

Ayam

Reference Manual

Randolf Schultz (randolf.schultz@gmail.com)

5. Jun 2024

This is the documentation of Ayam 1.35 – a free 3D modelling environment for the RenderMan interface. Please note, that this document is a reference manual, more detailed explanations of how to actually model with Ayam are given in the tutorials. This document has been written using the SGML-Tools (LinuxDoc) formatting system to generate files in a variety of text formats from one source file. There are HTML and PDF versions of this document prepared. In addition, the provided SGML-source can be used to generate other formats.

Overview

1	Introduction	15
2	The Ayam GUI	18
3	Modelling Actions	75
4	Objects, Properties, and Tags	107
5	NURBS Modelling Tools	273
6	Scripting Interface	345
7	Import and Export	485
8	Miscellaneous	511
9	Index	541

Contents

1	Introduction	15
1.1	About this Manual	15
1.2	About Ayam	16
1.2.1	History	16
1.2.2	Features Overview	16
1.2.3	Coordinate Systems and Units	17
1.2.4	Rational Style	17
2	The Ayam GUI	18
2.1	Anatomy of the Main Window	19
2.1.1	Objects	19
2.1.2	Properties	24
2.1.3	The Console	27
2.2	Main Menu	28
2.3	Main Window Keyboard Shortcuts	41
2.4	Anatomy of a View	42
2.5	View Menu	42
2.6	View Window Shortcuts and Actions	48
2.7	Drawing Modes	50
2.8	The Toolbox	53
2.9	Object Search	54
2.9.1	Search Expressions	54
2.9.2	Search Actions	56
2.10	Preferences	58
2.10.1	Main Preferences	59
2.10.2	Modelling Preferences	62
2.10.3	Drawing Preferences	64
2.10.4	RIB-Export Preferences	66
2.10.5	Miscellaneous Preferences	71
2.11	GUI Scaling	74

3	Modelling Actions	75
3.1	Modelling Actions Overview	76
3.2	Transforming Objects or Selected Points	77
3.3	Selecting Objects by Picking	78
3.3.1	Selecting Individual Objects	78
3.3.2	Drag-selecting Multiple Objects	78
3.3.3	Ambiguous Picking	79
3.4	Selecting/Tagging Points	79
3.5	Setting the Mark	82
3.6	Intermediate Parameter GUIs	84
3.7	Moving Objects or Selected Points	85
3.8	Rotating Objects or Selected Points	86
3.9	Rotating Objects or Selected Points about a Point	87
3.10	Scaling Objects or Selected Points	88
3.11	Scaling Objects or Selected Points about a Point	89
3.12	Editing Points	90
3.12.1	Expression Support	92
3.12.2	Context Menu	92
3.12.3	Keyboard Shortcuts	92
3.13	Editing Weights	93
3.14	Snapping Points to the Grid	95
3.15	Snapping Objects to the Grid	95
3.16	Snapping Points to the Mark	96
3.17	Snapping Objects to the Mark	96
3.18	Inserting or Deleting Points	97
3.19	Manipulating the Multiplicity of Points	98
3.20	Finding Points on Curves	99
3.21	Finding Points on Surfaces	100
3.22	Selecting Boundary Curves	101
3.23	Interactively Splitting Curves	102
3.24	Selecting/Tagging Boundary Points	103
3.25	Selecting Faces	104
3.26	Editing in Local Spaces	105

4	Objects, Properties, and Tags	107
4.1	Object Types Overview	109
4.1.1	Scene Organization	109
4.1.2	CSG/Solid Primitives	109
4.1.3	Freeform Curves	110
4.1.4	Freeform Surfaces	110
4.1.5	Curve Tool Objects	110
4.1.6	Surface Tool Objects	111
4.1.7	Polygonal and Subdivision Objects	111
4.1.8	Scripts and Plugins	112
4.2	Scene Organization Objects	113
4.2.1	Root Object	113
4.2.2	View Object	116
4.2.3	Camera Object	118
4.2.4	Light Object	119
4.2.5	Material Object	125
4.2.6	Level Object	128
4.2.7	Instance Object	131
4.2.8	Clone Object	134
4.2.9	Mirror Object	137
4.2.10	Select Object	139
4.2.11	RiInc Object	140
4.2.12	RiProc Object	141
4.3	CSG/Solid Primitives	142
4.3.1	Box Object	142
4.3.2	Sphere Object	143
4.3.3	Disk Object	144
4.3.4	Cone Object	145
4.3.5	Cylinder Object	146
4.3.6	Torus Object	147
4.3.7	Paraboloid Object	148
4.3.8	Hyperboloid Object	149
4.4	Freeform Curve Objects	150

4.4.1	NCurve (NURBS Curve) Object	150
4.4.2	ICurve (Interpolating Curve) Object	155
4.4.3	ACurve (Approximating Curve) Object	158
4.4.4	NCircle (NURBS Circle) Object	160
4.4.5	More Curve Types	161
4.5	Curve Tool Objects	162
4.5.1	ConcatNC (Concatenate NURBS Curves) Object	162
4.5.2	ExtrNC (Extract NURBS Curve) Object	165
4.5.3	OffsetNC (Offset NURBS Curves) Object	168
4.6	Freeform Surface Objects	170
4.6.1	NPatch (NURBS Patch) Object	170
4.6.2	IPatch (Interpolating Patch) Object	174
4.6.3	APatch (Approximating Patch) Object	176
4.6.4	BPatch (Bilinear Patch) Object	178
4.6.5	PatchMesh Object	179
4.7	Surface Tool Objects	182
4.7.1	Revolve Object	182
4.7.2	Extrude Object	184
4.7.3	Swing Object	187
4.7.4	Sweep Object	190
4.7.5	Birail1 Object	195
4.7.6	Birail2 Object	198
4.7.7	Skin Object	201
4.7.8	Gordon Object	204
4.7.9	Bevel Object	208
4.7.10	Cap Object	211
4.7.11	ConcatNP (Concatenate NURBS Patches) Object	214
4.7.12	ExtrNP (Extract NURBS Patch) Object	217
4.7.13	OffsetNP (Offset NURBS Surfaces) Object	219
4.7.14	Text Object	221
4.7.15	Trim Object	223
4.8	Polygonal and Subdivision Objects	225
4.8.1	PolyMesh Object	225

4.8.2	SDMesh Object	227
4.9	Script and Custom Objects	229
4.9.1	Script Object	229
4.9.2	Custom Objects Management	238
4.9.3	SfCurve (Superformula Curve) Object	239
4.9.4	BCurve (Basis Curve) Object	241
4.9.5	SDCurve (Subdivision Curve) Object	243
4.9.6	Metaball Object	245
4.9.7	SDNPatch Object	248
4.10	Standard Properties	251
4.10.1	Transformations Property	251
4.10.2	Attributes Property	252
4.10.3	Material Property	252
4.10.4	Shader Properties	252
4.10.5	Caps Property	255
4.10.6	Bevels Property	256
4.10.7	Tags Property	257
4.11	Tags	258
4.11.1	RiAttribute Tag	259
4.11.2	RiOption Tag	260
4.11.3	TC (Texture Coordinates) Tag	260
4.11.4	PV (Primitive Variable) Tag	262
4.11.5	RiHider Tag	263
4.11.6	RiDisplay Tag	263
4.11.7	AsWire Tag	264
4.11.8	NoExport Tag	264
4.11.9	SaveMainGeom Tag	264
4.11.10	SavePaneLayout Tag	265
4.11.11	TP (Tessellation Parameter) Tag	265
4.11.12	DC (Depth Complexity) Tag	265
4.11.13	NP (New Property) Tag	266
4.11.14	RP (Remove Property) Tag	266
4.11.15	BNS (Before Notify Script) Tag	267

4.11.16 ANS (After Notify Script) Tag	268
4.11.17 UMM/VMM (U/V Min Max) Tag	269
4.11.18 BP (Bevel Parameters) Tag	269
4.11.19 CP (Cap Parameters) Tag	269
4.11.20 MN (Mean Normal) Tag	270
4.11.21 MP (Mean Point) Tag	270
4.11.22 SB (Selected Boundary) Tag	270
4.11.23 XML Tag	271
4.11.24 Internal Tags	271
5 NURBS Modelling Tools	273
5.1 General Remarks	273
5.2 Curve Creation Tools	274
5.2.1 Circular B-Spline Tool	274
5.2.2 NURBCircle Tool	275
5.2.3 Rectangle Tool	276
5.2.4 TrimRect Tool	277
5.2.5 Tween Curve Tool	278
5.3 Curve Modification Tools	279
5.3.1 Revert Tool	279
5.3.2 Open Tool	280
5.3.3 Close Tool	281
5.3.4 Refine Tool	282
5.3.5 Refine Knots Tool	283
5.3.6 Refine Knots With Tool	284
5.3.7 Coarsen Tool	285
5.3.8 Elevate Tool	286
5.3.9 Reduce Tool	287
5.3.10 Extend Tool	288
5.3.11 Clamp Tool	289
5.3.12 Unclamp Tool	290
5.3.13 Insert Knot Tool	291
5.3.14 Remove Knot Tool	292

5.3.15	Remove Superfluous Knots Tool	293
5.3.16	Fair Tool	294
5.3.17	Concat Tool	296
5.3.18	Split Tool	297
5.3.19	Curve Trim Tool	298
5.3.20	Reparameterisation Tool	299
5.3.21	Plot Curvature Tool	300
5.3.22	Interpolate Tool	301
5.3.23	Approximate Tool	302
5.3.24	Shift Closed Curve Tool	303
5.3.25	To XY Tool	304
5.3.26	Make Compatible Tool	305
5.3.27	Rescale Knots to Range Tool	306
5.3.28	Rescale Knots to Mindist Tool	306
5.3.29	Collapse Points Tool	307
5.3.30	Explode Points Tool	307
5.4	Surface Creation Tools	308
5.4.1	NURBSphere Tool	308
5.4.2	NURBSphere2 Tool	308
5.4.3	Revolve Tool	309
5.4.4	Swing Tool	309
5.4.5	Extrude Tool	310
5.4.6	Sweep Tool	310
5.4.7	Bevel Tool	311
5.4.8	Cap Tool	312
5.4.9	Birail1 Tool	312
5.4.10	Birail2 Tool	313
5.4.11	Gordon Tool	313
5.4.12	Skin Tool	314
5.4.13	Trim Tool	315
5.4.14	Tween Surfaces Tool	316
5.5	Surface Modification Tools	317
5.5.1	Revert U Tool	317

5.5.2	Revert V Tool	317
5.5.3	Swap UV Tool	317
5.5.4	Close U Tool	318
5.5.5	Close V Tool	318
5.5.6	Refine Knots Surface Tool	319
5.5.7	Refine Knots With Surface Tool	320
5.5.8	Elevate Surface Tool	321
5.5.9	Reduce Surface Tools	322
5.5.10	Clamp Surface Tool	323
5.5.11	Unclamp Surface Tool	324
5.5.12	Insert Knot Surface Tool	325
5.5.13	Remove Knot Surface Tool	326
5.5.14	Remove Superfluous Knots Surface Tools	327
5.5.15	Split Surface Tool	328
5.5.16	Interpolate Surface Tool	329
5.5.17	Approximate Surface Tools	330
5.5.18	Fair Surface Tool	331
5.5.19	Make Surfaces Compatible Tool	332
5.5.20	Rescale Knots to Range Surface Tool	333
5.5.21	Rescale Knots to Mindist Surface Tool	333
5.6	Conversion Tools	334
5.6.1	Extract Curve Tool	334
5.6.2	Extract Patch Tool	334
5.6.3	Break into Curves Tool	335
5.6.4	Build from Curves Tool	336
5.6.5	Tessellation Tool	337
6	Scripting Interface	345
6.1	Global Variables and Arrays	346
6.1.1	Global Variables	346
6.1.2	The Global Array ay	346
6.1.3	The Global Array ayprefs	346
6.1.4	The Global Array aymark	346

6.1.5	Property Management and Data Arrays	347
6.2	Procedures and Commands	349
6.2.1	Getting Help on Scripting Interface Commands	349
6.2.2	Managing Objects	350
6.2.3	Interrogating Objects	371
6.2.4	Selecting Objects	375
6.2.5	Selecting Points	377
6.2.6	Selecting Faces	378
6.2.7	Manipulating Properties	379
6.2.8	Clipboard Operations	381
6.2.9	Hierarchy Operations	383
6.2.10	Transformations	384
6.2.11	Manipulating Shaders	387
6.2.12	Manipulating Tags	388
6.2.13	Manipulating Curves	390
6.2.14	Manipulating Surfaces	392
6.2.15	Manipulating NURBS Curves	393
6.2.16	Manipulating NURBS Surfaces	400
6.2.17	Manipulating PolyMesh Objects	412
6.2.18	Manipulating Points and Normals	413
6.2.19	Vector Algebra	418
6.2.20	Updating the GUI	420
6.2.21	Managing Preferences	421
6.2.22	Custom Objects / Plugins	421
6.2.23	Applying Commands to a Number of Objects	422
6.2.24	Scene IO	423
6.2.25	RIB Export	424
6.2.26	Reporting Errors	424
6.2.27	Property GUI Management	425
6.2.28	Miscellaneous	433
6.3	Expression Support in Dialog Entries	439
6.4	Scripting Interface Examples	440
6.4.1	Moving Objects	440

6.4.2	Moving NURBS points	441
6.4.3	Easy Sweep	442
6.4.4	Toolbox Buttons	444
6.5	Distributed Helper Scripts	446
6.5.1	Repair Ayam	446
6.5.2	Test Ayam	447
6.5.3	Use Ayam as Command Line Converter	447
6.5.4	Shader Parsing	448
6.5.5	Convert Everything to Polygons	449
6.5.6	Convert Everything to NURBS patches	449
6.5.7	Restrict the Console	449
6.5.8	Color the Focus Ring	449
6.5.9	Decouple Hierarchy View Link	450
6.5.10	Shuffle Objects Up/Down	450
6.5.11	Open/Close All Tree Levels	450
6.5.12	Automatic About Center Actions	451
6.5.13	Automatic Point Actions	451
6.5.14	Save Selected Points	452
6.5.15	Revert Cursor Key Behavior	452
6.5.16	Access Core Functions from the Toolbox	452
6.5.17	Switch Dialogs to Kdialog	452
6.5.18	Switch Dialogs to Zenity	452
6.5.19	Switch File Dialogs to Tcl	453
6.5.20	Use Aqsis from Application Directory	453
6.5.21	Use Pixie from Library Directory	453
6.5.22	Replace Icons	453
6.5.23	Dynamic Tree	453
6.5.24	Control Vertex View	454
6.5.25	Curvature Plot	455
6.5.26	NURBS for X3DOM	456
6.6	Distributed Script Objects	460
6.6.1	Truncated Cone	460
6.6.2	Disk with Hole	461

6.6.3	Box with Cylindrical Topology	462
6.6.4	NURBS Circle with Triangular Base	463
6.6.5	Helix	464
6.6.6	Spiral	465
6.6.7	Oval	466
6.6.8	FilletNC	467
6.6.9	DualSweep	468
6.6.10	Translational Surface	469
6.6.11	Extrusion along Normal	470
6.6.12	Create Polyhedrons from Conway Notations	471
6.7	JavaScript Scripting Interface	473
6.7.1	Accessing JavaScript from Tcl and Script Objects	473
6.7.2	JavaScript Functions	474
6.7.3	Data Conversion	476
6.7.4	Complete Examples	477
6.8	Lua Scripting Interface	479
6.8.1	Accessing Lua from Tcl and Script Objects	479
6.8.2	Lua Functions	480
6.8.3	Data Conversion	482
6.8.4	Complete Examples	483
7	Import and Export	485
7.1	Import and Export Plugin Management	485
7.2	Import and Export Plugin Overview	485
7.3	RenderMan Interface Bytestream (RIB) Import	487
7.3.1	RIB Primitive Support	487
7.3.2	RIB Import Options	488
7.4	RenderMan Interface Bytestream (RIB) Export	488
7.5	Mops Import	489
7.6	AutoCAD DXF Import	489
7.6.1	DXF Entity Support	489
7.6.2	DXF Import Options	490
7.7	AutoCAD DXF Export	491

7.7.1	Ayam Object and Properties Support	491
7.7.2	DXF Export Options	491
7.8	Wavefront OBJ Import	492
7.8.1	Wavefront OBJ Statement Support	492
7.8.2	Wavefront OBJ Import Options	493
7.8.3	Wavefront Material Import	494
7.9	Wavefront OBJ Export	497
7.9.1	Ayam Object and Properties Support	497
7.9.2	Wavefront OBJ Export Options	497
7.9.3	Wavefront Material Export	498
7.10	3DMF (Apple) Import	499
7.10.1	3DMF Primitive and Attribute Support	499
7.10.2	3DMF Import Options	500
7.11	3DMF (Apple) Export	500
7.11.1	Ayam Object and Properties Support	500
7.11.2	Trim Curves Support	501
7.11.3	3DMF Export Options	502
7.12	3DM (Rhino) Import	502
7.12.1	3DM Object Support	502
7.12.2	3DM Import Options	502
7.13	3DM (Rhino) Export	503
7.13.1	Ayam Object and Properties Support	503
7.13.2	3DM Export Options	504
7.14	X3D (Web3D) Import	505
7.14.1	X3D Element Support	505
7.14.2	X3D Attribute Support	508
7.14.3	X3D Import Options	508
7.15	X3D (Web3D) Export	509
7.15.1	Ayam Object and Properties Support	509
7.15.2	Wire-frame Support	509
7.15.3	X3D Export Options	510

8	Miscellaneous	511
8.1	The Undo System	511
8.2	The Modelling Concept Tool-Objects	511
8.3	Scene File Management	514
8.3.1	Opening Scene Files	514
8.3.2	Inserting Scene Files	515
8.3.3	Saving Scene Files	515
8.4	Ayamrc File	516
8.4.1	Changing Keyboard Shortcuts	517
8.4.2	Hidden Preference Settings	517
8.4.3	RiOption and RiAttributes Database	525
8.5	Command Line Arguments	526
8.6	Environment Variables	527
8.7	Plugins Overview	528
8.8	Shader Parsing Plugins	529
8.9	Viewport Rendering	530
8.10	Automatic Instancing	531
8.11	Importance Driven Rendering (IDR)	531
8.12	CSG preview using the AyCSG plugin	532
8.13	Increasing Drawing Speed	536
8.14	Modelling Without Views	536
8.15	Restrictions and Implementation Deficiencies	536
8.16	How to Join the Fun	537
8.17	References	538
8.18	Acknowledgements	539
8.19	Trademarks	540
9	Index	541

1 Introduction

This section contains general information about this manual and Ayam.

1.1 About this Manual

As this document is a reference manual, it is probably pointless to read it from the beginning to the end (except maybe for the next section, explaining the basics of the user interface). Instead, just look up the documentation of the things you are interested in via the table of contents or the index. Cross references will then guide you to other important parts of the documentation. Again: this manual has a rather large index, please use it (see section 9 [Index \(page 541\)](#))!

This document is organized in the following way:

After this first introductory section, the user interface of Ayam is explained and basic handling instructions for the application are given in the second section.

In the third section all interactive modelling actions are documented.

The fourth section details all object types and object properties, followed by documentation on all NURBS modelling tools in the fifth section.

In the sixth section the Tcl scripting interface is explained and the following seventh section has all information about the import and export plugins.

The eighth section is the dreaded miscellaneous section that contains documentation not fitting elsewhere.

In this manual, the following typographic conventions are used:

- keyboard shortcuts: <Ctrl+c> (press control *and* c key), for shortcuts like <Ctrl+Shift+t> an abbreviated version: <Ctrl+T> will be used;
- names (of object types, menu entries, properties, or property elements): "A Name";
- Tcl code examples:

```
set riopt(runtime) { a b }
```

- Object hierarchies:

```
+--Parent_Object (Type)
|  |--First_Child_Object (Type)
|  |--Second_Child_Object (Type)
|  |--[Third_Child_Object_may_be_present_or_not (Type) ]
|  |--Empty_Level (Level)
|  +-Sub_Level (Level)
|     |--First_Child_Object_of_Sub_Level (Type)
|     |--\--Last_Child_Object_of_Sub_Level (Type)
|     |--\--Last_Child_Object (Type)
+-Next_Parent_Object (Type)
```

- Commands:
 - » command arguments

1.2 About Ayam

Ayam is a free 3D modelling environment for the RenderMan Interface, distributed under the modified BSD licence (no advertisement clause).

1.2.1 History

Ayam is in development since 1997 and was formerly known as "The Mops". Ayam formed the software base of the authors PhD work. New versions of Ayam were released in approximately half year intervals since.

1.2.2 Features Overview

Here is a short summary of the Ayam feature set:

- RIB (RenderMan Interface Bytestream) export and import.
- Support for NURBS curves, interpolating and approximating curves, (trimmed) NURBS surfaces, interpolating surfaces, bilinear and bicubic patches and patch meshes, Boxes, Quadrics (Sphere, Disk, Cylinder, Cone, Hyperboloid, Paraboloid and Torus), MetaBalls, polygonal meshes, subdivision meshes and more.
- All primitives may be combined with the common CSG-operations: Intersection, Difference, and Union.
- NURBS modelling includes extrude, revolve, sweep, birail, skin, and gordon operations (with caps, holes, and bevels) realized as Tool-Objects (see also section 8.2 [The Modelling Concept Tool-Objects \(page 511\)](#)).
- Wavefront OBJ export and import, Rhino 3DM export and import, AutoCAD DXF export and import, Web3D X3D export and import, Apple Quicktime 3D Metafile (3DMF) export and import.
- Custom objects that may freely implement their representations (using OpenGL and RIB) and even small GUIs to edit their type specific parameters may be written by the user and dynamically loaded at runtime.
- Scripting interfaces: Tcl, JavaScript, Lua.
- Script objects.
- Miscellaneous: (automatic) instancing, arbitrary number of modelling views, object clipboard, independent property clipboard, console, n-level undo.

Since Ayam 1.12, dynamic loading of custom objects and certain plugins are also available on the Win32 platform (they were not available before).

Ayam is primarily aimed at the Linux, IRIX, and Win32 platforms. On those platforms BMRT (Blue Moon Rendering Tools, a RenderMan compliant renderer by Larry Gritz) is available. Even though the distribution of BMRT is stopped it is still the recommended renderer for Ayam. Despite of this, Ayam may be used on many more platforms with any RenderMan compliant renderer.

For platforms where BMRT is not available (e.g. FreeBSD or NetBSD), Ayam may be compiled with code from the Affine Toolkit with limited functionality (see also the file INSTALL). In this case, no parsing of slc compiled shaders will be possible.

Since Ayam 1.6 it is also possible to completely replace the BMRT shader parsing and RIB writing code with code from the Aqsis project, thus completely eliminating the need for BMRT. Furthermore, shader parsing plugins are available for all major RenderMan compliant renderers allowing a tight integration of Ayam with any of those renderers.

1.2.3 Coordinate Systems and Units

Ayam uses a right-handed coordinate system as used by OpenGL but as opposed to RenderMan (the latter is using a left-handed coordinate system).

In the default modelling view of type "Front", the positive X-axis points to the right, the positive Y-axis points upwards and the Z-axis points outside the screen, to the user. See also the following image.

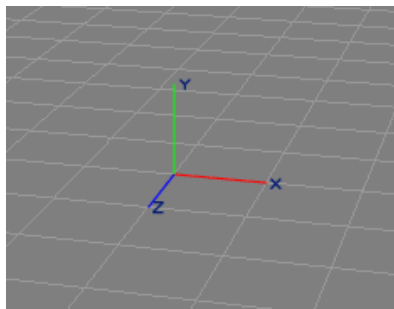


Figure 1: Right Handed Coordinate System

All coordinate values in Ayam are dimensionless. It is up to the user to define what a value of 1.0 means: e.g. one centimeter or one meter.

Due to the limited precision of floating point arithmetics used in Ayam, coordinate values should be defined in the range between -10.0 and 10.0 for most accurate results.

1.2.4 Rational Style

Ayam stores rational coordinates in *euclidean* rational style, i.e. the weight of a control point is not pre-multiplied with the coordinates (this will be done internally when handing the coordinates over for drawing or export purposes).

In Ayam versions prior to 1.19 the weights were always multiplied with the coordinate values (i.e. *homogeneous* rational coordinates) to allow faster drawing with OpenGL/GLU, which expects rational coordinate values to be delivered this way. But a negative side effect of this approach was that modification of coordinates or weights for modelling purposes was unnecessarily complicated.

Ayam can still display the coordinates in both styles, controlled by the preference option "RationalPoints" (see also section 2.10.2 [Modelling Preferences \(page 62\)](#)), but internally the coordinates are kept in euclidean rational style.

Scene files from older versions of Ayam are converted automatically to the new rational style when read by Ayam 1.19 and above, but loading of scene files written by Ayam 1.19 and above into older versions of Ayam requires manual intervention (e.g. by means of a script).

2 The Ayam GUI

This section describes the user interface of Ayam.

The user interface of Ayam is composed of three types of windows: a main window, a toolbox and an arbitrary number of view windows. The main window displays the object hierarchy and allows to edit object properties. The toolbox window is for easy creation of objects and starting of modelling actions and tools. The modelling actions are then carried out in view windows, where also the scene is displayed.

Until Ayam 1.14 all those windows were always separate top level windows – a so called *floating windows GUI mode*. Since version 1.14, a new GUI mode is available where the main window, three view windows and the toolbox are integrated in one top level window. This mode is called *single window GUI mode*, see also the image above. The new single window GUI mode is enabled by default. All sub windows are in panes, the space occupied by a sub window may be adjusted by dragging the mouse at the borderlines of the panes. The number of views is not limited to three, albeit all extra views will become extra top level windows.

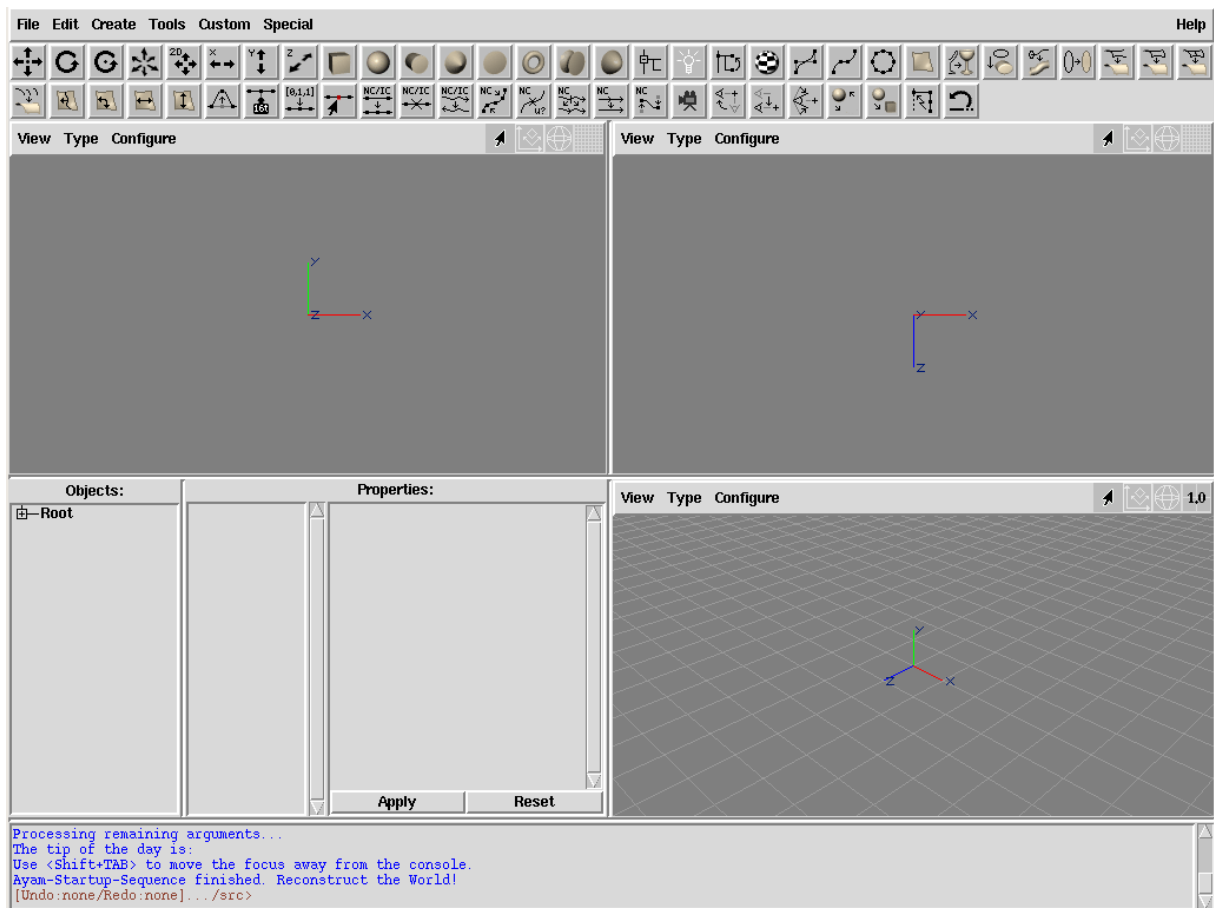


Figure 2: Single Window GUI Mode

The next sections document the three types of windows (main, toolbox, and views) in detail.

2.1 Anatomy of the Main Window

The main window is split into three major areas:

1. an area named "Objects:"
2. an area labeled "Properties:"
3. and a text widget (the so called "Console")

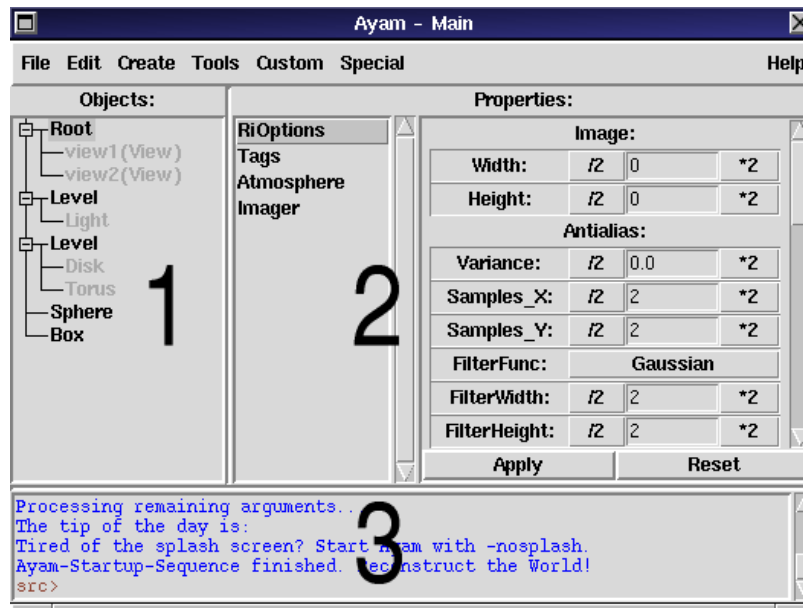


Figure 3: The Main Window

The relative sizes of the three areas are managed by a so called paned geometry management. To change the relative size of the console, move the mouse pointer to the upper border of the console until the pointer changes and then drag the border. The same goes for the right border of the objects section.

2.1.1 Objects

The default representation of the object hierarchy is a tree view. The second available representation is a simple listbox (as known from "The Mops"). The label "Objects" may be used to switch between the two representations of the object hierarchy quickly (using a double click). It is also possible to switch between both representations using the context menu.

The two representations have very different properties regarding speed, use of resources, and versatility. The tree is, due to the drag and drop operations, much more versatile but also slower.

Both representations manage a so called *current level*. This level is the scene level that is displayed in the object listbox. In the tree view the current level is drawn in black while all other levels are grayed out. Selection of objects may take place in the current level only!

After the start-up of Ayam you will notice, that there is a first object called "Root" in the top level of the scene, even though the scene seems to be empty. See section 4.2.1 Root Object (page 113) for more information regarding this special object, and what it is good for. Note, that this object can not be deleted or copied.

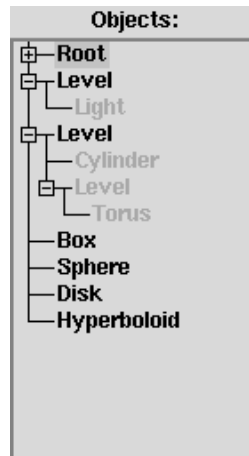
Object Tree View:

Figure 4: Object Tree View

The object tree view is quite complex and may be slow on slow machines (of the Pentium 90 class), especially when dealing with scenes that contain many objects. This should not be a problem nowadays. Nevertheless, Ayam tries to keep tree update delays as low as possible, but this works only if the scene uses the hierarchy and changes happen in sub levels (not the root level). Further speedups may be achieved with the help of the DTree script (see section [6.5.23 Dynamic Tree \(page 453\)](#)).¹

In the tree view, objects may be selected using the left mouse button. Multiple selection of objects is possible by holding down the <Shift> or <Ctrl> key while clicking on objects.

Double clicking on objects with child objects toggles display of the child level. The same may be accomplished using single clicks on the well known plus/minus symbols in front of the name of those objects.

Drag and drop operation is also possible to move objects in the hierarchy and to initiate special actions like connecting materials to objects. However, this last feature is documented in section [4 Objects, Properties, and Tags \(page 107\)](#) as it is object type specific.

The rightmost mouse button opens a context menu with basic tree and clipboard operations:

- "Tree/Rebuild" completely removes the tree nodes, rebuilds the hierarchy, makes the top level current, and clears the object selection,
- "Tree/Expand All" opens all nodes with child nodes,
- "Tree/Collapse All" closes all nodes with child nodes,
- "Tree/Expand Selected" opens all selected nodes,
- "Tree/Collapse Selected" closes all selected nodes,
- "Switch to Listbox" removes the tree view and replaces it with the object listbox (see below).
- "Deselect Object" deselects the currently selected object(s).
- "Copy Object", "Cut Object", "Paste Object", "Delete Object" are standard clipboard operations as documented in section [2.2 Main Menu \(page 31\)](#).
- "Help on Object" displays the help of the selected object.

¹ Since 1.22.

The scene may also be navigated and objects may be selected using the keyboard:¹

- <Up> and <Down> move the selection to the previous or next object in the current level.
- <Shift-Up> and <Shift-Down> will not move the selection, but rather extend it in the respective direction.²
- <Home> and <End> select the first or last object in the current level,
- <Shift+Home> and <Shift+End> extend the selection to the first or last object in the current level respectively.³ The Root object, however, will always be omitted.
- <Right> enters the (first) selected object,
- <Left> enters the parent level,
- <Ctrl+a> and <Ctrl+n> select or de-select all objects in the current level. If the current level is the root level, the Root object will not be selected by <Ctrl+a>.
- <Space> toggles display of the child objects of the selected object(s).
- <Shift+Space> toggles display of all sub-levels of the selected object(s).⁴
- <+> opens all sub-levels of the selected object(s).⁵
- <-> closes all sub-levels of the selected object(s).⁶
- <Alt+Up> and <Alt+Down> shuffle the selected objects in the current level in the respective direction, unless the Root object is selected and unless the first object is selected for up-movement or the last object for down-movement.⁷
- <^> moves the focus to the first view (in single window GUI mode).⁸

If those shortcuts do not work you may need to move the keyboard input focus away from (internal) view windows, the property GUI, or the console using <Tab> or <Shift+Tab> first.

Another way of moving the focus (and cleaning up the application state) is by using the <Esc> key: In property GUIs and the console, pressing <Esc> moves the focus away to the main window or object selection window.

Pressing <Esc> twice in a view window will also reset the focus to the main window/object selection window.⁹ Pressing <Esc> twice in the object selection window will additionally clear the selection (this implies removal of the currently displayed property GUI) and change the current level to the root level.

Thus, if you feel lost anywhere in Ayam, just press <Esc> twice or thrice.

¹ Since 1.6. ² Since 1.7. ³ Since 1.11. ⁴ Since 1.18. ⁵ Since 1.21. ⁶ Since 1.21. ⁷ Since 1.21. ⁸ Since 1.24. ⁹ Since 1.15.

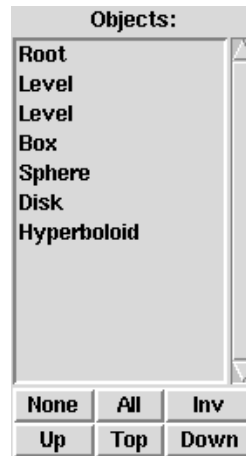
Object Listbox:

Figure 5: Object Listbox

The object listbox displays the object hierarchy of the current scene. Using this listbox you may browse through the hierarchy of the scene with your mouse and you may select one or more objects.

Browsing and selecting should be very intuitive: Use a double click to enter a level (or an object with child objects), use a single click to select objects, multiple objects may be selected using click and drag, or holding down the <Shift> or <Ctrl> key while clicking. Keyboard operation is also possible if the listbox has the input focus.

A ". . ." is displayed as the first element of the current level if you are "inside" a level or another object. A double click on the ". . ." takes you to the parent level. The buttons below the listbox may be used to change the selection or to quickly jump through the hierarchy. They should be self explanatory.

The rightmost mouse button opens a small context menu:

- "Switch to Tree" removes the listbox and replaces it with the tree view (see above).
- "Deselect Object" deselects the currently selected object(s).
- "Copy Object", "Cut Object", "Paste Object", "Delete Object" are standard clipboard operations as documented in section 2.2 Main Menu (page 31).
- "Help on Object" displays the help of the selected object.

The scene may also be navigated and objects may be selected using the keyboard:¹

- <Up> and <Down> move the selection to the previous or next object in the current level.
- <Shift-Up> and <Shift-Down> will not move the selection, but rather extend it in the respective direction.²
- <Home> and <End> select the first or last object in the current level,
- <Shift+Home> and <Shift+End> extend the selection to the first or last object in the current level respectively.³ The Root object, however, will always be omitted.
- <Right> enters the (first) selected object,
- <Left> enters the parent level,

¹ Since 1.6. ² Since 1.7. ³ Since 1.11.

- `<Ctrl+a>` and `<Ctrl+n>` select or de-select all objects in the current level. If the current level is the root level, the Root object will not be selected by `<Ctrl+a>`.
- `<Alt+Up>` and `<Alt+Down>` move the selected objects in the current level in the respective direction, unless the Root object is selected and unless the first object is selected for up-movement or the last object for down-movement.¹
- `<^>` moves the focus to the first view (in single window GUI mode).²

If those shortcuts do not work you may need to move the keyboard input focus away from (internal) view windows, the property GUI, or the console using `<Tab>` or `<Shift+Tab>` first.

Another way of moving the focus (and cleaning up the application state) is by using the `<Esc>` key: In property GUIs and the console, pressing `<Esc>` moves the focus away to the main window or object selection window.

Pressing `<Esc>` twice in a view window will also reset the focus to the main window/object selection window.³

Pressing `<Esc>` twice in the object selection window will additionally clear the selection (this implies removal of the currently displayed property GUI) and change the current level to the root level.

Thus, if you feel lost anywhere in Ayam, just press `<Esc>` twice or thrice.

¹ Since 1.21. ² Since 1.24. ³ Since 1.15.

2.1.2 Properties

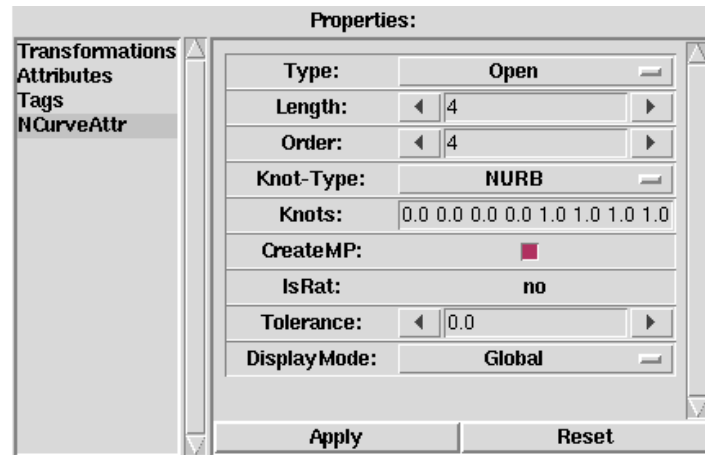


Figure 6: Properties

The listbox right next to the object hierarchy displays the properties of the currently selected object.

If there are multiple selected objects, the properties listbox will display no properties at all.

Unlike the object tree/listbox, where multiple entries can be selected, only one property may be selected. If a property is selected, the associated GUI will be shown in the appropriate area (on the right hand side).

Also the keyboard may be used to select properties: just press one of the $\langle 0 \rangle$ – $\langle 9 \rangle$ keys (most comfortably using the numeric keypad). $\langle 0 \rangle$ always selects the last and often the only object type specific property, whereas $\langle 1 \rangle$ selects the first property, which contains the linear transformations in most cases.¹

The properties listbox also has a context menu. The entries in this menu allow to:

- Deselect the current property,
- Copy/Paste the currently selected property to/from the property clipboard (see below),
- Add/Remove a property to/from the currently selected object by means of NP/RP tags (see also sections 4.11.13 NP (New Property) Tag (page 266) and 4.11.14 RP (Remove Property) Tag (page 266)),
- display the help section of the currently selected property.

Property GUIs

All property GUIs use more or less standardized GUI elements that are organized in list form, see also the image above. The lists may be scrolled if they get too long to fit into the window.

What properties exactly will be shown, and how the GUIs look alike depends on the selected object and the selected property. This is documented comprehensively in section 4 Objects, Properties, and Tags (page 107).

If Ayam is in floating windows GUI mode and the elements of the current property GUI do not fit horizontally into the screen space that is defined by the main window size, Ayam can automatically resize the main window. This behaviour may be controlled using the preference setting "AutoResize" (see section 2.10 Preferences (page 58)).

¹ Since 1.8

If an object and a property are selected and a different object is selected, the property GUI that has the same index as the previously selected property in the properties listbox will be selected and shown. This is not necessarily a property of the same type. To avoid that or to clear the property GUI for fast browsing through the scene you may either double click on the "Properties" label, hit the <Esc> key three times, or use the context menu of the properties listbox to de-select the current property.

The various things that may be changed using a property GUI will normally not be applied to the selected object until the "Apply"-button is pressed.

However, holding down the "Shift" button while interacting with the property GUI or pressing the "Return" key when entry widgets have the keyboard input focus will lead to an instant application of all changed values.¹

Note that property GUIs of custom objects may offer interactive elements that also do an instant "Apply" operation. But most property GUI elements of the core objects of Ayam do not change anything until the "Apply"-button is used.

All changes to the arguments of a property that have been made since either opening the property or the last "Apply" operation (whatever was last) can be reverted with the "Reset"-button. Mind that this does not use the undo mechanism of Ayam but rather copies Tcl data, just like the property clipboard.

If a property GUI element has the keyboard input focus (it is then usually displayed with a black rim around it), all the keyboard shortcuts for the main menu and scene navigation will have no effect until the keyboard input focus is moved away from the property GUI. This may be accomplished easily using the <Esc> key.

Property Clipboard

With the help of the property clipboard arbitrary property data may be copied from one and then pasted to another object. See the "Edit" menu for access to this functionality.

Pasting a property to multiple selected objects does work too. This is a great way to apply e.g. a parameterised surface shader to a bigger number of material objects, without going the long way of setting a new shader and entering parameters for it for every material object.

Since you may not want to copy and paste whole properties all the time, you may even mark single parameters with a double click on the labels of the parameters. The selected parameters will then be preceded by an exclamation mark (!) in the property GUI.

If this property is then copied, all marked parameters will be omitted.

It is also possible to copy just the selected parameters using "Copy Marked Prop".

¹ Since 1.8.3.

A simple example for such advanced use of the property clipboard:

Our task is to give a big number of material objects the same color, but they already have different opacity settings. Copying the complete attribute property would destroy the individually adjusted opacity values. We can solve this by copying just the color attribute, but leave all other attributes as they are:

1. Change the color of a first material object using the "Attributes" property GUI. (Do not forget the "Apply" button!)
2. Mark the color parameter as to be copied using a double click on the text "Color"; it should read "!Color" now.
3. Copy just the color parameter to the property clipboard, using "Copy Marked Prop" in the "Edit" menu or the hot key <Ctrl+I>.
4. Select all other material objects.
5. Paste the property using "Paste Property" or <Ctrl+V>.
6. All done.

An even more advanced application of the property clipboard is to paste property values to different types of properties (e.g. pasting parameters from a surface shader to a displacement shader, or pasting a radius value from a sphere to a disk). This can be done using "Paste Property to Selected" in the "Special/Clipboard" sub-menu.

However, special care must be taken when pasting incomplete properties to objects which do not have complete properties already. Do not paste an incomplete shader property to an object which does not already have the same shader.

2.1.3 The Console

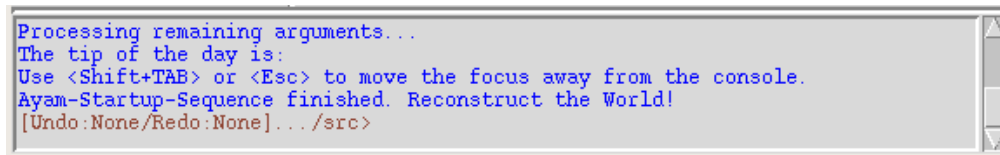


Figure 7: Console

The third part of the main window is the console. The console is mainly for unobtrusive text output (informative, warning, and error messages). If something does not work as advertised, the console may be worth a look.

The console captures the `stderr` and `stdout` channels of the Tcl-interpreter Ayam is running in. It is also possible to redirect all internal Tcl error messages, that would normally cause a Tcl error dialog window to appear, to the console using the preference setting "Misc/RedirectTcl" (see section 2.10.5 Miscellaneous Preferences (page 71)).

Furthermore, commands or even complete new Tcl procedures can be directly entered into the console. However, this is a feature for an advanced user that studied section 6 Scripting Interface (page 345). You need to explicitly click into the console to give it the input focus and thus enable input.

An important thing to know is that the keyboard shortcuts for the various main menu entries do not work if the console has the input focus. Instead, other keyboard shortcuts (related to the console) are in effect. How do you get out of this? Simply press `<Shift+Tab>` or `<Esc>` to move the focus away from the console and enable the main menu shortcuts again.

Note that the `<Tab>` key alone does not move the focus away from the console. `<Tab>` instead completes names of files, commands (procedures), variables, and widgets. You may try this out by typing `tip` in the console, then press `<Tab>`. The console automatically completes `tip` to `tipoftheDay` (the procedure that prints out the tip of the day, just try it).

Remember that many commands of the Ayam scripting interface work in background: without update of object selection widget, property GUI or redrawing of view windows. But it is possible to enforce an immediate update of the GUI and redrawing of all views by using `<Shift-Enter>` instead of `<Enter>` when entering commands.

Another simple demonstration of the consoles capabilities:

- Create ten box objects by clicking on the box icon ten times.
- Select all ten boxes.
- Go to the console by clicking into it.
- Enter the following: `forAll {movOb $i 0 0; rotOb [expr $i*10] 0 0}`

This example uses three procedures:

- `forAll`: allows to execute a command for each of the selected objects, or for each object in the current level if no objects are selected.
- `movOb`: moves the selected object(s).
- `rotOb`: rotates the selected object(s).

See section 6 [Scripting Interface \(page 345\)](#) for a listing of all the available commands.

Note that the example uses a side effect (the variable "i" that holds the index of the currently processed object) to calculate the amount of the movement and rotation.

For more information regarding the console, please refer to the appropriate documentation by the original author Jeffrey Hobbs (see the console context menu, that can be opened with the right mouse button).

2.2 Main Menu

This section discusses the main menu bar.

Note that many menu entries have keyboard shortcuts that are displayed in each entry. But those shortcuts only work if the main window has the keyboard input focus *and* the input focus is not in the console or in a property GUI element (i.e. a data entry field). Hit <Esc> to set the focus to the object selection widget and thus enable the main menu keyboard shortcuts.

These keyboard shortcuts may be adapted using the *ayamrc* file (see section 8.4 [Ayamrc File \(page 516\)](#)).

Another way of navigating the menu is via the <Alt> or <Menu> key, pressed together with the various underlined characters in the menu entries. Once a menu is open, it may also be navigated with the cursor keys. The <Return> key invokes the currently selected menu entry and the <Esc> key closes the menu.

The "File" menu deals with standard file operations:

- "New", clears the current scene (deletes all objects) and reloads the working environment.
This operation cannot be undone!

The standard keyboard shortcut for this operation is <Ctrl+N>.

- "Open", clears the current scene and closes all views, then loads a new scene from disk.

All objects from the file will be read. A backup copy of the file will be made before loading (depending on the preference setting "Main/BakOnReplace"). See section 8.3.1 [Opening Scene Files \(page 514\)](#) for a more detailed discussion.

This operation cannot be undone!

The standard keyboard shortcut for this operation is <Ctrl+o>.

Also files supported by any of the import plugins may be imported using this route.¹

Note that this only works if the selected file has a file name extension. The appropriate plugin will be loaded automatically (from the list of plugin directories in the preferences) if needed and the import options dialog of the plugin will be opened, with the "FileName" option already set. Mind that in this case, no backup copy of the file will be made. See also section 7 [Import and Export \(page 485\)](#).

- "Insert", inserts the objects and views of an Ayam scene file into the current scene.

All objects from the file will be read. If the file to be inserted contains a Root or View objects, the new objects will be created in the top level of the scene. Otherwise, if just geometric objects are in the scene file (i.e. the file was created using "Special/Save Selected"), the new objects will be inserted in the current level of the scene. See section 8.3.2 [Inserting Scene Files \(page 515\)](#) for a more detailed discussion.

This operation cannot be undone!

The standard keyboard shortcut for this operation is <Ctrl+i>.

Also files supported by any of the import plugins may be imported using this route. See above ("Open").²

¹ Since 1.13. ² Since 1.13.

- "Save as", saves the current scene to a file asking for a new file name. All objects in the scene will be saved to the scene file, but if the current scene was loaded from a file without Root object (and thus without view windows), Root and views will be omitted from the saved scene file as well. See section [8.3.3 Saving Scene Files \(page 515\)](#) for a more detailed discussion.

The standard keyboard shortcut for this operation is <Ctrl+S>.

Also files supported by any of the export plugins may be exported using this route. Just pick a file name with the desired extension (see above, "Open").¹

- "Save", saves the scene to a file. If the scene has not been saved before (the scene file name is "unnamed") Ayam will ask for a file name first. All objects in the scene will be saved to the scene file, but if the current scene was loaded from a file without Root object (and thus without view windows), root and views will be omitted from the saved scene file as well. See section [8.3.3 Saving Scene Files \(page 515\)](#) for a more detailed discussion.

The standard keyboard shortcut for this operation is <Ctrl+s>.

- "Import/", since Ayam 1.13 this sub menu is initially empty. To gain access to the menu entries described here, the respective plugin must be loaded.
 - "Import/Apple 3DMF", import a scene from the Apple 3DMF format, see section [7.10 3DMF \(Apple\) Import \(page 499\)](#) for more information.
 - "Import/AutoCAD DXF", import a scene from the AutoCAD DXF format, see section [7.6 AutoCAD DXF import \(page 489\)](#) for more information.
 - "Import/Mops", import a scene from The Mops, see section [7.5 Import of Mops Scenes \(page 489\)](#) for more information.
 - "Import/RenderMan RIB", import a Renderman Interface Bytestream, see section [7.3 RenderMan RIB Import \(page 487\)](#) for more information.
 - "Import/Rhino 3DM", import a scene from the Rhino 3DM format, see section [7.12 3DM \(Rhino\) Import \(page 502\)](#) for more information.
 - "Import/Wavefront OBJ", import a scene from the Wavefront OBJ format, see section [7.8 Wavefront OBJ Import \(page 492\)](#) for more information.
 - "Import/Web3D X3D", import a scene from the XML based X3D format published by the Web3D Consortium, see section [7.14 X3D \(Web3D\) Import \(page 505\)](#) for more information.
- "Export/", since Ayam 1.13 this sub menu initially only contains the "RenderMan RIB" entry. To gain access to the other menu entries described here, the corresponding plugin must be loaded.
 - "Export/RenderMan RIB", exports the current scene to a RIB file, asking which camera (which view) to use. The standard keyboard shortcut for this operation is <Ctrl+E>.
 - "Export/Apple 3DMF", export a scene to the Apple 3DMF format, see section [7.11 3DMF \(Apple\) Export \(page 500\)](#) for more information.
 - "Export/Rhino 3DM", export a scene to the Rhino 3DM format, see section [7.13 3DM \(Rhino\) Export \(page 503\)](#) for more information.
 - "Export/Wavefront OBJ", exports the current scene to a Wavefront OBJ file, see also section [7.9 Wavefront OBJ export \(page 497\)](#).
 - "Export/Web3D X3D", export a scene to the XML based X3D format published by the Web3D Consortium, see section [7.15 X3D \(Web3D\) Export \(page 509\)](#) for more information.

¹ Since 1.13.

- "Load Plugin", loads a file containing a custom object or a plugin. Depending on the platform Ayam is running on, these are files with the file name extension ".so" or ".dll".

If loading of a plugin fails, further attempts may also fail, even if the cause of the initial failure is eliminated. Just restart Ayam.

See section 8.7 Plugins Overview (page 528) for more information regarding Ayam plugins.

- "Save Prefs", save the current preference settings to the *ayamrc* file after making a backup copy of this file (see section 8.4 Ayamrc File (page 516) for more information about this file).
- "1.", "2.", "3.", "4.", immediately replace the current scene with the one in the menu entry. The menu entries are updated and rotated upon successful loading and saving of a scene so that the first entry always contains the scene that was loaded (or saved) last.

The standard keyboard shortcuts for this operation are <Ctrl+1> to <Ctrl+4>.

- "Exit!", remove all temporary files, save preferences (if the preference setting "Main/AutoSavePrefs" is turned on) and quit the application.

The standard keyboard shortcut for this operation is <Ctrl+q>.

The "Edit" menu contains object and property clipboard operations, selection and undo actions, hosts the object search, and lets you open the preferences editor:

- "Copy", copies the currently selected object(s) into the clipboard.
This is a deep copy, reference counters of master or material objects may be increased and there are special rules for instances, see also section 4.2.7 [Instances and the Object Clipboard \(page 132\)](#).
This operation cannot be undone!
The standard keyboard shortcut for this operation is <Ctrl+c>.
- "Cut", moves the currently selected object(s) into the clipboard.
This operation cannot be undone!
The standard keyboard shortcut for this operation is <Ctrl+x>.
- "Paste", copies the object(s) from the clipboard to the current level of the scene.
This operation cannot be undone!
The standard keyboard shortcut for this operation is <Ctrl+v>.
Note that the content of the clipboard remains intact after this operation, this means that the operation can be used multiple times. Objects can be moved out of the clipboard (clearing it) using the menu entry "Special/Clipboard/Paste (Move)".
Also note that referenced objects, when moved into the clipboard with "Cut", can not be moved out of it using a simple "Paste", use "Special/Clipboard/Paste (Move)" instead! See also section 4.2.7 [Instances and the Object Clipboard \(page 132\)](#).
- "Delete", removes the selected object(s) and their children from the scene.
This operation may fail for referenced objects or for parent objects with referenced objects among their children. In case of failure, an error will be reported and the undeletable object(s) will be moved to the end of the current level.
This operation cannot be undone!
- "Select All", selects all objects in the current level (except for the Root object).
The standard keyboard shortcut for this operation is <Ctrl+a>.
- "Select None", de-selects all currently selected objects.
The standard keyboard shortcut for this operation is <Ctrl+n>.
- "Copy Property", copies the currently selected property of the currently selected object to the property clipboard (the property clipboard is completely independent from the normal object clipboard). Marked parameters will be omitted!
The standard keyboard shortcut for this operation is <Ctrl+C>.
See also section 2.1.2 [Property Clipboard \(page 25\)](#) for more information.
- "Copy Marked Prop", copies the currently marked parameters of the currently selected property of the currently selected object to the property clipboard (the property clipboard is completely independent from the normal object clipboard).
The standard keyboard shortcut for this operation is <Ctrl+I>.
- "Paste Property", copies all property data from the property clipboard to the currently selected object(s). The data will get pasted to the property type saved by the last copy operation. It will not get pasted to the currently selected property; use "Special/Clipboard/Paste Property to selected" for that.
The standard keyboard shortcut for this operation is <Ctrl+V>.

- "Undo", perform the undo operation (see section 8.1 The Undo System (page 511) for more information).

The standard keyboard shortcut for this operation is <Ctrl+z>.

- "Redo", perform the redo operation (see section 8.1 The Undo System (page 511) for more information).

The standard keyboard shortcut for this operation is <Ctrl+y>.

- "Material", searches for the material object currently associated with the selected object and selects it for editing. If the selected object has no material yet, a new material will be created: Ayam will prompt for the name of the new material, the material object will be created, if successful, the material will be linked to *all* currently selected objects (even if they are already linked to other materials), see also section 4.2.5 Material Object (page 125).

The standard keyboard shortcut for this operation is <Ctrl+m>.

- "Master", searches for the master object of the currently selected instance object and selects it for editing, see also section 4.2.7 Instance Object (page 131).

The standard keyboard shortcut for this operation is <Ctrl+M>.

- "Search", opens the object search dialog, see also section 2.9 Object Search (page 54).

The standard keyboard shortcut for this operation is <Ctrl+f>.

- "Preferences", opens the preferences dialog (see section 2.10 Preferences (page 58) for more information).

The standard keyboard shortcut for this operation is <Ctrl+p>.

The "Create" menu entries let you create objects. In contrast to the object creation via the toolbox some menu entries open small dialogs, where parameters for the object to be created may be adjusted. The entry fields in those dialogs support Tcl expressions as detailed in section 6.3 Expression Support in Dialog Entries (page 439). Here are the entries of the Create menu:

- "NURBCurve", create a new NURBS curve. A small dialog box will pop up, where the length of the new curve may be specified. See also section 4.4.1 NCurve Object (page 150). This dialog also contains a "AddArgs" entry field where additional command line arguments to the "crtOb NCurve" command may be specified, as outlined in section 6.2.2 Creating Objects (page 351).¹
- "ICurve", create a new interpolating curve. A small dialog box will pop up, where the length of the new curve may be specified. See also section 4.4.2 ICurve Object (page 155). This dialog also contains a "AddArgs" entry field where additional command line arguments to the "crtOb ICurve" command may be specified, as outlined in section 6.2.2 Creating Objects (page 353).²
- "ACurve", create a new approximating curve. A small dialog box will pop up, where the length of the new curve may be specified. See also section 4.4.3 ACurve Object (page 158). This dialog also contains a "AddArgs" entry field where additional command line arguments to the "crtOb ACurve" command may be specified, as outlined in section 6.2.2 Creating Objects (page 355).³
- "NCircle", create a new NURBS circle (NCircle object). A small dialog box will pop up, where the radius, start angle, and end angle of the new circle may be specified. See also section 4.4.4 NCircle Object (page 160).

For the other entries see section 5.2 Curve Creation Tools (page 274).

- "NURBPatch", create a new NURBS patch. A small dialog box will pop up, where the width and height of the new patch may be specified. See also section 4.6.1 NPatch Object (page 170). This dialog also contains a "AddArgs" entry field where additional command line arguments to the "crtOb NPatch" command may be specified, as outlined in section 6.2.2 Creating Objects (page 356).⁴
- "IPatch", create a new interpolating patch.⁵ A small dialog box will pop up, where the width and height of the new patch may be specified. See also section 4.6.2 IPatch Object (page 174). This dialog also contains a "AddArgs" entry field where additional command line arguments to the "crtOb IPatch" command may be specified, as outlined in section 6.2.2 Creating Objects (page 358).
- "APatch", create a new approximating patch.⁶ A small dialog box will pop up, where the width and height of the new patch may be specified. See also section 4.6.3 APatch Object (page 176). This dialog also contains a "AddArgs" entry field where additional command line arguments to the "crtOb APatch" command may be specified, as outlined in section 6.2.2 Creating Objects (page 360).
- "BPatch", create a new bilinear patch. See also section 4.6.4 BPatch Object (page 178).
- "PatchMesh", create a new patch mesh. A small dialog box will pop up, where the width and height of the new patch mesh may be specified. See also section 4.6.5 PatchMesh Object (page 179). This dialog also contains a "AddArgs" entry field where additional command line arguments to the "crtOb PatchMesh" command may be specified, as outlined in section 6.2.2 Creating Objects (page 362).⁷

For the other entries see section 5.4 Surface Creation Tools (page 308).

- "Solid", create a new solid primitive object, for use in CSG. "Box", "Sphere", "Disk", "Cone", "Cylinder", "Torus", "Hyperboloid" or "Paraboloid" may be selected.

¹ Since 1.19. ² Since 1.19. ³ Since 1.19. ⁴ Since 1.19. ⁵ Since 1.20. ⁶ Since 1.30. ⁷ Since 1.25.

- "Level", creates a new hierarchy object. "Level" just groups objects, "Union", "Intersection", "Difference", and "Primitive" are CSG operations. See also section [4.2.6 Level Object \(page 128\)](#).
- "Light", create a new light source. See also section [4.2.4 Light Object \(page 119\)](#).
- "Custom Object", create a new custom object. If this sub-menu is empty no custom object has been loaded yet. See also section [4.9.2 Custom Object \(page 238\)](#).
- "View", a new View window will be opened. See also section [4.2.2 View Object \(page 116\)](#).
- "Instance", create an instance of the currently selected object, see section [4.2.7 Instance Object \(page 131\)](#) for more information regarding instances.
- "Clone", create a Clone object, see section [4.2.8 Clone Object \(page 134\)](#).
- "Mirror", create a Mirror object, see section [4.2.9 Mirror Object \(page 137\)](#).
- "Material", create a new material. A small dialog box will pop up, where the name of the new material must be specified. See also section [4.2.5 Material Object \(page 125\)](#).
- "Camera", create a new camera. Camera objects may be used to temporarily save view camera settings, see section [4.2.3 Camera Object \(page 118\)](#).
- "RiInc", create a new RIB-include object. Those objects may be used to include objects into your scenes that just exist as a piece of RIB, see also section [4.2.11 RiInc Object \(page 140\)](#).
- "RiProc", create a new RI procedural object, see also section [4.2.12 RiProc Object \(page 141\)](#).
- "Script", create a new Script object, see also section [4.9.1 Script Object \(page 229\)](#).
- "Select", create a new Select object, see also section [4.2.10 Select Object \(page 139\)](#).
- "Text", create a new Text object, see also section [4.7.14 Text Object \(page 221\)](#).

The "Tools" menu hosts modelling tools to create complex objects or modify existing objects. Some tools open dialog windows to request parameters. The entry fields in those dialogs support Tcl expressions as detailed in section 6.3 Expression Support in Dialog Entries (page 439). The entries of the "Tools" menu are:

- "Last (None)", this menu entry allows quick access to the last used entry/tool in the "Tools" menu hierarchy.¹ The label of the entry will be changed appropriately when a tool was started, e.g. to "Last (Revert U)" after the "Tools/Surface/Revert U" menu entry/tool was used. The corresponding keyboard shortcut is <Ctrl+t>. To repeat the last used tool with the same set of parameters (and without opening the parameter dialog window again) the shortcut <Ctrl+T> can be used instead.
- "Create", "Curve", and "Surface", are sub-menus with various NURBS based creation and modelling tools, that are explained in depth in section 5 NURBS Modelling Tools (page 273).
- "PolyMesh": sub-menu for polygonal mesh related tools:
 - "Merge": merges all currently selected PolyMesh objects into a single PolyMesh object, without checking for doubly used points, loops, or faces. The currently selected PolyMesh objects will not be changed by this tool. But the merge-tool can delete them immediately after the merging operation, when the "RemoveMerged"-option is enabled. If the merge fails, the original objects will not be removed.
If the "OptimizeNew"-option is enabled, the "Optimize"-tool (see below) will be started right after the merge operation with the newly created merged object as argument.
The option "MergePVTags" controls whether the merge tool should also merge all PV tags.
 - "Split": splits the selected faces from the selected PolyMesh objects off and into a second PolyMesh object. To select faces see also section 3.25 Selecting Faces (page 104).
The original selected PolyMesh objects will be changed, the selected faces will be removed.
Since the split operation does not create optimized new objects, the "Optimize"-tool (see below) may be started immediately after splitting using the "OptimizeNew"-option.
 - "Optimize": optimizes the selected PolyMesh object(s) by removing all multiply used (and unused) control points (if the option "OptimizeCoords" is enabled) or multiply used faces (not implemented yet).
The option "NormalEpsilon" additionally controls which points are considered equal by comparing also the vertex normals (if present):
A value of "0.0" means, the normals must be bitwise identical, a value of "Inf" means, the normals are totally ignored, any other value defines the maximum angle (in degrees) between two normals to be considered equal.²
The option "OptimizePV" determines whether the PV tags should also be optimized.³
If "OptimizeSelected" is enabled, only the selected points of the PolyMesh are processed.
Removing multiply used control points using the "Optimize"-tool may decrease the memory consumption of the control points by a factor of about six, depending on the connectivity of the original mesh.
 - "Connect": connects the first two selected PolyMesh objects via their selected boundaries. This tool can e.g. be used to close gaps between tessellated NURBS surfaces that are incompatible to achieve a watertight tessellation.
All control points of the respective boundary must be selected. The select boundary points modelling action can be used to easily select all those points interactively (see also section 3.24 Selecting Boundary Points (page 103)). The PolyMesh objects must be optimized at least in the direct neighborhood of the boundary (all control points used by the faces on the boundary).

¹ Since 1.13. ² Since 1.23. ³ Since 1.23.

The boundary edges are first offset into the interior of the respective mesh along the surface tangent, then a strip of new triangles will be generated. The new triangles will be put into a new (third) PolyMesh object.

The offset values can be adjusted; they are interpreted relative to the shortest edge incident to each control point to be moved. Note that too large offset values may lead to degenerate triangles/meshes for boundary faces with unfavorable shape.

- "Quadrangulate": refines a purely triangular PolyMesh by replacing each triangle with three quads.
- "Gen. Face Normals": Generates face normals for the selected PolyMesh object(s) using the robust Newell algorithm. The generated normals will be stored in a PV tag.
- "Gen. Smooth Normals": Generates smooth vertex normals for the selected PolyMesh object(s), averaging the surrounding face normals of each vertex. The face normals will be weighted by the vertex face-centroid distance, which takes both, face area and face shape, into account. Vertices of hole loops will just get the respective face normal.
Already existing vertex normals will be destroyed.
If face normals exist, they will be used, otherwise, new face normals will be generated automatically using the same algorithm as implemented in the "Gen. Face Normals" tool above.
- "Rem. Smooth Normals": Remove all smooth vertex normals from the selected PolyMesh object(s). Afterwards, the objects will be shaded in a faceted look.
- "Flip Smooth Normals": Flips/reverts all smooth vertex normals of the selected PolyMesh object(s).
- "Flip Loops": Flips/reverts all loops of the selected PolyMesh object(s).
- "Points": sub-menu for tools that work on points:
 - "Select All Points", selects all points of the currently selected object(s).
 - "Deselect All Points", de-selects all points of the currently selected object(s).
 - "Invert Selection", selects all points of the currently selected object(s) that are not selected, and de-select all points that are currently selected.
 - "Collapse Points", collapses all currently selected points to one, see also [5.3.29 Collapse Tool \(page 307\)](#).
 - "Explode Points", explodes all currently selected multiple points, see also [5.3.30 Explode Tool \(page 307\)](#).
 - "Apply To All", applies the transformations encoded in the transformations property of the selected objects to all points of those objects. This will have the effect of resetting the transformations property to the default values without (visibly) changing the points of the selected objects.
 - "Apply To Selected", applies the transformations encoded in the transformations property of the selected objects to the selected points. This will reset the transformations property without (visibly) changing the selected points. The points currently not selected will be transformed, however!
 - "Center All Points (3D)", moves all points of the selected objects so that their common center (the center of gravity) is the center of the respective objects coordinate system. Note that, currently, this works on each of the selected objects separately!
 - "Center All Points (2D-XY)", "Center All Points (2D-YZ)", "Center All Points (2D-XZ)": work like the center 3D tool but just center in the designated plane.
- "Show", "Hide" set and unset the "Hide" attribute of the selected object(s) thus making them invisible or visible again. Note that hidden objects may be excluded from RIB-Export, when the preference setting "RIB-Export/ExcludeHidden" is activated.

- "Show All" and "Hide All" set and unset the "Hide" attribute of all objects in the scene (including the Root object and all views!) regardless of the currently selected objects (and without changing the current selection). These operations can not be undone using the undo system.
- "Convert", starts the convert action that has been registered for the type of the selected object(s). The exact behaviour depends on the type of the selected object(s): a Revolve object will e.g. be converted to a level containing NURBS patches that make up the surface of revolution and the caps. This operation can not be undone, i.e. the newly created objects will not be removed, using the undo system. Potentially present "TP" and "TC" tags will be preserved. This can be configured via the hidden preference setting "ConvertTags", see section 8.4.2 [Hidden Preference Settings](#) (page 518) for more information.
- "Convert (In Place)", starts the convert action as outlined above, but replaces the original objects with the new converted ones. This operation, in contrast to the simple conversion above, can be undone.
- "Force Notification", force the notification callbacks of all selected objects (or all objects in the scene if no objects are selected) to be called. The notification callbacks are used by objects like e.g. Revolve to be informed about changes of their child objects to properly adapt to those changes (for more information on notification, see also section 8.2 [The Modelling Concept Tool-Objects](#) (page 511)).

The "Custom" menu is initially empty. Custom objects and plugins may create entries here.

The "Special" menu contains seldom used tools:

- "Save Selected as", saves just the currently selected objects to a scene file. Note that Ayam will not check, whether the objects are saved with their materials. It is also possible to save instance objects without their master objects. This will lead to errors while loading such a scene later on.
- "Save Environment", saves the Root object and all views to a so called environment scene file, which is read automatically on program startup and when the scene is reset using the main menu entry "File/New". Initially, the file requester that asks for the name of the new environment uses the value of the preference setting "Main/EnvFile". Note that there will be no check whether loading of that environment on next start up is enabled in the preferences. Note also, that the files saved using "Save Environment" just contain the Root object and all views. In order to include geometric objects in the environment or to exclude the Root object and just save view objects use "File/Save" or "Special/Save Selected as" respectively.
- "Clipboard/Paste (Move)", moves objects from the clipboard back to the scene (clearing the clipboard). This is the only way to get referenced objects out of the clipboard. This operation may fail, if the result is an illegal instance configuration.
- "Clipboard/Replace", replaces the currently selected object(s) with the object clipboard content, moving the replaced objects into the clipboard. If multiple objects are selected in non consecutive sequences, only the first consecutive sequence or single object is replaced. This operation may fail, if the result is an illegal instance configuration.
- "Clipboard/Copy (Add)", copies the selected objects to the clipboard without cleaning it beforehand.
- "Clipboard/Cut (Add)", moves the selected objects to the clipboard without cleaning it beforehand.
- "Clipboard/Clear", clears the object clipboard. This operation may fail if there are referenced objects (Material objects or master objects of instances) in the clipboard.
- "Clipboard/Paste Property to Selected" paste the property from the property clipboard to the currently selected property of the currently selected object. No type check of the properties will take place! This way it is possible to e.g. copy a radius value from a sphere to a cylinder or to copy settings from a displacement shader to a surface shader (as long as the copied arguments of both shaders have the same names and types).
- "Instances/Resolve all Instances", converts all instance objects to copies of their respective master objects. If the menu entry is used, a small dialog opens where the scope of the resolve operation may be adjusted and the operation may be started¹.
In contrast to converting the selected Instance objects via e.g. the menu entry "Tools/Convert", this operation does not stop until no Instances are left in the processed object hierarchies.
- "Instances/Automatic Instancing", opens a small dialog, where the automatic instantiation may be parameterised and started (this algorithm automatically creates instances from equal objects). See section 8.10 Automatic Instancing (page 531) for more information.

¹ Since 1.21.

- "Tags/Add RiOption", pops up a small dialog box, where a RiOption may be selected, parameterised, and added as tag to the Root object (see [4.11.2 RiOption Tag \(page 260\)](#)). The Root object does not have to be selected and the current selection will not be changed by this action.
The database of available RiOptions is controlled by the *ayamrc* file (see also section [8.4.3 RiOption and RiAttributes Database \(page 525\)](#)).
- "Tags/Add RiAttribute", pops up a small dialog box, where a RiAttribute may be selected, parameterised, and added as tag to the currently selected object(s) (see [4.11.1 RiAttribute Tag \(page 259\)](#)).
The database of available RiAttributes is controlled by the *ayamrc* file (see also section [8.4.3 RiOption and RiAttributes Database \(page 525\)](#)).
- "Tags/Edit TexCoords", opens the texture coordinates editor. (see also section [4.11.3 TC \(Texture Coordinates\) Tag \(page 260\)](#)).
- "Tags/Add Property", opens a dialog where "NP" tags can be managed easily for all selected objects (see also section [4.11.13 NP \(New Property\) Tag \(page 266\)](#)).
- "Tags/Remove Property", opens a dialog where "RP" tags can be managed easily for all selected objects (see also section [4.11.14 RP \(Remove Property\) Tag \(page 266\)](#)).
- "Tags/Add Tag", opens a dialog where a new tag can be added to the selected objects (see also section [4.11 Tags \(page 258\)](#)).
- "RIB-Export/From Camera", writes a complete RIB of the current scene with the camera transformations taken from the currently selected Camera object. The dimensions of the rendered image will be taken from the RiOptions of the Root object. If those values are zero, default values of 400 pixels width and 300 pixels height will be used. The type of the projection written will be *perspective*. Otherwise the RIB looks exactly the same as if exported via main menu "File/Export/RenderMan RIB".
- "RIB-Export/Selected Objects", exports only the selected objects to a RIB. Note that instances will always be resolved, hidden objects and objects with "NoExport" tags are treated as on normal export operations, and light objects are simply ignored. Note also that the created RIB, since it e.g. lacks camera transformation and WorldBegin/End directives, may not be rendered directly by a RenderMan compliant renderer (unless the renderer is really forgiving about mis-structured RIBs). The main use of this feature is to aid in the creation of RIBs that may be easily included in other RIBs using e.g. the ReadArchive directive. See also section [4.2.11 RiInc Object \(page 140\)](#).
- "RIB-Export/Create All ShadowMaps", for each light in the scene, the light is selected, then the shadow map will be created in the same way as with the menu entry "Create ShadowMap" below (i.e. this creates all shadow maps for the current scene regardless of selection). See also section [4.2.4 Using ShadowMaps \(page 121\)](#).
- "RIB-Export/Create ShadowMap", the scene is exported to a RIB file using the currently selected light source for the camera settings and with a RiDisplay statement that results in a shadow map to be produced when rendered; then the "SMRender" command will be called. See also sections [2.10.4 RIB-Export Preferences \(page 66\)](#) and [4.2.4 Using ShadowMaps \(page 121\)](#).
Note that before doing anything, Ayam will check whether shadow maps are enabled in the RIB export preferences and offer to adapt the preference setting accordingly. This must be done because *only then* exported RIBs for image rendering will actually use the shadow maps.
- "Enable Scripts" enables all disabled script tags and objects. Objects and tags in the object clipboard are not affected!

- "Select Renderer" opens a dialog where the renderer for direct rendering from a view may be chosen. The changes will have effect on all preference settings that control direct rendering from a view, except whether RenderGUIs should be used. If the "ScanShaders" checkmark is activated, Ayam will additionally try to load the corresponding shader parsing plugin (see also section 8.8 Shader Parsing Plugins (page 529)) and rescan for compiled shaders. Note that in order for the "ScanShaders" feature to work properly the "Main/Shaders" and "Main/Plugins" preference settings have to be set correctly (see also section 2.10.1 Main Preferences (page 59)).
- "Scan Shaders" initiates the shader parsing with the built in shader parser or the currently loaded shader parsing plugin (see also section 8.8 Shader Parsing Plugins (page 529)).
- "Reset Preferences" removes the current *ayamrc* file, where the preferences are saved; after a restart of Ayam, all preferences will be reset to factory defaults. See also section 8.4 Ayamrc File (page 516) for more information about the *ayamrc* file.
- "Reset Layout" resets the pane layout in single window GUI mode so that the upper internal views get an approximately even share of the available horizontal screen space and the object selection widget and property GUI are completely visible.
- "Toggle Toolbox" closes or opens the toolbox window (see 2.8 The Toolbox Window (page 53)). The state of the toolbox is also remembered in the saved preferences.¹ This option is not available if Ayam is in single window GUI mode.
- "Toggle TreeView" toggles between object tree view and object listbox (see also section 2.1.1 Objects (page 19) for more information about both representations). This state is also remembered in the saved preferences.²
- "Zap Ayam" iconifies all currently open windows of Ayam. If one of the iconified windows is de-iconified later, all other zapped windows will be de-iconified as well.

The "Help" menu:

- "Help", opens a web browser and displays the documentation, the URL to display is taken from the "Docs" preference setting. See also section 2.10.1 Main Preferences (page 59).
The standard keyboard shortcut for this operation is <F1>.
- "Help on object", opens a web browser and displays documentation about the currently selected type of object, the URL to display is derived from the "Docs" preference setting.
- "Help on property", opens a web browser and displays documentation about the currently selected property, the URL is derived in the same way as for the "Help on object" entry above.
- "Show Shortcuts", displays some important shortcuts for modelling actions, you may leave this window open when doing your first steps in modelling with Ayam.
- "About", displays some version, copyright, and trademark information.
- "Show Tooltips", enables tool tips (balloon help) for various user interface elements (including the toolbox buttons).

¹ Since 1.3. ² Since 1.3.

2.3 Main Window Keyboard Shortcuts

An important group of shortcuts is available on the function keys:

<F1> has already been mentioned, it opens a web browser and displays the URL from the "Docs" preference setting.

<F2> and <F3> lower and raise the global GLU sampling tolerance value respectively, allowing fast adjustment of the NURBS drawing/shading quality.

<F4> toggles between display of NURBS control cage and true curves / surface outlines.

<F5> rebuilds the object tree and issues a complete notification, then all views (even those with disabled AutoDraw¹) will be drawn. It is therefore helpful to update the complete GUI after changes made to the scene using the scripting interface in the console.

<F6> toggles lazy notification.

<Ctrl+A> is bound to the "Apply" and <Ctrl+R> to the "Reset" button of the property GUI.

The object selection can be manipulated by the cursor keys, see also section 2.1.1 [object tree shortcuts](#) (page 21) and section 2.1.1 [object list shortcuts](#) (page 22).

The whole application with all open windows may be iconified (zapped) using the shortcut <Ctrl+Z>. If any of the windows iconified by zap is de-iconified, all other windows iconified by zap will be de-iconified as well.

Many main menu entries have direct keyboard shortcuts, displayed directly in the menu entries, see also section 2.2 [Main Menu](#) (page 28).

Note that the main window keyboard shortcuts only work if the main window has the keyboard input focus *and* the input focus is not in the console or in a property GUI element (i.e. a data entry field). In doubt, hit <Esc> first to set the focus to the object selection widget and thus enable the main window keyboard shortcuts.

All these shortcuts can be adapted using the *ayamrc* file (see section 8.4 [Ayamrc File](#) (page 516)).

¹ Since 1.26.

2.4 Anatomy of a View

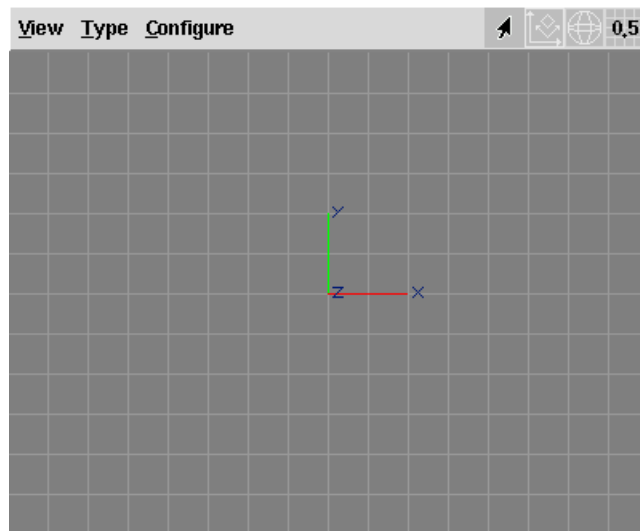


Figure 8: A View Window

The view window is split into a menu bar and a OpenGL-widget, where interaction and drawing takes place. The title of the view window gives information about name, current type, and the currently active modelling action of the view.

The current modelling action, modelling mode, drawing mode and grid size are also displayed as a set of icons on the right hand side of the view menu bar.

2.5 View Menu

This section discusses the view menu bar.

Note that many menu entries have keyboard shortcuts that are displayed in each entry. But those shortcuts only work if the view window has the keyboard input focus. The shortcuts are adaptable using the *ayamrc* file (see section 8.4 [Ayamrc File \(page 516\)](#)).

Another way of navigating the menu is via the "Menu" or "Alt" key, pressed together with the various underlined characters in the menu entries.

Here are all entries of the "View" menu:

- "Quick Render": the scene is exported to a RIB file using the camera settings of the current view; then the "QRender" command will be called. See also section 2.10.4 [RIB-Export Preferences \(page 66\)](#). As the renderer will be executed in the background, Ayam will remain responsive and it is also possible to run multiple renderers in parallel.

Note that the RIB export will *not* use the `RiOption` settings for image size from the Root object but the current view window size instead. Also note that the environment variable `SHADERS` will be adapted to the current value of the preference setting `Shaders` for rendering.

- "Render": the scene is exported to a RIB file using the camera settings of the current view; then the "Render" command will be called. See also section 2.10.4 [RIB-Export Preferences \(page 66\)](#). As the renderer will be executed in the background, Ayam will remain responsive and it is also possible to run multiple renderers in parallel.

Note that the RIB export will *not* use the `RiOption` settings for image size from the Root object but the current view window size instead. Also note that the environment variable `SHADERS` will be adapted to the current value of the preference setting `Shaders` for rendering.

- "Render to File": the scene is exported to a RIB file using the camera settings of the current view and with a `RiDisplay` statement that results in an image file to be produced when rendered; then the "FRender" command will be called. See also section 2.10.4 RIB-Export Preferences (page 66). As the renderer will be executed in the background, Ayam will remain responsive and it is also possible to run multiple renderers in parallel.

The image file name will be derived from the scene file name. Note that the RIB export will *not* use the `RiOption` settings for image size from the Root object but the current view window size instead. Also note that the environment variable `SHADERS` will be adapted to the current value of preference setting `Shaders` for rendering.

- "Export RIB" exports the scene to a RIB file. This does exactly the same as the main menu entry "File/Export/RenderMan RIB", except that the current view will already be selected in the dialog box.
- "Redraw": forces the view window to be drawn, this is particularly useful if automatic redrawing of the view has been disabled.
- "Reset": restores the camera settings of the view to those natural to the current view type.
- "Open PPrev", "Close PPrev": those menu entries are just available, if the compile time option `AYENABLEPPREV` has been set. This option is *not* set for the official Ayam binaries. Permanent preview (PPrev) continuously writes a RIB stream to a (fast) RenderMan renderer, a frame for each redraw operation of the view window that was used to open the preview. This way, the RenderMan renderer immediately displays all changes in the scene. This is a great way to test many different camera or light settings without the need to manually start a rendering process and close the preview window for each different setting. As the RIB client library usually is not able to handle multiple open RIB streams simultaneously, RIB-Export and direct rendering from view windows are not available until the permanent preview window is closed.
- "Create All ShadowMaps": for each light in the scene, the light is selected, then the shadow map will be created in the same way as with the menu entry "Create ShadowMap" below (i.e. this creates all shadow maps for the current scene regardless of selection). See also section 4.2.4 Using ShadowMaps (page 121).
- "Create ShadowMap": the scene is exported to a RIB file using the currently selected light source for the camera settings and with a `RiDisplay` statement that results in a shadow map to be produced when rendered; then the "SMRender" command will be called. See also sections 2.10.4 RIB-Export Preferences (page 66) and 4.2.4 Using ShadowMaps (page 121).

Note that before doing anything, Ayam will check whether shadow maps are enabled in the RIB export preferences and offer to adapt the preference setting accordingly. This must be done because *only then* exported RIBs for image rendering will actually use the shadow maps.

- "Close": the View window will be removed. This entry is not available for the internal views of Ayam in single window GUI mode.

The "Type" menu entries:

- "Front"
- "Side"
- "Top"
- "Perspective"
- "Trim"

may be used to change the type of the view, which restrains the scope of certain modelling actions. See sections [4.2.2 View Object \(page 116\)](#), [2.6 View Window Shortcuts and Actions \(page 48\)](#), and [3 Modelling Actions \(page 75\)](#) for more information.

The "Configure" menu may be used to change preferences of the view. Some preferences are outlined in greater detail in section [4.2.2 ViewAttrib \(page 117\)](#).

- "Automatic Redraw", toggles whether the view should be redrawn, whenever the scene changes. If this is disabled, a redraw can be enforced using the menu entry "View/Redraw" or the corresponding keyboard shortcut <Ctrl+d>.
- "Drawing Mode" determines whether the view should draw a wire-frame representation ("Drawing Mode/Draw"), a shaded one ("Drawing Mode/Shade"), a representation where the wires of the draw mode are drawn over the shaded representation ("Drawing Mode/ShadeAndDraw"), or hidden wires and silhouettes ("Drawing Mode/HiddenWire"). See also section [2.7 Drawing Modes \(page 50\)](#).
- "Modelling Mode" allows to switch the modelling coordinate system from world space ("Global") to the space defined by the current parent object ("Local (Level) ") or even the space defined by the currently selected object ("Local (Object) "). See also section [3.26 Editing in Local Spaces \(page 105\)](#).
- "Draw Selection only", if this is enabled, just the currently selected objects (and their children) will be drawn.
- "Draw Level only", if this is enabled, just the objects of the current level (and their children) will be drawn.
- "Draw Object CS", if this is enabled, small coordinate systems (three colored lines) will be drawn at the base of each objects coordinate system.
- "AntiAlias Lines", if this is enabled, all lines will be anti-aliased (smoothed).
- "Draw BGImage", if this is enabled, the background image will be drawn.
- "Set BGImage", may be used to set the current background image of the view, which should be a TIFF file. This image may also be specified using the view attribute BGImage.
- "Draw Grid", if this is enabled the grid will be drawn.
- "Use Grid", if this is enabled the grid will be used to constrain modelling actions to grid coordinates. See also section [3 Modelling Actions \(page 75\)](#).
- "Set Gridsize", may be used to change the size of the grid associated with this view. Another way to change the grid size is to use the grid icon menu on the rightmost side, see below.
- "Half Size", change width and height to the half of the current values. This is not available for the internal views in single window GUI mode.

- "Double Size", change width and height to the double of the current values. This is not available for the internal views in single window GUI mode.
- "From Camera", copy camera settings from the currently selected camera object to the view.
- "To Camera", copy camera settings to the currently selected camera object from the view.
- "Set FOV", lets you specify a field of view value for the view, and adapts the zoom accordingly. This is just working for perspective views, of course.
- "Zoom to Object", adapt the camera settings, so that the currently selected objects are centered in the view. In addition, clipping planes and light position may be adapted for very small or very large objects.
- "Zoom to All", adapt the camera settings, so that all objects are centered in the view, regardless of current level and selected objects.
- "Align to Object", align the view to the coordinate system of the currently selected object or to the parent object of the current level if no object is currently selected (see also section 3.26 [Editing in Local Spaces](#) (page 105)).

Apart from the text based menus documented above, there are also some icon based menus in the view window menu bar:

The "Action" icon menu hosts basic selection actions.¹ The menu button of this menu also always conveys the currently active modelling action. See also the image below:

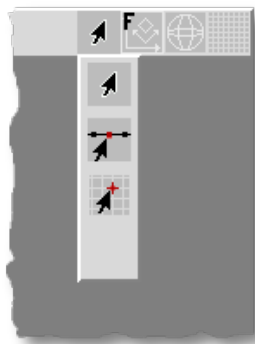


Figure 9: Action Icon Menu

The selection actions are:

- Pick, see section 3.3 [Selecting Objects by Picking](#) (page 78),
- Tag, see section 3.4 [Selecting/Tagging Points](#) (page 79),
- Mark, see section 3.5 [Setting the Mark](#) (page 82),
- None, disables all actions.

¹ Since 1.24.

The action icon menu can be configured using a dialog that opens when the rightmost mouse button is pressed on the menu button or via the "SetActionMenu" entry in the "Misc" preference section.¹ See also the image below.

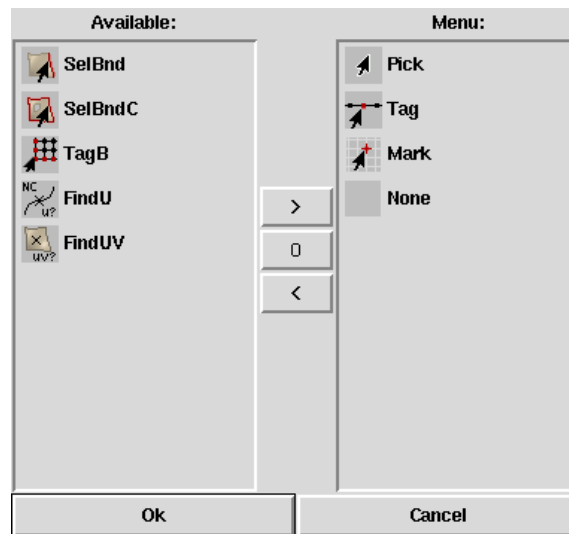


Figure 10: Action Icon Menu Configuration

Selected entries can be moved to the respective other side using the "<" and ">" buttons, the new entry will be inserted above the selected entry on the other side; the "□" button resets to the last saved configuration. The additional selection actions are:

- SelFace, see section 3.25 Selecting Faces (page 104),
- FindU, see section 3.20 Finding Points on Curves (page 99),
- FindUV, see section 3.21 Finding Points on Surfaces (page 100),
- SelBnd, see section 3.22 Selecting Boundary Curves (page 101),
- SelBndC, see section 3.22 Selecting Boundary Curves (page 101), and
- TagB, see section 3.24 Selecting/Tagging Boundary Points (page 103).

The "Modelling Mode" icon menu may be used to quickly change the current modelling mode (global or local, see also section 3.26 Editing in Local Spaces (page 105)). Apart from a different icon, the local modes will display a **L** or **O** in the lower right corner of the icon.

The icon, additionally, conveys whether objects or points would be modified by a modelling action: for points, a red dot will be present in the upper right corner of the icon (see also section 3.2 Transforming Objects or Selected Points (page 77)). The modelling scope may also be toggled directly by clicking on the menu button with the rightmost mouse button.²

Furthermore, the type of the view will be displayed in the upper left corner of the icon as letter **F**, **S**, **T**, or **P** for front, side, top, or perspective views, respectively. Views of type Trim get no designating letter in the icon. See also the image below:

¹ Since 1.29. ² Since 1.26.



Figure 11: Modelling Mode Icon Menu

The "Drawing Mode" icon menu may be used to quickly change the current drawing mode, drawing, shading, drawing and shading, or hidden wires. See also the image below:

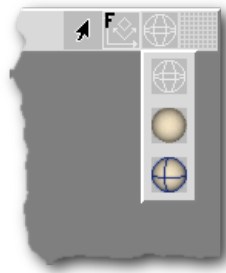


Figure 12: Drawing Mode Icon Menu

See section 2.7 [Drawing Modes \(page 50\)](#) for a discussion of the different drawing modes.

Finally, the "Grid" icon menu may be used to quickly change the current grid size:

On the right hand side in the view menu bar there is a little icon that displays the current grid size. You may click on the icon to display a menu with predefined grid size values. Choosing one of the values 0.1, 0.25, 0.5, or 1.0 will set the grid size of the view to the chosen value and will additionally enable drawing of the grid and snapping to the grid. The entry "X" allows to set a custom grid value. The last entry will set the grid size to 0.0 and disable drawing of and snapping to the grid. If a grid size other than 0.1, 0.25, 0.5, or 1.0 is in effect for the view, a generic icon (with a X instead of a number) will be displayed in the icon menu. See also the image below:

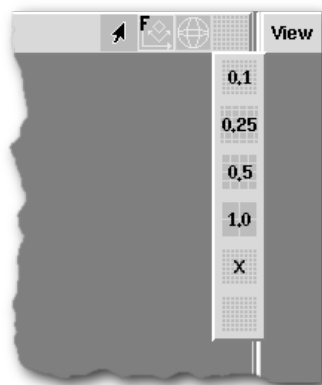


Figure 13: Grid Icon Menu

2.6 View Window Shortcuts and Actions

Important keyboard commands of a view window (aside from the view menu shortcuts) are discussed in this section.

Note that the view keyboard shortcuts only work if the view window has the keyboard input focus.

These shortcuts can be adapted using the *ayamrc* file (see section 8.4 *Ayamrc File* (page 516)).

Keyboard shortcuts directly modifying the camera, that is associated with the view window, are:

- <Left>, <Up>, <Right>, <Down> rotate viewer around the current camera aim point.
- <Shift+Left>, <Shift+Up>, <Shift+Right>, <Shift+Down> pan the view.
- <+>, <->, or <Add>, <Sub> (on the numeric keypad) zoom the view.
- <Ctrl+5> (on the numeric keypad!) resets the views camera (From and To) to the view type dependent default values.
- <. > pans the view to the mark.

See section 6.5.15 *Revert Cursor Key Behavior* (page 452) for a script that swaps the rotate and pan cursor key bindings in parallel views.

Interactive actions modifying the camera, that is associated with the view window, are:

- Using <v> you may move the view with your mouse.
- Using <V> you move the camera in the direction it is looking. Note that this affects both, from and to setting of the virtual camera. Furthermore, this movement will have no visible effect in parallel views.
- <R> (note the case!) starts rotating the virtual camera around the point it is looking to.
- Rotating the view is also possible in any modelling mode, by holding down the <Alt>-key while dragging the mouse.
- <Z> starts zooming the view. Dragging the mouse up zooms in and dragging the mouse down zooms out.
- Zooming the view into a rectangular region defined through a mouse drag is also possible in any modelling mode, by holding down the <Shift>-key.¹

See also the table below.




Name	Shortcut	Icon
Pan View	<v>	
Zoom View	<Z>	
Rotate View	<R>	

Table 1: Interactive View Actions Overview

You may also pan/move the view by dragging with the rightmost mouse button and zoom the view with the middle mouse button without affecting any other active view or modelling action.

¹ Since 1.7.

If you have a wheel mouse and it is configured to send Mouse4 and Mouse5 button events, Ayam will zoom the view when you turn the wheel.

Similarly the view can be rolled when the mouse wheel is turned with the `<Control>` key held down.¹

`<PgUp>` and `<PgDown>` allow to cycle through the view types.²

`<Ctrl+PgUp>` and `<Ctrl+PgDown>` cycle through the drawing modes.³

`<Insert>` and `<Delete>` cycle through the grid sizes.

`<Shift-Insert>` and `<Shift-Delete>` modify the current grid size.⁴

Using the menu entry "Zoom to Object" or the corresponding shortcut `<BackSpace>` the views camera settings can be changed so that the selected objects will be displayed centered in the view window. This is handy to search for objects or if the user is simply lost in space.

Closely related to the latter is the "Zoom to All" action, bound to the `<Shift-BackSpace>` key. This action adjusts the camera settings so that all objects in the scene will be visible, regardless of current level and object selection.

Note that both, zoom to object and zoom to all, adjust the clipping planes to the extents of the affected objects and also adapt the position of the light source used for shading.⁵ This facilitates working with very large and very tiny objects.

Using the menu entry "Align to Object" or the shortcut `<Ctrl+a>` (`<L>` for internal views) the views camera settings can be changed so that the view is aligned to the coordinate system of the currently selected object. This is handy for modelling in local coordinate systems (e.g. when editing the control points of some planar curve defined in the XY-plane that has been rotated around the Y-axis). See also section 3.26 [Editing in Local Spaces \(page 105\)](#).

It is also possible to move through the scene hierarchy and change the selection directly in view windows:⁶

- `<Ctrl+4>`, `<Ctrl+6>` (on the numeric keypad!) move up and down in the hierarchy respectively, also selecting the parent or first child.
- `<Ctrl+2>`, `<Ctrl+8>` (on the numeric keypad!) select the next or previous object.
- `<Ctrl+Shift+2>`, `<Ctrl+Shift+8>` (on the numeric keypad!) extend the current selection to include the next or previous object.

¹ Since 1.29. ² Since 1.15. ³ Since 1.16. ⁴ Since 1.21. ⁵ Since 1.21. ⁶ Since 1.18.

2.7 Drawing Modes

This section explains the draw modes available in Ayam.

The draw mode "Draw" shows a simple wire-frame representation of the scene, see the following image:

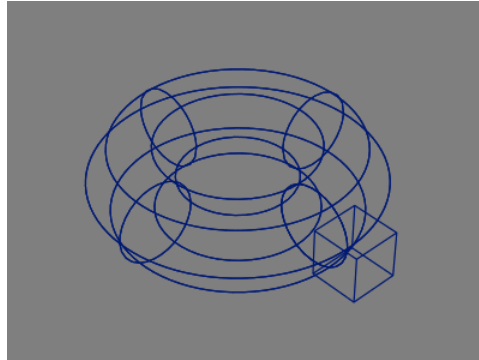


Figure 14: Draw Mode "Draw"

The draw mode "Shade" displays the scene as shaded surface lit by a single headlight, see also the following image:

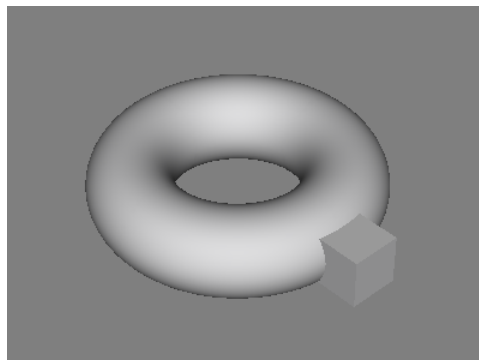


Figure 15: Draw Mode "Shade"

The draw mode "ShadeAndDraw" combines the images of the modes "Shade" and "Draw", see also the following image:

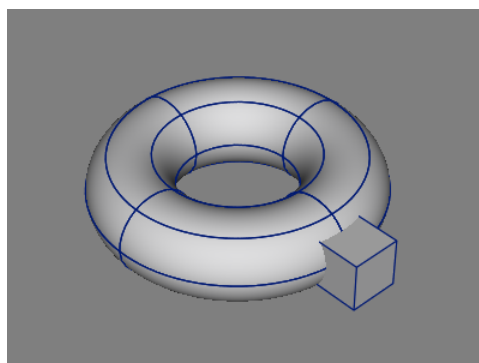


Figure 16: Draw Mode "ShadeAndDraw"

The drawing mode "HiddenWire" works like the shade and draw mode, where the surface shading part is only used to remove hidden bits. In addition, silhouette edges are detected and drawn. For best results, use it with anti-aliasing.

The silhouette detection is shading the scene a second time in a special multi-colored lighting setup. Then, edges are detected in the Z-buffer and color buffer data. The edge detection in the Z-buffer profits from tight-fitting clipping planes. A good clipping plane setup can be easily achieved by the "Zoom to Object" action (shortcut <BackSpace>).

If anti-aliasing is not enabled, the resulting edge map is processed by morphological thinning and removal of pixels in the direct vicinity of pixels already drawn, to get one pixel wide lines.

Finally, the edge map is used as a largely transparent texture (except for the edges) on a full-screen quad, which is drawn as last object. In case of anti-aliasing, the quad will be drawn four times. See also the image below.

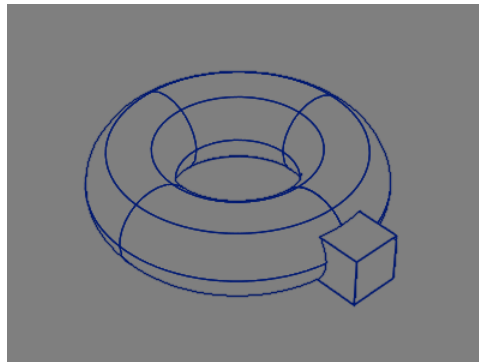


Figure 17: Draw Mode "HiddenWire"

The silhouette detection is a slow process and therefore only performed when no interactive action is in progress. The following image shows what will be drawn during interactive actions, compare it also to the complete image above.

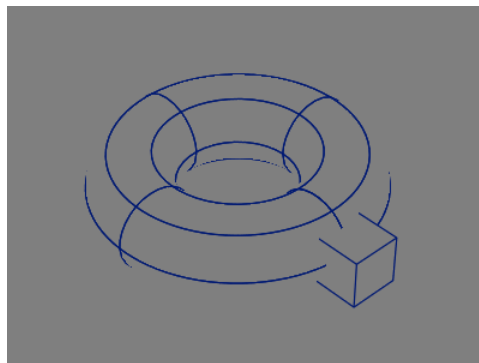


Figure 18: Draw Mode "HiddenWire" without Silhouettes

The drawing mode "Isophotes" allows to assess the surface quality of parametric surfaces by visually analyzing the continuity of lines drawn on said surfaces. This way, discontinuities and irregularities that are not visible in the shaded drawing mode can be detected more easily.

This drawing mode also shows discontinuities in the transitions between adjoining surfaces, see also the example image below, showing a transition between a NURBS torus and a NURBS cylinder that is not curvature continuous.

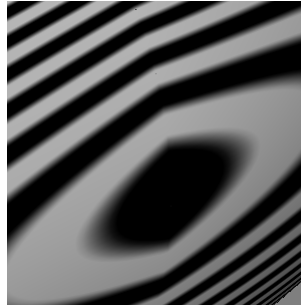


Figure 19: "Isophotes" Example (Transition between NURBS Torus and Cylinder)

2.8 The Toolbox

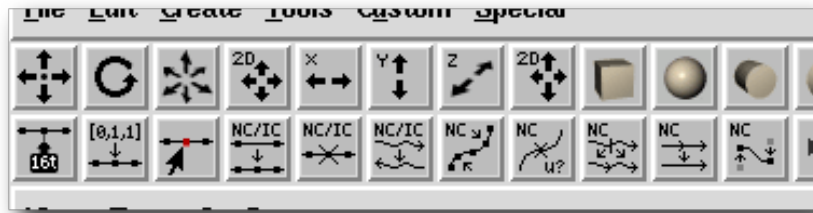


Figure 20: The Toolbox

The toolbox displays some buttons that start interactive modelling actions, modelling tools, or create objects. The toolbox can be opened and closed using the main menu entry "Special/Toggle Toolbox" unless Ayam is in single window GUI mode.

Note that in contrast to the keyboard shortcuts of the view windows, the buttons in the toolbox may switch to the modelling actions for all open views. This is the case if Ayam is in single window GUI mode *and* AutoFocus is enabled or if Ayam is in multi window GUI mode. For more information about the actions see section 3 Modelling Actions (page 75).

Furthermore, the modelling scope may be immediately switched to *point* when starting a transformation action using the rightmost mouse button.¹ See also section 3.2 Transforming Objects or Selected Points (page 77).

When creating objects, holding down the <Ctrl> key while pressing the corresponding toolbox button will keep the objects selected. This is especially useful for objects that would move the currently selected objects to themselves as children (Level, Revolve, Skin etc.).

Several other tool buttons also change their behaviour with modifier keys, check the tooltips.

The toolbox window may be configured by the user using the hidden preference setting "toolBoxList" in the *ayamrc* file. Using this setting you may select from certain groups of buttons and change the order in which they appear in the toolbox window. See section 8.4.2 Hidden Preference Settings (page 523) for more information.

The toolbox is also open for extension by scripts, see section 6.4.4 Script Examples (Toolbox Buttons) (page 444) for examples.

You may also resize the window to change from the vertical standard layout to a horizontal one, optimizing the use of precious screen space. After resizing, the toolbox will re-layout the buttons, warning you if the space is too small for all buttons to display. If the window is too big for the desired layout and the hidden preference setting "ToolBoxShrink" is switched on, the toolbox will shrink wrap the window to match the space occupied by the buttons. Furthermore, using the hidden preference setting "ToolBoxTrans" the toolbox window can be made transient. It will then (depending on the window manager or its configuration) get a different or no decoration, no icon, and will always be iconified when the main window gets iconified.

¹ Since 1.26.

2.9 Object Search

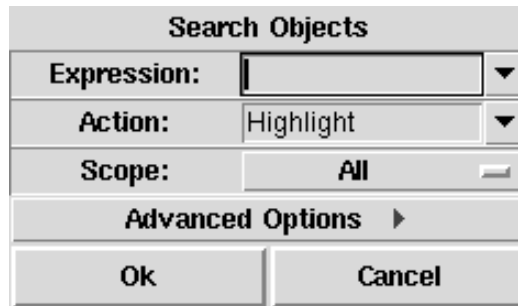


Figure 21: Object Search Dialog

The object search facility allows to find objects in the scene hierarchy according to many different, even script defined criteria and highlight them in the tree view or execute arbitrary actions on them.¹ Object search is controlled by a dialog, see also the image above.

Note, that the object search dialog is not closed immediately after a search operation, this allows multiple search operations with possibly refined parameters.

The first two options "Expression" and "Action" let the user specify which objects to find and what to do with them, they are explained in depth in the respective sections below.

Using the parameter "Scope" the search may be limited to certain sets of objects. By default, when scope is "All", all objects in the scene are processed, even if the current level is not the root level and no objects are selected.

If the scope is "Selection" just the selected objects and their children will be searched.

If the scope is "Level" all objects of the current level regardless of selection but again including all children will be searched.

If the scope is "Collection" just the objects found by the previous search are searched.

Note that the clipboard is never searched.

More options are made available, when the "Advanced Options" button is pressed.

The option "HighlightColor" defines the color to be used by the "Highlight" action.

If the options "ClearHighlight" or "ClearClipboard" are switched on, the highlights and the object clipboard are cleared before a search starts respectively.

Finally, "InvertMatch" reverts the search logic: if enabled, the search finds all objects for whose the search expression delivers a negative result.

2.9.1 Search Expressions

When just a simple string is used as search expression, this string is matched against the types, names, and materials of the objects to be searched in a case insensitive manner.² The matching allows usage of "*" (matches any sequence of characters), "?" (matches a single character), and character ranges specified like this "[A-z]". To match any of *?[] precede them with a backslash.

¹ Since 1.21. ² Since 1.22.

If the search expression starts with a "\$", "(", or "[" it is considered a special expression.

The following special search expressions are defined:

Type

find objects of a certain type, for example

```
$type == "Sphere"
```

finds all spheres.

Name

find objects of a certain name, for example

```
$name == "objname"
```

finds all objects with the name objname.

Material Name

find objects of a certain material, for example

```
$mat == "matname"
```

finds all objects with the material matname.

Property Value

find objects with a certain property value, for example

```
$SphereAttr(Radius) == 0.5
```

finds all spheres with radius 0.5.

Master

find the master of the currently selected instance.

Instances

find all instances of the currently selected master, or if the currently selected object is an instance, find all instances with the same master.

Procedure Call

find objects for which a procedure or command returns 1, for example

```
[hasChild]
```

finds all objects that have children.

See section [6.2.3 Interrogating Objects \(page 371\)](#) for a listing of available commands.

Complex logical expressions may also be created, like e.g.

```
($type == "Sphere") && ($mat == "wood")
```

which finds all wooden sphere objects. Due to the way the expressions are evaluated, the use of the braces is imperative even for a little bit more complex expressions that use the negation operator:

```
(![hasChild])
```

Also note that all string based comparisons of e.g. types or names via "==" are case sensitive.

The default value menu of the "Expression" entry is pre-seeded with meaningful entries when the search dialog is opened. What exactly appears in this menu is controlled by the first of the selected objects. For instance, if the selected object is a material, the material entry will contain the material name fetched from this object and will be one of the first entries, so that searching for objects of this material is just a matter of two mouse clicks. If the object is an instance, the first two entries in this menu will be "Master" and "Instances".

2.9.2 Search Actions

During a search, for every object that matches the given expression the object will be selected and an action will be executed. The following special actions are defined:

Highlight

All found objects will be colored in the object tree, see also the following image.

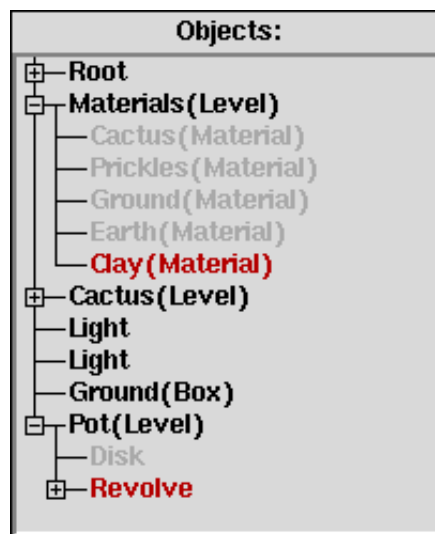


Figure 22: Object Search Result of Search String "clay"

Highlighted objects stay highlighted until the tree view is updated completely (e.g. via the keyboard shortcut <F5>) or partially (for instance after drag and drop or object clipboard operations).

Count

The objects matching the search expression will be counted and the result output to the console.

Collect

The node names of the found objects will be collected in the Tcl list "ObjectSearch(nodes)". This list can then be used for post-processing by scripts without the restrictions imposed by the "forAll" command.

Copy

The found objects will be added to the clipboard.

Delete

The node names of the found objects will be collected, then the list of objects will be processed so that delete is safe, then the objects will be deleted from the scene.

In addition to these special actions, a command / procedure call, like e.g.

```
convOb -inplace
```

or

```
addTag NoExport ""
```

can also be specified as action. However, as the object search is based on the "forAll" scripting interface command, certain restrictions apply:

- The action must not change the scene hierarchy, i.e. no objects must be deleted, cut, or created in the command / procedure call. Changing the objects themselves, even converting them in-place, however, is allowed.
- The script has no access to global variables, unless special measures are taken (see section 6.2.23 [Applying Commands to a Number of Objects](#) (page 422)).

See also section 6.2 [Procedures and Commands](#) (page 349) for a listing of available commands.

If the search action is empty, the objects will just be searched and a count of the matches will be reported.¹

The special action "Highlight" also collects the found node names in the global Tcl list "ObjectSearch(nodes)".

¹ Since 1.26.

2.10 Preferences

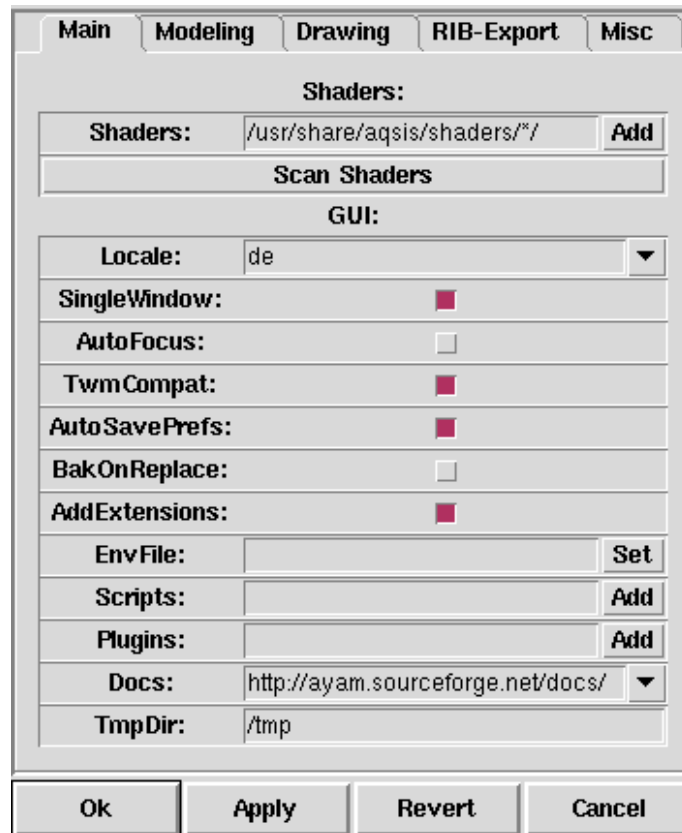


Figure 23: Preferences Dialog

The preferences dialog may be opened using the main menu entry "Edit/Preferences" or the shortcut <Ctrl+p>.

Use

- "Ok" to close the preference editor and apply all changes,
- "Apply" to apply the changes, but leave the editor open,
- "Revert" to reset everything to the state before the preference editor opened¹ (these are not the factory defaults, to get back to the factory defaults, restart Ayam with the command line option "-failsafe" or use the main menu entry "Special/Reset Preferences"),
- "Cancel" to close the dialog without applying any changes. All changes done after the last press of "Apply" will be lost.

Note that while the preference editor is open, AutoFocus is temporarily turned off and changes to the "AutoFocus" preference setting will only be realized after the editor is closed.

The preferences are divided into five sections as follows.

¹ Since 1.27.

2.10.1 Main Preferences

The "Main" section contains the most important application setup related preference settings.

Note that unused settings will not be shown, e.g. on the Win32 platform, the "TwmCompat" setting will be hidden.¹

- "Shaders" contains a number of paths (separated by a colon ":" on Unix and by a semicolon ";" on Win32) where Ayam looks for compiled shaders (e.g. files with the extension ".slc" that have been compiled with slc from BMRT). Using the "Add" button, paths may be added with a directory chooser dialog. After changes to this setting (also after "Revert"!), use the "Scan for Shaders!" button below.

Note that the environment variable SHADERS will be adapted to match the contents of this preference setting, so that renderers started by Ayam see exactly the same set of shaders as Ayam. However, this does of course not affect any renderers that are started outside the Ayam context, i.e. when rendering exported RIB files from a shell.

Since Ayam 1.22, the shader paths support globbing, i.e. it is possible to specify

```
"/usr/share/aqsis/shaders/*/"
```

to include

```
"/usr/share/aqsis/shaders/surface/"
"/usr/share/aqsis/shaders/displacement/"
"/usr/share/aqsis/shaders/volume/"
"/usr/share/aqsis/shaders/light/"
```

etc. with just a single entry. More complex styles of globbing, using e.g. the question mark, are also supported.

- "Scan for Shaders!" initiates a rebuild of the internal shader database. All shaders in the directories specified by the "Shaders" preference setting will be scanned and entered in that database. Watch the console for error messages that may appear while scanning. See also sections [4.10.4 Shader Parsing](#) (page 253) and [8.8 Shader Parsing Plugins](#) (page 529) for more information on scanning shaders.

The next sub-section contains GUI (user interface) related settings.

- "Locale", sets a language for the balloon help texts, the default value menu shows all currently available locales.

The value will have no effect until Ayam is restarted!

- "SingleWindow" toggles, whether Ayam should create just one main window with internal views and toolbox or use the old floating windows style GUI. The new single window GUI mode is enabled by default.
- "AutoResize" toggles, whether the main window should be resized horizontally according to the property GUI whenever a new GUI is displayed.

This option is not available in the single window GUI mode.

- "AutoFocus" controls whether Ayam should automatically move the focus to a view or the main window, when the mouse pointer enters it. However, depending on the operating system, window manager, or their settings, this may only work correctly, when a window of Ayam already has the

¹ Since 1.14.

focus. On some operating systems or window managers auto focus might not work at all. Furthermore, note that moving the focus to a toplevel window might also raise it.

In single window GUI mode, auto focus additionally manages the input focus of the internal views, tree view, property GUI, and console.

Note that while the preference editor is open, auto focus is always temporarily turned off and changes to the "AutoFocus" preference setting will only be realized after the preferences editor is closed.

- "TwmCompat" changes, how Ayam tells the window manager new geometries of windows. This option has to be toggled if Ayam fails to correctly remember the geometry of the main window between two invocations or if the main window jumps downward when properties are selected.

This option is not available on the Win32 platform and on Mac OS X Aqua.

- "AutoSavePrefs", if this is switched on, Ayam will write the preferences settings to the *ayamrc* file when the program quits. See also section 8.4 [Ayamrc File \(page 516\)](#).
- "BakOnReplace", controls whether Ayam will make a backup copy of each scene file loaded via the main menu entry "File/Open" or via the most recently used list. The backup file will be placed right next to the loaded file and get an additional file name extension according to the hidden preference setting "BackupExt".
- "AddExtensions", this option may be used to let Ayam automatically add file name extensions to saved files (if they do not have already a proper extension).
- "EnvFile" specifies the working environment scene file. This file typically contains some view objects to create a standard 2-, 3-, or 4-view working environment and is automatically loaded upon each start of Ayam (unless the hidden preference option "LoadEnv" is switched off) and upon each clearing of the scene using "File/New" (unless the hidden preference option "NewLoadsEnv" is switched off). See also section 8.4.2 [Hidden Preference Settings \(page 520\)](#).
- "Scripts" is a list of Tcl script files that will be executed on startup. The scripts do *not* have to be specified with full path and filename.¹ Furthermore, the scripts and plugins directories (as specified using the "Plugins" option below) are automatically searched, and the ".tcl" file name extension is automatically added, so that a setting of "colfocus;loadayslx" would load the "colfocus.tcl" script from the scripts directory and the ayslx shader parsing plugin (via the "loadayslx.tcl" script) from the plugins directory.

If non-absolute paths are used, they are considered to be relative to the current directory of Ayam on application startup (which is typically the directory where the Ayam executable resides).

Multiple entries have to be separated by a colon (":") on Unix and by a semicolon (";") on Win32. If unsure about the correct syntax, just use the "Add" button.

See section 6.5 [Distributed Helper Scripts \(page 446\)](#) for documentation on the scripts distributed with Ayam.

- "Plugins" is a list of directories that contain custom objects or plugins. Those directories will e.g. be searched for custom objects when unknown object types are encountered while reading Ayam scene files. If a matching custom object is found, it will be automatically loaded into Ayam, so that scene loading may proceed without an error. If non-absolute paths are used they are considered to be relative to the directory where the Ayam executable resides.

Thus, the default value "plugins" leads to Ayam searching for plugins in the directory "plugins" located in the directory, where the Ayam executable resides:

```
.../ayam/bin/Ayam.exe
.../ayam/bin/plugins/
.../ayam/bin/plugins/ayslx.dll
```

¹ Since 1.16.

Multiple entries have to be separated by a colon (":") on Unix and by a semicolon (";") on Win32. If unsure about the correct syntax, just use the "Add" button.

See section [8.7 Plugins Overview \(page 528\)](#) for more information regarding Ayam plugins.

- "Docs" is an URL that points to the documentation in HTML format.
- "TmpDir" is the directory, where temporary RIB files are created, e.g. when rendering directly from view windows.

The preference settings "ListTypes", "MarkHidden", "LoadEnv", and "NewLoadsEnv" are hidden preference settings since Ayam 1.14, see section [8.4.2 Hidden Preference Settings \(page 520\)](#).

2.10.2 Modelling Preferences

The next section of the preferences, "Modelling", contains modelling related settings:

- "ScopeManagement" controls how Ayam decides whether to transform selected points or objects: in the "Explicit" mode this is controlled manually using the keyboard shortcuts <o> and <p> respectively, whereas in the "Implicit" mode selected editable points are preferred. The implicit mode is the new default.¹ See also section 3.2 Transforming Objects or Selected Points (page 77).

- "PickEpsilon" is used by the single point editing actions (see section 3.12 Editing Points (page 90)) to determine which point of an object to modify and by the tag point and find knot actions (see section 3.4 Selecting Points (page 79) and section 3.20 Finding Points on Curves (page 99)) to determine which points/knots to select.

Smaller values mean more exact picking. The value of "PickEpsilon" should be positive and is expressed in terms of object coordinates, however, also the view zoom factor modifies the effective epsilon value in a way that for zoomed-in views a smaller value will be used (and vice versa).

- "HandleSize" controls the drawing size of editable points. Note, that there is no check, whether the size value is supported by the OpenGL implementation used.
- "LazyNotify" determines whether notification shall occur on all mouse movements or just on release of the mouse button, for the interactive modelling actions. This option may also be toggled easily using the keyboard shortcut <F6>.

Notification is the mechanism used to inform objects that rely on certain child objects (e.g. the Revolve tool object) about changes of their child objects, so that the parent can adapt to the child objects automatically (see also section 8.2 The Modelling Concept Tool-Objects (page 511)).

- "CompleteNotify" determines when notification of all objects depending on references of changed objects takes place:
 - "Never", for manual control of complete notification (manual complete notification can be carried out using the main menu entry "Tools/Force Notification" or using the keyboard shortcut <F5>);
 - "Always", a complete notification is done whenever the normal notification is carried out;
 - "Lazy", the complete notification runs only after a modelling action finished (when the mouse button is released).

Note that complete notify also updates objects that implicitly depend on updated objects via instances.

- "EditSnaps" determines, whether points should be snapped to the grid when a grid is defined and in use for the single point modelling actions.
- "Snap3D" controls whether points that are snapped to grid coordinates (in single point editing actions and when grids are active) should be influenced in all three dimensions, or just the two dimensions determined by the type of the view.
- "SelFacePoints" allows to toggle the automatic synchronization of the point selection with the face selection; if enabled, all points of a face will be selected after a selection of said face, see also section 3.25 Selecting Faces (page 104).
- "FlashPoints" controls flashing of editable points in the single point modelling actions when they would be modified by a click and drag action. Note that a change of this preference option only takes effect for the next invocation of the single point editing modelling action.

Also note, that there is a similar, albeit hidden, preference option that controls the flashing of picked objects ("FlashObjects"); see also section 8.4.2 Hidden Preference Settings (page 519).

¹ Since 1.30.

- "RationalPoints" determines the display style of rational points.¹

In the *euclidean* style the weights are not multiplied in, whereas in the *homogeneous* style the weights are multiplied in prior to display. This preference setting also influences the control hull display of NURBS curves and surfaces and some modelling actions like setting the mark from points. See also the following image, showing two standard nine point NURBS circles, where every second control point has a weight value of 0.75, with their control hulls in euclidean and homogeneous style.

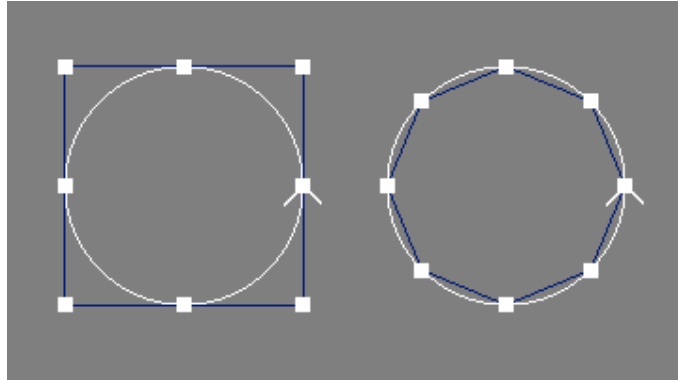


Figure 24: Euclidean (left) vs. Homogeneous (right) Display of Rational Points

- "GlobalMark" toggles whether each view should manage its own mark (off), or whether there should be just one global mark (on, default²). Note that enabling this preference setting will not immediately lead to a global mark set in all windows, one rather needs to set a new mark that will then become global. See also section 3.5 [Setting the Mark \(page 82\)](#) for more information about the mark.
- "CreateAt" controls the position where new objects are to be created. By default, this is the current mark position.³ Level and tool objects are always created with default translation parameters.
- "CreateIn" controls the orientation of newly created objects (except for Level and tool objects).⁴ By default, objects will be created without rotation attributes so that the orientation depends on the current level.⁵
- "DefaultAction" determines the modelling action that will be active after a press of the <Esc> key in a view window.
- "PickCycle" exchanges the object pick candidates dialog with a cycling mechanism:⁶ when multiple candidates are available, the first click selects the first object in the list and each further click on the same position selects the next object in the list of candidates. See also section 3.3 [Selecting Objects by Picking \(page 78\)](#) for more information about picking objects.
- "UndoLevels" determines the number of modelling steps that should be saved in the undo buffer. If "UndoLevels" is set to 0, the undo system will be disabled completely.⁷ For more information, see also the section 8.1 [The Undo System \(page 511\)](#).

¹ Since 1.22. ² Since 1.21. ³ Since 1.30. ⁴ Since 1.30. ⁵ Since 1.31. ⁶ Since 1.23. ⁷ Since 1.21.

2.10.3 Drawing Preferences

The preferences in the "Drawing" section let you specify how objects are being drawn:

- "Tolerance" controls the display quality of NURBS curves and surfaces. Smaller tolerance settings lead to higher quality but also slower display. Useful values range from 1 to 100. In the GLU drawing/shading modes this value is directly used as GLU sampling tolerance value. The STESS drawing/shading modes derive a quality/sampling factor from this setting, that leads to a sampling quality that is roughly equivalent to the corresponding GLU mode for a standard viewing distance.

Using the keyboard shortcuts <F2> and <F3> the tolerance may also be set easily.

This setting has no effect for objects that override it using a local tolerance setting different from 0.

- "NPDisplayMode" sets the display mode for NURBS patches. Either the control hull (or control polygon) is drawn ("ControlHull"), or just the outlines of the polygons created by the tessellation ("OutlinePoly"), or just the outlines of the patch ("OutlinePatch"). The latter are available in GLU and STESS variants.

The GLU variants tessellate the surface according to the current camera transformation. Zooming into an object increases the sampling rate. This approach is slow, needs to be done for every view window separately, but delivers high quality where needed. In addition, surfaces outside the current viewing volume are culled. Due to the dynamic sampling, the NURBS parameterisation can not be judged easily.

STESS, on the other hand, is tessellating in a uniform way and therefore the NURBS parameterisation can be inferred easily. The tessellation is done only once, even if there are multiple views, and then cached. Consequently, the startup cost and memory requirements of STESS are higher than those of GLU, but repeated drawing is much faster, even though culling in STESS is currently done on the basis of tessellated triangles, not entire surfaces.

Another aspect to consider is that GLU operates on the floating point data type "float" and imposes a serious constraint on knot values: knots that differ in less than 10E-4 are considered equal. In contrast, STESS operates on the floating point data type "double" and uses a more relaxed epsilon of 10E-6.¹

Finally, the application of instancing does not speed up drawing with GLU, but lowers the tessellation and memory costs of STESS, as instances use the cached tessellation of the respective master objects.

Note that the "NPDisplayMode" setting also influences display of shaded NURBS: ControlHull shades the control polygon², OutlinePoly (GLU) and OutlinePatch (GLU) use GLU, and OutlinePatch (STESS) shades the STESS tessellation. The shaded STESS tessellation of non-planar trimmed NURBS surfaces is of low quality.

Note also, that the "NPDisplayMode" setting has no effect for objects that override it locally using a "DisplayMode" attribute different from "Global".

Toggling between drawing of hulls and outlines may also be done easily using the keyboard shortcut <F4>.

- "NCDisplayMode" sets the display mode for NURBS curves; just like for surfaces the control hull (control polygon) or the curve or a combination of both may be displayed.

See also the discussion of GLU vs. STESS above.

Note that this setting has no effect for objects that override it locally using a "DisplayMode" attribute different from "Global".

Toggling between drawing of hulls and curves may also be done easily using the keyboard shortcut <F4>.

¹ Ayam may be recompiled with a different epsilon if this is still limiting. ² Since 1.22.

- "ToleranceA" is the GLU sampling tolerance that is used for display of NURBS curves or patches in all interactive actions, i.e. while interactively moving objects or editing control points.

Negative values are interpreted as multiplicand in the following way. A value of -2 means: use the value from the "Tolerance" setting multiplied by 2, e.g. if the "Tolerance" is set to 40, a value of 80 will be used in actions.¹

The default value 0.0 means no change from the "Tolerance" setting.

In contrast to the "Tolerance" setting above, objects can *not* override this setting locally.

- "NPDisplayModeA" controls the display mode of NURBS patches for all interactive actions. The first and default value ("NPDisplayMode") means: no change.

In contrast to the "NPDisplayMode" setting above, objects can *not* override this setting locally.

- "NCDisplayModeA" controls the display mode of NURBS curves for all interactive actions. The first and default value ("NCDisplayMode") means: no change.

In contrast to the "NCDisplayMode" setting above, objects can *not* override this setting locally.

- "UseGUIScale" determines, whether the GUI scale factor shall be used to adapt the line widths, sizes of handles, and other annotations when drawing the scene.² For more information, see also the section [2.11 GUI Scaling \(page 74\)](#).

- "UseMatColor" determines, whether the shaded representation uses the color defined by the material of an object for rendering. If disabled, the color defined by the "Shade" option (below) will be used instead.

- "Background", "Object", "Selection", "Grid", "Tag", "Shade", and "Light" set the colors that will be used when drawing or shading the respective primitives.

¹ Since 1.22. ² Since 1.31.

2.10.4 RIB-Export Preferences

The "RIB-Export" section of the preferences contains settings that affect how RIBs are created.

- "RIBFile" allows to set the file Ayam is exporting RenderMan Interface Bytestreams (RIBs) to. Note that some file names have special meaning:

If "RIBFile" is set to "Scene" (this is the default) the RIB file name will be derived from the name of the currently loaded scene with the last extension replaced by ".rib". If "RIBFile" is set to "Scenefile", the leading path will be stripped from the scene name additionally. Use "Scenefile", for rendering with shadow maps. This way the scene will use relative paths to load the shadow maps and the RIBs may be moved around more easily.

"Ask" is another special setting, that allows to select a different file name each time the scene is exported. A file selection dialog will pop up, after the selection of the view to export. The same effect may be achieved by leaving "RIBFile" totally empty.

If "RIBFile" is set to "rendrib", libribout.a does not create a RIB file at all, but immediately pipes the resulting byte stream into rendrib (the BMRT renderer) for rendering. The same goes for "rgl". Moreover, file names that start with a pipe symbol "|" will cause the program after the pipe symbol to be started by libribout and the written RIB to be piped into. This works e.g. with Photorealistic RenderMan, try it out with "|render". In the latter cases of direct rendering, you will probably want to set up the RIB to render to the display (read leave the "Image" preference setting empty. However, when you use these options of direct rendering, be warned, that for the time of the rendering Ayam will be frozen (it will neither respond to mouse clicks nor will it update any windows), until the rendering is finished and the display window of the renderer is closed.

- "Image" specifies the image file that will be created, when the exported RIB file is rendered. When this option is set to "RIB", rendering will create image files that are named as the exported RIB file (with the last file extension replaced by ".tif"). Again, setting it to "Ask" will cause a dialog box to appear, each time the scene is exported to RIB.

Note that in contrast to the "RIBFile" option, leaving the field totally empty is not equal to entering "Ask" but generates RIB files that will be set up to render to the display.

- "ResInstances", if this is enabled all instance objects are resolved (temporarily) before being written to the RIB file.
- "CheckLights", if this is enabled Ayam will check the current scene for lights before RIB export. If no lights or no lights that are actually switched on are to be found in the scene, a distant headlight will be added to the scene automatically for RIB export.
- "DefaultMat" determines a default material setting that should be written in the top level of the RIB, so that it is in effect for all objects, that are not connected to a material object. Many RenderMan compliant renderers will not render the objects at all, if no material is defined. The default "matte", writes just a simple Surface "matte" (without parameters) to the RIB. The setting "default" looks for a material object named "default" and writes it's complete shaders and attributes, if it does not find such a material it falls back to "matte". The setting "none" does not write any default material setting.
- "RISstandard" determines whether Ayam should omit all non standard RenderMan interface options and attributes on RIB export.
- "WriteIdent" determines, whether Ayam should write special RiAttributes (RiAttribute "identifier" ["name"]) with the names of the objects to the RIB to aid in RIB file debugging.
- "ExcludeHidden" causes hidden objects not to be exported to RIB files.

The following sub-section of preference settings is concerned with settings for rendering directly from Ayam.

- "SetRenderer" allows to select which renderer preferences to edit (Preview, QuickPreview, File, ShadowMaps, or All). Depending on the selection, different sets of options will be shown below for the respective renderer.
- "RenderMode" allows to switch between different methods of forcing the preview renderer to render to the screen:
 - "CommandLineArg" via a command line argument, e.g. `-d` for rendrib from BMRT,
 - "RiDisplay" via a RiDisplay statement in the exported RIB, necessary for e.g. PRMan and RDC,
 - "Viewport" via a RenderMan display driver as specified below.

The "Viewport" rendering will directly appear in the Ayam view window see also section 8.9 [Viewport Rendering \(page 530\)](#).

- "Render" determines the command that should be executed upon preview rendering of a view using the view menu "View/Render" or the keyboard shortcut `<Ctrl+R>`. The placeholder "%s" denotes the name of the RIB file. The command string may contain additional (command line) parameters that should reflect other renderer preference options, i.e. "RenderMode" and "RenderUI". Please check with the documentation of your renderer.

Examples:

- `rendrib %s`
for rendering with BMRT in render mode "RiDisplay"
- `aqsis %s`
for rendering with Aqsis in render mode "RiDisplay" or "Viewport"
- `aqsis -d %s`
for rendering with Aqsis in render mode "CommandLineArg"

- "RenderUI" enables the preview renderer user interface (*Rendering GUI*), which consists of a simple progress bar, a label that displays the estimated remaining rendering time, a checkbox to control ringing the bell when the rendering is finished, and a cancel button to stop the rendering. See also the image below.

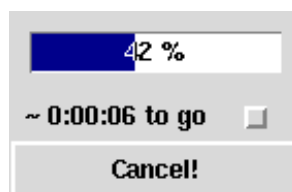


Figure 25: Rendering GUI Dialog

The Rendering GUI is displayed when a renderer is invoked directly from a view window using one of the "Render" view menu entries (or the equivalent keyboard shortcut). The Rendering GUI also reroutes the renderer console output to the Ayam console, standard and diagnostic channels are handled separately. Proper work of all Rendering GUI features depends on the existence of two external programs: "cat" and "kill" (those programs should be available on every Unix platform). If you do not have those programs in your path, do not enable the RenderUI option.

On the Win32 platform a working "cat.exe" is distributed with Ayam and Ayam is set up to use an internal kill command "w32kill" so that the rendering GUI should work out of the box despite the

general unavailability of said commands on this platform.¹ See also section 8.4.2 [Hidden Preference Settings](#) (page 520).

If this option is not enabled, the renderer console output will be handled jointly; also, some output may be delayed and will appear only after the renderer is finished.

- "RenderPT" is a string that contains a progress output template used by Ayam to determine the current percentage of completion of the rendering for display in the Rendering GUI. The placeholder "%d" denotes the position of the percentage number in the output of the renderer. For `rendrib` from BMRT 2.6 this should be set to "R90000 %d" and the special command line option "-Progress" should be used for the renderer command (see "Render" above). For `rendrib` from BMRT 2.5 it should be set to "Done computing %d" and no special option has to be given to the renderer. If the output of the renderer contains variable strings before the progress number or the renderer uses terminal control characters, a second variant of parsing the output using regular expressions is available.² In this case, the progress template should be a complete regexp command for Tcl that parses the output of the renderer with a regular expression and puts the extracted progress number into a variable named "percent". The output to parse will be inserted in the regexp command template in place of "string". The dummy output variable is needed, because Tcl regexp always outputs the complete matching line into the first variable.

Here is an example that works with Pixie-1.2.1, which outputs strings like

```
fish.rib (222): - 10.00 percent
```

(note the variable component containing the RIB name):

```
regexp -- {^.* - ([0-9\+])} string dummy percent
```

This regexp command eats the variable component away and then looks for the single dash right in front of the percent number.

Gelato output looks like this

```
Elapsed:      10.1440s  Done:   64%  |*****|
```

and uses terminal control characters to render the progress bar in ASCII; this calls for a different approach that parses backwards (hence the use of \$ at the end):

```
regexp -- {([0-9\+])%([ |\*]+)$} string dummy percent
```

This regexp command eats away all pipe symbols, asterisks, and spaces *backwards*, starting from the last character and including the percent sign, which comes right behind the sought after percentage number.

- "DisplayDriver" sets the RenderMan display driver to use for the "Viewport" render mode, at the moment the only sensible settings of this are "fifodspy" and "pixiefifodspy". See also section 8.9 [Viewport Rendering](#) (page 530).
- "QRenderMode" see "RenderMode" above.
- "QRender" determines the command that should be executed upon quick rendering a view using the view menu "View/Quick Render" or the keyboard shortcut <Ctrl+r>. The placeholder "%s" denotes the name of the RIB file. The command string may contain additional (command line) parameters that should reflect other renderer preference options, i.e. "QRenderMode" and "QRenderUI". Please check with the documentation of your renderer.

Examples:

```
- rgl %s
  for rendering with rgl from BMRT in render mode "CommandLineArg"
```

¹ Since 1.4. ² Since 1.6.

- `rendrib -d 4 %s`
for rendering with BMRT in render mode "CommandLineArg"
 - `aqsis %s`
for rendering with Aqsis in render mode "RiDisplay" or "Viewport"
- "QRenderUI", enables the Rendering GUI for quick rendering, see discussion of "RenderUI" below.
- "QRenderPT", progress template for quick rendering, see discussion of "RenderPT" below.
- "QDisplayDriver" sets the RenderMan display driver to use for the "Viewport" render mode, at the moment the only sensible settings of this are "fifodspy" and "pixiefifodspy".
- "FRenderMode" see "RenderMode" above.
- "FRender", renderer command to use for direct rendering to image files (via view menu entry "Render to File"). The placeholder "%s" denotes the name of the RIB file. The command string may contain additional (command line) parameters that should reflect other renderer preference options, e.g. "FRenderUI". Please check with the documentation of your renderer.
- "FRenderUI", enables the Rendering GUI for the direct rendering to image files, see discussion of "RenderUI" above.
- "FRenderPT", progress template for the direct rendering to image files, see discussion of "RenderPT" above.
- "FDisplay" sets the RenderMan display to use for rendering to an image file, the syntax of this string is equivalent to the value string of a RiDisplay tag see also section 4.11.6 RiDisplay Tag (page 263). If this string starts with "image", the image part will be replaced by the image file name that would be used for normal RIB export.

Examples:

- `image.exr,exr_dspy,rgba`
for rendering to OpenEXR files (instead of TIFF)
 - `image.tif,file,z`
for rendering only the depth buffer to (single channel) TIFF files
- "SMRender", renderer command to use for the rendering of shadow maps (e.g. view menu entry "View/Create All ShadowMaps"), see also section 4.2.4 Using ShadowMaps (page 121). The placeholder "%s" denotes the name of the RIB file. The command string may contain additional (command line) parameters that should reflect other renderer preference options, please check with the documentation of your renderer.
- "SMRenderUI", enables the Rendering GUI for the rendering of shadow maps, see discussion of "RenderUI" above.
- "SMRenderPT", progress template for the rendering of shadow maps, see discussion of "RenderPT" above.
- "AutoCloseUI" controls whether the Rendering GUI dialog window should be closed automatically when the renderer signals that the rendering finished.

The following sub-section of preference settings is concerned with shadow maps.

- "ShadowMaps" determines, whether shadow maps should be created/used:
 - "Never" shadow maps will not be created/used, and shadows will be created by the renderer using algorithms like ray tracing.
 - "Automatic" shadow maps will be created and used each time the exported RIB is rendered.
 - "Manual", shadow maps will be used each time the exported RIB is rendered. The shadow maps will be created/rendered on user request only (e.g. using the view menu entry: "View/Create All ShadowMaps").

The mode "Manual" should be used, when rendering directly from view windows with rendering mode "CommandLineArg".

Note that it is not sufficient to enable this option for an arbitrary scene to render using shadow maps, individual light sources that shall use shadow maps have to be parameterised for this as well, see section 4.2.4 Using ShadowMaps (page 121).

- "SMFileFormat", designates the file format of the shadow maps, use "zfile" for RenderMan and "shadow" for Gelato.
- "SMFileType", type of shadow maps to be created, currently available types are "z" – normal shadow maps (for RenderMan renderers and Gelato), "avgz" – Woo shadow maps (for Gelato only!), and "volz" – volume shadow maps (for Gelato only!).
- "SMChangeShaders", toggles, whether Ayam should automatically prepend a "shadow" to light shader names for lights that use shadow maps upon RIB export. Note, that for rendering with Gelato, this option should be disabled.
- "PPRender" is the name of the renderer to use for the permanent preview feature (see also section 2.5 View Menu (page 42)). This setting is just available, if the compile time option AYENABLEPPREV has been set. This option is *not* set for the official Ayam binaries.

Note that many renderer related preferences can be set at once using the select renderer tool via the main menu "Special/Select Renderer" (see also section 2.2 Special Menu (page 38)). In fact, using "Special/Select Renderer" first, then fine tuning the renderer setup using the preferences editor is the suggested way to switch Ayam to a certain RenderMan renderer.

2.10.5 Miscellaneous Preferences

The "Misc" section of the preferences contains the dreaded miscellaneous settings.

The first sub-section deals with error message handling:

- "RedirectTcl" controls whether error messages stemming from Tcl/Tk should be redirected to the console, rather than be handled by Tcls sometimes annoying error handling dialog box. However, this dialog box with the built in stack trace can also become very handy, while writing and debugging Tcl scripts.
- "Logging" determines, whether any messages should be written to the file specified by the "LogFile" option. If logging is enabled, the log file should be cleared manually from time to time, as Ayam will always append to the file.

In contrast to what is output to the console, the messages in the log file will include date and time.¹

- "LogFile"; path and filename of the file where to write the log messages to, see also the "Logging" option above.

When overridden by the command line option "-log", both settings, "Logging" and "LogFile", will turn into a information display.² See also section [8.5 Command Line Arguments \(page 526\)](#).

- "ErrorLevel", this option controls how many messages should be written to the Ayam console. Available values are:
 - "Silence" no output,
 - "Errors" only error messages,
 - "Warnings" only error and warning messages, and finally
 - "All" (default) all messages, even informative, should be written to the console.

Note, that the error level does *not* control which messages will be written to the log file.

¹ Since 1.33. ² Since 1.33.

The last sub-section contains miscellaneous user interface related preferences:

- "ExportSetsCD": controls whether or not the current directory of Ayam should be set by an export operation.
- "ImportSetsCD": controls whether or not the current directory of Ayam should be set by an import operation.
- "SetActionMenu"; opens the action icon menu configuration dialog, see also section [2.5 Action Icon Menu \(page 46\)](#).
- "TclPrecision"; this is the precision Tcl handles floating point number to string conversions with. You may want to decrease this value to about 5 if any numbers in the entry fields are represented in an exact, but also too lengthy and hard to read fashion, like 0.4999999 instead of 0.5. Note that some model precision is lost in doing so. The default value used by Tcl is 12 and results in no loss of precision. The default value used by Ayam is 6 and should result in a good balance between precision and readability.

Related hidden preference options are "NormalizeDigits", "NormalizePoints", and "NormalizeTrafos", see also section [8.4.2 Hidden Preference Settings \(page 520\)](#).

- "SaveDialogGeom" controls whether the geometry of various dialog windows should be remembered by Ayam, the settings available are
 - "Never": the dialog windows are always opened in standard size, centered on the screen;
 - "WhileRunning": the window geometry will be remembered as long as Ayam is running;
 - "Always": the window geometry will be stored in the saved preferences, thus, also surviving a restart of Ayam.

Note that the height of the preferences dialog window will always be adapted to the currently open preferences section, no matter how "SaveDialogGeom" is set.

For more geometry saving related information, see also sections [4.11.9 SaveMainGeom \(page 264\)](#) and [4.11.10 SavePanelLayout \(page 265\)](#).

- "SMethod"; is the sampling method used by the NURBS tessellation facility that converts NURBS surfaces to PolyMesh objects.

Six sampling methods are available:

1. "ParametricError" ensures that the distance between the tessellated surface and the original surface is no point bigger than the value specified by "SParamU".
2. The sampling method "PathLength" ensures that no edge of a polygon generated by the tessellation is longer than the value specified by "SParamU" and the tessellation method
3. "DomainDistance" (the default in Ayam up to version 1.22) simply tessellates the NURBS surface into equally sized pieces with regard to parametric space; "SParamU" and "SParamV" control the number of sampling points in U and V direction respectively per unit length in parameter space. However, this leads to different numbers of samples for knot vectors of different total length.
4. "NormalizedDomainDistance" ensures that the tessellation creates the same number of sample points (as given via "SParamU" and "SParamV") for knot vectors of any total length in parameter space¹ and
5. "AdaptiveDomainDistance" additionally adds sample points depending on the number of control points (width or height of the patch) to provide a better adaptation to complex patches.²
6. "AdaptiveKnotDistance", finally, normalizes the number of sample points to the number of knot intervals and the total length of the knot vector.³

The sampling method "AdaptiveKnotDistance" is the default since Ayam 1.23.

Note that the sampling method controlled by this preference option is *not* used for NURBS display in Ayam. See also [2.10.3 Drawing Preferences \(page 64\)](#).

- "SParamU"; is a parameter for the sampling method above.
The default value for the sampling method "AdaptiveKnotDistance" is 3. Bigger values lead to better quality and more tessellated polygons.
The default value for the sampling method "DomainDistance" is 8. Bigger values lead to better quality and more tessellated polygons.
The default value for the sampling method "PathLength" is 1.5. Smaller values lead to better quality and more tessellated polygons.
The default value for the sampling method "ParametricError" is 0.25. Smaller values lead to better quality and more tessellated polygons.

Note that "SParamU" is expressed in object space units for the "PathLength" and "ParametricError" sampling methods.

- "SParamV"; is just available for the sampling methods "DomainDistance", "NormalizedDomainDistance", "AdaptiveDomainDistance", and "AdaptiveKnotDistance".
The default value is equal to the respective value of the "SParamU" parameter above.

See also section [5.6.5 Tessellation Tool \(page 337\)](#) for example tessellations.

¹ Since 1.9. ² Since 1.9. ³ Since 1.23.

2.11 GUI Scaling

Since version 1.31 the Ayam GUI can be scaled to better accommodate to high resolution displays. The scaling is fractional with the exception of all pixel based GUI elements (icon images).

The GUI scale factor must be set using the command line option `"-guiscale"`, see also section [8.5 Command Line Arguments \(page 526\)](#). If a value greater than 1.0 is specified, the GUI will be scaled as follows:

- the fonts used by Ayam will be scaled using the GUI scale directly; however note, that as font sizes are integer values, not every change in scale factor will lead to a different font size,
- Tk is instructed to adapt by setting a `"tk scaling"` value that is manipulated using the GUI scale factor directly,
- icons (and therefore the tool buttons) will be scaled to double size when the GUI scale value reaches 1.5,
- the handles of the panes will be enlarged when the icons scale,
- object tree graphics will be enlarged when the GUI scale value is larger than 1.5,
- cascade menu indicators and scrollbars will be sized according to the new default font sizes,
- the keyboard focus ring width will be enlarged when the icons scale,
- if the drawing preference option `"UseGUIScale"` is enabled, line widths, handle sizes, and other annotations will also be adapted to the scale factor.

3 Modelling Actions

This section contains the documentation of the interactive modelling actions available in Ayam.

Before invoking any modelling action one or more objects should be selected using the object hierarchy in the main window, the pick action, or selection manipulating keyboard shortcuts.

Every action can be started with a key press (a shortcut) when the keyboard focus is in a view window or by clicking on the associated button in the toolbox window. Using a keyboard shortcut starts that action in the current view only, the other views are not affected. In multi window GUI mode, starting an action from the toolbox window will cause the action to be started in all view windows that are currently open simultaneously. In single window GUI mode, the action will be started in the current internal view only (unless "AutoFocus" is enabled).

It is perfectly ok to start and work with many different actions at the same time in different views: one can have e.g. a view, where objects are moved, a second where objects are rotated and a third, where objects are picked. The layout, drawing style, and grids may also differ between the different views. Together with the feature, that the selection may be changed while actions are active and even panning and zooming of views is possible (using the rightmost and middle mouse buttons respectively) while actions are active and without breaking them, this is the key to unsurpassed flexibility in modelling using Ayam.

To break an action, the <Esc> key may be used.

The default action for all views, which is also in effect after use of the <Esc> key, is "None", "Pick", or "Edit" (depending on the preference setting "Modelling/DefaultAction"). See [section 3.3 Selecting Objects by Picking \(page 78\)](#) for more information about picking objects.

Note that modelling actions that are directly modifying transformations or points are not available in perspective views, but selection actions and also the numeric point editing action do work fine.

If an action is in effect for a view, the views title will be changed appropriately.

A modelling action is performed by clicking into the view with the leftmost mouse button to mark a point in space or to pick a vertex and then by dragging the mouse.

The effects of a modelling action can be undone using <Ctrl+z> and restored using <Ctrl+y> (see [section 8.1 The Undo System \(page 511\)](#) for more information).

Grids are available to restrict the modelling actions to certain points and help in exact modelling.

Also note that the middle and rightmost mouse buttons may be used to zoom and move the view while modelling actions are active.

For actions that modify the camera of a view please see [section 2.6 View Window Shortcuts and Actions \(page 48\)](#).

3.1 Modelling Actions Overview

The following tables give an overview over the various modelling actions.

These actions allow to select objects, points, faces or curves for use with other actions or tools.







Name	Shortcut	Icon	Name	Shortcut	Icon
Objects	<P>		Points	<t>	
Faces	<f>		Boundary Points		
Boundary Curve	<n>		Trim Curve	<k>	

Table 2: Selection Actions Overview

These actions allow to transform objects or selected points.

























Name	Shortcut	Icon	Name	Shortcut	Icon
Move	<m>		Move X	<mx>	
Move Y	<my>		Move Z	<mz>	
Rotate	<r>		Rotate About	<ra>	
Rotate X	<rx>		Rotate X About	<rxa>	
Rotate Y	<ry>		Rotate Y About	<rya>	
Rotate Z	<rz>		Rotate Z About	<rza>	
Scale 3D	<s>		Scale 3D About	<Sa>	
Scale 2D	<s>		Scale 2D About	<sa>	
Scale 1D X	<sx>		Scale 1D X About	<sxa>	
Scale 1D Y	<sy>		Scale 1D Y About	<sya>	
Scale 1D Z	<sz>		Scale 1D Z About	<sza>	
Stretch 2D	<T>		Stretch 2D About	<Ta>	

Table 3: Transformation Actions Overview

These actions allow to edit parametric curves or surfaces.







Name	Shortcut	Icon	Name	Shortcut	Icon
Edit	<e>		Edit Numeric	<E>	
Edit Weights	<w>		Reset Weights	<W>	
Insert Points	<i>		Delete Points	<d>	

Table 4: Edit Actions Overview

3.2 Transforming Objects or Selected Points

Many modelling actions either work on objects transformation attributes or coordinates of selected points. Since Ayam 1.30 this is controlled implicitly by the point selection: if editable points are selected, they will be transformed instead of the object. If there are objects with *and* without selected editable points in the selection, the latter will not be transformed at all.

Using the preference setting "ScopeManagement" (see section 2.10.2 Modelling Preferences (page 62)) a explicit scope management mode can be enabled. For this mode there are two keyboard shortcuts: <o> and <p>. <o> switches to object transformations and <p> to point transformations. These shortcuts can be used anytime, even in the middle of any other actions. If points are to be modified, a little red point will appear in the modelling mode icon in the view menu as shown in the image below:

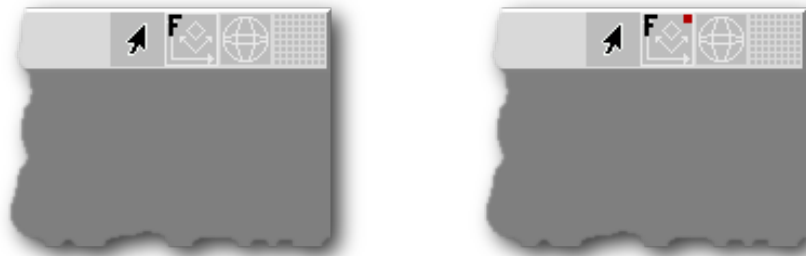


Figure 26: View Transforming Objects (left) and Points (right)

Note that those shortcuts just switch the current view, to modify all open views, just press o or p twice: <oo> and <pp>.¹

Also note that the modelling scope may also be toggled by clicking on the modelling mode icon with the rightmost mouse button or by starting the action using this mouse button.²

¹ Since 1.21. ² Since 1.26.

3.3 Selecting Objects by Picking

This section describes techniques that can be used for selecting one or more objects within a view window with the mouse.

When the view window's action is "Pick", objects that appear within this view can be selected with the mouse. This action may be invoked by pressing <P> or by making this action the default action using the preference setting "Modelling/DefaultAction". If picking is the default action, it will be automatically enabled when any other action is broken using <Esc>.

3.3.1 Selecting Individual Objects

Selecting objects within a view is a straightforward operation that uses standard methods. You will use the following two selection operations most frequently:

- To select a single object within a view, move the cursor to the object and click mouse button 1 (the leftmost one). Once you select an object, any objects previously selected are unselected automatically.
- To select an additional object, move the cursor to the object and <Control>+Click (again with the leftmost mouse button). Previously selected objects remain selected, and the newly picked object is added to the selection. Notice that the picked item must belong to the same level as the previously selected objects. An alternative method for selecting multiple objects is to drag a rectangle around them. For more information see [3.3.2 Drag-selecting Multiple Objects \(page 78\)](#)

Note that the Root object and View objects can not be pick-selected at all.

3.3.2 Drag-selecting Multiple Objects

You can select multiple objects using the <Control>+Click method described in [section 3.3.1 Selecting Individual Objects \(page 78\)](#). An additional method for selecting multiple objects is to drag a rectangle around those objects. However only objects that belong to the current level can be picked within a drag-selection. If you want to select multiple objects that belong to another level you must change the current level by either selecting it in the tree/listbox or by picking one object from that level on a view.

The procedure for drag-selecting multiple objects also uses a standard method:

1. Imagine a rectangle that encloses only the objects you want to select.
2. Click at one corner of the rectangle and, while continuing to press the mouse button, drag until you have enclosed all the objects.
3. Release the mouse button. All the valid objects inside or crossing the rectangle are selected and any objects previously selected are unselected automatically.

Note that if you press <Control> during the drag-selection, objects that are enclosed by the rectangle will be added to the current selection instead of replacing it.

3.3.3 Ambiguous Picking

In some cases Ayam is unable to differentiate between the objects you have selected and other nearby or related objects. This ambiguity can arise as follows:

- Imagine a small square surrounding the cursor. When you click an object, any other valid objects that fall inside this square are also considered to be possible selections. For example, if you select an item that is positioned very close to another one, Ayam may consider both items to be possible selections.
- If your model is three-dimensional (which is likely to happen), imagine a line that is perpendicular to the screen and that passes through the cursor and into the model. When you pick an object, any objects that intersect this line are considered to be possible candidates for selection.

If the selection is ambiguous Ayam displays a window that contains a list of the possible candidates for selection (see image below), or, if the "PickCycle" option is activated, Ayam just cycles through the candidate objects by further clicks on the same position.¹

When a name in the candidate list is selected, the corresponding object is highlighted. Click "Ok" when you have determined which object to select or "Cancel" to close the list and keep the previous selection unchanged.

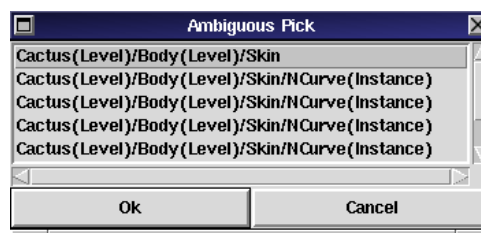


Figure 27: List of Picking Candidates

It is also possible to use a double click in the list of selection candidates to select an object and immediately close the dialog window. Moreover, as the list of candidates immediately gets the keyboard focus, the cursor keys and <Enter> can also be used to select the object.²

Notes:

- While the list of ambiguous candidates is opened no other objects can be picked within the views.
- It is possible to use the "Zoom to Object" action (shortcut <BackSpace>) while the ambiguous select listbox is open to get a better view of the temporarily selected object.
- The tolerance used to determine whether an object should be picked or not can be adjusted (see "PickTolerance" in 8.4.2 Hidden Preference Settings (page 521)).

3.4 Selecting/Tagging Points

The modelling action "Select Points" (shortcut: <t>; for **t**ag points) may be applied to objects that support single point editing or read only points.

To select or de-select a point it can directly be clicked upon or a rectangular region can be dragged with the mouse around the points in question.

¹ Since 1.23. ² Since 1.11.

Only points within a certain distance from the mouse pointer click position will be considered picked (see preference option "Modelling/PickEpsilon"). Points that would be picked will flash in a different color when the mouse pointer moves over them (this is also controlled by the preference option "Modelling/FlashPoints").

Selected points will be drawn in dark red when the select points modelling action or a modelling action that would modify the selected points is active (see the following image).

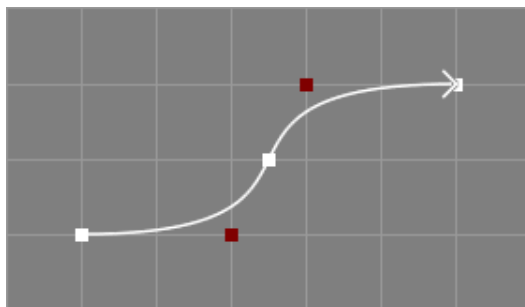


Figure 28: Object With Selected Points (red)

The selected editable points may be modified subsequently using the modelling actions Move, Rotate, and Scale; refer also to the discussion in section 3.2 [Transforming Objects or Selected Points](#) (page 77). See also the table below.

Name	Shortcut	Icon
Tag Points	<t>	

Table 5: Tag Action

After the pick (the selection of a point), the picked point will be added to the list of selected points for the selected object. If the selected point is already in that list it will be removed from the list instead, the picked point will be deselected.

All points can be selected easily using the keyboard shortcut <A>.

The list of selected points will not be deleted from the object until an explicit deselection is performed using the keyboard shortcut <N> or if certain modelling tools are used.

The point selection can also be inverted using the keyboard shortcut <I>.

All three shortcuts above can be used anytime without interfering with any other active modelling actions.

Note that the list of selected points is not copied, if the object is copied using the clipboard. The selection is also not saved to Ayam scene files. But selected points survive undo/redo.¹

Furthermore, the selection action itself is not an operation recorded in the undo buffer, thus cannot directly be undone.

The point selection also does not interfere with single point modelling actions: It is perfectly legal to select some points, move them using the move action, then switch to single point editing, edit some other or even one of the selected points, switch back to the selection action, add other points to the selection or delete some points from the selection, switch to rotate, rotate the selected points and so on.

¹ Since 1.16.

A bigger number of points may be added to the selection using a click and drag operation. All points that are inside the rectangular region defined by the click and drag will be added to the selection. In fact, this approach is the only way to safely add points to the selection that occlude each other. Clicking always only adds/removes single points.

Holding down the <Ctrl>-key while dragging the mouse removes all selected points within the rectangular region defined by the drag from the selection.¹

The exact behaviour of picking multiple points of a NURBS curve or surface depends on the attribute "CreateMP":

If "CreateMP" is enabled, picking a multiple point will always select all points that make up the multiple point.

If "CreateMP" is disabled, picking a multiple point will only select the first of the points that make up the multiple point (but drag selection can be used to select all points nevertheless).

¹ Since 1.16.1.

3.5 Setting the Mark

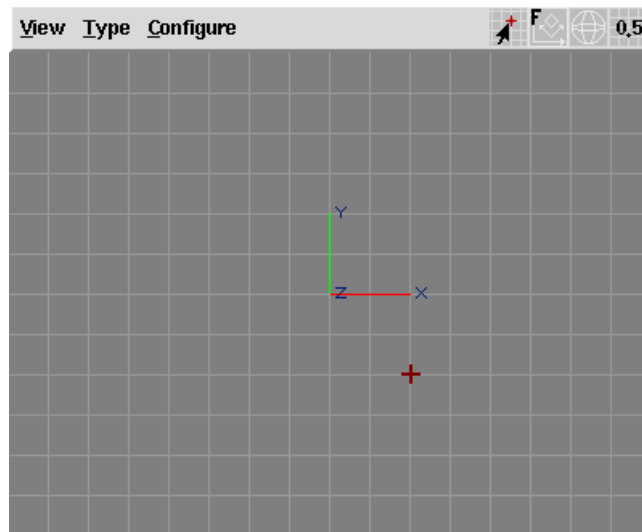


Figure 29: Front View With Mark Set At 1,-1,0

This action may be invoked to mark a point in space for perusal of other modelling actions that e.g. rotate or scale about a point. The marked point will be shown as a little red cross in the view window (see also the image above).

The keyboard shortcut to invoke the set mark action is `<a>`. This action is also active as intermediate action for e.g. rotate about, where the keyboard shortcuts `<r>` and then `<a>` would be used in a sequence and after setting the mark, the rotate about action would take over again.

See also the table below.

Name	Shortcut	Icon
Set Mark	<code><a></code>	

Table 6: Set Mark Action

A single click with the left mouse button sets the mark. If grids are active in the view, the mark will be snapped to the nearest grid coordinates.

Note, that, using a single mouse click, only two dimensional mark coordinates can be specified. Therefore, in versions prior to 1.21, the third coordinate was always set to zero upon a click. This is no longer the case. If the preference option "Modelling/GlobalMark" is enabled, the third coordinate is *not* reset anymore. This way all three coordinates of the mark can be defined by clicking two times in appropriate views, e.g. first in a front view for the x and y coordinates and then in a side view for the missing z coordinate.

The mark can also be set to a point of one of the selected objects by clicking on it. In this case, the mark coordinates are always immediately three dimensional.¹

If the mouse is dragged instead of clicked, the mark is set to the bounding box center of all points in the drag rectangle.²

¹ Since 1.21. ² Since 1.25.

Additionally, when the set mark action is active, the following keyboard shortcuts are available:

<Return> accept the current mark (useful, if one first rotates about a point then decides to also scale about the same point: <ra>, drag mouse, <sa>, <Return>, drag mouse ...),

<c> set the mark to the center of gravity of all currently selected objects coordinate systems,

<C> set the mark to the center of gravity of all currently selected points.

 set the mark to the center of the bounding box of all currently selected points.

<0> – <9> open a intermediate parameter GUI where the three mark coordinate values (separated by spaces) may be entered.¹

The image below demonstrates the difference between the center of the bounding box and the center of gravity.

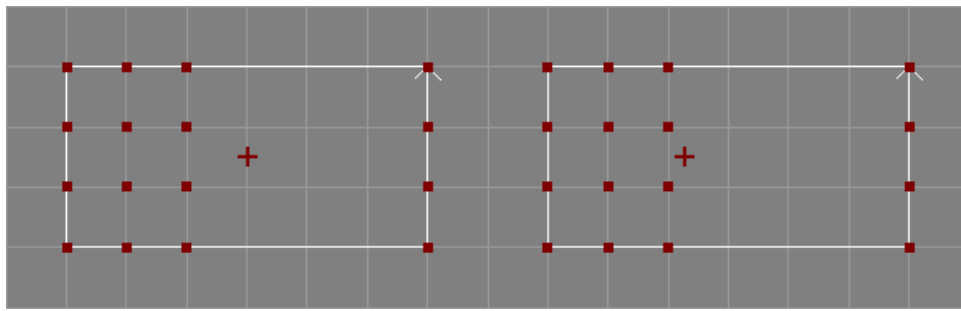


Figure 30: Bounding Box Center Mark (left) and Center of Gravity Mark (right)

The mark may also directly be manipulated using the corresponding view object property (see also section 4.2.2 ViewAttrib Property (page 117)).

Since Ayam 1.21, the mark may also be set without activating the set mark action and without interfering with other modelling actions by double clicking with the rightmost mouse button.

Another way of setting the mark is via the "FindU" special action for curves (see also section 3.20 Finding Points on Curves (page 99)) or the "FindUV" special action for surfaces (see also section 3.21 Finding Points on Surfaces (page 100)) .

The mark can also be used to remember an important point in space and get back to it later using the pan to mark action (keyboard shortcut <. >).

The current mark can be cleared by pressing <D> anytime.

Finally, remember that certain operations like e.g. changing the view type will also clear the mark.

¹ Since 1.30.

3.6 Intermediate Parameter GUIs

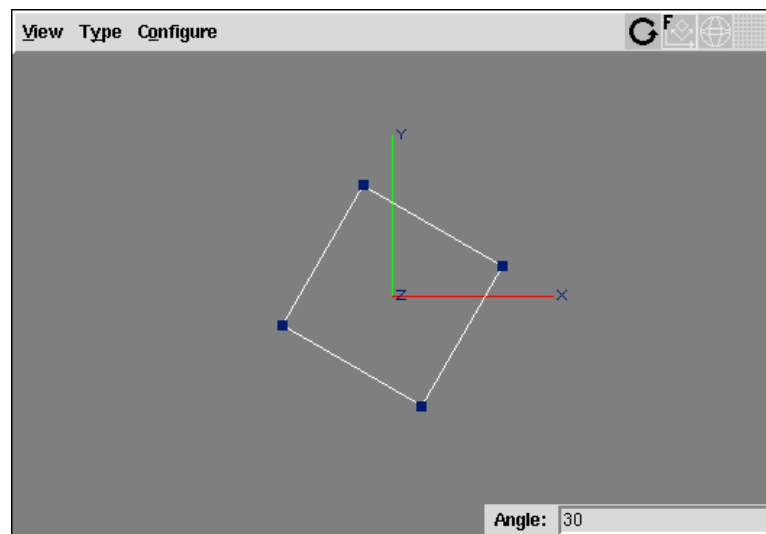


Figure 31: Intermediate Parameter GUI Example

Several interactive modelling actions support intermediate parameter GUIs.¹ Those are simple entry fields that appear in the lower right corner of the view window when e.g. a number key is pressed while the action is active and allow to parameterise the action in quick and exact fashion. See also the image above, where the intermediate parameter GUI for the Rotate action is shown.

The parameter value can be committed using the `<Enter>` or `<Return>` key.

The parameter GUI can also be used multiple times / kept open when the `<Shift>` key is held down while committing the parameter value.

Currently only actions that require one parameter are supported.

¹ Since 1.27.

3.7 Moving Objects or Selected Points

Using the modelling action "Move" (shortcut: <m>) selected objects or selected (tagged) points can be moved.

Note that the objects/points will be moved in the XY-plane for Front-views, the ZY-plane for Side-views, and the XZ-plane for Top-views only, no matter how the view is rotated.

The move action may be restricted to a certain axis by pressing <x>, <y>, or <z> right after the <m>.¹

For an overview of the move actions see also the table below.





Name	Shortcut	Icon
Move	<m>	
Move X	<mx>	
Move Y	<my>	
Move Z	<mz>	

Table 7: Move Actions Overview

See also the documentation of the corresponding scripting interface commands [6.2.10 movOb \(page 384\)](#) and [6.2.10 movPnts \(page 384\)](#).

¹ Since 1.17.

3.8 Rotating Objects or Selected Points

Using the modelling action "Rotate" (shortcut: <r>) objects or selected (tagged) points can be rotated.

The axis of rotation is always parallel to the Z-axis in Front-views, the Y-axis in Top-views, and the X-axis in Side-views of the local object coordinate system. The orientation of the object coordinate system may change in respect to the world coordinate system if a sequence of rotate modelling actions is applied.

The rotate action may be restricted to a certain axis by pressing <x>, <y>, or <z> right after the <r>.¹ If a restricted rotate action is active, the rotation axis used will be displayed in the view as stippled color coded line.

In object local views, using a rotation action will not immediately re-align the view.

Note that if multiple objects are selected, each object is rotated around the center of its own local coordinate system. Only the Rotate_X, Rotate_Y, or Rotate_Z property of the selected objects will be changed by this action.





Name	Shortcut	Icon
Rotate	<r>	
Rotate X	<rx>	
Rotate Y	<ry>	
Rotate Z	<rz>	

Table 8: Rotate Actions

See also the documentation of the corresponding scripting interface commands [6.2.10 rotOb \(page 384\)](#) and [6.2.10 rotPnts \(page 385\)](#).

¹ Since 1.34.

3.9 Rotating Objects or Selected Points about a Point

Using the rotate about modelling action objects or the selected (tagged) points of the selected objects can be interactively rotated about a specific point in space.

The rotation axis is determined in the same way as for the normal rotate action.

The rotate about action may be restricted to a certain axis by pressing `<x>`, `<y>`, or `<z>` right after the `<a>` or setting the mark.¹ If a restricted rotate action is active, the rotation axis used will be displayed in the view as stippled color coded line.

In object local views, using a rotation about action will not immediately re-align the view.

To start the rotate about action, invoke the normal rotate action, then press `<a>`. See also the table below.





Name	Shortcut	Icon
Rotate About	<code><ra></code>	
Rotate X About	<code><rax></code>	
Rotate Y About	<code><ray></code>	
Rotate Z About	<code><raz></code>	

Table 9: Rotate About Actions

To rotate about a different point, the intermediate set mark action must be restarted (simply press `<a>` again).

After setting the mark, the action works the same way as the Rotate action, except that it rotates the selected object(s) or points about the mark. This, consequently, also works with multiple selected objects. Note that this action does not only change the `Rotate_X`, `Rotate_Y`, or `Rotate_Z` properties of the selected objects, but also the `Translate_X`, `Translate_Y`, or `Translate_Z` properties.

To avoid degenerated coordinates due to roundoff errors it is highly suggested to use a grid or the intermediate parameter GUI with this action.

See also the section [6.5.12 Automatic About Center Actions \(page 451\)](#) for a script, that modifies the rotate action to rotate about the current selections center automatically.

¹ Since 1.34.

3.10 Scaling Objects or Selected Points

There are several different actions available to interactively scale objects or the selected (tagged) points of the selected objects:

The modelling action "Scale 3D" (shortcut: <S>, note the capital S!) scales all three axes of the selected objects or the selected (tagged) points of the selected objects by the same factor.

The modelling action "Scale 2D" (shortcut: <s>) scales just two axes of the selected objects or the selected (tagged) points of the selected objects. Those axes are XY in a Front-view, ZY in a Side-view, and XZ in a Top-view.

It is also possible to restrict the scaling of objects or selected points to just one axis. For that <x>, <y>, or <z> must be pressed right after <s> (e.g. <sx> for scale 1D about x).

Since Ayam 1.17, direct access to the 1D scale modelling actions "Scale X" (old shortcut: <x>), "Scale Y" (old shortcut: <y>), and "Scale Z" (old shortcut: <z>) is no longer available.

The modelling action "Stretch 2D" (shortcut: <T>) works much like "Scale 2D" but the scale factor for each axis may be different. Never start this action by a click near one of the axes to be changed, as this will cause very big scale factors for the other axis. Try it first with a centered box by starting from one of the box corners, then try it once starting on the X-axis.

For an overview of the scale actions see also the table below.

Name	Shortcut	Icon
Scale 3D	<S>	
Scale 2D	<s>	
Scale 1D X	<sx>	
Scale 1D Y	<sy>	
Scale 1D Z	<sz>	
Stretch 2D	<T>	

Table 10: Scaling Actions Overview

See also the documentation of the corresponding scripting interface commands [6.2.10 scalOb \(page 384\)](#) and [6.2.10 scalPnts \(page 385\)](#).

3.11 Scaling Objects or Selected Points about a Point

Using the scale about modelling actions objects or the selected (tagged) points of the selected objects can be interactively scaled about a specific point in space.

To start a scale about action, invoke the normal scale action, then press <a> (e.g. <sa> for scale 2D about, <sya> for scale 1D Y about). For an overview of the scale about actions see also the table below.







Name	Shortcut	Icon
Scale 3D About	<Sa>	
Scale 2D About	<sa>	
Scale 1D X About	<sxa>	
Scale 1D Y About	<sya>	
Scale 1D Z About	<sza>	
Stretch 2D About	<Ta>	

Table 11: Scaling About Actions Overview

To scale about a different point, the intermediate set mark action must be restarted (simply press <a> again).

After setting the mark, the action works the same way as the scale action, except that it scales the selected object(s) or points about the mark. This, consequently, also works with multiple selected objects. Note that this action does not only change the Scale_X, Scale_Y, or Scale_Z properties of the selected objects, but also the Translate_X, Translate_Y, or Translate_Z properties.

Also note, that the three dimensional scaling about the mark occurs in all three dimensions, however, in the intermediate interactive set mark action only two dimensional coordinates can be specified using a single mouse click. To specify a true three dimensional mark the corresponding view object attributes can be used or a point can be selected and the mark set to it (shortcut <C>). See [section 3.5 Setting the Mark \(page 82\)](#) for other means of setting a three dimensional mark.

To avoid degenerated coordinates due to roundoff errors it is highly suggested to use a grid or the intermediate parameter GUI with this action.

See also the [section 6.5.12 Automatic About Center Actions \(page 451\)](#) for a script, that modifies the scale actions to scale about the current selections center automatically.

3.12 Editing Points

To edit the points of an object two actions ("Edit" and "Numeric Point Edit") are available. Those actions may be applied to objects that support single point editing only. Such objects usually draw their selectable points using small white rectangular handles when a modelling action is active (see also the image below).

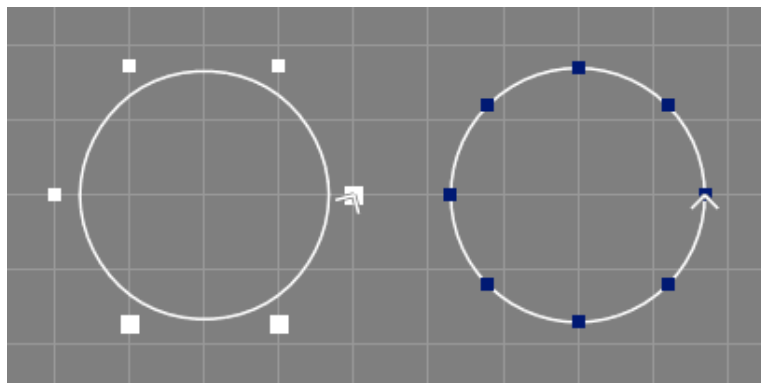


Figure 32: Objects With Editable (left) and With Read-only Points (right)

In contrast to e.g. the move action, all point edit actions require the handle of the point to be picked directly. Only points within a certain distance from the mouse pointer click position will be considered picked (see preference option "Modelling/PickEpsilon"). Points that would be picked will flash in a different color when the mouse pointer moves over them (this is also controlled by the preference option "Modelling/FlashPoints").

For an overview of the point edit actions see also the table below.

Name	Shortcut	Icon
Edit	<e>	
Numeric Edit	<E>	

Table 12: Edit Points Overview

The modelling action "Edit" (shortcut: <e>) works much like the move action, but it moves single points instead of objects. Since Ayam 1.18 it is possible to edit points of multiple selected objects (in former versions only points from the first selected object were considered).

If a multiple point is edited, this action modifies all single points that make up the multiple point, i.e. it is not possible to move single points apart from a multiple point using this action. Temporarily disable the "CreateMP" property or explode the multiple point to do that.

If the preference option "Modelling/EditSnaps" is enabled, the picked point will be moved to the nearest grid coordinates first, otherwise the grid just controls the displacement for the edit action. The snapping can occur in 2D or 3D, depending on the preference option "Modelling/Snap3D". Moreover, while snapping a point, the mouse pointer will warp to the new position of the point (so that the user does not lose track of it). Warping the mouse pointer currently does not work on Mac OS X Aqua.

When the "Ctrl"-key is held down, while selecting curve points, this action will insert a new control point into the curve and allow to move this new point to a new position.¹

The modelling action "Numeric Point Edit" (shortcut: <E>) starts an intermediate point selection action and upon a click with the mouse opens a small window where the coordinates of the selected editable point(s) may be changed directly by entering new coordinate values on the keyboard (see image below). If there are already selected points, the dialog window will be opened immediately.²

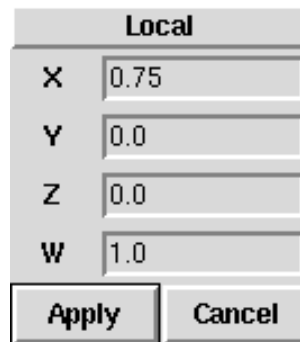


Figure 33: Numeric Point Edit Dialog

Nothing will be changed unless the "Return" key is pressed in a coordinate entry field or the "Apply" button is used.

In contrast to the normal point selection action, where further clicks add points to the selection, clicking on new points, while the edit point dialog is open, deselects the old point(s), selects the new point and loads its coordinate values into the entry fields. In case of multiple selected objects, the deselection is only performed on the object with a new selection.

Note that the w coordinate setting will be ignored for selected points that do not have weight information (are not rational).

Using the small menu on top of the coordinate window one can determine whether editing takes place in local/object space or global/world space.

This modelling action may affect multiple points of multiple selected objects.³

Snapping points of different objects together is now possible too: Just select two objects, start numeric point editing (press <E>), pick a point on the first object (where a point of the second object should be snapped to), the point edit window opens, now drag select the point of the second object (do *not* use a single click for selection as this would also load new coordinates) and press apply.

The numeric point editing action also supports read only points, their coordinates may be retrieved, but applying any values will have no effect.⁴

When applying changes, single coordinate entry fields may be empty to prevent a change of the corresponding coordinate value.⁵

If a multiple point is edited, this action modifies all single points that make up the multiple point, i.e. single points can not be moved apart from a multiple point using the numeric point edit action. Temporarily disable the "CreateMP" property or explode the multiple point to do that.

Notice that the numeric point editing dialog may stay open all the time.⁶

¹ Since 1.32. ² Since 1.26. ³ Since 1.18. ⁴ Since 1.18. ⁵ Since 1.20. ⁶ Since 1.4.

Furthermore, it is not necessary that the original object stays selected while working with the numeric point edit dialog, other objects may be selected to e.g. infer new point coordinates from their properties and apply them to the original object. Furthermore notice that the coordinate values displayed in the numeric point editing window will not update when the point is modified by another modelling action (e.g. in another view). Simply click on the point again in a view where the numeric point editing action is active, to update the coordinate values in the numeric point editing dialog.

Even though the dialog may display point coordinates in degraded accuracy (due to floating point to string conversion) the original point data is unchanged and its accuracy is not affected as long as the new data is not applied. See also the discussion of the "TclPrecision" preference option in section 2.10.5 Miscellaneous Preferences (page 71)).

3.12.1 Expression Support

The numeric point edit value fields support Tcl expressions. This allows e.g. to set a point to exactly¹ 1/3 by entering:

```
[expr 1.0/3.0]
```

3.12.2 Context Menu

The numeric point edit dialog also has a context menu.² The entries in this menu allow to:

- "Clear" all entries,
- "Reset" all entries to the last fetched values (undo edits),
- "Fetch Mark", set all entries to the mark coordinates of the current view (W is cleared),
- "Fetch First", set all entries to the first of the currently selected points, or
- "Fetch Last", set all entries to the last of the currently selected points.

3.12.3 Keyboard Shortcuts

The following keyboard shortcuts are in effect for the numeric edit dialog:

- <x>, <y>, <z>, <w>, move the focus to the respective entry.
- <Return> / <Enter>, apply all changes.
- <Shift-Return> / <Shift-Enter>, apply all changes and close the dialog.
- <Shift-Delete>, clear all entries.
- <Esc> close the dialog.

¹ in the limits imposed by the floating point data type double ² Since 1.21.

3.13 Editing Weights

In the NURBS context, weights are mainly used to construct perfect circles, circular arcs, and spheres. However, they also present another way of influencing the curves or surfaces shape in relation to the control point with the weight: Weights smaller than 1.0 repel the curve or surface from the control point, whereas weights larger than 1.0 attract the curve or surface to the control point. See also the following image:

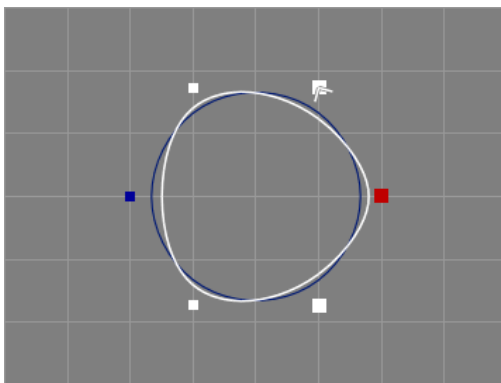


Figure 34: Weights Example (Blue: $w = 0.5$, Red: $w = 2.0$)

To edit the weights of an object two actions ("Edit Weights" and "Reset Weights") are available. Those actions may be applied to objects that support single point editing and support rational coordinates only. Those are objects of type NCurve, NPatch, and PatchMesh.

The modelling action "Edit Weights" (shortcut: $\langle w \rangle$) allows to change the w coordinate of a single point by dragging the mouse left or right. The initial click always resets the weight to 1.0. For every 10 pixels the mouse is dragged, a value of 0.1 will be added to or removed from the weight respectively. If a grid is active, the 0.1 will be multiplied by the grid value so that a grid of 0.5 will add/remove weight in terms of 0.05 increments/decrements.

When the $\langle \text{Control} \rangle$ key is held down while picking the control point to edit, the weight will not be reset.¹

The modelling action "Reset Weights" (shortcut: $\langle W \rangle$) resets the w coordinate of the picked point to 1.0. Drag selection of points to reset is also possible.²

In both actions the current weight of a control point is visualized by color: repelling weights ($w < 1.0$) are drawn in tones of blue that get darker as the weight approaches 0.0 and attracting weights ($w > 1.0$) are drawn in tones of red that get darker as the weight approaches 3.0. Control points with zero or invalid weights are drawn in black. Control points with negative weights are drawn in green. Control points with no weight ($w = 1.0$) stay white, see also the figure below.

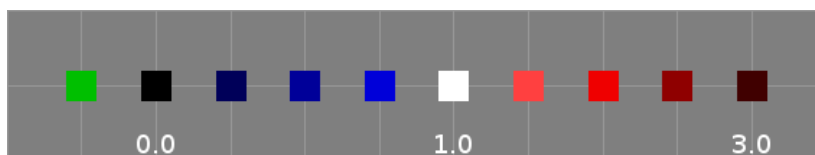


Figure 35: Weight Scale

The weights may also be reset for all points using the keyboard shortcut: $\langle \text{Ctrl-W} \rangle$.

¹ Since 1.25. ² Since 1.25.

Also note that, if the preference option "RationalPoints" is set to *homogeneous*, changing the weight of a control point also changes its (displayed) position. See also section [2.10.2 Modelling Preferences](#) (page 62).

For an overview of the weight edit actions see also the table below.


Name	Shortcut	Icon
Edit Weights	<w>	
Reset Weights	<W>	

Table 13: Edit Weights Overview

3.14 Snapping Points to the Grid

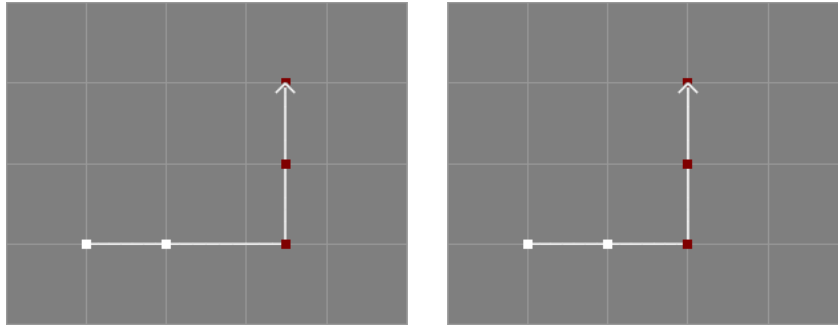


Figure 36: Snapping Points to the Grid

There are two actions available for snapping points to the current grid of a view.¹

The actions are initiated using the shortcuts `<g>` and `<G>`. If an object has selected (tagged) editable points, only those points will be snapped to the grid, otherwise all editable points of the object will be snapped to the grid (see also the image above). If `<g>` is used, the snapping only occurs in the modelling plane associated with the view (i.e. in 2D). To snap all three coordinate values to the grid use `<G>`. Note that the snapping also occurs, if the view has the preference option "Use Grid" turned off. This action can be used without affecting other active actions.

3.15 Snapping Objects to the Grid

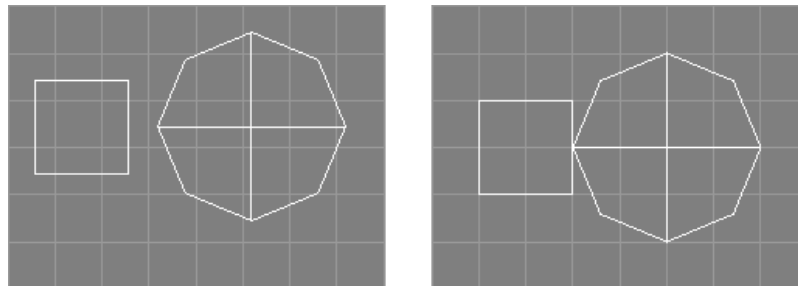


Figure 37: Snapping Objects to the Grid

There are two actions available for snapping objects to the current grid of a view.²

The actions are initiated using the shortcuts `<f>` and `<F>`.

If `<f>` is used, the snapping only occurs in the modelling plane associated with the view (i.e. in 2D). To snap all three coordinate values to the grid use `<F>`. Note that the snapping also occurs, if the view has the preference option "Use Grid" turned off. This action can be used without affecting other active actions.

¹ Since 1.11. ² Since 1.26.

3.16 Snapping Points to the Mark

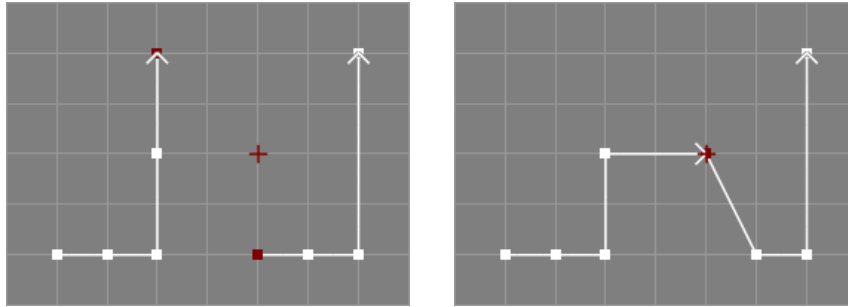


Figure 38: Snapping Points to the Mark

The snap points to mark action moves all selected points to the current mark.¹ See also the image above.

This action is initiated using the shortcut `<M>`. All selected editable points of the selected objects are snapped to the current mark coordinates (see section 3.5 [Setting the Mark \(page 82\)](#) for more information about the mark) immediately. This action can be used without affecting other active actions.

3.17 Snapping Objects to the Mark

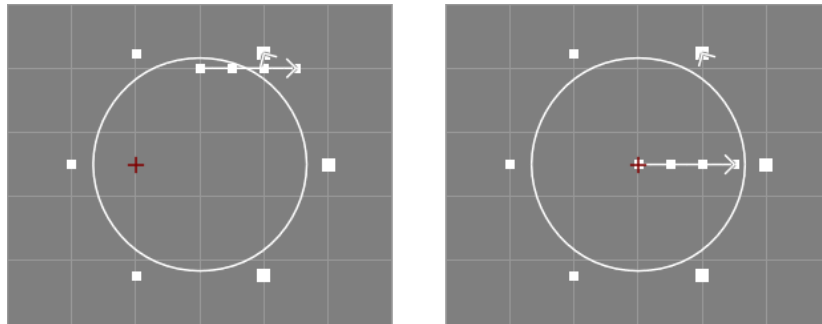


Figure 39: Snapping Objects to the Mark

The snap objects to mark action moves all selected objects to the current mark.² See also the image above.

This action is initiated using the shortcut `<K>`. All selected objects are snapped to the current mark coordinates (see section 3.5 [Setting the Mark \(page 82\)](#) for more information about the mark) immediately, providing an easy way to move objects over long distances or to just put an object "here". This action can be used without affecting other active actions.

¹ Since 1.18. ² Since 1.18.

3.18 Inserting or Deleting Points

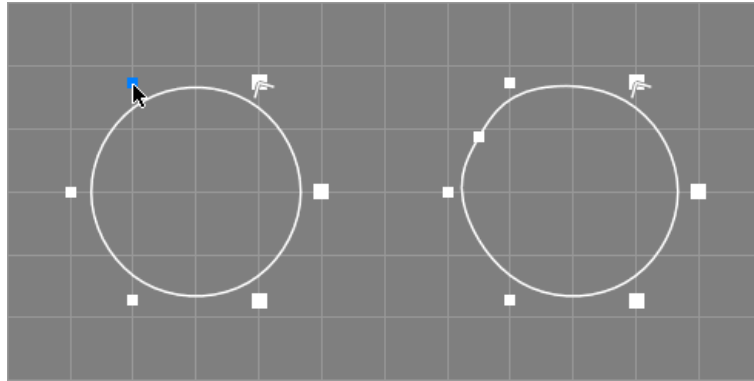


Figure 40: Inserting Points

The modelling action "Insert Point" (shortcut: `<i>`) may be applied to NURBS, interpolating, and approximating curves (objects of type NCurve, ICurve, and ACurve) only. A new control point will be inserted in the curve right after the picked point. The new point will be inserted in the middle between the selected point and the next point, changing the shape of the curve. See also the image above.

It is also possible to insert control points into certain types of NURBS curves without changing their shape using knot insertion; see also the insert knot tool section [5.3.13 Insert Knot Tool \(page 291\)](#).

If the `<Ctrl>`-key is held down while clicking on a point, the new point can interactively be placed on a freely chosen position.¹ This is done by initially giving the new point the coordinates of the clicked point, from there, the new point can be dragged to the final position.

The modelling action "Delete Point" (shortcut: `<d>`) may be applied to NURBS, interpolating, and approximating curves (objects of type NCurve, ICurve, and ACurve) only. The selected control point will be deleted from the curve. Deleting points from a curve with knot type "Custom" may currently lead to an incorrect knot sequence, please check and correct the new sequence manually. See also the table below.



Name	Shortcut	Icon
Insert Points	<code><i></code>	
Delete Points	<code><d></code>	

Table 14: Insert/Delete Points Actions

¹ Since 1.25.

3.19 Manipulating the Multiplicity of Points

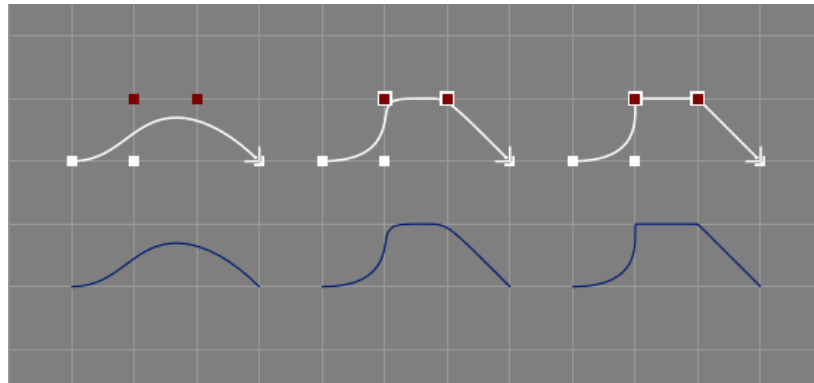


Figure 41: Successively Increasing the Multiplicity of Selected Points

To manipulate the multiplicity of selected NURBS curve control points there are two modelling actions available.¹

The actions are initiated using the shortcuts `<*>` (to increase the multiplicity) and `</>` (to decrease the multiplicity).

Increasing the multiplicity of a control point can be used to easily construct sharp corners and linear curve segments (see also the image above).

The multiplicity of the selected control points will only be raised to the order of the curve and, conversely, never be decreased below 1. Trying to increase/decrease beyond those limits will not result in any error.

The selected points stay selected, so that the actions can be applied multiple times.

Both actions can be invoked anytime without breaking other actions.

¹ Since 1.20.

3.20 Finding Points on Curves

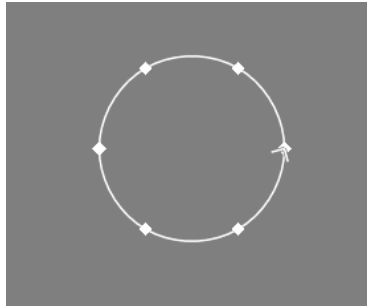


Figure 42: Breakpoint Display in Find Point on Curve Action

The modelling action "FindU" (shortcut: <u>) may be applied to NURBS curves (objects of type NCurve or objects that provide NCurve objects) only. This action may be used to get the corresponding parametric value u from an arbitrary point on a curve. For every picked point the appropriate value for u is calculated, stored in the global variable "u", and additionally written to the console. The mark is set to the coordinates of the point on the curve that results from evaluating the curve at the parametric value u . See also section 3.5 Setting the Mark (page 82) for more information about the mark. Remember to exactly pick a point on the curve or nearby, otherwise the calculation may fail.

This action can also be used to get the curve coordinates from a distinct value (breakpoint) from the knot vector.¹ Those points are displayed as rhombuses and can simply be picked, see also the image above. If a breakpoint is picked, its parametric value is put into the global variable u and the mark is set to the corresponding, three dimensional, point on the curve. Drag selection of breakpoints is also possible. A drag selection of multiple breakpoints, either with one drag or a second drag with the <Ctrl> key held down, leads to a knot range selection which will be stored in a "UMM" tag.² The selected range will be displayed as a pair of bracket symbols ("[" and "]"") on the curve when the "FindU" action is active. A selected knot range can then e.g. be perused by the knot refine tool. An empty drag selection will remove the current knot range selection by removing the "UMM" tag.

See also the table below.

Name	Shortcut	Icon
Find U	<u>	

Table 15: Find Point on Curve Action

¹ Since 1.25. ² Since 1.29.

3.21 Finding Points on Surfaces

The modelling action "FindUV" (shortcut: <U>) may be applied to NURBS surfaces (objects of type NPatch or objects that provide NPatch objects) only. This action may be used to get the corresponding parametric values u and v from a point on a surface. For every picked point the appropriate values for u and v are calculated, stored in the global variables " u " and " v ", and additionally written to the console. The mark is set to the coordinates of the point on the surface that results from evaluating the surface at the parametric values u and v . See section 3.5 [Setting the Mark \(page 82\)](#) for more information about the mark. Remember to exactly pick a point on the surface or nearby, otherwise the calculation may fail. As the icon suggests, this action works best in the shaded drawing mode.

This action can also be used to get the surface coordinates from a pair of distinct values (breakpoints) from the knot vectors.¹ Those points are displayed as rhombuses and can simply be picked. If a breakpoint is picked, its parametric values are put into the global variables u and v and the mark is set to the corresponding, three dimensional, point on the surface. Drag selection of breakpoints is also possible.

See also the table below.


Name	Shortcut	Icon
Find UV	<U>	

Table 16: Find Point on Surface Action

¹ Since 1.29.

3.22 Selecting Boundary Curves

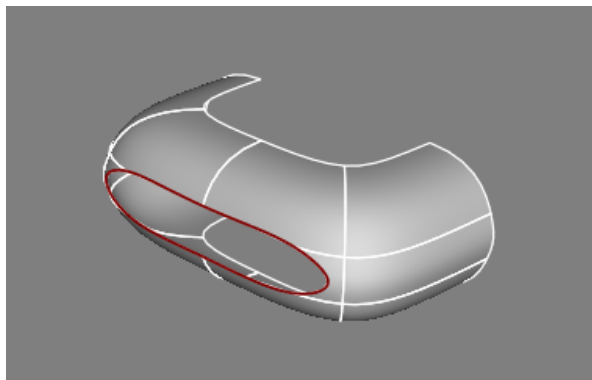


Figure 43: Display of Selected Boundary Curve (red) Example

The modelling actions "SelBnd" (shortcut: <n>) and "SelBndC" (shortcut: <k>) may be applied to NURBS surfaces (objects of type NPatch or objects that provide NPatch objects) only.¹

These actions may be used to select a boundary curve for further processing by e.g. the various tool object creation tools (see section 5.4 Surface Creation Tools (page 308)).

The selected boundaries will be drawn in red color when one of these actions is active. See also the image above, where a trim boundary of a Trim object has been selected.

The selection will be stored in SB tags and thus survive scene saving and restoring. See also section 4.11.22 SB (Selected Boundary) Tag (page 270).

Drag selection is supported by both actions. An empty drag selection de-selects all currently selected boundaries.

Holding down the <Ctrl>-key while dragging the mouse removes all selected boundaries within the rectangular region defined by the drag from the selection.

See also the table below.



Name	Shortcut	Icon
SelBnd	<n>	
SelBndC	<k>	

Table 17: Select Boundary Curve Actions

¹ Since 1.29.

3.23 Interactively Splitting Curves

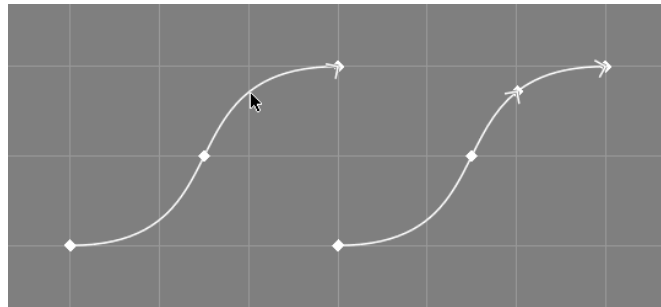


Figure 44: Interactive Curve Splitting

The modelling action "Split Curve" (shortcut $\langle | \rangle$) may be applied to NURBS curves (objects of type NCurve) only. Using this action a NURBS curve may be split into two new curves at a point on the curve that is specified by picking a point on the curve; see also the corresponding NURBS curve tool [5.3.18 Split Tool](#) (page 297).

Remember to exactly pick a point on the curve or nearby otherwise the calculation of the parametric value for the split will fail. The selected curve will be changed by this action, and a new curve will be created. It is currently not possible to undo the changes of a split!

This action also displays the distinct values (breakpoints) from the knot vector and allows to pick them just like the "FindU" action.¹ Grids are not utilised by this action. See also the table below.


Name	Shortcut	Icon
Split	$\langle \rangle$	

Table 18: Split Curve Action

¹ Since 1.25.

3.24 Selecting/Tagging Boundary Points

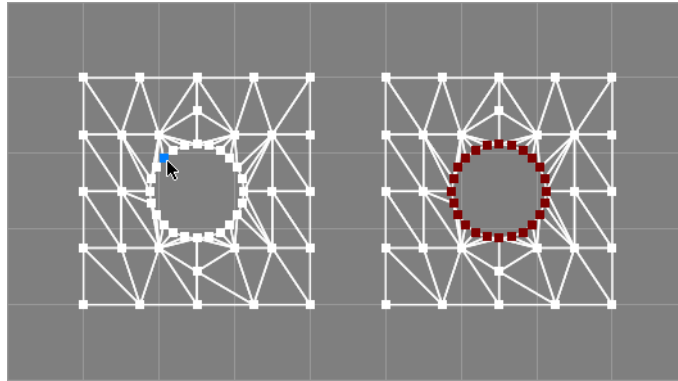


Figure 45: Selecting Boundary Points

The modelling action "Select Boundary Points" (shortcut) may be applied to PolyMesh objects only.

This action can be used to select all control points of a PolyMesh boundary after picking a point on that boundary.¹ Selected boundaries are needed for the "Connect" polymesh tool, see also [2.2 PolyMesh \(page 35\)](#). See also the image above.

Note that the PolyMesh object must be optimized for best results. See also the table below.


Name	Shortcut	Icon
Select Boundary		

Table 19: Select Boundary Points Action

¹ Since 1.23.

3.25 Selecting Faces

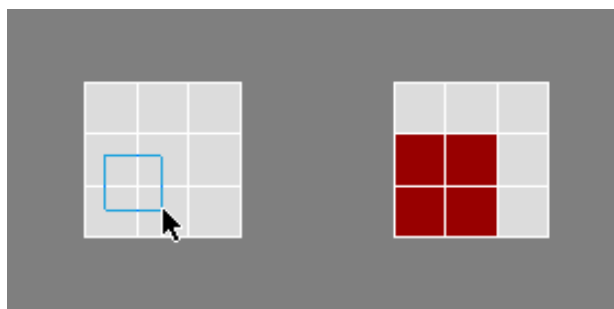


Figure 46: Selecting Faces Example

The modelling action "Select Faces" (shortcut `<f>`) may be applied to PolyMesh, SDMesh, and SDNPatch objects only.

This action can be used to select individual faces of a mesh for the application of further tools.

Normal mouse clicks toggle the selection state of a face.

Drag selection is also possible but just adds to the selection. An empty drag selection de-selects all currently selected faces.

Holding down the `<Ctrl>`-key while dragging the mouse removes all selected faces within the rectangular region defined by the drag from the selection.

Selected faces are shaded in red in the shaded drawing modes, in wireframe drawing mode a red point will be displayed in the barycenter of each selected face, see also the image above.

In the shaded drawing modes, a successful selection requires front facing normals, which can not be guaranteed to be present when e.g. polygonal data is created by conversion from arbitrary NURBS data. The flip loops polymesh tool can be used to fix this.

In the wireframe drawing mode, the normals are ignored, but this can lead to unwanted face selections (on the front *and* back of an object).

See also the table below.


Name	Shortcut	Icon
Select Face	<code><f></code>	

Table 20: Select Face Action

See also the corresponding scripting interface command `selFace` in section 6.2.6 Selecting Faces (page 378).

3.26 Editing in Local Spaces

Normally, all editing takes place in world space and the input plane of all modelling actions is constrained to the world XY-, ZY-, or XZ-plane (depending on the type of the view – "Front", "Side", or "Top" respectively).

However, if a view is *aligned* and switched to *local*, editing in object space is possible. The input plane of an aligned local view will match the XY-, ZY-, or XZ-plane of the local object space, again depending on the type of the view. Editing and other modelling actions take place in that plane.

This means that in an aligned local view a planar parameter curve of a Skin object may be edited and the parameter curve is guaranteed to remain planar all the time, even if both objects, curve and Skin, are rotated and scaled arbitrarily via their transformation attributes.

Furthermore, grids will also act as if defined in local object space. Note that in contrast to their normal behaviour, grids can also be scaled differently in X-window and Y-window coordinates in aligned local views (if the local object space is deformed this way).

To make a view *object-local* and aligned the object must be selected first. Now, pressing <1> *twice* makes the current view object-local (in external views <Ctrl+1> can also be used). In Ayam versions prior to 1.18 the view also needed to be aligned manually using <L> (or <Ctrl+a> in external views). Since Ayam 1.18, the view is automatically aligned to the selected object or current level when cycling through the global/local modes. Manually aligning a view may still be necessary after certain changes to the camera of the view e.g. by the rotate view action. Note that realigning is also always possible by cycling the modelling mode three times using <1>. But let us get back to our local view.

To illustrate local views a little bit further, see the following example images:

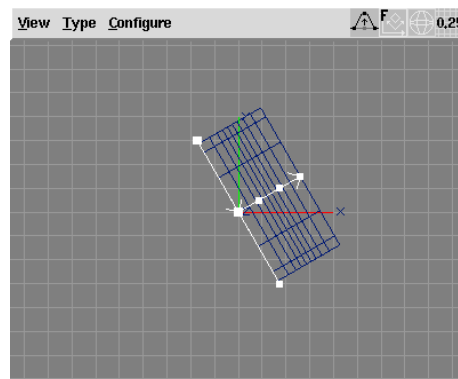


Figure 47: Global Front View with Rotated Sweep

A Sweep object with a circular B-Spline curve as cross section (rotated about the Y-axis by 90 degrees using the transformation attributes) and a straight standard curve as trajectory. The Sweep itself is rotated about the Z-axis by 30 degrees. The view is a front view, the cross section and trajectory are both selected. Note how inadequate the grid spacing would be to edit the trajectory curve (it is e.g. near impossible to edit the curve and keep it straight).

» Press <1>.

The view has been switched to level-local using a single press of the <1> key (see the modelling mode icon, it is displaying a **L** in the lower right corner). The view is now aligned to the 30 degree rotated space of the Sweep object (note the Root object coordinate system, it is tilted). The grid is also rotated (with respect

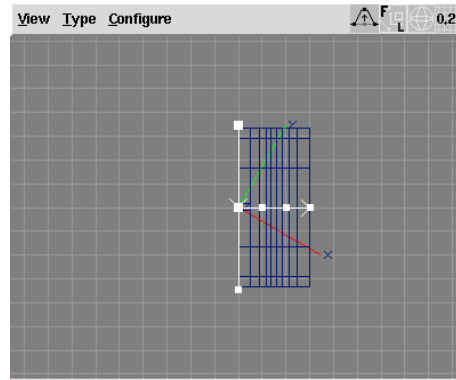


Figure 48: Local (Level) Aligned View

to the root coordinate system) but now much more useful for editing the trajectory, in fact it is perfectly aligned to the trajectory.

» Press <1> again.

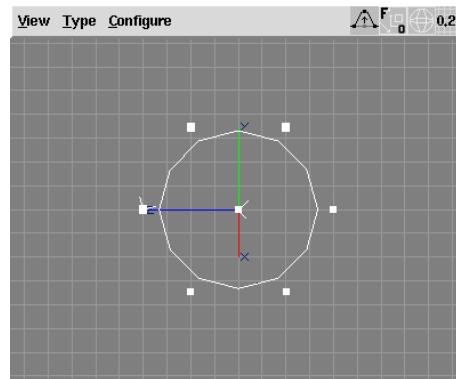


Figure 49: Local (Object) Aligned View

Now the view is object-local and aligned to the first of the selected objects (the cross section curve); look at the modelling mode icon, it is now displaying a **O** in the lower right corner. As the selected cross section curve was rotated by 90 degrees around the Y-axis, the front view now displays a circle. Note again the orientation of the Root object coordinate system. The cross section curve can now be edited safely, it will stay planar.

Note, that object-local aligned views only align to the first of multiple selected objects; if all selected objects share the same transformation attributes, everything is fine, but if not, some objects will be misaligned.

Also remember: when a view is level-local, changing the current level does not automatically realign the view. If the view is object-local, changing the object selection will also possibly lead to a misaligned view. To realign the view after changing the current level or object selection simply press <L>. Note that aligning the view changes the aim point of the view camera to the origin of the respective local coordinate system which might not be appropriate for modelling purposes; so, after a selection change, one may want to also zoom the view to the newly selected object: <LO>.

Finally, note that aligning views only works for spaces defined by the transformation attributes, it is *not* possible to align to e.g. a planar curve that has the default transformation attributes and is rotated via the control points.

4 Objects, Properties, and Tags

This section informs about the different object types of Ayam and about the property GUIs that appear in the properties section of the main window if a single object and a property have been selected.

Note that the "Help on object" and "Help on property" main menu entries in Ayam can be used to directly jump to the appropriate sub-section of this part of the documentation.

Documentation on the standard properties (Transformations, Attributes, Material, Shader, and Tags) can be found in [section 4.10 Standard Properties \(page 251\)](#).

Furthermore, this section contains information about all tag types, see [section 4.11 Tags \(page 258\)](#).

In the next sections general object capabilities will be briefly documented in tables like this:

Type	Parent of	Material	Converts to/Provides	Point Edit
Object Type	No / Object Type ⁺ / [*]	Yes/No	N/A / Children / Object Type ⁺ / [*]	Yes/No [*]

Table 21: Object Capabilities Template

The capabilities are:

- **Type:** the type name as displayed in the object tree view and understood by the "crtOb" scripting interface command;
- **Parent of:** the object is a parent object (can have child objects of the designated type),
⁺ – multiple child objects may be present (one or many),
^{*} – multiple child objects must be present (many),
 note that the type of the child object(s) does not need to match directly, the child(ren) must rather provide an object of the appropriate type (see also [section 8.2 The Modelling Concept Tool-Objects \(page 511\)](#));
- **Material:** the object can be associated with a material;
- **Converts to/Provides:** type of converted or provided objects (Children means, the provided objects of the children are delivered upstream),
⁺ – multiple objects may be provided (one or many),
^{*} – multiple objects will be provided (many);
- **Point Edit:** the object has editable points,
^{*} – read only points are supported.

Example:

Type	Parent of	Material	Converts to/Provides	Point Edit
Revolve	NCurve	Yes	NPatch ⁺	No*

Table 22: Object Capabilities Example

Explanation:

- **Type:** the Revolve object has the type name "Revolve", i.e. it can be created from scripts using the command "crtOb Revolve";
- **Parent of:** the Revolve object has one NCurve (or NCurve providing object) as child;
- **Material:** the Revolve object can be associated with a material;
- **Converts to/Provides:** the Revolve object converts to (and provides) one or multiple NPatch objects;
- **Point Edit:** the Revolve object has no editable points, it does not support single point modelling actions; however, read only points are supported (the control points of the underlying NPatch object can be read and selected).

4.1 Object Types Overview

This section provides an overview on the object types available in Ayam (since there are so many). The object types are grouped by application in the following sections.

4.1.1 Scene Organization

These objects help to organize/structure the scene; view, camera, light, and material objects are also listed here (even though deserving an own section each):

Type	Parent of	Material	Converts to / Provides	Point Edit
Root	View ⁺	No	N/A	No
Level	Any ⁺	Yes	N/A / Children ⁺	No
Clone	Any ⁺	No	Children ⁺	No*
Mirror	Any ⁺	No	Children ⁺	No*
Instance	No	No	Master	No*
Select	Any ⁺	No	N/A / Children ⁺	No
RiInc	Any	No	N/A	No
RiProc	No	No	N/A	No
View	NPatch	No	N/A	Yes
Camera	No	No	N/A	Yes
Light	No	No	N/A	Yes
Material	No	N/A	N/A	No

Table 23: Scene Organization

For complete documentation, see section [4.2 Scene Organization Objects \(page 113\)](#).

4.1.2 CSG / Solid Primitives

These objects serve as geometric primitives in CSG hierarchies:

Type	Parent of	Material	Converts to / Provides	Point Edit
Box	No	Yes	NPatch*	No*
Sphere	No	Yes	NPatch ⁺	No*
Disk	No	Yes	NPatch	No*
Cone	No	Yes	NPatch ⁺	No*
Cylinder	No	Yes	NPatch ⁺	No*
Torus	No	Yes	NPatch ⁺	No*
Paraboloid	No	Yes	NPatch ⁺	No*
Hyperboloid	No	Yes	NPatch ⁺	No*

Table 24: CSG / Solid Primitives

For complete documentation, see section [4.3 CSG / Solid Primitives \(page 142\)](#).

4.1.3 Freeform Curves

These objects are mainly used as child objects for the surface generating tool objects:

Type	Parent of	Material	Converts to/Provides	Point Edit
NCurve	No	No	N/A	Yes
ICurve	No	No	NCurve	Yes
ACurve	No	No	NCurve	Yes
NCircle	No	No	NCurve	No*

Table 25: Freeform Curves

For complete documentation, see section [4.4 Freeform Curve Objects \(page 150\)](#).

4.1.4 Freeform Surfaces

These objects enable direct manipulation of freeform surfaces:

Type	Parent of	Material	Converts to/Provides	Point Edit
NPatch	NCurve ⁺ /Level ⁺	Yes	PolyMesh	Yes
IPatch	No	Yes	NPatch	Yes
APatch	No	Yes	NPatch	Yes
BPatch	No	Yes	NPatch	Yes
PatchMesh	No	Yes	NPatch	Yes

Table 26: Freeform Surfaces

For complete documentation, see section [4.6 Freeform Surface Objects \(page 170\)](#).

4.1.5 Curve Tool Objects

These objects modify existing curves or create new curves:

Type	Parent of	Material	Converts to/Provides	Point Edit
ConcatNC	NCurve ⁺	No	NCurve	No*
ExtrNC	NPatch	No	NCurve	No*
OffsetNC	NCurve	No	NCurve	No*

Table 27: Curve Tool Objects

For complete documentation, see section [4.5 Curve Tool Objects \(page 162\)](#).

4.1.6 Surface Tool Objects

These objects create freeform surfaces from curves or other surfaces:

Type	Parent of	Material	Converts to / Provides	Point Edit
Revolve	NCurve	Yes	NPatch ⁺	No*
Extrude	NCurve ⁺	Yes	NPatch ⁺	No*
Swing	NCurve*	Yes	NPatch ⁺	No*
Sweep	NCurve*	Yes	NPatch ⁺	No*
Birail1	NCurve*	Yes	NPatch ⁺	No*
Birail2	NCurve*	Yes	NPatch ⁺	No*
Skin	NCurve*	Yes	NPatch ⁺	No*
Gordon	NCurve* / Level / NPatch	Yes	NPatch ⁺	No*
Bevel	NCurve ⁺	Yes	NPatch	No*
Cap	NCurve ⁺	Yes	NPatch	No*
Text	No	Yes	NPatch ⁺	No*
Trim	NPatch / NCurve ⁺ / Level ⁺	Yes	NPatch	No*
ConcatNP	NPatch* / NCurve*	Yes	NPatch ⁺	No*
ExtrNP	NPatch	Yes	NPatch ⁺	No*
OffsetNP	NPatch	Yes	NPatch ⁺	No*

Table 28: Surface Tool Objects

For complete documentation, see section [4.7 Surface Tool Objects \(page 182\)](#).

4.1.7 Polygonal and Subdivision Objects

These objects complement the Ayam feature set and allow objects modelled in the polygonal or subdivision modelling paradigms to be included in Ayam scenes:

Type	Parent of	Material	Converts to / Provides	Point Edit
PolyMesh	No	Yes	SDMesh	Yes
SDMesh	No	Yes	PolyMesh	Yes

Table 29: Polygonal and Subdivision Objects

For complete documentation, see section [4.8 Polygonal and Subdivision Objects \(page 225\)](#).

4.1.8 Scripts and Plugins

These objects create/modify arbitrary other objects from scripts or define entirely new object types via the custom object plugin mechanism.

Type	Parent of	Material	Converts to/Provides	Point Edit
Script	Any ⁺	No	Any	No*
MetaObj	MetaComp ⁺	Yes	PolyMesh	No
MetaComp	No	No	N/A	No
SDNPatch	No	Yes	PolyMesh	Yes
SfCurve	No	No	NCurve	No*
SDCurve	No	No	NCurve	Yes
BCurve	No	No	NCurve	Yes

Table 30: Scripts and Plugins

For complete documentation, see section [4.9 Script and Custom Objects \(page 229\)](#).

A number of Script objects scripts are already distributed with Ayam:

Type	Parent of	Material	Converts to/Provides	Point Edit
Helix	No	No	NCurve	No*
Spiral	No	No	NCurve	No*
TCone	No	No	Hyperboloid	No*
CBox	No	No	NPatch	No*
TCircle	No	No	NCurve	No*
Oval	No	No	NCurve	No*
DualSweep	NCurve*	No	NPatch	No*
TSurf	NCurve*	No	NPatch	No*
ExtrudeN	NCurve	No	NPatch	No*
Polyhedron	No	No	PolyMesh	No*

Table 31: Distributed Script Object Scripts

For complete documentation, see section [6.6 Distributed Script Objects \(page 460\)](#).

4.2 Scene Organization Objects

These objects help to organize/structure the scene; view, camera, light, and material objects are also listed here (even though deserving an own section each).

4.2.1 Root Object

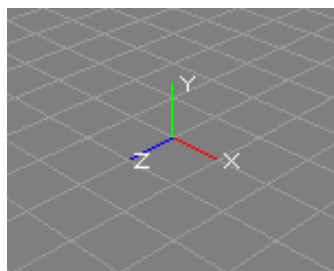


Figure 50: Root Object

There is always exactly one Root object in the scene. This object is something special in that it can not be deleted or copied. The Root object holds rendering options global to the scene like `RiOptions`, atmosphere and imager shaders. Furthermore, all currently open view windows are represented as child objects of the Root object.

If the Root object is hidden, the little red/green/blue coordinate system will not be drawn in any view.

The Root object also aids in per-scene window geometry management using `SaveMainGeom` and `SavePanelLayout` tags (see also sections [4.11.9 SaveMainGeom \(page 264\)](#) and [4.11.10 SavePanelLayout \(page 265\)](#)).

The following table briefly lists some capabilities of the Root object.

Type	Parent of	Material	Converts to/Provides	Point Edit
Root	View ⁺	No	N/A	No

Table 32: Root Object Capabilities

The global scene rendering options are documented in the following sections.

RiOptions Property

This property carries RenderMan Interface options. Both, standard and BMRT specific options may be set using this property. For the sake of brevity only a short description of the available options will be given here. Please refer to the documentation of the RenderMan Interface and the documentation of BMRT for more detailed information about the options.

The `RiOptions` property consists of the following elements:

- "Width", "Height", if greater than zero this value will be used for the image size instead of the corresponding dimension of the view window, but only for real RIB export operations, not for the QuickRender and not for the Render actions in view windows. QuickRender and Render actions will always use the dimensions of the view window instead.

- "StdDisplay", if this is enabled, a standard display statement will be written to the RIB, which looks like this:

```
Display "unnamed.tif" "file" "rgba"
```

If this option is disabled, at least one RiDisplay tag should be added to the Root object (see also section 4.11.6 RiDisplay Tag (page 263)), otherwise the exported RIB will not contain a RiDisplay statement. This option has no effect on RIBs created by the QuickRender and Render actions in view windows.

- "Variance", maximum allowed variance of two pixel values. The default 0.0 causes no setting in the RIB. If the variance is > 0.0 no pixel samples setting will be written to the RIB. Various sources discourage the use of variance based sampling, because e.g. the number of samples actually taken (and therefore the rendering time) might not easily be predicted anymore.
- "Samples_X", "Samples_Y" number of samples taken per pixel.
- "FilterFunc", function used to filter final pixel values.
- "FilterWidth", "FilterHeight" size of the pixel filter.
- "ExpGain", Exposure
- "ExpGamma", Exposure Gamma
- "RGBA_ONE", "RGBA_MIN", "RGBA_MAX", "RGBA_Dither", specify quantisation and dithering
- "MinSamples", "MaxSamples", minimum and maximum number of samples per pixel (for variance based sampling).
- "MaxRayLevel", maximum number of recursive rays.
- "ShadowBias", minimum distance that one object has to be in order to shadow another object.
- "PRManSpec", toggles behaviour of BMRT's specular() function between PRMan compatible (default) and RI standard compatible.
- "RadSteps", number of radiosity steps, the default 0 leads to no radiosity calculations to be performed.
- "PatchSamples", minimum number of samples per patch to calculate the radiosity form factors for this patch.
- "Textures", "Shaders", "Archives" and "Procedurals" are search paths for the renderer.
- "TextureMem" and "GeomMem" determine how much memory rendrib (from BMRT) should use at maximum to cache textures and tessellated geometry.

Renderer specific options may also be set with RiOption tags (see section 4.11.2 RiOption Tag (page 260)), most comfortably added via the main menu "Special/Tags/Add RiOption" (see section 2.2 Tags Special Menu (page 39)).

Imager, Atmosphere Property

The Imager and Atmosphere properties let you define shaders for the Root object, please refer to section 4.10.4 Shader Properties (page 252) for information on how to deal with shader property GUIs.

Imager shaders are executed once for every rendered pixel, they may e.g. be used to set a specific background color or backdrop image.

Atmosphere shaders are volume shaders that may be used to implement global atmospheric optical effects like fog.

RIB Export

The Root object appears in RIB output in different places as collection of RenderMan Interface options and imager as well as atmosphere shaders.

The exact RIB statements used depend on the configuration of the object and the preference setting "RIB-Export/RISstandard".

The Root object is the only object to support RiOptions, RiHider, and RiDisplay tags (see also [section 4.11 Tags \(page 258\)](#)).

4.2.2 View Object

Every view window (see also section 2.4 [Anatomy of a View \(page 42\)](#)) has a corresponding view object as a child object of the Root object. Using the properties of the view object, camera settings, the type of the view, and other things related to the view can be changed. Note that deleting the object that represents a view, will not close the view window. You will just lose a way to configure it. Please, do not mess with the objects in other ways (e.g. copy them), you are asking for trouble otherwise!

Each view is associated with a virtual camera. The type of the view determines the default up-vector of that camera. If the type is "Top" the up-vector corresponds to the world Z-axis, else the world Y-axis. The type of the view, additionally, determines the so called *input plane* of the view. Interactive modelling actions in a view are limited to that input plane (unless the view is switched to local modelling; see also section 3.26 [Editing in Local Spaces \(page 105\)](#)).¹

The standard input planes are as following: Front – XY-plane, Side – ZY-plane, Top – XZ-plane, Trim – XY-plane.

In perspective views no interactive modelling actions are possible, but the camera may be modified, objects can be picked, and points selected.

Views of type "Trim" are used to edit trim curves of NPatch objects only. They display those trim curves as normal NURBS curves when the current level is inside a NPatch. The extensions of the patch in parameter-space are also drawn as a dashed rectangle. The trim curves should completely lie inside this rectangle. Note that picking of objects currently does not work in views of type "Trim".

The following table briefly lists some capabilities of the View object.

Type	Parent of	Material	Converts to/Provides	Point Edit
View	NPatch	No	N/A	Yes

Table 33: View Object Capabilities

The next sections detail the properties of the view object.

Camera Property

This section describes all elements of the "Camera" property:

- "From" is the point where the camera (that is attached to the view) is situated.
- "To" is the point the camera is looking to.
- "Up" is the up-vector of the camera.
- "Near" defines the near clipping plane. A value of 0.0 means a default value (that depends on the type of the view) should be used. Otherwise, near should always be positive for perspective views, and smaller than far.
- "Far" defines the far clipping plane. A value of 0.0 means a default value (that depends on the type of the view) should be used. Otherwise, far should always be bigger than near.
- "Roll" defines an angle by which the camera is rotated around the axis that is defined by the points from and to.
- "Zoom" is a zoom factor.

¹ Since 1.4.

Note that the up-vector is *not* checked for erroneous values (e.g. pointing in the direction of from-to) when applying the changes of the "Camera" property.

ViewAttrib Property

This section describes the elements of the "ViewAttrib" property:

- "Type" specifies the type of the view. Front, Side, Top (all parallel), Perspective and Trim (again parallel) may be selected.
- "Width" and "Height" control the size of the view window. It is currently not possible to resize internal views with these elements.
- "Redraw" toggles automatic redrawing of the view. If this is disabled, no drawing takes place in the view until an explicit redraw is requested (using the view menu, or the shortcut <Ctrl+d>).
- "DrawingMode" allows to specify the drawing mode of the view: "Draw" draws a wire-frame, "Shade" draws lighted surfaces. Note that the lighting is in no way an exact (or even similar) representation of the light information as specified with Light objects in the scene. Instead, a single light source, located at the camera origin (a headlight), will be used. "ShadeAndDraw" combines surfaces and wire-frames. "HiddenWire" shows hidden wire-frames and silhouettes. See also section 2.7 Drawing Modes (page 50).
- "DrawSel" toggles drawing of selected objects. If this is enabled, only the current selected objects will be drawn.
- "DrawLevel" toggles drawing of the objects of the current level only. If this is enabled, only the objects of the current level will be drawn.
- "Grid" is the grid size, 0.0 means no grid.
- "DrawGrid" toggles drawing of the current grid.
- "UseGrid" toggles, whether the current grid should be used by the interactive modelling actions. See also section 3 Modelling Actions (page 75).
- "ModellingMode" enables editing in local object spaces. See also section 3.26 Editing in Local Spaces (page 105).
- "DrawBG" controls whether the background image (specified by the "BGImage" option below) should be drawn. If a NPatch object is present as child of the View object, the image will be mapped onto this object instead of filling the complete view window background.
- "BGImage" is the name of a TIFF image file, that will be used as texture for the background image. Ayam will read this image file when the changes to the "ViewAttrib" property are applied, but also reread the image file if the notification callback of the view object is invoked (e.g. using the main menu entry "Tools/Force Notification").
- "Mark" is the marked point (in world coordinates) for the rotate and scale about modelling actions.
- "SetMark" controls whether the data from the "Mark" entries above should be used as new mark coordinates when the changes to the "ViewAttrib" property are applied.
- "EnableUndo" allows to control undo for the interactive view actions, e.g. panning or zooming a view. If this option is disabled, these actions will not be recorded in the undo system and also do not change the scene changed state.
This option is switched on by default.

Drag and Drop Support

View objects act in special ways, when certain objects are dropped onto them in the tree view:

When a Camera object is dropped onto a View object, the camera settings of the Camera object will be copied to the views camera.

When a Light object of type "Spot" is dropped onto a View object, the views camera will be changed, so that the user looks along the light to see what objects of the scene are lit by the light object (this works best with perspective views that have equal width and height).

It is also possible, to directly drag arbitrary objects from the tree view to a view window: for geometric objects, the view then performs a zoom to object operation, for cameras and light sources the views camera will be changed as if the object was dropped onto a View object in the tree view (see the above description).¹

4.2.3 Camera Object

Camera objects are used to temporarily save camera settings of views. Therefore, they have just two properties explained above, see sections [4.2.2 Camera \(page 116\)](#) and [4.10.2 Attributes Property \(page 252\)](#).

The following table briefly lists some capabilities of the Camera object.

Type	Parent of	Material	Converts to/Provides	Point Edit
Camera	No	No	N/A	Yes

Table 34: Camera Object Capabilities

Drag and Drop Support

When a View object is dropped onto a Camera object the camera settings from the view will be copied to the Camera object.

RIB Export

Camera objects never appear in RIB output.

¹ Since 1.8.

4.2.4 Light Object

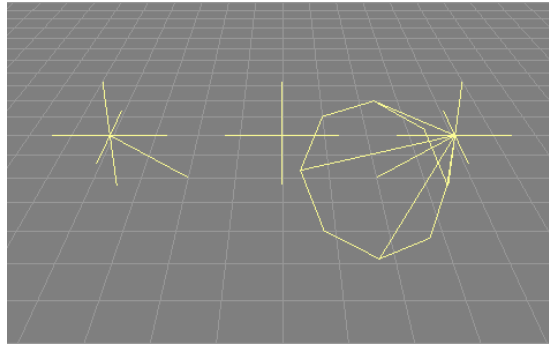


Figure 51: Lightsource Examples (l: Distant, m: Point, r: Spot)

Light objects represent light sources.

There are currently four different light types available in Ayam: "Point", "Distant", "Spot", and "Custom" (see also the image above).

Point-, Distant-, and Spotlights:

These standard light sources have well defined parameters that will be displayed in the "LightAttr" property. Please refer to the RenderMan documentation for more information about these standard light sources (see section 8.17 References (page 538)).

Custom Lights:

Light sources of type "Custom" use the attached light shader.

Note that Ayam is trying to guess from the names of the light shader arguments to draw the light. The names "from" and "to" denote location and destination of the light source. Those names should not be used for other things in the light shaders.

In contrast to the light sources as defined in the RenderMan interface, Ayam light sources are always global by default. This means, regardless of the place of a light source in the scene hierarchy, it will always light all other objects (unless the "IsLocal" attribute is used).

Note that the effect of a light source can not be previewed in shaded Ayam views, currently. However it is possible to estimate the effect of a spot light source by simply dropping it into a perspective view window; the view will then adapt the camera attributes (from, to, and zoom) of the view to show the objects lit by the spot.

The following table briefly lists some capabilities of the Light object.

Type	Parent of	Material	Converts to/Provides	Point Edit
Light	Yes	No	N/A	Yes

Table 35: Light Object Capabilities

LightAttr Property

The parameter "Type" determines the type of the light source.

Depending on the type of the light source, the light attribute property contains different sets of parameters.

Parameters that are not displayed will not be used on RIB export, consequently. When the type of a light source is changed, the property GUI will be adapted to show only the options available for the new light source type. Note that this adaptation happens when the "Apply"-button is used.

"IsOn" allows you to switch the light off or on. The default value is on.

"IsLocal" controls whether the light source should light just local objects (objects, that are defined in the same level in the scene hierarchy as the light source object or below it) or all objects in the scene. The default is off, all objects in the scene are lit. The "IsLocal" attribute is ignored for lights that are defined in the root level of the scene. Mind also that shadow maps will always contain shadows from all objects in the scene, regardless of the "IsLocal" attribute of the light source.

Using the light attribute "Shadows" you may determine whether the light source should cast shadows. The default is off, no shadows. Note that this option will not magically enable shadows on renderers that create shadows by shadow maps. It will merely be interpreted by raytracing renderers like BMRT.

The parameter "Samples" determines the number of times to sample an area light source, independent of pixel samples, the default value is 1. This attribute is available for custom lights only.

"UseSM" determines, whether shadow maps should be created and used for this light source. The resolution of the shadow map may be determined by the attribute "SMRes". If "SMRes" is 0, a default of 256 by 256 pixels will be used. These options are for renderers that do not support raytraced shadows like e.g. Aqsis only.

For lights of type "Distant" the "Scale" attributes of the "Transformations" property of the light object may be used to scale the camera transformation used for the creation of the corresponding shadow map. Values of 1 for "Scale_X" and "Scale_Y" create a shadow map that is sized 1 by 1 units in world space.

All other parameters that may appear in the "LightAttr" property are the standard parameters for the standard RenderMan light sources: distant, point, and spot:

- "From" and "To" denote position and target of the light source as point in space. You may edit both points using standard point editing actions (see also [section 3 Modelling Actions \(page 75\)](#)).
- "Color" is the color of the light emitted by the light source.
- "Intensity" is the intensity of the light emitted by the light source. Note that the standard point and spot lights have a quadratic falloff (with distance), that requires the intensity to be set to quite high values in order to achieve some illumination effect (e.g. around 30 for the standard distance of "From" and "To" of a spot light).
- "ConeAngle" is the angle of the beam of a spot light.
- "ConeDAngle" (cone delta angle) is the angle that determines a falloff area at the edge of the beam of a spot light.
- "BeamDistrib" (beam distribution) determines, how the light falls off in the beam of the spot light. Larger values result in narrower lit areas.

In order to ease the parameterisation of spot lights, the light source object may be dropped onto a view object or into a view window (preferably one with a perspective viewing transformation and with equal width and height) to see what objects of the scene are actually lit by the light object.

Using ShadowMaps

Using shadow maps requires the global preference setting "RIB-Export/ShadowMaps" to be switched to "Automatic" or "Manual" (both settings will be explained below). Furthermore, for each light source for which a shadow map should be created, the attributes "IsOn" and "UseSM" have to be switched on.

Automatic Creation of ShadowMaps

If the preference setting "RIB-Export/ShadowMaps" is set to "Automatic", Ayam will create a special version of the RIB on export, that creates all shadow maps automatically upon rendering. This is done by rendering depth images from the position of every light source that casts shadows. Special light source shaders later pick up these depth images and calculate shadows. This approach implies, that the scene is rendered multiple times. To reduce the size of the RIB, the objects to be rendered are written to a second RIB file named "<ribfilename>.obj.rib". This file is read from the main RIB several times via "ReadArchive". The RIB contains multiple frames which may also be rendered separately if the frame number is known. To help picking the right frame number for the image (e.g. to re-render just the final image, when only a material setting was changed, and no shadow casting lights were moved and no shadow casting geometry was changed), a comment with the frame number of the last frame (the image) will be written as last statement to the RIB.

Because multiple files (RIBs and shadow maps) are used, it is suggested to change the preference setting "RIB-Export/RIBFile" to "Scenefile". This will strip the leading absolute path component from the filenames so that the exported scene may be moved from one system to another more easily.

Do not render directly from a view window to the display when the "ShadowMaps" "RIB-Export" preference option is set to "Automatic". The renderer may not write image files when the command line option to render directly to the display (-d for rendrib, or -fb for Aqsis) is in use. Consequently, this may also inhibit writing of the shadow maps, so that the resulting image will look wrong, or the renderer will only render the shadow map to the display and then simply stop.

Manual Creation of ShadowMaps

If the preference setting "RIB-Export/ShadowMaps" is set to "Manual", the exported scene will not create the shadow maps upon rendering but rather expects them to be present already. They can be created manually using the view menu entries "View/Create ShadowMap", "View/Create All ShadowMaps" or the main menu entries "Special/RIB-Export/Create ShadowMap", "Special/RIB-Export/Create All ShadowMaps". Rendering the scene multiple times will be faster because the shadow maps will not be re-created each time. However, the user is responsible to update the shadow maps after certain scene changes, like for instance changes to the shadow casting geometry or light sources.

ShadowMap Types

Ayam supports three different methods for the creation of shadow maps for certain types of light sources: point, distant, and spot:

The point method is used with lights of type "Point" and custom lights that have a light shader argument named "from". Six shadow maps pointing in all possible axis aligned directions and named "<rib>.point<num>_<dir>.shd" (where "<rib>" is the name of the RIB, "<num>" is the number of the light source that makes use of shadow maps and "<dir>" is one of "x+", "x-", "y+", "y-", "z+", or "z-") will be created.

The `distant` method is used with lights of type `"Distant"` and custom lights that have a light shader argument named `"from"` and a light shader argument named `"to"`. One shadow map is created and named `"<rib>.dist<num>.shd"`. By default, the size of the shadow map is 1 by 1 units in world space, but this may be adapted using the scale transformation attributes of the light object.

The `spot` method is used with lights of type `"Spot"` and custom lights that have a light shader argument named `"from"`, a light shader argument named `"to"`, and a light shader argument named `"coneangle"`. One shadow map is created and named `"<rib>.spot<num>.shd"`. The `spot` method uses the cone angle (and additionally the delta cone angle, if present) argument to determine the size of the shadow map in world space.

If a light object of type `"Spot"`, `"Distant"` or `"Point"` is used, Ayam automatically changes the name of the exported light shader to `"shadowspot"`, `"shadowdistant"`, and `"shadowpoint"` respectively. Additionally, the shader will be parameterised to use the created shadow maps. If the light source is of type `"Custom"`, no automatic renaming and adjusting of the shader takes place. This means, you have to make sure that the shader really uses the shadow maps, by selecting the right shader and parameterising it accordingly. See the discussion above for the names of the shadow map files. Those file names, most probably, will have to be entered as parameter to the light shader.

For example, you will not get any shadows if you use a light source of type `"Custom"` with the normal `"distantlight"` shader attached, even though Ayam is able to create the necessary shadow maps. The normal `"distantlight"` shader just makes no use of the shadow maps. You have to manually switch to a shader that makes use of the shadow maps (`"shadowdistant"` in this case) to actually get shadows.

Here is a short example for a scene using a shadow map:

1. Go to the preferences (section "RIB-Export") and set "ShadowMaps" to "Automatic".
2. Create two boxes.
3. Open the "Transformations" property of the second box.
4. Translate it by X: 0.0, Y: -1.0, Z: 0.0.
5. Scale it by X: 4.0, Y: 1.0, Z: 4.0.
6. Create a light source.
7. Open the "LightAttr" property.
8. Change the type to "Spot". Press "Apply".
9. Now change the parameters of the spot light to "IsOn": Yes, "Intensity": 18.0, "UseSM": Yes, "ConeAngle": 45.0, "BeamDistrib": 3.0, "From": -2, 2, 2, "To": 1, 0, -1; leave all other parameters at their default values.
10. Create a new view and make it perspective (Menu: "Type/Perspective").
11. Export a RIB from that perspective view (Menu: "View/Export RIB").
12. Render the RIB with a RenderMan compliant renderer, that uses shadow maps, e.g. Photorealistic RenderMan (prman) or Aqsis.

This scene is distributed with Ayam as an example scene named "shadowmaps.ay", see also the following image from that scene that was created with an additional point light and Aqsis as renderer:

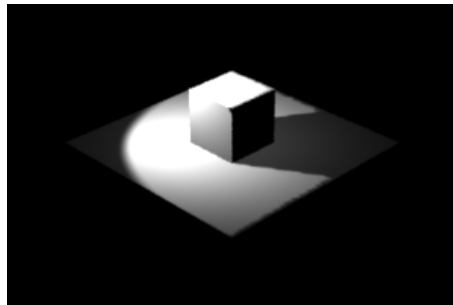


Figure 52: Shadowmaps Example

Note that for Aqsis you should add a `RiHider hidden,depthfilter,s,midpoint` tag to your Root object if shadow maps are in use. Other renderers might require additional tweaking using shadow bias `RiOption` tags. Please consult the documentation of your renderer on how to achieve the best results using shadow maps.

Using Area Lights

The common idealized standard light sources "Point", "Distant" and "Spot" have no own geometric extension in space. This means, shadows resulting from such light sources will have sharp borders which does not look too natural. Good looking soft shadows may be generated using area lights. See also the image below.

Area lights may be created by simply placing a single object as child object of a "Custom" light object that has the "arealight" shader attached, i.e. the scene hierarchy looks like this:

```
+--AreaLight (Light)
  \--AreaLightGeometry (Sphere)
```

This child object determines the geometry (place *and* extension) of the light source. According to L. Gritz, Spheres and Cylinders work best as area light geometry for BMRT, because of special sampling code.

An example:

- Create a custom light object.
- Assign the "arealight" light shader to it.
- Create a sphere.
- Drag and drop the sphere onto the light object so that it becomes a child of the light object.
- Transform the sphere object to your hearts content; the position and size of the object determines the position and size of the light source.

There is an example scene named "arealight.ay" distributed with Ayam, see also the following images from that scene that were created with BMRT 2.6 as renderer:

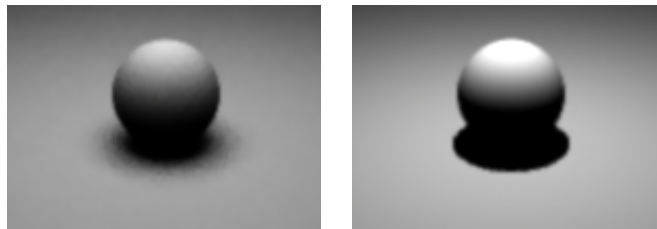


Table 36: Area Light (l) and Point Light (r)

4.2.5 Material Object

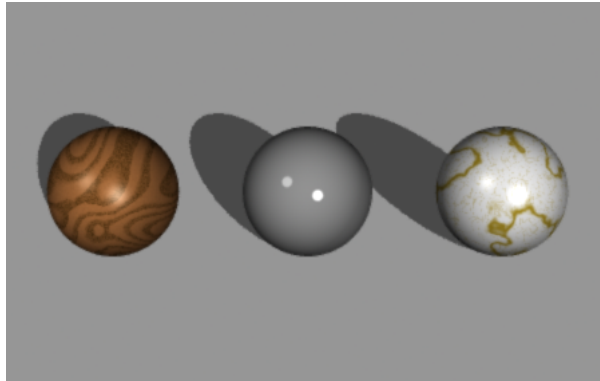


Figure 53: Material Examples (l: wood, m: ceramic, r: veined marble)

Material objects are used to attach RenderMan attributes and shaders to geometric objects. See also the image above that was rendered with BMRT 2.6 and uses materials with the following shaders, which are distributed with the renderer: *wood2*, *ceramic*, *veinedmarble*.

As long as geometric objects are connected to a material object, this material object may not be deleted.

The following table briefly lists some capabilities of the Material object.

Type	Parent of	Material	Converts to / Provides	Point Edit
Material	No	N/A	N/A	No

Table 37: Material Object Capabilities

RiAttributes Property

Using this property standard and BMRT specific RenderMan attributes may be set. Please refer to the documentation of the RenderMan interface and the documentation of BMRT for more detailed information about the RenderMan specific attributes.

- "Color", the color of the object. If one of the entries is set to a negative value (e.g. -1), the color will not be set at all for this object, i.e. no RiColor call will be emitted upon export.
- "Opacity", the opacity of the object, the default (255 255 255) means the object is totally opaque. If one of the entries is set to a negative value (e.g. -1), the opacity will not be set at all for this object, i.e. no RiOpacity call will be emitted upon export.
- "ShadingRate", determines how often shaders are evaluated for a sample.
- "Interpolation", determines how return values computed by the shaders are interpolated across a geometric primitive.
- "Sides", determines how many sides of the surface of a geometric primitive should be shaded.
- "BoundCoord", sets the coordinate system in which the displacement bound is expressed.
- "BoundVal", displacement bound value.
- "TrueDisp", toggles true displacements on or off. Default off.

- "CastShadows", determines how the object casts shadows: the default "Os" means the object casts shadows according to its opacity; "None" object does not cast any shadows; "Opaque" the object is completely opaque and casts shadows; "Shade" the object has a complex opacity pattern determined by its surface shader, that is used in shadow calculations.
- "Camera", "Reflection", and "Shadow" toggle visibility attributes.

Surface, Displacement, Interior, Exterior Property

These properties let you define shaders for the material object, please refer to [section 4.10.4 Shader Properties \(page 252\)](#) for information on how to deal with shader property GUIs.

Surface shaders may be used to procedurally encode lighting models and textures. Displacement shaders may procedurally deform the object while rendering. Interior and Exterior shaders are so called volume shaders that may be used to capture special optical effects, encoding how light is affected while passing through an object.

MaterialAttr Property

The MaterialAttr property contains attributes related to the management of material objects:

- "Materialname" denotes the name of the material. Note that material names have to be unique in a scene. If two materials with the same name exist, only the first material created with this name is "registered" and thus may be connected to geometric objects.
- "Refcount" shows how many geometric objects are connected to (are of) this material. Note that connected or referring geometric objects not necessarily have to live in the scene, they may as well temporarily reside in the object clipboard.
- "Registered" displays whether this material may be connected to geometric objects, see the discussion about material names above.

Drag and Drop Support

When geometric objects are dropped onto a material object they will be connected to this material object.

RIB Export

Material objects only appear in RIB output if connected to a geometric object (e.g. a Box).

The exact RIB statements used depend on the configuration of the material and the preference setting "RIB-Export/RISstandard".

If all elements of the MaterialAttr property are left on their default values, only color and opacity will be written to the RIB:

```
RiColor(...);
RiOpacity(...);
```

After the elements of the MaterialAttr property, the surface, displacement, interior, and exterior shaders (if attached to the material) will be exported, all shader parameters will be properly declared:

```
RiDeclare("Ka", "float");  
...  
RiSurface("Ka", 0.9, ...);  
...
```

After the material description custom `RiAttributes` and texture coordinates from tags will be exported.

No attempt is being made to re-order or sort objects in a level according to their attached materials, they will rather be exported in the order of their appearance in the level and thus each object with a material will also be prepended by a full material specification as described above.

Here is a complete example of a material description with the `wood2` surface shader applied to a sphere:

```
...  
Color 0.862745 0.862745 0.862745  
Opacity 1 1 1  
Declare "Ka" "float"  
Declare "Kd" "float"  
Declare "Ks" "float"  
Declare "roughness" "float"  
Declare "ringscale" "float"  
Declare "txtscale" "float"  
Declare "lightwood" "color"  
Declare "darkwood" "color"  
Declare "grainy" "float"  
Surface "wood2" "Ka" [1] "Kd" [0.75] "Ks" [0.4] "roughness" [0.1]  
"ringscale" [15] "txtscale" [1] "lightwood" [0.686275 0.439216 0.247059]  
"darkwood" [0.34902 0.219608 0.0784314] "grainy" [1]  
Sphere 1 -1 1 360  
...
```

4.2.6 Level Object

Level objects may be used to build object hierarchies and perform CSG operations.

Ayam does not offer a layer concept, but by grouping objects using levels and the hide/show tools, layer functionality may be emulated to a certain extent.

Organizing the scene and working in levels also increases the speed of object tree updates, as only the current level and its sub-levels are subject to a tree update if something in the object hierarchy changes.

Note that child objects of a level inherit the levels transformations, material, attributes, and shaders. Inheritance of e.g. transformations means:

If you have a NURBS patch in a level that is translated to (10,0,0), the origin of the local coordinate system of the NURBS patch will be situated at (10,0,0). If you decide to move the patch by a value of 5 in X direction by setting a corresponding value in the Transformations property of the patch object, the local coordinate system of the patch will be placed at (15,0,0) in world coordinates, i.e. a control point with the coordinate values (1,0,0) will be at (16,0,0) in terms of world coordinates.

Note also that since Ayam 1.12, Level objects provide their child objects to their parent objects as a list. This means the following hierarchy is now valid:

```
+-Skin
  +-Level
    |-NCurve
    |-NCurve
    |-ICurve
    \-NCurve
```

All NURBS curves and objects that may be converted to NURBS curves (in this example: the ICurve) will be provided to the Skin by the Level object. Transformation attributes of the Level will be added to the provided objects. Objects that do not provide the wanted type will be silently ignored.

The following table briefly lists some capabilities of the Level object.

Type	Parent of	Material	Converts to/Provides	Point Edit
Level	Any ⁺	Yes	N/A / Children ⁺	No

Table 38: Level Object Capabilities

LevelAttr Property

Levels do not have many object type specific properties, you may just modify the type of the level using the attribute "Type".

Levels of type "Level" just group objects and inherit attributes.

Levels of type "Union", "Intersection", and "Difference" are used to build CSG hierarchies. Additionally, they inherit attributes. Note that Ayam is currently not able to correctly display the results of CSG operations, all objects are always drawn completely, even though a CSG operation would cut parts away.

However, since Ayam 1.8 there is a plugin available that is able to preview the results of CSG operations, see also section 8.12 CSG preview using the AyCSG plugin (page 532).

The object hierarchy to cut away a part of a box using a sphere looks like this:

```
+-Level_of_Type_Difference (Level)
  |-Box
  \-Sphere
```

More than two objects may be arguments of a CSG operation:

```
+-Level_of_Type_Difference (Level)
  |-Box
  |-Sphere
  \-Sphere
```

In this example, the two spheres would cut away parts of the box.

New solid primitives may be created with levels of type "Primitive".

```
+-Level_of_Type_Difference (Level)
  +-Level_of_Type_Primitive (Level)
    | |-Sphere_blue
    | \-Disk_red
    \-Box_grey
```

In this example an open sphere with "ThetaMax" 180.0 (a hemisphere) is manually capped by a disk object. The two objects need to be placed into a level of type "Primitive" because each object alone is an open surface and therefore not a valid CSG primitive. Both objects that form the new primitive use a different material. In addition, a grey box cuts away a part from the multi colored hemisphere. The above CSG hierarchy is available as example scene file "multicolcsg.ay".

See also this image:

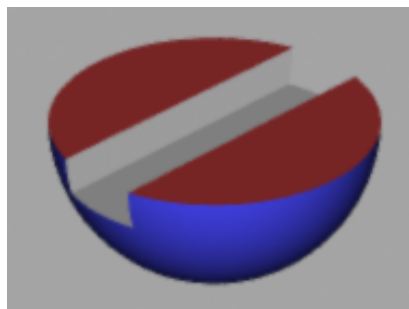


Figure 54: Multicolor CSG Example

Note that Ayam is not able to check, whether your new primitive obeys the rule of total closeness. For instance, if the disk in the above example would not totally cap the sphere (this happens if the disk "ThetaMax" is not 360.0 or if it is not placed exactly at the sphere) Ayam would not complain upon RIB export. The rendered image would expose serious errors, however.

Furthermore, it is not necessary to enclose normal child objects (e.g. quadrics with the "Closed" attribute set to on) of CSG levels in primitive levels for RIB export. This is done by Ayam automatically where needed.

RIB Export

The exact representation of a Level in RIB output depends on its type.

Normal Level objects appear in RIB output as Transformation hierarchies:

```
RiTransformBegin();  
RiTranslate(...);  
RiRotate(...);  
RiScale(...);  
  
«Children RIB output»  
  
RiTransformEnd();
```

Level objects of type Union, Difference, or Intersection will additionally emit a call to SolidBegin, and each child will be properly declared as primitive, e.g.:

```
RiTransformBegin();  
«Level Transformations»  
RiSolidBegin(RI_DIFFERENCE);  
  
RiSolidBegin(RI_PRIMITIVE);  
«Child #1 RIB output»  
RiSolidEnd();  
  
RiSolidBegin(RI_PRIMITIVE);  
«Child #2 RIB output»  
RiSolidEnd();  
  
RiSolidEnd();  
RiTransformEnd();
```

4.2.7 Instance Object

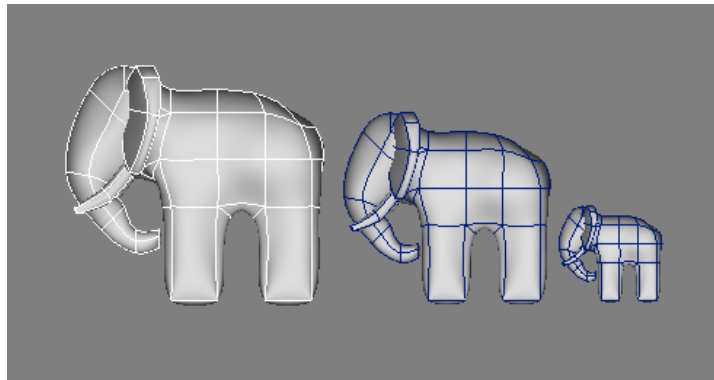


Figure 55: Instancing Example

Instance objects are transformed lightweight copies of single or groups of objects and can therefore help to save memory consumption in scenes with many similar or repeating objects. See also the image above, where there is only one set of patches stored for the three elephants. The amount of saved space can be very high, depending heavily on the actual scene and at what levels in the hierarchy instances are used. If there are no similar objects in the scene, however, instancing can hardly be utilised. Similar means "the same except for the transformation property" in this context.

The term instance is unfortunately misleading (and can be very confusing if you are accustomed to the terminology of object oriented programming), but it is the term that seems to be used and understood by most computer graphic artists for this mechanism. A better term would be *link*, as an instance object has the same basic properties as a link in a Unix file system. A link is just a pointer to an original file, the same goes for an instance object: it is just a pointer to an original object (here also called master object). A link can be placed anywhere on the file system, an instance object can be placed anywhere in the scene hierarchy, and additionally, it can be transformed.

While the original purpose of instances is to save memory, in the tool object context instances also serve as a means to transport geometric data across the scene hierarchy to make tool objects depend on each other (see also section 8.2 [The Modelling Concept Tool-Objects \(page 511\)](#)). Note that in the tool object context, instance objects are the only objects, that are subject to a second round of provision.

Some simple rules for instancing:

- No instances may be created of objects of the following types: Root, View, Instance, Material, Light. Do not try to fool Ayam and create instances of levels that contain aforementioned types of objects, things will go awry!
- It is allowed, however, to put some instances into a level object and create instances of this level (this is sometimes called *hierarchical instancing*).
- But instances of a level may not be put into the very same level or one of its children (this would be *recursive instancing*, which is not supported by Ayam).
- The original/master object may not be deleted from the scene as long as there are instances of that object in the scene or in the object clipboard.

If deleting of an object fails, and the error message complains about the reference counter not being zero, then the last rule was about to be violated. Clean the clipboard using the menu "Special/Clipboard/Paste (Move) " and delete or resolve all references first.

Ayam can also create instances for complete scenes automatically (see section 8.10 Automatic Instancing (page 531)).

To easily find the master object of an instance, just select the instance, then use the main menu entry: "Edit/Master".

The following table briefly lists some capabilities of the Instance object.

Type	Parent of	Material	Converts to/Provides	Point Edit
Instance	No	No	Master	No*

Table 39: Instance Object Capabilities

Instances without Transformations (References)

Instance objects support the "RP" tag type in a special way: if the "Transformations" property is removed using a "RP" tag, the instance does not provide objects with an own set of transformation attributes (to ease hierarchy building with e.g. "ExtrNC"/"ExtrNP" objects, where only pointers to already existing objects are required and where it is expected, that the instance reflects the master exactly, including its transformation attributes).¹

The extract curve/surface tools automatically add such a tag to the instances they create.

To create such a tag manually, select the Instance object and enter into the Ayam console:

```
»addTag RP Transformations
```

This special case of an instance is sometimes also called *reference*.

Instances and the Object Clipboard

This section contains additional information on instances and what happens to them when they are copied and pasted using the object clipboard.

All copies of instance objects, that are created by the object clipboard, always point to the same original master object. Therefore, it is not possible to copy a master object and some instances of it, so that the new instances point to the newly created master.

For example, when the following two objects are copied and pasted back to the scene

```
| -NCurve    <----- .
|
| -Instance  -----'
```

¹ Since 1.16.

the following scene hierarchy results:

```

|-NCurve    <-----+- .
|
|           | |
|-Instance  -----' |
|           | !
|-NCurve    |
|           |
|-Instance  -----'

```

The new instance still points to the original master and *not* to the copy of the master.

Nevertheless, it is possible to move complete sets of masters and their instances through the scene hierarchy using drag and drop in the tree view or using the object clipboard with "Edit/Cut" and then "Special/Clipboard/Paste (Move)".

Conversion Support

An Instance object may be converted to an ordinary object using the main menu entry "Tools/Convert". This process is also called resolving the instance.

To resolve all instance objects in a scene to normal objects, also the main menu entry: "Special/Instances/Resolve all Instances" can be used.

RIB Export

The RIB export of instances does *not* use the RiInstance facility of the RenderMan interface, but rather the more flexible ReadArchive mechanism.

This means, every master object in the scene will be written in a separate archive (RIB file) on disk, and every instance will cause that archive file to be read when rendering the RIB file. The resulting RIB file is not self contained: multiple files need to be transferred to other systems when rendering shall occur there.

This behaviour can be changed using the RIB export preference setting "ResInstances": If this option is enabled, all instances will be resolved temporarily to normal objects before being exported to RIB. The resulting RIB file will then be self contained but probably also much larger.

4.2.8 Clone Object

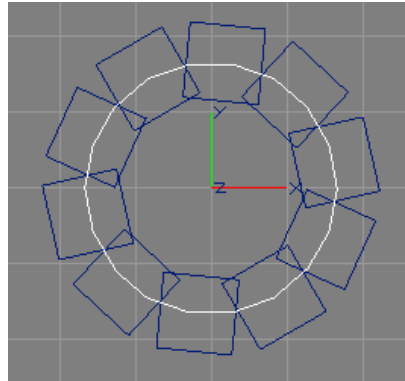


Figure 56: Clone Object (blue) with Trajectory Curve (white)

The Clone object allows to easily create and control an arbitrary number of instances of a single object, hereafter called the cloned object. The instances can be transformed (each by a certain amount expressed as difference between two instances) or placed on a trajectory curve (see also the image above).

If a second object is present as child of the Clone object it is treated as trajectory (or path) curve automatically. The process of placing the clones on the trajectory is very similar to the sweeping operation (see also section 4.7.4 Sweep Object (page 190)).

Thus, the default object hierarchy of a Clone object looks like this:

```
+-Clone
  |-Cloned-Object
  \-[Trajectory (NCurve) ]
```

If you use a trajectory curve to place the clones, you may want to hide the cloned object and also add a "NoExport" tag to it. Otherwise the original object will appear twice, on the trajectory and on its normal, untransformed, position. Note that the transformation attributes of the cloned object will be completely overwritten when placing the clones on the trajectory curve. If the cloned object has distinct scale or rotation attributes it should be put inside a level object like this:

```
+-Clone
  +-Level
  | \-Cloned-Object with non-standard Scale/Rotation
  \-Trajectory (NCurve)
```

It is not possible to create clones from objects that may not be master objects of instance objects, e.g. it is not possible to clone light objects or material objects. However, (since Ayam 1.7) it is possible to use instances as parameter objects.

If an instance object is used as cloned object on a trajectory it can be placed in a level and the "NoExport" tag can be added to the level object (as adding tags to Instance objects is more involved), see the following hierarchy for an example:

```

+-Clone
  +-Level with NoExport tag
  | \-Instance
  \-Trajectory (NCurve)

```

Since Ayam 1.20 the mirror facility of the Clone object is realized through the new Mirror object (see also section 4.2.9 Mirror Object (page 137)). The mirror facility was integrated into the Clone object before.

The following table briefly lists some capabilities of the Clone object.

Type	Parent of	Material	Converts to / Provides	Point Edit
Clone	Any ⁺	No	Children ⁺	No*

Table 40: Clone Object Capabilities

CloneAttr Property

The following attributes control the cloning process:

- "NumClones" is the number of clones to create.
- "Rotate" is only used, if a trajectory curve is present. If it is enabled all clones will be aligned according to the normal of the trajectory curve. Otherwise the rotation attributes will not be touched when placing the clone on the trajectory.
- "Translate_X", "Translate_Y", "Translate_Z", "Rotate_X", "Rotate_Y", "Rotate_Z", "Scale_X", "Scale_Y", "Scale_Z", those attributes control the transformation of the instances created by the Clone object. These attributes specify difference values between two instances: the clone "n+1" is offset by "Translate_X", "Translate_Y", and "Translate_Z" from the previous clone "n". It is also rotated by "Rotate_X", "Rotate_Y", and "Rotate_Z" and scaled by "Scale_X", "Scale_Y", "Scale_Z" in relation to the previous clone. Note however, that the transformation attributes do not affect the first clone.

The transformation attributes are also in effect if a trajectory curve is present, they will be applied after moving of the instance to the trajectory and rotating it.¹

The following table summarizes which transformation attributes are used in the respective clone modes.

Mode	Use Child Transform	Use CloneAttrib Transform	Use Clone Transform
Clone	No	Yes	Yes
Trajectory	Yes	Yes	Yes
Mirror	Yes	N/A	Yes

Table 41: Clone Parameterisation Examples

¹ Since 1.13.

Conversion Support

The Clone object may be converted to ordinary objects using the main menu entry "Tools/Convert".

Upon conversion a Level object will be created, that contains the original object *and* the clones.

RIB Export

Clone objects appear in RIB output as a number of real objects, each with different transformation attributes.

As the original objects will also appear in the RIB output, it is suggested to add a "NoExport" tag to the original if the Clone is in trajectory mode.

4.2.9 Mirror Object

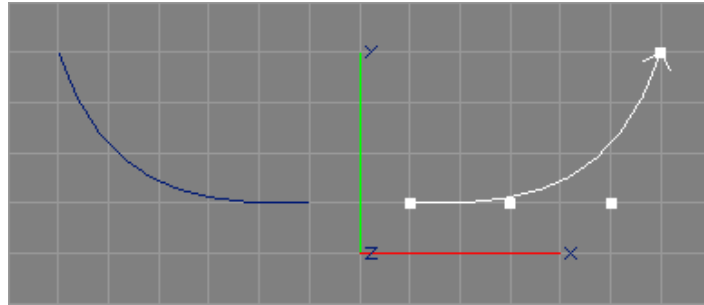


Figure 57: Mirrored Object (blue) From Curve (white)

The Mirror object allows to easily create and control an arbitrary number of mirrored instances of a number of objects.¹

The original object(s) *and* their mirrored counterparts will be provided by the Mirror object to the respective parent object (normally, tool objects do not provide their unmodified children). Additionally, the order of the mirrored objects will be reversed so that it is possible to use a single Mirror object (with one or multiple NURBS curves as children) as parameter object of e.g. a Skin object:

```

+-Skin
  +-Mirror
    \-NCurve
+-Skin
  +-Mirror
    |-Curve_1 (NCurve)
    |-Curve_2 (NCurve)
    \-Curve_3 (NCurve)

```

The first Skin object in the above example can only work, because also the unmodified NCurve is provided by the Mirror object. The second Skin object can only work properly, because the order of the provided objects is reversed, so that it gets to see the curves in the following sequence: "NCurve_1", "NCurve_2", "NCurve_3", "Mirrored_NCurve_3", "Mirrored_NCurve_2", "Mirrored_NCurve_1".

The following table briefly lists some capabilities of the Mirror object.

Type	Parent of	Material	Converts to/Provides	Point Edit
Mirror	Any ⁺	No	Children ⁺	No*

Table 42: Mirror Object Capabilities

MirrorAttr Property

The following attributes control the mirror process:

- "Plane" allows to select the plane about which the mirroring should occur (YZ-, XZ-, or XY-plane).

¹ Since 1.20.

Conversion Support

The Mirror object may be converted to ordinary objects using the main menu entry "Tools/Convert".

Upon conversion a Level object will be created, that contains the original objects and the mirrored counterparts (the latter in reverse order).

RIB Export

Mirror objects appear in RIB output as a number of real objects, each with different transformation attributes.

4.2.10 Select Object

The Select object may be used in hierarchies of tool objects to select one object from a list of provided objects.¹

Also multiple objects and ranges (even decreasing ranges that lead to reversed orders) may be selected.²

In the following example hierarchy, a single patch from multiple provided patches of the Sweep object (the swept surface, a bevel, or a cap) could be selected by the Select object and delivered upstream to the ExtrNC object.

```

+-Sweep          <----- .
+-Revolve         |
+-ExtrNC          |
+-Select         |
  \-Instance_of_Sweep(Instance) --'

```

Note that this example just serves illustrative purpose; the hierarchy presented is not exactly useful, as the ExtrNC object has a selector facility built in. Consequently, the Select object should be used in scenarios, where a selector facility does not exist or is hard to implement, as e.g. in Script object scripts.

The following table briefly lists some capabilities of the Select object.

Type	Parent of	Material	Converts to/Provides	Point Edit
Select	Any ⁺	No	N/A / Children ⁺	No

Table 43: Select Object Capabilities

SelectAttrib Property

The following attribute controls the selection process:

- "Indices" designates the object(s) to select. The index values are zero based. Multiple indices must be separated by ", ", ranges can be specified like this "1-4", reversed ranges are allowed ("4-1") and will create an object list of reversed order. The special index "end" (or abbreviated "e") designates the last of all the provided objects. An index may appear multiple times, leading to multiple copies of the selected object to be delivered upstream. The index space spans over all provided objects of the desired type from all child objects. This means provided objects from multiple different child objects of the Select object can be mixed. Syntactically incorrect ranges and indices are silently ignored. Examples:
 - "0, 2" – delivers the first and third provided objects upstream;
 - "end-0" – delivers all provided objects in reversed order upstream;
 - "0, 0, 0" – delivers three copies of the first provided object upstream;
 - "0, 4-end, 1" – delivers the first, the fifth (if there are so many) up to the last, and the second object upstream.

RIB Export

Select objects never appear in RIB output.

¹ Since 1.14. ² Since 1.16.

4.2.11 RiInc Object

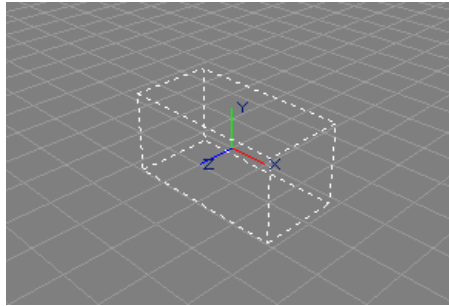


Figure 58: RiInc Example

RiInc objects may be used to include objects or whole scene parts into your scenes that are just available as a piece of RIB, a so called *archive*. They can therefore help to manage huge scenes that are too cumbersome to handle in total in Ayam.

RiInc objects draw a simple bounding box with stippled lines instead of the archive contents; those will only become part of the scene when it is exported to a RIB. See also the above image. If a RiInc object has children, they will be drawn instead of the bounding box. This allows to better convey the actual shape of the objects in the included archive.¹ As the bounding box, also the children never appear in the RIB.

Note that Ayam can not check, whether the bounding box or children actually match the content of the included archive.

Also note that archives can be created easily using the main menu entry "Special/RIB-Export/Selected Objects".

The following table briefly lists some capabilities of the RiInc object.

Type	Parent of	Material	Converts to/Provides	Point Edit
RiInc	Any	No	N/A	No

Table 44: RiInc Object Capabilities

RiIncAttr Property

The following attributes control the inclusion process:

- Using "File" the filename of the RIB archive to be included is specified.
- "Width", "Height", and "Length" determine the size of a box, that will be drawn as a geometric representation of the RIB archive (if no children are present).

RIB Export

RiInc objects export as single

```
ReadArchive <filename>
```

statement.

¹ Since 1.26.

4.2.12 RiProc Object

RiProc objects may be used to include procedural objects or external archives into your scenes. In contrast to archives included via a RiInc object, these archives will only be included by the RenderMan renderer when the specified bounding box is visible to the current camera.

The following table briefly lists some capabilities of the RiProc object.

Type	Parent of	Material	Converts to/Provides	Point Edit
RiProc	No	No	N/A	No

Table 45: RiProc Object Capabilities

RiProcAttr Property

The following attributes control the RiProc object:

- "Type" defines the type of the procedural object which is one of "DelayedReadArchive", "RunProgram", or "DynamicLoad".
- Using "File" you specify the filename of the RIB archive, program, or dynamic shared object (depending on the type of the procedural object).
- Using "Data" additional arguments may be supplied to procedural objects of type "RunProgram" and "DynamicLoad".
- "MinX", "MaxX", "MinY", "MaxY", "MinZ", and "MaxZ" specify the size of the bounding box of the objects that the procedural will create or the archive contains.

RIB Export

RiProc objects export as single

```
RiProcedural
```

statement.

4.3 CSG/Solid Primitives

These objects serve as geometric primitives in CSG hierarchies.

4.3.1 Box Object

A solid box, centered at the origin of the object coordinate system.

The following table briefly lists some capabilities of the Box object.

Type	Parent of	Material	Converts to/Provides	Point Edit
Box	No	Yes	NPatch*	No*

Table 46: Box Object Capabilities

BoxAttr Property

The following parameters control the shape of the box:

- "Width" is the width of the box (size of the box in direction of the X-axis of the objects coordinate system).
- "Length" is the length of the box (size of the box in direction of the Z-axis of the objects coordinate system).
- "Height" is the height of the box (size of the box in direction of the Y-axis of the objects coordinate system).

Conversion Support

A box object may be converted to three NURBS patches using the main menu entry "Tools/Convert".¹

See section 6.6.3 Cylindrical Box (page 462) for a Script object that converts to a single NURBS patch.

RIB Export

The box object will always be exported as six bilinear patches.

¹ Since 1.8.2.

4.3.2 Sphere Object

A sphere, centered at the origin of the object coordinate system.

The following table briefly lists some capabilities of the Sphere object.

Type	Parent of	Material	Converts to/Provides	Point Edit
Sphere	No	Yes	NPatch ⁺	No*

Table 47: Sphere Object Capabilities

SphereAttr Property

The following parameters control the shape of the sphere:

- "Closed" toggles whether the object should be automatically sealed (closed by matching cap surfaces).
Only when this option is enabled, the sphere may be used in CSG operations safely.
- "Radius" is the radius of the sphere, default is 1.
- "ZMin" is the lower limit of the sphere on the Z-axis, default is -1.
- "ZMax" is the upper limit of the sphere on the Z-axis, default is 1.
- "ThetaMax" is the sweeping angle of the sphere in degrees, default is 360.

Conversion Support

A Sphere object may be converted to NURBS patches using the main menu entry "Tools/Convert". This conversion obeys all parameters of the sphere.¹

If the sphere is closed, an enclosing Level object will be created and the caps follow the sphere in the following order: disk-shaped cap at zmin, disk-shaped cap at zmax, cap at theta 0, cap at thetamax.

RIB Export

The Sphere object appears in RIB output as simple

```
RiSphere(...);
```

or, if "Closed" is enabled and "ZMin", "ZMax", or "ThetaMax" have other than the default values, as complex CSG hierarchy of at most two spheres, two cylinders, and eight disks.

¹ Since 1.8.2.

4.3.3 Disk Object

A disk in the XY-plane, centered at the origin of the object coordinate system.

The following table briefly lists some capabilities of the Disk object.

Type	Parent of	Material	Converts to/Provides	Point Edit
Disk	No	Yes	NPatch	No*

Table 48: Disk Object Capabilities

DiskAttr Property

The following parameters control the shape of the disk:

- "Radius" is the radius of the disk, default is 1.
- "ZMin" displaces the disk along the Z-axis, default is 0.
- "ThetaMax" is the sweeping angle of the disk in degrees, default is 360.

Conversion Support

A Disk object may be converted to a NURBS patch using the main menu entry "Tools/Convert". This conversion obeys all parameters of the disk.¹

RIB Export

The Disk object will always be exported as simple disk:

```
RiDisk(...);
```

¹ Since 1.8.2.

4.3.4 Cone Object

A cone, centered at the origin of the object coordinate system, with the base in the XY-plane.

The standard cone is always pointed. See section 6.6.1 [Truncated Cone \(page 460\)](#) for a Script object that implements a truncated cone.

The following table briefly lists some capabilities of the Cone object.

Type	Parent of	Material	Converts to/Provides	Point Edit
Cone	No	Yes	NPatch ⁺	No*

Table 49: Cone Object Capabilities

ConeAttr Property

The following parameters control the shape of the cone:

- "Closed" toggles whether the object should be automatically sealed (closed by matching cap surfaces).
Only when this option is enabled, the cone may be used in CSG operations safely.
- "Radius" is the radius of the cone at the base, default is 1.
- "Height" is the height of the cone, default is 1.
- "ThetaMax" is the sweeping angle of the cone in degrees, default is 360.

Conversion Support

A Cone object may be converted to NURBS patches using the main menu entry "Tools/Convert". This conversion obeys all parameters of the cone.¹

If the cone is closed, an enclosing Level object will be created and the caps follow the cone in the following order: disk-shaped cap at the base, cap at theta 0, cap at thetamax.

RIB Export

The Cone object appears in RIB output as simple

```
RiCone(...);
```

or, if "Closed" is enabled and "ThetaMax" has a different than the default value, as complex CSG hierarchy of at most one cone, one disk, and two polygons.

¹ Since 1.8.2.

4.3.5 Cylinder Object

A cylinder, centered at the origin of the object coordinate system.

The following table briefly lists some capabilities of the Cylinder object.

Type	Parent of	Material	Converts to/Provides	Point Edit
Cylinder	No	Yes	NPatch ⁺	No*

Table 50: Cylinder Object Capabilities

CylinderAttr Property

The following parameters control the shape of the cylinder:

- "Closed" toggles whether the object should be automatically sealed (closed by matching cap surfaces).
Only when this option is enabled, the cylinder may be used in CSG operations safely.
- "Radius" is the radius of the cylinder, default is 1.
- "ZMin" determines the Z location of the base, default is -1.
- "ZMax" determines the Z location of the top, default is 1.
- "ThetaMax" is the sweeping angle of the cylinder in degrees, default is 360.

Conversion Support

A cylinder object may be converted to NURBS patches using the main menu entry "Tools/Convert". This conversion obeys all parameters of the cylinder.¹

If the cylinder is closed, an enclosing Level object will be created and the caps follow the cylinder in the following order: disk-shaped cap at zmin, disk-shaped cap at zmax, cap at theta 0, cap at thetamax.

RIB Export

The cylinder object appears in RIB output as simple

```
RiCylinder(...);
```

or, if "Closed" is enabled and "ThetaMax" has a different than the default value, as complex CSG hierarchy of at most one cylinder, two disks, and two polygons.

¹ Since 1.8.2.

4.3.6 Torus Object

A torus, centered at the origin of the object coordinate system. A torus is a donut like shape, that results from sweeping a small circle (that has been displaced along X sufficiently) around the Z-axis.

The following table briefly lists some capabilities of the Torus object.

Type	Parent of	Material	Converts to / Provides	Point Edit
Torus	No	Yes	NPatch ⁺	No*

Table 51: Torus Object Capabilities

TorusAttr Property

The following parameters control the shape of the torus:

- "Closed" toggles whether the object should be automatically sealed (closed by matching cap surfaces).
Only when this option is enabled, the torus may be used in CSG operations safely.
- "MajorRad" is the radius of the torus, measured from the Z-axis to the center of the swept smaller circle, default is 0.75.
- "MinorRad" is the radius of the swept circle, default is 0.25.
- "PhiMin" determines an angle to limit the swept circle, default is 0.
- "PhiMax" determines an angle to limit the swept circle, default is 360.
- "ThetaMax" is the sweeping angle of the torus in degrees, default is 360.

Conversion Support

A torus object may be converted to NURBS patches using the main menu entry "Tools/Convert". This conversion obeys all parameters of the torus.¹

If the torus is closed, an enclosing Level object will be created and the caps follow the torus in the following order: disk-shaped cap at theta 0, disk-shaped cap at thetamax, ring-shaped cap at phimin 0, ring-shaped cap at phimax.

RIB Export

The torus object appears in RIB output as simple

```
RiTorus(...);
```

or, if "Closed" is enabled and "PhiMin", "PhiMax", or "ThetaMax" have different than the default values, as complex CSG hierarchy of at most one one torus, two disks, and two hyperboloids.

¹ Since 1.8.2.

4.3.7 Paraboloid Object

A paraboloid, centered at the origin of the object coordinate system.

The following table briefly lists some capabilities of the Paraboloid object.

Type	Parent of	Material	Converts to/Provides	Point Edit
Paraboloid	No	Yes	NPatch ⁺	No*

Table 52: Paraboloid Object Capabilities

ParaboloidAttr Property

The following parameters control the shape of the paraboloid:

- "Closed" toggles whether the object should be automatically sealed (closed by matching cap surfaces).
Only when this option is enabled, the paraboloid may be used in CSG operations safely.
- "RMax" is the radius of the paraboloid at a Z of "ZMax", the base of the paraboloid surface, default is 1.
- "ZMin" limits the paraboloid surface on the Z-axis, must be positive, default is 0.
- "ZMax" limits the paraboloid surface on the Z-axis and determines the Z location of the base, must be positive, default is 1.
- "ThetaMax" is the sweeping angle of the paraboloid in degrees, default is 360.

Conversion Support

A paraboloid object may be converted to NURBS patches using the main menu entry "Tools/Convert". This conversion obeys all parameters of the paraboloid.¹

If the paraboloid is closed, an enclosing Level object will be created and the caps follow the paraboloid in the following order: disk-shaped cap at zmin, disk-shaped cap at zmax, cap at theta 0, cap at thetamax.

RIB Export

The paraboloid object appears in RIB output as simple

```
RiParaboloid(...);
```

or, if "Closed" is enabled and "ZMin", "ZMax", or "ThetaMax" have different than the default values, as complex CSG hierarchy of at most one paraboloid, two disks, and two bicubic patches.

¹ Since 1.8.2.

4.3.8 Hyperboloid Object

A hyperboloid, centered at the origin of the object coordinate system. The shape of the hyperboloid will be created by sweeping a line specified by two points in space around the Z-axis. Thus, disk, cylinder, and cone are special cases of the hyperboloid.

The following table briefly lists some capabilities of the Hyperboloid object.

Type	Parent of	Material	Converts to/Provides	Point Edit
Hyperboloid	No	Yes	NPatch ⁺	No*

Table 53: Hyperboloid Object Capabilities

HyperboloidAttr Property

The following parameters control the shape of the hyperboloid:

- "Closed" toggles whether the object should be automatically sealed (closed by matching cap surfaces).
Only when this option is enabled, the hyperboloid may be used in CSG operations safely.
- "P1_X", "P1_Y" and "P1_Z", define point one, default is (0, 1, -0.5).
- "P2_X", "P2_Y" and "P2_Z", define point two, default is (1, 0, 0.5).
- "ThetaMax" is the sweeping angle of the hyperboloid in degrees, default is 360.

Conversion Support

A hyperboloid object may be converted to NURBS patches using the main menu entry "Tools/Convert". This conversion obeys all parameters of the hyperboloid.¹

If the hyperboloid is closed, an enclosing Level object will be created and the caps follow the hyperboloid in the following order: disk-shaped cap at P1, disk-shaped cap at P2, non-planar cap at theta 0, non-planar cap at thetamax.

RIB Export

The hyperboloid object appears in RIB output as simple

```
RiHyperboloid(...);
```

or, if "Closed" is enabled and "ThetaMax" has a different than the default value, as complex CSG hierarchy of at most one hyperboloid, two disks, and two bilinear patches.

Note that due to a bug in BMRT that is still present in V2.3.6 the "Closed" option does not work properly when "ThetaMax" has a different than the default value and the hyperboloid has no displacement shader. In fact, using a displacement shader with a km (amount of displacement) of 0.0 is a work-around for this bug (found by T. E. Burge).

¹ Since 1.8.2.

4.4 Freeform Curve Objects

These objects are mainly used as child objects for the surface generating tool objects.

4.4.1 NCurve (NURBS Curve) Object

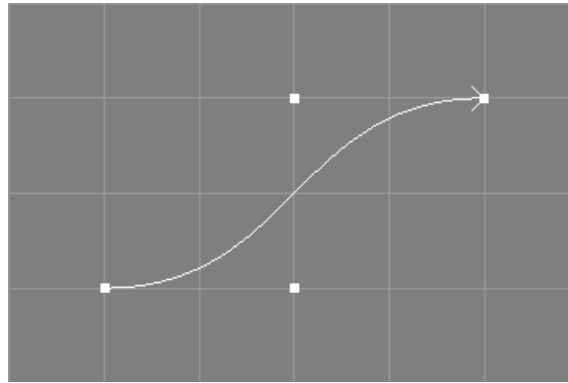


Figure 59: A simple NURBS Curve

The NCurve object is the most used basic object for NURBS modelling in Ayam because NURBS curves are used to build more complex smoothly shaped surfaces using operations like extrude, revolve, sweep or skin. NURBS curves can be open or closed and used to emulate Bezier and B-Spline curves easily. In addition, for easier modelling, they support multiple points, as explained in section 4.4.1 Multiple Points (page 153).

The following table briefly lists some capabilities of the NCurve object.

Type	Parent of	Material	Converts to/Provides	Point Edit
NCurve	No	No	N/A	Yes

Table 54: NCurve Object Capabilities

NCurve objects may be created via the toolbox, the main menu, several tools as documented in section 5.2 Curve Creation Tools (page 274), or the scripting interface, see also section 6.2.2 Creating Objects (page 351).

Tools that process NCurve objects are documented in 5.3 Curve Modification Tools (page 279).

The next section details the NCurve object property.

NCurveAttr Property

The first section of the NCurveAttr property contains curve specific settings:

- "Type": This attribute replaces the "Closed" attribute since Ayam 1.9.

The type "Open" is for the standard open NURBS curve.

If the type is "Closed", the first and last control point of the curve will be made identical. This will close the curve but without any guaranteed continuity. Such a closed curve will e.g. be created by the NURBS circle tool. It is important to know, that identical start/end control points alone can not guarantee that the curve is closed if the knot vector is not clamped. If in doubt, use the clamp tool or a knot vector of type "NURB", "Chordal", or "Centripetal".

If the type is "Periodic", the last p control points of the curve will be made identical to the first p where p is the degree of the curve (read order-1). This will close the curve with guaranteed continuity. Note that for a cubic spline (order 4) you will need at least 6 control points to make it periodic. It is important to know, that the multiple control points alone can not guarantee that the curve is closed if the knot vector has no periodic extensions. If in doubt, switch the curve to knot type "B-Spline", "Chordal", or "Centripetal".

You may also want to enable the creation of multiple points using the "CreateMP" attribute (see below) for closed and periodic curves so that single point editing actions modify all multiple control points.

- "Length" is the number of control points of the curve.
- "Order" is the order of the curve.
- "Knot-Type": Using "Knot-Type" you may select from "NURB", "Bezier", "B-Spline", "Custom", "Chordal", and "Centripetal" knot types.

The knot type "NURB" will generate uniformly distributed knot values ranging from 0.0 to 1.0, where the multiplicity of the knots at the ends will be of order of the curve (the knot vector will be clamped). This guarantees that the curve will touch the control points at the ends of the curve. An example "NURB" knot vector for a curve of length 5 and order 4 would be:

```
{ 0.0 0.0 0.0 0.0 0.5 1.0 1.0 1.0 1.0 }.
```

The knot type "Bezier" will generate just 0.0 and 1.0 values. Note that the order of the curve has to be equal to the length of the curve if "Bezier" knots are generated. Otherwise, the generated knot sequence is illegal. The resulting curve looks and behaves exactly like a real Bezier curve, interpolating the control points at the ends and so on. An example "Bezier" knot vector for a curve of length 5 and order 5 would be:

```
{ 0.0 0.0 0.0 0.0 0.0 1.0 1.0 1.0 1.0 1.0 }.
```

The knot type "B-Spline" will generate uniformly distributed knot values (without any multiple knots). The resulting curve looks and behaves like a B-Spline curve. It is *not* interpolating the control end points. An example "B-Spline" knot vector for a curve of length 5 and order 4 would be:

```
{ 0.0 0.125 0.25 0.375 0.5 0.625 0.75 0.875 1.0 }.
```

The knot types "Chordal" and "Centripetal" will generate knot values whose distribution reflect the distances of the control points. This only works, if there are free knots in the knot vector, i.e. knots that are not subject to clamping or periodic extensions (the default NURBS curve with 4 control points and order 4 has *none*). For open curves, the generated knot vector will be clamped, for periodic curves, proper periodic extensions will be created. Those knot types are mainly useful for curves with unevenly distributed control points that will be sampled uniformly (in parametric space) later on and where it is expected, that the uniform sampling in parameter space results in evenly distributed sample points in coordinate space, e.g. if the curves are used as Sweep, Birail, or Clone trajectory, or surfaces are created from them that use implicit texture coordinates or a uniform tessellation strategy. The "Chordal" and "Centripetal" knots will ensure a more uniform distribution of the sample points on the curve in such cases (see also the example image below). An example "Chordal" knot vector for an open curve of length 5 and order 4 would be:

```
{ 0.0 0.0 0.0 0.0 0.388889 1.0 1.0 1.0 1.0 }.
```

The image below illustrates the use of two curves with uniform (NURB) vs. chordal knot vectors as Sweep trajectories. The upper Sweep with the uniform knot vector has much more unevenly distributed/sized sections and exhibits more severe self intersection problems. Please note that the shapes of the curves differ slightly.

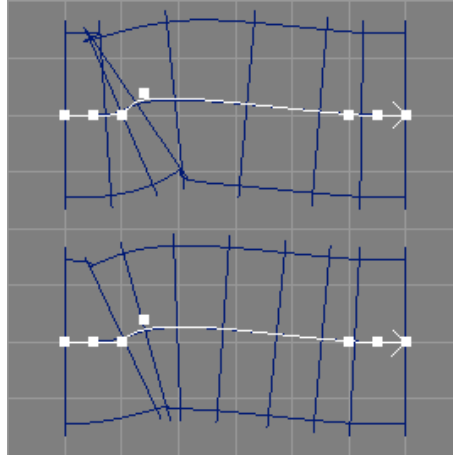


Figure 60: Sweeps Created From Curves With Uniform (upper) And Chordal (lower) Knot Vectors

- "Knots" allows to enter own custom knot sequences. Note that this only works if the "Knot-Type" (above) is "Custom". Otherwise, the "Knots" parameter just displays the automatically generated knot sequence and changes to the values have no effect.

Each knot vector has $l + o$ knot values, where l is the length of the curve and o its order. The knot values must be strictly monotonous (increasing) and the maximum number of equal values in a row must not exceed the order of the curve.

- "CreateMP" toggles, whether multiple points should be created for this curve. See also the discussion in section 4.4.1 Multiple Points (page 153).
- "IsRat" informs, whether the curve is rational (has any weight values different from 1.0).¹

The GLU-parameters control the appearance of the curve when curve/surface display is enabled.

- "Tolerance" is in fact GLU sampling tolerance, used to control the quality of the sampling when rendering a curve. Smaller tolerance settings lead to higher quality but also slower display. A setting of 0.0 means, that the global preference setting "Drawing/Tolerance" should be used.
- "DisplayMode" determines how the curve should be drawn. The control hull (control polygon) or the curve or a combination of both may be displayed. The setting "Global" means, that the global preference setting "Drawing/NCDisplayMode" should be used.

When changing more than one of the above values the changes will be applied in the order of the values in the property. The sum of the changed values should describe a valid NURBS curve. It is perfectly legal to change the length of the curve, its order, and switch to a custom knot vector (be sure to actually enter a valid new knot vector) at once. Ayam will check your changes and fall back to certain default values if e.g. your knot sequence is wrong. Check the console for any messages after pressing the "Apply" button!

When the curve type is changed using the NCurveAttr property Ayam may also have to change the position of some control points as follows:

- When the type is changed from "Open" to "Closed", the last control point is moved to be identical to the first one. In addition, if the current knot type of the curve is "B-Spline", it will be reset to knot type "NURB".
- When the type is changed from "Open" or "Closed" to "Periodic", the last p control points will be moved to be identical to the first p , where p is the degree of the curve ($order - 1$). For a cubic

¹ Since 1.9.

curve (order 4), consequently, the last three control points will be moved. In addition, if the current knot type of the curve is "NURB" or "Bezier" it will be changed to "B-Spline" automatically.

When changing the order of a periodic curve (and not touching the length) Ayam will automatically add or remove control points to/from the curve so that the shape of the curve remains largely intact and the periodic extensions plausible.¹

Multiple Points

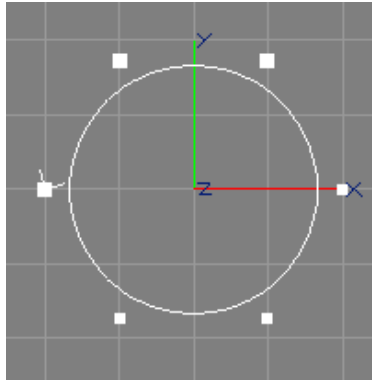


Figure 61: A NURBS Curve with Multiple Points (big handles)

The NURBS curves of Ayam support so called multiple points. A multiple point is made of a number of different control points that have the same coordinates. Multiple points will be drawn with a bigger handle than normal points (see image above).

Single point interactive modelling actions (e.g. edit) will always modify all the control points that make up a multiple point.

Selecting/tagging a multiple point will always select all control points of that multiple point.

Modelling actions that work with selected points (e.g. translate) will only modify all control points of a multiple point, if it is completely selected. This selection state is conveyed by drawing a completely filled bigger handle.² Individual points of a multiple point may be selected by

- turning off the "CreateMP" property of the curve,
- using the "selPnts" scripting interface command (see also section 6.2.5 selPnts command (page 377)), or
- using the "CVView" property (see also section 6.5.24 CVView property (page 454)).

Note that the control points that make up a multiple point do not have to be consecutive (in the control point vector of the NURBS curve).

Multiple points may also be created using the collapse tool, and split up again using the explode tool (see sections 5.3.29 Collapse Tool (page 307) and 5.3.30 Explode Tool (page 307) for more information regarding those tools).

¹ Since 1.18. ² Since 1.28.

RIB Export

NCurve objects never directly appear in RIB output (only indirectly as trim curve).

Scripting Support

To create NCurve objects using the scripting interface see section [6.2.2 crtOb NCurve command](#) (page 351). For scripting interface commands that manipulate NURBS curve objects see section [6.2.15 Manipulating NURBS Curves](#) (page 393).

4.4.2 ICurve (Interpolating Curve) Object

The ICurve object creates a global interpolating NURBS curve from n 3D non-rational ordered data points. The curve may be open or closed, the order of the curve may be configured, the parameterisation may be adapted, and end derivatives may be specified. The open versions create $n + 2$ NURBS control points, and the closed ones $n + 3$.

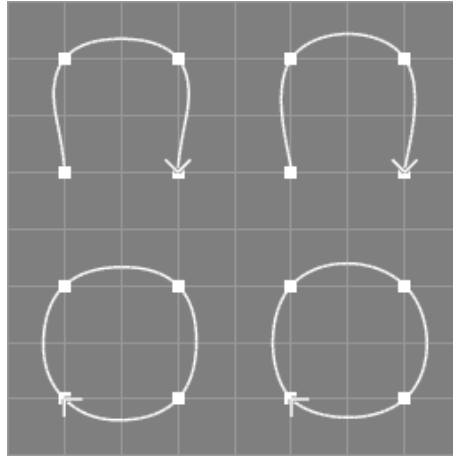


Figure 62: Different ICurves resulting from identical data point configurations (Upper Left: Open, Order 3; Upper Right: Open, Order 4; Lower Left: Closed, Order 3; Lower Right: Closed, Order 4)

The image above shows some interpolating curves, the left ones are of order 3 (quadratic curves), the right ones are of order 4 (cubic curves), the upper open, and the lower closed ones. The interpolation fidelity for the closed curves was tuned by adjusting the "SDLen" and "EDLen" parameters (all set to 0.2), see also the discussion of the parameters below.

In all parameterisation modes, knot averaging will be used to determine the knot vector of the interpolating curve.

Note that the axis of symmetry for closed interpolating curves crosses the first data point (in contrast to open interpolating or closed approximating curves, where it crosses between the last and first data point). For example, the closed interpolating curves in the above example image are indeed both symmetric, but the axis of symmetry is crossing the first and third data point and is, thus, rotated by 45 degrees.

This object makes use of the provide mechanism. It marks itself as providing a NCurve object (it creates and uses NURBS curves internally anyway) and all other objects that work with the provide mechanism (e.g. revolve, sweep, extrude, and skin) are able to work with an ICurve object instead of an object of type NCurve.

The following table briefly lists some capabilities of the ICurve object.

Type	Parent of	Material	Converts to/Provides	Point Edit
ICurve	No	No	NCurve	Yes

Table 55: ICurve Object Capabilities

ICurve objects may be created via the toolbox, the main menu, or the scripting interface, see also 6.2.2 Creating Objects (page 353).

ICurveAttr Property

The following parameters control the interpolation process:

- The "Type" parameter controls whether the interpolated curve should be open or closed.
- "Length" is the number of data points to interpolate.
- The next parameter "Order" determines the desired order of the interpolating curve. If the specified order is bigger than the number of control points used by the interpolating NURBS curve, then the order is silently changed to match the number of control points.
- The parameter "ParamType" switches the parameterisation between "Chordal" (default), "Centripetal", and "Uniform". The centripetal method generates a better parameterisation than the default (chordal) if the input data contains sharp turns. The uniform method is available since Ayam 1.20 and generates worse parameterisations (that lead to wiggles and overshooting) in the general case but it might help in some edge cases.
- "Derivatives" allows to choose between automatic and manual derivatives.
 If automatic derivatives are switched on, the direction of the derivatives will be determined from the first, second, second to last, and last data points for open curves and from the second and second to last data points for closed curves. In addition, the respective derivative vector will be scaled by "SDLen" and "EDLen".
 If manual derivatives are switched on, two more editable points appear in the single point editing modes. Those additional points directly control the derivatives for the endpoints of the interpolating curve. The parameters "SDLen" and "EDLen" do not influence those derivatives.
- The parameters "SDLen" and "EDLen" are used to control the length of the first and last derivative (if automatically generated from the data points, i.e. when "Derivatives" above is switched to automatic).
- See section 4.4.1 NCurveAttr (page 150) for a description of the parameters: "Tolerance" and "DisplayMode".
- Finally, a "NCInfo" field informs about the actual configuration of the created NURBS curve.

The parameters "Mode", "Closed", and "IParam" are gone since Ayam 1.16. "Closed" was replaced by "Type", "IParam" by "SDLen" and "EDLen", and the "Mode" is now determined automatically from the desired order.

See also the related tool to create interpolating NURBS curves: [5.3.22 Interpolate Tool \(page 301\)](#).

Conversion Support

The interpolating curve may be converted to an ordinary NURBS curve using the main menu entry "Tools/Convert".

RIB Export

ICurve objects never directly appear in RIB output (only indirectly as trim curve).

Scripting Support

To create ICurve objects using the scripting interface see section [6.2.2 crtOb ICurve command \(page 353\)](#).

4.4.3 ACurve (Approximating Curve) Object

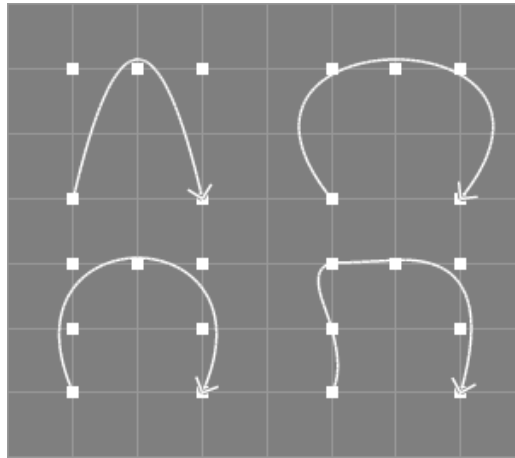


Figure 63: Different ACurves (Upper Left: ALength 3, Order 3; Upper Right: ALength 4, Order 4; Lower Left: ALength 4, Order 4; Lower Right: ALength 6, Order 3)

The ACurve object creates an approximating NURBS curve with m control points from n 3D non-rational ordered data points (see image above).¹

The number of desired output control points must be smaller than or equal to the number of data points to approximate ($m \leq n$). The approximation algorithm used is of the least squares variety. If the number of control points approaches the number of data points, undesired wiggles in the output curve may occur.

This object makes use of the provide mechanism. It marks itself as providing a NCurve object (it creates and uses NURBS curves internally anyway) and all other objects that work with the provide mechanism (e.g. Revolve, Sweep, Extrude, and Skin) are able to work with an ACurve object instead of an object of type NCurve.

The following table briefly lists some capabilities of the ACurve object.

Type	Parent of	Material	Converts to/Provides	Point Edit
ACurve	No	No	NCurve	Yes

Table 56: ACurve Object Capabilities

ACurve objects may be created via the main menu or the scripting interface, see also [6.2.2 Creating Objects](#) (page 355).

ACurveAttr Property

The following parameters control the approximation process:

- "Length" determines the number of data points to approximate.
- "ALength" is the number of (distinct) control points to use for the approximating NURBS curve. The total number of distinct control points must be smaller than or equal to the number of data points.
- The curve can be closed with the parameter "Closed". For closed approximations, the total number of control points will be "ALength + Order - 1". The following table illustrates this relationship.

¹ Since 1.15.

Length	ALength	Order	Closed	Output Length
10	5	3	No	5
10	5	3	Yes	8
10	4	4	Yes	8
5	4	3	No	4
5	4	3	Yes	7

Table 57: ACurve Parameterisation Examples

- For symmetric data point configurations, the approximating curve is not necessarily symmetric. With the parameter "Symmetric" a symmetric result can be enforced (see image below), albeit at the cost of about double runtime and a slightly worse parameterisation. For closed symmetric curves, "ALength" must be smaller than "Length".¹

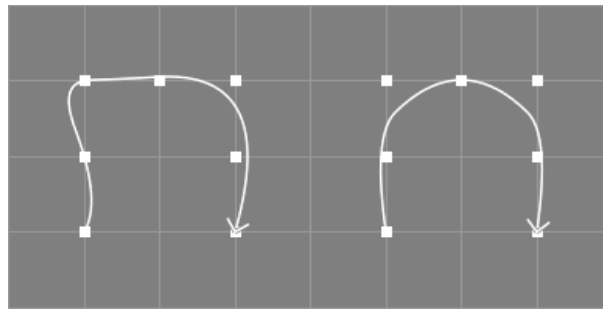


Figure 64: Left: asymmetric ACurve, Right: symmetric ACurve

- The parameter "Order" specifies the desired order of the approximating NURBS curve. Currently, only orders higher than 2 are supported.
- See section 4.4.1 [NCurveAttr](#) (page 150) for a description of the parameters: "Tolerance" and "DisplayMode".
- Finally, a "NCInfo" field informs about the actual configuration of the created NURBS curve.

See also the related tool to approximate existing NURBS curves: [5.3.23 Approximate Tool](#) (page 302).

Conversion Support

The approximating curve may be converted to an ordinary NURBS curve using the main menu entry "Tools/Convert".

RIB Export

ACurve objects never directly appear in RIB output (only indirectly as trim curve).

Scripting Support

To create ACurve objects using the scripting interface see section [6.2.2 crtOb ACurve](#) command (page 355).

¹ Since 1.32.

4.4.4 NCircle (NURBS Circle) Object

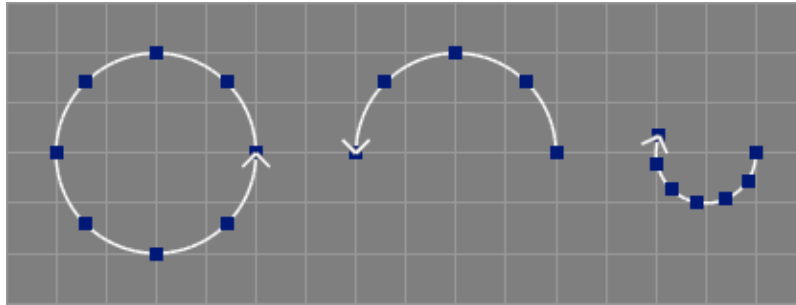


Figure 65: Different NCircle Objects (l: default, m: TMax 180, r: Radius 0.5, TMax -200)

The NCircle object creates a circular NURBS curve or a circular arc in the XY-plane centered at the origin with designated radius and start/end angles (see image above).¹

A full circle is created using nine rational control points in a rectangular configuration, circular arcs are created with fewer control points. See section 6.6.4 [NURBS Circle with Triangular Base \(page 463\)](#) for a Script object that implements a circle with less control points.

In order to revert the created NURBS curve the start/end angles may be used, e.g. "TMin" 0.0, "TMax" -360.0 for a reverse full circle.²

The following table briefly lists some capabilities of the NCircle object.

Type	Parent of	Material	Converts to/Provides	Point Edit
NCircle	No	No	NCurve	No*

Table 58: NCircle Object Capabilities

NCircleAttr Property

The following parameters control the shape of the circle or arc.

- "Radius" is the radius of the circle.
- "TMin" (ThetaMin) controls the starting angle of the circle or arc to be created. Negative values are allowed.
- "TMax" (ThetaMax) controls the end angle of the circle or arc to be created. Negative values are allowed.
- See section 4.4.1 [NCurveAttr \(page 150\)](#) for a description of the parameters: "Tolerance" and "DisplayMode".
- Finally, a "NCInfo" field informs about the actual configuration of the created NURBS curve.

¹ Since 1.12. ² Since 1.15.

Conversion Support

The circular curve/arc may be converted to an ordinary NURBS curve using the main menu entry "Tools/Convert".

RIB Export

NCircle objects never directly appear in RIB output (only indirectly as trim curve).

4.4.5 More Curve Types

More basic curve types are available via plugins and scripts, those are:

- Basis Curve: [4.9.4 BCurve object \(page 241\)](#),
- Subdivison Curve: [4.9.5 SDCurve object \(page 243\)](#),
- Superformula Curve: [4.9.3 SFCurve object \(page 239\)](#),
- NURBS circle with triangular base: [6.6.4 TCircle script object \(page 463\)](#),
- Helix Curve: [6.6.5 Helix script object \(page 464\)](#), and
- Spiral Curve: [6.6.6 Spiral script object \(page 465\)](#).

4.5 Curve Tool Objects

These objects modify existing curves or create new curves.

4.5.1 ConcatNC (Concatenate NURBS Curves) Object

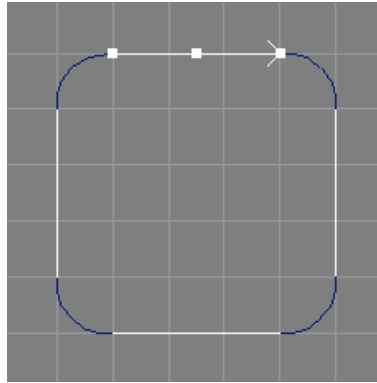


Figure 66: ConcatNC Object (blue) from a Linear Curve and 3 Instances (white)

The ConcatNC object concatenates all child objects (which should be NURBS curves or provide NURBS curves) to a single NURBS curve. Since the ConcatNC object also provides a NURBS curve, it is possible to use it as child object for another ConcatNC object (with possibly different parameters) or as a parameter object for a tool object that works with NURBS curves such as Revolve or Extrude.

The concatenation process works as follows:

1. The orders of all parameter curves will be elevated to the maximum order of all the parameter curves (see also section 5.3.8 [elevate tool \(page 286\)](#) for more information on elevation) and all curves will be clamped (see also section 5.3.11 [clamp tool \(page 289\)](#) for more information on clamping).
2. If the parameter "FillGaps" is enabled, fillet curves will be created for every gap between the parameter curves of the ConcatNC object. If "Closed" and "FillGaps" are enabled, an additional fillet is created to close the curve.
3. Now the control points of all parameter curves and fillets are simply copied into a new big control point vector, without checking for double points. This means that for parameter curves that touch at their respective ends, at least double control points in the new concatenated curve will result.
4. If "Closed" is enabled, the curve will be closed.

The knot sequence of the new concatenated curve will be of type "NURB" or a custom knot vector will be computed (depending on the setting of "Knot-Type").

If "Knot-Type" is "NURB", the shape of the concatenated curve will differ from the parameter curves if any of the parameter curves has a custom knot vector with non equidistant knots.

If "Knot-Type" is "Custom", the shape of the parameter curves will be preserved completely, but the knot vector of the concatenated curve will then contain internal multiple knots.

Attributes like display mode and tolerance for the new concatenated curve are simply taken from the first parameter curve.

For best results, only clamped curves should be used as parameter objects.

The following table briefly lists some capabilities of the ConcatNC object.

Type	Parent of	Material	Converts to/Provides	Point Edit
ConcatNC	NCurve ⁺	No	NCurve	No*

Table 59: ConcatNC Object Capabilities

ConcatNCAttr Property

The following parameters control the concatenation process:

- Using "Closed" a closed concatenated curve may be created, even if the parameter curves do not touch. If also "FillGaps" (see below) is enabled, an additional fillet will be created for the last and the first child curve to close the concatenated curve. If "FillGaps" is not enabled, the concatenated curve will be closed with the same algorithm that is also used by the close curve tool (possibly changing its shape).
- If "Revert" is enabled, the orientation of the concatenated curve will be reversed.
- "FillGaps" creates fillet curves for all gaps between the parameter curves of the ConcatNC object. No fillet will be created if the end points of two parameter curves match.

The fillet curves will initially be created from four control points. The outer fillet control points are the parameter curve end points and the inner fillet control points will be positioned on the tangent of the respective parameter curve end point (see also the discussion of "FTLength" below). Thus, the transitions between parameter curve and fillet should be at least G1 continuous. Degree elevation will be used to raise the degree of the fillet to that of the concatenated curve; this may introduce additional control points in the fillet but its shape does not change and the transition continuity is also not affected.

If the order of the resulting concatenated curve is 2, only simple fillets, connecting the parameter curves by straight lines, will be generated.

- "FTLength" determines the distance of the inner fillet control points from their respective end points. This value can be adapted for smaller/larger gaps between parameter curves.

If the "FTLength" parameter is 0.0, C1 continuous fillets will be created by global curve interpolation instead of the G1 fillets outlined above.

- "Knot-Type" sets the knot type of the concatenated curve:
 - If "Knot-Type" is "NURB" a simple knot vector with equidistant knots is generated, which leads to a concatenated curve that does not exactly preserve the shapes of the original curves if any of the parameter curves has a custom knot vector with non equidistant knots. Furthermore, all transitions between parameter curves are always smoothed out.
 - If "Knot-Type" is "Custom", the knot vector is composed from the knot vectors of the original curves, and thus, their shapes may be preserved completely. Note, that potential discontinuities of any level, even gaps between the parameter curves are also fully preserved. However, this comes at the price of internal multiple knots. A problem with these knots is, that the resulting curve is *not* differentiable in these places anymore, which in turn can be problematic for operations like sweeps.

- Finally, a "NCInfo" field informs about the actual configuration of the created NURBS curve.

Conversion Support

The concatenated curve may be converted to an ordinary NURBS curve using the main menu entry "Tools/Convert".

RIB Export

ConcatNC objects never directly appear in RIB output (only indirectly as trim curve).

4.5.2 ExtrNC (Extract NURBS Curve) Object

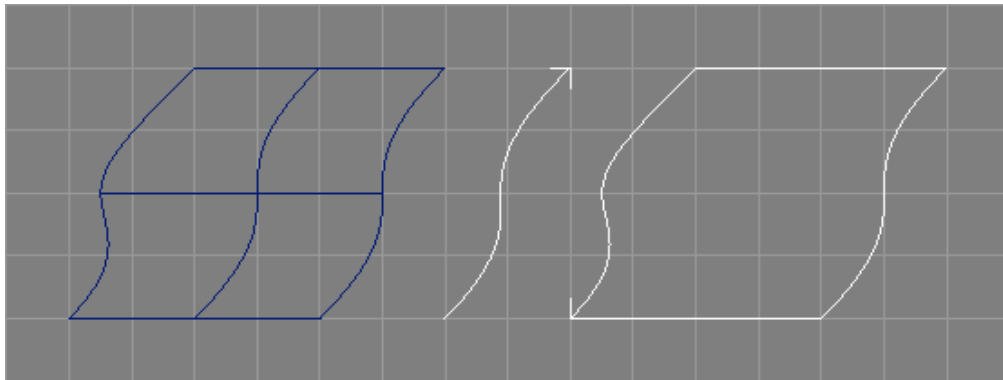


Figure 67: Extracted Curves (white) from Arbitrarily Shaped Surface (blue)

The ExtrNC object extracts a NURBS curve from a NURBS patch object, for use as parameter object for other tool objects, like e.g. Revolve (see image above). It also works with NURBS patch providing objects, so that the following example hierarchy is valid:

```

|-NPatch    <-----'.
+-Skin      |
  +-ExtrNC  |
    | \-Instance_of_NPatch(Instance) --'
    \-NCurve

```

Depending on the parameters of the ExtrNC object, the Skin object above will have one boundary in common with a boundary or an iso-curve of the NPatch object. Note that using an instance object of some other surface object (as shown in the above example) is in fact the recommended way of using the ExtrNC object. Therefore, the main menu entry "Tools/Create/ExtrNC" will automatically create an instance of the currently selected object and move it to the newly created ExtrNC object.

As the geometry of the extracted curve is completely defined by the master surface, ExtrNC objects do not support own transformation attributes.¹

Note that the extraction of any curves currently completely ignores potentially present trimming information of the NPatch object, i.e. the extracted curves will not be trimmed.

However, complete trim boundaries may be extracted.²

The following table briefly lists some capabilities of the ExtrNC object.

Type	Parent of	Material	Converts to/Provides	Point Edit
ExtrNC	NPatch	No	NCurve	No*

Table 60: ExtrNC Object Capabilities

¹ Since 1.19. ² Since 1.21.

ExtrNCAttr Property

The extraction process is controlled by the following attributes:

- "Side" controls which curve should be extracted from the surface. Available values are:
 - "U0", "Un": extract upper or lower boundary curve (along width);
 - "V0", "Vn": extract left or right boundary curve (along height);
 - "U", "V" extract curve along width and height respectively at specified parametric value (see below).¹
 - "Boundary": extract the complete boundary curve of the patch, by extracting four boundary curves, automatically selecting the proper method depending on the knot type of the surface (e.g. "U0" or "U"), reversing two and concatenating all non degenerated curves to the result.² Note that if the surface has differing orders, the partial curves in the direction with the lower order will be elevated to the higher order before concatenation and therefore those parts of the resulting boundary curve will *not* be compatible to the underlying surface anymore.
 - "Middle_U", "Middle_V": create a curve from the patch data that is the "middle axis" (simply the medium of all control points of a patch in the designated dimension).³ This option is e.g. useful to re-engineer swept surfaces, delivered as simple patches.
 - "Trim", all trim curves and loops of the current patch will appear here.⁴ Note, that their extraction will result in an approximation of the 3D representation of the corresponding trim boundary only. In particular, the extracted curve will not be compatible to the trim curve. The approximation quality can be adjusted using "Parameter" (see below).

Note that if "Side" is "U0", "Un", "V0", or "Vn" the extraction process just copies the respective boundary control points, which is fast but only works correctly for clamped knot vectors. To extract a boundary from a surface with e.g. a B-Spline knot vector, "U0" should *not* be used but rather "U" with "Parameter" set to 0.0 and "Relative" enabled as "U" uses a different, more expensive, extraction process (which involves knot insertion).

- "Parameter" controls the parametric value in U or V direction in the parameter space of the NURBS patch object where the curve should be extracted or the approximation quality of extracted trim curves. Consequently, this parameter is only used when "Side" is "U", "V" or "Trim". The valid range of parameter values depends on the knot vectors of the NURBS patch. If a trim boundary is to be extracted, this parameter controls the sampling just like the preference setting "Drawing/Tolerance": smaller values lead to dense sampling and higher quality, the useful range of values is 0.01 to 100.
- "Relative" controls whether or not the parametric value delivered via "Parameter" above should be interpreted in a relative way. If enabled, a parametric value of 0.5 always extracts from the middle of the knot vector/surface, regardless of the actual knot values, and the valid range for "Parameter" is then consequently 0.0-1.0.⁵
- "Revert" immediately reverts the extracted curve.
- "CreatePVN" controls creation of a PV or NT tag that contains just the normals or normals and tangents on the surface. These tags can then be used to e.g. control a 3D offset curve. The normals and tangents will be derived from the surrounding surface control points except for trim boundary extraction where the underlying surface will be evaluated exactly.

Note that the tag will be present on provided/converted curve objects only.

When extracting a middle axis, this option is ignored.

¹ Since 1.8.1. ² Since 1.13. ³ Since 1.15. ⁴ Since 1.21. ⁵ Since 1.15.

- "PatchNum" allows to select a patch from a list of patches delivered e.g. by a beveled Extrude object as child of the ExtrNC object. This way it is possible to extract a curve from a bevel or cap surface of e.g. said Extrude object.
- Finally, a "NCInfo" field informs about the actual configuration of the extracted NURBS curve.

See section [4.4.1 NCurveAttr \(page 150\)](#) for a description of the other two attributes "DisplayMode" and "Tolerance".

Conversion Support

The extracted curve may be converted to an ordinary NURBS curve using the main menu entry "Tools/Convert".

RIB Export

ExtrNC objects never directly appear in RIB output (only indirectly as trim curve).

4.5.3 OffsetNC (Offset NURBS Curves) Object

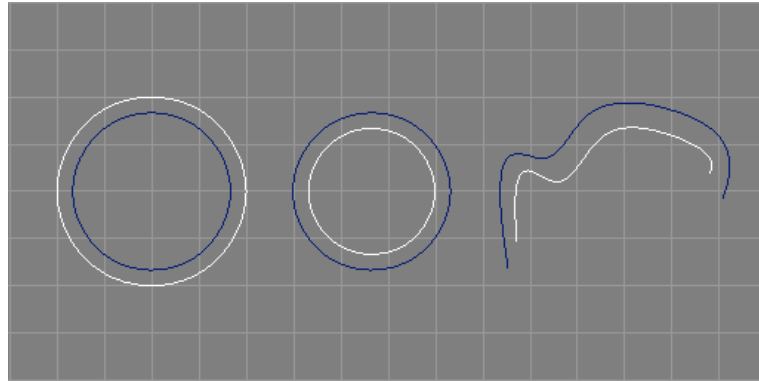


Figure 68: Offset Curves (white) from NURBS Curves (blue) with Offset 0.2, -0.2, and 0.3

The OffsetNC object creates offset curves from planar NURBS curves using different algorithms.¹ See also the image above.

The offset curve will always match the original curve in type, length, order, and knots. No attempt is made to prevent collisions or self intersections. Rational curves are not supported well.

The available offset algorithms are:

Point

offsets each control point along the normal derived from the surrounding control points; the offset curve created by this algorithm may come too near the original curve at sharp convex features whose exact forms are not preserved well either,

Section

this algorithm offsets all control polygon sections in the direction of their normal and places new control points at intersection points of the lines defined by the offset sections; this algorithm is better in avoiding self intersections of the offset curve but the offset curve may be too far away from the original curve at sharp convex or concave features (regions of high curvature),

Hybrid

this algorithm offsets the curve two times using the Point and Section algorithms and then mixes the results so that the bad features of the two algorithms cancel out each other,²

3DPVN³

this algorithm creates true three dimensional offsets from non planar curves using a primitive variable tag that contains normal information for the curve (vertex normals). Such tags can be created manually or automatically e.g. when extracting curves from surfaces using the "ExtNC" object.

3DPVNB

Three dimensional offset like 3DPVN above, but offsets in the direction of the binormal.⁴

PlaneNormal⁵

this algorithm creates true three dimensional offsets from planar curves using the mean normal of the plane, the curve is defined in. Since all points are moved by the same amount, the shape and planarity of the curve will be preserved.

¹ Since 1.14. ² Since 1.19. ³ Since 1.18. ⁴ Since 1.30. ⁵ Since 1.35.

As the geometry of the offset curve is completely defined by the master curve and the offset parameter, OffsetNC objects do not support own transformation attributes.¹

The "Bevel3D" offset algorithm has been removed since Ayam 1.19.

The following table briefly lists some capabilities of the OffsetNC object.

Type	Parent of	Material	Converts to/Provides	Point Edit
OffsetNC	NCurve	No	NCurve	No*

Table 61: OffsetNC Object Capabilities

OffsetNCAttr Property

The following parameters control the offsetting process:

- The first parameter "Mode" determines, which algorithm to use for the offsetting process.
- Using "Revert" the direction of the offset curve may be reversed.
- "Offset" determines the distance between original curve and offset curve. Negative values are allowed.
- See section 4.4.1 [NCurveAttr \(page 150\)](#) for a description of the parameters: "Tolerance" and "DisplayMode".
- Finally, a "NCInfo" field informs about the actual configuration of the created NURBS curve.

Conversion Support

The offset curve may be converted to an ordinary NURBS curve using the main menu entry "Tools/Convert".

RIB Export

OffsetNC objects never directly appear in RIB output (only indirectly as trim curve).

¹ Since 1.19.

4.6 Freeform Surface Objects

These objects enable direct manipulation of freeform surfaces.

4.6.1 NPatch (NURBS Patch) Object

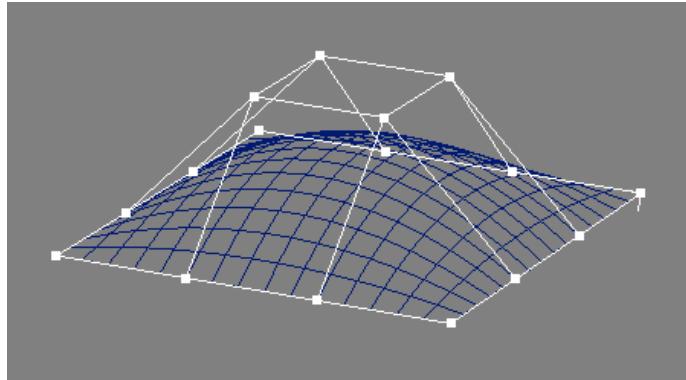


Figure 69: NURBS Patch: Surface (blue) from Control Polygon (white)

The NPatch object allows to model NURBS surfaces in a direct way, e.g. by modifying control points (see also the image above). Note that using NPatch objects should be seen as last resort, only to be used when the flexibility of all the NURBS surface creating tool objects is not sufficient to achieve a certain shape.

Like NCurve objects, NPatch objects mark their last control point with a small arrow. Note that the arrow points in the V direction (height).

NPatch objects also support the concept of multiple points, see section 4.4.1 Multiple Points (page 153) for more information regarding this.¹

The following table briefly lists some capabilities of the NPatch object.

Type	Parent of	Material	Converts to/Provides	Point Edit
NPatch	NCurve ⁺ /Level ⁺	Yes	PolyMesh	Yes

Table 62: NPatch Object Capabilities

NPatch objects may be created via the toolbox, the main menu, some tools as documented in section 5.4 Surface Creation Tools (page 308), or the scripting interface, see also 6.2.2 Creating Objects (page 356).

Tools that process NPatch objects are documented in section 5.5 Surface Modification Tools (page 317).

NPatchAttr Property

The first section of the NPatchAttr property contains patch specific settings:

- "Width" and "Height" control the dimensions of the patch. Similar to the "Length" parameter of the NCurve object, changes to "Width" or "Height" add or remove internal control points (i.e. to double the resolution of a 4 by 4 NURBS patch in U direction, change the "Width" from 4 to 7; this will lead to an additional control point inserted into every section of the original patch).
- "Order_U" and "Order_V" set the orders of the patch.

¹ Since 1.10.

- "Knot-Type_U"/"Knot-Type_V" and "Knots_U"/"Knots_V": For a discussion of the "Knot-Type" and "Knots" parameters, please see section 4.4.1 NCurveAttr (page 150).
- "CreateMP" toggles, whether multiple points should be created for this surface. See also the discussion in section 4.4.1 Multiple Points (page 153).¹
- "IsRat" informs, whether the patch is rational (has any weight values different from 1.0).²

The next parameters control the appearance of the patch for display in Ayam:

- "Tolerance" is in fact the GLU sampling tolerance used to control the quality of the sampling when rendering the patch. Smaller tolerance settings lead to higher quality but also slower display. A setting of 0.0 means, that the global preference setting "Drawing/Tolerance" should be used.
- "DisplayMode" sets the display mode, either the control hull is drawn ("ControlHull"), or just the outlines of the polygons created by the tessellation ("OutlinePolygon"), or just the outlines of the patch ("OutlinePatch"). The default setting ("Global") means, that the value of the global preference setting "Drawing/NPDisplayMode" should be used instead.

Trim Curves

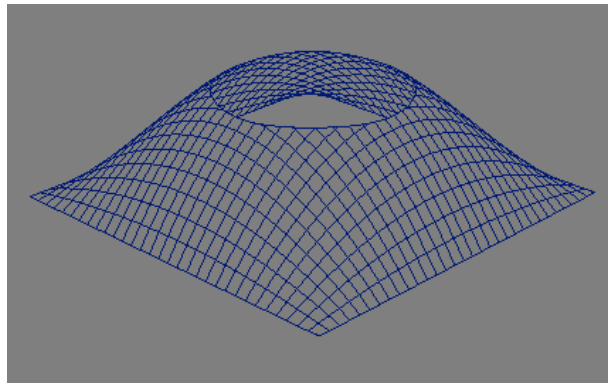


Figure 70: A Trimmed NURBS Patch

Trim curves may be used to cut out certain parts of a NURBS patch (see also the image above). They are simple 2D NURBS curves defined in the parametric space of the associated NURBS patch. Trim curves have to be defined as child objects of the NURBS patch object they belong to. In contrast to other child objects, however, they do not inherit the transformation attributes of the parent object. Trim curve editing can take place in views of type "Trim", that draw the boundaries of the parametric space of the corresponding NURBS patch as rectangle, but otherwise act as normal "Front" views.

Note that the direction of the trim curve determines which part of the NURBS patch should be cut out. The Revert tool ("Tools/Curve" menu) can be used to easily change the direction of a trim curve.

Some special restrictions apply to trim curves:

- All trim curves should entirely lie in the (u,v) parameter space of the NURBS patch (remember the rectangle in the Trim view). Note that this restriction does not apply to the control points, but the curves! It is ok to have control points outside the rectangle if the defined curve is inside the rectangle.
- The last point of a trim curve must be identical to the first point.

¹ Since 1.10. ² Since 1.9.

- Trim loops (multiple trim curves that form loops) are possible too; the last point of each curve in the loop must be identical to the first point of the next curve in the loop and the first point of the first curve of a loop must be identical to the last point of the last curve of that loop.
- To mark a set of curves to be a loop, they must be placed in a level object. The order of the curves in this level is the order of the loop. The transformation attributes of this level object are fully ignored for trimming.
- Drawing trimmed NURBS patches with certain implementations of OpenGL may require a special trim curve (a rectangular piecewise linear curve that encloses the whole NURBS patch) to be present. Such a curve may be generated with the `TrimRect` tool. This tool can be found in the "Tools/Create" menu. This curve is needed if you want to cut out a hole with a single trim curve. This curve is generally not needed if you want to render the patch with BMRT but it should not hurt if it is present.
- If there are nested trim curves, their direction must alternate.
- Trim curves may not intersect each other or them self.

Note that Ayam is not checking whether the trim curves follow these rules.

Warning: Certain OpenGL implementations may be easily crashed drawing trimmed NURBS patches with trims that do not follow the aforementioned rules. When in doubt or while heavy modelling, switch to wire-frame drawing and switch off shading temporarily and you will be on the safe side.

NURBS curve providing objects are also supported as trim curves.¹ When objects provide multiple NURBS curves, those do *not* form a single loop, but are seen as individual loops.

Caps and Bevels

The NPatch object supports the standard caps as lined out in section 4.10.5 Caps Property (page 255) and the standard bevels as lined out in section 4.10.6 Bevels Property (page 256).

The boundary names are U0, U1, V0, and V1.

Integration is not supported; the corresponding option is silently ignored.

Conversion Support

A NPatch object may be converted to a PolyMesh object using the main menu entry "Tools/Convert".

This process is also called tessellation and thus, the tessellation parameters from TP tags will be used in the conversion process (if present) (see also section 4.11.11 TP Tag (page 265)).

If bevels or caps are present, an enclosing Level object will be created and the tessellated bevels or caps follow the tessellated NPatch in the following order: U0, U1, V0, V1.

¹ Since 1.5.

RIB Export

NPatch objects will be exported as NURBS patch primitives:

```
RiNuPatch (...);
```

PV tags are supported and also trim curves may appear.

Multiple TC tags are also supported.¹

Scripting Support

To create NPatch objects using the scripting interface see section [6.2.2 crtOb NPatch command](#) (page 356). For scripting interface commands that manipulate NURBS patch objects see section [6.2.16 Manipulating NURBS Surfaces](#) (page 400)).

¹ Since 1.24.

4.6.2 IPatch (Interpolating Patch) Object

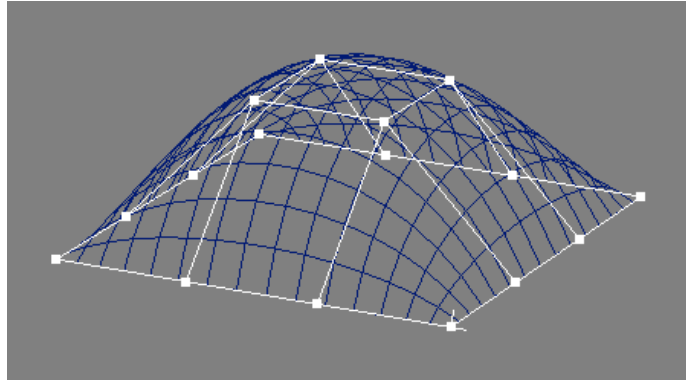


Figure 71: Interpolating surface (blue) from data points (white)

An IPatch forms a surface defined by interpolating a regular grid of three dimensional and non rational data points (see also the image above).

The following table briefly lists some capabilities of the IPatch object.

Type	Parent of	Material	Converts to/Provides	Point Edit
IPatch	No	Yes	NPatch	Yes

Table 63: IPatch Object Capabilities

IPatch objects may be created via the main menu or the scripting interface, see also 6.2.2 Creating Objects (page 358).

IPatchAttr Property

The IPatchAttr property contains the following elements:

- "Width" and "Height" control the dimensions of the patch. Similar to the "Length" parameter of the NCurve object, changes to "Width" or "Height" add or remove internal control points (i.e. to double the resolution of a 4 by 4 interpolating patch in U direction, change the "Width" from 4 to 7; this will lead to an additional control point inserted into every section of the original patch).
- "Order_U" and "Order_V" set the desired interpolation orders of the patch. If any of these values is set to 2, no explicit interpolation will take place in this dimension (the surface will implicitly interpolate the data points due to the low order).
- "Close_U" and "Close_V" allow to create closed surfaces in the respective dimension.
- "Knot-Type_U" and "Knot-Type_V" switches the parameterisation between "Chordal" (default), "Centripetal", and "Uniform". The centripetal method generates a better parameterisation than the default (chordal) if the input data contains sharp turns. The uniform method generates worse parameterisations (that lead to wiggles and overshooting) in the general case but it might help in some edge cases.
- "Deriv_U" and "Deriv_V" toggle whether
 - "None": no end derivatives,
 - "Automatic": automatically created (from data points) derivatives, scaled by the additional parameters "SDLen_U", "EDLen_U", "SDLen_V", and "EDLen_V",

- "Manual": completely manually controlled end derivatives (appearing as additional control points in point editing, if enabled)

should be used in the interpolation.

The next parameters control the appearance of the patch for display in Ayam:

- "Tolerance" is in fact the GLU sampling tolerance used to control the quality of the sampling when rendering the patch. Smaller tolerance settings lead to higher quality but also slower display. A setting of 0.0 means, that the global preference setting "Drawing/Tolerance" should be used.
- "DisplayMode" sets the display mode, either the control hull is drawn ("ControlHull"), or just the outlines of the polygons created by the tessellation ("OutlinePolygon"), or just the outlines of the patch ("OutlinePatch"). The default setting ("Global") means, that the value of the global preference setting "Drawing/NPDisplayMode" should be used instead.
- Finally, a "NPInfo" field informs about the actual configuration of the created NURBS patch.

See also the related tool to create interpolating NURBS surfaces: [5.5.16 Interpolate Surface Tool \(page 329\)](#).

Caps and Bevels

The IPatch object supports the standard caps as lined out in [section 4.10.5 Caps Property \(page 255\)](#) and the standard bevels as lined out in [section 4.10.6 Bevels Property \(page 256\)](#).

The boundary names are U0, U1, V0, and V1.

Conversion Support

The interpolated surface may be converted to an ordinary NURBS patch using the main menu entry "Tools/Convert".

RIB Export

IPatch objects will be exported as NURBS patch primitives:

```
RiNuPatch(...);
```

PV tags are currently not supported.

Multiple TC tags are also supported.¹

Scripting Support

To create IPatch objects using the scripting interface see [section 6.2.2 crtOb IPatch command \(page 358\)](#).

¹ Since 1.24.

4.6.3 APatch (Approximating Patch) Object

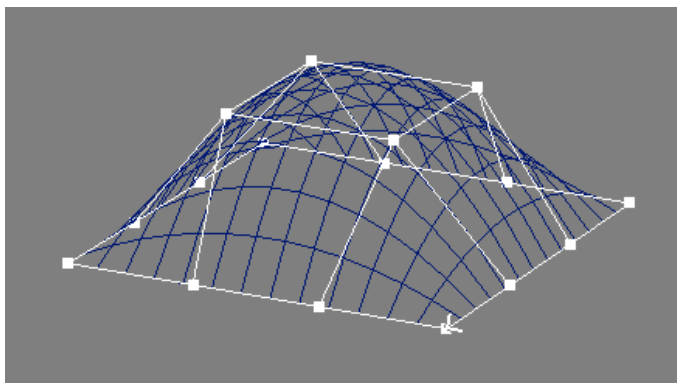


Figure 72: Approximating surface (blue) from data points (white)

An APatch forms a surface defined by approximating a regular grid of three dimensional and non rational data points (see also the image above).¹ The approximation will occur first in U then in V direction, first in V then in U direction, just in U, or just in V direction.

The following table briefly lists some capabilities of the APatch object.

Type	Parent of	Material	Converts to/Provides	Point Edit
APatch	No	Yes	NPatch	Yes

Table 64: APatch Object Capabilities

APatch objects may be created via the main menu or the scripting interface, see also [6.2.2 Creating Objects](#) (page 360).

APatchAttr Property

The APatchAttr property contains the following elements:

- "Width" and "Height" control the number of data points of the patch to approximate. Similar to the "Length" parameter of the NCurve object, changes to "Width" or "Height" add or remove internal control points (i.e. to double the resolution of a 4 by 4 approximating patch in U direction, change the "Width" from 4 to 7; this will lead to an additional control point inserted into every section of the original patch).
- "Mode" determines the order in which the partial approximations occur.
- "AWidth" and "AHeight" set the desired number of NURBS control points. Both values must be smaller or equal to "Width" and "Height" respectively.
- "Order_U" and "Order_V" set the desired approximation orders of the patch.
- "Knot-Type_U" and "Knot-Type_V" switches the parameterisation between "Chordal" (default) and "Centripetal". The centripetal method generates a better parameterisation than the default (chordal) if the input data contains sharp turns. If the approximation mode is "U" or "V" more knot types will be available for the dimension where no approximation occurs. Those knot types are: "Bezier", "B-Spline", and "NURB". See section [4.4.1 NCurveAttr Property](#) (page 150) for their documentation.

¹ Since 1.30.

- "Close_U" and "Close_V" allow to create closed surfaces in the respective dimension.

The next parameters control the appearance of the patch for display in Ayam:

- "Tolerance" is in fact the GLU sampling tolerance used to control the quality of the sampling when rendering the patch. Smaller tolerance settings lead to higher quality but also slower display. A setting of 0.0 means, that the global preference setting "Drawing/Tolerance" should be used.
- "DisplayMode" sets the display mode, either the control hull is drawn ("ControlHull"), or just the outlines of the polygons created by the tessellation ("OutlinePolygon"), or just the outlines of the patch ("OutlinePatch"). The default setting ("Global") means, that the value of the global preference setting "Drawing/NPDisplayMode" should be used instead.
- Finally, a "NPInfo" field informs about the actual configuration of the created NURBS patch.

Caps and Bevels

The APatch object supports the standard caps as lined out in section 4.10.5 Caps Property (page 255) and the standard bevels as lined out in section 4.10.6 Bevels Property (page 256).

The boundary names are U0, U1, V0, and V1.

Conversion Support

The approximating surface may be converted to an ordinary NURBS patch using the main menu entry "Tools/Convert".

RIB Export

APatch objects will be exported as NURBS patch primitives:

```
RiNuPatch(...);
```

PV tags are currently not supported.

Multiple TC tags are also supported.

Scripting Support

To create APatch objects using the scripting interface see section 6.2.2 crtOb APatch command (page 360).

4.6.4 BPatch (Bilinear Patch) Object

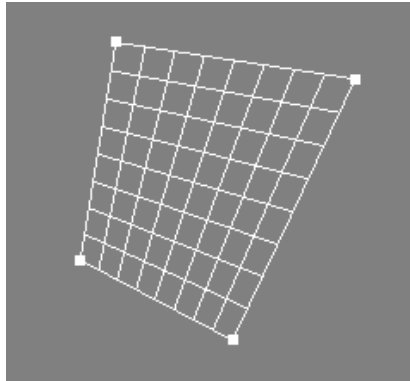


Figure 73: Bilinear Patch

A BPatch is a simple bilinear patch defined by four control points. BPatch objects are e.g. used internally to build box objects, see also [4.3.1 Box Object \(page 142\)](#).

The following table briefly lists some capabilities of the BPatch object.

Type	Parent of	Material	Converts to/Provides	Point Edit
BPatch	No	Yes	NPatch	Yes

Table 65: BPatch Object Capabilities

BPatchAttr Property

The BPatchAttr property allows to directly control the four points defining the geometry of the patch:

- "P1_X", "P1_Y", "P1_Z", first point.
- "P2_X", "P2_Y", "P2_Z", second point.
- "P3_X", "P3_Y", "P3_Z", third point.
- "P4_X", "P4_Y", "P4_Z", fourth point.

Conversion Support

The bilinear patch may be converted to an ordinary NURBS patch using the main menu entry "Tools/Convert".

RIB Export

BPatch objects will be exported as bilinear patch primitives:

```
RiPatch(RI_BILINEAR, ...);
```

PV tags are supported.

4.6.5 PatchMesh Object

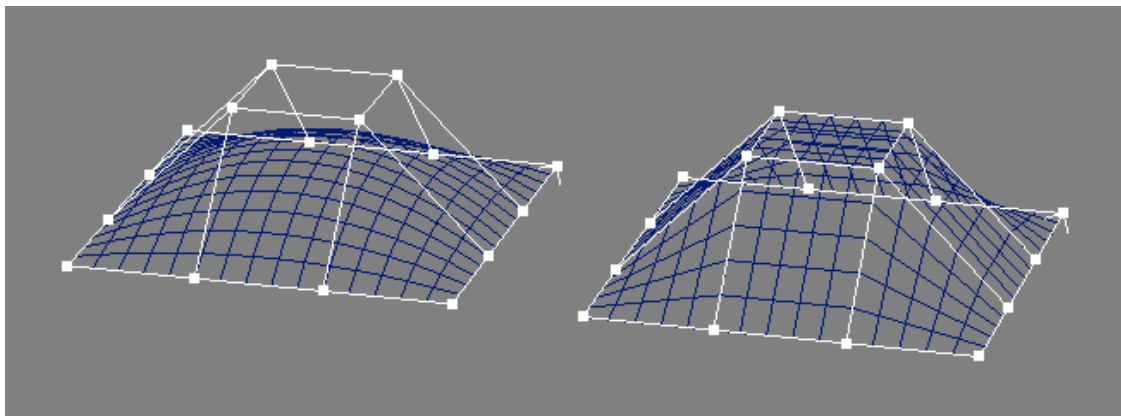


Figure 74: Bicubic (left) and Bilinear (right) PatchMesh Surfaces (blue) from their Respective Control Polygons (white)

The PatchMesh object may be used to model with bicubic and bilinear patch meshes, see the image above for two examples.

Like NCurve objects, PatchMesh objects mark their last control point with a small arrow. Note that the arrow points in the V direction (height).

The following table briefly lists some capabilities of the PatchMesh object.

Type	Parent of	Material	Converts to/Provides	Point Edit
PatchMesh	No	Yes	NPatch	Yes

Table 66: PatchMesh Object Capabilities

PatchMeshAttr Property

The first section of the PatchMeshAttr property contains patch specific settings:

- "Type" may be set to "Bicubic" or "Bilinear".
- "Width" and "Height" control the dimensions of the patch. Note that for bicubic patch meshes, the basis type may impose restrictions on the valid values for width and height; e.g. for the basis type "Bezier" (with step size 3) valid values are: 4, 7, 10, Another basis type that imposes this restriction is "Hermite" (with step size 2), where valid values are: 4, 6, 8,
Closing the surface complicates the matter a bit further. A valid width for a closed bicubic patch mesh of basis type "Bezier" is e.g. 6; it is actually still 7, but the last point is equal to the first and will be omitted.
The arrow buttons of the dimension entry fields add/subtract the current step size of the patch to/from the respective value when the <Control> key is held down.¹
- "Close_U" and "Close_V" determine, whether the patch mesh should be closed in U and V direction respectively.
- "IsRat" informs, whether the patch mesh is rational (has any weight values different from 1.0).²

¹ Since 1.25. ² Since 1.21.

- "BType_U" and "BType_V" control the basis type for bicubic patches. The following basis types are available: "Bezier", "B-Spline", "Catmull-Rom", "Hermite", "Power", and "Custom". In the latter case ("Custom"), additional parameters may be set. Those are "Step_U"/"Step_V" (the step size of the basis) and "Basis_U"/"Basis_V" the basis itself. When a basis type is changed to "Custom", the initial values of the basis matrix will be of those of the previous type.¹

Also note, that changing a basis type via the property GUI is not the same as converting a patch mesh via the `tobasisPM` command, see also section 6.2.16 [tobasisPM command](#) (page 411).

Please see the RenderMan Companion for a more in depth discussion of the different basis types.

The parameters "BType_U", "BType_V", "Step_U", "Step_V", "Basis_U", and "Basis_V" are only available to bicubic patch meshes.

The next parameters control the appearance of the patch for display in Ayam:

- "Tolerance" is in fact GLU sampling tolerance, used to control the quality of the sampling when rendering the patch. A setting of 0.0 means, that the global preference setting "Drawing/Tolerance" should be used.
- "DisplayMode" sets the display mode, either the control hull is drawn, or just the outlines of the polygons created by the tessellation (OutlinePolygon), or just the outlines of the patch (OutlinePatch). The default setting (Global) means, that the global preference setting "Drawing/DisplayMode" should be used.

Caps and Bevels

The PatchMesh object supports the standard caps as lined out in section 4.10.5 [Caps Property](#) (page 255) and the standard bevels as lined out in section 4.10.6 [Bevels Property](#) (page 256).²

The boundary names are U0, U1, V0, and V1.

Conversion Support

The patch mesh may be converted to an ordinary NURBS patch using the main menu entry "Tools/Convert". In Ayam versions prior to 1.21, conversion (and shaded display) did not work for patch meshes with the basis types Catmull-Rom, Hermite, or Custom. This is no longer the case.

¹ Since 1.24. ² Since 1.24.

RIB Export

PatchMesh objects will be exported as patch mesh primitives:

```
RiPatchMesh (...);
```

PV tags are supported.

Multiple TC tags are also supported.¹

Scripting Support

To create PatchMesh objects using the scripting interface see section [6.2.2 crtOb PatchMesh command](#) (page 362).

¹ Since 1.24.

4.7 Surface Tool Objects

These objects create freeform surfaces from curves or other surfaces.

4.7.1 Revolve Object



Figure 75: Revolve Object (left: Curve, right: Surface of Revolution)

The Revolve object forms a surface of revolution from a NURBS curve, see the image above for an example.

The Revolve object has the generating NURBS curve as child object and watches its changes and adapts to it automatically.

The axis of revolution is always the Y-axis. The parameter curve should be defined in the XY-plane. If not, the curve will be projected to this plane before revolving.

The following table briefly lists some capabilities of the Revolve object.

Type	Parent of	Material	Converts to/Provides	Point Edit
Revolve	NCurve	Yes	NPatch ⁺	No*

Table 67: Revolve Object Capabilities

RevolveAttr Property

The parameter "ThetaMax" specifies the sweeping angle of the revolution just like for an ordinary RenderMan quadric primitive.

The Revolve object also supports a B-Spline mode that may be enabled by setting the parameter "Sections" to a value higher than 0.¹

In this mode, a circular B-Spline is used as basis for the surface of revolution instead of the standard NURBS circle. Depending on the number of sections chosen, the surface of revolution does not exactly interpolate the parameter curve, but the surface may be edited more easily after a possible conversion to an ordinary NURBS patch object, because the control points will not be rational if the revolved curve is also not rational. Note that also the B-Spline mode can realize arbitrary "ThetaMax" values.²

In addition to the number of sections, in B-Spline mode it is possible to control the order of the surface of revolution using the parameter "Order". If "Order" is 0, a standard value of 3 will be used.

¹ Since 1.8. ² Since 1.18.

See section 4.6.1 [NPatchAttr \(page 170\)](#) for a description of the other two attributes "DisplayMode" and "Tolerance".

Caps and Bevels

The Revolve object supports the standard caps as lined out in section 4.10.5 [Caps Property \(page 255\)](#) and the standard bevels as lined out in section 4.10.6 [Bevels Property \(page 256\)](#).

The boundary names are:

Upper – curve formed by revolving the start point of the cross section,

Lower – curve formed by revolving the end point of the cross section,

Start – cross section, and

End – cross section at end of revolution.

Conversion Support

The surface of revolution and the bevels and caps may be converted to ordinary NURBS patches using the main menu entry "Tools/Convert".

If bevels or caps are present, an enclosing Level object will be created and the bevels or caps follow the surface of revolution in the following order: upper, lower, start, end.

Integrated bevels or caps do not appear as extra objects.

The Revolve object provides a list of NURBS patch objects in the same order as created upon conversion.

RIB Export

Revolve objects will be exported as NURBS patch primitives:

```
RiNuPatch(...);
```

If bevels or caps are present, those follow as potentially trimmed NURBS patch primitives in the following order: upper, lower, start, end.

PV tags are supported.¹

Multiple TC tags are also supported.²

¹ Since 1.20. ² Since 1.24.

4.7.2 Extrude Object



Figure 76: Extrude Object (left: Curve, middle: normal Extrusion, right: Extrusion with Caps)

The Extrude object forms a linear extrusion from a number of planar NURBS curves, see the image above for an example.

The first curve determines the outline and the other curves determine holes in the extrusion object. Holes may be used by objects that form e.g. letters.

The object has the generating NURBS curves as child objects, watches them and adapts to them automatically.

Consequently, the object hierarchy of an Extrude object may look like this:

```
+--Extrude
  |--Outline(NCurve)
  |--[Hole_1(NCurve)
  |   ...
  |--Hole_n(NCurve) ]
```

The Extrude object can generate caps, if the generating curves are closed. Cap generation may fail, if the outer curve has weights and the curve itself leaves the convex hull of the control polygon. Be careful when using curves with weights!

The sharp corners between caps and extrusion may be beveled.

The axis of the extrusion is always the Z-axis. The parameter curves should be defined in the XY-plane. If not, they will be squashed down to this plane. See section [5.3.25 To XY Tool \(page 304\)](#) for information on how to easily achieve curves in the XY-plane.

The dimensions and orders of the extruded surface(s) will be taken from the respective parameter curves as follows: width and order in U direction will be 2, height and order in V direction are taken from the parameter curve.

The following table briefly lists some capabilities of the Extrude object.

Type	Parent of	Material	Converts to/Provides	Point Edit
Extrude	NCurve ⁺	Yes	NPatch ⁺	No*

Table 68: Extrude Object Capabilities

ExtrudeAttr Property

The parameter "Height" determines how big in Z direction the extrusion should be. Note that the height of the bevels will not be taken into account here, if the extrusion height is 1.0 and beveling (upper and lower) is switched on with radius 0.1 the resulting object extends 1.2 units in Z direction.

The Extrude object can automatically generate caps, that are trimmed NURBS patches. Using "StartCap" and "EndCap" you determine whether such caps should be generated, default is off (no caps). Note that this feature does only work properly, if the generating NURBS curves are closed and not self intersecting, this is because the generating curves themselves are used as trim curves for the caps. Warning, Ayam will not check whether the parameter curves conform to these criteria. Ayam, however, automatically detects the correct orientation of the curves (and reverts them if necessary).

Since Ayam 1.10 the bevel parameters of the Extrude object are saved in bevel parameter tags and the property GUI changed to conform to all other bevel supporting tool objects. The old options "LowerBevel", "UpperBevel", "BevelType", and "BevelRadius" are no longer available. They were replaced with new dynamic tag creating bevel property GUI sections that are accessible through the new command entries "Add Start Bevel!" and "Add End Bevel!" respectively. If one of those entries is used, a corresponding bevel parameter tag is created and more options will be made available in the property GUI to adjust the bevel parameters or remove the tag again. A more thorough discussion of those options is available in section 4.7.9 BevelAttr Property (page 209).

See section 4.6.1 NPatchAttr (page 170) for a description of the other two attributes "DisplayMode" and "Tolerance".

Using Holes and Bevels

All curves forming holes in the extruded object must be defined inside (geometrically) the first curve (the outline curve). Additionally, they may not intersect each other or them self and there can not be hole curves inside hole curves. If there are bevels and caps, allow some extra spacing between the curves (for the bevels). Ayam will not check whether the parameter curves conform to these criteria.

With the direction of the curve, the direction of the bevel is determined as well (should it round outwards or inwards?). If the bevels of the holes look wrong try to revert the generating curves of the holes. Note that beveling does not work well with open curves. It is suggested to always use closed curves for beveling. Beveling may lead to self intersecting trim curves in sharp corners of an extrusion. Decrease the bevel radius or round the corners of the extruded curve (using insertion of additional control points) if cap generation fails due to self intersecting bevels.

Another special issue shall be noted: If there are holes, the corresponding bevels will be scaled with the hole curve object transformation values. Thus, to achieve equally sized bevels for outline and holes, possible scale transformations should be carried out on the hole curve control points, rather than on the hole curve object transformation attributes.

Conversion Support

The extruded surface, the bevels, and the caps may be converted to ordinary NURBS patches using the main menu entry "Tools/Convert".

If bevels or caps are present, an enclosing Level object will be created and the bevels and caps follow the extruded surface in the following order: end bevel, end cap, start bevel, start cap.

The Extrude object provides a list of NURBS patch objects in the same order as created upon conversion.

RIB Export

Extrude objects will be exported as NURBS patch primitives:

```
RiNuPatch(...);
```

If caps or bevels are present, those follow as potentially trimmed NURBS patch primitives in the following order: end bevel, end cap, start bevel, start cap.

PV tags are supported but all NURBS patch primitives will get the same set of tags.¹

¹ Since 1.20.

4.7.3 Swing Object

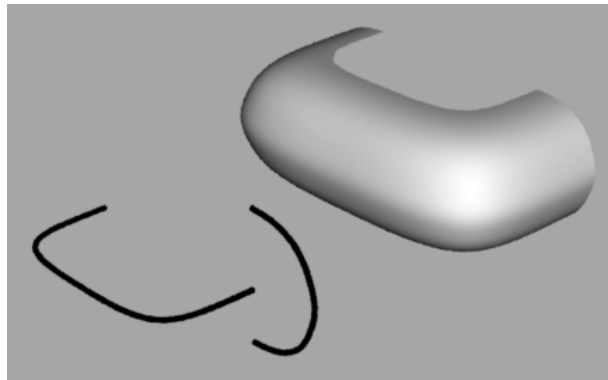


Figure 77: Swing Object (left: Curves, right: Resulting Swung Surface)

The Swing object forms a surface that results from rotating a NURBS curve (called *cross section* or *profile*) around an axis while scaling it according to a second NURBS curve (called *trajectory* or *path*).¹ This process is sometimes also called *rotational sweep*. See the image above for an example.

The Swing object has the generating NURBS curves as child objects and watches their movements and adapts to them automatically. The first curve is the cross section, the second is the trajectory.

The object hierarchy of a Swing object, thus, looks like this:

```
+--Swing
  |--Cross_Section(NCurve)
  \--Trajectory(NCurve)
```

The swing operation will occur around the Y-axis, i.e. the trajectory should be defined in the XZ-plane.

The cross section curve should be defined in the YZ-plane and the trajectory should start here. See also the image below.

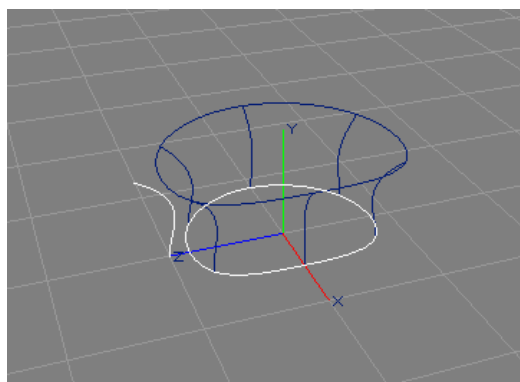


Figure 78: Swing Cross Section (l), Trajectory (r), Surface (blue)

¹ Since 1.14.

Note that the swing create tool will automatically modify the control points of NCurve, ICurve, and ACurve objects so that they are defined in the proper plane. For all other object types, the transformation attributes will be set (Rotate_Y is set to 90 if currently set to 0).¹

The dimensions and orders of the swung surface will be taken from the respective parameter curves as follows: width and order in U direction from the trajectory, height and order in V direction from the cross section.

The following table briefly lists some capabilities of the Swing object.

Type	Parent of	Material	Converts to/Provides	Point Edit
Swing	NCurve*	Yes	NPatch ⁺	No*

Table 69: Swing Object Capabilities

SwingAttr Property

The swung surface is completely controlled by the parameter curves.

See [section 4.6.1 NPatchAttr \(page 170\)](#) for a description of the attributes "DisplayMode" and "Tolerance".

To help in the exact configuration of the swung surface, the "NPInfo" field always displays the parameters of the created NURBS patch.

Caps and Bevels

The Swing object supports the standard caps as lined out in [section 4.10.5 Caps Property \(page 255\)](#) and the standard bevels as lined out in [section 4.10.6 Bevels Property \(page 256\)](#).

The boundary names are:

Start – cross section,

End – cross section at end of sweep,

Upper – curve formed by sweeping the start point of the cross section, and

Lower – curve formed by sweeping the end point of the cross section.

¹ Since 1.24.

Conversion Support

The swung surface and the caps may be converted to ordinary NURBS patches using the main menu entry "Tools/Convert".

If bevels or caps are present, an enclosing Level object will be created and the bevels or caps follow the swung surface in the following order: upper, lower, start, end.

Integrated bevels or caps do not appear as extra objects.

The Swing object provides a list of NURBS patch objects in the same order as created upon conversion.

RIB Export

Swing objects will be exported as NURBS patch primitives:

```
RiNuPatch (...);
```

If bevels or caps are present, those follow as potentially trimmed NURBS patch primitives in the following order: upper, lower, start, end.

PV tags are supported.¹

Multiple TC tags are also supported.²

¹ Since 1.20. ² Since 1.24.

4.7.4 Sweep Object

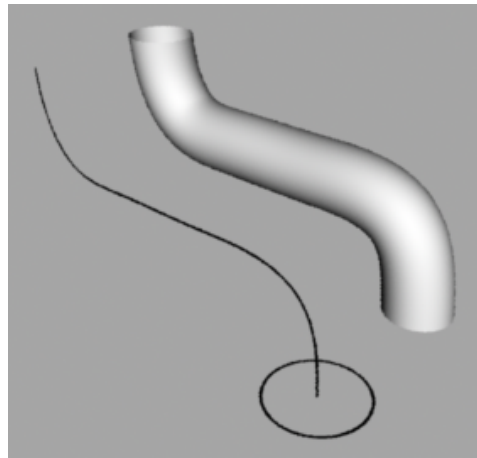


Figure 79: Sweep Object (left: Curves, right: Resulting Swept Surface)

The Sweep object forms a surface that results from moving a NURBS curve (called *cross section* or *profile*) along a second NURBS curve (called *trajectory* or *path*). See the image above for an example.

The cross section may be scaled while sweeping using a third curve, the *scaling function*. Swept surfaces may be closed in the direction of the trajectory and they may also be periodic.¹

The Sweep object has the generating NURBS curves as child objects and watches their movements and adapts to them automatically. The first curve is the cross section, the second is the trajectory, and the third curve represents the scaling function.

The object hierarchy of a Sweep object, thus, looks like this:

```
+--Sweep
  |--Cross_Section(NCurve)
  |--Trajectory(NCurve)
  \--[Scaling_Function(NCurve)]
```

Note that the "Translate" attributes of the cross section curve will be fully ignored. All other transformation attributes (of cross section and trajectory) will be used to determine place, orientation, and size of the Sweep object. The cross section curve has to be defined in the YZ-plane of the Sweep objects coordinate system. See also the image below.

This means that a simple circular curve as e.g. created with the toolbox has to be rotated by 90 degrees around the Y-axis. This can be done either by modifying the control points, or by setting the transformation attributes accordingly. Later editing of this curve has to be done in a Side view. Note that the sweep create tool will automatically modify the control points of NCurve, ICurve, and ACurve objects so that they are defined in the proper plane. For all other object types, the transformation attributes will be set (Rotate_Y is set to 90 if currently set to 0).²

The scaling function is sampled for each section and the Y-component of the coordinates of the current curve point will be used as scale factor that is applied to the cross section in Y-direction.

¹ Since 1.10. ² Since 1.24.

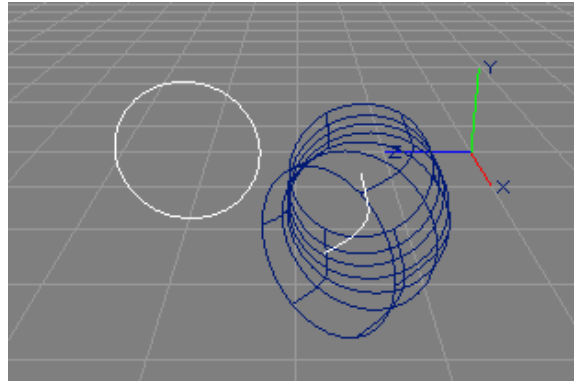


Figure 80: Sweep Cross Section (l), Trajectory (r), Surface (blue)

If any sample point of the scaling function has a Z-component different from zero, the Z-component will be used to independently scale the cross section in X-direction, otherwise the Y-component will be used to also scale the cross section in X-direction.¹

This implies, that e.g. a scaling function that does nothing should be a linear curve from (0,1,1) to (1,1,1). Scale components that are less than or equal to zero will be silently ignored.

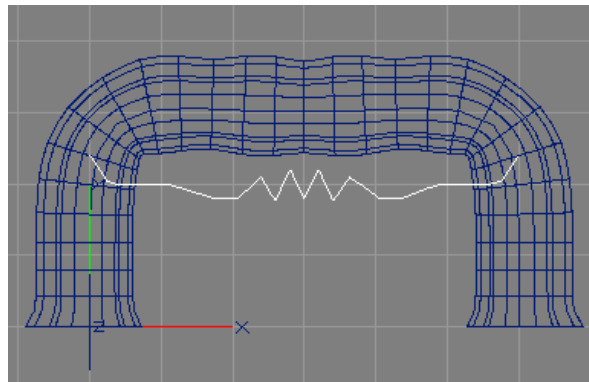


Figure 81: Sweep Object (blue) with Scaling Function (white)

Here is a short example for the creation of a sweep:

1. Create a circular B-Spline curve using the toolbox. (This will be our cross section.)
2. Create a simple NURBS curve using the toolbox. (This will be our trajectory.)
3. Select both curves. (Select the first curve, hold down the "Shift" key and select the other curve.)
4. Create the Sweep object using the toolbox.
5. Observe that the cross section curve was rotated automatically to the YZ-plane.
6. Now you may enter the Sweep object and modify e.g. the second curve, the trajectory. (Press <e>, then drag some control points around.)
7. To modify the cross section you would need to switch to a view of type "Side". (Use the views "Type" menu or the <PgDwn> keyboard shortcut while the view has the input focus.)

Section 6.4.3 Easy Sweep (page 442) has an example script that automates creation and parameterisation of a suitable cross section curve.

¹ Since 1.13.

Translational surfaces are a similar creation algorithm, see also section 6.6.10 Translational Surface (page 469).

The following table briefly lists some capabilities of the Sweep object.

Type	Parent of	Material	Converts to/Provides	Point Edit
Sweep	NCurve*	Yes	NPatch ⁺	No*

Table 70: Sweep Object Capabilities

SweepAttr Property

The "Type" attribute controls whether the swept surface should be open, closed, or periodic in the direction of the trajectory curve.¹

If "Interpolation" is enabled, an additional interpolation will be run on the swept surface in U direction so that all section curves will be interpolated by the swept surface. Instead of a NURBS knot vector, the swept surface will then get a Chordal knot vector (calculated by knot averaging) and the swept surface will follow the trajectory more closely. See the image below for an example.

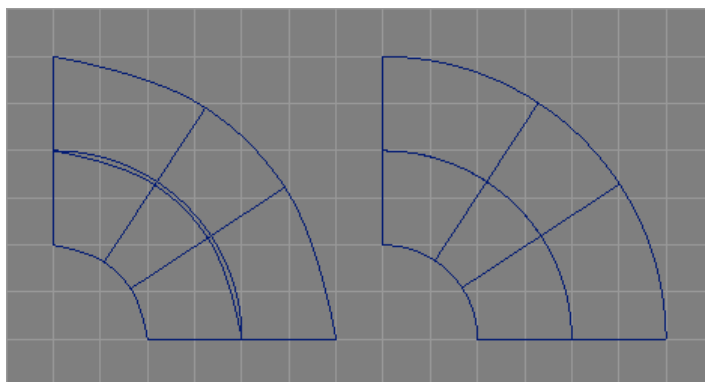


Figure 82: Sweep Along a Quarter Circle without (left) and with (right) Interpolation Enabled

The third parameter, "Sections", determines how many control points (in U direction) should be used, when generating the sweep NURBS patch. The sweep NURBS patch has sections+1 control points in U direction for open and closed sweeps, whereas sections+order control points will be created for periodic sweeps.

Zero is a valid setting for the "Sections" parameter and used as default value.² In this case the number of sections is directly derived from the length of the trajectory curve plus one (except for trajectory curves of length 2, where the number of sections is 1).

For example, if "Sections" is zero, for a standard NURBS curve of length 4, the number of sections used is 5 and the width of the created NURBS patch is 6, for a curve with just 2 control points, the number of sections used is 1 and the width of the resulting patch is 2. See the table below for more examples.

Moreover, if "Sections" is zero, the order of the sweep in U direction is taken from the trajectory curve. Otherwise, the order of the created patch depends on the number of sections as follows: for 1 and 2 sections the order will be 2 and 3 respectively, in all other cases it will be 4.

If "Rotate" is enabled, the cross sections will be rotated so that they are always perpendicular to the trajectory, this option is enabled by default.

¹ Since 1.10. ² Since 1.13.

See section 4.6.1 [NPatchAttr](#) (page 170) for a description of the other two attributes "DisplayMode" and "Tolerance".

To help in the exact configuration of the swept surface, the "NPInfo" field always displays the parameters of the created NURBS patch.

The following table shows some example parameter configurations for the Sweep object.

Sections	Trajectory Length	Trajectory Order	Sweep Length	Sweep Order
0	2	2	2	2
0	5	4	6	4
0	6	5	7	5
4	6	5	5	4
10	6	5	11	4

Table 71: Sweep Parameterisation Examples

Caps and Bevels

The Sweep object supports the standard caps as lined out in section 4.10.5 Caps Property (page 255) and the standard bevels as lined out in section 4.10.6 Bevels Property (page 256).

The boundary names are:

Start – cross section,

End – cross section at end of sweep,

Left – curve formed by sweeping the start point of the cross section, and

Right – curve formed by sweeping the end point of the cross section.

Conversion Support

The swept surface, the bevels and the caps may be converted to ordinary NURBS patches using the main menu entry "Tools/Convert".

If bevels or caps are present, an enclosing Level object will be created and the bevels and caps follow the swept surface in the following order: start bevel, start cap, end bevel, end cap, left bevel, left cap, right bevel, right cap.

Integrated bevels or caps do not appear as extra objects.

The Sweep object provides a list of NURBS patch objects in the same order as created upon conversion.

RIB Export

Sweep objects will be exported as NURBS patch primitives:

```
RiNuPatch(...);
```

If caps or bevels are present, those follow as potentially trimmed NURBS patch primitives in the following order: start bevel, start cap, end bevel, end cap, left bevel, left cap, right bevel, right cap.

PV tags are supported.¹

Multiple TC tags are also supported.²

¹ Since 1.20. ² Since 1.24.

4.7.5 Birail1 Object

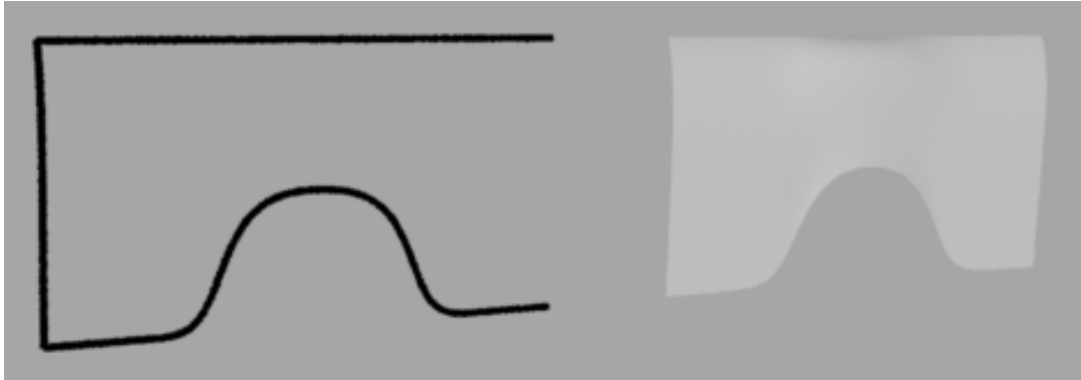


Figure 83: Birail1 Object (left: Curves, right: Resulting Swept Surface)

The Birail1 object forms a surface by sweeping a *cross section* (or *profile*) curve along two so called *rail* curves. See the image above for an example.

The object hierarchy of a Birail1 object, thus, looks like this:

```
+--Birail1
  |--Cross_Section (NCurve)
  |--Rail1 (NCurve)
  \--Rail2 (NCurve)
```

When the cross section touches the rail curves in their respective starting points, the resulting surface will interpolate the rail curves. The direction of the cross section curve will be parallel to the V parametric dimension (height) and the direction of the rail curves will be parallel to the U parametric dimension (width) of the resulting surface. Height and width of the surface will be derived from the length of the cross section curve and the number of sections, respectively.

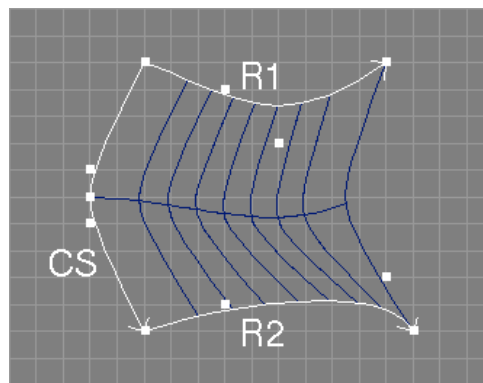


Figure 84: Valid Configuration of Parameter Curves (white) for Birail1 (blue)

The image above shows a valid configuration of parameter curves for the Birail1 object. Mind the direction of the rail curves (R1 and R2) with regard to the cross section curve (CS) and the fact that the cross section curve touches the starting points of the rail curves.

Note that the cross section curve does not have to be two dimensional, and, in contrast to the normal Sweep object, it also does not have to be defined in a special plane. Also note that the precision with which the resulting surface will interpolate the rail curves depends on the number of sections chosen.

The Birail1 object watches the child objects and adapts to them automatically via the notification mechanism.

See also section 6.6.9 DualSweep (page 468) for a script object that creates a similar surface.

The following table briefly lists some capabilities of the Birail1 object.

Type	Parent of	Material	Converts to/Provides	Point Edit
Birail1	NCurve*	Yes	NPatch ⁺	No*

Table 72: Birail1 Object Capabilities

Birail1Attr Property

The following parameters control the birailing process.

Similar to the Sweep object, the "Type" attribute controls whether the birailed surface should be open, closed, or periodic in the direction of the rail curves.

The parameter "Sections" determines how many sections (in U direction) should be used, when generating the birailed NURBS patch. The birailed NURBS patch always has sections+1 control points in U direction.

Also zero is a valid setting for the "Sections" parameter and used as default value.¹

If "Sections" is zero the number of sections is directly derived from the length of the first rail curve plus one (except for curves of length 2, where it is 1). See the table below for examples.

Moreover, if "Sections" is zero, the order of the birail in U direction is taken from the first rail curve. Otherwise, the order of the created patch depends on the number of sections as follows: for 1 and 2 sections the order will be 2 and 3 respectively, in all other cases it will be 4.

See section 4.6.1 NPatchAttr (page 170) for a description of the other two attributes "DisplayMode" and "Tolerance".

To help in the exact configuration of the birailed surface, the "NPInfo" field always displays the parameters of the created NURBS patch.

¹ Since 1.13.

The following table shows some example parameter configurations for the Birail1 object.

Sections	Rail1 Length	Rail1 Order	Birail1 Length	Birail1 Order
0	2	2	2	2
0	5	4	6	4
0	6	5	7	5
4	6	5	5	4
10	6	5	11	4

Table 73: Birail1 Parameterisation Examples

Caps and Bevels

The Birail1 object supports the standard caps as lined out in section 4.10.5 Caps Property (page 255) and the standard bevels as lined out in section 4.10.6 Bevels Property (page 256).

The boundary names are:

Start – cross section,

End – cross section at end of sweep,

Left – curve formed by sweeping the start point of the cross section, and

Right – curve formed by sweeping the end point of the cross section.

Conversion Support

The birailed surface, the bevels, and the caps may be converted to ordinary NURBS patches using the main menu entry "Tools/Convert".

If bevels or caps are present, an enclosing Level object will be created and the bevels and caps follow the birailed surface in the following order: end bevel, end cap, start bevel, start cap, left bevel, left cap, right bevel, right cap.

Integrated bevels or caps do not appear as extra objects.

The Birail1 object provides a list of NURBS patch objects in the same order as created upon conversion.

RIB Export

Birail1 objects will be exported as NURBS patch primitives:

```
RiNuPatch(...);
```

If caps or bevels are present, those follow as potentially trimmed NURBS patch primitives in the following order: end bevel, end cap, start bevel, start cap, left bevel, left cap, right bevel, right cap.

PV tags are supported.¹

Multiple TC tags are also supported.²

¹ Since 1.20. ² Since 1.24.

4.7.6 Birail2 Object

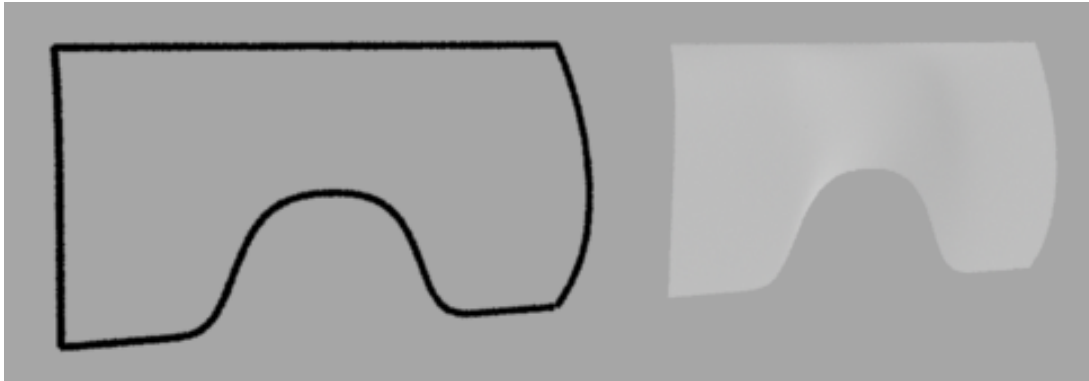


Figure 85: Birail2 Object (left: Curves, right: Resulting Swept Surface)

The Birail2 object forms a surface by sweeping a *cross section* (or *profile*) curve along two so called *rail* curves, while morphing it into a second cross section (or profile) curve. See also the image above for an example.

The morphing process may be controlled by a fifth parameter curve, the *interpolation control* curve. The object hierarchy of a Birail2 object, thus, looks like this:

```
+--Birail2
  |--Cross_Section1 (NCurve)
  |--Rail1 (NCurve)
  |--Rail2 (NCurve)
  |--Cross_Section2 (NCurve)
  \--[Interpolation_Control (NCurve) ]
```

When the cross sections touch the rail curves in their respective starting or end points, the resulting surface will interpolate the rail curves. The direction of the cross section curves will be parallel to the V parametric dimension (height) and the direction of the rail curves will be parallel to the U parametric dimension (width) of the resulting surface. Height and width of the surface will be derived from the length of the cross section curves and the number of sections, respectively.

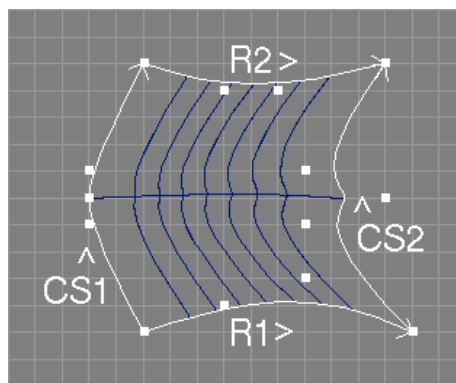


Figure 86: Valid Configuration of Parameter Curves (white) for Birail2 (blue)

The image above shows a valid configuration of parameter curves for the Birail2 object. Mind the direction of the rail curves (R1 and R2) with regard to the two cross section curves (CS1 and CS2) and the fact, that all curves touch at their respective end points.

Note that the cross section curves do not have to be two dimensional, and, in contrast to the normal Sweep object, they also do not have to be defined in a special plane. Furthermore, they do not have to be compatible in terms of length, order, and knots. Incompatible curves will be made compatible before birailing automatically; the height of the resulting surface, however, is not easily predictable anymore in this case. Also note that the precision with which the resulting surface will interpolate the rail curves depends on the number of sections chosen.

If a fifth curve is present as parameter object, this curve will control the morphing (interpolation) process. The y coordinate of this curve at a specific point, which should have a value between 0 and 1, determines the ratio of control of the first cross section (0) and the second cross section (1) over the interpolated curve. Thus, a straight line running from point (0,0) to (1,1) will be equivalent to the standard linear interpolation that would be carried out if no interpolation control curve were present. Note, however, that the interpolation control curve has no influence on the first and last copy of the respective cross section curve, unless the "InterpolCtrl" option is used.¹

The Birail2 object watches the child objects and adapts to them automatically via the notification mechanism.

The following table briefly lists some capabilities of the Birail2 object.

Type	Parent of	Material	Converts to/Provides	Point Edit
Birail2	NCurve*	Yes	NPatch ⁺	No*

Table 74: Birail2 Object Capabilities

Birail2Attr Property

The following parameters control the birailing process.

The parameter "Sections" determines how many sections (in U direction) should be used, when generating the birailed NURBS patch. The birailed NURBS patch always has sections+1 control points in U direction.

Also zero is a valid setting for the "Sections" parameter and used as default value.²

If "Sections" is zero the number of sections is directly derived from the length of the first rail curve plus one (except for curves of length 2, where it is 1). See the table below for examples.

Moreover, if "Sections" is zero, the order of the birail in U direction is taken from the first rail curve. Otherwise, the order of the created patch depends on the number of sections as follows: for 1 and 2 sections the order will be 2 and 3 respectively, in all other cases it will be 4.

The parameter "InterpolCtrl" allows the interpolation controlling curve full influence on the birailed surface. If "InterpolCtrl" is disabled, the first and last border of the resulting surface will always exactly match the parameter curves (CS1 and CS2 respectively), regardless of the interpolation control curve.

See section 4.6.1 NPatchAttr (page 170) for a description of the other two attributes "DisplayMode" and "Tolerance".

¹ Since 1.10. ² Since 1.13.

To help in the exact configuration of the birailed surface, the "NPInfo" field always displays the parameters of the created NURBS patch.

The following table shows some example parameter configurations for the Birail2 object.

Sections	Rail1 Length	Rail1 Order	Birail2 Length	Birail2 Order
0	2	2	2	2
0	5	4	6	4
0	6	5	7	5
4	6	5	5	4
10	6	5	11	4

Table 75: Birail2 Parameterisation Examples

Caps and Bevels

The Birail2 object supports the standard caps as lined out in section 4.10.5 Caps Property (page 255) and the standard bevels as lined out in section 4.10.6 Bevels Property (page 256).

The boundary names are:

Start – first cross section,

End – second cross section,

Left – curve formed by sweeping the start point of the first cross section, and

Right – curve formed by sweeping the end point of the first cross section.

Conversion Support

The birailed surface, the bevels, and the caps may be converted to ordinary NURBS patches using the main menu entry "Tools/Convert".

If bevels or caps are present, an enclosing Level object will be created and the bevels and caps follow the birailed surface in the following order: end bevel, end cap, start bevel, start cap, left bevel, left cap, right bevel, right cap.

Integrated bevels or caps do not appear as extra objects.

The Birail2 object provides a list of NURBS patch objects in the same order as created upon conversion.

RIB Export

Birail2 objects will be exported as NURBS patch primitives:

```
RiNuPatch(...);
```

If caps or bevels are present, those follow as potentially trimmed NURBS patch primitives in the following order: end bevel, end cap, start bevel, start cap, left bevel, left cap, right bevel, right cap.

PV tags are supported.¹

Multiple TC tags are also supported.²

¹ Since 1.20. ² Since 1.24.

4.7.7 Skin Object

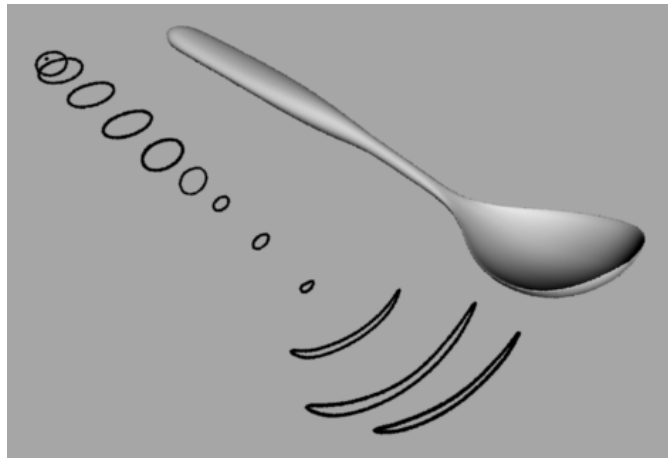


Figure 87: Skin Object (left: Curves, right: Resulting Skinned Surface)

The Skin object forms a surface defined by a set of *cross section* curves, where the first and last curve will always be interpolated by the surface (this process is sometimes also called *lofting*). See also the image above.

When only two parameter curves are used, the skin forms a so called *ruled surface*.

The complete template for the Skin object hierarchy, consequently, looks like this:

```
+--Skin
  | -Curve_1 (NCurve)
  | -Curve_2 (NCurve)
  | -[ ...
  \ -Curve_n (NCurve) ]
```

Note that in contrast to the build from curves tool, the curves may be of arbitrary length and order. It is e.g. possible to use a parameter curve of order 2 and length 6 with a second curve of order 4 and length 4 and a third curve with order 3 and 5 control points. If the curves are of different length or order, they will all be converted internally until they are compatible. Be warned, that this process may consume a considerable amount of time because all unclamped curves have to be converted to clamped ones; then, for every curve with low order degree elevation has to be done; then a uniform knot vector has to be found; then all curves have to be refined using this new knot vector; interpolation adds another dimension of complexity. If you experience lags when editing the child curves of a Skin object try to switch to lazy notification.

A Skin object will also use all the curves of a tool object, that provides multiple curves, e.g. a Clone object in mirror mode.¹

The direction of the parameter curves will be parallel to the V dimension (height) of the skinned surface. The number of the parameter curves will define the U dimension (width) of the skinned surface.

Also note that the resulting patch may be quite complex, even though the curves are not, if the orders or knot vectors of the curves do not match. For example, a skinned patch from two curves of length 4 but one with order 4 and the other with order 2 will result in a patch with a width of 2 and a height of 10.

¹ Since 1.9.

The Skin object has the generating NURBS curves as child objects and watches their changes and adapts to them automatically.

The following table briefly lists some capabilities of the Skin object.

Type	Parent of	Material	Converts to/Provides	Point Edit
Skin	NCurve*	Yes	NPatch ⁺	No*

Table 76: Skin Object Capabilities

SkinAttr Property

The following parameters control the skinning process.

The first parameter "Interpolation" controls whether the inner curves should also be interpolated by the skinning surface.

The second parameter "Order_U" determines the order of the resulting surface in U direction (the order in V direction is determined by the curves). The order may not be higher than the number of curves used. If the specified value is higher than the number of curves, the order of the generated surface will be silently set to the number of curves. If "Order_U" is 0, a default value equal to the number of parameter curves but not higher than 4 will be used.

Using the next parameter "Knot-Type_U", the type of the knot vector that should be used in the U direction of the skinned surface can be adapted. If the knot type is "Bezier" and the specified order (see above) does not exactly match the number of skinned curves, then the order will be silently adapted to the number of skinned curves. The knot type "Custom" creates a chord length parameterisation.¹

If interpolation is enabled, the knot types "Chordal", "Centripetal", and "Uniform" will be used in the interpolation, all other types will lead to a chordal parameterisation of the interpolation.²

See section 4.6.1 [NPatchAttr \(page 170\)](#) for a description of the other two attributes "DisplayMode" and "Tolerance".

To help in the exact configuration of the skinned surface, the "NPInfo" field always displays the parameters of the created NURBS patch.

Caps and Bevels

The Skin object supports the standard caps as lined out in section 4.10.5 [Caps Property \(page 255\)](#) and the standard bevels as lined out in section 4.10.6 [Bevels Property \(page 256\)](#).

The boundary names are:

Start – first parameter curve,

End – last parameter curve,

Left – curve through start points of parameter curves, and

Right – curve through end points of parameter curves.

¹ Since 1.7. ² Since 1.30.

Conversion Support

The skinned surface, the bevels, and the caps may be converted to ordinary NURBS patches using the main menu entry "Tools/Convert".

If bevels or caps are present, an enclosing Level object will be created and the bevels and caps follow the skinned surface in the following order: start bevel, start cap, end bevel, end cap, left bevel, left cap, right bevel, right cap.

Integrated bevels or caps do not appear as extra objects.

The Skin object provides a list of NURBS patch objects in the same order as created upon conversion.

RIB Export

Skin objects will be exported as NURBS patch primitives:

```
RiNuPatch(...);
```

If caps or bevels are present, those follow as potentially trimmed NURBS patch primitives in the following order: start bevel, start cap, end bevel, end cap, left bevel, left cap, right bevel, right cap.

PV tags are supported.¹

Multiple TC tags are also supported.²

¹ Since 1.20. ² Since 1.24.

4.7.8 Gordon Object

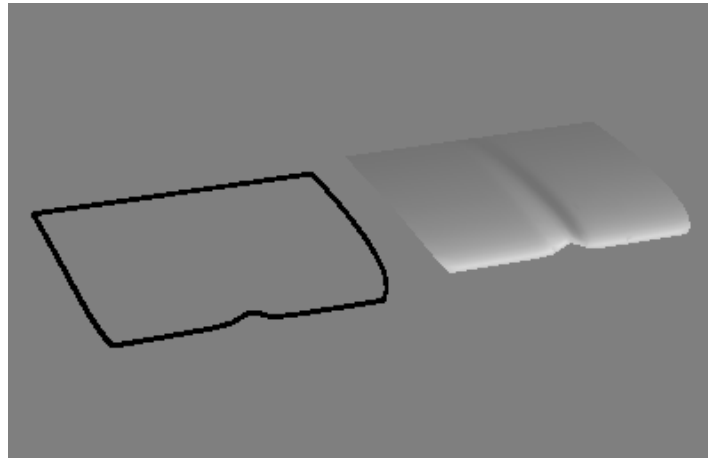


Figure 88: Gordon Object (left: Curves, right: Resulting Gordon Surface)

The Gordon object forms a surface defined by two sets of intersecting curves (a network of curves), where all curves will always be interpolated by the surface (see image above). The image below shows the simplest configuration of such a network, consisting of four parameter curves. Note the arrangement and the direction of the curves. Also note that this configuration is in fact equivalent to a *Coons patch*.

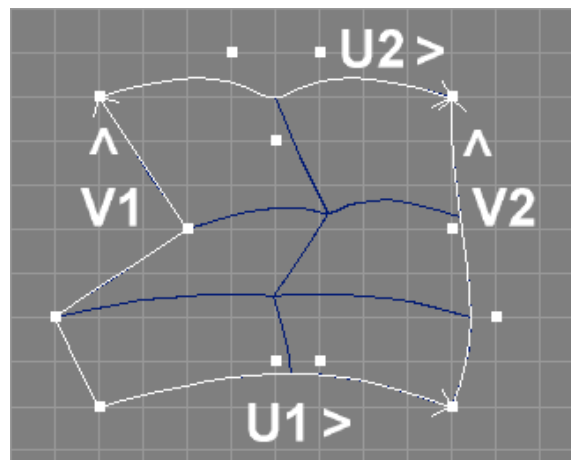


Figure 89: Gordon Surface with Parameter Curves (white)

The curves may be of arbitrary length and order. It is e.g. possible to use a first curve of order 2 and length 6 with a second curve of order 4 and length 4 and a third curve with order 3 and 5 control points for the U parametric dimension.

Note that all intersection points of a curve with other curves have to have the same parametric value in each of the other curves in the same set/direction. All curves are so called *isoparametric* curves.

Also note, that in the general case only non-rational curves can be used as parameter curves for a Gordon surface. If the parameter curves are rational, the weight information of the curves will simply be ignored. However, since Ayam 1.13 there is a special case allowed: if exactly four parameter curves are present, their weight information will be used properly. Mind that for a correct surface interpolation the curves weights have to match in the respective end points.

The Gordon object has the generating NURBS curves as child objects and watches their changes and adapts to them automatically. Separation of the two sets of curves has to be done using an empty Level object. The first set of curves determines the U direction and the second set of curves the V direction of the Gordon surface. For the example surface in the image above, the child objects of the Gordon object would have to look like this in the Ayam object tree view:

```
+--Gordon
  |--U1 (NCurve)
  |--U2 (NCurve)
  |--Level
  |--V1 (NCurve)
  \--V2 (NCurve)
```

The creation of a Gordon surface is computationally expensive. It involves (interpolated) skinning of the two sets of parameter curves, finding the intersection points of the two sets of parameter curves, interpolating the matrix of intersection points, making the three resulting surfaces compatible, and finally combining the three surfaces into the resulting Gordon surface. If there are lags while editing the parameter curves of a Gordon surface, consider switching to lazy notification.

In order to ease the computationally intensive intersection detection for Ayam an additional parameter object, separated from the two sets of parameter curves by a second empty Level object, may be specified. This parameter object should be a NURBS patch object that describes all intersection points by its control points. If this object is present, no intersection points will be calculated internally and the points specified via this object will be used instead. A "NoExport" tag should be added to this patch, to prevent it from appearing in RIB output.

The object hierarchy of a Gordon object using such a patch may look like this:

```
+--Gordon
  |--U1 (NCurve)
  |--U2 (NCurve)
  |--Level
  |--V1 (NCurve)
  |--V2 (NCurve)
  |--Level
  \--Intersections (NPatch)
```

The complete template for the Gordon object hierarchy, consequently, is as follows:

```
+--Gordon
  |--U1 (NCurve)
  |--U2 (NCurve)
  |--[ ...
  |--Un (NCurve) ]
  |--Level
  |--V1 (NCurve)
  |--V2 (NCurve)
  |--[ ...
  |--Vn (NCurve) ]
  |--[Level
  \--Intersections (NPatch) ]
```

The Gordon object watches the child objects and adapts to them automatically via the notification mechanism.

The following table briefly lists some capabilities of the Gordon object.

Type	Parent of	Material	Converts to/Provides	Point Edit
Gordon	NCurve*/Level/NPatch	Yes	NPatch ⁺	No*

Table 77: Gordon Object Capabilities

GordonAttr Property

The following parameters of the Gordon object control the creation of the Gordon surface.

If the parameter "WatchCorners" is switched on, Ayam will check for all four outer parameter curves, whether they touch in their endpoints. If not, the endpoints will be corrected. If Ayam can determine which curve was modified last, the other curve that should meet at the endpoint in question will be modified. If Ayam finds no information on modifications, the U curves take precedence (i.e. the V curves will be modified). Note that this works only properly with clamped curves. Furthermore, only NCurve, ICurve, or ACurve objects will be modified, but end point data will be derived from any objects providing a NCurve (e.g. ExtrNC).¹

The parameters "Order_U" and "Order_V" determine the desired order of the resulting surface in U and V direction. However, depending on the number and configuration of curves used in the U or V direction, it may not be possible to create a Gordon surface of the desired order. If "Order_U" or "Order_V" are 0, a default value of 4 will be used.

See section 4.6.1 NPatchAttr (page 170) for a description of the other two attributes "DisplayMode" and "Tolerance".

To help in the exact configuration of the Gordon surface, the "NPInfo" field always displays the parameters of the created NURBS patch.

Caps and Bevels

The Gordon object supports the standard caps as lined out in section 4.10.5 Caps Property (page 255) and the standard bevels as lined out in section 4.10.6 Bevels Property (page 256).

The boundary names are:

U0 – first curve of first set,

U1 – last curve of first set,

V0 – first curve of second set, and

V1 – last curve of second set.

Conversion Support

The Gordon surface, the bevels, and the caps may be converted to ordinary NURBS patches using the main menu entry "Tools/Convert".

¹ Since 1.21.

If bevels or caps are present, an enclosing Level object will be created and the bevels and caps follow the Gordon surface in the following order: u0 bevel, u0 cap, u1 bevel, u1 cap, v0 bevel, v0 cap, v1 bevel, v1 cap.

Integrated bevels or caps do not appear as extra objects.

The Gordon object provides a list of NURBS patch objects in the same order as created upon conversion.

RIB Export

Gordon objects will be exported as NURBS patch primitives:

```
RiNuPatch(...);
```

If caps or bevels are present, those follow as potentially trimmed NURBS patch primitives in the following order: u0 bevel, u0 cap, u1 bevel, u1 cap, v0 bevel, v0 cap, v1 bevel, v1 cap.¹

PV tags are supported.²

Multiple TC tags are also supported.³

Triangular Gordon Surfaces

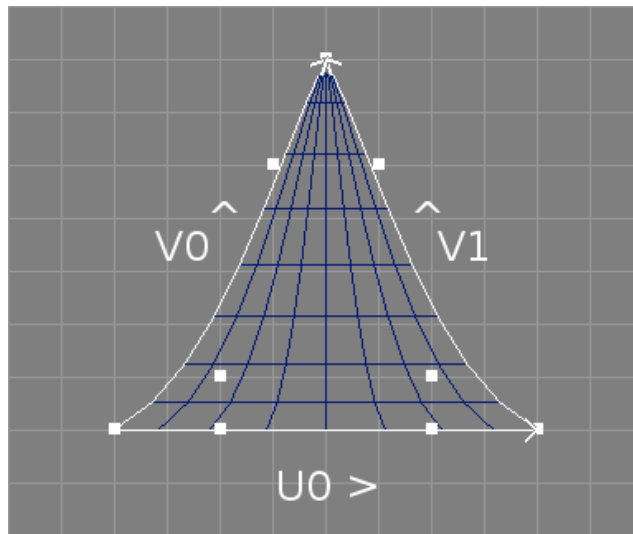


Figure 90: Triangular Gordon Surface (blue) From Three Curves (white)

Apart from the Coons patch mode, Gordon objects also support another special mode where a triangular surface is formed from three parameter curves.⁴ See also the image above. In this mode the three parameter curves do not have to be separated by a Level object. The created NURBS surface is degenerate at $v = 1.0$, however, a special tessellation is used for shading these surfaces so that no artefacts should be visible.

This tessellation can also be accessed using the scripting interface command "convOb PolyMesh".

As they are a special case of Coons patches, triangular Gordon surfaces also support rational coordinates.

¹ Since 1.21. ² Since 1.20. ³ Since 1.24. ⁴ Since 1.30.

4.7.9 Bevel Object

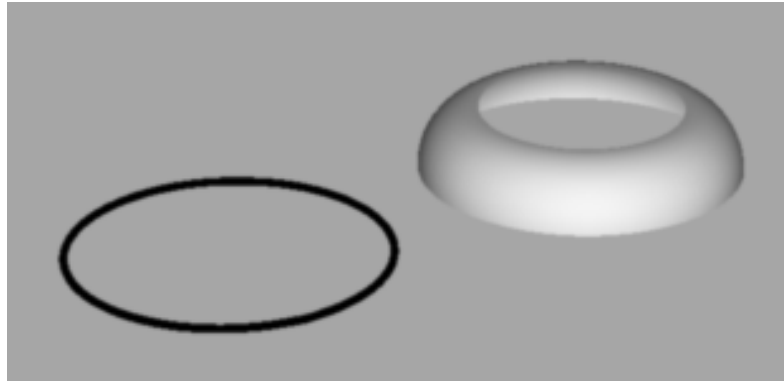


Figure 91: Bevel Object (left: Curve, right: Resulting Bevelled Surface)

The Bevel object forms a bevelled surface from a single parameter curve. See also the image above. The bevel cross section shape may be defined by a second curve. Consequently, the template for the object hierarchy of a Bevel object looks like this:

```
+-Bevel
  |-NCurve
  \-[NCurve]
```

Bevels are also available as properties of different tool objects (e.g. Extrude or Sweep). In fact, Bevel objects use the same creation algorithm as bevel properties but offer increased flexibility in terms of e.g. material settings. Surfaces created from bevel properties always share the material settings of the progenitor tool object. In contrast, Bevel objects may have their own material settings. Bevel objects are available in Ayam since version 1.10.

Note that the parameter curve of a Bevel object should be closed and planar to achieve best results; see section 5.3.25 [To XY Tool \(page 304\)](#) for information on how to easily achieve this. If the curve is closed or periodic, the appropriate curve type should be set in the curve object, otherwise the bevelled surface may expose defects.

Since Ayam 1.19 the Bevel object supports a second parameter curve that defines the bevels cross section shape. It should be defined in the XY-plane and run from (0, 0) to (1, 1). If this curve is present, the Bevel type parameter is ignored as the shape of the bevel is already completely defined. Note that even though the curve should end at (1, 1), this is not mandatory and therefore allows for bevels of differing width and height to be created.

The Bevel object watches the child object and adapts to it automatically via the notification mechanism.

The following table briefly lists some capabilities of the Bevel object.

Type	Parent of	Material	Converts to/Provides	Point Edit
Bevel	NCurve ⁺	Yes	NPatch	No*

Table 78: Bevel Object Capabilities

BevelAttr Property

The following parameters of the Bevel object control the creation of the bevelled surface:

- "BevelType" determines the shape of the bevel by choosing from a set of cross section curves:
 - "Round" a quarter circle,
 - "Linear" a straight bevel,
 - "Ridge" a more complex ridged surface,
 - "RoundToCap" a surface that starts in the direction of a progenitor surface tangent and rounds off to a cap surface (tangents must be present as tag on the parameter curve), this type provides a smoother transition from the progenitor surface to the bevel than a round bevel, see also the image below,
 - "RoundToNormal" a surface that starts in the direction of a progenitor surface tangent and rounds off to a mean normal, see also the image below, normals and tangents must be present as tag on the parameter curve, the mean normal can also be provided by a MN tag, see section 4.11.20 MN (Mean Normal) Tag (page 270),
 - Any curves defined in a global level named "Bevels" also appear as potential bevel type. Those curves should follow the rules laid out above for the second parameter curve of the Bevel object.

See also the following image for a comparison of some bevel types.

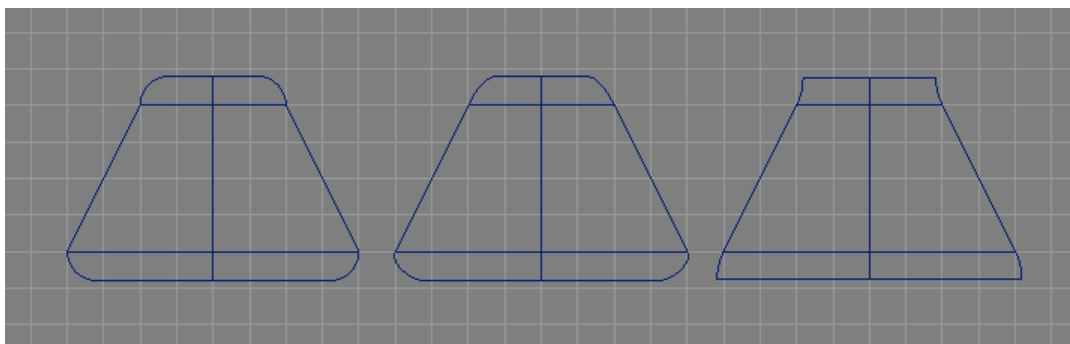


Figure 92: Bevel Types (l: Round, m: RoundToCap, r: RoundToNormal)

Note that the bevel types "RoundCapped" and "LinearCapped" are no longer available¹, use the "Caps" property to create caps.

- "Radius" controls the size and direction of the bevelled surface when seen from the top of the parameter curve. Note that the size of the bevel is expressed in units defined by the object coordinate system of the controlling object. Scale values of the controlling object also affect the bevel size. Negative values are allowed and reverse the surface.
- "Revert" allows to revert the sense of the bevelled surface, should it round inwards or outwards? The sense may also be controlled using the direction of the parameter curve and, additionally, the sense in a different dimension may also be affected by using negative values for the bevel radius.
- "Force3D" switches to 3D mode for all bevel types; if this is enabled, non-planar parameter curves are supported, but normals and tangents must be present as PV tag on the parameter curve. However, when bevels are created via the Bevels property, the normals and tangents will be extracted from the progenitor surface automatically and no PV tags need to be created.

¹ Since 1.21.

To create the tag with the normal and tangent information, the "Extract" option of the ExtrNC object can be used. Otherwise PV tags of appropriate name, storage class, and data type must be created manually or with the help of the scripting interface. Consider the following example script:

```
crtOb NCurve -length 4 -order 2 -cv {0 0 0 1 1 0 0 1 0 1 0 1 0 0 0 1}
uS;sL
addTag PV "N,varying,n,4, -1,0,0, 1,0,0, 0,1,0, -1,0,0"
addTag PV "T,varying,n,4, 0,0,1, 0,0,1, 0,0,1, 0,0,1"
```

Also note that the planarity and exact shape defined by rational coordinate values (e.g. circular arcs) will not be preserved by the bevel types that support non-planar curves with one exception: RoundToCap bevels on fully planar curves/boundaries do preserve the boundary shape fully (unless overridden by "Force3D").¹

See section 4.6.1 NPatchAttr (page 170) for a description of the two attributes "DisplayMode" and "Tolerance" of the "BevelAttr" property.

To help in the exact configuration of the bevel surface, the "NPInfo" field always displays the parameters of the created NURBS patch.

Caps

The Bevel object supports the standard caps as lined out in section 4.10.5 Caps Property (page 255).

The boundary names are:

Start – start of bevel cross section,

End – end of bevel cross section.

Conversion Support

The bevelled surface may be converted to an ordinary NURBS patch using the main menu entry "Tools/Convert".

If caps are present, an enclosing Level object will be created and the caps follow the bevel surface in the following order: start cap, end cap.

RIB Export

Bevel objects will be exported as NURBS patch primitives:

```
RiNuPatch(...);
```

If caps are present, those follow as potentially trimmed NURBS patch primitives in the following order: start cap, end cap.

PV tags are supported.²

Multiple TC tags are also supported.³

¹ Since 1.23. ² Since 1.20. ³ Since 1.24.

4.7.10 Cap Object

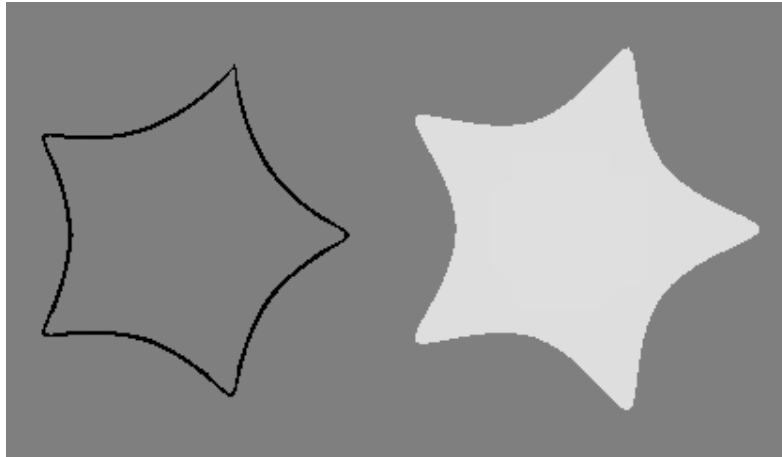


Figure 93: Cap Object (left: Curve, right: Resulting Cap Surface)

The Cap object forms a surface that fills a closed NURBS curve. See also the image above.

Four different types of cap surfaces with different characteristics and parameter curve requirements are supported: *Trim*, *Gordon*, *Simple*, and *Simple3D*.¹

The **Trim** cap type requires planar parameter curves but they may be concave. Furthermore, if multiple curves are present as child objects, the curves following the first curve define holes in the cap surface, similar to the parameter curves of an extruded surface (see also section 4.7.2 [Using Holes and Bevels](#) (page 185)).

Consequently, the template for the object hierarchy of a Cap object in Trim mode looks like this:

```
+--Cap
  |--Outline(NCurve)
  |--[Hole1(NCurve)]
  +-[Hole2(Level)
    |--Part1(NCurve)
    \--Part2(NCurve)]
```

Note that the cap generation may fail, if the control points of the first curve have weights and the curve leaves the convex hull of the control polygon.

The **Gordon** cap type supports only a single parameter curve but this curve may be non planar. Internally the Cap object will split the parameter curve into four sections and build a Gordon surface from the four sections (see the following image for an example).

The **Simple** cap type just extends the parameter curve linearly to a middle point, not supporting non planar curves well and not supporting concave curves at all but ensuring compatibility with the progenitor curve/surface (which may be important for tessellation or further surface processing).

The **Simple3D** cap type extends the parameter curve to a middle point via an additional planar and circular ring of control points, therefore rounding more smoothly to the middle (especially useful for non planar parameter curves or parameter curves with discontinuities), and retains all other characteristics of the Simple type.

¹ Since 1.21.

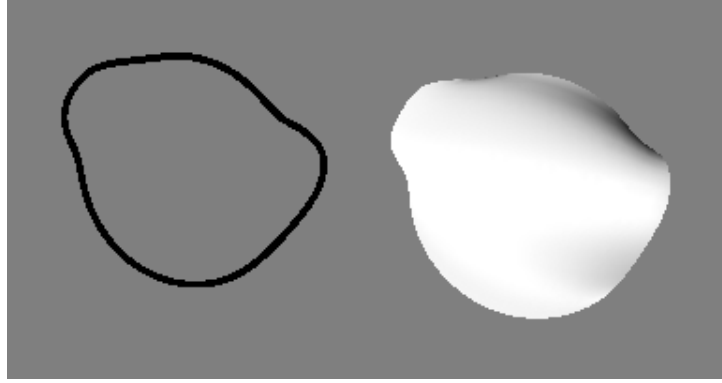


Figure 94: Cap from Non-Planar Curve (left: Curve, right: Resulting Cap Surface)

A MP tag can be set to the Cap object to control the middle point in both simple modes (see section 4.11.21 MP (Mean Point) Tag (page 270)).

See also the following table for an overview on the available cap types:

Type	Planar	Concave	Holes	Compatibility / Integration
Trim	Yes	Yes	Yes	No
Gordon	No	No	No	No
Simple	Yes	No	No	Yes
Simple3D	No	No	No	Yes

Table 79: Cap Types Overview

The Cap object watches the child objects and adapts to them automatically via the notification mechanism.

The following table briefly lists some capabilities of the Cap object.

Type	Parent of	Material	Converts to / Provides	Point Edit
Cap	NCurve ⁺	Yes	NPatch	No*

Table 80: Cap Object Capabilities

CapAttr Property

The following parameters control the cap creation process.

The attribute "Type" allows to select one of the following cap creation methods:

- "Trim": a trimmed NURBS surface,
- "Gordon": an untrimmed Gordon surface,
- "Simple": a simple cap that extends linearly to a middle point,
- "Simple3D": a simple cap with an additional ring of control points that therefore rounds more smoothly to the middle especially for non planar parameter curves.

See also the general discussion about the Cap object above.

The "Fraction" parameter allows to adjust the placement of the additional control point ring in Simple3D mode.

See section 4.6.1 [NPatchAttr](#) (page 170) for a description of the two attributes "DisplayMode" and "Tolerance" of the "CapAttr" property.

To help in the exact configuration of the cap surface, the "NPInfo" field always displays the parameters of the created NURBS patch.

Conversion Support

The cap surface may be converted to an ordinary NURBS patch using the main menu entry "Tools/Convert".

RIB Export

Cap objects will be exported as NURBS patch primitives:

```
RiNuPatch(...);
```

PV tags are supported.¹

¹ Since 1.20.

4.7.11 ConcatNP (Concatenate NURBS Patches) Object

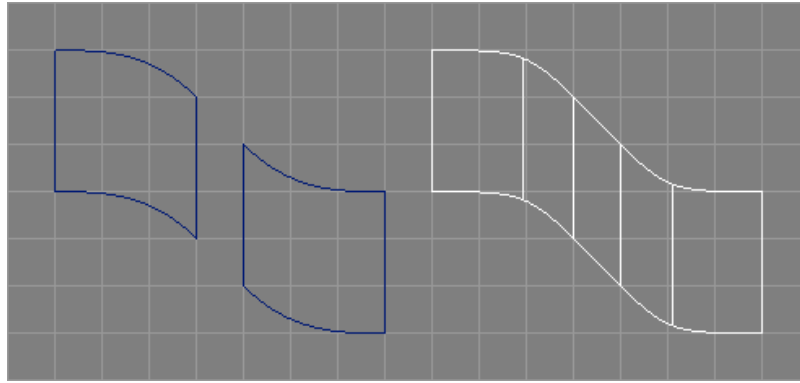


Figure 95: ConcatNP Object (white) From Two NURBS Patches (blue)

The ConcatNP object concatenates all child objects, which should be NURBS patches or provide NURBS patches to a single NURBS patch (see also the image above).¹

The concatenation simply breaks all surfaces into curves, makes the curves compatible, and joins them to the concatenated surface.

Also NURBS curves or objects that provide NURBS curves can be used as parameter objects.²

Eventually present trim curves will be copied and transformed to the appropriate place and orientation in the concatenated surface, according to the new knot domain.³

Attributes like display mode and tolerance for the new concatenated patch are simply taken from the first parameter patch.

Since the ConcatNP object also provides a NURBS patch, it is possible to use it as child object for another ConcatNP object (with possibly different parameters). This way, a hierarchy of ConcatNP objects can be used to emulate patch based modelling to certain extents.

For best results, only clamped surfaces should be used as parameter objects.

The following table briefly lists some capabilities of the ConcatNP object.

Type	Parent of	Material	Converts to/Provides	Point Edit
ConcatNP	NPatch*/NCurve*	Yes	NPatch ⁺	No*

Table 81: ConcatNP Object Capabilities

ConcatNPAttr Property

The following parameters control the concatenation process.

- Using "Type", open, closed, or periodic concatenated patches may be created. If a closed surface is to be created and also "FillGaps" (see below) is enabled, an additional fillet will be created for the last and the first child surface to close the concatenated surface.
- "Order" is the desired order of the concatenated surface (in U direction), the default value (0) leads to a cubic surface. If the desired order is 1, the respective order from the first of the parameter surfaces

¹ Since 1.16. ² Since 1.20. ³ Since 1.21.

is taken. If the desired order is higher than the number of curves (i.e. the total number of control points of all surfaces in their desired directions plus the number of eventually present parameter curves), it will be lowered to the number of curves silently.

- "FillGaps" creates fillet surfaces for all gaps between the parameter surfaces of the ConcatNP object. No fillet will be created if the end curves of two parameter surfaces match or if parameter curves are present between the parameter surfaces in question.

Similar to the fillets for concatenated curves, the fillet surface will be constructed from four control points (in U direction). However, the tangent vectors will not be calculated directly, but instead derived from the respective control points.

- "FTLength" determines the distance of the inner fillet control points from their respective end points. This value can be adapted for smaller/larger gaps between parameter surfaces. If this parameter is negative, the distance between the two surfaces in the respective border points will be multiplied in so that a more pleasing fillet shape results in configurations where the distances between the respective border points vary a lot (see also the image below).

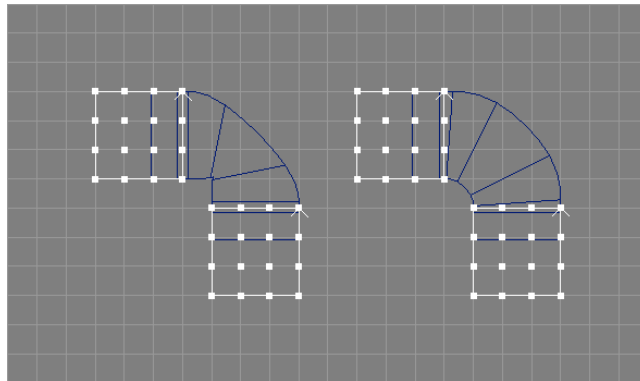


Figure 96: Concatenated Surfaces (blue) with Fillets (left: FTLength 0.3, right: FTLength -0.3)

- If "Revert" is enabled, the orientation of the concatenated surface will be reversed (in U direction).
- The "Knot-Type" parameter allows to choose a knot vector for the concatenated surface in U direction. Similar to the ConcatNC object, only "Custom" knots allow to preserve the shapes of the parameter surfaces completely, but this comes at the price of multiple internal knots (see also 4.5.1 [ConcatNCAttr Property \(page 163\)](#)). In addition, for "Custom" knots, all parameter surfaces will be elevated to a common maximum order or at least be clamped in the respective direction prior to the splitting to curves but after the fillet creation. Furthermore trim curves of the progenitor surfaces will only be copied to the resulting surface, if the "Knot-Type" parameter is "Custom".
- The "UVSelect" option is a string that can be used to control the splitting direction for each parameter surface individually. Valid characters in this string are "u", "U", "v", and "V". The uppercase variants lead to a reverted surface in the respective direction. To connect two surfaces that share the same orientation "over a corner" "UVSelect" should be set to "uV" (see also the image below). The default value for "UVSelect", an empty string, is equivalent to "u" for all patches. Also incomplete strings will lead to "u" for all remaining patches. There is no need to specify a value for fillets, those will always be created in a way so that they can be split along the U direction.
- "Compatible" controls whether the curves should be made compatible before the concatenated surface is built from them. If this option is turned off (the default), the curves are made compatible. Turn this option on if the surfaces to be concatenated are already compatible and their knot vectors are unclamped. The output surface will then also be unclamped (in V direction).

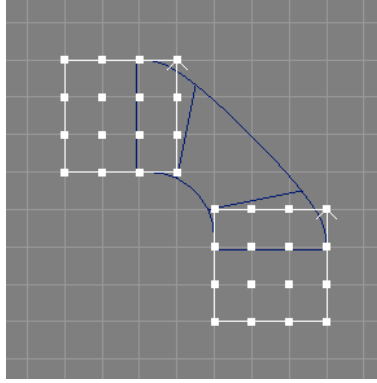


Figure 97: Concatenating two Surfaces with UVSelect == "uV"

- Finally, a "NPInfo" field informs about the actual configuration of the created NURBS patch.

Caps and Bevels

The ConcatNP object supports the standard caps as lined out in section 4.10.5 Caps Property (page 255) and the standard bevels as lined out in section 4.10.6 Bevels Property (page 256).

The boundary names are U0, U1, V0, and V1.

Conversion Support

The concatenated surface may be converted to an ordinary NURBS patch using the main menu entry "Tools/Convert".

If bevels or caps are present, an enclosing Level object will be created and the bevels and caps follow the concatenated surface in the following order: U0 bevel, U0 cap, U1 bevel, U1 cap, V0 bevel, V0 cap, V1 bevel, V1 cap.

Integrated bevels or caps do not appear as extra objects.

The ConcatNP object provides a list of NURBS patch objects in the same order as created upon conversion.

RIB Export

ConcatNP objects will be exported as NURBS patch primitives:

```
RiNuPatch(...);
```

If caps or bevels are present, those follow as potentially trimmed NURBS patch primitives in the following order: U0 bevel, U0 cap, U1 bevel, U1 cap, V0 bevel, V0 cap, V1 bevel, V1 cap.¹

PV tags are supported but all NURBS patch primitives will get the same set of tags.²

Multiple TC tags are also supported.³

¹ Since 1.21. ² Since 1.20. ³ Since 1.24.

4.7.12 ExtrNP (Extract NURBS Patch) Object

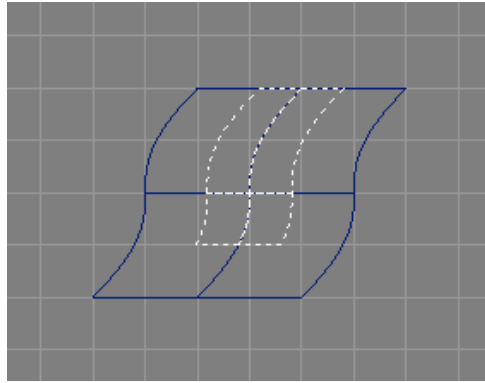


Figure 98: Extracted Surface (white) from Progenitor Surface (blue)

The ExtrNP object extracts a NURBS patch from another NURBS patch object, for use as parameter object for other tool objects (see image above).¹

It also works with NURBS patch providing objects, so that the following example hierarchy is valid:

```
--NPatch
+-ExtrNP
  \ Instance_of_NPatch(Instance)
```

Note that using an instance object of some other surface object (as shown in the above example) is in fact the recommended way of using the ExtrNP object. Therefore, the main menu entry "Tools/Create/ExtrNP" will automatically create an instance of the currently selected object and move it to the newly created ExtrNP object.

As the geometry of the extracted surface is completely defined by the master surface, ExtrNP objects do not support own transformation attributes.² However, if a NP tag makes the Transformation property available again, the transformation attributes will be employed as usual.³

Also note that eventually present trim curves will *not* be honored properly.

The following table briefly lists some capabilities of the ExtrNP object.

Type	Parent of	Material	Converts to/Provides	Point Edit
ExtrNP	NPatch	Yes	NPatch ⁺	No*

Table 82: ExtrNP Object Capabilities

ExtrNPAttr Property

The extraction process is controlled by the following attributes:

- "UMin", "UMax", "VMin", and "VMax" are parametric values that control which part of the original surface is to be extracted. The valid range of parameter values depends on the knot vectors of the original surface.

¹ Since 1.14. ² Since 1.19. ³ Since 1.21.

- "Relative" controls whether the parametric values should be interpreted in a relative way.¹ If enabled, a parametric value of 0.5 always extracts from the middle of the knot vector, regardless of the actual knot values, and the valid range for the parametric values is then consequently 0.0-1.0.
- "PatchNum" allows to select a patch from a list of patches delivered e.g. by a beveled Extrude object as child of the ExtrNP object. This way it is possible to extract a patch from a bevel or cap surface of e.g. a Revolve object.
- See section 4.6.1 NPatchAttr (page 170) for a description of the other two attributes "DisplayMode" and "Tolerance".
- Finally, a "NPInfo" field informs about the actual configuration of the extracted NURBS surface.

Caps and Bevels

The ExtrNP object supports the standard caps as lined out in section 4.10.5 Caps Property (page 255) and the standard bevels as lined out in section 4.10.6 Bevels Property (page 256).

The boundary names are U0, U1, V0, and V1.

Conversion Support

The extracted surface may be converted to an ordinary NURBS patch using the main menu entry "Tools/Convert".

If bevels or caps are present, an enclosing Level object will be created and the bevels or caps follow the extracted surface in the following order: U0, U1, V0, V1.

Integrated bevels or caps do not appear as extra objects.

The ExtrNP object provides a list of NURBS patch objects in the same order as created upon conversion.

RIB Export

ExtrNP objects will be exported as NURBS patch primitives:

```
RiNuPatch(...);
```

If bevels or caps are present, those follow as potentially trimmed NURBS patch primitives in the following order: U0, U1, V0, V1.

PV tags are supported but all NURBS patch primitives will get the same set of tags.²

Multiple TC tags are also supported.³

¹ Since 1.15. ² Since 1.20. ³ Since 1.24.

4.7.13 OffsetNP (Offset NURBS Surfaces) Object

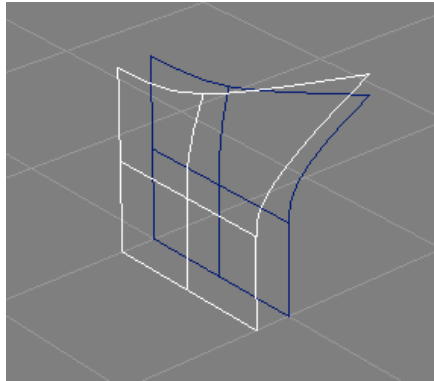


Figure 99: Offset Surface (white) from NURBS Surface (blue) with Offset 0.2

The OffsetNP object creates offset surfaces from NURBS surfaces using a simple algorithm: each control point is moved along a normal vector obtained by all direct neighboring control points.¹ See also the image above.

The offset surface will always match the original surface in width, height, orders, and knots.

The offsetting also works for closed and periodic surfaces in any possible combinations in the two dimensions.² Degenerate surfaces are supported.³ However note that rational surfaces are still not supported. No attempt is made to prevent collisions or self intersections.

Trim curves are copied verbatim from the parameter surface to the offset surface.

As the geometry of the offset surface is completely defined by the master surface and the offset parameter, OffsetNP objects do not support own transformation attributes.⁴

The following table briefly lists some capabilities of the OffsetNP object.

Type	Parent of	Material	Converts to / Provides	Point Edit
OffsetNP	NPatch	Yes	NPatch ⁺	No*

Table 83: OffsetNP Object Capabilities

OffsetNPAttr Property

The following parameters control the offsetting process:

- "Offset" determines the distance between original surface and offset surface. Negative values are allowed.
- See section 4.6.1 NPatchAttr (page 170) for a description of the other two attributes "DisplayMode" and "Tolerance".
- Finally, a "NPInfo" field informs about the actual configuration of the created NURBS surface.

¹ Since 1.17. ² Since 1.19. ³ Since 1.21. ⁴ Since 1.19.

Caps and Bevels

The OffsetNP object supports the standard caps as lined out in section 4.10.5 Caps Property (page 255) and the standard bevels as lined out in section 4.10.6 Bevels Property (page 256).

The boundary names are U0, U1, V0, and V1.

Conversion Support

The offset surface may be converted to an ordinary NURBS patch using the main menu entry "Tools/Convert".

If bevels or caps are present, an enclosing Level object will be created and the bevels or caps follow the offset surface in the following order: U0, U1, V0, V1.

Integrated bevels or caps do not appear as extra objects.

The OffsetNP object provides a list of NURBS patch objects in the same order as created upon conversion.

RIB Export

OffsetNP objects will be exported as NURBS patch primitives:

```
RiNuPatch(...);
```

If bevels or caps are present, those follow as potentially trimmed NURBS patch primitives in the following order: U0, U1, V0, V1.

PV tags are supported but all NURBS patch primitives will get the same set of tags.¹

Multiple TC tags are also supported.²

¹ Since 1.20. ² Since 1.24.

4.7.14 Text Object



Figure 100: Text Object set in Verdana

Text objects may be used to easily create objects that form letters or even whole words in very high quality. For that, they parse TrueType font description files, extract the Bezier curves from the font description, sort the curves, connect them properly and finally extrude them. As with the Extrude objects, caps and bevels may be created automatically.

Parsing of TrueType font descriptions is quite tricky. For the sake of brevity and ease of the implementation, Ayam does not support elaborate TrueType features like kerning tables, that e.g. control distances between certain letters (You are not going to typeset a book with Ayam anyway, aren't you?). Therefore you might experience wrong letter distances from time to time. If this happens, just create a Text object for each letter, and arrange the objects as you like.

The Text object can be converted to ordinary NURBS patches using the main menu entry "Tools/Convert".

The following table briefly lists some capabilities of the Text object.

Type	Parent of	Material	Converts to/Provides	Point Edit
Text	No	Yes	NPatch ⁺	No*

Table 84: Text Object Capabilities

TextAttr Property

The following attributes control the creation of the text objects.

- Using "FontName" a TrueType font description file is specified. Those files usually have the file name extension ".ttf". Only real TrueType font files, containing Bezier curve font descriptions, are supported. There are also rastered, bitmap containing TrueType font description files, those will not work.
- Using "String" you specify the letters to be created. This entry (and the corresponding data structures) are Unicode clean. This means you can put any Unicode letters into this entry. You should of course make sure, that the specified letters are included in the selected font file.
- "Height" controls the height of the extruded object.
- "Revert" reverts the sense of inside-outside detection mechanism for the cap generation. Depending on the actual font description file (or even letter) you may need to toggle this to get caps.

- "UpperCap", "LowerCap", work like for the Extrude object (see section 4.7.2 [ExtrudeAttr Property](#) (page 185) for a more exhaustive description of those parameters).
- "Add Lower Bevel!", "Add Upper Bevel!": Since Ayam 1.10 the bevel parameters of the text object are saved in bevel parameter tags and the property GUI changed to conform to all other bevel supporting tool objects. The old options "LowerBevel", "UpperBevel", "BevelType", "BevelRadius", and "RevertBevels" are no longer available. They were replaced with new dynamic tag creating bevel property GUI sections that are accessible through the new command entries "Add Lower Bevel!" and "Add Upper Bevel!" respectively. If one of those entries is used, a bevel parameter tag is created and more options will be made available in the property GUI to adjust the bevel parameters or remove the tag again. A more thorough discussion of those options is available in section 4.7.9 [BevelAttr Property](#) (page 209). Just one note: for some fonts, the bevel radius has to be set to really small values (about 0.001) to get proper bevels and caps. This is because of sharp corners in some letters that lead to self overlapping borders of the bevel surfaces with otherwise perfectly normal values for the bevel radius.

See section 4.6.1 [NPatchAttr](#) (page 170) for a description of the other two attributes "DisplayMode" and "Tolerance".

Conversion Support

The extruded surfaces, the bevels, and the caps, may be converted to ordinary NURBS patches using the main menu entry "Tools/Convert".

If bevels or caps are present, an enclosing Level object will be created and the caps follow the extruded surfaced in the following order: end bevel, end cap, start bevel, start cap.

The Text object provides a list of NURBS patch objects in the same order as created upon conversion.

RIB Export

Text objects will be exported as NURBS patch primitives:

```
RiNuPatch(...);
```

If caps or bevels are present, those follow as potentially trimmed NURBS patch primitives in the following order: end bevel, end cap, start bevel, start cap.

PV tags are supported but all NURBS patch primitives will get the same set of tags.¹

¹ Since 1.20.

4.7.15 Trim Object

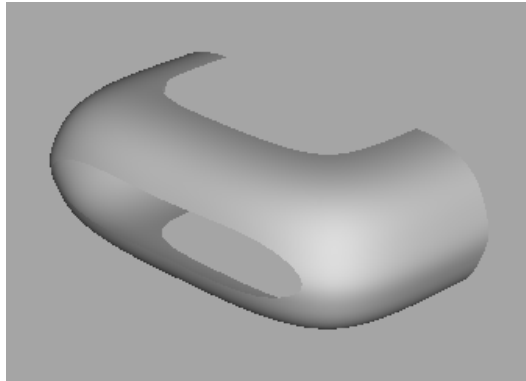


Figure 101: Trimmed Swing Object Example

The Trim object may be used in hierarchies of tool objects to trim NURBS patch providing objects otherwise unavailable to trimming like e.g. a Revolve object.¹

See also the image above which depicts the swung surface example object further trimmed by a Trim object.

The first child of the Trim object is the NURBS patch providing object and the second object is the trim curve (defined in the parametric space of the NURBS surface). More curves and loops may follow. All parameter curves must obey the rules for trimming as outlined in section 4.6.1 Trim Curves (page 171). The surface may already be trimmed and there may be multiple provided patches, however, only one of them will be trimmed by the Trim object.

The object hierarchy of a Trim object, thus, looks like this:

```
+-Trim
  |-Surface (Revolve)
  |-Trim_1 (NCurve)
+-[Trim_2 (Level)
  | |-NCurve
  | \-NCurve
  | ...
  \-Trim_n (ICurve) ]
```

The following table briefly lists some capabilities of the Trim object.

Type	Parent of	Material	Converts to/Provides	Point Edit
Trim	NPatch/NCurve ⁺ /Level ⁺	Yes	NPatch	No

Table 85: Trim Object Capabilities

¹ Since 1.16.

TrimAttrib Property

The following parameters control the trimming process:

- "PatchNum" allows to select a patch, should the NURBS patch providing object deliver a list. This way, a bevel of an extrusion might be trimmed.
- "ScaleMode" controls how the trim curve is scaled to the NURBS patch parameter space:
In mode "Absolute" no scaling happens.
In mode "Relative" the trim curves are expected to be defined between 0 and 1 (in x *and* y dimension) and will be scaled to the patch appropriately, no matter how the parametric space of the patch actually looks like (i.e. it works the same for a patch where the knots range from 0 to 1, 0 to 2, or even 3 to 3.5).

Conversion Support

The trimmed surface may be converted to an ordinary NURBS patch using the main menu entry "Tools/Convert".

RIB Export

Trim objects will be exported as NURBS patch primitives:

```
RiNuPatch(...);
```

PV tags are currently not supported.

4.8 Polygonal and Subdivision Objects

These objects complement the Ayam feature set and allow objects modelled in the polygonal or subdivision modelling paradigms to be included in Ayam scenes.

4.8.1 PolyMesh Object

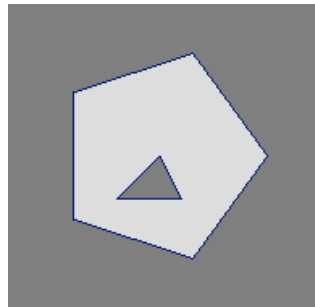


Figure 102: PolyMesh Example

The PolyMesh object may be used to include objects that have been modeled using the polygonal modelling paradigm in Ayam scenes. See the image above for a simple example.

There are just a few special modelling actions for this type of object (see section 2.2 [PolyMesh tools](#) (page 35)), but its control points can be selected and modified as it can be done with other object types, e.g. curves.

The PolyMesh object is equivalent to the general points polygons primitive of the RenderMan interface. This means, each PolyMesh object may contain multiple general (convex or concave) polygons, which in turn may consist of an outer loop and an arbitrary number of inner loops that describe holes in the polygon (see also the image above, showing a polygonal mesh with one pentagonal face and a triangular hole). The loops use a point indexing scheme to efficiently reuse coordinate values. This general approach requires a so called tessellation to be carried out, in order for the PolyMesh object to be shaded. For the tessellation, Ayam uses routines of the GLU library.

Avam is able to automatically create face normals for PolyMeshes. They will be calculated while tessellating the PolyMesh and be perpendicular to the plane determined by the first three vertices of the outer loop of a polygon. Furthermore, Ayam supports vertex normals (normals stored for every control point).

Note that storing single triangles in PolyMesh objects will lead to a real waste of memory. The merge tool (main menu "Tools/PolyMesh/Merge") can be used to combine many PolyMesh objects into a single PolyMesh object.

The following table briefly lists some capabilities of the PolyMesh object.

Type	Parent of	Material	Converts to/Provides	Point Edit
PolyMesh	No	Yes	SDMesh	Yes

Table 86: PolyMesh Object Capabilities

PolyMeshAttr Property

The PolyMeshAttr GUI just displays some information about the PolyMesh object:

- "NPolys" the number of polygons.
- "NControls" the total number of control points defined.
- "HasNormals" is 1 if the object uses vertex normals, else it is 0.

Conversion Support

PolyMesh objects may be converted to SDMesh objects using the main menu entry "Tools/Convert".¹

Note that no verification of the usability of the mesh as base mesh for a subdivision surface is carried out. Usually, such meshes have to be manifold and may not contain T-junctions.

RIB Export

PolyMesh objects will be exported as `RiPointsGeneralPolygons` primitives (regardless of whether the actual configuration would fit into a simpler polygonal primitive of the RenderMan interface, e.g. a `RiGeneralPolygon`).

PV tags are supported.

Scripting Support

For scripting interface commands that manipulate PolyMesh objects see section 6.2.17 [Manipulating Poly-Mesh Objects \(page 412\)](#)).

¹ Since 1.11.

4.8.2 SDMesh Object

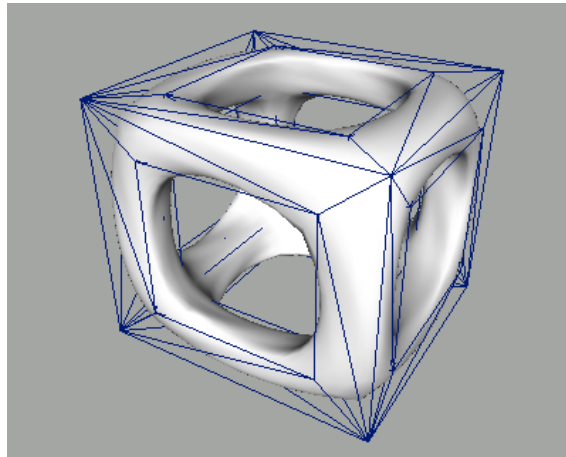


Figure 103: SDMesh object with original polygonal mesh (blue)

The SDMesh object may be used to include objects that have been modeled using the subdivision modelling paradigm in Ayam scenes (see also the image above).

There are no special modelling actions for this type of object, but its control points can be selected and modified as it can be done with other object types, e.g. curves.

The SDMesh object is equivalent to the Subdivision Mesh primitive of the RenderMan interface. This means, each SDMesh object may contain multiple faces with arbitrary number of vertices that form a polygonal mesh. This polygonal mesh is then successively refined using a subdivision scheme and, depending on the number of refinement (or subdivision) steps, results in a more or less smooth surface. There are several different subdivision schemes, but the scheme currently supported by most RenderMan compliant renderers is named "Catmull-Clark".

Tags may be specified for faces, edges, or vertices to control the subdivision process (e.g. to create sharp corners or edges in the resulting surface). All tags known from the RenderMan interface (hole, crease, corner, and interpolateboundary) are supported by Ayam, but they may currently not be changed by the user.

Unless the "subdiv" plugin (available since Ayam 1.19) is loaded, Ayam is not able to do the subdivision and show the resulting smooth surface. All that is shown in wire-frame and shaded views is the original polygonal mesh.

The following table briefly lists some capabilities of the SDMesh object.

Type	Parent of	Material	Converts to/Provides	Point Edit
SDMesh	No	Yes	PolyMesh	Yes

Table 87: SDMesh Object Capabilities

SDMeshAttr Property

The SDMeshAttr GUI just displays some information about the SDMesh object:

- "Scheme", is the subdivision scheme, currently available schemes are Catmull-Clark and Loop.¹

¹ Since 1.11.

- "Level" is the number of subdivision steps that should be carried out when subdividing the mesh for preview. This subdivision needs the "subdiv" plugin.
- "DrawSub" allows to switch between the control polygon and the subdivided polygon outlines when drawing the mesh.
- "NFaces", the number of faces.
- "NControls", the total number of control points defined.

Conversion Support

SDMesh objects may be converted to PolyMesh objects.¹

Note however that only the original, unrefined, control polygon (i.e. the base mesh) will be converted unless the "Level" attribute is not zero *and* the "subdiv" plugin is loaded.

RIB Export

SDMesh objects will be exported as subdivision mesh primitives:

```
RiSubdivisionMesh(...);
```

PV tags are supported.

¹ Since 1.11.

4.9 Script and Custom Objects

These objects create/modify arbitrary other objects from scripts or define entirely new object types via the custom object plugin mechanism.

4.9.1 Script Object

```
crtOb NCurve -l 30 -kt 1
sL
for {set i 0} {$i < 30} {incr i} {
    set x [expr $i*cos($i*20.0)/10.0]
    set y [expr $i*sin($i*20.0)/10.0]
    setPnt $i $x $y 0 1
}
```



Table 88: Script Object Example: Tcl Code (l), Resulting Curve (r)

Script objects are the most flexible object type of Ayam. They may be used to create new objects, modify existing objects, or realise mechanisms like constraints using small scripts that are embedded in the Script objects themselves.

Those small embedded scripts may employ functionality from Tcl and the Tcl scripting interface of Ayam (see also section 6 [Scripting Interface](#) (page 345)). See also the table above, depicting a simple example script object that creates a NURBS curve and puts the control points of the curve in a spiral in the XY-plane. This example is also distributed with Ayam.

Script objects may also use arbitrary, plugin provided, scripting languages, like JavaScript, provided by the "jsinterp" plugin (see also: 6.7 [JavaScript Scripting Interface](#) (page 473)).¹

The binary and source distributions of Ayam contain several example scripts for Script objects in the "ayam/bin/scripts" and "ayam/src/scripts" directories, respectively. In addition, there are example scene files using Script objects in the "ayam/scn/scripts" directory, see also section 6.6 [Distributed Script Objects](#) (page 460).

The following table briefly lists some capabilities of the Script object.

Type	Parent of	Material	Converts to/Provides	Point Edit
Script	Any ⁺	No	Any ⁺	No*

Table 89: Script Object Capabilities

¹ Since 1.18.

Script Object Usage

The script of a Script object will be run each time the script is modified and each time the notification callback of the Script object is called (e.g. because one of the children of the Script object changed). As long as the script of a Script object is executed, Ayam will not process any events except for checking whether the script emergency hotkey <Ctrl+C>, that may also be used to escape from infinite loops in the Ayam console, is pressed. Calling commands and procedures that lead to the processing of events or that are slow because they manipulate or update the GUI of Ayam should be avoided. In particular, the following procedures and commands should *not* be used: `uS`, `uCR`, `uCL`, `selOb`, `plb_update`, `undo`!

As a Script objects script is fired with the notification of said object, and recursive notification is blocked by the Ayam core, the script can not rely on automatic multilevel notification when creating hierarchies of objects, e.g. for tool objects. Instead, the notification of a tool object created by the script has to be triggered manually by the script.

If a script fails, the Script object will be disabled by means of the "Active" attribute.¹ Furthermore, the script line where the error occurred will be highlighted in the script editor.

Script objects may also create their own property GUIs for e.g. script parameters.² This may be accomplished by adding tags of type "NP" with the name of the new property as value to the Script object (menu entry "Special/Tags/Add Property"). The script itself is responsible for data management and property GUI creation (see section 4.9.1 [ScriptAttr Property \(page 231\)](#) below).

There is also a tag type to remove properties ("RP").³ Using this tag, the actual script code can be hidden and thus users are blocked from unintentionally changing it.

Starting with Ayam 1.16, the environment for running Script objects scripts has been refined to allow more complex scripts (that traverse the scene or use the clipboard) to be written. When a script is running,

- the current level is initially the child level of the respective Script object,
- the object clipboard is saved and cleared before running the script and also re-established after the script finished, and
- the "CreateAt" and "CreateIn" options are both disabled (i.e. objects will be created with default transformation attributes).

RIB Export

Script objects will be exported to RIB files as the objects they create/modify.

¹ Since 1.24. ² Since 1.8.2. ³ Since 1.12.

ScriptAttr Property

This section discusses the available Script object types and additional controlling parameters.

- If "Active" is disabled, the script will not be run.
- "Type" is the type of the Script object. Three types of Script objects are currently available:
 1. "Run", the script will be run and no special action will take place.
 2. "Create", the script will be run and will create and parameterise new objects. After running the script, the newly created object(s) will automatically be moved into the internal data structure of the Script object. The Script object will look like and act as an object of the type that the script created. If the script creates e.g. a NCurve object, the Script object may be used as parameter object of a tool object that needs a NCurve, e.g. a Sweep:

```
+--Sweep
  |--Cross_Section (Script)
  \--Path (NCurve)
```

If the newly created object has to be selected by the script code for further parameterisation purposes, the selection should be done using the scripting interface command "sL" (which performs a hidden selection in the safe interpreter context). Consequently, the most simple example script for a Script object of type "Create" looks like this:

```
crtOb NCurve
```

or, with further parameterisation:

```
crtOb NCurve
sL
setProperty NCurveAttr (Order) 2
```

3. "Modify", if the Script object has child objects, these child objects will be temporarily moved into the internal data structure of the Script object. A copy of all child objects will be created as new children of the Script object. A selection of the new child objects will be established, then the script will be run. Usually, the script modifies one of the selected objects (moves control points, adds tags, or does something similar). Afterwards, the two sets of objects will be exchanged, the modified objects will be moved to the internal data structure of the Script object while the unmodified original child objects will again be child objects of the Script object. The modified objects will henceforth be provided upstream to potential parents. If certain actions in the script shall be restricted to one of the child objects of the Script object, the "withOb" command may be used to accomplish this easily. The Script object will look like and act as an object of the type of the first child object of the Script object. If the Script object has e.g. a NCurve object as first child, the Script object may be used as parameter object of a tool object that needs a NCurve, e.g. a Sweep:

```
+--Sweep
  +-Cross_Section (Script)
  | \-NCurve
  \--Path (NCurve)
```

A simple example script for a Script object of type "Modify" that needs a single NURBS curve as child object may look like this:

```
revertC
```

Note: In order to make this work for objects that are not supported by the `revertC` command directly but provide NURBS curves (e.g. `ExtrNC` objects or instances of NURBS curves) the code has to look like this:

```
convOb -inplace; revertC
```

- "Script" is the script code. The corresponding widget is a text widget that allows to directly edit the code.

Many standard mouse and keyboard shortcuts, e.g. for clipboard handling and undo are available. In addition, syntax highlighting and dynamic brace highlighting are available. Furthermore, line numbers can be shown in an additional widget to the left of the text area (this is initially hidden, see the context menu).¹ It is also possible to edit the code in an external editor and copy it to the Script object using the operating system clipboard and the "Paste (Replace)" context menu entry of the text widget. In contrast to the other property GUI elements, the text widget has a resize handle in the lower left corner, and a dynamically appearing scroll bar, see also the image below.

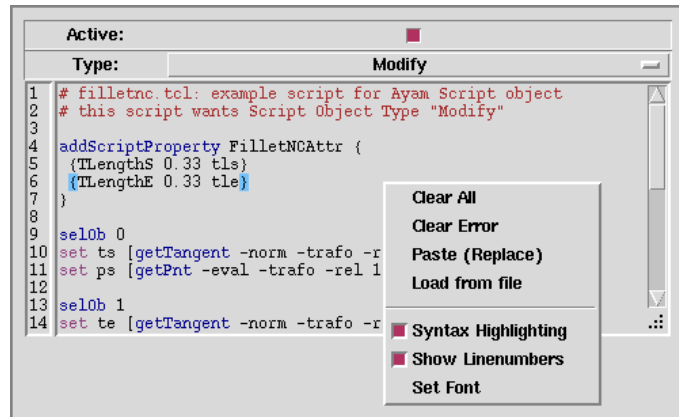


Figure 104: Script Property GUI with Context Menu and Resize Handle

Finally, the font used to display the script may be adjusted using the corresponding entry from the context menu.

Script Parameters and Property GUIs

If the script has a property GUI it is important to save the property data array between multiple copies of a Script object (to make them truly unique and not operating on the same set of parameters) and to scene files.

This can be accomplished using a comment in the first line of the script. If it looks like this:

```
# Ayam, save array: <arrayname>
```

then the global Tcl array <arrayname> will be saved with the Script object to Ayam scene files. The array must contain an entry "SP" that lists all individual parameters of the Script object. Note that only parameters from this list will be saved to Ayam scene files. Note also, that "SP" must *never* contain "SP". All copies of a Script object must share the same set of individual vs. shared parameters. If Script objects need to store differing/individual amounts of parameter data, lists can be used as individual parameters.

¹ Since 1.35.

Scripts in Foreign Languages

Script objects may also use arbitrary, plugin provided, scripting languages.¹ To switch to a different language, the first line of the script must be a comment (in the syntax of the other language) with the keyword "use:" followed by the language name, as provided by the corresponding plugin, e.g. for JavaScript the first line should look like this:

```
/* Ayam, use: JavaScript */
```

The special comments for saving of array items and language switching can be used in conjunction like this:

```
/* Ayam, use: JavaScript, save array: MyArr */
```

Safe Interpreter

In Ayam versions prior to 1.16 Script object scripts could use any functionality of Tcl, Tk, and the Tcl scripting interface of Ayam which posed a huge security risk. This is no longer the case. Script objects scripts now run in a safe interpreter with reduced instruction set. They can no longer write to the file system, get onto the network, or confuse the application state. Direct access to Tk is also completely blocked, but Script objects still can have their own property GUIs (refer to the examples below).

In particular, the following Tcl commands are *not* available in the safe interpreter: `cd`, `encoding`, `exec`, `exit`, `fconfigure`, `file`, `glob`, `load`, `open`, `pwd`, `socket`, `source`, `unload`; `auto_exec_ok`, `auto_import`, `auto_load`, `auto_load_index`, `auto_qualify`, `unknown` (the missing `unknown` and autoloading facilities lead to further unavailability of commands normally available via autoloading, like e.g. `parray`, `history`).

The `puts` command is available in a limited fashion: only access to the `stdout` and `stderr` channels is allowed.

Avyam scripting interface commands that directly manipulate the user interface are also not available (`uS`, `rV` etc.). Please refer to the documentation of the scripting interface commands about their availability in the safe interpreter (see section 6.2 Procedures and Commands (page 349)).

In addition, access to global variables deserving protection like `env`, `ay`, `ayprefs` is not allowed. In fact, the safe interpreter has a completely separate set of variables. Transfer of data between both interpreters must be arranged manually from the Ayam interpreter (i.e. with scripts that run in the Ayam console).

With the help of scripts, that run in the Ayam interpreter, more commands may be transferred to or made available in the safe interpreter. But this may, of course, open security holes again.

Full access from Script objects to the complete scripting interface may be re-enabled by recompiling Ayam. If this is enabled and scene files containing Script objects are loaded, Ayam will raise a warning dialog, offering to temporarily disable all Script objects that will be read. The Script objects will be disabled using their "Active" Script object property and may be enabled again after careful inspection of the script code manually or using the main menu entry "Special/Enable Scripts".

¹ Since 1.18.

Transformation Support

Even though, initially, Script objects do not show and use the Transformations property, they can support transformations under certain circumstances. To enable this support, a NP tag with the value "Transformations" must be added to the Script object (i.e. the Transformations property must be visible in order to be effective).

Now, the result(s) of the script can, additionally, be manipulated by the standard interactive modelling actions, like move or scale.

Note, that upon provide or conversion the transformations from the Script object will be added to the transformations of the objects created by the script in the same way as by the delegate transformations tool, which fails for complex setups (e.g. shear transformations). However, if the objects created by the script have editable points, this problem can be avoided by using the "applyTrafo" command in the script.

Conversion Support

Script objects convert to the objects they create/modify using the main menu entry "Tools/Convert".

When converting in place, some special rules are in effect:

1. If there is just one created/modified object, its tags are appended to the tags of the Script object and its material setting only takes precedence if there is actually a material set.
2. If there are multiple created/modified objects, the Script object is transformed into a Level object, keeping its tags and material settings.
3. The created/modified objects will become children of the new Level object (with tags and material properties unchanged).
4. The current children of the Script object will be removed prior to conversion. If this fails (e.g. due to references), they may end up in the object clipboard.

If the script created a master object and instances of this master, normal conversion will *not* be able to duplicate this relationship (due to the copy semantics of instance objects, see also section 4.2.7 [Instances and the Object Clipboard](#) (page 132)). To get around this, just copy the Script object and then use in place conversion on the copy.

Caps and Bevels

Script objects of type "Create" and "Modify" support the standard caps as lined out in section 4.10.5 [Caps Property](#) (page 255) and the standard bevels as lined out in section 4.10.6 [Bevels Property](#) (page 256).¹

The boundary names are U0, U1, V0, and V1.

By default, the corresponding properties are not available. To make them visible, NP tags need to be added to the Script object.

The caps and bevels from the Script object will be added to all created/modified objects that do not possess own caps and bevels already and that support caps and bevels.

¹ Since 1.26.

Parametric Line Example

This section illustrates the development of a Script object for parametric lines, otherwise unavailable in Ayam.

We start with a simple version that first creates a NURBS curve object with two control points and then places the control points each at \pm half the desired line length on the X-axis. Just copy the following code to the Script property of a Script object of type "Create" and activate it.

```
set length 1
crtOb NCurve -length 2
sL
setPnt 0 [expr {-length/2.0}] 0.0 0.0 1.0
setPnt 1 [expr {length/2.0}] 0.0 0.0 1.0
```

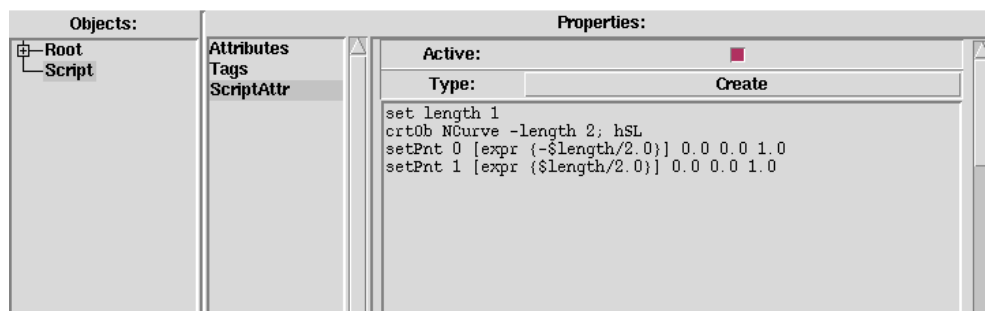


Figure 105: Simple Script for Parametric Line Primitive

This code works, but if lines of a different length than 1 are needed, the user must edit the script which is not very convenient and error prone.

A complete, easy to use, and safe GUI for the length parameter can be added by changing the script code to:

```
addScriptProperty LineAttr {{Length 2.0 length}}
crtOb NCurve -length 2
sL
setPnt 0 [expr {-length/2.0}] 0.0 0.0 1.0
setPnt 1 [expr {length/2.0}] 0.0 0.0 1.0
```

resulting in a new clickable graphical user interface as can be seen in the following image:

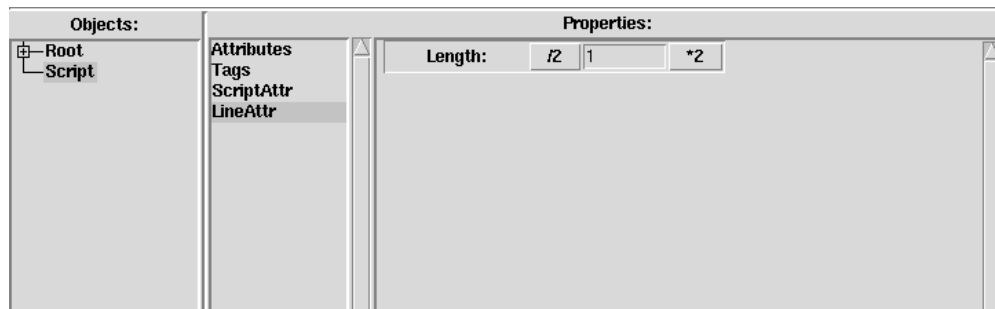


Figure 106: Parametric Line Primitive with Parameter GUI

The newly inserted call to `addScriptProperty` does quite a bit of complicated setup/management code, which is also all too easy to do wrong, given that there are actually two interpreters involved and certain things must/must not be done on the first script run (e.g. after being read from a scene file).

The GUIs created this way are, however, limited to four basic GUI element types and all members of the respective data arrays will be saved to scene files. See also section 6.2.27 `addScriptProperty` (page 431).

The manual but more flexible way of creating/managing a property GUI would look like this:

```
# Ayam, save array: LineAttrData
if { ![info exists ::LineAttrData] } {
    array set ::LineAttrData {
        Length 1
        SP {Length}
    }
}
if { ![info exists ::LineAttrGUI] } {
    set w [addPropertyGUI LineAttr]
    addParam $w LineAttrData Length
}
set length $::LineAttrData(Length)
crtOb NCurve -length 2
sL
setPnt 0 [expr {- $length/2.0}] 0.0 0.0 1.0
setPnt 1 [expr { $length/2.0}] 0.0 0.0 1.0
```

To actually see the property GUI, a "NP" (new property) tag with the value "LineAttr" would also have to be added to the Script object.

The manual GUI setup code first creates a Tcl array essential to manage the data of an Ayam object property (LineAttrData). Then, the LineAttr property GUI is created and a GUI element is added to the GUI using "addParam". Note that the "addPropertyGUI" command expects for a property named "SomePropertyName" a corresponding property data array named "SomePropertyNameData" to exist. The GUI setup code should just run once, therefore it checks for the presence of the variable "LineAttrGUI" (which is created on the first run of "addPropertyGUI") first. See also sections 6.2.27 Property GUI Management (page 425) and 6.1.5 Global Property Management and Data Arrays (page 347) for more information about property GUIs and the Ayam scripting interface.

Finally, to enable saving of the parameter value in the new property "LineAttr" to scene files, a comment must be prepended to the script ("Ayam, save array: LineAttrData"), and to enable multiple and individually parameterised copies of this Script object, a "SP" entry needs to be added to the "LineAttrData" array as well.

The complete script is also available as example script file "scripts/crtlinegui.tcl" in the Ayam distribution.

Hierarchy Building Example

This example script demonstrates the scene traversal and hierarchy building capabilities available to Script objects since Ayam 1.16.

Create a Script object, and add two children to it, a box and a NURBS curve (order 2, knot type: chordal works best). Then add the following script to the Script object:

```
# this script needs object type "Modify" and two children:
# a box/sphere and a curve
withOb 1 {estlenNC len}
cutOb
crtOb Clone
goDown -1
pasOb -move
goUp
sL
getProp
set CloneAttrData(NumClones) [expr round($len)]
setProp
```

This little script first determines the length of the curve, then it creates a Clone object and moves the children of the Script object to it (via the object clipboard). Finally, the Clone object is parameterised, so that the trajectory is completely filled (assuming that each copy of the first child needs one length unit on the trajectory) with objects. The curve can now be modified using interactive modelling actions, or its length can be changed, and the trajectory will always be completely filled with an appropriate number of box objects. See example image below (compare the Clone configurations from the two different trajectory curves):

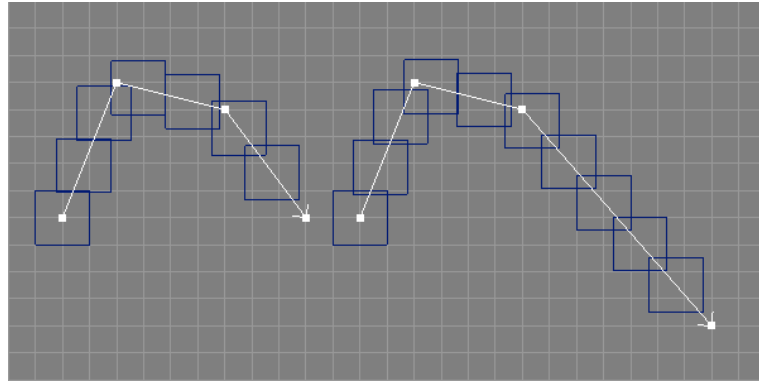


Figure 107: Hierarchy Building Script Object Example

4.9.2 Custom Objects Management

Custom objects are plugins that extend the Ayam capabilities by defining entirely new types of e.g. geometric objects. This may be done easily, because the Ayam core is written in a modelling paradigm independent way.

Custom objects may also define own modelling tools. Those can usually be found in the "Custom" main menu.

Unlike other modelling helper plugins, custom object plugins will be loaded automatically with the scene files that contain such objects. Note, that this only works properly if the preference option "Main/Plugins" is correctly set.

Several custom object plugins are already distributed with Ayam. Those are documented in the next sections.

4.9.3 SfCurve (Superformula Curve) Object

The SfCurve object creates a superformula curve from four parameters named m , $n1$, $n2$, and $n3$ (see also the image below). The superformula is a generalization of the superellipse; in polar coordinates it is defined as:

$$r(t) = \left(\cos\left(\frac{m \times t}{4}\right)^{n2} + \sin\left(\frac{m \times t}{4}\right)^{n3} \right)^{-\frac{1}{n1}} \quad (1)$$

where r is the radius and t the angle. The SfCurve object allows to specify start and end values for t as well as the number of sample points in between.

The generated NURBS curve is always closed, but the order may be configured freely.

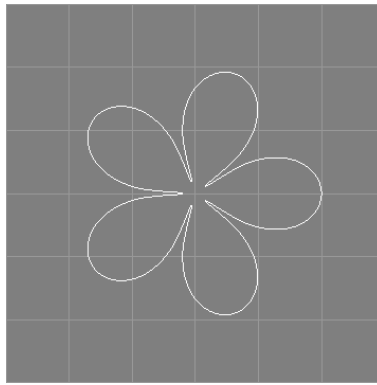


Figure 108: Superformula Curve from Parameters 10, 22, -11, 3

The following table briefly lists some capabilities of the SfCurve object.

Type	Parent of	Material	Converts to / Provides	Point Edit
SfCurve	No	No	NCurve	No*

Table 90: SfCurve Object Capabilities

SfCurveAttr Property

The SfCurve object provides the following parameters:

- The parameters "M", "N1", "N2", and "N3" control the superformula.
- "TMin" is the start angle.
- "TMax" is the end angle.

Note that the angle defined by "TMin" and "TMax" may actually be larger than 360, as can be seen in the image below:

Also note that e.g. setting "TMin" to 1 and "TMax" to 361 may deliver better results than using the default values.

- "Sections" is the number of sample points. Curves with sharp features may need high values of about 100.
- "Order" is the desired order of the curve.

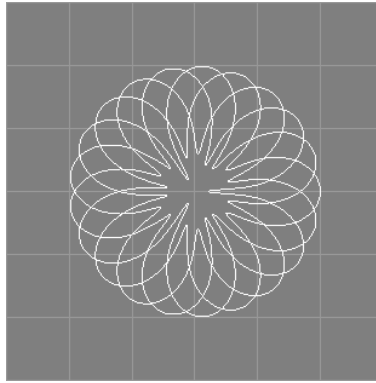


Figure 109: Superformula Curve with TMax 1440

- See section 4.4.1 [NCurveAttr](#) (page 150) for a description of the parameters: "Tolerance" and "DisplayMode".

Operation Support

The superformula curve may be converted to an ordinary NURBS curve using the main menu entry "Tools/Convert".

Furthermore, the superformula curve fully supports the revert, refine, and coarsen operations.¹

RIB Export

SfCurve objects never directly appear in RIB output (only indirectly as trim curve).

¹ Since 1.26.

4.9.4 BCurve (Basis Curve) Object

The BCurve (basis curve) object creates a cubic parametric curve from a number of rational control points and a basis (which is a 4 by 4 matrix).¹ The basis defines the interpretation of the control points, similar to the bicubic patch mesh primitive of the RenderMan interface, see also section 4.6.5 [PatchMesh Object \(page 179\)](#). The BCurve therefore represents a range of parametric curves like B-Spline, Bezier, Catmull-Rom, and Hermite splines.

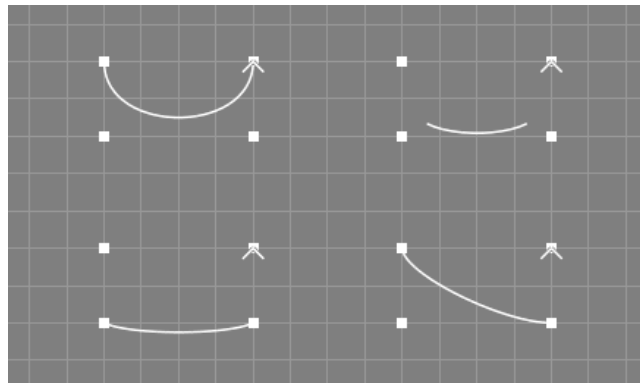


Figure 110: BCurves of Basis Types *Bezier* (ul), *B-Spline* (ur), *Catmull-Rom* (ll), and *Hermite* (lr)

See also the image above depicting simple example curves of some basis types.

The following table briefly lists some capabilities of the BCurve object.

Type	Parent of	Material	Converts to/Provides	Point Edit
BCurve	No	No	NCurve	Yes

Table 91: BCurve Object Capabilities

BCurveAttr Property

The BCurve object provides the following parameters:

- "Closed" determines whether the curve is closed.
- "Length" is the number of control points, valid values depend on the basis type/step below, see also the corresponding discussion in section 4.6.5 [PatchMeshAttr Property \(page 179\)](#). The arrow buttons of this entry field add/subtract the current step size of the curve to/from the respective value when the <Control> key is held down.
- "BType" is the basis type. The following basis types are available: "Bezier", "B-Spline", "Catmull-Rom", "Hermite", "Power", and "Custom".
- "Basis" is the basis.
- "BStep" is the step size, this value must be 1, 2, 3, or 4.
- See section 4.4.1 [NCurveAttr \(page 150\)](#) for a description of the parameters: "Tolerance" and "DisplayMode".

The entries "Basis" and "BStep" are only visible, if the basis type is "Custom".

¹ Since 1.25.

Operation Support

The basis curve may be converted to an ordinary NURBS curve using the main menu entry "Tools/Convert".

Furthermore, the basis curve fully supports the revert, open, close, refine, and coarsen operations.¹

RIB Export

BCurve objects never directly appear in RIB output (only indirectly as trim curve).

Scripting Interface

The BCurve object plugin defines a scripting interface command to convert BCurve objects to a different basis, see also section [6.2.16 tobasisBC](#) (page 411).

¹ Since 1.26.

4.9.5 SDCurve (Subdivision Curve) Object

The SDCurve (subdivision curve) object creates a curve from a number of non-rational control points.¹ Subdivision curves are similar to subdivision surfaces, but a polygon is subdivided to a limit curve instead of a polygonal mesh to a limit surface, see also section 4.8.2 SDMesh Object (page 227).

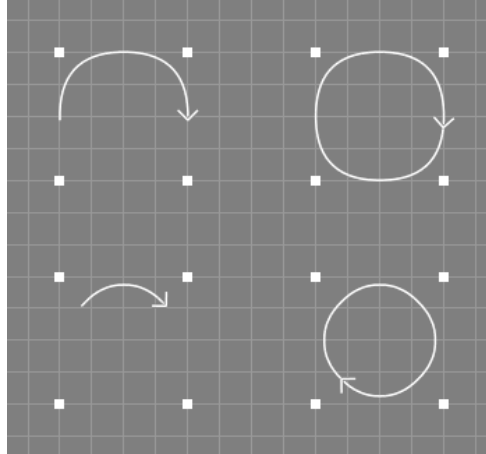


Figure 111: SDCurve Examples of Level 4 (Chaikin: u, Cubic: l)

See also the image above depicting four simple example curves.

The following table briefly lists some capabilities of the SDCurve object.

Type	Parent of	Material	Converts to / Provides	Point Edit
SDCurve	No	No	NCurve	Yes

Table 92: SDCurve Object Capabilities

SDCurveAttr Property

The SDCurve object provides the following parameters:

- "Closed" determines whether the curve is closed.
- "Length" is the number of control points.
- "Type" is the subdivision method to use, the currently implemented methods are "Chaikin" and "Cubic".
For each iteration, the "Chaikin" method inserts two new points into each section of the original control polygon, all old points are removed.
The "Cubic" method inserts one new point into each section of the original control polygon, the original points will not be deleted but rather moved to the barycenter of the two surrounding new points and the original point.
- "Level" determines the number of iterations of the subdivision algorithm to carry out.
- "SLength" is the number of points generated by the subdivision algorithm.

¹ Since 1.26.

Operation Support

The subdivision curve may be converted to an ordinary NURBS curve of order 2 using the main menu entry "Tools/Convert".

Arbitrary curve objects can also be converted to subdivision curves using the provided conversion tool that is accessible via the main menu entry "Custom/SDCurve/From Curve". This tool supports NCurve, ICurve, and ACurve objects directly, all other curve objects need just to provide their points to be supported. Furthermore, the subdivision curve fully supports the revert, open, close, refine, and coarsen operations.

RIB Export

SDCurve objects never directly appear in RIB output (only indirectly as trim curve).

4.9.6 Metaball Object

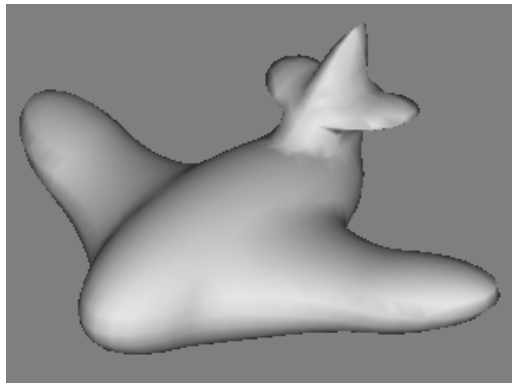


Figure 112: A Metaball Object from Six Meta Components

A metaball object is a custom object (see also section 4.9.2 Custom Object (page 238)). It allows you to model with implicit surfaces in realtime.

To start modelling you should first create a "MetaObj" object using the menu entry "Create/Custom Object/MetaObj" (if this menu entry is not available, you have to load the "metaobj" plugin using the menu entry "File/Load Plugin" first). "Create/Custom Object/MetaObj" creates a so called meta world with a single meta component (a sphere) in it. The meta world is represented by a "MetaObj" object and the component by a "MetaComp" object which is a child of the "MetaObj" object.

The complete template for the MetaObj object hierarchy, consequently, looks like this:

```
+--MetaWorld(MetaObj)
  |-C1(MetaComp)
  |-[ ...
  \-Cn(MetaComp) ]
```

The following table briefly lists some capabilities of the MetaObj and MetaComp objects.

Type	Parent of	Material	Converts to/Provides	Point Edit
MetaObj	MetaComp ⁺	Yes	PolyMesh	No
MetaComp	No	No	N/A	No

Table 93: MetaObj/MetaComp Object Capabilities

Meta components live only in a meta world, therefore it makes no sense to create "MetaComp" objects in other places except as a child of a "MetaObj" object. Type, parameters, and transformation attributes of the meta components define the function of an implicit surface. The "MetaObj" object, that represents the meta world, evaluates this function on a regular three-dimensional grid and creates a polygonal representation for a specific function value (the so called threshold value).

MetaObjAttr Property

The following attributes control the creation of the implicit surface:

- With the parameter `"NumSamples"` you specify the resolution of the three-dimensional regular grid, on which the implicit function is evaluated, in each dimension. A higher number of samples results in better quality but more polygons are created and more CPU power and memory are needed. For modelling you should set this to a lower value of about 40. For final rendering you may increase this to about 160.
- `"IsoLevel"`, defines the threshold value for that a polygonal representation of the implicit function should be created. Normally, you should not need to change this value.
- To show the actual bounds of the meta world, you may enable the `"ShowWorld"` parameter.

New in Ayam 1.5 is an adaptive calculation mode of the implicit surface. It may be switched on using the new attribute `"Adaptive"`. In the adaptive calculation mode, Ayam tries to vary the resolution of the resulting polygonal mesh according to the features of the implicit surface in order to capture fine details, even though a coarse grid is used. This is not done using a successively refined grid but by a refinement of the triangles created by the original algorithm (see also XXXX). You may control the adaptation process using three parameters: `"Flatness"`, `"Epsilon"`, and `"StepSize"`. If `"Adaptive"` is set to `"automatic"`, Ayam will not use the adaptive calculation while a modelling action is in progress. This mode has been introduced, because the adaptive mode may consume a considerable amount of CPU resources.

While modelling with meta balls you may add other `"MetaComp"` objects to the `"MetaObj"` object and parameterise them. A `"MetaComp"` object has the following properties.

MetaCompAttr Property

- `"Formula"` specifies the type of the meta component. The following types are available: Metaball, Torus, Cube, Heart, and Custom. The latter gives you the possibility to use your own formulas.
- With the parameter `"Negative"` you define a component with a negative effect on the implicit function value. Negative components are not visible on their own but they are useful for modelling holes. Just try it.

The other parameter are specific to the type of the component:

Metaball

- `"Radius"` sets the radius of the metaball
- `"EnergyCoeffA"`, `"EnergyCoeffB"`, and `"EnergyCoeffC"` are some parameters for the metaball formula. Usually you can leave those parameters at their default values. If you change them, be careful.

Torus

- `"Ri"` is the inner radius of the torus,
- `"Ro"` is the outer radius of the torus,
- `"Rotate"` rotates the torus about 90 degree.

Cube

- "EdgeX", "EdgeY", and "EdgeZ", let you define the sharpness of the edges of the cube.

Custom

- "Expression" is a piece of Tcl script, that represents your own custom formula for a meta component. The expression may call any Tcl commands to calculate a field value from the current grid position, which is given in the global variables "x", "y", and "z". The expression simply has to return the calculated field value.¹ Here is an example for a custom expression:

```
expr {pow($x, 4) + pow($y, 4) + pow($z, 4) }
```

Note that those expressions are called many times and since they are programmed in Tcl, this can be quite slow. You should use any tricks (like the curly braces in the expr-statement above) to speed up the expression.

Also note that in former versions of Ayam, the global variable "f" had to be set to return the field value. This is no longer the case, but old expression code that ends with `set f [expr ...]` is still valid.

Conversion Support

Metaball objects may be converted to PolyMesh objects using the main menu entry "Tools/Convert".

RIB Export

Metaball objects will be exported as RiPointsGeneralPolygons primitives (regardless of whether the actual configuration would fit into a simpler polygonal primitive of the RenderMan interface, e.g. a RiGeneralPolygon).

PV tags are currently not supported.

¹ Since 1.27.

4.9.7 SDNPatch Object

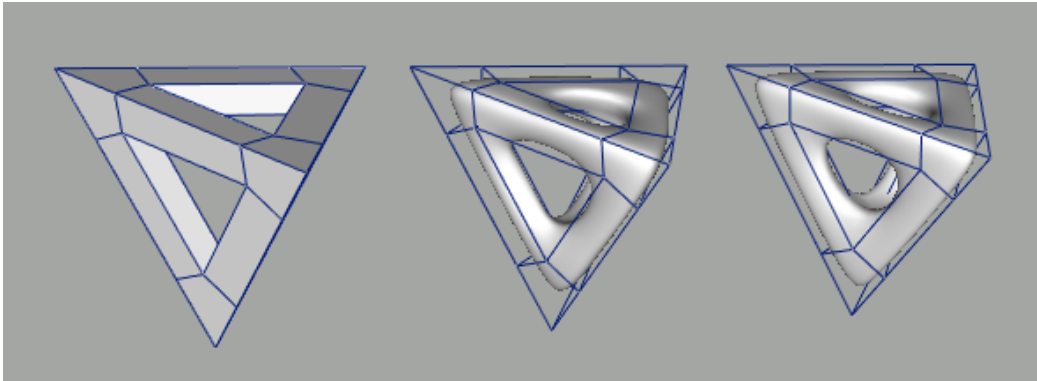


Figure 113: SDNPatches, l: control mesh, m: subdivided mesh with knot, r: subdivided mesh without knot

The SDNPatch custom object is available since Ayam 1.16 and allows to model with Subdivision NURBS, which extend the traditional subdivision scheme with knot values and rational coordinates. See also the image above, where in the middle mesh a knot value has been set in the left hand side of the mesh. The SDNPatch plugin is based on libsnurbs by Tom Cashman.

There are some special modelling actions for Subdivision NURBS defined (see below) and there are PLY import/export facilities. Furthermore, there are two conversion operations that convert NURBS patch and PolyMesh objects to SDNPatch objects. Thus, also SDMesh objects may be converted to SDNPatch objects in two steps: first to a PolyMesh then to the SDNPatch.

Please note that the plugin is still in experimental state, there is limited error checking and crashes may occur, if the special modelling actions are used.

The following table briefly lists some capabilities of the SDNPatch object.

Type	Parent of	Material	Converts to/Provides	Point Edit
SDNPatch	No	Yes	PolyMesh	Yes

Table 94: SDNPatch Object Capabilities

SDNPatchAttr Property

The SDNPatchAttr property allows to set the following SDNPatch specific attributes:

- "Degree" is the degree of the subdivision NURBS surface, the only valid values are currently 3, 5, and 7.
- "Level" is the subdivision level, a high level leads to many polygons and a smooth surface; useful values range from 0 to 5.
- "IsRat" informs, whether the patch is rational (has any weight values different from 1.0).
- "NPolys" is the number of polygons generated by the subdivision algorithm.

SDNPatch Modelling Actions

This section, briefly, explains the special modelling actions defined for the SDNPatch custom object. All modelling actions can be started via the "Custom/SDNPatch" main menu.

Most of these actions require selected faces. To select a face, use the corresponding selection action first, see also section 3.25 [Selecting Faces](#) (page 104).

- "Face Extrude", new faces are inserted in the mesh at all edges of the selected faces, the selected faces themselves are displaced along their respective normals. This operation has two parameters¹ that control the offset of the extrusion operation (parameter "Length") and a scaling factor applied to the new displaced set of control points (parameter "Scale"). The length parameter may be negative, to revert the direction of the extrusion.
- "Face Remove", the selected face and its four neighbors are removed from the mesh. A new face is created that closes the hole left by the removed faces. This operation is the inverse of the face extrusion above.
- "Face Merge", the first two selected faces are removed from the mesh, the neighboring patches of the second face are connected to the neighboring faces of the first face. The decision, which vertices of the faces will actually be connected, depends on the relative vertex distances: i.e. the faces may need to be moved near to each other to make clear, how the connection shall be established.
- "Face Connect", the first two selected faces are removed from the mesh. The respective holes are then filled by creating a set of new faces, that connect the respective edges. The decision, between which vertices of the faces the new faces are built, depends on the relative vertex distances: i.e. the faces may need to be moved near to each other to make clear, how the connection shall be established.
- "Reset All Knots", set all knot values to 1.0 (the default).
- "Set Knot", set the knot value of the selected edge. In order to select a edge for this operation, just select all the control points defining the edge.
- "Revert", reverts all faces.
- "Merge", merges multiple SDNPatch objects into one new.
- "Import PLY", import a PLY file.
- "Export PLY", export the currently selected SDNPatch object to a PLY file.

In addition, there are two conversion operations that convert NURBS patch objects (or NURBS patch providing objects) and PolyMesh objects (or PolyMesh providing objects) to SDNPatch objects.

Note that the PolyMesh to SDNPatch conversion only accepts closed quadrilateral polygon meshes (triangles are omitted) and expects an optimized mesh (i.e. adjacent faces should share the same vertices).

¹ Since 1.17.

Creation Support

As the standard SDNPatch creation method only creates flat open patches there is another way to create more complex closed SDNPatch meshes. The menu entry "Create/Custom/SDNCube" allows to create a SDNPatch mesh in cube shape with desired number of patches per dimension.

Conversion Support

SDNPatch objects may be converted to PolyMesh objects using the main menu entry "Tools/Convert".

RIB Export

SDNPatch objects will be exported as `RiPointsGeneralPolygons` primitives (regardless of whether the actual configuration would fit into a simpler polygonal primitive of the RenderMan interface, e.g. a `RiGeneralPolygon`).

PV tags are currently not supported.

4.10 Standard Properties

Most Ayam objects have standard properties. They are used to control transformations and common attributes of objects. The following sections describe the standard properties "Transformations", "Attributes", "Material", "Shaders", and "Tags".

4.10.1 Transformations Property

Use the "Transformations" property to edit the location, orientation, and size of an object.

The corresponding property GUI contains the following elements:

- "Reset All!" immediately resets all transformation attributes to the default values.
- "Translate_X (_Y, _Z)" is the displacement of the object from the world origin in X (Y, Z) direction.
- "Rotate_X (_Y, _Z)" is the angle (in degrees) of the rotation of the object around the X (Y, Z) axis. See below for more information on how to use these entries.
- "Quaternion" the quaternion that is used to determine the orientation of the object in space.
- "Scale_X (_Y, _Z)" determines a scale factor that will be applied to the object in the direction of the local X (Y, Z) axis.

The transformations are applied to the object in the following order: Scale, Rotation, Translation.

How to use the rotation attributes?

The orientation of an object in space may be expressed using so called Euler angles. This notation (simply three angles determining a rotation about the principal axes of the coordinate system) suffers from a phenomenon called *gimbal lock*.

To avoid gimbal locks, Ayam internally holds the orientation of an object in a quaternion. This quaternion not only holds information about the angles but also about the order in which partial rotations occurred.

It is important to know, that the values of the angles of the rotation property must not be read in a way that the object will first be rotated around X by x-angle degrees then around Y by y-angle degrees then around Z by z-angle degrees. In fact, no information about the order in which partial rotations occurred may be derived from that three values. This implies, that e.g. the values 0 0 45 may actually denote a different orientation than the very same values 0 0 45.

Rotating an object is easy, simply *add* the amount about which the object shall be rotated to the value currently displayed in the appropriate entry.

For example, if an object is to be rotated 45 degrees about X and the x-angle entry displays a 30, enter 75. Then press the apply button.

If multiple entries are changed the rotations made will be in the order X (if changed) then Y (if changed) then Z (if changed). Do not change more than one entry at once unless you exactly know what you are doing.

Undoing a single rotation works in the same way, just use a subtraction instead of an addition.

Undoing all rotations (resetting the object to its original orientation) is simple too: enter 0 for all three entries at once, then press apply.

To copy just the orientation of an object to other objects using the property clipboard, all Translate and Scale property elements should be marked/selected via double clicks, then "Edit/Copy Property" can be used. Just marking the Rotate elements and then using "Edit/Copy Marked Prop" will *not* work, because then the quaternion will not be copied properly.

4.10.2 Attributes Property

The "Attributes" property of an object contains currently:

- "Objectname", the name of the object. It is also displayed in the object listbox or tree and may be written to RIB streams.
- "Hide", if this attribute is set this object is not drawn. It may also be excluded from RIB export.
- "HideChildren", if this attribute is set, the child objects of this object are not drawn. This attribute is e.g. used by "NPatch" objects to prevent the trim curves from being drawn in normal views.
- "RefCount", just displays how many objects point to this object e.g. through master-instance or object-material relationships. Objects with a reference count higher than zero may not be deleted.

4.10.3 Material Property

The "Material" property allows you to connect geometric objects to material objects (see also section 4.2.5 [Material Object \(page 125\)](#)). The material property GUI consist of the following elements:

- "Clear Material!" immediately clears any connection of the current object to its material.
- "Add/Edit Material!" adds a material to the current object (if it has none) and immediately selects the new material object for editing. If the current object already has a material, this material object is searched for and selected for editing.
- "Materialname" is the name of the material of this object. If the name is changed, the object will be disconnected from the old material and connected to the new material. An easier way to connect geometric objects to material objects is to simply drop the geometric objects onto the material object using drag and drop in the tree view.

4.10.4 Shader Properties

Shader properties are used to attach shaders of a certain type to objects. The name of the property contains the type of the shader, e.g. light shaders may be attached using a property named "LightShader" only. Other types of shaders or shader properties available are: "Surface", "Displacement", "Interior", "Exterior", "Atmosphere", and "Imager".

Each shader property GUI, even if no shader is attached to an object, starts with the "Set new shader."-button. This button allows to select a new shader of the appropriate type. If you press the "Set new shader."-button, a dialog with a list of shaders pops up. If this list is empty, Ayam is probably not set up properly (or you simply do not have shaders of the appropriate type). Check the preference setting "Main/Shaders". After a new shader has been set, the arguments of the shader will be parsed and a GUI will be generated to allow the arguments of the shader to be filled with values.

If the option "KeepParameters" is used, the parameter values of the previously set shader will be carried over to the new shader, if their types match.¹

¹ Since 1.34.

The `Delete shader.`-button may be used to delete the current shader from the selected object.

The `Default Values.`-button resets all arguments of the shader to the default values. See also section 4.10.4 [Working with Shaders \(page 254\)](#) below.

All other elements of the shader property GUI depend on the currently attached shader, i.e. they represent the arguments of the shader function.

Shader Parsing

Parsing a shader incorporates detecting the type of the shader and reading the names, types, and default values of all shader arguments.

Shaders will be parsed on the following occasions:

- application startup,
- "Scan for Shaders!" button in preferences dialog,
- "Special/Scan Shaders" menu entry,
- "ScanShaders" option in "Select Renderer" dialog,
- "Default Values" button in shader property GUI.

Note that currently, Ayam only works properly with shaders that have at most two dots in their file name and that Ayam will simply skip all array arguments (and emit a warning message) while parsing a shader. Those array arguments consequently never appear in the shader property GUIs and RIBs exported by Ayam. Also note that default values for shader arguments of type color will be silently clamped to the range 0-255.

Some shaders use array arguments to define transformation matrices. If this is the case and you have access to the shader source code you may want to modify those shaders to enable working with the transformation matrix carrying shader arguments. To do this, just change all definitions of transformation matrix carrying floating point arrays to real matrices. For instance, if the shader contains a

```
"float a_matrix_parameter[16]"
```

change this to

```
"matrix a_matrix_parameter".
```

Note that these changes of the shader argument definitions probably also require changes of the shader source code that uses those arguments. Ayam is able to deal with matrices because of their fixed size of 16 float values, and because libslcargs is able to deliver the default values for a matrix (but not for an array!).

If Ayam has been compiled without a shader parsing library (e.g. without libslcargs), Ayam will parse XML files created by `sl2xml` from the K-3D project (see ["http://www.k-3d.org/"](http://www.k-3d.org/)) instead of compiled shaders. The `Set new shader.`-button will in this case always open a file requester, allowing you to select a XML file, that has been created by `sl2xml`. Furthermore, the `Default Values.`-button will not be available; you have to use `Set new shader.` instead.

From version 1.3 on, Ayam also supports shader parsing plugins to allow parsing of shaders compiled with different shader compilers, see also section 8.8 [Shader Parsing Plugins \(page 529\)](#).

Finally, since version 1.25, there is a script that parses shader source code, see also section 6.5.4 [Shader Parsing \(page 448\)](#).

Working with Shaders

The "Default Values."-button resets all arguments of the shader to the default values. Additionally, the compiled shader will be parsed again and the property GUI will be adapted (new shader arguments will appear, removed shader arguments will disappear). Therefore, this button is quite handy when dealing with changing shaders: just edit the shader, recompile it, then back in Ayam just hit the "Default Values."-button. However, note that this destroys any possibly carefully adjusted shader argument values.

To keep these old shader argument values select them using double-clicks on the parameter names in the shader property GUI and then use e.g. "Edit/Copy Marked Prop" (see also the description of the property clipboard in section [2.1.2 Properties \(page 24\)](#)). After loading the default values, the saved argument values can then be re-established using "Edit/Paste Property". Care should be taken to only copy values with matching types.

4.10.5 Caps Property

Many tool object have a "Caps" property to easily close the created surface.

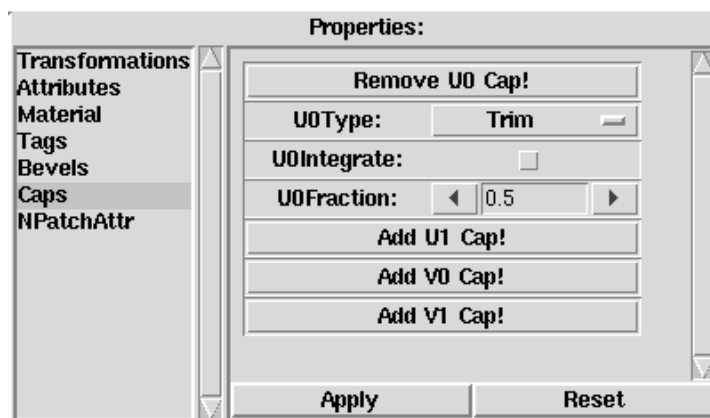


Figure 114: Caps Property of NPatch Object

The exact names of the attributes available in this property are object type specific as each object types boundary names are different: the boundary names of a Skin object are e.g. "Left", "Right", "Start", and "End", whereas the boundary names of a NPatch object are "U0", "U1", "V0", and "V1" (see also the image above). When the mouse cursor hovers over any of the Add/Remove buttons, the corresponding surface boundary is flashing in all views to help decide where to put the cap.¹

To reduce GUI clutter, initially, the Caps property only contains buttons that allow to enable/disable the associated cap and control appearance of further options.

For each enabled cap a menu button will appear that allows to set the corresponding cap type (see section [4.7.10 CapAttr Property \(page 212\)](#) for more information on the available cap types).

In addition, the Caps property allows to create caps that integrate with the progenitor surface of the corresponding tool object (after the potential integration of bevels) via the additional parameter "Integrate". However, integration is only supported for the cap types "Simple" and "Simple3D". If integration is enabled and there is a bevel that is not integrating with the progenitor surface, the cap will integrate into the bevel instead. Integration will change the number of provided objects and the parameters of the progenitor surface.

Similar to Cap objects, MP tags can be set to object to control the middle point in both simple modes (see also section [4.11.21 MP \(Mean Point\) Tag \(page 270\)](#)).

¹ Since 1.32.

4.10.6 Bevels Property

Many tool object have a "Bevels" property to easily round the otherwise sharp borders of the created surface.

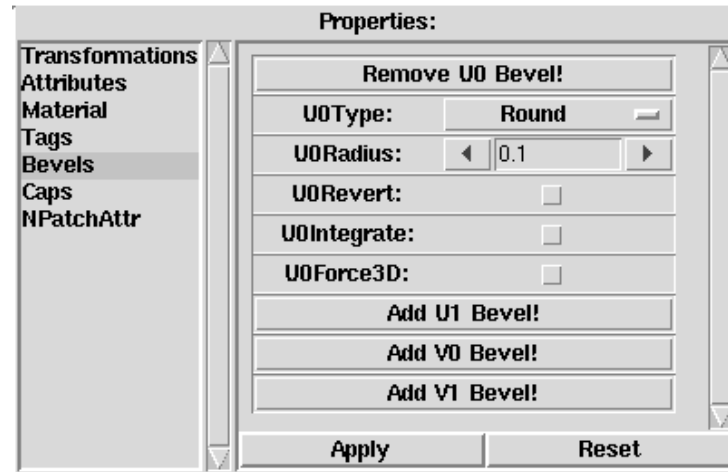


Figure 115: Bevels Property of NPatch Object

The exact names of the attributes available in this property are object type specific as each object types boundary names are different: the boundary names of a Skin object are e.g. "Left", "Right", "Start", and "End", whereas the boundary names of a NPatch object are "U0", "U1", "V0", and "V1" (see also the image above). When the mouse cursor hovers over any of the Add/Remove buttons, the corresponding surface boundary is flashing in all views to help decide where to put the bevel.¹

To reduce GUI clutter, initially, the Bevels property only contains buttons that allow to enable/disable the associated bevel and control appearance of further options.

Many attributes of the Bevels property are also available for the Bevel object (see also [4.7.9 BevelAttr Property \(page 209\)](#)). In addition to those attributes, the Bevels property allows to integrate the created bevel surface with the progenitor surface of the corresponding tool object via the additional parameter "Integrate". Integration will change the number of provided objects and the parameters of the progenitor surface.

¹ Since 1.32.

4.10.7 Tags Property

Use the "Tags" property to edit the tags of an object. See also the image below.



Figure 116: Tags Property GUI with Context Menu

All tag entries have a context menu, where the corresponding tag can be copied/added to the property clipboard.¹

Long and multiline tag values will be displayed in abbreviated fashion (with an ellipsis – . . .).

The tags property GUI also consists of the following standard elements (see also above image):

- "Remove all Tags!" immediately removes all tags from the object.
- "Remove Tag!" is a menu, that allows to quickly select and remove a single tag from the object.
- "Add Tag!" opens a small dialog window, where a new tag type and value may be entered (see also the image below). The tag type line has a default value menu, containing the most important tag types.²

After pressing the "Ok" button, a new entry will be added to the tags property, displaying the new tag. Just click on the entry to get back to the dialog, to remove the tag using "Clear" then "Ok", or to change the type or value of the tag.

The tag edit dialog supports multiline tag editing.³ Just enlarge the dialog window to switch to multiline editing. Another way to control multiline mode is to use the keyboard shortcuts <Alt-Down> and <Alt-Up> respectively. Note that in multiline mode, the <Enter>, <Return>, and also <Tab> key will behave differently.

The tag value widget also has a context menu, allowing text to be fetched from the system clipboard or files.



Figure 117: Add/Edit Tag Dialog

The next sub-sections describe the tag types currently available in Ayam and the plugins distributed with Ayam. Note that foreign extensions and plugins may define their own types.

¹ Since 1.21. ² Since 1.24. ³ Since 1.24.

4.11 Tags

Tags provide an easy way to attach arbitrary information (e.g. additional RenderMan interface attributes, special attributes for plugins, or even scripts) to objects. A tag consists of two strings, one defining the type and one defining the value of the tag.

Tags may be manipulated via the tags property GUI (see section 4.10.7 Tags Property (page 257)) or the scripting interface (see section 6.2.12 Manipulating Tags (page 388)). In addition, there is a corresponding sub menu in the main menu (see section 2.2 Tags Menu (page 39)).

The following two tables contain a compact list of tag names and short explanations for all tag types that are known in Ayam and in all accompanying extensions (plugins); tag types marked with an asterisk (*) are for internal use only.

Name	Description
ANS	After Notify Script
AsWire	X3D export control
BNS	Before Notify Script
BP	Bevel Parameter
CP	Cap Parameter
CIDR	Importance Driven Rendering
CCIDR	Importance Driven Rendering
DANS	Disabled After Notify Script
DBNS	Disabled Before Notify Script
DC	Depth Complexity
HC *	Has Children
IDR	Importance Driven Rendering
IIDR	Importance Driven Rendering
MI *	Material ID
MN	Mean Normal
mn *	master name
MP	Mean Point
NC *	Notify Count
NM *	Notify Master
NO *	Notify Object
NoExport	Export Control
NP	New Property
NT	Normals Tangents

Table 95: Tags Overview (1/2), * – internal

Name	Description
OI *	Object ID
PV	Primitive Variable
RiAttribute	RenderMan Export
RiDisplay	RenderMan Export
R3IDR	Importance Driven Rendering
RIDR	Importance Driven Rendering
RiHider	RenderMan Export
RiOption	RenderMan Export
RP	Remove Property
SaveMainGeom	Geometry Management
SavePaneLayout	Geometry Management
SB	Selected Boundary
SP	Selected Points
TC	Texture Coordinates
TI *	Texture ID
TM *	Transformation Matrix
TP	Tessellation Parameters
UMM	U Min Max
VMM	V Min Max
XML	XML Data

Table 96: Tags Overview (2/2), * – internal

4.11.1 RiAttribute Tag

The tag type "RiAttribute" can be used to attach arbitrary RenderMan interface attributes to objects. This is handy if a renderer with lots of RiAttributes that differ from the standard RiAttributes is in use.

"RiAttribute" tags attached to a geometric object override "RiAttribute" tags possibly attached to the material object of this geometric object.

In order to create a tag of type RiAttribute, the type string must be "RiAttribute". The syntax of the value string is as following:

```
<attrname>, <paramname>, <paramtype>, <param>
```

where

<attrname> is the name of the attribute (e.g. "render");

<paramname> is the name of the parameter (e.g. "displacementbound");

<paramtype> is a single character defining the type of the parameter (it may be one of f – float, g – float pair, i – integer, j – integer pair, s – string, c – color, p – point); and finally

<param> is the value of the parameter itself (e.g. a float: "1.2", an integer value: "3", a string: "on", a color: "1,1,1" or a point: "0.4,0.5,1.0").

Example

Some examples for valid RiAttribute tags:

```

RiAttribute
render, truedisplacement, i, 1

RiAttribute
dice, numprobes, j, 3, 3

RiAttribute
radiosity, specularcolor, c, 0.5, 0.5, 0.5

```

Notes

The "RiAttribute" tag handles just a single parameter at once. Also note that "RiAttribute" tags may be created much more easily using the menu entry "Special/Tags/Add RiAttribute". The database of RiAttributes for this GUI may be extended by editing the *ayamrc* file, see section 8.4 [Ayamrc File](#) (page 516).

4.11.2 RiOption Tag

The tag type "RiOption" can be used to attach arbitrary RenderMan interface options to the scene. This is handy if a renderer with lots of RiOptions that differ from the standard RiOptions is in use. However, they will be only used by the RIB exporter if they are attached to the "Root" object. The syntax is similar to the "RiAttribute" tag type, see above.

Example

```

RiOption
radiosity, steps, i, 16

RiOption
shadow, bias0, f, 0.01

```

Notes

RiOption tags may be created easily using the menu entry "Special/Tags/Add RiOption". Tags created with this GUI will always be added to the "Root" object. It does not have to be selected when the GUI is used. Furthermore, the database of RiOptions for this GUI may be extended by editing the *ayamrc* file, see section 8.4 [Ayamrc File](#) (page 516).

4.11.3 TC (Texture Coordinates) Tag

The tag type "TC" can be used to attach texture coordinates to objects or materials.

The "TC" tag always contains a list of eight comma separated float values, that specify a mapping for four 2D points (a quadrilateral) in texture space from the default values (0,0), (1,0), (0,1), and (1,1) to the new specified values.

Example

```

TC
0, 0, 10, 0, 0, 10, 10, 10

```

Changes the texture coordinate space so that more and smaller tiles of a texture would be displayed on a primitive.

```
TC
0,0,0,1,1,0,1,1
```

Flips the texture coordinate space over two corners. A shader normally generating vertical stripes will create horizontal stripes now.

```
TC
0,1,0,0,1,1,1,0
```

Turns the texture coordinate space by 90 degrees. A shader normally generating vertical stripes will create horizontal stripes now.

Notes

"TC" tags attached to a geometric object override "TC" tags possibly attached to the material object of this geometric object.

The exact behaviour of an object equipped with a "TC" tag depends heavily on the shader and its use of the texture coordinates.

Note also that using "TC" tags, the texture coordinates of entire primitives are changed. To change the texture coordinates of sub-primitives (e.g. of single control points of a NURBS patch) "PV" (Primitive Variable) tags must be used instead.

To ease setting of "TC" tag values Ayam provides a special graphical editor as outlined below.

The texture coordinate editor may be opened using the main menu entry "Special/Tags/Edit TexCoords" and lets you edit texture coordinate tags in an intuitive way.

For that, the current texture coordinates are displayed as a black polygon in a canvas with regard to the original (default) values, that are displayed in gray. Small arrows point to positive s and t direction respectively.

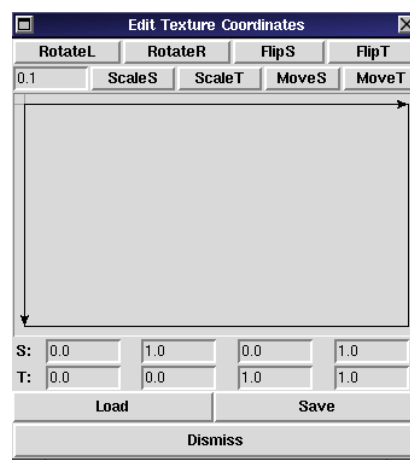


Figure 118: Texture Coordinate Editor

The "RotateR" and "RotateL" buttons shift the coordinate values between the four points. This results in a 90 degree rotation of the texture space.

The "FlipS" and "FlipT" buttons flip the texture coordinate values in s and t direction respectively. This is useful, if, for example, a texture mapping shall be corrected for an image that appears upside down.

The next buttons allow to move (using "MoveS" and "MoveT") and scale (using "ScaleS" and "ScaleT") the texture coordinates by a specific amount that is given in the first entry field.

The "Load" and "Save" menu buttons allow to:

- load the default texture coordinate values ((0,0), (1,0), (0,1), (1,1)),
- load texture coordinates from a selected BPatch object: The xy coordinates of the four points of the selected BPatch will be interpreted as st coordinates. This allows for more complex transformations of the texture coordinates e.g. rotations about an angle of 45 degrees. For that just create a BPatch object, rotate it accordingly, then load the coordinates into the texture coordinate editor.
- load TC tags from the selected object,
- save the texture coordinates to a BPatch object,
- save TC tags to a selected object. Note that it is not possible to directly save the TC tag to multiple selected objects. But the property clipboard can be used to copy the tag after saving to a single object.

Note that the tag numbers in the menu entries count TC tags only.

The texture coordinate dialog is mode-less, it may stay open while modeling.

4.11.4 PV (Primitive Variable) Tag

The tag type "PV" can be used to attach arbitrary data to geometric primitives and even sub-primitives. With the help of primitive variables texture coordinates can be attached to the vertices of a NURBS patch primitive or distinct colors to the faces or even to single vertices of a polygonal mesh. In the latter case, the data is properly interpolated by the RenderMan renderer before it is handed over to the surface shader.

When rendering, all data defined in a "PV" tag is handed over to the surface shader that is attached to the respective geometric primitive using additional shader parameters. For RIB export, proper "RiDeclare" statements will be created automatically by Ayam.

However, Ayam does not check, whether the shaders actually use the data from the "PV" tag.

The syntax of the value string of a PV tag is as following:

```
<name>,<detail>,<type>,<ndata>,<data>
```

where

<name> is the name of the primitive variable;

<detail> (or storage class) should be one of "uniform", "varying", "vertex", or "constant";

<type> is a single character describing the type of the data (one of "c" (color), "f" (float), "g" (float[2]), "n" (normal), "p" (point), "s" (string), or "v" (vector), see also the documentation of the "RiAttribute" tag above);

<ndata> is an integer number describing how many data elements will follow; and

<data> is a comma separated list consisting of <ndata> elements of type <type>.

Examples

```
PV
mycolor,constant,c,1,0,1,0
```

adds a single color value (0,1,0), which is the same all over the primitive, the respective surface shader should have a parameter "color mycolor";

```
PV
mys,varying,f,4,0.1,0.2,0.3,0.4
```

could be used to add a distinct float value to each corner point of a four point NURBS patch (of order, width, and height 2), the respective surface shader should have a parameter "varying float mys".

Notes

The following data types are *not* supported: "i", "j". Support for the data types "n" (normal), and "v" (vector) was added in Ayam 1.17.

Not all geometric objects currently honour PV tags on RIB export. The geometric objects currently supporting PV tags are: SDMesh, PolyMesh, PatchMesh, NPatch, and BPatch. Most tool objects that internally create NPatch objects also support PV tags.¹ Mind that the same set of tags will be used for all surfaces that make up the tool object, e.g. the swept surface, its bevels, and its caps.

Furthermore, the number of data elements, which depends on the detail or storage class, the type of geometric primitive, and the configuration of the geometric primitive is *not* checked by Ayam. Some RIB writing libraries, however, do verify the numbers and silently omit the primitive variable if there are mismatches. The RIB should be examined for the presence of the primitive variable after export, especially, if PV tags were added or edited manually.

4.11.5 RiHider Tag

The tag type "RiHider" can be used to choose and parameterise different algorithms for hidden surface removal when rendering the exported scene with a RenderMan compliant renderer. RiHider tags have to be attached to the root object in order to be used. The syntax of a RiHider tag is quite similar to a Ri-Attribute tag: "<type>, <parameterlist>" where "<type>" is the name of an algorithm and "<parameterlist>" is a comma separated list of triplets consisting of name, type, and value of a parameter.

Example

A RiHider tag could look like this:

```
RiHider
hidden,depthfilter,s,midpoint
```

4.11.6 RiDisplay Tag

The tag type "RiDisplay" can be used to add output files of different type (e.g. containing depth-buffer information) to the scene or to directly control the output format when rendering the exported scene with a RenderMan compliant renderer. RiDisplay tags have to be attached to the Root object in order to be used.

¹ Since 1.20.

The syntax of a RiDisplay tag is as follows: "<name>, <type>, <mode>, <parameterlist>", where

"<name>" is a file or device name,

"<type>" specifies the destination of the image data (e.g. screen or file),

"<mode>" specifies which information should be stored or displayed (e.g. color values: rgb, or depth values: z), and

"<parameterlist>" is a optionally present comma separated list of triplets consisting of name, type, and value of a parameter.

Example

A RiDisplay tag to add output of the depth-buffer information to the file "imagez.tif" could look like this:

```
RiDisplay
imagez.tif, file, z
```

Notes

The name will be automatically changed to "+name" on RIB export if it does not already start with a plus (except for the very first RiDisplay statement).

4.11.7 AsWire Tag

The tag type "AsWire" switches export of certain objects from surface to wire-frame mode. The value string of this tag is ignored. All that counts is the presence of the tag. Note that only X3D export honours this tag.

4.11.8 NoExport Tag

The tag type "NoExport" can be used to exclude certain objects from exported RIBs. The value string of this tag is ignored. All that counts is the presence of the tag. Child objects of objects with the "NoExport" tag will also be excluded from the RIB. Since Ayam 1.6, light objects also honour the "NoExport" tag. Note that regardless of potentially present "NoExport" tags, RIB archives will be created for all referenced objects all the time (even if "NoExport" tags are added to all instances).

4.11.9 SaveMainGeom Tag

The tag type "SaveMainGeom" can be used to save the geometry of the main window and the toolbox window (if open) to a scene file. For that the scene saving code checks for the presence of a "SaveMainGeom" tag for the Root object and fills it with the current geometry information. The scene reading code checks for the presence of a "SaveMainGeom" tag for the Root object after replacing a scene and re-establishes the geometries of main and toolbox window.

This tag is only filled with meaningful data and the geometry is only restored if Ayam is in multi window GUI mode.

4.11.10 SavePaneLayout Tag

The tag type "SavePaneLayout" can be used to save the relative sizes of the internal windows of the main window when Ayam runs in the single window GUI mode to a scene file. For that the scene saving code checks for the presence of a "SavePaneLayout" tag for the Root object and fills it with the current geometry information. The scene reading code checks for the presence of a "SavePaneLayout" tag for the Root object after replacing a scene and re-establishes the geometries of the internal windows.

This tag is only filled with meaningful data and the pane layout is only restored if Ayam is in single window GUI mode.

4.11.11 TP (Tessellation Parameter) Tag

The tag type "TP" can be used to save tessellation parameters to objects of type "NPatch" (and objects that may be converted to "NPatch" objects). Those tessellation parameters will be used when the NPatch object is tessellated for e.g. a conversion to a PolyMesh object. The syntax of the TP tag is: "<smethod>, <sparamu>, <sparamv>[, <refinetrim>]" where <smethod> is an integer value between 1 and 6, describing which sampling method to use (1 – ParametricError, 2 – PathLength, 3 – DomainDistance, 4 – NormalizedDomainDistance, 5 – AdaptiveDomainDistance, and 6 – AdaptiveKnotDistance) and <sparamu> and <sparamv> are float values describing the respective parameter value for the chosen sampling method. The second parameter value is ignored for the sampling methods 1 and 2.

The last value, "refinetrim", is an integer between 0 and 5 controlling how many times the trim curves are to be refined before tessellation for improved tessellation fidelity along trim edges. The "refinetrim" value may be omitted and defaults to 0.

Note that the syntax of the "TP" tag changed in Ayam 1.9, the old syntax only allowed one parameter.

TP tags may be easily created using the tessellation GUI, that can be started with the main menu entry "Tools/Surface/Tessellate" (see also section 5.6.5 Tessellation Tool (page 337)).

Example

A TP tag could look like this:

```
TP
1, 0.5, 0.6
```

4.11.12 DC (Depth Complexity) Tag

The tag type "DC" is only used by the AyCSG CSG preview plugin to store the depth complexity of CSG primitives. The syntax of the DC tag is: "<dcval>" where "<dcval>" is a positive integer value describing the depth complexity of the CSG primitive. See also section 8.12 CSG preview using the AyCSG plugin (page 532) for more information regarding the depth complexity value.

Example

A DC tag (valid for e.g. a torus) could look like this:

```
DC
2
```

4.11.13 NP (New Property) Tag

The tag type "NP" (new property) may be used to add new property GUIs to single objects. The value of the tag is the name of a new property. The necessary code to manage the property data and the windows that make up the property GUI itself have to be present in the Tcl context of Ayam before the user clicks on the new property in the property list box.

Example

```
NP
Transformations
```

This tag can e.g. be added to Script objects that create objects.

Any script controlled property GUI of a Script object must also be enabled using a NP tag like this:

```
NP
MyProperty
```

Notes

NP tags can be managed more easily using a specialized dialog that is available in the main menu "Special/Tags/Add Property".

4.11.14 RP (Remove Property) Tag

The tag type "RP" (remove property) may be used to remove GUI access to a property from single objects. The value of the tag is the name of the property to be removed. The GUI access will be blocked by simply omitting the property from the property listbox. Note well: the property is still present and active in the objects themselves and values may still be set using the scripting interface.

Example

```
RP
Script
```

removes direct access to the Script property of a Script object. Ideally, the Script object also has a "NP" tag, to allow direct control of script parameters. This way, the user does not see the script (code), just a clean parameter GUI.

Here is another example: the tag

```
RP
Transformations
```

can be added to Instance objects to turn them into references (instances without own transformation attributes).

Notes

RP tags can be managed more easily using a specialized dialog that is available in the main menu "Special/Tags/Remove Property".

4.11.15 BNS (Before Notify Script) Tag

The tag type "BNS" (before notify script) may be used to add scripts to an object, that will be run *before* the notification of that object starts. A notification, in turn, will be executed because e.g. one of the children of the object changed (see also section 8.2 [The Modelling Concept Tool-Objects \(page 511\)](#)). When the script runs, the respective object with the BNS tag will already be selected. Current level and selection will be restored when the script finishes, i.e. neither needs to be done by the script.

Example

A simple BNS tag could look like this:

```
BNS
puts "notify callback about to fire"
```

A more useful example (that is also available as example scene "ayam/scn/scripts/bnstag.ay"):

```
BNS
getProp;
set ::RevolveAttrData(Sections) [expr int($::RevolveAttrData(ThetaMax)/10)];
setProp
```

This tag computes the number of sections from the ThetaMax value of a Revolve object. Whenever the ThetaMax is changed, the number of sections adapt, so that each section always spans the same angle. The function call `int()` is needed because the C code for the `setProp` command of the `RevolveAttr` property only looks for integer data in the variable `Sections` and without `int()` the variable would contain incompatible floating point data.

A second similar example would be:

```
BNS
getProp;
set ::ACurveAttrData(ALength) [expr int($::ACurveAttrData(Length)/2)];
setProp
```

This tag automatically derives a good value for the `ALength` parameter of a `ACurve` object from the number of data points to approximate. When this tag is added to a `ACurve` object, the interactive insert/delete point actions can be used more freely.

Here is another shorter example; the BNS tag

```
BNS
applyTrafo -all
```

makes sure, that a `NCurve` or `NPatch` object always has the default transformations, as any changes to the transformation attributes will immediately be applied to the control points and the transformation attributes will be reset in the progress.

Notes

In Ayam versions prior to 1.16 BNS tag scripts could use any functionality of Tcl, Tk, and the Tcl scripting interface of Ayam which posed a huge security risk. This is no longer the case. BNS tag scripts now run

in a safe interpreter with reduced instruction set. They can no longer write to the file system, get onto the network, or confuse the application state, see also section [4.9.1 Safe Interpreter \(page 233\)](#). Consequently, the warning dialog that appeared when files with BNS tags were loaded is also gone.

The original functionality can still be re-enabled by recompiling Ayam. If this is enabled and scene files containing BNS tags are loaded, Ayam will again raise a warning dialog, offering to temporarily disable all such tags that will be read for obvious security reasons. To disable a BNS tag, Ayam simply changes its type from "BNS" to "DBNS" (disabled before notify script). It will not be executed then. Disabled notify script tags may be enabled after careful inspection by simply changing their type back to "BNS" or by using the main menu entry "Special/Enable Scripts".

If the script of a BNS tag fails, the tag will be disabled by changing the type to DBNS.¹

Also note that BNS tag scripts must not change the scene hierarchy, i.e. they must not create or delete objects or use the object clipboard.

4.11.16 ANS (After Notify Script) Tag

The tag type "ANS" (after notify script) may be used to add scripts to an object, that will be run *after* the notification of that object completed. The notification, in turn, will be executed because e.g. one of the children of the object changed (see also section [8.2 The Modelling Concept Tool-Objects \(page 511\)](#)). When the script runs, the respective object with the ANS tag will already be selected. Current level and selection will be restored when the script finishes, i.e. neither needs to be done by the script.

Example

A simple ANS tag could look like this:

```
ANS
puts "notify callback completed"
```

Notes

In Ayam versions prior to 1.16 ANS tag scripts could use any functionality of Tcl, Tk, and the Tcl scripting interface of Ayam which posed a huge security risk. This is no longer the case. ANS tag scripts now run in a safe interpreter with reduced instruction set. They can no longer write to the file system, get onto the network, or confuse the application state, see also section [4.9.1 Safe Interpreter \(page 233\)](#). Consequently, the warning dialog that appeared when files with ANS tags were loaded is also gone.

The original functionality can still be re-enabled by recompiling Ayam. If this is enabled and scene files containing ANS tags are loaded, Ayam will again raise a warning dialog, offering to temporarily disable all such tags that will be read for obvious security reasons. To disable a ANS tag, Ayam simply changes its type from "ANS" to "DANS" (disabled after notify script). It will not be executed then. Disabled notify script tags may be enabled after careful inspection by simply changing their type back to "ANS" or by using the main menu entry "Special/Enable Scripts".

If the script of a ANS tag fails, the tag will be disabled by changing the type to DANS.²

Also note that ANS tag scripts must not change the scene hierarchy, i.e. they must not create or delete objects or use the object clipboard.

¹ Since 1.24. ² Since 1.24.

4.11.17 UMM/VMM (U/V Min Max) Tag

The tag types "UMM" (u min max) and "VMM" (v min max) may be used to store additional parametric domain trimming values to NURBS curve and NURBS patch objects. Note that the GLU NURBS display modes do not honor those tags, but some export plugins do (RIB, Wavefront OBJ, 3DM (Rhino)). Those tags will also be created by RIB import or Wavefront OBJ import.

Example

An UMM tag could look like this:

```
UMM
0.4, 0.6
```

4.11.18 BP (Bevel Parameters) Tag

The tag type "BP" (bevel parameters) is used by all bevel supporting tool objects to store their bevel parameter information. See also sections [4.10.6 Bevels property \(page 256\)](#) and [4.7.9 BevelAttr Property \(page 209\)](#).

The syntax of the BP tag is:

```
"<side>,<type>,<radius>,<revert>"
```

where

<side> is an integer value from 0 - 3 defining the side of the surface, to add the bevel to,

<type> is an integer value defining the type of the bevel,

<radius> is a floating point value defining the radius of the bevel, and

<revert> is either 0 or 1 and may be used to revert the bevel.

Example

A BP tag could look like this:

```
BP
0,0,0.1,0
```

4.11.19 CP (Cap Parameters) Tag

The tag type "CP" (cap parameters) is used by all cap supporting tool objects to store their cap parameter information. See also sections [4.10.5 Caps property \(page 255\)](#) and [4.7.10 CapAttr Property \(page 212\)](#).

The syntax of the CP tag is:

```
"<side>,<type>,<integrate>,<fraction>"
```

where

<side> is an integer value from 0 - 3 defining the side of the surface, to add the cap to,

<type> is an integer value from 0 - 3 defining the type of the cap,

<integrate> determines whether to integrate the cap into the progenitor surface (0 or 1)

<fraction> a floating point parameter value for the Simple3D cap type.

Example

A CP tag could look like this:

```
CP
0, 0, 0, 0.5
```

4.11.20 MN (Mean Normal) Tag

The tag type "MN" (mean normal) can be added to the Bevel object to control the mean/target normal of the "RoundToNormal" bevel mode. The tag value is a list of three comma separated floating point numbers, a 3D vector. The vector does not have to be normalized.

Example

A MN tag could look like this:

```
MN
0.0, 1.0, 0.1
```

4.11.21 MP (Mean Point) Tag

The tag type "MP" (mean point) can be added to the Cap object or objects with caps to control the middle point of the "Simple" and "Simple3D" caps. This is useful if e.g. the automatically calculated mean point of the parameter curve is off. The tag value is a list of three comma separated floating point numbers, a 3D vector, followed by an integer value between 0 and 3 designating the cap. On Cap objects, the cap number may be omitted.

Example

A MP tag could look like this:

```
MP
0.5, 0.25, 0.1, 0
```

4.11.22 SB (Selected Boundary) Tag

The tag type "SB" (selected boundary) is used by the select boundary action(s) to store their results, but it can also be created manually or by other means. The tag supports two notations, a simple one, consisting of an integer value and a complex one, where an object index and a boundary index are provided. The latter needs to be specified for tool objects that provide multiple NPatch objects.

Examples

A SB tag in simple notation could look like this:

```
SB
1
```

and in complex notation:

```
SB
o:2b:4
```

4.11.23 XML Tag

The tag type "XML" can be used to enrich XML based export formats with arbitrary user defined data (attributes and nodes).¹ It can currently only be added to Material and NURBS surface objects for X3D export (see also section 7.15 X3D (Web3D) Export (page 509)). The value string of this tag must be a valid XML node hierarchy where the outer node specifies the target node of the XML data. Attributes of the outer node will be added to the target XML node for export, already existing attributes will be overwritten (i.e. the XML data tag will be processed after the XML export converted the corresponding Ayam object to a XML node hierarchy). Likewise, all child nodes of the outer node will be added to the matching target XML node. CDATA sections are also supported properly (this is important for GLSL shader source code).

Examples

XML tags could look like this:

```
XML
<Material shininess='0.7' />
```

to add the shininess attribute to a Material.

```
XML
<Appearance>
<ComposedShader language='GLSL'>
...
</ComposedShader>
</Appearance>
```

to add a GLSL shader to a Material.

4.11.24 Internal Tags

The following tags are used by Ayam internally only; they will *not* appear in the tags property GUI and they can *not* be deleted or changed using the scripting interface.

- **OI (Object ID) Tag**

This tag is used by the RIB exporter and the scene storage facility to establish links between instance objects and the master objects they are pointing to.

- **MI (Material ID) Tag**

This tag is used by the RIB exporter and the scene storage facility to establish links between material objects and the objects they are assigned to.

- **HC (Has Children) Tag**

This tag is used by the scene storage facility.

- **TM Tag**

Internal binary tag to store transformation matrices, used by the AyCSG plugin.

- **mdn Tag**

Internal tag used by the X3D import/export plugin. Renamed from mn since Ayam 1.26.

¹ Since 1.24.

- NC Tag
Internal tag used by the notification mechanism.
- NO/NM Tags
These tags are internal binary tags that transport the notification across the scene.
- NT Tag
This tag is an internal binary tag that carries normals and tangents for curves extracted from surfaces.

5 NURBS Modelling Tools

This section describes NURBS curve and surface related modelling tools of Ayam.

5.1 General Remarks

All NURBS modelling tools are accessible via the "Tools" menu of the main window or the toolbox. In addition, there are also corresponding scripting interface commands. See sections [6.2.15 Manipulating NURBS Curves \(page 393\)](#) and [6.2.16 Manipulating NURBS Surfaces \(page 400\)](#).

Many modifying tools work on multiple selected objects and execute the selected operation on all those selected objects in the order of their appearance in the current level.

Tools that take only NURBS curves or only NURBS surfaces from the selection will warn if the selection contains objects of unsuitable type, but processing will continue regardless.

In case of an error, however, the processing of multiple selected objects immediately stops, possibly leaving modified *and* unmodified objects behind.

If an operation executed successfully on an object, the point selection may be removed from the object.

The notification of the selected objects will be run immediately after the tool operation. The notification of the parent object(s), however, will be run after processing of all selected objects finished (for more information on notification, see also section [8.2 The Modelling Concept Tool-Objects \(page 511\)](#)).

Tools that create tool objects may use the clipboard to move the new children around, i.e. the clipboard contents may be lost afterwards.

Most tools open a dialog window where additional operation parameters can be adjusted, see also the figure below.

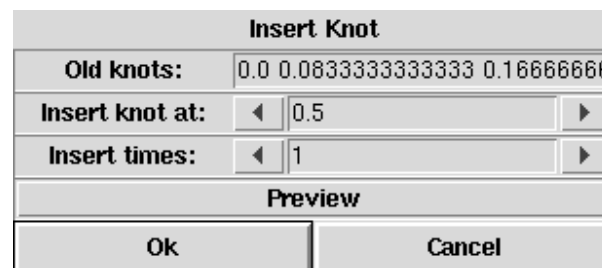


Figure 119: Tool Dialog Example

The "Ok"-button starts the tool and "Cancel" just closes the dialog. Furthermore, the currently set parameters can be tried using the "Preview" button.¹ A successful preview will be signaled by a green tick mark on the button, a failed preview (e.g. caused by unsuitable parameters) by a red exclamation mark. Note, that the meaning of the "Ok"- and "Cancel"-buttons change, when a preview was generated (green tick mark present). In this case, "Ok" simply closes the dialog and "Cancel" restores the selected objects to the state before the dialog opened. Also note, that repeated use of preview just starts the tool again on the selected objects in their original state.

If the tool parameters are changed, after a successful preview was generated, the green tick mark will change to a question mark.²

¹ Since 1.30. ² Since 1.35.

5.2 Curve Creation Tools

These tools create parametric curve objects.

5.2.1 Circular B-Spline Tool

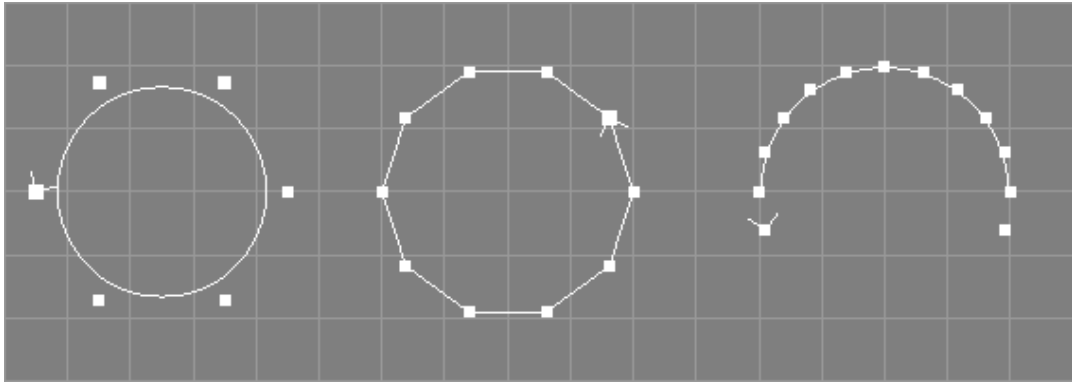


Figure 120: B-Spline Curves Created by the Circular B-Spline Tool

Arguments

Radius, Arc, Sections, Order.

Operation

This tool creates a non-rational B-Spline curve with $\text{Sections}+1+\text{Order}$ ($\text{Arc}=360.0$) or $\text{Sections}+1+(\text{Order}/2)$ (other Arc values) control points in the XY-plane. The control points are arranged in a circle of the given radius, centered around the origin. This gives the curve a circular appearance (see image above) but it is *not* a true circle: If only few control points are used, the radius of the circular curve is clearly smaller than the specified radius value (see the left curve in the image above). Furthermore, shape, parameterisation, and curvature of the B-Spline curve are not exactly as one would expect from a circle. To create true circular curves, the NURBCircle tool (see below) should be used instead.

Notes

Sections must be at least 1.

If Arc is 360.0, the first n control points of the new curve will be identical to the last n (where n is $\text{Order}-1$). Compare the left and middle curves in the image above which are of order 4 and 2 respectively, the first having 3 and the latter just having $2-1=1$ equal control points. If Arc is 360.0, the curve will also be marked periodic and the generation of multiple points will be enabled, so that point edit actions know that they may need to move two points (see also section 4.4.1 [Multiple Points](#) (page 153)).

If Arc is smaller than 360.0, the curve will be open, and, as is natural for a B-Spline curve, will not interpolate the first and last control points unless the order is 2 (see the right curve in the image above, which was created with an arc value of 180.0, 10 sections, and order 4).

5.2.2 NURBCircle Tool

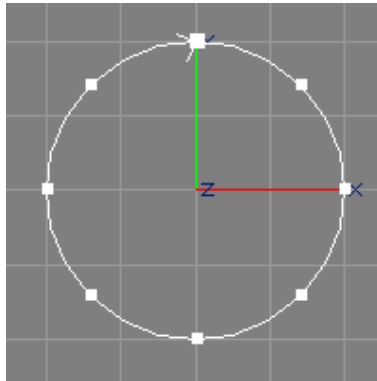


Figure 121: A NURBS Circle

Arguments

Radius, Arc.

Operation

The NURBCircle tool creates a circular NURBS curve of desired radius and arc in the XY-plane, centered around the origin (see also the image above). The order of the curve will be 3. The number of control points used differs according to the arc, e.g. 9 points for full circles, 5 for half circles, 3 for a quarter circle.

Notes

The NURBS curve created by the NURBCircle tool is rational (uses weights). This means, editing the curve (e.g. moving control points) may lead to unpredicted results (the curve does not behave exactly as wished). If the curve is to be modified further, a closed B-Spline, created with the Circular B-Spline tool (see above), should be used instead. Additionally, the created curve will be marked as closed and the generation of multiple points will be enabled, so that point edit actions know that they may need to move two points. See also section [4.4.1 Multiple Points \(page 153\)](#).

A NURBS circle created by this tool can be used to easily create a NURBS torus by moving the circle along X a bit and then revolving it. The amount of movement determines the radius of the torus, whereas the radius of the circle determines the thickness.

5.2.3 Rectangle Tool

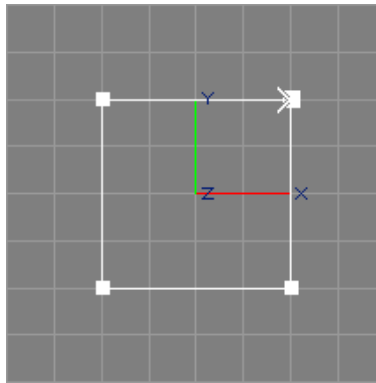


Figure 122: A Rectangle

Arguments

Width, Height

Operation

The rectangle tool creates a non-rational, piecewise linear, planar, centered NURBS curve of rectangular shape and twice the specified width and height in the XY-plane (see also the image above).

Notes

The created curve will be marked as closed and the generation of multiple points will be enabled, so that point edit actions know that they may need to move two points. See also section [4.4.1 Multiple Points](#) (page 153).

5.2.4 TrimRect Tool

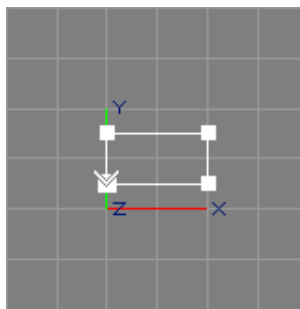


Figure 123: A TrimRect

Arguments

The TrimRect tool takes the selected NPatch objects (or NPatch providing objects¹) from the selection. If the current level is inside a NPatch object, the respective parent object is used.

Operation

For each selected patch, the TrimRect tool creates a non-rational, piecewise linear, planar NURBS curve of rectangular shape in the XY-plane, that fits in the (u, v) parameter space of this NURBS patch, for use as outer trim curve. See also the image above, depicting a rectangle for a NPatch defined on the knot intervals

0.0, 0.0, 1.0, 1.0

and

0.0, 0.08 $\bar{3}$, 0.1 $\bar{6}$, 0.25, . . . , 0.75, 0.8 $\bar{3}$, 0.91 $\bar{6}$, 1.0.

The created curve will be marked as closed and the generation of multiple points will be enabled, so that point edit actions know that they may need to move two points. See also section [4.4.1 Multiple Points](#) (page 153).

If a NPatch object is selected, the new curve will be linked as child of this NPatch object. Otherwise it will be linked to the current level.

The "CreateAt" and "CreateIn" options are ignored.

See section [4.6.1 Trim Curves](#) (page 171) for a more detailed discussion of trim curves and how to use the rectangular curve created by the TrimRect tool.

¹ Since 1.22.

5.2.5 Tween Curve Tool

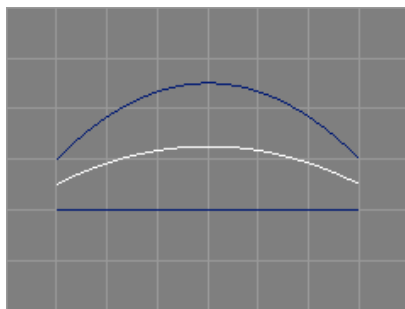


Figure 124: Tweened Curve (white) from two Parameter Curves (blue) with $r = 0.5$

Arguments

The tween curve tool takes two NURBS curves from the selection and requests a parameter r . In addition, an *append* option can be set.¹

Operation

The selected NURBS curves will be interpolated (tweened) and a new curve incorporating features from both of the original curves will be created. See also the image above. The parameter r defines the ratio of influence of the first and the second curve (the latter is using a ratio of $1 - r$).

```
|-NCurve1          |-NCurve1
|-NCurve2          ==> |-Tweened_Curve (NCurve)
                    |-NCurve2
```

If a third curve is selected, the parameter r is ignored and this third curve defines the ratio of influence with its y coordinates. Unless the *append* option is set, the new curve will be created after the first curve.

Notes

The two curves must be of the same length and order. They need not be defined on the same knot vector, however.

The interpolation control curve does not need to be compatible with any of the original curves.

If one of the curves knot vector types is "Custom" or the respective knot vector types are different, the resulting knot vector type will be "Custom" and the knot values will also be interpolated/tweened. Otherwise a matching knot vector will be generated according to the type.

The original NURBS curves will not be deleted by this tool.

See also the documentation of the corresponding scripting interface command [6.2.15 tweenNC](#) (page 395) and the related tool for surfaces [5.4.14 Tween Surfaces Tool](#) (page 316).

To tween NURBS curve providing objects or incompatible NURBS curve objects, a Script object must be used, as shown in the distributed example scene file "tweenc.ay" and script "tweenc.tcl".

¹ Since 1.29.

5.3 Curve Modification Tools

These tools modify parametric curve objects.

Unless noted otherwise, PV tags are *not* supported/modified by these tools.

5.3.1 Revert Tool

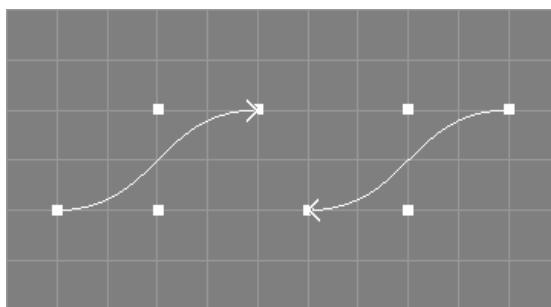


Figure 125: Revert Tool (l: Original Curve, r: Reversed Curve)

Arguments

The revert tool takes all NCurve, ICurve, ACurve, NCircle, and ConcatNC objects from the selection. In addition, plugin defined curve object types can support the revert operation.¹

Operation

The direction of the selected curves will be reversed. This tool also reverts the relative knot distances of NURBS curves so that for example a NURBS curve defined on the (asymmetric) knot vector

$[000 \ 0.75 \ 1 \ 1]$

will get the new knot vector

$[000 \ 0.25 \ 1 \ 1]$

after reversal. This ensures that the shape of a NURBS curve does not change during reversal. Interpolating and approximating curves may change their shape as the underlying interpolation/approximation algorithms are not direction-invariant.

For interpolating curves, the end derivatives are also reversed properly.

Notes

The revert tool can be conveniently invoked from view windows using the keyboard shortcut `< ! >` (exclamation mark).

Eventually selected points will still be selected after this operation.

See also the documentation of the corresponding scripting interface command [6.2.13 revertC](#) (page 390) and the related tools for surfaces [5.5.1 Revert U Surface Tool](#) (page 317) and [5.5.2 Revert V Surface Tool](#) (page 317).

¹ Since 1.26.

5.3.2 Open Tool

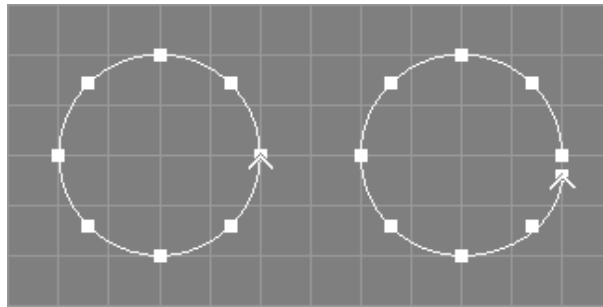


Figure 126: Open Tool (l: Closed Curve, r: Opened Curve)

Arguments

The open tool takes all NCurve, ICurve, ACurve, and ConcatNC objects from the selection. In addition, plugin defined curve object types can support the open operation.¹

Operation

The curves will be opened. For closed NCurve objects, the last point will be moved away from the first (multiple end points will be managed correctly), see also the image above.

For periodic NURBS curves, the last p points will be moved away from the first p (where p is the degree of the curve).

For ACurve, ICurve, and ConcatNC objects the "Closed" attribute will be cleared and no control points will be modified.

Notes

The open tool can be conveniently invoked from view windows using the keyboard shortcut < (>.

See also the documentation of the corresponding scripting interface command [6.2.13 openC](#) (page 390).

¹ Since 1.26.

5.3.3 Close Tool

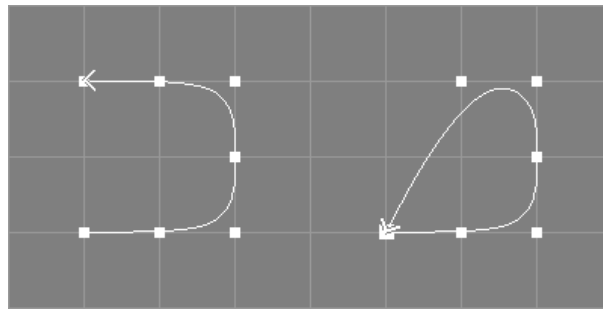


Figure 127: Close Tool (l: Open Curve, r: Closed Curve)

Arguments

The close tool takes all NCurve, ICurve, ACurve, and ConcatNC objects from the selection. In addition, plugin defined curve object types can support the close operation.¹

Operation

The curves will be closed. For NCurve objects the last point will be made identical to the first, however, this does not guarantee a closed curve shape unless the knot vector is clamped, see also the image above.

For ACurve, ICurve, and ConcatNC objects the "Closed" attribute will be set and no control points will be modified.

Notes

The close tool can be conveniently invoked from view windows using the keyboard shortcut <) >.

To close a curve with a fillet, the ConcatNC tool object can be used instead of this tool, see also section 4.5.1 ConcatNC object (page 162).

See also the documentation of the corresponding scripting interface command [6.2.13 closeC](#) (page 390).

¹ Since 1.26.

5.3.4 Refine Tool

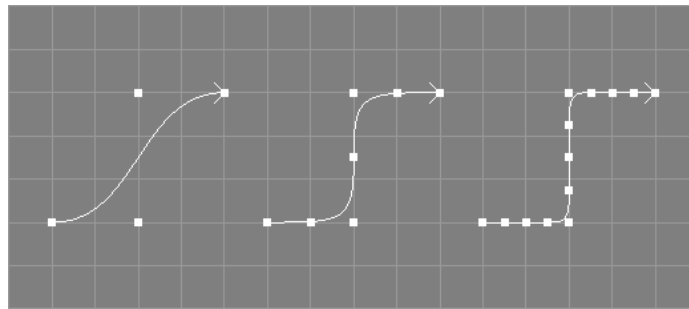


Figure 128: Successive Application of Refine Tool

Arguments

The refine tool takes all NCurve, ICurve, and ACurve objects from the selection.¹ In addition, plugin defined curve object types can support the refine operation.²

Operation

The selected curves will be refined by inserting a control point in the middle of each control point interval, changing the shape of the curve. The original control points will not be changed. For periodic NURBS curves, no control points will be inserted in the last p intervals (where p is the degree of the curve), this allows to maintain the periodicity, see also the image below where a periodic curve of length 9 (8 sections) has been refined, resulting in a periodic curve of length 15 (not 17).

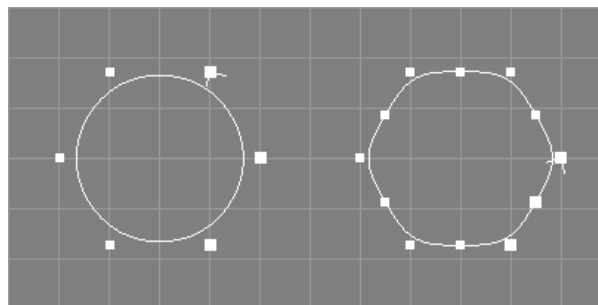


Figure 129: Refining a Periodic NURBS Curve

If there are selected points, only the intervals between the first and the last selected point are refined, see also the image below.

The point selection will be adapted so that the refine tool can be applied multiple times.

Notes

The refine tool can be conveniently invoked from view windows using the keyboard shortcut `<#>`.³

See also the documentation of the corresponding scripting interface command [6.2.13 refineC](#) (page 390).

¹ ICurve and ACurve since 1.21. ² Since 1.26. ³ Since 1.26.

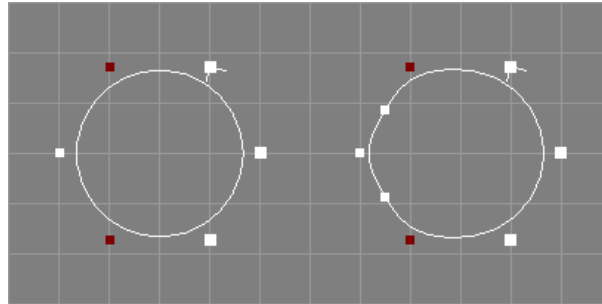


Figure 130: Refining a Selected Region

5.3.5 Refine Knots Tool

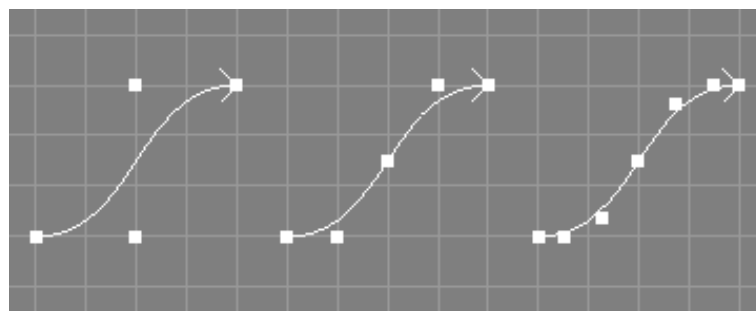


Figure 131: Successive Application of Refine Knots Tool

Arguments

The refine knots tool takes a number of NURBS curves from the selection.

Operation

The knot vectors of the selected NURBS curves will be refined by inserting a knot in the middle of each inner knot interval without changing the shape of the curve. New control points will be added and the position of old control points may be changed in the progress. See also the image above where a curve defined on the knot vector

`[0000 1 1 1 1]`

is successively refined to the knot vector

`[0000 0.5 1 1 1 1]`

and then

`[0000 0.25 0.5 0.75 1 1 1 1]`.

Notes

Because new knots are inserted into inner intervals only, the clamping state of the knot vectors does not change. Furthermore, knot vectors of type "NURB" will not change in type, other knot vectors will be changed to type "Custom".

The curve type of periodic curves will *not* be preserved.

The point selection will be removed from the processed objects.

See also the documentation of the corresponding scripting interface command [6.2.15 refineknNC](#) (page 395) and the related tool for surfaces [5.5.6 Refine Knots Surface Tool](#) (page 319).

5.3.6 Refine Knots With Tool

Arguments

The refine knots with tool takes a number of NURBS curves from the selection and requests a vector of new knot values.

Operation

The knot vectors of the selected NURBS curves will be refined by inserting all knots from the specified vector at once. New control points will be added and the position of old control points may be changed in the progress. The resulting knot vector must be valid, otherwise an error will be reported and the respective curve is not changed.

Notes

Note that the shapes of the curves do not change, but the position of certain control points does.

The point selection will be removed from the processed objects.

See also the documentation of the corresponding scripting interface command [6.2.15 refineknNC](#) (page 395) and the related tool for surfaces [5.5.7 Refine Knots With Surface Tool](#) (page 320).

5.3.7 Coarsen Tool

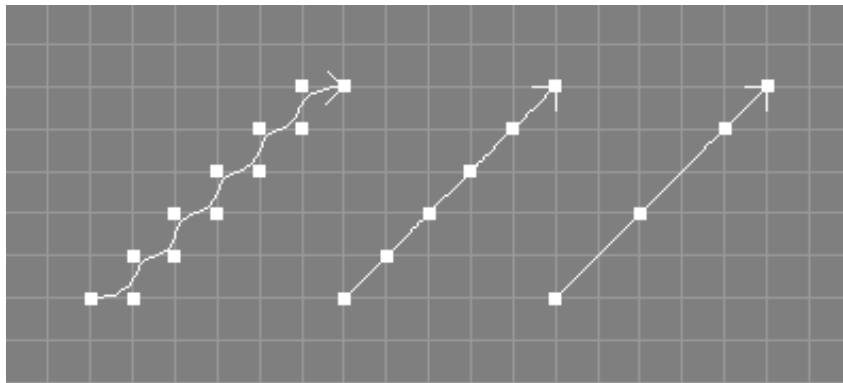


Figure 132: Successive Application of Coarsen Tool

Arguments

The coarsen tool takes all NCurve, ICurve, and ACurve objects from the selection.¹ In addition, plugin defined curve object types can support the coarsen operation.²

Operation

Every second control point in the control vectors of the selected curves will be deleted. If the new length of the curve would be smaller than the current order, the coarsen tool will not change anything.

Notes

For periodic NURBS curves the coarsen tool will not remove control points from the first (last) p intervals (where p is the degree of the curve).

For closed curves, the coarsen tool will not remove the last point.

The coarsen tool will also remove knot values from NURBS curves with custom knot vectors.

If there are selected points, only the intervals between the first and the last selected point are affected.

The point selection will be modified so that the tool may be applied multiple times.³

The coarsen tool can be conveniently invoked from view windows using the keyboard shortcut `<Shift-#>`.⁴

See also the documentation of the corresponding scripting interface command [6.2.13 coarsenC](#) (page 390).

¹ ICurve and ACurve since 1.26. ² Since 1.26. ³ Since 1.27. ⁴ Since 1.26.

5.3.8 Elevate Tool

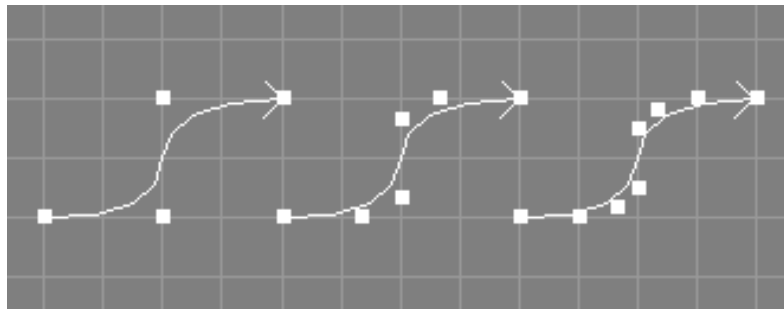


Figure 133: Successive Application of Elevate Tool (Order 3 (left), 4 (middle), 5 (right))

Arguments

The elevate tool takes a number of NURBS curves from the selection and additionally requests an integer value.

Operation

The order of the selected NURBS curves will be raised by the specified integer value without changing the shape of the curve. See also the example image above.

Notes

If the knot vector of the curve is not clamped, it will be clamped automatically. The knot type of the curve will be changed to "Custom". New control points will be added and the position of old control points may be changed in the progress.

The point selection will be removed from the processed objects.

See also the documentation of the corresponding scripting interface command [6.2.15 elevateNC](#) (page 393) and the related tool for surfaces [5.5.8 Elevate Surface Tool](#) (page 321).

5.3.9 Reduce Tool

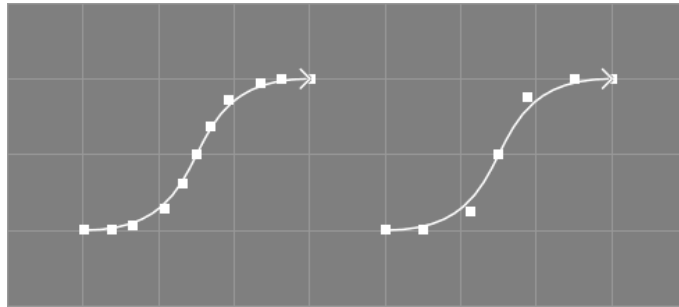


Figure 134: Reduce Tool (Order 5 (left), 4 (right))

Arguments

The reduce tool takes a number of NURBS curves from the selection and additionally requests a tolerance value.

Operation

The order of the selected NURBS curves will be decreased by one if the shape of the reduced curves does not deviate from the original curves by the given tolerance in any point.

See also the example image above where a curve of order five and length 11, defined on the knot vector

`[0 0 0 0 0 0.25 0.25 0.5 0.5 0.75 0.75 1 1 1 1 1]`,

has been reduced to order four and is now of length seven and defined on the knot vector

`[0 0 0 0 0.25 0.5 0.75 1 1 1 1]`.

Notes

If the knot vector of the curve is not clamped, it will be clamped automatically. The knot type of the curve will be changed to "Custom". Control points will be removed and the position of old control points may be changed in the progress.

The point selection will be removed from the processed objects.

The ability to reduce a certain curve can be improved by increasing the multiplicity of internal knots first. Furthermore, if the shape of the reduced curve has priority over the distribution of control points suitable for further editing, the curve approximation tool may be able to deliver a result even if curve reduction fails, see section [5.3.23 Approximate Tool \(page 302\)](#).

See also the documentation of the corresponding scripting interface command [6.2.15 reduceNC \(page 394\)](#) and the related tools for surfaces [5.5.9 Reduce Surface Tools \(page 322\)](#).

5.3.10 Extend Tool

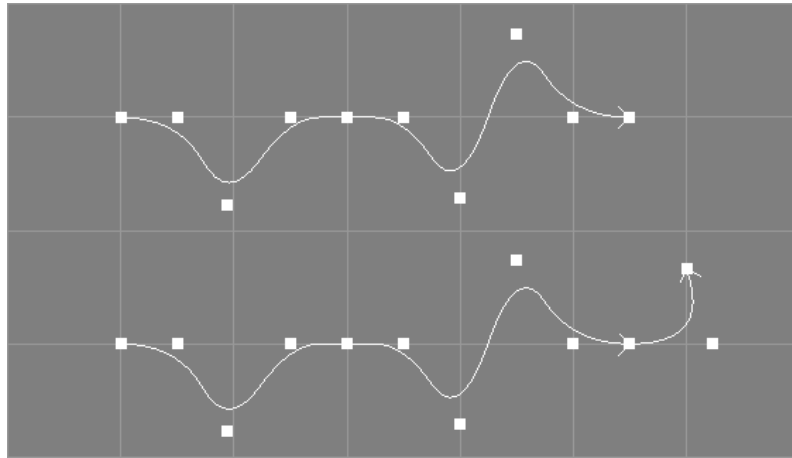


Figure 135: Extend Tool Example (Original Curve (upper), Original Curve and Extended Curve Superimposed (lower))

Arguments

The extend tool takes a number of NURBS curves from the selection and additionally requests a point in space.

Operation

The selected NURBS curves will be extended to the specified point in space without changing the already existing shape of the curves.

The knot vector of the curves will be rescaled to the range $[0, 1]$. If the knot vector of a curve is not clamped at the end, it will be clamped automatically. The knot type of each curve will be changed to "Custom". A new control point will be added and the position of old control points may be changed in the progress.

Notes

Due to the constraints that only one control point will be added to the curve and the current shape must be maintained, the new curve may form unusual/unwanted big arcs if the new point deviates too much from the tangent in the endpoint of the original curve.

The point selection will be removed from the processed objects.

See also the documentation of the corresponding scripting interface command [6.2.15 extendNC](#) (page 393).

5.3.11 Clamp Tool

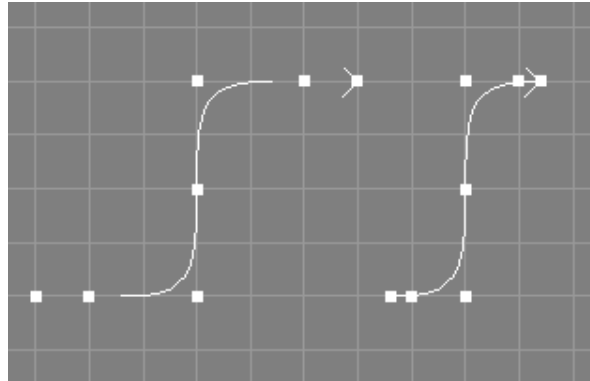


Figure 136: Clamp Tool (left: original curve, right: clamped curve)

Arguments

The clamp tool takes a number of NURBS curves from the selection.

Operation

The knot vectors of the selected NURBS curves will be changed using knot insertion so that the first and the last knot have a multiplicity equal to the order of the curve, without changing the shape of the curve. The curve will interpolate the first and the last control point afterwards (unless the weights of those points are not 1.0).

See also the image above, where a curve of length seven and order four, originally defined on the knot vector:

`[0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1]`

has been clamped and is now defined on the knot vector:

`[0.3 0.3 0.3 0.3 0.4 0.5 0.6 0.7 0.7 0.7 0.7]`.

Notes

The knot type of the curves will be changed to "Custom".

In Ayam versions prior to 1.18 it was an error if the curve was already clamped at either side, this is no longer the case. Furthermore, curves with multiple knots in the end region(s) could not be clamped, this works ok now.

See also the documentation of the corresponding scripting interface command [6.2.15 clampNC](#) (page 393) and the related tool for surfaces [5.5.10 Clamp Surface Tool](#) (page 323).

5.3.12 Unclamp Tool

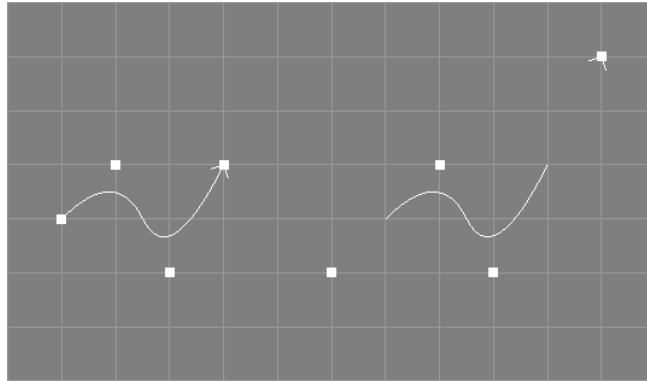


Figure 137: Unclamp Tool (left: original curve, right: unclamped curve)

Arguments

The unclamp tool takes a number of NURBS curves from the selection.

Operation

The knot vectors of the selected NURBS curves will be changed so that there are no multiple knots at the respective ends of the knot vectors. The shape of the curve will not change but the position of some control points will. The knot type of the curve can be changed to type "Custom".

See also the image above where a NURBS curve of length four and order three, defined on the knot vector

$[0\ 0\ 0\ 0.5\ 1\ 1\ 1]$,

has been unclamped and is now using the new knot vector:

$[0\ 0.1333\ 0.3333\ 0.5\ 0.6667\ 0.8333\ 1]$.

Notes

Unclamping is *not* an exact reversal of clamping (and vice versa).

Furthermore, as the unclamp operation only works on completely clamped curves, the unclamp tool may need to clamp the curve first.

See also the documentation of the corresponding scripting interface command [6.2.15 unclampNC](#) (page 393) and the related tool for surfaces [5.5.11 Unclamp Surface Tool](#) (page 324).

5.3.13 Insert Knot Tool

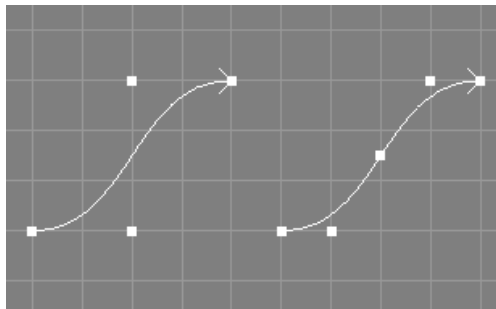


Figure 138: Insert Knot Tool (left: original curve, right: a knot has been inserted one time at $u = 0.5$)

Arguments

The insert knot tool takes a number of NURBS curves from the selection and requests two additional values, a parametric value u and an integer value i .

Operation

A knot with the specified value u will be inserted i times into the knot vector of the selected curves, without changing the shape of the curves.

See also the image above, where a knot has been inserted at $u = 0.5$ one time into a curve of length four and order four that was originally defined on the knot vector:

$[0000 \ 1111]$.

The processed curve of length five is defined on the knot vector:

$[0000 \ 0.5 \ 1111]$.

Notes

The knot type of the curves will be changed to "Custom".

The point selection will be removed from the processed objects.

See also the documentation of the corresponding scripting interface command [6.2.15 insknNC](#) (page 394) and the related tool for surfaces [5.5.12 Insert Knot Surface Tool](#) (page 325).

5.3.14 Remove Knot Tool

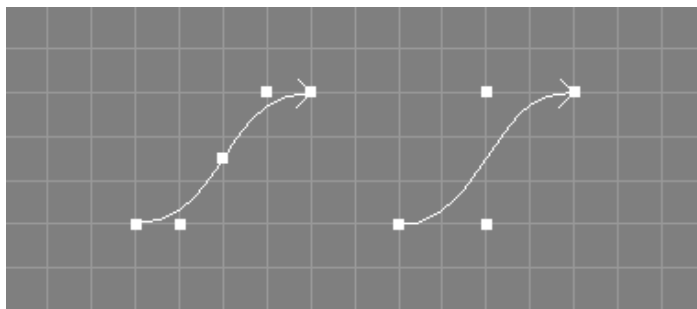


Figure 139: Remove Knot Tool (left: original curve, right: the knot at $t = 0.5$ has been removed one time)

Arguments

The remove knot tool takes a number of NURBS curves from the selection and requests three additional values, a parametric value t , an integer value r , and a tolerance value tol .

Operation

The knot at the specified parametric value t will be removed r times from the knot vector of the selected curves if the shape of the resulting curve does not deviate more than tol from the original curve in any point. Since Ayam 1.20 the knot to remove may also be specified using its (zero based) index in the knot vector; for instance by entering "-i 3" instead of "0.5" for the knot vector:

```
[000 0.5 111].
```

If the knot can not be removed r times due to the tolerance given, an error is reported and the original curve is left unchanged.

This operation also fails, if the knot removal would lead to a curve of lower order.

See also the image above, where the knot $t = 0.5$ has been removed one time from a curve of length five and order four that was originally defined on the knot vector:

```
[0000 0.5 1111].
```

The processed curve of length four is defined on the knot vector:

```
[0000 1111].
```

Notes

If tol is "0.0" the remove knot tool tries to work without changing the shape of the curve, i.e. only superfluous knots will be removed. If tol is "Inf" (infinity) the specified knot will be removed regardless of potential curve changes.

The point selection will be removed from the processed objects.

See also the documentation of the corresponding scripting interface command [6.2.15 remknNC](#) (page 394) and the related tool for surfaces [5.5.13 Remove Knot Surface Tool](#) (page 326).

5.3.15 Remove Superfluous Knots Tool

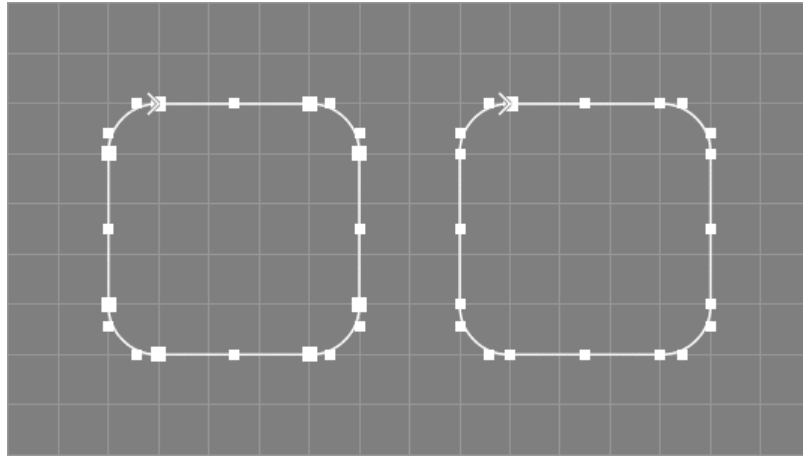


Figure 140: Remove Superfluous Knots Tool (left: original curve, right: superfluous knots have been removed)

Arguments

The remove superfluous knots tool takes a number of NURBS curves from the selection and requests a tolerance value *tol*.

Operation

All knots that do not contribute to the shape of the curve will be removed from the knot vector of the selected curves if the shape of the resulting curve does not deviate more than *tol* from the original curve in any point.

It is no error if no knots can be removed.

See also the example image above where a curve created by the concatenation algorithm with knot type "Custom" and therefore superfluous knots has been cleaned up by the remove superfluous knots tool. The original curve of order three with 28 control points is defined on the knot vector:

```
[000 111 1.5 222 333 3.5 444 555 5.5 666 777 7.5 888].
```

The processed curve has 21 control points and is defined on the knot vector:

```
[000 11 1.5 22 33 3.5 44 55 5.5 66 77 7.5 888].
```

Notes

The point selection will be removed from the processed objects.

See also the documentation of the corresponding scripting interface command [6.2.15 remsuknNC](#) (page 394) and the related tools for surfaces [5.5.14 Remove Superfluous Knots Surface Tools](#) (page 327).

5.3.16 Fair Tool

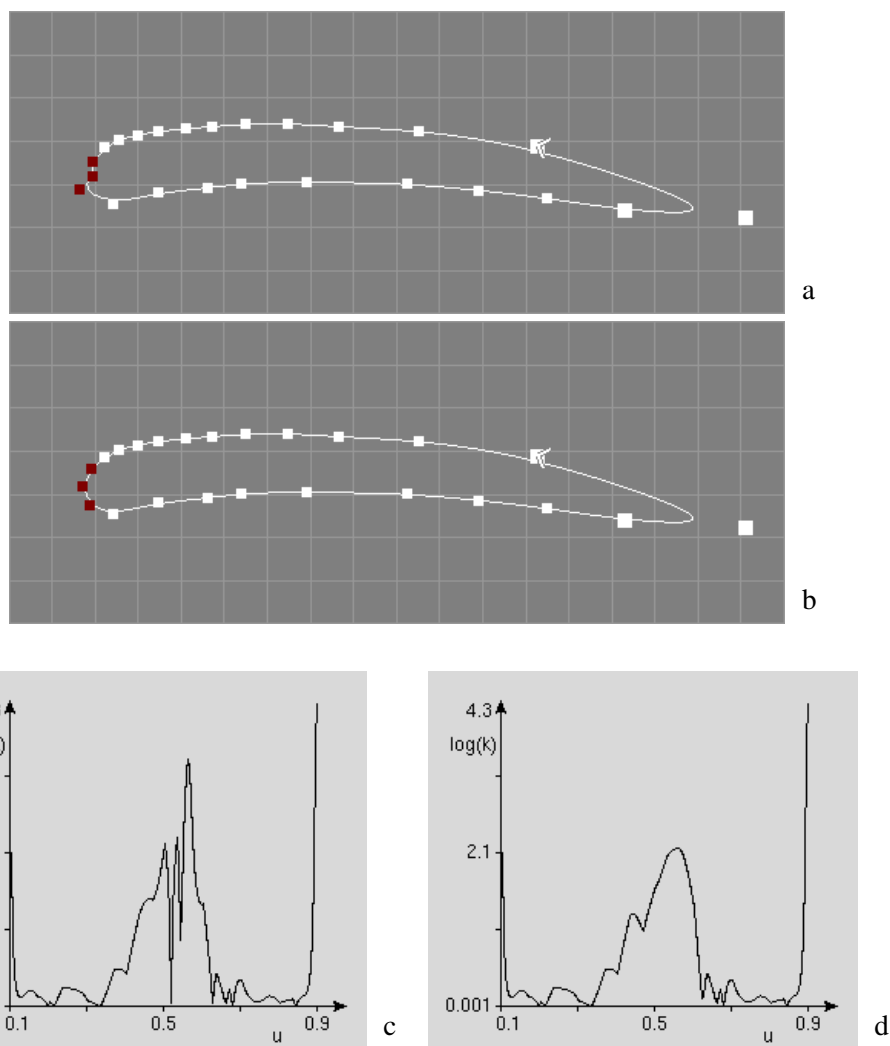


Table 97: Fair Tool (a: Original, b: Processed, c: Curvature of a, d: Curvature of b)

Arguments

The fairing tool takes a number of NURBS curves from the selection and requests a tolerance value *tol*.

Operation

The control points of the curves will be modified so that the shape of the curve is somewhat smoother. The movement of a single control point is restricted by the specified tolerance value. This can also be used to apply the tool multiple times until the curve shape meets the requirements.¹

If the option "Fair Worst" is enabled, only the control point that would be moved the longest distance will be modified.²

If there are selected points, only those will be modified.

See also the example images above where a kink on the leading edge of a wing profile (a) has been removed by applying the fairing tool with a tolerance value of 0.1 two times, yielding the lower curve (b). Compare also the curvature plots of both curves (c and d).

¹ Since 1.28. ² Since 1.29.

Notes

The fairing algorithm uses the four direct and indirect neighboring control points of each point to process. This poses a problem for the end points of an open curve. Therefore, the control vector of open curves will be extended by a simple linear extrapolation algorithm, so that every point of the original curve has four neighbors. However, the fairing quality for the first, second, second to last, and last control point of an open curve will be somewhat lower.

The original algorithm only delivers optimal results for cubic curves (of order four), using curves of other degrees will work too but yield worse results.

Due to the fact that the algorithm uses already faired points as input points for other points to fair, it is *not* direction invariant if used on all or multiple neighboring points. Also, control points already placed on very good if not optimal positions may be moved away to less favorable positions. Further problems arise for multiple control points, used to model sharp corners, as the fairing algorithm does not differentiate between wanted and unwanted non-continuous curve features. If in doubt, just select a single control point to fair.

See also the documentation of the corresponding scripting interface command [6.2.15 fairNC \(page 399\)](#) and the related tool for surfaces [5.5.18 Fair Surface Tool \(page 331\)](#).

5.3.17 Concat Tool

Arguments

The concat tool takes all NURBS curves and NURBS curve providing objects from the selection and requests parameters for closing, knot type, and fillet creation.¹

Operation

The selected curves will be concatenated and a new NURBS curve will be created.

--NCurve		--NCurve
--NCurve	==>	--NCurve
		--NCurve

Notes

The order of the new curve is taken from the first curve. If one of the curves has weights, the resulting curve will have weights too.

The original NURBS curves will not be deleted by this tool.

See also section [4.5.1 ConcatNC Object \(page 162\)](#) and the documentation of the corresponding scripting interface command [6.2.13 concatC \(page 391\)](#).

¹ Since 1.27.

5.3.18 Split Tool

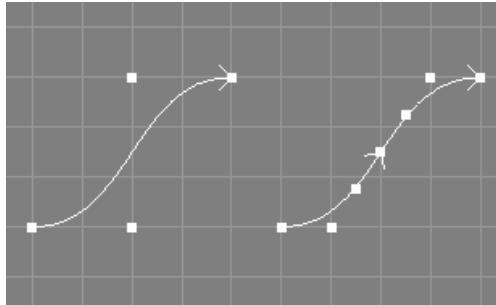


Figure 141: Split Tool (left: original curve, right: resulting split curves for $u=0.5$)

Arguments

The split curves tool takes a number of NURBS curves from the selection and additionally requests a parametric value u . In addition, a *relative* and an *append* option can be set.

Operation

The selected NURBS curves will be split into two NURBS curves at the designated parametric value u . The splitting process involves application of knot insertion, so that all new curves will get a custom knot vector. The valid range of parametric values depends on the knot vector of the original curve unless the *relative* option is set, where it must be in the range (0.0, 1.0).

The new curve(s) will be inserted into the level right after the respective curve(s) to split unless the *append* option is set.¹

--NCurve		--NCurve
--Sphere	==>	--NCurve
		--Sphere

See also the image above, where a curve of length and order four, originally defined on the knot vector $[0000 \ 1111]$,

was split at parametric value 0.5 resulting in a new curve of length and order four that is defined on the knot vector

$[0.50.50.50.5 \ 1111]$.

The original curve will be defined on the knot vector

$[0000 \ 0.50.50.50.5]$

after this operation.

Notes

The original selected NURBS curve objects will be changed.

The point selection will be removed from the original curves.

See also the documentation of the corresponding scripting interface command [6.2.15 splitNC](#) (page 395) and the related tool for surfaces [5.5.15 Split Surface Tool](#) (page 328).

¹ Since 1.24.

5.3.19 Curve Trim Tool

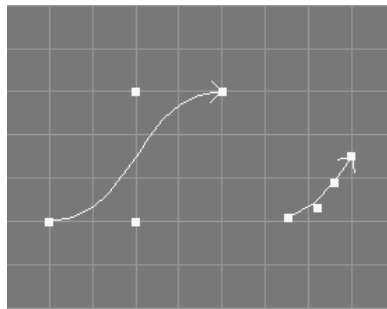


Figure 142: Trim Tool (left: original curve, right: resulting trimmed curve for $u_{min} = 0.1$, $u_{max} = 0.5$)

Arguments

The trim curves tool takes all selected NURBS curves from the selection and additionally requests two parametric values, u_{min} and u_{max} .

Operation

The selected NURBS curves will be trimmed to the designated parametric range (u_{min} , u_{max}).

See also the image above, where a curve of length and order four, originally defined on the knot vector $[0 0 0 0 \ 1 1 1 1]$, was trimmed at the parametric values 0.1 0.5 resulting in a new curve of length and order four that is defined on the knot vector $[0.1 0.1 0.1 0.1 \ 0.5 0.5 0.5 0.5]$.

Notes

The trimming process involves the application of knot insertion so that the curves will get a custom knot vector.

The point selection will be removed from the original objects.

See also the documentation of the corresponding scripting interface command [6.2.15 trimNC](#) (page 396).

5.3.20 Reparameterisation Tool

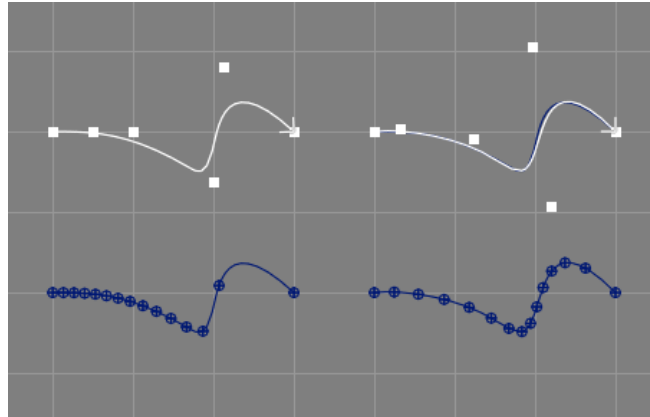


Figure 143: Reparameterisation Tool (ul: original curve, ur: reparameterised curve (white) over original (blue), ll: original curve used as clone trajectory, lr: reparameterised curve used as clone trajectory)

Arguments

The reparameterisation curves tool takes a number of NURBS curves from the selection and additionally requests a type t .

Operation

The selected NURBS curves will be approximately reparameterised to the given knot type t .

See also the image above, where a curve of length six and order four, originally defined on the knot vector

$[0000 \ 0.8 \ 0.9 \ 1111]$,

was reparameterised to type "Centripetal" resulting in a new curve of length six and order four that is defined on the knot vector

$[0000 \ 0.323 \ 0.5 \ 1111]$.

Notes

The reparameterisation process involves a tessellation followed by an approximation, so that

- the shape of the new curve will slightly differ from the original curve,
- the new knot vector will be of type "Custom",
- any weights will be removed / reset to 1.0,
- periodic curves will be converted to closed curves.

See also the documentation of the corresponding scripting interface command [6.2.15 reparamNC](#) (page 396)

5.3.21 Plot Curvature Tool

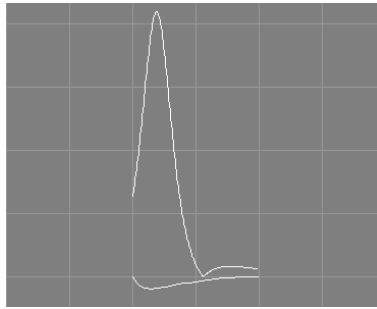


Figure 144: Curvature Plot (top) of simple NURBS curve (bottom)

Arguments

The plot curvature tool takes a number of NURBS curves from the selection and requests three additional values: the number of data points, the width value and the height value.

Operation

A new NURBS curve, depicting the curvature of the selected NURBS curve, will be created for each of the selected NURBS curves. The curvature plots will have a length defined by the number of data points and will be scaled to the specified width and by the specified height value. See also the image above.

5.3.22 Interpolate Tool

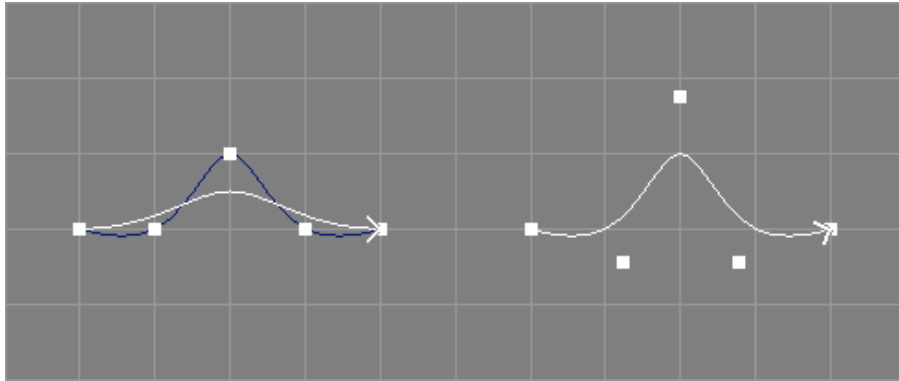


Figure 145: Interpolate Curve Tool (left-white: original curve, left-blue and right: interpolating curve)

Arguments

The interpolate tool takes a number of NURBS curves from the selection and requests five additional parameters.

Operation

The curves will be modified so that they interpolate all original control points with the desired interpolation order, parameterisation type and end derivatives. For a detailed description of the parameters see section 4.4.2 [ICurveAttr Property](#) (page 156). See also the image above.

Notes

Eventually present weight information of the original curves will be ignored.

The point selection will be removed from the original objects.

See also the documentation of the corresponding scripting interface command [6.2.16 interpNC](#) (page 406) and the related tool for surfaces [5.5.16 Interpolate Surface Tool](#) (page 329).

5.3.23 Approximate Tool

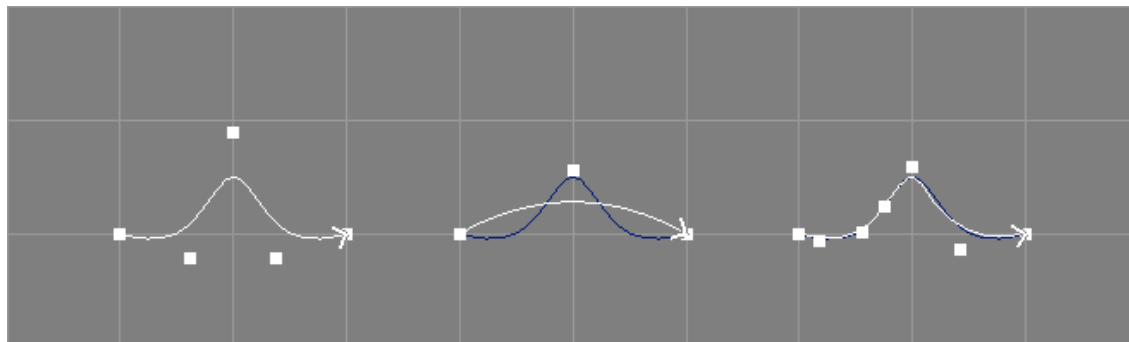


Figure 146: Approximate Curve Tool (left: original curve, middle and right: approximating curve)

Arguments

The approximate tool takes a number of NURBS curves from the selection and requests additional parameters: new length, new order, whether to create a closed curve, a parameterization type, and a tessellation parameter.

Operation

The curves will be modified so that they approximate their respective original curve shape(s) using a new non-rational NURBS curve with an adjustable number of new control points, order and parameterization type. The centripetal parameterization is better suited than the default (chordal) if the input data contains sharp turns. The uniform type is optimal if regular/symmetric distribution of output control points is desired and the resulting curve is to be edited manually further on. The data points to approximate are created by tessellating the original curves. The tessellation parameter controls how fine this tessellation shall be, larger values lead to more data points and possibly higher approximation fidelity but also slower processing. A value of 0 disables the tessellation completely and the original control points will be used as approximation input.¹

The total number of data points generated by the tessellation is $nd + (nd - 1) \times t$, where nd is the total number of distinct knots in the valid range of the knot vector and t the tessellation parameter.

See also the image above where a curve with seven original control points of order four has been approximated by a curve with just three control points and order three (middle) and by a curve with seven control points and order three (left).

Notes

The point selection will be removed from the original objects.

Potentially present weights will be removed from the processed curves.

See also the documentation of the corresponding scripting interface command [6.2.15 approxNC](#) (page 398), and the related tool for surfaces [5.5.17 Approximate Surface Tools](#) (page 330).

Approximation is also used by the [5.3.20 Reparameterisation Tool](#) (page 299).

¹ Since 1.29.

5.3.24 Shift Closed Curve Tool

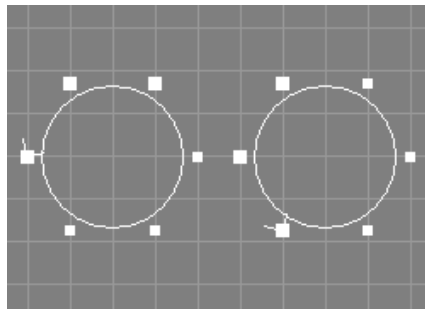


Figure 147: Shift Closed Curve Tool (left: Closed B-Spline Curve, right: Shifted Curve)

Arguments

The shift closed curve tool takes a number of closed curves (NCurve, ICurve, and ACurve objects are supported), from the selection and requests one additional integer parameter i .

Operation

The control points of the curve(s) will be shifted i times.

The parameter i may be negative to revert the direction of the shifting.

For a simple closed curve, shifting with $i = 1$, the first control point will get the coordinates of the former last control point. This means, positive shifts occur in the direction of the curve. Note that for closed and periodic NURBS curves, the multiple points will be managed correctly.

Notes

Eventually selected points will still be selected after this operation. See also the image above.

See also the documentation of the corresponding scripting interface command [6.2.13 shiftC](#) (page 391).

5.3.25 To XY Tool

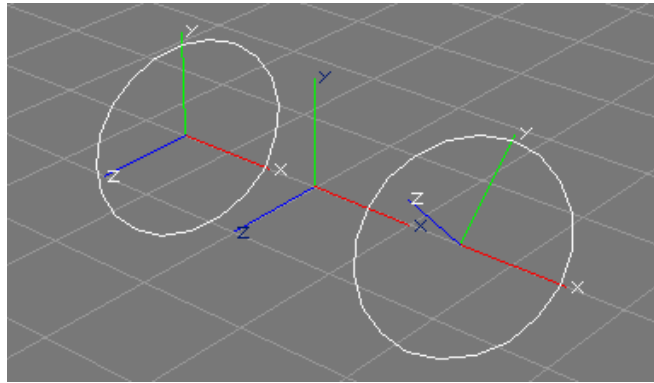


Figure 148: To XY Tool (left: original curve, right: modified curve)

Arguments

The To XY tool takes a number of NCurve, ACurve, or ICurve objects from the selection. The curves should be planar. Curves that form a single straight line are not supported.

Operation

The control points of the curves will be rotated, so that they are in the XY-plane of the respective object space defined by the curve objects. Additionally, the rotation attributes of the curve objects will be changed so that the curve does not change its orientation with regard to other objects or the world space. The scale attributes will be reset to 1.0.

See also the image above, where the left curve, planar but not defined in the XY-plane will be changed, so that it is defined in the XY-plane (mind the two different object coordinate systems in conjunction with the world coordinate system in the middle).

Notes

A reverse operation, apart from undo, would be to apply the current transformation attributes to the control points of the curves.

See also the documentation of the corresponding scripting interface command [6.2.13 toXYC](#) (page 391).

5.3.26 Make Compatible Tool

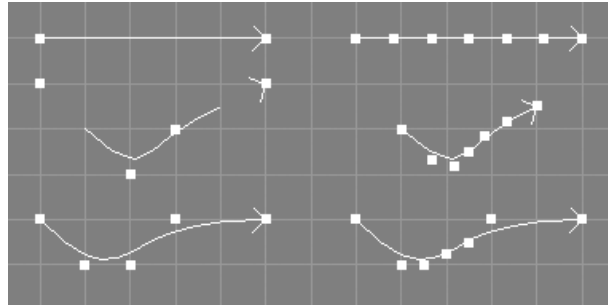


Figure 149: Make Compatible Tool (left: Original Curves, right: Compatible Curves)

Arguments

The make compatible tool takes a number of NURBS curves from the selection and requests a level.

Operation

The curves will be made compatible, so that, based on the level parameter given, they are of the same length, order and defined on the same knot vector.

This eases the application of the build from curves tool (see section 5.6.4 [Build from Curves Tool](#) (page 336)) or the application of skinning (see section 4.7.7 [Skin Object](#) (page 201)).

Notes

This tool does not change the geometry of the curves. However, since clamping, degree elevation, and knot insertion may be used on the curves, their order, knot vectors, and control points may be changed.

The orders of all curves will simply be raised to the maximum order of all curves; no attempt is made to check, whether lowering orders would lead to a simpler result.

The point selection will be removed from the original objects.

See also the documentation of the corresponding scripting interface command [6.2.15 makeCompNC](#) (page 397) and the related tool for surfaces [5.5.19 Make Surfaces Compatible Tool](#) (page 332).

5.3.27 Rescale Knots to Range Tool

Arguments

The rescale knots to range tool takes a number of NURBS curves from the selection and requests a range (two float values).

Operation

The knot vectors of the curves will be scaled, so that their first and last values match the given range.

Notes

Since Ayam 1.20 the knot type of the curve does not have to be "Custom" anymore. Furthermore, rescaling the knots does not change the knot type.

This tool does not change the geometry of the curves.

See also the documentation of the corresponding scripting interface command [6.2.15 rescaleknNC](#) (page 395) and the related tool for surfaces [5.5.20 Rescale Knots to Range Surface Tool](#) (page 333).

5.3.28 Rescale Knots to Mindist Tool

Arguments

The rescale knots to mindist tool takes a number of NURBS curves from the selection and request a minimum distance value.

Operation

The knot vectors of the curves will be scaled, so that no two knots have a distance smaller than the given minimum distance (except for multiple knots).

Notes

Since Ayam 1.20 the knot type of the curve does not have to be "Custom" anymore. Furthermore, rescaling the knots does not change the knot type.

This tool does not change the geometry of the curves.

See also the documentation of the corresponding scripting interface command [6.2.15 rescaleknNC](#) (page 395) and the related tool for surfaces [5.5.21 Rescale Knots to Mindist Surface Tool](#) (page 333).

5.3.29 Collapse Points Tool

Arguments

The collapse tool expects a number of selected objects with selected (tagged) control points (see section 3.4 Selecting Points (page 79) for information on how to select (tag) control points).

Operation

The selected control points will be made a single multiple point, all points will get the coordinate values of the respective center point. This means that the shape of the processed curve/surface can change.

Notes

This command used to only create multiple points for NURBS curve/surface objects and clear the point selection.

In view windows this tool is bound to the <q> key.

5.3.30 Explode Points Tool

Arguments

The explode tool expects some objects with a number of selected (tagged) multiple points (see section 3.4 Selecting Points (page 79) for information on how to select (tag) control points).

Operation

The points forming the selected multiple points will be moved away from each other. The shapes of the processed objects will change.

Notes

This command used to only remove already existing multiple points of NURBS curves/surfaces that were created by the "CreateMP" option or the collapse tool above before.

In view windows this tool is bound to the <q> key.

5.4 Surface Creation Tools

These tools create parametric surface objects.

5.4.1 NURBSphere Tool

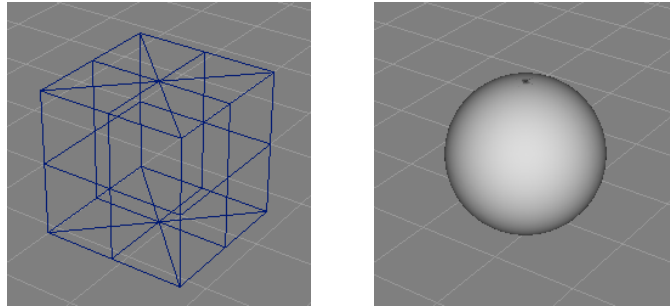


Table 98: NURB Sphere (l: Control Polygon, r: Shaded Surface)

Arguments

Radius.

Operation

The NURBSphere tool creates a half circle NURBS curve of given radius and revolves it about the Y-axis thus forming a sphere.

Notes

Due to the construction method, the NURBS surface is degenerate at the poles which can lead to shading artefacts (compare the image above).

5.4.2 NURBSphere2 Tool

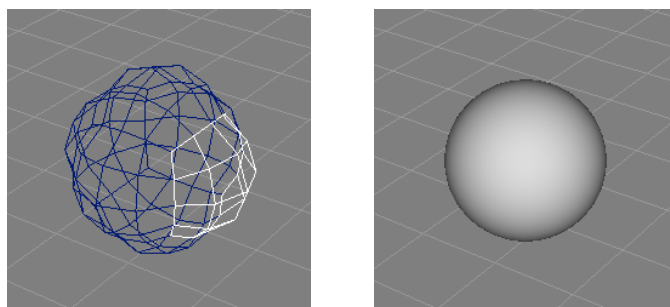


Table 99: Cobb NURB Sphere (l: Control Polygon, r: Shaded Surface)

Arguments

None.

Operation

The NURBSphere2 tool creates a Cobb NURB sphere of radius 1.0, consisting of six NURBS patches arranged in a configuration resembling a box, see also the above image.

Notes

The NURBS patches are of comparatively high order (5). In contrast to the standard NURBS sphere, the Cobb sphere is not degenerate and no shading artefacts result.

5.4.3 Revolve Tool**Arguments**

The revolve tool takes all selected objects or their selected boundaries.

Operation

The tool creates a Revolve object, and moves the selected objects to it.

```
| -NCurve      ==>    +-Revolve
                      \ -NCurve
```

For selected boundaries, properly parameterised ExtrNC objects will be created:

```
| -NPatch      ==>    +-NPatch  <----- .
                      +-Revolve      |
                      +-ExtrNC       |
                      \ -Instance  --'
```

Notes

See section 4.7.1 [Revolve Object \(page 182\)](#) for more information regarding the revolve object.

This tool uses the object clipboard to move the objects so that the original clipboard contents are lost when this tool finishes.

Holding down the <Ctrl> key while clicking on the corresponding icon in the toolbox will keep the parameter curve selected and the current level will also be changed.

5.4.4 Swing Tool**Arguments**

The swing tool takes all selected objects or their selected boundaries.

Operation

The tool creates a Swing object, moves the selected objects to it, and rotates the first curve to the YZ-plane and the second curve to the XZ-plane. If the respective curve object is of type NCurve, ICurve, or ACurve, the control points will be modified and the rotation attributes will be reset, objects of other types will be rotated by their transformation attributes.¹ The rotation will be recorded in the undo buffer.

¹ Since 1.24.

```

|-Cross_section (NCurve)          +-Swing
|-Trajectory (NCurve)             ==>  |-Cross_section (NCurve)
                                     \-Trajectory (NCurve)

```

For selected boundaries, properly parameterised ExtrNC objects will be created:

```

|-NPatch          ==>  |-NPatch  <----- .
                      +-Swing      |
                      +-ExtrNC     |
                      \-Instance  --'

```

Notes

See section 4.7.3 [Swing Object \(page 187\)](#) for more information regarding the Swing object.

This tool uses the object clipboard to move the objects so that the original clipboard contents are lost when this tool finishes.

Holding down the <Ctrl> key while clicking on the corresponding icon in the toolbox will keep the parameter curves selected and the current level will also be changed.

5.4.5 Extrude Tool

Arguments

The extrude tool takes all selected objects or their selected boundaries.

Operation

The tool creates an Extrude object, and moves the selected objects to it.

```

|-Outline (NCurve)          +-Extrude
|-Hole (NCurve)             ==>  |-Outline (NCurve)
                                     \-Hole (NCurve)

```

For selected boundaries, properly parameterised ExtrNC objects will be created:

```

|-NPatch          ==>  |-NPatch  <----- .
                      +-Extrude      |
                      +-ExtrNC     |
                      \-Instance  --'

```

Notes

See section 4.7.2 [Extrude Object \(page 184\)](#) for more information regarding the Extrude object.

This tool uses the object clipboard to move the objects so that the original clipboard contents are lost when this tool finishes.

Holding down the <Ctrl> key while clicking on the corresponding icon in the toolbox will keep the parameter curve selected and the current level will also be changed.

5.4.6 Sweep Tool

Arguments

The sweep tool takes all selected objects or their selected boundaries.

Operation

The tool creates a Sweep object, moves the selected objects to it, and offers to rotate the first curve object to the YZ-plane (if it is not already defined in this plane). If this curve object is of type NCurve, ICurve, or ACurve, the control points will be modified and the rotation attributes will be reset, objects of other types will be rotated by their transformation attributes.¹ The rotation will be recorded in the undo buffer.

```
| -Cross_section (NCurve)          +-Sweep
| -Trajectory (NCurve)             ==>  | -Cross_section (NCurve)
                                         \ -Trajectory (NCurve)
```

For selected boundaries, properly parameterised ExtrNC objects will be created:

```
| -NPatch          <----- .
| -NPatch          ==>  +-Sweep          |
                     +-ExtrNC          |
                     \ -Instance --'
```

Notes

See section 4.7.4 [Sweep Object \(page 190\)](#) for more information regarding the Sweep object.

This tool uses the object clipboard to move the objects so that the original clipboard contents are lost when this tool finishes.

Holding down the <Ctrl> key while clicking on the corresponding icon in the toolbox will keep the parameter curve selected and the current level will also be changed.

5.4.7 Bevel Tool

Arguments

The bevel tool takes all selected objects or their selected boundaries.

Operation

The tool creates a Bevel object, and moves the selected objects to it.

```
| -NCurve          +-Bevel
                  ==>  \ -NCurve
```

For selected boundaries, properly parameterised ExtrNC objects will be created:

```
| -NPatch          <----- .
| -NPatch          ==>  +-Bevel          |
                     +-ExtrNC          |
                     \ -Instance --'
```

Notes

See section 4.7.9 [Bevel Object \(page 208\)](#) for more information regarding the Bevel object.

This tool uses the object clipboard to move the objects so that the original clipboard contents are lost when this tool finishes.

Holding down the <Ctrl> key while clicking on the corresponding icon in the toolbox will keep the parameter curve selected and the current level will also be changed.

¹ Since 1.24.

5.4.8 Cap Tool

Arguments

The cap tool takes all selected objects or their selected boundaries.

Operation

The tool creates a Cap object, and moves the selected objects to it.

```
|-Outline (NCurve)          +-Cap
|-Hole (NCurve)             ==>  |-Outline (NCurve)
                                \-Hole (NCurve)
```

For selected boundaries, properly parameterised ExtrNC objects will be created:

```
|-NPatch          ==>  |-NPatch  <----- .
                      +-Cap      |
                      +-ExtrNC   |
                      \-Instance --'
```

Notes

See section [4.7.10 Cap Object \(page 211\)](#) for more information regarding the Cap object.

This tool uses the object clipboard to move the objects so that the original clipboard contents are lost when this tool finishes.

Holding down the <Ctrl> key while clicking on the corresponding icon in the toolbox will keep the parameter curve selected and the current level will also be changed.

5.4.9 Birail1 Tool

Arguments

The birail1 tool takes all selected objects or their selected boundaries.

Operation

The tool creates a Birail1 object, and moves the selected objects to it.

```
|-Cross_section (NCurve)    +-Birail1
|-Rail1 (NCurve)            ==>  |-Cross_section (NCurve)
|-Rail2 (NCurve)            \-Rail1 (NCurve)
                                \-Rail2 (NCurve)
```

For selected boundaries, properly parameterised ExtrNC objects will be created:

```
|-NPatch          ==>  |-NPatch  <----- .
                      +-Birail1  |
                      +-ExtrNC   |
                      \-Instance --'
```

Notes

See section [4.7.5 Birail1 Object \(page 195\)](#) for more information regarding the Birail1 object.

This tool uses the object clipboard to move the objects so that the original clipboard contents are lost when this tool finishes.

Holding down the <Ctrl> key while clicking on the corresponding icon in the toolbox will keep the parameter curve selected and the current level will also be changed.

5.4.10 Birail2 Tool

Arguments

The birail2 tool takes all selected objects or their selected boundaries.

Operation

The tool creates a Birail2 object, and moves the selected objects to it.

```
|-Cross_section1 (NCurve)      +-Birail2
|-Rail1 (NCurve)               |-Cross_section1 (NCurve)
|-Rail2 (NCurve)               ==>  |-Rail1 (NCurve)
|-Cross_section2 (NCurve)      |-Rail2 (NCurve)
                                \-Cross_section2 (NCurve)
```

For selected boundaries, properly parameterised ExtrNC objects will be created:

```
|-NPatch      ==>  |-NPatch  <-----,
                  +-Birail2      |
                  +-ExtrNC        |
                  \-Instance  --'
```

Notes

See section 4.7.6 [Birail2 Object \(page 198\)](#) for more information regarding the Birail2 object.

This tool uses the object clipboard to move the objects so that the original clipboard contents are lost when this tool finishes.

Holding down the <Ctrl> key while clicking on the corresponding icon in the toolbox will keep the parameter curve selected and the current level will also be changed.

5.4.11 Gordon Tool

Arguments

The gordon tool takes all selected objects or their selected boundaries.

Operation

The tool creates a Gordon object, and moves the selected objects to it.

```
|-NCurve      +-Gordon
|-NCurve      |-NCurve
|-Level       |-NCurve
|-NCurve      ==>  |-Level
|-NCurve      |-NCurve
               \-NCurve
```

For selected boundaries, properly parameterised ExtrNC objects will be created:

```

|-NPatch      ==>  |-NPatch  <----- .
                  +-Gordon    |
                  +-ExtrNC    |
                  \-Instance  --'

```

Notes

See section 4.7.8 [Gordon Object \(page 204\)](#) for more information regarding the Gordon object.

This tool uses the object clipboard to move the objects so that the original clipboard contents are lost when this tool finishes.

Holding down the <Ctrl> key while clicking on the corresponding icon in the toolbox will keep the parameter curve selected and the current level will also be changed.

5.4.12 Skin Tool

Arguments

The skin tool takes all selected objects or their selected boundaries.

Operation

The tool creates a Skin object, and moves the selected objects to it.

```

|-NCurve      +-Skin
|-NCurve      ==>  |-NCurve
|-NCurve      |-NCurve
                \-NCurve

```

For selected boundaries, properly parameterised ExtrNC objects will be created:

```

|-NPatch      +-Skin
|-NPatch      ==>  +-Skin
                  +-ExtrNC
                  \-Instance

```

Notes

See section 4.7.7 [Skin Object \(page 201\)](#) for more information regarding the Skin object.

This tool uses the object clipboard to move the objects so that the original clipboard contents are lost when this tool finishes.

Holding down the <Ctrl> key while clicking on the corresponding icon in the toolbox will keep the parameter curve selected and the current level will also be changed.

5.4.13 Trim Tool

Arguments

The trim tool takes the selected objects from the selection.

Operation

The tool creates a Trim object, and moves the selected objects to it.

-NPatch		+-Trim
-NCurve	==>	-NPatch
-NCurve		-NCurve
		\-NCurve

Notes

See section [4.7.15 Trim Object \(page 223\)](#) for more information regarding the Trim object.

This tool uses the object clipboard to move the objects so that the original clipboard contents are lost when this tool finishes.

Holding down the <Ctrl> key while clicking on the corresponding icon in the toolbox will keep the parameter curve selected and the current level will also be changed.

5.4.14 Tween Surfaces Tool

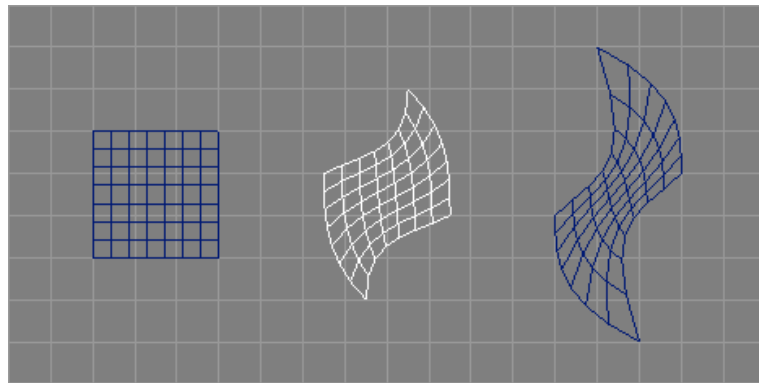


Figure 150: Tweened Surface (white) from two Parameter Surfaces (blue) with $r = 0.5$

Arguments

The tween tool takes two NURBS patches from the selection and requests a parameter r . In addition, an *append* option can be set.¹

Operation

The selected NURBS patches will be interpolated (tweened) and a new patch incorporating features from both of the original patches will be created. The parameter r defines the ratio of influence of the first and the second patch (the latter is using a ratio of $1 - r$).

```
|-NPatch1          |-NPatch1
|-NPatch2          ==> |-Tweened_Surface (NPatch)
                    |-NPatch2
```

If a third surface is selected, the parameter r is ignored and this third surface defines the ratio of influence with its y coordinates.² Unless the *append* option is set, the new surface will be created after the first surface.

Notes

The two patches must be of the same width, height, uorder, and vorder. They need not be defined on the same knot vectors, however.

The interpolation control surface does not need to be compatible with any of the original patches.

If one of the patches knot vector types is "Custom" or the respective knot vector types are different, the resulting knot vector type will be "Custom" and the knot values will also be interpolated/tweened. Otherwise a matching knot vector will be generated according to the type.

The original NURBS patches will not be deleted by this tool.

Trim curves are completely ignored by this tool.

See also the documentation of the corresponding scripting interface command [6.2.16 tweenNP](#) (page 405) and the related tool for curves [5.2.5 Tween Curves Tool](#) (page 278).

¹ Since 1.29. ² Since 1.25.

5.5 Surface Modification Tools

These tools modify parametric surface objects.

Unless noted otherwise, PV tags are *not* supported/modified by these tools.

5.5.1 Revert U Tool

Arguments

The revert u tool takes a number of NURBS patches, IPatch, BPatch, or PatchMesh objects from the selection.

Operation

The control point arrays of the selected objects will be reversed in the U dimension (width). For NURBS patches this tool also reverts the relative knot distances of the corresponding knot vector so that the surface does not change in shape. However, trim curves will not be modified.

For IPatch objects the respective derivatives are reverted.

Notes

PV tags will not be modified.

See also the documentation of the corresponding scripting interface command [6.2.14 revertuS](#) (page 392) and the related tool for curves [5.3.1 Revert Tool](#) (page 279).

5.5.2 Revert V Tool

Arguments

The revert v tool takes a number of NURBS patches IPatch, BPatch, or PatchMesh objects from the selection.

Operation

The control point arrays of the selected objects will be reversed in the V dimension (height). For NURBS patches this tool also reverts the relative knot distances of the corresponding knot vector so that the surface does not change in shape. However, trim curves will not be modified.

For IPatch objects the respective derivatives are reverted.

Notes

PV tags will not be modified.

See also the documentation of the corresponding scripting interface command [6.2.14 revertvS](#) (page 392) and the related tool for curves [5.3.1 Revert Tool](#) (page 279).

5.5.3 Swap UV Tool

Arguments

The swap uv tool takes a number of NURBS patches, IPatch, BPatch, or PatchMesh objects from the selection.

Operation

The U and V dimension of the selected objects will be swapped (width, height, and all other dimension related parameters will be exchanged) without altering the shape of the patches.

Notes

The point selection will be removed from the original objects.

PV tags will not be modified.

Also trim curves will not be modified.

See also the documentation of the corresponding scripting interface command [6.2.14 swapuvS](#) (page 392).

5.5.4 Close U Tool**Arguments**

The close u tool takes a number of NURBS patches from the selection and requests some operation parameters.

Operation

According to the "mode" parameter value, the surfaces will be closed by copying certain control point lines. If the "extend" parameter is enabled no old control points will be changed but the width of the patch will be increased.

5.5.5 Close V Tool**Arguments**

The close v tool takes a number of NURBS patches from the selection and requests some operation parameters.

Operation

According to the "mode" parameter value, the surfaces will be closed by copying certain control point lines. If the "extend" parameter is enabled no old control points will be changed but the height of the patch will be increased.

5.5.6 Refine Knots Surface Tool

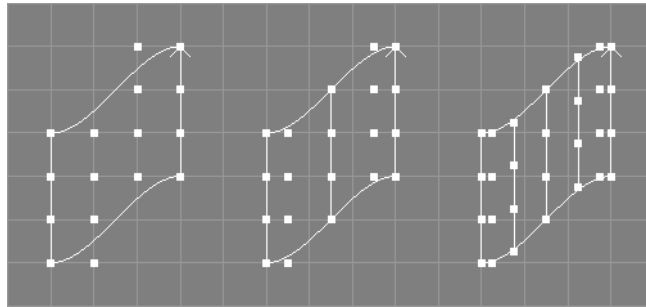


Figure 151: Successive Application of Refine Knots Surface (U) Tool

Arguments

The refine knots surface tool takes a number of NURBS patches from the selection.

Operation

The respective knot vectors of the selected NURBS patches will be refined by inserting a knot in the middle of each inner knot interval without changing the shape of the patches. New control points will be added and the position of old control points may be changed in the progress. See also the image above where a surface defined on the knot vector

$[0000 \ 1111]$

is successively refined to the knot vector

$[0000 \ 0.5 \ 1111]$

and then

$[0000 \ 0.25 \ 0.5 \ 0.75 \ 1111]$.

Notes

Because new knots are inserted into inner intervals only, the clamping state of the knot vectors does not change. Furthermore, knot vectors of type "NURB" will not change in type, other knot vectors will be changed to type "Custom".

The point selection will be removed from the original objects.

There are also tools available that refine only the U or V knots.

See also the documentation of the corresponding scripting interface commands [6.2.16 refineuNP](#) (page 403) and [6.2.16 refinevNP](#) (page 403) and the related tool for curves [5.3.5 Refine Knots Tool](#) (page 283).

5.5.7 Refine Knots With Surface Tool

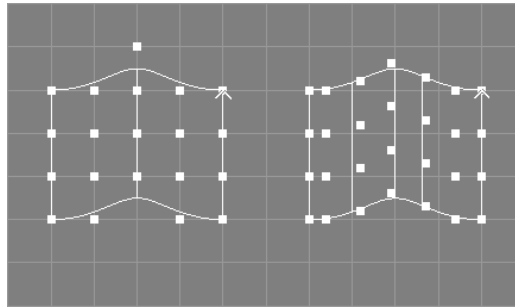


Figure 152: Refine Knots With Surface (U) Tool (left: original surface, right: refined surface)

Arguments

The refine knots with surface tool takes a number of NURBS patches from the selection and requests a vector of new knot values.

Operation

The respective knot vectors of the selected NURBS patches will be refined by inserting all knots from the specified vector at once. The shape of the patch will not be changed. New control points will be added and the position of old control points may be changed in the progress.

See also the example image above where a surface defined on the knot vector

`[0000 0.5 1111]`

has been refined with the new knots 0.2 and 0.7 in U direction to the new knot vector:

`[0000 0.2 0.5 0.7 1111]`.

The resulting knot vector must be valid, otherwise an error will be reported and the respective patch is not changed.

Notes

The respective knot type of the patch may be changed to "Custom".

The point selection will be removed from the original objects.

There are also tools available that refine only the U or V knots.

See also the documentation of the corresponding scripting interface commands [6.2.16 refineuNP](#) (page 403) and [6.2.16 refinevNP](#) (page 403) and the related tool for curves [5.3.6 Refine Knots With Tool](#) (page 284).

5.5.8 Elevate Surface Tool

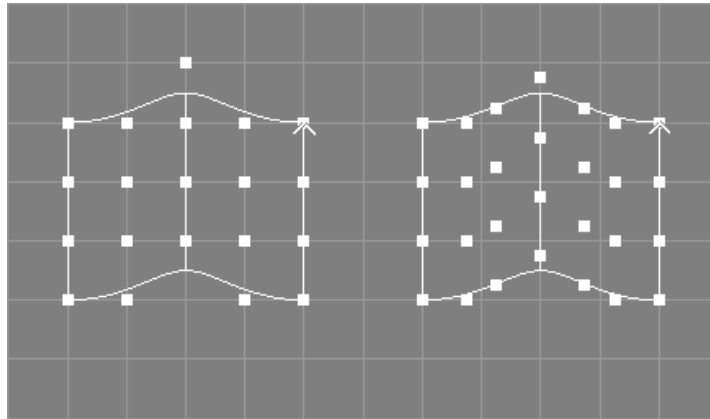


Figure 153: Elevate Surface Tool (left: original surface, right: elevated surface)

Arguments

The elevate surface tool takes a number of NURBS patches from the selection and additionally requests two integer values.

Operation

The order of the selected NURBS patches will be raised by the specified integer values without changing the shape of the patches. New control points will be added and the position of old control points may be changed in the progress.

See also the image above, where a surface of order four and width five, originally defined on the u knot vector:

```
[0000 0.5 1111]
```

has been elevated to order five in u direction and is now of width seven and defined on the u knot vector:

```
[00000 0.5 0.5 11111].
```

Notes

If the knot vector of the patch is not clamped, it will be clamped automatically.

The knot type of the patch will be changed to "Custom".

The point selection will be removed from the original objects.

There are also tools available that elevate a patch in U or V direction only.

See also the documentation of the corresponding scripting interface commands [6.2.16 elevatenuNP](#) (page 403) and [6.2.16 elevatevNP](#) (page 403) as well as the related tool for curves [5.3.8 Elevate Tool](#) (page 286).

5.5.9 Reduce Surface Tools

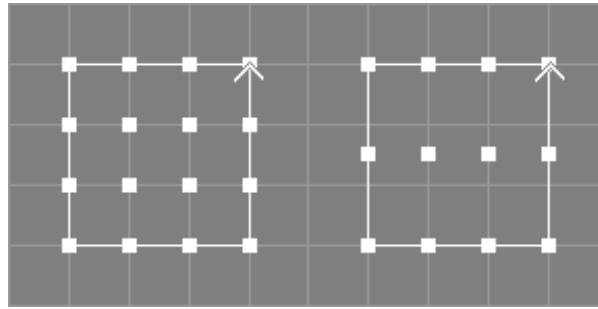


Figure 154: Reduce Tool (Order 4 (left), 3 (right))

Arguments

The reduce surface tools take a number of NURBS surfaces from the selection and additionally request a tolerance value.

Operation

The order of the selected NURBS surfaces will be decreased by one if the shape of the reduced surfaces does not deviate from the original surfaces by the given tolerance in any point.

See also the example image above where a surface of order four and height four, defined on the v knot vector

$[0000 \ 1111]$,

has been reduced in v direction to order three and is now of height three and defined on the v knot vector

$[000 \ 111]$.

Notes

If the respective knot vector of the surface is not clamped, it will be clamped automatically. The respective knot type of the surface will be changed to "Custom". Control points will be removed and the position of old control points may be changed in the progress.

The point selection will be removed from the processed objects.

The ability to reduce a certain surface can be improved by increasing the multiplicity of internal knots first.

See also the documentation of the corresponding scripting interface commands [6.2.16 reduceuNP](#) (page 404) and [6.2.16 reducevNP](#) (page 404) as well as the related tool for curves [5.3.9 Reduce Tool](#) (page 287).

5.5.10 Clamp Surface Tool

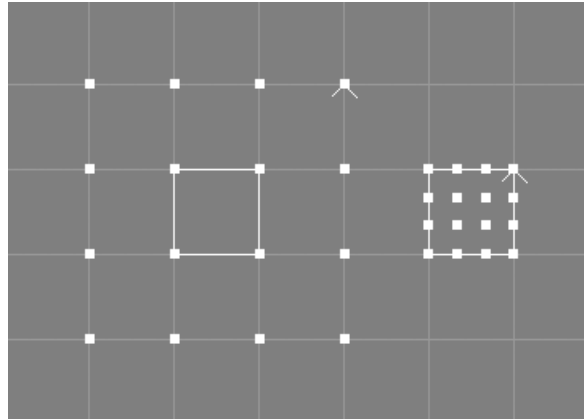


Figure 155: Clamp Surface Tool (left: Original Patch with B-Spline Knot Vectors, right: Clamped Patch)

Arguments

The clamp surface tool takes a number of NURBS patches from the selection.

Operation

The knot vectors of the selected NURBS patches will be changed using knot insertion so that the first and the last knot (in each direction) have a multiplicity equal to the order of the patch (in the respective direction).

See also the image above, where a surface of width/height four and both orders also four, originally defined on the B-Spline knot vectors:

```
[0 0.1428 0.2857 0.4285 0.5714 0.7142 0.8571 1.0]
```

has been clamped and is now defined on the knot vectors:

```
[0.4285 0.4285 0.4285 0.4285 0.5714 0.5714 0.5714 0.5714].
```

Notes

The shape of the patches will not change but the position of some control points will. The patches interpolate the first and the last control points in the respective direction afterwards (unless the weights of those points are not 1.0). The knot types of the patches will be changed to type "Custom".

There are also tools available that clamp a patch in U or V direction only.

See also the documentation of the corresponding scripting interface commands [6.2.16 clampuNP](#) (page 400) and [6.2.16 clampvNP](#) (page 400) and the related tool for curves [5.3.11 Clamp Tool](#) (page 289).

5.5.11 Unclamp Surface Tool

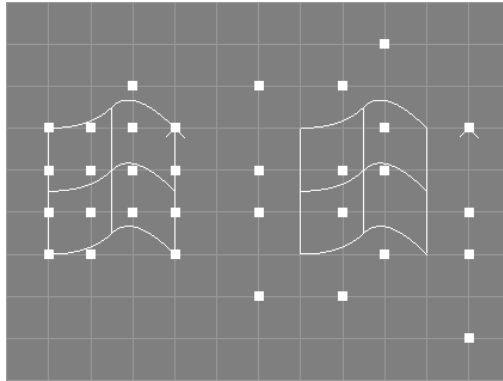


Figure 156: Unclamp Surface Tool (left: Original Patch, right: Unclamped Patch)

Arguments

The unclamp surface tool takes a number of NURBS patches from the selection.

Operation

The knot vectors of the selected NURBS patches will be changed so that there are no multiple knots at the respective ends of the knot vectors. The shape of the patches will not change but the position of some control points will. The knot types of the patches may be changed to type "Custom".

See also the image above, where a surface of width/height four and both orders three, originally defined on the knot vectors:

$[0\ 0\ 0\ 0.5\ 1\ 1\ 1]$

has been unclamped and is now defined on the knot vectors:

$[0\ 0.1666\ 0.3333\ 0.5\ 0.6666\ 0.8333\ 1]$.

Notes

Unclamping is *not* an exact reversal of clamping (and vice versa).

Furthermore, as the unclamp operation only works on completely clamped surfaces, the unclamp tool may need to clamp the surface first.

There are also tools available that unclamp a patch in U or V direction only.

See also the documentation of the corresponding scripting interface commands [6.2.16 unclampuNP](#) (page 400) and [6.2.16 unclampvNP](#) (page 401) and the related tool for curves [5.3.12 Unclamp Tool](#) (page 290).

5.5.12 Insert Knot Surface Tool

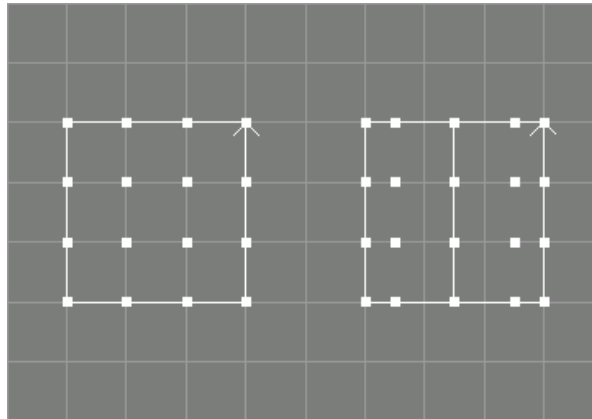


Figure 157: Insert Knot Surface Tool (left: original patch, right: a knot has been inserted one time at $t = 0.5$)

Arguments

The insert knot surface tool takes a number of NURBS patches from the selection and requests two additional values, a parametric value t and an integer value i .

Operation

A knot with the specified value t will be inserted i times into the knot vector of the selected patches, without changing their shape.

See also the image above, where a knot has been inserted at $u = 0.5$ one time into a surface of width four and order four that was originally defined on the u knot vector:

$[0000 \ 1111]$.

The processed surface of width five is defined on the u knot vector:

$[0000 \ 0.5 \ 1111]$.

Notes

The knot type of the patch will be changed to "Custom".

This tool does not change the geometry of the patches.

The point selection will be removed from the original objects.

See also the documentation of the corresponding scripting interface commands [6.2.16 insknuNP](#) (page 401) and [6.2.16 insknvNP](#) (page 402) and the related tool for curves [5.3.13 Insert Knot Tool](#) (page 291).

5.5.13 Remove Knot Surface Tool

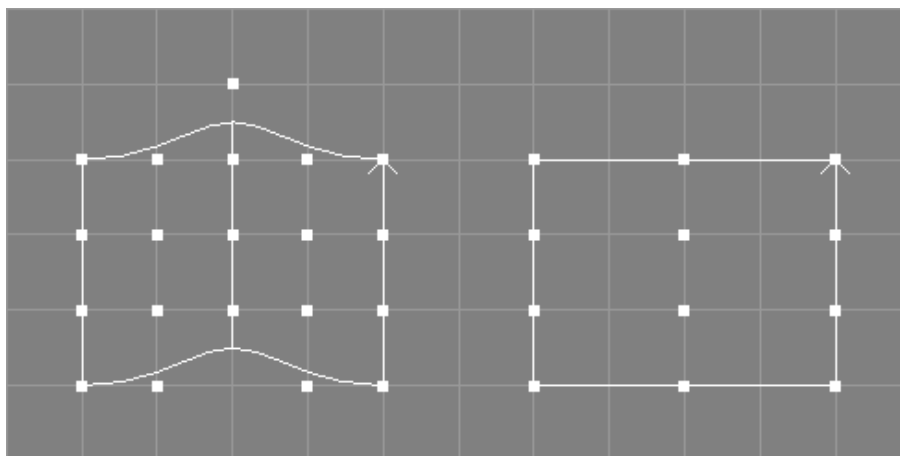


Figure 158: Remove Knot Surface Tool (left: original surface, right: the knot at $t=0.5$ has been removed one time)

Arguments

The remove knot surface tool takes a number of NURBS surfaces from the selection and requests three additional values, a parametric value t , an integer value r , and a tolerance value tol .

Operation

The knot at the specified parametric value t will be removed r times from the knot vector of the selected surfaces if the shape of the resulting surfaces does not deviate more than tol from the original surfaces in any point. Since Ayam 1.20 the knot to remove may also be specified using its (zero based) index in the knot vector; for instance by entering "-i 3" instead of "0.5" for the knot vector:

```
[000 0.5 111].
```

If the knot can not be removed r times due to the tolerance given, an error is reported and the original surface is left unchanged.

This operation also fails, if the knot removal would lead to a surface of lower order.

See also the image above, where the knot $t=0.5$ has been removed one time from a surface of width five and order four that was originally defined on the u knot vector:

```
[0000 0.5 1111].
```

The processed surface of width four is defined on the knot vector:

```
[0000 1111].
```

Notes

If tol is "0.0" the remove knot tool tries to work without changing the shape of the surface, i.e. only superfluous knots will be removed. If tol is "Inf" (infinity) the specified knot will be removed regardless of potential surface changes.

The point selection will be removed from the original objects.

See also the documentation of the corresponding scripting interface commands [6.2.16 remknuNP](#) (page 402), and [6.2.16 remknvNP](#) (page 402) and the related tool for curves [5.3.14 Remove Knot Tool](#) (page 292).

5.5.14 Remove Superfluous Knots Surface Tools

Arguments

The remove superfluous knots tools take a number of NURBS surfaces from the selection and requests a tolerance value *tol*.

Operation

All knots that do not contribute to the shape of the surface will be removed from the knot vector of the selected surfaces if the shape of the resulting surface does not deviate more than *tol* from the original surface in any point.

It is no error if no knots can be removed.

Notes

The point selection will be removed from the processed objects.

See also the documentation of the corresponding scripting interface commands [6.2.16 remsuknuNP](#) (page 402) and [6.2.16 remsuknvNP](#) (page 403). See also the related tool for curves [5.3.15 Remove Superfluous Knots Tool](#) (page 293).

5.5.15 Split Surface Tool

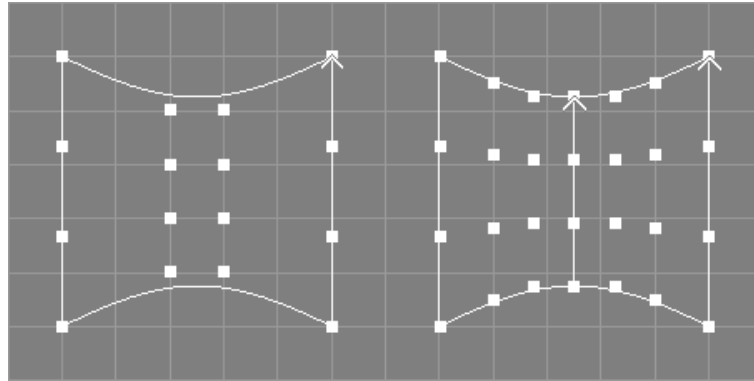


Figure 159: Split Surface Tool (left: original surface, right: the surface was split at $t=0.5$)

Arguments

The split surface tool takes a number of NURBS patches from the selection and requests a parametric value t (in U or V parametric dimension, respectively). In addition, a *relative* and an *append* option can be set.

Operation

The patches will be split at the parametric value t into two patches (in U or V parametric dimension, respectively) using knot insertion. The valid range of parametric values depends on the knot vector of the original surface unless the *relative* option is set, where it must be in the range (0.0, 1.0).

The new surface(s) will be inserted into the level right after the respective surface(s) to split unless the *append* option is set.¹

```
| -NPatch          | -NPatch
| -Sphere          ==> | -NPatch
                   | -Sphere
```

Notes

The original NURBS patch object(s) will be modified.

The point selection will be removed from the original objects.

Eventually present trim curves will *not* be honored properly.

See also the documentation of the corresponding scripting interface commands [6.2.16 splituNP](#) (page 404), [6.2.16 splitvNP](#) (page 404) and the related tool for curves [5.3.18 Split Tool](#) (page 297).

¹ Since 1.24.

5.5.16 Interpolate Surface Tool

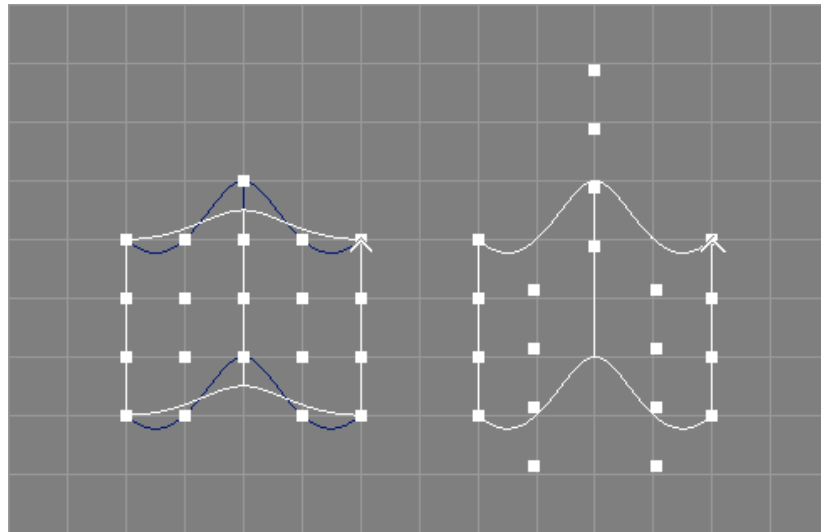


Figure 160: Interpolate Surface Tool (left-white: original surface, left-blue and right: interpolating surface)

Arguments

The interpolate surface tool takes a number of NURBS patches from the selection and requests an additional parameter o .

Operation

The patches will be modified so that they interpolate all original control points with the desired interpolation order o . See also the image above.

Notes

Eventually present weight information of the original patches will be ignored.

The point selection will be removed from the original objects.

See also the documentation of the corresponding scripting interface commands [6.2.16 interpuNP](#) (page 406) and [6.2.16 interpvNP](#) (page 406), as well as the related tool for curves [5.3.22 Interpolate Tool](#) (page 301).

5.5.17 Approximate Surface Tools

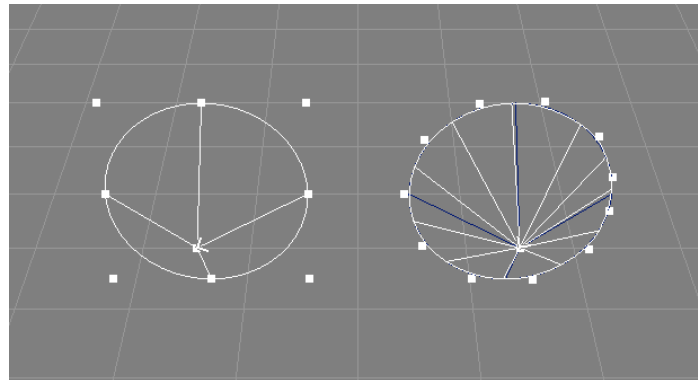


Figure 161: Approximate Surface Tool (left: original surface, right: approximating (white) and original (blue) surface superimposed)

Arguments

The approximate surface tools take a number of NURBS surfaces from the selection and request additional parameters. Depending on the approximation direction determined by the concrete tool selection, those are a subset or all of: width and height, orders, knot types, whether to create a closed surface, and tessellation parameters.

Operation

The surfaces will be modified so that they approximate their respective original surface shape(s) using a new non-rational NURBS surface with an adjustable number of new control points and order(s).

The centripetal knot types are better suited than the default (chordal) if the input data contains sharp features. The uniform knot types are optimal if regular/symmetric distribution of output control points is desired and the resulting surface is to be edited manually further on.

The data points to approximate are created by tessellating the original surfaces. The tessellation parameters control how fine this tessellation shall be, larger values lead to more data points and possibly higher approximation fidelity but also slower processing. A tessellation parameter of 0 disables the tessellation completely and the original control points will be used as approximation input.

See also the image above where a standard cone surface with nine rational control points in V direction of order three has been approximated by a periodic non rational surface with 11 control points and order three.

Notes

The point selection will be removed from the original objects.

Potentially present weights will be removed from the processed surfaces.

See also the documentation of the corresponding scripting interface command [6.2.16 approxNP](#) (page 407), and the related tool for curves [5.3.23 Approximate Tool](#) (page 302).

5.5.18 Fair Surface Tool

Arguments

The fair surface tool takes a number of NURBS surfaces from the selection and requests a mode and a tolerance value. In addition, a "Fair Worst" option can be set.

Operation

The control points of the surface will be modified so that the shape of the surface is somewhat smoother. The movement of a single control point is restricted by the specified tolerance value. This can also be used to apply the tool multiple times until the surface shape meets the requirements.

As the surface fairing algorithm is derived from the curve fairing algorithm, the option "Mode" allows to select in which direction the algorithm shall be applied, or, for the modes "UV" and "VU", in which order the dimensions shall be processed.

If the option "Fair Worst" is enabled, only the control point that would be moved the longest distance in each control line in the currently processed direction will be modified.

If there are selected points, only those will potentially be modified.

Notes

The fairing algorithm uses the four direct and indirect neighboring control points of each point to process. This poses a problem for the end points of an open surface. Therefore, the control vector of open surfaces will be extended by a simple linear extrapolation algorithm, so that every point of the original surface has four neighbors. However, the fairing quality for the first, second, second to last, and last control points of an open surface will be somewhat lower.

The original algorithm only delivers optimal results for cubic surfaces (of order four), using surfaces of other degrees will work too but yield worse results.

Due to the fact that the algorithm uses already faired points as input points for other points to fair, it is *not* direction invariant if used on all or multiple neighboring points. Also, control points already placed on very good if not optimal positions may be moved away to less favorable positions. Further problems arise for multiple control points, used to model sharp corners, as the fairing algorithm does not differentiate between wanted and unwanted non-continuous surface features. If in doubt, just select a single control point to fair.

See also the documentation of the corresponding scripting interface command [6.2.16 fairNP](#) (page 410) and the related tool for curves [5.3.16 Fair Tool](#) (page 294).

5.5.19 Make Surfaces Compatible Tool

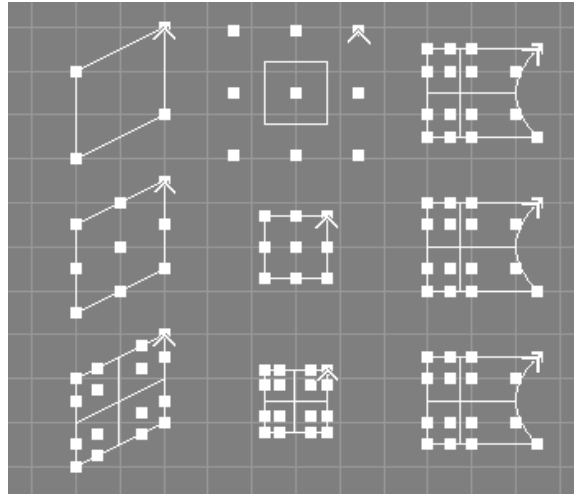


Figure 162: Make Compatible Tool (u: Original, m: Compatible by Order, l: Fully Compatible Surfaces)

Arguments

The make compatible tool takes a number of NURBS surfaces from the selection and requests a side and a level.

Operation

The surfaces will be made compatible, so that, based on the side and level parameters given, they are of the same orders, width/height, and defined on the same knot vector.

Notes

This tool does not change the geometry of the surfaces. However, since clamping, degree elevation, and knot insertion may be used on the surfaces, their order, knot vectors, and control points may be changed.

The orders of all surfaces will simply be raised to the maximum order of all surfaces; no attempt is made to check, whether lowering orders would lead to a simpler result.

The point selection will be removed from the original objects.

See also the documentation of the corresponding scripting interface command [6.2.16 makeCompNP](#) (page 409) and the related tool for curves [5.3.26 Make Compatible Tool](#) (page 305).

5.5.20 Rescale Knots to Range Surface Tool

Arguments

The rescale knots to range surface tool takes a number of NURBS patches from the selection and requests a range.

Operation

The knot vectors of the patches will be scaled, so that their first and last values match the given range. Trim curves, if present, will also be scaled to match the new range.

Notes

Since Ayam 1.20 the knot type of the surface does not have to be "Custom" anymore. Furthermore, rescaling the knots does not change the knot type.

This tool does not change the geometry of the patches.

See also the documentation of the corresponding scripting interface command [6.2.16 rescaleknNP](#) (page 401) and the related tool for curves [5.3.27 Rescale Knots to Range Tool](#) (page 306).

5.5.21 Rescale Knots to Mindist Surface Tool

Arguments

The rescale knots to mindist surface tool takes a number of NURBS patches from the selection and request a minimum distance value.

Operation

The knot vectors of the patches will be scaled, so that no two knots have a distance smaller than the given minimum distance (except for multiple knots). Trim curves, if present, will also be scaled to match the new range.

Notes

Since Ayam 1.20 the knot type of the surface does not have to be "Custom" anymore. Furthermore, rescaling the knots does not change the knot type.

This tool does not change the geometry of the patches.

See also the documentation of the corresponding scripting interface command [6.2.16 rescaleknNP](#) (page 401) and the related tool for curves [5.3.28 Rescale Knots to Mindist Tool](#) (page 306).

5.6 Conversion Tools

These tools convert between parametric objects.

5.6.1 Extract Curve Tool

Arguments

The extract curve tool takes the first of the selected objects from the selection.

Operation

The tool creates an instance from the first of the selected objects then creates an ExtrNC object and moves the instance to it.

```
|-NPatch ==> |-NPatch
              +-ExtrNC
              \-Instance_of_NPatch (Instance)
```

Notes

This tool uses the object clipboard to move the objects around so that the original clipboard contents are lost when this tool finishes.

See section [4.5.2 ExtrNC Object \(page 165\)](#) for more information regarding the ExtrNC object.

5.6.2 Extract Patch Tool

Arguments

The extract patch tool takes the first of the selected objects from the selection.

Operation

The tool creates an instance from the first of the selected objects then creates an ExtrNP object and moves the instance to it.

```
|-NPatch ==> |-NPatch
              +-ExtrNP
              \-Instance_of_NPatch (Instance)
```

Notes

This tool uses the object clipboard to move the objects around so that the original clipboard contents are lost when this tool finishes.

See section [4.7.12 ExtrNP object \(page 217\)](#) for more information regarding the ExtrNP object.

See also the documentation of the corresponding scripting interface command [6.2.16 extrNP \(page 405\)](#).

5.6.3 Break into Curves Tool

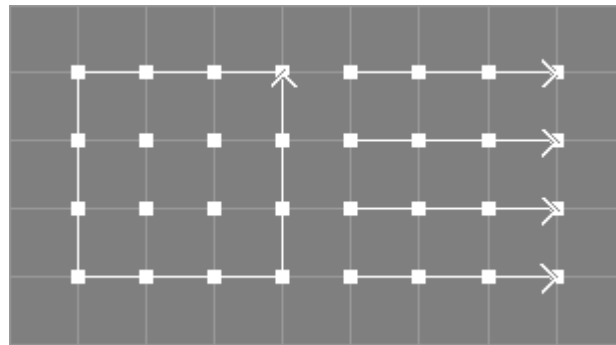


Figure 163: Break into Curves Example

Arguments

The break into curves tool takes all NURBS patches from the selection and requests some parameters.

Operation

The selected NURBS patches will be broken into NURBS curves, along the given direction (U or V). If the option "ApplyTrafo" is set, the transformations of the NPatch objects will be applied to the control points and the NCurve objects will be created with default transformation attributes, otherwise the NCurve objects will get the transformation attributes of the respective NPatch. See also the image above.

If the option "ReplaceOriginal" is enabled, the curves will replace each original NPatch object, however, by default new curve objects will be appended to the current level:

```
|-NPatch                                |-NPatch
                                     |-NCurve
                                     ==> |-NCurve
                                     |-NCurve
                                     |-NCurve
```

Notes

See also the documentation of the corresponding scripting interface command [6.2.16 breakNP](#) (page 408).

5.6.4 Build from Curves Tool

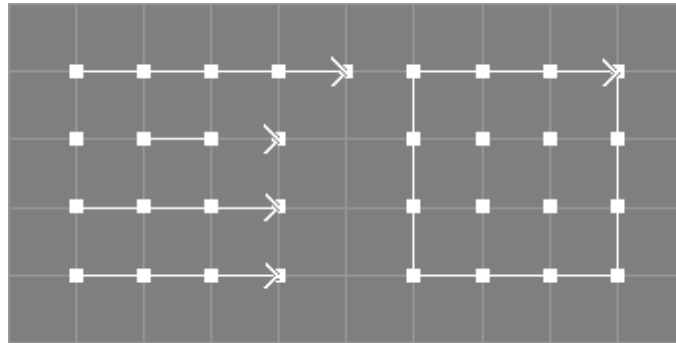


Figure 164: Build from Curves Example

Arguments

The build from curves tool takes a number of NURBS curves from the selection and requests some parameters.

Operation

The selected NURBS curves will be parsed, all curves that are of equal length or longer than the first selected curve will be used to form a new NURBS patch of the following dimensions:

Width: number of used curves, Height: length of the first selected curve. If the supplied order is 0 (this is the default), the order of the new surface in U direction (Order_U) will be equal to the number of used curves for numbers of two to four and four for bigger numbers of used curves.

The knot type in U direction (Knot-Type_U) will always be NURB. Other parameters (Order_V, Knot-Type_V, Knots_V) are taken from the first curve.

See also the image above; note, that the extraneous control point of the upper curve and the knot vector of the curve below are ignored. To avoid this, the make compatible tool can be used first, see section 5.3.26 Make Compatible Tool (page 305).

If the "ApplyTrafo" option is enabled the transformation attributes of the individual curve objects will be applied to the control points, otherwise they will simply be ignored.

If the "ReplaceOriginal" option is enabled, the new surface object will replace the first of the curves and the other curves will be removed. However, by default the new NPatch object will be appended to the current level:

```

|-NCurve          |-NCurve
|-NCurve          |-NCurve
|-NCurve          ==> |-NCurve
|-NCurve          |-NCurve
                  |-NPatch

```

Notes

See also the documentation of the corresponding scripting interface command 6.2.16 buildNP (page 408).

5.6.5 Tessellation Tool

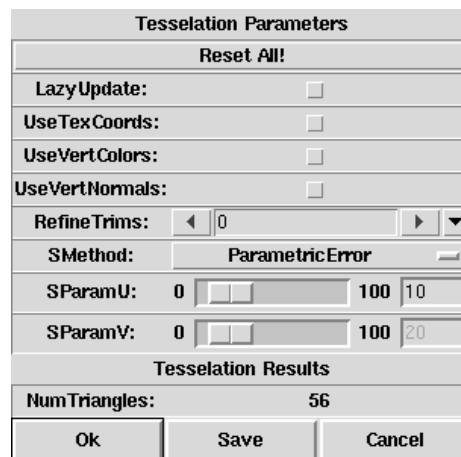


Figure 165: GUI of Tessellation Tool

Arguments

The tessellation tool takes all NURBS patches and NURBS patch providing objects from the selection.

Operation

A modal dialog box (see image above) will pop up, that allows to select a tessellation method via a drop-down menu and to tune the corresponding tessellation parameter(s) using a slider and an entry widget. The initial method and parameter values will be derived from the "TP" tag of the first of the selected objects (if it has such a tag).

The selected or provided NURBS patches will be tessellated with the chosen method and parameters. The PolyMesh objects created by the tessellation will immediately be displayed in all view windows instead of the original objects. Whenever tessellation method or parameters are changed, the tessellation will be recomputed and displayed, thus, allowing an immediate estimation of the tessellation quality and the number of created polygonal elements.

If the option "LazyUpdate" is enabled, updates of the tessellation that normally occur while dragging the slider(s) will be deferred until the mouse button is released. The initial value of this option will be determined from the "LazyNotify" preference setting.

The options "UseTexCoords" and "UseVertColors" control processing of texture coordinates and vertex colors stored as PV tags in the objects to be tessellated (see also section 4.11.4 PV Tag (page 262)). When enabled, the resulting PolyMesh objects will also have PV tags of the respective type. Note that the PV tags must be named as defined by the hidden preference options "PVTexCoordName" and "PVColorName" respectively. By default those are set to "st" and "Cs".

If the "UseTexCoords" option is enabled, but no matching PV tag is present, texture coordinates will be generated from the data provided by a TC tag or (if no TC tag is present) from the knot values of the NURBS surfaces to be tessellated.¹ See also section 4.11.3 TC (Texture Coordinates) Tag (page 260).

The option "UseVertNormals" controls processing of vertex normals stored as PV tags in the objects to be tessellated (see also section 4.11.4 PV Tag (page 262)). When enabled, the resulting

¹ Since 1.21.

PolyMesh objects will have normals derived from these tags instead of normals derived from the NURBS surface.

Note that the PV tags must be named as defined by the hidden preference option "PVNormalName". By default this is set to "N", so that a correct example tag looks like this:

```
PV N,varying,n,4,0,0,1,0,0,1,1,1,0,0,0,1
```

The option "RefineTrims" controls how many times the trim curves are to be refined before tessellation for improved tessellation fidelity along trim edges.¹ See also the images below.

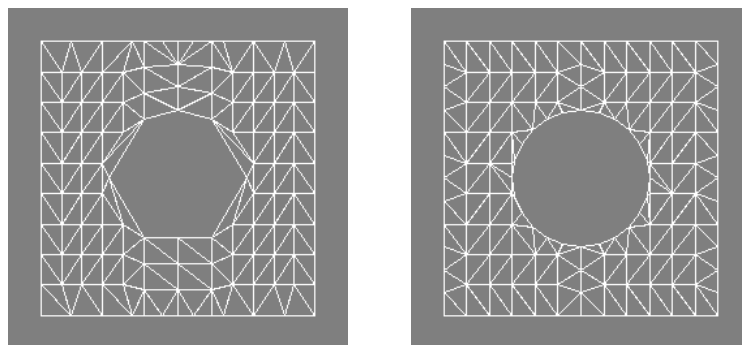


Table 100: RefineTrims Example (left: 0, right: 2)

The option "Primitives" allows to select a primitive creation method. These methods are explained in greater detail in section 5.6.5 Primitive Creation Methods (page 342).

If the "Ok" button is pressed, the tessellation tool will be closed and all selected objects will be replaced by their tessellated counterparts. The original objects will be deleted.

If referenced/master objects are among the selected objects or their children, and if any of their references are not selected or a child of the selected objects (i.e. would not be deleted also), then those master objects will not be deleted but moved to the clipboard instead.

If the "Save" button is used, the tessellation dialog will be closed and "TP" tags containing the currently selected method and parameter value will be added to all selected objects. These tags can be evaluated later, when the respective objects are converted to PolyMesh objects (e.g. upon export). Since Ayam 1.11 the conversion mechanism of objects keeps the "TP" tags intact, so that e.g. tessellation parameters saved to a Sphere or a Revolve object will also be preserved, and can also be used later when the object will eventually be converted to a PolyMesh (via a NURBS patch).

If "Cancel" is used, the dialog will be closed and all selected objects remain unchanged.

Notes

The tessellation tool will block most other parts of Ayam while it is running, i.e. it is not possible to change the selection or run other tools. It is, however, allowed to adjust view parameters while the tessellation tool is open, to examine the tessellation result more closely or from different viewing angles.

The values of the parameter slider bounds may be changed by simply entering values that are out of the current bounds into the respective entry widget, then pressing the <Tab> key. Range and resolution of the slider will be recalculated automatically from the value in the entry widget.

PV tags that do not provide enough data for the surface to be tessellated will be silently ignored.

¹ Since 1.21.

The PolyMesh objects created by this tool will *not* be optimized.

The tessellation facility is based on the GLU (V1.3+) NURBS tessellator.

Sampling Methods and Parameters

Six sampling methods are available:

1. "ParametricError" ensures that the distance between the tessellated surface and the original surface is no point bigger than the value specified by "SParamU".
2. The sampling method "PathLength" ensures that no edge of a polygon generated by the tessellation is longer than the value specified by "SParamU" and the tessellation method
3. "DomainDistance" (the default up to 1.22) simply tessellates the NURBS surface into equally sized pieces with regard to parametric space; "SParamU" and "SParamV" control the number of sampling points in U and V direction respectively per unit length. This leads to different numbers of samples for knot vectors of different total length in parameter space.
4. "NormalizedDomainDistance" ensures that the tessellation creates the same number of sample points (as given via "SParamU" and "SParamV") for knot vectors of any total length in parameter space¹ and
5. "AdaptiveDomainDistance" additionally adds sample points depending on the number of control points (width or height of the patch) to provide a better adaptation to complex patches.²
6. "AdaptiveKnotDistance" normalizes the number of sample points to the number of knot intervals and the total length of the knot vector.³

"SParamU" is a parameter for the sampling method above.

The default value for the sampling method "AdaptiveKnotDistance" is 3. Higher values lead to better quality and more tessellated polygons.

The default value for the sampling method "DomainDistance" is 8. Higher values lead to better quality and more tessellated polygons.

The default value for the sampling method "PathLength" is 1.5. Smaller values lead to better quality and more tessellated polygons.

The default value for the sampling method "ParametricError" is 0.25. Smaller values lead to better quality and more tessellated polygons.

Note that "SParamU" is expressed in object space units for the "PathLength" and "ParametricError" sampling methods.

"SParamV" is just available for the sampling methods "DomainDistance", "NormalizedDomainDistance", "AdaptiveDomainDistance", and "AdaptiveKnotDistance".

The default value is equal to the respective value of the "SParamU" parameter above.

See also the next two images and corresponding tables that allow to compare the results of four main sampling methods with different parameters.

¹ Since 1.9. ² Since 1.9. ³ Since 1.23.

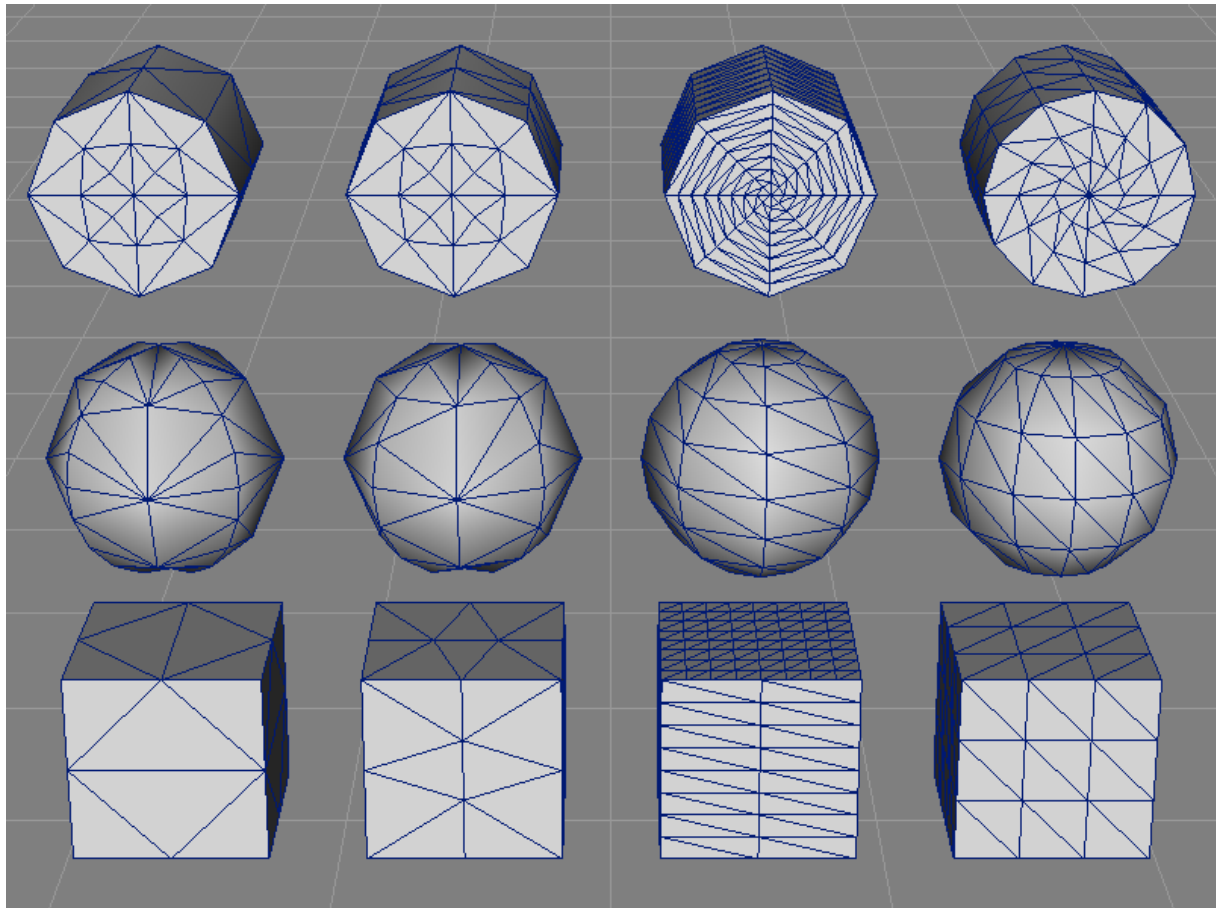


Figure 166: Sampling Method Comparison, Defaults (l: ParametricError, ml: PathLength, mr: DomainDistance, r: AdaptiveKnotDistance)

Method	ParametricError	PathLength	DomainDistance	AdaptiveKnotDistance
Parameter	0.25	1.5	8 / 8	3 / 3
Cylinder	88	104	368	192
Sphere	96	80	112	120
Box	36	60	384	108
Total	220	244	864	420

Table 101: Sampling Methods Parameters and Results (1/2)

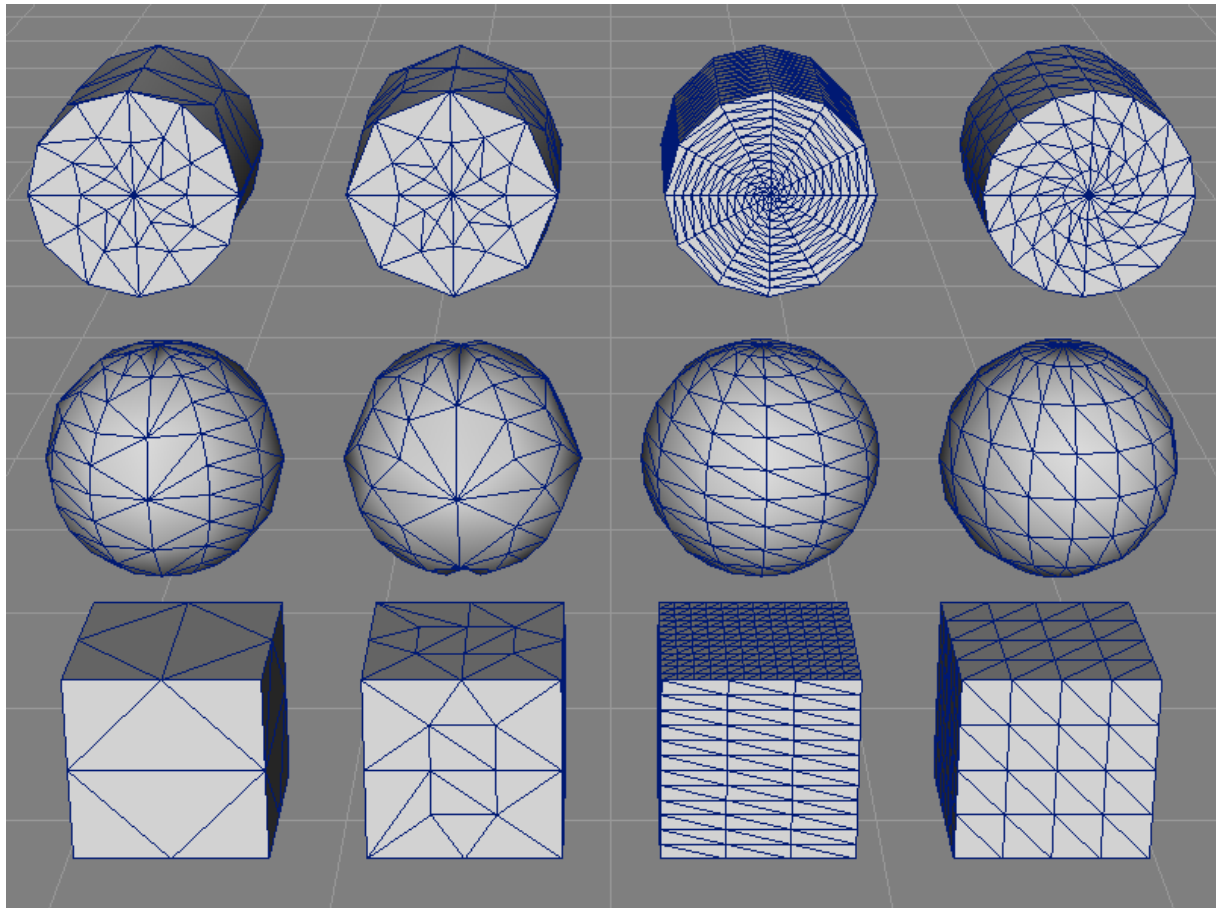


Figure 167: Sampling Method Comparison, Higher Quality (l: ParametricError, ml: PathLength, mr: DomainDistance, r: AdaptiveKnotDistance)

Method	ParametricError	PathLength	DomainDistance	AdaptiveKnotDistance
Parameter	0.125	0.9	12 / 12	4 / 4
Cylinder	152	168	840	352
Sphere	216	160	264	224
Box	36	108	864	192
Total	404	436	1968	768

Table 102: Sampling Methods Parameters and Results (2/2)

Primitive Creation Methods

The primitive creation methods determine, how the OpenGL primitives sent from the GLU NURBS tessellator (`GL_TRIANGLES`, `GL_TRIANGLE_FAN`, `GL_TRIANGLE_STRIP`, and `GL_QUAD_STRIP`) are turned into polygon faces. Four primitive creation methods are available:

1. "Triangles": the output contains only triangles. Even if quads are originally emitted from the tessellator, those will be torn down to triangles.
2. "TrianglesAndQuads": the output contains triangles and possibly quads (as originally emitted from the tessellator).¹ Note that quads will be emitted from the tessellator with higher probability in the *DomainDistance* and *KnotDistance* sampling methods.
3. "Quads": the output only contains quads that will be synthesized from neighboring triangles if they form a planar quad. The pairing algorithm is not very sophisticated, only one solution will be tried. Furthermore, triangles without suitable neighbor will be used to form degenerate quads.²
4. "QuadrangulatedTriangles": the output contains only quads that stem from refined/quadrangulated triangles.³ The input of the quadrangulation are the triangles as created by the "Triangles" method. Each triangle will be refined/replaced by three quads. Four new points will be introduced for that. These new points will be placed directly on the (still) underlying NURBS surface. The quads are guaranteed not to be degenerate (contrary to those created by the "Quads" method) but they may be non-planar if the tessellated surface is highly curved and the initial sampling is sufficiently coarse.
Degeneracies will be detected and proper actions with regard to normal interpolation will be taken.
Vertex colors supplied via PV tags are currently not supported by this method.

See also the following images and tables for a comparison of the primitive creation method results.

¹ Since 1.23. ² Since 1.23. ³ Since 1.30.

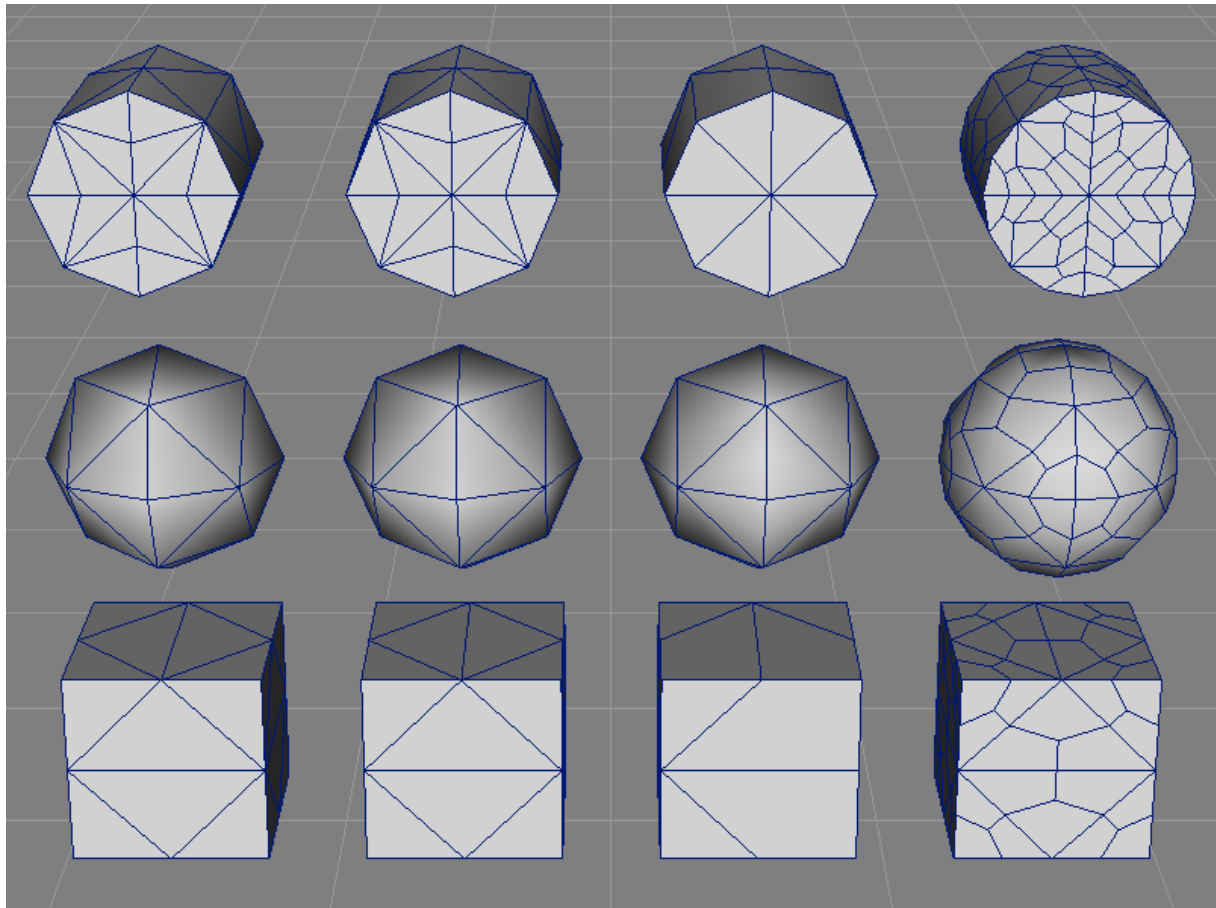


Figure 168: Primitive Creation Method Comparison, Sampling Method: Parametric Error (0.65) (l: Triangles, ml: TrianglesAndQuads, mr: Quads, r: QuadrangulatedTriangles)

Method	Triangles	TrianglesAndQuads	Quads	QuadrangulatedTriangles
Cylinder	56t	56t	32q	168q
Sphere	32t	32t	32q	96q
Box	36t	36t	24q	108q
Total	124t	124t	88q	372q

Table 103: Primitive Creation Method Comparison (1/2)

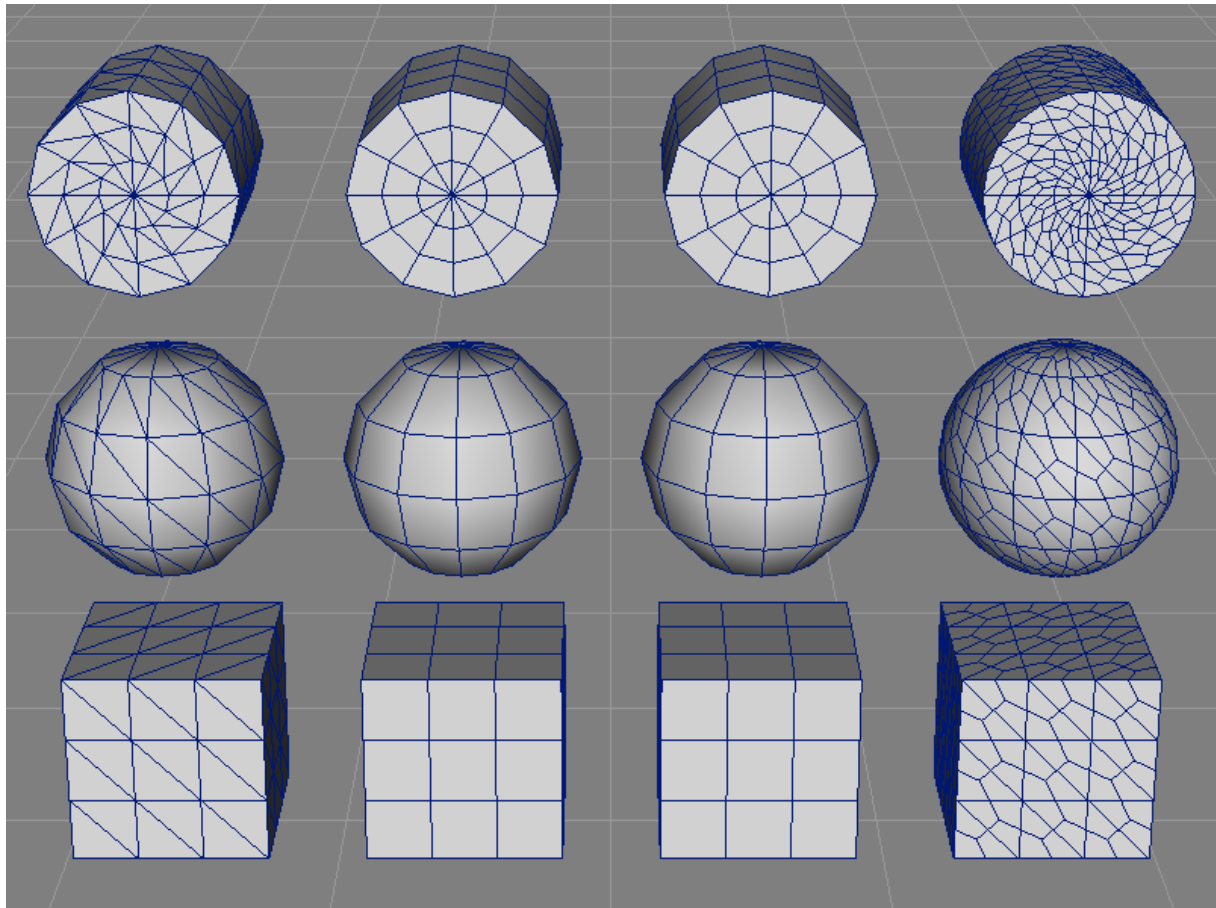


Figure 169: Primitive Creation Method Comparison, Sampling Method: AdaptiveKnotsDistance (3.0/3.0)
 (l: Triangles, ml: TrianglesAndQuads, mr: Quads, r: QuadrangulatedTriangles)

Method	Triangles	TrianglesAndQuads	Quads	QuadrangulatedTriangles
Cylinder	56t	24t / 84q	100q	576q
Sphere	120t	24t / 48q	72q	360q
Box	108t	54q	54q	324q
Total	284t	48t / 186q	226q	1260q

Table 104: Primitive Creation Method Comparison (2/2)

6 Scripting Interface

The Ayam scripting interface consists of a number of Tcl procedures and Tcl commands that are also used internally by the application. For instance, the main menu entry "File/New" calls the scripting interface command "newScene" (among other commands). Using the Ayam scripting interface means to call these procedures or commands, possibly in a mix with standard Tcl script code.

Furthermore, using Tcl and its introspection capabilities, the code Ayam consists of could easily be modified at runtime. This is, however, not recommended for good reasons (unless intimate knowledge about the Ayam source code is obtained first). So watch out for already existing procedures and commands when implementing your own. Using procedures and commands not listed in this documentation is dangerous too. Implementation and interfaces of those procedures and commands may change in future versions of Ayam without notice.

In Tcl, all variables, procedures, and commands are case sensitive, it really is "sL" and *not* "s1" and *not* "SL".

The scripting interface may be used directly from the console of Ayam. One can, of course, also put scripts in Tcl script files, that may be loaded at any time into Ayam using the console and the Tcl command "source". Script files can also be made to run on every application startup automatically using the preference setting "Main/Scripts". Moreover, on the X11 and Aqua window systems, Ayam is able to execute script code sent via the Tk "send" command or the AppleScript "tell" command from external applications.

In contrast to other modelling environments, in Ayam there is yet another way to run scripts. In Ayam, scripts may also be attached to Script objects and run when the notification mechanism updates the scene. See also section [4.9.1 Script object \(page 229\)](#). Even normal objects can trigger scripts upon notification using BNS or ANS tags. See also sections [4.11.15 Before Notify Script \(page 267\)](#) and [4.11.16 After Notify Script \(page 268\)](#).

Note that most of the scripting interface commands listed in this documentation work in the background, without changing anything to the Ayam GUI and Ayam view windows, for the sake of execution speed. To make any changes to the scene visible, the various parts of the GUI (property GUIs, view windows) need to be updated explicitly (see also section [6.2.20 Updating the GUI \(page 420\)](#)).

However, since Ayam 1.13 it is also possible to automatically run GUI updating commands in the console with every issued command by using <Shift+Return> instead of <Return>.

Also note that even though no updates of the GUI take place when using the scripting interface, all notification processes are carried out immediately regardless. The scene will be consistent and up to date when the scripting interface command returns.

Scene changes from the scripting interface can also be recorded in the undo buffer, but this must be arranged manually too (see the documentation of the undo command: [6.2.28 Undo \(page 434\)](#)).

From scripts it may be necessary to check whether an error occurred during the execution of a command. All commands return TCL_OK in *any* case, so checking their return value avails to nothing, but they set the global Tcl variable "ay_error" to a value higher than 1 if an error occurred. This variable needs to be set to zero before and checked after the operation in question to see whether the operation performed successfully, see the following example:

```
proc myProc { } {  
    set ::ay_error 0  
    copOb  
    if { $::ay_error > 1 } {  
        ayError 2 "myProc" "Error copying object!"  
    }  
}
```

6.1 Global Variables and Arrays

Several global variables and arrays exist in the Ayam Tcl context, that may be useful for scripts.

6.1.1 Global Variables

The global variable "ay_error" holds the current error state. See also section 6.2.26 Reporting Errors (page 424).

The global variable "i" is used by all "forAll" command variants. See also section 6.2.23 Applying Commands to a Number of Objects (page 422).

The global variables "u" and "v" are set by the find u/uv actions. See also sections 3.20 Finding Points on Curves (page 99) and 3.21 Finding Points on Surfaces (page 100).

6.1.2 The Global Array ay

The global array "ay" holds application state variables. Furthermore, the paths to important widgets can be found (e.g. the tree widget for the object hierarchy or the currently active view) in this array. Use "parray ay" in the console to see what is there. More documentation to come.

6.1.3 The Global Array ayprefs

The global array "ayprefs" holds preferences data. The *complete* array is saved in the *ayamrc* file upon exit, so be careful when adding new elements to this array. See also section 8.4 Ayamrc File (page 516). Use "parray ayprefs" in the console to see what is there. More documentation to come.

Note that changes to this array on the Tcl side do not immediately take effect as the data needs first to be transferred to the C context using the "setPrefs" command. See also section 6.2.21 Managing Preferences (page 421).

6.1.4 The Global Array aymark

The global array "aymark" holds a copy of the current mark position.¹ It is updated whenever the mark is set using e.g. the set mark action (see section 3.5 Setting the Mark (page 82)). Manual changes to this array have no effect on the mark.

¹ Since 1.21.

6.1.5 Property Management and Data Arrays

For every object type, a corresponding global variable exists, that contains the property names of this object type as a list. The variable name is just the object type name followed by `_props` (for properties). For example for the `NCurve` object type, this variable is named `"NCurve_props"` and the list it contains looks like this:

```
{ Transformations Attributes Tags NCurveAttr }
```

This list is consulted each time a single object is selected in the tree view or object list widget and its content is used to populate the properties listbox widget. See also section [2.1.2 Properties \(page 24\)](#). Note that the list can be further manipulated by `"NP"` and `"RP"` tags of the selected object.

For every property, a corresponding global array exists, where the property is managed. For the Transformations property, this array looks like this:

```
Transformations {
  arr  transfPropData
  sproc setTrafo
  gproc getTrafo
  w     fTrafoAttr
}
```

The first entry, `"arr"`, designates the name of the global property data array (thus, transformation data is stored in an array called `"transfPropData"`). This array only holds useful data when it has been filled explicitly by the so called get-property callback.

The entries `"sproc"` and `"gproc"` designate the set-property and get-property callbacks (procedures or commands). Those are called, when the `"Apply"` button is used or the property is selected in the properties listbox respectively. If `sproc` or `gproc` are empty strings (`" "`), standard callbacks named `"setProp"` or `"getProp"` will/should be used to get or set the property values. But for the Transformations property, the `"setTrafo"` and `"getTrafo"` commands should be used. The flexibility gained by individual `sproc/gproc` procedures is used to sanitize user input, run additional GUI update code or realize dynamic property GUIs.

The last entry, `"w"`, is the name of the main property GUI window. To get the full, usable, widget path of this window, the current value of the array entry `"ay(pca)"` needs to be prepended: `$ay(pca).$Transformations(w)`.

Note that for many object types the property variable and the object type specific property management arrays only exist after an object type specific initialization procedure, derived from the type name, is called, e.g. `"init_Box"`. Those types are: `ACurve`, `Bevel`, `Birail1`, `Birail2`, `Box`, `BPatch`, `Cap`, `ConcatNC`, `ConcatNP`, `Cone`, `Cylinder`, `Disk`, `ExtrNC`, `ExtrNP`, `Hyperboloid`, `ICurve`, `IPatch`, `NCircle`, `OffsetNC`, `OffsetNP`, `PatchMesh`, `Paraboloid`, `Revolve`, `RiInc`, `RiProc`, `Script`, `SDMesh`, `Select`, `Sphere`, `Sweep`, `Swing`, `Torus`, and `Trim`.

Since Ayam 1.16, the global property management array may be created easily using the new scripting interface command `"addPropertyGUI"`. See also section [6.2.27 Property GUI Management \(page 425\)](#).

The following global arrays and callbacks to get or set the data exist:

property	array	get-property callback	set-property callback
Transformations	transfPropData	getTrafo	setTrafo
Attributes	attrPropData	getAttr	setAttrp
Material	matPropData	getMat	setMat
Tags	tagsPropData	getTagssp	setTagssp
MaterialAttr	MaterialAttrData	""	setMaterialAttrp
Surface	ay_shader	shader_getSurf	shader_setSurf
Displacement	ay_shader	shader_getDisp	shader_setDisp
Interior	ay_shader	shader_getInt	shader_setInt
Exterior	ay_shader	shader_getExt	shader_setExt
Light	ay_shader	light_getShader	light_setShader
LightAttr	LightAttrData	light_getAttr	""
ViewAttrib	ViewAttribData	""	setViewAttr
Camera	CameraData	""	setCameraAttr
NCurveAttr	NCurveAttrData	""	""
NPatchAttr	NPatchAttrData	""	""
ICurveAttr	ICurveAttrData	""	""
NCircleAttr	NCircleAttrData	""	""

Table 105: Property Arrays and Callbacks

Note that this list is pretty much incomplete, however the information can always be inferred easily using the "parray" command in the Ayam console:

```
»parray NCurveAttr
```

See also section [6.2.7 Manipulating Properties \(page 379\)](#) for more information on how to edit property values from the scripting interface.

6.2 Procedures and Commands

This section provides documentation on the most important scripting interface procedures and commands of Ayam sorted by category.

Note that the "help" command in the Ayam console can be used to directly jump to the appropriate subsection of this part of the documentation.

All procedures and commands are documented in the following scheme:

- **Synopsis:** "command param1 param2 [optionalparam1]" (syntax of the command and its parameters),
- **Background:** does the command run in the background, **Undo:** can the result of the command be undone, **Safe:** is the command available in the safe interpreter (for Script objects and notify script tags), **Type:** Procedure – this is actually not a command but a procedure implemented in Tcl
- **Description:** detailed description of the command and its parameters,
- **Notes:** additional information completing the detailed description,
- **Example:** "command 1 2" (example application of the command with explanation of expected results).

6.2.1 Getting Help on Scripting Interface Commands

Since Ayam 1.8.2 a scripting interface command named "help" is available, that displays the help of scripting interface commands using a web browser (similar to the "Help on Object" feature):

- **Synopsis:** "help command"
- **Background:** N/A, **Undo:** No, **Safe:** No, **Type:** Procedure
- **Description:** Fire up a web browser and display the help for the designated Ayam scripting interface command or procedure.
- **Example:** "help help" displays the help of the help procedure.

6.2.2 Managing Objects

To create new objects the `"crtOb"` command can be used.

- Synopsis: `"crtOb type [args]"`
- Background: Yes, Undo: No, Safe: Yes
- Description: New objects may be created with the command `"crtOb"`, `"type"` may be derived from the object type names, as displayed in the tree view. The new object will be created and linked to the scene as last object in the current level, no part of the GUI (object selection widget, property GUI, views) will be updated. Furthermore, the new object will *not* be selected. Depending on the type, further arguments may (or have to) be given; some object types expect other objects to be selected upon creation.

All arguments consist of a option name part and a value part (i.e. it is `"-center 1"` and *not* `"-center"` and also *not* `"-center=1"`). The option names can be abbreviated. Useful default and fallback values exist (see below). The arguments can be mixed freely (their order is not important) and repeated. If arguments are repeated, only the last set value is used, even if this leads to errors and application of fallback values later on.

Here is a comprehensive list of available arguments sorted by object type:

- **"NCurve"**: NURBS curves accept the following arguments:
 - * **"-length"**: length of the new curve, the length defaults to 4.
 - * **"-order"**: order of the new curve, the order defaults to 4. If a value greater than the length is specified, the order will be made identical to the length value.
 - * **"-kt"**: the knot type of the new curve, must be one of 0 – Bezier, 1 – BSpline, 2 – NURB, 3 – Custom, 4 – Chordal, 5 – Centripetal. A knot vector of specified type will automatically be created. The knot type defaults to 2 – NURB. If a custom knot vector is specified using the **"-kv"** option below, the knot type will always be set to 3 – Custom.
 - * **"-kv"**: the knot vector of the new curve. The value of this option is a list of floating point numbers of length curve length plus curve order, e.g. for a curve with 2 control points and order 2, specify 4 knots: **"-kv {0.0 0.0 1.0 1.0}"**. The knot vector defaults to an automatically created knot vector of the type specified by the **"-kt"** option above.
 - * **"-kn"**: the knot vector of the new curve. The value of this option is a variable name (with optional array and namespace specifier). The value of this variable must be compatible to the **"-kv"** option above.
 - * **"-cv"**: the control vector of the new curve. The value of this option is a list of floating point numbers that describe the 4D euclidean rational (weight *not* multiplied in) coordinates of the control points.
 This list may also only specify one point, which is then taken as starting point and DX/DY/DZ (see below) are used to create the missing control points automatically.
 To specify a complete control vector, this list should have length $\times 4$ elements, e.g. for a curve of length 3, specify 12 values: **"-cv {0.0 0.0 0.0 1.0 1.0 0.0 0.0 1.0 2.0 0.0 0.0 1.0}"**.
 - * **"-cn"**: the control vector of the new curve. The value of this option is a variable name (with optional array and namespace specifier). The value of this variable must be compatible to the **"-cv"** option above.
 - * **"-dx"**: the value of this option specifies the distance of automatically created control points in the x dimension, default is 0.25.
 - * **"-dy"**: the value of this option specifies the distance of automatically created control points in the y dimension, default is 0.0.
 - * **"-dz"**: the value of this option specifies the distance of automatically created control points in the z dimension, default is 0.0.
 - * **"-center"**: If the value of the **"-center"** option is 1, the new curve will be centered. The default value is 0, no centering. This option is only in effect if no **"-cv"** option is specified.
 - * **"-createmp"**: The **"-createmp"** option toggles creation of multiple points. The default value is 0.

Examples:

1. **"crtOb NCurve"**

creates a curve with length 4, order 4, standard (clamped) NURBS knot vector, control points at 0 0 0, 0.25 0 0, 0.5 0 0, 0.75 0 0 (all weights 1).

2. **"crtOb NCurve -center 1"**

creates a centered curve with length 4, order 4, standard (clamped) NURBS knot vector, control points at -0.375 0 0, -0.125 0 0, 0.125 0 0, 0.375 0 0 (all weights 1).

3. **"crtOb NCurve -length 5 -center 1 -dx 0.5"**

creates a centered curve with length 5, order 4, standard (clamped) NURBS knot vector, control points at -1 0 0, -0.5 0 0, 0 0 0, 0.5 0 0, 1 0 0 (all weights 1).

In versions of Ayam prior to 1.17, NURBS curves only accepted the **"-length"** argument.

See also section [4.4.1 NCurve Object](#) (page 150).

- **"ICurve"**: Interpolating curves accept the following arguments:
 - * **"-type"**: the type of the new curve, must be one of 0 – Open, 1 – Closed; default is 0.
 - * **"-length"**: length (number of data points to interpolate) of the new curve; the length defaults to 4.
 - * **"-order"**: order of the new curve, the order defaults to 4. If a value greater than the length is specified, the order will be made identical to the length value.
 - * **"-pt"**: the parameter type of the new curve, must be one of 0 – Chordal, 1 – Centripetal, 2 – Uniform; default is 0.
 - * **"-cv"**: the control vector of the new curve. The value of this option is a list of floating point numbers that describe the 3D (non rational) coordinates of the control points. This list may also only specify one point, which is then taken as starting point and DX/DY/DZ (see below) are used to create the missing control points automatically. To specify a complete control vector, this list should have curve length \times 3 elements, e.g. for a curve of length 3, specify 9 values: `"-cv {0.0 0.0 0.0 1.0 0.0 0.0 2.0 0.0 0.0}"`.
 - * **"-cn"**: the control vector of the new curve. The value of this option is a variable name (with optional array and namespace specifier). The value of this variable must be compatible to the **"-cv"** option above.
 - * **"-dx"**: the value of this option specifies the distance of automatically created control points in the x dimension, default is 0.25.
 - * **"-dy"**: the value of this option specifies the distance of automatically created control points in the y dimension, default is 0.0.
 - * **"-dz"**: the value of this option specifies the distance of automatically created control points in the z dimension, default is 0.0.
 - * **"-center"**: If the value of the **"-center"** option is 1, the new curve will be centered. The default value is 0, no centering. This option is only in effect if no **"-cv"** option is specified.
 - * **"-derivs"**: the value of this option controls whether user defined end derivatives should be used: 0 – no, 1 – yes, default is 0.
 - * **"-sdlen"**: the value of this option specifies the relative length (in relation to the distance of the first and second control point) of the start derivative, default is 0.125.
 - * **"-sderiv"**: is the start derivative, specified as a list of three float values. The derivative is specified relative to the first control point. The start derivative defaults to an automatically created derivative of a direction taken from the first two control points and length specified by the **"-sdlen"** option.
 - * **"-edlen"**: the value of this option specifies the relative length (in relation to the distance of the second to last and last control point) of the end derivative, default is 0.125.
 - * **"-ederiv"**: is the end derivative, specified as a list of three float values. The derivative is specified relative to the last control point. The end derivative defaults to an automatically created derivative of a direction taken from the last two control points and length specified by the **"-edlen"** option.

Examples:

1. **"crtOb ICurve"**

creates a curve with length 4, order 4, data points at 0 0 0, 0.25 0 0, 0.5 0 0, 0.75 0 0.

2. **"crtOb ICurve -l 5 -sderiv {0.0 -0.5 0.0} -ederiv {0.0 -0.5 0.0} -derivs 1 -center 1"**

creates a curve with length 5, order 4, data points at -0.5 0 0, -0.25 0 0, 0 0 0, 0.25 0 0, 0.5 0 0, end derivatives pointing straight upwards with length 0.5 in their respective end.

In versions of Ayam prior to 1.17, interpolating curves only accepted the **"-length"** argument. See also section [4.4.2 ICurve Object](#) (page 155).

- **"ACurve"**: Approximating curves accept the following arguments:
 - * **"-type"**: the type of the new curve, must be one of 0 – Open, 1 – Closed; default is 0.
 - * **"-length"**: length (number of data points to approximate) of the new curve, the length defaults to 4.
 - * **"-alength"**: number of NURBS control points to use for the approximating curve, must be smaller than the length above; this parameter defaults to length-1.
 - * **"-order"**: order of the new curve, the order defaults to 3. If a value greater than the output length is specified, the order will be made identical to the output length value. Currently, only values larger than 2 are supported.
 - * **"-symmetric"**: toggles creation of symmetric curves, must be one of 0 – Asymmetric, 1 – Symmetric; default is 0.
 - * **"-cv"**: the control vector of the new curve. The value of this option is a list of floating point numbers that describe the 3D (non rational) coordinates of the control points. This list may also only specify one point, which is then taken as starting point and DX/DY/DZ (see below) are used to create the missing control points automatically. To specify a complete control vector, this list should have curve length \times 3 elements, e.g. for a curve of length 3, specify 9 values: **"-cv {0.0 0.0 0.0 1.0 0.0 0.0 2.0 0.0 0.0}"**.
 - * **"-cn"**: the control vector of the new curve. The value of this option is a variable name (with optional array and namespace specifier). The value of this variable must be compatible to the **"-cv"** option above.
 - * **"-dx"**: the value of this option specifies the distance of automatically created control points in the x dimension, default is 0.25.
 - * **"-dy"**: the value of this option specifies the distance of automatically created control points in the y dimension, default is 0.0.
 - * **"-dz"**: the value of this option specifies the distance of automatically created control points in the z dimension, default is 0.0.
 - * **"-center"**: If the value of the **"-center"** option is 1, the new curve will be centered. The default value is 0, no centering. This option is only in effect if no **"-cv"** option is specified.

Examples:

1. **"crtOb ACurve -length 6"**
creates an approximating curve from six data points: 0 0 0, 0.25 0 0, 0.5 0 0, 0.75 0 0, 1 0 0, 1.25 0 0.
2. **"crtOb ACurve -l 5 -center 1"**
creates a centered approximating curve from five data points: -0.5 0 0, -0.25 0 0, 0 0 0, 0.25 0 0, 0.5 0 0.

In versions of Ayam prior to 1.17, approximating curves only accepted the **"-length"** argument.

See also section [4.4.3 ACurve Object \(page 158\)](#).

- **"NPatch"**: NURBS patches accept the following arguments:
 - * **"-width"**: width of the new patch, the width defaults to 4.
 - * **"-height"**: height of the new patch, the height defaults to 4.
 - * **"-uorder"**: order of the new patch in U parametric dimension, the order defaults to 4. If a value greater than the width is specified, the order will be made identical to the width value.
 - * **"-ukt"**: the U knot type of the new patch, must be one of 0 – Bezier, 1 – BSpline, 2 – NURB, 3 – Custom, 4 – Chordal, 5 – Centripetal. A knot vector of specified type will automatically be created. The knot type defaults to 2 – NURB. If a custom knot vector is specified using the **"-ukv"** option below, the knot type will always be set to 3 – Custom.
 - * **"-ukv"**: the U knot vector of the new patch. The value of this option is a list of floating point numbers of length width plus patch U order, e.g. for a patch with width 2 and U order 2, specify 4 knots: **"-ukv {0.0 0.0 1.0 1.0}"**. The knot vector defaults to an automatically created knot vector of the type specified by the **"-ukt"** option above.
 - * **"-un"**: the U knot vector of the new patch. The value of this option is a variable name (with optional array and namespace specifier). The value of this variable must be compatible to the **"-ukv"** option above.
 - * **"-vorder"**: order of the new patch in V parametric dimension, the order defaults to 4. If a value greater than the height is specified, the order will be made identical to the height value.
 - * **"-vkt"**: the V knot type of the new patch, must be one of 0 – Bezier, 1 – BSpline, 2 – NURB, 3 – Custom, 4 – Chordal, 5 – Centripetal. A knot vector of specified type will automatically be created. The knot type defaults to 2 – NURB. If a custom knot vector is specified using the **"-vk v"** option below, the knot type will always be set to 3 – Custom.
 - * **"-vk v"**: the V knot vector of the new patch. The value of this option is a list of floating point numbers of length height plus patch V order, e.g. for a patch with height 2 and V order 2, specify 4 knots: **"-vk v {0.0 0.0 1.0 1.0}"**. The knot vector defaults to an automatically created knot vector of the type specified by the **"-vkt"** option above.
 - * **"-vn"**: the V knot vector of the new patch. The value of this option is a variable name (with optional array and namespace specifier). The value of this variable must be compatible to the **"-vk v"** option above.
 - * **"-cv"**: the control vector of the new patch. The value of this option is a list of floating point numbers that describe the 4D euclidean rational (weight *not* multiplied in) coordinates of the control points. This list may also only specify one point, which is then taken as starting point and UDX/UDY/UDZ and VDX/VDY/VDZ (see below) are used to create the missing control points automatically. To specify a complete control vector, this list should have width × height × 4 elements, e.g. for a patch of width 2 and height 2, specify 16 values: **"-cv {0.0 0.0 0.0 1.0 1.0 0.0 0.0 1.0 2.0 0.0 0.0 1.0 2.0 1.0 0.0 1.0}"**.
 - * **"-cn"**: the control vector of the new patch. The value of this option is a variable name (with optional array and namespace specifier). The value of this variable must be compatible to the **"-cv"** option above.
 - * **"-udx"**: the value of this option specifies the distance of automatically created control points in the x dimension between points in a row (U parametric dimension, along width), default is 0.25.
 - * **"-udy"**: the value of this option specifies the distance of automatically created control points in the y dimension between points in a row (U parametric dimension, along width), default is 0.0.
 - * **"-udz"**: the value of this option specifies the distance of automatically created control

points in the z dimension between points in a row (U parametric dimension, along width), default is 0.0.

- * **"-vdx"**: the value of this option specifies the distance of automatically created control points in the x dimension between points in a column (V parametric dimension, along height), default is 0.0.
- * **"-vdy"**: the value of this option specifies the distance of automatically created control points in the y dimension between points in a column (V parametric dimension, along height), default is 0.25.
- * **"-vdz"**: the value of this option specifies the distance of automatically created control points in the z dimension between points in a column (V parametric dimension, along height), default is 0.0.
- * **"-center"**: If the value of the **"-center"** option is 1, the new patch will be centered. The default value is 0, no centering. This option is only in effect if no **"-cv"** option is specified.
- * **"-cretemp"**: The **"-cretemp"** option toggles creation of multiple points. The default value is 0.

Examples:

1. **"crtOb NPatch"**
creates a flat patch with width 4, height 4, uorder 4, vorder 4, standard (clamped) NURBS knot vectors, control points arranged in a equidistant grid in the XY-plane from 0 0 0 to 0.75 0.75 0 (all weights 1).
2. **"crtOb NPatch -vdy 0 -vdz 0.25"**
creates the same patch as above in the XZ-plane (ground plane).
3. **"crtOb NPatch -udy 0.25"**
creates a sheared version of the standard NURBS patch in the XY-plane.
4. **"crtOb NPatch -udy 0.25 -vdz 0.25"**
creates a 3D sheared version of the standard NURBS patch.
5. **"crtOb NPatch -width 2 -height 2 -center 1 -udx 2 -vdy 2"**
creates a centered patch with width 2, height 2, uorder 2, vorder 2, standard (clamped) NURBS knot vectors, control points at -1 0 0, 1 0 0, -1 1 0, 1 1 0 (all weights 1).
6. **"crtOb NPatch -width 3 -height 2 -uorder 2 -cv {-1 0 1 1 1 0 1 1 -1 0 0 1 1 0 0 1 -1 1 0 1 1 1 0 1}"**
creates a angular patch in the XZ- and XY-plane (remove **-uorder 2** to get a smooth shape).

In versions of Ayam prior to 1.17, NURBS patches only accepted the **"-width"** and **"-height"** argument.

See also section [4.6.1 NPatch Object \(page 170\)](#).

- **"IPatch"**: Interpolating patches accept the following arguments:
 - * **"-width"**: width of the new patch, the width defaults to 4.
 - * **"-height"**: height of the new patch, the height defaults to 4.
 - * **"-uorder"**: order of the new patch in U parametric dimension, the order defaults to 4. If a value greater than the width is specified, the order will be made identical to the width value. A value of 0 switches off interpolation along U.
 - * **"-ukt"**: the U parameterisation type, must be one of 0 – Chordal (default), 1 – Centripetal, 2 – Uniform.
 - * **"-vorder"**: order of the new patch in V parametric dimension, the order defaults to 4. If a value greater than the height is specified, the order will be made identical to the height value. A value of 0 switches off interpolation along V.
 - * **"-vkt"**: the V parameterisation type, must be one of 0 – Chordal (default), 1 – Centripetal, 2 – Uniform.
 - * **"-deriv_u"**: the end derivative mode for U, must be one of 0 – None (default), 1 – Automatic, or 2 – Manual. In manual mode full derivative vectors must be provided via **"-ederiv_u"** and **"-sderiv_u"**.
 - * **"-edlen_u"**: the length of automatically calculated end derivatives at end of patch in U (default 0.125).
 - * **"-sdlen_u"**: the length of automatically calculated end derivatives at start of patch in U (default 0.125).
 - * **"-ederiv_u"**: end derivatives for U at end of patch. The value of this option is a list of $3 \times \text{height}$ floating point numbers. There is no default value.
 - * **"-sderiv_u"**: end derivatives for U at start of patch. The value of this option is a list of $3 \times \text{height}$ floating point numbers. There is no default value.
 - * **"-deriv_v"**: the end derivative mode for V, must be one of 0 – None (default), 1 – Automatic, or 2 – Manual. In manual mode full derivative vectors must be provided via **"-ederiv_v"** and **"-sderiv_v"**.
 - * **"-edlen_v"**: the length of automatically calculated end derivatives at end of patch in V (default 0.125).
 - * **"-sdlen_v"**: the length of automatically calculated end derivatives at start of patch in V (default 0.125).
 - * **"-ederiv_v"**: end derivatives for V at end of patch. The value of this option is a list of $3 \times \text{width}$ floating point numbers. There is no default value.
 - * **"-sderiv_v"**: end derivatives for V at start of patch. The value of this option is a list of $3 \times \text{width}$ floating point numbers. There is no default value.
 - * **"-cv"**: the control vector of the new patch. The value of this option is a list of floating point numbers that describe the 3D non rational coordinates of the data points to be interpolated. This list may also only specify one point, which is then taken as starting point and UDX/UDY/UDZ and VDX/VDY/VDZ (see below) are used to create the missing control points automatically. To specify a complete control vector, this list should have $\text{width} \times \text{height} \times 3$ elements, e.g. for a patch of width 2 and height 2, specify 12 values: **"-cv {0.0 0.0 0.0 1.0 0.0 0.0 2.0 0.0 0.0 2.0 1.0 0.0}"**.
 - * **"-cn"**: the control vector of the new patch. The value of this option is a variable name (with optional array and namespace specifier). The value of this variable must be compatible to the **"-cv"** option above.
 - * **"-udx"**: the value of this option specifies the distance of automatically created control

points in the x dimension between points in a row (U parametric dimension, along width), default is 0.25.

- * `"-udy"`: the value of this option specifies the distance of automatically created control points in the y dimension between points in a row (U parametric dimension, along width), default is 0.0.
- * `"-udz"`: the value of this option specifies the distance of automatically created control points in the z dimension between points in a row (U parametric dimension, along width), default is 0.0.
- * `"-vdx"`: the value of this option specifies the distance of automatically created control points in the x dimension between points in a column (V parametric dimension, along height), default is 0.0.
- * `"-vdy"`: the value of this option specifies the distance of automatically created control points in the y dimension between points in a column (V parametric dimension, along height), default is 0.25.
- * `"-vdz"`: the value of this option specifies the distance of automatically created control points in the z dimension between points in a column (V parametric dimension, along height), default is 0.0.
- * `"-center"`: If the value of the `"-center"` option is 1, the new patch will be centered. The default value is 0, no centering. This option is only in effect if no `"-cv"` option is specified.

Examples:

1. `"crtOb IPatch"`
creates a flat patch with width 4, height 4, uorder 4, vorder 4, chordal parameterisation, data points arranged in a equidistant grid in the XY-plane from 0 0 0 to 0.75 0.75 0.
2. `"crtOb IPatch -vdy 0 -vdz 0.25"`
creates the same patch as above in the XZ-plane (ground plane).
3. `"crtOb IPatch -udy 0.25"`
creates a sheared version of the standard IPatch in the XY-plane.
4. `"crtOb IPatch -udy 0.25 -vdz 0.25"`
creates a 3D sheared version of the standard IPatch.
5. `"crtOb IPatch -width 3 -height 3 -center 1 -udx 2 -vdy 2"`
creates a centered patch with width 3, height 3, uorder 3, and vorder 3.

See also section 4.6.2 IPatch Object (page 174).

- **"APatch"**: Approximating patches accept the following arguments:
 - * **"-mode"**: the value of this option specifies the approximation mode, it must be one of 0 - UV, 1 - VU, 2 - U, or 3 - V.
 - * **"-width"**: number of data points in U direction, the width defaults to 4.
 - * **"-height"**: number of data points in V direction, the height defaults to 4.
 - * **"-awidth"**: desired width of the NURBS patch approximating the data points, defaults to 3. This value must be \leq the width.
 - * **"-aheight"**: desired height of the NURBS patch approximating the data points, defaults to 3. This value must be \leq the height.
 - * **"-uorder"**: order of the new patch in U parametric dimension, the order defaults to 3. If a value greater than *awidth* is specified, the order will be made identical to the *awidth* value.
 - * **"-ukt"**: the U parameterisation type, must be one of 0 – Chordal (default), 1 – Centripetal.
 - * **"-vorder"**: order of the new patch in V parametric dimension, the order defaults to 3. If a value greater than *aheight* is specified, the order will be made identical to the *aheight* value.
 - * **"-vkt"**: the V parameterisation type, must be one of 0 – Chordal (default), 1 – Centripetal.
 - * **"-cv"**: the control vector of the new patch. The value of this option is a list of floating point numbers that describe the 3D non rational coordinates of the data points to be approximated.
 This list may also only specify one point, which is then taken as starting point and UDX/UDY/UDZ and VDX/VDY/VDZ (see below) are used to create the missing control points automatically.
 To specify a complete control vector, this list should have $\text{width} \times \text{height} \times 3$ elements, e.g. for a patch of width 4 and height 4, specify 48 values.
 - * **"-cn"**: the control vector of the new patch. The value of this option is a variable name (with optional array and namespace specifier). The value of this variable must be compatible to the **"-cv"** option above.
 - * **"-udx"**: the value of this option specifies the distance of automatically created control points in the x dimension between points in a row (U parametric dimension, along width), default is 0.25.
 - * **"-udy"**: the value of this option specifies the distance of automatically created control points in the y dimension between points in a row (U parametric dimension, along width), default is 0.0.
 - * **"-udz"**: the value of this option specifies the distance of automatically created control points in the z dimension between points in a row (U parametric dimension, along width), default is 0.0.
 - * **"-vdx"**: the value of this option specifies the distance of automatically created control points in the x dimension between points in a column (V parametric dimension, along height), default is 0.0.
 - * **"-vdy"**: the value of this option specifies the distance of automatically created control points in the y dimension between points in a column (V parametric dimension, along height), default is 0.25.
 - * **"-vdz"**: the value of this option specifies the distance of automatically created control points in the z dimension between points in a column (V parametric dimension, along height), default is 0.0.
 - * **"-center"**: If the value of the **"-center"** option is 1, the new patch will be centered. The default value is 0, no centering. This option is only in effect if no **"-cv"** option is specified.

Examples:

1. **"crtOb APatch"**
creates a flat patch with width 4, height 4, awidth 3, aheight 3, uorder 3, vorder 3, chordal parameterisation, data points arranged in a equidistant grid in the XY-plane from 0 0 0 to 0.75 0.75 0.
2. **"crtOb APatch -vdy 0 -vdz 0.25"**
creates the same patch as above in the XZ-plane (ground plane).
3. **"crtOb APatch -udy 0.25"**
creates a sheared version of the standard APatch in the XY-plane.
4. **"crtOb APatch -udy 0.25 -vdz 0.25"**
creates a 3D sheared version of the standard APatch.
5. **"crtOb APatch -width 3 -height 3 -center 1 -udx 2 -vdy 2"**
creates a centered patch with width 3, height 3, uorder 3, and vorder 3.

See also section 4.6.3 APatch Object (page 176).

- **"PatchMesh"**: patch meshes accept the following arguments:
 - * **"-type"**: type of the new patch mesh, must be one of 0 – bilinear or 1 – bicubic, defaults to 1.
 - * **"-width"**: width of the new patch, the width defaults to 4. Valid values for bicubic patch meshes depend on closedness and step size in the U direction. See also the discussion in section 4.6.5 [PatchMeshAttr Property](#) (page 179).
 - * **"-height"**: height of the new patch, the height defaults to 4. Valid values for bicubic patch meshes depend on closedness and step size in the V direction. See also the discussion in section 4.6.5 [PatchMeshAttr Property](#) (page 179).
 - * **"-closeu"**: determines, whether the patch mesh is closed in U direction; must be either 0 – open (default) or 1 – closed.
 - * **"-closev"**: determines, whether the patch mesh is closed in V direction; must be either 0 – open (default) or 1 – closed.
 - * **"-ubt"**: the U basis type, must be one of 0 – Bezier (default), 1 – B-Spline, 2 – Catmull-Rom, 3 – Hermite, 4 – Power, 5 – Custom.
 - * **"-vbt"**: the V basis type, must be one of 0 – Bezier (default), 1 – B-Spline, 2 – Catmull-Rom, 3 – Hermite, 4 – Power, 5 – Custom.
 - * **"-ubasis"**: the U basis; must be 16 float values in column major order. If set, the U basis type will automatically set to Custom and the step size should be specified (if different from 3).
 - * **"-ubn"**: the U basis. The value of this option is a variable name (with optional array and namespace specifier). The value of this variable must be compatible to the **"-ubasis"** option above.
 - * **"-vbasis"**: the V basis; must be 16 float values in column major order. If set, the V basis type will automatically set to Custom and the step size should be specified (if different from 3).
 - * **"-vbn"**: the V basis. The value of this option is a variable name (with optional array and namespace specifier). The value of this variable must be compatible to the **"-vbasis"** option above.
 - * **"-ustep"**: the U basis step size; must be 1, 2, 3, or 4. Can be omitted for all non custom basis types.
 - * **"-vstep"**: the V basis step size; must be 1, 2, 3, or 4. Can be omitted for all non custom basis types.
 - * **"-cv"**: the control vector of the new patch mesh. The value of this option is a list of floating point numbers that describe the 3D rational coordinates of the control points. This list may also only specify one point, which is then taken as starting point and UDX/UDY/UDZ and VDX/VDY/VDZ (see below) are used to create the missing control points automatically.
To specify a complete control vector, this list should have $\text{width} \times \text{height} \times 4$ elements, e.g. for a bilinear patch mesh of width 2 and height 2, specify 16 values: **"-cv {0.0 0.0 0.0 1.0 1.0 0.0 0.0 1.0 2.0 0.0 0.0 1.0 2.0 1.0 0.0 1.0}"**.
 - * **"-cn"**: the control vector of the new patch mesh. The value of this option is a variable name (with optional array and namespace specifier). The value of this variable must be compatible to the **"-cv"** option above.
 - * **"-udx"**: the value of this option specifies the distance of automatically created control points in the x dimension between points in a row (U parametric dimension, along width), default is 0.25.
 - * **"-udy"**: the value of this option specifies the distance of automatically created control points in the y dimension between points in a row (U parametric dimension, along width),

default is 0.0.

- * `"-udz"`: the value of this option specifies the distance of automatically created control points in the z dimension between points in a row (U parametric dimension, along width), default is 0.0.
- * `"-vdx"`: the value of this option specifies the distance of automatically created control points in the x dimension between points in a column (V parametric dimension, along height), default is 0.0.
- * `"-vdy"`: the value of this option specifies the distance of automatically created control points in the y dimension between points in a column (V parametric dimension, along height), default is 0.25.
- * `"-vdz"`: the value of this option specifies the distance of automatically created control points in the z dimension between points in a column (V parametric dimension, along height), default is 0.0.
- * `"-center"`: If the value of the `"-center"` option is 1, the new patch will be centered. The default value is 0, no centering. This option is only in effect if no `"-cv"` option is specified.

Examples:

1. **`"crtOb PatchMesh"`**
creates a flat bicubic patch mesh of width 4, height 4, Bezier basis, control points arranged in a equidistant grid in the XY plane from 0 0 0 to 0.75 0.75 0.
2. **`"crtOb PatchMesh -vdy 0 -vdz 0.25"`**
creates the same patch mesh as above in the XZ plane (ground plane).
3. **`"crtOb PatchMesh -udy 0.25"`**
creates a sheared version of the standard patch mesh in the XY plane.
4. **`"crtOb PatchMesh -udy 0.25 -vdz 0.25"`**
creates a 3D sheared version of the standard patch mesh.
5. **`"crtOb PatchMesh -type 0 -width 3 -height 3 -center 1 -udx 2 -vdy 2"`**
creates a large centered bilinear patch mesh of width 3 and height 3.

See also section [4.6.5 PatchMesh Object](#) (page 179).

- **"PolyMesh"**: Polygonal meshes accept the following arguments:
 - * **"-polys"**: the value of this option specifies the number of polygons/faces in the mesh. The number of polygons defaults to 0.
 - * **"-loops"**: the value of this option specifies the number of loops per polygon. It is therefore a list of positive integer values of a length equal to the value of the **"-polys"** option. The default value of this option is a list of proper length with all elements set to 1 (only normal polygons, without holes, are specified).
 - * **"-nverts"**: the value of this option specifies the number of vertices per loop. It is therefore a list of positive integer values of a length equal to the sum of all elements of the **"-loops"** option. The default value of this option is a list of proper length with all elements set to 3 (only triangles are in the mesh).
 - * **"-iverts"**: the value of this option specifies all the (zero based) indices of the vertices of all loops. It is therefore a list of integer values of a length equal to the sum of all elements of the **"-nverts"** option. The default value of this option is a list of proper length with the elements set to a sequence of integers so that the control points are used in the same order as specified via the **"-cv"** option (0, 1, 2, 3, ...).
 - * **"-cv"**: the control points of the new mesh. The value of this option is a list of floating point numbers that describe the 3D (non rational) coordinates of the control points. The indices specified via the **"-iverts"** option point to this list. If the **"-vnormals"** option is 1, also vertex normals are specified in this list (directly following the coordinate values of each control point) and stride is 6, otherwise stride is 3. This list must have a length of stride by the highest value in the list provided via the **"-iverts"** option. The default value of this option is an empty list, this implies that this option must be specified to create a non-empty PolyMesh object.
 - * **"-cn"**: the control points of the new mesh. The value of this option is a variable name (with optional array and namespace specifier). The value of this variable must be compatible to the **"-cv"** option above.
 - * **"-vnormals"**: determines whether vertex normals are present. The default value is 0 – no vertex normals are present.

Examples:

1. `"crtOb PolyMesh -p 1 -cv {0 0 0 1 0 0 0 1 0}"`
creates a polymesh with a single triangular face.
2. `"crtOb PolyMesh -p 2 -cv {0 0 0 1 0 0 1 1 0 0 1 0} -iv {0 1 2 0 2 3}"`
creates a polymesh with two connected (vertex sharing) triangular faces.
3. `"crtOb PolyMesh -p 3 -cv {0 0 0 1 0 0 1 1 0 0 1 0 1.5 0 0 1.5 1 0} -iv {0 1 2 0 2 3 1 4 5 2} -nv {3 3 4}"`
creates a polymesh with two triangles and one quad all connected (vertex sharing).
4. `"crtOb PolyMesh -p 1 -loops {2} -cv {0 0 0 1 0 0 0 1 0 .25 .25 0 .5 .25 0 .25 .5 0}"`
creates a polymesh with one triangular face that has a triangular hole.

Notes:

Besides checking the lengths of arrays, maximum indices, and minimum number of vertices per loop, there is *no* error checking. Undetected errors include: degenerated or non-planar polygons, hole loops geometrically outside or touching the outline loop, disagreeing loop winding orders between faces, non-manifold meshes, unused control points. Those errors may be problematic for further processing steps – some may even go unnoticed in Ayam and will only be detected later in other applications. See also section 4.8.1 PolyMesh Object (page 225).

- **"SDMesh"**: Subdivision meshes accept the following arguments:
 - * **"-scheme"**: the value of this option specifies the subdivision scheme, it may be set to 0 – Catmull-Clark or 1 – Loop only. Default is 0.
 - * **"-faces"**: the value of this option specifies the number of faces in the mesh. The number of faces defaults to 0.
 - * **"-nverts"**: the value of this option specifies the number of vertices per face. It is therefore a list of positive integer values of a length equal to the number of faces. The default value of this option is a list of proper length with all elements set to 3 (only triangles are in the mesh).
 - * **"-verts"**: the value of this option specifies all the (zero based) indices of the vertices of all faces. It is therefore a list of integer values of a length equal to the sum of all elements of the **"-nverts"** option. The default value of this option is a list of proper length with the elements set to a sequence of integers so that the control points are used in the same order as specified via the **"-cv"** option (0, 1, 2, 3, ...).
 - * **"-cv"**: The value of this option is a list of floating point numbers that describe the 3D (non rational) coordinates of the control points. The indices specified via the **"-verts"** option point to this list. This list must have a length of 3 multiplied by the highest value in the list provided via the **"-verts"** option. The default value of this option is an empty list, this implies that this option must be specified to create a non-empty SDMesh object.
 - * **"-cn"**: the control points of the new mesh. The value of this option is a variable name (with optional array and namespace specifier). The value of this variable must be compatible to the **"-cv"** option above.
 - * **"-tags"**: the value of this option specifies a number of tags. It is therefore a list of positive integer values of arbitrary length. The only allowed values are 0 – hole, 1 – corner, 2 – crease, and 3 – interpolateboundary. The default value of this option is an empty list: no tags.
 - * **"-args"**: the value of this option specifies the number of integer and floating point arguments per tag. It is therefore a list of positive integer values of length: double number of tags. The even entries specify the number of integer and the odd entries the number of floating point arguments per tag. The content of this list is partially dictated by the **"-tags"** option, e.g. a crease entry has at least two integer arguments and one floating point argument. The default value of this option is list of proper length, with all elements set to zero (no tags have any arguments).
 - * **"-intargs"**: the value of this option specifies the integer arguments of all tags. It is therefore a list of integer values of length sum of all even elements given by the **"-args"** option.
 - * **"-doubleargs"**: the value of this option specifies the floating point arguments of all tags. It is therefore a list of double values of length sum of all odd elements given by the **"-args"** option.

Examples:

1. `"crtOb SDMesh -f 4 -v {0 1 3 1 2 3 0 3 2 0 2 1} -cv {0 0 0 1 0 0 0 0 -1 0.5 1 -0.5}"`
creates a tetrahedral (four triangular faces) mesh.
2. `"crtOb SDMesh -f 4 -v {0 1 3 1 2 3 0 3 2 0 2 1} -cv {0 0 0 1 0 0 0 0 -1 0.5 1 -0.5} -tags {1} -args {1 1} -intargs {0} -doubleargs {3.0}"`
creates a tetrahedral mesh with a semi-sharp corner.
3. `"crtOb SDMesh -f 4 -v {0 1 3 1 2 3 0 3 2 0 2 1} -cv {0 0 0 1 0 0 0 0 -1 0.5 1 -0.5} -tags {1} -args {2 1} -intargs {0 1} -doubleargs {10.0}"`
creates a tetrahedral mesh with a crease.

Notes:

Besides checking the lengths of arrays, maximum indices, and minimum number of vertices per face, there is *no* error checking. Undetected errors include: degenerated or non-planar faces, faces with unsuitable vertex counts (for the selected subdivision scheme), non-manifold meshes, unused control points, wrong tag arguments. Those errors may be problematic for further processing steps – some may even go unnoticed in Ayam and will only be detected later in other applications.

See also section 4.8.2 [SDMesh Object](#) (page 227).

- **"Script"**: Script objects accept the following arguments:¹
 - * **"-type"**: the value of this option specifies the script type, it may be set to "run", "modify", or "create" (without quotes) only.
 - * **"-active"**: the value of this option specifies whether to activate the script immediately or not. It may be set to "yes", "no", 0, or 1.
 - * **"-script"**: the value of this option is the script code itself.
 - * **"-file"**: the value of this option is the full path and file name of a script to load into the new Script object.

See also section [4.9.1 Script Object \(page 229\)](#).

- **"Level"**: these objects can be given an additional type argument, this argument may be one of: "0" (level – default), "1" (union), "2" (intersection), "3" (difference), or "4" (primitive).

Examples:

1. **"crtOb Level"**
creates a simple level object.
2. **"crtOb Level 3"**
creates a CSG difference level object.

See also section [4.2.6 Level Object \(page 128\)](#).

- **"Material"**: Materials must be given an additional argument giving the name of the new material. If a material with the chosen name already exists, no object will be created.

Example:

1. **"crtOb Material Wood"**
creates a material named Wood.

See also section [4.2.5 Material Object \(page 125\)](#).

- **"Instance"**: creates an instance of the selected object.

Example:

1. **"crtOb Sphere; selOb -1; crtOb Instance"**
creates a Sphere and an instance of it.

See also section [4.2.7 Instance Object \(page 131\)](#).

- ...

- Example: Create a sphere and update the GUI: `"crtOb Sphere; uS; rV"`.

The following helper commands, create certain often used curves:

crtNCircle – create NURBS circle:

- Synopsis: `"crtNCircle [-r radius] [-a arc]"`
- Background: Yes, Undo: No, Safe: Yes
- Description: This command creates a circular NURBS curve with radius as defined via the `-r` option and arc as defined via the `-a` option. The curve always starts on the positive X-axis. The radius defaults to 1.0 and the arc to 360.0. The arc option supports negative values. See also section [5.2.2 NURB Circle Tool \(page 275\)](#).

¹ Since 1.29.

crtNParabola – create a parabola:

- Synopsis: `"crtNParabola [-ymin ymin] [-ymax ymax] [-rmax r]"`
- Background: Yes, Undo: No, Safe: Yes
- Description: This command creates a rational NURBS curve with the shape of a parabola in the XY-plane that starts at desired *ymin* and ends at specified *ymax* value with the given radius value.¹ The default values are *ymin* = 0.0, *ymax* = 1.0, and *rmax* = 1.0.

crtNHyperbola – create a hyperbola:

- Synopsis: `"crtNHyperbola [-ymin ymin] [-ymax ymax] [-a a] [-b b]"`
- Background: Yes, Undo: No, Safe: Yes
- Description: This command creates a rational NURBS curve with the shape of a hyperbola in the XY-plane that starts at desired *ymin* and ends at specified *ymax* value with the given values for the semi-axes, *a* and *b*.² The default values are *ymin* = -0.5, *ymax* = 0.5, *a* = 0.5, and *b* = 0.5.

crtNConicArc – create a conic arc:

- Synopsis: `"crtNConicArc -p0 p0 -t0 t0 -p2 p2 -t2 t2 -p1 p1 [-k]"`
- Background: Yes, Undo: No, Safe: Yes
- Description: This command creates a rational NURBS curve with the shape of a conic arc defined by start and end point (*p0* and *p2*) and corresponding tangents (*t0* and *t2*) as well as another point on the arc (*p1*).³
This command may create zero weights (infinite control points) for arcs $\geq 180^\circ$ if the optional parameter `-k`, for `keep_zero_weights`, is present. However, these infinite control points are *not* supported by the majority of NURBS processing algorithms in Ayam and curves/surfaces employing them do not display correctly using any of the GLU NURBS display modes.
- Example: Create a centered half sphere of radius 1 in the XY-plane:
`"crtNConicArc -p0 {-1 0 0} -t0 {0 1 0} -p2 {1 0 0} -t2 {0 -1 0} -p1 {0 1 0}"`.

crtClosedBS – create closed (circular) B-Spline:

- Synopsis: `"crtClosedBS [-s sections] [-o order] [-a arc] [-r radius]"`
- Background: Yes, Undo: No, Safe: Yes
- Description: This command creates a circular B-Spline curve with desired number of sections (defaults to 6), order (defaults to 4), arc (defaults to 360.0, negative values are allowed), and radius (defaults to 1.0). The curve always starts on the positive X-axis. See also section 5.2.1 [Circular B-Spline Tool \(page 274\)](#).

crtNRect – create a rectangular NURBS curve:

- Synopsis: `"crtNRect [-w width] [-h height]"`
- Background: Yes, Undo: No, Safe: Yes
- Description: This command creates a centered rectangular NURBS curve of twice the given width and height in the XY-plane.

¹ Since 1.31. ² Since 1.31. ³ Since 1.31.

crtTrimRect – create a rectangular bounding trim curve:

- Synopsis: `"crtTrimRect "`
- Background: Yes, Undo: No, Safe: Yes
- Description: This command creates a rectangular NURBS curve that fits the parameter space of the currently selected NPatch object (or the current parent object, if it is a NPatch). The `"CreateAt "` and `"CreateIn "` options are ignored. See also section 5.2.4 TrimRect Tool (page 277).

The following helper commands create certain often used surfaces:

crtNSphere – create a sphere:

- Synopsis: `"crtNSphere [-radius r] "`
- Background: Yes, Undo: No, Safe: Yes
- Description: This command creates a rational NURBS surface with the shape of a sphere with the given radius value.¹ The default value is $r = 1.0$.

crtNCylinder – create a cylinder:

- Synopsis: `"crtNCylinder [-height h] [-radius r] "`
- Background: Yes, Undo: No, Safe: Yes
- Description: This command creates a rational NURBS surface with the shape of a cylinder with the given height and radius values.² The base of the cylinder is centered in the XZ-plane and it extends along the Y-axis. The default values are $h = 1.0$, $r = 1.0$.

crtNCone – create a cone:

- Synopsis: `"crtNCone [-height h] [-radius r] "`
- Background: Yes, Undo: No, Safe: Yes
- Description: This command creates a rational NURBS surface with the shape of a cone with the given height and radius values.³ The base of the cone is centered in the XZ-plane and it extends along the Y-axis. The default values are $h = 1.0$, $r = 1.0$.

crtNDisk – create a disk:

- Synopsis: `"crtNDisk [-radius r] "`
- Background: Yes, Undo: No, Safe: Yes
- Description: This command creates a rational NURBS surface with the shape of a disk with the given radius value.⁴ The disk is centered in the XZ-plane. The default value is $r = 1.0$.

crtNTorus – create a torus:

- Synopsis: `"crtNTorus [-minorradius minr] [-majorradius majr] "`
- Background: Yes, Undo: No, Safe: Yes
- Description: This command creates a rational NURBS surface with the shape of a torus with the given radius values.⁵ The torus will be created by revolving a circle, that is displaced from the center of the XY-plane along the X-axis, around the Y-axis. The default values are $minr = 0.25$, $majr = 1.0$.

¹ Since 1.31. ² Since 1.31. ³ Since 1.31. ⁴ Since 1.31. ⁵ Since 1.31.

crtNParaboloid – create a paraboloid:

- Synopsis: `"crtNParaboloid [-ymin ymin] [-ymax ymax] [-rmax rmax]"`
- Background: Yes, Undo: No, Safe: Yes
- Description: This command creates a rational NURBS surface with the shape of a paraboloid with the given *ymin*, *ymax*, and *rmax* values.¹ The paraboloid will be created by revolving a parabola around the Y-axis. The default values are *ymin* = 0.0, *ymax* = 1.0, *rmax* = 1.0.

crtNHyperboloid – create a hyperboloid:

- Synopsis: `"crtNHyperboloid [-ymin ymin] [-ymax ymax] [-a a] [-b b]"`
- Background: Yes, Undo: No, Safe: Yes
- Description: This command creates a rational NURBS surface with the shape of a hyperboloid with the given *ymin*, *ymax*, and *a/b* values.² The hyperboloid will be created by revolving a hyperbola around the Y-axis. The default values are *ymin* = -1.0, *ymax* = 1.0, *a* = 0.25, *b* = 0.5.

delOb – delete object(s):

- Synopsis: `"delOb"`
- Background: Yes, Undo: No, Safe: Yes
- Description: Delete the selected object(s) and their children from the scene. This operation fails for the root object. This operation may fail for master objects (objects with instances/references) or for parent objects with master objects among their children. In case of failure, an error will be reported and the undeletable object(s) will be moved to the end of the current level. The root object, however, will never be moved.

See also the [6.2.3 candelOb scripting interface command](#) (page 372).

- Notes: This command always clears the object selection.

¹ Since 1.31. ² Since 1.31.

6.2.3 Interrogating Objects

getType:

- Synopsis: "getType [varname] "
- Background: Yes, Undo: No, Safe: Yes
- Description: This command writes the type of the selected object into the variable designated by the `varname` argument. The types are the well known strings that are displayed in the hierarchy list box if the objects are not named (NPatch, NCurve, Sphere, etc.). If no variable name is specified the command returns the respective result(s).¹ If multiple objects are selected, a list is returned.

getName:

- Synopsis: "getName [-s] [varname] "
- Background: Yes, Undo: No, Safe: Yes
- Description: This command writes the name of the selected object into the variable designated by the `varname` argument. If the object has no name, a warning message will be emitted (unless the option `"-s"` is given). If no variable name is specified the command returns the respective result(s).² If multiple objects are selected, a list is returned.
See also section 6.2.28 `nameOb` scripting interface command (page 436).

hasChild:

- Synopsis: "hasChild"
- Background: Yes, Undo: No, Safe: Yes
- Description: This command returns 1 if the selected object has child objects, otherwise 0 is returned. If multiple objects are selected, a list is returned.

hasMat – has material:

- Synopsis: "hasMat "
- Background: Yes, Undo: No, Safe: Yes
- Description: This command returns 1 if the selected object has a material assigned, otherwise 0 is returned. If multiple objects are selected, a list is returned.

hasRefs – has references:

- Synopsis: "hasRefs "
- Background: Yes, Undo: No, Safe: Yes
- Description: This command returns 1 if the selected object has references (e.g. it is a master), otherwise 0 is returned. If multiple objects are selected, a list is returned.

¹ Since 1.25. ² Since 1.25.

candelOb – delete object(s):

- Synopsis: "candelOb"
- Background: Yes, Undo: No, Safe: Yes
- Description: This command returns 1 if the selected object(s) can be deleted without errors and without moving objects to the end of the current level, otherwise 0 is returned.

hasTrafo – has transformations:

- Synopsis: "hasTrafo [-r|-s|-t]"
- Background: Yes, Undo: No, Safe: Yes
- Description: This command returns 1 if the selected object has non-default transformation attributes, otherwise 0 is returned. If multiple objects are selected, a list is returned.
If the "-r" flag is provided, the command only checks the rotation attributes.
If the "-s" flag is provided, the command only checks the scale attributes.
If the "-t" flag is provided, the command only checks the translation attributes.

isCurve:

- Synopsis: "isCurve"
- Background: Yes, Undo: No, Safe: Yes
- Description: This command returns 1 if the selected object is (or provides / converts to) a parametric curve, otherwise 0 is returned. If multiple objects are selected, a list is returned.

isSurface:

- Synopsis: "isSurface"
- Background: Yes, Undo: No, Safe: Yes
- Description: This command returns 1 if the selected object is (or provides / converts to) a parametric surface, otherwise 0 is returned. If multiple objects are selected, a list is returned.

getBB – get the bounding box:

- Synopsis: "getBB"
- Background: Yes, Undo: No, Safe: Yes
- Description: This command returns the axis parallel bounding box of the selected object(s) as a list of six double values (xmin, ymin, zmin, xmax, ymax, zmax). Note, that for multiple selected objects the union of all individual bounding boxes will be returned and *not* the individual bounding boxes in a list.

getPlaneNormal – get the plane normal of an object:

- Synopsis: "getPlaneNormal [-t]"
- Background: Yes, Undo: No, Safe: Yes
- Description: This command returns the plane normal(s) of the selected object(s) as a list of three double values.
If the option "-t" is given, the normal will be transformed by the objects transformation attributes.
If the object is not planar, a mean normal will be computed. If the normal can not be computed because e.g. the selected curve is a straight line or degenerate the returned vector will be 0.0, 0.0, 0.0.
This command supports arbitrary object types that just need to provide their points.

isClosed:

- Synopsis: `"isClosed [-u|-v]"`
- Background: Yes, Undo: No, Safe: Yes
- Description: This command returns 1 if the selected object is a closed parametric curve or surface, otherwise 0 is returned.¹ The optional parameter `"-u"/"-v"` determines which dimension of a surface to check.

If multiple objects are selected, a list is returned.

This command supports arbitrary object types that just need to provide NURBS curves or surfaces.

- Examples:

1. `"crtOb NCurve; selOb -1; isClosed"`
returns `"0"`,
2. `"crtOb NCircle; selOb -1; isClosed"`
returns `"1"`.

isPlanar – check for planarity:

- Synopsis: `"isPlanar"`
- Background: Yes, Undo: No, Safe: Yes
- Description: This command returns 1 if the selected object is a planar parametric curve or surface, otherwise 0 is returned.² If multiple objects are selected, a list is returned.

This command supports arbitrary object types that just need to provide NURBS curves or surfaces.

isDegen – check for degeneracy:

- Synopsis: `"isDegen"`
- Background: Yes, Undo: No, Safe: Yes
- Description: This command returns 1 if the selected object is a degenerated parametric curve (point shape) or surface (point or line shape), otherwise 0 is returned.³ If multiple objects are selected, a list is returned.

This command supports arbitrary object types that just need to provide NURBS curves or surfaces.

- Examples:

1. `"crtOb NCurve; selOb -1; isDegen"`
returns `"0"`,
2. `"crtOb NCurve -dx 0; selOb -1; isDegen"`
returns `"1"`.

¹ Since 1.27. ² Since 1.27. ³ Since 1.27.

isParent:

- Synopsis: "isParent"
- Background: Yes, Undo: No, Safe: Yes
- Description: This command returns 1 if the selected object is a potential parent, otherwise 0 is returned.¹ If multiple objects are selected, a list is returned.
- Examples:
 1. `"crtOb NCurve; selOb -1; isParent"`
returns "0",
 2. `"crtOb NPatch; selOb -1; isParent"`
returns "1".

isTrimmed:

- Synopsis: "isTrimmed"
- Background: Yes, Undo: No, Safe: Yes
- Description: This command returns 1 if the selected object is a non trivially trimmed parametric surface, otherwise 0 is returned.² If multiple objects are selected, a list is returned.
This command supports arbitrary object types that just need to provide a NURBS surface.

isHidden:

- Synopsis: "isHidden"
- Background: Yes, Undo: No, Safe: Yes
- Description: This command returns 1 if the selected object is hidden, otherwise 0 is returned.³ If multiple objects are selected, a list is returned.

¹ Since 1.27. ² Since 1.27. ³ Since 1.30.

6.2.4 Selecting Objects

These commands are probably the most important ones, because many other scripting interface commands operate on selected objects only:

selOb – select object(s):

- Synopsis: `"selOb ([-get|-end|-clear] | [index] | [first-last])"`
- Background: Yes, Undo: No, Safe: Yes
- Description: Use this command to set or clear the current selection, `index` may be a single (zero based) index, an ordered list of indices or a range.
If the `"-get"` option is specified, the current selection will be returned just as the `"getSel"` command does.
If no index is given or the `"-clear"` option is specified, the current selection will be cleared.
If the single index is `-1` or the `"-end"` option is specified, the last object in the current level will be selected.
- Examples:
 1. `"selOb"`
clears the current selection,
 2. `"selOb 0"`
selects the first object in the current level,
 3. `"selOb 0 1"`
selects the first two objects in the current level, but
 4. `"selOb 1 0"`
only selects the second object in the current level,
 5. `"selOb 0 3-8"`
selects the first and the fourth up to the ninth objects in the current level,
 6. `"selOb 1-end"`
selects all objects in the current level except for the first, and
 7. `"selOb -1"`
selects the last object in the current level.

sL – select last object:

- Synopsis: "sL [count]"
- Background: No, Undo: No, Safe: Yes
- Description: Select the last object in the current level and, if run in the Ayam interpreter, update the GUI.
If an argument is provided, it determines the number of objects to select.
If run in the safe interpreter (e.g. from a Script objects script) this command will create a hidden selection. This command is often called in a sequence after creating a new object like this:

```
# create object
crtOb NCurve
# update tree
uCR
# select new object
sL
```

In Script objects scripts the above example command sequence would leave out the "uCR" command, as access to the GUI is blocked anyway in this context:

```
# create object
crtOb NCurve
# select new object
sL
```

- Note: In the main Tcl interpreter, sL is a procedure that operates on information in the Tcl context alone. Calling sL on outdated hierarchy data (e.g. by leaving out uS or uCR after object creation) will possibly lead to the wrong objects being selected.

hSL – hidden select last object:

- Synopsis: "hSL [count]"
- Background: Yes, Undo: No, Safe: No
- Description: Select the last object in the current level but do not update the GUI.
If an argument is provided, it determines the number of objects to select.
This command does not need an updated hierarchy in the Tcl context. In fact it is completely oblivious of the GUI and therefore safe to use directly after e.g. script based object creation.
Note: prior to Ayam 1.18 this command used to be available in the safe interpreter. This is no longer the case, use "sL" instead.

sP – hidden select parent object:

- Synopsis: "sP [-]"
- Background: Yes, Undo: No, Safe: Yes (*only*)
- Description: Select the current parent object but do not update the GUI. If the optional argument "-" is given, the original selection is restored (from the previous call to this command without this argument).
This command does not need an updated hierarchy in the Tcl context. In fact it is completely oblivious of the GUI and therefore safe to use directly after e.g. script based object creation.
Note that this command is only available in the safe interpreter.

6.2.5 Selecting Points

This command manipulates the point selection.

selPnts – manage the point selection:

- Synopsis: `"selPnts [(-count|-get) [vname] | -has | -all | -none | index1 index2 ...]"`
- Background: Yes, Undo: No, Safe: Yes
- Description: This command manages the point selection.
 - If called without arguments, this command deselects all points.
 - If the argument is `"-count"`, this command puts the number of all currently selected points into the variable specified by the `"vname"` argument. If no variable name is provided, the number is returned.¹
 - If the argument is `"-get"`, this command puts the indices of all currently selected points into the variable specified by the `"vname"` argument. If no variable name is provided, the indices are returned.²
 - If the argument is `"-has"`, this command returns `"1"` if at least one of the selected objects has selected points. Otherwise `"0"` is returned.³
 - If the argument is `"-all"`, this command selects all points.
 - If the argument is `"-none"`, this command de-selects all points.
 - If the argument contains an index, the corresponding point(s) will be added to the selection; already selected points will not be deselected. The index is zero-based and always one dimensional (even for surfaces). Multiple indices may be provided.
- Notes: In contrast to the tag points modelling action, selecting a single point of a multiple point (of NCurve and NPatch objects) via this command only selects this single point.
- Examples: Given that a single NCurve object is selected,
 1. `"selPnts -all"`
selects all points of the curve, and
 2. `"selPnts 0 2"`
selects the first and third point of the curve.

¹ Since 1.26. ² Since 1.26. ³ Since 1.26.

6.2.6 Selecting Faces

This command manipulates the face selection.

selFace – manage face selection:¹

- Synopsis: `"selFace [-count | -get | -has | -set | -clear | -toggle | ind1 ind2 ...]"`
- Background: Yes, Undo: No, Safe: Yes
- Description: This command manages the face selection.
 - If called without arguments, this command returns the face selection state as a list of face indices.
 - If the argument is `"-count"`, this command returns the total number of all currently selected faces.
 - If the argument is `"-get"`, this command returns the indices of all currently selected faces.
 - If the argument is `"-has"`, this command returns `"1"` if at least one of the selected objects has selected faces. Otherwise `"0"` is returned.
 - If the argument is `"-set"`, this command selects all or the faces designated by the index arguments.
 - If the argument is `"-clear"`, this command de-selects all or the faces designated by the index arguments.
 - If the argument is `"-toggle"`, this command reverts the selection state of all or the faces designated by the index arguments.
 - If the argument(s) contains only indices, the corresponding faces(s) will be added to the selection; already selected faces will not be deselected. The index is zero-based.
- Examples: Given that a single PolyMesh object is selected,
 1. `"selFace -clear"`
de-selects all faces of the mesh, and
 2. `"selFace 0 2"`
selects the first and third face of the mesh.

¹ Since 1.33.

6.2.7 Manipulating Properties

These procedures allow easy access to object properties from the scripting interface:¹

getProperty – get single property value

- Synopsis:
`"getProperty (propname(elemname) | elemname) [varname] [-s|-i]"`
- Background: Yes, Undo: No, Safe: Yes, Type: Procedure
- Description: This procedure gets a single property element named `elemname` from the property named `propname` of the currently selected object and writes the result into the variable named `varname`. Multiple selected objects are supported and lead to a list of values produced in the output variable.²
 If the `"varname"` is omitted, the procedure will return the fetched value(s), otherwise the returned value will be `"1"` for successful operations and `"0"` else.³
 If just the `elemname` is provided, the corresponding property element from the type specific attribute property will be retrieved.⁴
 If the `"-s"` option is specified, no errors will be reported.⁵
 If the `"-i"` option is specified, `"RP"` tags will be ignored.⁶
- Notes: This procedure runs more slowly than calling the appropriate get-procedure and accessing the data array that is associated with the property in question directly, especially if multiple values are to be fetched.
 However, the name of the data array and the appropriate procedure are inferred automatically and errors are caught and reported. Furthermore, potential object type initialization and `"NP"/"RP"` tags are also handled transparently.
 See below for more information regarding direct access of property values.
- Example: Given that a single Sphere object is selected, its radius may be retrieved in the variable `"r"` easily using the command

```
getProperty SphereAttr(Radius) r
```

In contrast to using `"getProperty"`, here is an equivalent example for the direct (fast) access of property values:

```
getProp  
set r $::SphereAttrData(Radius)
```

See also section 6.1.5 Global Property Management and Data Arrays (page 347).

¹ Since 1.9. ² Since 1.21. ³ Since 1.27. ⁴ Since 1.35. ⁵ Since 1.27. ⁶ Since 1.27.

setProperty – set single property value

- Synopsis: `"setProperty (propname(elemname) | elemname) value"`
- Background: Yes, Undo: Yes, Safe: Yes, Type: Procedure
- Description: This procedure sets a single property element named `elemname` of the property named `propname` for the currently selected object to the new value given in `value`. Multiple selected objects are supported.¹ If just the `elemname` is provided, the corresponding property element from the type specific attribute property will be set.²
- Notes: This procedure runs more slowly than accessing the data array that is associated with a property and calling the appropriate set-procedure directly, especially if multiple values are to be set. However, the name of the data array and the appropriate procedure are inferred automatically and errors are caught and reported. See below for more information regarding direct access of property values.
- Example: Given that a single Sphere object is selected, its radius may be set to the new value `"3.0"` easily using the command

```
setProperty SphereAttr(Radius) 3.0
```

or, using the simple syntax available since Ayam 1.35:

```
setProperty Radius 3.0
```

In contrast to using `"setProperty"`, here is an equivalent example for the direct (fast) access of property values:

```
getProp  
set SphereAttrData(Radius) 3.0  
setProp
```

See also section [6.1.5 Global Property Management and Data Arrays \(page 347\)](#).

¹ Since 1.21. ² Since 1.35.

6.2.8 Clipboard Operations

These commands operate the object clipboard:

copOb – copy object(s):

- Synopsis: "copOb [-append]"
- Background: Yes, Undo: No, Safe: Yes
- Description: Copy the selected object(s) to the object clipboard after clearing it.
This is a deep copy, reference counters of master or material objects may be increased and there are special rules for instances, see also section [4.2.7 Instances and the Object Clipboard \(page 132\)](#).
If the option "-append" is used, the clipboard will not be cleared before this operation.
- Notes: May fail if the clipboard contains referenced objects. If the append option is set, this operation only fails if there is not enough memory.

cutOb – cut object(s):

- Synopsis: "cutOb [-append]"
- Background: Yes, Undo: No, Safe: Yes
- Description: Move the selected object(s) and their children into the object clipboard after clearing it.
If the option "-append" is used, the clipboard will not be cleared before this operation.
- Notes: May fail if the clipboard contains referenced objects. If the append option is set, this operation never fails.
This command always clears the object selection.

pasOb – paste object(s):

- Synopsis: "pasOb [-move]"
- Background: Yes, Undo: No, Safe: Yes
- Description: Copy all object(s) from the object clipboard to the current level.
This is a deep copy, reference counters of master or material objects may be increased and there are special rules for instances, see also section [4.2.7 Instances and the Object Clipboard \(page 132\)](#).
If the option "-move" is given, the objects are moved and not copied, i.e. after a "pasOb -move", the clipboard is empty and no reference counters will be changed.
- Notes: This operation may fail if the result would be an illegal instance configuration (i.e. recursive references).

repOb – replace clipboard content with selected object(s):

- Synopsis: "repOb"
- Background: Yes, Undo: No, Safe: Yes
- Description: Swap all objects from the object clipboard with the currently selected object(s). If multiple objects are selected in non consecutive sequences, only the first consecutive sequence or single object is replaced.
- Notes: This operation may fail if the result would be an illegal instance configuration (i.e. recursive references).

clearClip – clear object clipboard:

- Synopsis: "clearClip"
- Background: Yes, Undo: No, Safe: Yes
- Description: Clears the object clipboard.
- Notes: May fail if the clipboard contains referenced objects.

The following procedures operate the property clipboard, which is totally independent from the object clipboard.

copyProp – copy a property to the property clipboard

- Synopsis: "copyProp [mode]"
- Background: Yes, Undo: No, Safe: No, Type: Procedure
- Description: Copy the currently selected property from the currently selected object to the property clipboard. If `mode` is 0 (default), all marked entries will be omitted. If `mode` is 1 only the marked entries will be copied. Property entries are usually marked by double clicks on the respective entry labels but they can also be marked programmatically by adding the respective property element names to the global array "pclip_omit".

pasteProp – paste a property

- Synopsis: "pasteProp"
- Background: Yes, Undo: Yes, Safe: No, Type: Procedure
- Description: Copy the property from the property clipboard to the currently selected object.

6.2.9 Hierarchy Operations

These commands manipulate the current level of Ayam:

goDown:

- Synopsis: "goDown index"
- Background: Yes, Undo: No, Safe: Yes
- Description: Enter the object determined by `index`. If `index` is 0 and the current level is inside some object the parent level will be entered instead. If `index` is -1, the last object of the current level will be entered.
- Notes: The object selection is *not* synchronized.
- Example: "**crtOb Level; goDown -1; crtOb Sphere**" creates a simple Level, enters it, and creates a Sphere as child of this level.

goUp:

- Synopsis: "goUp"
- Background: Yes, Undo: No, Safe: Yes
- Description: Go one level up in the object hierarchy.
- Notes: The object selection is *not* synchronized.

goTop:

- Synopsis: "goTop"
- Background: Yes, Undo: No, Safe: Yes
- Description: Go to the top level of the object hierarchy.
- Notes: The object selection is *not* synchronized.

The following commands move objects around in the hierarchy:

upOb – shuffle object(s) backward in the current level:

- Synopsis: "upOb"
- Background: Yes, Undo: No, Safe: Yes
- Description: Shuffle the currently selected object(s) backwards (towards the first object) in the current level. Non contiguous selections are supported. If the first of the selected objects is the first in the current level, nothing will be changed, i.e. either all selected objects move, or none. The moved objects stay selected, i.e. this command can be used multiple times in a row.

downOb – shuffle object(s) forward in the current level:

- Synopsis: "downOb"
- Background: Yes, Undo: No, Safe: Yes
- Description: Shuffle the currently selected object(s) forwards (towards the last object) in the current level. Non contiguous selections are supported. If the last of the selected objects is the last in the current level, nothing will be changed, i.e. either all selected objects move, or none. The moved objects stay selected, i.e. this command can be used multiple times in a row.

6.2.10 Transformations

These commands transform objects or selected points of objects:

movOb – move objects:

- Synopsis: `"movOb dx dy dz"`
- Background: Yes, Undo: Yes, Safe: Yes
- Description: Move the selected object(s) by `dx` in direction of the objects X-axis, by `dy` in direction of the objects Y-axis, and by `dz` in direction of the objects Z-axis.

rotOb – rotate objects:

- Synopsis: `"rotOb (dx dy dz | -a ax ay az a) "`
- Background: Yes, Undo: Yes, Safe: Yes
- Description: Rotate the selected object(s) by `dx` degrees around the objects X-axis, then by `dy` degrees around objects Y-axis, and finally by `dz` degrees around the objects Z-axis. Note the order of the rotations.
If the parameters start with an `-a`, they denote an rotation axis followed by the angle in degrees.¹
- Examples:
 1. `"rotOb 0 0 45"`
rotates the selected objects by an angle of 45° about the Z-axis.
 2. `"rotOb -a 1 1 0 45"`
rotates the selected objects by an angle of 45° about the axis `"1 1 0"`.

scalOb – scale objects:

- Synopsis: `"scalOb dx dy dz"`
- Background: Yes, Undo: Yes, Safe: Yes
- Description: Scale the selected object(s) by a factor of `dx` in direction of the objects X-axis, by a factor of `dy` in direction of the objects Y-axis, and by a factor of `dz` in direction of the objects Z-axis.
- Note: A scale factor of zero is generally a bad idea and thus will be changed to 1.0 silently.

movPnts – move selected points:

- Synopsis: `"movPnts dx dy dz"`
- Background: Yes, Undo: Yes, Safe: Yes
- Description: Move the selected points by `dx` in direction of the objects X-axis, by `dy` in direction of the objects Y-axis, and by `dz` in direction of the objects Z-axis.

¹ Since 1.27.

rotPnts – rotate selected points:

- Synopsis: `"rotPnts (dx dy dz | -a ax ay az a) "`
- Background: Yes, Undo: Yes, Safe: Yes
- Description: Rotate the selected points by `dx` degrees around the objects X-axis, then by `dy` degrees around objects Y-axis, and finally by `dz` degrees around the objects Z-axis. Note the order of the rotations.
If the parameters start with an `-a`, they denote an rotation axis followed by the angle in degrees.¹
- Examples:
 1. `"rotPnts 0 0 45"`
rotates the selected points by an angle of 45° about the Z-axis.
 2. `"rotPnts -a 1 1 0 45"`
rotates the selected points by an angle of 45° about the axis `"1 1 0"`.

scalPnts – scale selected points:

- Synopsis: `"scalPnts dx dy dz "`
- Background: Yes, Undo: Yes, Safe: Yes
- Description: Scale the selected points by a factor of `dx` in direction of the objects X-axis, by a factor of `dy` in direction of the objects Y-axis, and by a factor of `dz` in direction of the objects Z-axis.
- Note: A scale factor of zero is generally a bad idea and thus will be changed to 1.0 silently.

delegTrafo – delegate transformations:

- Synopsis: `"delegTrafo"`
- Background: Yes, Undo: Yes, Safe: Yes
- Description: Delegates the transformations associated with the selected objects to their child objects. Additionally, the transformations of the selected objects will be reset to the default values.
This operation fails for complex setups (i.e. if the combination of parent and child transformation is a shear transformation).

applyTrafo – apply transformations:

- Synopsis: `"applyTrafo [-sel] "`
- Background: Yes, Undo: Yes, Safe: Yes
- Description: Applies the transformations encoded in the transformation attributes of the selected objects to their points. Either all points (default) or just the selected ones (if the option `"-sel"` is given) are modified. It is no error, if an object has no points at all or if the points are readonly. Additionally, if any points of an object are modified, the transformations of this object will be reset to the default values.

¹ Since 1.27.

normTrafos – normalize transformation values:

- Synopsis: "normTrafos"
- Background: Yes, Undo: Yes, Safe: Yes
- Description: Normalize all transformation attribute values of the selected objects. The values will be rounded to the number of significant digits specified via the hidden preference setting "NormalizeDigits" (by default 6).

normPnts – normalize points:

- Synopsis: "normPnts"
- Background: Yes, Undo: Yes, Safe: Yes
- Description: Normalize all selected points of the selected objects. The values will be rounded to the number of significant digits specified via the hidden preference setting "NormalizeDigits" (by default 6).

normVar – normalize variable(s):

- Synopsis: "normVar varname"
- Background: Yes, Undo: Yes, Safe: Yes
- Description: Normalize the value of the designated variable(s). The value of the variable will be rounded to the number of significant digits specified via the hidden preference setting "NormalizeDigits" (by default 6). Lists are also supported.¹

normVal – normalize double value(s):

- Synopsis: "normVar value"
- Background: Yes, Undo: Yes, Safe: Yes
- Description: Returns the normalized value(s). Lists are supported. The value(s) will be rounded to the number of significant digits specified via the hidden preference setting "NormalizeDigits" (by default 6).

¹ Since 1.30.

6.2.11 Manipulating Shaders

These commands operate the shader properties:

shaderSet:

- Synopsis: `"shaderSet shadertype [varname]"`
- Background: Yes, Undo: Yes, Safe: Yes
- Description: Set the shader of type `shadertype` for the selected object. Type may be one of `"surface"`, `"displacement"`, `"light"`, `"imager"`, `"atmosphere"`, `"exterior"` or `"interior"`. If `varname` is not given, the shader in question is deleted from the object instead. Otherwise, `varname` points to an associative array that contains the data (arguments) of the shader. Example content may be created with the `"shaderGet"` command below. The data is *not* checked against the internal shader database for correctness or completeness. This command fails, if the selected object does not support a shader of given type.

shaderGet:

- Synopsis: `"shaderGet shadertype varname"`
- Background: Yes, Undo: No, Safe: Yes
- Description: Get the shader of type `shadertype` for the selected object. Type may be one of `"surface"`, `"displacement"`, `"light"`, `"imager"`, `"atmosphere"`, `"exterior"` or `"interior"`. The shader data will be written to an associative array pointed to by `varname`.

6.2.12 Manipulating Tags

These commands may be used to modify the tags of an object (see also section 4.11 Tags (page 258)).

When processing tags of unknown or unregistered type, a corresponding warning message may be emitted. This warning can be inhibited using the hidden preference setting "WarnUnknownTag" or by registering the tag type using the "registerTag" command (see below).

setTag:

- Synopsis: "setTag type value"
- Background: Yes, Undo: Yes, Safe: Yes
- Description: Set a tag with type-string `type` and value-string `value` to the currently selected object(s).
If a tag of matching type already exists, the new value will be set to the first matching tag. Otherwise a new tag will be added to the selected object(s);
It is legal to deliver "" as value parameter. This is e.g. needed for the "NoExport" tag type.
- Examples:
 1. **"setTag NoExport ""**
sets or adds a "NoExport" tag to the selected objects.
 2. **"setTag RP Transformations"**
sets or adds a "RP" (remove property) tag to the selected objects that hides the Transformations property GUI.

addTag:

- Synopsis: "addTag type value"
- Background: Yes, Undo: Yes, Safe: Yes
- Description: Add a tag with type-string `type` and value-string `value` to the currently selected objects.
It is legal to deliver "" as value parameter. This is e.g. needed for the "NoExport" tag type.
Potentially existing tags of the designated type will *not* be changed.
- Examples:
 1. **"addTag NoExport ""**
adds a "NoExport" tag to the selected objects.
 2. **"addTag RP Transformations"**
adds a "RP" (remove property) tag to the selected objects that hides the Transformations property GUI.

hasTag:

- Synopsis: "hasTag type [value]"
- Background: Yes, Undo: No, Safe: Yes
- Description: This command returns "1" if the selected object has at least one tag of the designated type. Otherwise "0" is returned. If multiple objects are selected, a list of values is returned.¹
If a value is provided, this string is matched against the potential tag value, it may also contain any of *?[] for more complex forms of matching.

¹ Since 1.29.

delTags:

- Synopsis: `"delTags [type]"`
- Background: Yes, Undo: Yes, Safe: Yes
- Description: Delete all tags of designated type from the currently selected objects. If the type parameter is omitted or `"all"`, *all* tags are deleted from the currently selected objects.
- Examples:
 1. **"delTags"**
removes all tags from the selected objects.
 2. **"delTags RP"**
removes all `"RP"` tags from the selected objects.

getTags:

- Synopsis: `"getTags tvname vvname"`
- Background: Yes, Undo: No, Safe: Yes
- Description: Get all tags from the currently selected object and put them as lists into two variables named `tvname` for the tag types and `vvname` for the tag values.

setTags:

- Synopsis: `"setTags taglist"`
- Background: Yes, Undo: Yes, Safe: Yes
- Description: Clear all tags from the currently selected objects and set all new tags from the list `taglist`. The tag types are taken from the list elements with even index numbers and the tag value-strings from the list elements with odd index numbers.
- Examples:
 1. **"setTags {RP Transformations RP Attributes}"**
replaces all tags from the selected objects with two `"RP"` tags.

getTag:

- Synopsis: `"getTag type [vname]"`
- Background: Yes, Undo: No, Safe: Yes
- Description: Get the value of a specific tag from the currently selected object and put it into the designated variable. If there is no tag of the specified type, the variable will be set to `" "`. If no variable name is provided, the value of the tag is returned.

registerTag:

- Synopsis: `"registerTag type"`
- Background: Yes, Undo: No, Safe: Yes
- Description: This command registers the designated tag type.

6.2.13 Manipulating Curves

These commands operate on parametric curves:

openC – open curve:

- Synopsis: "openC"
- Background: Yes, Undo: Yes, Safe: Yes
- Description: Open the selected curve objects.
See also section [5.3.2 Open Tool \(page 280\)](#).

closeC – close curve:

- Synopsis: "closeC"
- Background: Yes, Undo: Yes, Safe: Yes
- Description: Close the selected curve objects.
See also section [5.3.3 Close Tool \(page 281\)](#).

refineC – refine curve:

- Synopsis: "refineC"
- Background: Yes, Undo: Yes, Safe: Yes
- Description: Refine the control vector of the selected curve objects by inserting a new control point into every control point interval, changing the shapes of the curves. If a curve has selected points, only the interval between the first and the last of the selected points is refined. For periodic NURBS curves, the last p intervals are not refined (where p is the degree of the curve).
See also section [5.3.4 Refine Tool \(page 282\)](#).

coarsenC – coarsen curve:

- Synopsis: "coarsenC"
- Background: Yes, Undo: Yes, Safe: Yes
- Description: Remove every second control point from the selected curve objects. If a curve has selected points, only the interval between the first and the last of the selected points is affected. For periodic NURBS curves, the last p intervals are not affected (where p is the degree of the curve).
- Notes: Prior to Ayam 1.26 this command was named "coarsenNC", the old name is still available for compatibility but its use is deprecated.
See also section [5.3.7 Coarsen Tool \(page 285\)](#).

revertC – revert curves:

- Synopsis: "revertC"
- Background: Yes, Undo: Yes, Safe: Yes
- Description: Revert the direction of the selected curve objects.
See also section [5.3.1 Revert Tool \(page 279\)](#).

shiftC – shift control points of a (closed) curve:

- Synopsis: `"shiftC i"`
- Background: Yes, Undo: Yes, Safe: Yes
- Description: shifts the control points of the selected NCurve, ACurve, and ICurve objects by an amount specified by the parameter `i` (which may be negative to revert the direction of the shifting). For a simple closed curve, shifting with `i=1`, the first control point will get the coordinates of the former last control point. This means, positive shifts occur in the direction of the curve. Note that for closed and periodic NURBS curves, the multiple points will be managed correctly.
See also section [5.3.24 Shift Closed Curve Tool \(page 303\)](#).

toXYC – rotate curve to XY-plane

- Synopsis: `"toXYC"`
- Background: Yes, Undo: Yes, Safe: Yes
- Description: rotates the control points of the selected planar NCurve, ACurve, and ICurve objects to their respective XY-plane, modifying the transformation attributes accordingly.
See also section [5.3.25 To XY Tool \(page 304\)](#).

concatC – concatenate curves:

- Synopsis: `"concatC [-c closed | -k knottype | -f fillets | -fl len]"`
- Background: Yes, Undo: No, Safe: No
- Description: Concatenate the selected curve objects into a single NURBS curve. The order of the new curve is taken from the first curve.
The option `"-c"` allows to create a closed curve (0 – open, 1 – closed; default 0).
The option `"-k"` allows to set a knot type (0 – NURB, 1 – Custom; default 0 – NURB).
The option `"-f"` determines whether or not fillets should be created (0 – no, 1 – yes; default 0).
The option `"-fl"` allows to set the length of the fillets.
See also section [4.5.1 ConcatNCAttr Property \(page 163\)](#) for more information on these options.

6.2.14 Manipulating Surfaces

These commands operate on parametric surfaces:

revertuS – revert surfaces:

- Synopsis: "revertuS"
- Background: Yes, Undo: Yes, Safe: Yes
- Description: Revert the direction of the selected surfaces in the u parametric dimension.
See also section 5.5.1 Revert U Tool (page 317).

revertvS – revert surfaces:

- Synopsis: "revertvS"
- Background: Yes, Undo: Yes, Safe: Yes
- Description: Revert the direction of the selected surfaces in the v parametric dimension.
See also section 5.5.2 Revert V Tool (page 317).

swapuvS – swap dimensions of surfaces:

- Synopsis: "swapuvS"
- Background: Yes, Undo: Yes, Safe: Yes
- Description: Swap the dimensions of the selected surfaces (objects of type NPatch, IPatch, BPatch, and PatchMesh), thus exchanging width and height, without altering the shape of the surfaces.
See also section 5.5.3 Swap UV Tool (page 317).

concatS – concatenate surfaces:

- Synopsis: "concatS [-o order | -t type | -k knottype | -u uvselect]"
- Background: Yes, Undo: No, Safe: Yes
- Description: Concatenate the selected surface objects into a single NURBS patch.
The option "-o" determines the desired order of the surface in U direction.
The option "-t" allows to set a surface type (0 – open, 1 – closed, 3 – periodic; default 0).
The option "-k" allows to set a knot type (default 1 – NURB).
Finally, the "-u" option allows to specify the uv-select-string.
See also section 4.7.11 ConcatNPAttr Property (page 214) for more information on these options.

capS – cap surfaces:

- Synopsis: "capS [-mp mp] side"
- Background: Yes, Undo: Yes, Safe: Yes
- Description: Extend the selected surface objects to a single point, effectively capping them.¹ For NURBS surfaces, the respective border is copied to order-1 multiplicity, so that
 - there will be a sharp corner between original surface and cap surface,
 - the cap surface will be flat (if the original border is flat).
 The option "-mp" allows to specify the point to extend to.
The side parameter must be an integer between 0 and 3, where 0 and 1 extend the surface at U0 and Un respectively, and 2 and 3 extend the surface at V0 and Vn respectively.

¹ Since 1.33.

6.2.15 Manipulating NURBS Curves

These are more specialized commands to change NURBS curve properties:

clampNC – clamp NURBS curve:

- Synopsis: `"clampNC [-s|-e]"`
- Background: Yes, Undo: Yes, Safe: Yes
- Description: Clamp the knot vector of the selected NURBS curves without changing the shape of the curves. The knot type of the clamped curve will be changed to "Custom" and the knots will have o equal values at the desired side(s), where o is the order of the curve.

If the side parameter is omitted, both sides are clamped. If the side parameter is "-s" only the start, and if it is "-e" only the end is clamped.

In Ayam versions prior to 1.18 it was an error if the curve was already clamped at either side, this is no longer the case. Furthermore, curves with multiple knots in the end region(s) could not be clamped, this works ok now.

See also section [5.3.11 Clamp Tool \(page 289\)](#).

unclampNC – unclamp NURBS curve:

- Synopsis: `"unclampNC [-s|-e]"`
- Background: Yes, Undo: Yes, Safe: Yes
- Description: Unclamp the knot vector of the selected NURBS curves without changing the shape of the curves. The knot type of the clamped curve can be changed to "Custom".

If the side parameter is omitted, both sides are unclamped. If the side parameter is "-s" only the start, and if it is "-e" only the end is unclamped.

See also section [5.3.12 Unclamp Tool \(page 290\)](#).

extendNC – extend NURBS curve:

- Synopsis: `"extendNC (x y z [w] | -vn varname | -m)"`
- Background: Yes, Undo: Yes, Safe: Yes
- Description: Extend the selected NURBS curves to a given point or to the mark without changing the current shape of the curve.

See also section [5.3.10 Extend Tool \(page 288\)](#).

elevateNC – increase order of NURBS curve:

- Synopsis: `"elevateNC [n]"`
- Background: Yes, Undo: Yes, Safe: Yes
- Description: Increases the order of the selected NURBS curves without changing the shape of the curves by n . If the parameter n is omitted, a default value of 1 is used. The knot type of the elevated curves will be changed to "Custom".

See also section [5.3.8 Elevate Tool \(page 286\)](#).

reduceNC – decrease order of NURBS curve:

- Synopsis: `"reduceNC [tol]"`
- Background: Yes, Undo: Yes, Safe: Yes
- Description: Decrease the order of the selected NURBS curves by one if the shape of the reduced curves does not deviate from the original curves by `tol` in any point. If the parameter `tol` is omitted, a default value of 0.0 is used, i.e. the order is only reduced if the curve does not change. The knot type of the reduced curves will be changed to "Custom".

See also section [5.3.9 Reduce Tool \(page 287\)](#).

insknNC – insert knot into NURBS curve:

- Synopsis: `"insknNC u r"`
- Background: Yes, Undo: Yes, Safe: Yes
- Description: Insert a new knot at the position specified by `u` `r` times. The valid range for `u` is determined by the current knot vector `U` as follows: $U[p] \leq u \leq U[n]$, where `p` is the degree (order-1) of the curve and `n` is the length of the curve. The knot type of the curves will always be changed to custom but the shape of the curves will not change. See also section [5.3.13 Insert Knot Tool \(page 291\)](#).

remknNC – remove knot from NURBS curve:

- Synopsis: `"remknNC (u | -i ind) r [tol]"`
- Background: Yes, Undo: Yes, Safe: Yes
- Description: Remove a knot at the position specified by `u` (`u` must be in the valid range of the knot vector of the selected curve) `r` times from the curve. Since Ayam 1.20 the knot to remove may also be specified using its (zero based) index in the knot vector (i.e. use `"remknNC -i 3 1"` instead of `"remknNC 0.5 1"` for the knot vector `"0 0 0 0.5 1 1 1"`).

Note that the shape of the curve may be changed by this tool unless the parameter `tol` is specified. If `tol` is specified the new curve does not deviate from the original curve more than `tol` in any point on the curve. If the knot can not be removed `r` times due to the tolerance, an error is reported and the original curve is left unchanged.

This operation also fails, if the knot removal would lead to a curve of lower order.

See also section [5.3.14 Remove Knot Tool \(page 292\)](#).

remsuknNC – remove superfluous knots from NURBS curve:

- Synopsis: `"remsuknNC [tol]"`
- Background: Yes, Undo: Yes, Safe: Yes
- Description: Remove all knots from the NURBS curve that do not contribute to its shape. This may lead to control point removal and changes, but the shape of the curve does not change unless a large tolerance value is specified via the `tol` parameter. The default value of this parameter is 0.0.

It is no error if no knots can be removed.

See also section [5.3.15 Remove Superfluous Knots Tool \(page 293\)](#).

refineknNC – refine knots of NURBS curve:

- Synopsis: `"refineknNC [{u1 u2 un}] "`
- Background: Yes, Undo: Yes, Safe: Yes
- Description: Refine the knot vector of the selected NURBS curve without changing the shape of the curve with n new knots $\{u1\ u2\ un\}$. If no list of new knots is given, a new knot is inserted into each interval in the old knot vector.

See also section 5.3.5 Refine Knots Tool (page 283).

tweenNC – interpolate (tween) curves:

- Synopsis: `"tweenNC [r] "`
- Background: Yes, Undo: No, Safe: Yes
- Description: Interpolate (tween) between the first two selected NURBS curves, creating a new curve that incorporates features from the selected curves. The parameter r defines the ratio of influence of the first and the second curve (the latter using $1-r$). This parameter defaults to 0.5. The first two curves must be of the same length and order. They need not be defined on the same knot vector, however. If a third curve is selected, the parameter r is ignored and this third curve defines the ratio of influence with its y coordinates.

See also section 5.2.5 Tween Curve Tool (page 278).

rescaleknNC – rescale knots of NURBS curves:

- Synopsis: `"rescaleknNC [-r rmin rmax | -d mindist] "`
- Background: Yes, Undo: Yes, Safe: Yes
- Description: Rescale the knot vector(s) of the selected NURBS curve(s) to the range $[0.0, 1.0]$ (if no argument is present) or to the range $[rmin, rmax]$ if the `"-r"` argument is given or to the minimum distance `mindist` if the `"-d"` argument is used. Scaling to a minimum distance ensures that all knots (except for multiple knots) have a distance bigger than `mindist` afterwards.

Since Ayam 1.20 the knot type of the curve does not have to be `"Custom"` anymore. Furthermore, rescaling the knots does not change the knot type.

This operation does not change the shape of the curve.

See also section 5.3.27 Rescale Knots to Range Tool (page 306).

splitNC – split NURBS curve

- Synopsis: `"splitNC [-a|-r] u"`
- Background: Yes, Undo: No, Safe: Yes
- Description: splits the selected NURBS curves at the designated parametric value u into two curves, creating one new curve and *modifying the original* selected curve. If the option `"-r"` is present, the parametric value is interpreted in a relative way and should therefore be in the range $[0, 1]$.¹

If the option `"-a"` is present, the new curve(s) will be appended to the current level. Otherwise the new curve(s) will be inserted into the level right after the respective curve(s) to split.²

See also section 5.3.18 Split Tool (page 297).

¹ Since 1.28. ² Since 1.24.

extrNC – extract NURBS curve:

- Synopsis: "extrNC [-relative] umin umax"
- Background: Yes, Undo: No, Safe: Yes
- Description: Extracts a sub-curve from the selected NCurve or NCurve providing objects. The extracted curve(s) will be appended as new object(s) to the current level of the scene. The sub-curve to be extracted is specified by the parametric values `umin` and `umax` which have to be in the respective valid knot range.

If the optional argument "`-relative`" is specified, the parametric values are interpreted in a relative way and must consequently be in the range `[0, 1]`.

trimNC – trim NURBS curve

- Synopsis: "trimNC [-relative] umin umax"
- Background: Yes, Undo: Yes, Safe: Yes
- Description: trims the selected NURBS curve to the designated parametric range (`umin-umax`), modifying the original selected curve.

If the optional argument "`-relative`" is specified, the parametric values are interpreted in a relative way and must consequently be in the range `[0, 1]`.

See also section 5.3.19 [Curve Trim Tool](#) (page 298).

estlenNC – estimate length of NURBS curve:

- Synopsis: "estlenNC [-trafo | -refine n] [varname]"
- Background: Yes, Undo: No, Safe: Yes
- Description: estimate the length of the currently selected NURBS curve or NURBS curve providing object and put the result into the designated variable.
If multiple curves are selected or provided, a list of results is created.
If the optional parameter "`-trafo`" is given, the transformation attributes of the curve will be applied to the control points for the length estimation.
If the optional parameter "`-refine`" is given, the curve will be refined `n` times prior to the length estimation which increases the accuracy of the estimation.¹
If no variable name is specified the command returns the respective result(s).²

reparamNC – reparameterise a NURBS curve:

- Synopsis: "reparamNC type"
- Background: Yes, Undo: Yes, Safe: Yes
- Description: reparameterise all selected NURBS curves to have chordal knots (type: 0), or centripetal knots (type: 1). The knot type of the curve will be changed to "`Custom`". See also section 5.3.20 [Reparameterisation Tool](#) (page 299).

¹ Since 1.27. ² Since 1.25.

isCompNC:

- Synopsis: `"isCompNC [-l level]"`
- Background: Yes, Undo: No, Safe: Yes
- Description: This command returns 1 if the selected NURBS curves are compatible (i.e. defined on the same knot vector), otherwise it returns 0.
If `"level"` is 0, only the orders of the curves are compared.
If `"level"` is 1, only the orders and lengths of the curves are compared.

makeCompNC – make NURBS curves compatible

- Synopsis: `"makeCompNC [-f | -l level]"`
- Background: Yes, Undo: Yes, Safe: Yes
- Description: makes the selected NURBS curves compatible i.e. of the same order and defined on the same knot vector.
If the option `"-f"` is present, there will be no prior compatibility check.
If `"level"` is 0, only the orders will be adapted.
If `"level"` is 1, only the orders and lengths will be adapted.
See also section [5.3.26 Make Compatible Tool \(page 305\)](#).

interpNC – interpolate NURBS curve:

- Synopsis: `"interpNC [-order order | -ptype type | -closed (0|1) | -sdlen length | -edlen length]"`
- Background: Yes, Undo: Yes, Safe: Yes
- Description: Interpolate the selected NURBS curves with desired order and parameterisation type. Order defaults to 4 and must be higher than 2. The parameterisation type must be one of 0 – Chordal, 1 – Centripetal, or 2 – Uniform, default is Chordal. The interpolation can create a closed curve if requested by the `"-closed"` parameter but this will also increase the length of the resulting NURBS curve. The default is to create an open curve for open input curves and a closed curve for closed or periodic input curves.
Using the options `"-sdlen"` and `"-edlen"` (which both default to 0.0) the length of automatically created start/end derivatives can be adjusted. If any of these are not 0.0, a different interpolation algorithm will be used, which increases the length of the resulting NURBS curve.
The curve will interpolate all current control points after the interpolation and the position of certain control points will be changed in this process so that, after interpolation, the new control points will *not* be interpolated by the curve. The curve will rather interpolate the *old* control point positions.
The knot type of the interpolated curves will be changed to `"Custom"`.
See also section [5.3.22 Interpolate Tool \(page 301\)](#).

approxNC – approximate NURBS curve:

- Synopsis:

```
"approxNC [-order o | -length l | -closed (0|1) | -tessellate t |
-ptype (0|1|2)]"
```

- Background: Yes, Undo: Yes, Safe: Yes
- Description: Approximate the control points or a tessellation of the selected NURBS curves with a new non-rational NURBS curve of desired order and length.

The `"-order"` parameter defaults to the order of the respective NURBS curve and must be at least 2. If a smaller value is supplied the command will silently fall back to the default. The order must also be smaller or equal to the length of the new curve.

The `"-length"` parameter defaults to the length of the respective NURBS curve and must be higher than 2. If a smaller value is supplied the command will silently fall back to the default. The length must not be higher than the number of data points to approximate.

If the `"-closed"` parameter is set to `"1"` a closed periodic curve will be created. By default, whether or not a closed curve shall be created will be derived from the type of the curve to be approximated.

The `"-tessellate"` parameter allows to control the tessellation of the curve to be approximated. If this parameter is not present or zero, no tessellation will be carried out and the control points of the curve will be approximated. The tessellation parameter directly controls the number of intermediate points to generate for every distinct knot interval in the originating curves knot vector. The total number of data points generated by the tessellation is $nd + (nd - 1) \times t$, where nd is the total number of distinct knots in the valid range of the knot vector.

The `"-ptype"` parameter allows to choose a parameterization type: `"0"` centripetal, `"1"` chordal, or `"2"` uniform.

The knot type of the processed curves will be changed to `"Custom"`.

See also section [5.3.23 Approximate Tool \(page 302\)](#).

- Examples:

1. `"crtOb NCurve; selOb -1; approxNC -l 3 -o 3"`
creates a standard NURBS curve and approximates the control points of it with a new curve of length 3 and order 3.
2. `"crtOb NCurve; selOb -1; approxNC -l 5 -o 3 -t 3"`
creates a standard NURBS curve and approximates a tessellation of it with a new curve of length 5 and order 3.

curvatNC – compute curvature:

- Synopsis: `"curvatNC [-r] -u u"`
- Background: Yes, Undo: No, Safe: Yes
- Description: Compute and return the curvature of the selected NURBS curve or NURBS curve providing object at the designated parametric value `u`.
If the option `"-r"` is present, the parametric value is interpreted in a relative way and should therefore be in the range `[0, 1]`.
If multiple objects are selected, a list of curvature values is returned.
See also section [5.3.21 Plot Curvature Tool \(page 300\)](#).

torsionNC – compute torsion:

- Synopsis: `"torsionNC [-r] -u u"`
- Background: Yes, Undo: No, Safe: Yes
- Description: Compute and return the torsion of the selected NURBS curve or NURBS curve providing object at the designated parametric value `u`.¹
If the option `"-r"` is present, the parametric value is interpreted in a relative way and should therefore be in the range `[0, 1]`.
If multiple objects are selected, a list of torsion values is returned.

distNC – compute distance between two NURBS curves:²

- Synopsis: `"distNC [mode] "`
- Background: Yes, Undo: No, Safe: Yes
- Description: This command computes and returns the distance between the first two selected NURBS curves. The mode parameter controls whether the arithmetic mean (0, default), minimum (1), or maximum (2) distance shall be computed.
This command supports arbitrary object types that just need to provide NURBS curves.³

fairNC – improve curve shape:

- Synopsis: `"fairNC [tol] "`
- Background: Yes, Undo: Yes, Safe: Yes
- Description: Improve the curve shape by moving control points so that the curvature is distributed more evenly.
If a tolerance value is present the processed control points do not move more than the given value.
If points are selected, only these will be processed.
See also section [5.3.16 Fair Tool \(page 294\)](#).

¹ Since 1.30. ² Since 1.33. ³ Since 1.34.

6.2.16 Manipulating NURBS Surfaces

These are more specialized commands to change NURBS surface properties:

clampuNP – clamp NURBS patch in U direction:

- Synopsis: `"clampuNP [-s|-e]"`
- Background: Yes, Undo: Yes, Safe: Yes
- Description: Clamp the U direction knot vector of the selected NURBS patches without changing the shape of the patches. The knot type will be changed to "Custom" and the knots will have o equal values at start and end (where o is the order of the patch in U direction).

If the side parameter is omitted, both sides are clamped. If the side parameter is "-s" only the start, and if it is "-e" only the end is clamped.

In Ayam versions prior to 1.18 it was an error if the patch was already clamped at either side, this is no longer the case. Furthermore, patches with multiple knots in the end region(s) could not be clamped, this works ok now.

See also section 5.5.10 Clamp Surface Tool (page 323).

clampvNP – clamp NURBS patch in V direction:

- Synopsis: `"clampvNP [-s|-e]"`
- Background: Yes, Undo: Yes, Safe: Yes
- Description: Clamp the V direction knot vector of the selected NURBS patches without changing the shape of the patches. The knot type will be changed to "Custom" and the knots will have o equal values at start and end (where o is the order of the patch in V direction).

If the side parameter is omitted, both sides are clamped. If the side parameter is "-s" only the start, and if it is "-e" only the end is clamped.

In Ayam versions prior to 1.18 it was an error if the patch was already clamped at either side, this is no longer the case. Furthermore, patches with multiple knots in the end region(s) could not be clamped, this works ok now.

See also section 5.5.10 Clamp Surface Tool (page 323).

unclampuNP – unclamp NURBS patch in U direction:

- Synopsis: `"unclampuNP [-s|-e]"`
- Background: Yes, Undo: Yes, Safe: Yes
- Description: Unclamp the U direction knot vector of the selected NURBS patches without changing the shape of the patches. The knot type can be changed to "Custom".

If the side parameter is omitted, both sides are unclamped. If the side parameter is "-s" only the start, and if it is "-e" only the end is unclamped.

See also section 5.5.11 Unclamp Surface Tool (page 324).

unclampvNP – unclamp NURBS patch in V direction:

- Synopsis: `"unclampvNP [-s|-e]"`
- Background: Yes, Undo: Yes, Safe: Yes
- Description: Unclamp the V direction knot vector of the selected NURBS patches without changing the shape of the patches. The knot type can be changed to "Custom".

If the side parameter is omitted, both sides are unclamped. If the side parameter is "-s" only the start, and if it is "-e" only the end is unclamped.

See also section 5.5.11 [Unclamp Surface Tool](#) (page 324).

rescaleknNP – rescale knots of NURBS patches:

- Synopsis: `"rescaleknNP [-r[u|v] rmin rmax | -d[u|v] mindist]"`
- Background: Yes, Undo: Yes, Safe: Yes
- Description: Rescale the knot vector(s) of the selected NURBS patch(es) to the range [0.0, 1.0] (if no argument is present) or to the range [rmin, rmax] if the "-r" argument is given or to the minimum distance mindist if the "-d" argument is used. Scaling to a minimum distance ensures that all knots (except for multiple knots) have a distance bigger than mindist afterwards.

The "-ru", "-rv", "-du", and "-dv" variants scale only the designated dimension.

Trim curves, if present, will also be scaled to match the new range.

Since Ayam 1.20 the knot type of the curve does not have to be "Custom" anymore. Furthermore, rescaling the knots does not change the knot type.

This operation does not change the shape of the patch.

See also sections 5.5.20 [Rescale Knots to Range Surface Tool](#) (page 333) and 5.5.21 [Rescale Knots to Mindist Surface Tool](#) (page 333).

- Example: `"rescaleknNP -ru 0.2 0.3"` scales the u knot vector of the selected NURBS patch objects to the new range [0.2, 0.3].

insknNP – insert knot into NURBS patch:

- Synopsis: `"insknNP u r"`
- Background: Yes, Undo: Yes, Safe: Yes
- Description: Insert a new knot in U direction at the position specified by u, r times. u must be in the valid range of the corresponding knot vector of the selected patches. The valid range is determined by the current knot vector U as follows: $U[p] \leq u \leq U[n]$, where p is the degree (order-1) of the patch in U direction and n is the width of the patch. The u knot type of the patches will always be changed to "Custom" but the shape of the patches will not change.

See also section 5.5.12 [Insert Knot Surface Tool](#) (page 325).

insknvNP – insert knot into NURBS patch:

- Synopsis: `"insknvNP v r"`
- Background: Yes, Undo: Yes, Safe: Yes
- Description: Insert a new knot in V direction at the position specified by `v`, `r` times. `v` must be in the valid range of the corresponding knot vector of the selected patches. The valid range is determined by the current knot vector `V` as follows: $V[p] \leq v \leq V[n]$, where `p` is the degree (order-1) of the patch in V direction and `n` is the height of the patch. The `v` knot type of the patches will always be changed to "Custom" but the shape of the patches will not change.

See also section [5.5.12 Insert Knot Surface Tool \(page 325\)](#).

remknuNP – remove u knot from NURBS surface:

- Synopsis: `"remknuNP (u | -i ind) r [tol]"`
- Background: Yes, Undo: Yes, Safe: Yes
- Description: Remove a knot at the position specified by `u` (`u` must be in the valid range of the knot vector of the selected surface) `r` times from the surface. Since Ayam 1.20 the knot to remove may also be specified using its (zero based) index in the knot vector (i.e. use `"remknuNP -i 3 1"` instead of `"remknuNP 0.5 1"` for the knot vector `"0 0 0 0.5 1 1 1"`). Note that the shape of the surface may be changed by this tool unless the parameter `tol` is specified. If `tol` is specified, the new surface does not deviate from the original surface more than `tol` in any point. If the knot can not be removed `r` times due to the tolerance, an error is reported and the original surface is left unchanged.

This operation also fails, if the knot removal would lead to a surface of lower order.

See also section [5.5.13 Remove Knot Surface Tool \(page 326\)](#).

remknvNP – remove v knot from NURBS surface:

- Synopsis: `"remknvNP (v | -i ind) r [tol]"`
- Background: Yes, Undo: Yes, Safe: Yes
- Description: Remove a knot at the position specified by `v` (`v` must be in the valid range of the knot vector of the selected surface) `r` times from the surface. Since Ayam 1.20 the knot to remove may also be specified using its (zero based) index in the knot vector (i.e. use `"remknvNP -i 3 1"` instead of `"remknvNP 0.5 1"` for the knot vector `"0 0 0 0.5 1 1 1"`). Note that the shape of the surface may be changed by this tool unless the parameter `tol` is specified. If `tol` is specified, the new surface does not deviate from the original surface more than `tol` in any point. If the knot can not be removed `r` times due to the tolerance, an error is reported and the original surface is left unchanged.

This operation also fails, if the knot removal would lead to a surface of lower order.

See also section [5.5.13 Remove Knot Surface Tool \(page 326\)](#).

remsknuNP – remove superfluous knots from NURBS surface:

- Synopsis: `"remsknuNP [tol]"`
- Background: Yes, Undo: Yes, Safe: Yes
- Description: Remove all knots from the NURBS surface that do not contribute to its shape (in `u` direction). This may lead to control point removal and changes, but the shape of the surface does not change unless a large tolerance value is specified via the `tol` parameter. The default value of this parameter is 0.0.

It is no error if no knots can be removed.

See also section [5.3.15 Remove Superfluous Knots Surface Tools \(page 293\)](#).

remsuknvNP – remove superfluous knots from NURBS surface:

- Synopsis: "remsuknvNP [tol]"
- Background: Yes, Undo: Yes, Safe: Yes
- Description: Remove all knots from the NURBS surface that do not contribute to its shape (in v direction). This may lead to control point removal and changes, but the shape of the surface does not change unless a large tolerance value is specified via the `tol` parameter. The default value of this parameter is 0.0.

It is no error if no knots can be removed.

See also section [5.5.14 Remove Superfluous Knots Surface Tools \(page 327\)](#).

refineuNP – refine NURBS surface in U direction:

- Synopsis: "refineuNP [{u1 u2 un}]"
- Background: Yes, Undo: Yes, Safe: Yes
- Description: Refine the selected NURBS surfaces in u direction with n new knots {u1 u2 un} without changing their shape. If no list of new knots is given, a new knot is inserted into each interval in the old knot vector. The u knot type of the refined surfaces may be changed to "Custom".

See also section [5.5.6 Refine Knots Surface Tool \(page 319\)](#).

refinevNP – refine NURBS surface in V direction:

- Synopsis: "refinevNP [{v1 v2 vn}]"
- Background: Yes, Undo: Yes, Safe: Yes
- Description: Refine the selected NURBS surfaces in v direction with n new knots {v1 v2 vn} without changing their shape. If no list of new knots is given, a new knot is inserted into each interval in the old knot vector. The v knot type of the refined surfaces may be changed to "Custom".

See also section [5.5.6 Refine Knots Surface Tool \(page 319\)](#).

elevateuNP – elevate NURBS surface in U direction:

- Synopsis: "elevateuNP [n]"
- Background: Yes, Undo: Yes, Safe: Yes
- Description: Elevate the u order of the selected NURBS surfaces without changing the shape of the surfaces by n. If the parameter n is omitted, a default value of 1 is used. The u knot type of the elevated surfaces will be changed to "Custom".

See also section [5.5.8 Elevate Surface Tool \(page 321\)](#).

elevatevNP – elevate NURBS surface in V direction:

- Synopsis: "elevatevNP [n]"
- Background: Yes, Undo: Yes, Safe: Yes
- Description: Elevate the v order of the selected NURBS surfaces without changing the shape of the surfaces by n. If the parameter n is omitted, a default value of 1 is used. The v knot type of the elevated surfaces will be changed to "Custom".

See also section [5.5.8 Elevate Surface Tool \(page 321\)](#).

reduceuNP – decrease order of NURBS surface:

- Synopsis: "reduceuNP [tol]"
- Background: Yes, Undo: Yes, Safe: Yes
- Description: Decrease the order of the selected NURBS surfaces in *u* direction by one if the shape of the reduced surfaces does not deviate from the original surfaces by *tol* in any point. If the parameter *tol* is omitted, a default value of 0.0 is used, i.e. the order is only reduced if the surface does not change. The knot type of the reduced surfaces will be changed to "Custom".

See also section 5.5.9 Reduce Surface Tools (page 322).

reducevNP – decrease order of NURBS surface:

- Synopsis: "reducevNP [tol]"
- Background: Yes, Undo: Yes, Safe: Yes
- Description: Decrease the order of the selected NURBS surfaces in *v* direction by one if the shape of the reduced surfaces does not deviate from the original surfaces by *tol* in any point. If the parameter *tol* is omitted, a default value of 0.0 is used, i.e. the order is only reduced if the surface does not change. The knot type of the reduced surfaces will be changed to "Custom".

See also section 5.5.9 Reduce Surface Tools (page 322).

splituNP – split NURBS patch:

- Synopsis: "splituNP [-a|-r] u"
- Background: Yes, Undo: No, Safe: Yes
- Description: Splits the selected NPatch objects into two patches at the parametric value *u*, creating one new NPatch object and *modifying the original* selected NPatch object. If the option "-r" is present, the parametric value is interpreted in a relative way and should therefore be in the range [0, 1].

If the option "-a" is present, the new NPatch object(s) will be appended to the current level. Otherwise the new NPatch object(s) will be inserted into the level right after the respective NPatch object(s) to split. This is the new default.¹

See also section 5.5.15 Split Surface Tools (page 328).

splitvNP – split NURBS patch:

- Synopsis: "splitvNP [-a|-r] v"
- Background: Yes, Undo: No, Safe: Yes
- Description: Splits the selected NPatch objects into two patches at the parametric value *v*, creating one new NPatch object and *modifying the original* selected NPatch object. If the option "-r" is present, the parametric value is interpreted in a relative way and should therefore be in the range [0, 1].

If the option "-a" is present, the new NPatch object(s) will be appended to the current level. Otherwise the new NPatch object(s) will be inserted into the level right after the respective NPatch object(s) to split. This is the new default.²

See also section 5.5.15 Split Surface Tools (page 328).

¹ Since 1.24. ² Since 1.24.

extrNP – extract NURBS patch:

- Synopsis: "extrNP [-relative] umin umax vmin vmax"
- Background: Yes, Undo: No, Safe: Yes
- Description: Extracts a sub-patch from the selected NPatch or NPatch providing objects. The extracted patch(es) will be appended as new object(s) to the current level of the scene. The sub-patch to be extracted is specified by the parametric values `umin`, `umax`, `vmin`, and `vmax` which have to be in the respective valid knot range.

If the optional argument "`-relative`" is specified, the parametric values are interpreted in a relative way and must consequently be in the range `[0, 1]`.

See also section [5.6.2 Extract Patch Tool \(page 334\)](#).

tweenNP – interpolate (tween) surfaces:

- Synopsis: "tweenNP [`r`]"
- Background: Yes, Undo: No, Safe: Yes
- Description: Interpolate (tween) between the first two selected NURBS patches, creating a new patch that incorporates features from the selected patches. The parameter `r` defines the ratio of influence of the first and the second patch (the latter using `1-r`). This parameter defaults to 0.5.

The two patches must be of the same width, height, `uorder`, and `vorder`. They need not be defined on the same knot vectors, however.

If a third surface is selected, the parameter `r` is ignored and this third surface defines the ratio of influence with its `y` coordinates.

See also section [5.4.14 Tween Surfaces Tool \(page 316\)](#).

interpuNP – interpolate NURBS surface in U direction:

- Synopsis: `"interpuNP [-order order | -ktype type | -closed (0|1) | -sdlen length | -edlen length]"`
- Background: Yes, Undo: Yes, Safe: Yes
- Description: Interpolate the selected NURBS surfaces in U direction with desired order and parameterisation type. Order defaults to 4 and must be higher than 2. The parameterisation type must be one of 0 – Chordal, 1 – Centripetal, or 2 – Uniform, default is Chordal. The interpolation can create a closed surface if requested by the `"-closed"` parameter but this will also increase the width of the resulting NURBS surface. The default is to create an open surface for open input surfaces and a closed surface for closed or periodic input surfaces.

Using the options `"-sdlen"` and `"-edlen"` (which both default to 0.0) the length of automatically created start/end derivatives can be adjusted. If any of these are not 0.0, a different interpolation algorithm will be used, which increases the width of the resulting NURBS surface.

The surface will interpolate all current control points after the interpolation and the position of certain control points will be changed in this process so that, after interpolation, the new control points will *not* be interpolated by the surface. The surface will rather interpolate the *old* control point positions.

The u knot type of the interpolated surfaces will be changed to `"Custom"`.

If the `"-closed"` option is not present, the interpolated surface will be closed for closed and periodic surfaces.

See also section [5.5.16 Interpolate Surface Tool \(page 329\)](#).

interpvnP – interpolate NURBS surface in V direction:

- Synopsis: `"interpvnP [-order order | -ktype type | -closed (0|1) | -sdlen length | -edlen length]"`
- Background: Yes, Undo: Yes, Safe: Yes
- Description: Interpolate the selected NURBS surfaces in V direction with desired order and parameterisation type. Order defaults to 4 and must be higher than 2. The parameterisation type must be one of 0 – Chordal, 1 – Centripetal, or 2 – Uniform, default is Chordal. The interpolation can create a closed surface if requested by the `"-closed"` parameter but this will also increase the height of the resulting NURBS surface. The default is to create an open surface for open input surfaces and a closed surface for closed or periodic input surfaces.

Using the options `"-sdlen"` and `"-edlen"` (which both default to 0.0) the length of automatically created start/end derivatives can be adjusted. If any of these are not 0.0, a different interpolation algorithm will be used, which increases the height of the resulting NURBS surface.

The surface will interpolate all current control points after the interpolation and the position of certain control points will be changed in this process so that, after interpolation, the new control points will *not* be interpolated by the surface. The surface will rather interpolate the *old* control point positions.

The v knot type of the interpolated surfaces will be changed to `"Custom"`.

See also section [5.5.16 Interpolate Surface Tool \(page 329\)](#).

approxNP – approximate NURBS surface:¹

- Synopsis: `"approxNP [-m mode | -aw awidth | -ah aheight | -ou uorder | -ov vorder | -kt(u|v) ptu|ptv | -c(u|v) 0|1 | -tp(u|v) tu|tv]"`

- Background: Yes, Undo: Yes, Safe: Yes

- Description: Approximate the control points or a tessellation of the selected NURBS surfaces in desired direction(s) with a new non-rational surface of desired dimensions (awidth, aheight), orders and parameterisation type.

The mode parameter determines whether to just approximate along U (0), V (1), U and then V (2), or V and then U (3). If just approximating in one direction, the approximation parameters of the other direction will be ignored and the values from the original surface will be used instead.

The order parameters default to 4 and must be higher than 2. They must also not be higher than the respective dimension of the approximating surface.

The awidth and aheight parameters must not be higher than the number of data points in the respective dimension to approximate.

The parameterisation type must be one of 0 – Chordal, 1 – Centripetal, or 2 – Uniform; default is Chordal. For one dimensional approximations, the knot type of the other dimension may not be uniform and will be overridden to be of type "BSpline" if a closed/periodic surface is created.

The approximation can create a closed surface if requested by the "-cu" / "-cv" parameter(s).

The "-tpu" / "-tpv" parameters allows to control the tessellation of the surface to be approximated. If these parameters are not present or zero, no tessellation will be carried out and the control points of the surface will be approximated. The tessellation parameters directly control the number of intermediate points to generate for every distinct knot interval in the originating surfaces knot vectors. The number of data points per dimension generated by the tessellation is $nd + (nd - 1) \times t$, where nd is the total number of distinct knots in the valid range of the respective knot vector and t the respective tessellation parameter.

The knot type of the approximated surfaces will be changed to "Custom".

See also section 5.5.17 Approximate Surface Tools (page 330).

- Examples:

1. `"crtOb NPatch; selOb -1; approxNP -aw 3 -ou 3"`

creates a standard NURBS surface and approximates the control points of it with a new surface of width 3 and order 3.

2. `"crtOb NPatch; selOb -1; approxNP -aw 5 -ou 3 -tpu 3"`

creates a standard NURBS surface and approximates a tessellation of it with a new surface of width 5 and order 3.

¹ Since 1.30.

breakNP – break NURBS patch into curves:

- Synopsis: `"breakNP [-r | -a | (-u | -v)]"`
- Background: Yes, Undo: No, Safe: Yes
- Description: Breaks the selected NPatch objects into NURBS curves, along parametric dimension U or V (depending on whether the option `"-u"` or `"-v"` is specified, default is U).

If the option `"-a"` is specified, the transformations of the NPatch objects will be applied to the control points and the NCurve objects will be created with default transformation attributes, otherwise the control points will be copied verbatim and the NCurve objects will get the transformation attributes of the respective NPatch.

If the option `"-r"` is specified, the new curve objects will replace each NPatch object instead of being appended to the current level. They will also be selected immediately.¹

See also section 5.6.3 Break into Curves Tool (page 335).

buildNP – build NURBS patch from curves:

- Synopsis: `"buildNP [-r|-a (0|1) | -o order | -t type | -k knottype]"`
- Background: Yes, Undo: No, Safe: Yes
- Description: Builds a NURBS patch from the selected NURBS curves. The width of the new patch depends on the number of provided curves and the surface type.

The option `"-a"` controls, whether the transformation attributes of the NURBS curves shall be applied to the respective control points before building the patch (default 1 – yes).

The option `"-o"` determines the desired order of the surface in U direction (default $\min(4, width)$).

The option `"-t"` allows to set a surface type (0 – open, 1 – closed, 3 – periodic; default 0).

The option `"-k"` allows to set a knot type (0 – Bezier, 1 – B-Spline, 2 – NURB, 4 – Chordal, 5 – Centripetal, default 2 – NURB). Custom knots are not supported.

If the option `"-r"` is present, the new NPatch objects will replace the first selected NCurve object instead of being appended to the current level, the other NCurve objects will be removed.

See also section 5.6.4 Build from Curves Tool (page 336).

¹ Since 1.27.

isCompNP:

- Synopsis: "isCompNP [(-u|-v) | -l level]"
- Background: Yes, Undo: No, Safe: Yes
- Description: This command returns 1 if the selected NURBS surfaces are compatible (i.e. defined on the same knot vector), otherwise it returns 0.

If the option "-u" is given, only the U dimension will be checked.

If the option "-v" is given, only the V dimension will be checked.

If "level" is 0, only the orders of the surfaces are compared.

If "level" is 1, width/height and orders of the surfaces are compared.

makeCompNP – make NURBS surfaces compatible

- Synopsis: "makeCompNP [-f | (-u|-v) | -l level]"
- Background: Yes, Undo: Yes, Safe: Yes
- Description: makes the selected NURBS surfaces compatible i.e. of the same order and defined on the same knot vectors.

If the option "-f" is present, there will be no prior compatibility check.

If the option "-u" is given, only the U dimension will be adapted.

If the option "-v" is given, only the V dimension will be adapted.

If "level" is 0, only the orders will be adapted.

If "level" is 1, only the orders and lengths will be adapted.

See also section 5.5.19 Make Compatible Tool (page 332).

curvatNP – compute Gaussian curvature:

- Synopsis: "curvatNP [-r] -u u -v v"
- Background: Yes, Undo: No, Safe: Yes
- Description: Compute and return the Gaussian curvature of the selected NURBS surface or NURBS surface providing object at the designated parametric values u and v.
If the option "-r" is present, the parametric values are interpreted in a relative way and should therefore be in the range [0, 1].
If multiple objects are selected, a list of curvature values is returned.

fairNP – improve surface shape:

- Synopsis: "fairNP [-m mode | -w | -t tol]"
- Background: Yes, Undo: Yes, Safe: Yes
- Description: Improve the surface shape by moving control points so that the curvature is distributed more evenly.
The value of the option "-m" (mode) controls how the surface shall be faired: 0 – U direction only, 1 – along V direction only, 2 – first along U direction, then along V direction, and 3 – first along V direction, then along U direction. The mode defaults to 0 – U direction only.
If the option "-w" (worst) is present, only the point that would move the farthest distance in a row/column will be changed.
If a tolerance value is present the processed control points do not move more than the given value.
If points are selected, only these will be processed.

tobasisPM – convert PatchMesh to a different basis:

- Synopsis: `"tobasisPM [-t type | -s step | -b basis]"`
- Background: Yes, Undo: Yes, Safe: Yes
- Description: Converts all selected bicubic PatchMesh objects to a different basis. Both dimensions will be converted to the same basis; it is not an error if one dimension is already of the target basis type.

The option `"-t"` controls the new basis type (0 – "Bezier", 1 – "B-Spline", 2 – "Catmull-Rom", 3 – "Hermite", 4 – "Power", 5 – "Custom"), default is 1 (conversion to "B-Spline").

The option `"-s"` determines the new step size (1 to 4), it defaults to the natural step size of the target basis type and thus can be omitted safely, unless the target type is "Custom", in which case the step size *must* be specified.

The option `"-b"` allows to convert to a custom basis and thus is a list of 16 floating point values specifying a 4 by 4 basis matrix in column major order. If `"-b"` is given, the target type defaults to "Custom" and the option `"-t"` can be omitted.

- Example:
`"tobasisPM -t 0"`
 converts the selected bicubic patch meshes to the Bezier basis.

tobasisBC – convert BCurve objects to a different basis:

- Synopsis: `"tobasisBC [-t type | -s step | -b basis]"`
- Background: Yes, Undo: Yes, Safe: Yes
- Description: Converts all selected BCurve objects to a different basis.

The option `"-t"` controls the new basis type (0 – "Bezier", 1 – "B-Spline", 2 – "Catmull-Rom", 3 – "Hermite", 4 – "Power", 5 – "Custom"), default is 1 (conversion to "B-Spline").

The option `"-s"` determines the new step size (1 to 4), it defaults to the natural step size of the target basis type and thus can be omitted safely, unless the target type is "Custom", in which case the step size *must* be specified.

The option `"-b"` allows to convert to a custom basis and thus is a list of 16 floating point values specifying a 4 by 4 basis matrix in column major order. If `"-b"` is given, the target type defaults to "Custom" and the option `"-t"` can be omitted.

- Example:
`"tobasisBC -t 0"`
 converts the selected basis curves to the Bezier basis.

6.2.17 Manipulating PolyMesh Objects

These are more specialized commands to change PolyMesh properties:

genfnPo – generate face normals:

- Synopsis: "genfnPo"
- Background: Yes, Undo: Yes, Safe: Yes
- Description: Generate face normals for the selected PolyMesh object(s) using the robust Newell algorithm.

The generated normals will be stored in a PV tag.

gensnPo – generate smooth normals:

- Synopsis: "gensnPo"
- Background: Yes, Undo: Yes, Safe: Yes
- Description: Generate smooth vertex normals for the selected PolyMesh object(s), averaging the surrounding face normals of each vertex. The face normals will be weighted by the vertex face-centroid distance, which takes both, face area and face shape, into account. Vertices of hole loops will just get the respective face normal.

Already existing vertex normals will be destroyed.

If face normals already exist, they will be used, otherwise, new face normals will be generated using the same algorithm as implemented in the "genfnPo" command above.

remsnPo – remove smooth normals:

- Synopsis: "remsnPo"
- Background: Yes, Undo: Yes, Safe: Yes
- Description: Removes all smooth/vertex normals from the selected PolyMesh object(s).

flipPo – flip normals or loops:

- Synopsis: "flipPo [0|1|2]"
- Background: Yes, Undo: Yes, Safe: Yes
- Description: Flips smooth/vertex normals of the selected PolyMesh object(s). By default, and if the parameter is 0, also all loops will be flipped. If the parameter is 1, only the normals are reversed. If the parameter is 2, only the loops are reversed.

6.2.18 Manipulating Points and Normals

Use these commands to read or manipulate the control points or normals of objects that support point editing.

getPnt – get point(s):

- **Synopsis:** `"getPnt [-trafo | -world | -eval | -relative] (index | indexu indexv | u | u v) [(varx vary varz [varw] | -vn varname | -all [varname] | -sel [varname])]"`

- **Background:** Yes, **Undo:** No, **Safe:** Yes

- **Description:** Get a control point of the currently selected object and write the coordinate values into the variables `varx`, `vary`, `varz`, and (if the object supports rational coordinates) `varw`.

The index arguments needed depend on the type of the selected object, e.g. reading the points of a NURBS curve requires just one index parameter (`index`), whereas reading the points of a NURBS patch requires two index parameters (`indexu` and `indexv`) to be specified.

If the optional argument `"-trafo"` is given, the coordinates will additionally be transformed by the values given in the objects Transformation property.

If the optional argument `"-world"` is used, the coordinates will additionally be transformed to world space.

If the optional argument `"-eval"` is specified, the `"indexu"` and `"indexv"` values are interpreted as parametric values of a NURBS curve or surface and the corresponding point on the curve or surface is delivered in `varx`, `vary`, and `varz`.

If the optional argument `"-relative"` is specified, the parametric value for NURBS curve or surface evaluation is interpreted in a relative way and must consequently be in the range $[0, 1]$.

If the alternative argument `"-vn"` is given, the coordinate values will be appended to the list variable specified by `"varname"`.

If the alternative argument `"-all"` is used, all coordinate values of the selected objects will be appended to the list variable specified by `"varname"`.

If the alternative argument `"-sel"` is used, the coordinate values of the currently selected points of the selected objects will be appended to the list variable specified by `"varname"`.¹

If any of the variable name arguments are omitted, the command returns the respective results.²

- **Notes:** In Ayam versions prior to 1.20, only global variables were written, this is no longer the case.

- **Examples:**

1. `"getPnt 1 x y z w"`

gets the coordinate values of the second point of the selected NURBS curve and writes the values to the variables `"x"`, `"y"`, `"z"`, and `"w"`.

2. `"getPnt -eval 0.5 x y z"`

gets the curve point at parametric value `"0.5"` and writes the coordinate values to the variables `"x"`, `"y"`, and `"z"`.

¹ Since 1.28. ² Since 1.27.

setPnt – set point(s):

- **Synopsis:** `"setPnt [-world] (index | indexu indexv) (x y z [w] | -vn varname) | (-all|-sel) varname"`
- **Background:** Yes, **Undo:** Yes, **Safe:** Yes
- **Description:** Set a control point of the currently selected object to the coordinates x, y, z, and w or to coordinates from a list, or set all control points from a list of coordinate values.

The index arguments needed depend on the type of the selected object, e.g. manipulating the points of a NURBS curve requires just one index parameter (index), whereas manipulating the points of a NURBS patch requires two index parameters (indexu and indexv) to be specified.

If the optional parameter `"-world"` is given, the coordinate values are expressed in world space and will be transformed to appropriate object space coordinates before setting.

If the optional parameter `"w"` is omitted, but the selected object has rational points, a default value of 1.0 will be used for the weight.

If the alternative parameter `"-vn"` is used, the coordinate values will be read from the variable specified by `"varname"` which must be a list of double values.

If the alternative parameter `"-all"` is provided, all control points of the selected objects will be set and the coordinate values will be read from the variable specified by `"varname"` which must be a list of double values.

If the alternative argument `"-sel"` is used, all selected control points of the selected objects will be set and the coordinate values will be read from the variable specified by `"varname"` which must be a list of double values.¹

When reading data from list variables, no precision will be lost as there are no double-string-double conversions involved.

- **Examples:**
 1. **"setPnt 1 0.0 0.2 0.3 1.0"**
sets the coordinate values of the second point of the selected NURBS curve object to "0.0 0.2 0.3 1.0".
 2. **"setPnt -world 0 0 0 0"**
sets the first point of the selected NURBS curve object to the world origin, regardless of the transformation attributes of the curve object (or any of its potential parent objects).
 3. **"setPnt 2 1 0.0 0.2 0.3"**
sets the coordinate values of the second point in the third column of the control mesh of the selected NURBS patch object to "0.0 0.2 0.3 1.0".

¹ Since 1.28.

getNormal – get normal(s):

- Synopsis: `"getNormal [-cv | -relative | -trafo | -world | -norm] (index | u v) [(varx vary varz | -vn varname)]"`
- Background: Yes, Undo: Yes, Safe: Yes
- Description: Get a normal of the currently selected object and write the normal vector component values into the variables `varx`, `vary`, and `varz`.¹

This operation fails for PolyMesh objects, if there are no vertex normals.

The index arguments needed depend on the type of the selected object, e.g. reading the normal of a PolyMesh requires just one index parameter (`index`), whereas evaluating the normal of a NURBS patch requires two parameters (`u` and `v`) to be specified.

If the optional argument `"-cv"` is specified, the `"u"` and `"v"` values are interpreted as index parameters and the surface is not evaluated but the normal is computed from the surrounding control points.

If the optional argument `"-relative"` is specified, the parametric values for NURBS surface evaluation are interpreted in a relative way and must consequently be in the range $[0, 1]$.

If the alternative argument `"-vn"` is given, the coordinate values will be appended to the list variable specified by `"varname"`.

If the optional argument `"-trafo"` is specified, the returned normal will, additionally, be transformed by the object orientation attributes.

If the optional argument `"-world"` is specified, the returned normal will, additionally, be transformed by the object and all parent level orientation attributes.

If the optional argument `"-norm"` is specified, the returned vector will be scaled to a length of 1.

- Examples:
 1. `"getNormal 1 x y z"`
gets the normal values of the second point of the selected PolyMesh and writes the values to the variables `"x"`, `"y"`, `"z"`.
 2. `"getNormal 0.5 0.5"`
evaluate the normal of the selected NURBS surface at parametric values `"0.5"` `"0.5"` and returns the result as list.

¹ Since 1.30.

setNormal – set normal:

- Synopsis: `"setNormal index (x y z | -vn varname) "`
- Background: Yes, Undo: Yes, Safe: Yes
- Description: Set a vertex normal of the selected PolyMesh object(s) to the values delivered by the x, y, and z arguments, or to values from a list.¹

This operation fails if there are currently no vertex normals.

If the alternative parameter `"-vn"` is used, the normal values will be read from the variable specified by `"varname"` which must be a list of double values.

When reading data from list variables, no precision will be lost as there are no double-string-double conversions involved.

- Examples:

1. `"setNormal 1 0.0 0.2 0.3"`

sets the normal values of the second control point of the selected PolyMesh object to `"0.0 0.2 0.3"`.

2. `"setNormal 0 -vn normal"`

sets the normal values of the first control point of the selected PolyMesh object to the values from the list variable `"normal"`.

¹ Since 1.30.

getTangent – get first derivative(s)/tangent(s):¹

- Synopsis: `"getTangent [-relative | -trafo | -world | -norm] (u | u v) (varx vary varz [varu varv varw] | -vn varname) "`

- Background: Yes, Undo: Yes, Safe: Yes

- Description: Compute the first derivative/tangent of the currently selected parametric curve object(s) and write the tangent vector component values into the variables `varx`, `vary`, and `varz`; for parametric surface objects both tangents (in `u` and `v` direction) will be computed and, consequently, also `varu`, `varv`, and `varw` will be written to. If the optional argument `"-relative"` is specified, the parametric values are interpreted in a relative way and must consequently be in the range `[0, 1]`.

If the alternative argument `"-vn"` is given, the resulting values will be appended to the list variable specified by `"varname"`.

If the optional argument `"-trafo"` is specified, the returned tangent will, additionally, be transformed by the object orientation attributes.

If the optional argument `"-world"` is specified, the returned tangent will, additionally, be transformed by the object and all parent level orientation attributes.

If the optional argument `"-norm"` is specified, the returned vector will be scaled to a length of 1.

- Examples:

1. `"getTangent 0.5 x y z"`

compute the first derivative/tangent values of the selected NURBS curve and writes the values to the variables `"x"`, `"y"`, `"z"`.

2. `"getTangent 0.5 0.5"`

evaluate the first derivatives of the selected NURBS surface at parametric values `"0.5"` `"0.5"` and returns the result as list.

¹ Since 1.30.

6.2.19 Vector Algebra

These procedures implement simple vector algebra for vectors stored as lists.

l3add – add two three element vectors:

- Synopsis: "l3add l1 l2"
- Background: No, Undo: No, Safe: Yes, Type: Procedure
- Description: returns the element-wise sum of two three element lists (l1 and l2) as another three element list.

l3sub – subtract two three element vectors:

- Synopsis: "l3sub l1 l2"
- Background: No, Undo: No, Safe: Yes, Type: Procedure
- Description: returns the element-wise subtraction of two three element lists (l1 and l2) as another three element list.

l3mul – multiply two three element vectors:

- Synopsis: "l3mul l1 l2"
- Background: No, Undo: No, Safe: Yes, Type: Procedure
- Description: returns the element-wise multiplication of two three element lists (l1 and l2) as another three element list.

l3scal – multiply a three element vector with a scalar:

- Synopsis: "l3scal l c"
- Background: No, Undo: No, Safe: Yes, Type: Procedure
- Description: returns the element-wise multiplication a three element list (l) with a scalar (c).

l3dot – compute the dot product of two three element vectors:

- Synopsis: "l3dot l1 l2"
- Background: No, Undo: No, Safe: Yes, Type: Procedure
- Description: returns the dot product of two three element lists (l1 and l2).

l3cross – compute the cross product of two three element vectors:

- Synopsis: "l3cross l1 l2"
- Background: No, Undo: No, Safe: Yes, Type: Procedure
- Description: returns the cross product of two three element lists (l1 and l2) as three element list.

l3to4 – convert 3D coordinates to rational style:

- Synopsis: "l3to4 l3 (w) "
- Background: No, Undo: No, Safe: Yes, Type: Procedure
- Description: returns the list of 3D points as list of 4D points (with a rational weight component, specified as w). The weight defaults to 1.0.
- Example:

```
crtOb ICurve; selOb -1;  
crtOb NCurve -length [getProperty Length] -cv [l3to4 [getPnt -all]]
```

Creates an interpolating curve, fetches its three dimensional data points, converts them to rational style (with default weight value 1.0) and creates a NURBS curve from these coordinates.

l4to3 – convert rational coordinates to 3D:

- Synopsis: "l4to3 l4"
- Background: No, Undo: No, Safe: Yes, Type: Procedure
- Description: returns the list of rational 3D points without the weight components.
- Example:

```
crtNCircle; selOb -1;  
crtOb ICurve -length [getProperty Length] -cv [l4to3 [getPnt -all]]
```

Creates a NURBS circle, fetches its rational coordinates, and creates an interpolating curve from the 3D positions of these coordinates.

6.2.20 Updating the GUI

These procedures update various parts of the Ayam user interface:

rV – redraw all views:

- Synopsis: "rV"
- Background: No, Undo: No, Safe: No, Type: Procedure
- Description: Redraws all currently open views, except for iconified views and views where automatic redraw has been turned off.

uS – update select:

- Synopsis: "uS [update_prop maintain_selection]"
- Background: No, Undo: No, Safe: No, Type: Procedure
- Description: Update the object listbox or tree view after a change to the object hierarchy. If update_prop is 0 no update of the property GUIs will take place.
If maintain_selection is 1 the old selection will be established again.
If both arguments are omitted update_prop defaults to 1 and maintain_selection to 0.
- Deficiencies: uS completely removes the object tree from the tree widget and rebuilds it, which can be a very time consuming operation (depending on the complexity of the scene). There are some options to speed this process up:
 - If there were just changes to the current level (and below), the global array entry "ay(ul)" (UpdateLevel) may be set to the current level before calling "uS". This will not remove and update the complete scene but just the part below "ay(ul)". Example:

```
global ay; set ay(ul) $ay(CurrentLevel); uS;
```

- If objects have been created and thus just need to be added to the current level of the object tree view, the command "uCR" may be used instead of "uS".
- If just names or types of objects of the current level changed, the command "uCL cl" may be used instead of "uS".

uCL – update current level:

- Synopsis: "uCL mode [args]"
- Background: No, Undo: No, Safe: No, Type: Procedure
- Description: Update only the current level of the object listbox or tree view after changes. See also the discussion of "uS" above. The parameter "mode" may be "cl" or "cs", where "cl" is the normal operation mode, and "cs" just clears the selection.

uCR – update current level after create:

- Synopsis: "uCR"
- Background: No, Undo: No, Safe: No
- Description: Update only the current level of the object listbox or tree view after objects have been created and need to be added to the current level. See also the discussion of "uS" above.

plb_update – property listbox update:

- Synopsis: "plb_update"
- Background: No, Undo: No, Safe: No, Type: Procedure
- Description: Clear the current property GUI, ask the currently selected object for a list of properties and insert them in the property listbox, then rebuild the property GUI of the property with the same index in the property listbox as the property selected before plb_update was started (this is not necessarily a property of the same type).

Since Ayam 1.13 it is also possible to automatically run GUI updating commands in the console by using `<Shift+Return>` instead of `<Return>`. The commands from the hidden preference setting "AUCOMMANDS" will be executed after the commands from the command line, if the `<Shift>` key is held down. `<Shift+Return>` may also be used without commands on the command line. By default, the "AUCOMMANDS" are "uS; rV; ", leading to updated object tree, property GUI, and views.

6.2.21 Managing Preferences

These commands manage preferences data:

getPrefs – get preferences data:

- Synopsis: "getPrefs"
- Background: No, Undo: No, Safe: No
- Description: Copy preferences data from the C to the Tcl context.

setPrefs – set preferences data:

- Synopsis: "setPrefs"
- Background: No, Undo: No, Safe: No
- Description: Copy preferences data from the Tcl to the C context. This is necessary after a change to the global Tcl array "ayprefs" to let all changes take effect.

See also section [2.10 Preferences \(page 58\)](#)

6.2.22 Custom Objects/Plugins

This command manages custom objects (plugins):

loadPlugin – load custom object/plugin:

- Synopsis: "loadPlugin name"
- Background: No, Undo: No, Safe: No, Type: Procedure
- Description: Load a custom object/plugin. If name is a complete filename and the designated file exists, it will be loaded directly. Otherwise, the file to load will be searched for in the list of configured plugin directories (see "Plugins" preference setting).

Note that it is currently not possible to unload a custom object or plugin from Ayam.

6.2.23 Applying Commands to a Number of Objects

These procedures and commands help to apply arbitrary commands to a number of selected objects.

forAll:

- Synopsis: `"forAll [(-recursive|-r) r | (-type|-t) t] command"`
- Background: depends, Undo: depends, Safe: No, Type: Procedure
- Description: The forAll procedure executes command for all objects that have been selected currently, or for every object of the current level if nothing has been selected.

If `r` is 1 (this is the default) then forAll will recurse into every object (if it has child objects) before the execution of command. If `r` is 2, the recursion will happen after the execution of command. If `r` is 0, only objects from the current level will be processed.

If the `"-type"` option is given, only objects of the specified type `t` will be processed.

Prior to Ayam 1.22, potential errors from the command(s) were suppressed and processing continued regardless. But this made interactive usage and debugging unnecessarily difficult. From 1.22 on, errors are reported to the user and processing immediately stops. Errors may still be suppressed using the `"catch"` command like this: `"forAll { catch { commands } }"`

As the command will potentially be called multiple times, result values of any kind can not be delivered using the `"return"` command but should rather be collected in global variables. In fact, returning any value that is not `-1` will be interpreted as error and the processing will stop immediately. Returning `-1` in a recursive forAll will stop the processing without raising an error.

The global variable `"ay(CurrentLevel)"` will be maintained while a recursive forAll browses the scene. In addition, the global variable `"i"` will be set to the index of the current object.

Note that forAll will run slowly if a property GUI is displayed. If the current property is de-selected first (using e.g. the property context menu), it will run much faster.

Furthermore, the current selection is correctly maintained.¹

- Deficiencies:
 - A recursive forAll will e.g. also descend into NURBS patches (if they have trim curves) and apply the command to the trim curves, which might not exactly be intended. In this case, the `"-type"` option can be used.
 - The command will not have access to global arrays unless e.g. one of the following constructs is in use:


```
"forAll { uplevel #0 { commands } }"
```

```
"forAll { global arrayname; commands }"
```
 - It is not possible to use commands that change the object hierarchy (e.g. deleting or inserting objects). The commands may just modify existing objects.

withOb – execute a command on certain selected object(s):

- Synopsis: `"withOb index [do] command"`
- Background: depends on command, Undo: depends on command, Safe: Yes
- Description: Use this command to execute command on a single object (designated by the zero based index) from a multiple selection without changing the selection state of any objects.
- Example:

1. `"withOb 2 {movOb 0 1 0}"`

moves the third object from multiple selected objects. All objects stay selected.

¹ Since 1.9.

6.2.24 Scene IO

These commands help to load scenes from and save them to Ayam scene files:

replaceScene:

- Synopsis: `"replaceScene filename"`
- Background: Yes, Undo: No, Safe: No
- Description: clears the current scene, then loads a new scene from the file designated by the `"filename"` parameter.
- Notes: In contrast to using the main menu, further house keeping tasks detailed in section [8.3.1 Opening Scene Files \(page 514\)](#) will *not* be executed. In particular, this command does *not* check or modify the scene changed state and does *not* set the most recently used list. Furthermore, automatic import is not working.

insertScene:

- Synopsis: `"insertScene filename"`
- Background: Yes, Undo: No, Safe: No
- Description: inserts a scene from the file designated by the `"filename"` parameter.
- Notes: In contrast to using the main menu, some house keeping tasks detailed in section [8.3.2 Inserting Scene Files \(page 515\)](#) will *not* be executed. In particular, this command does *not* modify the scene changed state.

saveScene:

- Synopsis: `"saveScene filename [selected]"`
- Background: Yes, Undo: No, Safe: No
- Description: saves the current scene to the file designated by the `"filename"` parameter. If the optional parameter `"selected"` is 1, only the selected objects will be saved.
- Notes: In contrast to using the main menu, some house keeping tasks detailed in section [8.3.3 Saving Scene Files \(page 515\)](#) will *not* be executed. In particular, this command does *not* modify the scene changed state and does *not* set the most recently used list. Furthermore, automatic export and saving of window geometries are not working.

newScene:

- Synopsis: `"newScene"`
- Background: Yes, Undo: No, Safe: No
- Description: clears the current scene.
- Notes: In contrast to using the main menu, this command does *not* check or modify the scene changed state.

6.2.25 RIB Export

This command allows to export the current scene to a RenderMan Interface Bytestream (RIB):

wrib – RIB export:

- Synopsis: `"wrib filename [-image imagename] [-smonly | -selonly | -objonly]"`
- Background: Yes, Undo: No, Safe: No
- Description: exports the current scene to a RIB file designated by "filename".

If the argument "-image" is given, the RIB file will create an image file named "imagename" upon rendering.

The export will use the camera transformation from the currently selected Camera object, unless one of the following options is specified.

If the argument "-smonly" is provided, a RIB to render shadow maps will be created and the argument of "-image" will be ignored. See also section [4.2.4 Using ShadowMaps \(page 121\)](#).

If the argument "-selonly" is used, only the selected (geometric) objects will be exported, which will result in a RIB file not suitable for rendering (no setup, camera transformation, or lights are in it) but for inclusion into other scenes via RiArchive. See also section [4.2.11 RiInc Object \(page 140\)](#).

Likewise "-objonly" leads to a RIB file containing all objects in the scene but not suitable for rendering.

The "wrib" command always needs a selected camera object (unless the "-selonly" or "-objonly" options are given); if there is none or if the camera transformations of the camera associated with a view window shall be used, the corresponding Togl callback for the view might be used like this instead:

```
.view1.f3D.togl wrib -file filename.rib
```

The Togl callback understands the same options as the "wrib" command.

- Notes: In Ayam versions prior to 1.15, the filename had to be prepended by a "-filename ", this is no longer the case.

6.2.26 Reporting Errors

This command is for error reporting from scripts:

ayError:

- Synopsis: `"ayError code place detail"`
- Background: No, Undo: No, Safe: No
- Description: This command reports errors or warnings. `ayError` should be preferred to `puts` because the error reporting mechanism of Ayam features consistently formatted output, compression of repeated messages, and logging. The code parameter should be one of: 1 – warning, 2 – error, 3 – flush messages, 4 – unspecified output. There are more codes defined (see `ayam.h`, look for Return/Error Codes) but they are generally not needed in the Tcl script context. The place parameter should describe the procedure where the error occurred. The detail parameter is the string to be output.
- Notes: The actual output in the Ayam console depends on the preference option "ErrorLevel" see section [2.10.5 Miscellaneous Preferences \(page 71\)](#).

6.2.27 Property GUI Management

These procedures help to manage property GUIs. See also section 6.1.5 Property Management and Data Arrays (page 347).

addPropertyGUI:

- Synopsis: `"addPropertyGUI name"`
- Background: No, Undo: No, Safe: Yes, Type: Procedure
- Description: This procedure sets up a property GUI management array and creates an enclosing frame for GUI elements whose window name will be returned.

The array will be set up in a way that the data array of the property will be named as the property with the string `Data` appended, i.e. for `MyProperty` it will be `MyPropertyData`.

The get/set procedure entries will be left empty.

After the creation of property GUI elements, `"NP"` tags must be used to make the new property visible to the user. See also section 4.11.13 NP (New Property) Tag (page 266).

A complete example is available in section 4.9.1 Script Object Examples (page 235).

- Example: `"set w [addPropertyGUI MyProperty]"`

The following procedures add user interface elements to the property GUIs created by `"addPropertyGUI"`.

addParam:

- Synopsis: `"addParam window arrayname paramname [defaults]"`
- Background: No, Undo: No, Safe: Yes, Type: Procedure
- Description: This procedure adds a property GUI element for a single integer or floating point number parameter.

The `"window"` parameter should contain the window name as returned by `"addPropertyGUI"` above.

The `"arrayname"` parameter is the name of the corresponding data array of the property.

The `"paramname"` parameter is the name of the parameter.

The `"defaults"` parameter is a list of default values. Those values will be presented to the user as an additional drop down menu on the right side of the interface element.

- Example: `"addParam $w MyPropertyData MyFloat {0.1 0.5 1.5}"`

addString:

- Synopsis: `"addString window arrayname paramname [defaults]"`
- Background: No, Undo: No, Safe: Yes, Type: Procedure
- Description: This procedure adds a property GUI element for a single string parameter.
The "window" parameter should contain the window name as returned by "addPropertyGUI" above.
The "arrayname" parameter is the name of the corresponding data array of the property.
The "paramname" parameter is the name of the parameter.
The "defaults" parameter is a list of default values. Those values will be presented to the user as an additional drop down menu on the right side of the interface element. If the list contains an entry "...", selecting this entry will clear the string entry field and move the input focus to the field.
- Example: `"addString $w MyPropertyData MyString {"a" "b" "abc"}"`

addCheck:

- Synopsis: `"addCheck window arrayname paramname [onoffvalues]"`
- Background: No, Undo: No, Safe: Yes, Type: Procedure
- Description: This procedure adds a property GUI element for a single boolean parameter realized by a check-button.
The "window" parameter should contain the window name as returned by "addPropertyGUI" above.
The "arrayname" parameter is the name of the corresponding data array of the property.
The "paramname" parameter is the name of the parameter.
The optional "onoffvalues" parameter is a list of two values that will be used when setting the corresponding variable when the check-button is enabled or disabled. The default values are 0 and 1.
- Example: `"addCheck $w MyPropertyData MyBool"`

addColor:

- Synopsis: `"addColor window arrayname paramname [defaults]"`
- Background: No, Undo: No, Safe: Yes, Type: Procedure
- Description: This procedure adds a property GUI element for a single color parameter.
The "window" parameter should contain the window name as returned by "addPropertyGUI" above.
The "arrayname" parameter is the name of the corresponding data array of the property.
The "paramname" parameter is the name of the parameter.
The "defaults" parameter is a list of default values. Those values will be presented to the user as an additional drop down menu on the right side of the interface element.
- Example: `"addColor $w MyPropertyData MyColor"`

addMatrix:

- Synopsis: `"addMatrix window arrayname paramname"`
- Background: No, Undo: No, Safe: Yes, Type: Procedure
- Description: This procedure adds a property GUI element for a four by four element matrix, e.g. a transformation matrix.
The "window" parameter should contain the window name as returned by "addPropertyGUI" above.
The "arrayname" parameter is the name of the corresponding data array of the property.
The "paramname" parameter is the name of the parameter. The individual matrix value variable names will be formed by appending a "_0" to "_15" to the "paramname".
- Example: **`"addMatrix $w MyPropertyData MyMatrix"`**

addMenu:

- Synopsis: `"addMenu window arrayname paramname choices"`
- Background: No, Undo: No, Safe: Yes, Type: Procedure
- Description: This procedure adds a property GUI element for a menu/multiple choice parameter, realized through a drop down menu. The value of the parameter will be the index of the chosen menu item.
The "window" parameter should contain the window name as returned by "addPropertyGUI" above.
The "arrayname" parameter is the name of the corresponding data array of the property.
The "paramname" parameter is the name of the parameter.
The "choices" parameter is a list of strings that will be presented in the menu.
In contrast to the other user interface element generating procedures, the corresponding entry in the property data array *must* exist before this procedure is called.
- Example: **`"addMenu $w MyPropertyData MyMenu {Choice1 Choice2}"`**

addFile:

- Synopsis: `"addFile window arrayname paramname [defaults]"`
- Background: No, Undo: No, Safe: No, Type: Procedure
- Description: This procedure adds a property GUI element for a file name.
The "window" parameter should contain the window name as returned by "addPropertyGUI" above.
The "arrayname" parameter is the name of the corresponding data array of the property.
The "paramname" parameter is the name of the parameter.
The "defaults" parameter is a list of default values. Those values will be presented to the user as an additional drop down menu on the right side of the interface element.
- Notes: There is a variation of this procedure for file names meant to be used for saving: `"addSFile"`
- Example:
`"addFile $w MyPropertyData MyFile {"/tmp/file1" "/tmp/file2"}"`

addCommand:

- Synopsis: `"addCommand window name text command"`
- Background: No, Undo: No, Safe: Yes, Type: Procedure
- Description: This procedure adds a property GUI element to let the user initiate a command with the help of a push-button.
The `"window"` parameter should contain the window name as returned by `"addPropertyGUI"` above.
The `"name"` parameter is the name of the corresponding button widget. The names must be unique in each property GUI.
The `"text"` parameter is the string to be put on the button.
The `"command"` parameter is the command to be executed when the button is pushed.
- Example: `"addCommand $w b1 PushMe {puts pushed}"`

addText:

- Synopsis: `"addText window name text"`
- Background: No, Undo: No, Safe: Yes, Type: Procedure
- Description: This procedure adds a property GUI element to display static information, e.g. a parameter section name.
The `"window"` parameter should contain the window name as returned by `"addPropertyGUI"` above.
The `"name"` parameter is the name of the corresponding label widget. The names must be unique in each property GUI.
The `"text"` parameter is the string to be displayed.
- Example: `"addText $w t1 "Angular Parameters:""`

addInfo:

- Synopsis: `"addInfo window arrayname paramname"`
- Background: No, Undo: No, Safe: Yes, Type: Procedure
- Description: This procedure adds a property GUI element to display dynamic textual information.
The `"window"` parameter should contain the window name as returned by `"addPropertyGUI"` above.
The `"arrayname"` parameter is the name of the corresponding data array of the property.
The `"paramname"` parameter is the name of the parameter whose value is to be displayed. Whenever the value of this variable changes, the corresponding label will be updated automatically.
If a second variable named like `"paramname"` but with an additional trailing `"Ball"` exists, its value will be displayed as balloon help text, when the mouse pointer hovers over the label. This way longer or more complex information can be presented.
- Example: `"addInfo $w MyPropertyData NumGeneratedElems"`

addProgress:

- Synopsis: `"addProgress window arrayname paramname"`
- Background: No, Undo: No, Safe: No, Type: Procedure
- Description: This procedure adds a property GUI element to display progress information.
The "window" parameter should contain the window name as returned by "addPropertyGUI" above.
The "arrayname" parameter is the name of the corresponding data array of the property.
The "paramname" parameter is the name of the parameter where the progress is stored in percent.
- Example: **`"addProgress $w MyPropertyData Progress"`**

addVSpace:

- Synopsis: `"addVSpace window name height"`
- Background: No, Undo: No, Safe: Yes, Type: Procedure
- Description: This procedure adds an empty property GUI element with defined height to improve the layout of a property GUI.
The "window" parameter should contain the window name as returned by "addPropertyGUI" above.
The "name" parameter is the name of the corresponding widget. The names must be unique in each property GUI.
The "height" parameter is the desired height in pixels.
- Example: **`"addVSpace $w v1 20"`**

addOptionToggle:

- Synopsis: `"addOptionToggle window arrayname paramname text cmd"`
- Background: No, Undo: No, Safe: Yes, Type: Procedure
- Description: This procedure adds a property GUI element that allows to toggle the visibility of other property GUI elements.
The "window" parameter should contain the window name as returned by "addPropertyGUI" above.
The "arrayname" parameter is the name of the property management array of the property.
The "paramname" parameter is the name of a variable that contains the current visibility state.
The "text" parameter is the text to display on the toggle widget, this is usually a string like "Advanced Options".
The "cmd" parameter is the name of a procedure that, based on the current visibility state, creates or destroys the other property GUI elements.

- Example:

```
proc toggleAdvanced { } {
    global MyProp
    set w $MyProp(w)
    if { $MyProp(ShowAdvanced) } {
        addCheck $w MyPropData AdvancedOption
    } else {
        catch {destroy $w.fAdvancedOption}
    }
}

set w [addPropGUI MyProp]
set MyProp(w) $w
addCheck $w MyPropData CommonOption
addOptionToggle $w MyProp ShowAdvanced "Advanced Options" \
    toggleAdvanced
```

addScriptProperty:

- Synopsis: `"addScriptProperty propName propargs"`
- Background: No, Undo: No, Safe: Yes, Type: Procedure
- Description: This procedure manages a property for a Script object script.

The `"propname"` parameter is the base name of the property.

The `"propargs"` parameter is a list of parameter specifications of the property.

This procedure

- creates the appropriate data arrays in safe and main interpreter;
- arranges for all parameters from the parameter specifications to be saved between multiple instances of the script object and to scene files (by adding a DA tag);
- creates and populates the property GUI, based on the default value types in the parameter specifications;
- copies the parameters from property GUI to designated variables for easy perusal by the script object code;
- adds a NP tag to the script object so that the (new) property GUI is visible to the user.

Even though the name of this procedure suggests otherwise, it can and should be invoked on every run of the script objects code (unlike code that manages data arrays and GUI manually).

Each parameter specification is a list of three values:

`"{ Name Defaultvalue ScriptContextVariableName }"`

where:

- `"Name"` is the string to be used in the property GUI as parameter name,
- `"Defaultvalue"` is the initial value to be used for this parameter, the type of the default value determines the type of property GUI element to be created:
 - * simple double and integer values lead to standard number parameter GUI elements (`addParam`),
 - * `true` or `false` lead to check boxes (`addCheck`), but the values in the script variable will still be 1 or 0 (and not `true` or `false`),
 - * menu components will be created, when the default value is a list (`addMenu`), each element of this list is a string to display in the menu, the zero based index in the list is the integer value to be put into the corresponding script variable, the first entry from this list is the default,
 - * other types of values lead to simple string entry fields (`addString`),
- `"ScriptContextVariableName"` is a short variable name where the value from the property GUI element will be stored for use in the script.

- Example:

```
addScriptProperty MyProp {  
  { FloatParam 0.1 fp }  
  { BoolParam false bp }  
  { MenuParam { "Model" "Mode2" } mp }  
  { StringParam "texture.tif" tp }  
}
```

leads to the following property GUI:

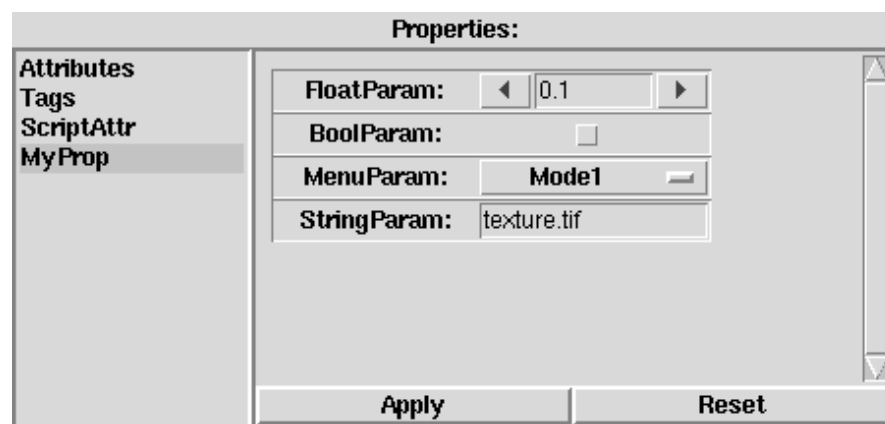


Figure 170: addScriptProperty Procedure Example

and the script code can then go on and use the parameter values from the variables `fp`, `bp`, `mp`, and `tp` respectively.

6.2.28 Miscellaneous

Miscellaneous commands:

convOb:

- Synopsis: "convOb [-inplace [type] | -check type]"
- Background: Yes, Undo: depends (if -inplace: Yes), Safe: Yes
- Description: This command calls the registered converter for the selected objects which usually creates new objects of a different type.

If the option "-inplace" is used, the new object(s) will replace the old object(s). If, additionally, a target type is specified, the conversion will be repeated until the object is of the requested type (and no conversion is attempted if the object is already of the requested type).¹

If the option "-check" is given, the convOb command does not convert but checks, whether a conversion to a given object type would be successful, the result of the check will be returned as 0 – no, or 1 – yes. Note, that only immediate, one step, conversions are checked, not multi step conversions as carried out by in-place conversions with target type.

¹ Since 1.25.

undo:

- Synopsis: "undo [redo | save opname [0|1] | clear | rewind]"
- Background: Yes, Undo: Yes, Safe: No
- Description:
 - If called without arguments, this command performs the undo operation.
 - If the argument is "redo", this command performs the redo operation.
 - If the argument is "save", the currently selected objects are saved to the undo buffer for future undo operations. The name of the now following modelling operation *has* to be provided in a second argument ("opname"). This name will be displayed in the default console prompt, to inform the user about which operation would be undone/redone, if undo/redo would be used (e.g. "[Undo:MoveObj/Redo:none].../bin>"). Since Ayam 1.13, a third argument may be given, that controls whether all the children of the selected objects should also be saved. This may be needed if the modelling action that follows the undo save is about to change the selected objects and also their children. Note: undo save does not fail if no objects are selected.
 - If the argument is "clear", all currently saved states will be cleared from the undo buffer.
 - The argument "rewind" is available since Ayam 1.14. With this command you can undo the last undo save operation. This may be necessary, if a modelling operation failed. Care should be taken, however, to *not* rewind the undo state, when a modelling operation only failed for some (not for all) of the selected objects.
- Example:

```
undo save "MovOb"  
set ay_error ""  
movOb 0 1 0  
if { $ay_error > 1 } {  
    undo rewind  
}
```

- Notes: See also section 8.1 The Undo System (page 511).

runTool:

- Synopsis: `"runTool vars labels commands title [doc]"`
- Background: Yes, Undo: No, Safe: No, Type: Procedure
- Description: This procedure opens a dialog window to request a set of parameter values, specified via `vars` and `labels`, and if the dialog is closed via the "Ok"-button, runs the specified commands substituting the values provided by the user into the script.

"vars" is a list of variable names, array names are allowed. If the specified variables already exist, their current values will be used to infer a parameter type for presenting a type specific GUI component to the user. The type detection uses the `string is` facility of Tcl. If the current value is of type integer and there is a second variable (named like the original variable with the appendix `_1`), a menu component will be used and the menu choices will be taken from the second variable.

"labels" is a list of label strings for the parameters.

"commands" is the code that calls the tool and runs additional GUI updating. If the code contains strings like `%0`, those will be replaced by the corresponding parameter value requested before the code runs.

"title" is the title string of the dialog window. It can be used to convey the tool name or description to the user.

"doc" is an optional argument that contains a documentation URL for the tool. This URL will be opened when the <F1>-key is pressed in the dialog.

- Examples:

1.

```
runTool {x} {"X:"} {moveOb %0 0 0;plb_update;rV} "MoveX"
```

opens a dialog to request a single parameter and moves the selected object(s) along the X-axis by the specified amount.

2.

```
runTool {x y} {"X:" "Y:"} {moveOb %0 %1 0;plb_update;rV} "MoveXY"
```

opens a dialog to request two parameters and moves the selected object(s) along the X-axis and the Y-axis by the specified amounts.

3.

```
set l 0
set l_1 {"Start" "End" "Both"}
runTool {l} {"Side:"} {
  switch %0 {
    0 {clampNC -s}
    1 {clampNC -e}
    2 {clampNC}
  }
  plb_update;rV
} "Clamp"
```

opens a dialog with a menu GUI component and clamps the selected NURBS curve(s) on the chosen side(s).

notifyOb:

- Synopsis: `"notifyOb [-all | -modified | -parent]"`
- Background: Yes, Undo: No, Safe: Yes*
- Description: This command executes the notification of the selected object(s) and their parents, or, if no object is selected, for all objects of the scene. This includes running BNS or ANS tag scripts (if present) and executing the notification of referenced objects. If the `"-modified"` parameter is used, only modified objects will be notified.

If the `"-all"` parameter is used, all objects will be notified regardless of the selection.

If the `"-parent"` parameter is used, only the current parent object of the current level will be notified.

Prior to Ayam 1.20 this command was named `"forceNot"`, the old name is still available for compatibility but its use is deprecated.

*: Since 1.21 this command is also available in the safe interpreter with limited functionality: only the notification callbacks of the selected objects will be executed. No BNS or ANS tags will be considered, notification of parents will not be done, and the complete notification will also be omitted. Hence, in the safe interpreter, this command ignores all parameters.

See also section [8.2 The Modelling Concept Tool-Objects \(page 511\)](#).

nameOb:

- Synopsis: `"nameOb name"`
- Background: Yes, Undo: Yes, Safe: Yes
- Description: This command sets the name of the selected object(s).
See also section [6.2.3 getName scripting interface command \(page 371\)](#).

setMark:

- Synopsis: "`setMark [x y z | 1]`"
- Background: Yes, Undo: No, Safe: No, Type: Procedure
- Description: This procedure sets the global mark to the specified coordinates.
If there is only one argument, it is expected to be a list of three floating point values.
If no arguments are provided, the global mark is cleared.
This procedure only works, if there is at least one view.
See also section [3.5 Setting the Mark \(page 82\)](#).
- Example: "**setMark 0 0 0**"

getMark:

- Synopsis: "`getMark`"
- Background: Yes, Undo: No, Safe: No, Type: Procedure
- Description: This procedure returns the current global mark as a list of three floating point values.
This procedure only returns meaningful data, if there is at least one view.
See also section [3.5 Setting the Mark \(page 82\)](#).

clearMark:

- Synopsis: "`clearMark`"
- Background: Yes, Undo: No, Safe: No, Type: Procedure
- Description: This procedure clears the global mark.
This procedure only works, if there is at least one view.
See also section [3.5 Setting the Mark \(page 82\)](#).

tmpGet:

- Synopsis: `"tmpGet tmpdir varname [ext]"`
- Background: Yes, Undo: No, Safe: No
- Description: This command calculates a name for a temporary file in `tmpdir` and puts the complete name (with the optionally present extension appended) into the variable designated by `varname`.

whatis:

- Synopsis: `"whatis this"`
- Background: Yes, Undo: N/A, Safe: No, Type: Procedure
- Description: Print information about the named object `"this"` in the Tcl scripting context to the console. This procedure can be used to inquire about the existence of variables, widgets, or whether or not something is a command or a procedure.
- Example: `"whatis ."` leads to the output:

```
. is a command.  
. is a widget.
```

addToProc:

- Synopsis: `"addToProc procedure addition"`
- Background: Yes, Undo: No, Safe: No, Type: Procedure
- Description: This procedure adds the code from `addition` to the Tcl procedure `procedure`.
- Note: This procedure uses the introspection facilities of Tcl and works only correctly for procedures, that end with a single `"return;"` statement.

6.3 Expression Support in Dialog Entries

Various entries of dialogs for object creation and modelling tools support Tcl variables and expressions.

It is e.g. possible to enter

```
$::u
```

instead of a numeric knot value in the insert knot tool parameter dialog to infer the parametric value from the global variable `u` (that may have been set before using the find `u` modelling action) and insert a knot at the picked point.

It is also possible to enter complex mathematical expressions:

```
[expr sin(45)]
```

or call into own procedures (that have to return appropriately typed values):

```
[myproc]
```

where "myproc" is defined elsewhere (e.g. in a Tcl script file loaded via the "Scripts" preference setting) as follows:

```
proc myproc { } {  
    return [expr sin(45)];  
}
```

.

Repeated calling of the tool without opening the dialog (using the keyboard shortcut <Ctrl+T>), will execute the provided expression again. This means, a number of curves with increasing length can be created by entering into the Ayam console

```
set ::myvar 1
```

then entering for the length in the create NURBS curve dialog:

```
[incr ::myvar]
```

then pressing <Ctrl+T> multiple times.

6.4 Scripting Interface Examples

Here are some complete example scripts for the Ayam Tcl scripting interface.

All examples may be copied from the documentation and pasted directly into the console of Ayam.

6.4.1 Moving Objects

The following example script shows how to move a selected object to a specified position in space.

```
proc placeOb { x y z } {
    global transfPropData

    # copy Transformations-property data to
    # global array "transfPropData"
    getTrafo

    # set array values according to procedure parameters
    set transfPropData(Translate_X) $x
    set transfPropData(Translate_Y) $y
    set transfPropData(Translate_Z) $z

    # copy Transformations-property data from
    # global array "transfPropData" to selected object
    setTrafo
}
# placeOb
```

In order to move all selected objects to 1 1 1 you may enter the following into the console:

```
forAll -recursive 0 {placeOb 1 1 1}
```

But perhaps you would rather like a small GUI for that? No problem, the following snippet adds an entry to the custom menu that opens a small requester for the x-, y-, and z-values and calls the "placeOb" procedure (defined above) with them:

```
global ay
$ay(cm) add command -label "Place Object(s)" -command {
    runTool {x y z} {"X:" "Y:" "Z:"} {forAll -recursive 0 {placeOb %0 %1 %2}};
    plb_update; rV
} "Place Object(s)"
}
```

The trailing "plb_update; rV" command ensures that the GUI is updated properly and all views display the new position of the moved objects.

6.4.2 Moving NURBS points

The following example script snippet shows how to move control points of a NURBS curve.

```
# first, we create a new NURBS curve with 30 control points
set len 30
crtOb NCurve -length $len
# update selection
uS
# select last object (the newly created curve)
sL
# prepare moving
set i 0
set r 3.0
set angle 0
set angled [expr 3.14159265/2.0]
while { $i < $len } {

    set x [expr $r*cos($angle)]
    set y [expr $r*sin($angle)]
    set z [expr $i/3.0]

    # move control point to new position
    setPnt $i $x $y $z 1.0

    set angle [expr $angle + $angled]
    incr i
}
# redraw all views
rV
```

Now use this as path for a Sweep. For instance, using the next small script.

6.4.3 Easy Sweep

The following example script shows how to easily create a sweep from a selected path curve (avoiding the manual and lengthy creation and parameterisation of a suitable cross section).

```
proc easySweep { } {  
  # first, we create a sweep object  
  crtOb Sweep  
  
  # now, we need to move the selected curve (path) to  
  # the sweep and create a cross-section curve there too  
  # for that, we move the currently selected curve to the clipboard  
  cutOb  
  
  # enter the Sweep (the last object in the current level)  
  goDown -1  
  
  # now, we create a new curve (a closed B-Spline suitable as cross section)  
  crtClosedBS -s 8  
  
  # select the new object  
  selOb 0  
  
  # now, we rotate and scale the curve  
  rotOb 0 90 0  
  scalOb 0.25 0.25 1.0  
  
  # move trajectory back (we use "-move", because we  
  # really want to move (and not copy) the curve object  
  pasOb -move  
  
  # go up to where we came from  
  goUp  
  
  # finally, update the GUI...  
  uS  
  sL  
  
  # ...and redraw all views  
  rV  
}  
# easySweep
```

Run this procedure by selecting a NURBS curve object, then type into the console:

```
» easySweep
```

This command may be added to the main menu as well:

```
global ay
$ay(cm) add command -label "Easy Sweep" -command {
  easySweep
}
```

After running the above script there should now be a new menu entry "Custom/Easy Sweep" that calls the `easySweep` procedure.

6.4.4 Toolbox Buttons

Here is another example script that shows how buttons may be added to the toolbox. `myImage` should be an image created e.g. from a GIF file of the size 25 by 25 pixels.

```
global ay ayprefs

# create an image from a GIF file:
image create photo myImage -format gif -file /home/user/giffile

set b $ay(tbw).mybutton

# if the button does not already exist:
if { ![winfo exists $b] } {

    # create it:
    button $b -padx 0 -pady 0 -image myImage -command myCommand

    # tell Ayam about the new button:
    # you can use "linsert", to insert the button in a specific
    # place or just append to the end of the list using "lappend"
    lappend ay(toolbuttons) mybutton

    # display the button:
    toolbox_layout

    # from now on, the button will be under the
    # automatic toolbox layout management
}
```

This example shows that

1. toolbox buttons have to be created in a frame whose path and window name are stored in the global variable "`ay(tbw)`" (this is "`.tbw.f`" for multi-window GUI configurations or "`.fv.fTools.f`" for single-window GUI configurations),
2. Ayam manages a list of all toolbox buttons in the global variable "`ay(toolbuttons)`", the order in that list is the order in which the buttons appear in the toolbox,
3. automatic layout management is carried out by the procedure "`toolbox_layout`".

Adding buttons with just text is a little bit more involved, as the sizes of those buttons often do not fit well in the icon button scheme with its constant button size. However, the procedure "`toolbox_add`" can be of considerable help.¹

See also the script "`scripts/topoly.tcl`" for an example.

¹ Since 1.14.

The following example script adds two buttons to the bottom of the toolbox spanning the whole window (this works best with the standard toolbox layout of 4 by 12 buttons used in the multi-window GUI configuration):

```
global ay

# create a frame:
set f [frame $ay(tbw).fcollex]

# calculate the row number below the last row:
set row [expr [lindex [grid size $ay(tbw)] 1] + 1]

# now display the frame at calculated row, spanning the whole window:
grid $f -row $row -column 0 -columnspan [lindex [grid size $ay(tbw)] 0] \
    -sticky we
# create two buttons inside the frame:
button $f.b1 -width 5 -text "Coll." -command { collMP; rV; }
button $f.b2 -width 5 -text "Expl." -command { explMP; rV; }
pack $f.b1 $f.b2 -side left -fill x -expand yes
```

6.5 Distributed Helper Scripts

This sections contains the documentation of some helper scripts that are distributed with Ayam.

The helper scripts may be run via the context menu of the console, the Tcl `"source"` command in the console, or the `"Scripts"` preference setting of Ayam on each start (the latter except for the so called *external* scripts `"repairAyam.tcl"`, `"bgconvert.tcl"`, and `"aytest.tcl"`).

All other, *internal* scripts may be combined arbitrarily except for `"kdialog.tcl"`, `"zdialog.tcl"`, and `"intfd.tcl"`, which are mutually exclusive.

6.5.1 Repair Ayam

The *external* Tcl script `"repairAyam.tcl"` may be used to repair the application state of Ayam, should it be stuck e.g. in an endless loop of Tcl error messages.¹

On Unix systems `"repairAyam"` may be started from any shell simply by typing

```
» ./repairAyam.tcl
```

or

```
» wish repairAyam.tcl
```

on the command prompt; if the script detects that it is running on Unix and not in Ayam it will send itself to the Tcl interpreter Ayam is running in using the Tk send command. On Mac OS X Aqua (not X11!) AppleScript events will be used instead of the Tk send command. If this does not work as expected `"repairAyam.tcl"` may still be run via the Ayam console (as on Win32).

On Win32 `"repairAyam.tcl"` has to be started from the Ayam console using the command:

```
» source scripts/repairAyam.tcl
```

or via the consoles context menu: `"Console/Load File"`.

The script `"repairAyam.tcl"` should be considered a *last resort* to help saving the current state of modified objects.

The script will close all views, clean up the application state variables, reset the mouse cursor and the console prompt, and try to update important main window widgets.

Furthermore, the script will also clear the console and try to break potential endless loops running e.g. in the console or in Script objects.²

After running `"repairAyam.tcl"` the scene (or the most important objects currently worked on) should be immediately saved to a new scene file, *not* the file currently loaded, using `"File/Save As"` or `"Special/Save Selected"`) and Ayam should be restarted afterwards.

Simply saving the scene using `"File/Save"` or `<Ctrl+s>` should be avoided because views were possibly deleted.

¹ Since 1.8.2. ² Since 1.9.

6.5.2 Test Ayam

The *external* script "aytest.tcl" allows to test the Ayam implementation by creating objects with various combinations of parameters and executing standard operations and modelling actions on them.

6.5.3 Use Ayam as Command Line Converter

The *external* Tcl script "bgconvert.tcl" converts scene files from one 3D file format to another, with the help of Ayam which is running in the background.¹

In the most simple form, bgconvert may be used from a Unix command line (or shell script) like this:

```
»bgconvert.tcl infile.x3d outfile.dxf
```

The above command would load the X3D file "infile.x3d" into Ayam and export the scene as DXF file to "outfile.dxf".

For a successful conversion Ayam has to run and the plugins required for the import and export processes need to be available and properly configured (check the "Plugins" preference setting). The plugins necessary for the conversion will be loaded automatically.

Import and export options may also be given like this:

```
»bgconvert.tcl "infile.rib -p 1" outfile.dxf
```

In the example above the "-p 1" option switches on reading of partial RIB files.

Available options and their syntax may be inquired from the import and export plugin Tcl scripts (e.g. "plugins/rrib.tcl").

¹ Since 1.15.

6.5.4 Shader Parsing

The Tcl script "slxml.tcl" switches the shader parsing machinery of Ayam to recognize XML based meta information embedded in shading language comments.¹ This way, the shader database for the Material objects can be built up from shading language source files, instead of compiled shaders.

The XML tags will not be parsed by a real XML parser, so that well formedness is not an issue. However, the attributes must be ordered in a certain way. To specify a shader, use:

```
<shader type="stype" name="sname">
```

where `stype` is one of `surface`, `displacement`, `volume`, `light`, `imager`, or `transformation` and `sname` is the shader name that must also match the file name of the shader source file sans extensions.

To specify a shader parameter, use a line like:

```
<argument name="argname" type="argtype" value="argval">
```

where `argname` is the name of the parameter, `argtype` is one of `float`, `string`, `matrix`, `color`, `point`, `normal`, or `vector` and `argval` is the default value.

See also the following example shader source:

```
/* myshader.sl:
 * Author: Randolph Schultz
 * <shader type="surface" name="myshader">
 * <argument name="Ka" type="float" value="0.5">
 * <argument name="Kd" type="float" value="0.9">
 * <argument name="ic" type="color" value="0 1 0">
 */
surface myshader(float Ka = 0.5, Kd = 0.9; color ic = color (0, 1, 0));
{
    color mycolor;

    ...

    Ci = Cs*mycolor*(Ka*ambient()+Kd*diffuse(faceforward(normalize(N), I)));
}
```

The following restrictions/caveats (in contrast to the normal shader parsing) apply:

- there can only be one shader per shader source file,
- shader file name and shader name must match,
- shaders for which there is no source code available are not supported (unless the parameter information is known, in which case a shader source file with matching comment can be faked),
- there is no control over whether the meta data matches the actual shader signature,
- there is no control over whether the compiled shader, that is used for rendering, matches the shader source code.

¹ Since 1.25.

6.5.5 Convert Everything to Polygons

The script `"topoly.tcl"` recursively browses through the scene and converts everything to a polygonal representation.¹

After running the script, there is a new button in the toolbox named `"ToPolyMesh"`. Additionally, there is a corresponding entry in the `"Custom"` main menu. Pressing the button or using the menu entry immediately starts the conversion process.

Since the changes of the conversion can not be undone, the conversion will not run if the scene contains unsaved changes.

The conversion will use the current parameters from the preference settings `"SMethod"`, `"SParamU"`, and `"SParamV"`; `"TP"` tags (if present) will override these parameters. TP tags may be created easily using the tessellation tool, see also section 5.6.5 Tessellation Tool (page 337).

6.5.6 Convert Everything to NURBS patches

The script `"tonpatch.tcl"` recursively browses through the scene and converts everything to a NURBS patch representation effectively flattening the tool object hierarchy.²

After running the script, there is a new button in the toolbox named `"ToNPatch"`. Additionally, there is a corresponding entry in the `"Custom"` main menu. Pressing the button or using the menu entry immediately starts the conversion process.

Since the changes of the conversion can not be undone, the conversion will not run if the scene contains unsaved changes.

6.5.7 Restrict the Console

The script `"2lcons.tcl"` (for two line console) may be used to restrict the screen space occupied by the console.

Normally, the Ayam console is resized with the main window and occupies a varying amount of screen space. After running the script, the console will always resize to exactly two lines of text. Different values may be chosen easily by adapting the script.

6.5.8 Color the Focus Ring

The script `"colfocus.tcl"` (for **colored focus**) may be used to paint the focus ring in a more visible color.

After running the script, the focus ring will be painted in blue (instead of black): focused sub-windows (views, console, object tree) will be more easily recognizable. Other colors may be used by editing the script.

¹ Since 1.13. ² Since 1.14.

6.5.9 Decouple Hierarchy View Link

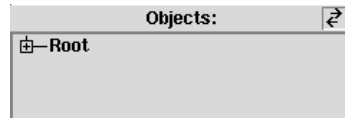


Figure 171: Link Button

The script `"link.tcl"` places a small button into the object hierarchy title area that controls the, otherwise permanent, link between the hierarchy and the views.¹ If this button is disabled, selection actions and changes to the hierarchy or to the objects themselves (via the property GUI) do not result in redraw operations.

Toggling the state to `"on"` will result in a single redraw.

This script may be used with all other scripts that place buttons in the designated area. The buttons will be placed from the right to the left in the order of the script execution.

6.5.10 Shuffle Objects Up/Down

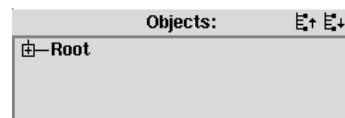


Figure 172: Shuffle Objects Up/Down Buttons

The scripts `"shufup.tcl"` and `"shufdown.tcl"` place small buttons into the object hierarchy title area that shuffle the selected objects up or down in the current level.² See also the image above.

See also the discussion about multiple hierarchy button scripts in section [6.5.9 Decouple Hierarchy View Link](#) (page 450).

6.5.11 Open/Close All Tree Levels

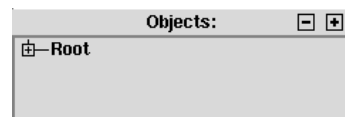


Figure 173: Open/Close All Tree Levels Buttons

The scripts `"opent.tcl"` and `"closet.tcl"` place small buttons into the object hierarchy title area that allow to open and close all tree levels.³ See also the image above.

See also the discussion about multiple hierarchy button scripts in section [6.5.9 Decouple Hierarchy View Link](#) (page 450).

¹ Since 1.30. ² Since 1.30. ³ Since 1.30.

6.5.12 Automatic About Center Actions

The script `"aac.tcl"` (for **automatic about center**) may be used to switch all modelling actions to their about variants with the mark set to the center of the current selection automatically.

After running the script, invoking e.g. the scale 2D action using the shortcut `<s>` will:

- if the modelling mode is object,
 1. set the mark to the center of all selected objects
 2. invoke the about variant of scale 2D

(this is the equivalent of `<sac>`)

- if the modelling mode is point,
 1. set the mark to the center of all selected points
 2. invoke the about variant of scale 2D

(this is the equivalent of `<saC>`)

The script modifies all rotate and scale actions (including their axis confined variants).

Note, that the mark is not reset to a new center, when the selection changes. After a selection change (e.g. by selecting points in a different view) simply restart the action to transform about the new center.

To rotate or scale about a different point than the center, the mark may still be set manually using `<a>`.

To temporarily disable the modified behavior, the global keyboard shortcut `<F11>` can be used.

6.5.13 Automatic Point Actions

Note that since the introduction of the `"ScopeManagement"` preference setting (see [2.10.2 Modelling Preferences \(page 62\)](#)) in Ayam 1.30 this script is obsolete and kept only for backwards compatibility. Furthermore, to ensure complete compatibility this script will set the scope management mode to `"Explicit"` when loaded.

The script `"apnt.tcl"` (for **automatic point**) may be used to switch the modelling mode to point modelling automatically after a point selection.

After running the script, selecting (tagging) a point using the select point action (shortcut `<t>`) will automatically switch the view to point modelling so that the next modelling actions (e.g. move, via shortcut `<m>`) will always transform the points and not modify the objects transformations. Note that currently the switch to point modelling will also occur, if no points are actually selected, it is just the mouse click that counts.

Selecting all points via the keyboard shortcut `<A>` will additionally switch to point modelling and de-selecting all points via `<N>` will additionally switch to object modelling.¹

It is also still possible to switch back to object modelling anytime via the keyboard shortcut `<o>`.

To temporarily disable the modified behavior, the global keyboard shortcut `<F12>` can be used.

¹ Since 1.21.

6.5.14 Save Selected Points

The script `"ssp.tcl"` (for **s**ave **s**electe**d** **p**oints) allows to save the point selection to tags of type `SP`.¹

After running the script, two new buttons appear in the toolbox that allow to save and restore the point selection respectively. See also the table below.



Operation	Icon
Save	
Restore	

Table 106: Save/Restore Selected Points

There are also two corresponding entries in the custom menu.

Note that the tags can be saved to scene files and also copied to different objects.

6.5.15 Revert Cursor Key Behavior

The script `"rc.tcl"` (for **r**evert **c**ursor) may be used to get more useful cursor key behavior in primary modelling views (parallel views).

After running the script, the keyboard shortcuts for rotating and panning in parallel views are swapped, e.g. just pressing `<Left>` key will then pan the view, instead of rotating it.

The shortcuts will be swapped again, when the view changes type to `"Perspective"`.

6.5.16 Access Core Functions from the Toolbox

The script `"zap.tcl"` demonstrates, how arbitrary core functionality that is just available through a main menu entry or the scripting interface might be accessed easily via the toolbox window.

After running the script `"zap.tcl"`, there will be a new toolbox button, labeled `"Zap!"`, that simply runs the `zap` command (which iconifies the complete application).

6.5.17 Switch Dialogs to Kdialog

The script `"kdialog.tcl"` switches all file dialogs of Ayam to use the `kdialog` application of the KDE project instead of the native Tk file dialog.

The script also adds a custom main menu entry to revert any changes.

6.5.18 Switch Dialogs to Zenity

The script `"zdialog.tcl"` switches all file dialogs of Ayam to use the `zenity` application of the Gnome project instead of the native Tk file dialog.

The script also adds a custom main menu entry to revert any changes.

¹ Since 1.21.

6.5.19 Switch File Dialogs to Tcl

The script `"intfd.tcl"` switches all file dialogs of Ayam to use the Tcl/Tk internal version instead of the native file dialogs provided by the operating system.

6.5.20 Use Aqsis from Application Directory

The script `"useaqsisapp.tcl"` sets up Ayam to use Aqsis from the application directory structure (`"/Applications/Aqsis.app"`) on Mac OS X. This is the default installation location of Aqsis on Mac OS X.

The script adapts the executable and shader search paths. Furthermore, environment variables vital for Aqsis to work will be set up properly.

Note that the script does not change the `"RIB-Export/Renderer"` preferences, you still have to switch to Aqsis using the main menu `"Special/Select Renderer"` once.

6.5.21 Use Pixie from Library Directory

The script `"usepixie.tcl"` sets up Ayam to use Pixie from the `"/Library/pixie"` directory on Mac OS X. This is the default installation location of Pixie on Mac OS X.

The script adapts the executable, shared library, and shader search paths. Furthermore, environment variables vital for Pixie to work will be set up properly.

Note that the script does not change the `"RIB-Export/Renderer"` preferences, you still have to switch to Pixie using the main menu `"Special/Select Renderer"` once.

6.5.22 Replace Icons

The script `"myicons.tcl"` allows to replace icons of the Ayam user interface, e.g. toolbox icons, with user defined ones. The new icons must be GIF image files of size 25 by 25 and reside in the `"icons"` directory relative to the Ayam executable.

The names of the image files may be obtained from the script or by the following scripting interface command (in the Ayam console):

```
» image names
```

Action icon variants (e.g. for the scale about actions) can also be created automatically by changing the `"createVariants"` variable in the script file.

6.5.23 Dynamic Tree

The script `"dtree.tcl"` (dynamic/fast tree) replaces some of BWidgets tree code for faster interaction in scenes with many objects.

The original BWidget tree widget (and accompanying script code in Ayam) creates a canvas item for every object in the scene, even if it is not visible in the tree widget. Therefore, tree update operations that occur

e.g. after loading of a scene or after drag-and-drop operations, can become very slow, when a scene has many objects.

If the dtree script is active, there are only as many canvas items as there are nodes visible in the current scroll region. Therefore, working with many objects in long lists becomes much faster for operations that require tree updates. However, operations like opening or closing sub trees or even just scrolling in the tree will become a bit slower, as canvas items need to be created constantly.

6.5.24 Control Vertex View

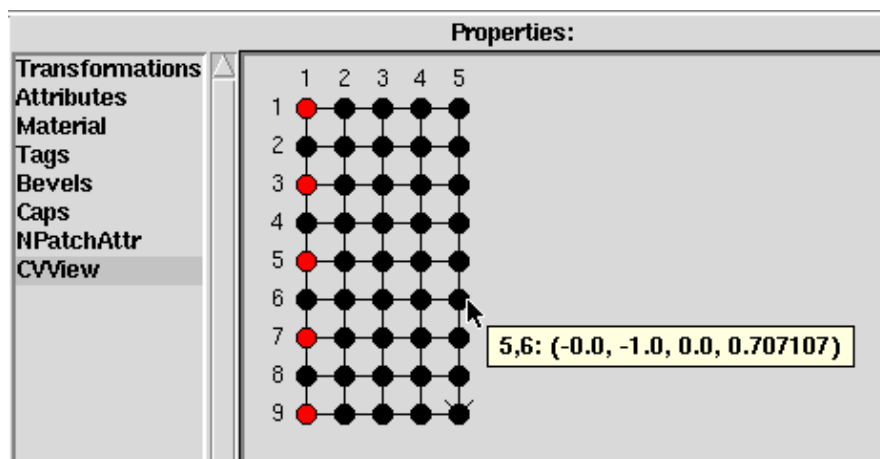


Figure 174: Control Vertex View Example

The script "cvview.tcl" (control vertex view) allows to view the control points of NURBS surface or curve objects as property, see also the image above.

The script currently supports the following object types: APatch, IPatch, NPatch, PatchMesh, ACurve, ICurve, NCurve, BCurve, and SDCurve. In addition also objects that provide NPatch/NCurve objects and show a NPInfo/NCInfo entry in their parameter property are supported.¹

If the script is loaded, there is a new entry in the "Custom" menu ("Add CVView to Object") that allows to add a new property ("CVView") to an individual surface or curve object.

The "CVView" property GUI displays all control vertices in a regular grid. The coordinates of each vertex will be shown when the mouse pointer hovers over it. This also works if the control points lump together in 3D space or have unusual values.

Additionally, the current point selection is visualized by painting the selected vertices in red and the selection may also be adjusted by clicking on the circles. Drag selection is also possible.²

¹ Since 1.26. ² Since 1.28.

6.5.25 Curvature Plot

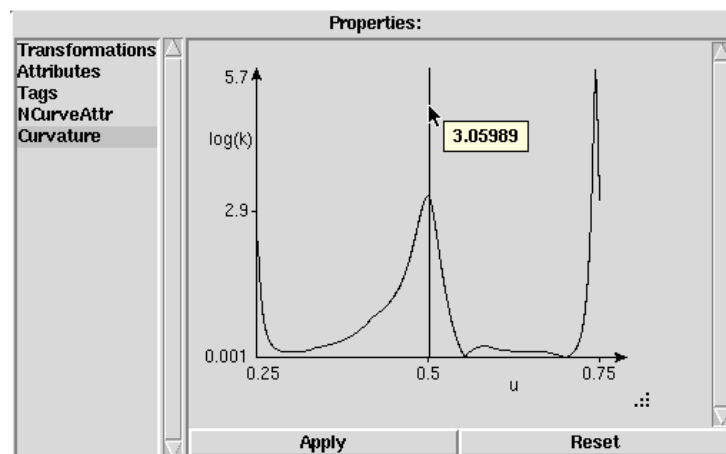


Figure 175: Curvature Plot Example

The script "curvature.tcl" allows to view the curvature of NURBS curves and NURBS curve providing objects as property, see also the image above.

If the script is loaded, there is a new entry in the "Custom" menu ("Add Curvature View to Object") that allows to add a new property ("Curvature") to an individual curve object.

The "Curvature" property GUI displays the curvature of the curve as live and interactive diagram. The diagram will be updated if the curve changes. The diagram can be zoomed by dragging with the leftmost mouse button and panned by dragging with the rightmost mouse button (in zoomed state). Zooming is also possible with the mouse wheel.

The zoomed state is conveyed by prepending/appending "..." to/on the respective label of the horizontal scale.

Note that when panning, the new section of the curve will be re-scaled to completely fill the Y-axis.

For NURBS curve objects the real knot value range will be shown on the x axis whereas for NURBS curve providing objects a relative knot value range ([0, 1]) will be displayed.

A click on the "k" label toggles between absolute and logarithmic scaling of the Y-axis.

There is a resize handle that allows to adapt the size of the diagram to the property canvas.

6.5.26 NURBS for X3DOM

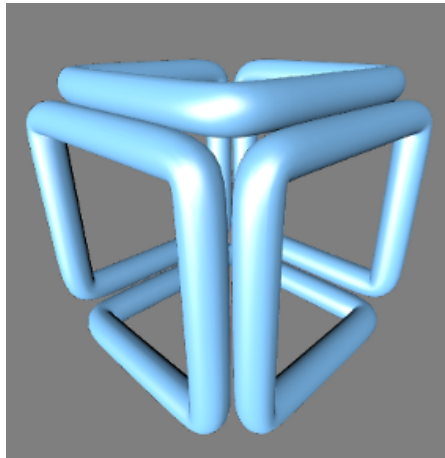


Figure 176: X3dom NURBS Display

The "x3dom-nurbs" script implements the `<NurbsPatchSurface>` and `<NurbsTrimmedSurface>` X3D NURBS nodes for x3dom (see <http://www.x3dom.org/>) in JavaScript.

After loading of the scene into the web browser, the NURBS surfaces from these nodes are tessellated into `<IndexedTriangleSet>` nodes. This allows to directly publish NURBS models on the web without prior conversion to a polygonal representation which is cumbersome, inflexible, and leads to higher bandwidth consumption.

The tessellator is based on idea and example code from A. J. Chung and A. J. Field: *"A Simple Recursive Tessellator for Adaptive Surface Triangulation"* in Journal of Graphics Tools Vol. 5, Iss. 3, 2000.

The implementation spans four script files:

- `x3dom-nurbs-nodes.js` – interface of the tessellator to x3dom
- `x3dom-nurbs-pool.js` – worker management (current pool size is 3)
- `x3dom-nurbs-worker.js` – a worker
- `x3dom-nurbs-tess.js` – the tessellator

In order to use the tessellator just add the following to your XHTML file after inclusion of "x3dom.js":

```
<script type="text/javascript" src="x3dom-nurbs-pool.js"/>
<script type="text/javascript" src="x3dom-nurbs-nodes.js"/>
```

As the tessellator is fully automatic, no further adjustments are needed.

Proper XHTML files can be created using the X3D export in x3dom-mode, see section 7.15.3 X3D (Web3D) export options (page 510).

While the tessellator runs in the background an initial polygonal representation that is directly derived from the control polygon of the NURBS surface is shown. Additionally, a busy prompt is displayed, see also the image below. As the busy prompt is derived from the x3dom loading prompt, it may also be styled with the "x3dom-progress" style in "x3dom.css".

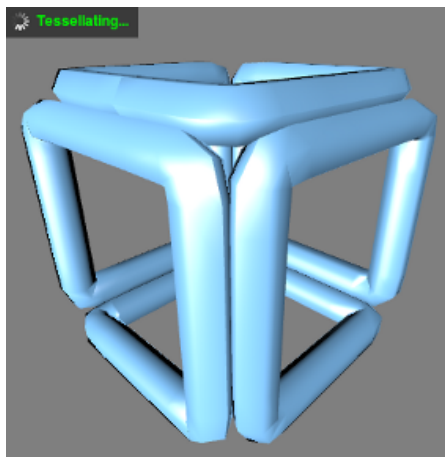


Figure 177: X3dom NURBS Busy Prompt

The tessellation runs in so called webworkers, processes that run in parallel to the main browser thread, in order to not block user interaction and employ multi-core CPUs. The webworker pool is currently hard coded to only use three such webworkers at any given time. This value may be adapted easily in `"x3dom-nurbs-pool.js"` (e.g. if your average target CPU has more cores). Note that each webworker will tessellate a single NURBS surface, i.e. scenes with only one surface will not benefit from the parallelism.

By default, the tessellator tries to create a tessellation that represents all important surface features in a way that a visual inspection from mid to close viewing distance does not reveal the nature of the underlying triangular representation. This may be too fine/slow for objects that are never viewed from close distance or too coarse for very detailed objects.

Therefore, and in concordance with the respective suggestions of the X3D specification, the tessellation quality and speed may be adjusted using the `"uTessellation"` and `"vTessellation"` attributes of the `<NurbsPatchSurface>` or `<NurbsTrimmedSurface>` nodes as explained in the next sections.

Object Space Based Sampling

If no "uTessellation" attribute is specified, or its value is positive, the tessellator uses the so called *object space sampling* mode. In this mode the tessellator subdivides an initial set of triangles recursively until all edges of those triangles are shorter than a given threshold value (in object space). The threshold value is set automatically so that an object extending one unit by one unit in object space is subdivided to about 15 by 15 by two triangles. The "uTessellation" attribute value is simply multiplied into this automatically determined threshold.

Therefore "uTessellation" values larger than 1.0 lead to a coarser and faster tessellation, whereas values smaller than 1.0 lead to a finer and slower tessellation.

Note that in this mode the value of the "vTessellation" attribute is *not* considered at all.

Also note that in highly curved regions of the surface and at trim edges, even smaller triangles than determined by the edge length criterion may be created.

Parametric Space Based Sampling

If the "uTessellation" attribute value is negative, the tessellator switches to *parametric space sampling*. In this mode also the "vTessellation" attribute is considered.

As in the object space sampling mode an automatic threshold value is computed, but here the extension of the object in parametric space (i.e. the number of control points) is used. This actually results in two threshold values, one for each parametric dimension. The "uTessellation" and "vTessellation" attributes are multiplied into those thresholds.

Therefore, values larger than -1.0 (in absolute value) lead to a coarser and faster tessellation in the respective dimension. Values smaller than -1.0 (in absolute value) lead to a finer and slower tessellation in the respective dimension. This means a value of -2.0 leads to roughly half as many triangles compared to the default and a value of -0.5 leads to twice as many triangles when compared to the default in the respective parametric dimension.

As the edge length computation is simpler and no curvature analysis is taking place, the parametric space sampling is considerably faster than object space sampling.

Note that at trim curve edges even smaller triangles than determined by the edge length criterion may be created.

Additional Attributes

To aid in parameterisation of the "uTessellation" and "vTessellation" attributes, a "normalPerVertex" attribute can be added to the respective `<NurbsPatchSurface>` or `<NurbsTrimmedSurface>` node. This attribute will then also be set for the corresponding `<IndexedTriangleSet>` node that is created by the tessellator. If the value of this attribute is "false", x3dom will display this surface in a flat shaded style and the tessellated triangles will be visible allowing easier judgment and adjustment of the tessellation quality.

Restrictions and Implementation Deficiencies

Due to memoization of surface points, parametric values must not exceed:

$$\frac{1.7976931348623157 \times 10^{308}}{10 \times 10^{10}} = 1.7976931348623157 \times 10^{297} \quad (2)$$

Texture coordinates are always directly derived from the parametric values.

Surface normals are not computed by the tessellator, but by x3dom. This can lead to normals that are off for very coarse tessellations but is much faster. Another benefit of this approach is, that normals in non-differentiable surface points (e.g. the poles of the standard NURBS sphere) do not flip.

There is *no* support for the following nodes:

`<NurbsTextureCoordinate>`,
`<NurbsSet>`,
`<NurbsSweptSurface>`,
`<NurbsSwungSurface>`,
`<NurbsCurve>`, and all
`<Nurbs*Interpolator>` nodes.

6.6 Distributed Script Objects

These scripts implement Script objects, see also section 4.9.1 Script Object (page 229).

6.6.1 Truncated Cone

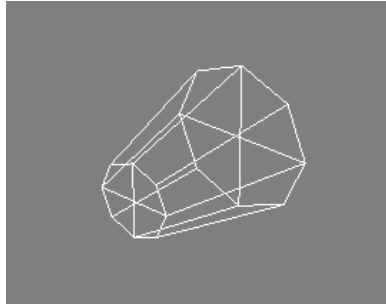


Figure 178: Truncated Cone with ZMax 2.0

The file `"tcone.tcl"` is a Script object script that creates truncated cones with arguments similar to the cylinder primitive, see also the above image.

This script must be used in a Script object of type `"Create"` (see section 4.9.1 Script Object (page 229)). For convenience, there is also a property GUI; one must add a `"NP"` tag of value `"TConeAttr"` to the Script object to see it.

These are the parameters of the truncated cone:

- `"Closed"` toggles whether the object should be automatically sealed (closed by matching cap surfaces).
Only when this option is enabled, the cone may be used in CSG operations safely.
- `"ThetaMax"` is the sweeping angle of the cone in degrees, default is 360.
- `"ZMin"` is the base of the cone, default is 0.
- `"ZMax"` is the peak of the cone, default is 1.
- `"RMin"` is the radius of the cone at the base, default is 1.
- `"RMax"` is the radius of the cone at the peak, default is 0.5.

Internally, the script creates a Hyperboloid; further information about conversion capabilities and RIB export may be found in section 4.3.8 Hyperboloid (page 149).

An example scene file containing such an object is distributed with Ayam, see the file:

`"ayam/scn/scripts/tcone.ay"`.

6.6.2 Disk with Hole

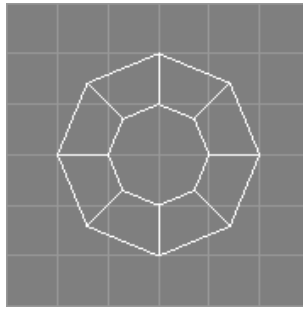


Figure 179: Disk with Hole Example

The file `"hdisk.tcl"` is a Script object script that creates a disk with a hole, also called an *annulus*. See also the above image.

This script must be used in a Script object of type `"Create"` (see section 4.9.1 Script Object (page 229)). For convenience, there is also a property GUI; one must add a `"NP"` tag of value `"HDiskAttr"` to the Script object to see it.

These are the parameters of the disk with hole:

- `"ThetaMax"` is the sweeping angle of the disk in degrees, default is 360.
- `"RMin"` is the inner radius of the disk or radius of the hole, default is 0.5.
- `"RMax"` is the outer radius of the disk, default is 1.0.

Internally, the script creates a Hyperboloid; further information about conversion capabilities and RIB export may be found in section 4.3.8 Hyperboloid (page 149).

An example scene file containing such an object is distributed with Ayam, see the file:

`"ayam/scn/scripts/hdisk.ay"`.

6.6.3 Box with Cylindrical Topology

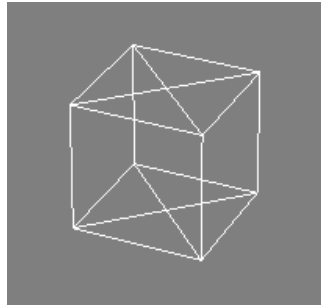


Figure 180: Box with Cylindrical Topology

The file `"cbbox.tcl"` is a Script object script that creates a NURBS surface in the form of a box with a cylindrical topology, see also the image above.

This script must be used in a Script object of type `"Create"` (see section [4.9.1 Script Object \(page 229\)](#)). For convenience, there is also a property GUI; one must add a `"NP"` tag of value `"CBoxAttr"` to the Script object to see it.

These are the parameters of the cylindrical box:

- `"Width"` extension of the box along the X-axis, default is 1.0.
- `"Depth"` extension of the box along the Z-axis, default is 1.0.
- `"Height"` extension of the box along the Y-axis, default is 1.0.

In contrast to the standard Box object, the cylindrical box converts to a single NURBS patch. However, this patch has poles and, consequently, shading artefacts may appear on the top and down side. Texturing this object is also challenging.

An example scene file containing such an object is distributed with Ayam, see the file:

`"ayam/scn/scripts/cbbox.ay"`.

6.6.4 NURBS Circle with Triangular Base

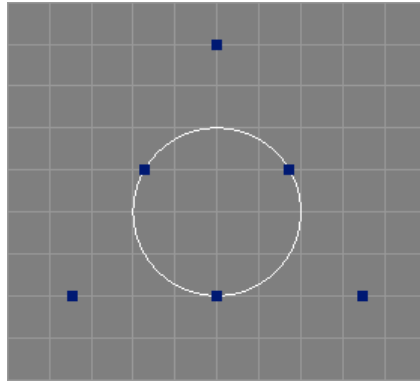


Figure 181: NURBS Circle with Triangular Base

The file `"tcircle.tcl"` is a Script object script that creates NURBS circles with a triangular base, see also the above image.

In contrast to the standard nine point NURBS circle with rectangular base, the triangular circle consists of just seven control points. This can save some memory, e.g. in long sweep objects. However, the parameterisation of the triangular circle is also slightly worse. Furthermore, only full circles that start on the positive X-axis are supported.

This script must be used in a Script object of type `"Create"` (see section 4.9.1 Script Object (page 229)). For convenience, there is also a property GUI; one must add a `"NP"` tag of value `"TCircleAttr"` to the Script object to see it.

These are the parameters of the triangular circle:

- `"Radius"` radius of the circle, default is 1.0.

An example scene file containing such an object is distributed with Ayam, see the file:

`"ayam/scn/scripts/tcircle.ay"`.

6.6.5 Helix

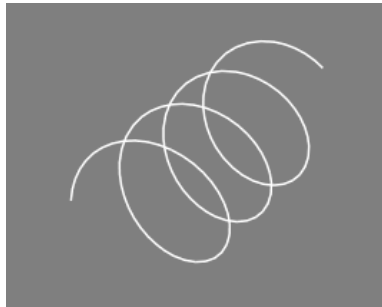


Figure 182: Helix Example

The script `"helix.tcl"` creates a NURBS curve that forms a helix, see also the above image.

This script must be used in a Script object of type `"Create"` (see section 4.9.1 Script Object (page 229)). There is also a property GUI provided; one must add a `"NP"` tag of value `"HelixAttr"` to the Script object to see it.

These are the parameters of the helix:

- `"Length"` is the number of control points in the curve, default is 30.
- `"Radius"` is the radius of the helix, default is 2.0.
- `"Angle"` is the offset angle of from one point to the next, default is 45.0.
- `"DZ"` is the offset along z from one point to the next, default is 0.25.

An example scene file containing such an object is distributed with Ayam, see the file:

`"ayam/scn/scripts/helix.ay"`.

6.6.6 Spiral



Figure 183: Spiral Example

The script `"spiral.tcl"` creates a NURBS curve that forms a spiral, see also the above image.

This script must be used in a Script object of type `"Create"` (see section 4.9.1 Script Object (page 229)). There is also a property GUI provided; one must add a `"NP"` tag of value `"SpiralAttr"` to the Script object to see it.

These are the parameters of the spiral:

- `"Length"` is the number of control points in the curve, default is 30.
- `"Angle"` is the offset angle of from one point to the next, default is 45.0.
- `"RMin"` is the start radius value, default is 0.1.
- `"RDiff"` is the radius difference from one point to the next, default is 0.1.

An example scene file containing such an object is distributed with Ayam, see the file:

`"ayam/scn/scripts/spiral.ay"`.

6.6.7 Oval

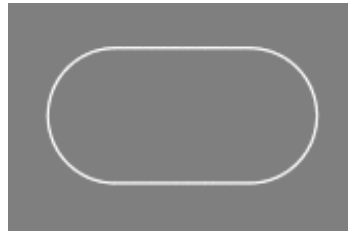


Figure 184: NURBS Oval

The file `"oval.tcl"` is a Script object script that creates NURBS curves in the form of an oval, here: two half circles connected by straight lines, see also the above image.

This script must be used in a Script object of type `"Create"` (see section [4.9.1 Script Object \(page 229\)](#)). For convenience, there is also a property GUI; one must add a `"NP"` tag of value `"OvalAttr"` to the Script object to see it.

These are the parameters of the oval:

- `"Length"` distance of the two half circle apexes on the x axis, default is 2.0.
- `"Radius"` radius of the half circles, default is 1.0.

An example scene file containing such an object is distributed with Ayam, see the file:

`"ayam/scn/scripts/oval.ay"`.

6.6.8 FilletNC

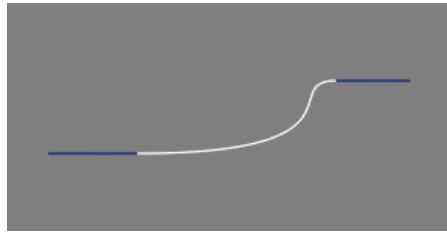


Figure 185: NURBS Fillet Curve

The file `"filletnc.tcl"` is a Script object script that creates a fillet NURBS curve between two other curves with adjustable individual tangent lengths, see also the above image.

This script must be used in a Script object of type `"Modify"` (see section 4.9.1 Script Object (page 229)) and instances of the curves to connect should be children of the Script object. For convenience, there is also a property GUI; one must add a `"NP"` tag of value `"FilletNCAttr"` to the Script object to see it.

These are the parameters of the fillet:

- `"TanLenS"` tangent length at start of fillet,
- `"TanLenT"` tangent length at end of fillet.

An example scene file containing such an object is distributed with Ayam, see the file:

`"ayam/scn/scripts/filletnc.ay"`.

If instead of two instances of individual curves, two instances of a single curve are present as children, the fillet will close the parameter curve. An example scene file of this special setup is distributed with Ayam, see the file:

`"ayam/scn/scripts/filletncc.ay"`.

6.6.9 DualSweep

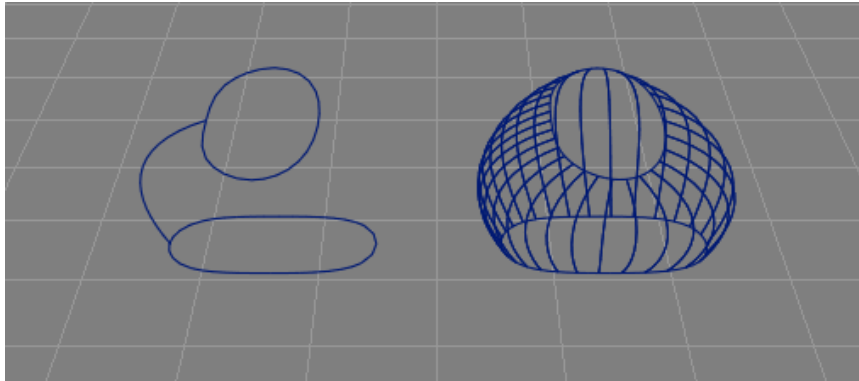


Figure 186: DualSweep Example

The file "dualsweep.tcl" is a Script object script that creates a NURBS surface from three parameter curves similar to birailing but with cross section curves perpendicular to both rails, see also the above image.

The first curve is the so called *cross section*. This curve must be planar and defined in the YZ-plane. It is the leftmost open curve in the upper example image.

The second curve is the first *rail* curve. This curve should start in the starting point of the cross section curve. It is the lower closed curve in the upper example image.

The third parameter curve is the second *rail* curve. This curve should start in the end point of the cross section curve. It is the upper closed curve in the upper example image.

This script must be used in a Script object of type "Modify" (see section 4.9.1 Script Object (page 229)). For convenience, there is also a property GUI; one must add a "NP" tag of value "DualSweepAttr" to the Script object to see it.

These are the parameters of the DualSweep:

- "Type" similar to the corresponding property of the Sweep object, this allows to set the type of the resulting surface (open, closed, or periodic).
- "Sections" number of sections to use, also works in similar fashion as for the Sweep object (see also 4.7.4 SweepAttr Property (page 192)). The default value is 0, i.e. the number of sections are determined from the parameter curves dimensions.

An example scene file containing such an object is distributed with Ayam, see the file:

"ayam/scn/scripts/dualsweep.ay".

6.6.10 Translational Surface

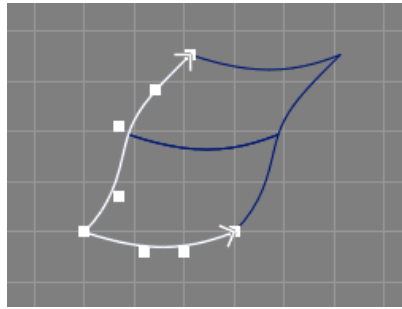


Figure 187: Translational Surface Example

The script `"tsurf.tcl"` creates a translational surface from two parameter curves, see also the above image. The control points of the translational surface are created by copying the control points of the first curve n times, where n is the number of control points of the second curve, while also offsetting the points according to the offset of the n th control point to the first control point of the second curve.

The curves do not need to touch in any point, but if they start in the same point, the translational surface will interpolate both curves.

Rational parameter curves are currently not supported.

This script must be used in a Script object of type "Modify" (see section 4.9.1 Script Object (page 229)). As the surface is completely defined by the parameter curves, there are no additional parameters and there is, consequently, also no property GUI.

An example scene file containing such an object is distributed with Ayam, see the file:

`"ayam/scn/scripts/tsurf.ay"`.

6.6.11 Extrusion along Normal

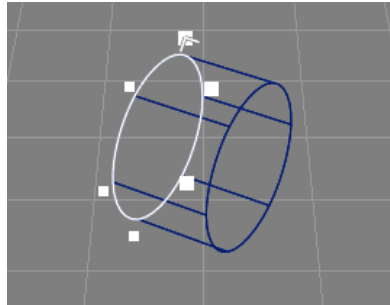


Figure 188: Extrusion along Normal Example

The script `"extruden.tcl"` extrudes a NURBS curve along its normal, see also the above image.

If the curve object has a MN tag, it will take precedence over the normal computation via `"getPlaneNormal"`. This is faster and also allows to create sheared extrusions.¹

Internally, the script creates a Skin from two curves, therefore, arbitrarily oriented and non-planar curves are supported.

This script must be used in a Script object of type `"Modify"` (see section 4.9.1 [Script Object \(page 229\)](#)). For convenience, there is also a property GUI; one must add a `"NP"` tag of value `"ExtrudeNAttr"` to the Script object to see it.

These are the parameters of the extrusion:

- `"Height"` is the amount of displacement of the second curve along the normal, default is 1.0.

An example scene file containing such an object is distributed with Ayam, see the file:

`"ayam/scn/scripts/extruden.ay"`.

¹ Since 1.26.

6.6.12 Create Polyhedrons from Conway Notations

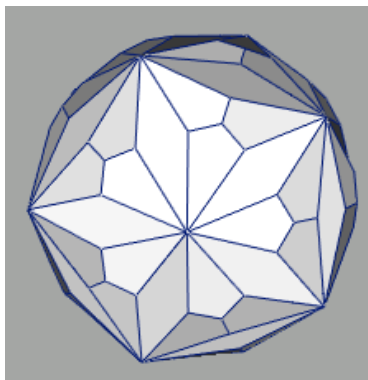


Figure 189: Polyhedron generated from Conway notation: "jtD"

Since Ayam 1.18 there is a complete example script for the JavaScript scripting interface distributed as "polyhedron.js" which creates polyhedrons from Conway notations. The script is based on the online Polyhedron VRML generator by George W. Hart:

http://www.georgehart.com/virtual-polyhedra/conway_notation.html

This script must be used in a Script object of type "Create" (see section 4.9.1 Script Object (page 229)). For convenience, there is also a property GUI; to make this GUI visible a "NP" tag of value "PolyhedronAttr" must be added to the Script object.

The Conway notation defines a set of operations executed consecutively on a seed/basic shape. The script currently supports the following seeds and operations (information taken from George W. Harts fine web pages, see above).

Seeds:

The Platonic solids are denoted T, O, C, I, and D, according to their first letter. Other polyhedra which are implemented here include prisms: P_n, antiprisms: A_n, and pyramids: Y_n, where n is a number (3 or greater) which must be specified to indicate the size of the base, e.g. , Y₃=T, P₄=C, and A₃=O.

Operations:

Currently, d, t, k, a, j, s, g, e, b, o, m, r, and p are defined. They are motivated by the operations needed to create the Archimedean solids and their duals from the Platonic solids. The following tables explain the operations in more detail:

These abbreviated explanations were again taken from George W. Hart.

Letter	Name	Description
d	dual	The dual of a polyhedron has a vertex for each face, and a face for each vertex, of the original polyhedron, e.g. $dC=O$.
$t / \tau n$	truncate all / just n -fold vertices	Truncating a polyhedron cuts off each vertex, producing a new n -sided face for each n -fold vertex.
k / kn	kis all / just n -sided faces	The kis operation divides each n -sided face into n triangles. A new vertex is added in the center of each face.
a	ambo	The ambo operation can be thought of as truncating to the edge midpoints. It produces a polyhedron, aX , with one vertex for each edge of X .
j	join	The join operator is dual to ambo, so $jX=dadX=daX$. jX is like kX without the original edges of X .
e	expand	Each face of X is separated from all its neighbors and reconnected with a new 4-sided face, corresponding to an edge of X . An n -gon is then added to connect the 4-sided faces at each n -fold vertex.
s	snub	The snub operation can be thought of as eC followed by the operation of slicing each of the new 4-fold faces along a diagonal into two triangles. With a consistent handedness to these cuts, all the vertices of sX are 5-fold.
g	gyro	The dual operation to s is g . g is like k but with the new edges connecting the face centers to the $1/3$ points on the edges rather than the vertices.
b	bevel	The bevel operation can be defined by $bX=taX$.
o	ortho	Dual to e , $oX=deX=jjX$. oX has the effect of putting new vertices in the middle of each face of X and connecting them, with new edges, to the edge midpoints of X .
m	meta	Dual to b , m is like k and o combined; new edges connect new vertices at the face centers to the old vertices and new vertices at the edge midpoints.

Table 107: Conway Notation Operations

Letter	Name	Description
r	reflect	Changes a left-handed solid to right handed, or vice versa, but has no effect on a reflexible solid. So $rC=C$, but compare sC and rsC .
p	propellor	Makes each n -gon face into a "propellor" of an n -gon surrounded by n quadrilaterals, e.g. pT is the tetrahedrally stellated icosahedron. Try pkD and $pt6kT$. p is a self-dual operation, i.e., $dpx=pX$ and $dpX=pdX$, and p also commutes with a and j , i.e. $paX=apX$.

Table 108: Additional Operations

6.7 JavaScript Scripting Interface

This section contains the documentation of the JavaScript scripting interface which is available after loading of the "jsinterp" plugin.

The JavaScript scripting interface exists since Ayam 1.18 and is based on the Mozilla SpiderMonkey JavaScript engine.

Upon loading, the "jsinterp" plugin creates *one* JavaScript context that lives (with all variables and objects defined therein) until Ayam exits.

6.7.1 Accessing JavaScript from Tcl and Script Objects

The JavaScript functionality may be accessed from the Tcl scripting interface via the "jsEval" command. The command can be used either to directly execute JavaScript code provided via the command's argument (Tcl code in **bold**):

```
» jsEval {var a = 0; a = a + 5.5; tclset("a", a);}
```

or to execute JavaScript code from a file:

```
» jsEval -f scriptfile.js
```

Note, that this command is not available in the safe interpreter.

Furthermore, Script object scripts may also be implemented in JavaScript, provided the first line of the script is a comment that instructs Ayam to use the JavaScript interpreter:

```
/* Ayam, use:  JavaScript */  
var a = 0;  
...
```

Note that the JavaScript scripting context inherits the limitations of the calling Tcl context. For example, when running in a Script object, the following code fails:

```
tcleval("exit");
```

because the Tcl command "exit" is not available in the safe interpreter. The command will not fail, when the calling context is the main Tcl interpreter; one can e.g. type into the Ayam console:

```
» jsEval {tcleval("exit");}
```

and Ayam quits (see also section: [4.9.1 Safe Interpreter \(page 233\)](#)).

6.7.2 JavaScript Functions

This subsection informs about the global functions additionally available in the Ayam JavaScript interpreter.

Those are converted Tcl commands (e.g. `crtOb()`), `tcleval()`, `tclvar()`, and `tclset()`.

Converted Commands:

The functionality of Ayam is accessible from JavaScript via a larger set of global functions, named as the corresponding Tcl commands. For instance, Ayam objects can be created in JavaScript using a function call like this:

```
crtOb("NCircle");
```

or, with additional arguments:

```
crtOb("NCircle", "-radius", 3.0);
```

In general, all commands available in the safe Ayam Tcl interpreter are also available as function (refer to section 6.2 Procedures and Commands (page 349) for a more or less complete list of those commands).

Note that Tcl procedures are generally not available as global JavaScript function, but they can be called using `tcleval()` as documented in the next paragraph.

`tcleval()`:

This global JavaScript function allows to evaluate arbitrary Tcl scripts, delivered as string argument:

```
var a = 42;
a = tcleval("puts " + a + "; return 5;");
tcleval("puts " + a);
/* expected output: 42 5 */
```

The `tcleval()` function provides access to all the functionality of Ayam that is just available as a Tcl procedure. Note that return values are properly transferred back to JavaScript according to the rules for data conversion as documented below. However, due to an intermediate conversion to string data, the overhead of such a call is considerable and bulk data transport should be arranged by other means, see below.

`tclvar()`:

Using the JavaScript function `tclvar()` a link between a Tcl variable and a corresponding variable in the JavaScript context may be established. The `tclvar()` function essentially creates a write trace on the Tcl variable, so that changes on the Tcl side are always automatically reflected on the JavaScript side:

```
tclvar("a");
tcleval("set a 42");
tcleval("puts " + a);
/* expected output: 42 */
```

Mind that the corresponding variable on the JavaScript side does *not* exist until the first write operation on the Tcl variable occurs. The Tcl variable, in turn, does not have to exist, when the `tclvar()` function

is called (i.e. all the work is done in the trace callback). If the variable name contains a namespace specifier, this namespace has to exist, when `"tclvar()"` is called.

Even though it looks a perfect fit, `"tclvar()"` can not be used to manage a property data array (if the array contains components to be saved to Ayam scene files). This is, because upon reading a scene file with such saved array items, the items will be read (and put into the Tcl context) before the script can establish the write trace using `"tclvar()"` and the data from the scene file never arrives in the JavaScript context. There is no easy way to get around this. A suggested way to manage a property data array is shown in the complete examples section below.

tclset() :

The third global JavaScript function is `"tclset()"` that allows to efficiently set Tcl variables from the JavaScript context avoiding conversion to string data and back. For example:

```
var a = 3.3;
var b = new Array(1, 3, 5);
tclset("a", a);
tclset("b", b);
```

sets the Tcl variable `"a"` to the floating point value 3.3, and `"b"` to a list of integer values { 1 3 5 }. Note that the variable names may also point to Tcl array elements, for instance

```
tclset("SphereAttrData(Radius)", 1.2);
```

sets the Radius element in the SphereAttrData array; or contain namespace specifiers, for example

```
tclset("::MyNameSpace::Radius", 1.2);
```

sets the Radius variable in the MyNameSpace namespace.

6.7.3 Data Conversion

When data is transferred from the Tcl to the JavaScript side (e.g. while converting return values of `"tcl eval ()"` or variable values linked via `"tcl var ()"`), the following conversions are in effect: Scalar data types will be converted to their directly matching counterparts, except for Booleans, which will be converted to integer values. Lists will be converted to Array objects (nesting is allowed and will produce accordingly nested arrays). Associative arrays will be converted to objects with named properties. Unicode strings are currently not supported. See also the table below.

Tcl	JavaScript
Boolean (true, false)	Integer (1, 0)
Integer (2)	Integer (2)
Double (3.14)	Double (3.14)
String ("mystr")	String ("mystr")
List ({0 1 2})	Array ((0, 1, 2))
Array (mya(mye) = 0.1)	Object (mya.mye = 0.1)

Table 109: Tcl to JavaScript Data Conversions

When data is transferred from the JavaScript side to the Tcl side (e.g. as function argument), the following conversions are in effect: Scalar data types will be converted to their directly matching counterparts, Array objects will be converted to lists (nesting is allowed and will produce accordingly nested lists). Unicode strings and objects of a type other than Array (e.g. Boolean) are currently not supported. See also the following table.

JavaScript	Tcl
Integer (2)	Integer (2)
Double (3.14)	Double (3.14)
String ("mystr")	String ("mystr")
Array ((0, 1, 2))	List ({0 1 2})

Table 110: JavaScript to Tcl Data Conversions

The transport/conversion of object properties (to e.g. associative array elements) can be arranged manually like this:

```
var a = new Object();
a.b = 3.14;
tclset("a(b)", a.b);
```

6.7.4 Complete Examples

This section contains two complete examples for Script objects written in JavaScript.

For the first example use Script object type "Modify" and put a Sphere as child object of the Script object.

```
/* Ayam, use: JavaScript */
tclvar("SphereAttrData");
getProp();
if(SphereAttrData)
{
    tclset("SphereAttrData(ZMin)", -SphereAttrData.Radius);
    tclset("SphereAttrData(ZMax)", SphereAttrData.Radius);
    setProp();
}
```

The above script will make sure, that the ZMin and ZMax parameters of the Sphere object always match its radius.

First, a link from the original Sphere object property data array "SphereAttrData" is established, so that when "getProp()" (a converted Tcl Ayam command) is called, also the JavaScript object "SphereAttrData" is filled with meaningful data.

The next line (the if) is a safety measure that prevents the script from failing if the child object of the Script object is not a Sphere object.

Now the radius value is transferred back to Tcl directly into the property data array to the ZMin and ZMax entries respectively with the help of "tclset()".

Finally, the modified property is transferred back to the Sphere object again with a converted Tcl Ayam command "setProp()".

The next example shows, how to manage a property GUI in a JavaScript implemented Script object script. Use Script object type "Create" and add a tag "NP MyProp" to see the property GUI.

```

/* Ayam, use: JavaScript, save array: MyPropData */
var MyPropData = new Object();
if(!tclevel("info exists MyPropData;"))
{
    /* initial script run (but not when loaded from scene file!) */
    MyPropData.MyItem = tclevel("set MyPropData(MyItem) 1.0;");
    tclevel("set MyPropData(SP) {MyItem};");
}
else
{
    /* all following script runs (and also when loaded from scene file!) */
    MyPropData.MyItem = tclevel("set MyPropData(MyItem);");
}
if(!tclevel("info exists MyPropGUI;"))
{
    tclevel("set ::phw [addPropertyGUI MyProp \"\" \"\"];");
    tclevel("addParam $::phw MyPropData MyItem;");
}
crtOb("Sphere");
sL();
getProp();
tclset("SphereAttrData(Radius)", MyPropData.MyItem);
tclset("SphereAttrData(ZMin)", -MyPropData.MyItem);
tclset("SphereAttrData(ZMax)", MyPropData.MyItem);
setProp();

```

This example demonstrates how to manage property data using the JavaScript object variable "MyPropData". The property data can be saved to and read from Ayam scene files with the help of a mirroring array variable on the Tcl side (also named "MyPropData"). To make this work properly, the initialisation of the JavaScript object must be constrained to the first script run: when the property data was read from a scene file, initialisation must not be run, instead the read data must be fetched from the Tcl context. This is what the first "if" statement, checking for existence of the mirroring Tcl array variable, in above example is all about.

Following this scheme of dual mirroring data structures on the Tcl and JavaScript sides, now the property GUI is created, which is also constrained to just one script run by a similar "if" statement.

After the GUI, a Sphere object is created and parameterised according to the data in the property GUI, which is used as radius, zmin, and zmax value.

6.8 Lua Scripting Interface

This sections contains the documentation of the Lua scripting interface which is available after loading of the "luainterp" plugin.¹

Upon loading, the "luainterp" plugin creates *one* Lua context that lives (with all variables and objects defined therein) until Ayam exits.

6.8.1 Accessing Lua from Tcl and Script Objects

The Lua functionality may be accessed from the Tcl scripting interface via the "luaEval" command. The command can be used either to directly execute Lua code provided via the commands argument (Tcl code in **bold**):

```
» luaEval {a = 0; a = a + 5.5; tclset("a", a);}
```

or to execute Lua code from a file:

```
» luaEval -f scriptfile.lua
```

Note, that this command is not available in the safe interpreter.

Furthermore, Script object scripts may also be implemented in Lua, provided the first line of the script is a comment that instructs Ayam to use the Lua interpreter:

```
-- Ayam, use:  Lua
a = 0
...
```

Note that the Lua scripting context inherits the limitations of the calling Tcl context. For example, when running in a Script object, the following code fails:

```
tcleval("exit")
```

because the Tcl command "exit" is not available in the safe interpreter. The command will not fail, when the calling context is the main Tcl interpreter; one can e.g. type into the Ayam console:

```
» luaEval {tcleval("exit")}
```

and Ayam quits (see also section: [4.9.1 Safe Interpreter \(page 233\)](#)).

¹ Since 1.21.

6.8.2 Lua Functions

This subsection informs about the global functions additionally available in the Ayam Lua interpreter.

Those are converted Tcl commands (e.g. `"crtOb()"`), `"tcleval()"`, `"tclvar()"`, and `"tclset()"`.

Converted Commands:

The functionality of Ayam is accessible from Lua via a larger set of global functions, named as the corresponding Tcl commands. For instance, Ayam objects can be created in Lua using a function call like this:

```
crtOb("NCircle")
```

or, with additional arguments:

```
crtOb("NCircle", "-radius", 3.0)
```

In general, all commands available in the safe Ayam Tcl interpreter are also available as function (refer to section 6.2 Procedures and Commands (page 349) for a more or less complete list of those commands).

Note that Tcl procedures are generally not available as global Lua function, but they can be called using `"tcleval()"` as documented in the next paragraph.

`tcleval()`:

This Lua function allows to evaluate arbitrary Tcl scripts, delivered as string argument:

```
a = 42
a = tcleval("puts " .. a .. "; return 5;")
tcleval("puts " .. a)
-- expected output: 42 5
```

The `"tcleval()"` function provides access to all the functionality of Ayam that is just available as a Tcl procedure. Note that return values are properly transferred back to Lua according to the rules for data conversion as documented below. However, due to an intermediate conversion to string data, the overhead of such a call is considerable and bulk data transport should be arranged by other means, see below.

`tclvar()`:

Using the Lua function `"tclvar()"` a link between a Tcl variable and a corresponding variable in the Lua context can be established. The `"tclvar()"` function creates a write trace on the Tcl variable, so that changes on the Tcl side are always automatically reflected on the Lua side:

```
tclvar("a")
tcleval("set a 42")
tcleval("puts " .. a)
-- expected output: 42
```

Mind that the corresponding variable on the Lua side does *not* exist until the first write operation onto the Tcl variable occurs. The Tcl variable, in turn, does not have to exist, when the `"tclvar()"` function is

called (i.e. all the work is done in the trace callback). If the variable name contains a namespace specifier, this namespace has to exist, when `"tclvar()"` is called.

Even though it looks a perfect fit, `"tclvar()"` can not be used to manage a property data array (if the array contains components to be saved to Ayam scene files). This is, because upon reading a scene file with such saved array items, the items will be read (and put into the Tcl context) before the script can establish the write trace using `"tclvar()"` and the data from the scene file never arrives in the Lua context. There is no easy way to get around this. A suggested way to manage a property data array is shown in the complete examples section below.

tclset() :

The third global Lua function is `"tclset()"` that allows to efficiently set Tcl variables from the Lua context avoiding conversion to string data and back. For example:

```
a = 3.3
b = {1, 3, 5}
tclset("a", a)
tclset("b", b)
```

sets the Tcl variable `"a"` to the floating point value 3.3, and `"b"` to a list of integer values `{ 1 3 5 }`. Note that the variable names may also point to Tcl array elements, for instance

```
tclset("SphereAttrData(Radius)", 1.2)
```

sets the Radius element in the SphereAttrData array; or contain namespace specifiers, for example

```
tclset("::MyNameSpace::Radius", 1.2)
```

sets the Radius variable in the MyNameSpace namespace.

6.8.3 Data Conversion

When data is transferred from the Tcl to the Lua side (e.g. while converting return values of `"tcl eval ()"` or variable values linked via `"tcl var ()"`), the following conversions are in effect: Scalar data types will be converted to their directly matching counterparts. Lists will be converted to array-tables (nesting is allowed and will produce accordingly nested tables). Associative arrays will be converted to tables with properly named keys. Unicode strings are currently not supported. See also the table below.

Tcl	Lua
Boolean (true, false)	Boolean (true, false)
Integer (2)	Integer (2)
Double (3.14)	Double (3.14)
String ("mystr")	String ("mystr")
List ({0 1 2})	Array ({0, 1, 2})
Array (mya(mye) = 0.1)	Table (mya.mye = 0.1)

Table 111: Tcl to Lua Data Conversions

When data is transferred from the Lua side to the Tcl side (e.g. as function argument), the following conversions are in effect: Scalar data types will be converted to their directly matching counterparts, array-tables will be converted to lists (nesting is allowed and will produce accordingly nested lists). Sparse and mixed tables are currently not supported. Unicode strings are also currently not supported. See also the following table.

Lua	Tcl
Boolean (true, false)	Boolean (true, false)
Integer (2)	Integer (2)
Double (3.14)	Double (3.14)
String ("mystr")	String ("mystr")
Array ({0, 1, 2})	List ({0 1 2})

Table 112: Lua to Tcl Data Conversions

The transport/conversion of table entries (to e.g. associative array elements) can be arranged manually like this:

```
a.b = 3.14
tclset("a(b)", a.b)
```

6.8.4 Complete Examples

This section contains two complete examples for Script objects written in Lua.

For the first example use Script object type "Modify" and put a Sphere as child object of the Script object.

```
-- Ayam, use: Lua
tclvar("SphereAttrData")
getProp()
if SphereAttrData then
    tclset("SphereAttrData(ZMin)", -SphereAttrData.Radius)
    tclset("SphereAttrData(ZMax)", SphereAttrData.Radius)
    setProp()
end
```

The above script will make sure, that the ZMin and ZMax parameters of the Sphere object always match its radius.

First, a link from the original Sphere object property data array "SphereAttrData" is established, so that when "getProp()" (a converted Tcl Ayam command) is called, also the Lua object "SphereAttrData" is filled with meaningful data.

The next line (the if) is a safety measure that prevents the script from failing if the child object of the Script object is not a Sphere object.

Now the radius value is transferred back to Tcl directly into the property data array to the ZMin and ZMax entries respectively with the help of "tclset".

Finally, the modified property is transferred back to the Sphere object again with a converted Tcl Ayam command "setProp()".

The next example shows, how to manage a property GUI in a Lua implemented Script object script. Use Script object type "Create" and add a tag "NP MyProp" to see the property GUI.

```
-- Ayam, use: Lua, save array: MyPropData

if tcleval("info exists MyPropData;") == 0 then
    -- initial script run (but not when loaded from scene file!)
    MyPropData = {}
    MyPropData.MyItem = tcleval("set MyPropData(MyItem) 1.0;")
    tcleval("set MyPropData(SP) {MyItem};")
else
    -- all following script runs (and also when loaded from scene file!)
    MyPropData = {}
    MyPropData.MyItem = tcleval("set MyPropData(MyItem);")
end

if tcleval("info exists MyPropGUI;") == 0 then
    -- create property GUI "MyProp"
    tcleval("set ::phw [addPropertyGUI MyProp \"\" \"\"];")
    tcleval("addParam $::phw MyPropData MyItem;")
end

crtOb("Sphere")
sL()
getProp()
tclset("SphereAttrData(Radius)", MyPropData.MyItem)
tclset("SphereAttrData(ZMin)", -MyPropData.MyItem)
tclset("SphereAttrData(ZMax)", MyPropData.MyItem)
setProp()
```

This example demonstrates how to manage property data using the Lua object variable "MyPropData". The property data can be saved to and read from Ayam scene files with the help of a mirroring array variable on the Tcl side (also named "MyPropData"). To make this work properly, the initialisation of the Lua object must be constrained to the first script run: when the property data was read from a scene file, initialisation must not be run, instead the read data must be fetched from the Tcl context. This is what the first "if" statement, checking for existence of the mirroring Tcl array variable, in above example is all about.

Following this scheme of dual mirroring data structures on the Tcl and Lua sides, now the property GUI is created, which is also constrained to just one script run by a similar "if" statement.

After the GUI, a Sphere object is created and parameterised according to the data in the property GUI, which is used as radius, zmin, and zmax value.

7 Import and Export

This section contains the documentation of all import and export modules of Ayam.

7.1 Import and Export Plugin Management

Except for RIB export, all import/export modules of Ayam are plugins that need to be loaded into the application before possible usage. Loading of an import/export plugin may be done in three different ways:

1. *explicitly* via the main menu entry "File/Load Plugin",
2. *automatically* on application startup via a script (by adding e.g. "plugins/loaddx fio.tcl" to the "Scripts" preference setting),
3. *implicitly* via normal scene IO. Implicit loading means one can simply use the main menu entries "File/Open" and "File/Save as" (or the corresponding keyboard shortcuts) and specify a filename with the appropriate extension (e.g. ".dxf").¹ Ayam will then load the matching plugin ("dxfio") and will also open the import (or export) options dialog with the "FileName" option already set to the filename chosen before. Of course, implicit plugin loading requires that the "Plugins" preferences are correctly set.

7.2 Import and Export Plugin Overview

The following table lists the Ayam features supported by the various import plugins.

Feature	RIB	OBJ	3DMF(Apple)	DXF	3DM(Rhino)	X3D
Quadrics	Yes	No	Yes	No	Yes	Yes
Trimmed NURBS	Yes	Yes	Yes	No	Yes	Yes
Parametrics	No	No	No	No	No	Yes
Curves	No	Yes	Yes	Yes	Yes	Yes
Transformations	Yes	No	Yes	No	No	Yes
Hierarchy	Yes	No	Yes	No	No	Yes
Instances	Yes	No	No	No	No	No
CSG	Yes	No	No	No	No	No
Materials	Yes	Yes	No	No	No	No

Table 113: Ayam Features Supported by Various Import Plugins

Not all features of Ayam are supported in the various export options. The following table gives an overview of the supported features per export file format.

¹ Since 1.13.

Feature	RIB	OBJ	3DMF(Apple)	DXF	3DM(Rhino)	X3D
Quadrics	Yes	No ^a	Some ^d	No ^b	Some ^d	Some ^d
Trimmed NURBS	Yes	Yes	Yes	No ^b	Yes ^c	Yes
Parametrics	No ^a	No ^a	No ^a	No ^b	Some ^e	Some ^e
Curves	No	Yes	Yes	Yes	Yes	Yes
Transformations	Yes	No	Yes	No	No	Yes
Hierarchy	Yes	No	Yes	No	No	Yes
Instances	Yes	No	No	No	No	Yes
CSG	Yes	No	No	No	No	No
Materials	Yes	Yes	No	No	No	No

Table 114: Ayam Features Supported by Various Export Formats

^a: will be converted to NURBS

^b: will be converted to PolyMeshes

^c: 3D trim curves exported as PolyLines

^d: some quadrics are converted to NURBS (refer to plugin documentation)

^e: some parametrics are converted to NURBS (refer to plugin documentation)

Note that a successful export of a 3D scene to a different application not only depends on Ayam but also on the importing application. For instance, many applications claim to read files in the Wavefront OBJ format but only import polygonal data or, even worse, only triangles from such files. By default, Ayam tries to preserve as much information/design intent as possible in the respective export format leading e.g. to the use of NURBS in Wavefront OBJ files. Consequently, to successfully transfer an Ayam scene to a different application, in some cases, manual conversion of the NURBS objects in the Ayam scene to polygonal geometry may be necessary. There is a script provided that helps in doing this ("`topoly.tcl`", see also section 6.5.5 [Convert Everything to Polygons](#) (page 449)).

Ayam is not perfect either, as in most import options material and animation data is completely ignored.

The following table gives an overview of the file format versions supported by the various import and export plugins. Import of files from a different version should be considered unsupported.

Format	RIB	OBJ	3DMF(Apple)	DXF	3DM(Rhino)	X3D
Version	3.0	3.0	1.0	14	3.0	3.1

Table 115: Supported File Format Versions Overview

The next sections document the various import and export plugins in detail.

7.3 RenderMan Interface Bytestream (RIB) Import

Using the "rrib" (for **Read RIB**) plugin RenderMan Interface Bytestreams of version 3.0 can be imported into Ayam. This plugin is based on the Affine library by Thomas E. Burge. Start importing a RIB using the menu entry "File/Import/RenderMan RIB" (if this menu entry is not available, the rrib plugin must be loaded using the menu entry "File/Load Plugin" first).

7.3.1 RIB Primitive Support

The RIB plugin supports import of the following geometric primitives:

- Quadrics (Sphere, Disk, Cylinder, Cone, Paraboloid, Hyperboloid, Torus),
- bilinear and bicubic patches and patch meshes,
- NURBS patches (with trim curves),
- (general) polygons and (general) polygon meshes,
- subdivision meshes (with all tags).

Furthermore, the plugin supports reading of CSG, object instances, archives, light sources (including area-lights), arbitrary linear transformations (except shear transformations), arbitrary RiOptions and RiAttributes, shaders (except transformation shaders and without array arguments), arbitrary primitive variables (e.g. varying or vertex)¹, and procedural objects and delayed read archives².

Texture coordinates will import as TC tags (see also section 4.11.3 [TC \(Texture Coordinates\) Tag](#) (page 260)). The handedness of the scene (set via RiOrientation) is tracked and converted to the right-handed default of Ayam.³

The RIB plugin does not support reading of curves, implicit surfaces (blobby models) and calls to the RenderMan interface that are not so much useful for a RIB import like e.g. RiMakeTexture.

Unsupported geometric primitives and other calls to the RenderMan interface are silently ignored.

Also note that for NURBS patches and bicubic patch meshes, points of type "P" will be promoted to "Pw". Points of type "Pz" are not supported by the plugin. Trimming of NURBS patches by clamping their knot ranges is also not supported (however, UMM/VMM tags will be created, that contain the new knot minimum and maximum values).⁴ See also section 4.11.17 [UMM/VMM \(U/V Min Max\) Tag](#) (page 269).

Furthermore, objects of type (general) polygon and polygon mesh will always be promoted to general polygon meshes.

Object instances are resolved to normal objects while importing. Instances may be easily created again using Automatic Instancing (see section 8.10 [Automatic Instancing](#) (page 531)). If there are multiple objects in a RenderMan object instance, an enclosing Level object will be created.⁵

Procedural objects will not be evaluated, instead, RiProc objects will be created, that carry all arguments and create the same sequence of RIB requests upon export as was read upon import.

Note that in the case of serious syntactic errors of the RIB file more informative error messages are printed to the stderr channel of Ayam (which is not redirected to the Ayam console).

¹ Since 1.7. ² Since 1.9. ³ Since 1.22. ⁴ Since 1.9. ⁵ Since 1.22.

7.3.2 RIB Import Options

The RIB import may be controlled via different options:

- "ScaleFactor", determines a global scale factor to be applied to all imported objects.
- "ReadFrame", specifies the number of the frame in the RIB to read. A value of -1 means, all frames are to be read. If a frame number is specified and this frame does not show up in the RIB as "FrameBegin <yournumber>" nothing will be imported.
- "ReadCamera": if this is switched on, a Camera object will be created when the RIB plugin encounters a "WorldBegin". You may drag this camera object onto a perspective View object in Ayam after import to see through the camera of the imported RIB.
- "ReadOptions", controls whether RiOptions are to be imported from the RIB to the scene. Note that those RiOptions will overwrite the current global settings in the Ayam scene.
- "ReadLights", if this is enabled the lights from the RIB will be imported.
- "ReadMaterial", controls whether material objects are to be created for the imported objects. All material objects are created in a special level named "Materials" in the top level of the scene. The plugin tries to keep the number of generated material objects as low as possible by comparing with already existing materials in this level. This also works with material objects that exist before the RIB plugin is invoked (as long as they reside in this special level).
- "ReadPartial", this option is useful if you want to import partial RIBs (e.g. archives) that do not contain a "WorldBegin". Be careful with this option (i.e. use it only if reading of a RIB fails), as it switches reading of all types of objects on, regardless of the RIB structure.
- "ReadSTrim" if switched off, no simple trims (trims that only run along the border of a NURBS patch and actually do not trim the surface) will be imported if they are the only trim curves.
- "RescaleKnots" allows to rescale the knot vectors of NURBS patches and trim curves so that the distances between different knots are not smaller than the given value. Using a value of 1.0e-04 leads to NURBS that may be safely drawn using GLU. The default value 0.0 means no scaling.
- "Progress": displays the progress of the import in terms of line numbers of the file to import (archives do not advance the progress).

7.4 RenderMan Interface Bytestream (RIB) Export

RenderMan Interface Bytestream (RIB) export is the most important export module of Ayam and in fact justifies its existence. All features of the Ayam object and scene structure are supported (hierarchy, CSG, instances, materials, lights etc.). Furthermore, Ayam also supports direct rendering from view windows, rendering in multiple passes for shadow maps, and permanent previews (where a RenderMan renderer is directly coupled to an Ayam view window).

The documentation on RIB export is spread over the Ayam documentation, this section gives some general information and otherwise just points to the real documentation sections.

RIB export is always available, it does not need a plugin to be loaded. The corresponding main menu entry is "File/Export/RenderMan RIB" and the corresponding keyboard shortcut is <Ctrl+E>. To control the RIB export and rendering, many options exist that are documented in [section 2.10.4 RIB-Export preferences \(page 66\)](#).

There are also some special ways to export RIBs available in the main menu: "Special/RIB-Export"; this is documented in section 2.2 [Special Menu](#) (page 38).

Ayam can not only export scenes as RIB but also call various RenderMan renderers to directly render the exported RIB files to the screen or to an image file. Documentation on how to export/render directly from a view window can be found in section 2.5 [View Menu](#) (page 42).

Invoking RIB export is also possible using the scripting interface, see the section 6.2.25 [RIB export](#) (page 424) for more information.

RIB export always honors "NoExport" tags and the "HideChildren" attribute.

7.5 Mops Import

In older versions of Ayam, Mops scenes could be imported using the main menu entry: "File/Import Mops". Since Ayam 1.13 Mops import is a plugin named "mopsi". After loading the plugin, Mops scenes may be imported using the main menu entry "File/Import/Mops".

Ayam is able to import most elements of a Mops scene except for RiAttributes attached to arbitrary geometric objects, because attributes and shaders are managed by material objects in Ayam. However, if a Mops object has a surface or displacement shader, a material object with the shaders from the Mops object and its RiAttributes will be automatically created and linked with the geometric object while importing. Only Mops objects with surface or displacement shaders are considered because otherwise a material object would have to be created for every imported Mops object. The material objects are named "mat0", "mat1" and so on. Make sure, that the current scene in Ayam does not contain material objects with those names, otherwise Mops import will not be able to create material objects for the scene to import.

The import options "ResetDM" and "ResetST" control, whether GLU display mode and tolerance settings (see sections 4.4.1 [NCurveAttr](#) (page 150), and 4.6.1 [NPatchAttr](#) (page 170) for more information about display mode and tolerance) of NURBS primitives should be reset to using global preference values (the default in Ayam) instead of using the values from the Mops scene file.

7.6 AutoCAD DXF Import

The "dxfio" plugin allows to import AutoCAD DXF (drawing interchange format) files into Ayam with the help of the Dime library (from Systems in Motion, Kongsberg SIM AS, Norway). This plugin only supports files up to version 14 of the file format.

Start importing a DXF file using the main menu entry "File/Import/AutoCAD DXF" (if this menu entry is not available, the dxfio plugin must be loaded using the menu entry "File/Load Plugin" first).

Note that the entire DXF file is read into memory before any geometry is created.

7.6.1 DXF Entity Support

The DXF import plugin supports reading of the following DXF entities: 3DFACE, ARC, CIRCLE, ELLIPSE, LINE, SOLID, TRACE, BLOCK, INSERT, POLYLINE, LWPOLYLINE, and SPLINE. Entities not listed here will be silently ignored.

3DFACE entities are imported as PolyMesh objects if either only the first three points are unique (the entity describes a triangle) or the face is planar else as BPatch objects.

ARC, CIRCLE, and ELLIPSE entities will be read as NCircle objects with corresponding "TMin", "TMax" parameters (for arcs and ellipses) and scale transformation values (for ellipses) set.

POLYLINE entities are completely supported:

- Polylines will be imported as NCurve objects;
- PolyMeshes and PolyFaceMeshes will be imported as PolyMesh objects;
- B-Spline and Bezier surfaces will be imported as NPatch objects.

LINE, LWPOLYLINE, and SPLINE entities will be imported as NCurve objects.

SOLID and TRACE entities are imported as BPatch objects.

BLOCK and INSERT entities will be converted to appropriate master (referenced) and instance objects (references).

Bulges and extrusions are *not* supported.

The following table comprehensively lists the supported DXF entities and their Ayam counterparts that will be created upon import.

DXF Entity	Avam Object
3DFACE	PolyMesh / BPatch
ARC	NCircle
CIRCLE	NCircle
ELLIPSE	NCircle
LINE	NCurve
SOLID	BPatch
TRACE	BPatch
POLYLINE	NCurve / PolyMesh / NPatch
LWPOLYLINE	NCurve
SPLINE	NCurve
INSERT	Instance

Table 116: DXF Import Conversion Table

7.6.2 DXF Import Options

The DXF import may be controlled via different options:

- "FileName": is the path and name of the DXF file to import.
- "ReadCurves": if this is disabled, no freeform curves will be imported.
- "ScaleFactor": allows to apply a scaling factor to all imported objects.
- "ReadLayers": using this import option, a single layer or a range of layers may be selected for import. By default, all entities from all layers will be imported.

- "RescaleKnots": allows to rescale the knot vectors of imported NURBS curves so that the distances between different knots are not smaller than the given value. A "RescaleKnots" value of 1.0e-04 leads to NURBS that may be safely drawn using GLU. The default value 0.0 means no scaling.
- "Progress": displays the progress of the import; from 0 to 50 percent, Dime is reading the DXF file; from 50 to 100 percent, the dxfio plugin is converting the DXF entities to Ayam objects.

7.7 AutoCAD DXF Export

The "dxfio" plugin allows to export Ayam scenes to AutoCAD DXF (drawing interchange format) files with the help of the Dime library (from Systems in Motion, Kongsberg SIM AS, Norway).

Start exporting to a DXF file using the main menu entry "File/Export/AutoCAD DXF" (if this menu entry is not available, the dxfio plugin must be loaded using the menu entry "File/Load Plugin" first).

Note that the entire Ayam scene is converted to a corresponding DXF model in memory before it is written to the DXF file.

Avaya only creates entities of type POLYLINE and SPLINE and misses very much information that could be saved to other formats (e.g. normals and texture coordinates). Therefore, the DXF export format should be avoided if possible.

7.7.1 Ayam Object and Properties Support

The export functionality of the dxfio plugin currently covers export of all boxes, quadrics, NURBS, PolyMeshes, instances, clones, script objects (of type "Create" or "Modify"), and objects that may be converted to NURBS curves or surfaces or to PolyMeshes. However, all boxes and quadrics will always be converted to NURBS surfaces and NURBS surfaces will be tessellated to PolyMeshes for export.

The scene hierarchy and CSG operations are fully ignored, all objects will be written as if combined by the union operator.

All transformations will be applied to the control points of the exported objects.

PolyMesh objects will be exported to POLYLINE (subtype PolyFaceMesh) entities. If a PolyMesh object contains faces with holes or with more than four points, it will be tessellated for export. Eventually existing normals will not be exported.

NURBS curves will be exported as SPLINE entities.

Instance objects are resolved for export.

Light sources, as well as Cameras, Views, and Materials are not exported.

Clamping the knot ranges of NURBS curves or surfaces via UMM/VMM tags is not supported. Texture coordinates will not be exported.

7.7.2 DXF Export Options

The DXF export may be controlled via different options:

- "FileName": is the path and name of the DXF file to export to.

- "ScaleFactor": allows to apply a scaling factor to all exported objects.
- "WriteSelected": exports only the selected objects.
- "ObeyNoExport": ignores all objects with "NoExport" tags.
- "IgnoreHidden": ignores all hidden objects.
- "WriteCurves": if this is disabled, no freeform curves will be exported.
- "TopLevelLayers": controls whether the top level Level objects in the Ayam scene to be exported should be interpreted as layers. If this option is enabled, all objects in these levels will be placed on the respective layer. Objects that are not in one of those levels will be written to the default layer. Furthermore, top level object names will become layer names.
- "Progress": displays the progress of the export; from 0 to 50 percent, the dxflib plugin is converting the Ayam objects to DXF entities; from 50 to 100 percent, Dime is writing the DXF file.

7.8 Wavefront OBJ Import

Since Ayam 1.8.3 a Wavefront OBJ (version 3.0) import facility is available and since Ayam 1.13 it is a plugin ("objio") that needs to be loaded before import. Since Ayam 1.34 also materials from accompanying MTL files can be imported. The corresponding main menu entry is "File/Import/Wavefront OBJ" (if this menu entry is not available, the objio plugin must be loaded using the menu entry "File/Load Plugin" first).

7.8.1 Wavefront OBJ Statement Support

Wavefront OBJ import supports reading of polygonal lines and faces with vertex normals and texture coordinates (the latter are read as primitive variable tags); statements: `v`, `vt`, `vn`, `l`, `f`.

Furthermore, freeform curves and surfaces (NURBS) with trim curves and with texture coordinates (again read as primitive variable tags) are supported; statements: `vp`, `cstype`, `deg`, `curv`, `curv2`, `surf`, `parm`, `trim`, `hole`, `end`.

Freeform curves and surfaces of basis type `bmatrix`, `cardinal` and `taylor` are currently not supported. Also, import of special curves (statement `scr`) and points (statement `sp`) as well as surface connectivity information (statement `con`) are currently not supported.

Furthermore, trimming of freeform curves and surfaces by additionally clamping their knot ranges will be realized by UMM/VMM tags, that contain the new knot minimum and maximum values.¹ See also section 4.11.17 UMM/VMM (U/V Min Max) Tag (page 269).

The grouping statements `g` and `o` are supported as means to import object names.

Since Ayam 1.34 materials from accompanying MTL files can be imported; statements: `mtllib`, `usemtl`. See also section 7.8.3 Wavefront Material Import (page 494).

Unsupported statements will be silently ignored.

Wavefront OBJ import expects the file to be syntactically correct. The plugin is not very good in detecting and reporting errors. If the import fails, a third party tool should be used first to check whether the Wavefront OBJ file is valid at all.

¹ Since 1.9.

Also, note that the objio plugin supports Wavefront version 3.0 syntax only, files that use older syntax will not be imported correctly.

7.8.2 Wavefront OBJ Import Options

The following options control the Wavefront OBJ import process:

- "FileName": is the name of the Wavefront OBJ file (version 3.0)
- "MergeFaces": controls whether consecutive polygonal faces should be merged into a single PolyMesh object for import. Note that the merged PolyMesh objects probably needs to be optimized if there are vertices used by multiple faces (main menu "Tools/PolyMesh/Optimize").
- "MergePVTags": controls whether the PV tags of PolyMesh objects should be merged as well if they are subject to automatic merging (see above).
- "ReadCurves": if this is disabled, no freeform curves will be imported. This option does *not* influence the import of trim curves.
- "ReadSTrim": if switched off, no simple trims (trims that only run along the border of a NURBS patch and actually do not trim the surface) will be imported if they are the only trim curves.
- "ReadTexCoords": controls if and how texture coordinates will be imported:
 - "None": disables texture coordinate import,
 - "Separately": leads to texture coordinates in two primitive variables ("s" and "t") which can be used directly by standard RenderMan shaders (e.g. "paintedplastic"),
 - "Combined": leads to texture coordinates in one primitive variable which would be easier to be used by exports to non RIB formats.
- "ReadMaterials": toggles reading of MTL files.
- "UseMaterials": switches, whether or not imported geometric objects will be connected to corresponding materials.
- "RescaleKnots": allows to rescale the knot vectors of NURBS curves, patches, and trim curves so that the distances between different knots are not smaller than the given value. A "RescaleKnots" value of $1.0e-04$ leads to NURBS that may be safely drawn using GLU. The default value 0.0 means no scaling.
- "ScaleFactor": allows to apply a scaling factor to all imported objects.
- "RationalStyle": determines how rational coordinates are stored in the file to be imported (see also section [1.2.4 Rational Style \(page 17\)](#));
- "DefaultIllum": specifies the default illumination model to use, when reading materials without illum statements. See also section [7.8.3 Wavefront Material Import \(page 494\)](#).
- "Progress": displays the progress of the import; from 0 to 100 percent, the objio plugin is reading lines from the Wavefront OBJ file and creating Ayam objects. The number may be a bit off occasionally, as the progress meter just counts lines and assumes a fixed medium line length of 28 characters.

7.8.3 Wavefront Material Import

Import of Wavefront MTL files is available since Ayam 1.34. The import of an MTL file is usually triggered by the corresponding `mtllib` statement in a Wavefront OBJ file, but MTL files can also be imported directly.

Regardless of how it was started and whether or not any objects from the corresponding Wavefront OBJ file actually use these materials, the material import always creates Ayam Material objects for *every* material in the MTL file.

The following material statements are supported: `newmtl`, `Ka`, `Kd`, `Ks`, `Tf`, `d`, `Ns`, `Ni`, `map_Ka`, `map_Kd`, `map_Ks`, `map_Ns`, `map_d`, `disp`, `bump`, `refl`, `illum`.

Decals (statement `decal`) are *not* supported.

Color statements (`Ka`, `Kd`, `Ks`, and `Tf`) support the RGB and CIEXYZ color spaces, but values declared in CIEXYZ will be converted to RGB for import. Spectral curves specified via the `spectral` option are *not* supported.

The dissolve statement (`d`) is fully supported, this includes the `-halo` option.

Import of bump maps (statement `bump`) supports the bump multiplier option (`-bm`).

All other texture map options (`-blendu`, `-blendv`, `-cc`, `-clamp`, `-mm`, `-imfchan`, `-o`, `-s`, `-t`, `-texres`) are *not* supported.

The only reflection map type supported by the `refl` statement is sphere ("*lat-long*" in RenderMan terms).

All normal illumination models (0-9) are fully supported, but 8 will be mapped to 3 and 9 to 4.

Materials not specifying an illumination model will use a model selected by the following set of rules:

- if a reflection map is used (`refl` statement), the model is at least 3,
- if the material is transparent (`d` statement), the model is at least 4,
- if the optical density is specified (`Ni` statement), the model is 7.

If none of the above rules apply, the illumination model is 2.

These rules can be overridden by the `"DefaultIllum"` import option.

Note that a wrongly specified illumination model can lead to the discarding of parameters that are specified in the MTL file but unused by the illumination model.

The various illumination models are realized through a set of RenderMan surface and displacement shaders that are distributed in the `"ayam/scn/shaders"` directory. These are named `RSmtli0.sl` to `RSmtli7.sl` and `RSmtld.sl` and need to be compiled for the respective target renderer before possible usage.

From 0 to 7, the illumination models get more capable, complex, have more parameters but also require more rendering time.

As the various shaders may also be used outside the Wavefront OBJ import context, they are explained in greater detail in the next sections.

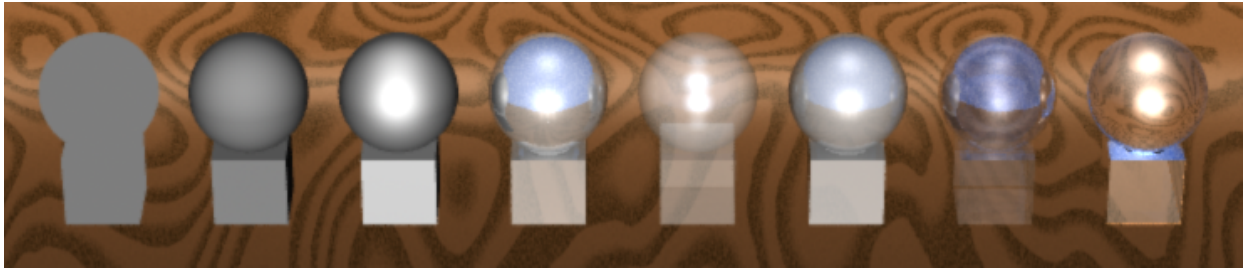


Figure 190: Illumination Models (0-7) Examples

RSmtli0 – Constant

This shader implements the most basic illumination model for constant color, i.e. ignoring any light source information, but nevertheless supporting opacity and bump mapping albeit the bump mapping only serves to distort the normal for the dissolve halo effect. Rendered objects will appear flat. A similar standard RenderMan shader would be `constant`.

RSmtli1 – Diffuse

The illumination model of this shader adds a diffuse term, i.e. uses light sources. Rendered objects will appear three dimensional, a similar standard RenderMan shader would be `matte`.

RSmtli2 – Specular

This shader adds specular highlights whose acuteness can be controlled via a parameter (`Ns`). Rendered objects will appear as if made from plastic, similar standard RenderMan shaders would be `plastic` or `paintedplastic`.

RSmtli3 – Reflection

This shader adds reflections from either ray tracing or a reflection map, a similar standard RenderMan shader would be `shinymetal`.

RSmtli4 – FakeGlass

This shader adds specular highlight saving to the opacity formula and can therefore be used to approximate glass without ray tracing. However, objects behind the fake glass will not appear distorted and the color of the transmitted light will also not be influenced.

RSmtli5 – FresnelReflect

This shader improves the raytraced reflections at grazing angles by employing the Fresnel equations.

RSmtli6 – Raytracing

This shader implements full ray traced reflections and refractions with coloring of transmitted light but without any Fresnel effects.

RSmtli7 – Fresnel

This shader improves the raytraced reflections and refractions at grazing angles by employing the Fresnel equations.

See also the image above, rendered with BMRT 2.6, scene distributed with Ayam as `"mtls.ay"`.

RSmtl Shaders and Material Attributes

Note that in contrast to established use in other RenderMan shaders, the `RSmtlix` surface shaders completely ignore the standard color and opacity attributes. To avoid confusion it is therefore suggested to set one component of these attributes to `-1` for Ayam Material objects that make use of one of these shaders. The color preview of the shaded drawing mode of Ayam is aware of this and will use the color set with the `Kd` surface shader parameter instead.

RenderMan Renderer Support

Not all RSmtl shaders are supported by all RenderMan renderers. The `RSmtli0` to `RSmtli4` shaders may be used with all renderers including pure REYES renderers (e.g. Aqsis or Pixie with the `stochastic` hider) whereas the other shaders, `RSmtli5` to `RSmtli7`, require ray tracing capable renderers, (e.g. BMRT or Pixie with the `raytrace` hider).

When using a REYES renderer with the `RSmtli3` shader, the `"refl_map"` parameter must be set.

7.9 Wavefront OBJ Export

Since Ayam 1.7, it is possible to export scenes or objects to the Wavefront OBJ format (version 3.0). Since Ayam 1.13, Wavefront OBJ export is a plugin ("objio") that needs to be loaded before export. The corresponding main menu entry is "File/Export/Wavefront OBJ" (if this menu entry is not available, the objio plugin must be loaded using the menu entry "File/Load Plugin" first).

7.9.1 Ayam Object and Properties Support

The Wavefront export currently supports the following objects:

- NCurve and objects that may be converted to NCurve objects (e.g. ICurve, ConcatNC, ExtrNC),
- NPatch (with trim curves) and objects that may be converted to NPatch objects (e.g. BPatch, PatchMesh, Revolve, Sweep, Extrude, Skin, Cap, Gordon, Birail1, Birail2, Text),
- Quadrics will be automatically converted to NURBS surfaces,¹
- PolyMesh and objects that may be converted to PolyMesh objects (e.g. MetaObj), faces with holes are not supported by the Wavefront OBJ format and will be tessellated to triangles for export automatically,
- Box (as six polygonal faces),
- Instance, Clone, Mirror; those will be resolved to normal objects for export as Wavefront OBJ does not support referenced geometry.

Since the Wavefront OBJ format does not support separate transformation attributes, all transformation attributes will be used to transform the coordinate values (the control points) of the exported objects.

The hierarchy of the Ayam scene will be squashed to one level, transformations will be properly delegated to the child objects.

CSG operations are fully ignored, all objects will be written as if combined by the union operator.

The Wavefront OBJ export supports material information which gets written to accompanying MTL files.² See also section 7.9.3 Wavefront Material Export (page 498).

Texture coordinates from primitive variable tags will be exported.³ Also TC tags are supported, but PV tags take precedence.⁴

UMM/VMM tags are used to trim the knot vectors of exported NURBS objects.⁵ See also section 4.11.17 UMM/VMM (U/V Min Max) Tag (page 269).

Light sources, as well as Cameras, and Views will not be exported.

Object names will be exported as `o` statements.

7.9.2 Wavefront OBJ Export Options

The following parameters control the Wavefront OBJ export:

- "FileName": is the filename of the Wavefront OBJ or MTL file;
- "WriteHeader": controls export of a header;

¹ Since 1.8.3. ² Since 1.34. ³ Since 1.8.3. ⁴ Since 1.35. ⁵ Since 1.9.

- "Header": allows to adjust the content of the header, the place holders %d, %t, and %u will be replaced by the current date, time, and user name respectively;
- "WriteSelected": exports only the currently selected object(s);
- "TessPoMesh" automatically tessellates all PolyMesh objects to triangles for export;
- "WriteCurves": toggles writing of NURBS curves and NURBS curve providing objects to the exported Wavefront OBJ file (This option does *not* influence the export of trim curves.);
- "WriteMaterials": toggles export of an accompanying MTL file, containing all materials used by the exported geometric objects, and prepends all objects in the OBJ file with a corresponding usemtl statement, which will be empty for objects without material if there was a material set before;
- "DefaultIllum": determines the illumination model to be used for all Material objects with unsupported surface shaders;
- "ScaleFactor": allows to apply a scaling factor to all exported objects;
- "RationalStyle": determines how to write rational coordinates (see also section 1.2.4 Rational Style (page 17));
- "Progress": displays the progress of the export; from 0 to 100 percent, the objio plugin is writing the Ayam objects to the Wavefront OBJ file.

7.9.3 Wavefront Material Export

Export of Ayam Material objects to MTL files is available since Ayam 1.34. The creation of a MTL file can be triggered by the "WriteMaterials" export option or by using a export filename with an .mtl-extension. In the latter case, all Material objects in the scene will be written to the MTL file (or just the selected Material objects, if the "WriteSelected" export option is enabled).

All materials created by a previous import of a Wavefront material are fully supported. See also section 7.8.3 Wavefront Material Import (page 494).

In addition, materials using one of the following RenderMan standard surface and displacement shaders are supported: "constant", "matte", "plastic", "paintedplastic", "metal", "shinymetal", "bumpy". Furthermore, there is support for the "glass" surface shader with the following parameters: "Ka", "Kd", "Ks", "roughness", "specularcolor", and "eta".

The following table comprehensively lists the illumination model used by the respective RenderMan surface shader upon export.

Surface Shader	Illumination Model
constant	0
matte	1
plastic	2
paintedplastic	2
metal	2
shinymetal	3
glass	7

Table 117: Wavefront MTL Export Illumination Model Table

If any of these materials has a opacity set, a corresponding dissolve statement (d) will be exported.

Materials with unsupported shaders will be written using a "Kd" statement with values taken from the material Color attribute, a "d" (dissolve) statement with values taken from the material Opacity attribute (if not set to fully opaque), and a "illum" statement according to the "DefaultIllum" export option.¹

7.10 3DMF (Apple) Import

Using the "mfio" plugin scenes may be imported from the 3DMF format (QuickDraw 3D Metafile) from Apple with the help of a free 3DMF parser created by Duet Development Corp. and distributed by Apple. Start importing a 3DMF file using the menu entry "File/Import/Apple 3DMF" (if this menu entry is not available, the mfio plugin must be loaded using the menu entry "File/Load Plugin" first).

The mfio plugin only supports the 3DMF version 1.0!

7.10.1 3DMF Primitive and Attribute Support

The mfio plugin supports import of the following geometric primitives:

- Polyline, Triangle, TriGrid, Polygon, general Polygon, Box,
- NURBS curve, NURBS surface (with trim curves),
- Ellipsoid, Cylinder, Cone, Disk, and Torus.

The following table comprehensively lists the supported 3DMF primitives and their Ayam counterparts that will be created upon import.

3DMF Primitive	Avam Object
Polyline	NCurve
Triangle	PolyMesh
TriGrid	PolyMesh
Polygon	PolyMesh
Box	Box
Ellipsoid	Sphere
Cylinder	Cylinder
Cone	Cone
Disk	Disk
Torus	Torus
NURBCurve	NCurve
NURBCurve2D	NCurve
NURBPatch	NPatch
Container	Level

Table 118: 3DMF (Apple) Import Conversion Table

¹ Since 1.35.

The following transformations are supported in 3DMF import:

- Scale,
- Translate,
- Rotate, RotateQuaternion, RotateAxis (if axis is X, Y, or Z).

Furthermore, the import plugin reads the structure of the scene from Container objects. Reference objects will be resolved to normal objects while importing. Instances may be easily created again using Automatic Instancing (see section [8.10 Automatic Instancing](#) (page 531)).

Support for import of lights, camera attributes as well as material attributes other than material color and opacity is currently not available.

7.10.2 3DMF Import Options

The following parameters, additionally, control the 3DMF import:

- "FileName": is the filename of the 3DMF file to import;
- "ScaleFactor": The "ScaleFactor" option allows to apply a scaling factor to all imported objects.
- "ReadCurves": If the "ReadCurves": import option is switched off, no curves will be imported. This option does *not* influence the import of trim curves.
- "ReadSTrim" if switched off, no simple trims (trims that only run along the border of a NURBS patch and actually do not trim the surface) will be imported if they are the only trim curves.
- "RescaleKnots": allows to rescale the knot vectors of NURBS curves, patches, and trim curves so that the distances between different knots are not smaller than the given value. A "RescaleKnots" value of 1.0e-04 leads to NURBS that may be safely drawn using GLU. The default value 0.0 means no scaling.

7.11 3DMF (Apple) Export

Using the "mfio" plugin scenes may be exported to the 3DMF format (QuickDraw 3D Metafile) from Apple with the help of a free 3DMF parser created by Duet Development Corp. and distributed by Apple. Start exporting to a 3DMF file using the menu entry "File/Export/Apple 3DMF" (if this menu entry is not available, the mfio plugin must be loaded using the menu entry "File/Load Plugin" first).

The mfio plugin only supports the 3DMF version 1.0!

7.11.1 Ayam Object and Properties Support

The mfio export supports the following geometric objects:

- NURBS curve and NURBS surface (with trim curves) including all NURBS curve/surface providing objects, such as ICurve, Skin etc.,
- Sphere, Disk, Cone, Cylinder, Torus,
- Box, and PolyMesh.

Objects of types not listed here will be converted to NURBS (if possible) or to PolyMesh objects automatically for export.

All transformations are supported and will be written as Translate, Rotate, and Scale transformations, respectively.

All Instance objects will be resolved for export. Level objects (regardless of type) will be written as Container objects.

If an object has a material, the color and opacity of the material will be written as DiffuseColor and TransparencyColor, if the respective red color component has a value different from -1.

Support for export of lights, camera attributes as well as material attributes other than material color and opacity is currently not available.

7.11.2 Trim Curves Support

The 3DMF file format specification for version 1.0 is unfortunately very terse, when it comes to trim curves. There is no clearly defined way of specifying them in conjunction with their respective NURBS surface. Furthermore, the method presented here is the only way that works with the free provided 3DMF parser. Trimmed NURBS patches will be written by Ayam like this:

```
Container (
    NURBPatch ( ...
    )
    [transformations & attributes of NURBS patch]
    TrimCurves ( )
    NURBCurve2D ( ...
    )
    Container (
        NURBCurve2D ( ...
        )
        NURBCurve2D ( ...
        )
    )
)
```

1. There will always be an enclosing `Container` for a NURBS patch.
2. If the patch is trimmed, after the `NURBPatch` or potentially present transformations and attributes of the patch, a `TrimCurves` element will follow (which does *not* contain the trim curves but is empty).
3. The trim curves follow now as 2D NURBS curves (`NURBCurve2D` objects for simple trims) or `Container` objects (for trim loops) with multiple 2D NURBS curves until the end of the enclosing container.
4. The transformation attributes of the trim curves will be applied to the NURBS curve control points for export (there will be no transformations or attributes for the trim curve elements).

The Ayam 3DMF import expects the trim curves to be delivered in this manner and can then indeed read them back accurately.

7.11.3 3DMF Export Options

The following parameters, additionally, control the 3DMF export:

- "FileName": is the filename of the 3DMF file to export;
- "WriteBinary": This option controls whether the text version or the binary version of the 3DMF file format should be used for export.
- "ScaleFactor": This option allows to apply a scaling factor to all exported objects.
- "WriteSelected": exports only the selected objects.
- "WriteCurves": If this option is disabled, no curves will be written to the exported 3DMF file. This option does *not* influence the export of trim curves.

7.12 3DM (Rhino) Import

Since version 1.8.2 Ayam contains a plugin named "onio" that may import scenes from the Rhino 3DM file format using the OpenNURBS toolkit (hence the name of the plugin onio – **OpenNURBS IO**) by Robert McNeel & Associates.

Start importing from a Rhino 3DM file using the menu entry "File/Import/Rhino 3DM" (if this menu entry is not available, the onio plugin must be loaded using the menu entry "File/Load Plugin" first).

The onio plugin only supports import of 3DM files of version 3.0 and earlier.

7.12.1 3DM Object Support

The import functionality of the onio plugin currently covers import of all NURBS and BRep objects and objects that may be converted to NURBS with routines from the OpenNURBS toolkit (those objects are: PolylineCurve, PolyCurve, LineCurve, ArcCurve, CurveOnSurface, RevSurface, SumSurface, and PlaneSurface). References will be resolved. Names will be imported, but converted to an ASCII representation. Since Ayam 1.8.3 also Mesh objects will be imported to PolyMesh objects, texture coordinates will be read and appropriate PV tags will be created for them.

The following table comprehensively lists the supported Rhino 3DM primitives and their Ayam counterparts that will be created upon import.

7.12.2 3DM Import Options

The 3DM import process is controlled by the following options:

- "ScaleFactor": This option allows to apply a scaling factor to all imported objects.
- "Accuracy": This option controls the tolerance of OpenNURBS internal operations, in this case the value is mostly used for conversion operations to the NURBS form.
- "ReadCurves": If this option is switched off, no curves will be imported. This option does *not* influence the import of trim curves.
- "ReadLayers": Using this import option, a single layer or a range of layers may be selected for import. By default, all objects from all layers will be imported.

Rhino 3DM Primitive	Ayam Object
PolyLineCurve	NCurve
PolyCurve	NCurve
LineCurve	NCurve
ArcCurve	NCurve
CurveOnSurface	NCurve
Mesh	PolyMesh
NurbsCurve	NCurve
NurbsSurface	NPatch
RevSurface	NPatch
SumSurface	NPatch
PlaneSurface	NPatch

Table 119: 3DM (Rhino) Import Conversion Table

- **"ReadSTrim"**: This option helps to ignore single bounding trim loops of NURBS surfaces. Importing this single bounding trim loop would just make the Ayam scene more complex than needed in many cases. If **"ReadSTrim"** is switched off, no simple trims (trims that only run along the border of a NURBS patch and actually do not trim the surface) will be imported if they are the only trim curves.
This option replaces the **"IgnoreFirstTrim"** import option available before Ayam 1.13 with slightly different semantics.
- **"RescaleKnots"**: allows to rescale the knot vectors of NURBS curves, patches, and trim curves so that the distances between different knots are not smaller than the given value. A **"RescaleKnots"** value of 1.0e-04 leads to NURBS that may be safely drawn using GLU. The default value 0.0 means no scaling. Since Ayam 1.13 also eventually present trim curves will be scaled properly to the new knot ranges of NURBS patches.
- **"Progress"**: displays the progress of the import; from 0 to 50 percent, OpenNURBS is reading the 3DM file into memory; from 50 to 100 percent, the onio plugin is converting the 3DM objects to Ayam objects.

7.13 3DM (Rhino) Export

Since version 1.8.2 Ayam contains a plugin named **"onio"** that exports scenes to the Rhino 3DM file format using the OpenNURBS toolkit (hence the name of the plugin onio – **OpenNURBS IO**) by Robert McNeel & Associates.

Start exporting to a Rhino 3DM file using the menu entry **"File/Export/Rhino 3DM"** (if this menu entry is not available, the onio plugin must be loaded using the menu entry **"File/Load Plugin"** first).

The onio plugin exports 3DM files of version 3.0.

7.13.1 Ayam Object and Properties Support

The export functionality of the onio plugin currently covers export of all boxes, quadrics, NURBS, poly-meshes, instances, clones, script objects (of type **"Create"** or **"Modify"**) and objects that may be converted to NURBS curves or surfaces.

Even though export of planar cap surfaces of various tool objects is fully supported, the export of general trimmed NURBS patches is not well supported. This is because of a missing feature (pushing up 2D trim curves to 3D curves for arbitrary NURBS surfaces) in the OpenNURBS toolkit. A coarse polygonal 3D representation of the 2D trim curves will be created automatically, so that general trimmed NURBS patches may be exported, albeit with lower quality and bigger file size as would be necessary.¹

UMM/VMM tags are used to trim the knot vectors of exported NURBS objects.² See also section 4.11.17 UMM/VMM (U/V Min Max) Tag (page 269).

Since the Rhino 3DM file format does not support hierarchy and transformation attributes per object, the hierarchy of the Ayam scene will be squashed and all transformation attributes will be applied to the control points of the objects for export. CSG operations are fully ignored, all objects will be written as if combined by the union operator. Furthermore, all instance objects will be resolved to normal objects.

All objects will be written to the first layer, the default layer (unless the "TopLevelLayers" option is used). Object names will be exported as well. Names of level objects will be prepended to the names of their child objects. The object hierarchy:

```
+--Arm(Level)
  |--MySphere(Sphere)
  \--MyCylinder(Cylinder)
```

for instance, leads to two objects in the Rhino file named "Arm/MySphere" and "Arm/MyCylinder".

7.13.2 3DM Export Options

The 3DM export process is controlled by the following options:

- "ScaleFactor": The "ScaleFactor" option allows to apply a scaling factor to all exported objects.
- "Accuracy": The "Accuracy" option controls the tolerance of internal OpenNURBS operations (currently those are: pushing up 2D trim curves to 3D curves and checking NURBS surfaces for planarity).
- "WriteSelected": exports only the selected objects.
- "ObeyNoExport": ignores all objects with "NoExport" tags.
- "IgnoreHidden": ignores all hidden objects.
- "WriteCurves": If this option is disabled, no curves will be written to the exported Rhino 3DM file. This option does *not* influence the export of trim curves.
- "QuadAsBRep": If this option is enabled spheres, cylinders, cones, and torii will not be exported as collection of NURBS surfaces (as converted by Ayam) but as BRep objects (as converted by the OpenNURBS toolkit). However, not all features of the quadric objects will be translated in this case:
 - The BRep sphere does not support ZMin, ZMax, and ThetaMax.
 - The BRep cylinder does not support ThetaMax (base caps will be created if the cylinder is closed).
 - The BRep cone does not support ThetaMax (a base cap will be created, if the cone is closed).
 - The BRep torus does not support PhiMin, PhiMax, and ThetaMax.

¹ Since 1.9. ² Since 1.9.

The "QuadAsBRep" option has no effect on the export of disks, hyperboloids, and paraboloids. Those will always be exported as NURBS surfaces.

- "TopLevelLayers": controls whether the top level Level objects in the Ayam scene to be exported should be interpreted as layers. If this option is enabled, all objects in these levels will be placed on the respective layer. Objects that are not in one of those levels will be written to the default layer. Furthermore, top level object names will become layer names.
- "Progress": displays the progress of the export; from 0 to 50 percent, the onio plugin is converting the Ayam objects to 3DM objects; from 50 to 100 percent OpenNURBS is writing the 3DM file.

7.14 X3D (Web3D) Import

Since version 1.13 Ayam provides a plugin named "x3dio" that may import scenes from the XML based X3D file format published by the Web3D Consortium. The XML parser used in this plugin is based on Expat and SCEW.

Binary and compressed versions of X3D, as well as VRML files are *not* supported. Only pure XML files are read by the x3dio plugin.

Start importing from a Web3D X3D file using the menu entry "File/Import/Web3D X3D" (if this menu entry is not available, the x3dio plugin must be loaded using the menu entry "File/Load Plugin" first).

7.14.1 X3D Element Support

The import functionality of the x3dio plugin currently covers import of the following X3D elements (grouped by components):

- **Geometry3D:** Box, Sphere, Cylinder, Cone, ElevationGrid, Extrusion, IndexedFaceSet, IndexedTriangleSet, IndexedTriangleStripSet, IndexedTriangleFanSet, TriangleSet, TriangleStripSet, TriangleFanSet, IndexedLineSet, and LineSet.

Cylinders with just one cap are imported as two objects (a Cylinder and a Disk). In all other cases and also for Cones, the "Closed" attribute of the Ayam object is set according to the cap information of the X3D element.

ElevationGrids are imported as bilinear patch meshes.

- **Geometry2D:** Arc2D, ArcClosed2D, Circle2D, Disk2D, Polyline2D.

Arcs and Circles are imported as NCircle objects. Closed arcs and Polylines are imported as NURBS curves. Disks with an inner radius > 0.0 are imported as flat Hyperboloids (otherwise as Disks).

- **NURBS:** NurbsCurve, NurbsCurve2D, NurbsPatchsurface, NurbsTrimmedSurface, Contour2D, ContourPolyline2D, NurbsSweptSurface, NurbsSwungSurface, NurbsSet.

All NURBS elements are fully supported.

- **CAD:** QuadSet, IndexedQuadSet, CADLayer, CADAssembly, CADPart, CADFace.

CADLayer objects will be imported as top level Level objects. CADAssembly and CADPart objects will be imported as level objects.

- **Light sources:** DirectionalLight, PointLight, SpotLight.

The lights will be directly mapped to the standard RenderMan light sources distant, point, and spot, respectively. Therefore, the "radius" and the "attenuation" attributes of point and spotlights are not supported. However, point and spotlights still have a quadratic falloff with distance.

- **Navigation:** Viewpoint.

Viewpoint elements will be imported as view objects (with corresponding view window) or camera objects depending on the "ReadViewpoints" import option.

- **Non geometric / Scene structure:** Transformation, Shape, Group, StaticGroup, Inline.

Shear transformations are not supported.

The semantics for inlining are currently not fully standards compliant. By default, DEFs in inlined files live in their own namespace. It is not possible to USE a DEF from an inlined file in the inlining file. However, if the "MergeInlineDefs" import option is switched on, *all* DEF namespaces (of inlining and inlined files) will be merged into one big namespace. Now it would be possible to USE a DEF from an inlined file in the inlining file. But beware, this only works correctly, if the DEF names in all the files are unique. It is not possible to transfer single definitions from an inlined file to the inlining file or from the inlining file to the inlined file.

Also note: Inline URLs that do not point to the file system are not supported.

Unsupported X3D elements will be silently ignored. Prototyping and dynamic scenes as well as scripts are not supported.

The following table comprehensively lists the supported Web3D X3D primitives and their Ayam counterparts that will be created upon import.

X3D Primitive	Ayam Object	X3D Primitive	Ayam Object
<i>Geometry3D:</i>		<i>CAD:</i>	
Box	Box	QuadSet	PolyMesh
Sphere	Sphere	IndexedQuadSet	PolyMesh
Cylinder	Cylinder	CADLayer	Level
Cone	Cone	CADAssembly	Level
ElevationGrid	PaMesh	CADPart	Level
Extrusion	PolyMesh	CADFace	Level
IndexedFaceSet	PolyMesh	<i>Light:</i>	
IndexedTriangleSet	PolyMesh	DirectionalLight	Light
IndexedTriangleStripSet	PolyMesh	SpotLight	Light
IndexedTriangleFanSet	PolyMesh	PointLight	Light
TriangleSet	PolyMesh		
TriangleStripSet	PolyMesh		
TriangleFanSet	PolyMesh		
IndexedLineSet	NCurve		
LineSet	NCurve		
<i>Geometry2D:</i>			
Arc2D	NCircle		
ArcClosed2D	NCurve		
Circle2D	NCircle		
Polyline2D	NCurve		
Disk2D	Disk / Hyperboloid		
<i>NURBS:</i>			
NurbsCurve	NCurve		
NurbsCurve2D	NCurve		
NurbsPatchSurface	NPatch		
NurbsTrimmedSurface	NPatch		
NurbsSweptSurface	Sweep		
NurbsSwungSurface	Swing		

Table 120: X3D (Web3D) Import Conversion Table

7.14.2 X3D Attribute Support

The "solid", "ccw", and "convex" attributes are always ignored.

The "DEF" and "USE" attributes are supported for all elements, however, the corresponding master-instance relationships are completely resolved upon import. Instances of objects may be easily created again using Automatic Instancing (please refer to section 8.10 Automatic Instancing (page 531)).

Normals will currently not be generated automatically for any X3D element and thus the "creaseAngle" attribute will be fully ignored. Note however, that if normals are specified explicitly, e.g. for an "IndexedFaceSet" element, they will be imported correctly.

Furthermore, if normals, colors, or texture coordinates are provided to an element that imports as a PolyMesh (IndexedFaceSet and the likes) proper PV (primitive variable) tags will be created.¹

Moreover, if there is an index provided to the normals, colors, or texture coordinates, the relevant affected data (e.g. the vertex coordinates) will be expanded properly (as RenderMan does not support multiple different indices on the vertex related data). Note that currently the mere presence of an index will lead to this expansion. The potential normal, color, or texture coordinate index is not checked for, whether it is identical to the vertex coordinate index (and thus no expansion would be necessary in the first place).

The "DEF" attributes will be converted to object names in some cases.

7.14.3 X3D Import Options

The following options further control the X3D import process:

- "FileName": is the name of the X3D file to be imported
- "ReadCurves": if this is disabled, no freeform curves will be imported. This option does *not* influence the import of trim curves.
- "ReadViewpoints": controls whether viewpoints should be read as view, camera, or not at all.
- "ReadSTrim": if switched off, no simple trims (trims that only run along the border of a NURBS patch and actually do not trim the surface) will be imported if they are the only trim curves.
- "RescaleKnots": allows to rescale the knot vectors of NURBS curves, patches, and trim curves so that the distances between different knots are not smaller than the given value. A "RescaleKnots" value of 1.0e-04 leads to NURBS that may be safely drawn using GLU. The default value 0.0 means no scaling.
- "ScaleFactor": allows to apply a scaling factor to all imported objects.
- "RationalStyle": determines how rational coordinates are stored in the file to be imported (see also section 1.2.4 Rational Style (page 17));
- "Progress": displays the progress of the import; from 0 to 50 percent, the x3dio plugin is reading the XML file, from 50 to 100 percent the x3dio plugin is creating Ayam objects.

¹ Since 1.17.

7.15 X3D (Web3D) Export

Since version 1.13 Ayam provides a plugin named "x3dio" that exports scenes to the XML based X3D file format published by the Web3D Consortium. The XML parser used in this plugin is based on Expat and SCEW.

Start exporting to a X3D file using the menu entry "File/Export/Web3D (X3D) " (if this menu entry is not available, the x3dio plugin must be loaded using the menu entry "File/Load Plugin" first).

7.15.1 Ayam Object and Properties Support

The export functionality of the x3dio plugin currently covers export of all boxes, quadrics, NURBS, Poly-Meshes, instances, clones, script objects (of type "Create" or "Modify") and objects that may be converted to NURBS curves or surfaces or to PolyMeshes (e.g. SDMesh objects).

Some NURBS tool objects can be exported as parametric NURBS nodes, e.g. Sweep objects may be exported as NurbsSweptSurface nodes.

The scene hierarchy (level objects) will be converted to a matching transform node hierarchy.

CSG operations are fully ignored, all objects will be written as if combined by the union operator.

Texture coordinates from primitive variable tags will be exported. Also TC tags are supported, but PV tags take precedence.¹

Clamping the knot ranges of NURBS via UMM/VMM tags is not supported.

NURBS surfaces support XML tags to set "uTessellation" and/or "vTessellation" attributes.

PolyMesh objects will be exported to IndexedFaceSet nodes. PolyMesh faces with holes are automatically tessellated. PolyMesh faces with more than three points are tessellated if the export option "TessPoMesh" is used. All tessellated faces will be exported to a second IndexedFaceSet element.

Light sources are exported if they are point, distant, or spot lights.

Cameras and Views are exported as Viewpoint nodes. Note however, that X3D always assumes a perspective viewing transformation. This means, views of type "Front", "Side", or "Top" will not be exported properly.

Material objects are supported in a very minimalist way, i.e. only the material color will be exported as "diffuseColor" attribute. However, XML tags can be used to add more attributes or even shaders to the exported materials, see also section 4.11.23 XML tag (page 271).

Object names will be converted to DEF attributes.

Instances can be resolved or exported as USE/DEF pairs.

7.15.2 Wire-frame Support

The X3D export allows to export wire-frames instead of surfaces for NPatch and PoMesh objects.

To switch an object to wire-frame export, just add a tag "AsWire" to the object.

¹ Since 1.35.

7.15.3 X3D Export Options

The following parameters, additionally, control the X3D (Web3D) export:

- "FileName": is the filename of the X3D file;
- "ScaleFactor": allows to apply a scaling factor to all exported objects (this will be realized by an extra Transform-node in the scene hierarchy);
- "WriteSelected": if enabled, only the currently selected object(s) will be exported;
- "ObeyNoExport": toggles export of objects with "NoExport" tags;
- "IgnoreHidden": toggles export of hidden objects;
- "WriteCurves": toggles writing of NURBS curves and NURBS curve providing objects to the exported file (This option does *not* influence the export of trim curves.);
- "WriteViews": controls whether views should be exported as view points;
- "WriteParametrics": toggles writing of the following tool objects as plain NURBS surface or the following higher level primitives:
 - Revolve as NurbsSwungSurface,
 - Swing as NurbsSwungSurface,
 - Sweep as NurbsSweptSurface,
 - Extrude as NurbsSweptSurface;
- "WriteMaterials": controls whether appearance and material nodes should be exported for each shape;
- "ResolveInstances": controls whether instance objects should be resolved to normal objects or exported as DEF/USE pairs. Note, that no attempt is made to detect whether the master objects (the DEFs) will be exported for all exported instances (USE); this may happen especially in conjunction with the X3D export options "WriteSelected" or "IgnoreHidden", incomplete/erroneous X3D files may result unless the "ResolveInstances" option is switched on;
- "TopLevelLayers": arranges for all top level objects in the Ayam scene to be exported as CADLayer-nodes, instead of Transform-nodes;
- "RationalStyle": determines how to write rational coordinates (see also section [1.2.4 Rational Style \(page 17\)](#));
- "WriteX3dom": toggles X3DOM-export mode. If this option is enabled, the X3D export will
 1. first read a XHTML template file from the export target directory (named "x3dom-template.xhtml"),
 2. then inject the scene to be exported into the X3D "<Scene>"-tag of the template (already present child elements of this tag will be preserved),
 3. add width and height of the view window that was exported last as attributes to the "<X3D>"-tag,
 4. and finally write the XHTML document to the file specified via the "FileName" export option.
- "Progress": displays the progress of the export; from 0 to 50 percent, the x3dio plugin is creating a XML document in memory and from 50 to 100 percent this XML document will be written to the X3D file.

8 Miscellaneous

This section contains all information about Ayam not fitting into the other main sections.

8.1 The Undo System

With the help of the undo system mistakes made while modelling may be corrected.

Note, that only modifications of objects can be undone. This includes changes made by interactive modelling actions, changes made using property GUIs, but also changes to views (type changes or changes to the camera settings associated with a view, unless explicitly disabled using the view attribute "EnableUndo"). It is currently *not* possible to undo any changes to the object hierarchy, including clipboard (e.g. cut, paste) and drag and drop operations. If an object is deleted, it is gone! If an object is, accidentally, moved using drag and drop, undo will *not* help.

The undo system works by storing copies of the different states of changed objects in an undo buffer. It is possible to step backwards through the saved states using <Ctrl+z> (undo) but also forward using <Ctrl+y> (redo).

The storage space occupied by the undo buffer may be adjusted using the preferences option "Modelling/UndoLevels". A value of 0 completely disables the undo system.¹ The value 1 means that there is always one state of the scene that can be restored, plus, a potential undo operation can also always be undone using redo.

The changes that would be undone or redone are shown in abbreviated form in the default prompt of the Ayam console and also in the main menu entries "Edit/Undo" and "Edit/Redo".²

Several actions will completely clear the undo buffer (i.e. no undo is possible after one of those actions): New Scene, Open (Replace) Scene, and Close View.

Furthermore, undo/redo operations will also modify objects that reside in the object clipboard (if they have saved states in the undo buffer). This means that the following sequence of operations leads to a sphere placed at "0, 0, 0":

```
create Sphere (at 0,0,0)
move Sphere (to 1,1,0)
cut Sphere (to clipboard)
undo
paste Sphere (from clipboard)
```

8.2 The Modelling Concept Tool-Objects

This section introduces the modelling concept *Tool-Objects*, as used in Ayam.

In a standard modelling application, to create a surface of revolution, one would either first create a curve then call the revolve tool to get an appropriate surface (losing the curve as object, or even keeping it, but without relation to the surface), or call the revolve tool first, which would then require the user to draw a curve with similar results: the created surface will typically lose the relation to the surface of revolution tool and the curve (even if kept intact) will lose the relation to the surface. There is no easy way to change

¹ Since 1.21. ² Since 1.14.

parameters for the surface creation or to change the geometry of the curve later on without doing it all over again.

The modelling concept Tool-Objects overcomes these drawbacks by transforming the revolve *tool* into a scene *object*.

The following example hierarchy shows two objects in a parent-child relation:

```

+-Tool_Object (Revolve)
|               ^
|               :
|   <Notification>
|               :
\ -Parameter_Object (NCurve)

```

The parent object is called *Tool-Object* and the child object is called *Parameter-Object*. There is a flow of information from the parameter object to the tool object. This information flow is controlled by the so called *Notification* mechanism. The notification mechanism makes sure that whenever the parameter objects change, the tool object is informed so that it may adapt to the changes. For interactive modelling actions, the notification will be carried out while dragging the mouse or after release of the mouse button (i.e. after the modelling action finished), depending on the main preference setting "LazyNotify".

In the example above, a NURBS curve is the parameter object and the tool object is creating a surface of revolution from the curve data. The NURBS curve and parameters of the Revolve object may change at any time. When saved to an Ayam scene file, no surface data will be written, leading to very small files that additionally contain a modelling history and capture design intent to a certain degree.

Tool objects may be parameter objects of other tool objects:

```

+-Tool_Object (ExtrNP)
  +-Parameter_and_Tool_Object (Revolve)
    \ -Parameter_Object (NCurve)

```

and there may be more than one parameter object per tool object:

```

+-Tool_Object (Skin)
  |-Parameter_Object_1 (NCurve)
  |-Parameter_Object_2 (NCurve)
  | ...
  \ -Parameter_Object_n (NCurve)

```

Tool objects create new geometric objects from the information delivered by the parameter object(s) or they modify the parameter object(s) and deliver them to their respective parent object.

The pure hierarchical dependency scheme may be broken up by instance objects:

```

+-Tool_Object (Revolve)
  \ -Parameter_Object (NCurve) ----- .
+-Tool_Object (Revolve)                | !
  \ -Parameter_Object (Instance_of_NCurve) <- '

```


In the scene above, the second Revolve tool object depends on the shape of the first parameter object. The Instance object transports the data from one part of the hierarchy to another. The notification mechanism is aware of this and initiates updates in the scene hierarchy wherever needed and efficiently (not updating any objects twice) according to the "CompleteNotify" main preference setting. In the example above, due to complete notification, the second Revolve object will get updated automatically whenever the original NCurve object changes.

Two other mechanisms exist, that help tool objects to get the information they need and to increase flexibility in hierarchy building and modelling: *Provision* and *Conversion*. Both mechanisms are quite similar and convert objects temporarily/transparently (provision) or finally (conversion) from one type to another, e.g. they convert an ICurve (interpolating curve) to a plain NCurve (NURBS curve).

Due to the provision mechanism, an ICurve object could be used instead of an NCurve as parameter object in all examples above easily. And with the help of the conversion mechanism, the Revolve tool objects could be converted to plain NPatch objects (e.g. for modelling operations not available to Revolve objects).

Note that instance objects are subject to a second round of provision, i.e. the master does not need to be of the wanted type but rather provide the wanted type.

Even though, theoretically, every tool could be implemented as tool object, this has not been done in Ayam (mainly, because this would unnecessarily increase the code base). Only the most often used tools that convey and capture much design intent were implemented as tool objects (those are surface or curve creating tools). But also seldom used tools can be elevated to nearly full tool object capabilities through the employment of the concept of scripting objects (see section 4.9.1 Script Object (page 229)).

This can be done easily by e.g. creating Script objects of type "Modify" that call those tools from their script (after possible conversion of the provided object(s) to a proper type, the tool may need to operate on). Even property GUIs to let the user adjust tool parameters as they know it from the other Ayam objects can be added and different tools can be combined in single objects with normal script code for unmatched flexibility. But let us see a simple example first:

```
+-Skin
+-Script
  \-ExtrNC
```

In the scene hierarchy above, the Script object could be simply reverting the extracted curve with code like this:

```
convOb -inplace; revertC
```

effectively elevating the tool "revertC" to an object.

A more useful example can be found in the Marsrakete sample scene (as available from the Ayam home page). Here, a curve extracted from a patch is trimmed to the right length using a script like this:

```
convOb -inplace; trimNC 0.0 0.5
```

. See section 4.9.1 Script Object Example (page 235) for information on how this script might be expanded to support a GUI and more error checking.

8.3 Scene File Management

This section contains exhaustive information on what exactly happens, when Ayam reads or writes a scene file.

8.3.1 Opening Scene Files

When reading a scene using "File/Open" or the MRU (most recently used) list:

1. Ayam will first check the scene changed state of the currently loaded scene and warn the user, if the current scene contains unsaved changes,
2. then Ayam will clear the undo buffer and the current scene (the clipboard content is not touched, except for instance objects, whose masters are cleared with the scene: those instance objects will be deleted from the clipboard),
3. if the new file appears not to be an Ayam scene file (judged solely by the file name extension) Ayam will try to import the file using an import plugin responsible for the file (automatically loading the matching plugin if not already loaded),
4. Ayam will change the application working directory to the directory of the scene file,
5. now, a backup copy of the file to read will be made (depending on the preference setting "Main/BakOnReplace"), a potentially existing old backup copy will be silently overwritten, note: even if the backup fails, scene reading continues,
6. the header of the scene file will be read to derive the scene file format version,
7. the file will be read with all objects, *
8. after reading of all objects, all instances will be connected to their masters using the information stored in "OI" tags; if a master can not be found in the scene, the respective instance object(s) will be removed,
9. all objects will be connected to their materials using the information stored in "MI" tags; if no matching material can be found, the connection will not be established,
10. a complete notification will be run,
11. if the file contains a Root object with "SaveMainGeom" and/or "SavePaneLayout" tags, the respective window/widget geometries from the tags will be re-established,
12. if no errors occurred during reading, the current scene name will be set to that of the file, otherwise the current scene name will be reset to "unnamed" (to avoid clobbering good scene files that failed to load for some reason with a single, maybe even unintentional, press of <Ctrl+s>),
13. the file name will be put to the first most recently used files entry in the file menu,
14. the scene changed state will be reset to "unchanged",
15. if a Root object was read, a save views flag will be set, otherwise it will be cleared and following save operations will not save the root and any views (i.e. scene files without root and views stay scene files without root and views, even if new views are opened and parameterised).

* When reading objects from a file:

1. if a view object is read, a new view window will be opened (except for the first three view objects in single window GUI mode, where only the configuration of the view objects read will be copied to the already existing view objects of the three internal views),

2. if an object is of a currently undefined type (i.e. defined by a plugin), Ayam will derive a potential plugin name from the object type name and attempt to load the plugin, then scene reading continues,
3. if a material object is read, Ayam will immediately register it; if this registration fails (because there is already a material with that name registered; this material can *only* be in the clipboard, as all other objects were deleted before opening the file), the new material object will be renamed by appending a number and the registration is attempted again, those steps will be repeated until successful (prior to version 1.21 Ayam did not attempt to rename the material and other objects were then connected to the already existing material),
4. if loading of an object fails, Ayam skips to the next object and continues to read from the file.

8.3.2 Inserting Scene Files

In contrast to reading scenes via "File/Open" or the MRU list, reading scenes via "File/Insert" does some things differently:

1. the current scene will not be cleared before reading the file,
2. the current level will not be reset before reading the file; if the scene file to be inserted contains no root and no views, the new objects will be created in the current level, otherwise the objects will be created in the topmost level of the scene, which will also be the new current level in any case (after reading),
3. no backup copy will be made of the file to be inserted,
4. if a material is missing from the file, and a matching (by name) material exists in the scene, the objects will be connected to this material,
5. view windows will be opened for every view object read (the internal views will not be changed),
6. the current directory will only be changed during the file read operation,
7. the current file name will not be changed,
8. the scene changed state will be set to "changed",
9. the save views flag will not be changed, this means that inserting a scene file without root and views into an empty scene (e.g. directly after application startup or after "File/New") and then saving the scene, will enrich this scene file with root and views (in contrast to loading this scene via "File/Open").

8.3.3 Saving Scene Files

When saving a scene using "File/Save":

1. Ayam will first check the current scene file name, if the name is "unnamed", a new file name will be requested,
2. if the file to save to appears not to be an Ayam scene file (judged solely by the file name extension) Ayam will try to export the current scene using an export plugin responsible for the file type instead of saving to an Ayam scene file (automatically loading the matching plugin if not already loaded),
3. the geometry of the main window and internal widgets is saved to "SaveMainGeom" and/or "SavePaneLayout" tags (if present in the Root object),

4. "OI" tags and "MI" tags will be created to allow instances to be connected to their respective masters and objects to their materials, when reading the scene file again,
5. then, all objects from the scene will be saved to the file; if the save views flag was cleared while reading the previous scene file, root and views will be omitted (i.e. scene files without root and views stay scene files without root and views, even if new views were opened and parameterised while the scene was loaded),
6. if no errors occurred during writing of the scene file, the file name will be put to the first most recently used files entry in the file menu, and the scene changed state will be set to "unchanged",
7. if errors occurred, the current scene changed state will be kept.

8.4 Ayamrc File

To customize Ayam beyond the capabilities of the preferences dialog, the *ayamrc* file may be used. This file is either pointed to by the environment variable `AYAMRC` or is determined as following:

- On Unix it is "`~/ .ayamrc`", where "`~`" denotes the home directory of the current user.
- On the Win32 platform (Windows95 - XP) it is "`$(HOME) / ayamrc`" if the environment variable `HOME` exists, else "`$(TEMP) / ayamrc`".
- On Mac OS X Aqua (not X11!) it is "`~/Library/Preferences/ .ayamrc`", where "`~`" denotes the home directory of the current user.
- On Mac OS X X11 (not Aqua!) it is "`~/ .ayamrc`", where "`~`" denotes the home directory of the current user.

The *ayamrc* file is read on each start of Ayam and saved again on exit (if the preference setting "`Main/AutoSavePrefs`" is enabled).

The *ayamrc* file contains:

1. preference settings (including some hidden settings that require just occasional tweaking and are not reachable using the GUI preference editor),
2. position and size of the main window and the toolbox window,
3. position and size of various dialog windows (if the preference setting "`SaveDialogGeom`" is set accordingly),
4. keyboard shortcuts to menu entries and modelling actions,
5. `RiOption` and `RiAttribute` databases.

You may edit the file with any text editor (while Ayam is *not* running), but keep in mind, that the file will be parsed by Tcl. Should you, for some reason, destroy your *ayamrc* file so that Ayam does not start correctly anymore you can always start Ayam with the command line option "`-failsafe`". When the application is left the next time, or the main menu entry "`File/Save Prefs`" is invoked, a correct *ayamrc* file will be created again. All preference settings will be reset to factory defaults and all your edits of the *ayamrc* file will be lost, however.

Another way to reset the *ayamrc* file is to simply delete the file manually or using the main menu entry "`Special/Reset Preferences`".

To reset single elements to factory defaults, just remove the corresponding lines from the *ayamrc* file.

Finally, resetting single preference settings without a text editor is also possible with the help of the scripting interface by manipulating the global "ayprefs" array. The following example leads to a reset of the tolerance preference setting to its factory default for the *next* start of Ayam.

```
»unset ayprefs(Tolerance)
```

After unsetting single elements of the preferences array, Ayam should be restarted.

8.4.1 Changing Keyboard Shortcuts

You may adapt the keyboard shortcuts used in the GUI to your special needs using the *ayamrc* file. Note that if you do that, the GUI (the menu entries and the "Show Shortcuts" window) will adapt to your changes but certainly neither this documentation, nor the reference card (unless recreated using the script "refcard.tcl"), nor the tutorials.

Ayam does *not* check for clashes in key bindings. This means, the last set binding for a key will be used.

On Unix, the output of the program "xev" and the manual page for the "bind" command of Tk provide helpful information about which strings may be used to describe key presses. You can also directly use the Ayam console to infer key names, just enter:

```
»toplevel .keytest; bind .keytest <Key> {puts %K}
```

into the Ayam console. Now you can activate the new top level window and type on your keyboard while the Ayam console prints the names of the keys.

For convenience, the special string "Ctrl" will be replaced by "Control" before a shortcut is handed to the bind command.

Example:

```
set aymainshortcuts(Prefs) {Ctrl-p}
```

sets the keyboard shortcut for opening of the preferences editor to <Ctrl+p>. See the *ayamrc* file itself for a complete listing of available shortcuts.

8.4.2 Hidden Preference Settings

The *ayamrc* file currently contains the following adjustable hidden preference settings:

- "AALineWidth", line width used for drawing the lines of unselected objects (blue lines in standard color configuration) when anti-aliasing is enabled. The default value is 1.3. Ayam is not checking, whether the specified value is supported by the OpenGL implementation used.
- "AAFudge", offset added to any line ends to accommodate for different OpenGL implementations wrt. anti-aliasing (default 1.0).
- "AASelLineWidth", line width used for drawing the lines of selected objects (white lines in standard color configuration) when anti-aliasing is enabled. The default value is 1.5 (a tad higher than "AALineWidth" above, to make the perceived line width equal compared to normal lines, in the case of drawing a selected curve over a unselected resulting surface). Ayam is not checking, whether the specified value is supported by the OpenGL implementation used.

- "AddViewParams" allows to add custom parameters to the view OpenGL widget creation, like e.g. "-stereo true". The default value is "" (empty string).
- "ALFileTypes", "ALPlugins" two lists that describe file name extensions and corresponding plugins that import and export files of the type designated by the file name extensions. The default values of these options are

```
{ ".rib" ".3dm" ".obj" ".3dmf" ".mop" ".dxf" ".x3d" }
```

and

```
{ "rrib" "onio" "objio" "mfio" "mopsi" "dx fio" "x3dio" }
```
- "AllowWarp": controls whether the mouse pointer should be moved to the new position of points snapped to the grid while editing (valid values: 0, 1; default: 1 – yes).
- "AskScriptDisable" controls the warning dialog that appears if scenes with Script objects or tags are loaded. The default value is 1 – yes, warn about Script objects and tags. Valid values are 0 and 1.
See also section 4.9.1 Safe Interpreter (page 233).
- "AUCommands", commands that will be run in the console when <Shift+Return> is used instead of <Return>. See also section 6.2.20 Updating the GUI (page 420). The default value is "uS;rV;", leading to a complete update of the object hierarchy, the property GUI, and all view windows.
See also section 2.1.3 The Console (page 27).
- "AutoCloseUITimeOut", time in ms to wait after the renderer signalled completion to avoid killing it prematurely, default: 1000 – wait one second.
- "AvoidPwlCurve", determines whether NURBS drawing and tessellation avoid the use of the GLU "PwlCurve"-function to specify polygonal trim curves. This function uses less memory and is faster than the alternative but can lead to a tessellation that is clearly too coarse for the desired tessellation quality (valid values: 0, 1; default: 1 – yes).
- "BackupExt": is the file name extension to be used for backup files. Default values are "~" for Unix and ".bak" for Win32.
- "Balloon": time in ms until the tooltip window appears (default: 1500 – 1.5s).
- "Cat": name of a program that can read from and write to a pipe (used by the Rendering GUI) (default: "cat") (a setting of "cat" will be automatically replaced by "cat.exe" on Win32).
- "ConsoleTakeFocus": can be used to exclude the console from focus traversal via <Tab> when set to 0. The default value is 1. Valid values are 0 and 1.
See also section 2.1.3 The Console (page 27).
- "ConsoleCursorEnd", if switched on, the first click into the console will move the cursor to the input prompt, ready for command input, instead of moving the cursor to the point of the click. This option is enabled by default. Valid values are 0 and 1.
- "ConvertTags", a comma separated list of tag names. These tags will be copied verbatim to converted and provided objects. If any of "BP" or "CP" are in this list, the conversion of tool objects with caps or bevels to NPatch objects does not generate multiple objects in a Level object but just one object with all caps and bevels¹. Default value: "TP, TC, BP, CP".
See also section 2.2 Tools Menu (page 36).

¹ Since 1.24.

- "CullFaces", enables culling/discarding of back facing polygons prior to display with OpenGL. This can improve the visual quality of shaded surfaces at their edges. Note however, that culling back faces can also disrupt the CSG preview using AyCSG. Default value: 0 – no. Valid values are 0 and 1.
- "CycleHiddenWire", determines whether the slow "HiddenWire" drawing mode is included in the list of modes that are cycled by keyboard (via <Ctrl+PageUp>/<Ctrl+PageDown>), default: 0 – no. Valid values are 0 and 1.
- "CyclePerspective", determines whether the "Perspective" view type is included in the list of types that are cycled by keyboard (via <PageUp>/<PageDown>), default: 0 – no. Valid values are 0 and 1.
- "DailyTips": a list of strings that appear as tips on startup in the console (default: large).
- "DisableFailedScripts": specifies whether scripts in Script objects or BNS/ANS tags should be disabled when they fail (default: 1 – yes). Valid values are 0 and 1.
- "DisplayPath": is the display search path to be used in the viewport rendering mode, it should point to the "fifodspy.so" or "fifodspy.dll" file, respectively. If set to an empty string, no "searchpath" option for "display" will be emitted and the path to the display driver should be configured by other means, e.g. in the renderer configuration.
- "DynamicMRU": controls whether loading a scene from one of the most recently used main menu entries shall change the MRU order (count as used). Valid values are 0 and 1, default is 0.
- "EFlush": time in ms between two flushes of the error message buffer (default: 2000 – 2s).
- "FDShowHidden": controls whether the standard file dialog on Unix shows hidden files (default 1 – yes). Valid values are 0 and 1.
- "FDShowHiddenBtn": controls whether the standard file dialog on Unix shows an extra button that toggles display of hidden files (default 0). Valid values are 0 and 1.
- "FixDialogTitlees", determines whether or not the titles of the message boxes that are normally displayed in the window frame, shall be prepended to the respective content of the message boxes. This may be necessary because on some systems the title string might be displayed in an unreadable font or not at all. On Mac OS X Aqua (not X11!) this option is enabled by default. On all other systems this option is disabled by default. Valid values are 0 and 1.
- "FixImageButton", enables a workaround for buttons (e.g. in the toolbox) that stay depressed when used. Valid values are 0 and 1. This option is disabled by default on all platforms.
- "FixX11Menu" enables a workaround for non-sticky menus on X11 (displaced menus do not stay open). This option is enabled by default and not used on the Win32 and Aqua platforms. Valid values are 0 and 1.
- "FlashObjects" enables flashing of objects while object picking. When enabled, this option needs a considerable amount of resources due to constant picking operations while the mouse pointer travels over the view window and also leads to a lot of visual noise. Therefore it is disabled by default. Valid values are 0 and 1.
- "FocusFollowsWheel" controls whether mouse wheel events in a view window that has not the keyboard input focus should move the focus to that window. Valid values are 0 and 1. This option is enabled by default but due to the nature of the focus wheel interaction only useful on X11 and Aqua.

- "HLAyamAPI": foreground color to use for the Ayam API (commands and procedures) when highlighting scripts in the Script object script editor. The color can be specified as named color or in the form #RRGGBB, where RR, GG, and BB are a numeric specification of the red, green, and blue intensities, respectively, of the color using hexadecimal digits. Default value: blue4.
- "HLCommands": foreground color for highlighted Tcl commands, default value: #904080.
- "HLComments": foreground color for highlighted comments, default value brown.
- "HLStrings": foreground color for highlighted strings, default value brown4.
- "HLVars": foreground color for highlighted variables, default value DarkGreen.
- "IconGamma": this setting may be used to adapt the contrast of all icons (in view menu and the toolbox) to your display gamma. If you are on a SGI it is recommended to set this to about "0.7". The default value "" (empty string) leads to no changes of any icon images.
- "KeepNTmpFiles": how many incarnations of the scene in RIB form (which actually may be split in more than one file due to e.g. instances) created when directly rendering from a view window should be kept on disk (default: 5)
- "KeepParamGUI": determines whether the intermediate parameter GUIs should be kept open after a parameter has been entered or rather disappear immediately. The default value is 0 – the GUIs disappear immediately. See also section 3.6 [Intermediate Parameter GUIs](#) (page 84).
- "KeepToolPreview": determines whether or not the editing of tool parameters should keep the current preview (and change the green tick mark to a question mark) or revert the preview completely. Valid values are 0 and 1. The default value is 1, the current preview is kept.
- "Kill": name of a program that kills other processes and accepts a process id as argument; used by the Rendering GUI to cancel rendering processes (default: Unix "kill", Win32 "w32kill" – an internal kill command that has been introduced in Ayam 1.4).
- "LineWidth", line width used for drawing the lines of unselected objects (blue lines in standard color configuration). The default value is 1.0. Ayam is not checking, whether the specified value is supported by the OpenGL implementation used.
See also "SelLineWidth", "AALineWidth", and "AASelLineWidth".
- "ListTypes" determines, whether the type of an object should be displayed in braces in the tree view or listbox. Valid values are 0 and 1. The default value is 1 – yes, list the types.
- "LoadEnv", controls whether the environment scene file should be read on each application start. Valid values are 0 and 1. The default value is 0 – no file will be read. Saving an environment scene file via the menu entry "Special/Save Environment" or setting a value different from an empty string ("") to the "EnvFile" preference setting will set this option to 1 automatically.
- "MarkHidden" determines, whether hidden objects should be marked (using a preceding exclamation mark) in the tree view or object listbox. Valid values are 0 and 1. The default value is 1 – yes, mark hidden objects.
- "MaxTagLen": the maximum number of characters to be displayed in the buttons in the Tag Property GUI (default: 30).

- "NewLoadsEnv", if this is switched on, Ayam will read the scene file specified by "EnvFile" when the scene is cleared using the main menu entry "File/New". Valid values are 0 and 1. The default value is 1 – yes, load the environment file on "File/New".
- "NormalizeDigits", determines the number of digits to the right of the decimal point, the normalizing processes should leave intact (default: 6).
- "NormalizeMark", controls whether the mark coordinate values should be normalized after setting the mark from the center of selected points (valid values: 0, 1; default: 1 – yes).
- "NormalizePoints", controls whether the coordinate values of points modified by interactive modelling actions should be normalized (valid values: 0, 1; default: 1 – yes).
- "NormalizeTrafos", controls whether the transformation attribute values should be normalized after interactive modelling actions (valid values: 0, 1; default: 1 – yes).
- "PanDist": the distance about which panning in views by keyboard occurs; positive values are absolute in pixels, negative values are relative to the current window dimension; a value of -10 means a tenth of the current window width/height. The default value is -10.
- "PaneMargins" is a list of currently five floating point values that are used as a safety margin for the panes when they are resized (1: console vs. hierarchy, 2: hierarchy vs. upper-views, 3: lower-view vs. property, 4: property vs. hierarchy, 5: upper-view-2 vs. upper-view-1). These values control the minimum size of a pane expressed in an inverse (1/x) and relative way: the smaller the number, the bigger the margin. The safety margin of the uppermost horizontal pane (that divides the upper internal views from the hierarchy and the third view) is e.g. a bit larger so that the main menu may not be obscured easily (the corresponding value is 5.0). The default value for the console (20.0) leads to a small margin, so that the console may be shrunk to 2 or even 1 lines of text. The default values are { 20.0 5.0 10.0 10.0 10.0 }
- "PickCycleMax": the maximum number of objects for pick cycling. If "PickCycle" is enabled and there are more candidates for a selection than specified by this option, the object selection listbox will take over regardless. The default value is 5.¹
- "PickTolerance": the tolerance used to determine whether an object should be picked or not (default: 5); this setting determines the size of a rectangular area around the picked point in pixels, all objects that are inside or touch this area are considered picked.
- "PolyOffset0", "PolyOffset1" two float values, that control the offsetting of shaded surfaces in the shade and draw drawing mode (so that the curves always appear on top of the surfaces). Default values are 1.0, 1.0.
- "Prompt": controls the prompt for the Ayam console. See also section 2.1.3 The Console (page 27). If this option is set to an empty string, a default of

```
\[Undo:$ay(undoo)/Redo:$ay(redoo)\]\[Repeat:$ay(repo)\].../[file
tail [pwd]]>
```

will be used, which displays the name of the operations that can be undone and redone, the tool to be repeated, and the last component of the current directory of Ayam like this:

```
[Undo:None/Redo:None] [Repeat:None] .../scn>
```

If this option would be set to "[pwd]>" the prompt would display the full path name of the current directory instead.

¹ Since 1.26.

To display the value of an arbitrary Tcl variable in the prompt (e.g. designating the current level in the scene hierarchy) a write trace must be bound to that variable. The write trace in turn must call the procedure "ayam_updateprompt" and may e.g. be established using a small script like this:

```
trace variable <vname> ayam_updateprompt
```

- "PVTexCoordName", default name for texture coordinate PV tags, the default value is "st".
- "PVNormalName", default name for vertex normal PV tags, the default value is "N".
- "PVCOLORName", default name for vertex color PV tags, the default value is "Cs".
- "ResetScriptEnv", determines whether or not to reset the property management and data array of a Script object script, when the script code was changed in the script editor.¹ Valid values are 0 and 1, default is 1.
- "RotateCrossSection", controls whether the creation of sweeps should rotate the cross section curve to the YZ-plane. If set to 0, no rotation occurs, if set to 1, the rotation will be attempted without any checks, and if set to 2 (this is the default²) the cross section is checked and a requester is raised offering to rotate the curve only if it is not already in the proper plane. See also section 5.4.6 Sweep Tool (page 310).
- "SafeAutoFocus" disables AutoFocus (see section 2.10.1 GUI preference settings (page 59)) when certain dialog windows are open, so that they do not get shuffled under other windows by accidental mouse movements on systems where the window manager does only auto raise in conjunction with auto focus. Valid values are 0 and 1. This option is enabled by default on Win32.
- "SDMode", silhouette detection mode for the hidden wire drawing mode, 0 – off, 1 – z-buffer, 2 – color, 3 – z-buffer and color. The default is 3.³
- "SelBndFactor", floating point value to multiply with the line width of selected lines when drawing selected boundary curves.⁴ The default value is 1.5. Ayam is not checking, whether the resulting value is supported by the OpenGL implementation used.
- "SelectLast" determines whether clicks into the tree widget (but not on any node) should select the last object of the current level. Valid values are 0 and 1. Default 1 – yes.⁵
- "SelLineWidth", line width used for drawing the lines of selected objects (white lines in standard color configuration). The default value is 1.0. Ayam is not checking, whether the specified value is supported by the OpenGL implementation used.
- "SelXOR_R", "SelXOR_G", "SelXOR_B": determine a color value that is used for drag selection rectangles. Note that the color is not used directly but combined with the color value of already drawn pixels by XOR. The default values are 255 for the red, 128 for the green, and 0 for the blue component.
- "ShiftTab", allows to set a specific keyboard symbol for systems where pressing the Shift together with the Tab key does not produce "<Shift-Tab>" (the default) but some other symbol like e.g. "<ISO_Left_Tab>" (many, but not all, modern X11 systems often use this).
- "SimpleToolGUI" controls whether the standard UI elements (as known from the property GUIs) should be used in tool dialogs or just simple entry widgets. Valid values are 0 and 1. Default 0, use standard UI elements.⁶
- "StripShaderArch" determines whether a second extension (the architecture) should be stripped from the file names of compiled shaders when scanning for shaders. Valid values are 0 and 1. Default 1 – yes.⁷

¹ Since 1.35. ² Since 1.25. ³ Since 1.21. ⁴ Since 1.30. ⁵ Since 1.22. ⁶ Since 1.22. ⁷ Since 1.22.

- "SwapMB", allows to swap mouse buttons 2 and 3 on Mac OS X Aqua (not X11!) for the mouse bindings specified in "SwapMBSC" below, because on Mac OS X Aqua, traditionally, the naming of the middle and rightmost mouse button is reversed compared to X11/Win32.
This option is enabled by default on Mac OS X Aqua and allows to use the same set of mouse bindings (the same *ayamrc* file) for X11 and Aqua without sacrificing user experience. The middle mouse button, by default, zooms the view, and the right one moves the view. Valid values are 0 and 1.
- "SwapMBSC" contains the mouse bindings to be swapped, when "SwapMB" above is activated. The default value of this option is

```
{ "ayviewshortcuts (MoveVButton) " "ayviewshortcuts (ZoomVButton) " }
```
- "TextFontsDir": initial directory for the font file selection file requester of the Text object; on UNIX this is by default set to `"/usr/share/fonts/truetype"` and on Windows to `"C:/Windows/Fonts"`.
- "toolBoxList": a list of sections or groups of buttons describing the appearance of the toolbox window (default, using all available sections: {trafo trafo2 solids misco nurbs toolobjs points nctools1 nctools2 camera misc}). See also section 2.8 [The Toolbox Window](#) (page 53).
- "ToolBoxShrink", controls whether the toolbox window should shrink wrap around its contents after a resize operation. This option is not used in single window GUI mode. Valid values are 0 and 1. Default is 1 – yes.
- "ToolBoxTrans", decides if the toolbox window should be made transient. It will then, depending on the window manager or its configuration, get a different or no decoration, no icon (or no entry in the task bar on Win32), and will always be iconified when the main window gets iconified. Not used in single window GUI mode. Valid values are 0 and 1. The default value is 1 – yes.
- "UseInternalFD" switches to an internal file dialog for loading of plugins. This option is only used on Mac OS X Aqua (not X11!), because there the normal file dialog will not enter application bundle directory structures. This option is enabled by default on Mac OS X Aqua and not used on any other platform. (valid values: 0, 1; default: 0 – no) See also the documentation of a related helper script 6.5.19 [Switch File Dialogs to Tcl](#) (page 453).
- "WarnPnts" controls the warning messages for missing selected points (valid values: 0, 1; default: 1 – yes).¹
- "WarnPropPasteToSel": should "Special/Clipboard/Paste Property to Selected" raise a warning requester? (valid values: 0, 1; default: 1 – yes)
- "WarnRendering" controls whether or not a warning message should appear when exiting Ayam with a viewport rendering in progress (valid values: 0, 1; default: 1 – yes).²
- "WarnUnknownTag" controls the warning messages for unknown tag types (valid values: 0, 1; default: 1 – yes).
- "Wait": set this to "waitPid" to enable a work around for zombie processes created by the Rendering GUI. This is e.g. necessary for the Linux platform.
- "WheelRoll", a float value that controls the amount of rolling (expressed in degrees) to apply when rolling the view with the mouse wheel.

¹ Since 1.27. ² Since 1.32.

- "WheelZoom", a float value that controls the zoom factor for view windows when either the mouse wheel is used or the <+>/<-> keys; the default value is 0.5. A value of 0.75 leads to more fine grained zoom control but slower zooming and a value of 0.25 to coarse but fast zooming.
- "WheelZoomToCursor" controls whether zooming a view with the mouse wheel zooms to the mouse cursor or to the current camera aim point. Valid values are "Always", "Modifier", "Parallel", and "Never":
In the "Modifier" setting, Ayam will only zoom to the mouse cursor, when the "Shift"-key is held down when turning the mouse wheel.
In the "Parallel" setting, Ayam will only zoom "Front", "Side", and "Top" views to the mouse cursor.
The default value is "Modifier".¹

¹ Since 1.33.

8.4.3 RiOption and RiAttributes Database

With the *ayamrc* file, also the database of RiOptions and RiAttributes may be adapted to the current target RenderMan rendering system.

Renderer specific options and attributes can then be added to the scenes using tags and the main menu entries "Special/Tags/Add RiOption" and "Special/Tags/Add RiAttribute", see also sections 4.11.1 RiAttribute Tag (page 259) and 4.11.2 RiOption Tag (page 260).

The syntax for a new RiOption is quite simple as the following example shows:

```
set riopt(runtime) {
  { verbosity s { "silent" "normal" "stats" "debug" } }
}
```

This snippet sets the section "runtime" and adds a single option, "verbosity", to it. The option is declared to be of type string using "s" and provided with a list of default values: "{ "silent" "normal" "stats" "debug" }".

To add another option to this section, say the option "op" which shall be an integer value, the aforementioned snippet needs to be changed to:

```
set riopt(runtime) {
  { verbosity s { "silent" "normal" "stats" "debug" } }
  { op i }
}
```

As you can see, it is not mandatory to provide default values. Be sure to correctly close all the curly braces, otherwise the next start of Ayam may fail.

Available types of parameters are:

- i: a scalar integer value,
- j: a pair of integer values,
- f: a scalar float value,
- g: a pair of float values,
- s: a string value,
- p: a point in space (simply three float values), the default values (if provided) are three float values in curly braces, such as {0.0 1.0 0.0},
- c: a color, the default values (if provided) are three float values in curly braces, such as {1.0 1.0 1.0}.

8.5 Command Line Arguments

This section documents the command line arguments supported by Ayam.

- `"-help"`: if this option is present, Ayam does not start but instead display a short help message about the available command line arguments.
- `"-guiscale x"`: this argument sets the GUI scale factor, which must be a floating point value between 1.0 and 3.0, values outside this range will be silently clamped to this range. See also section [2.11 GUI Scaling \(page 74\)](#).
- `"-nosplash"`: if this option is present, Ayam will not show the splash screen upon start.
- `"-noview"`: if this option is present, Ayam will not attempt to open a view window (only relevant in single window GUI mode, see section [2 The Ayam GUI \(page 18\)](#)).
- `"-failsafe"`: if this option is present, Ayam will not load the ayamrc file and also no environment scene file; see also section [8.4 Ayamrc File \(page 516\)](#).
- `"-log logfile"`: if this option is present, Ayam will use the specified file `"logfile"` as log file and also log the application start sequence. The logging related preference settings will be ignored/overridden by this option.

Command line arguments that follow any of these options are considered to be file names of Ayam scene files. All these scene files will be loaded upon application start.

If multiple scene file arguments are specified, the second up until the last scene files will be inserted into the scene from the first argument, i.e. different argument orders lead to different resulting scene hierarchies. See also sections [8.3 Scene File Management \(page 514\)](#) and [2.2 File Menu \(page 28\)](#).

8.6 Environment Variables

This section documents the environment variables used by Ayam.

- "AYAMRC": designates the full filename of the *ayamrc* file (see section 8.4 [Ayamrc File \(page 516\)](#)).
- "HOME": path to the *ayamrc* file (used on Win32 if "AYAMRC" is not set).
- "TEMP": path to the *ayamrc* file (used on Win32 if "AYAMRC" and "HOME" are not set); also initial value of the "TmpDir" preference setting (used on Win32 if no *ayamrc* file exists, that specifies "TmpDir").
- "AYNOSPLASH": if this variable is set to 1, the splash screen will not be shown.
- "BROWSER": filename of the preferred WWW browser (used to display the documentation URL).
- "NETSCAPE": (if "BROWSER" does not exist) filename of the Netscape WWW browser (used to display the documentation URL).
- "SHADERS": initial value of "Shaders" preference setting (used if no *ayamrc* file exists).

8.7 Plugins Overview

This section serves as a overview of the various plugins available in Ayam.

There are currently five major types of plugins for Ayam:

shader parsing plugins

aysdr, ayslb, ayslc, ayslo, ayslo3d, ayslx, ayso.

See also section [8.8 Shader Parsing Plugins \(page 529\)](#).

custom objects

metaobj, sdnpatch, sfcurve, sdcurve, bcurve, csphere.

See also section [4.9.2 Custom Objects \(page 238\)](#).

import/export plugins

dx fio, mfio, mopsi, objio, onio, rrib, x3dio.

See also section [7 Import and Export \(page 485\)](#).

scripting language plugins

- jsinterp – JavaScript scripting interface (see section [6.7 JavaScript Scripting Interface \(page 473\)](#)),
- luainterp – Lua scripting interface (see section [6.8 Lua Scripting Interface \(page 479\)](#)).

modelling helper plugins

- AyCSG – CSG rendering (see section [8.12 CSG preview using the AyCSG plugin \(page 532\)](#)),
- IDR – Importance Driven Rendering (see section [8.11 Importance Driven Rendering \(page 531\)](#)),
- aydnd – inter-application drag and drop,
- subdiv – Catmull-Clark and Loop subdivision for the SDMesh object (see section [4.8.2 SDMesh Object \(page 227\)](#)).

Furthermore, the Ayam distribution contains two plugins that are destined to be loaded by the respective RenderMan renderer and drive the *Viewport* rendering mode: `fifodspy` and `pixiefifodspy`.

See also sections [2.10.4 RIB-Export Preferences \(page 66\)](#) and [8.9 Viewport Rendering \(page 530\)](#).

8.8 Shader Parsing Plugins

The following plugins are provided to allow parsing of compiled shaders:¹ "ayslb" for Air, "ayslx" for Aqsis, "ayso" for RDC, "ayslo" for PRMan, "ayslo3d" for 3Delight, "aysdr" for Pixie², and "aygso" for Gelato³.

After loading of one of the aforementioned plugins, Ayam will be able to parse shaders compiled with the shader compiler of the respective renderer.

There can only be one active shader parsing plugin. You can not first load ayslb and then ayslx and expect Ayam to parse slb *and* slx shaders.

A shader parsing plugin may be loaded automatically on startup of Ayam using one of the provided Tcl scripts: "loadayslb.tcl", "loadayslo.tcl", "loadayslo3d.tcl", "loadayslx.tcl", "loadayso.tcl", "loadaysdr.tcl", and "loadaygso.tcl". To automatically load a plugin simply add the appropriate script to the preference setting "Main/Scripts" using the "Add" button in the preferences editor.

Additionally, those scripts may be further adapted to set a different "Shaders" preference setting or to immediately scan for shaders after loading of the plugin. For that, just remove the leading hash-marks (#) from the corresponding lines in the script. Note that changing the scripts for immediate shader parsing is not necessary if the shader parsing plugin is loaded automatically on startup of Ayam, as the loading of the scripts (and therefore also of the plugin) will happen before the Ayam startup sequence executes the initial shader scanning pass. The shader search path used for the initial shader scanning pass is taken from the "Shaders" preference setting.

If a shader parsing plugin is loaded manually or via unchanged load script, the shaders search path must be adapted manually. Furthermore a shader scan must be initiated manually too. Both actions may be carried out using the preferences editor. Scanning for shaders may also be started using the main menu: "Special/Scan Shaders".

To better accommodate the fast changing world of RenderMan renderers, since Ayam 1.11 all shader parsing plugins are Ayam version independent (but still renderer version dependent *and* Tcl version dependent). This allows to distribute updated shader parsing plugins without updating Ayam too and thus in a higher frequency. Furthermore, compiling a shader parsing plugin is now much easier.

For shader parsing without plugins, see also section 6.5.4 [Shader Parsing \(page 448\)](#).

¹ Since 1.3. ² Since 1.6. ³ Since 1.11.

8.9 Viewport Rendering

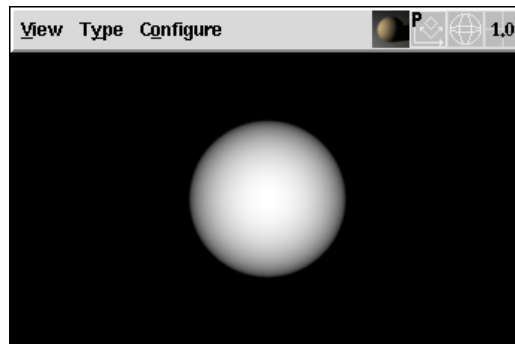


Figure 191: Viewport Rendering Example

The viewport rendering mode allows to display an image in a view window while it is being rendered for preview purposes, see also the image above.

After the renderer finished, the image can be zoomed and panned using the mouse wheel and the rightmost mouse button respectively.

Hitting the <Esc> key removes the rendered image and restores the normal view display.

To make this work, the RenderMan renderer must load a special plugin (also called *display driver*) that is distributed with Ayam.

There are actually two of these plugins provided: `fifodspy` for standard RenderMan renderers and `pixiefifodspy` for Pixie (to cater for its non-standard display driver interface). The correct plugin must be set in the corresponding "DisplayDriver" preference option, see also section 2.10.4 RIB-Export preferences (page 66). See also hidden preference option 8.4.2 DisplayPath (page 519).

The machine type (32 or 64 bit) of the concrete plugin used must match that of the renderer and may in fact be different from that of Ayam as one can use a 32 bit renderer from a 64 bit Ayam and vice versa.

The image data is sent via a named pipe (FIFO), which can be seen as a special type of file, that can be written and read by two processes simultaneously.

Here is what is going on under the hood:

1. Ayam will write a RIB that instructs the renderer to use the `fifodspy` display driver,
2. Ayam will start the renderer on this RIB and wait for the FIFO to appear,
3. the renderer reads the RIB and searches for the display driver plugin and loads it,
4. the initialization of the display driver will create the FIFO whose existence instructs Ayam to fully install the viewport rendering machinery, that continuously monitors the FIFO for new image data,
5. whenever a part of the image (scanline or bucket) is computed, it will be sent via the FIFO from the renderer to Ayam which immediately displays it in the view window,
6. upon completion, a special message is sent that disables the monitoring and the FIFO will be removed.

If step 3 fails, i.e. the renderer does not find or can not load the display driver, after 5s Ayam will complain about this and offer to either keep on waiting or to cancel the rendering.

8.10 Automatic Instancing

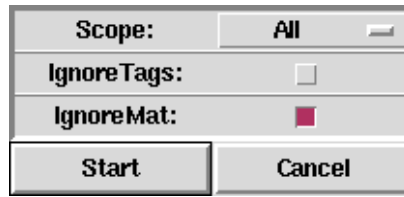


Figure 192: Automatic Instancing Dialog

Automatic Instancing is available via the main menu entry: "Special/Instances/Automatic Instancing". When this menu entry is selected a small parameter dialog will appear, see also the image above.

Automatic Instancing creates instances from all instantiable objects using a simple algorithm that recursively compares objects (see also section 4.2.7 [Instance object \(page 131\)](#)). Two objects are considered to be instantiable when all their attributes, except for the transformation attributes, are the same. The comparison of materials and tags may be turned off. Also the scope of the automatic instantiation can be set to the selected objects, the current level, or all objects in the scene (regardless of current level and selection).¹

The instancing algorithm is able to create instances of grouping objects too (objects with child objects, e.g. levels or tool-objects like revolve). However, in order for two grouping objects to be instantiated not only all child objects and the grouping objects have to be instantiable, but the child objects also have to be in the right order. It is not sufficient, that for every child of the potential master, a matching child of the potential instance exists. Instantiation of grouping objects may drastically decrease the total number of objects in a scene.

Note that before the automatic instantiation starts, all currently existing instances will be resolved. After instantiation some statistics will be displayed in the console.

More information about this subject can be found in:

Schultz, R., and Schumann, H.: "Automatic Instancing of Hierarchically Organized Objects", in: Kunii T.L. (ed.): Spring Conference on Computer Graphics (SCCG 2001) Conference Proceedings, Budmerice, Slovakia, 25-28 April 2001, ISBN 80-223-1606-7

8.11 Importance Driven Rendering (IDR)

The importance driven rendering plugin may be used to drastically reduce rendering times while developing a scene. It works in three main steps:

1. Importance values are assigned to elements of the scene.
2. Two rendering passes are started according to the assigned importance values. Elements of different importance values are mutually masked out using "RiMatte" statements.
3. The resulting partial images are composed to a single resulting image, which is then displayed.

The parameterisation of the two rendering passes ensures, that the total rendering time is lower than the rendering time of a single pass with high quality.

¹ Since 1.21.

Many options exist to assign importance and parameterise the rendering passes:

Elements of the scenes may be geometric objects, regions in image space, or regions in object space. Importance values are currently just binary values. Assignment may take place manually (using IDR tags) or half-automatic by derivation of importance from currently selected or changed objects. To avoid inconsistency in the resulting images, importance values may be propagated between (geometrically or hierarchically) near objects, or between objects that are related (e.g. from a material to a geometric object).

Parameterisation of the two rendering passes currently includes selection of a different renderer and the possibility to reduce rendering resolution and shading rate. To further reduce rendering times for raytracing renderers, the size of the region to render may be automatically adapted to the elements of the current importance value (including an optimisation run that balances renderer startup times and times needed to render regions not originally occupied by two regions to merge).

Furthermore, caching of partial images is possible. However, the implementation of this feature is not very sophisticated at the moment, as it uses the Unix text tool "diff" to decide whether two RIB streams are identical and hence need no re-rendering.

To start using IDR:

1. load a scene (e.g. the cactus example scene),
2. load the IDR plugin (menu "File/Load Plugin"),
3. open the IDR control window using the main menu "Custom/Open IDR",
4. set the assign mode to "Selection",
5. select an object in the scene (e.g. the object named "Pot"),
6. then press the "Render!" button.

Compare the rendering time with a full render from the view window.

IDR requires that at least the renderer of the second rendering pass honours RiMatte. Since rgl does not honour RiMatte, it is sometimes necessary to simply exclude objects of different importance value. No wrong images are to be expected from this, as rgl does not calculate other than local lighting effects.

More information about this subject can be found in:

Schultz, R., and Schumann, H.: "Importance Driven Rendering - Using Importance Information in the Rendering Process", in: Hamza M., Sarfraz M. (ed.): Computer Graphics and Imaging (CGIM 2001) Conference Proceedings, Honolulu, Hawaii, 13-16 August 2001, ISBN 0-88986-303-2

8.12 CSG preview using the AyCSG plugin

The AyCSG plugin may be used to resolve and preview CSG operations. For this, the plugin uses image based CSG rendering algorithms provided by the OpenCSG library by Florian Kirsch. The OpenCSG library, currently, supports the *Goldfeather* and the *SCS* algorithm. The latter only works properly with convex primitives. Since both, Goldfeather and SCS, are image based rendering algorithms, there is no limit in geometric object types that may be used in CSG hierarchies. You may e.g. use Quadrics, NURBS, and Metaballs in every possible combination. You just have to make sure, that every CSG primitive describes a closed space.

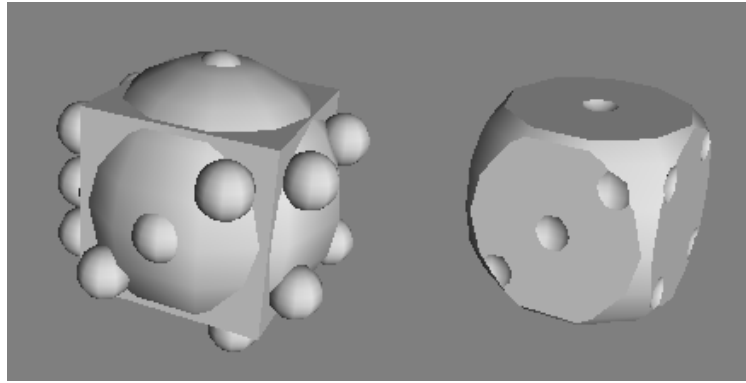


Figure 193: CSG preview example (left without, right with CSG)

In order for the CSG rendering algorithms to work properly, the depth complexity (convexity) of a primitive must be known. The depth complexity of a primitive determines the maximum number of forward oriented surfaces any ray through this primitive would pass. A regular sphere has a depth complexity of 1, a torus of 2, but do not confuse depth complexity with genus, they are different values. A 3D representation of the letter A e.g. has a genus of 1 but a depth complexity of 3. The depth complexity of a primitive can be stored in a "DC" tag. A torus would e.g. get a tag named "DC" with the value "2". If no "DC" tag is present for a primitive, a default value for the depth complexity of "1" will be used. If the depth complexity is not specified correctly, rendering errors, like missing parts of surfaces, will occur.

Note that the correct operation of AyCSG not only depends on the depth complexity but also the winding order of the OpenGL primitives (triangles or quads) used for drawing of the CSG primitives. The winding order has to be consistent in a scene, so that the rendering algorithm can decide what is inside and what is outside by looking at a single OpenGL primitive. For all quadric primitives of Ayam the winding order is always consistent. However, for NURBS patches the winding order depends on the orientation of the patch dimensions. If NURBS patches are used in CSG operations one, consequently, may need to revert the patches (e.g. using the "RevertU" tool, see [5.5.1 Revert U tool \(page 317\)](#)). If the winding order of some of the primitives in a CSG hierarchy is not right, the respective primitives will not be effective in the CSG operations to the extent that the rendered image becomes completely empty.

The AyCSG rendering obeys the "Draw Selection only" and "Draw Level only" view options as well as the hide attribute of objects. If the CSG rendering fails for complete complex scenes, you might still get a preview of the important CSG using objects by selecting them and enabling the "Draw Selection only" view option.

Also note that CSG rendering requires fast graphics hardware (the more fillrate, the better). Furthermore, your OpenGL subsystem has to support the PBuffers extension and, depending on the rendering options chosen, a stencil buffer. Speedups may be achieved using the "GL_ARB_occlusion_query" or "GL_NV_occlusion_query" extensions (if available to you).

Once the AyCSG plugin is loaded successfully you can render the CSG preview in any view window using the keyboard shortcut <Ctrl+C> or using the new button in the menu bar of every view window (see image below). If you hold down <Shift> while pressing the button the view will start to continually render CSG (the button stays pressed to signify this) until you click onto the button again.

The AyCSG plugin supports the following options, that are available through the main menu entry "Custom/AyCSG Preferences":

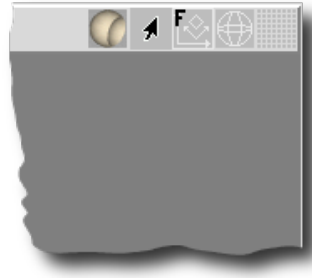


Figure 194: View With AyCSG Icon

- "Algorithm" allows to switch between the Goldfeather and SCS algorithm. Note again that the SCS algorithm only works correctly for convex primitives. The "Automatic" setting chooses one of the algorithms based on whether concave primitives (depth complexity > 1) are present or not. As this decision is made based on the depth complexity from DC tags, the only valid setting for the "DCSampling" preference below is then "NoDCSampling".
- "DCSampling" determines a depth complexity sampling strategy. Quoting from the OpenCSG documentation, the following options are available:

"NoDCSampling": Does not employ the depth complexity. This essentially makes the algorithm $O(n^2)$, but with low constant costs. This is the standard Goldfeather algorithm, DC tags *must* be present for primitives with a depth complexity greater than one, or rendering errors may occur.

"OcclusionQuery": Uses occlusion queries to profit implicitly from depth complexity without calculating it. This is especially useful for the SCS algorithm where this strategy is applied at shape level, resulting in a $O(n \times k')$ algorithm (where $k' \leq k$), without significant constant overhead. This strategy requires hardware occlusion queries, i.e. the OpenGL extension "GL_ARB_occlusion_query" or "GL_NV_occlusion_query". If this is enabled (and the "Algorithm" is set to Goldfeather), DC tags need *not* to be present.

"DCSampling": Calculates the depth complexity k using the stencil buffer. This makes the algorithm $O(n \times k)$, but with high constant costs. In case of the Goldfeather algorithm, the literature denotes this as layered Goldfeather algorithm. Note that this option requires a stencil buffer which must be enabled on the creation of a view window using additional view parameters (preferences option "AddViewParams", see also the section on [8.4.2 hidden preference settings \(page 517\)](#)). If stencil buffers are not enabled, the AyCSG plugin will automatically add the relevant option to the Ayam preferences but this only has an effect on newly created view windows (i.e. you might also want to restart Ayam, if you switch to this sampling strategy). If "DCSampling" is enabled (and the "Algorithm" is set to Goldfeather), DC tags need *not* to be present as the depth complexity of primitives is calculated internally.
- "OffscreenType": This option allows to switch between two offscreen rendering techniques, PBuffers and Frame Buffer Objects; depending on your graphics hardware and driver, one or the other might work better for you.
- "Optimization": Currently unused.
- "CalcBBS" determines whether bounding boxes should be calculated and used for speed up (not working at the moment).

The following table shows an overview of the available options their restrictions and implied requirements.

Algorithm	DCSampling	Concave Primitives	DC Tags needed	Requirements
Goldfeather	OcclusionQuery	Yes	No	occlusion query extension
Goldfeather	DCSampling	Yes	No	stencil buffer
Goldfeather	NoDCSampling	Yes	Yes	None
Automatic	NoDCSampling	Yes	Yes	None
SCS	NoDCSampling	No	No	None
SCS	OcclusionQuery	No	No	None
SCS	DCSampling	No	No	stencil buffer

Table 121: AyCSG Options Overview

The default settings are to use the Goldfeather algorithm with occlusion queries, as this combination allows complex CSG hierarchies, has widespread hardware support, requires no exotic view argument modifications, and also no DC tags.¹

See also: *Kirsch F. and Doellner J.: "Rendering Techniques for Hardware-Accelerated Image-Based CSG", in: Skala V. (ed.): Journal of WSCG'04, 221-228, ISSN 1213-6972*

¹ Since 1.25.

8.13 Increasing Drawing Speed

In case of sluggish response of the user interface of Ayam (not accounting for long tree update operations) several things to increase drawing speed can be done:

- Hide objects or complete object hierarchies using "Hide" in the "Tools" menu.
- Disable drawing of true NURBS curves/surfaces, if you can. Use the ControlHull display modes (hit "<F4>" to toggle).
- If you need to see curves/surfaces, try to increase the (GLU) sampling tolerance of the objects (use a value of about 60.0, hit "<F3>" multiple times).
- Switch the primary modelling view to draw just the selected object(s) or the current level.
- Iconify views you do not need, they will not be redrawn then.
- Switch off automatic redrawing of slow redrawing (e.g. shaded) views, and control their redraw by pressing <Ctrl+d> manually.
- Do not create unnecessary caps, they are trimmed NURBS patches that render very slowly.
- Disable "UseMatColor".

8.14 Modelling Without Views

To work with very large/complex scenes, it may be necessary to turn off all views. In Ayam this can be done in the multi window GUI mode (see section 2.10.1 [GUI preference settings \(page 59\)](#)). If there are many objects, it is also advisable to switch the object selection widget from tree view mode to list box mode (menu: "Special/Toggle TreeView").

Modelling without views does not work on Apple Mac OS X Aqua if GLU functionality is involved (as every GLU functionality on Mac OS X needs a GL context). GLU is needed for the PolyMesh and NURBS tessellation.

8.15 Restrictions and Implementation Deficiencies

Due to the way Ayam is implemented, several restrictions and deficiencies exist:

- Almost all strings in Ayam (scene file names, object names, material names) are restricted to be 7-bit ASCII. If the scene is not to be transported between platforms (e.g. between UNIX and Win32), using 8-bit ASCII should be fine, however.
- The maximum scene depth (i.e. the maximum number of nested levels) depends on the maximum stack size of the operating system Ayam is running on (due to the use of recursion almost everywhere when traversing the scene, e.g. for drawing purposes).
- Ayam internally operates on double precision math, however, no control over roundoff error in lengthy calculations exists. Saving of scene files only uses the precision dictated by the current standard C library. This may degrade the precision of the models. The usage of double precision numbers via the Tcl scripting interface or the GUI also can reduce model precision due to double-string-double-conversions (the precision of those conversion operations can be adjusted by the user via the "TclPrecision" preference setting, however).

- The maximum array size Ayam can handle is 2GB (this is also true on 64Bit platforms), therefore, the maximum NURBS patch dimensions are for example 8192*8192, the maximum number of control points in a PolyMesh or SDMesh is 89.478.485 (without vertex normals, with vertex normals only half as many) resulting, in the worst case, in 29.826.162 triangles (without vertex normals). None of these limits are actively enforced.
- Materials may only be assigned to complete objects, not to certain parts of objects.
- Undo/Redo are not possible for scene structure changes.
- In the user interface, colors are always represented as three 8-bit RGB values, even though the RenderMan interface allows e.g. for different color sample numbers or floating point samples instead of integers to be used for colors.
- There are no acceleration structures for e.g. drawing and object tree updates. This limits the number of objects that Ayam can handle simultaneously without big lags in redraws or after scene structure changes to about 1000. However, Ayam objects should be high-level objects, not single polygons, anyway, i.e. one usually gets away with about 100 objects for moderately complex scenes, the geometry-heavy Marsrakete uses just 62 objects.
- It is not possible to create a X11 using executable on the Win32 platform.

Ayam user interface deficiencies:

- The application state is often communicated via window title strings only. But certain window systems do not display those titles well, and they are not to be seen in the single window GUI mode at all.

8.16 How to Join the Fun

Helping to make Ayam even better will spice up your life too. Here is how to do it:

1. Write/translate tutorials.
2. Translate some balloon help texts. Create a new translation script file by entering into the Ayam console


```
»ms_create lang
```

 (replacing `lang` with a two character language designation like `it` for italian). Now edit/translate the new script file. Test it out by adding it to the "Scripts" preference setting.
3. Create and submit example objects, scenes, and images.
4. Implement custom objects like trees, landscape, sky, XSplines, T-Splines, or whatever you can think of. Note that the license of Ayam does not prevent you from implementing your custom object as shareware or even commercial software. However, free software is preferred for obvious reasons.
5. Donate source to improve several critical parts of the modeler, some ideas are: better (more exact) lighting simulation (is this possible to do with OpenGL at all?), transformation widgets, true support for subdivision surfaces, lift some deficiencies (see above), import/export plugins. The project page of Ayam on SourceForge lists some more tasks and you are always welcome to discuss such matters in the public forum.

Please do not implement custom objects like simple triangles or polygons. This would be something that really is not intended by the Ayam Team, and it would surely show the limits of the current design of all code operating on the scene structure.

Ayam objects should be high-level objects!

Reading the last paragraph you might think that we are a bit biased against polygonal models. We are not. Polygonal models are the only way to preview complex geometry using hardware accelerated graphics, for the moment. But even while RenderMan supports rendering of polygonal models, their use as a primitive is not recommended for good reasons. In other words, use polygonal models in the modeler as quick representation of your higher level objects, but please, if you are going to actually render something, do not use that polygonal representation. If you want to go a complete polygonal way instead, voila, there are good modelers out there.

8.17 References

Suggested reading:

- Advanced RenderMan: Creating CGI for Motion Pictures by Tony Apodaca and Larry Gritz (Morgan-Kaufmann, 1999)
- The RenderMan Companion: A Programmer's Guide to Realistic Computer Graphics by Steve Upstill (Addison-Wesley, 1989)
- Textures and Modelling: A Procedural Approach by Ebert, Musgrave, Peachey, Perlin, and Worley (Academic Press, 1994)

WWW resources related to Ayam:

- If you are reading this document from a local file system, maybe an updated version is available from the internet:
`http://ayam.sourceforge.net/docs/ayam.html`
- Ayam Tutorial #1:
`http://ayam.sourceforge.net/tut1/tutorial1.html`
- The Ayam FAQ:
`http://ayam.sourceforge.net/faq.html`

WWW resources related to RenderMan:

- The RenderMan Repository: `http://www.renderman.org/`
The original resource is now unfortunately offline, try your luck with the [Internet Archive copy from 12. Dec 2011](#) .
- The RenderMan Academy: `http://www.rendermanacademy.com/`
The original resource is now unfortunately offline, try your luck with the [Internet Archive copy from 19. Dec 2007](#) .

8.18 Acknowledgements

First of all, I would like to express a big "Thank you!" to Bernd (Pink) Sieker. He is the first real Mops user and beta tester, who urged me during the last years via E-Mail and on IRC to fix this particular bug, add some essential features, move the lights again etc. pp. in countless iterations. Bernd, without your help I surely would not be that far, thanks!

Furthermore, I would like to thank the following people:

- Hynek Schlawack: Ayam Makefile configuration GUI
- Florian Kirsch: OpenCSG
- Bertrand Coconnier: implementation of object picking
- Hermann Birkholz: initial implementation of the tree widget, shadow map support, AI, and IDR
- Frank (Copper) Pagels: MetaBalls custom object, TTF parser, providing a lot of good music
- Alain Siksik: French translation of documentation
- Stephen Echavia: Icons
- Larry Gritz: BMRT
- Benjamin Bederson, Brian Paul et. al.: The Togl Widget
- Jeffrey Hobbs: tkMegaWidget set
- Jan Nijtmans: Tcl/Tk PlusPatches, tcl2c, Wrap
- Georgios Petasis: tkdnd
- George Peter Staplin: ctext
- Thomas E. Burge: The Affine Toolkit
- Apple, Duet Development Corp.: 3DMF parser
- Mark J. Kilgard: GLUT
- Les A. Piegl and Wayne Tiller: The NURBS Book
- W. T. Hewitt and D. Yip: The NURBS Procedure Library
- Philippe Lavoie: The NURBS++ Library
- Tom Cashman: snurbs (Subdivision NURBS) library
- George W. Hart: Conway notation polyhedron generator
- Everyone involved in the development of Tcl/Tk, OpenGL, The RenderMan Interface

8.19 Trademarks

OpenGL (R) is a registered trademark of Silicon Graphics, Inc.

The RenderMan (R) Interface Procedures and Protocol are: Copyright 1988, 1989, 2000 Pixar All Rights Reserved

RenderMan (R) is a registered trademark of Pixar

The Affine Libraries and Tools are Copyright (c) 1995, 1996, 1997, 1998 Thomas E. Burge All rights reserved.

Affine (R) is a registered trademark of Thomas E. Burge.

TIFF Software is Copyright (c) 1988-1997 Sam Leffler Copyright (c) 1991-1997 Silicon Graphics, Inc.

Dime is Copyright (c) 1998-1999 Systems In Motion, SA All rights reserved.

AutoCAD (R) is a registered trademark of Autodesk, Inc.

DXF (R) is a registered trademark of Autodesk, Inc.

9 Index

Overview:

O – page 541
A – page 541
B – page 542
C – page 543
D – page 545
E – page 546
F – page 547
G – page 548
H – page 548
I – page 549
J – page 550
K – page 550
L – page 550
M – page 551
N – page 552
O – page 552
P – page 553
Q – page 554
R – page 554
S – page 557
T – page 560
U – page 561
V – page 562
W – page 562
X – page 563
Y – page 563
Z – page 563

Index:

O

2lcons.tcl: 6.5.7 example/helper script (page 449)
 3DM:
 • 7.12 3DM (Rhino) Import (page 502),
 • 7.13 3DM (Rhino) Export (page 503)
 3DMF:
 • 7.10 3DMF (Apple) Import (page 499),
 • 7.11 3DM (Apple) Export (page 500)
 3DPVN: 4.5.3 offset type (page 168)
 3DPVNB: 4.5.3 offset type (page 168)

A

aac.tcl: 6.5.12 example/helper script (page 451)
 AAFudge: 8.4.2 hidden preference setting (page 517)
 AALineWidth, AASelLineWidth: 8.4.2 hidden preference setting (page 517)
 About: 2.2 main menu entry (page 40)
 Accuracy:
 • 7.12.2 3DM (Rhino) import option (page 502),
 • 7.13.2 3DM (Rhino) export option (page 504)
 ACurve: 4.4.3 ACurve object (page 158)
 Action:
 • 3 interactive actions (page 75),
 • 2.9 object search option (page 54)
 Active: 4.9.1 Script attribute (page 231)
 Adaptive: 4.9.6 MetaObj attributes (page 246)
 AdaptiveDomainDistance, AdaptiveKnotDistance: 2.10.5 NURBS tessellation mode (page 73)
 Add RiOption, RiAttribute, Property, Tag: 2.2 Tag management (page 39)
 Add Tag: 2.2 Tag management (page 39)
 AddExtensions: 2.10.1 preference setting (page 59)
 addPropertyGUI: 6.2.27 scripting interface command (page 425)
 addScriptProperty: 6.2.27 scripting interface command (page 431)
 addTag: 6.2.12 scripting interface command (page 388)
 addToProc: 6.2.28 scripting interface command (page 438)
 AddViewParams: 8.4.2 hidden preference setting (page 517)
 ALength: 4.4.3 ACurve attribute (page 158)
 ALFileTypes, ALPlugins: 8.4.2 hidden preference setting (page 517)
 AllowWarp: 8.4.2 hidden preference setting (page 517)
 Algorithm: 8.12 AyCSG plugin option (page 532)
 Align to Object:
 • 2.5 view menu entry (page 44),
 • 2.6 view action (page 48)
 Angle: 6.6.6 Spiral attribute (page 465)

- ANS: 4.11.16 tag type (page 268)
- AntiAlias Lines: 2.5 view menu entry (page 44)
- Annulus: 6.6.2 Disk with Hole (page 461)
- apnt.tcl: 6.5.13 example/helper script (page 451)
- Append:
- 5.3.18 split curve tool option (page 297),
 - 5.5.15 split surface tool option (page 328),
 - 5.2.5 tween curve tool option (page 278),
 - 5.4.14 tween surface tool option (page 316)
- Approximate:
- 4.4.3 ACurve object (page 158),
 - 4.6.3 APatch object (page 176),
 - 5.3.23 curve tool (page 302),
 - 5.5.17 surface tool (page 330)
- approxNC: 6.2.15 scripting interface command (page 398)
- approxNP: 6.2.16 scripting interface command (page 407)
- Apply: 2.1.2 apply property GUI (page 24)
- Apply To All/Selected: 2.2 main menu entry (page 36)
- ApplyTrafo:
- 5.6.4 build NURBS patch tool option (page 336),
 - 5.6.3 break NURBS patch tool option (page 335)
- applyTrafo: 6.2.10 scripting interface command (page 385)
- Archives:
- 4.2.1 RenderMan interface option (page 113),
 - 4.2.11 RiInc object (page 140),
 - 4.2.12 RiProc object (page 141)
- Area Light: 4.2.4 creation of (page 124)
- Array: 4.2.8 Clone object (page 134)
- AskScriptDisable: 8.4.2 hidden preference setting (page 517)
- Atmosphere: 4.2.1 shader property (page 114)
- Attributes:
- 4.10.2 attributes property (page 252),
 - 4.2.5 RenderMan/BMRT attributes property (page 125),
 - 4.11.1 RiAttribute tag type (page 259)
- AUCommands: 8.4.2 hidden preference setting (page 517)
- AutoCAD DXF:
- 7.6 AutoCAD DXF import (page 489),
 - 7.7 AutoCAD DXF export (page 491)
- AutoCloseUI: 2.10.4 preference setting (page 66)
- AutoCloseUITimeOut: 8.4.2 hidden preference setting (page 517)
- AutoFocus: 2.10.1 preference setting (page 59)
- Automatic Instancing: 2.2 main menu entry (page 38)
- Automatic Redraw: 2.5 view menu entry (page 44)
- AutoResize: 2.10.1 preference setting (page 59)
- AutoSavePrefs: 2.10.1 preference setting (page 59)
- AvoidPwlCurve: 8.4.2 hidden preference setting (page 517)
- ayamrc: 8.4 Ayamrc File (page 516)
- ayError: 6.2.26 scripting interface command (page 424)
- aytest.tcl: 6.5.2 example/helper script (page 447)
- ## B
- B-Spline:
- 4.4.1 knot type (page 150),
 - 4.6.5 PatchMesh basis type (page 179),
 - 4.9.4 BCurve basis type (page 241)
- Backdrop: 4.2.1 shader property (page 114)
- Background: 2.10.3 preference setting (page 64)
- Background Image: 4.2.2 View attribute (page 117)
- BackupExt: 8.4.2 hidden preference setting (page 518)
- BakOnReplace: 2.10.1 preference setting (page 59)
- Balloon: 8.4.2 hidden preference setting (page 518)
- Basis: 4.9.4 BCurve attribute (page 241)
- Basis_U, Basis_V: 4.6.5 PatchMesh attribute (page 179)
- BCurve: 4.9.4 BCurve object (page 241)
- BeamDistrib: 4.2.4 Light attribute (page 119)
- Bevel:
- 4.7.9 Bevel object (page 208),
 - 5.4.7 Bevel tool (page 311),
 - 4.10.6 Bevels property (page 256),
 - 4.7.2 using bevels (page 185)
- Bevel3D: 4.5.3 offset mode (page 169)
- BevelRadius, BevelRevert, BevelType: 4.7.9 Bevel attribute (page 209)
- Bezier:

- 4.4.1 knot type (page 150),
 - 4.6.5 PatchMesh basis type (page 179),
 - 4.9.4 BCurve basis type (page 241)
- bgconvert.tcl: 6.5.3 example/helper script (page 447)
- BGImage: 4.2.2 View attribute (page 117)
- bicubic/bilinear PatchMesh: 4.6.5 PatchMesh object (page 179)
- Birail1:
- 4.7.5 Birail1 object (page 195),
 - 5.4.9 Birail1 tool (page 312)
- Birail2:
- 4.7.6 Birail2 object (page 198),
 - 5.4.10 Birail2 tool (page 313)
- BNS: 4.11.15 tag type (page 267)
- Bound: 4.2.5 RenderMan/BMRT attribute (page 125)
- BoundCoord: 4.2.5 RenderMan/BMRT attribute (page 125)
- Box: 4.3.1 Box object (page 142)
- BP: 4.11.18 tag type (page 269)
- BPatch: 4.6.4 BPatch object (page 178)
- Break: 5.6.3 NPatch tool (page 335)
- breakNP: 6.2.16 scripting interface command (page 408)
- Breakpoint: 3.20 breakpoint display (page 99)
- BStep: 4.9.4 BCurve attribute (page 241)
- BType: 4.9.4 BCurve attribute (page 241)
- BType_U, BType_V: 4.6.5 PatchMesh attribute (page 179)
- Build: 5.6.4 NCurve tool (page 336)
- buildNP: 6.2.16 scripting interface command (page 408)
- C**
- CalcBBS: 8.12 AyCSG plugin option (page 532)
- Camera:
- 4.2.3 Camera object (page 118),
 - 4.2.2 view property (page 116)
- candelOb: 6.2.3 scripting interface command (page 372)
- Cap:
- 4.7.10 Cap object (page 211),
 - 5.4.8 Cap tool (page 312),
 - 4.10.5 Caps property (page 255)
- capS: 6.2.14 scripting interface command (page 392)
- CastShadows: 4.2.5 RenderMan/BMRT attribute (page 125)
- Cat: 8.4.2 hidden preference setting (page 518)
- Catmull-Clark: 4.8.2 SDMesh subdivision scheme (page 227)
- Catmull-Rom:
- 4.6.5 PatchMesh basis type (page 179),
 - 4.9.4 BCurve basis type (page 241)
- cbox.tcl: 6.6.3 example/helper script (page 462)
- Chaikin: 4.9.5 curve subdivision method (page 243)
- Chamfer: 4.7.9 Bevel object (page 208)
- CheckLights: 2.10.4 preference setting (page 66)
- Chordal:
- 4.4.1 knot type (page 150),
 - 4.4.2 interpolation type (page 155),
 - 5.3.23 approximation type (page 302)
- Center All Points: 2.2 main menu entry (page 36)
- Centripetal:
- 4.4.1 knot type (page 150),
 - 4.4.2 interpolation type (page 155),
 - 5.3.23 approximation type (page 302)
- Circle:
- 5.2.2 NURBS circle tool (page 275),
 - 4.4.4 NCircle object (page 160)
- Clamp:
- 5.3.11 NCurve tool (page 289),
 - 5.5.10 NPatch tool (page 323)
- clampNC: 6.2.15 scripting interface command (page 393)
- clampuNP: 6.2.16 scripting interface command (page 400)
- clampvNP: 6.2.16 scripting interface command (page 400)
- Clear: 2.2 main menu entry (page 38)
- ClearClipboard: 2.9 object search option (page 54)
- clearClip: 6.2.8 scripting interface command (page 382)
- ClearHighlight: 2.9 object search option (page 54)
- clearMark: 6.2.28 scripting interface command (page 437)
- Clipboard:
- 2.2 edit menu entries (page 31),
 - 2.2 special menu entries (page 38),
 - 6.2.8 scripting interface commands (page 381)

- Clone: 4.2.8 Clone object (page 134)
- Close: 5.3.3 close curve tool (page 281)
- closeC: 6.2.13 scripting interface command (page 390)
- Closed:
 - 4.3.2 Sphere attribute (page 143),
 - 4.3.4 Cone attribute (page 145),
 - 4.3.5 Cylinder attribute (page 146),
 - 4.3.6 Torus attribute (page 147),
 - 4.3.7 Paraboloid attribute (page 148),
 - 4.3.8 Hyperboloid attribute (page 149),
 - 4.4.1 NCurve attribute (page 150),
 - 4.5.1 ConcatNC attribute (page 163),
 - 4.9.4 BCurve attribute (page 241),
 - 4.9.5 SDCurve attribute (page 243)
- Closed B-Spline: 5.2.1 closed B-Spline tool (page 274)
- closet.tcl: 6.5.11 example/helper script (page 450)
- Close_U, Close_V:
 - 4.6.5 PatchMesh attribute (page 179),
 - 4.6.2 IPatch attribute (page 174)
- Close U, Close V:
 - 5.5.4 close u surface tool (page 318),
 - 5.5.5 close v surface tool (page 318)
- Coarsen: 5.3.7 coarsen tool (page 285)
- CoarsenC: 6.2.13 scripting interface command (page 390)
- Cobb NURBS Sphere: 5.4.2 create surface tool (page 308)
- Collapse:
 - 5.3.29 collapse points tool (page 307),
 - 2.1.1 tree context menu entry (page 20)
- colfocus.tcl: 6.5.8 example/helper script (page 449)
- Color:
 - 4.2.5 RenderMan/BMRT attribute (page 125),
 - 4.2.4 Light attribute (page 119)
- Compatible: 4.7.11 ConcatNP attribute (page 214)
- CompleteNotify: 2.10.2 preference setting (page 62)
- Concat: 5.3.17 Concat tool (page 296)
- concatC: 6.2.13 scripting interface command (page 391)
- ConcatNC: 4.5.1 ConcatNC object (page 162)
- ConcatNP: 4.7.11 ConcatNP object (page 214)
- concatS: 6.2.14 scripting interface command (page 392)
- Cone: 4.3.4 Cone object (page 145)
- ConeAngle, ConeDAngle: 4.2.4 Light attribute (page 119)
- Connect: 2.2 Connect PolyMesh tool (page 35)
- Console: 2.1.3 main window component (page 27)
- ConsoleCursorEnd: 8.4.2 hidden preference setting (page 518)
- ConsoleTakeFocus: 8.4.2 hidden preference setting (page 518)
- Context Menu:
 - 2.1.1 object tree context menu (page 20),
 - 2.1.1 object listbox context menu (page 22),
 - 2.1.2 property listbox context menu (page 24),
 - 2.1.3 console context menu (page 28),
 - 4.9.1 Script object context menu (page 232),
 - 4.10.7 Tags property context menu (page 257),
 - 3.12.2 Numeric point edit context menu (page 92)
- Convert: 2.2 main menu entry (page 35)
- ConvertTags: 8.4.2 hidden preference setting (page 518)
- convOb: 6.2.28 scripting interface command (page 433)
- Coons Patch: 4.7.8 Gordon object (page 204)
- copOb: 6.2.8 scripting interface command (page 381)
- Copy: 2.2 main menu entry (page 31)
- Copy (Add): 2.2 main menu entry (page 38)
- Copy Property:
 - 2.2 main menu entry (page 31),
 - 2.1.2 copying properties (page 24)
- Corner: 4.8.2 subdivision mesh tag (page 227)
- CP: 4.11.19 tag type (page 269)
- Crease: 4.8.2 subdivision mesh tag (page 227)
- Create: 2.2 create menu (page 33)
- CreateAt: 2.10.2 preference setting (page 62)
- CreateIn: 2.10.2 preference setting (page 62)
- CreateMP:
 - 4.4.1 NCurve attribute (page 150),
 - 4.6.1 NPatch attribute (page 170)
- Create ShadowMap:
 - 2.2 main menu entry (page 38),

- 2.5 view menu entry (page 42)
 - crtClosedBS: 6.2.2 scripting interface command (page 368)
 - crtNCircle: 6.2.2 scripting interface command (page 367)
 - crtNCone: 6.2.2 scripting interface command (page 369)
 - crtNConicArc: 6.2.2 scripting interface command (page 368)
 - crtNCylinder: 6.2.2 scripting interface command (page 369)
 - crtNDisk: 6.2.2 scripting interface command (page 369)
 - crtNHyperbola: 6.2.2 scripting interface command (page 368)
 - crtNHyperboloid: 6.2.2 scripting interface command (page 370)
 - crtNParabola: 6.2.2 scripting interface command (page 368)
 - crtNParaboloid: 6.2.2 scripting interface command (page 370)
 - crtNRect: 6.2.2 scripting interface command (page 368)
 - crtNSphere: 6.2.2 scripting interface command (page 369)
 - crtNTorus: 6.2.2 scripting interface command (page 369)
 - crtTrimRect: 6.2.2 scripting interface command (page 369)
 - crtOb: 6.2.2 scripting interface command (page 350)
 - CSG:
 - 4.2.6 Level object (page 128),
 - 8.12 AyCSG CSG preview plugin (page 532)
 - Cubic: 4.9.5 curve subdivision method (page 243)
 - CullFaces: 8.4.2 hidden preference setting (page 518)
 - curvatNC: 6.2.15 scripting interface command (page 399)
 - curvatNP: 6.2.16 scripting interface command (page 410)
 - Curvature: 5.3.21 NCurve tool (page 300)
 - curvature.tcl: 6.5.25 example/helper script (page 455)
 - Curves: 4.4 curve primitives (page 150)
 - Custom:
 - 2.2 custom menu (page 37),
 - 4.4.1 knot type (page 150)
 - 4.2.4 light type (page 119),
 - 4.6.5 PatchMesh basis type (page 179),
 - 4.9.4 BCurve basis type (page 241)
 - Custom Object: 4.9.2 object type (page 238)
 - Cut: 2.2 main menu entry (page 31)
 - Cut (Add): 2.2 main menu entry (page 38)
 - cutOb: 6.2.8 scripting interface command (page 381)
 - cvview.tcl: 6.5.24 example/helper script (page 454)
 - CycleHiddenWire: 8.4.2 hidden preference setting (page 518)
 - CyclePerspective: 8.4.2 hidden preference setting (page 518)
 - Cylinder: 4.3.5 Cylinder object (page 146)
- ## D
- DANS: 4.11.16 tag type (page 268)
 - Data: 4.2.12 RiProc attribute (page 141)
 - DC: 4.11.12 tag type (page 265)
 - DCSampling: 8.12 AyCSG plugin option (page 532)
 - DefaultAction: 2.10.2 preference setting (page 62)
 - DefaultIllum:
 - 7.8.2 Wavefront OBJ import option (page 493),
 - 7.9.2 Wavefront OBJ export option (page 497)
 - DefaultMat: 2.10.4 preference setting (page 66)
 - Degree:
 - 4.9.7 SDNPatch attribute (page 248),
 - see also 9 Order (page 553),
 - Elevation: see also 9 Elevate (page 546),
 - Reduction: see also 9 Reduce (page 555)
 - DelayedReadArchive: 4.2.12 RiProc attribute (page 141)
 - delegTrafo: 6.2.10 scripting interface command (page 385)
 - Delete:
 - 2.2 main menu entry (page 31),
 - 3.18 delete points (page 97)
 - delOb: 6.2.2 scripting interface command (page 370)
 - Deselect All Points: 2.2 main menu entry (page 36)
 - Deselect Object: 2.1.1 tree context menu entry (page 20)
 - Derivatives: 4.4.2 ICurve attribute (page 156)

- Derivatives_U, Derivatives_V: 4.6.2 IPatch attribute (page 174)
- Difference: 4.2.6 Level object (page 128)
- Direct Editing: 3.12 edit points (page 90)
- DisableFailedScripts: 8.4.2 hidden preference setting (page 519)
- Disk: 4.3.3 Disk object (page 144)
- Displacement: 4.2.5 shader property (page 126)
- Display: 4.11.6 RiDisplay tag type (page 263)
- DisplayDriver: 2.10.4 preference setting (page 66)
- DisplayPath: 8.4.2 hidden preference setting (page 519)
- DisplayMode:
 - 4.4.1 NCurve attribute (page 150),
 - 4.6.1 NPatch attribute (page 170)
- Distant: 4.2.4 light type (page 119)
- distNC: 6.2.15 scripting interface command (page 399)
- DBNS: 4.11.15 tag type (page 267)
- Docs: 2.10.1 preference setting (page 59)
- Double Size: 2.5 view menu entry (page 44)
- downOb: 6.2.9 scripting interface command (page 383)
- dualsweep.tcl: 6.6.9 example/helper script (page 468)
- DrawGrid, DrawLevel, DrawSel, DrawBG: 4.2.2 View attribute (page 117)
- Draw BGImage: 2.5 view menu entry (page 44)
- Draw Grid: 2.5 view menu entry (page 44)
- Draw Level only: 2.5 view menu entry (page 44)
- Draw Object CS: 2.5 view menu entry (page 44)
- Draw Selection only: 2.5 view menu entry (page 44)
- Drawing modes: 2.7 Drawing Modes (page 50)
- DrawSub: 4.8.2 SDMesh attribute (page 227)
- dtree.tcl: 6.5.23 example/helper script (page 453)
- DXF:
 - 7.6 DXF import plugin (page 489),
 - 7.7 DXF export plugin (page 491)
- DynamicLoad: 4.2.12 RiProc attribute (page 141)
- DynamicMRU: 8.4.2 hidden preference setting (page 519)
- DZ: 6.6.5 Helix attribute (page 464)
- Edit:
 - 3.12 edit points actions (page 90),
 - 2.2 edit menu (page 31)
- Edit Local:
 - 2.5 view menu entry (page 44),
 - 3.26 Editing in Local Space (page 105)
- Edit TexCoords: 2.2 Tag management (page 39)
- EditSnaps: 2.10.2 preference setting (page 62)
- EFlush: 8.4.2 hidden preference setting (page 519)
- Elevate:
 - 5.3.8 NCurve tool (page 286),
 - 5.5.8 NPatch tool (page 321)
- elevateNC: 6.2.15 scripting interface command (page 393)
- elevateuNP: 6.2.16 scripting interface command (page 403)
- elevatevNP: 6.2.16 scripting interface command (page 403)
- EnableUndo: 4.2.2 View attribute (page 117)
- Enable Scripts: 2.2 main menu entry (page 38)
- EndBevel: 4.7.2 Extrude attribute (page 185)
- EndCap: 4.7.2 Extrude attribute (page 185)
- EnvFile: 2.10.1 preference setting (page 59)
- Environment Variables: 8.6 Environment Variables (page 527)
- Epsilon: 4.9.6 MetaObj attributes (page 246)
- ErrorLevel: 2.10.5 preference setting (page 71)
- estlenNC: 6.2.15 scripting interface command (page 396)
- evaluate curve/surface: 6.2.18 scripting interface command (page 413)
- ExcludeHidden: 2.10.4 preference setting (page 66)
- Expand: 2.1.1 tree context menu entry (page 20)
- ExpGain, ExpGamma: 4.2.1 RenderMan interface option (page 113)
- Explode: 5.3.30 explode points tool (page 307)
- Export:
 - 7.13 3DM (Rhino) export (page 503),
 - 7.11 3DMF (Apple) export (page 500),
 - 7.7 AutoCAD DXF Export (page 491),
 - 7.9 OBJ (Wavefront) export (page 497),
 - 7.4 RIB Export (page 488),
 - 7.14 X3D Export (page 505)
- Export RIB: 2.2 main menu entry (page 28)
- ExportSetsCD: 2.10.5 preference setting (page 71)

Expression:

- 2.9 object search attribute (page 54)
- 4.9.6 MetaComp attribute (page 246)

extendNC: 6.2.15 scripting interface command (page 393)

Extend: 5.3.10 extend curve tool (page 288)

Exterior: 4.2.5 shader property (page 126)

Extract:

- 5.6.1 extract curve tool (page 334),
- 5.6.2 extract patch tool (page 334)

ExtrNC:

- 4.5.2 ExtrNC object (page 165),
- 5.6.1 ExtrNC tool (page 334),
- 6.2.15 scripting interface command (page 396)

ExtrNP:

- 4.7.12 ExtrNP object (page 217),
- 5.6.2 ExtrNP tool (page 334),
- 6.2.16 scripting interface command (page 405)

Extrude:

- 4.7.2 Extrude object (page 184),
- 5.4.5 Extrude tool (page 310)

extruden.tcl: 6.6.11 example/helper script (page 470)

F

Face Extrude/Remove/Merge/Connect: 4.9.7 SD-NPatch modelling action (page 249)

Face normals: 2.2 PolyMesh tool (page 35)

Face selection: 3.25 selection action (page 104)

Fairing:

- 5.3.16 curve tool (page 294),
- 5.5.18 surface tool (page 331)

fairNC: 6.2.15 scripting interface command (page 399)

fairNP: 6.2.15 scripting interface command (page 399)

FAQ: 8.17 Ayam FAQ WWW reference (page 538)

Far: 4.2.2 camera property (page 116)

FDdisplay: 2.10.4 preference setting (page 66)

FDSHOWHidden, FDSHOWHiddenBtn: 8.4.2 hidden preference setting (page 519)

File:

- 2.2 file menu (page 28),

- 4.2.11 RiInc attribute (page 140),

- 4.2.12 RiProc attribute (page 141)

filletnc.tcl: 6.6.8 example/helper script (page 467)

FillGaps:

- 4.5.1 ConcatNC attribute (page 163),
- 4.7.11 ConcatNP attribute (page 214)

FilterFunc: 4.2.1 RenderMan interface option (page 113)

FilterWidth: 4.2.1 RenderMan interface option (page 113)

Find: 2.9 object search (page 54)

FindU: 3.20 modelling action (page 99)

FindUV: 3.21 modelling action (page 100)

FixDialogTitles: 8.4.2 hidden preference setting (page 519)

FixImageButtons: 8.4.2 hidden preference setting (page 519)

FixX11Menu: 8.4.2 hidden preference setting (page 519)

FlashObjects: 8.4.2 hidden preference setting (page 519)

FlashPoints: 2.10.2 preference setting (page 62)

Flatness: 4.9.6 MetaObj attributes (page 246)

Flip Loops/Smooth Normals: 2.2 polymesh tool (page 35)

flipPo: 6.2.17 scripting interface command (page 412)

FocusFollowsWheel: 8.4.2 hidden preference setting (page 519)

FontName: 4.7.14 Text attribute (page 221)

forAll: 6.2.23 scripting interface command (page 422)

Force3D: 4.7.9 Bevel attribute (page 209)

Force Notification: 2.2 main menu entry (page 35)

Formula: 4.9.6 MetaComp attribute (page 246)

Fraction: 4.7.10 Cap attribute (page 212)

FRender, FRenderPT, FRenderUI: 2.10.4 preference setting (page 66)

From:

- 4.2.2 camera property (page 116),
- 4.2.4 Light attribute (page 119)

From Camera:

- 2.2 main menu entry (page 38),
- 2.5 view menu entry (page 44)

Front: 2.5 view menu entry (page 44)

FTLength:

- 4.5.1 ConcatNC attribute (page 163),
- 4.7.11 ConcatNP attribute (page 214)

G

Gaussian curvature: 6.2.16 scripting interface command (page 410)

Gen. Face/Smooth Normals: 2.2 polymesh tool (page 35)

genfnPo: 6.2.17 scripting interface command (page 412)

gensnP: 6.2.17 scripting interface command (page 412)

getBB: 6.2.3 scripting interface command (page 372)

getMark: 6.2.28 scripting interface command (page 437)

getName: 6.2.3 scripting interface command (page 371)

getNormal: 6.2.18 scripting interface command (page 415)

getPlaneNormal: 6.2.3 scripting interface command (page 372)

getPnt: 6.2.18 scripting interface command (page 413)

getPrefs: 6.2.21 scripting interface command (page 421)

getProperty: 6.2.7 scripting interface command (page 379)

getTag: 6.2.12 scripting interface command (page 389)

getTags: 6.2.12 scripting interface command (page 389)

getTangent: 6.2.18 scripting interface command (page 417)

getType: 6.2.3 scripting interface command (page 371)

Gimbal Lock: 4.10.1 avoiding gimbal locks (page 251)

Global: 3.26 Editing in Local Spaces (page 105)

GlobalMark: 2.10.2 preference setting (page 62)

GLU: 2.10.3 preference setting (page 64)

goDown: 6.2.9 scripting interface command (page 383)

goTop: 6.2.9 scripting interface command (page 383)

Gordon:

- 4.7.8 Gordon object (page 204),
- 5.4.11 Gordon tool (page 313)

goUp: 6.2.9 scripting interface command (page 383)

Grid:

- 4.2.2 View attribute (page 117),
- 2.10.3 drawing preference setting (page 64)

Group: 4.2.6 Level object (page 128)

GUI Scaling: 2.11 GUI Scaling (page 74)

H

Half Size: 2.5 view menu entry (page 44)

handedness: 1.2.3 Coordinate Systems and Units (page 17)

HandleSize: 2.10.2 preference setting (page 62)

hasChild: 6.2.3 scripting interface command (page 371)

hasMat: 6.2.3 scripting interface command (page 371)

hasRefs: 6.2.3 scripting interface command (page 371)

hasTag: 6.2.12 scripting interface command (page 388)

hasTrafo: 6.2.3 scripting interface command (page 372)

HDisk: 6.6.2 Script object script (page 461)

Header: 7.9.2 Wavefront OBJ export option (page 497)

Height:

- 4.2.1 RenderMan interface option (page 113),
- 4.3.1 Box attribute (page 142),
- 4.3.4 Cone attribute (page 145),
- 4.6.1 NPatch attribute (page 170),
- 4.6.2 IPatch attribute (page 174),
- 4.6.5 PatchMesh attribute (page 179),
- 4.7.2 Extrude attribute (page 185),
- 4.2.2 View attribute (page 117),
- 4.7.14 Text attribute (page 221),
- 4.2.11 RiInc attribute (page 140)

helix.tcl: 6.6.5 Script object script (page 464)

Help:

- 2.2 main menu entry (page 40),
- 6.2.1 scripting interface command (page 349)

Help on object: 2.2 main menu entry (page 40)

Help on property: 2.2 main menu entry (page 40)

Hermite:

- 4.6.5 PatchMesh basis type (page 179),
- 4.9.4 BCurve basis type (page 241)

Hidden Preferences: 8.4.2 hidden preference settings (page 517)

HiddenWire:2.5 view menu entry (page 44)

Hide:

- 2.2 main menu entry (page 35),
- 4.10.2 attribute (page 252)

Hide All: 2.2 main menu entry (page 35)

Hider: 4.11.5 RiHider tag type (page 263)

HighlightColor: 2.9 object search option (page 54)

HLAyamAPI, HLCommands, HLComments, HLStrings, HLVars: 8.4.2 hidden preference setting (page 519)

Hole:

- 4.7.2 using extrusion holes (page 185),
- 4.8.2 subdivision mesh tag (page 227)

hSL: 6.2.4 scripting interface command (page 376)

Hybrid: 4.5.3 offset type (page 168)

Hyperboloid: 4.3.8 Hyperboloid object (page 149)

I

IconGamma: 8.4.2 hidden preference setting (page 520)

ICurve: 4.4.2 ICurve object (page 155)

IDR: 8.11 IDR plugin (page 531)

IgnoreHidden: 7.13.2 3DM (Rhino) export option (page 504)

IgnoreNormals: 2.2 Optimize PolyMesh tool option (page 35)

Image: 2.10.4 preference setting (page 66)

Imager: 4.2.1 shader property (page 114)

Import:

- 7.12 3DM (Rhino) Import (page 502),
- 7.10 3DMF (Apple) Import (page 499),
- 7.6 AutoCAD DXF Import (page 489),
- 7.5 Mops Import (page 489),
- 7.3 RIB Import (page 487),
- 7.8 Wavefront OBJ Import (page 492),
- 7.14 X3D Import (page 505)

Importance Driven Rendering: 8.11 IDR plugin (page 531)

ImportSetsCD: 2.10.5 preference setting (page 71)

Include: 4.2.11 RiInc object (page 140)

Indices: 4.2.10 Select attribute (page 139)

Input Plane: 3.26 Editing in Local Spaces (page 105)

Insert:

- 2.2 main menu entry (page 28),
- 3.18 insert points (page 97)

insknNC: 6.2.15 scripting interface command (page 394)

insknNP: 6.2.16 scripting interface command (page 401)

insknvNP: 6.2.16 scripting interface command (page 402)

Insert Knot:

- 5.3.13 NCurve tool (page 291),
- 5.5.12 NPatch tool (page 325)

insertScene: 6.2.24 scripting interface command (page 423)

Instance: 4.2.7 Instance object (page 131)

Instant Apply: 2.1.2 instant apply facility (page 24)

Integrate:

- 4.10.5 Bevel attribute (page 255),
- 4.10.6 Cap attribute (page 256)

Intensity: 4.2.4 Light attribute (page 119)

Interior: 4.2.5 shader property (page 126)

interpNC: 6.2.15 scripting interface command (page 397)

Interpolate:

- 4.7.4 Sweep attribute (page 192),
- 4.7.7 Skin attribute (page 202),
- 5.3.22 NCurve tool (page 301),
- 5.5.16 NPatch tool (page 329)

Interpolate Boundary: 4.8.2 subdivision mesh tag (page 227)

Interpolation: 4.10.2 RenderMan/BMRT attribute (page 252)

InterpolCtrl: 4.7.6 Birail2 attribute (page 199)

interpuNP: 6.2.16 scripting interface command (page 406)

interpvpNP: 6.2.16 scripting interface command (page 406)

Intersection: 4.2.6 Level object (page 128)

intfd.tcl: 6.5.19 example/helper script (page 453)

InvertMatch: 2.9 object search option (page 54)

Invert Selection: 2.2 main menu entry (page 36)

IPatch: 4.6.2 IPatch object (page 174)

isClosed: 6.2.3 scripting interface command (page 373)

isCompNC: 6.2.15 scripting interface command (page 397)

isCompNP: 6.2.16 scripting interface command (page 409)

isCurve: 6.2.3 scripting interface command (page 372)

isDegen: 6.2.3 scripting interface command (page 373)

isHidden: 6.2.3 scripting interface command (page 374)

IsLocal: 4.2.4 Light attribute (page 119)

isParent: 6.2.3 scripting interface command (page 374)

isPlanar: 6.2.3 scripting interface command (page 373)

IsoLevel: 4.9.6 MetaObj attribute (page 246)

IsOn: 4.2.4 Light attribute (page 119)

Isophotes: 2.7 drawing mode (page 52)

IsRat:

- 4.4.1 NCurve attribute (page 150),
- 4.6.1 NPatch attribute (page 170),
- 4.6.5 PatchMesh attribute (page 179)

isSurface: 6.2.3 scripting interface command (page 372)

isTrimmed: 6.2.3 scripting interface command (page 374)

J

JavaScript: 6.7 JavaScript Scripting Interface (page 473)

K

kdiallog.tcl: 6.5.17 example/helper script (page 452)

KeepNTmpFiles: 8.4.2 hidden preference setting (page 520)

KeepParamGUI: 8.4.2 hidden preference setting (page 520)

KeepParams: 4.10.4 shader management option (page 252)

KeepToolPreview: 8.4.2 hidden preference setting (page 520)

Keyboard Shortcuts:

- 2.3 main window shortcuts (page 41),
- 2.2 main menu shortcuts (page 28),
- 2.6 view window shortcuts (page 48),
- 2.5 main menu shortcuts (page 42),
- 2.1.1 object tree shortcuts (page 21),
- 2.1.1 object list shortcuts (page 22)

Kill: 8.4.2 hidden preference setting (page 520)

Knots: 4.4.1 NCurve attribute (page 150)

Knot-Type:

- 4.4.1 NCurve attribute (page 150),
- 4.5.1 ConcatNC attribute (page 163),
- 4.7.11 ConcatNP attribute (page 214)

Knot-Type_U:

- 4.6.1 NPatch attribute (page 170),
- 4.6.2 IPatch attribute (page 174),
- 4.7.7 Skin attribute (page 202)

Knot-Type_V:

- 4.6.1 NPatch attribute (page 170),
- 4.6.2 IPatch attribute (page 174)

L

l3add, l3sub, l3mul, l3scal, l3cross, l3dot:
6.2.19 vector algebra procedures (page 418)

l3to4, l4to3: 6.2.19 coordinate list procedures (page 419)

Last (None): 2.2 main menu entry (page 35)

Lathe: 4.7.1 Revolve object (page 182)

LazyNotify: 2.10.2 preference setting (page 62)

LazyUpdate: 5.6.5 tessellation option (page 337)

Length:

- 4.3.1 Box attribute (page 142),
- 4.4.1 NCurve attribute (page 150),
- 4.4.3 ACurve attribute (page 158),
- 4.4.2 ICurve attribute (page 156),
- 4.2.11 RiInc attribute (page 140),
- 4.9.4 BCurve attribute (page 241),
- 4.9.5 SDCurve attribute (page 243),
- 6.6.5 Helix attribute (page 464),
- 6.6.6 Spiral attribute (page 465)

Level:

- 4.2.6 Level object (page 128),
 - 4.9.5 SDCurve attribute (page 243),
 - 4.8.2 SDMesh attribute (page 227),
 - 4.9.7 SDNPatch attribute (page 248)
- Light:
- 4.2.4 Light object (page 119),
 - 2.10.3 preference setting (page 64)
- LightAttr: 4.2.4 property (page 119)
- LightShader: 4.2.4 light (page 119)
- Linear: 4.7.9 Bevel type (page 209)
- LineWidth: 8.4.2 hidden preference setting (page 520)
- link.tcl: 6.5.9 example/helper script (page 450)
- ListTypes: 8.4.2 hidden preference setting (page 520)
- List View: 2.1.1 List View (page 22)
- LoadEnv: 8.4.2 hidden preference settings (page 520)
- Load Plugin: 2.2 main menu entry (page 28)
- Local:
- 4.2.2 View attribute (page 117),
 - 3.26 Editing in Local Spaces (page 105)
- Locale: 2.10.1 preference setting (page 59)
- Loft: 4.7.7 Skin object (page 201)
- LogFile: 2.10.5 preference setting (page 71)
- Logging: 2.10.5 preference setting (page 71)
- Loop: 4.8.2 SDMesh subdivision scheme (page 227)
- LowerBevel: 4.7.14 Text attribute (page 221)
- LowerCap: 4.7.14 Text attribute (page 221)
- Lua: 6.8 Lua Scripting Interface (page 479)
- M**
- MajorRad: 4.3.6 Torus attribute (page 147)
- Make Compatible:
- 5.3.26 NCurve tool (page 305),
 - 5.5.19 NPatch tool (page 332)
- makeCompNC: 6.2.15 scripting interface command (page 397)
- makeCompNP: 6.2.16 scripting interface command (page 409)
- Mark:
- 4.2.2 View attribute (page 117),
 - 3.5 modelling action (page 82)
- MarkHidden: 8.4.2 hidden preference settings (page 520)
- Master: 2.2 edit menu entry (page 31)
- Material:
- 4.2.5 Material object (page 125),
 - 2.2 edit menu entry (page 31)
- Materialname:
- 4.2.5 attribute (page 126),
 - 4.10.3 material property attribute (page 252)
- MaxRayLevel: 4.2.1 RenderMan interface option (page 113)
- MaxTagLen: 8.4.2 hidden preference settings (page 520)
- MaxX, MaxY, MaxZ: 4.2.12 RiProc attribute (page 141)
- Menu:
- 2.2 main menu (page 28),
 - 2.5 view menu (page 42),
 - 2.1.1 tree context menu (page 20),
 - 2.1.1 listbox context menu (page 22),
 - 3.12.2 numeric point edit menu (page 92)
- Merge: 2.2 PolyMesh tool (page 35)
- MergeFaces 7.8 Wavefront OBJ import option (page 492)
- MergePVTags 7.8 Wavefront OBJ import option (page 492)
- MetaObj, MetaComp: 4.9.6 MetaObj object (page 245)
- mfio Plugin:
- 7.10 3DMF (Apple) import (page 499),
 - 7.11 3DMF (Apple) export (page 500)
- MinorRad: 4.3.6 Torus attribute (page 147)
- MinSamples, MaxSamples: 4.2.1 RenderMan interface option (page 113)
- MinX, MinY, MinZ: 4.2.12 RiProc attribute (page 141)
- Mirror: 4.2.9 Mirror object (page 137)
- MN: 4.11.20 tag type (page 270)
- Modelling: 3 Modelling Actions (page 75)
- Modelling Modes:
- 3.2 Transforming Objects or Selected Points (page 77),
 - 3.26 Editing in Local Spaces (page 105)
- Mode: 4.5.3 OffsetNC attribute (page 169)
- Mops Import: 7.5 Import of Mops Scenes (page 489)
- Move:

- 2.6 view action (page 48),
- 3.7 modelling action (page 85)

movOb: 6.2.10 scripting interface command (page 384)

movPnts: 6.2.10 scripting interface command (page 384)

MP: 4.11.21 tag type (page 270)

MTL:

- 7.8.3 Wavefront MTL import (page 494),
- 7.9.3 Wavefront MTL export (page 498)

Multiple Point: 4.4.1 Multiple Points (page 153)

Multiplicity: 3.19 modelling action (page 98)

myicons.tcl: 6.5.22 example/helper script (page 453)

N

nameOb: 6.2.28 scripting interface command (page 436)

NCDisplayMode: 2.10.3 preference setting (page 64)

NCDisplayModeA: 2.10.3 preference setting (page 64)

NCircle: 4.4.4 NCircle object (page 160)

NCurve: 4.4.1 NCurve object (page 150)

Near: 4.2.2 camera property (page 116)

Negative: 4.9.6 MetaComp attribute (page 246)

Network Surface: 4.7.8 Gordon object (page 204)

New: 2.2 main menu entry (page 28)

NewLoadsEnv: 8.4.2 hidden preference settings (page 520)

newScene: 6.2.24 scripting interface command (page 423)

NoExport: 4.11.8 tag type (page 264)

NormalEpsilon: 2.2 optimize PolyMesh tool option (page 35)

NormalizedDomainDistance: 2.10.5 NURBS tessellation mode (page 73)

NormalizeDigits, NormalizeMark, NormalizePoints, NormalizeTrafos: 8.4.2 hidden preference setting (page 520)

normPnts: 6.2.10 scripting interface command (page 386)

normTrafos: 6.2.10 scripting interface command (page 386)

normVal: 6.2.10 scripting interface command (page 386)

normVar: 6.2.10 scripting interface command (page 386)

notifyOb: 6.2.28 scripting interface command (page 436)

NP: 4.11.13 tag type (page 266)

NPatch: 4.6.1 NPatch object (page 170)

NPDisplayMode: 2.10.3 preference setting (page 64)

NPDisplayModeA: 2.10.3 preference setting (page 64)

NumClones: 4.2.8 Clone attribute (page 135)

Numeric Edit: 3.12 modelling action (page 91)

NumSamples: 4.9.6 MetaObj attribute (page 246)

NURB: 4.4.1 knot type (page 150)

NURBCircle: 5.2.2 NURBS circle tool (page 275)

NURBCurve: 4.4.1 NCurve object (page 150)

NURBPatch: 4.6.1 NPatch object (page 170)

NURBS: 6.2.15 scripting interface commands (page 393)

NURBSphere: 5.4.1 NURBS sphere tool (page 308)

O

ObeyNoExport: 7.13.2 3DM (Rhino) export option (page 504)

OBJ:

- 7.8 Wavefront OBJ import (page 492),
- 7.9 Wavefront OBJ export (page 497)

Object: 2.10.3 preference setting (page 64)

Objectname: 4.10.2 attribute (page 252)

Objects:

- 2.1.1 main window component (page 19),
- 4 object types (page 107)

OffsetNC: 4.5.3 OffsetNC object (page 168)

OffsetNP: 4.7.13 OffsetNP object (page 219)

OI: 4.11.24 tag type (page 271)

Open:

- 2.2 main menu entry (page 28),
- 5.3.2 open tool (page 280)

openC: 6.2.13 scripting interface command (page 390)

OpenNURBS:

- 7.12 3DM (Rhino) Import (page 502),
- 7.13 3DM (Rhino) Export (page 503)

opent.tcl: 6.5.11 example/helper script (page 450)

- Optimize: 2.2 PolyMesh tool (page 35)
 - OptimizeCoords: 2.2 optimize PolyMesh tool option (page 35)
 - OptimizeFaces: 2.2 optimize PolyMesh tool option (page 35)
 - OptimizeNew: 2.2 merge PolyMesh tool option (page 35)
 - OptimizePV: 2.2 optimize PolyMesh tool option (page 35)
 - Option: 4.11.2 RiOption tag type (page 260)
 - Order:
 - 4.4.1 NCurve attribute (page 150),
 - 4.4.3 ACurve attribute (page 158),
 - 4.4.2 ICurve attribute (page 156),
 - 4.7.1 Revolve attribute (page 182),
 - 4.7.11 ConcatNP attribute (page 214),
 - 4.9.3 SfCurve attribute (page 239)
 - Order_U:
 - 4.6.1 NPatch attribute (page 170),
 - 4.6.2 IPatch attribute (page 174),
 - 4.7.7 Skin attribute (page 202),
 - 4.7.8 Gordon attribute (page 206)
 - Order_V:
 - 4.6.1 NPatch attribute (page 170),
 - 4.6.2 IPatch attribute (page 174),
 - 4.7.8 Gordon attribute (page 206)
 - Outliner: 2.1.1 object tree view (page 20)
 - oval.tcl: 6.6.7 example/helper script (page 466)
- P**
- P1, P2: 4.3.8 Hyperboloid attribute (page 149)
 - Pan: 2.6 view action (page 48)
 - Pan to Mark: 2.6 view action (page 48)
 - PanDist: 8.4.2 hidden preference setting (page 521)
 - PaneMargins: 8.4.2 hidden preference setting (page 521)
 - Paraboloid: 4.3.7 Paraboloid object (page 148)
 - Parameter: 4.5.2 ExtrNC attribute (page 166)
 - Parameter Object: 8.2 modelling concept (page 511)
 - ParametricError: 2.10.5 NURBS tessellation mode (page 73)
 - ParamType: 4.4.2 ICurve attribute (page 156)
 - pasOb: 6.2.8 scripting interface command (page 381)
 - Paste: 2.2 main menu entry (page 31)
 - Paste (Move): 2.2 main menu entry (page 38)
 - Paste Property: 2.2 main menu entry (page 31)
 - Paste Property to Selected: 2.2 main menu entry (page 38)
 - Patch Based Modelling: 4.7.11 ConcatNP object (page 214)
 - PatchMesh: 4.6.5 PatchMesh object (page 179)
 - PatchNum:
 - 4.5.2 ExtrNC attribute (page 166),
 - 4.7.12 ExtrNP attribute (page 217),
 - 4.2.10 Select attribute (page 139),
 - 4.7.15 Trim attribute (page 224)
 - PatchSamples: 4.2.1 RenderMan interface option (page 113)
 - PathLength: 2.10.5 NURBS tessellation mode (page 73)
 - Perspective: 2.5 view menu entry (page 44)
 - PhiMax, PhiMin: 4.3.6 Torus attribute (page 147)
 - Pick: 3.3 selecting objects by picking (page 78)
 - PickCycle: 2.10.2 preference setting (page 62)
 - PickCycleMax: 8.4.2 hidden preference setting (page 521)
 - PickEpsilon: 2.10.2 preference setting (page 62)
 - PickTolerance: 8.4.2 hidden preference setting (page 521)
 - Pivot:
 - 3.5 set mark action (page 82),
 - 3.9 rotate about action (page 87),
 - 3.11 scale about action (page 89)
 - Plane: 4.2.9 Mirror attribute (page 137)
 - PlaneNormal: 4.5.3 offset type (page 168)
 - Plot Curvature: 5.3.21 NCurve tool (page 300)
 - Plugins: 8.7 Plugins Overview (page 528)
 - Point:
 - 4.2.4 light type (page 119),
 - 4.5.3 offset type (page 168)
 - Points: 6.2.18 scripting interface commands (page 413)
 - Point Size: 2.10.2 see preference setting HandleSize (page 62)
 - polyhedron.js: 6.6.12 example/helper script (page 471)
 - PolyMesh:
 - 4.8.1 PolyMesh object (page 225),
 - 2.2 PolyMesh tools (page 35),

- 6.2.17 scripting interface commands (page 412)

PolyOffset: 8.4.2 hidden preference setting (page 521)

Power:

- 4.6.5 PatchMesh basis type (page 179),
- 4.9.4 BCurve basis type (page 241)

PPRender: 2.10.4 preference setting (page 66)

Preferences:

- 2.10 Preferences (page 58),
- 8.4.2 hidden preference settings (page 517),
- 6.2.21 scripting interface commands (page 421)

Preview: 5 tool GUI element (page 273)

Procedural Objects: 4.2.12 RiProc object (page 141)

Procedurals: 4.2.1 RenderMan interface option (page 113)

Prompt: 8.4.2 hidden preference setting (page 521)

Properties: 2.1.2 main window component (page 24)

Property Clipboard: 2.1.2 property clipboard (page 25)

Property GUI: 6.2.27 scripting interface commands (page 425)

Primitive: 4.2.6 Level object (page 128)

Primitive Variable: 4.11.4 tag type (page 262)

Primitives: 5.6.5 tessellation option (page 337)

PRManSpec: 4.2.1 RenderMan interface option (page 113)

PV: 4.11.4 tag type (page 262)

PVColorName: 8.4.2 hidden preference setting (page 521)

PVNormalName: 8.4.2 hidden preference setting (page 521)

PVTexCoordName: 8.4.2 hidden preference setting (page 521)

Q

QuadAsBRep: 7.13.2 3DM (Rhino) export option (page 504)

Quadrangulate: 2.2 polymesh tool (page 35)

QuadrangulatedTriangles: 5.6.5 primitive creation method (page 342)

Quadrics: 4.3.2 quadric primitives (page 143)

QDisplayDriver: 2.10.4 preference setting (page 66)

QRender, QRenderPT, QRenderUI: 2.10.4 preference setting (page 66)

Quaternion: 4.10.1 transformations property (page 251)

QuickDraw 3D Metafile:

- 7.10 3DMF import (page 499),
- 7.11 3DMF export (page 500)

Quick Render: 2.5 view menu entry (page 42)

R

Radius:

- 4.3.2 Sphere attribute (page 143),
- 4.3.3 Disk attribute (page 144),
- 4.3.4 Cone attribute (page 145),
- 4.3.5 Cylinder attribute (page 146),
- 4.4.4 NCircle attribute (page 160),
- 4.7.9 Bevel attribute (page 209),
- 4.9.6 MetaComp attribute (page 246),
- 6.6.5 Helix attribute (page 464),
- 6.6.6 Spiral attribute (page 465)

RadSteps: 4.2.1 RenderMan interface option (page 113)

RationalPoints: 2.10.2 preference setting (page 62)

RationalStyle:

- 1.2.4 concept explanation (page 17),
- 7.10.2 3DMF import option (page 500),
- 7.11.3 3DMF export option (page 502),
- 7.8.2 Wavefront OBJ import option (page 493),
- 7.9.2 Wavefront OBJ export option (page 497),
- 7.14.3 X3D import option (page 508),
- 7.15.3 X3D export option (page 510)

rc.tcl: 6.5.15 example/helper script (page 452)

RDiff: 6.6.6 Spiral attribute (page 465)

ReadCamera: 7.3.2 RIB import option (page 488)

ReadCurves:

- 7.6.2 DXF import option (page 490),
- 7.10.2 3DMF (Apple) import option (page 500),
- 7.8 Wavefront OBJ import/export option (page 492),
- 7.12.2 3DM (Rhino) import option (page 502),

- 7.14.3 X3D import option (page 508)
- ReadFrame: 7.3.2 RIB import option (page 488)
- ReadLayers:
 - 7.6.2 DXF import option (page 490),
 - 7.12.2 3DM (Rhino) import option (page 502)
- ReadLights: 7.3.2 RIB import option (page 488)
- ReadMaterial: 7.3.2 RIB import option (page 488)
- ReadMaterials: 7.8.2 Wavefront OBJ import option (page 493)
- ReadOptions: 7.3.2 RIB import option (page 488)
- ReadPartial: 7.3.2 RIB import option (page 488)
- ReadSTrim:
 - 7.8 Wavefront OBJ import option (page 492),
 - 7.12.2 3DM (Rhino) import option (page 502),
 - 7.3.2 RIB import option (page 488),
 - 7.14.3 X3D import option (page 508)
- ReadTexCoords: 7.8.2 Wavefront OBJ import option (page 493)
- Rebuild: 2.1.1 tree context menu entry (page 20)
- Rectangle: 5.2.4 TrimRect tool (page 277)
- RedirectTcl: 2.10.5 preference setting (page 71)
- Redo:
 - 8.1 The Undo System (page 511),
 - 2.2 main menu entry (page 31),
 - 6.2.28 scripting interface command (page 434)
- Redraw:
 - 2.5 view menu entry (page 42),
 - 8.13 speeding up (page 536),
 - 4.2.2 View attribute (page 117)
- Reduce:
 - 5.3.9 NCurve tool (page 287),
 - 5.5.9 NPatch tool (page 322)
- reduceNC: 6.2.15 scripting interface command (page 394)
- reduceuNP: 6.2.16 scripting interface command (page 404)
- reducevNP: 6.2.16 scripting interface command (page 404)
- Refcount: 4.10.2 attribute (page 252)
- Reference Counter:
 - 4.2.7 Instance object (page 131),
 - 4.2.5 Material object (page 125)
- Reference: 4.2.7 instance type (page 132)
- References: 8.17 literature and WWW references (page 538)
- Refine:
 - 5.3.4 curve tool (page 282),
 - 5.5.6 surface tool (page 319)
- Refine Knots:
 - 5.3.5 NCurve tool (page 283),
 - 5.5.6 NPatch tool (page 319)
- Refine Knots with:
 - 5.3.6 NCurve tool (page 284),
 - 5.5.7 NPatch tool (page 320)
- refineC: 6.2.13 scripting interface command (page 390)
- refineknNC: 6.2.15 scripting interface command (page 395)
- RefineTrims: 5.6.5 tessellation option (page 337)
- refineuNP: 6.2.16 scripting interface command (page 403)
- refinevNP: 6.2.16 scripting interface command (page 403)
- registerTag: 6.2.12 scripting interface command (page 389)
- Relative:
 - 4.5.2 ExtrNC attribute (page 166),
 - 4.7.12 ExtrNP attribute (page 217)
- Rem. Smooth Normals: 2.2 PolyMesh tool (page 35)
- remknNC: 6.2.15 scripting interface command (page 394)
- remknuNP: 6.2.16 scripting interface command (page 402)
- remknvNP: 6.2.16 scripting interface command (page 402)
- remsnPo: 6.2.17 scripting interface command (page 412)
- remsuknNC: 6.2.15 scripting interface command (page 394)
- remsuknuNP: 6.2.16 scripting interface command (page 402)
- remsuknvNP: 6.2.16 scripting interface command (page 403)
- Remove Property: 2.2 Tag management (page 39)
- Remove Knot:
 - 5.3.14 NCurve tool (page 292),
 - 5.5.13 NPatch tool (page 326)

- RemoveMerged: 2.2 merge PolyMesh tool option (page 35)
- Render:
- 2.5 view menu entry (page 42),
 - 2.10.4 preference setting (page 66)
- Renderer: 2.2 select a different renderer (page 38)
- RenderMode: 2.10.4 preference setting (page 66)
- RenderPT, RenderUI: 2.10.4 preference setting (page 66)
- Repair Ayam: 6.5.1 Tcl helper script (page 446)
- Reparameterise: 5.3.20 NURBCurve tool (page 299)
- reparamNC: 6.2.15 scripting interface command (page 396)
- Replace: 2.2 main menu entry (page 38)
- ReplaceOriginal:
- 5.6.4 build NURBS patch tool option (page 336),
 - 5.6.3 break NURBS patch tool option (page 335)
- replaceScene: 6.2.24 scripting interface command (page 423)
- repOb: 6.2.8 scripting interface command (page 381)
- Rescale Knots to Mindist:
- 5.3.28 NURBCurve tool (page 306),
 - 5.5.21 NURBPatch tool (page 333)
- Rescale Knots to Range:
- 5.3.27 NURBCurve tool (page 306),
 - 5.5.20 NURBPatch tool (page 333)
- rescaleknNC: 6.2.15 scripting interface command (page 395)
- rescaleknNP: 6.2.16 scripting interface command (page 401)
- RescaleKnots:
- 7.6.2 DXF import option (page 490),
 - 7.12.2 3DM (Rhino) import option (page 502),
 - 7.3 RIB import option (page 487),
 - 7.8 Wavefront OBJ import option (page 492)
- Reset:
- 2.1.2 reset property GUI (page 24),
 - 2.5 view menu entry (page 42)
- ResetDM, ResetST: 7.5 Mops import option (page 489)
- ResetScriptEnv: 8.4.2 hidden preference setting (page 522)
- Reset Layout: 2.2 main menu entry (page 38)
- Reset Preferences: 2.2 main menu entry (page 38)
- Reset Weights: 3.13 reset weights action (page 93)
- ResInstances: 2.10.4 preference setting (page 66)
- Resolve all Instances: 2.2 main menu entry (page 38)
- ResolveInstances: 7.15.3 X3D (Web3D) export option (page 510)
- Revert:
- 5.3.1 Revert tool (page 279),
 - 4.7.14 Text attribute (page 221),
 - 4.5.1 ConcatNC attribute (page 163),
 - 4.5.3 OffsetNC attribute (page 169),
 - 4.5.2 ExtrNC attribute (page 166),
 - 4.7.11 ConcatNP attribute (page 214)
- RevertBevels: 4.7.14 Text attribute (page 221)
- revertC: 6.2.13 scripting interface command (page 390)
- RevertU, RevertV:
- 5.5.1 Revert U tool (page 317),
 - 5.5.2 Revert V tool (page 317)
- Revolve:
- 4.7.1 Revolve object (page 182),
 - 5.4.3 Revolve tool (page 309)
- RGBA_ONE, RGBA_MIN, RGBA_MAX, RGBA_Dither: 4.2.1 RenderMan interface option (page 113)
- Rhino:
- 7.12 3DM (Rhino) Import (page 502),
 - 7.13 3DM (Rhino) Export (page 503)
- RiAttribute:
- 4.2.5 property (page 125),
 - 4.11.1 tag type (page 259),
 - 8.4.3 RiOption and RiAttributes Database (page 525)
- RIB export:
- 2.10.4 preferences (page 66),
 - 2.2 main menu (page 28),
 - 2.5 view menu (page 42),
 - 6.2.25 scripting interface commands (page 424)
- RIB import: 7.3 RIB Import (page 487)
- RIBFile: 2.10.4 preference setting (page 66)
- RiDisplay: 4.11.6 tag type (page 263)
- Ridge: 4.7.9 Bevel type (page 209)
- RiHider: 4.11.5 tag type (page 263)
- RiInc: 4.2.11 RiInc object (page 140)

RiOption: 4.11.2 tag type (page 260)

RiOptions:

- 4.2.1 RenderMan interface options property (page 113),
- 8.4.3 RiOption and RiAttributes Database (page 525)

RiProc: 4.2.12 RiProc object (page 141)

RISandard: 2.10.4 preference setting (page 66)

RMax:

- 4.3.7 Paraboloid attribute (page 148),
- 6.6.2 HDisk attribute (page 461),
- 6.6.1 Truncated cone attribute (page 460)

RMin:

- 6.6.1 Truncated cone attribute (page 460),
- 6.6.2 HDisk attribute (page 461),
- 6.6.6 Spiral attribute (page 465)

Roll: 4.2.2 Camera attribute (page 116)

Root: 4.2.1 Root object (page 113)

Rotate:

- 2.6 view action (page 48),
- 3.8 modelling action (page 86),
- 4.7.4 Sweep attribute (page 192),
- 4.2.8 Clone attribute (page 135)

RotateCrossSection: 8.4.2 hidden preference setting (page 522)

Rotation:

- 4.10.1 transformations property (page 251),
- 4.10.1 using the transformations property (page 251)

Rotational Sweep: 4.7.3 Swing object (page 187)

rotOb: 6.2.10 scripting interface command (page 384)

rotPnts: 6.2.10 scripting interface command (page 385)

Round, RoundToCap, RoundToNormal: 4.7.9 Bevel type (page 209)

RP: 4.11.14 tag type (page 266)

RRIB: 7.3 RIB import plugin (page 487)

Ruled surface: 4.7.7 Skin object (page 201)

RunProgram: 4.2.12 RiProc attribute (page 141)

runTool: 6.2.28 scripting interface command (page 435)

rV: 6.2.20 scripting interface command (page 420)

SafeAutoFocus: 8.4.2 hidden preference setting (page 522)

Samples: 4.2.4 Light attribute (page 119)

Samples_X: 4.2.1 RenderMan interface option (page 113)

Samples_Y: 4.2.1 RenderMan interface option (page 113)

Save: 2.2 main menu entry (page 28)

Save as: 2.2 main menu entry (page 28)

Save Environment: 2.2 main menu entry (page 38)

Save Prefs: 2.2 main menu entry (page 28)

Save Selected as: 2.2 main menu entry (page 38)

SaveDialogGeom: 2.10.5 preference setting (page 71)

SaveMainGeom: 4.11.9 tag type (page 264)

SavePanelLayout: 4.11.10 tag type (page 265)

saveScene: 6.2.24 scripting interface command (page 423)

SB: 4.11.22 tag type (page 270)

Scale:

- 4.10.1 transformations property (page 251),
- 3.10 modelling action (page 88)

ScaleFactor:

- 7.6.2 DXF import option (page 490),
- 7.7.2 DXF export option (page 491),
- 7.12.2 3DM (Rhino) import option (page 502),
- 7.13.2 3DM (Rhino) export option (page 504),
- 7.8 Wavefront OBJ import option (page 492),
- 7.9 Wavefront OBJ export option (page 497),
- 7.3.2 RIB import option (page 488)

ScaleMode 4.7.15 Trim attribute (page 224)

scalOb 6.2.10 scripting interface command (page 384)

scalPnts 6.2.10 scripting interface command (page 385)

Scan Shaders:

- 2.2 main menu entry (page 38),
- 2.10.1 preference dialog (page 59),
- 4.10.4 shader parsing (page 253)

Scheme: 4.8.2 SDMesh attribute (page 227)

Scope: 2.9 object search option (page 54)

ScopeManagement: 2.10.2 preference setting (page 62)

- Script:
 - 6 scripting interface (page 345),
 - 4.9.1 Script object (page 229)
- Scripts:
 - 2.10.1 preference setting (page 59),
 - 6.5 distributed helper scripts (page 446)
- SDCurve: 4.9.5 SDCurve object (page 243)
- SDLen: 4.4.2 ICurve attribute (page 156)
- SDMesh: 4.8.2 SDMesh object (page 227)
- SDNCube: 4.9.7 SDNPatch Creation Support (page 250)
- SDNPatch: 4.9.7 SDNPatch object (page 248)
- Search: 2.9 object search (page 54)
- Section: 4.5.3 offset type (page 168)
- Sections:
 - 4.7.4 Sweep attribute (page 192),
 - 4.7.1 Revolve attribute (page 182),
 - 4.7.5 Birail1 attribute (page 196),
 - 4.7.6 Birail2 attribute (page 199),
 - 4.9.3 SfCurve attribute (page 239)
- SelBnd, SelBndC: 3.22 select boundary action (page 101)
- SelBndFactor: 8.4.2 hidden preference setting (page 522)
- SelFacePoints: 2.10.2 preference setting (page 62)
- Select: 4.2.10 Select object (page 139)
- Select (Faces):
 - 3.25 selection action (page 104),
 - 6.2.6 select faces scripting interface command (page 378)
- Select (Objects):
 - 2.1.1 select objects with the tree/listbox (page 19),
 - 3.3 select objects action (page 78),
 - 6.2.4 select objects scripting interface commands (page 375)
- Select (Points):
 - 3.4 select points action (page 79),
 - 6.2.5 select points scripting interface commands (page 377)
- Select All: 2.2 main menu entry (page 31)
- Select All Points: 2.2 main menu entry (page 36)
- Select None: 2.2 main menu entry (page 31)
- Select Renderer: 2.2 main menu entry (page 39)
- Selected Objects: 2.2 main menu entry (page 38)
- Selection: 2.10.3 preference setting (page 64)
- SelectLast: 8.4.2 hidden preference setting (page 522)
- SelLineWidth: 8.4.2 hidden preference setting (page 522)
- selOb: 6.2.4 scripting interface command (page 375)
- selPnts: 6.2.5 scripting interface command (page 377)
- SelXOR_R, SelXOR_G, SelXOR_B: 8.4.2 hidden preference setting (page 522)
- Set BGImage: 2.5 view menu entry (page 44)
- Set Gridsize: 2.5 view menu entry (page 44)
- Set FOV: 2.5 view menu entry (page 44)
- SetActionMenu: 2.10.5 preference setting (page 71)
- SetMark:
 - 4.2.2 View attribute (page 117),
 - 6.2.28 scripting interface command (page 437)
- setNormal: 6.2.18 scripting interface command (page 416)
- setPnt: 6.2.18 scripting interface command (page 414)
- setPrefs: 6.2.21 scripting interface command (page 421)
- SetRenderer: 2.10.4 preference setting (page 66)
- setTag: 6.2.12 scripting interface command (page 388)
- setTags: 6.2.12 scripting interface command (page 389)
- setProperty: 6.2.7 scripting interface command (page 380)
- SfCurve: 4.9.3 SfCurve object (page 239)
- Shade:
 - 2.10.3 preference setting (page 64),
 - 2.5 view menu entry (page 44)
- Shader: 4.10.4 properties (page 252)
- Shader Parsing: 8.8 shader parsing plugins (page 529)
- Shaders:
 - 2.10.1 preference setting (page 59),
 - 4.2.1 RenderMan interface option (page 113),
 - 6.2.11 scripting interface commands (page 387)
- ShadingRate: 4.10.2 RenderMan/BMRT attribute (page 252)

- ShadowBias: 4.2.1 RenderMan interface option (page 113)
- ShadowMaps:
 - 2.10.4 preference setting (page 66),
 - 4.2.4 using shadowmaps (page 121)
- Shadows: 4.2.4 Light attribute (page 119)
- Sheared extrusion: 6.6.11 example/helper script (page 470)
- Shift: 5.3.24 curve tool (page 303)
- shiftC: 6.2.13 scripting interface command (page 391)
- ShiftTab: 8.4.2 hidden preference setting (page 522)
- Show: 2.2 main menu entry (page 35)
- Show All: 2.2 main menu entry (page 35)
- Show Shortcuts: 2.2 main menu entry (page 40)
- Show Tooltips: 2.2 main menu entry (page 40)
- Shuffle: 2.1.1 tree keyboard shortcut (page 21)
- shufup.tcl: 6.5.10 example/helper script (page 450)
- shufdown.tcl: 6.5.10 example/helper script (page 450)
- Side:
 - 2.5 view menu entry (page 44),
 - 4.5.2 ExtrNC attribute (page 166)
- SimpleToolGUI: 8.4.2 hidden preference setting (page 522)
- SingleWindow: 2.10.1 preference setting (page 59)
- Skin:
 - 4.7.7 Skin object (page 201),
 - 5.4.12 Skin tool (page 314)
- sL: 6.2.4 scripting interface command (page 376)
- slxml.tcl: 6.5.4 example/helper script (page 448)
- SMChangeShaders: 2.10.4 preference setting (page 66)
- SMethod:
 - 5.6.5 tessellation tool parameter (page 337),
 - 2.10.5 preference setting (page 73)
- SMFileFormat: 2.10.4 preference setting (page 66)
- SMFileType: 2.10.4 preference setting (page 66)
- Smooth normals: 2.2 PolyMesh tool (page 35)
- SMRender, SMRenderUI, SMRenderPT: 2.10.4 preference setting (page 66)
- SMRes: 4.2.4 Light attribute (page 119)
- Snap Points to Grid: 3.14 modelling action (page 95)
- Snap Points to Mark: 3.16 modelling action (page 96)
- Snap3D: 2.10.2 preference setting (page 62)
- Solids: 4.3 solid primitives (page 142)
- sP: 6.2.4 scripting interface command (page 376)
- SP: 4.9.1 Script object (page 231)
- SParamU, SParamV:
 - 5.6.5 tessellation tool parameter (page 337),
 - 2.10.5 preference setting (page 73)
- Special: 2.2 special menu (page 38)
- Sphere:
 - 4.3.2 Sphere object (page 143),
 - 5.4.1 NURBS sphere tool (page 308)
- spiral.tcl: 6.6.6 example/helper script (page 465)
- Split:
 - 5.3.18 NCurve tool (page 297),
 - 5.5.15 NPatch tool (page 328),
 - 2.2 PolyMesh tool (page 35)
- Split Curve: 3.23 modelling action (page 102)
- splitNC: 6.2.15 scripting interface command (page 395)
- splituNP: 6.2.16 scripting interface command (page 404)
- splitvNP: 6.2.16 scripting interface command (page 404)
- Spot: 4.2.4 light type (page 119)
- ssp.tcl: 6.5.14 example/helper script (page 452)
- Standard Properties: 4.10 Standard Properties (page 251)
- StartBevel: 4.7.2 Extrude attribute (page 185)
- StartCap: 4.7.2 Extrude attribute (page 185)
- StdDisplay: 4.2.1 RenderMan interface option (page 113)
- Step: 4.9.4 BCurve attribute (page 241)
- Step_U, Step_V: 4.6.5 PatchMesh attribute (page 179)
- StepSize: 4.9.6 MetaObj attributes (page 246)
- STESS: 2.10.3 preference setting (page 64)
- Stretch: 3.10 modelling action (page 88)
- String: 4.7.14 Text attribute (page 221)
- StripShaderArch: 8.4.2 hidden preference setting (page 522)
- Subdivision Curve: 4.9.5 SDCurve object (page 243)
- Subdivision Mesh: 4.8.2 SDMesh object (page 227)
- Subdivision NURBS: 4.9.7 SDNPatch object (page 248)
- Superformula Curve: 4.9.3 SfCurve object (page 239)

Surface: 4.2.5 shader property (page 126)
 Surfaces: 4.6 surface primitives (page 170)
 SwapMB: 8.4.2 hidden preference setting (page 522)
 Swap UV: 5.5.3 surface tool (page 317)
 swapuvS 6.2.14 scripting interface command (page 392)
 Sweep:
 • 4.7.4 Sweep object (page 190),
 • 5.4.6 Sweep tool (page 310)
 Swing:
 • 4.7.3 Swing object (page 187),
 • 5.4.4 Swing tool (page 309)
 Symmetric: 4.4.3 ACurve attribute (page 158)
 Symmetry: 4.2.9 Mirror object (page 137)

T

Tag: 2.10.3 preference setting (page 64)
 Tags:
 • 4.11 tags property (page 258),
 • 6.2.12 scripting interface commands (page 388),
 • 4.8.2 subdivision mesh tags (page 227)
 TC: 4.11.3 tag type (page 260)
 tcircle.tcl: 6.6.4 example/helper script (page 463)
 tclevel:
 • 6.7.2 JavaScript scripting interface function (page 474),
 • 6.8.2 Lua scripting interface function (page 480)
 TclPrecision: 2.10.5 preference setting (page 71)
 tciset:
 • 6.7.2 JavaScript scripting interface function (page 475),
 • 6.8.2 Lua scripting interface function (page 481)
 tcivar:
 • 6.7.2 JavaScript scripting interface function (page 474),
 • 6.8.2 Lua scripting interface function (page 480)
 tcone.tcl: 6.6.1 Script object script (page 460)
 Tessellate: 5.6.5 NPatch tool (page 337)
 TessPoMesh 7.9 Wavefront OBJ export option (page 497)

Test Ayam: 6.5.2 example/helper script (page 447)
 Text: 4.7.14 Text object (page 221)
 TextFontsDir: 8.4.2 hidden preference setting (page 523)
 Textures: 4.2.1 RenderMan interface option (page 113)
 Texture Coordinates:
 • 4.11.3 tag type (page 260),
 • 4.11.3 texture coordinate editor (page 261)
 ThetaMax:
 • 4.3.2 Sphere attribute (page 143),
 • 4.3.3 Disk attribute (page 144),
 • 4.3.4 Cone attribute (page 145),
 • 4.3.5 Cylinder attribute (page 146),
 • 4.3.6 Torus attribute (page 147),
 • 4.3.7 Paraboloid attribute (page 148),
 • 4.3.8 Hyperboloid attribute (page 149),
 • 4.7.1 Revolve attribute (page 182),
 • 6.6.1 Truncated cone attribute (page 460),
 • 6.6.2 HDisk attribute (page 461)
 Threshold: 4.9.6 MetaObj attributes (page 246)
 TMax, TMin:
 • 4.4.4 NCircle attribute (page 160),
 • 4.9.3 SfCurve attribute (page 239)
 TmpDir: 2.10.1 preference setting (page 59)
 To:
 • 4.2.2 camera property (page 116),
 • 4.2.4 Light attribute (page 119)
 tobasisPM: 6.2.16 scripting interface command (page 411)
 To Camera: 2.5 view menu entry (page 44)
 Toggle Toolbox: 2.2 main menu entry (page 38)
 Toggle TreeView: 2.2 main menu entry (page 38)
 Tolerance:
 • 2.10.3 preference setting (page 64),
 • 4.4.1 NCurve attribute (page 150),
 • 4.6.1 NPatch attribute (page 170)
 ToleranceA: 2.10.3 preference setting (page 64)
 tonpatch.tcl: 6.5.6 example/helper script (page 449)
 Tool Objects: 8.2 modelling concept (page 511)
 Toolbox: 2.8 toolbox window (page 53)
 toolBoxList: 8.4.2 hidden preference setting (page 523)
 ToolBoxShrink: 8.4.2 hidden preference setting (page 523)

ToolBoxTrans: 8.4.2 hidden preference setting (page 523)

Tools: 2.2 tools menu (page 35)

Top: 2.5 view menu entry (page 44)

TopLevelLayers:

- 7.7.2 DXF export option (page 491),
- 7.13.2 3DM (Rhino) export option (page 504),
- 7.15.3 X3D (Web3D) export option (page 510)

topoly.tcl: 6.5.5 example/helper script (page 449)

torsionNC: 6.2.15 scripting interface command (page 399)

Torus: 4.3.6 Torus object (page 147)

To XY: 5.3.25 To XY tool (page 304)

toXYC: 6.2.13 scripting interface command (page 391)

TP: 4.11.11 tag type (page 265)

Transformations:

- 4.10.1 transformations property (page 251),
- 6.2.10 scripting interface commands (page 384)

Translate:

- 2.6 view action (page 48),
- 3.7 modelling action (page 85)

Translation: 4.10.1 transformations property (page 251)

translational surface: 6.6.10 example/helper script (page 469)

Tree View: 2.1.1 Tree View (page 20)

Triangular Gordon: 4.7.8 Gordon surface (page 207)

Trim:

- 4.7.15 Trim object (page 223),
- 5.4.13 Trim tool (page 315),
- 2.5 view menu entry (page 44)

Trim Curve: 4.6.1 using trim curves (page 171)

trimNC: 6.2.15 scripting interface command (page 396)

TrimRect: 5.2.4 TrimRect tool (page 277)

TrueDisp: 4.10.2 RenderMan/BMRT attribute (page 252)

Truncated Cone: 6.6.1 example/helper script (page 460)

tsurf.tcl: 6.6.10 example/helper script (page 469)

Tutorial: 8.17 Tutorial WWW reference (page 538)

Tween:

- 5.2.5 NCurve tool (page 278),
- 5.4.14 NPatch tool (page 316)

tweenNC: 6.2.15 scripting interface command (page 395)

tweenNP: 6.2.16 scripting interface command (page 405)

TwmCompat: 2.10.1 preference setting (page 59)

Type:

- 4.2.6 Level attribute (page 128),
- 4.2.4 Light attribute (page 119),
- 4.6.5 PatchMesh attribute (page 179),
- 4.2.2 View attribute (page 117),
- 4.9.1 Script attribute (page 231),
- 4.2.12 RiProc attribute (page 141),
- 4.7.10 Cap attribute (page 212),
- 4.4.1 NCurve attribute (page 150),
- 4.4.2 ICurve attribute (page 156),
- 4.7.11 ConcatNP attribute (page 214),
- 4.9.5 SDCurve attribute (page 243)

U

uCL: 6.2.20 scripting interface command (page 420)

uCR: 6.2.20 scripting interface command (page 420)

UMax, UMin: 4.7.12 ExtrNP attribute (page 217)

UMM: 4.11.17 tag type (page 269)

Unclamp:

- 5.3.12 NCurve tool (page 290),
- 5.5.11 NPatch tool (page 324)

unclampNC: 6.2.15 scripting interface command (page 393)

unclampuNP: 6.2.16 scripting interface command (page 400)

unclampvNP: 6.2.16 scripting interface command (page 401)

Undo:

- 8.1 The Undo System (page 511),
- 2.2 main menu entry (page 31),
- 6.2.28 scripting interface command (page 434)

UndoLevels: 2.10.2 preference setting (page 62)

Uniform:

- 4.4.2 interpolation type (page 155),
- 5.3.23 approximation type (page 302)

Union: 4.2.6 Level object (page 128)
 upOb: 6.2.9 scripting interface command (page 383)
 UpperBevel: 4.7.14 Text attribute (page 221)
 UpperCap: 4.7.14 Text attribute (page 221)
 up-vector: 4.2.2 camera property (page 116)
 uS: 6.2.20 scripting interface command (page 420)
 useaqsisapp.tcl: 6.5.20 example/helper script (page 453)
 UseGrid:

- 4.2.2 View attribute (page 117),
- 2.5 view menu entry (page 44)

 UseGUIScale: 2.10.3 preference setting (page 64)
 UseInternalFD: 8.4.2 hidden preference setting (page 523)
 UseMatColor: 2.10.3 preference setting (page 64)
 UseMaterials: 7.8.2 Wavefront OBJ import option (page 493)
 UseTexCoords: 5.6.5 tessellation option (page 337)
 UseVertColors: 5.6.5 tessellation option (page 337)
 UseVertNormals: 5.6.5 tessellation option (page 337)
 usepixie.tcl: 6.5.21 example/helper script (page 453)
 UseSM: 4.2.4 Light attribute (page 119)
 UVSelect: 4.7.11 ConcatNP attribute (page 214)

V

Variance: 4.2.1 RenderMan interface option (page 113)
 View:

- 4.2.2 View object (page 116),
- 2.4 Anatomy of a View (page 42)

 ViewAttrib: 4.2.2 property (page 117)
 Viewport: 2.10.4 preference setting (page 66)
 Viewport Rendering: 8.9 rendering mode (page 530)
 VMax, VMin: 4.7.12 ExtrNP attribute (page 217)
 VMM: 4.11.17 tag type (page 269)

W

Wait: 8.4.2 hidden preference setting (page 523)
 WarnPnts: 8.4.2 hidden preference setting (page 523)
 WarnPropPasteToSel: 8.4.2 hidden preference setting (page 523)

WarnRendering: 8.4.2 hidden preference setting (page 523)
 WarnUnknownTag: 8.4.2 hidden preference setting (page 523)
 WatchCorners: 4.7.8 Gordon attribute (page 206)
 Watertight Tessellation: 2.2 Connect PolyMesh tool (page 35)
 Wavefront OBJ:

- 7.8 Wavefront OBJ import (page 492),
- 7.9 Wavefront OBJ export (page 497)

 Weight:

- 3.13 single point weight editing (page 93),
- 1.2.4 rational style (page 17)

 whatis 6.2.28 scripting interface command (page 438)
 WheelRoll: 8.4.2 hidden preference setting (page 523)
 WheelZoom: 8.4.2 hidden preference setting (page 523)
 WheelZoomToCursor: 8.4.2 hidden preference setting (page 523)
 Width:

- 4.2.1 RenderMan interface option (page 113),
- 4.3.1 Box attribute (page 142),
- 4.6.1 NPatch attribute (page 170),
- 4.6.2 IPatch attribute (page 174),
- 4.6.5 PatchMesh attribute (page 179),
- 4.2.2 View attribute (page 117),
- 4.2.11 RiInc attribute (page 140)

 withOb: 6.2.23 scripting interface command (page 422)
 writb: 6.2.25 scripting interface command (page 424)
 WriteCurves:

- 7.9.2 Wavefront OBJ import/export option (page 497),
- 7.13.2 3DM (Rhino) export option (page 504),
- 7.15.3 X3D (Web3D) export option (page 510)

 WriteHeader: 7.9.2 Wavefront OBJ export option (page 497)
 WriteMaterials:

- 7.9.2 Wavefront OBJ import/export option (page 497),
- 7.15.3 X3D (Web3D) export option (page 510)

WriteParameters: 7.15.3 X3D (Web3D) export option (page 510)

WriteIdent: 2.10.4 preference setting (page 66)

WriteSelected:

- 7.13.2 3DM (Rhino) export option (page 504),
- 7.9 Wavefront OBJ export option (page 497)

WriteViews: 7.15.3 X3D (Web3D) export option (page 510)

WriteX3dom: 7.15.3 X3D (Web3D) export option (page 510)

X

X3D:

- 7.14 X3D (Web3D) Import (page 505),
- 7.15 X3D (Web3D) Export (page 509)

x3dom-nurbs: 6.5.26 NURBS for X3DOM (page 456)

XML: 4.11.23 XML tag (page 271)

Y

.

Z

Zap Ayam: 2.2 main menu entry (page 38)

zap.tcl: 6.5.16 example/helper script (page 452)

zdialog.tcl: 6.5.18 example/helper script (page 452)

ZMax, ZMin:

- 4.3.2 Sphere attribute (page 143),
- 4.3.3 Disk attribute (page 144),
- 4.3.5 Cylinder attribute (page 146),
- 4.3.7 Paraboloid attribute (page 148),
- 6.6.1 Truncated cone attribute (page 460)

Zoom:

- 4.2.2 camera property (page 116),
- 2.6 view action (page 48)

Zoom to All:

- 2.5 view menu entry (page 44),
- 2.6 view action (page 48)

Zoom to Object:

- 2.5 view menu entry (page 44),
- 2.6 view action (page 48)