# Ayam

Randolf Schultz (randolf.schultz@gmail.com) 20. Mar 2012

This is the documentation of Ayam 1.20 – 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, one can use the provided SGML-source to generate other formats.

## Overview

# Contents

# 1 Introduction

This section contains general information about this manual and Ayam.

## 1.1 About this Manual

Since 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). 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 327))!

This document is organized in the following way:

After this first section, the user interface of Ayam is introduced 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)
```

## 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.

### 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 306)).
- 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.
- 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.

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.

NURBS curves and surfaces support rational coordinate values by allowing a fourth value (the weight) to be specified for each control point. In Ayam versions prior to 1.19 those weights were always multiplied with the coordinate values (i.e. homogenous 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 appproach was that modification of coordinates or weights for modelling purposes was unnecessarily complicated. Therefore, since Ayam 1.19, rational coordinates are now euclidean rational, the weight is not pre-multiplied with the coordinates anymore (this will be done internally when handing the coordinates over for drawing or export purposes). Scene files from older versions are converted automatically to the new coordinate scheme when read by Ayam 1.19, but loading of scene files written by Ayam 1.19 into older versions of Ayam requires conversion (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 split into 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.

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.

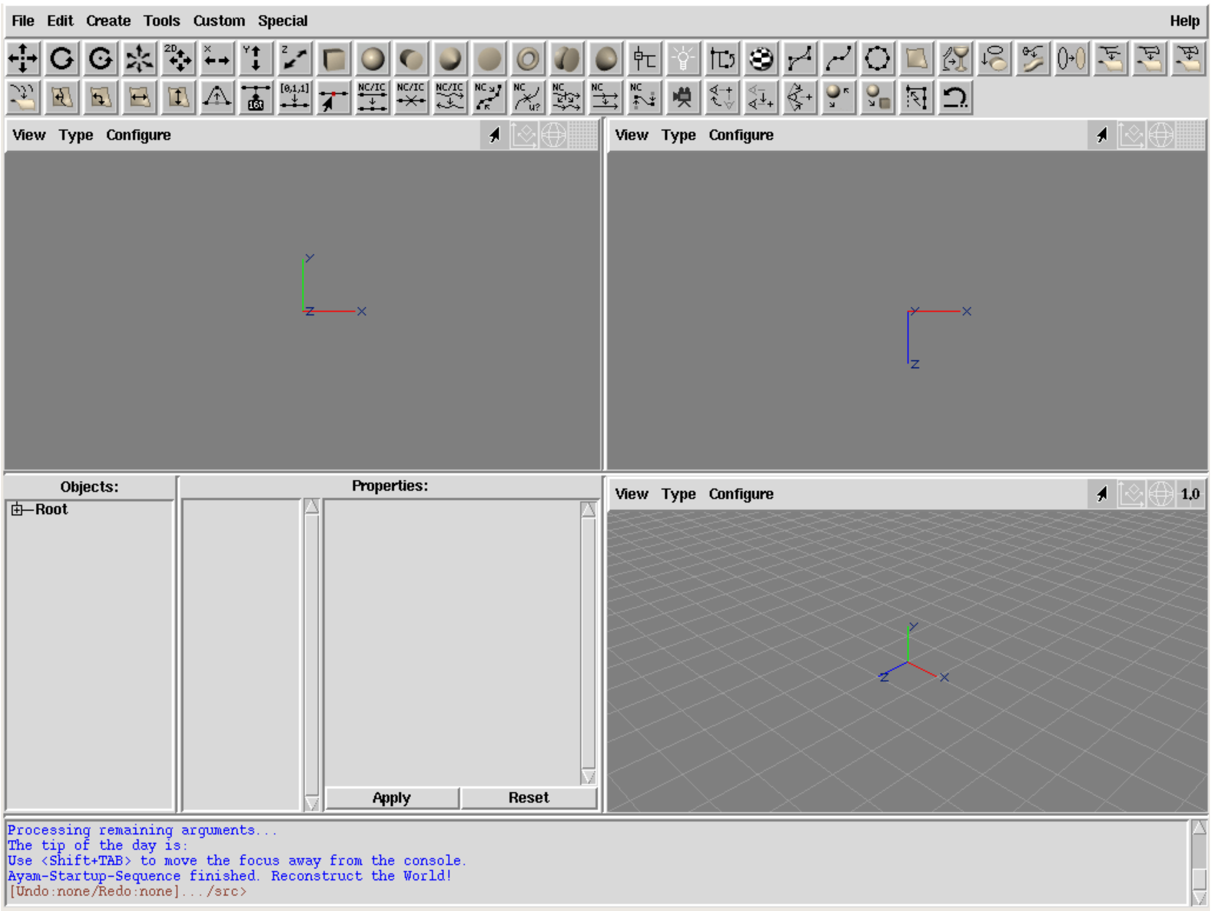



Figure 1: Single Window GUI Mode

Until Ayam 1.14 all windows, main, toolbox, and views were always single 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.

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")

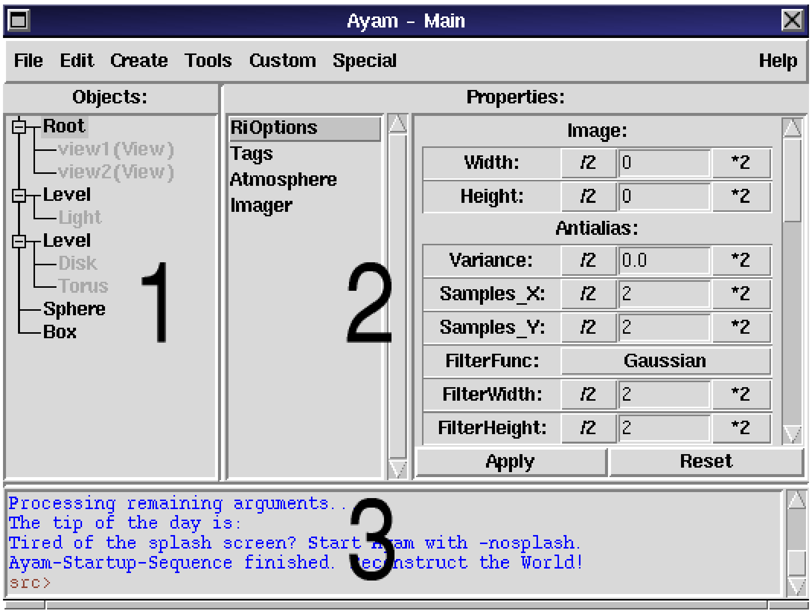



Figure 2: The Main Window

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 Root Object (page 77) for more information regarding this special object, and what it is good for. Note that one can not delete or copy this object.

**Object Tree View:**

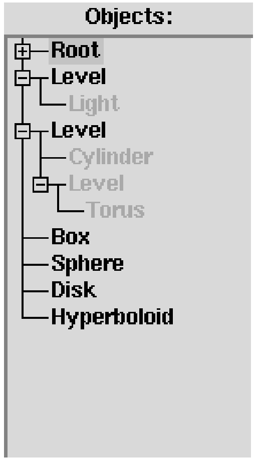



Figure 3: Object Tree View

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).

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 72) 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/Toggle Selected"` toggles display of all sub-levels of the selected objects,
- `"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 30).
- `"Help on Object"` displays the help of the selected object.

Since Ayam 1.6 the scene may be navigated and objects may be selected using the keyboard alone:

- <`Up`> and <`Down`> move the selection to the previous or next object.

- `<Shift-Up>` and `<Shift-Down>` will not move the selection, but rather extend it in the respective direction.[1]
- `<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.[2] 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).[3]

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.[4] 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.

**Object Listbox:**

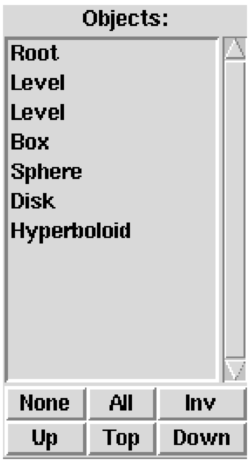



Figure 4: Object Listbox

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

[1] Since 1.7. [2] Since 1.11. [3] Since 1.18. [4] Since 1.15.

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).
- "Copy Object", "Cut Object", "Paste Object", "Delete Object" are standard clipboard operations as documented in section 2.2 Main Menu (page 30).
- "Help on Object" displays the help of the selected object.

Since Ayam 1.6 the scene may be navigated and objects may be selected using the keyboard alone:

- <Up> and <Down> move the selection to the previous or next object.
- <Shift-Up> and <Shift-Down> will not move the selection, but rather extend it in the respective direction.[1]
- <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.[2] 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>.

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.

### 2.1.2 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 you can select multiple entries, 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).

[1] Since 1.7. [2] Since 1.11. [3] Since 1.15.

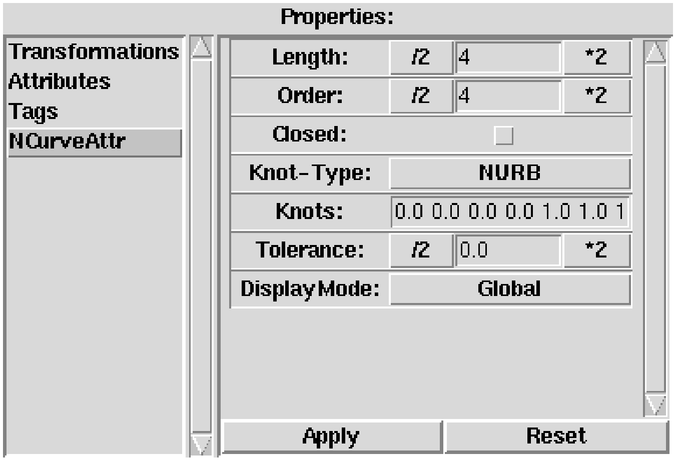



Figure 5: Properties

Also the keyboard may be used to select properties: just press one of the <0>-<9> keys (most comfortably using the numeric keypad). <0> always selects the last and often the only object type specific property, whereas <1> selects the first property, which often contains the standard transformations.[1]

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

If Ayam is in floating windows GUI mode and the elements of the current property GUI do not fit 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 2.9 Preferences (page 45)).

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.

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 72).

Here are some general remarks about object properties:

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.

Holding down the "`Shift`" button while interacting with the property GUI or pressing "`Return`" when entry widgets have the keyboard input focus will also lead to an instant apply.[2]

You can undo all changes to the arguments of a property that have been made after the last "`Apply`" operation with the "`Reset`"-button. This does, however, not use the undo mechanism of Ayam.

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

[1] Since 1.8 [2] Since 1.8.3.

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. You may accomplish this easily using the <Esc> key.

A property may be copied and pasted to another object, see the "Edit" menu. You can also paste property values to different types of properties (e.g. pasting parameters from a surface shader to the displacement shader) using "Paste to selected" in the "Special/Clipboard" sub-menu.

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".

A simple example for 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.

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

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.

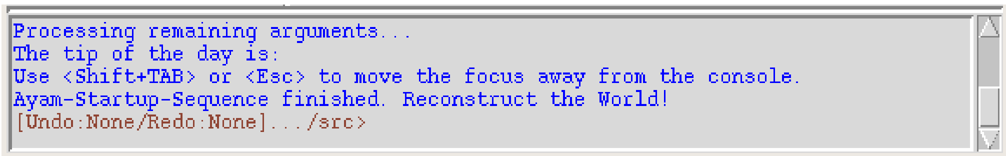



Figure 6: Console

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 2.9.5 Miscellaneous Preferences (page 52)).

One can of course also enter commands or even complete new Tcl procedures into the console. However, this is a feature for an advanced user that studied section 6 Scripting Interface (page 222). 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 automagically 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 one can 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 0 {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 222) 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 you may open with your 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.

Note that the main menu 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). Hit `<Esc>` to set the focus to the object selection widget and thus enable the main menu keyboard shortcuts.

One can adapt these shortcuts using the `"ayamrc"` file (see section 8.4 Ayamrc File (page 311)).

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!
- `"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 309) for a more detailed discussion.
  This operation cannot be undone!

  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 285).
- `"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 310) for a more detailed discussion.
  This operation cannot be undone!

  Also files supported by any of the import plugins may be imported using this route. See above (`"Open"`).[2]
- `"Save as"`, saves the current scene 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 311) for a more detailed discussion.

  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"`).[3]

[1] Since 1.13. [2] Since 1.13. [3] Since 1.13.

- `"Save"`, saves the scene. 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 311) for a more detailed discussion.
- `"Import/"`, since Ayam 1.13 this sub menu is initially empty. You need to load a plugin first to gain to the menu entries described here.
- `"Import/Apple 3DMF"`, import a scene from the Apple 3DMF format, see section 7.10 3DMF (Apple) Import (page 294) for more information.
- `"Import/AutoCAD DXF"`, import a scene from the AutoCAD DXF format, see section 7.6 AutoCAD DXF import (page 289) for more information.
- `"Import/Mops"`, import a scene from The Mops, see section 7.5 Import of Mops Scenes (page 289) for more information.
- `"Import/Rhino 3DM"`, import a scene from the Rhino 3DM format, see section 7.12 3DM (Rhino) Import (page 297) for more information.
- `"Import/Wavefront OBJ"`, import a scene from the Wavefront OBJ format, see section 7.8 Wavefront OBJ Import (page 292) 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 300) for more information.
- `"Export/"`, since Ayam 1.13 this sub menu initially only contains the `"RenderMan RIB"` entry. You need to load a plugin first to gain access to the other menu entries described here.
- `"Export/RenderMan RIB"`, exports the current scene to a RIB, asking which camera (which view) to use.
- `"Export/Apple 3DMF"`, export a scene to the Apple 3DMF format, see section 7.11 3DMF (Apple) Export (page 296) for more information.
- `"Export/Rhino 3DM"`, export a scene to the Rhino 3DM format, see section 7.13 3DM (Rhino) Export (page 299) for more information.
- `"Export/Wavefront OBJ"`, exports the current scene to a Wavefront OBJ file, see also section 7.9 Wavefront OBJ export (page 293).
- `"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 303) for more information.
- `"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"`. See section 8.6 Plugins Overview (page 317) 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 311) 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.
- `"Exit!"`, remove all temporary files, save preferences (if the preference setting `"Main/AutoSavePrefs"` is turned on) and quit the application.

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

- "Copy", copies the currently selected object(s) into the clipboard.
  This operation cannot be undone!
- "Cut", moves the currently selected object(s) into the clipboard.
  This operation cannot be undone!
- "Paste", copies the object(s) from the clipboard to the current level of the scene.
  This operation cannot be undone!

  Note that the content of the clipboard remains intact after this operation, this means that you can paste multiple times! You can move objects 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 4.10.2 Instances and the Object Clipboard (page 98).
- "Delete", removes the selected object(s) from the scene.
  This operation cannot be undone!
- "Select All", selects all objects in the current level (except for the root object).
- "Select None", de-selects all currently selected objects.
- "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!
- "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).
- "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.
- "Undo", perform undo operation (see section 8.1 The Undo System (page 306) for more information).
- "Redo", perform redo operation (see section 8.1 The Undo System (page 306) for more information).
- "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 objects).
- "Master", searches for the master object of the currently selected instance object and selects it for editing, see also section 4.10 Instance Object (page 97).
- "Preferences", opens the preferences dialog (see section 2.9 Preferences (page 45) for more information).

The "Create" menu entries let you create objects. In contrast to the object creation via the toolbox some menu entries present you with small dialogs, where you may adjust parameters for the object to be created. The entry fields in those dialogs support Tcl expressions as detailed in section 6.3 Expression Support in Dialog Entries (page 267). Here are the entries of the Create menu:

- "NURBCurve", create a new NURBS curve. A small dialog box will pop up, where you may specify the length of the new curve. See also section 4.22 NCurve Object (page 110). This dialog also contains a "AddArgs" entry field where you may specify additional command line arguments to the "crtOb NCurve" command, as outlined in section 6.2.2 Creating Objects (page 226).[1]
- "ICurve", create a new interpolating curve. A small dialog box will pop up, where you may specify the length of the new curve. See also section 4.23 ICurve Object (page 114). This dialog also contains a "AddArgs" entry field where you may specify additional command line arguments to the "crtOb ICurve" command, as outlined in section 6.2.2 Creating Objects (page 228).[2]
- "ACurve", create a new approximating curve. A small dialog box will pop up, where you may specify the length of the new curve. See also section 4.24 ACurve Object (page 116). This dialog also contains a "AddArgs" entry field where you may specify additional command line arguments to the "crtOb ACurve" command, as outlined in section 6.2.2 Creating Objects (page 230).[3]
- "NCircle", create a new NURBS circle. See also section 4.25 NCircle Object (page 118).
- "NURBPatch", create a new NURBS patch. A small dialog box will pop up, where you may specify the width and height of the new patch. See also section 4.29 NPatch Object (page 127). This dialog also contains a "AddArgs" entry field where you may specify additional command line arguments to the "crtOb NPatch" command, as outlined in section 6.2.2 Creating Objects (page 232).[4]
- "IPatch", create a new interpolating patch.[5] A small dialog box will pop up, where you may specify the width and height of the new patch. See also section 4.30 IPatch Object (page 130). This dialog also contains a "AddArgs" entry field where you may specify additional command line arguments to the "crtOb IPatch" command, as outlined in section 6.2.2 Creating Objects (page 235).
- "BPatch", create a new bilinear patch. See also section 4.31 BPatch Object (page 132).
- "PatchMesh", create a new patch mesh. See also section 4.32 PatchMesh Object (page 133).
- "Solid", create a new solid primitive object, for use in CSG. "Box", "Sphere", "Disk", "Cone", "Cylinder", "Torus", "Hyperboloid" or "Paraboloid" may be selected.
- "Level", creates a new hierarchy object. "Level" just groups objects, "Union", "Intersection", "Difference", and "Primitive" are CSG operations. See also section 4.7 Level Object (page 89).
- "Light", create a new light source. See also section 4.5 Light Object (page 82).
- "Custom Object", create a new custom object. If this sub-menu is empty no custom object has been loaded yet. See also section 4.51 Custom Object (page 182).
- "View", a new View window will be opened. See also section 4.3 View Object (page 79).
- "Instance", create an instance of the currently selected object, see section 4.10 Instance Object (page 97) for more information regarding instances.
- "Clone", create a clone object, see section 4.8 Clone Object (page 92)
- "Mirror", create a Mirror object, see section 4.9 Mirror Object (page 95)

[1] Since 1.19. [2] Since 1.19. [3] Since 1.19. [4] Since 1.19. [5] Since 1.20.

- `"Material"`, create a new material. A small dialog box will pop up, where you have to specify the name of the new material. See also section 4.6 Material Object (page 87).
- `"Camera"`, create a new camera. Camera objects may be used to temporarily save view camera settings, see section 4.4 Camera Object (page 81).
- `"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 4.12 RiInc Object (page 101).
- `"RiProc"`, create a new procedural object, see also section 4.13 RiProc Object (page 101).
- `"Script"`, create a new script object, see also section 4.50 Script Object (page 176).
- `"Select"`, create a new select object, see also section 4.11 Select Object (page 100).
- `"Text"`, create a new text object, see also section 4.48 Text Object (page 173).

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 267). 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 "ToolsSurface/Revert U"/-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 202).
- "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 you may let the merge-tool delete them immediately after the merging operation, by enabling the "RemoveMerged"-option. If the "OptimizeNew"-option is enabled, the "Optimize"-tool (see below) will be started after the merge operation with the newly created merged object as argument.
  - "Split": splits the faces from the selected PolyMesh objects off and into a second PolyMesh object. The faces to be split off are selected by selecting all their control points with the select points modelling action (see also section 3.9 Selecting Points (page 62)). The original selected PolyMesh objects will be changed, the selected faces will be removed. The changes to the original objects cannot be undone. 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). If the option "IgnoreNormals" is enabled, the optimize-tool will consider points with equal coordinates but differing normals to be equal (and optimize them). 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.

- "Points": sub-menu for tools that work on points:
  - "Select All Points", 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.
  - "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.

[1] Since 1.13.

  - "`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)`", "`Points/Center All Points (2D-YZ)`", "`Points/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!) regardles 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.
- "`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.
- "`Highlight Material`", colours all objects of the same material in red color in the tree view. This tool expects a selected material object or a normal object that has a material attached. It will not work for material objects that are not registered or have no normal objects that refer to them. It will also not work for objects that have no material assigned. If the reference counter of a material object is not zero, but "`Highlight Material`" reports 0 objects found, the referring objects probably reside in the object clipboard. You can clear the color from the tree view with the keyboard shortcut `<Ctrl+l>`.

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 disk. 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 on program startup and "`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 using "`Save Environment`" you can just save environment files that contain the root object and all views. If you want to include geometric objects in your environment or if you want to exclude the root object and just save views you have to 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.
- "`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.
- "`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! One may e.g. copy the 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 instances of the current level (and its child objects) to normal objects. If objects are selected, only those selected objects will be processed.[1]
- "`Instances/Automatic Instancing`", pops up a small dialog box, where you may parameterise and start the automatic instantiation algorithm (that automatically creates instances from equal objects). See section 8.8 Automatic Instancing (page 319) for more information regarding automatic instancing.
- "`Tags/Add RiOption`", pops up a small dialog box, where you may select and parameterise a RiOption tag to add as tag to the Root object (see 4.55.2 RiOption Tag (page 193)). The Root object does not have to be selected and the current selection will not be changed by this action.
- "`Tags/Add RiAttribute`", pops up a small dialog box, where you may select and parameterise a RiAttribute tag to add as tag to the currently selected object(s) (see 4.55.1 RiAttribute Tag (page 192)).
- "`Tags/Edit TexCoords`", opens the texture coordinates editor. (see also section 4.55.3 TC (Texture Coordinates) Tag (page 193)).
- "`RIB-Export/From Camera`", writes a complete RIB of the current scene with the camera transformations taken from the currently selected camera object. The size of the rendered image will be taken from the RiOptions of the root object. If they 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`".

[1] Since 1.14.

- "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 let you create RIBs that may be easily included in other RIBs using e.g. the ReadArchive directive.
- "RIB-Export/Create ShadowMap", creates the shadow maps from the currently selected light source. See also section 4.5.2 Using ShadowMaps (page 83).
- "RIB-Export/Create All ShadowMaps", creates all shadow maps for the current scene (regardless of selection). See also section 4.5.2 Using ShadowMaps (page 83).
- "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 8.7 Shader Parsing Plugins (page 318)) 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 2.9.1 Main Preferences (page 46)).
- "Scan Shaders" initiates the shader parsing with the built in shader parser or the currently loaded shader parsing plugin (see also section 8.7 Shader Parsing Plugins (page 318)).
- "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 311) 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 available 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 44)). From version 1.3 on, Ayam remembers the state of the toolbox in the saved preferences. This option is not available if Ayam is in single window GUI mode.
- "Toggle TreeView" toggles object tree view and object listbox. From version 1.3 on, Ayam remembers whether the tree view or the object listbox is open in the saved preferences (see also section 2.1.1 Objects (page 21) for more information about both representations).
- "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.
- "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, this feature will not work with frame redirects e.g. "http://www.ayam3d.org/";

use "http://ayam.sourceforge.net/docs/" or a "file:"-URL as base URL in the "Docs" preference setting instead!

- "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).

## 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. 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 22) and section 2.1.1 object list shortcuts (page 24).

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). Hit <Esc> first to set the focus to the object selection widget and thus enable the main window keyboard shortcuts.

One can adapt all these shortcuts using the "ayamrc" file (see section 8.4 Ayamrc File (page 311)).

## 2.4 Anatomy of a View

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. Those view menu keyboard shortcuts only work if the view window has the key-

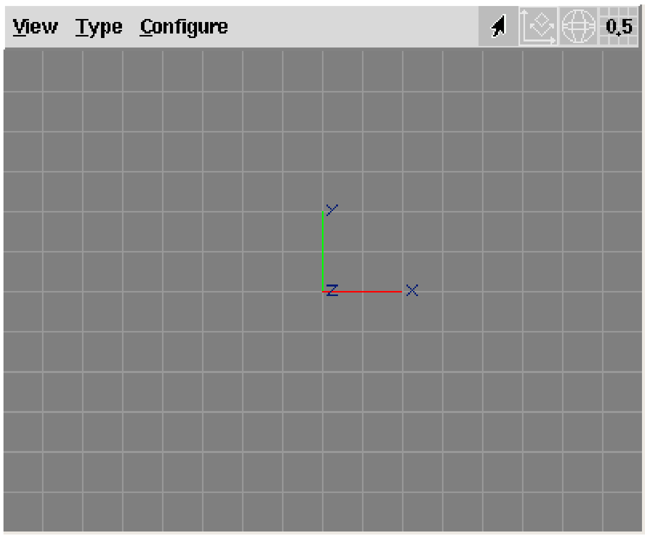



Figure 7: A View Window

board input focus. The shortcuts are adaptable using the "ayamrc" file (see section 8.4 Ayamrc File (page 311)).

Here are all meny 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 (see the preferences) will be called. Note that the RIB export will not use the RiOption settings for image size but the current window size instead. Also note that the environment variable SHADERS will be adapted to 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 (see the preferences) will be called. Note that the RIB export will not use the RiOption settings for image size but the current window size instead. Also note that the environment variable SHADERS will be adapted to the preference setting Shaders for rendering.
- "Redraw": forces the OpenGL-widget to be drawn, this is particularly useful if automatic redrawing of the view has been disabled.
- "Export RIB" exports the scene to a RIB. 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.
- "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 ShadowMap": creates the shadow maps for the currently selected light source. See also section 4.5.2 Using ShadowMaps (page 83).

- "Create All ShadowMaps": creates all shadow maps for the current scene (regardless of selection). See also section 4.5.2 Using ShadowMaps (page 83).
- "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 4.3 View Object (page 79), 2.6 View Window Shortcuts and Actions (page 41), and 3 Modelling Actions (page 55) for more information.

The "Configure" menu may be used to change preferences of the view. Some preferences are outlined in greater detail in section 4.3.2 ViewAttrib (page 80).

- "Automatic Redraw", toggles whether the view should be redrawn, whenever the scene changes. If this is disabled, you can force a redraw using the menu entry "View/Redraw" or the corresponding keyboard shortcut <Ctrl+d>.
- "Drawing Mode" determines whether the view should draw a wireframe representation ("Drawing Mode/Draw") or a shaded one ("Drawing Mode/Shade") or, new in Ayam 1.6, a representation where the curves of the draw mode are drawn over the shaded representation ("Drawing Mode/ShadeAndDraw").
- "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 3.19 Editing in Local Spaces (page 70).
- "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. You can also set this image 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.

- "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.
- "Double Size", change width and height to the double of the current values.
- "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.
- "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.

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

The "Modelling mode" icon menu may be used to quickly change the current modelling mode (global or local, see also section 3.19 Editing in Local Spaces (page 70)). 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 will currently modified by a modelling action (for points, a red dot will be present in the upper right corner of the icon). 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:

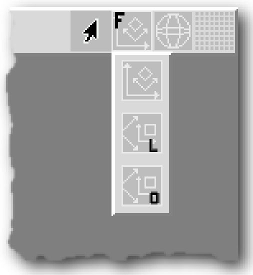



Figure 8: Modelling Mode Icon Menu

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

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:

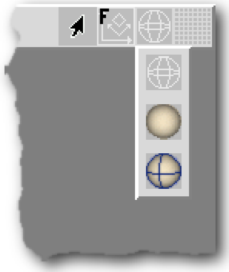


Figure 9: Drawing Mode Icon Menu

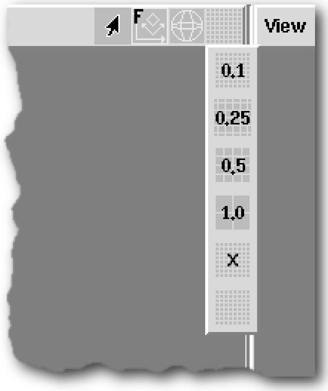



Figure 10: Grid Icon Menu

## 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.

One can adapt these shortcuts using the "`ayamrc`" file (see section 8.4 Ayamrc File (page 311)).

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

- `<Left>`, `<Up>`, `<Right>`, `<Down>` rotate viewer around origin.
- `<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 dependend default values.
- `<.>` pans the view to the mark.

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.
- <o> starts zooming the view. Moving the mouse up zooms in and moving 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.

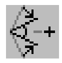
| Name | Shortcut | Icon |
|---|---|---|
| Pan View | <v> |  |
| Zoom View | <V> |  |
| Rotate View | <R> |  |

Table 1: 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.

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.

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

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

Using the menu entry "Zoom to Object" or the corresponding shortcut <Ctrl+o> (<O> for internal views) one can change the views camera settings 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.

Using the menu entry "Align to Object" or the shortcut <Ctrl+a> (<L> for internal views) one can change the views camera settings so that it 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 3.19 Editing in Local Spaces (page 70).

It is also possible to move through the scene hierarchy and change the selection directly in view windows:[4]

- <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 1.7. [2] Since 1.15. [3] Since 1.16. [4] Since 1.18.

## 2.7 Selecting Objects within a View

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

When the view windows action is "Pick" you can pick (select) objects that appear within this view with the mouse. You can invoke this action by pressing <P> or make 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>.

### 2.7.1 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 2.7.2 Drag-selecting Multiple Objects (page 43)

### 2.7.2 Drag-selecting Multiple Objects

You can select multiple objects using the <Control>+Click method described in section 2.7.1 Selecting Individual Objects (page 43). 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.

### 2.7.3 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.


Figure 11: List of Ambiguous Candidates

If the selection is ambiguous Ayam displays a window that contains a list of the possible candidates for selection. When you click a name in the list, 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.
It is also possible to use a double click in the list of selection candidates to select an object and immediately close the dialog.[1]

Notes:

- While the list of ambiguous candidates is opened you can not pick other objects within the views.
- It is possible to use the `"Zoom to Object"` action (shortcut <O>) 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 313)).

## 2.8 The Toolbox Window

The toolbox window displays some buttons that start interactive modelling actions, modelling tools, or create objects. You can open and close it using the main menu entry `"Special/Toggle Toolbox"`.

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 55).

[1] Since 1.11.


Figure 12: The Toolbox

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 313) for more information.

The toolbox is also open for extension by scripts, see section 6.4.4 Script Examples (Toolbox Buttons) (page 272) 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.

## 2.9 Preferences

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,

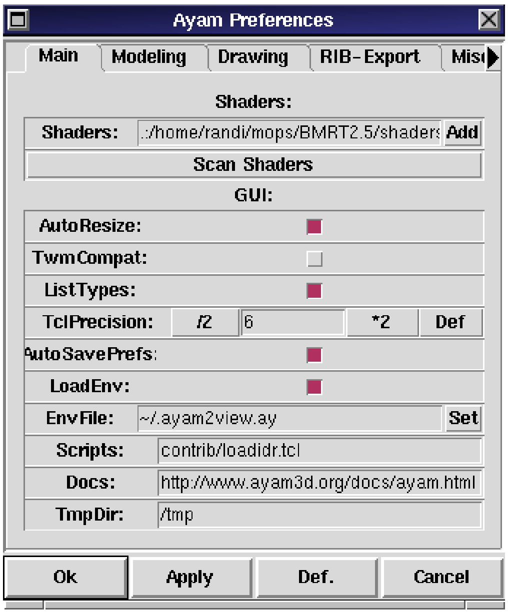



Figure 13: Preferences Dialog

- "`Apply`" to apply the changes, but leave the editor open,
- "`Revert`" to reset to the settings that have been loaded on program startup (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.

### 2.9.1 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, you may easily add another path, simply press the button and select a file from the directory you want to add. Note that currently the filename of a compiled shader may contain just two "`.`" at maximum. 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 shaders as Ayam through the "`Shaders`" preference setting.

  Note that this option does of course not affect any renderers that are started outside the Ayam context.

[1] Since 1.14.

- "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 section 4.54.4 Shader Parsing (page 191) 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 your operating system, window manager, or their settings, this may only work correctly, when a window of Ayam already has the focus. On some operating systems or window managers AutoFocus might not work at all. Furthermore, note that moving the focus to a window might also raise it.

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

  While the preference editor is open, AutoFocus is 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 MacOSX Aqua.
- "AutoSavePrefs", if this is switched on, Ayam will write the preferences settings to the ayamrc file when the program quits.
- "BakOnReplace", if this is switched on, Ayam will make a backup copy of each scene file it loads 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"; This file typically contains some view objects to create a standard 2-, 3-, or 4-view working environment and is automatically loaded upon startup of Ayam (if the hidden preference option "LoadEnv" is switched on) and upon clearing the scene using "File/New" (if the hidden preference option "NewLoadsEnv" is switched on). See also section 8.4.2 Hidden Preference Settings (page 313).

- "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 script from the scripts directory and the ayslx shader parsing plugin from the plugins directory. If non absolute paths are used they are 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.
- "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 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
  ```

  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.
- "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 313).

### 2.9.2 Modelling Preferences

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

- "PickEpsilon" is used by the single point editing actions (see section 3.10 Editing Points (page 63)) to determine which point of an object has been selected. A smaller "PickEpsilon" means 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 "PickEpsilon" value in a way that for zoomed-in views a smaller value will be used (and vice versa).

  In older versions of Ayam (prior to 1.8), a value of 0.0 was allowed and default. This is no longer the case, the new default value is 0.05.
- "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>.

[1] Since 1.16.

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 automagically (see also section 8.2 The Modelling Concept Tool-Objects (page 306)).

- "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.
- "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.
- "GlobalMark" toggles whether each view should manage its own mark (off, default), or whether there should be just one global mark (on). 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.
- "DefaultAction" determines the modelling action that will be active after a press of the <Esc> key in a view window.
- "UndoLevels" determines the number of modelling steps that should be saved in the undo buffer. Useful values range from 2 to 50 (depending on how much memory you like to spend). If you set "UndoLevels" to -1, the undo system will be disabled completely. For more information, see also the section 8.1 The Undo System (page 306).

### 2.9.3 Drawing Preferences

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

- "Tolerance" is in fact GLU sampling tolerance, used to control the quality of the sampling when rendering a NURBS curve or NURBS patch using GLU. Smaller tolerance settings lead to higher quality. Useful values range from 1 to 100. This setting has no effect for objects that override it using a local tolerance setting different from 0.

  Using the keyboard shortcuts <F2> and <F3> the GLU tolerance may also be set easily.
- "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 tesselation (OutlinePoly), or just the outlines of the patch (OutlinePatch).

Note that this setting does not affect shaded patches.

Note also, that this setting has no effect for objects that override it using a local DisplayMode setting 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, The control hull (control polygon) or the curve or a combination of both may be displayed.

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

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

- "UseMatColor" determines, whether the shaded representation uses the color defined by the material of an object for rendering.
- "Background", "Object", "Selection", "Grid", "Tag", "Shade", and "Light" let you set colors that will be used when drawing or shading.

### 2.9.4 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 filenames have special meaning: If you set "RIBFile" 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 you set "RIBFile" to "Scenefile", the leading path will be stripped from the scene name additionally. Use "Scenefile", if you render with shadow maps. This way the scene will use relative paths to load the shadow maps and you may move the RIBs around more easily.

  "Ask" is another special setting, that allows to select a different filename each time you export a RIB file. 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 you set "RIBFile" 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, filenames 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 mouseclicks 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 you render the exported RIB file. You may set it to "RIB", this 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 you export to a RIB file. 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 RiSurface "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.
- "RIStandard" 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.
- "ShadowMaps" determines, whether shadow maps should be used, when writing light sources. It is not sufficient to switch this on to render using shadow maps, light sources that shall use shadow maps have to be parameterised as well, see section 4.5.2 Using ShadowMaps (page 83). If "ShadowMaps" is set to "Automatic", the exported RIBs will automatically render and use all shadow maps; if it is set to "Manual", the shadow maps will be rendered on user request only (e.g. using the view menu entry: "View/Create All ShadowMaps"). "Manual" should be used, when rendering directly from view windows with shadow maps.
- "ExcludeHidden" causes hidden objects not to be exported to RIB files.
- "RenderMode" allows to switch between two different methods of forcing a renderer to render to the screen (via a RiDisplay statement in the exported RIB, necessary for e.g. PRMan and RDC; or via a command line argument, e.g. -d for rendrib from BMRT).
- "QRender" determines the command that should be executed, upon quick rendering a view, %s denotes the name of the RIB file.
- "QRenderUI", enables the Rendering GUI for quick rendering, see discussion of "RenderUI" below.
- "QRenderPT", progress template for quick rendering, see discussion of "RenderPT" below.
- "Render" determines the command that should be executed, upon normal rendering of a view, %s denotes the name of the RIB file.
- "RenderUI" enables the renderer user interface (Rendering GUI), which consists of a simple progress bar, a label that displays the estimated or elapsed rendering time, a checkbutton to control ringing the bell when the rendering is finished, and a cancel button. This GUI is displayed when a renderer is invoked directly from a view window using the "Render" view menu entry (or the equivalent keyboard shortcut). Proper work of this GUI 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 you may also use an internal kill command "w32kill" that has been introduced in Ayam 1.4. See also section 8.4.2 Hidden Preference Settings (page 313).

- "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 special symbol "%d" denotes the position of the percentage number in the output of the renderer. For rendrib from BMRT2.6 this should be set to "R90000 %d" and the special command line option "-Progress" should be used. For rendrib from BMRT2.5 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, a second variant of parsing the output using regular expressions is available since Ayam 1.6. In this case, the progress template should be a complete regexp command for Tcl that parses the string contained in the variable named "string" and puts the parsed progress number into a variable named "progress". Here is an example that works with Pixie-1.2.1, which outputs strings like "fish.rib (222):  - 10.00 percent":

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

- "SMRender", renderer to use for the rendering of shadow maps (e.g. view menu entry "View/Create All ShadowMaps"), see also section 4.5.2 Using ShadowMaps (page 83). %s denotes the name of the RIB file.
- "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.
- "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 "shader" to light shader names for lights that use shadow maps upon RIB export. Not changing the shader names is necessary for Gelato.
- "PPRender" is the name of the renderer to use for the permanent preview feature (see also section 2.5 View Menu (page 37)). 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 you can set many renderer related preferences at once using the select renderer tool via the main menu "Special/Select Renderer" (see also section 2.2 Special Menu (page 35)). In fact, using "Special/Select Renderer" first, then fine tuning your renderer setup using the preferences editor is the suggested way to switch Ayam to a certain RenderMan renderer.

### 2.9.5 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 error messages should be written to the file specified by "LogFile". If this is enabled, you should clear the log manually from time to time, as Ayam will always append to "LogFile".
- "LogFile"; see above.
- "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.

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

- "SaveAddsMRU"; if this is switched on, saving to a file will add that file to the most recently used files list in the main menu for quick access.
- "TclPrecision"; this is the precision Tcl handles floating point numbers with. You may want to decrease this number 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 you may lose model precision 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.
- "SavePrefsGeom" controls when the geometry of the preferences editor should be remembered by Ayam, "Never": the window is always opened in standard size, centered on the screen; "WhileRunning": the window width and position will be remembered as long as Ayam is running; "Always": the window width and position will be remembered in the saved preferences, thus, also surviving a restart of Ayam. Note that the height of the preferences window will always be adapted to the currently open preferences section, no matter how "SavePrefsGeom" is set.
- "SMethod"; is the sampling method used by the NURBS to PolyMesh (tesselation) facility (based on GLU V1.3+). Five methods are available:

  1. "DomainDistance" (the default) simply tesselates the NURBS 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 length in parameter space.
  2. The tesselation method "PathLength" ensures that no edge of a polygon generated by the tesselation is longer than the value specified by "SParamU" and the tesselation method
  3. "ParametricError" ensures that the distance between the tesselated surface and the original surface is no point bigger than the value specified by "SParamU".
  4. "NormalizedDomainDistance" ensures that the tesselation creates the same number of sample points (as given via "SParamU" and "SParamV") for any knot vector and
  5. "AdaptiveDomainDistance" additionally adds sample points dependend on the number of control points (width or height of the patch) to provide a better adaptation to complex patches.

The tesselation methods,"`NormalizedDomainDistance`" and "`AdaptiveDomainDistance`" are available since Ayam 1.9 to overcome the limitations of the "`DomainDistance`" method.

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

- "`SParamU`"; is a parameter for the sampling method above. The default value for the sampling method "`DomainDistance`" is 10. Higher values lead to better quality and more tesselated polygons. The default value for the sampling method "`PathLength`" is 30. Smaller values lead to better quality and more tesselated polygons. The default value for the sampling method "`ParametricError`" is 0.5. Smaller values lead to better quality and more tesselated polygons.
- "`SParamV`"; is just available for the tesselation modes "`DomainDistance`", "`NormalizedDomainDistance`", and "`AdaptiveDomainDistance`".

# 3 Modelling Actions

Before invoking any modelling action you should select one or more objects 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: you can have e.g. a view, where you move objects, a second where you rotate objects and a third, where you pick objects. The layout, drawing style, and grids may also differ between the different views. Together with the feature, that you may change the selection while actions are active and even pan and zoom views (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"` or `"Pick"` (depending on the preference setting `"Modelling/DefaultAction"`). See section 2.7 Selecting Objects within a View (page 43) for more information about picking objects.

Note that the modelling actions are not available in perspective views.

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.

You may undo/redo the effects of a modelling action using `<Ctrl+z>` and `<Ctrl+y>` (see section 8.1 The Undo System (page 306) for more information).

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

Also note that you may use the middle and rightmost mouse button 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 41).

## 3.1 Modelling Actions Overview

This section gives an overview over the various modelling actions. Note that a more complete collection of shortcuts is available as separate reference card document in the Ayam distribution.

| Name | Shortcut | Icon | Name | Shortcut | Icon |
|---|---|---|---|---|---|
| *Transform:* | | | | | |
| Move | <m> | | Rotate | <r> | |
| Move X | <mx> | | Rotate About | <ra> | |
| Move Y | <my> | | Set Mark | <a> | |
| Move Z | <mz> | | | | |
| 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> | |
| *Edit:* | | | | | |
| Edit | <e> | | Tag Points | <t> | |
| Edit Weights | <w> | | Insert Points | <i> | |
| Edit Numeric | <E> | | Delete Points | <d> | |

Table 2: Modelling Actions Overview

## 3.2 Transforming Objects or Selected Points

Many modelling actions either work on objects transformation attributes or coordinates of selected points. Before Ayam 1.18 this was controlled implicitly by the point selection: if points were selected, they would be transformed instead of the object.

There are now two keyboard shortcuts (<o> and <p>) that let the user control this behaviour directly per view:[1] <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:

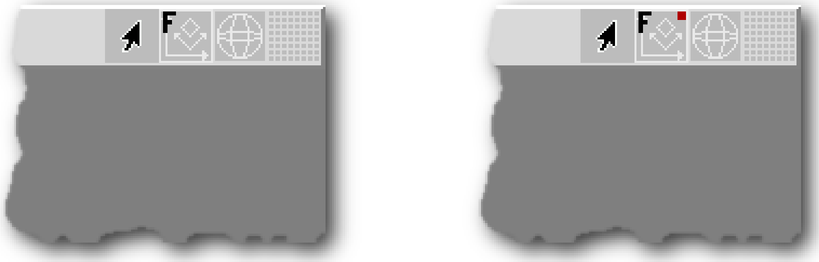


Figure 14: View Transforming Objects (left) and Points (right)

See also the section 6.5.7 Automatic Point Actions (page 276) for a script, that, partially, brings back the old implicit behaviour.

## 3.3 Moving Objects or Selected Points

Using the modelling action "Move" (shortcut: <m>) you may move selected objects or the selected (tagged) points of the selected objects.

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>.[2]

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

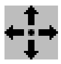
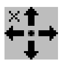
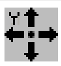
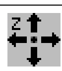
| Name | Shortcut | Icon |
|---|---|---|
| Move | <m> |  |
| Move X | <mx> |  |
| Move Y | <my> |  |
| Move Z | <mz> |  |

Table 3: Move Actions Overview

[1] Since 1.18. [2] Since 1.17.

## 3.4 Rotating Objects or Selected Points

Using the modelling action "`Rotate`" (shortcut: <`r`>) you may rotate objects or the selected (tagged) points of the selected objects.

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.

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.

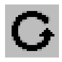
| Name | Shortcut | Icon |
|---|---|---|
| Rotate | <`r`> |  |

Table 4: Rotate Action

## 3.5 Rotating Objects or Selected Points About a Point

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

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

| Name | Shortcut | Icon |
|---|---|---|
| Rotate About | <`ra`> |  |

Table 5: Rotate About Action

To rotate about a different point, one needs to restart the intermediate set mark action (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 grids with this action.

See also the section 6.5.6 Automatic About Center Actions (page 275) for a script, that modifies the rotate action to rotate about the current selections center automatically.

## 3.6 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 big 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 one needs to press <x>, <y>, or <z> 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.

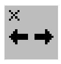
| Name | Shortcut | Icon |
|---|---|---|
| Scale 3D | <S> |  |
| Scale 2D | <s> |  |
| Scale 1D X | <sx> |  |
| Scale 1D Y | <sy> |  |
| Scale 1D Z | <sz> |  |
| Stretch 2D | <T> |  |

Table 6: Scaling Actions Overview

## 3.7 Scaling Objects or Selected Points About a Point

Using the scale about modelling actions one may interactively scale objects or the selected (tagged) points of the selected objects 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.

To scale about a different point, one needs to restart the intermediate set mark action (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.

| 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 7: Scaling About Actions Overview

Also note, that the three dimensional scaling about the mark occurs in all three dimensions, however, in the interactive set mark action one is only able to specify two dimensional coordinates using a mouse click (the third coordinate is always set to zero). To specify a true three dimensional mark one can use the corresponding view object attributes or select a point and set the mark to it (shortcut <C>).

To avoid degenerated coordinates due to roundoff errors it is highly suggested to use grids with this action.

See also the section 6.5.6 Automatic About Center Actions (page 275) for a script, that modifies the scale actions to scale about the current selections center automatically.

## 3.8 Setting the Mark

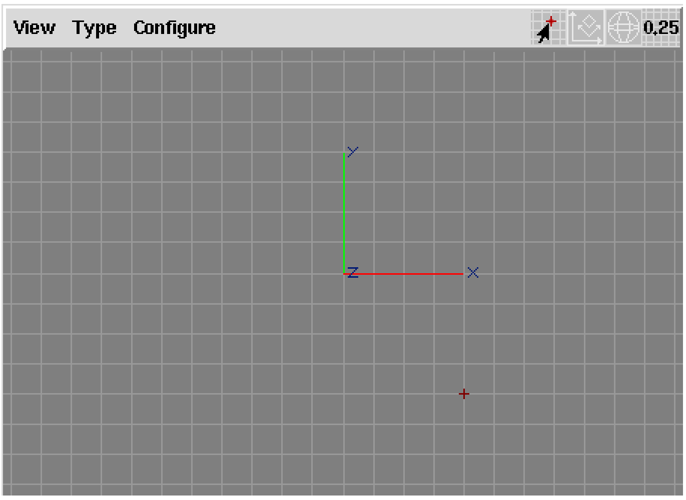



Figure 15: Front View With Mark Set At 1,-1,0

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 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.

See also the table below.

| Name | Shortcut | Icon |
|---|---|---|
| Set Mark | `<a>` |  |

Table 8: Set Mark Action

A single click with the mouse sets the mark. If grids are active in the view, the mark will be snapped to the nearest grid coordinates. Additionally, when this 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.
- `<b>` set the mark to the center of the bounding box of all currently selected points. The image below demonstrates the difference between the center of the bounding box and the center of gravity.

The marked point will be shown as a little red cross in the view window (see also the image below).

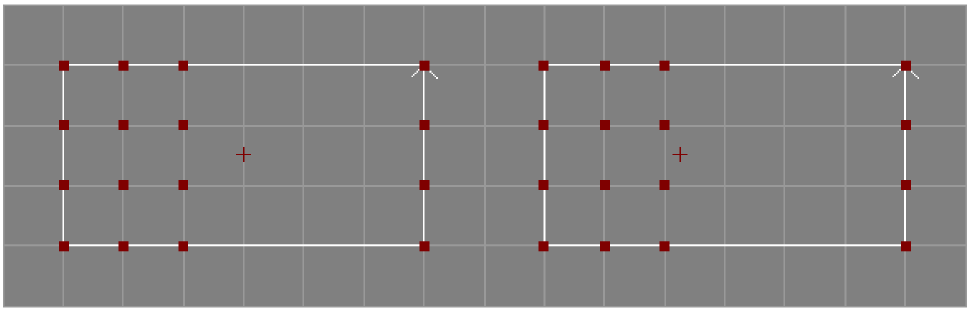


Figure 16: Bounding Box Center Mark (left) and Center of Gravity Mark (right)

The mark may also directly be set using the corresponding view object property (see also section 4.3.2 ViewAttrib Property (page 80)).

Another way of setting the mark is via the "FindU" special action for curves (see also section 3.16 Finding Points on Curves (page 68)) or the "FindUV" special action for surfaces (see also section 3.17 Finding Points on Surfaces (page 68)) .

Also note, that, using a mouse click, one is only able to specify two dimensional mark coordinates (the third coordinate is always set to zero).

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 `<.>`).

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

## 3.9 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.

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).

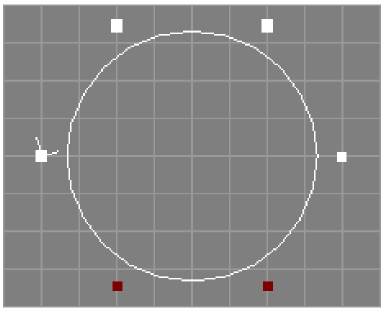


Figure 17: Object With Selected Points (red)

The selected editable points may be modified subsequently using the modelling actions Move, Rotate, and Scale as discussed above when the view is switched to point transformations using the shortcut <`p`>. See also the table below.

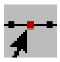
| Name | Shortcut | Icon |
|---|---|---|
| Tag Points | <`t`> |  |

Table 9: 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 iself 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.

One may also add a bigger number of points 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.[2]

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 one can use drag select to select all points nevertheless).

### 3.10 Editing Points

To edit the points of an object three actions (`"Edit"`, `"Edit Weights"`, and `"Numeric Point Edit"`) are available. All 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).

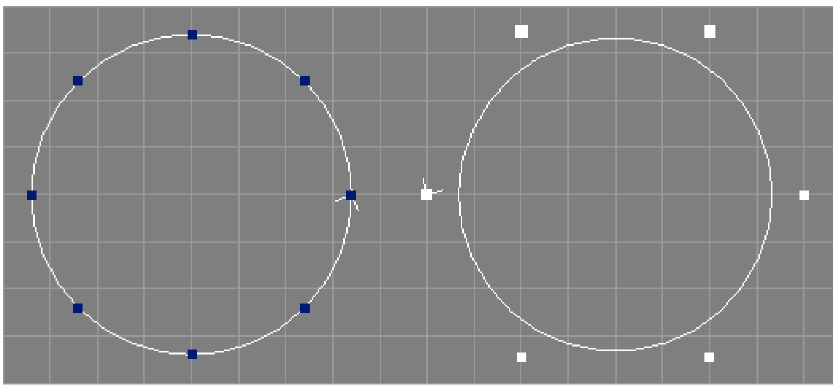


Figure 18: Objects With Read Only (left) and With Editable Points (right)

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 dif-

[1] Since 1.16. [2] Since 1.16.1.

ferent 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.

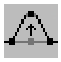
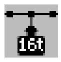
| Name | Shortcut | Icon |
|---|---|---|
| Edit | <e> |  |
| Edit Weights | <w> |  |
| Numeric Edit | <E> |  |

Table 10: Edit Actions 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. you can not move single points apart from a multiple point using the point edit 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 MacOSX Aqua.
- The modelling action "Edit Weights" (shortcut: <w>) changes the w coordinate of a single point by dragging the mouse left or right. The weights may be reset for all points using the shortcut: <W>. Furthermore, it is not possible to edit the weights of multiple selected objects, only the first selected object is considered. If a NURBS curve has multiple points, this action modifies all single points that make up the multiple point.
- 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 you may change the coordinates of the selected editable point(s) by entering new coordinate values on the keyboard (see image below).

  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, 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.

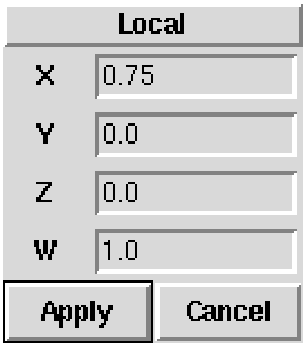



Figure 19: Numeric Point Edit Dialog

This modelling action may affect multiple points of multiple selected objects.[1]

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.[2]

Single coordinate entry fields may be empty to prevent a change of the corresponding coordinate value.[3]

If a multiple point is edited, this action modifies all single points that make up the multiple point, i.e. you can not move single points 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.[4]

Furthermore, it is not necessary that the original object stays selected while working with the numeric point edit dialog, you may select other objects 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. 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 2.9.5 Miscellaneous Preferences (page 52)).

## 3.11 Snapping Points to the Grid

There are two actions available for snapping points to the current grid of a view.[5]

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

[1] Since 1.18. [2] Since 1.18. [3] Since 1.20. [4] Since 1.4. [5] Since 1.11.

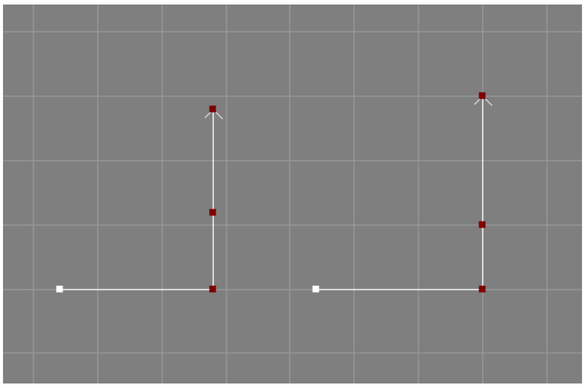


Figure 20: Snapping Points to the Grid

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.

## 3.12 Snapping Points to the Mark

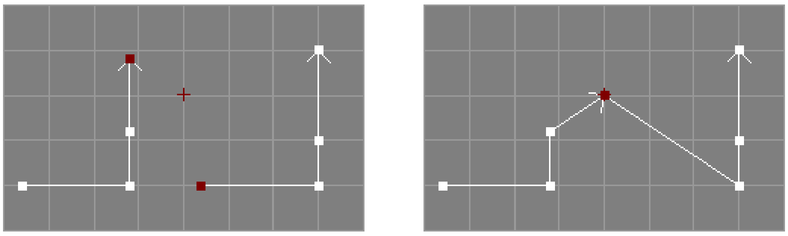


Figure 21: Snapping Points to the Mark

The snap points to mark action moves all selected points to the current mark.[1]

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 3.8 Setting the Mark (page 60) for more information about the mark) immediately. This action can be used without affecting other active actions. See also the image above.

## 3.13 Snapping Objects to the Mark

The snap objects to mark action moves all selected objects to the current mark.[2]

This action is initiated using the shortcut <K>. All selected objects are snapped to the current mark coordinates (see section 3.8 Setting the Mark (page 60) 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.

## 3.14 Inserting or Deleting 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. 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 5.24 Insert Knot Tool (page 211).

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 | <i> |  |
| Delete Points | <d> |  |

Table 11: Insert/Delete Points Actions

## 3.15 Manipulating the Multiplicity of Points

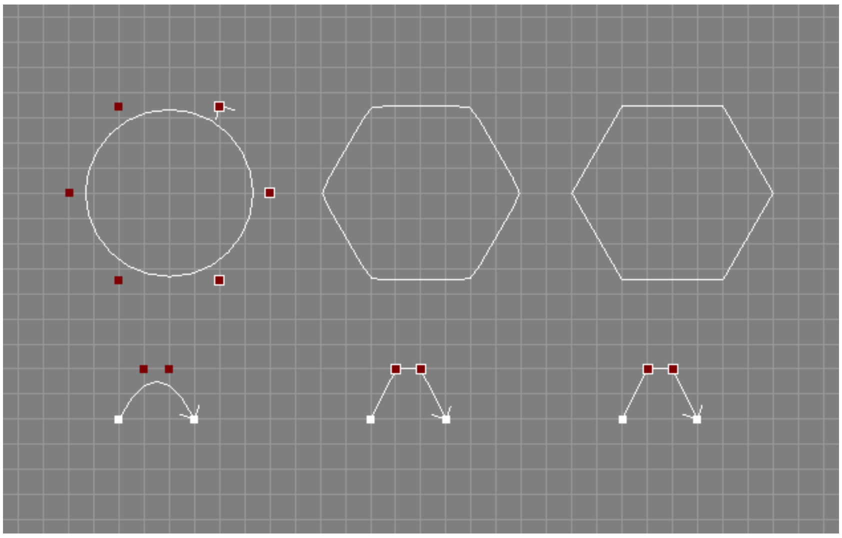


Figure 22: Successively Increasing the Multiplicity of Selected Points

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).

[1] Since 1.20.

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.

## 3.16 Finding Points on Curves

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 a 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 position of the picked point (see section 3.8 Setting the Mark (page 60) for more information about the mark). Remember to exactly pick a point on the curve or nearby, otherwise the calculation may fail. See also the table below.

| Name | Shortcut | Icon |
|---|---|---|
| Find U | <u> |  |

Table 12: Find Point on Curve Action

## 3.17 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 position of the picked point (see section 3.8 Setting the Mark (page 60) 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. See also the table below.

| Name | Shortcut | Icon |
|---|---|---|
| Find UV | <U> |  |

Table 13: Find Point on Surface Action

## 3.18 Interactively Splitting Curves

The modelling action "`Split Curve`" (shortcut <c>) may be applied to NURBS curves (objects of type NCurve) only. Using this action you may split a NURBS curve into two new curves at a point on the curve that may be specified by picking a point on the curve. 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!

## 3.19 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 view used).

However, if a view is *aligned* and switched to *local*, one can also edit in local object space. This means one can e.g. edit a planar parameter curve of a skin object where both objects (curve and skin) are rotated and scaled arbitrarily and make sure that the parameter curve remains planar all the time.

In other words, this means that the input plane of an aligned local view will match the XY-, ZY-, or XZ-plane of the local object space, depending on the type of the view (`"Front"`, `"Side"`, or `"Top"`). Editing and other modelling actions take place in that plane.

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).

How to make a view local and align it?

All you need to do is to first select an object and then press <`l`> *twice* to make the view object-local (one can also use <`Ctrl+l`> in external views). In Ayam versions prior to 1.18 one also needed to manually align the view 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, if the camera of the view is e.g. changed by a view action (however, realigning is also always possible by cycling the modelling mode three times). But let us get back to our local view.

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

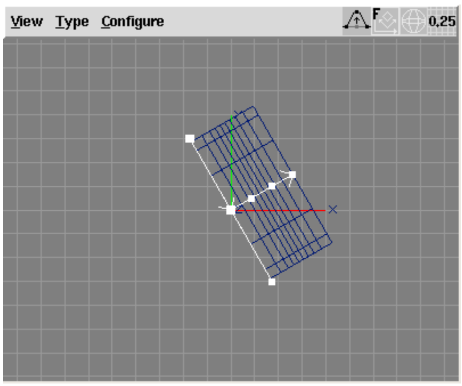



Figure 23: Global Front View with Rotated Sweep

A Sweep object with a circular B-Spline curve as cross section (rotated about the y axis by 90 degrees) 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 <`l`>.

The view has been switched to level-local using a single press of the <`l`> 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

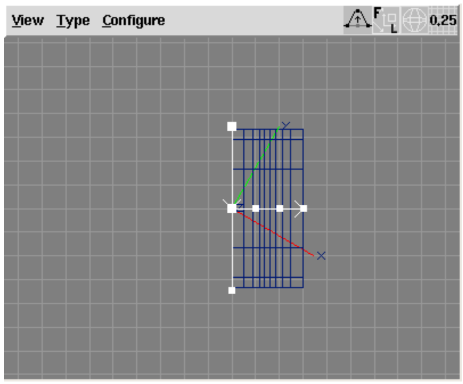



Figure 24: Local (Level) Aligned View

the Sweep object (note the Root object coordinate system, it is tilted). The grid is also rotated (with respect to the root coordinate system) but now much more useful for editing the trajectory, in fact it is perfectly aligned to the trajectory.

Press <l> again.

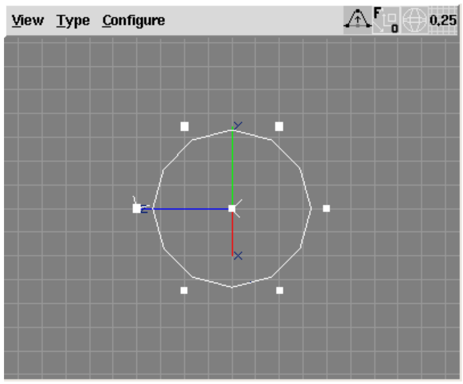



Figure 25: Local (Object) Aligned View

Now the view is object-local and aligned to the first of the selected objects (the cross section curve), note again 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 again displays a circle. Note again the orientation of the Root object coordinate system. One can safely edit the cross section curve, it will stay planar.

Note, that object-local aligned views only align to the first of multiple selected objects; if all selected objects have 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>.

# 4 Objects, Properties, and Tags

This section informs you 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.

Documentation on the standard properties (Transformations, Attributes, Material, Shader, and Tags) can be found in section 4.54 Standard Properties (page 189).

Furthermore, this section contains information about all tag types, see section 4.55 Tags (page 192).

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 14: 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(s) must rather provide an object of the appropriate type (see also section 8.2 The Modelling Concept Tool-Objects (page 306));
- **Material**: the object can be associated with a material;
- **Converts to/Provides**: type of converted/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 15: 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 the scene apart from geometric objects:

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

Table 16: Scene Organization

### 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 17: CSG/Solid Primitives

### 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 18: Freeform Curves

### 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 |
| BPatch | No | Yes | NPatch | Yes |
| PatchMesh | No | Yes | NPatch | Yes |

Table 19: Freeform Surfaces

### 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 20: Curve Tool Objects

### 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 | 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 21: Surface Tool Objects

### 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 22: Polygonal and Subdivision Objects

### 4.1.8 Scripts and Plugins

These objects create/modify arbitrary other objects from scripts or define 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 |

Table 23: Scripts and Plugins

## 4.2 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 you hide the Root object, 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 SavePaneLayout tags (see also section 4.55 Tags (page 192).

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

### 4.2.1 RiOptions Property

The RiOptions 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 you disable this option, be sure to add atleast one RiDisplay tag to the root object (see also section 4.55.6 RiDisplay Tag (page 196)), otherwise your 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 tesselated geometry.

### 4.2.2 Imager, Atmosphere Property

The Imager and Atmosphere properties let you define shaders for the Root object, please refer to section 4.54.4 Shader Properties (page 190) 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.

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

### 4.2.3 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/RIStandard".

The Root object is the only object to support RiOptions, RiHider, and RiDisplay tags (see also section 4.55 Tags (page 192).

## 4.3 View Object

Every view window (see also section 2.4 Anatomy of a View (page 37)) has a corresponding view object as a child object of the root object. You can change camera settings, the type of the view, and other things related to the view using the properties of the view object. 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 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 3.19 Editing in Local Spaces (page 70)).[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 you may position the camera, pick objects, and select points.

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

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 using drag and drop in the tree view 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 using drag and drop in the tree view 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 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).[2]

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 24: View Object Capabilities

The next sections detail the properties of the view object.

### 4.3.1 Camera Property

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

[1] Since 1.4. [2] Since 1.8.

- "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. 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. 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.

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.

### 4.3.2 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 wireframe, "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 wireframes.
- "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.
- "ModellingMode" enables editing in local object spaces. See also section 3.19 Editing in Local Spaces (page 70).
- "DrawBG" controls whether the background image should be drawn.
- "BGImage" is the name of a TIFF file, that will be used as texture for the background image. Ayam will read this image once when you apply the changes to the "ViewAttrib" property but 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.
- "EnableUndo" is switched on by default and allows to disable undo for view actions, e.g. panning or zooming a view, these actions then also do not change the scene changed state.

## 4.4 Camera Object

Camera objects are used to temporarily save camera settings of views. Therefore, they has just have two properties explained above, see sections 4.3.1 Camera (page 79) and 4.54.2 Attributes Property (page 190).

When a view object is dropped onto a camera object using drag and drop in the tree view, the camera settings from the view will be copied to the camera object.

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 25: Camera Object Capabilities

### 4.4.1 RIB Export

Camera objects never appear in RIB output.

## 4.5 Light Object

Light objects let you bring light into your scenes.

The behaviour of a light source object depends heavily on the type of the light source. There are currently four different light types available in Ayam: "Custom", "Point", "Distant" and "Spot".

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.

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 the standard light sources (see section 8.15 references (page 325)).

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 you can not preview the effect of a light source 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 show you 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 26: Light Object Capabilities

### 4.5.1 LightAttr Property

Depending on the type of the light source, the light attribute property contains different parameters. Parameters that are not displayed will not be used on RIB export, consequently.

Using "Type" you can change the type of the light source. When you change the type of a light source, the property GUI will be adapted to show only the options available for the new light source type, but only after you used the "Apply"-button.

"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 attribute "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 PRMan or 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 interactive actions (page 55)).
- "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, you may drop the light source object 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.

### 4.5.2 Using ShadowMaps

Using shadow maps requires the global preference setting "RIB-Export/ShadowMaps" to be switched on. 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 "<scene>.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 you 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 you may move the exported scene 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". Your 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 render the shadow map to the display and simply stop.

**Manual Creation of ShadowMaps**

If the preference setting "RIB-Export/ShadowMaps" is set to "Manual", the exported scene will not render the shadow maps but rather expects them to be present already. You can create them manually (hence the name "Manual") 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". The manual approach has the advantage, that the shadow maps will not be re-created each time you render the scene.

**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 get shadows.

**ShadowMap Mini Tutorial**

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".

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.

### 4.5.3 Using AreaLights

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.

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:

```
+-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.

## 4.6 Material Object

Material objects are used to attach RiAttributes and shaders to geometric objects.

When geometric objects are dropped onto a material object using drag and drop in the tree view they will be connected to this material object.

While 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 27: Material Object Capabilities

### 4.6.1 RiAttributes Property

Using this property standard and BMRT specific 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 you set one of the entries to a negative value (e.g. -1), the color will not be set at all for this object.
- `"Opacity"`, the opacity of the object, the default 255 255 255 means the object is totally opaque. If you set one of the entries to a negative value (e.g. -1), the opacity will not be set at all for this object.
- `"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 it's 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 it's surface shader, that is used in shadow calculations.
- `"Camera"`, `"Reflection"`, and `"Shadow"` toggle visibility attributes.

### 4.6.2 Surface, Displacement, Interior, Exterior Property

These properties let you define shaders for the material object, please refer to section 4.54.4 Shader Properties (page 190) 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.

### 4.6.3 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.

### 4.6.4 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/RIStandard"`.

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.

## 4.7 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. the control point (1,0,0) will be at (16,0,0).

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 28: Level Object Capabilities

### 4.7.1 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 8.10 CSG preview using the AyCSG plugin (page 320).

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:

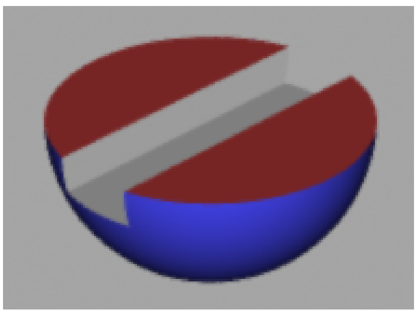


Figure 26: Multicolor CSG Example

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.

### 4.7.2 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 contain 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();
```

## 4.8 Clone Object

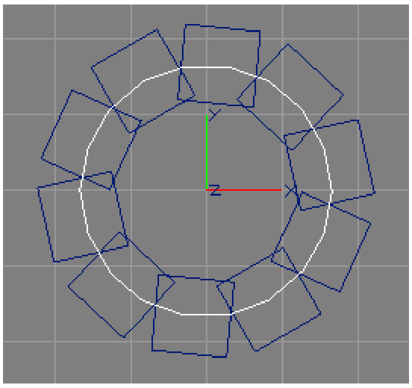


Figure 27: Clone Object (blue) with Trajectory Curve (white)

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 4.38 Sweep Object (page 146)).

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 4.9 Mirror Object (page 95)). 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 29: Clone Object Capabilities

The following attributes further control the cloning process:

### 4.8.1 CloneAttr Property

- "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 30: Clone Parameterisation Examples

[1] Since 1.13.

### 4.8.2 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.

### 4.8.3 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.

## 4.9 Mirror Object

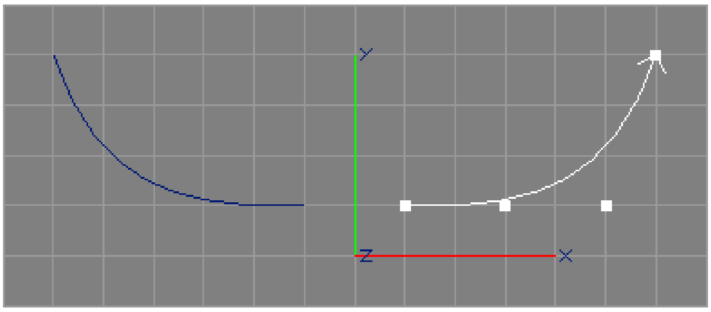



Figure 28: Mirrored Object (blue) From Curve (white)

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
  | NCurve_1(NCurve)
  | NCurve_2(NCurve)
  \ NCurve_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 31: Mirror Object Capabilities

The following attributes further control the mirror process:

### 4.9.1 MirrorAttr Property

- `"Plane"` allows to select the plane about which the mirroring should occur (YZ-, XZ-, or XY-plane).

[1] Since 1.20.

### 4.9.2 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).

### 4.9.3 RIB Export

Mirror objects appear in RIB output as a number of real objects, each with different transformation attributes.

## 4.10 Instance Object

Instance objects help to save memory consumption in scenes with many similar or repeating objects or transport geometric data across the scene hierarchy.

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. 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.

Normally, the sole purpose of instance objects is to save space, in memory and on the disk. But 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 306)). Note that in the tool object context, instance objects are the only objects, that are subject to a second round of provision.

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, one can hardly utilise instancing. Similar means "the same except for the transformation property" in this context.

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 one may not put instances of a level 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 8.8 Automatic Instancing (page 319)).

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 32: Instance Object Capabilities

#### 4.10.1 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.

#### 4.10.2 Instances and the Object Clipboard

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. All copies of instance objects always point to the same original master object.

For example, when the following two objects are copied and pasted back to the scene

```
--NCurve   <-----
                |
--Instance  -----
```

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.

It is possible to move masters and instances through the scene hierarchy using drag and drop in the tree view or using the clipboard with "Edit/Cut" and then "Special/Clipboard/Paste (Move)".

[1] Since 1.16.

### 4.10.3 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, one can also use the main menu entry: `"Special/Instances/Resolve all Instances"`.

### 4.10.4 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. You can change this behaviour using the RIB export preference setting `"ResInstances"`: If `"ResInstances"` 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.

## 4.11 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 33: Select Object Capabilities

### 4.11.1 SelectAttrib Property

- `"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 one can mix provided objects from multiple child objects of the Select object. 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.

### 4.11.2 RIB Export

Select objects never appear in RIB output.

[1] Since 1.14. [2] Since 1.16.

## 4.12 RiInc Object

RiInc objects may be used to include objects or whole scene parts into your scenes that, for some reason, are just available as a piece of RIB.

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

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

Table 34: RiInc Object Capabilities

The following attributes control the inclusion process:

### 4.12.1 RiIncAttr Property

- Using `"File"` you specify the filename of the RIB to be included.
- `"Width"`, `"Height"`, and `"Length"` specify the size of a box, that will be drawn as a geometric representation of the RIB.

## 4.13 RiProc Object

RiProc objects may be used to include procedural objects or external archives into your scenes.

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 35: RiProc Object Capabilities

The following attributes control the RiProc object:

### 4.13.1 RiProcAttr Property

- `"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.

## 4.14 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 36: Box Object Capabilities

The following parameters further control the shape of a box:

### 4.14.1 BoxAttrib Property

- "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).

### 4.14.2 Conversion Support

A box object may be converted to three NURBS patches using the main menu entry "Tools/Convert".[1]

### 4.14.3 RIB Export

The box object will always be exported as six bilinear patches.

[1] Since 1.8.2.

## 4.15 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 37: Sphere Object Capabilities

The following parameters further control the shape of a sphere:

### 4.15.1 SphereAttr Property

- `"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.

### 4.15.2 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.

### 4.15.3 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.

## 4.16 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 38: Disk Object Capabilities

The following parameters further control the shape of a disk:

### 4.16.1 DiskAttr Property

- `"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.

### 4.16.2 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]

### 4.16.3 RIB Export

The disk object will always be exported as simple disk:

```
RiDisk(...);
```

[1] Since 1.8.2.

## 4.17 Cone Object

A cone, centered at the origin of the object coordinate system, with the base in the XY plane.

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 39: Cone Object Capabilities

The following parameters further control the shape of a cone:

### 4.17.1 ConeAttr Property

- `"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.

### 4.17.2 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.

### 4.17.3 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.

## 4.18 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 40: Cylinder Object Capabilities

The following parameters further control the shape of a cylinder:

### 4.18.1 CylinderAttr Property

- `"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.

### 4.18.2 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.

### 4.18.3 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.

## 4.19 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 41: Torus Object Capabilities

The following parameters further control the shape of a torus:

### 4.19.1 TorusAttr Property

- `"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.

### 4.19.2 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.

### 4.19.3 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.

## 4.20 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 42: Paraboloid Object Capabilities

The following parameters further control the shape of a paraboloid:

### 4.20.1 ParaboloidAttr Property

- `"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.

### 4.20.2 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.

### 4.20.3 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.

## 4.21 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 43: Hyperboloid Object Capabilities

The following parameters further control the shape of a hyperboloid:

### 4.21.1 HyperboloidAttr Property

- `"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.

### 4.21.2 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.

### 4.21.3 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.

## 4.22 NCurve (NURBS Curve) Object

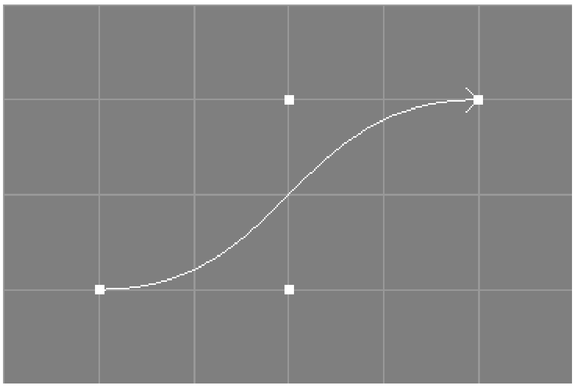


Figure 29: A simple NURBS Curve

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 4.22.2 Multiple Points (page 113).

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 44: NCurve Object Capabilities

The next section details the NCurve object property.

### 4.22.1 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 atleast 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 tesselation 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.

- `"Knots"` allows to enter own custom knot sequences. Note that `"Knots"` are not in use if `"Knot-Type"` (above) is of type `"NURB"`, `"B-Spline"`, `"Bezier"`, `"Chordal"`, or

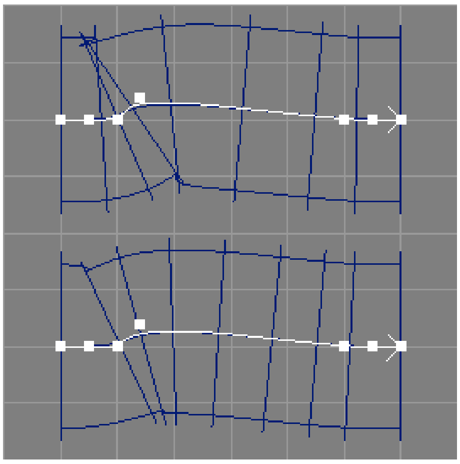


Figure 30: Sweeps Created From Curves With Uniform (upper) And Chordal (lower) Knot Vectors

"Centripetal". Note also, that the content of this parameter will be overwritten, if the knot type is switched to any of the aforementioned types.

- "CreateMP" toggles, whether multiple points should be created for this curve. See also the discussion in section 4.22.2 Multiple Points (page 113).
- "IsRat" informs you, whether the curve is rational (uses 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, it's 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 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.

[1] Since 1.9.

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]

### 4.22.2 Multiple Points

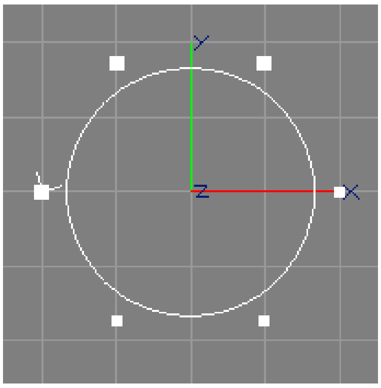



Figure 31: A NURBS Curve with Multiple Points (big handles)

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. Modelling actions will always modify all the control points that make up a multiple point. Multiple points will be drawn with a bigger handle than normal points (see image above). They may e.g. be used to create closed curves. 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 be created using the collapse tool, and split up again using the explode tool (see sections 5.32 Collapse Tool (page 214) and 5.33 Explode Tool (page 214) for more information regarding those tools). Note that even though you might have exploded some multiple points Ayam will re-create them automatically on several occasions like reading of a scene, inserting/deleting points, and applying the NCurveAttr property if they still have identical coordinate values. In other words, you should immediately edit the control points (move them apart) after exploding to avoid that they automatically collapse to a multiple point again! You may also totally inhibit creation of multiple points for a NURBS curve using the attribute `"CreateMP"`.

### 4.22.3 RIB Export

NCurve objects never directly appear in RIB output (only indirectly as trim curve).

[1] Since 1.18.

## 4.23 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 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.

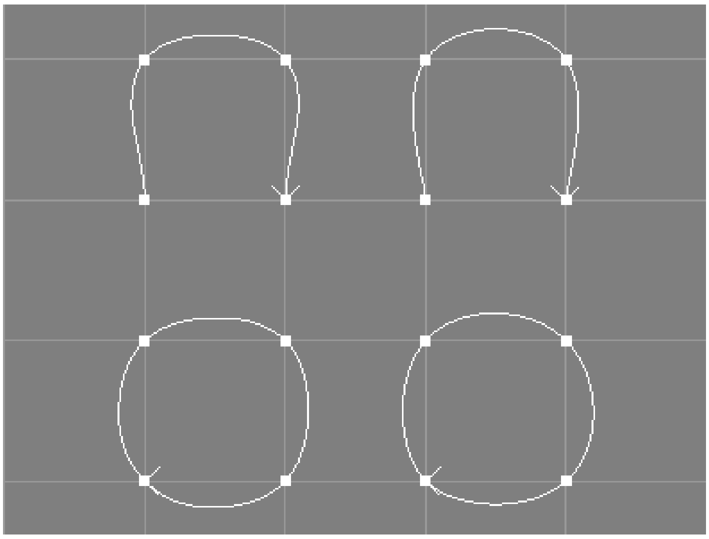


Figure 32: Different ICurves resulting from identical data point configurations (Upper Left: Open, Order 3; Upper Right: Open, Order 4; Lower Left: Closed, Order 3; Lower Right: Closed, Order 4)

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 45: ICurve Object Capabilities

The following parameters control the interpolation process:

### 4.23.1 ICurveAttr Property

- 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 dervivatives are is switched on, two more editable points appear in the single point editing modes. Those additional points directly control the derivatives in 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 4.22.1 NCurveAttr (page 110) 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.

### 4.23.2 Conversion Support

The interpolating curve may be converted to an ordinary NURBS curve using the main menu entry "Tools/Convert".

### 4.23.3 RIB Export

ICurve objects never directly appear in RIB output (only indirectly as trim curve).

### 4.24 ACurve (Approximating Curve) Object

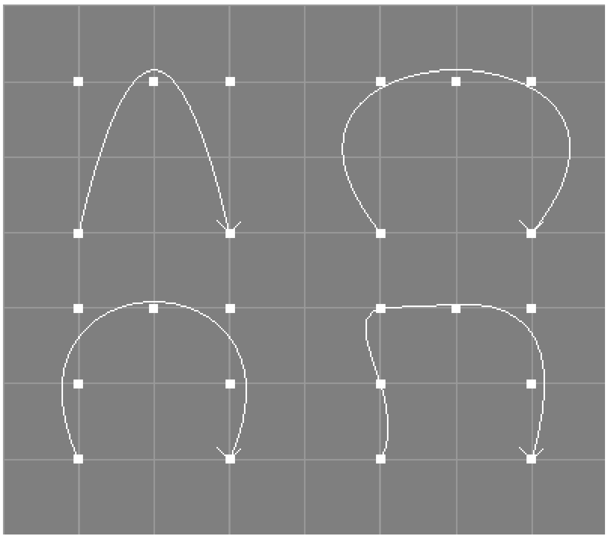


Figure 33: Different ACurves (Upper Left: ALength 3, Order 3; Upper Right: ALength 4, Order 4; Lower Left: ALength 4, Order 4; Lower Right: ALength 6, Order 3)

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 data points must be higher than or equal to the number of control points used. 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 46: ACurve Object Capabilities

The following parameters control the approximation process:

#### 4.24.1 ACurveAttr Property

- Using `"Length"` you determine the number of data points to approximate.
- Using `"ALength"` you determine 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.

[1] Since 1.15.

- 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.

| 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 47: 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.

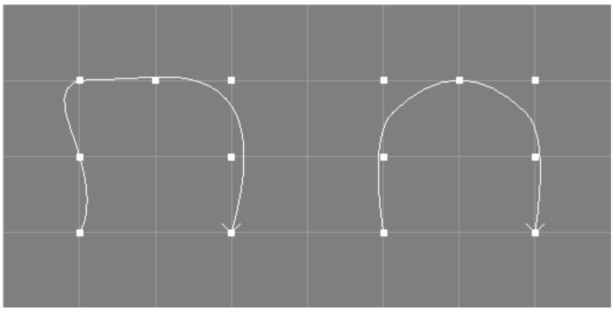


Figure 34: Left: asymmetric ACurve, Right: symmetric ACurve

- The parameter "Order" specifies the desired order of the approximating NURBS curve. Currently, only orders higher than 2 are supported.
- See section 4.22.1 NCurveAttr (page 110) for a description of the parameters: "Tolerance" and "DisplayMode".

- Finally, a "NCInfo" field informs about the actual configuration of the created NURBS curve.

### 4.24.2 Conversion Support

The approximating curve may be converted to an ordinary NURBS curve using the main menu entry "Tools/Convert".

### 4.24.3 RIB Export

ACurve objects never directly appear in RIB output (only indirectly as trim curve).

## 4.25 NCircle (NURBS Circle) Object

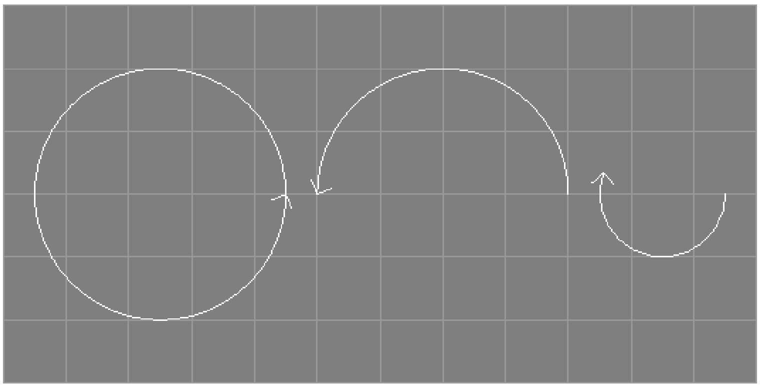


Figure 35: Different NCircle Objects (l: default, m: TMax 180, r: Radius 0.5, TMax 200)

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]

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 48: NCircle Object Capabilities

The following section describes the parameters of the circle or arc.

### 4.25.1 NCircleAttr Property

- "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 4.22.1 NCurveAttr (page 110) for a description of the parameters: "Tolerance" and "DisplayMode".

- Finally, a "NCInfo" field informs about the actual configuration of the created NURBS curve.

### 4.25.2 Conversion Support

The circular curve/arc may be converted to an ordinary NURBS curve using the main menu entry "Tools/Convert".

[1] Since 1.12. [2] Since 1.15.

### 4.25.3 RIB Export

NCircle objects never directly appear in RIB output (only indirectly as trim curve).

## 4.26 ConcatNC (Concatenate NURBS Curves) Object

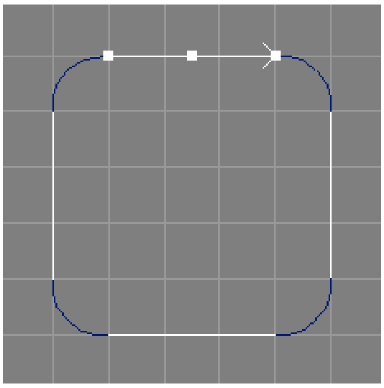


Figure 36: ConcatNC Object (blue) from a Linear Curve and 3 Instances (white)

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 5.19 elevate tool (page 208) for more information on elevation) and all curves will be clamped (see also section 5.23 clamp tool (page 210) 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, atleast 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 "`NURBS`" or a custom knot vector will be computed (depending on the setting of "`Knot-Type`"). If "`Knot-Type`" is "`NURBS`", 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.

Attributes like display mode and tolerance for the new concatenated curve are simply taken from the first parameter curve.

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 49: ConcatNC Object Capabilities

The following parameters further control the concatenation process:

### 4.26.1 ConcatNCAttr Property

- 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 atleast 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.
- `"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. 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. This also applies to potential discontinuities in transitions.

    However, this comes at the price of internal knots with high multiplicity. Note that such a curve is *not* differentiable in all places anymore.

- Finally, a `"NCInfo"` field informs about the actual configuration of the created NURBS curve.

### 4.26.2 Conversion Support

The concatenated curve may be converted to an ordinary NURBS curve using the main menu entry `"Tools/Convert"`.

### 4.26.3 RIB Export

ConcatNC objects never directly appear in RIB output (only indirectly as trim curve).

## 4.27 ExtrNC (Extract NURBS Curve) Object

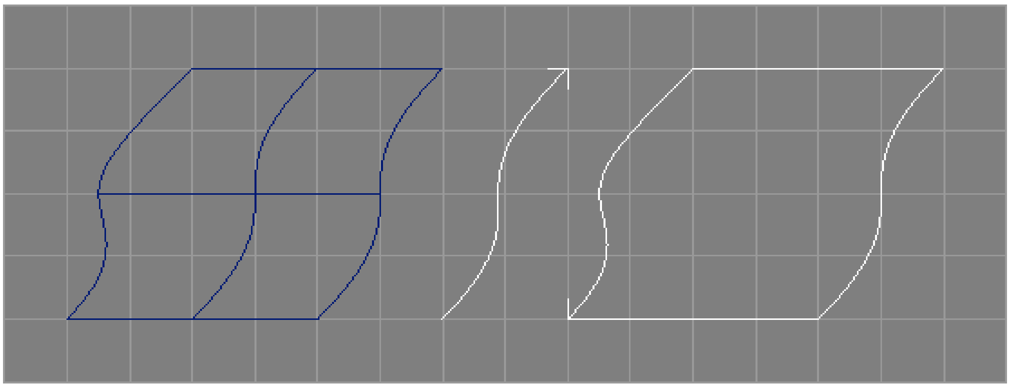


Figure 37: Extracted Curves (white) from Arbitrarily Shaped Surface (blue)

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.

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 50: ExtrNC Object Capabilities

The extraction process is controlled by the following attributes:

### 4.27.1 ExtrNCAttr Property

- `"Side"` controls which curve should be extracted from the surface. Available values are:

[1] Since 1.19.

  - "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.[2]
  - "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 useful to re-engineer swept surfaces, delivered as simple patches.

  Note that if "Side" is "U0", "Un", "V0", or "Vn" the extraction process just copies the respective boundary control points, which only works for clamped knot vectors. To extract a boundary from a surface with e.g. a B-Spline knot vector, one should *not* use "U0" but "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. This parameter is only used when "Side" is "U" or "V". The valid range of parameter values depends on the knot vectors of the NURBS patch.
- "Relative" controls whether the parametric value should be interpreted in a relative way. 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 "Parameter" is then consequently 0.0-1.0.[4]
- "Revert" immediately reverts the extracted curve.
- "CreatePVN" controls creation of a PV tag that contains the normals (derived from the surface control points) on the surface. This PV tag can then be used to control a 3D offset curve.
- "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 4.22.1 NCurveAttr (page 110) for a description of the other two attributes "DisplayMode" and "Tolerance".

### 4.27.2 Conversion Support

The extracted curve may be converted to an ordinary NURBS curve using the main menu entry "Tools/Convert".

### 4.27.3 RIB Export

ExtrNC objects never directly appear in RIB output (only indirectly as trim curve).

[1] Since 1.8.1. [2] Since 1.13. [3] Since 1.15. [4] Since 1.15.

## 4.28 OffsetNC (Offset NURBS Curves) Object

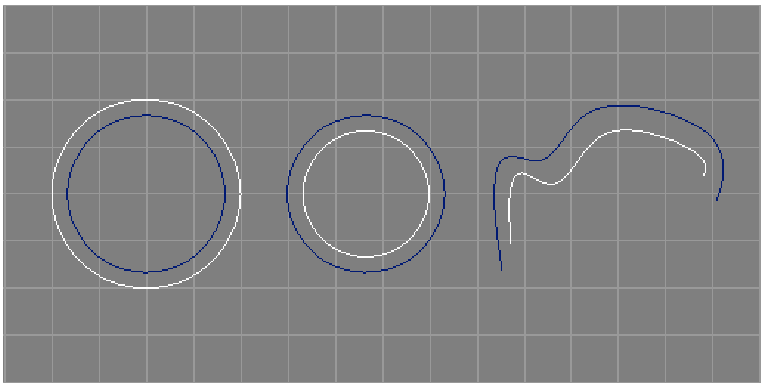


Figure 38: Offset Curves (white) from NURBS Curves (blue) with Offset 0.2, -0.2, and 0.3

The OffsetNC object creates offset curves from planar NURBS curves using four different algorithms.[1] The offset curve will always match the original curve in type, length, order, and knots. See also the image above.

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]

**`3DPVN`**
Since Ayam 1.18 there is a fourth offsetting algorithm available, that 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 `"ExtrNC"` object.

As the geometry of the offset curve is completely defined by the master curve, OffsetNC objects do not support own transformation attributes.[3]

The `"Bevel3D"` offset algorithm has been removed since Ayam 1.19.

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

The following parameters further control the offsetting process:

[1] Since 1.14. [2] Since 1.19. [3] Since 1.19.

| Type | Parent of | Material | Converts to / Provides | Point Edit |
|---|---|---|---|---|
| OffsetNC | NCurve | No | NCurve | No* |

Table 51: OffsetNC Object Capabilities

### 4.28.1 OffsetNCAttr Property

- The first parameter "Mode" determines, which algorithm to use for the offsetting process.
- Using "Revert" you may revert the direction of the offset curve.
- "Offset" determines the distance between original curve and offset curve. Negative values are allowed.
- See section 4.22.1 NCurveAttr (page 110) for a description of the parameters: "Tolerance" and "DisplayMode".
- Finally, a "NCInfo" field informs about the actual configuration of the created NURBS curve.

### 4.28.2 Conversion Support

The offset curve may be converted to an ordinary NURBS curve using the main menu entry "Tools/Convert".

### 4.28.3 RIB Export

OffsetNC objects never directly appear in RIB output (only indirectly as trim curve).

## 4.29 NPatch (NURBS Patch) Object

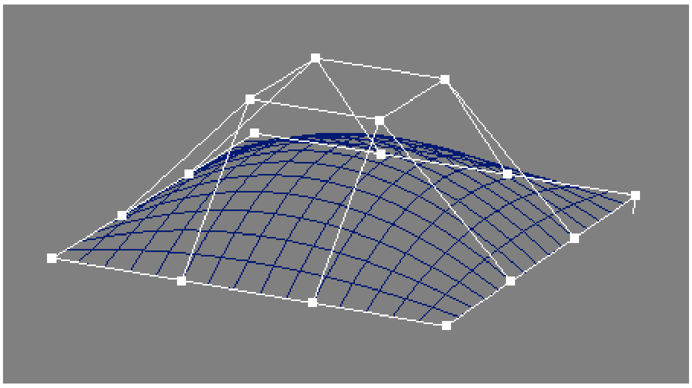


Figure 39: NURBS patch: surface (blue) from control polygon (white)

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 high enough 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 4.22.2 Multiple Points (page 113) 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 52: NPatch Object Capabilities

### 4.29.1 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.
- `"Knot-Type_U"`/`"Knot-Type_V"` and `"Knots_U"`/`"Knots_V"`: For a discussion of the `"Knot-Type"` and `"Knots"` parameters, please see section 4.22.1 NCurveAttr (page 110).
- `"CreateMP"` toggles, whether multiple points should be created for this surface. See also the discussion in section 4.22.2 Multiple Points (page 113).[2]
- `"IsRat"` informs you, whether the patch is rational (uses weight values different from 1.0).[3]

[1] Since 1.10. [2] Since 1.10. [3] Since 1.9.

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 tesselation (`"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.

### 4.29.2 Trim Curves

Trim curves may be used to cut out certain parts of a NURBS patch. 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 should 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. You can use the `Revert` tool (`Tools/NURBCurve` menu) 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.
- 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. You can find this tool 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 themself.

Note that Ayam is not checking whether your 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 wireframe 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]

### 4.29.3 Conversion Support

A NPatch object may be converted to a PolyMesh object using the main menu entry `"Tools/Convert"`.

This process is also called tesselation and thus, the tesselation parameters from TP tags will be used in the conversion process (if present) (see also section 4.55.10 TP Tag (page 198)).

### 4.29.4 RIB Export

NPatch objects will be exported as NURBS patch primitives:

```
RiNuPatch(...);
```

PV tags are supported and also trim curves may appear.

[1] Since 1.5.

## 4.30 IPatch (Interpolating Patch) Object

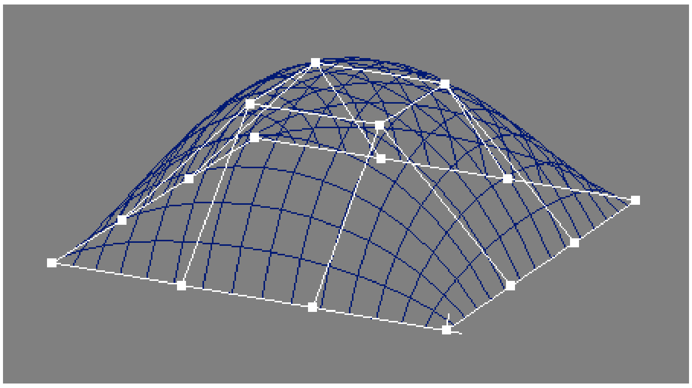


Figure 40: Interpolating surface (blue) from data points (white)

A 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 53: IPatch Object Capabilities

### 4.30.1 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 tesselation ("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.

### 4.30.2 Conversion Support

The interpolated surface may be converted to an ordinary NURBS patch using the main menu entry "Tools/Convert".

### 4.30.3 RIB Export

IPatch objects will be exported as NURBS patch primitives:

```
RiNuPatch(...);
```

PV tags are currently not supported.

## 4.31 BPatch (Bilinear Patch) Object

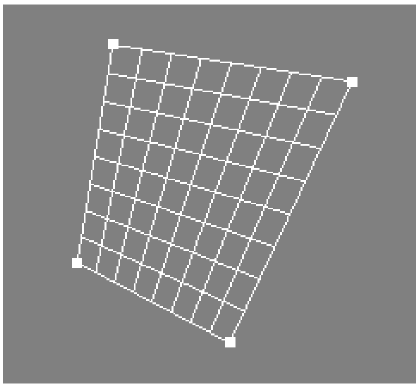


Figure 41: Bilinear Patch

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 4.14 Box Object (page 102).

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 54: BPatch Object Capabilities

### 4.31.1 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.

### 4.31.2 Conversion Support

The bilinear patch may be converted to an ordinary NURBS patch using the main menu entry `"Tools/Convert"`.

### 4.31.3 RIB Export

BPatch objects will be exported as bilinear patch primitives:

```
RiPatch(RI_BILINEAR, ...);
```

PV tags are supported.

## 4.32 PatchMesh Object

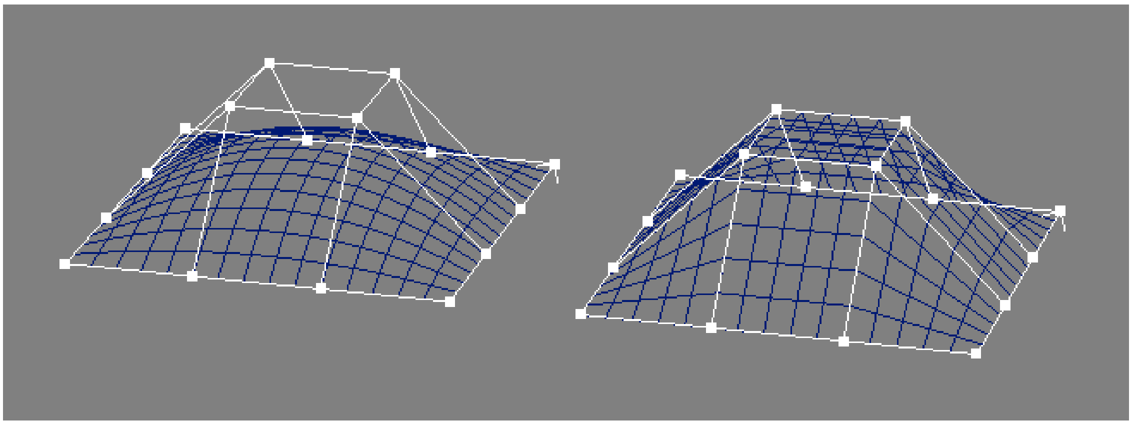


Figure 42: Bicubic (left) and Bilinear (right) PatchMesh Surfaces (blue) from their Respective Control Polygons (white)

The PatchMesh object may be used to model with bilinear and bicubic patch meshes.

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 55: PatchMesh Object Capabilities

### 4.32.1 PatchMeshAttr Property

The first section of the PatchMeshAttr property contains patch specific settings:

- `"Type"` may be set to `"Bilinear"` or `"Bicubic"`.
- `"Width"` and `"Height"` control the dimensions of the patch.
- `"Close_U"` and `"Close_V"` determine, whether the patch mesh should be closed in U and V direction respectively.
- `"BType_U"` and `"BType_V"` control the basis type for bicubic patches. You may choose between the basis types: `"Bezier"`, `"B-Spline"`, `"Catmull-Rom"`, `"Hermite"`, and `"Custom"`. In the latter case (`"Custom"`), additional parameters may be set. Those are `"Step_U"`/`"Step_V"` (the stepsize of the basis) and `"Basis_U"`/`"Basis_V"` the basis itself (please see the RenderMan Companion for a discussion of basis types).

The parameters `"BType_U"` and `"BType_V"` and consequently `"Step_U"`/`"Step_V"` and `"Basis_U"`/`"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 tesselation (OutlinePolygon), or just the outlines of the patch (OutlinePatch). The default setting (Global) means, that the global preference setting "Drawing/DisplayMode" should be used.

### 4.32.2 Conversion Support

The patch mesh may be converted to an ordinary NURBS patch using the main menu entry "Tools/Convert". However, this does not work for all possible types of patch meshes (e.g. patch meshes with the basis types Catmull-Rom, Hermite, or Custom can not be converted, currently). An internal NURBS patch representation is also in use when drawing the patch mesh (if the "DisplayMode" is not "ControlHull") and when shading the patch mesh. Consequently, there is currently no shaded representation of patch meshes of basis type Catmull-Rom, Hermite or Custom.

### 4.32.3 RIB Export

PatchMesh objects will be exported as patch mesh primitives:

```
RiPatchMesh(...);
```

PV tags are supported.

### 4.33 PolyMesh Object

The PolyMesh object may be used to include objects that have been modeled using the polygonal modelling paradigm in Ayam scenes.

There are no special modelling actions for this type of object, but you may select and modify single points as you can do it 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. The loops use a point indexing scheme to efficiently reuse coordinate values. This general approach requires a so called tesselation to be carried out, in order for the PolyMesh object to be shaded. For the tesselation, Ayam uses routines of the GLU library.

Ayam is able to automatically create face normals for PolyMeshes. They will be calculated while tesselating 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 a bunch of triangles each in its own PolyMesh object will lead to a real waste of memory. You may use the merge tool (main menu `"Tools/PolyMesh/Merge"`) to merge 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 56: PolyMesh Object Capabilities

#### 4.33.1 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.

#### 4.33.2 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.

[1] Since 1.11.

### 4.33.3 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.

## 4.34 SDMesh Object

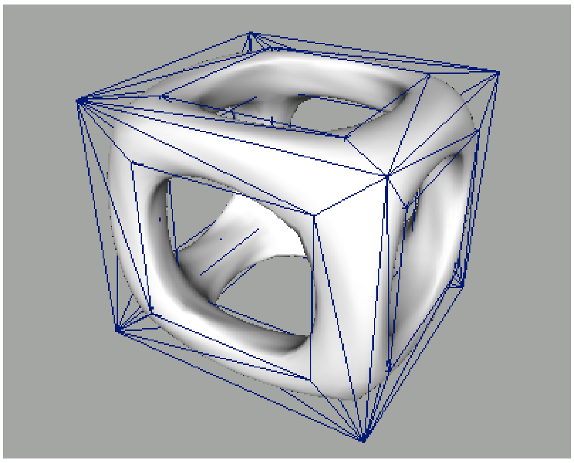


Figure 43: SDMesh object with original polygonal mesh (blue)

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 you may select and modify single points as you can do it 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 wireframe 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 57: SDMesh Object Capabilities

### 4.34.1 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]
- "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.

### 4.34.2 Conversion Support

SDMesh objects may be converted to PolyMesh objects.[2]

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.

### 4.34.3 RIB Export

SDMesh objects will be exported as subdivision mesh primitives:

```
RiSubdivisionMesh(...);
```

PV tags are supported.

[1] Since 1.11. [2] Since 1.11.

## 4.35 Revolve Object

Figure 44: Revolve Object (left: Curve, right: Surface of Revolution)

The Revolve object forms a surface of revolution from a NURBS curve.

The Revolve object has the generating NURBS curve as child object and watches its changes and adapts to it automagically.

The axis of revolution is always the Y axis. The parameter curve should be defined in the XY plane. If not, it will be squashed down to this plane before revolving it.

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 58: Revolve Object Capabilities

### 4.35.1 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.

The revolve object can automatically generate caps, which are trimmed NURBS patches. Using the parameters `"UpperCap"`, `"LowerCap"`, `"StartCap"`, and `"EndCap"`, you determine whether such caps

[1] Since 1.8. [2] Since 1.18.

should be generated, default is off (no caps).

If the side caps of a surface of revolution of an open curve are not created correctly, GLU complains about "intersecting or misoriented trim curves", try to revert the revolved curve.

See section 4.29.1 NPatchAttr (page 127) for a description of the other two attributes `"DisplayMode"` and `"Tolerance"`.

### 4.35.2 Conversion Support

The surface of revolution and the caps may be converted to ordinary NURBS patches using the main menu entry `"Tools/Convert"`.

If caps are present, an enclosing Level object will be created and the caps follow the surface of revolution in the following order: upper, lower, start, end.

The Revolve object provides a list of NURBS patch objects in the same order as created upon conversion.

### 4.35.3 RIB Export

Revolve objects will be exported as NURBS patch primitives:

```
RiNuPatch(...);
```

If caps are present, those follow as trimmed NURBS patch primitives in the following order: upper, lower, start, end.

PV tags are supported but all NURBS patch primitives will get the same set of tags.[1]

[1] Since 1.20.

## 4.36 Extrude Object

Figure 45: Extrude Object (left: Curve, middle: normal Extrusion, right: Extrusion with Caps)

The extrude object forms an extrusion from a number of planar NURBS curves.

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 automagically.

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 5.28 To XY Tool (page 213) 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 59: Extrude Object Capabilities

### 4.36.1 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 4.43.1 BevelAttr Property (page 163).

See section 4.29.1 NPatchAttr (page 127) for a description of the other two attributes "DisplayMode" and "Tolerance".

### 4.36.2 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 themself and one can not have hole curves inside hole curves. If there are bevels and caps, allow 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 one decides the direction of the bevel 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. One should 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.

### 4.36.3 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.

### 4.36.4 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.

## 4.37 Swing Object

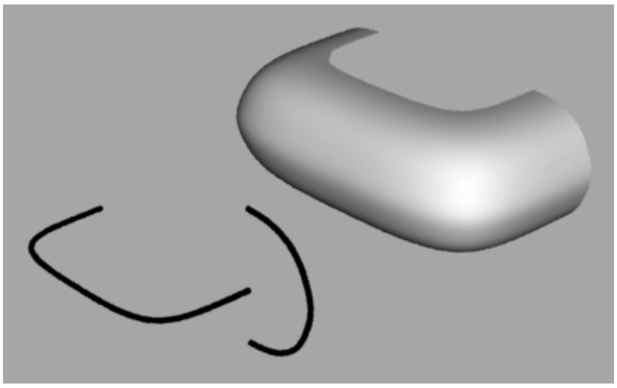


Figure 46: Swing Object (left: Curves, right: Resulting Swung Surface)

The Swing object forms a surface that results from rotating a NURBS curve (cross section or profile) around an axis while scaling it according to a second NURBS curve (trajectory or path).[1]
This process is sometimes also called *rotational sweep*.

The swing object has the generating NURBS curves as child objects and watches their movements and adapts to them automagically. 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.

The base plane for the swing operation is the YZ-plane. The cross section curve should be defined in this plane and the trajectory should start here.

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 60: Swing Object Capabilities

### 4.37.1 SwingAttr Property

The attributes `"UpperCap"`, `"LowerCap"`, `"StartCap"`, and `"EndCap"` may be used to automatically create cap surfaces, that close the Swing on the respective ends. Note that this works properly only if the corresponding parameter curve is closed and planar.

[1] Since 1.14.

See section 4.29.1 NPatchAttr (page 127) for a description of the other two attributes `"DisplayMode"` and `"Tolerance"`.

To help in the exact configuration of the Swing, the `"NPInfo"` field always displays the parameters of the created NURBS patch.

### 4.37.2 Conversion Support

The swung surface and the caps may be converted to ordinary NURBS patches using the main menu entry `"Tools/Convert"`.

If caps are present, an enclosing Level object will be created and the caps follow the swung surface in the following order: upper, lower, start, end.

The Swing object provides a list of NURBS patch objects in the same order as created upon conversion.

### 4.37.3 RIB Export

Swing objects will be exported as NURBS patch primitives:

```
RiNuPatch(...);
```

If caps are present, those follow as trimmed NURBS patch primitives in the following order: upper, lower, start, end.

PV tags are supported but all NURBS patch primitives will get the same set of tags.[1]

[1] Since 1.20.

## 4.38 Sweep Object

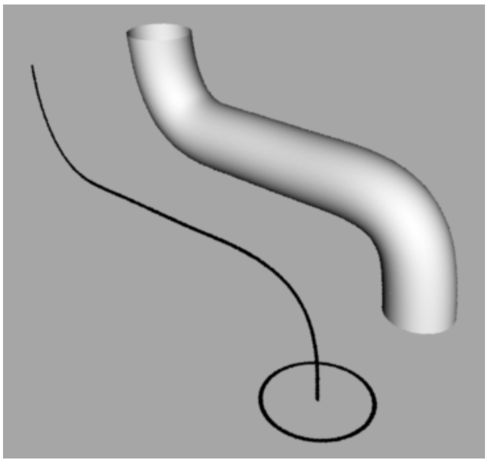


Figure 47: Sweep Object (left: Curves, right: Resulting Swept Surface)

The Sweep object forms a surface that results from moving a NURBS curve (cross section or profile) along a second NURBS curve (trajectory or path). 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, since Ayam 1.10, they may even be periodic.

The Sweep object has the generating NURBS curves as child objects and watches their movements and adapts to them automagically. 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 but it also has to be defined in the XY-plane of its own coordinate system. 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 using its transformation attributes to follow these rules. Later editing of this curve has to be done in a Side view (or in an aligned local Front view, if the Sweep object itself is transformed somehow).

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.

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]

[1] Since 1.13.

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.

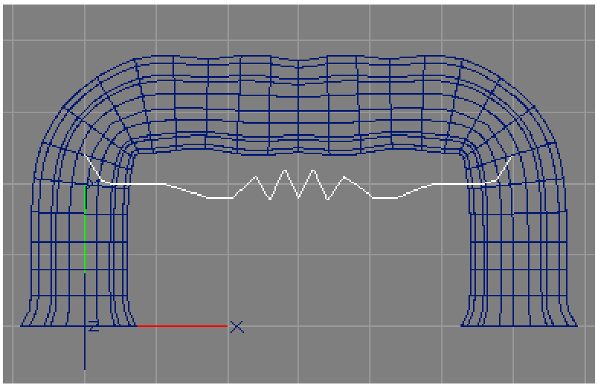


Figure 48: Sweep Object (blue) with Scaling Function (white)

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. Rotate the curve by 90 degrees around the Y-axis. (Use the "Transformations" property for that.)
3. Create a simple NURBS curve using the toolbox. (This will be our trajectory.)
4. Select both curves. (Select the first curve, hold down the "Shift" key and select the other curve.)
5. Create the Sweep object using the toolbox.
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 270) has an example script that automates creation and parameterisation of a suitable cross section curve.

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 61: Sweep Object Capabilities

### 4.38.1 SweepAttr Property

The "Type" attribute controls whether the swept surface should be open, closed, or periodic in the direction of the trajectory curve.[1]

[1] Since 1.10.

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.

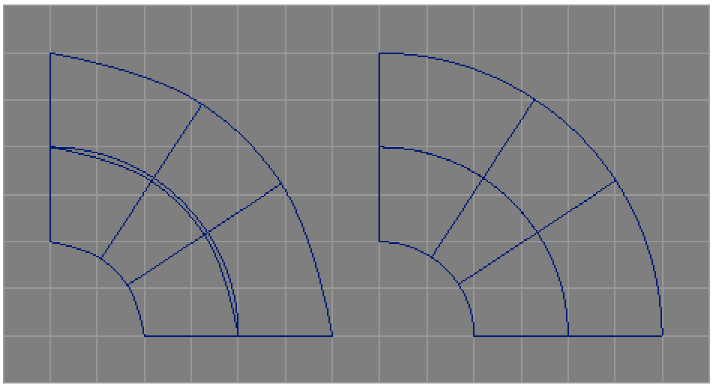


Figure 49: Sweep Along a Quarter Circle without (left) and with (right) Interpolation Enabled

The third parameter, "Sections", determines how many sections (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 sweep types whereas sections+order control points will be created for periodic sweep types.
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 trajectory curve plus one (except for trajectory curves of length 2, where it is 1). See the table below for examples.
Consequently, 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.
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 enabled by default.

The attributes "StartCap" and "EndCap" may be used to automatically create cap surfaces, that close the Sweep on both ends. Note that this works properly only if the cross section curve is closed and planar (defined in the XY plane).

Also bevels are available for sweeps.[2] They are accessible through the 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 4.43.1 BevelAttr Property (page 163).

See section 4.29.1 NPatchAttr (page 127) for a description of the other two attributes "DisplayMode" and "Tolerance".

[1] Since 1.13. [2] Since 1.10.

To help in the exact configuration of the sweep, 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 62: Sweep Parameterisation Examples

### 4.38.2 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.

The Sweep object provides a list of NURBS patch objects in the same order as created upon conversion.

### 4.38.3 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.

PV tags are supported but all NURBS patch primitives will get the same set of tags.[1]

[1] Since 1.20.

## 4.39 Birail1 Object

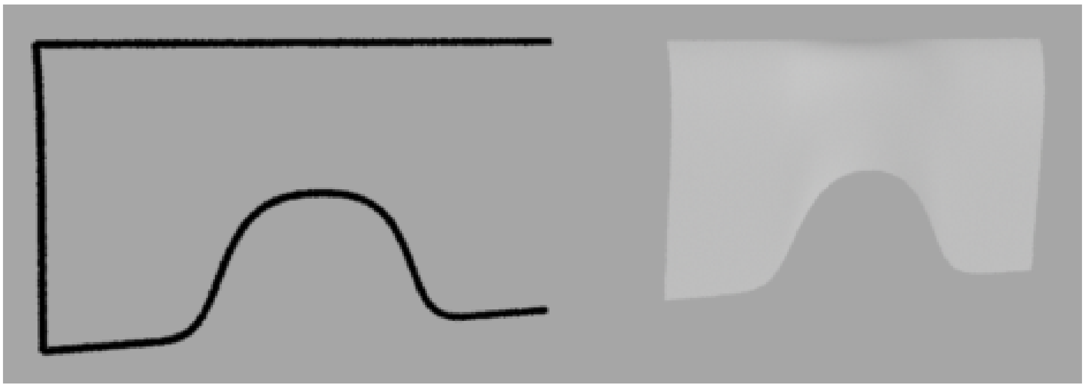


Figure 50: Birail1 Object (left: Curves, right: Resulting Swept Surface)

The Birail1 object forms a surface by sweeping a cross section (or profile) curve along two so called rail curves. 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.

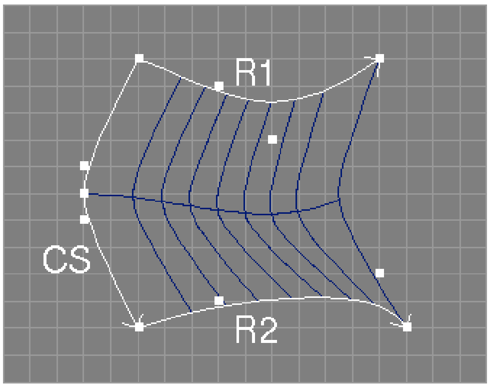



Figure 51: Valid Configuration of Parameter Curves (white) for Birail1 (blue)

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.

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 63: Birail1 Object Capabilities

The following parameters further control the birailing process:

### 4.39.1 Birail1Attr Property

Similar to the "Sweep", the "Type" attribute controls whether the swept surface should be open, closed, or periodic in the direction of the birail 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.

The attributes "StartCap" and "EndCap" may be used to automatically create cap surfaces, that close the birailed surface on the respective end. Note that this only works properly if the cross section curve is closed and planar (e.g. defined in the XY plane).

Since Ayam 1.10 bevels are available for birails. They 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 4.43.1 BevelAttr Property (page 163).

See section 4.29.1 NPatchAttr (page 127) for a description of the other two attributes "DisplayMode" and "Tolerance".

To help in the exact configuration of the birail, 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 birail 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 64: Birail1 Parameterisation Examples

### 4.39.2 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.

The Birail1 object provides a list of NURBS patch objects in the same order as created upon conversion.

### 4.39.3 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.

PV tags are supported but all NURBS patch primitives will get the same set of tags.[1]

[1] Since 1.20.

## 4.40 Birail2 Object

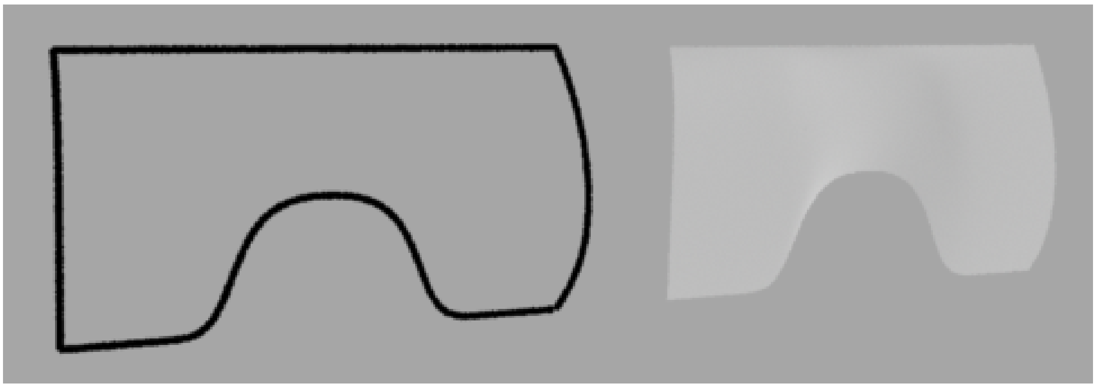


Figure 52: Birail2 Object (left: Curves, right: Resulting Swept Surface)

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. The morphing process may be controlled by a fifth parameter 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 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.

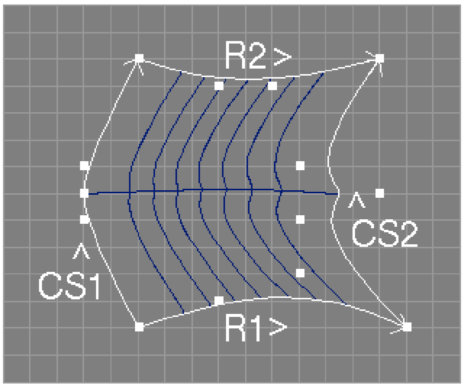



Figure 53: Valid Configuration of Parameter Curves (white) for Birail2 (blue)

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 65: Birail2 Object Capabilities

The following parameters control the birailing process:

### 4.40.1 Birail2Attr Property

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.

The attributes `"StartCap"` and `"EndCap"` may be used to automatically create cap surfaces, that close the birailed surface on the respective end. Note that this only works properly if the cross section curve is closed and planar (e.g. defined in the XY plane).

[1] Since 1.10. [2] Since 1.13.

Since Ayam 1.10 bevels are available for birails. They 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 4.43.1 BevelAttr Property (page 163).

See section 4.29.1 NPatchAttr (page 127) for a description of the other two attributes "DisplayMode" and "Tolerance".

To help in the exact configuration of the birail, the "NPInfo" field always displays the parameters of the created NURBS patch.

The following table shows some example parameter configurations for the birail 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 66: Birail2 Parameterisation Examples

### 4.40.2 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.

The Birail2 object provides a list of NURBS patch objects in the same order as created upon conversion.

### 4.40.3 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.

PV tags are supported but all NURBS patch primitives will get the same set of tags.[1]

[1] Since 1.20.

## 4.41 Skin Object

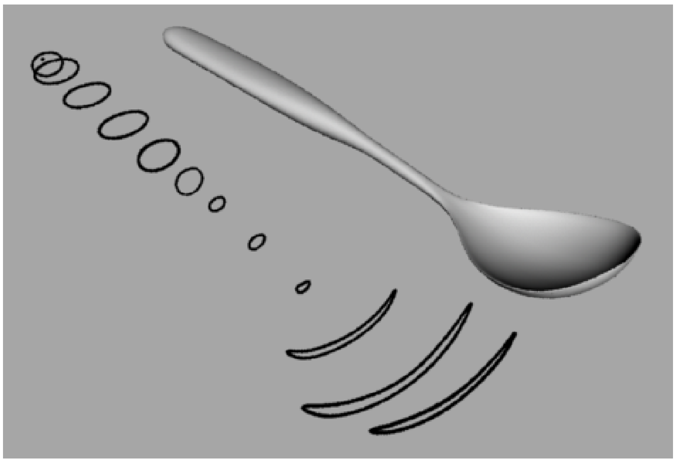


Figure 54: Skin Object (left: Curves, right: Resulting Skinned Surface)

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*). 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
 | C_1(NCurve)
 | C_2(NCurve)
 | [...
 \ C_n(NCurve)]
```

Note that in contrast to the build from curves tool, the curves may be of arbitrary length and order. One may e.g. easily 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. Since Ayam 1.9, 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.

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.

The skin object has the generating NURBS curves as child objects and watches their changes and adapts to them automagically.

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 67: Skin Object Capabilities

The following parameters control the skinning process:

### 4.41.1 SkinAttr Property

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 of 4 will be used.

Using the next parameter "`Knot-Type_U`", you can adapt the type of the knot vector that should be used in the U direction of the skinned surface. Note that this setting will have no effect if interpolation is switched on because then a chord length parameterisation will be used. 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. New in Ayam 1.7 is support for the knot type Custom, which creates a chord length parameterisation, even if interpolation is not enabled.

The attributes "`StartCap`" and "`EndCap`" may be used to automatically create cap surfaces to close the skinned surface on both ends. Note that this works only if the respective curves are closed and planar (e.g. defined in the XY plane). Furthermore, if the skin is not interpolating the respective parameter curves (this may be the case if the "`Knot-Type_U`" parameter is set to "`B-Spline`") the cap(s) will not be created in the right place. The cap(s) will always be created in the position of the first (last) parameter curve.

Since Ayam 1.10 bevels are available for skins. They 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 4.43.1 BevelAttr Property (page 163).

See section 4.29.1 NPatchAttr (page 127) for a description of the other two attributes "`DisplayMode`" and "`Tolerance`".

To help in the exact configuration of the skin, the "`NPInfo`" field always displays the parameters of the created NURBS patch.

### 4.41.2 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: end bevel, end cap, start bevel, start cap.

The Skin object provides a list of NURBS patch objects in the same order as created upon conversion.

### 4.41.3 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: 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.

## 4.42 Gordon Object

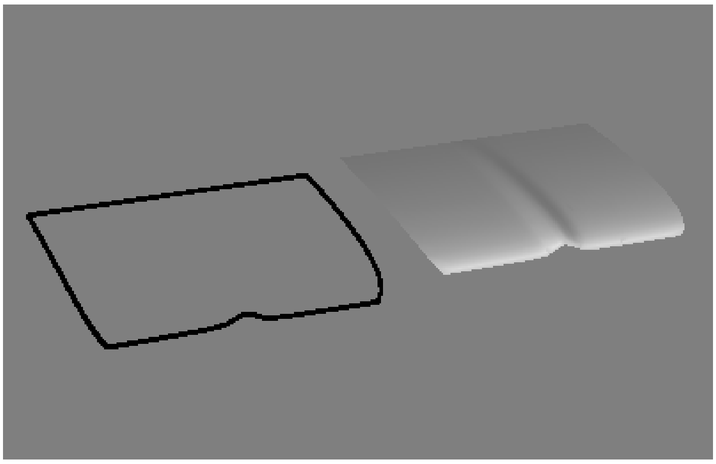


Figure 55: Gordon Object (left: Curves, right: Resulting Gordon Surface)

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.

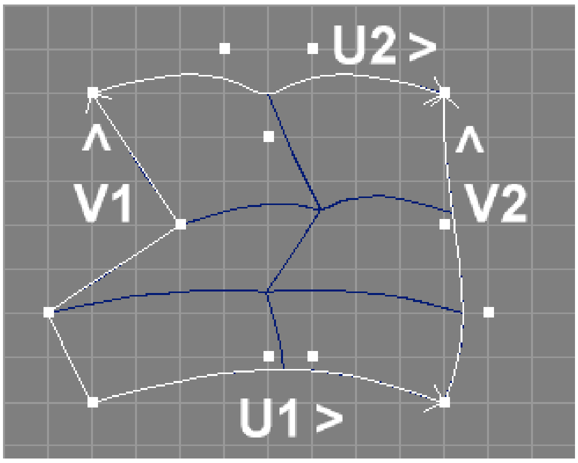



Figure 56: Gordon Surface with Parameter Curves (white)

The curves may be of arbitrary length and order. One may e.g. use a 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, however, 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 automagically. 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 one may specify a third argument (separated from the two sets of parameter curves by a second empty level object). This third argument should be a NURBS patch object that describes all intersection points (by its control points). If present, this intersection patch always takes precedence over the intersection points calculated internally. One may want to add a "`NoExport`" tag 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.

| Type | Parent of | Material | Converts to / Provides | Point Edit |
|---|---|---|---|---|
| Gordon | NCurve*/Level | Yes | NPatch+ | No* |

Table 68: Gordon Object Capabilities

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

The following parameters of the Gordon object further control the creation of the Gordon surface:

### 4.42.1 GordonAttr Property

If the parameter `"WatchCurves"` 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. Note that this works only properly with clamped curves and objects that directly contain editable control points (i.e. it works with NCurve and ICurve objects, but not with Instance or ConcatNC objects). If Ayam can determine which curve was modified last, the other curve that should meet at the endpoint in question will be modified by `"WatchCurves"`. If Ayam finds no information on modifications, the U curves take precedence (i.e. the V curves will be modified).

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 4.29.1 NPatchAttr (page 127) 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.

### 4.42.2 Conversion Support

The Gordon surface may be converted to an ordinary NURBS patch using the main menu entry `"Tools/Convert"`.

### 4.42.3 RIB Export

Gordon objects will be exported as NURBS patch primitives:

```
RiNuPatch(...);
```

PV tags are supported but all NURBS patch primitives will get the same set of tags.[1]

[1] Since 1.20.

## 4.43 Bevel Object

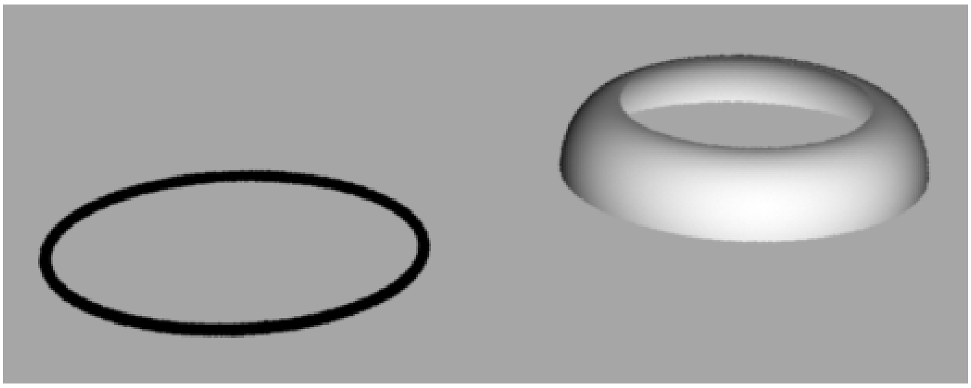


Figure 57: Bevel Object (left: Curve, right: Resulting Bevelled Surface)

The bevel object forms a bevelled surface from a single planar parameter curve. 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 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 5.28 To XY Tool (page 213) 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 run from 0,0 to 1,1. If present, the type parameter is ignored (with one notable exception) as the shape of the bevel is defined by the second curve. If the type is `"RoundCapped"` or `"LinearCapped"` the bevel will be extended to a cap. Note that even if the curve should end at 1,1, this is not mandatory and 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 69: Bevel Object Capabilities

The following parameters of the Bevel object further control the creation of the bevelled surface:

### 4.43.1 BevelAttr Property

- `"BevelType"` lets you choose between different shapes of bevels:
  - `"Round"` a quarter circle,
  - `"Linear"` a straight bevel,
  - `"Ridge"` a more complex ridged surface,
  - `"RoundCapped"` a quarter circle followed by a cap made by placing a single point in the center of the last bevel profile and extending the beveled surface to this point,
  - `"LinearCapped"` a straight bevel followed by a cap made by placing a single point in the center of the last bevel profile and extending the beveled surface to this point.

  Note that the bevel types `"RoundCapped"` and `"LinearCapped"` lead to degenerated NURBS surfaces. If applied shaders or textures look wrong, consider creation of real cap surfaces instead.
- `"BevelRadius"` controls the size 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 affect the bevel size.
- `"BevelRevert"` 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.

See section 4.29.1 NPatchAttr (page 127) for a description of the two attributes `"DisplayMode"` and `"Tolerance"` of the `"BevelAttr"` property.

To help in the exact configuration of the bevel, the `"NPInfo"` field always displays the parameters of the created NURBS patch.

### 4.43.2 Conversion Support

The bevelled surface may be converted to an ordinary NURBS patch using the main menu entry `"Tools/Convert"`.

### 4.43.3 RIB Export

Bevel objects will be exported as NURBS patch primitives:

```
RiNuPatch(...);
```

PV tags are supported.[1]

[1] Since 1.20.

## 4.44 Cap Object

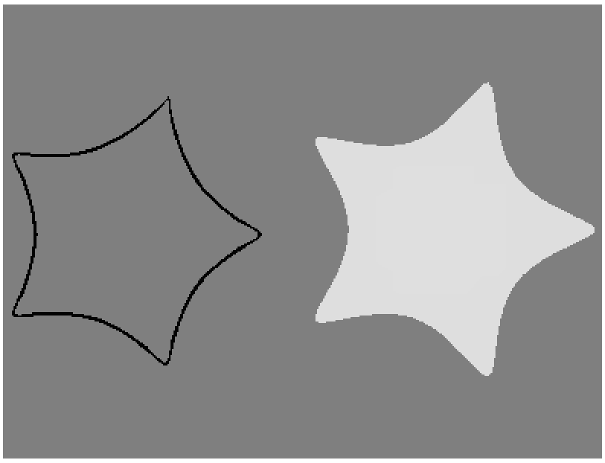


Figure 58: Cap Object (left: Curve, right: Resulting Cap Surface)

The cap object forms a surface that fills a closed planar NURBS curve. 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 4.36.2 Using Holes and Bevels (page 142)).

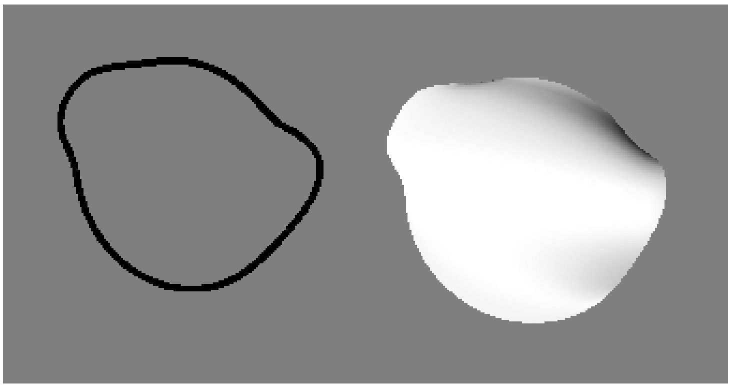


Figure 59: Cap from Non-Planar Curve (left: Curve, right: Resulting Cap Surface)

Since Ayam 1.13 the cap object also supports the so called Gordon mode. Here, only a single parameter curve may be present but, in contrast to the standard mode of operation (Trim), in the Gordon mode the parameter 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 image above for an example).

Consequently, the template for the object hierarchy of a Cap object looks like this:

```
+-Cap
 | Outline(NCurve)
 | [Hole1(NCurve)]
 +-[Hole2(Level)
  | Part1(NCurve)
  \ Part2(NCurve)]
```

Note that, in Trim mode, the curves have to be planar and defined in the XY plane; see section 5.28 To XY Tool (page 213) for information on how to easily achieve this. Furthermore, 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. Be careful when using weights!

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 70: Cap Object Capabilities

The following parameters control the cap creation process:

### 4.44.1 CapAttr Property

The attribute "Type" determines whether a trimmed NURBS surface should be created (type "Trim"), or an untrimmed Gordon surface (type "Gordon"), see also the general discussion about the cap object above.

See section 4.29.1 NPatchAttr (page 127) for a description of the two attributes "DisplayMode" and "Tolerance" of the "CapAttr" property.

To help in the exact configuration of the cap, the "NPInfo" field always displays the parameters of the created NURBS patch.

### 4.44.2 Conversion Support

The cap surface may be converted to an ordinary NURBS patch using the main menu entry "Tools/Convert".

### 4.44.3 RIB Export

Cap objects will be exported as NURBS patch primitives:

```
RiNuPatch(...);
```

PV tags are supported.[1]

[1] Since 1.20.

## 4.45 ConcatNP (Concatenate NURBS Patches) Object

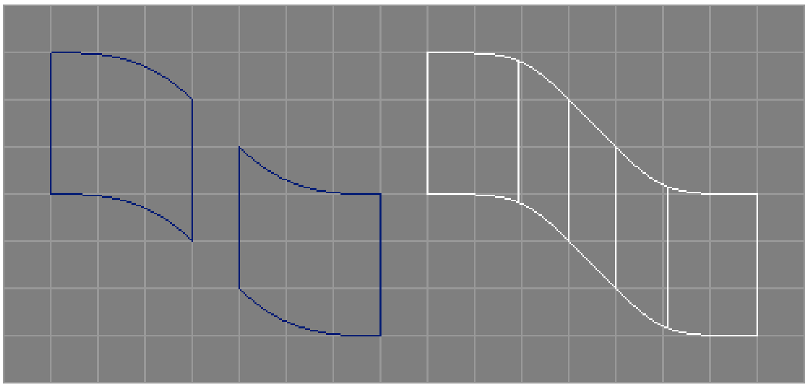


Figure 60: ConcatNP Object (white) From Two NURBS Patches (blue)

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 splits 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]

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).

Attributes like display mode and tolerance for the new concatenated patch are simply taken from the first parameter patch. Eventually present trim curves will currently *not* be honored properly.

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 71: ConcatNP Object Capabilities

The following parameters further control the concatenation process:

### 4.45.1 ConcatNPAttr Property

- 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 curve to close the concatenated curve.
- `"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 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.

[1] Since 1.16. [2] Since 1.20.

- "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 end points will be multiplied in so that a more pleasing fillet shape results in configurations where the distances vary a lot (see also the image below).

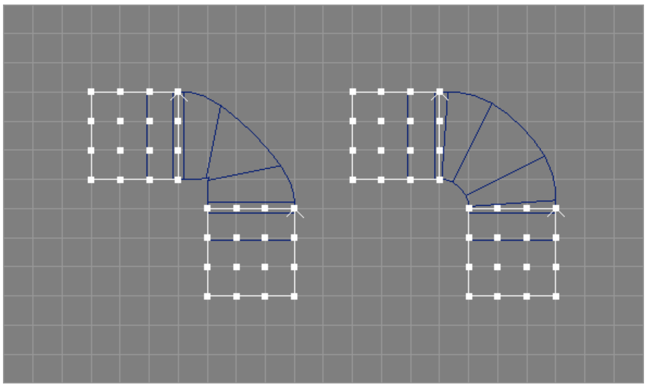


Figure 61: Concatenated Surfaces (blue) with Fillets (left: FTLength 0.3, right: FTLength -0.3

- 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 4.26.1 ConcatNCAttr Property (page 121)). In addition, for "Custom" knots, all parameter surfaces will be elevated to a common maximum order or atleast be clamped in the respective direction prior to the splitting to curves but after the fillet creation.
- 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. To connect two surfaces that share the same orientation "over a corner" one would e.g. need to set "UVSelect" to "uV" (see also the image below). The default, an empty string, is equivalent to "u" for all patches. Also incomplete strings 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 that they can be split along the U direction.

- Finally, a "NPInfo" field informs about the actual configuration of the created NURBS patch.

### 4.45.2 Conversion Support

The concatenated surface may be converted to an ordinary NURBS patch using the main menu entry "Tools/Convert".

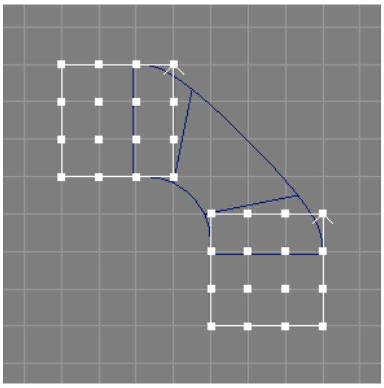


Figure 62: Concatenating two Surfaces with UVSelect == "uV"

### 4.45.3 RIB Export

ConcatNP objects will be exported as NURBS patch primitives:

```
RiNuPatch(...);
```

PV tags are supported.[1]

[1] Since 1.20.

## 4.46 ExtrNP (Extract NURBS Patch) Object

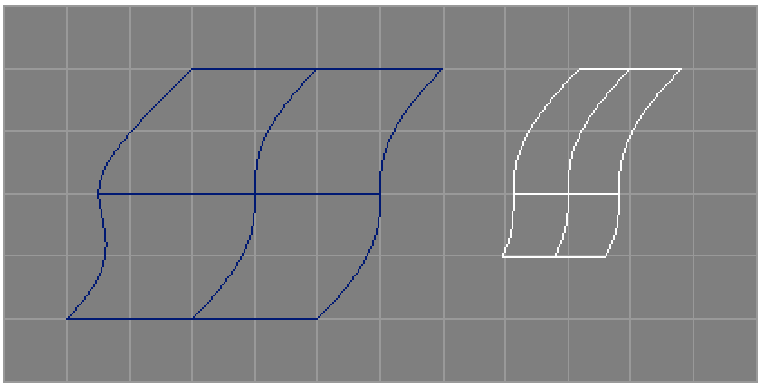


Figure 63: Extracted Surface (white) from Arbitrarily Shaped Surface (blue)

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]

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 72: ExtrNP Object Capabilities

The extraction process is controlled by the following attributes:

### 4.46.1 ExtrNPAttr Property

- `"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.

- "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 4.29.1 NPatchAttr (page 127) 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.

### 4.46.2 Conversion Support

The extracted surface may be converted to an ordinary NURBS patch using the main menu entry "Tools/Convert".

### 4.46.3 RIB Export

ExtrNP objects will be exported as NURBS patch primitives:

```
RiNuPatch(...);
```

PV tags are supported.[2]

[1] Since 1.15. [2] Since 1.20.

## 4.47 OffsetNP (Offset NURBS Surfaces) Object

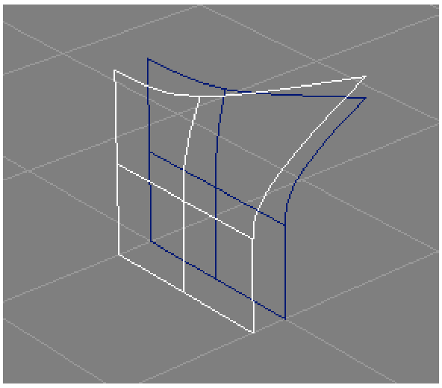


Figure 64: Offset Surface (white) from NURBS Surface (blue) with Offset 0.2

The OffsetNP object creates offset surfaces from NURBS surfaces using a simple algorithm.[1] The offsetting also works for closed and periodic surfaces in any possible combinations in the two dimensions.[2]

Note that degenerate and rational surfaces are still not supported.

The offset surface will always match the original surface in width/height, orders, and knots. See also the image above.

As the geometry of the offset surface is completely defined by the master surface, OffsetNP objects do not support own transformation attributes.[3]

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 73: OffsetNP Object Capabilities

The following parameters further control the offsetting process:

### 4.47.1 OffsetNPAttr Property

- `"Offset"` determines the distance between original surface and offset surface. Negative values are allowed.
- See section 4.29.1 NPatchAttr (page 127) 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.

### 4.47.2 Conversion Support

The offset surface may be converted to an ordinary NURBS patch using the main menu entry `"Tools/Convert"`.

[1] Since 1.17. [2] Since 1.19. [3] Since 1.19.

### 4.47.3 RIB Export

OffsetNP objects will be exported as NURBS patch primitives:

```
RiNuPatch(...);
```

PV tags are supported.[1]

[1] Since 1.20.

## 4.48 Text Object


Figure 65: 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.

You may convert the current Text object 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 74: Text Object Capabilities

The following attributes control the creation of the text objects.

### 4.48.1 TextAttr Property

- Using `"FontName"` you specify a TrueType font description file. 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 4.36.1 ExtrudeAttr Property (page 142) for a more exhaustive description of those parameters).
- `"Add Start Bevel!"`, `"Add End 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 Start Bevel!"` and `"Add End 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 4.43.1 BevelAttr Property (page 163). Just one note: for some fonts, the bevel radius has to be set to really small values (about 0.0008) 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 high values for the bevel radius.

See section 4.29.1 NPatchAttr (page 127) for a description of the other two attributes `"DisplayMode"` and `"Tolerance"`.

### 4.48.2 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.

### 4.48.3 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.

## 4.49 Trim Object

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] 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 curves must obey the rules for trimming as outlined in section 4.29.2 Trim Curves (page 128). 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 75: Trim Object Capabilities

The following parameters further control the trimming process:

### 4.49.1 TrimAttrib Property

- "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.

### 4.49.2 Conversion Support

The trimmed surface may be converted to an ordinary NURBS patche using the main menu entry "Tools/Convert".

### 4.49.3 RIB Export

Trim objects will be exported as NURBS patch primitives:

```
RiNuPatch(...);
```

PV tags are currently not supported.

[1] Since 1.16.

## 4.50 Script Object

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 222)).

Script objects may also use arbitrary, plugin provided, scripting languages, like e.g. JavaScript, provided by the "`jsinterp`" plugin (see also: 6.6 JavaScript Scripting Interface (page 280)).[1]

### 4.50.1 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. Ayam 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 Index of Procedures and Commands (page 224)).

In addition, access to global variables 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 (e.g. scripts that run in the Ayam console).

With the help of scripts, that run in the Ayam interpreter, more commands may be transfered to or made available in the safe interpreter. But this may, of course, open security holes again.

You can also still re-enable full access from script objects to the complete scripting interface by recompiling Ayam. If you do so, for security reasons, if 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 the "`Active`" script object property and may be enabled after careful inspection of the script code manually or using the main menu entry "`Special/Enable Scripts`".

### 4.50.2 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

[1] Since 1.18.

script emergency hotkey `<Ctrl+Shift+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!

Since Ayam 1.8.2 script objects may also create their own property GUIs for e.g. script parameters. This may be accomplished by adding tags of type "NP" with the name of the new property as value to the script object. The script itself is responsible for data management and property GUI creation.

Since Ayam 1.9 the parameters set via a new property GUI may be comfortably saved with scene files by just adding the following comment as first line to the script:

```
# Ayam, save array: <arrayname>
```

where arrayname designates the name of the global Tcl array the property GUI elements operate with. Note that with the introduction of individual parameters in Ayam 1.15 not all members of the arrays will be saved. Only members from the "SP" list (see below) will be considered, which gives script object developers fine grained control over what actually goes to the Ayam scene file.

Since Ayam 1.12 there is also a tag type to remove properties ("RP"). Using this tag, one can e.g. hide the actual script code and block users from unintentionally changing it.

Since Ayam 1.15 script objects can have individual sets of parameters, so that multiple copies of a script object can be used (without individual parameters all copies would share the same parameter values). For this, the new variable "SP" in the script data array contains a list of parameter names in that array that are individual (not shared by all copies of the script object).

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 the child level of the respective script object. Furthermore, the object clipboard is saved for running the script and re-established after the script finished.

Since Ayam 1.18 Script objects may also use arbitrary, plugin provided, scripting languages. 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 word "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 */
```

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.

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

The next section discusses the available script object types and additional controlling parameters.

| Type | Parent of | Material | Converts to / Provides | Point Edit |
|---|---|---|---|---|
| Script | Any+ | No | Any+ | No* |

Table 76: Script Object Capabilities

### 4.50.3 ScriptAttr Property

- 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:
  - "`Run`", the script will be run and no special action will take place.
  - "`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
```

  - "`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 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 providing NURBS curves (interpolating curves 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 standard Tcl text widget that allows to directly edit the code. 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. If the first line of the script is a comment like

```
# 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 objects need to store differing/individual amounts of parameter data, lists can be used as individual parameters.

#### 4.50.4 Script Object Examples

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 +/- 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
```

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 ("# ^^^" lines designate changed or added code):

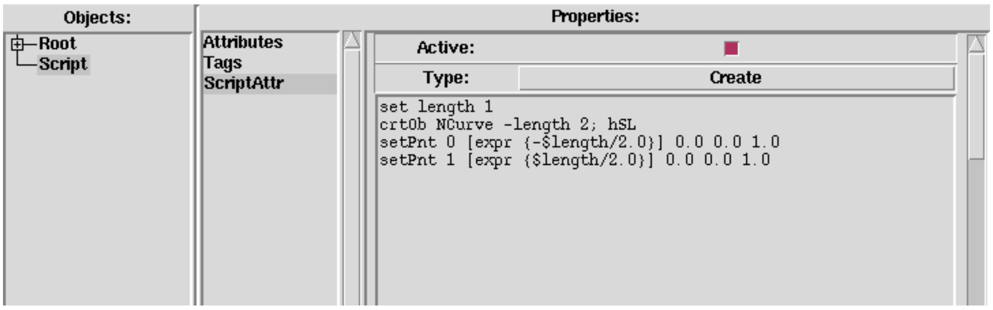



Figure 66: Simple Script for Parametric Line Primitive

```
# 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
```

and by adding a "NP" (new property) tag to the script object with the value "LineAttr", resulting in a new clickable graphical user interface as can be seen in the following image:


Figure 67: Parametric Line Primitive with Parameter GUI

The GUI setup code 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 set on the first run of "addPropertyGUI") first. See also section 6.1.4 Global Property Management and Data Arrays (page 223) 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.

In addition, a second example 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
pasmovOb
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 1 length unit on the trajectory) with objects. One may now modify the curve using interactive modelling actions, change its length, and the trajectory is always completely filled. See example image below (compare the Clone configurations from the two different trajectory curves):

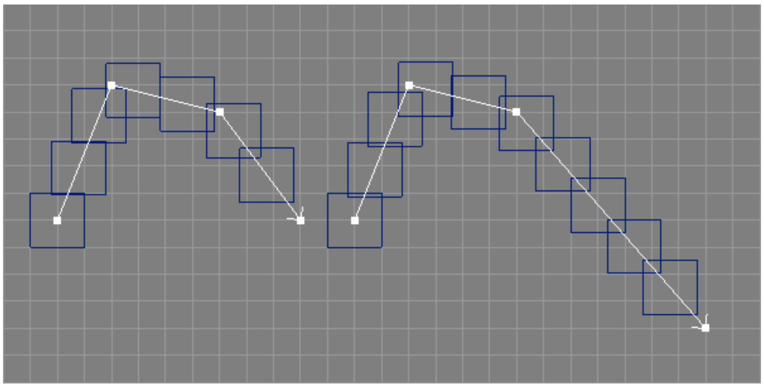


Figure 68: Hierarchy Building Script Object Example

### 4.51 Custom Objects

Custom objects are plugins that extend the Ayam capabilities by defining totally new types of e.g. geometric objects. This may be done easily, because the Ayam core is written in a modelling paradigm independent way.

A simple example of a custom object is the CSphere, which implements a simple sphere and has a new property named `"CSphereAttr"`. This property contains all parameters of a simple RenderMan Interface quadric sphere. A more complex example would be the MetaObj custom object. It is possible, but not planned for now, to integrate the T-Spline modelling paradigm into Ayam this way.

Since a custom object has total control over properties and representations, you should refer to the documentation of the respective custom object for more information regarding its properties.

One custom object is already distributed with Ayam. This Metaball custom object is documented in the next section:

## 4.52 Metaball Object

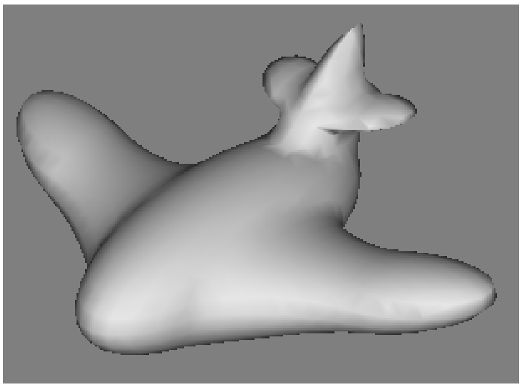


Figure 69: A Metaball Object from Six Meta Components

A metaball object is a custom object (see also section 4.51 Custom Object (page 182)). 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 77: 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).

This process may be further parameterised using the "MetaObjAttr" property:

### 4.52.1 MetaObjAttr Property

- 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.

### 4.52.2 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:

### 4.52.3 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.

### 4.52.4 Torus

- "Ri" the inner radius of the torus
- "Ro" the outer radius if the torus
- "Rotate" rotates the torus about 90 degree

#### 4.52.5 Cube

- "EdgeX", "EdgeY", and "EdgeZ", let you define the sharpness of the edges of the cube

#### 4.52.6 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 has to return the field value in the global variable "f". Here is an example for a custom expression:

```
set f [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.

#### 4.52.7 Conversion Support

Metaball objects may be converted to PolyMesh objects using the main menu entry "Tools/Convert".

#### 4.52.8 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.

## 4.53 SDNPatch Object

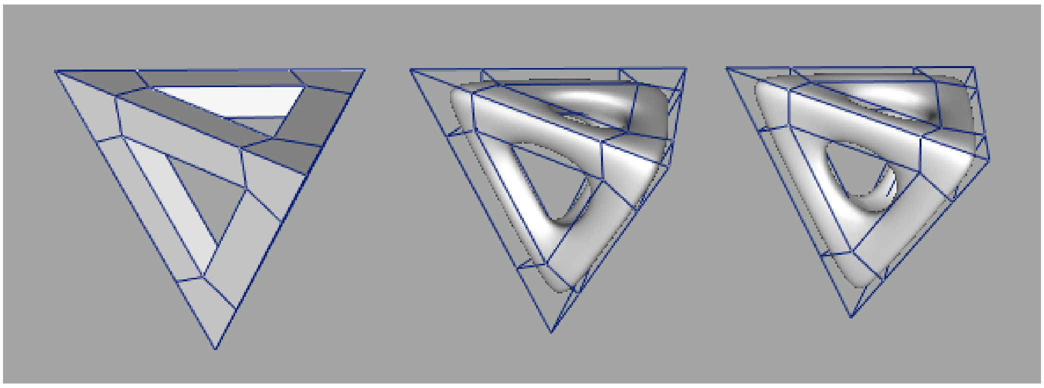


Figure 70: SDNPatches, l: control mesh, m: subdivided mesh with knot, r: subdivided mesh without knot

The SDNPatch custom object is available since Ayam 1.16 and allows to model with Subdivision NURBS, which extend the traditional subdivision schemes with knot values (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, SDMesh objects may be converted in two steps to SDNPatch objects, 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 78: SDNPatch Object Capabilities

### 4.53.1 SDNPatchAttr Property

The SDNPatchAttr property allows you 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.

### 4.53.2 SDNPatch Modelling Actions

This section, briefly, explains the special modelling actions defined for the SDNPatch custom object. In order to select a face or edge for such an operation, just select all the control points defining the face or edge. All modelling actions can be started via the "Custom/SDNPatch" main menu.

- "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. If this operation is picking the wrong direction, try to revert all faces first, see below. Since Ayam 1.17 this operation has two parameters 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 is removed from the mesh.
- "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 vertice distances: you need to move the two faces near each other to make clear, how the new connection shall be established.
- "Face Connect", 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 via a set of new faces. The decision, between which vertices of the faces the new faces are built, depends on the relative vertice distances: you need to move the two faces near each other to make clear, how the new 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.
- "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).

### 4.53.3 Conversion Support

SDNPatch objects may be converted to PolyMesh objects using the main menu entry "Tools/Convert".

### 4.53.4 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.

## 4.54 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".

### 4.54.1 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. Read the next section for more information on how to use these entries. Read it!
- "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 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 occured.

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 y-angle degrees then around Z z-angle degrees. In fact, no information about the order in which partial rotations occured may be derived from that three values. This implies, that e.g. the values 0 0 45 may denote a different orientation than the very same values 0 0 45 (no joke)!

But how do you get the three entries to do what you want? You either want to rotate the object around an axis by a given amount or you want to undo a rotation or undo all rotations.

Rotating an object is easy, simply *add* the amount about which you want to rotate the object to the value currently displayed in the appropriate entry. If you want to rotate about 45 degrees about X and the x-angle entry displays a 30, enter 75. Then press the apply button.
If you change multiple entries 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 state) is simple too: enter 0 for all three entries at once, then press apply.

If you want to copy just the orientation of an object to other objects using the property clipboard, mark/select all Translate and Scale property elements, then use `"Edit/Copy Property"`. Just marking the Rotate elements and then using `"Edit/Copy Marked Prop"` will *not* work, because the quaternion will not be copied properly.

### 4.54.2 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.

### 4.54.3 Material Property

The `"Material"` property allows you to connect geometric objects to material objects (see also section 4.6 Material Object (page 87)). 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 you change the name, 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.

### 4.54.4 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.

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 4.54.4 Working with Shaders (page 191) below.

All other elements of the shader property GUI depend on the currently attached shader.

**Shader Parsing**

If no plugin is loaded, the official Ayam binaries use libslcargs (from BMRT) to parse shaders that have been compiled with slc (the shader compiler from BMRT). Parsing incorporates detecting the type of the shader and detecting the names, types, and default values of all shader arguments.

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.

Many 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. 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/") 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 8.7 Shader Parsing Plugins (page 318).

**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 if you have to deal with changing shaders: just edit the shader, recompile it, then back in Ayam just hit the "Default Values."-button. Note that this destroys your possibly carefully adjusted shader argument values.

If you want to keep the old shader argument values when a shader changes, simply copy the shader property using the property clipboard (main menu: "Edit/Copy Property") before you load the new default values and paste the property back using "Edit/Paste Property" after loading of the new default values. Beware! This works properly only, if you do not change the type of existing shader arguments and if no shader arguments are removed in the new version of the shader.

You can also just copy certain parameter values (shader arguments whose types do not change) by selecting 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)).

#### 4.54.5 Tags Property

Use the "Tags" property to edit the tags of an object.

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.

The tags property GUI consists of the following standard elements:

- "Remove all Tags!" immediately removes all tags from the object.
- "Remove Tag!" is a menu, that allows you to select and remove a single tag from the object.
- "Add Tag!" opens a small dialog box, where you may enter a new tag type and value. Once you press 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 next sub-sections describe the tag types currently available in Ayam and the plugins distributed with Ayam. Note that extensions and plugins may define their own types.

## 4.55 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.

This section documents all currently known tag types.

### 4.55.1 RiAttribute Tag

**Description**

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.

"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 311).

### 4.55.2 RiOption Tag

**Description**

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.

**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 311).

### 4.55.3 TC (Texture Coordinates) Tag

**Description**

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 you change the texture coordinates of entire primitives only, if you want to change the texture coordinates of sub-primitives (e.g. of single control points of a NURBS patch) you would need to use a "`PV`" (Primitive Variable) tag 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.

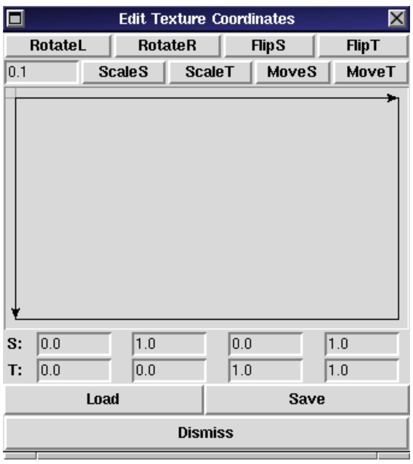



Figure 71: Texture Coordinate Editor

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 you, for example, want to correct a texture mapping 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 you 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 you can use the property clipboard 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 modeless, it may stay open while you model. The "Dismiss" button closes the dialog.

### 4.55.4 PV (Primitive Variable) Tag

#### Description

The tag type "PV" can be used to attach arbitrary data to geometric primitives and even sub-primitives. With the help of primitive variables you can e.g. attach your own texture coordinates to a NURBS patch primitive or attach 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 your 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.

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, check the number and silently omit the primitive variable if there are mismatches. Check your RIB for the presence of the primitive variable after export, especially, if you are adding or editing PV tags manually.

### 4.55.5 RiHider Tag

**Description**

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 RiAttribute tag: "<type>,<parameterlist>" where "<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
```

### 4.55.6 RiDisplay Tag

**Description**

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. The syntax of a RiDisplay tag is as follows: "<name>,<type>,<mode>,<parameterlist>", where name is e.g. 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 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.

### 4.55.7 NoExport Tag

**Description**

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).

### 4.55.8 SaveMainGeom Tag

**Description**

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.

### 4.55.9 SavePaneLayout Tag

**Description**

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.

#### 4.55.10 TP (Tesselation Parameter) Tag

**Description**

The tag type "TP" can be used to save tesselation parameters to objects of type "NPatch" (and objects that may be converted to "NPatch" objects). Those tesselation parameters will be used when the NPatch object is tesselated for e.g. a conversion to a PolyMesh object. The syntax of the TP tag is: "<tmethod>,<tparamu>,<tparamv>" where "<tmethod>" is an integer value between 1 and 3, describing which tesselation method to use (1 – ParametricError, 2 – PathLength, and 3 – DomainDistance) and "<tparamu>" and "<tparamv>" are float values describing the respective parameter value for the chosen tesselation method. The second parameter value is ignored for the tesselation methods 1 and 2.

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 tesselation GUI, that can be started with the main menu entry "Tools/Surface/Tesselate" (see also section 5.49 Tesselation Tool (page 220)).

**Example**

A TP tag could look like this:

```
TP 1,0.5,0.6
```

#### 4.55.11 DC (Depth Complexity) Tag

**Description**

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 8.10 CSG preview using the AyCSG plugin (page 320) for more information regarding the depth complexity value.

**Example**

A DC tag (valid for e.g. a torus) could look like this:

```
DC 2
```

#### 4.55.12 NP (New Property) Tag

**Description**

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
```

#### 4.55.13 RP (Remove Property) Tag

**Description**

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.

#### 4.55.14 BNS (Before Notify Script) Tag

**Description**

The tag type "BNS" (before notify script) may be used to add scripts to an object, that will be run *before* the notification callback of that object completed. The notification, in turn, will be executed because e.g. one of the children of the object changed.

**Example**

A simple BNS tag could look like this:

```
BNS puts "notify callback about to fire"
```

**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 4.50.1 Safe Interpreter (page 176). Consequently, the warning dialog that appeared when files with BNS tags were loaded is also gone.

You can still re-enable the old functionality by recompiling Ayam. If you do so, for security reasons, if scene files containing BNS tags are loaded, Ayam will again raise the warning offering to temporarily disable all such tags that will be read. 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".

#### 4.55.15 ANS (After Notify Script) Tag

**Description**

The tag type "ANS" (after notify script) may be used to add scripts to an object, that will be run *after* the notification callback of that object completed. The notification, in turn, will be executed because e.g. one of the children of the object changed.

**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 4.50.1 Safe Interpreter (page 176). Consequently, the warning dialog that appeared when files with ANS tags were loaded is also gone.

You can still re-enable the old functionality by recompiling Ayam. If you do so, for security reasons, if scene files containing ANS tags are loaded, Ayam will again raise the warning offering to temporarily disable all such tags that will be read. To disable a ANS tag, Ayam simply changes its type from "ANS" to "DANS" (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 "ANS" or by using the main menu entry "Special/Enable Scripts".

#### 4.55.16 UMM/VMM (U/V Min Max) Tag

**Description**

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 the RIB export does.

**Example**

An UMM tag could look like this:

```
UMM 0.4,0.6
```

#### 4.55.17 BP (Bevel Parameters) Tag

**Description**

The tag type "BP" (bevel parameters) is used by all bevel supporting tool objects to store their bevel information. 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, where the bevel should be applied to, "<type>" is an integer value from 0 - 4 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
```

### 4.55.18 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 original 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.
- NO/NM Tags These tags are internal binary tags, that transport the notification across the scene.

### 4.55.19 List of Known Tags

This section contains a comprehensive list of tag names, that are known since Ayam 1.10 and in all accompanying extensions (plugins).

`"RiAttribute", "RiOption", "RiHider", "RiDisplay", "NoExport", "TC", "PV", "SaveMainGeom", "SavePaneLayout", "TP", "MI", "OI", "DC", "TM", "NP", "RP", "BNS", "DBNS", "ANS", "DANS", "NO", "NM", "UMM", "VMM", "BP", "IDR", "IIDR", "RIDR", "R3IDR", "CIDR", "CCIDR"`

The following tags are for internal use (they will not be displayed in the Tags property of an object): `"MI", "OI"`

Tags of type `"NO", "NM"` are binary, they also never show up in the Tags property.

Documentation on all those tags can be found in the sections above.

# 5 NURBS Modelling Tools

This section describes NURBS curve and surface related modelling tools.

## 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.

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.

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 selected points will be removed from the object. Then the notification of the object will be run. The notification of the parent object(s) will be run after processing of all selected objects finished.

## 5.2 Circular B-Spline Tool

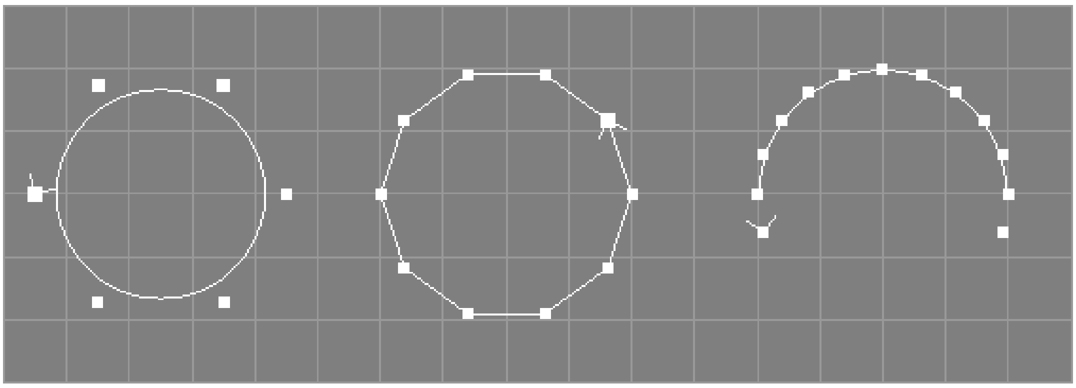


Figure 72: B-Spline Curves Created by the Circular B-Spline Tool

- Arguments: Radius, Arc, Sections, Order.
- Operation: This tool creates a non-rational B-Spline curve with Sections+1+Order (Arc=360.0) or Sections+1+(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.

- Note: Sections must be atleast 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 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 4.22.2 Multiple Points (page 113)). 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).

## 5.3 NURBCircle Tool

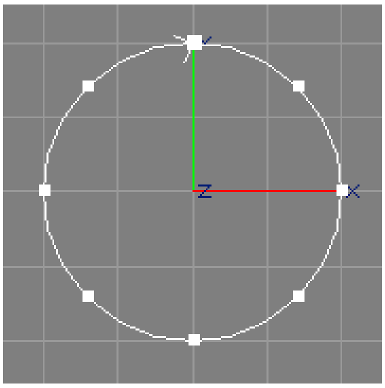



Figure 73: A NURBS Circle

- 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. 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.
- Note: 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). Use a closed B-Spline created with the Circular B-Spline tool (see above) if you want to edit the curve further. 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 4.22.2 Multiple Points (page 113). You can use a NURBS circle created by this tool 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.

## 5.4 TrimRect Tool

- Arguments: None
- Operation: The TrimRect tool creates a non-rational, two-dimensional piecewise linear NURBS curve of rectangular shape in the XY plane, that fits in the (u,v) parameter space of a NURBS patch, for use as trim curve.

- Note: To fit the curve to the parameter space of a NURBS patch, the NURBS patch object should be selected or the current level should be inside the NURBS patch. If no NURBS patch object is selected and the current level is not inside a NURBS patch, a curve with the coordinates (-1,-1), (-1,1), (1,1), and (1,-1) will be created 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 4.22.2 Multiple Points (page 113). See section 4.29.2 Trim Curves (page 128) for a more detailed discussion of trim curves and how to use the rectangular curve created by the TrimRect tool.

## 5.5 NURBSphere Tool

- Arguments: None.
- Operation: The NURBSphere tool creates a half circle NURBS curve and revolves it about the Y axis thus forming a sphere of radius 1.
- Note: The NURBS curve is deleted afterwards.

## 5.6 NURBSphere2 Tool

- Arguments: None.
- Operation: The NURBSphere tool creates a Cobb-NURBSphere, consisting of six NURBS patches.
- Note: The NURBS patches are of high order (5).

## 5.7 Revolve Tool

- Arguments: The revolve tool takes the selected objects from the selection.
- Operation: The tool creates a Revolve object, and moves the selected objects to it.

```
|-NCurve      ==>      +-Revolve
                        \-NCurve
```

- Note: See section 4.35 Revolve Object (page 139) 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.

## 5.8 Extrude Tool

- Arguments: The extrude tool takes the selected objects from the selection.
- Operation: The tool creates an Extrude object, and moves the selected objects to it.

```
|-NCurve               +-Extrude
|-NCurve      ==>       |-NCurve
                        \-NCurve
```

- Note: See section 4.36 Extrude Object (page 141) 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.

## 5.9 Sweep Tool

- Arguments: The sweep tool takes the selected objects from the selection.
- Operation: The tool creates a Sweep object, and moves the selected objects to it.

```
|-NCurve              +-Sweep
|-NCurve      ==>      |-NCurve
                      \-NCurve
```

- Note: See section 4.38 Sweep Object (page 146) 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.

## 5.10 Cap Tool

- Arguments: The cap tool takes the selected objects from the selection.
- Operation: The tool creates a Cap object, and moves the selected objects to it.

```
|-NCurve              +-Cap
|-NCurve      ==>      |-NCurve
                      \-NCurve
```

- Note: See section 4.44 Cap Object (page 164) 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.

## 5.11 Birail1 Tool

- Arguments: The birail1 tool takes the selected objects from the selection.
- Operation: The tool creates a Birail1 object, and moves the selected objects to it.

```
|-NCurve              +-Birail1
|-NCurve      ==>      |-NCurve
|-NCurve               |-NCurve
                      \-NCurve
```

- Note: See section 4.39 Birail1 Object (page 150) 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.

## 5.12 Birail2 Tool

- Arguments: The birail2 tool takes the selected objects from the selection.
- Operation: The tool creates a Birail2 object, and moves the selected objects to it.

```
|-NCurve              +-Birail2
|-NCurve               |-NCurve
|-NCurve      ==>      |-NCurve
|-NCurve               |-NCurve
                      \-NCurve
```

- Note: See section 4.40 Birail2 Object (page 153) 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.

## 5.13 Gordon Tool

- Arguments: The gordon tool takes the selected objects from the selection.
- Operation: The tool creates a Gordon object, and moves the selected objects to it.

```
|-NCurve              +-Gordon
|-NCurve               |-NCurve
|-NCurve      ==>      |-NCurve
|-NCurve               |-NCurve
                       \-NCurve
```

- Note: See section 4.42 Gordon Object (page 159) 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.

## 5.14 Skin Tool

- Arguments: The skin tool takes the selected objects from the selection.
- Operation: The tool creates a Skin object, and moves the selected objects to it.

```
|-NCurve              +-Skin
|-NCurve      ==>      |-NCurve
|-NCurve               |-NCurve
                       \-NCurve
```

- Note: See section 4.41 Skin Object (page 156) 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.

## 5.15 Revert Tool

- Arguments: The revert tool takes all NCurve, ICurve, and ACurve objects from the selection.
- Operation: The direction of the selected curves will be reversed. This tool also reverts the relative knot distances of NURBS curves so that a NURBS curve defined on the (asymmetric) knot vector

  `"0.0 0.0 0.0 0.75 1.0 1.0 1.0"` will get the knot vector

  `"0.0 0.0 0.0 0.25 1.0 1.0 1.0"` 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.
- Note: The direction of a curve is shown as a small arrow at the end of the curve. Eventually selected points will still be selected after this operation. See also the documentation of the corresponding scripting interface command 6.2.11 revertC (page 252).

## 5.16 Concat Tool

- Arguments: The concat tool takes two NURBS curves from the selection.
- Operation: The selected NURBS curves will be concatenated and a new third curve will be created.

```
|-NCurve            |-NCurve
|-NCurve     ==>    |-NCurve
                    |-NCurve
```

- Note: If one of the curves has weights, the resulting curve will have weights too. If the knot type of the first curve is `"Custom"`, it will be converted to `"NURB"`, otherwise the knot type of the new curve will be that of the first selected curve. Due to those changes of the knot values, the resulting curve might differ from the original curves. The original NURBS curves will not be deleted by this tool. See also section 4.26 ConcatNC Object (page 120).

## 5.17 Split Tool

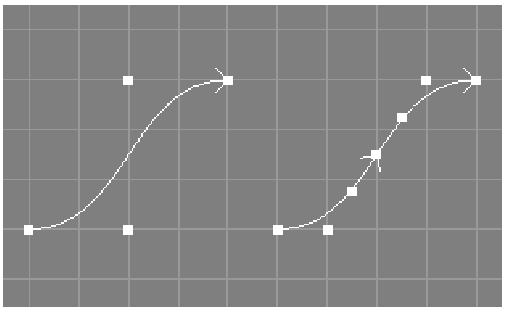


Figure 74: Split Tool (left: original curve, right: resulting split curves for t=0.5)

- Arguments: The split curves tool takes a single NURBS curve from the selection and additionally requests a parametric value.
- Operation: The selected NURBS curve will be split into two new NURBS curves at the designated parametric value. The splitting process involves application of knot insertion, so that both new curves will get a custom knot vector.

```
|-NCurve     ==>    |-NCurve
                    |-NCurve
```

- Note: The original selected NURBS curve will be changed and form the first of the two new curves, so you may want to keep a copy of it somewhere. See also the documentation of the corresponding scripting interface command 6.2.11 splitNC (page 252).

## 5.18 Trim Tool

- Arguments: The trim curves tool takes all selected NURBS curves from the selection and additionally requests two parametric values, umin and umax.
- Operation: The selected NURBS curves will be trimmed to the designated parametric range (umin, umax).

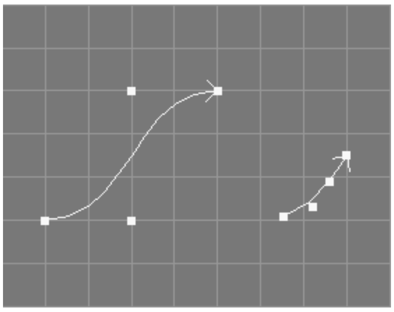


Figure 75: Trim Tool (left: original curve, right: resulting trimmed curve for umin=0.1, umax=0.5)

- Note: The trimming process involves the application of knot insertion so that the curves will get a custom knot vector. See also the documentation of the corresponding scripting interface command 6.2.11 trimNC (page 253).

## 5.19 Elevate Tool

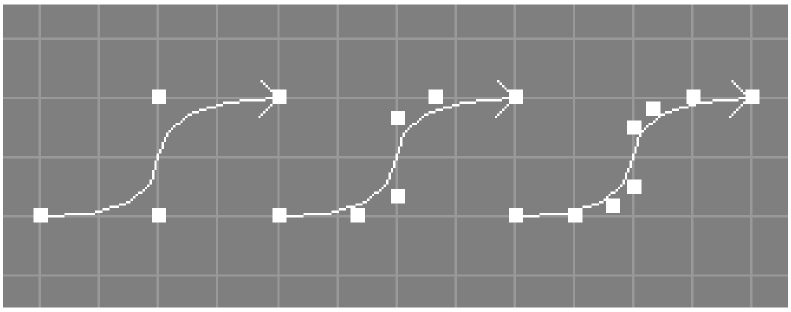


Figure 76: Successive Application of Elevate Tool (Order 3 (left), 4 (middle), 5 (right))

- 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.
- Note: 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. See also the documentation of the corresponding scripting interface command 6.2.11 elevateNC (page 250).

## 5.20 Refine Tool

- Arguments: The refine tool takes a number of NURBS curves from the selection.
- Operation: The selected NURBS 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 curves, no control points will be inserted in the last p intervals, this allows to maintain the periodicity, see also the image below where a periodic curve of length 9 (8 sections) has

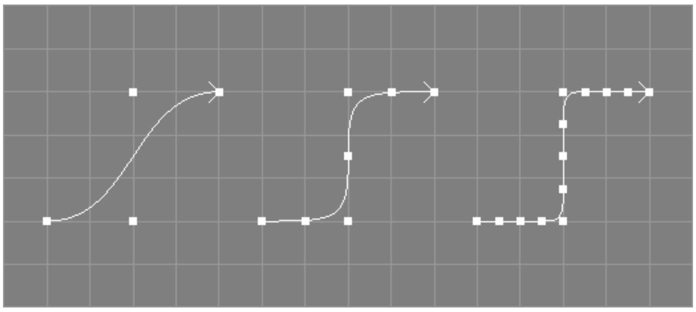


Figure 77: Successive Application of Refine Tool

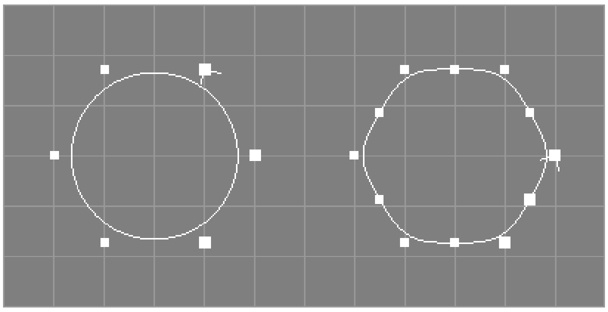


Figure 78: Refining a Periodic Curve

been refined, resulting in a periodic curve of length 15 (not 17). If there are selected points, only the intervals between the first and the last selected point are refined.

- See also the documentation of the corresponding scripting interface command 6.2.11 refineNC (page 251).

## 5.21 Refine Knots Tool

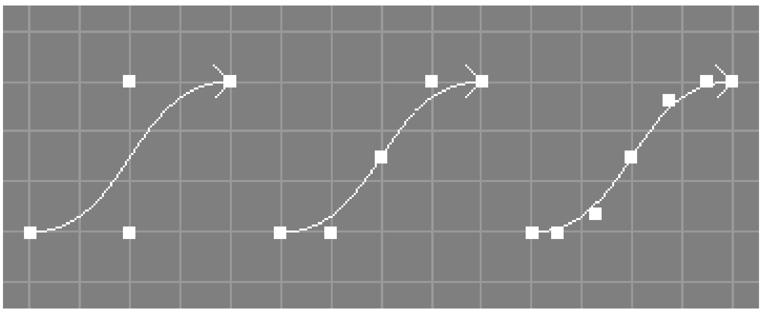


Figure 79: Successive Application of Refine Knots Tool

- 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.
- Note: Because a new knot is inserted in the middle of each interval, knot vectors of type NURB and B-Spline will not change in type. See the image above for an example of a successive refinement of a simple NURBS curve. Note that the shape of the curve does not change, but the position of certain

control points does. See also the documentation of the corresponding scripting interface command 6.2.11 refineNC (page 251).

## 5.22 Coarsen Tool

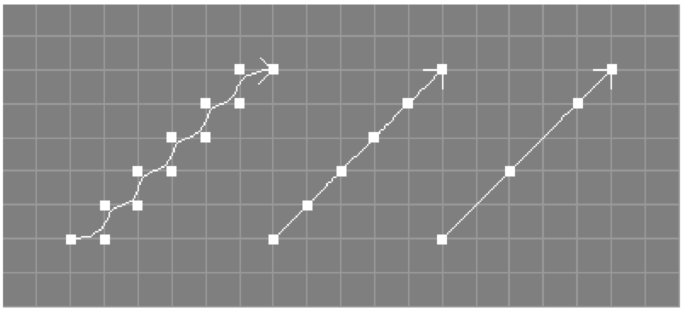


Figure 80: Successive Application of Coarsen Tool

- Arguments: The coarsen tool takes a number of NURBS curves from the selection.
- Operation: Every second control point in the control vectors of the selected NURBS curves will be deleted.
- Note: For periodic 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 curves with custom knot vectors. See also the documentation of the corresponding scripting interface command 6.2.11 coarsenNC (page 251).

## 5.23 Clamp Tool

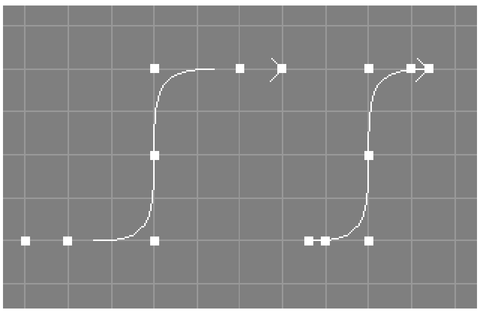


Figure 81: Clamp Tool (left: original curve, right: clamped curve)

- 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.
- Note: The knot type of the curves will be changed to `"Custom"`. The point selection will be removed from the original objects. 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 6.2.11 clampNC (page 250).

### 5.24 Insert Knot Tool

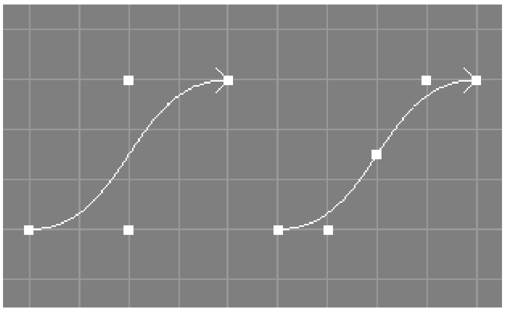


Figure 82: Insert Knot Tool (left: original curve, right: a knot has been inserted 1 time at t=0.5)

- Arguments: The insert knot tool takes a number of NURBS curves from the selection and requests two additional values, a parametric value t and an integer value i.
- Operation: The specified knot (t) will be inserted i times into the knot vector of the selected curves, without changing the shape of the curve(s).
- Note: The knot type of the curves will be changed to `"Custom"`. The point selection will be removed from the original objects. See also the documentation of the corresponding scripting interface command 6.2.11 insknNC (page 251).

### 5.25 Remove Knot Tool

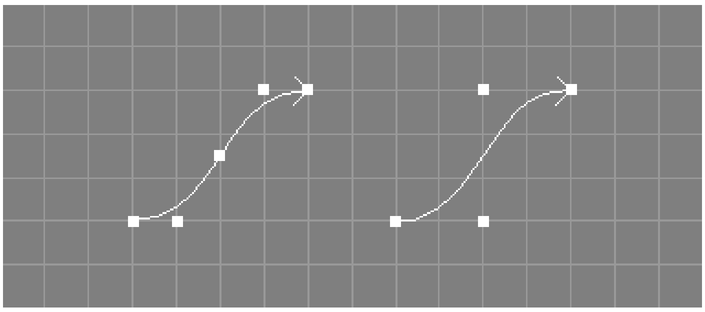


Figure 83: Remove Knot Tool (left: original curve, right: the knot at t=0.5 has been removed 1 time)

- 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 i, and a tolerance tol.
- Operation: The specified knot (t) will be removed i 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 (by entering -i index instead of a parametric value t, i.e. `"-i 4"` instead of `"0.5"`). If the knot can not be removed r times due to the tolerance, an error is reported and the original curve is left unchanged.

- Note: If tol is `"Inf"` (infinity) the tool tries to work without changing the shape of the curves, however, this is not guaranteed. The point selection will be removed from the original objects. See also the documentation of the corresponding scripting interface command 6.2.11 remknNC (page 251).

## 5.26 Plot Curvature Tool

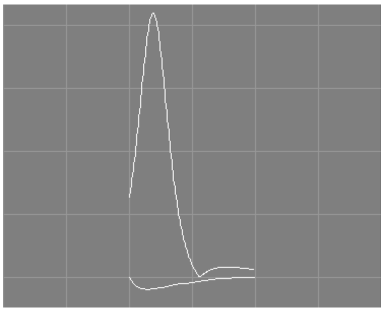


Figure 84: Curvature Plot (top) of simple NURBS curve (bottom)

- 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.

## 5.27 Shift Closed Curve Tool

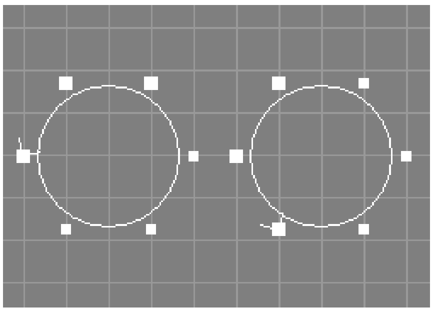


Figure 85: Shift Closed Curve Tool (left: Closed B-Spline Curve, right: Shifted Curve)

- 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.
- Operation: The control points of the curve(s) will be shifted. 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. The shifting process will be repeated according to the integer parameter i given. The parameter i may be negative to revert the direction of the shifting.

- Note: 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 6.2.11 shiftC (page 253).

## 5.28 To XY Tool

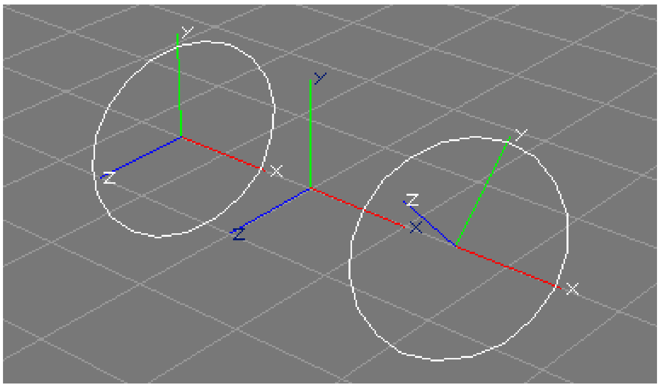



Figure 86: To XY Tool (left: original curve, right: modified curve)

- Arguments: The To XY tool takes a number of NURBS curves from the selection. The NURBS curves should be planar.
- Operation: The control points of the curve(s) will be rotated, so that they are in the XY plane of the respective object space defined by the NURBS curve object(s). Additionally, the rotation attributes of the NURBS curve object(s) will be changed so that the curve does not change its orientation with regard to other objects or the world space. 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).
- Note: 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 6.2.11 toXYNC (page 253).

## 5.29 Make Compatible Tool

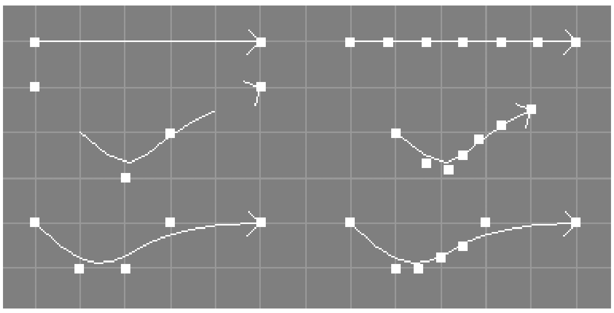


Figure 87: Make Compatible Tool (left: Original Curves, right: Compatible Curves)

- Arguments: The Make Compatible tool takes a number of NURBS curves from the selection.
- Operation: The curves will be made compatible, so that they are of the same order and defined on the same knot vector.
- Note: 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 point selection will be removed from the original objects. See also the documentation of the corresponding scripting interface command 6.2.11 makeCompNC (page 253).

## 5.30 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.
- Note: 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 6.2.11 rescaleknNC (page 252).

## 5.31 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).
- Note: 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 6.2.11 rescaleknNC (page 252).

## 5.32 Collapse Points Tool

- Arguments: The collapse tool expects a selected NURBS curve or NURBS patch and a number of selected (tagged) control points (see section 3.9 Selecting Points (page 62) 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 last tagged point.

## 5.33 Explode Points Tool

- Arguments: The explode tool expects a selected NURBS curve or NURBS patch and a number of selected (tagged) multiple points (see section 3.9 Selecting Points (page 62) for information on how to select (tag) control points).

- Operation: The points forming the selected multiple points will be made to simple points again and may be edited separately.
- Note: Even though you might have exploded some multiple points Ayam will re-create them on several occasions like reading of a scene, inserting/deleting points, and applying the NCurveAttr or NPatchAttr property if all single control points of the multiple point(s) still have identical coordinate values (and the `"CreateMP"` option of the curve or patch object is activated). You should immediately edit the control points (move them apart) after exploding to avoid that they automatically collapse to a multiple point again.

## 5.34 Swap UV Tool

- Arguments: The swap uv tool takes a number of NURBS patches, BPatch, or PatchMesh objects from the selection.
- Operation: The U and V dimension of the selected objects will be swapped (width and height will be exchanged) without altering the shape of the patches. See also the documentation of the corresponding scripting interface command 6.2.11 swapuvS (page 252).

## 5.35 Elevate UV Tool

- Arguments: The elevate uv 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.
- Note: 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"`. 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 original object. See also the documentation of the corresponding scripting interface commands 6.2.11 elevateuNP (page 256), and 6.2.11 elevatevNP (page 256).

## 5.36 Refine Surface Tool

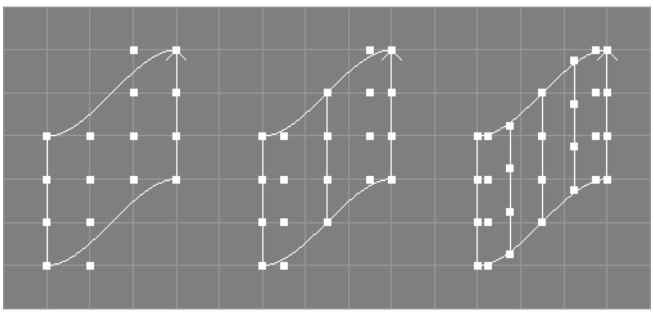


Figure 88: Successive Application of Refine Surface (U) Tool

- Arguments: The refine surface tool takes a number of NURBS patches from the selection.

- Operation: The respective knot vectors of the selected NURBS patches will be refined 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).
- Note: The respective knot type of the patch may be changed to `"Custom"`. The point selection will be removed from the original object. See also the documentation of the corresponding scripting interface commands 6.2.11 refineuNP (page 256), and 6.2.11 refinevNP (page 256).

## 5.37 Revert U Tool

- Arguments: The revert u tool takes a number of NURBS patches or 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.
- Note: See also the documentation of the corresponding scripting interface command 6.2.11 revertuS (page 252).

## 5.38 Revert V Tool

- Arguments: The revert v tool takes a number of NURBS patches or 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.
- Note: See also the documentation of the corresponding scripting interface command 6.2.11 revertvS (page 252).

## 5.39 Patch Clamp Tool

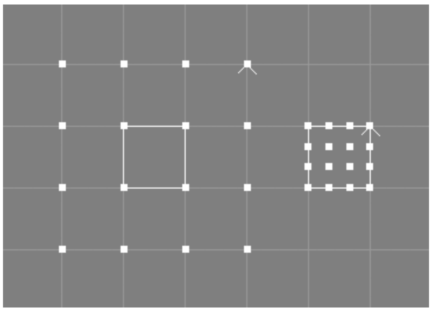


Figure 89: Patch Clamp Tool (left: Original Patch with B-Spline Knot Vectors, right: Clamped Patch)

- Arguments: The patch clamp tool takes a number of NURBS patches from the selection.
- Operation: The knot vectors of a selected NURBS patch 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).

- Note: The shape of the patch will not change but the position of some control points will. The patch interpolates the first and the last control point afterwards. The knot types of the patch will be changed to type `"Custom"`. The point selection will be removed from the original object. 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 6.2.11 clampuNP (page 254), and 6.2.11 clampvNP (page 254).

## 5.40 Patch Rescale Knots to Range Tool

- Arguments: The patch rescale knots to range 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.
- Note: 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 6.2.11 rescaleknNP (page 254).

## 5.41 Patch Rescale Knots to Mindist Tool

- Arguments: The rescale knots to mindist 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.
- Note: 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 6.2.11 rescaleknNP (page 254).

## 5.42 Patch Insert Knot Tool

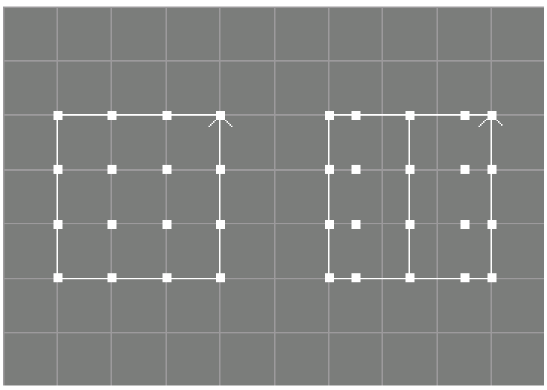


Figure 90: Patch Insert Knot Tool (left: original patch, right: a knot has been inserted 1 time at t=0.5)

- Arguments: The insert knot 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: The specified knot (t) will be inserted i times into the knot vector of the selected patches, without changing the shape of the patches.
- Note: 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 object. See also the documentation of the corresponding scripting interface commands 6.2.11 insknuNP (page 255), and 6.2.11 insknvNP (page 255).

## 5.43 Patch Remove Knot Tool

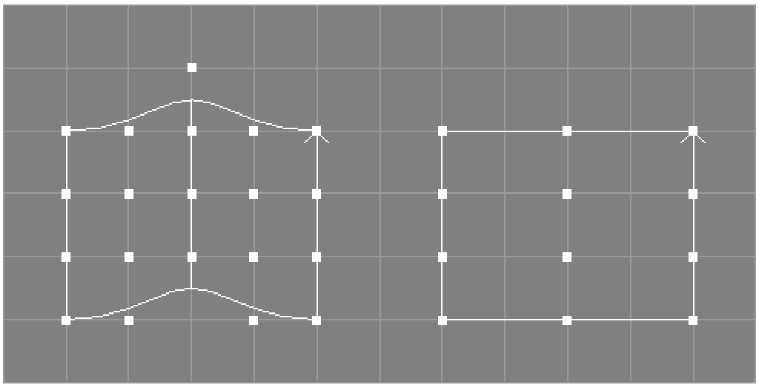


Figure 91: Remove Knot Tool (left: original surface, right: the knot at t=0.5 has been removed 1 time)

- Arguments: The remove knot tool takes a number of NURBS surfaces from the selection and requests three additional values, a parametric value t, an integer value i, and a tolerance tol.
- Operation: The specified knot (t) will be removed i 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 (by entering -i index instead of a parametric value t, i.e. `"-i 4"` instead of `"0.5"`). If the knot can not be removed r times due to the tolerance, an error is reported and the original surface is left unchanged.
- Note: If tol is `"Inf"` (infinity) the tool tries to work without changing the shape of the surfaces, however, this is not guaranteed. The point selection will be removed from the original objects. See also the documentation of the corresponding scripting interface commands 6.2.11 remknuNP (page 255), and 6.2.11 remknvNP (page 255).

## 5.44 Patch Split Tools

- Arguments: The patch split tools take a number of NURBS patches from the selection and request a parametric value t (in U or V parametric dimension, respectively).
- Operation: The patches will be split at the parametric value t into two patches (in U or V parametric dimension, respectively) using knot insertion.

```
|-NPatch      ==>      |-NPatch
                       |-NPatch
```

- Note: The original patch will be modified and a new patch will be created. The point selection will be removed from the original object. Trim curves will not be honored properly. See also the documentation of the corresponding scripting interface commands 6.2.11 splituNP (page 256), and 6.2.11 splitvNP (page 257).

## 5.45 Extract Curve Tool

- Arguments: The extract curve tool takes the first of the selected objects from the selection.
- Operation: The tool creates an instance the first of the selected objects then creates an ExtrNC object and moves the instance to it.

```
|-NPatch               |-NPatch
              ==>      +-ExtrNC
                        \-Instance_of_NPatch(Instance)
```

- Note: This tool uses the object clipboard to move the objects around so that the original clipboard contents are lost when this tool finishes. Eventually present trim curves will *not* be honored properly. See section 4.27 ExtrNC Object (page 123) for more information regarding the ExtrNC object.

## 5.46 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)
```

- Note: This tool uses the object clipboard to move the objects around so that the original clipboard contents are lost when this tool finishes. Eventually present trim curves will *not* be honored properly. See section 4.46 ExtrNP object (page 169) for more information regarding the ExtrNP object. See also the documentation of the corresponding scripting interface command 6.2.11 extrNP (page 257).

## 5.47 Split to Curves Tool

- Arguments: The split to curves tool takes a single NURBS patch from the selection.
- Operation: The selected NURBS patch will be split into NURBS curves, along direction U or V.

```
|-NPatch               |-NPatch
                       |-NCurve
              ==>      |-NCurve
                       |-NCurve
                       |-NCurve
```

- Note: The original NURBS patch object is not deleted. See also the documentation of the corresponding scripting interface command 6.2.11 splitNP (page 258).

## 5.48 Build from Curves Tool

- Arguments: The build from curves tool takes a number of NURBS curves from the selection.
- 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: length of the first selected curve, Height: number of used curves. The order 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.

```
|-NCurve                |-NCurve
|-NCurve                |-NCurve
|-NCurve      ==>       |-NCurve
|-NCurve                |-NCurve
                        |-NPatch
```

- Note: The original NURBS curves objects are not deleted. See also the documentation of the corresponding scripting interface command 6.2.11 buildNP (page 258).

## 5.49 Tesselation Tool

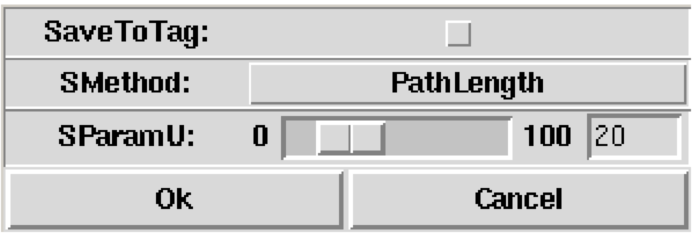



Figure 92: GUI of Tesselation Tool

- Arguments: The tesselation 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 tesselation method via a drop-down menu and to tune the corresponding tesselation 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 tesselated with the chosen method and parameters. The PolyMesh objects created by the tesselation will immediately be displayed in all view windows instead of the original objects. Whenever tesselation method or parameters are changed, the tesselation will be recomputed and displayed, thus, allowing an immediate estimation of the tesselation quality.

  If the preference option `"Modelling/LazyNotify"` is enabled, updates of the tesselation that normally occur while dragging the slider(s) will be deferred until the mouse button is released.

  If the `"Ok"` button is pressed to close the tesselation tool, all selected objects will be replaced by their tesselated counterparts; if `"Cancel"` is used, all selected objects remain unchanged.

If the check box "SaveToTag" is activated, closing the tesselation tool using "Cancel" will also add a "TP" tag containing the currently selected method and parameter value to all selected objects. This tag can be evaluated later, when the respective objects are converted to PolyMesh objects. Since Ayam 1.11 also the conversion mechanism of objects keeps the "TP" tags intact, so that one may e.g. save tesselation parameters for a Sphere or a Revolve object. Upon conversion of the Sphere or Revolve object via a NURBS patch to a PolyMesh, the tesselation parameters will be retained. Mind that the "SaveToTag" option will be activated automatically if any of the objects to tesselate already have a "TP" tag.

- Note: The tesselation tool will block most other parts of Ayam while it is running. It is, however, possible to adjust view parameters while the tesselation tool is open, to examine the tesselation result more closely or from different viewing angles. The initial values of the parameter slider bounds may be changed by simply entering values that are out of bounds into the respective entry widget, then pressing the <Tab> key. The resolution of the slider is calculated automatically from the resolution of the value entered in the entry widget.

See section 2.9.5 Miscellaneous Preferences (page 52) for a more extensive discussion of the tesselation methods and their parameter(s).

# 6 Scripting Interface

The Ayam scripting interface is mainly a number of Tcl procedures and Tcl commands that are also used internally by the application, e.g. the main menu entry "File/New" calls the scripting interface command "newScene" (among other commands). Using the scripting interface means to call these commands on your own possibly in a mix with standard Tcl script code.

Furthermore, using Tcl and its introspection facilities, you could easily modify the code Ayam consists of. This is, however, not recommended for good reasons (unless you read the Ayam source code and really know, what you are doing). 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* "sl" and *not* "SL".

The scripting interface may be used directly from the console of Ayam. You can, of course, also write scripts in your own Tcl script files, that may be loaded at any time into Ayam using the console and the Tcl command "source". You can also arrange for a script file to be executed automatically on every application startup 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 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 4.50 Script object (page 176). Even normal objects can trigger scripts upon notification using BNS or ANS tags. See also sections 4.55.14 Before Notify Script (page 199) and 4.55.15 After Notify Script (page 199).

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. If you want your changes to become visible you have to update the various parts of the GUI (property GUIs, view windows) explicitly (see also section 6.2.13 Updating the GUI (page 260)). However, since Ayam 1.13 it is also possible to automatically run GUI updating commands in the console by using <Shift+Return> instead of <Return> when issuing scripting interface commands.

If you want your changes to be recorded in the undo buffer, you have to arrange for this manually too (see the documentation of the undo command: 6.2.20 Undo (page 265)).

From scripts it may be necessary to check whether an error occured 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 occured. You need to set ay_error to zero before and check it after the operation in question to see whether the operation performed successfully:

```
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 "ay_error" variable holds the current error state. See also section 6.2.19 Reporting Errors (page 264).
- The variable "i" is used by all "forAll" command variants. See also section 6.2.16 Applying Commands to a Number of Objects (page 262).

### 6.1.2 The Global Array ay

The global array "ay" holds application state variables. Furthermore, you can find the paths to important widgets (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 311). 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 to be transferred to the C context using the "setPrefs" command. See also section 6.2.14 Managing Preferences (page 261).

### 6.1.4 The Global Property Management and Data Arrays

For every property, a corresponding global arrays 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"). The entries "sproc" and "gproc" designate the set-property and get-property callbacks (procedures or commands) respectively. If sproc or gproc are empty strings (""), standard callbacks named "setProp" or "getProp" should be used to get or set the property values. But for the transformations property, the "setTrafo" and "getTrafo" commands should be used. The last entry, "w", is the name of the main property GUI window. To get the full widget path of this window, the current value of ay(pca) needs to be prepended.

Note that the global property data array only holds useful data when the respective property GUI is active, or when it has been filled explicitly by the so called get-property callback.

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 | getTagsp | setTagsp |

Table 79: Property Arrays and Callbacks

Note that this list is pretty much incomplete, however you can always infer such information using commands like "parray Tags" in the Ayam console.

See also section 6.2.5 Manipulating Properties (page 244) for more information on how to edit property values from the scripting interface.

Since Ayam 1.16, the global property management array may be created easily using the new scripting interface command "addPropertyGUI".

## 6.2 Index of Procedures and Commands

This section provides documentation on the most important scripting interface commands and procedures of Ayam sorted by category.

All 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),
- **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
- Description: Fire up a web browser and display the help for the designated Ayam scripting interface command.
- Example: "`help help`" displays the help of the help command.

### 6.2.2 Creating 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 curve length by 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.

- "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 by 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.

- "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 control points to use for the approximating curve, the alength defaults to 3.
  * "-order": order of the new curve, the order defaults to 3. If a value greater than the length is specified, the order will be made identical to the length value.
  * "-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 by 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 6 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 5 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.

– "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 "-vkv" option below, the knot type will always be set to 3 – Custom.
  * "-vkv": 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: "-vkv {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 "-vkv" 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 by height by 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.
* "-createmp": The "-createmp" 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.

– "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 by 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 by 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_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 by 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 by height 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 width by height by 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.

– "PolyMesh": Polymeshes 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 go unnoticed in Ayam and will only be detected later in other applications.

- "`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 atleast 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 {2} -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 go unnoticed in Ayam and will only be detected later in other applications.

- "`Level`": Levels must be given an additional argument determining the type of the new level, this argument may be one of: "`0`" (level), "`1`" (union), "`2`" (intersection), "`3`" (difference), or "`4`" (primitive).

  Examples:

  1. **"crtOb Level 0"**
     creates a simple level object.
  2. **"crtOb Level 3"**
     creates a CSG difference level object.
- "`Material`": Materials must be given an additional argument giving the name of the new material.

  Example:

  1. **"crtOb Material Wood"**
     creates a material named Wood.
- "`Instance`": creates an instance of the selected object.
- ...

- Example: Create a sphere and update the GUI: "`crtOb Sphere; uS; rV`".

There are some helper commands, that 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.

crtClosedBS – create closed (circular) B-Spline:

- Synopsis: "`crtClosedBS sections [order [arc [radius]]]`"
- Background: Yes, Undo: No, Safe: Yes
- Description: This command creates a circular B-Spline curve with desired number of sections, 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.

crtNRect – create a rectangular NURBS curve:

- Synopsis: "`crtNRect`"
- Background: Yes, Undo: No, Safe: Yes
- Description: This command creates a rectangular NURBS curve that fits the parameter space of the currently selected NURBS patch object (or the current parent object, if it is a NPatch). If no NURBS patch is selected or parent, this command will create a centered quadratic curve with width 2.0 and height 2.0 instead.

### 6.2.3 Manipulating the Selection

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 [index]`"
- Background: Yes, Undo: No, Safe: Yes
- Description: Use this command to set or clear the current selection, index may be an ordered list of indices, a single index or empty. If no index is given, the current selection will be cleared.
- Examples: "`selOb`" clears the current selection, "`selOb 0`" selects the first object in the current level, "`selOb 0 1`" selects the first two objects in the current level.

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 index) from a multiple selection without changing the selection state of any objects.
- Example: "`withOb 2 {movOb 0 1 0}`" moves the third object from multiple selected objects. All objects stay selected.

sL – select last object:

- Synopsis: "`sL`"
- 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 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
```

---

hSL – hidden select last object:

- Synopsis: "`hSL`"
- Background: Yes, Undo: No, Safe: No
- Description: Select the last object in the current level but do not update the GUI. 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.

### 6.2.4 Selecting Points

This command manipulates the point selection.

selPnts – select points:

- Synopsis: "`selPnts [-all | index1 index2 ...]`"
- Background: Yes, Undo: No, Safe: Yes
- Description:
  - If called without arguments, this command deselects all points.
  - If the argument is "`-all`", this command 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.
- Example: Given that a single NCurve object is selected, its points may be selected using the command "`selPnts -all`"; the first and third point of the same curve may be selected using "`selPnts 0 2`".

### 6.2.5 Manipulating Properties

Since Ayam 1.9 two new commands are implemented that ease the manipulation of property values from the scripting interface:

getProperty – get single property value

- Synopsis: "`getProperty propname(elemname) varname`"
- Background: Yes, Undo: No, Safe: Yes
- Description: This command 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.
- Notes: In contrast to many other helper commands, the variable varname does not have to be global. This command runs more slowly than calling the appropriate get-procedure and accessing the data array that is associated with a property directly, especially if multiple values are to be fetched. 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 6.1.4 Global Property Management and Data Arrays (page 223).

setProperty – set single property value

- Synopsis: "`setProperty propname(elemname) value`"
- Background: Yes, Undo: Yes, Safe: Yes
- Description: This command sets a single property element named elemname of the property named propname for the currently selected object to the new value given in value.
- Notes: This command 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. 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
  ```

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 6.1.4 Global Property Management and Data Arrays (page 223).

### 6.2.6 Clipboard Operations

These commands operate the object clipboard:

copOb – copy object:

- Synopsis: "`copOb`"
- Background: Yes, Undo: No, Safe: Yes
- Description: Copy the selected object(s) to the object clipboard.

cutOb – cut object:

- Synopsis: "`cutOb`"

- Background: Yes, Undo: No, Safe: Yes
- Description: Move the selected object(s) into the object clipboard.

pasOb – paste object:

- Synopsis: "`pasOb`"
- Background: Yes, Undo: No, Safe: Yes
- Description: Copy the selected object(s) from the object clipboard to the current level.

delOb – delete object:

- Synopsis: "`delOb`"
- Background: Yes, Undo: No, Safe: Yes
- Description: Delete the selected object(s) from the scene.

pasmovOb – paste (move) object:

- Synopsis: "`pasmovOb`"
- Background: Yes, Undo: No, Safe: Yes
- Description: Move the objects from the object clipboard to the current level.

The following commands operate the property clipboard, which is totally independent from the object clipboard.

pclip_copy/copyProp – copy a property to the property clipboard

- Synopsis: "`pclip_copy mode`" or "`copyProp mode`"
- Background: Yes, Undo: No, Safe: No
- Description: Copy the currently selected property from the currently selected object to the property clipboard. If mode is 0, omit all marked entries, if mode is 1 copy just marked entries. Entries can also be marked programmatically by adding the respective entry names to the global array "`pclip_omit`". Note that you may call this procedure also using the shortcut "`copyProp`".

pclip_paste/pasteProp – paste a property

- Synopsis: "`pclip_paste`" or "`pasteProp`"
- Background: Yes, Undo: Yes, Safe: No
- Description: Copy the property from the property clipboard to the currently selected object. Note that you may call this procedure also using the shortcut "`pasteProp`".

### 6.2.7 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 other object (not the root) the parent level will be entered instead. If index is -1, the last object of the current level will be entered.

goUp:

- Synopsis: "`goUp`"
- Background: Yes, Undo: No, Safe: Yes
- Description: Go one level up in the object hierarchy.

goTop:

- Synopsis: "`goTop`"
- Background: Yes, Undo: No, Safe: Yes
- Description: Go to the top level of the object hierarchy.

### 6.2.8 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`"
- 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 then by dz degrees around the objects Z axis. Note the order of the rotations.

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.

rotPnts – rotate selected points:

- Synopsis: "`rotPnts dx dy dz`"
- 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 then by dz degrees around the objects Z axis. Note the order of the rotations.

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.

applyTrafo – apply transformations:

- Synopsis: "`applyTrafo sel | all`"
- Background: Yes, Undo: Yes, Safe: Yes
- Description: applies the transformations encoded in the transformation attributes of the selected objects to the points (either all points, or just the selected ones if there are any) of those objects. Additionally, the transformations of the selected objects will be reset to the default values.

### 6.2.9 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.

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 will be written to an array pointed to by varname.

### 6.2.10 Manipulating Tags

These commands may be used to modify the tags of an object:

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(s). It is legal to deliver "" as value parameter. This is e.g. needed for the "`NoExport`" tag.
- 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.

delTags:

- Synopsis: "`delTags type`"
- Background: Yes, Undo: Yes, Safe: Yes
- Description: Delete all tags of designated type from the currently selected objects(s). If type is "`all`", all tags are deleted from the currently selected objects(s).
- Examples:

1. **"delTags all"**
   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 objects and put them as lists into two variables named tvname for the tag types and vvname for the tag values.

setTags:

- Synopsis: "setTags tags"
- Background: Yes, Undo: Yes, Safe: Yes
- Description: Clear all tags from the currently selected object and set new tags. 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.

#### 6.2.11 Manipulating NURBS Curves and Surfaces

These are more specialized commands to change NURBS curve and surface properties:

clampNC – clamp NURBS curve:

- Synopsis: "clampNC [side]"
- 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 or 0, both sides are clamped. If the side parameter is 1 only the start, and if it is 2 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 5.23 Clamp Tool (page 210).

elevateNC – elevate NURBS curve:

- Synopsis: "elevateNC n"

- Background: Yes, Undo: Yes, Safe: Yes
- Description: Elevate the order of the selected NURBS curves without changing the shape of the curves by n. The knot type of the elevated curves will be changed to "`Custom`". See also section 5.19 Elevate Tool (page 208).

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 (u must be in the valid range of the knot vector of the selected curves) r times. The valid range is determined by the current knot vector U as follows: `U[p] <= u <= 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 5.24 Insert Knot Tool (page 211).

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 4 1`" instead of "`remknNC 0.5 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. See also section 5.25 Remove Knot Tool (page 211).

refineNC – refine NURBS curve:

- Synopsis: "`refineNC [-cv | {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} or refine the control vector (if "`-cv`" option is given). If no list of new knots is given a new knot is inserted into each interval in the old knot vector. See also section 5.20 Refine Tool (page 208).

coarsenNC – coarsen NURBS curve:

- Synopsis: "`coarsenNC`"
- Background: Yes, Undo: Yes, Safe: Yes
- Description: Remove every second control point from the selected NURBS curves. See also section 5.22 Coarsen Tool (page 210).

revertC – revert curves:

- Synopsis: `"revertC"`
- Background: Yes, Undo: Yes, Safe: Yes
- Description: Revert the direction of the selected NURBS, interpolating, and approximating curves. See also section 5.15 Revert Tool (page 206).

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 5.37 Revert U Tool (page 216).

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 5.38 Revert V Tool (page 216).

swapuvS – swap dimensions of surfaces:

- Synopsis: `"swapuvS"`
- Background: Yes, Undo: Yes, Safe: Yes
- Description: Swap the dimensions of the selected surfaces, thus exchanging width and height (without altering the shape of the surfaces). See also section 5.34 Swap UV Tool (page 215).

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 5.30 Rescale Knots to Range Tool (page 214).

splitNC – split NURBS curve

- Synopsis: `"splitNC u"`

- Background: Yes, Undo: No, Safe: Yes
- Description: splits the selected NURBS curve at designated parametric value into two curves, creating one new curve and *modifying the original* selected curve. See also section 5.17 Split Tool (page 207).

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 5.27 Shift Closed Curve Tool (page 212).

toXYNC – move NURBS curve to XY plane

- Synopsis: `"toXYNC"`
- Background: Yes, Undo: Yes, Safe: Yes
- Description: moves the control points of the selected NURBS curve objects to its XY plane. See also section 5.28 To XY Tool (page 213).

trimNC – trim NURBS curve

- Synopsis: `"trimNC 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. See also section 5.18 Trim Tool (page 207).

estlenNC – estimate length of NURBS curve:

- Synopsis: `"estlenNC varname"`
- Background: Yes, Undo: No, Safe: Yes
- Description: estimate the length of the currently selected curve and put the result into the designated variable.

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"`.

makeCompNC – make NURBS curves compatible

- Synopsis: "`makeCompNC`"
- Background: Yes, Undo: No, Safe: Yes
- Description: makes the selected NURBS curves compatible i.e. of the same order and defined on the same knot vector. See also section 5.29 Make Compatible Tool (page 213).

clampuNP – clamp NURBS patch in U direction:

- Synopsis: "`clampuNP [side]`"
- 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 or 0, both sides are clamped. If the side parameter is 1 only the start, and if it is 2 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 5.39 Patch Clamp Tool (page 216).

clampvNP – clamp NURBS patch in V direction:

- Synopsis: "`clampvNP [side]`"
- 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 or 0, both sides are clamped. If the side parameter is 1 only the start, and if it is 2 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 5.39 Patch Clamp Tool (page 216).

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. The "`-ru`", "`-rv`", "`-du`", and "`-dv`" variants scale only the designated dimension. Scaling to a minimum distance ensures that all knots (except for multiple knots) have a distance bigger than mindist afterwards. 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 5.40 Patch Rescale Knots to Range Tool (page 217) and 5.41 Patch Rescale Knots to Mindist Tool (page 217).

- 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).

insknuNP – insert knot into NURBS patch:

- Synopsis: "`insknuNP 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] <= u <= 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 5.42 Patch Insert Knot Tool (page 217).

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] <= v <= 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 5.42 Patch Insert Knot Tool (page 217).

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 4 1`" instead of "`remknuNP 0.5 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. See also section 5.43 Patch Remove Knot Tool (page 218).

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 4 1`" instead of "`remknvNP 0.5 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. See also section 5.43 Patch Remove Knot Tool (page 218).

refineuNP – refine NURBS surface in u direction:

- Synopsis: "`refineuNP`"
- Background: Yes, Undo: Yes, Safe: Yes
- Description: Refine the selected NURBS surfaces in u direction without changing their shape. The u knot type of the refined surfaces may be changed to "`Custom`". See also section 5.36 Refine Surface Tool (page 215).

refinevNP – refine NURBS surface in v direction:

- Synopsis: "`refinevNP`"
- Background: Yes, Undo: Yes, Safe: Yes
- Description: Refine the selected NURBS surfaces in v direction without changing their shape. The v knot type of the refined surfaces may be changed to "`Custom`". See also section 5.36 Refine Surface Tool (page 215).

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. The u knot type of the elevated surfaces will be changed to "`Custom`". See also section 5.35 Elevate UV Tool (page 215).

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. The v knot type of the elevated surfaces will be changed to "`Custom`". See also section 5.35 Elevate UV Tool (page 215).

splituNP – split NURBS patch:

- Synopsis: "`splituNP u`"
- Background: Yes, Undo: No, Safe: Yes

- Description: Splits the selected NPatch objects into two patches at parametric value u. The selected NPatch will be modified and a new NPatch object will be created and appended as new object to the current level of the scene. See also section 5.44 Patch Split Tools (page 218).

splitvNP – split NURBS patch:

- Synopsis: `"splitvNP v"`
- Background: Yes, Undo: No, Safe: Yes
- Description: Splits the selected NPatch objects into two patches at parametric value v. The selected NPatch will be modified and a new NPatch object will be created and appended as new object to the current level of the scene. See also section 5.44 Patch Split Tools (page 218).

extrNP – extract NURBS patch:

- Synopsis: `"extrNP umin umax vmin vmax"`
- Background: Yes, Undo: No, Safe: Yes
- Description: Extracts a sub-patch from the selected NPatch objects. The extracted patch will be appended as new object 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. See also section 5.46 Extract Patch Tool (page 219).

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 but this will also increase the width of the resulting NURBS surface (the default is to create an open surface). 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 is 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 interpolate the old control point positions. The u knot type of the interpolated surfaces will be changed to `"Custom"`.

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 but this will also increase the height of the resulting NURBS surface (the default is to create an open surface). 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 is 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 interpolate the old control point positions. The v knot type of the interpolated surfaces will be changed to `"Custom"`.

concatS – concatenate surfaces:

- Synopsis: `"concatS [-o order | -t type | -k knottype | -u uvselect]"`
- Background: Yes, Undo: No, Safe: No
- 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), and `"-k"` allows to set a knot type (default 1 - NURB). Finally, the `"-u"` option allows to specify the uv-select-string. See also section 4.45.1 ConcatNPAttr Property (page 166) for more information on these options.

splitNP – split NURBS patch:

- Synopsis: `"splitNP (u | v)"`
- Background: Yes, Undo: No, Safe: No
- Description: Splits the selected NPatch objects into NURBS curves, along parametric dimension u or v. See also section 5.44 Patch Split Tools (page 218).

buildNP – build NURBS patch:

- Synopsis: `"buildNPatch"`
- Background: Yes, Undo: No, Safe: No
- Description: Builds a NURBS patch from the selected NURBS curves. See also section 5.48 Build from Curves Tool (page 220).

#### 6.2.12 Manipulating Points

Use these two commands to read or manipulate single points of arbitrary objects. Note that the exact arguments needed depend on the type of the selected object, e.g. manipulating the points of a NURBS curve requires just one index parameter (indexu), whereas manipulating the points of a NURBS patch requires two index parameters (indexu and indexv) to be specified.

getPnt – get point(s):

- Synopsis: "`getPnt [-trafo | -world | -eval] (index | indexu indexv | u | u v (varx vary varz [varw] | -vn varname) | -all 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 varw.

  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 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`".
- 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 w`".
  2. **"`getPnt -eval 0.5 x y z w`"**
     gets the curve point at parametric value "`0.5`" and writes the values to the variables "`x y z w`".

setPnt – set point(s):

- Synopsis: "`setPnt [-world] (indexu [indexv] (x y z [w] | -vn varname) | -all 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.

  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.

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`".

### 6.2.13 Updating the GUI

These commands update various parts of the Ayam user interface:

rV – redraw all views:

- Synopsis: "`rV`"
- Background: No, Undo: No, Safe: No
- 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
- 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
- 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
- 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.

### 6.2.14 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 ayprefs Tcl array to let the changes take effect.

### 6.2.15 Custom Objects

This command manages custom objects (plugins):

io_lc – load custom:

- Synopsis: "`io_lc filename`"
- Background: No, Undo: No, Safe: No
- Description: Load the custom object (plugin) from file filename. Note that it is currently not possible to unload a custom object from Ayam.

### 6.2.16 Applying Commands to a Number of Objects

There are two commands that help to apply arbitrary commands to a number of objects, forAll and forAllT:

forAll:

- Synopsis: "`forAll recursive command`"
- Background: depends, Undo: depends, Safe: No
- Description: The forAll command executes command for all objects that have been selected currently, or for every object of the current level if nothing has been selected. If recursive is 1 then forAll will recurse into every object (if it has child objects) before the execution of command. Note that forAll will run slowly if a property GUI is displayed. You can make it run faster by de-selecting the property using e.g. the property context menu first.
- 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 what you want. Use "`forAllT`" in this case.
  - The command will not have access to global arrays unless e.g. one of the following constructs is in use:
    "`forAll 0 { uplevel #0 { commands } }`"
    "`forAll 0 { 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. Since Ayam 1.9 "`forAll`" correctly maintains the current selection.

forAllT:

- Synopsis: "`forAllT type recursive command`"

- Background: depends, Undo: depends, Safe: No
- Description: forAllT works the same way as forAll, with an additional type check. The command will not be executed if the type of the current object does not match the argument type. Note that forAllT will run slowly if a property GUI is displayed. You can make it run faster by de-selecting the property using e.g. the property context menu first.

  Note that the type strings will be converted to lowercase before comparison, so that it is legal to use forAllT e.g. this way:

  "`forAllT ncurve 0 {puts $i}`"
- Deficiencies:
  - The command will not have access to global arrays unless e.g. one of the following constructs is in use:"`forAllT ncurve 0 {uplevel #0 {commands} }`"
    "`forAllT ncurve 0 { 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. Since Ayam 1.9 "`forAllT`" correctly maintains the current selection.

#### 6.2.17 Scene IO

These commands help to load scenes from and save them to Ayam scene files:

newScene:

- Synopsis: "`newScene`"
- Background: Yes, Undo: No, Safe: No
- Description: clears the current scene.

replaceScene:

- Synopsis: "`replaceScene filename`"
- Background: Yes, Undo: No, Safe: No
- Description: clears the current scene, then loads a new scene from filename.

insertScene:

- Synopsis: "`insertScene filename`"
- Background: Yes, Undo: No, Safe: No
- Description: inserts a scene from filename.

saveScene:

- Synopsis: "`saveScene filename`"
- Background: Yes, Undo: No, Safe: No
- Description: saves the current scene to filename.

### 6.2.18 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.

  If the argument "`-smonly`" is provided, a RIB to render shadow maps will be created and the argument of "`-image`" will be ignored.

  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. 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.

### 6.2.19 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. You should always use ayError instead of puts because the error reporting mechanism of Ayam features consistently formatted output, compression of repeated messages, and logging. Code 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 not needed in the Tcl script context. Place should describe the procedure where the error occured. Detail is the string to be output.
- Notes: The actual output in the Ayam console depends on the preference option "`ErrorLevel`" see section 2.9.5 Miscellaneous Preferences (page 52).

### 6.2.20 Miscellaneous

Miscellaneous commands:

getType:

- Synopsis: "`getType varname`"
- Background: Yes, Undo: No, Safe: Yes
- Description: This command writes the type of the selected object into the variable varname. 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.).

tmpGet:

- Synopsis: "`tmpGet tmpdir varname`"
- Background: Yes, Undo: No, Safe: No
- Description: This command calculates a name for a temporary file in tmpdir and puts the complete name into varname.

hasChild:

- Synopsis: "`hasChild`"
- Background: Yes, Undo: No, Safe: Yes
- Description: This command returns 1 if the selected object has child objects, otherwise it returns 0.

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 306).

convOb:

- Synopsis: "`convOb [-inplace]`"
- Background: Yes, Undo: depends (if -inplace: Yes), Safe: Yes
- Description: This command calls the registered converter for the selected object(s). If the option "`-inplace`" is used, the new object(s) will replace the old object(s).

notifyOb:

- Synopsis: "`notifyOb [-all | -modified | -parent]`"
- Background: Yes, Undo: No, Safe: No
- Description: This command calls the registered notification callback for the selected object(s) and their parents, or, if no object is selected, for all objects of the scene.

  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.

addToProc:

- Synopsis: "`addToProc procedure addition`"
- Background: Yes, Undo: No, Safe: No
- Description: This command adds the code from addition to the procedure procedure.
- Note: This command 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.

One can e.g. 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]
```

.

Repeated calling of the tool without opening the dialog (using the keyboard shortcut <Ctrl+T>), will execute the provided expression again. This means, one can e.g. create a number of curves with increasing length 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.

You may copy and paste all examples directly from the documentation 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 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" -command {
 runTool {x y z} {"X:" "Y:" "Z:"} "forAll 0 {placeOb %0 %1 %2}"
 plb_update; rV
}
```

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 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 "pasmovOb" and _not_ "pasOb", because we
 # really want to move (and not copy) the curve object
 pasmovOb

 # go up to where we came from
 goUp

 # finally, update the GUI...
 uS
 sL

 # ...and redraw all views
 rV
}
# easySweep
```

Run this pocedure by selecting a NURBS curve object, then type into the console:

```
easySweep
```

You may add this command to the main menu as well:

```
global ay
$ay(cm) add command -label "Easy Sweep" -command {
 easySweep
}
```

After running the above script you should have 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 you may add buttons 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

if { $ayprefs(SingleWindow) } {
  set b .fv.fTools.f.mybutton
} else {
  set b .tbw.f.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, it will be under the
  # automatic toolbox layout management
}
```

This example shows that a) toolbox buttons have to be created in the frame "`.tbw.f`" for multi-window GUI configurations or "`.fv.fTools.f`" for single-window GUI configurations, b) Ayam manages a list of all toolbox buttons in the global array ay in "`ay(toolbuttons)`", the order in that list is the order in which the buttons appear in the toolbox, c) 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 the new 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.

Here is another example that 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):

```
# create a frame:
set f [frame .tbw.f.fcollex]

# calculate the row number below the last row:
set row [expr [lindex [grid size .tbw.f] 1] + 1]

# now display the frame at calculated row, spanning the whole window:
grid $f -row $row -column 0 -columnspan [lindex [grid size .tbw.f] 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 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, or the "`Scripts`" preference setting of Ayam on each start (the latter except for repairAyam.tcl and bgconvert.tcl).

### 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 one has to start "`repairAyam.tcl`" from the Ayam console using the command:

```
source scripts/repairAyam.tcl
```

[1] Since 1.8.2.

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.[1]

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.

### 6.5.2 Convert Everything to Polygons

The script "topoly.tcl" recursively browses through the scene and converts everything to a polygonal representation.[2]

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 tesselation tool, see also section 5.49 Tesselation Tool (page 220).

### 6.5.3 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.[3]

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.

[1] Since 1.9. [2] Since 1.13. [3] Since 1.14.

### 6.5.4 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.

### 6.5.5 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): focussed sub-windows (views, console, object tree) will be more easily recognizable.

### 6.5.6 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 temporarily disable the modified behaviour, the <F11> key can be used.

To rotate or scale about a different point than the center, the mark may still be set manually using <a>.

### 6.5.7 Automatic Point Actions

The script "`apnt.tcl`" (for automatic point) may be used to switch the modelling mode of the current or all views 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.

It is still possible to switch back to object modelling anytime via the keyboard shortcut <`o`>.

### 6.5.8 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 successfull 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`").

### 6.5.9 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).

### 6.5.10 Switch File 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.

[1] Since 1.15.

### 6.5.11 Switch File 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.

### 6.5.12 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.

### 6.5.13 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.

### 6.5.14 Create Polyhedrons from Conway Notations

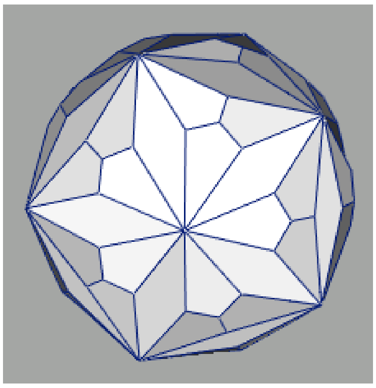


Figure 93: Polyhedron generated from Conway notation: jtD

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.

This script must be used in a Script object of type "`Create`". For convenience, there is also a property GUI; one must add a "`NP`" tag of value "`PolyhedronAttr`" to the script object to see it.

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 also section 8.15 References (page 325)).

**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: Pn, antiprisms: An, and pyramids: Yn, 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` / `t`*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` / `k`*n* | 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 80: 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., dpdX=pX and dpX=pdX, and p also commutes with a and j, i.e. paX=apX. |

Table 81: Additional Operations

## 6.6 JavaScript Scripting Interface

This sections 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.

### 6.6.1 Accessing JavaScript from Tcl and Script Objects

To access the JavaScript functionality from the Tcl scripting interface, the "`jsEval`" command can be used to directly execute JavaScript code (Tcl code in **bold**):

```
jsEval {var a = 0; a = a + 5.5; tclset("a", a);}
```

or from a file:

```
jsEval -f scriptfile.js
```

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");
```

it does not fail, when the calling context ist the main Tcl interpreter; one can e.g. type into the Ayam console:

```
jsEval {tcleval("exit");}
```

and Ayam quits (see also section: 4.50.1 Safe Interpreter (page 176)).

### 6.6.2 JavaScript Functions

This subsection informs about the global functions additionally available in the Ayam JavaScript interpreter.

Those are converted Tcl commands, "`tcleval`", "`tclvar`", and "`tclset`".

The functionality of Ayam is accessible from JavaScript via a larger set of global functions, named as the corresponding Tcl commands. One may e.g. create Ayam objects 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 converted function (refer to section 6.2 Index of Procedures and Commands (page 224) for a more or less complete list of those commands).

Note that many Tcl procedures are not available as global JavaScript function, but they can be called using "`tcleval`" as documented in the next paragraph.

**`tcleval`:**
Apart from Tcl commands converted to JavaScript functions, there is the global JavaScript function "`tcleval`", that allows to evaluate arbitrary Tcl scripts:

```
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.

**`tclvar`:**
Using the JavaScript function "`tclvar`" one may establish a link between a Tcl variable and a corresponding variable in the JavaScript context. The "`tclvar`" function establishes 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 onto the Tcl variable. 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).

Even though it looks a perfect fit, "`tclvar`" can not be used to manage a property data array. Atleast not 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: "`tclset("SphereAttrData(Radius)", 1.2);`".

### 6.6.3 Data Conversion

When data is transferred from the Tcl to the JavaScript side (e.g. while converting return values of "`tcleval`" or variable values linked via "`tclvar`"), 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 82: 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.

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);
```

| 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 83: JavaScript to Tcl Data Conversions

### 6.6.4 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(!tcleval("info exists MyPropData;"))
{
  /* initial script run (but not when loaded from scene file!) */
  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.MyItem = tcleval("set MyPropData(MyItem);");
}
if(!tcleval("info exists MyPropGUI;"))
{
  tcleval("set ::phw [addPropertyGUI MyProp \"\" \"\"];");
  tcleval("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 follows the creation of the property GUI which is also constrained to just one script run by an "`if`" statement.

Following the GUI, a sphere object is created and parameterized 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/loaddxfio.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 automatically load the matching plugin ("`dxfio`") and open the import (or export) options dialog with the "`FileName`" option already set to the filename chosen before. This feature 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 |

Table 84: 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 | Yes[c] |
| Parametrics | No[a] | No[a] | No[a] | No[b] | No[a] | 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 |

Table 85: Ayam Features Supported by Various Export Formats

[a] : will be converted to NURBS
[b] : will be converted to PolyMeshes
[c] : 3D trimcurves 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 quality of 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 as possible in the respective export format leading to the use of NURBS in Wavefront OBJ files. Consequently, to successfully transfer an Ayam scene to a different application, in some cases you may need to convert the NURBS objects in the Ayam scene to polygonal geometry before export. There is a script provided that helps you doing this (`"topoly.tcl"`, see also section 6.5.2 Convert Everything to Polygons (page 274)).

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 86: 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 **R**ead **RIB**) plugin you may import RenderMan Interface Bytestreams of version 3.0 into Ayam. Start importing a RIB using the menu entry `"File/Import/RenderMan RIB"` (if this menu entry is not available, you have to load the "rrib" plugin using the menu entry "File/Load Plugin" first).

### 7.3.1 RIB Primitive Support

The RRIB 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 archives[2] . Texture coordinates will import as TC tags (see also section 4.55.3 TC (Texture Coordinates) Tag (page 193)).

The RRIB 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 the knot ranges is also not supported (however, UMM/VMM tags will be created, that contain the new knot minimum and maximum values)[3]. See also section 4.55.16 UMM/VMM (U/V Min Max) Tag (page 200).

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 8.8 Automatic Instancing (page 319)).

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).

### 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.

[1] Since 1.7. [2] Since 1.9. [3] Since 1.9.

- "ReadFrame", specifies the number of the frame in the RIB to read. A value of -1 means, all frames are to be read. If you specify a frame number 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 RRIB 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) 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.

## 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). 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 2.9.4 RIB-Export preferences (page 50).

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 35).

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 38).

Invoking RIB export is also possible using the scripting interface, see the section 6.2.18 RIB export (page 264) 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 4.22.1 NCurveAttr (page 110), and 4.29.1 NPatchAttr (page 127) 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).

Start importing a DXF file using the main menu entry "File/Import/AutoCAD DXF" (if this menu entry is not available, you have to load the "dxfio" plugin 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 | Ayam 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 87: 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"`: By default, all entities from all layers will be imported. Using the `"ReadLayers"`: import option, a single layer or a range of layers may be selected for import.
- `"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, you have to load the "dxfio" plugin 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.

Ayam 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 tesselated 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 tesselated 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 dxfio 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. The corresponding main menu entry is "`File/Import/Wavefront OBJ`" (if this menu entry is not available, you have to load the "`objio`" plugin 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 and points (e.g. curve on surface) is currently not supported. Furthermore, trimming of freeform curves and surfaces by clamping their knot ranges is not supported (however, proper UMM/VMM tags will be created, that contain the new knot minimum and maximum values)[1]. See also section 4.55.16 UMM/VMM (U/V Min Max) Tag (page 200). No material and grouping information will be imported. 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, you might want to use a third party tool first to check whether the Wavefront OBJ file is valid at all.

Furthermore, 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`").

[1] Since 1.9.

- "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) 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.
- "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.

## 7.9 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, you have to load the "objio" plugin 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 (ICurve, ConcatNC, ExtrNC)
- NPatch (with trim curves) and objects that may be converted to NPatch objects (BPatch, PatchMesh, Revolve, Sweep, Extrude, Skin, Cap, Gordon, Birail1, Birail2, Text); since Ayam 1.8.3 also Quadrics will be automatically converted to NURBS surfaces and exported
- PolyMesh and objects that may be converted to PolyMesh objects (MetaObj), faces with holes are not supported by the Wavefront OBJ format and will be tesselated to triangles for export automatically
- Box
- Instance, Clone; both will be resolved to normal objects for export as Wavefront OBJ does not support referenced geometry
- Level, CSG operations are not supported and will be written as normal Level objects, transformations will be delegated to the child objects

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.

CSG operations are fully ignored, all objects will be written as if combined by the union operator.

The Wavefront OBJ export, currently, ignores all material information. Only the pure geometry information is written to the OBJ file. However, texture coordinates from primitive variable tags can be exported.[1]

UMM/VMM tags are used to trim the knot vectors of exported NURBS objects.[2] See also section 4.55.16 UMM/VMM (U/V Min Max) Tag (page 200).

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 file;
- `"WriteSelected"`: exports only the currently selected object(s);
- `"TessPoMesh"` automatically tesselates 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.);
- `"ScaleFactor"`: allows to apply a scaling factor to all exported objects;
- `"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.

## 7.10 3DMF (Apple) Import

Using the MFIO plugin you may import scenes 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, you have to load the `"mfio"` plugin 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.

The following transformations are supported in 3DMF import:

- Scale,

[1] Since 1.8.3. [2] Since 1.9.

| 3DMF Primitive | Ayam 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 88: 3DMF (Apple) Import Conversion Table

- 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 8.8 Automatic Instancing (page 319)).

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) 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 you may export scenes 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, you have to load the "`mfio`" plugin 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 saving trim curves in 3DMF 1.0. However, 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 ( ...
                    )
          )
)
```

There will always be an enclosing container for a NURBS patch. If the patch is trimmed, after the transformations and attributes of the patch, a `TrimCurves` element will follow (which does *not* contain the trim curves but is empty). The trim curves follow now as 2D NURBS curves (simple trims) or container objects (trim loops) with 2D NURBS curves until the end of the enclosing container. 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.

#### 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 – **O**pen**N**URBS **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, you have to load the `"onio"` plugin 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.

| 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 89: 3DM (Rhino) Import Conversion Table

### 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"`: By default, all objects from all layers will be imported. Using the `"ReadLayers"` import option, a single layer or a range of layers may be selected for import.
- `"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) 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 – **O**pen**N**URBS **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, you have to load the "onio" plugin 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, polymeshes, 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 4.55.16 UMM/VMM (U/V Min Max) Tag (page 200).

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".

[1] Since 1.9. [2] Since 1.9.

### 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.

  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, you have to load the `"x3dio"` plugin 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, TrimmedNurbsPatchsurface, 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, 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.

Shape elements that contain more than one geometric elements as child objects will be imported as level objects with the child objects using the default transformations and the level object using the current transformation attributes.

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 |
|---|---|---|---|
| *Geometry3D:* | | *CAD:* | |
| 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 | *Light:* | |
| IndexedTriangleSet | PolyMesh | DirectionalLight | Light |
| IndexedTriangleStripSet | PolyMesh | SpotLight | Light |
| IndexedTriangleFanSet | PolyMesh | PointLight | Light |
| TriangleSet | PolyMesh | | |
| TriangleStripSet | PolyMesh | | |
| TriangleFanSet | PolyMesh | | |
| IndexedLineSet | NCurve | | |
| LineSet | NCurve | | |
| *Geometry2D:* | | | |
| Arc2D | NCircle | | |
| ArcClosed2D | NCurve | | |
| Circle2D | NCircle | | |
| Polyline2D | NCurve | | |
| Disk2D | Disk / Hyperboloid | | |
| *NURBS:* | | | |
| NurbsCurve | NCurve | | |
| NurbsCurve2D | NCurve | | |
| NurbsSurface | NPatch | | |
| NurbsPatchsurface | NPatch | | |
| TrimmedNurbsPatchsurface | NPatch | | |
| NurbsSweptSurface | Sweep | | |
| NurbsSwungSurface | Swing | | |

Table 90: 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 8.8 Automatic Instancing (page 319)).

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, 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, of 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 index is not checked for, whether it is identical to the vertex coordinate index (and thus no expansion would be necessary).

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) 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.
- "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.

## 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, you have to load the "x3dio" plugin using the menu entry "File/Load Plugin" first).

[1] Since 1.17.

### 7.15.1 Ayam Object and Properties Support

The export functionality of the x3dio 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 (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.

Clamping the knot ranges of NURBS via UMM/VMM tags is not supported.

PolyMesh objects will be exported to IndexedFaceSet nodes. PolyMesh faces with holes are automatically tesselated. PolyMesh faces with more than three points are tesselated if the export option `"TessPoMesh"` is used. All tesselated 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.

Object names will be converted to DEF attributes.

Instances can be resolved or exported as USE/DEF pairs.

### 7.15.2 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;
- `"WriteSelected"`: exports only the currently selected object(s);
- `"ObeyNoExport"`: toggles export of objects with `"NoExport"` tags;
- `"IgnoreHidden"`: toggles export of hidden objects;
- `"TessPoMesh"`: automatically tesselates all PolyMesh objects to triangles for export;
- `"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;

- "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 child objects of top level objects in the Ayam scene to be put in CADLayer nodes;
- "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 elsewhere.

## 8.1 The Undo System

With the help of the undo system, one may correct mistakes made while modelling. However, only modifications of objects can be undone. This includes changes made by modelling actions, changes made using property GUIs, but also changes to views (type changes or changes to the camera settings associated with a view).

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 you delete an object, it is gone! If you, accidentally, move an object 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. You may 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 (`"Prefs/Modelling/UndoLevels"`). A value of -1 for `"UndoLevels"` completely disables the undo system. The values 0 and 1 are not allowed. The value 2 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"`[1].

Several actions will completely clear the undo buffer (no undo is possible after those actions): New Scene, Open (Replace) Scene, and Close View.

Note that undo/redo will also modify objects that currently 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

[1] Since 1.14.

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 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 4.50 Script Object (page 176)).

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 4.50.4 Script Object Example (page 179) 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 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. 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 instance 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 parameterized).

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 one 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 material object will be created, but objects from the file will not be be linked to the material object from the file, but to the material object that was already registered before,
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,
6. the current directory will only be changed during the file read operation,
7. 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), the material object will be created, but objects from the file will not be be linked to the material object from the file, but to the material object that was already registered before,

8. the current file name will not be changed,
9. the scene changed state will be set to "changed",
10. the save views flag will not be changed, this means that inserting a scene file without views into an empty scene (e.g. directly after application startup or after `"File/New"`) and then saving the scene, will enrich the 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 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 are opened and parameterized),
6. if no errors occured during saving the 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 occured, the current scene changed state will be kept.

## 8.4 Ayamrc File

To customize Ayam 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. keyboard shortcuts to menu entries and modelling actions
4. 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 the factory default for the *next* start of Ayam.

```
unset ayprefs(Tolerance)
```

### 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 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 your 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:

- "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.
- "AllowWarp": controls whether the mouse pointer should be moved to the new position of points snapped to the grid while editing (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.
- "AUCommands", commands that will be run in the console when <Shift+Return> is used instead of <Return>. See also section 6.2.13 Updating the GUI (page 260). The default value is "uS;rV;", leading to a complete update of the object hierarchy, the property GUI and all view windows.
- "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.
- "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.
- "DailyTips": a list of strings that appear as tips on startup in the console (default: large).
- "EFlush": time in ms between two flushes of the error message buffer (default: 2000 - 2s).

- "FixDialogTitles", this option prepends the title of a message box that is normally displayed in the window frame of the message box to the message in the box. 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.
- "FixImageButtons", enables a workaround for buttons (e.g. in the toolbox) that stay depressed when used. 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.
- "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)
- "Kill": name of a program that kills other processes and accepts a process id as argument (used by the Rendering GUI) (default: "kill") (a setting of "kill" will be automatically replaced by "kill.exe" on Win32) On the Win32 platform you may also use an internal kill command "w32kill" 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.
- "ListTypes" determines, whether the type of an object should be displayed in braces in the tree view or listbox. The default value is 1 – yes, list the types.
- "MarkHidden" determines, whether hidden objects should be marked (using a preceding exclamation mark) in the tree view or object listbox. 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 load the scene file specified by "EnvFile" also when the scene is cleared using the main menu entry "File/New". The default value is 1 – yes, load the environment file on "File/New".
- "PaneMargins", is a list of currently five floating point values, used as a safety margin for the panes: console vs. hierarchy, hierarchy vs. upper-views, lower-view vs. property, property vs. hierarchy, upper-view-2 vs. upper-view-1 (in this order). 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}
  ```

- "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. If set to an empty string, a default of

  ```
  \[Undo:$ay(undoo)/Redo:$ay(redoo)\]...[file tail [pwd]]>
  ```

  will be used, which displays the name of the operations that one can undo and redo and the last component of the current directory of Ayam like this:

  ```
  [Undo:None/Redo:None].../scn>
  ```

  .

  One may e.g. change this to "[pwd]>" to see just the full path name of the current directory.

  To display the value of a 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".
- "SafeAutoFocus" disables AutoFocus (see section 2.9.1 preference settings (page 47)) 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. This option is enabled by default on Win32.
- "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).
- "SwapMB", "SwapMBSC" allow to swap mouse buttons 2 and 3 on MacOSX/Aqua (not X11!) for specified mouse bindings ("SwapMBSC"), because on MacOSX/Aqua, traditionally, the naming of the middle and rightmost mouse button is reversed compared to X11/Win32. This option is enabled by default on MacOSX/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.
- "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})

- "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. 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 Windows), and will always be iconified when the main window gets iconified. Not used in single window GUI mode. The default value is 1 – yes.
- "UseInternalFD" switches to an internal file dialog for loading of plugins. This option is only used on MacOSX/Aqua (not X11!), because there the normal file dialog will not enter application bundle directory structures. This option is enabled by default on MacOSX/Aqua and not used on any other platform.
- "WarnPropPasteToSel": should "Special/Clipboard/Paste Property to Selected" raise a warning requester? (default: 1 – yes)
- "WarnUnknownTag" controls the warning messages for unknown tag types. (default: 1 – yes)
- "Wait": set this to "waitPid" if you want to enable the work around for zombie processes created by the Rendering GUI. This is e.g. necessary for the Linux platform.
- "WheelZoom", a float value that controls the zoom factor, for the mouse wheel; default value is 0.5.

### 8.4.3 RiOption and RiAttributes Database

With the ayamrc file, one may also adapt the database of RiOptions and RiAttributes to the current RenderMan rendering system.

One can then easily add renderer specific options and attributes to the scenes using tags and the main menu entries "Special/Tags/Add RiOption" and "Special/Tags/Add RiAttribute", see also sections 4.55.1 RiAttribute Tag (page 192) and 4.55.2 RiOption Tag (page 193).

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 you have to change the aforementioned snippet 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 0.0 0.0}{1.0 1.0 1.0}}`
- c: a color, the default values (if provided) are three float values in curly braces, such as `{{0.0 0.0 0.0}{1.0 1.0 1.0}}`

## 8.5 Environment Variables

This section documents the environment variables used by Ayam.

- `"AYAMRC"`: designates the full filename of the ayamrc file.
- `"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).

## 8.6 Plugins Overview

This section serves as a overview of the various plugins available in Ayam.

There are currently four major types of plugins for Ayam: shader parsing plugins, custom objects, import/export plugins, and modelling helper plugins:

1. shader parsing plugins:
   aysdr, ayslb, ayslc, ayslo, ayslo3d, ayslx, ayso.

   See also section 8.7 Shader Parsing Plugins (page 318).

2. custom objects:
   metaobj, sdnpatch, csphere.

   See also section 4.51 Custom Objects (page 182).

3. import/export plugins:
   dxfio, mfio, mopsi, objio, onio, rrib, x3dio.

   See also section 7 Import and Export (page 285).

4. modelling helper plugins:

   - jsinterp – JavaScript scripting interface (see section 6.6 JavaScript Scripting Interface (page 280)),
   - AyCSG – CSG rendering (see section 8.10 CSG preview using the AyCSG plugin (page 320)),
   - IDR – Importance Driven Rendering (see section 8.9 Importance Driven Rendering (page 319)),
   - aydnd – inter-application drag and drop
   - subdiv – Catmull-Clark and Loop subdivision for the SDMesh object (see section 4.34 SDMesh Object (page 137)).

## 8.7 Shader Parsing Plugins

The following plugins are provided to allow parsing of shaders[1] : `"ayslb"` for Air, `"ayslx"` for Aqsis, `"ayso"` for RDC, `"ayslo"` for PRMan, `"ayslo3d"` for 3Delight, `"aysdr"` for Pixie[2] , and `"aygso"` for Gelato.[3]

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 you load the shader parsing plugin automatically on startup of Ayam, as the loading of the scripts 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 you load a shader parsing plugin manually or via unchanged load script you need to adapt the shaders search path manually and also initiate a shader scan manually. 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.

[1] Since 1.3. [2] Since 1.6. [3] Since 1.11.

## 8.8 Automatic Instancing

Automatic Instancing is available via the main menu entry: `"Special/Instances/Automatic Instancing"`. Automatic Instancing creates instances from all instantiable objects, using a simple algorithm that recursively compares objects. The comparison of materials and tags may be turned off in the small dialog that pops up after selection of the menu entry `"Special/Instances/Automatic Instancing"`.

The 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.

Also note that the instantiation obeys the object selection:[1]
instances are only created from the selected objects. If no objects are selected, instances are created from all objects in the current level and below.

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*

## 8.9 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.

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).

[1] Since 1.14.

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 atleast 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*

## 8.10 CSG preview using the AyCSG plugin

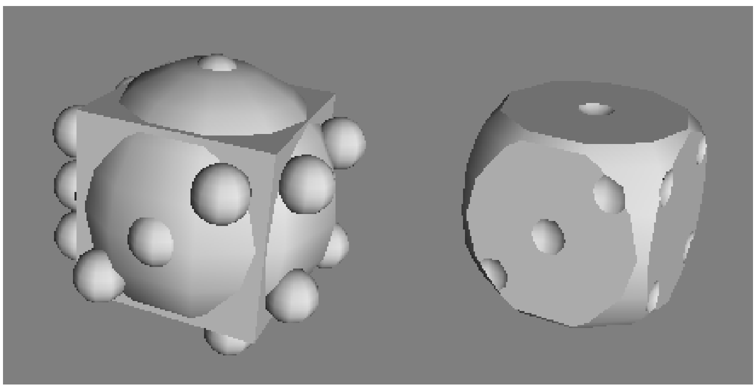


Figure 94: CSG preview example (left without, right with CSG)

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.

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 you fail to correctly specify the depth complexity, 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 5.37 Revert U tool (page 216)). 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+Shift+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 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"`:

- `"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.

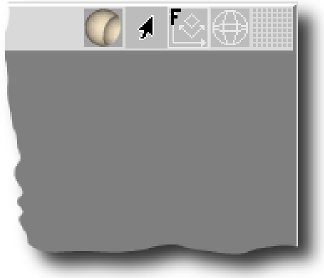


Figure 95: View With AyCSG Icon

- "`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*k') algorithm (where k' <= 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*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 313)). If stencil buffers are not enabled, Ayam will automatically add the relevant option to the 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 DCSampling). 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).

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*

## 8.11 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.
- If you need to see curves/surfaces, try to increase the (GLU) sampling tolerance of the objects (use a value of about 60.0).
- 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".

## 8.12 Modelling Without Views

To work with very large/complex scenes, it may be necessary to turn off all views. In Ayam one can do this in the multi window GUI mode (see section 2.9.1 GUI preference settings (page 47)). 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 MacOSX (Aqua) if GLU functionality is involved (as every GLU functionality on MacOSX needs a GL context). GLU is needed for the PolyMesh and NURBS tesselation.

## 8.13 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).
- 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 Marsrakete uses just 62 objects.

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.

## 8.14 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. Create and submit example objects, scenes, and images.
3. 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.
4. 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 or in the feature request tracker.
5. Donate money by registering ShellyLib. ShellyLibs source will be converted to a first high level custom object that creates objects of type seashell for Ayam. This object, however, will be Shareware.

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.

## 8.15 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:

- If you are reading this document from a local file system, maybe an updated version is available from the internet: `http://www.ayam3d.org/docs/ayam.html`
- Ayam Tutorial #1: `http://www.ayam3d.org/tut1/tutorial1.html`
- The Ayam FAQ: `http://www.ayam3d.org/faq.html`
- The RenderMan Repository: `http://www.renderman.org/`
- The RenderMan Academy: `http://www.rendermanacademy.com/`
- George W. Hart, Virtual Polyhedra
`http://www.georgehart.com/virtual-polyhedra/conway_notation.html`

## 8.16 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
- 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
- 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

# 9 Index

Overview:



Index:

**0**



**A**

**D**

**U**