Ayam

Randolf Schultz (rschultz@informatik.uni-rostock.de) 12. Mar 2005

This is the documentation of Ayam (1.8) - a free 3D modeling environment for the RenderMan interface. Please note, that this document is intended to be 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 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 for you. In addition, you can use the provided SGML-source to generate other formats.

Contents

1 Introduction

What is Ayam?

Ayam is a free 3D modeling environment for the RenderMan Interface (formerly known as `"The Mops"`).

On which platforms is Ayam available?

Ayam is primarily aimed at the platforms BMRT (Blue Moon Rendering Tools, a RenderMan compliant renderer by Larry Gritz (Exluna)) runs on. Those are Linux, IRIX, and Win32. 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 BMRT shader parsing and RIB writing by Aqsis, thus completely eliminating the need for BMRT.

What are the features of Ayam?

Here is a short summary:

- RIB (RenderMan Interface Bytestream) export and import.
- Support for NURBS curves, interpolating curves, (trimmed) NURBS 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 modeling includes extrude, revolve, sweep, birail, skin, and gordon operations (with caps, holes, and bevels) realized as Tool-Objects.
- 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 interface: Tcl.
- Misc: instancing, arbitrary number of modeling views, object clipboard, independent property clipboard, console, n level undo.

Due to limitations in the Win32 operating system(s) (no backlinking) and the current implementation of Ayam, dynamic loading of custom objects is currently not available on the Win32 platform!

How should one read this manual?

Since this manual is intended to be a reference manual only, 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, use it!

In this manual, the following typographic conventions are used:

- keyboard shortcuts: <Ctrl+c>
- 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)
```

2 The Ayam GUI

This section describes the user interface of Ayam.

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

The whole application with all 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.

2.1 Anatomy of the Main Window

The main window is split into three large areas:

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

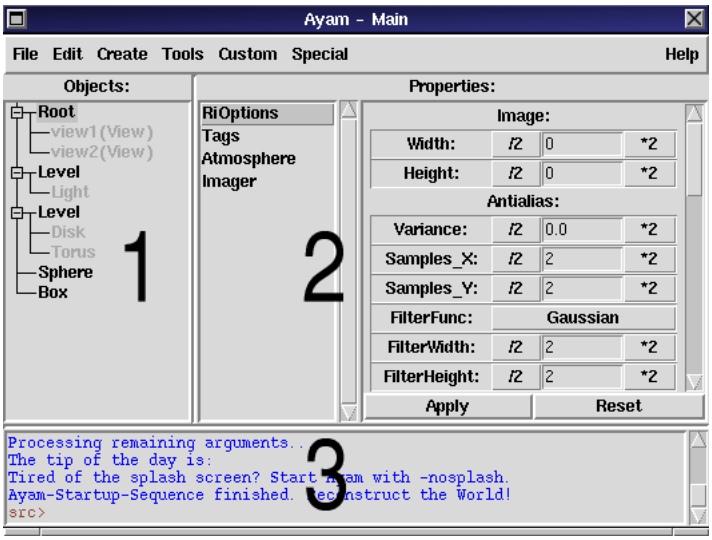

Figure 1: The Main Window

The relative sizes of the three areas are managed by a so called paned window management. To change the relative size of the console, you may move your 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 (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 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 only take place in the current level!

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 4.7 (Root Object) (page 47) for more information regarding this special object, and what it is good for. Note that you cannot delete or copy this object.

Tree View:

The tree view is very complex and may be slow on slow machines (of the Pentium 90 class), especially when dealing with large (deep nested) scenes. This should not be a problem nowadays.

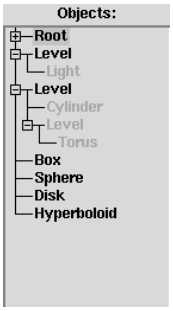

Figure 2: Tree View

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 (Ayam Objects and Properties) (page 38) 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, and selects the root object
- "Tree/Expand" opens all nodes with child nodes
- "Tree/Collapse" closes all nodes with child nodes
- "Switch to Listbox" removes the tree 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 16).

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, since Ayam 1.7 holding down the <Shift> key while pressing <Up> or <Down> will not move the selection, but rather extend it in the respective direction,
- <Home> and <End> select the first or last object in the current level,
- <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).

If those shortcuts do not work you may need to press <Esc> first (when a property GUI has the input focus) or <Shift-Tab> (when the console has the input focus).

Listbox:

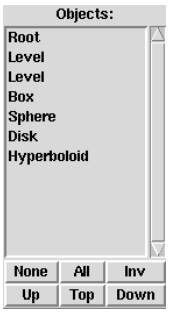

Figure 3: Listbox

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

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

The rightmost mouse button opens a small context menu:

- "Switch to Tree" removes the listbox and replaces it with the tree view (see above).
- "Copy Object", "Cut Object", "Paste Object", "Delete Object" are standard clipboard operations as documented in section 2.2 (main menu) (page 16).

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, since Ayam 1.7 holding down the <Shift> key while pressing <Up> or <Down> will not move the selection, but rather extend it in the respective direction,
- <Home> and <End> select the first or last object in the current level,
- <Right> enters the (first) selected object,
- <Left> enters the parent level,
- <Ctrl+a> and <Ctrl+n> select or de-select all object 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 press <Esc> first (when a property GUI has the input focus) or <Shift-Tab> (when the console has the input focus).

2.1.2 Properties

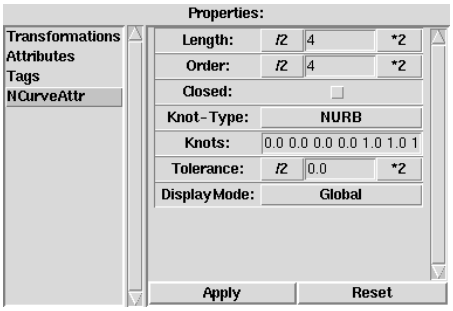

Figure 4: Properties

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 objects 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).

Since Ayam 1.8 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.

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 the elements of the property GUI do not fit into the screen space that is defined by the current window size, Ayam will automatically resize the main window when you select a property. You can toggle this behaviour using the preference setting "AutoResize" (see section 2.7 (Preferences) (page 27)).

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 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 in section 4 (Ayam Objects and Properties) (page 38).

Here are some general remarks:

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!

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

Note that property GUIs of custom objects may also 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.

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 properties 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 surface shader to a big number of material objects, without going the long way of setting a new shader and entering parameters for it for every object.

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

If you now copy this property all marked parameters will be omitted.

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

A simple example:

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 opacity values. We can accomplish 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!

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 text output (informative, warning and error messages). For this, the console captures the stderr and stdout channels of the Tcl-interpreter

Ayam is running in. You can also redirect all Tcl error messages that would normally cause a Tcl error requester to appear to the console using the preference setting `"Misc/RedirectTcl"` (see section 2.7.5 (Miscellaneous Preferences) (page 33)).

But you can also enter commands, new Tcl procedures and so on in the console. However, this is a feature for the advanced user that studied section 6 (The Tcl Scripting Interface) (page 101). 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`> 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).

Another simple demonstration of the consoles capabilities:

- Create ten boxes 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 (The Tcl Scripting Interface) (page 101) 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 The Main Menu

Another important part of the main window has not been discussed so far. This is the main menu bar. Note that many menu entries have keyboard shortcuts that are displayed in each entry. You can adapt the shortcuts using the file `"ayamrc"` (See section 7.2 (Ayamrc File) (page 119)).

The `File` menu deals with standard file operations:

- `New`, clears the current scene (deletes all objects) and reloads the working environment (if the preference setting `"Main/NewLoadsEnv"` is enabled).
- `Replace`, clears the current scene and closes all views, then loads a new scene from disk. All objects from the file will be read.
- `Insert`, inserts the objects and views of a ayam file into the current scene. All objects from the file will be read.
- `Save as`, saves the current scene asking for a new file name.
- `Save`, saves the scene. If the scene has not been saved before (read, you have not given it a file name) you will be asked for a file name.
- `Import Mops`, import a scene from The Mops, see section 7.4 (Import of Mops Scenes) (page 122) for more information.
- `Export/RenderMan RIB`, exports the current scene to a RIB, asking which camera (which view) to use.
- `Export/Wavefront OBJ`, exports the current scene to a Wavefront OBJ file, see also section 7.6 (Wavefront OBJ export) (page 124).
- `Load Custom`, 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 4.38 (Custom Objects) (page 89) for more information about custom objects.
- `Save Prefs`, save the current preference settings to the ayamrc file after making a backup copy of this file (see section 7.2 (Ayamrc File) (page 119) 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.
- `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.
- `Cut`, moves the currently selected object(s) into the clipboard.
- `Paste`, copies the object(s) from the clipboard to the current level of the scene. 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 using the menu entry `"Special/Clipboard/Paste (Move)"`.
- `Delete`, removes the selected object(s) from the scene.

- `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 a property from the property clipboard to the currently selected object(s).
- `Undo`, perform undo operation (see section 7.1 (The Undo System) (page 118) for more information)
- `Redo`, perform redo operation (see section 7.1 (The Undo System) (page 118) 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 first.
- `Master`, searches for the master object of the currently selected instance object and selects it for editing, see also section 4.22 (Instance Object) (page 67).
- `Preferences`, opens the preferences dialog (see section 2.7 (Preferences) (page 27) 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 requesters, where you may adjust parameters for the object to be created. 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 4.15 (NCurve Object) (page 62).
- `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 4.32 (ICurve Object) (page 83).
- `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 4.16 (NPatch Object) (page 63).
- `BPatch`, create a new bilinear patch. See also section 4.18 (BPatch Object) (page 65).
- `PatchMesh`, create a new patch mesh. See also section 4.19 (PatchMesh Object) (page 65).
- `Solid`, create a new solid primitive object, for use in CSG. `Box`, `Sphere`, `Disk`, `Cone`, `Cylinder`, `Torus`, `Hyperboloid` or `Paraboloid` may be selected. See also section 4.11 (Quadric Primitives) (page 52).
- `Level`, creates a new hierarchy object. `Level` just groups objects, `Union`, `Intersection`, `Difference`, and `Primitive` are CSG operations. See also section 4.12 (Level Object) (page 56).
- `Light`, create a new light source. See also section 4.14 (Light Object) (page 58).
- `Custom Object`, create a new custom object. If this sub-menu is empty no custom object has been loaded yet. See also section 4.38 (Custom Object) (page 89).

- `Instance`, create an instance of the currently selected object, see section 4.22 (Instance Object) (page 67) for more information regarding instances.
- `Clone`, create a clone object, see section 4.23 (Clone Object) (page 68)
- `View`, a new View window will be opened. See also section 4.8 (View Object) (page 49).
- `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 4.13 (Material Object) (page 56).
- `Camera`, create a new camera. Camera objects may be used to temporarily save view camera settings, see section 4.9 (Camera Object) (page 51).
- `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 4.36 (RInc Object) (page 87).

The `Tools` menu:

- `Create`, `NURBCurve`, and `NURBPatch`, are sub-menus with various NURBS based modeling tools, that are explained in depth in section 5 (NURBS Modeling Tools) (page 91).
- `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. Normally, the currently selected PolyMesh objects will not be changed by this tool. But you may let the merge-tool delete them immediately, if you enable 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.
  - `Optimize`: optimizes the selected PolyMesh object(s) by removing all multiply used 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.
- `Show`, `Hide` set and unset the `Hide` attribute of the selected object(s) thus making them invisible. 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 are not undoable.
- `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 is not undoable, i.e. the newly created objects will not be removed by 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, is undoable.
- `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 tree view again by pressing <`Ctrl+l`>.

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

The `Special` menu:

- `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/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! You 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).
- `Points/Select All Points`, selects all points of the currently selected object(s).
- `Points/Invert Selection`, selects all points of the currently selected object(s) that are not selected, and de-select all points that are currently selected.
- `Points/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 transforming the points.

- `Points/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 transforming the selected points. The points currently not selected will be transformed, however!
- `Instances/Resolve all Instances`, converts all instances of the current level (and its child objects) to normal objects.
- `Instances/Automatic Instancing`, pops up a small dialog box, where you may parameterize and start the automatic instantiation algorithm (that automatically creates instances from equal objects). See section 7.10 (Automatic Instancing) (page 127) for more information regarding automatic instancing.
- `Tags/Add RiOption`, pops up a small dialog box, where you may select and parameterize a RiOption tag to add as tag to the Root object (see 4.6.2 (RiOption Tag) (page 43)). 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 parameterize a RiAttribute tag to add as tag to the currently selected object(s) (see 4.6.1 (RiAttribute Tag) (page 42)).
- `Tags/Edit TexCoords`, opens the texture coordinates editor. (see also section 4.6.3 (TC (TextureCoordinates) Tag) (page 43)).
- `Select Renderer` opens a dialog where the renderer for direct rendering from a view may be choosen. The changes will have effect on all options 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 7.9 (Shader Parsing Plugins) (page 126))) and rescan for compiled shaders. Note that in order for the `"ScanShaders"` feature to work properly you also have to set the `"Main/Shaders"` and `"Main/Plugins"` preference setting correctly (see also section 2.7.1 (The Main Preferences) (page 28)).
- `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 RIB`.
- `RIB-Export/Selected Objects`, exports only the selected objects to a RIB. Note that instances will always be resolved, hidden objects and objects with `"NoExport"` tags are treated as on normal export operations, and light objects are simply ignored. Note also that this 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. ReadArchive.
- `RIB-Export/Create ShadowMaps`, creates the shadow maps for the current scene. See also section 4.14.2 (Using ShadowMaps) (page 59).
- `Toggle ToolBox` closes or opens the toolbox window (see 2.6 (The Tool Box Window) (page 26)). From version 1.3 on, Ayam remembers the state of the toolbox in the saved preferences.

- `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 10) 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 modeling actions, you may leave this window open when doing your first steps in modeling with Ayam.
- `About`, displays some version, copyright, and trademark information.
- `Show Tooltips`, enables tool tips (balloon help) for various user interface elements (including the tool box buttons).

2.3 Anatomy of a View

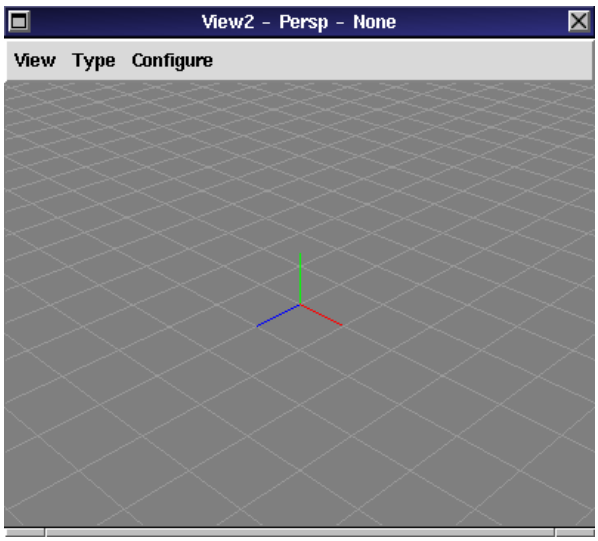

Figure 5: A View Window

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

2.4 The View Menu

Here are all meny entries of the "View" menu:

- Quick Render: the scene is exported to a RIB using the camera settings of the current view; then the "QRender" command (see the preferences) will be called. Note that the RIB export will override the RiOption settings for image size and use 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 using the camera settings of the current view; then the "Render" command (see the preferences) will be called. Note that the RIB export will override the RiOption settings for image size and use 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 Export RIB, except that the current view is already 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 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 more than one open RIB stream, RIB-Export and direct rendering from view windows is not available until the permanent view window is closed.
- Create ShadowMaps: creates the shadow maps for the current scene. See also section 4.14.2 (Using ShadowMaps) (page 59).
- Close: the View window will be removed.

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 modeling actions. See sections 4.8 (View Object) (page 49) and 2.4.1 (View Window Shortcuts and Actions) (page 24) for more information.

The "Configure" menu may be used to change preferences of the view. Some preferences are outlined in greater detail in section 4.8.2 (ViewAttrib) (page 50).

- Automatic Redraw, toggles whether the view should be redrawn, whenever the scene changes. If this is disabled, you can force a redraw using View/Redraw.
- 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).
- 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 BGImage, if this is enabled, a 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 modeling 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 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.
- Edit Local, if this is enabled modelling will take place in local object space and not in world space. The grid will be changed accordingly (interpreted as if defined in the local coordinate space). See also section 3.9 (Editing in Local Space) (page 38).

The "Grid" menu may be used to change the current grid size:

On the right hand side in the 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 value and additionally switch on drawing of the grid and snapping to the grid. The last entry, however, will set the grid size to 0.0 and switch off drawing and snapping to the grid. If a gridsize other than 0.1, 0.25, 0.5, or 1.0 is in effect for the view, a different icon with a small x will be displayed in the menu instead.

2.4.1 View Window Shortcuts and Actions

Important keyboard commands of a View:

- <Left>, <Up>, <Right>, <Down> rotate viewer around origin.
- <Add>, <Sub> (on the numeric keypad) zoom view.

Interactive actions modifying a view:

- 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 modeling mode, when holding down the <Alt>-key.
- <o> starts zooming the view. Moving the mouse up zooms in and moving the mouse down zooms out.
- Since Ayam 1.7 zooming the view into a rectangular region defined through a mouse drag is also possible in any modeling mode, when holding down the <Shift>-key.

You may also move the view by dragging with the rightmost mouse button and zoom the view with the middle mouse button.

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.

Using the menu entry "Zoom to Object" or the shortcut <Ctrl+o> you can change the views from to and zoom settings so that the selected objects will be displayed centered in the view window. This is handy, if you are e.g. searching for objects or simply lost in space.

Using the menu entry "Align to Parent" or the shortcut <Ctrl+a> you can change the views camera settings so that it is aligned to the coordinate system of the currently selected object. This is handy for modeling in local coordinate systems (e.g. editing the points of a 2D curve defined in the XY-plane that has been rotated around the Y-axis). See also section 3.9 (Editing in Local Space) (page 38).

2.5 Selecting Objects within a View

When the views action is `"Pick"` you can pick objects that appear within a view. You can invoke this action by pressing <`Shift+p`> or make this action the default action using the preference setting `"Modeling/DefaultAction"`. This section describes techniques that you can use for selecting one or more objects within a view.

2.5.1 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 2.5.2 (Drag-selecting Multiple Objects) (page 25)

2.5.2 Drag-selecting Multiple Objects

You can select multiple objects using the <`Control`>+Click method described in section 2.5.1 (Selecting Individual Objects) (page 25). 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.

2.5.3 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 6: List of Ambiguous Candidates

If your 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.

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 <`Ctrl+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 7.2.2 (Hidden Preference Settings)).

2.6 The Tool Box Window

The tool window displays some buttons that start interactive modeling actions or other modeling tools, or create objects.

Note that in contrast to the keyboard shortcuts of the view windows the buttons switch to the modeling actions for all available views. For more information about the actions see section 3 (Interactive Actions) (page 34).

The tool window may be configured by the user using the preference setting toolBoxList in the ayamrc file. You may select from the groups and change the order in which they appear in the tool window. See section 7.2.2 (Hidden Preference Settings) (page 120) for more information.

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 tool box will re-layout the buttons, warning you if the space is to small for all buttons to display, and if the window is too big for the desired layout shrink wrap the window to match the size occupied by the buttons. Furthermore, using the preference setting

Figure 7: The Tool Box

`Misc/ToolBoxTrans` the tool box 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.

2.7 Preferences

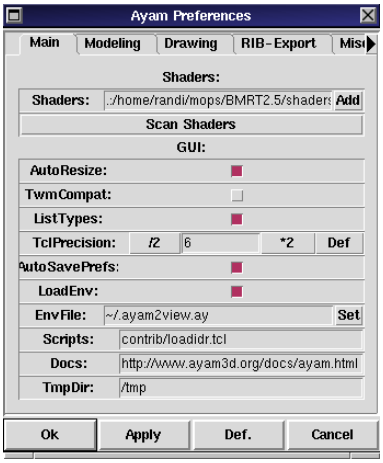

Figure 8: Preferences Dialog

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

Use

- `"Ok"` to close the preference editor and apply all changes,
- `"Apply"` to apply the changes, but leave the editor open,

- "Revert" to reset 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"!),
- "Cancel" to close the dialog without applying changes (done after the last press of "Apply").

The preferences are divided into five sections.

2.7.1 The Main Preferences

The "Main" section contains the most used preference settings:

- "Shaders" contains a number of paths (separated by ":" on Unix and ";" 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 add another path, simply press it and select a file inside 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.
- "Scan for Shaders!" initiates a rebuild of the internal shaders database. All shaders in the directories specified by the "Shaders" entry will be scanned and entered in that database. Watch the console for error messages that may appear while scanning. See also section 4.5.1 (Shader Parsing) (page 41) 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!
- "AutoResize" toggles, whether the main window should be resized horizontally according to the property GUI whenever a new GUI is displayed.
- "AutoFocus" controls whether Ayam should automatically move the focus to a view or the main window, when the mouse pointer enters it. Note that this may only work correctly, when a window of Ayam already has the focus. Notice also, that moving the focus to a window might also raise it (depending on your operating system or window manager).
- "TwmCompat" changes, how Ayam tells the window manager new geometries of windows. Might be useful, if you suffer from jumping windows.
- "ListTypes" determines, whether the type of an object should be displayed in brackets in the tree view or listbox.
- "MarkHidden" determines, whether hidden objects should be marked (using a preceding exclamation mark) in the tree view or object listbox.
- "AutoSavePrefs", if this is switched on, Ayam will write the preferences settings to the file ayamrc when the program quits.

- "BakOnReplace", if this is switched on, Ayam will make a backup copy of each scene file it loads via main menu "File/Replace" or via the most recently used list.
- "LoadEnv", if this is switched on, Ayam will load the scene file specified by "EnvFile" on startup.
- "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".
- "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 preference option "LoadEnv", see above, is switched on) and upon clearing the scene using "File/New" (if the preference option "NewLoadsEnv", see above, is switched on).
- "Scripts" is a list of Tcl scripts that will be executed on startup. The scripts have to be specified with full path and filename. 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.
- "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. Multiple entries have to be separated by a colon (:) on Unix and by a semicolon (;) on Win32.
- "Docs" is an URL that points to the documentation in HTML format.
- "TmpDir" is the directory, where temporary RIBs are created, e.g. when rendering directly from view windows.

2.7.2 Modeling Preferences

The next section of the preferences, "Modeling", contains modeling related settings:

- "PickEpsilon" is used by the single point editing actions to determine which point (vertice) of an object has been selected. A smaller "PickEpsilon" means more exact picking. The value of "PickEpsilon" should be positive. In older versions of Ayam (prior to 1.8), a value of 0.0 was allowed. This is no longer the case.
- "LazyNotify" determines whether notification shall occur on all mouse movements or just on release of the mouse button, for the interactive modeling actions. Notification is the mechanism used to inform objects that rely on certain child objects (e.g. the Revolve custom object) about changes of their child objects, so that the parent can adapt to the child objects automagically.
- "EditSnaps" determines, whether points should be snapped to the grid when a grid is defined and in use for the single point modeling actions.
- "Snap3D" controls, whether points that are snapped to grid coordinates (in single point editing actions, 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 modeling actions when they would be modified by a click and drag action. Note that a change of this preference option takes effect after the current modeling action has been switched off and the single point editing has been activated (again).
- "DefaultAction" determines the modelling action that should be active after a press of the <Esc> key.
- "UndoLevels" determines the number of modeling 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 7.1 (The Undo System) (page 118).

2.7.3 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.
- "DisplayMode" 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 just affects the drawing of patches if the view is not in shaded mode. Note also, that this setting has no effect for objects that override it using a local DisplayMode setting different from ”Global”.
- "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”.
- "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.

2.7.4 The 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 behind the 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 parameterized as well, see section 4.14.2 (Using ShadowMaps) (page 59). 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 (using the view menu entry: "View/Create 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 7.2.2 (Hidden Preference Settings) (page 120).
- "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 using the view menu entry "View/Create ShadowMaps", %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.
- `"PPRender"` is the name of the renderer to use for the permanent preview feature (see also section 2.4 (View Menu) (page 22)). This setting is just available, if the compile time option `AYENABLEPPREV` has been set. This option is not set for the official Ayam binaries.

2.7.5 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, if you write and debug 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.

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 list in the main menu for quick access.
- `"ToolBoxTrans"`; controls whether the tool box window should be declared as a transient window of the main window. 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.
- `"ToolBoxShrink"`; controls whether the tool box window should automatically shrink-wrap around the calculated layout of the buttons after a resize operation.
- `"RGTrans"`; controls whether the RenderUI-windows should be declared as a transient window of the main window. See the discussion of `"ToolBoxTrans"` above for more information about transient windows.
- `"HideTmpTags"` may be used to hide tags that are marked temporary (internal tag types do so) from the tag property GUI.
- `"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 loose information in doing so. The default value used by Tcl is 12 and results in no loss of information. 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+). Three methods are available: `"DomainDistance"` (the default) simply tesselates the NURBS into equally sized pieces with regard to parametric space; `"SParam"` controls the number of sampling points in u and v direction; `"PathLength"` ensures that no edge of a polygon generated by the tesselation is longer than the value specified by `"SParam"`; `"ParametricError"` ensures that the distance between the tesselated surface and the original surface is no point bigger than the value specified by `"SParam"`. Note that `"SParam"` is expressed in object space units for the last two sampling methods.

- `"SParam"`; 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.

3 Interactive Actions (Modeling)

Before invoking any modeling action you should select one or more objects using the main window or using the pick action!

Every action can be started with a key press (a shortcut) when the focus is in a view window or by pressing the associated button in the tool window. Using a shortcut starts that action in the current view only, the other views are not affected. Starting an action from the tool window will cause the action to be started in all view windows simultaneously.

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 `"Modeling/DefaultAction"`) See section 2.5 (Selecting Objects within a View) (page 25) for more information about picking objects.

Note that the modeling 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 to mark a point in space or to pick a vertex, then dragging the mouse.

You may undo/redo the effects of an action using <Ctrl+z> and <Ctrl+y> (see section 7.1 (The Undo System) (page 118) for more information).

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

Also note that you may use the middle and rightmost mouse button to zoom and move the view while modeling actions are active.

For actions that modify the camera of a view please see section 2.4.1 (View Window Shortcuts and Actions) (page 24).

3.1 Moving Objects or Selected Points

Using the modeling 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.

3.2 Rotating Objects or Selected Points

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

Note that if multiple objects are selected, each object is rotated around the center of its own local coordinate system. 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.

3.3 Rotating Objects or Selected Points around a Point

Using the modeling action "Rotate about" (shortcut: <a>) you may rotate objects or the selected (tagged) points of the selected objects around a specified point in space. The action requires a point to be specified using a single click after the action has been started. The point will then be marked by a little red cross. If you want to rotate about a different point, you need to restart the action (press <a> again).

After the first click, the action works the same way as the Rotate action, except that it rotates around the specified point. This also works with multiple selected objects. Note that this action does not only change the Rotate_X(_Y,_Z) properties of the selected objects, but also the Translate_X(_Y,_Z) properties.

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

3.4 Scaling Objects or Selected Points

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

The modeling 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 modeling 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.

The modeling actions "Scale X" (shortcut: <x>), "Scale Y" (shortcut: <y>), and "Scale Z" (shortcut: <z>) scale only one axis of the selected objects or the selected (tagged) points of the selected objects.

The modeling action "Stretch 2D" (shortcut: <Alt+s>) 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 vertices, then try it once starting on the X-axis.

3.5 Selecting Points

The modeling action "Select Points" (shortcut: <t>; tag points) may be applied to a NURBS curve, NURBS patch or objects that support single point editing only. Objects draw the selectable points using small rectangular handles. Selected points will be drawn in dark red.

The selected points may be modified using the modeling actions Move, Rotate, and Scale as discussed above. Selected points always take precedence for those modeling actions.

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. Note that the list of selected points will not be deleted from the object until a de-select is performed using the shortcut <D>.

Note that the list of selected points is not copied, if the object is copied using the clipboard. Undo and redo will destroy the list of selected points too!

However, 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.

You 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.

3.6 Editing Points

To edit the points of an object three actions ("Edit", "Edit Weights", and "Direct Point Edit") are available. All those actions may be applied to objects that support single point editing only. Objects mark themselves editable by drawing the editable points using small rectangular handles if one of the single point editing action is activated and the object is selected.

- The modeling action "Edit" (shortcut: <e>) works much like the move action, but it moves single points instead of objects. In contrast to the move action, you need to pick on the handle of the point you want to move. Furthermore, it is not possible to move points of multiple selected objects, only

the first selected object is considered. If a NURBS curve has multiple points, this action modifies all points that make up the multiple point.

- The modeling 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 points that make up the multiple point.

- The modeling action `"Direct Point Edit"` (shortcut: <p>) opens a small window where you may change the coordinates of the selected point directly by entering numbers. Note that the w coordinate setting will be ignored if the picked point does not have weight information (is not homogenous). Using the small menu on top of the coordinate window you may determine whether editing takes place in local object or global world space. If a NURBS curve has multiple points, this action modifies all points that make up the multiple point.

  Notice that since Ayam 1.4 the direct point editing dialog may stay open all the time. Furthermore, it is not necessary that the original object stays selected, you may select other objects to e.g. infer new point coordinates from their properties and apply them to the original object. However, certain actions like deleting objects, will also delete the reference to the selected points. In this case you will have to select the object and then a point to edit again. Furthermore notice that the coordinate values displayed in the direct point editing window will not update when the point is modified by another modeling action. Simply click on the point again in a view where the direct point editing action is active, to update the coordinate values in the direct point editing dialog. This modelling action also may only be applied to a single selected object.

3.7 Inserting or Deleting Points

The modeling action `"Insert Point"` (shortcut: <i>) may be applied to NURBS and interpolating curves (objects of type NCurve and ICurve) 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 the insert knot tool 5.21 (The Insert Knot Tool) (page 98).)

The modeling action `"Delete Point"` (shortcut: <d>) may be applied to NURBS and interpolating curves (objects of type NCurve and ICurve) 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.

3.8 Miscellaneous Actions

This section documents some special modeling actions.

- The modeling action `"FindU"` (shortcut: <u>) may be applied to NURBS curves (objects of type NCurve) only. This action may be used to get the corresponding parametric value u from a point on

a curve. Pick a point on the curve (not a control point!). If this is done, the appropriate value for u is calculated, stored in the global variable u, and additionally written to the console. A small cross is drawn at the position of the picked point. Remember to exactly pick a point on the curve or nearby, otherwise the calculation may fail and no value will be written to the console.

- The modeling 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!

3.9 Editing in Local Space

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, you can also edit in local object space. This means you can e.g. edit a two-dimensional parameter curve of a skin object where both objects (curve and skin) are rotated and scaled arbitrarily and make sure that the curve remains two-dimensional all the time.

All you need to do is to first select the curve and then press <`Ctrl+a`> to align the view and then <`Ctrl+l`> to make it local. In practice, 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"`).

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).

4 Objects and Properties

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.

4.1 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"`.

4.2 Transformations Property

Use this property to edit the location, orientation, and size of an object.

- `"Reset All!"` immediately resets all transformation attributes to the default values.
- `"Translation_X (_Y, _Z)"` is the displacement of the object from the world origin in X (Y, Z) direction.
- `"Rotation_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!
- `"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.
- `"Quat0 (1, 2, 3)"` the quaternion that is used to determine the orientation of the object in space. This quaternion is not here to be edited directly! The sole purpose of its appearance here is to allow copying and pasting of rotations.

The transformations are applied to the object in the following order: Scale, Rotation, Translation.

4.2.1 Using 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 until 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 the orientation of an object to other objects using the property clipboard, make sure that you select all Rotation and Quat property elements.

4.3 Attributes Property

The attributes of an object contain 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.
- `"Refcount"`, just displays how many objects point to this object. Objects with a reference count different from zero may not be deleted.

4.4 Material Property

The material property allows you to connect geometric objects to material objects (see also section 4.13 (Material Object) (page 56)). 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.

4.5 Shaders 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 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 4.5.2 (Working with Shaders) (page 41) below.

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

4.5.1 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 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.com/"`) 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 7.9 (Shader Parsing Plugins) (page 126).

4.5.2 Working with Shaders

The `"Default Values."`-button resets all arguments of the shader to the default values. For that, the compiled shader will be parsed again. 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. Note that this works only properly, 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 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 13)).

4.6 Tags Property

Use this property to edit the tags of an object.

Tags provide an easy way to attach arbitrary information 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 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 currently available tag types. Note that extensions and plugins may define their own types.

4.6.1 RiAttribute Tag

The tag type `"RiAttribute"` can be used to attach arbitrary RenderMan interface attributes to objects. This is handy if you use a renderer with lots of RiAttributes that differ from the standard RiAttributes.

`"RiAttribute"` tags attached to a geometric object override `"RiAttribute"` tags possibly attached to the material object of this geometric object.

In order to create a tag of type RiAttribute, the type string must be `"RiAttribute"`. The syntax of the value string is as following:

```
attrname,paramname,paramtype,param
```

where attrname is the name of the attribute (e.g. ”render”) paramname is the name of the parameter (e.g. ”displacementbound”) paramtype is a single character defining the type of the parameter (it may be one of `f` - float, `g` - float pair, `i` - integer, `j` - integer pair, `s` - string, `c` - color, `p` - point) and finally param is the value of the parameter itself (e.g. a float: ”1”, a string: ”on”, a color: ”1,1,1” or a point: ”0.4,0.5,1.0”).

Examples:

```
RiAttribute render,truedisplacement,i,1
```

```
RiAttribute dice,numprobes,j,3,3
```

```
RiAttribute radiosity,specularcolor,c,0.5,0.5,0.5
```

Note that 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 the user using the ayamrc file, see section 7.2 (Ayamrc File) (page 119).

4.6.2 RiOption Tag

The tag type `"RiOption"` can be used to attach arbitrary RenderMan interface options to objects. This is handy if you use a renderer with lots of RiOptions that differ from the standard RiOptions. 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. Note that 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! Furthermore, the database of RiOptions for this GUI may be extended by the user using the ayamrc file, see section 7.2 (Ayamrc File) (page 119).

4.6.3 TC (TextureCoordinates) Tag

The tag type `"TC"` can be used to attach texture coordinates to objects or materials. `"TC"` tags attached to a geometric object override `"TC"` tags possibly attached to the material object of this geometric object. The `"TC"` tag always contains a list of eight comma separated float values, that specify a mapping for four points (a quadrilateral) in texture space from the default values (0,0), (1,0), (0,1), and (1,1) to the new specified values.

Examples:

```
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.

Note that the exact behaviour depends heavily on the shader and its use of the texture coordinates!

Texture Coordinate Editor 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 texture coordinates are displayed as a black polygon in a canvas with regard to the original values, that are displayed in gray. Small arrows point to positive s and t direction respectively.

The `"RotateR"` and `"RotateL"` buttons shift the coordinate values between the four points. This results in a 90 degree rotation.

The `"FlipS"` and `"FlipT"` buttons flip the texture coordinate values in s and t direction respectively. This is useful, if you 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.

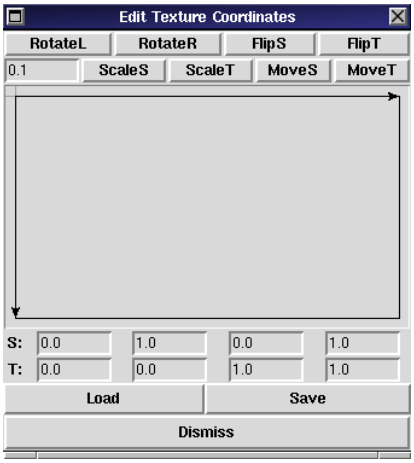

Figure 9: Texture Coordinate Editor

The "Load" and "Save" menu buttons allow you to:

- load the default values
- 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.
- 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 save the TC tag to multiple selected objects. Use the property clipboard to achieve this.

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.

4.6.4 PV (Primitive Variable) Tag

The tag type "PV" can be used to attach arbitrary data to geometric primitives and even sub-primitives. When rendering, all data is handed over to the surface shader attached to the respective geometric primitive. Note that Ayam does not check, whether your shader actually uses this data. 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 renderer before it is handed over to the shader.

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 "uniform", "varying", "vertex", or "constant"; "<type>" is a single character describing the type of the data (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 a certain number of 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;

```
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 standard NURBS patch (order, width, and height 4).

Note that not all geometric objects currently honour PV tags on RIB export. The geometric objects currently supporting PV tags are: SDMesh, PolyMesh, PatchMesh, NURBPatch, 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!

4.6.5 RiHider Tag

The tag type "RiHider" can be used to choose and parameterize different algorithms for hidden surface removal. 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.

An example for a RiHider tag could look like this:

```
RiHider hidden,depthfilter,s,midpoint
```

4.6.6 RiDisplay Tag

The tag type "RiDisplay" can be used to add more output files of different type (e.g. containing depth-buffer information) to the scene. 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. The name will be automatically changed to "+name" on RIB export if it does not already start with a plus.

An example for a RiDisplay tag could look like this:

```
RiDisplay imagez.tif,file,z
```

4.6.7 NoExport Tag

The tag type "NoExport" can be used to exclude certain objects from exported RIBs. The value string of this tag is ignored. All that counts is the presence of the tag. Child objects of objects with the "NoExport" tag will also be excluded from the RIB. Since Ayam 1.6, light objects also honour the "NoExport" tag. Note that regardless of potentially present "NoExport" tags, RIB archives will be created for all referenced objects all the time (even if "NoExport" tags are added to all instances).

4.6.8 SaveMainGeom Tag

The tag type "SaveMainGeom" can be used to save the geometry of the main window and the toolbox window (if open) with 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.

4.6.9 TP Tag

The tag type "TP" can be used to save tesselation parameters to objects of type "NPatch". 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>,<tparam>" where "<tmethod>" is an integer value between 1 and 3, describing which tesselation method to use (1 - ParametricError, 2 - PathLength, and 3 - DomainDistance) and "<tparam>" is a float value describing the respective parameter value for the chosen tesselation method.

TP tags may be easily created using the tesselation GUI, that can be started with the main menu entry "Tools/NPatch/Tesselate" (see also section 5.35 (The Tesselation Tool) (page 101)).

An example for a TP tag could look like this:

```
TP 1,0.5
```

4.6.10 DC Tag

The tag type "DC" is only used by the AyCSG CSG preview plugin to store the depth complexity of CSG primitives. The syntax of the DC tag is: "<dcval>" where "<dcval>" is a positive integer value describing the depth complexity of the CSG primitive. See also section 7.12 (CSG preview using the AyCSG plugin) (page 128) for more information regarding the depth complexity value.

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

```
DC 2
```

4.6.11 NP Tag

The tag type `"NP"` (new property) may be used to add new property GUIs to single objects. The value of the tag is the name of a new property. The necessary code to manage the property data and the windows that make up the property GUI itself have to be present in the Tcl context of Ayam before the user clicks on the new property in the property list box.

4.6.12 Internal Tags

The following tags are of no general use. They are used by Ayam internally only.

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.

The tag type `OI` is not meant to be used by the end user. Furthermore, changing the IDs manually avails to nothing as the tags are rebuilt before every export/save operation.

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.

The tag type `MI` is not meant to be used by the end user. Furthermore, changing the IDs manually avails to nothing as the tags are rebuilt before every export/save operation.

4.6.13 List of Known Tags

This section contains a comprehensive list of tag names, that are known in Ayam 1.8 and in the accompanying extensions.

```
"RiAttribute", "RiOption", "RiHider", "RiDisplay", "NoExport", "TC",
"PV", "SaveMainGeom", "TP", "MI", "OI", "DC", "NP", "IDR", "IIDR",
"RIDR", "R3IDR", "CIDR", "CCIDR"
```

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

4.7 Root Object

There is always exactly one Root object in the scene. This object is something special in that it cannot be deleted or copied. It holds options global to the scene like RiOptions, atmosphere and imager shaders. Furthermore, all currently open view windows are 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 global scene options are documented in the following sections.

4.7.1 RiOptions Property

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

- `"Width"`, `"Height"`, if greater than zero this value will be used for the image size instead of the corresponding dimension of the view window, but only for real RIB exports and not for the QuickRender and Render actions in view windows. QuickRender and Render 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 4.6.6 (RiDisplay Tag) (page 45)), otherwise your RIB will not contain a Display 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"`, `"FilterWidth"` size of the 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 pixels.
- `"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 RiStandard 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.

4.7.2 Imager, Atmosphere Property

The Imager and Atmosphere properties let you define shaders for the Root object, please refer to section 4.5 (Shaders Properties) (page 40) 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.

4.8 View Object

Every view window (see also section 2.3 (Anatomy of a View) (page 21)) 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 modeling actions in a view are limited to that input plane (unless the view is switched to local modeling; available since Ayam 1.4; see also section 3.9 (Editing in Local Space) (page 38)). The standard input planes are as following: Front - XY-plane, Side - ZY-plane, Top - XZ-plane, Trim - XY-plane.

In perspective views no interactive modeling actions are possible, but you may position the camera and pick objects.

Views of type `"Trim"` are very special. They are used to edit trimcurves of NURBPatch objects only. They display that trimcurves as normal NURBCurves when the current level is inside a NURBPatch. The extensions of the patch in parameter-space are drawn as a rectangle. The trimcurves 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 lighted by the light object (this works best with perspective views that have equal width and height).

Since Ayam 1.8 it is 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 accordingly (see the description of Drag-and-Drop with view objects above).

4.8.1 Camera Property

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

- `"From"` is the point where the camera (that is attached to the view) is situated.
- `"To"` is the point the camera is looking to.
- `"Up"` is the up vector of the camera.
- `"Near"` defines the near clipping plane. A value of 0.0 means a default value (that depends on the type of the view) should be used. 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.

4.8.2 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.
- `"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`>).
- `"Shade"` toggles shading of surfaces. Note that the lighting is in no way an exact (or even similar) representation of the light information you specified with Light objects! Instead, a single light source, located at the camera origin (a headlight), will be used!
- `"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 modeling actions.
- `"Local"` enables editing in local object space. See also section 3.9 (Editing in Local Space).
- `"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"`).

4.9 Camera Object

Camera objects are used to temporarily save camera settings of views. Therefore, they has just have two properties explained above, see sections 4.8.1 (Camera) (page 50) and 4.3 (Attributes Property) (page 40).

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.

4.10 Box Object

A solid box, centered at the origin of the object coordinate system. This object will always be exported as solid primitive in RIBs; consisting of six bilinear patches.

Since Ayam 1.8.2, a box object may be converted to three NURBS patches using the main menu entry `"Tools/Convert"`.

The following parameters further control the shape of a box:

4.10.1 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).

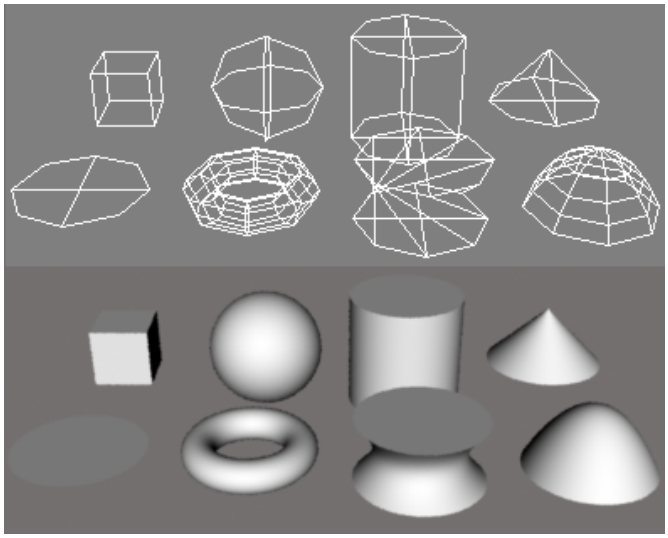

Figure 10: Box, and Quadric Primitives

4.11 Quadric Primitives

4.11.1 Sphere Object

A sphere, centered at the origin of the object coordinate system. This object will be exported as solid primitive or as simple sphere (depending on the `"Closed"` parameter of the SphereAttrib property) in RIBs.

Since Ayam 1.8.2, a sphere object may be converted to NURBS patches using the main menu entry `"Tools/Convert"`. This conversion obeys all parameters of the sphere.

The following parameters further control the shape of a sphere:

RiSphereAttr Property

- `"Closed"` toggles whether the object should be automatically sealed (closed) when exported to RIB. Note that if this option is in use and `"ZMin"`, `"ZMax"` or `"ThetaMax"` have other than the default values, a single sphere will be written (in the worst case) as a CSG hierarchy of two spheres, two cylinders and eight disks! But it may be used in CSG operations, safely.
- `"Radius"` is the radius of the sphere, default is 1.
- `"ZMin"` may be used to chop the sphere off at a certain place at Z.
- `"ZMax"` may be used to chop the off at a certain place at Z.
- `"ThetaMax"` is the sweeping angle of the sphere default is 360.

4.11.2 Disk Object

A disk, centered at the origin of the object coordinate system. This object will always be exported as simple disk in RIBs.

Since Ayam 1.8.2, 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.

The following parameters further control the shape of a disk:

RiDiskAttr 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, default is 360.

4.11.3 Cone Object

A cone, centered at the origin of the object coordinate system, with the base at the XY plane. This object will be exported as solid primitive or as simple cone (depending on the "Closed" parameter of the ConeAttrib property) in RIBs.

Since Ayam 1.8.2, a cone object may be converted to NURBS patches using the main menu entry "Tools/Convert". This conversion obeys all parameters of the cone.

The following parameters further control the shape of a cone:

RiConeAttr Property

- "Closed" toggles whether the object should be automatically sealed (closed) when exported to RIB. Note that if this option is in use and "ThetaMax" has a different than the default value, a single cone will be written (in the worst case) as a CSG hierarchy of a cone, a disk and two polygons! But it 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, default is 360.

4.11.4 Cylinder Object

A cylinder, centered at the origin of the object coordinate system. This object will be exported as solid primitive or as simple cylinder (depending on the "Closed" parameter of the RiCylinderAttr property) in RIBs. Note that the OpenGL representation of this object does not reflect the settings of the following parameters of the CylinderAttrib property: "Closed" and "ThetaMax".

Since Ayam 1.8.2, a cylinder object may be converted to NURBS patches using the main menu entry "Tools/Convert". This conversion obeys all parameters of the cylinder.

The following parameters further control the shape of a cylinder:

RiCylinderAttr Property

- `"Closed"` toggles whether the object should be automatically sealed (closed) when exported to RIB. Note that if this option is in use and `"ThetaMax"` has a different than the default value, a single cylinder will be written (in the worst case) as a CSG hierarchy of a cylinder, two disks and two polygons! But it 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, default is 360.

4.11.5 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. This object will be exported as solid primitive or as simple torus (depending on the `"Closed"` parameter of the RiTorusAttr property) in RIBs.

Since Ayam 1.8.2, a torus object may be converted to NURBS patches using the main menu entry `"Tools/Convert"`. This conversion obeys all parameters of the torus.

The following parameters further control the shape of a torus:

RiTorusAttr Property

- `"Closed"` toggles whether the object should be automatically sealed (closed) when exported to RIB. Note that if this option is in use and `"PhiMin"`, `"PhiMax"` or `"ThetaMax"` have different than the default values, a single torus will be written (in the worst case) as a CSG hierarchy of a torus, two disks and two hyperboloids! But it 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 -180.
- `"PhiMax"` determines an angle to limit the swept circle, default is 180.
- `"ThetaMax"` is the sweeping angle of the torus, default is 360.

4.11.6 Paraboloid Object

A paraboloid, centered at the origin of the object coordinate system. This object will be exported as solid primitive or as simple paraboloid (depending on the "Closed" parameter of the RiParaboloidAttr property) in RIBs.

Since Ayam 1.8.2, a paraboloid object may be converted to NURBS patches using the main menu entry "Tools/Convert". This conversion obeys all parameters of the paraboloid.

The following parameters further control the shape of a paraboloid:

RiParabAttr Property

- "Closed" toggles whether the object should be automatically sealed (closed) when exported to RIB. Note that if this option is in use and "ZMin", "ZMax" or "ThetaMax" have different than the default values, a single paraboloid will be written (in the worst case) as a CSG hierarchy of a paraboloid, two disks and two bicubic patches! But it may be used in CSG operations, safely.
- "RMax" is the radius of the paraboloid at a Z of "ZMax", 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 paraboloid, default is 360.

4.11.7 Hyperboloid Object

A hyperboloid, centered at the origin of the object coordinate system. This shape will be created by sweeping a line specified by two points in space around the Z axis. This object will be exported as solid primitive or as simple hyperboloid (depending on the "Closed" parameter of the RiHyperbAttr property) in RIBs.

Since Ayam 1.8.2, a hyperboloid object may be converted to NURBS patches using the main menu entry "Tools/Convert". This conversion obeys all parameters of the hyperboloid.

The following parameters further control the shape of a hyperboloid:

RiHyperbAttr Property

- "Closed" toggles whether the object should be automatically sealed (closed) when exported to RIB. Note that due to a bug in BMRT that is still present in V2.3.6 this 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). Also note that if this option is in use and "ThetaMax" has a different than the default value, a single hyperboloid will be written (in the worst case) as a CSG hierarchy of a hyperboloid, two disks and two bilinear patches!
- "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, default is 360.

4.12 Level Object

Level objects may be used to build object hierarchies and perform CSG operations. Note that child objects of a level inherit the levels transformations 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, you will finally arrive at 15,0,0.

4.12.1 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 cut parts away. Since Ayam 1.8 there is a plugin available, that is able to preview the CSG operations, see also section 7.12 (CSG preview using the AyCSG plugin) (page 128).

New solid primitives may be created with levels of type "Primitive". Note that Ayam is not able to check, whether your new primitive obeys the rule of total closeness.

4.13 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 object.

When geometric objects are connected to a material object this material object may not be deleted.

4.13.1 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.

4.13.2 Surface, Displacement, Interior, Exterior Property

These properties let you define shaders for the material object, please refer to section 4.5 (Shaders Properties) (page 40) 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.

4.13.3 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.

4.14 Light Object

In contrast to the light sources as defined in the RenderMan interface, Ayam light sources are always global. This means, regardless of the place of a light source in the scene hierarchy, it will always light all other objects.

The behaviour of a light source object depends heavily on the type of the light source. There are four different types available: `"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 lightsource. You should not use these names for other things in your 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 7.15 (references) (page 131)).

4.14.1 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 but only after you use 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) or all objects in the scene. The default is off, all objects in the scene are lighted! The `"IsLocal"` attribute is ignored for lights that are defined in the root level of the scene. Note that shadow maps will always contain shadows from all objects in the scene, regardless of local lights.

Using the light attribute `"Shadows"` you may determine whether the lightsource should cast shadows. The default is off, no shadows!

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 `"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 34)).
- "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 light sources.

In order to ease the parameterisation of spot lights, you may drop the light source object on to a view object (preferably one with a perspective viewing transformation and with equal width and height) to see what objects of the scene are actually lighted by the light object.

4.14.2 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.

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. 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 be rendered separately. To help you picking the right frame number for the image (e.g. to re-render just the image), 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 scene from one system to another more easily.

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 entry "View/Create ShadowMaps" or the main menu

entry "Special/RIB-Export/Create ShadowMaps". The manual approach has the advantage, that the shadow maps will not be re-created each time you render the scene.

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 parameterized 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 parameterizing it accordingly. See the discussion above for the names of the shadow map files. Those file names, 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 a "distantlight" shader attached, even though Ayam is able to create the necessary shadow maps. The "distantlight" shader just makes no use of them. You have to manually switch to a shader that makes use of the shadow maps ("shadowdistant" in this case) to get shadows.

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

1. Go to the preferences (section "RIB-Export") and set "ShadowMaps" to "Automatic".
2. Create two boxes.
3. Open the "Transformations" property of the second box.
4. Translate it by X: 0.0, Y: -1.0, Z: 0.0.
5. Scale it by X: 4.0, Y:1.0, Z:4.0.
6. Create a light source.
7. Open the "LightAttr" property.

8. Change the type to "Spot". Press "Apply".
9. Now change the parameters of the spot light to "IsOn": Yes, "Intensity": 18.0, "UseSM": Yes, "ConeAngle": 45.0, "BeamDistrib": 3.0, "From": -2, 2, 2, "To": 1, 0, -1; leave all other parameters at their default values.
10. Create a new view and make it perspective (Menu: "Type/Perspective").
11. Export a RIB from that perspective view (Menu: "View/Export RIB").
12. Render the RIB with a RenderMan compliant renderer, that uses shadow maps, e.g. Photorealistic RenderMan (prman) or Aqsis.

This scene is distributed with Ayam as an example scene named "shadowmaps.ay".

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.

Do not render directly from a view window if your renderer does 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. This command line option may inhibit writing of the shadow maps, so that the resulting image will look wrong.

4.14.3 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 naturally. 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 of the lightsource. 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 object to your hearts content; the position and size of the object determines the position and size of the lightsource!

There is an example scene named "arealight.ay" distributed with Ayam.

4.15 NURBCurve Object

NURBS curves are used to build more complex smoothly shaped objects using operations like extrude, revolve, sweep or skin. They can be closed and used to emulate Bezier and B-Spline curves easily.

4.15.1 Multiple Points

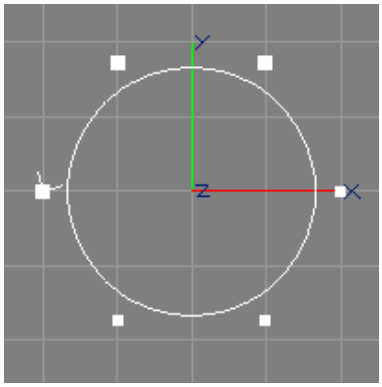

Figure 11: A NURBS Curve with Multiple Points (big handles)

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. Modeling 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 5.26 (The Collapse Tool) (page 99) and 5.27 (The Explode Tool) (page 99) for more information regarding those tools). Note that 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 NURBCurveAttrib property if they still have identical coordinate values. In other words, you should immediately edit the control points after exploding! You may also totally inhibit creation of multiple points for a NURBS curve using the attribute `"CreateMP"`.

4.15.2 NCurveAttrib Property

The first section of the NCurveAttrib property contains curve specific settings:

- `"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 and Custom knot sequences. If the knot type is not Custom, the next setting `"Knots"` will be ignored. Instead, knots of type NURB, Bezier or B-Spline will be generated. How do the different knot types affect the curve?

The knot type NURBS will generate knot values from 0.0 to 1.0, where the multiplicity of the knots at the ends will be of order of the curve. This guarantees that the curve will touch the control points at the ends of the curve.

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 it's ends and so on.

The knot type B-Spline will generate values without any multiple knots. The resulting curve looks and behaves like a B-Spline curve.

- `"Knots"` lets you enter your own custom knot sequences. Note that `"Knots"` are not in use if `"Knot-Type"` is of type NURB, B-Spline or Bezier!
- `"Closed"` toggles the closeness of the curve. If this is enabled, 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). All those points will be multiple points and single point editing actions will edit both points from now on.Note that for a cubic spline (order 4) you will need atleast 6 control points to close it. It is important to know, that the multiple points alone can not guarantee that the curve is closed if the knot type of the curve is Custom, if you really want a closed curve switch to type B-Spline.

  Also note that a NURBS circle as created by the NURBCircle tool is not a closed curve following this definition, it is nevertheless a closed curve.
- `"CreateMP"` toggles, whether multiple points should be created for this curve. See also the discussion in section 4.15.1 (Multiple Points) (page 62).

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. 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!

4.16 NURBPatch Object

4.16.1 NPatchAttrib Property

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

- "Width" and "Height" control the dimensions of the patch.
- "Order_U" and "Order_V" set the orders of the patch.
- "Knot-Type" and "Knots": For a discussion of the "Knot-Type" and "Knots" parameters, please see section 4.15.2 (NCurveAttrib) (page 62).

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. 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/DisplayMode" should be used instead.

4.17 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 control point of a trim curve must be identical to the first control point.
- Trim loops (multiple trim curves that form loops) are possible too; the last control point of each curve in the loop must be identical to the first control point of the next curve in the loop and the first control point of the first curve of a loop must be identical to the last control point of the last curve of that loop.
- To mark a set of curves to be a loop, they must be simply placed in a level. The order of the curves in this level is the order of the loop.
- Drawing trimmed NURBS patches with certain implementations of OpenGL may require a special trim curve (a 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 with trimmed NURBS patches with curves that do not follow the aforementioned rules! When in doubt or while heavy modeling, switch to wireframe drawing and switch off shading temporarily and you will be on the safe side.

Since Ayam 1.5 also NURBS curve providing objects are supported as trim curves.

4.18 BPatch Object

A BPatch is a bilinear patch (as it is used to build box objects, see 4.10 (Box Object) (page 51)).

4.18.1 BPatchAttrib Property

The BPatchAttrib property allows you to set 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.

4.19 PatchMesh Object

The PatchMesh object may be used to model with bilinear and bicubic patch meshes. The PatchMesh object may be converted to a NURBS patch representation. 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). The 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.

4.19.1 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.

4.20 PolyMesh Object

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

There are no special modeling 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 that, Ayam uses the tesselation 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.

4.20.1 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.

4.21 SDMesh Object

The SDMesh object may be used to include objects that have been modeled using the subdivision modeling paradigm in Ayam scenes.

There are no special modeling 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.

Furthermore, Ayam is currently 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.

4.21.1 SDMeshAttr Property

The SDMeshAttr GUI just displays some information about the SDMesh object:

- `"Scheme"`, subdivision scheme, currently only 0 (Catmull-Clark).
- `"NFaces"`, the number of faces.
- `"NControls"`, the total number of control points defined.

4.22 Instance Object

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 (master). A link can be placed anywhere on the file system, an instance object can be placed anywhere in the hierarchy, and additionally, it can be transformed (otherwise it would be pretty useless).

The sole purpose of instance objects is to save storage. The amount of saved diskspace can be very high, but this depends heavily on the scene. If there are no similar objects in the scene you can hardly use 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! You may, however, put some instances into a level object and create instances of this level (this is sometimes called hierarchical instancing). But you may not put instances of a level into the very same level (this would be recursive instancing, which is not supported by Ayam).
- The original 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 you cannot delete an object, and the error message tells you something about a reference counter, then you were about to violate the second rule. Clean the clipboard using the menu `"Special/Clipboard/Paste (Move)"` and delete or resolve all references.

Note that 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 master object. However, it is possible to move instances using Drag-and-Drop in the tree view or using the clipboard with `"Edit/Cut"` and then `"Special/Clipboard/Paste (Move)"`.

You can resolve an instance object at any time using the converter registered for objects of type `Instance` (simply select the instance object and use the menu entry `"Tools/Convert"`). To resolve all instance objects in a scene to normal objects, you may use the main menu entry: `"Special/Instances/Resolve all Instances"`.

The RIB export of instances does not use the RiInstance facility of the RenderMan interface, but the ReadArchive mechanism. This means, every original 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. You can change that behaviour using the preference setting `"ResInstances"`. If `"ResInstances"` is enabled, all instances will be resolved (temporarily) before being exported to RIB.

Ayam can also create instances automatically (see section 7.10 (Automatic Instancing) (page 127)).

To easily find the master object of an instance, just select the instance, then use the main menu entry: `"Edit/Master"`.

4.23 Clone Object

The Clone object allows you to easily create and control a number of instances of a single object. The instances will be created internally and transformed, each by a certain amount. The original object is the first child object of the Clone object. If a second object is present as child of the Clone object it is treated as trajectory (or path) curve, similar to the Sweep object (see section 4.26 (Sweep Object) (page 73)).

Thus, the object hierarchy of a Clone object may look like this:

```
+-Clone
 | Cloned-Object
 \ [Trajectory(NCurve)]
```

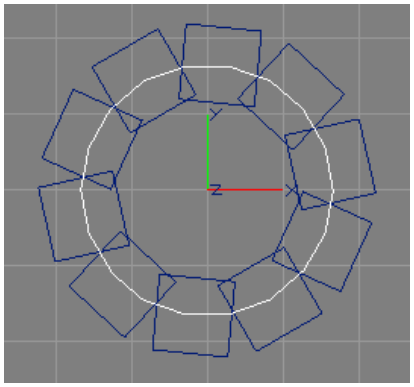

Figure 12: Clone Object with Trajectory Curve (white)

If you use a trajectory curve to place the clones, you may want to hide the parameter object and add a `"NoExport"` tag to it. Otherwise the original object will appear twice, on the trajectory and on its normal position. Note that the transformation attributes of the parameter object will be completely overwritten when placing the clones on the trajectory curve. If the parameter object has distinct scale or rotation attributes it should be placed 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. 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 parameter object it should be placed in a level (see above) and the `"NoExport"` tag should added to the level object (as you can not add tags to instance objects).

The following attributes further control the clone process:

4.23.1 CloneAttr Property

- `"NumClones"` 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.
- `"Mirror"` allows to choose between three different mirror modes. If mirroring is enabled, for each child of the Clone object, a mirrored counterpart will be created and all other parameters of the Clone object will be ignored.
- `"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. They are not used, if a trajectory curve is present.

4.24 Revolve Object

Figure 13: 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 of the coordinate system that is defined by the next higher level in the object hierarchy (the Y axis of the revolve object itself). The generating curve should lie in the XY plane of this coordinate system. If not, it will be squashed down to this plane!

The following simple experiment should make the last statements more clear, during all steps watch the movements of the revolution:

- Create a NURBCurve. Select it.
- Create a Revolve custom object using the menu entry (`Tools/NURBCurve/Revolve`).
- Select the Revolve object, and rotate it around Z. (The axis of the revolution changes. The generating NURBS curve, as child object, will also be rotated.)
- Now enter the Revolve object, select the child curve and edit the control points (Note how the Revolution changes).
- Rotate the curve around Z (Note how the Revolution changes).
- Switch to a Side view, edit the generating curve here in Z direction only (Revolution does not change!).

You may convert the current surface of revolution and the caps, if there are any, to ordinary NURBS patches using the main menu entry `"Tools/Convert"`.

4.24.1 RevolveAttr Property

Using the parameter `"ThetaMax"` you can specify the sweeping angle of the revolution just like with an ordinary RenderMan quadric.

Since Ayam 1.8 the Revolve object supports a B-Spline mode, that may be enabled by setting the new parameter `"Sections"` to a value higher than 2. 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 choosen, 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. 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 new parameter `"Order"`. If `"Order"` is 0, a standard value of 3 will be used. Note that the B-Spline mode is currently only available for full revolutions (`"ThetaMax"` should be 360.0).

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 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 trimcurves”), try to revert the revolved curve.

See section 4.16.1 (NPatchAttrib) (page 63) for a description of the other two attributes `"DisplayMode"` and `"Tolerance"`.

4.25 Extrude Object

Figure 14: Extrude Object (left: Curve, middle: normal Extrusion, right: Extrusion with Caps)

The extrude object forms an extrusion from a number of 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 and watches them and adapts to them automagically.

Consequently, the object hierarchy of an Extrude object may look like this:

```
+-Extrude
 | Outline(NCurve)
 | [Hole1(NCurve)]
 \ [Hole2(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 of the coordinate system that is defined by the next higher level in the object hierarchy (the Z axis of the extrude object itself). The generating curves should lie in the XY plane of this coordinate system. If not, they will be squashed down to this plane!

You may convert the current surface of extrusion and the caps and bevels, if there are any, to ordinary NURBS patches using the main menu entry `"Tools/Convert"`.

4.25.1 ExtrudeAttr Property

Using the parameter `"Height"` you determine how big in Z direction the extrusion should be. Note that the height of the bevels will not be taken into account here, if you have an extrusion with height 1.0 and you switch on beveling (upper and lower) with radius 0.1 you end up with an object that is 1.2 whatever big in Z direction.

The extrude object can automatically generate caps, that are trimmed NURBS patches. Using `"UpperCap"` and `"LowerCap"` 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 your curves conform to this criteria. Ayam, however, detects the correct orientation of the curves (and reverts them if necessary).

Using `"LowerBevel"` and `"UpperBevel"` you determine whether bevels should be created to round the otherwise sharp corners of the extrusion with the cap. The bevels may be controlled using `"BevelType"` and `"BevelRadius"`. `"BevelType"` allows to choose from `"Round"` (a complete round bevel, built using a quarter circle), `"Linear"` (not round at all, but just a connecting linear patch) and `"Ridge"` which forms a more complex beveling similar to some frames for (real world) images. The `"BevelRadius"` determines the size of the bevels. It is expressed in untransformed object coordinates of the generating curve, but the resulting radius of the bevel may differ a bit (depending on the shape of the curve).

See section 4.16.1 (NPatchAttrib) (page 63) for a description of the other two attributes `"DisplayMode"` and `"Tolerance"`.

4.25.2 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 you cannot have hole curves inside hole curves. Ayam will not check whether your curves conform to these criteria!

With the direction of the curve you decide 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. You should always use closed curves for beveling! Beveling may lead to self intersecting trimcurves 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.

4.26 Sweep Object

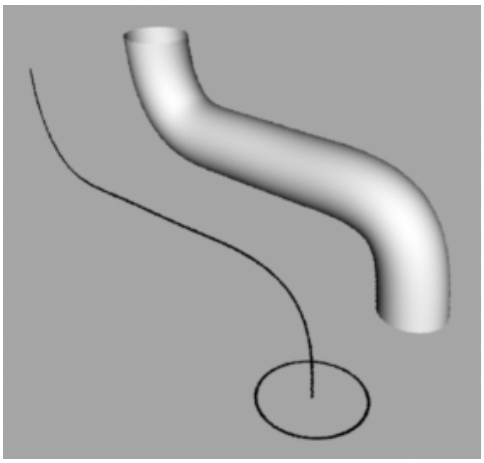

Figure 15: Sweep Object (left: Curves, right: Resulting Swept Surface)

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.

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, may look like this:

```
+-Sweep
 | Cross_Section(NCurve)
 | Path(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- and Z-direction. This implies, that a scaling function that does nothing should e.g. be linear from (0,1,0) to (1,1,0).

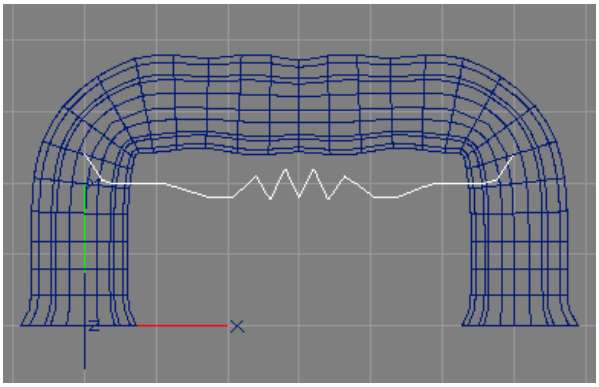

Figure 16: Sweep Object with Scaling Function (white)

Here is a short example for the creation of a sweep:

- Create a circular BSpline using the toolbox. (This will be our cross section.)
- Rotate it by 90 degrees around the Y-axis. (Use the `"Transformations"` property for that.)
- Create a simple NURBS curve using the toolbox. (This will be our trajectory.)
- Select both curves. (Select the first curve, hold down the `"Shift"` key and select the other curve.)
- Create the Sweep object using the toolbox.
- Now you may enter the Sweep object and modify e.g. the second curve, the trajectory. (Press `"e"`, then drag some control points around.)
- To modify the cross section you would need to switch to a view of type `"Side"`. (Use the <`Ctrl+s`> shortcut while the view has the input focus.)

Section 6.3.3 (Easy Sweep) (page 116) has an example script that automates creation and parameterisation of a suitable cross section curve.

You may convert the current sweep and the caps, if there are any, to ordinary NURBS patches using the main menu entry `"Tools/Convert"`.

4.26.1 SweepAttr Property

If `"Interpolation"` is enabled, all section curves will be interpolated by the swept surface.

The second parameter `"Sections"` determines how many sections (in u direction) should be used, when generating the sweep NURBS patch. The NURBS patch always has sections+1 control points in u direction.

If `"Rotate"` is enabled, the cross sections will be rotated so that they are always perpendicular to the trajectory, this is the 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).

See section 4.16.1 (NPatchAttrib) (page 63) for a description of the other two attributes `"DisplayMode"` and `"Tolerance"`.

4.27 Birail1 Object

Figure 17: 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.

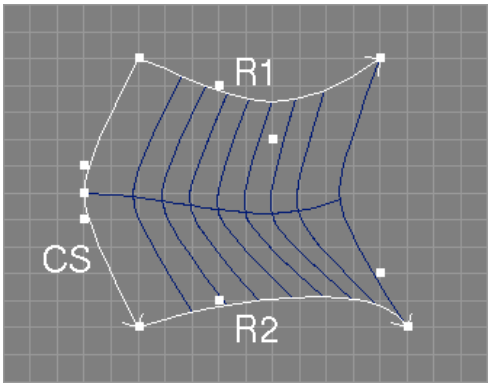

Figure 18: Valid Configuration of Parameter Curves (white) for Birail1 (blue)

The image above shows a valid configuration of parameter curves for the Birail1 object. Mind the direction of the rail curves (R1 and R1) 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 choosen.

The Birail1 object watches the child objects and adapts to them automatically via the notification mechanism.

You may convert the current birailed surface and the caps, if there are any, to ordinary NURBS patches using the main menu entry `"Tools/Convert"`.

The following parameters further control the birailing process:

4.27.1 Birail1Attr 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.

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).

See section 4.16.1 (NPatchAttrib) (page 63) for a description of the other two attributes `"DisplayMode"` and `"Tolerance"`.

4.28 Birail2 Object

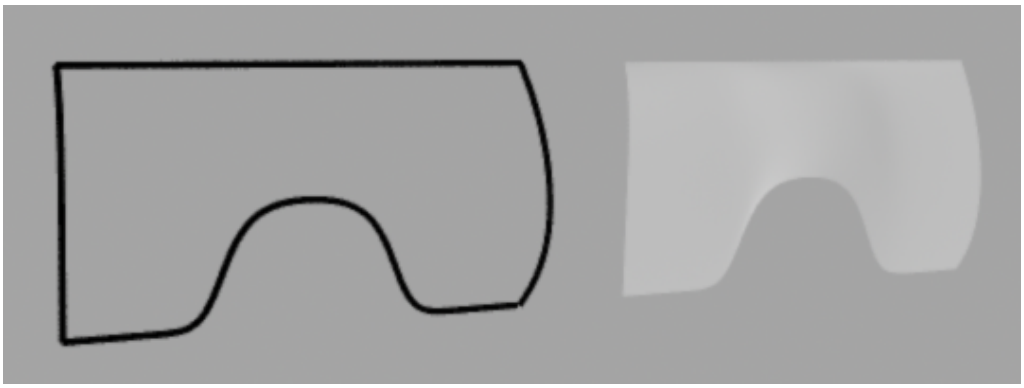

Figure 19: Birail2 Object (left: Curves, right: Resulting Swept Surface)

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.

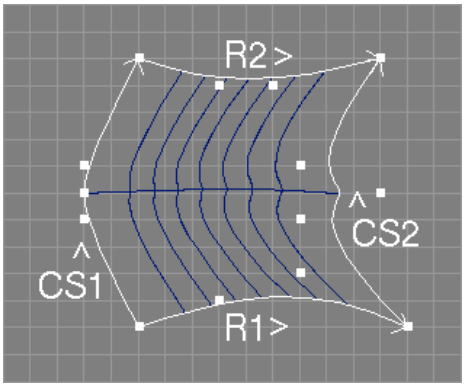

Figure 20: Valid Configuration of Parameter Curves (white) for Birail2 (blue)

The image above shows a valid configuration of parameter curves for the Birail2 object. Mind the direction of the rail curves (R1 and R1) 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 choosen.

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, will control 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 mimic 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. Those will always exactly match the parameter curves.

The Birail2 object watches the child objects and adapts to them automatically via the notification mechanism.

You may convert the current birailed surface and the caps, if there are any, to ordinary NURBS patches using the main menu entry `"Tools/Convert"`.

The following parameters further control the birailing process:

4.28.1 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.

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).

See section 4.16.1 (NPatchAttrib) (page 63) for a description of the other two attributes `"DisplayMode"` and `"Tolerance"`.

4.29 Skin Object

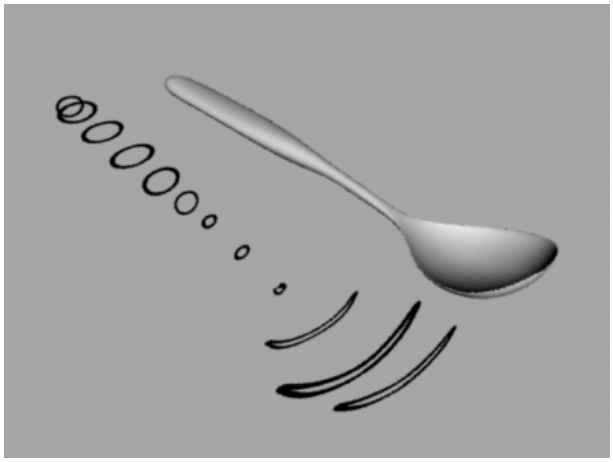

Figure 21: Skin Object (left: Curves, right: Resulting Skinned Surface)

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
 | C1(NCurve)
 | C2(NCurve)
 | [...
 \ Cn(NCurve)]
```

Note that in contrast to the build from curves tool, the curves may be of arbitrary length and order. You may e.g. easily skin 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. 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.

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. 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.

You may convert the current skin and the caps, if there are any, to ordinary NURBS patches using the main menu entry `"Tools/Convert"`.

4.29.1 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 lower than the number of curves used. If the specified value is lower 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 skin. 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 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.

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

See section 4.16.1 (NPatchAttrib) (page 63) for a description of the other two attributes `"DisplayMode"` and `"Tolerance"`.

4.30 Gordon Object

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 order and the direction of the curves.

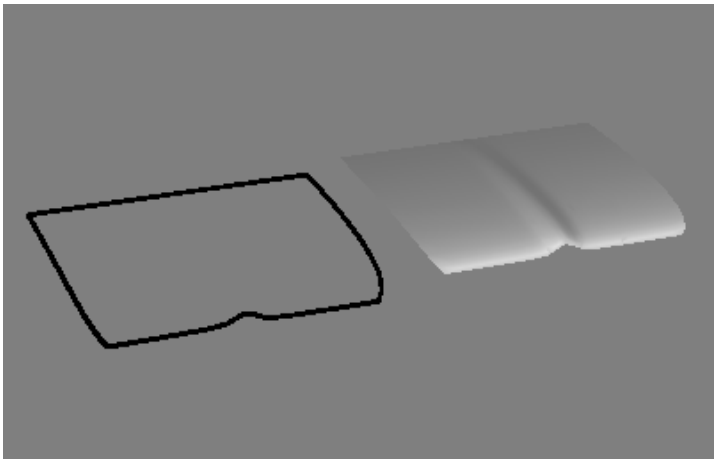

Figure 22: Gordon Object (left: Curves, right: Resulting Gordon Surface)

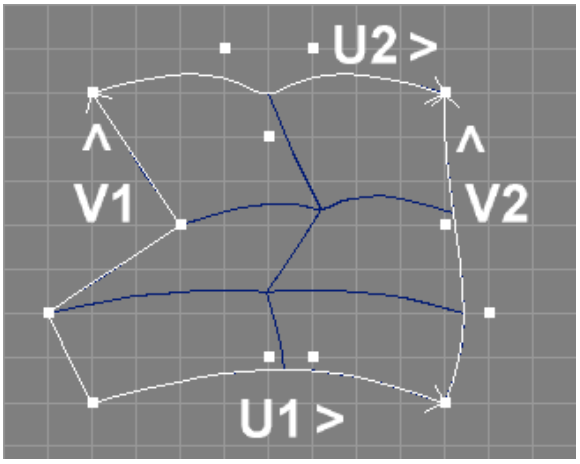

Figure 23: Gordon Surface with Parameter Curves (white)

The curves may be of arbitrary length and order. You 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 only non-rational curves can be used as parameter curves for a Gordon surface. If the parameter curves have weights, the weight information of the curves will simply be ignored.

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 you experience 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 you 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. You may want to add a `"NoExport"` tag to this patch. The object hierarchy of a Gordon object using such a patch, consequently looks like this:

```
+-Gordon
 | U1(NCurve)
 | U2(NCurve)
 | Level
 | V1(NCurve)
 | V2(NCurve)
 | Level
 \ Intersections(NPatch)
```

The complete template for the Gordon object hierarchy, consequently, looks like this:

```
+-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.

You may convert the current Gordon surface to an ordinary NURBS patch using the main menu entry `"Tools/Convert"`.

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

4.30.1 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 4.16.1 (NPatchAttrib) (page 63) for a description of the other two attributes `"DisplayMode"` and `"Tolerance"`.

4.31 Cap Object

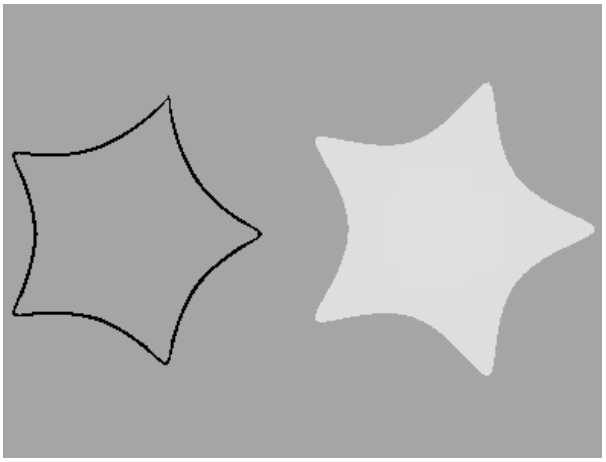

Figure 24: Cap Object (Curve, Resulting Cap Surface)

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 (see also section 4.25.2 (Using Holes and Bevels) (page 72)).

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 the curves have to be planar and defined in the XY plane. 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.

You may convert the current cap to an ordinary NURBS patch object using the main menu entry `"Tools/Convert"`.

4.31.1 CapAttr Property

See section 4.16.1 (NPatchAttrib) (page 63) for a description of the two attributes `"DisplayMode"` and `"Tolerance"` of the `"CapAttr"` property.

4.32 ICurve Object

The ICurve object creates an interpolating spline from n points in space. The curve is either a C2 cubic curve with n+2 control points or a global interpolating rational (Global4D) curve with n control points and with arbitrary order.

The global interpolation generates a bit smoother curves which look best with an order of 3, your mileage may vary however.

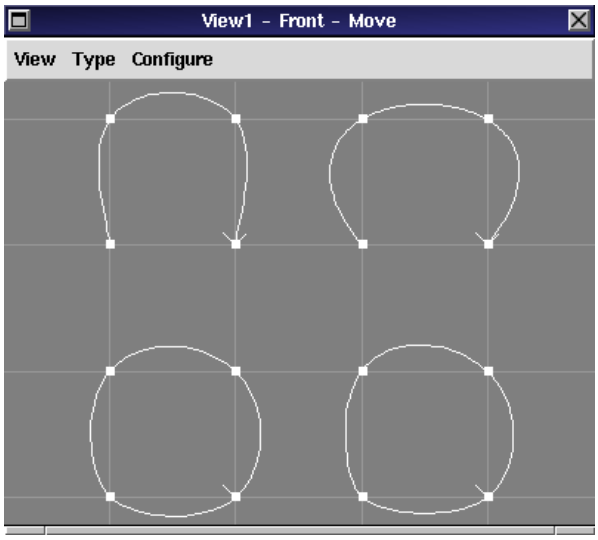

Figure 25: Different ICurves

The image above shows some interpolating curves, the left ones are C2 cubic, the right ones Global4D, the upper open, and the lower closed ones. The interpolation fidelity for the closed curves could be increased considerably by adjusting the `"IParam"` parameter, see below.

In both interpolation modes chord length parameterisation will be used to determine the knot vector of the interpolating curve.

This object makes use of the provide mechanism. It marks itself as providing a NURBCurve (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 NURBCurve.

You may convert the current ICurve to an ordinary NURBS curve using the main menu entry `"Tools/Convert"`.

4.32.1 ICurveAttr Property

- Using `"Length"` you determine the number of points to interpolate.
- The curve can be closed with the parameter `"Closed"`.
- The parameter `"Mode"` determines whether the curve should be a C2 cubic curve (with n+2 control points (n+3 if the curve is closed)) or a so called global interpolating curve (with n control points (n+3 if the curve is closed) and arbitrary order).
- The next parameter `"Order"` is used only if the mode is Global4D. It determines the order of the interpolating curve. If the specified order is bigger than the number of control points used by the interpolating curve, then the order is silently changed to match the number of control points.
- The parameter `"IParam"` is used to control the position of the second and last-1 control point of the interpolating curve. It scales the vectors used to position the aforementioned points. You can try to change (decrease) this value, if you are not satisfied with the shape of the curve near the first or last interpolated point. This parameter has no effect on open curves when the mode is Global4D.
- See section 4.15.2 (NCurveAttrib) (page 62) for a description of the last attribute: `"Tolerance"`.

4.33 ConcatNC Object

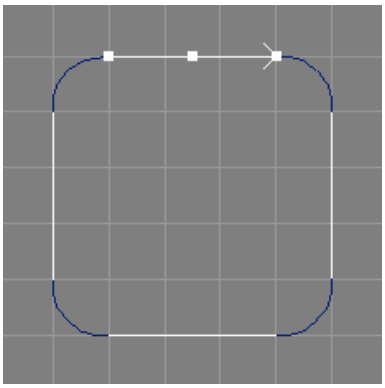

Figure 26: ConcatNC Object from a Linear Curve and 3 Instances (white)

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.

How does the concatenation process work? First, the orders of all child curves will be elevated to the maximum order of all the child curves (see section 5.18 (elevate tool) (page 96) for more information on elevation) and all curves will be clamped (see section 5.20 (clamp tool) (page 97) for more information on clamping). If `"FillGaps"` is enabled (see below), fillet curves will be created for every gap between the child curves of the ConcatNC object. If `"Closed"` and `"FillGaps"` are enabled an additional fillet is created to close the curve. Then, the control points of all child curves and fillets are simply copied into a new big control point vector, without checking for double points. Attributes like display mode and tolerance for the new concatenated curve are taken from the first child object. 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 child curves if any of the child curves has a custom knot vector with non equidistant knots. If `"Knot-Type"` is `"Custom"`, the shape of the child curves will be preserved.

You may convert the current ConcatNC object to an ordinary NURBS curve object using the main menu entry `"Tools/Convert"`.

The following attributes further control the concatenation process:

4.33.1 ConcatNCAttr Property

- Using `"Closed"` you may create a closed concatenated curve. If `"FillGaps"` (below) is enabled, an additional fillet will be created for the last and the first child curve to close the concatenated curve. If `"FillGaps"` (below) is not enabled, the concatenated curve will be closed with the same algorithm that is also used by the close curve tool (possibly changing the shape again!).
- `"FillGaps"`, creates fillet curves for all gaps between the child curves of the ConcatNC object. The fillet curves will be cubic Bezier curves. The direction of the tangents in the endpoints of the fillets and the gap enclosing curves will match, so that the transition should be G1 continuous.
- `"Revert"` the orientation of the concatenated curve will be reversed.
- `"FTLength"` determines a scale factor for the tangent vectors of the fillets. A value of 1.0 leads to vectors that are as long as the matching vectors in the control point arrays of the original curves. A value of 0.3 is the default value. You may need to tweak this, if you experience discontinuities in the transitions between original curves and fillets, especially, if `"Knot-Type"` (see below) is set to `"Custom"`.
- `"Knot-Type"` in fact toggles between two different modes of concatenation. 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 shape of the original curves. If `"Knot-Type"` is `"Custom"`, the knot vector is composed from the knot vectors of the original curves, and thus, their shape may be preserved completely.

4.34 ExtrNC Object

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.

The extraction process is controlled by the following attributes:

4.34.1 ExtrNCAttr Property

- `"Side"` controls, which curve should be extracted. Available values are `"U0"`, `"Un"`: extract boundary curve along width; `"V0"`, `"Vn"`: extract boundary curve along height; `"U"`, `"V"` extract curve along width and height respectively at specified parametric value (see below). The options `"U"`, `"V"` are not implemented in Ayam 1.8!
- `"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.

See section 4.16.1 (NPatchAttrib) (page 63) for a description of the other two attributes `"DisplayMode"` and `"Tolerance"`.

4.35 Text Object

Figure 27: 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 control distances between certain letters (You are not going to typeset a book with Ayam, aren't you?). Therefore you might experience wrong letter distances from time to time. If this happens, just create a Text object for each letter, and arrange the objects as you like.

The following attributes control the creation of the text objects.

4.35.1 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 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", "UpperBevel", "LowerBevel", "BevelType", and "BevelRadius" work like for the Extrude object (see section 4.25.1 (ExtrudeAttr Property) (page 72) for a more exhaustive description of those parameters). Just one note: for some fonts, the "BevelRadius" has to be set to really small values (about 0.0008) to get bevels and caps. This is because of sharp corners in some letters that lead to overlapping curves with high values for the "BevelRadius".
- "RevertBevels" can be turned on, if the extracted curves and therefore also the bevels are of wrong direction.

See section 4.16.1 (NPatchAttrib) (page 63) for a description of the other two attributes "DisplayMode" and "Tolerance".

4.36 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.

4.36.1 RiIncAttr Property

- Using "File" you specify the file 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.

4.37 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.

Theoretically, the scripts can use any functionality from Tcl and the Tcl scripting interface of Ayam (see also section 6 (The Tcl Scripting Interface) (page 101)). However, certain script object types may impose special constraints.

For security reasons, if scene files containing script objects are loaded, Ayam will raise a warning offering to disable all script objects that will be read.

The script will be run each time it is modified and each time the notification callback of the object is called (e.g. because one of the children of the script object changed). As long as the script of a script

object is executed, Ayam will not process any events except for checking whether the script emergency hotkey <Ctrl+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, sL, selOb, plb_update, cutOb, copOb, delOb, undo.

Since Ayam 1.8.2 script objects may also create their own property GUIs for e.g. script parameters. This is 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.

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.

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

4.37.1 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 a single new object. After running the script, the new created object will automatically be copied 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 NURBCurve object, the script object may be used as parameter object of a tool object that needs a NURBCurve, e.g. a Sweep:

```
+-Sweep
 | Cross_Section(Script)
 \ Path(NCurve)
```

    The newly created object has to be selected by the script code for parameterisation and copying. The selection should be done using the new scripting interface command "hSL" (hidden select last).
  - "Modify", if the script object has child objects, these child objects will be copied into the internal data structure of the script object. A selection of the copied 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). Note that the original child objects will not be modified. 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 NURBCurve object as first child, the script object may be used as parameter object of a tool object that needs a NURBCurve, e.g. a Sweep:

```
+-Sweep
 +-Cross_Section(Script)
 | \ NCurve
 \ Path(NCurve)
```

- `"Script"` is the script code. The 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 clipboard and the context menu entry `"Paste (Replace)"`.

4.38 Custom Objects

What is a custom object?

Think of it as a plugin that extends Ayam's capabilities by defining totally new types of e.g. geometric objects. A simple example is the CSphere custom object, which implements a sphere and has a new property named `"CSphereAttr"`. This property contains all parameters of a simple RenderMan Interface quadric sphere.

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

4.39 Metaball Object

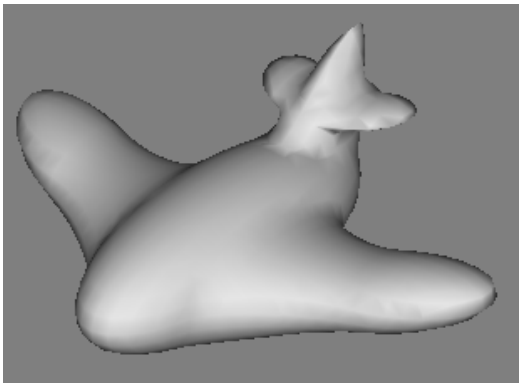

Figure 28: A Metaball Object from Six Meta Components

A metaball object is a custom object (see also section 4.38 (Custom Object) (page 89)). 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 Custom"` 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)]
```

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 parameterized using the "MetaObjAttr" property:

4.39.1 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 ressources.

While modelling with meta balls you may add other "MetaComp" objects to the "MetaObj" object and parameterize them. A "MetaComp" object has the following properties.

4.39.2 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:

4.39.3 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.

4.39.4 Torus

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

4.39.5 Cube

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

4.39.6 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.

5 NURBS Modeling Tools

This section describes NURBS curve and surface related modeling tools, which you can find in the "Tools" menu of the main window.

5.1 The Closed BSpline Tool

- Arguments: Number of control points (i).

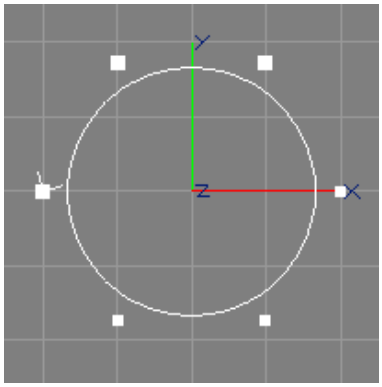

Figure 29: A Closed B-Spline

- Operation: This tool creates a closed cubic B-Spline curve with i+3 control points in the XY plane. The control points are arranged in a circle of radius 1 centered around the origin. This gives the curve a circular appearance (see image above) but it is not a true circle: If few control points are used, the radius of the circular curve is clearly smaller than 1. Furthermore, the parameterisation and curvature of the closed B-Spline are not exactly as one would expect from a circle. However, you can use the NURBCircle tool (see below) to create a true circle instead.
- Note: i must be atleast 3. The first three control points of the new curve will be identical to the last three. Additionally, the 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 4.15.1 (Multiple Points) (page 62).

5.2 The NURBCircle Tool

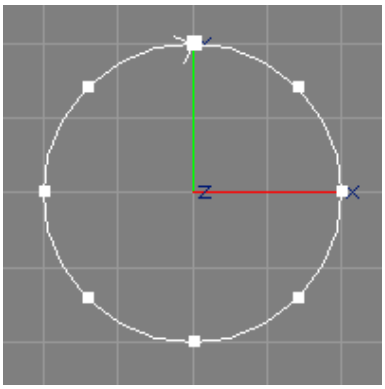

Figure 30: A NURBS Circle

- Arguments: None.
- Operation: The NURBCircle tool creates a full circular NURBS curve of radius 1 in the XY plane located at the origin.
- 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 Closed BSpline Tool (see above) if you want to edit the curve further. You can use this circle to easily create a NURBS-torus by moving the circle along X and then revolving it. The amount of movement determines the radius of the torus, whereas the radius of the circle determines the thickness.

5.3 The NURBCircleArc Tool

- Arguments: angle of the arc in degrees
- Operation: The NURBCircle tool creates a circular NURBS curve arc of radius 1, with the specified angle, in the XY plane, located at the origin.
- Note: The NURBS curve created by the NURBCircleArc 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).

5.4 The TrimRect Tool

- Arguments: None
- Operation: The TrimRect tool creates a 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. See section 4.17 (Trim Curves) (page 64) for a more detailed discussion of trim curves and how to use this rectangular curve.

5.5 The 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 The NURBSphere2 Tool

- Arguments: None.
- Operation: The NURBSphere tool creates a Cobb-NURBSphere, consisting of 6 NURBPatches.
- Note: The NURBPatches are of high order (5).

5.7 The 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.
- Note: See section 4.24 (Revolve Object) (page 70) 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 The 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.
- Note: See section 4.25 (Extrude Object) (page 71) 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 The 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.
- Note: See section 4.26 (Sweep Object) (page 73) 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 The 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.
- Note: See section 4.31 (Cap Object) (page 82) 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 The 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.
- Note: See section 4.27 (Birail1 Object) (page 75) 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 The 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.
- Note: See section 4.28 (Birail2 Object) (page 76) 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 The 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.
- Note: See section 4.30 (Gordon Object) (page 79) 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 The 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.
- Note: See section 4.29 (Skin Object) (page 78) 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 The Revert Tool

- Arguments: The revert tool takes all NURBS curves and ICurves from the selection.
- Operation: The direction of the selected NURBS curves will be reversed.
- Note: The direction of a NURBCurve is shown as a small arrow at the end of the curve.

5.16 The 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.
- 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. See also section 4.33 (ConcatNC Object) (page 84). The original NURBS curves will not be deleted by this tool.

5.17 The Split Tool

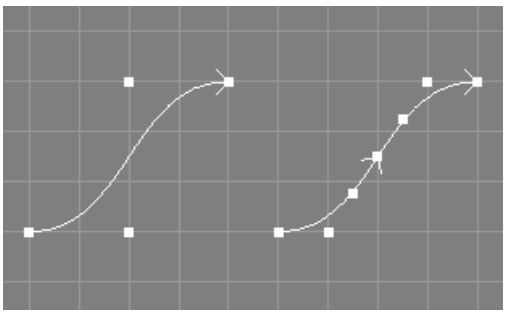

Figure 31: Left: original curve, Right: resulting split curves for t=0.5

- 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.
- 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.

5.18 The Elevate Tool

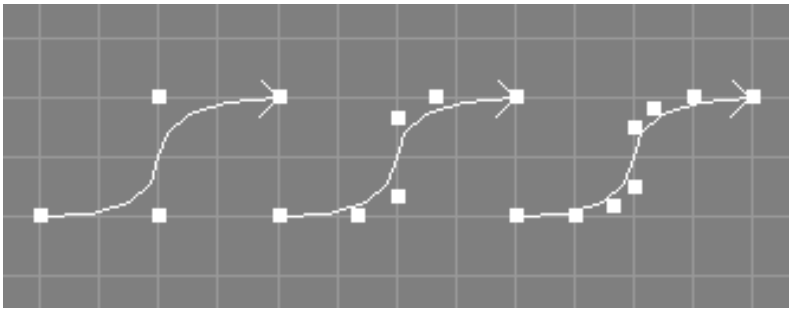

Figure 32: Successive Application of Elevate Tool (Order 3 (left), 4 (middle), 5 (right))

- 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.

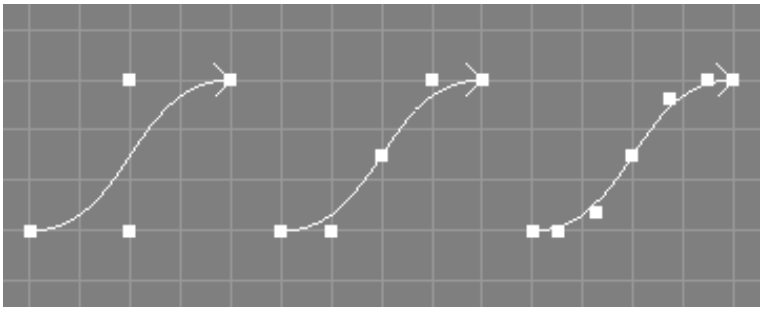

Figure 33: Successive Application of Refine Tool

5.19 The Refine Tool

- Arguments: The refine 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. If you want to refine a curve with new control points and not change the position of existing control points, use the NCAttribute property GUI instead (simply increase the length of the curve by oldlength-1).

5.20 The Clamp Tool

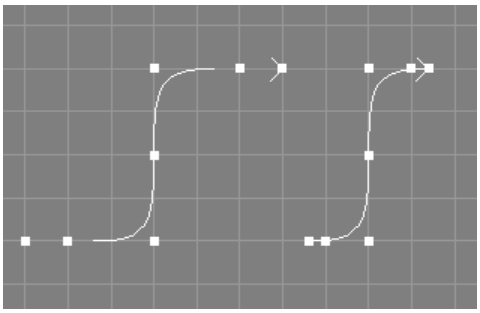

Figure 34: Left: original curve, Right: clamped curve

- 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. The knot type of the curve will be changed to custom.

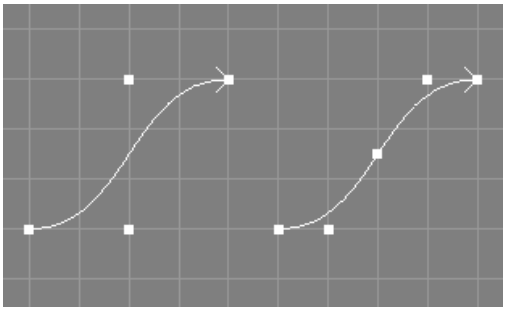

Figure 35: Left: original curve, Right: a knot has been inserted 1 time at t=0.5

5.21 The Insert Knot Tool

- 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). The knot type of the curve will be changed to custom.

5.22 The Plot Curvature Tool

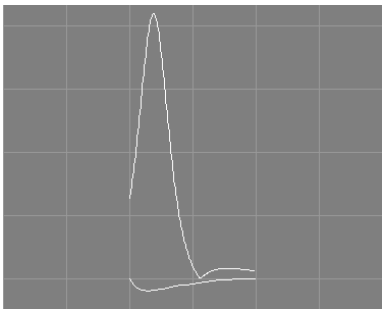

Figure 36: Curvature Plot (top) of simple NURBS curve (bottom)

- 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.

5.23 The Shift Closed B-Spline Tool

- Arguments: The shift closed B-Spline tool takes a number of NURBS curves, that should be closed B-Splines, from the selection and requests one additional integer parameter.

- Operation: The control points of the curve(s) will be shifted, so that the second control point will be the first after this operation. The shifting process will be repeated according to the parameter value given. Eventually selected points will still be selected after this operation.

5.24 The To XY Tool

- 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.

5.25 The Make Compatible Tool

- 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 afterwards.

5.26 The Collapse Points Tool

- Arguments: The collapse tool expects a selected NURBS curve and a number of selected (tagged) control points (see section 3.5 (Selecting Points) (page 36) 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.

5.27 The Explode Points Tool

- Arguments: The explode tool expects a selected NURBS curve and a number of selected (tagged) multiple points (see section 3.5 (Selecting Points) (page 36) 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 that 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 NURBCurveAttrib property if they still have identical coordinate values. In other words, you should immediately edit the control points after exploding!

5.28 The Swap UV Tool

- Arguments: The swap uv tool takes a number of NURBS patches or BPatch or PatchMesh objects from the selection.
- Operation: The u and v dimension of the selected objects will be swapped without changing the shape of the patches.

5.29 The 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.

5.30 The 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).

5.31 The 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).

5.32 The 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.
- Note: The NURBS patch is not deleted afterwards.

5.33 The 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. Other parameters (KnotType_U, Knots_U etc.) are taken from the first curve.
- Note: The NURBS curves are not deleted afterwards.

5.34 The Extract NC Tool

- Arguments: The Extract NC tool takes the selected objects from the selection.
- Operation: The tool creates a ExtrNC object, and moves the selected objects to it.
- Note: See section 4.34 (ExtrNC Object) (page 85) for more information regarding the ExtrNC object. This tool uses the object clipboard to move the objects so that the original clipboard contents are lost when this tool finishes.

5.35 The Tesselation Tool

- Arguments: The tesselation tool takes all NURBS patches 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 tune the corresponding tesselation parameter using a slider and an entry widget. The initial parameter values will be derived from the TP tag of the first of the selected objects (if it has such a tag). The selected NURBS patches will be tesselated with the chosen parameters. The PolyMesh objects created by the tesselation will immediately be displayed in all view windows instead of the original NURBS patches. If the preference option `"LazyNotify"` is enabled, updates of the tesselation that normally occur while dragging the slider will be deferred until the mouse button is released. If the `"Ok"` button is pressed to close the tesselation tool, all selected NURBS patch objects will be replaced by their tesselated counterparts; if `"Cancel"` is used, all NURBS patch 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 NURBS patch objects.
- Note: The tesselation tool will block all other parts of Ayam while it is running.

6 The Tcl Scripting Interface

The scripting interface is mainly a bunch of Tcl procedures and Tcl commands that a big part of Ayam uses internally.

Using Tcl, you could directly modify the code Ayam consists of. This is, however, not recommended for good reasons. 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 that commands may change in future versions.

The scripting interface may be used directly from the console of Ayam. You can, of course, also write procedures 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 to be executed automatically on startup using the preference setting `"Main/Scripts"`.

Note that most of the listed commands work in background, without changing anything to the Ayam GUI and Ayam view windows, for execution speed. If you want your changes to become visible you have to update the various parts of the GUI explicitly (see also section 6.2.12 (Updating the GUI) (page 110)).

From scripts it may be necessary to check wether 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 it to zero before and check it after the operation in question to see whether the operation performed successfully.

6.1 Global Variables

Several global variables exist in the Ayam Tcl context, that may be useful for scripts.

6.1.1 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.

6.1.2 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 entries. You can reset your `"ayamrc"` file anytime using the command line option `"-failsafe"`. Use `"parray ayprefs"` in the console to see what is there. More documentation to come.

More documentation to come.

6.2 Index of Procedures and Commands

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"`
- Description: Fire up a web browser and display the help for the designated Ayam scripting interface command.

6.2.2 Creating Objects

To create new objects the `"crtOb"` command can be used.

- Synopsis: `"crtOb type [args]"`
- Description: New objects may be created with the command `"crtOb"`, type may be derived from the type name, as displayed in the tree view.

  Depending on the type, further arguments may (or have to) be given, other types expect objects to be selected:

  - `"NCurve"`: NURBS curves accept a single integer as length of the new curve, the length defaults to 4. Example: `"crtOb NCurve -length 10; uS; rV"`
  - `"NPatch"`: NURBS patches accept two integers as width and height of the new patch, width and height both default to 4. Example: `"crtOb NPatch -width 2 -height 2; uS; rV"`
  - `"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), `"4"` (primitive).
  - `"Material"`: Materials must be given an additional argument giving the name of the new material. Example: `"crtOb Material default; uS; rV"`
  - `"Instance"`: creates an instance of the selected object.
  - ...

6.2.3 Manipulating the Selection

This command is probably the most important one, because a lot of the other commands operate on selected objects only.

selOb - select object(s):

- Synopsis: `"selOb [index]"`
- Description: Use this command to set or clear the current selection, index may be an ordered list of indizes, a single index or empty. If no index is given, the current selection will be cleared.

withOb - work on certain selected object(s):

- Synopsis: `"withOb index command"`
- Description: Use this command to execute command on a single object from a multiple selection designated by index.

6.2.4 Manipulating Properties

Every property has a corresponding global array, where the data of the property is saved. This global 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 data may be transferred back to the selected object using the so called set-property callback. The names of the array and the callbacks may be inferred from another global array that is named like the property itself, e.g. for the tags property the following global array is defined:

```
array set Tags {
arr   tagsPropData
sproc setTagsp
gproc getTagsp
w     fTagsAttr
}
```

If sproc or gproc are empty (`""`), a standard callback named `"setProp"` or `"getProp"` should be used.

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
- list incomplete, more to come...

6.2.5 Clipboard Operations

These commands operate the object clipboard:

copOb - copy object:

- Synopsis: `"copOb"`
- Description: Copy the selected object(s) to the object clipboard.

cutOb - cut object:

- Synopsis: `"cutOb"`
- Description: Move the selected object(s) into the object clipboard.

pasOb - paste object:

- Synopsis: `"pasOb"`

- Description: Copy the selected object(s) from the object clipboard to the current level.

delOb - delete object:

- Synopsis: `"delOb"`
- Description: Delete the selected object(s) from the scene.

cmovOb - paste (move) object:

- Synopsis: `"cmovOb"`
- 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"`
- 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. Note that you may call this procedure also using the shortcut `"copyProp"`.

pclip_paste/pasteProp - paste a property

- Synopsis: `"pclip_paste"`
- 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"`.

6.2.6 Hierarchy Operations

These commands manipulate the current level of Ayam.

goDown:

- Synopsis: `"goDown index"`
- 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.

goUp:

- Synopsis: `"goUp"`
- Description: Go one level up in the object hierarchy.

goTop:

- Synopsis: `"goTop"`
- Description: Go to the top level of the object hierarchy.

6.2.7 Transformations

movOb - move objects:

- Synopsis: `"movOb dx dy dz"`
- 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"`
- 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"`
- 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!

movSel - move selected points:

- Synopsis: `"movSel dx dy dz"`
- 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.

rotSel - rotate selected points:

- Synopsis: `"rotSel dx dy dz"`
- 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!

scalSel - scale selected points:

- Synopsis: `"scalSel dx dy dz"`
- 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"`
- 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"`
- 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.

6.2.8 Manipulating Shaders

shaderSet:

- Synopsis: `"shaderSet shadertype [varname]"`
- 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"`
- 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.

6.2.9 Manipulating Tags

addTag:

- Synopsis: `"addTag type value"`
- 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.

delTags:

- Synopsis: `"delTags type"`
- Description: Delete all tags of type-string type from the currently selected objects(s). If type is `"all"`, all tags are deleted from the currently selected objects(s).

getTags:

- Synopsis: `"getTags tvname vvname"`
- Description: Get all tags from the currently selected objects and put them as lists into two variables named tvname for the tag type-strings and vvname for the tag value-strings.

setTags:

- Synopsis: `"setTags tags"`
- Description: Clear all tags from the currently selected object and set new tags. The tag type-strings are taken from the list elements with even index numbers and the tag value-strings from the list elements with odd index numbers.

6.2.10 Manipulating NURBS Curves and Surfaces

clampNC - clamp NURBS curve:

- Synopsis: `"clampNC"`
- 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 start and end (where o is the order of the curve).

elevateNC - elevate NURBS curve:

- Synopsis: `"elevateNC n"`
- 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"`.

insknNC - insert knot into NURBS curve:

- Synopsis: `"insknNC u r"`
- 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!

refineNC - refine NURBS curve:

- Synopsis: `"refineNC [{u1 u2 un}]"`
- 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, if no list of new knots is given, add a new knot into each interval in the old knot vector. The knot type of the refined curve will be changed to `"Custom"`.

revertNC - revert NURBS curve:

- Synopsis: `"revertNC"`
- Description: Revert the direction of the selected NURBS curves.

rescaleKnNC - rescale knots of NURBS curve:

- Synopsis: `"rescaleKnNC [-r rmin rmax|-d mindist]"`
- 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. The knot type of the curve has to be `"Custom"`! This operation does not change the shape of the curve.

splitNP - split NURBS patch:

- Synopsis: `"splitNP (u|v)"`
- Description: splits the selected NURBPatch into NURBS curves, along parametric dimension u or v. See also section 5.17 (The Split Tool) (page 96).

buildNP - build NURBS patch:

- Synopsis: `"buildNPatch"`
- Description: builds a NURBPatch from the selected NURBS curves. See also section 5.33 (The Build from Curves Tool) (page 100).

6.2.11 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 object, e.g. a NURBS curve requires just one index parameter (indexu), whereas a NURBS patch requires two (indexu and indexv).

getPnt:

- Synopsis: `"getPnt [-trafo] indexu [indexv] varx vary varz [varw]"`
- Description: Get a control point of the currently selected object and write it's coordinates into the variables varx, vary, varz, and varw. If the special argument `"-trafo"` is given, the coordinates will be transformed by the values given in the objects Transformation property.

setPnt:

- Synopsis: `"setPnt indexu [indexv] x y z [w]"`
- Description: Set a control point of the currently selected object to the coordinates x, y, z, and w.

6.2.12 Updating the GUI

rV - redraw all views:

- Synopsis: `"rV"`
- 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]"`
- 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]"`
- 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"`
- 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"`
- 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).

6.2.13 Custom Objects

io_lc - load custom:

- Synopsis: `"io_lc filename"`
- Description: Load the custom object from file filename.

6.2.14 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"`
- 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 construct 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.

forAllT:

- Synopsis: `"forAllT type recursive command"`

- 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.

6.2.15 Scene IO

newScene:

- Synopsis: `"newScene"`
- Description: clears the current scene.

replaceScene:

- Synopsis: `"replaceScene filename"`
- Description: clears the current scene, then loads a new scene from filename.

insertScene:

- Synopsis: `"insertScene filename"`
- Description: inserts a scene from filename.

saveScene:

- Synopsis: `"saveScene filename"`
- Description: saves the current scene to filename.

6.2.16 Reporting Errors

ayError:

- Synopsis: `"ayError code place detail"`
- Description: This command reports errors or warnings. You should always use ayError instead of puts because the error reporting mechanism of Ayam features compression of repeated messages and logging to files. 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.

6.2.17 Miscellaneous

getType:

- Synopsis: `"getType varname"`
- 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"`
- Description: This command calculates a name for a temporary file in tmpdir and puts the complete name into varname.

hasChild:

- Synopsis: `"hasChild"`
- Description: This command returns 0 if the selected object has no child objects, otherwise it returns 1.

undo:

- Synopsis: `"undo [redo|save|clear]"`
- 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.

  - If the argument is `"clear"`, the undo buffer is cleared.

convOb:

- Synopsis: `"convOb [-inplace]"`
- 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).

forceNot:

- Synopsis: `"forceNot"`
- 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.

6.3 Scripting Interface Examples

Here are some example scripts for the Ayam Tcl scripting interface.

You may copy and paste all examples directly from the documentation into the console of Ayam.

6.3.1 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}"
 uS
}
```

6.3.2 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.

6.3.3 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
 uS

 # how does the current level look like?
 getLevel a b

 # enter the Sweep (the last object in the current level)
 goDown [expr [llength $a]-1]
 uS

 # now, we create a new curve (a closed BSpline suitable as cross section)
 crtClosedBS 8
 uS

 # how does the current level look like?
 getLevel a b

 # select last object (the newly created curve)
 selOb [expr [llength $a]-1]

 # now, we rotate and scale the curve
 rotOb 0 90 0
 scalOb 0.25 0.25 1.0

 # move trajectory back (we use "cmovOb" and _not_ "pasOb", because we
 # really move (and not copy) the curve
 cmovOb
 # go up to where we came from
 goUp

 # update GUI
 uS
 sL
```

```
 # 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.

6.3.4 Tool Box Buttons

Here is another example that shows how you may add buttons to the tool box. myImage should be an image created e.g. from a GIF file of the size 25 by 25 pixels.

```
global ay

# create an image from a GIF file:
image create photo myImage -format gif -file /home/user/a.png,height=4.0cm}}\fi

set b .tbw.f.mybutton

# if the button not already exists:
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
  lappend ay(toolbuttons) mybutton

  # display the button:
  # from now on, it will be under the
```

```
  # automatic tool box layout management
  toolbox_layout
}
```

This example shows that a) buttons have to be created in the frame ".tbw.f" b) Ayam manages a list of all buttons in the global array ay in "ay(toolbuttons)", the order in that list is the order in which buttons appear in the tool box 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.

Here is an example that adds two buttons to the bottom of the tool box spanning the whole window (this works best with the standard tool box layout of 4 by 12 buttons):

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

# create two buttons inside the frame:
button $f.b1 -width 5 -text "Coll." -command { collNC; rV; }
button $f.b2 -width 5 -text "Expl." -command { explNC; rV; }
pack $f.b1 $f.b2 -side left -fill x -expand yes

# 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]
```

7 Miscellaneous

7.1 The Undo System

Using the undo system you may correct mistakes you made while modeling. However, it is currently not possible to undo any changes to the object hierarchy, including clipboard 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! Only changes to objects are undoable. This includes changes made by modeling actions, changes made using property GUIs, but also changes to views (type changes or changes to the camera settings associated with a view).

The undo system works by storing copies of the different states of changed objects in an undo buffer. The storage space occupied by the undo buffer may be adjusted using the preferences (Prefs/Modeling/UndoLevels). Note that a value of -1 for UndoLevels completely disables the undo system. You may step backward through the saved states using <Ctrl+z> (undo) but also forward using <Ctrl+y> (redo).

Several actions will completely clear the undo buffer (no undo is possible after those actions): Delete Object, New Scene, 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.

7.2 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 Win32 platforms (Windows95-2000) it is `"$(HOME)/ayamrc"` if the environment variable `HOME` exists, else `"$(TEMP)/ayamrc"`.

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 tool box window
3. keyboard shortcuts to menu entries and modeling actions
4. RiOption and RiAttribute databases

You may edit the file by hand, but keep in mind, that the file will be parsed by Tcl. Should you, for some reason, destroy your ayamrc so that Ayam does not start correctly anymore you may start Ayam with the command line option `"-failsafe"`. When the application is left the next time, or the main menu entry `"Save Prefs"` is invoked a correct ayamrc will be created again. All preference settings will be reset to factory defaults, however!

7.2.1 Changing Keyboard Shortcuts

You may adapt the shortcuts used in the GUI to your special needs using the ayamrc file. Note that if you do that the GUI (the menu entries) will adapt to your changes but certainly neither this documentation nor the tutorials! Ayam does not check for clashes! 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. 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.

7.2.2 Hidden Preference Settings

The ayamrc file currently contains the following adjustable hidden preference settings:

- "Balloon": time in ms until the tooltip window appears (default: 1500 - 1.5s)
- "EFlush": time in ms between two flushes of the error message buffer (default: 2000 - 2s)
- "toolBoxList": a list of sections describing the appearance of the tool box window (default; using all available sections: {trafo trafo2 solids misco nurbs toolobjs points nctools1 nctools2 camera misc})
- "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.
- "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)
- "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 by Direct Rendering should be kept on disk (default: 5)
- "Wpclip_pastetosel": should "Special/Clipboard/Paste Property to Selected" raise a warning requester? (default: 1 - yes)
- "DailyTips": a list of strings that appear as tips on startup in the console
- "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
- "MaxTagLen": the maximum number of characters to be displayed in the buttons in the Tag Property GUI (default: 30)
- "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.
- "Prompt": controls the prompt for the 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 you can undo and redo and the last component of the current directory of Ayam. You may e.g. change this to "[pwd]>" to see just the full path name of the current directory. If you want to display the value of a variable in the prompt (e.g. designating the current level in the scene hierarchy) you need to bind a write-trace to that variable, that calls the procedure "update_prompt". This write-trace may e.g. be established using a small script: "trace variable <vname> update_prompt".

- "BackupExt": is the file name extension to be used for backup files. Default values are "~" for Unix and ".bak" for Win32.
- "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.
- "IconGamma": this setting may be used to adapt the contrast of all icons (in view menu and toolbox) to your display gamma. If you are on a SGI it is recommended to set this to about "0.7".

7.2.3 RiOption and RiAttributes Database

Using your ayamrc file, you may also adapt the database of RiOptions and RiAttributes to your rendering system.

You can then easily add those options and attributes to your scenes using tags and the main menu entries "Special/Tags/Add RiOption" and "Special/Tags/Add RiAttribute", see also sections 4.6.1 (RiAttribute Tag) (page 42) and 4.6.2 (RiOption Tag) (page 43).

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. But be careful with the brackets!

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`}}

7.3 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); initial value of the `"TmpDir"` preference setting (used on Win32 if no ayamrc file exists).
- `"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).

7.4 Import of Mops Scenes

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 4.15.2 (NCurveAttrib) (page 62), and 4.16.1 (NPatchAttrib) (page 63) for more information about tolerance and display mode) 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.

7.5 Import of RenderMan Interface Bytestreams (RIB)

Using the RRIB (Read RIB) plugin you may import RenderMan Interface Bytestreams into Ayam. Start importing a RIB using the menu entry `"Custom/RRIB/Import RIB"` (if this menu entry is not available, you have to load the `"rrib"` plugin using the menu entry `"File/Load Custom"` first).

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!), and (since Ayam 1.7) arbitrary primitive variables (e.g. varying or vertex).

The RRIB plugin does not support reading of curves, implicit surfaces (blobby models), procedural objects, delayed read archives, and other calls to the RenderMan Interface not that useful for a RIB import like e.g. RiMakeTexture. Unsupported geometric primitives and other calls to the RenderMan Interface are silently ignored.

The RIB import may be controlled via different options:

- `"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.

- `"ErrorLevel"`, this option controls how many error messages should be printed to the Ayam console while reading the RIB. Available values are: `"Silence"` no output except for severe errors, `"Errors"` all error messages should be printed, `"Warnings"`, all error and warning messages should be printed, and finally `"All"` all messages, even informative, should be printed. 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).

Note that for NURBS patches and bicubic patch meshes, points of type ”P” will be promoted to ”Pw”. Furthermore, objects of type (general) polygon and polygon mesh will 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 7.10 (Automatic Instancing) (page 127)).

7.6 Wavefront OBJ Export

Since Ayam 1.7, it is possible to export scenes or objects to the Wavefront OBJ format (version 3.0). The corresponding main menu entry is `"File/Export/Wavefront OBJ"`. The Wavefront export currently supports the following primitives:

- 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)
- 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 Wavefront OBJ export, currently, ignores all material information and possibly present primitive variable tags. Only the pure geometry information is written to the OBJ file.

The following parameters, additionally, control the Wavefront OBJ export: `"Selected"` - exports only the currently selected object(s); `"TessPoMesh"` - automatically tesselates all PolyMesh objects to triangles for export; `"OmitCurves"` - omits all NURBS curves and NURBS curve providing objects from the exported file.

7.7 3DMF Import Export (MFIO) Plugin

Using the MFIO plugin you may import and export scenes from (to) the 3DMF format (version 1.0 only!) from Apple. Start importing a 3DMF file using the menu entry `"Custom/3DMF/Import"` (if this menu

entry is not available, you have to load the `"mfio"` plugin using the menu entry `"File/Load Custom"` first).

Import supports the following primitives:

- Polyline, Triangle, TriGrid, Polygon, general Polygon, Box,
- NURBS curve, NURBS surface (with trim curves),
- Ellipsoid, Cylinder, Cone, Disk, and Torus.

The following transformations are supported:

- Scale,
- Translate,
- Rotate, RotateQuaternion, RotateAxis (if axis is X, Y, or Z).

Furthermore, the import plugin reads the structure of the scene from Container objects. Reference objects will be resolved to normal objects while importing. Instances may be easily created again using Automatic Instancing (see section 7.10 (Automatic Instancing) (page 127)).

The MFIO export supports the following primitives:

- 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.

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 red component has a value different from -1.

Support for import or export of lights, camera attributes as well as material attributes other than material color and opacity is currently not available.

7.8 The OpenNURBS IO (onio) Plugin

Since version 1.8.2 Ayam contains a plugin that may export to and import from the Rhino 3DM format using the OpenNURBS toolkit.

The export functionality of the onio plugin currently covers export of all boxes, quadrics, NURBS, 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 supported, the export of general trimmed NURBS patches is not supported. This is because of a missing feature (pushing up 2D trim curves to 3D curves for arbitrary NURBS surfaces) in the OpenNURBS toolkit.

The export options include "Accuracy": this 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). "WriteCurves": if this option is disabled, no curves will be written to the exported Rhino file. "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. However, not all features of the quadric objects will be translated. 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 option has no effect on the export of disks, hyperboloids, and paraboloids.

Since the Rhino 3DM 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). Object names will be written 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 named "Arm/MySphere" and "Arm/MyCylinder".

The import functionality of the onio plugin currently covers import of all NURBS and BRep objects and objects that may be converted to NURBS using the OpenNURBS toolkit (those are: PolylineCurve, PolyCurve, LineCurve, ArcCurve, CurveOnSurface, RevSurface, SumSurface, and PlaneSurface). References will be resolved.

The "Accuracy" import option controls the tolerance of OpenNURBS internal operations, in this case the value is mostly used for conversion operations to the NURBS form. If the "ReadCurves" import option is switched off, no curves will be imported. 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. Another import option is "IgnoreFirstTrim", this is useful if the first bounding trim loop of NURBS surfaces should be ignored. This trim loop is often not needed, because it encloses the whole surface that is not trimmed further in many cases.

7.9 Shader Parsing Plugins

Since Ayam 1.3, the following plugins are provided to allow parsing of shaders: "ayslb" for Air, "ayslx" for Aqsis, "ayso" for RDC, "ayslo" for PRMan, "ayslo3d" for 3Delight, and (since Ayam 1.6) "aysdr" for Pixie.

After loading of one of the aforementioned plugins, Ayam will be able to parse shaders compiled with the shader compiler of the respective renderer.

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", and "loadaysdr.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. However, the standard behavior while loading of such a plugin requires you to adapt the shaders search path manually and to scan manually for shaders too. Both actions may be carried out using the preferences editor.

This is, however, not necessary if you load the shader parsing plugin automatically on startup of Ayam, as the loading of the scripts will happen before the startup sequence executes the initial shader scanning pass.

7.10 Automatic Instancing

Automatic Instancing is available via the main menu entry: "Special/Instances/Automatic Instancing". Automatic Instancing creates instances from all instantiable objects from the current level and below, 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.

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

7.11 Importance Driven Rendering (IDR)

The importance driven rendering plugin may be used to drastically reduce rendering times while developing a scene. It works in three main steps:

1. Importance values are assigned to elements of the scene.
2. Two rendering passes are started according to the assigned importance values. Elements of different importance values are mutually masked out using "RiMatte" statements.
3. The resulting partial images are composed to a single resulting image, which is then displayed.

The parameterisation of the two rendering passes ensures, that the total rendering time is lower than the rendering time of a single pass with high quality.

Many options exist to assign importance and parameterize the rendering passes:

Elements of the scenes may be geometric objects, regions in image space, or regions in object space. Importance values are currently just binary values. Assignment may take place manually (using IDR tags) or half-automatic by derivation of importance from currently selected or changed objects. To avoid inconsistency in the resulting images, importance values may be propagated between (geometrically or hierarchically) near objects, or between objects that are related (e.g. from a material to a geometric object).

Parameterisation of the two rendering passes currently includes selection of a different renderer and the possibility to reduce rendering resolution and shading rate. To further reduce rendering times for Raytracing renderers, the size of the region to render may be automatically adapted to the elements of the current importance value (including an optimisation run that balances renderer startup times and times needed to render regions not originally occupied by two regions to merge).

Furthermore, caching of partial images is possible. However, the implementation of this feature is not very sophisticated at the moment, as it uses the Unix text tool `"diff"` to decide whether two RIB streams are identical and hence need no re-rendering.

To start using IDR:

1. load a scene (e.g. the cactus example scene),
2. load the IDR plugin (menu `"File/Load Custom"`),
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

7.12 CSG preview using the AyCSG plugin

The AyCSG plugin may be used to resolve and preview CSG operations. For this, the plugin uses image based CSG rendering algorithms provided by the OpenCSG library. The OpenCSG library, currently, supports the Goldfeather and the SCS algorithm. The latter only works properly with convex primitives. Since

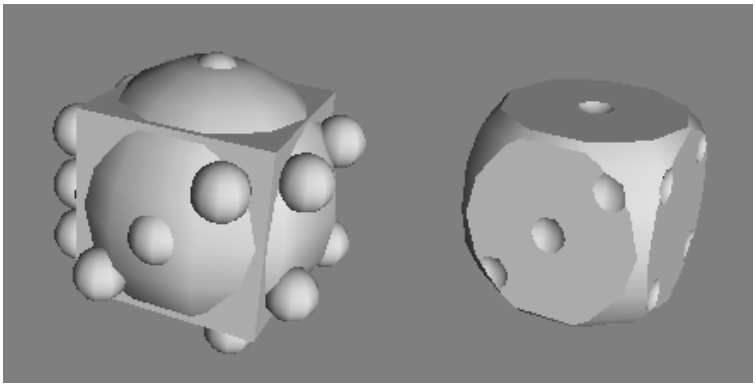

Figure 37: CSG preview example (left without, right with CSG)

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 should 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 you, consequently, may need to revert the patches (e.g. using the `"RevertU"` tool, see 5.30 (Revert U tool) (page 100)). 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, 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.

If you hold down <Shift> while pressing the button, the view will continually render CSG.

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.
- "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.

  "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".

  "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.
- "CalcBBS" determines whether bounding boxes should be calculated and used for speed up (not working at the moment).

7.13 Increasing drawing speed

In case of sluggish response of the user interface of Ayam (not accounting for long tree update operations) several things to increase drawing speed can be done:

- Hide objects or complete object hierarchies using "Hide" in the "Tools" menu.
- Disable drawing of true NURBS curves/surfaces, if you can. Use the ControlHull display modes.
- 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 modeling 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".

7.14 How can you help?

1. Write/translate tutorials.
2. Implement custom objects. This will be discussed a bit more later on.
3. 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?), CSG-preview with OpenGL, true support for subdivision surfaces. The project page of Ayam on SourceForge lists some more tasks.
4. Debug the MF3D code, which is currently not working well with binary files from different byte-order platforms.
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!

You can help by implementing more different high level objects like Trees, Landscape, Sky, Text, whatever you can think of. Note that the license of Ayam does not prevent you from implementing your object as Shareware or even Commercial software. However, Freeware is preferred for obvious reasons.

But please do me a favour and do not implement objects like simple triangles or polygons. This would be something that really is not intended by me, 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 I am a bit biased against polygonal models. I am not. Polygonal models are the only way to preview complex geometry using hardware accellerated 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.

7.15 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 Modeling: A Procedural Approach by Ebert, Musgrave, Peachey, Perlin, and Worley (Academic Press, 1994)

WWW resources:

- 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/`

7.16 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:

- 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
- 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
- Everyone involved in the development of Tcl/Tk, OpenGL, The RenderMan Interface

8 Index

A

B

C

S

T