Modular Procedural Rigging

Masters Thesis

Alexander Turusov

MSc Computer Animation and Visual Effects

N.C.C.A. - Bournemouth University

21st August 2009

Table of Contents

1. Introduction

Within the vast field of computer graphics and visual effects, rigging is an aspect which stays behind the scenes and yet plays a crucial role in any major production. 3D characters are traditionally the most complex CG elements, due to the amount of work required to make them look plausible to the viewer's eye. Many different areas of CG have to work together in order to produce a character and rigging is placed at the centre of this process, aiming to put everything together and bring the character to life. This makes rigging the most important element in character production, which is why it is important to make the rigging process as efficient as possible. The focus of this thesis is on streamlining and automating this crucial area of CG.

Because each character is different, there is no single way of building a character rig. The rigger is provided with a set of general-purpose tools and a number of requirements that the completed rig must fulfil. How to go about using those tools in order to achieve the desired result is a decision left entirely up to the person doing the rigging, hence the inner workings of different rigs vary greatly. This often results in bottlenecks and inconsistencies within the entire animation pipeline, especially when the rig is originally built by one person, but later has to be modified by someone else. Furthermore, it makes rigs difficult to reuse on other characters, whether as a whole or in parts. Such problems can hinder a production substantially due to people further down the pipeline, such as animators, waiting for a rig to be finished before they can start doing their part of the job. To tackle such problems most companies employ predefined standards, such as naming conventions and pre-built scripts. This does not solve the production problems entirely, however, because quite often characters are substantially different from each-other and reusing one character's rig within another character is impossible. Also, most people have their own varying preferences in how they work, therefore staying within the boundaries of predefined standards can be time-consuming and uncomfortable.

By taking the most commonly used application in rigging – Maya, and building upon its existing feature set, this master project aims at taking the solution to the previously mentioned production problems one step further. Although rigging is a greatly varying process, it can nevertheless be split up into a set of repetitive tasks. These tasks have been automated by developing a rigging toolkit that allows the user to build rigs without worrying about how these repetitive tasks are carried out. Instead, the user is provided with the ability to define how the rig should be built and the toolkit does the rest. This approach ensures that, although no two rigs are the same, they have all been built using the exact same methods and are therefore reusable and can easily be modified. Also, human errors are kept to a minimum due to the automation provided by this solution.

The toolkit works by taking repetitive rigging tasks and packing them into modules. Each module has its own distinct purpose. By arranging these modules into a list and executing them one by one, a rig is procedurally built. The user is essentially presented with the ability of defining what modules are to be used, in what order and with which parameters, while the task of building the rig via these modules is left to the system.

2. Previous Work

Procedural rigging is a relatively new concept and few implementations of it currently exist. Being a process integrated deeply into the production pipeline, making a universal tool which would fit the needs of any company, regardless of its pipeline, is notoriously difficult. Also, because the complexity of a rig is defined by the overall complexity of a character that can be achieved at the current level of technological development, the need for procedural rigging has only occurred in recent years.

The Setup Machine is one example of a procedural rigging solution. Implemented as a plug-in for Maya, it is aimed primarily at the games industry. It provides the user with a set of tools to build rigs based on predefined templates which contain instructions for constructing rig parts – arms, legs, tails etc. By connecting these parts a complete character rig is formed. This approach makes the rig building process fast and intuitive, but takes away a lot of control and flexibility from the user because the templates that are used to construct rig parts cannot be modified. Only a few options exist for defining how a certain part will behave once constructed.

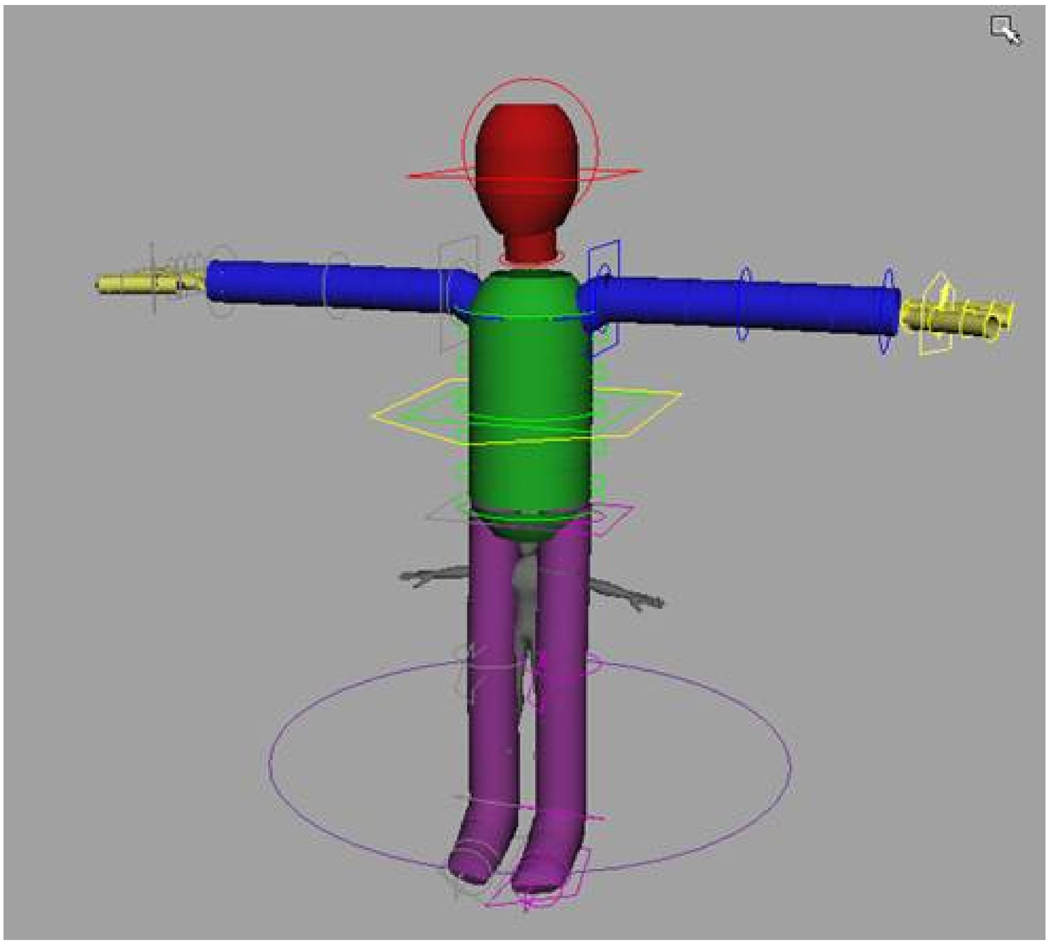


Figure 2.1: A biped rig constructed from several parts using The Setup Machine.

Face Robot is a standalone tool developed specifically for automated facial rigging and animation. It incorporates an intuitive solution to facial rigging which lets the user simply define key points on the target face, resulting in a rig generated entirely by the application. This rig can then be animated within Face Robot itself and exported into one of the main 3D applications along with the animation.

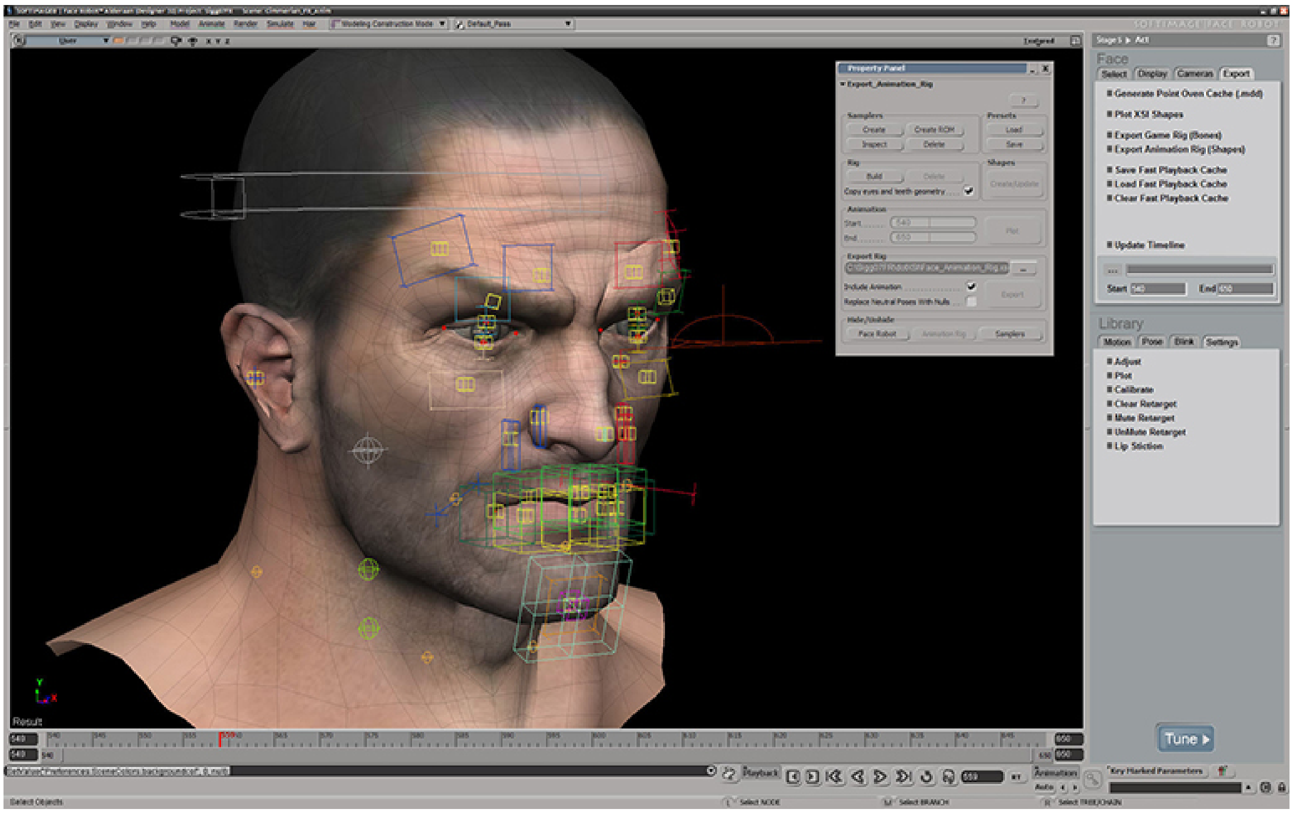


Figure 2.2: A facial rig being constructed in Face Robot.

MetaNode Rigging is a concept developed internally by Bungie and initially used in the production of Halo 3. It is a set of Maya tools which break up the rigging process into individual tasks attached to custom nodes that define the anatomy of a rig. Once the tasks attached to these nodes are carried out, a rig is constructed. The system is fully customizable and extensible through scripting.

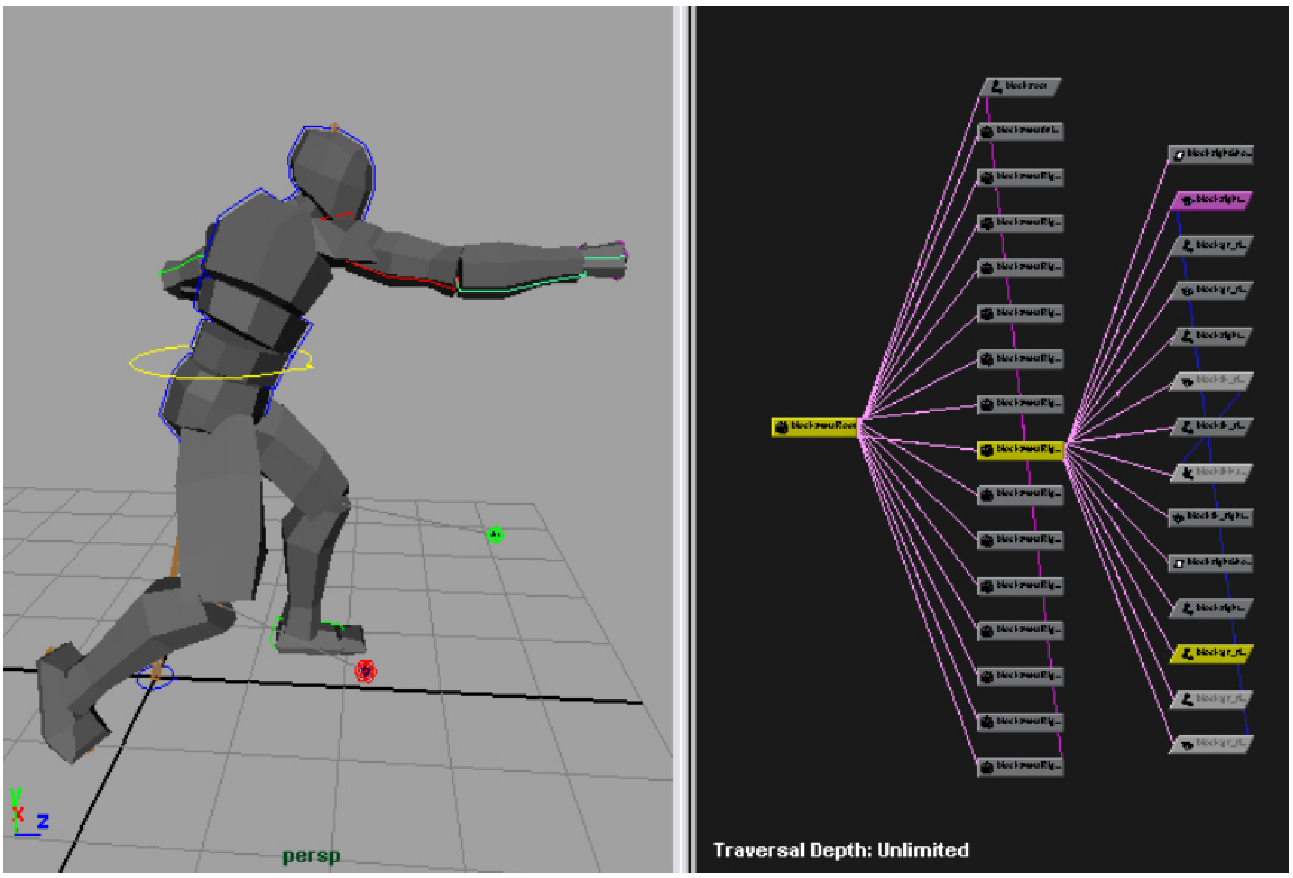


Figure 2.3: Rig constructed with the aid of a MetaNode network.

3. Technical Background

3.1 Animation Pipeline Overview

A typical animation pipeline consists of several interconnected stages. First of all, a concept is drawn for a character. Once the concept is finalized, the character is modeled based on that concept. Rigging comes next. This stage is usually split up into two separate phases – rigging and skinning. As soon as a skeleton is built, the character model can be skinned while the rigger is building the animation controls for that skeleton. Also the character is textured and shaded parallel to the rigging stage. Finally, once the rig has been completed, the character moves on to the animation stage. At this point the rig may not be completely finalized and can move back and forth between the animators and the rigger. This can create substantial delays in the production process, since the animators have to wait for rig updates before they can continue working on the animation. Once the animation is finished, the character moves on to the rendering stage – this process often outputs images split up into several layers. Finally, the compositing stage is reached in which layers produced at the rendering stage are combined.

Although the finer details of animation pipelines may vary in different companies, the general structure outlined above is maintained everywhere. The number of characters passing through such a pipeline during a single production can be anything from one to several hundreds, thus it is important that the pipeline is kept as efficient as possible.

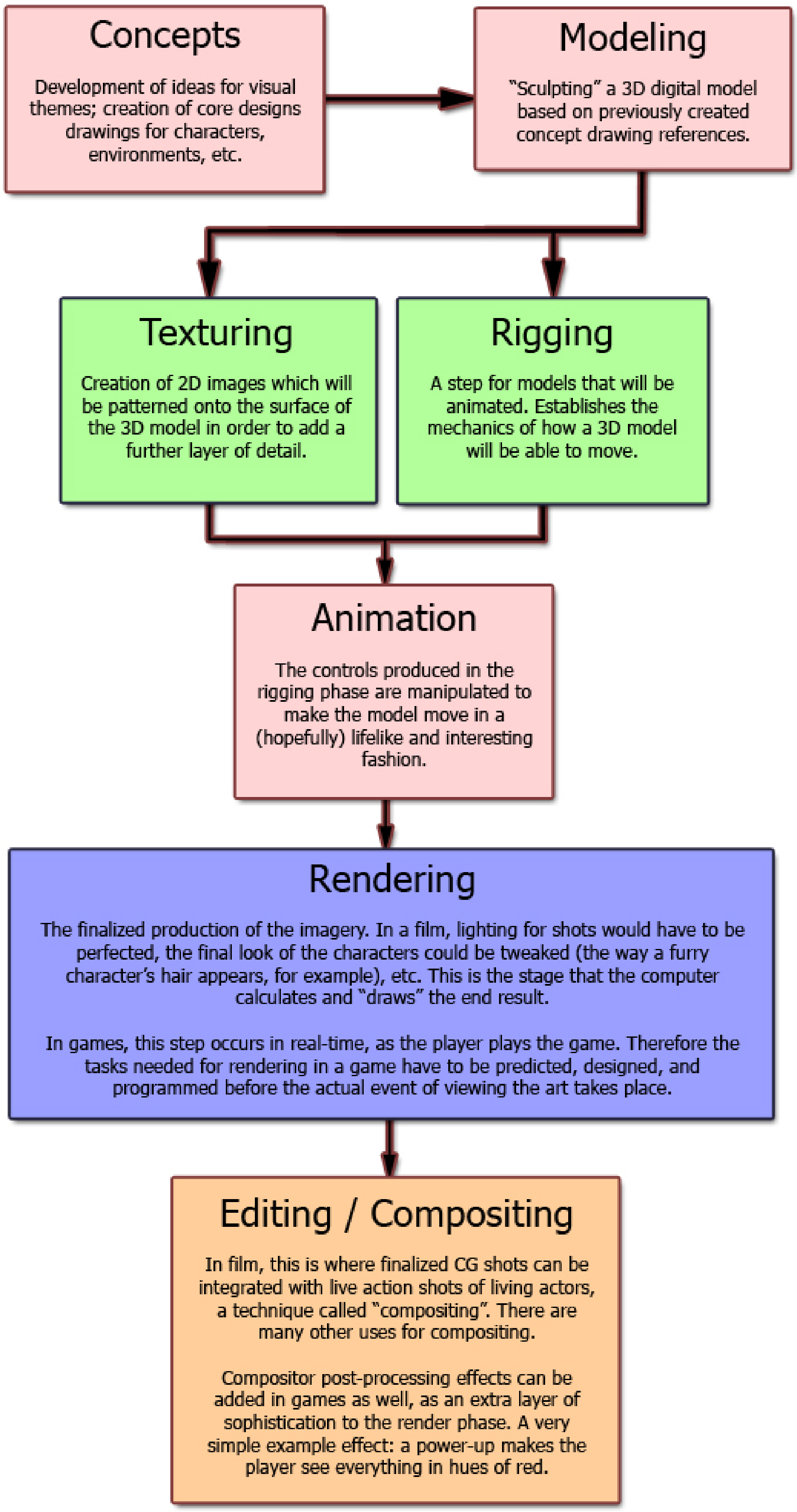

Concepts
Development of ideas for visual themes; creation of core designs drawings for characters, environments, etc.
Modeling
"Sculpting" a 3D digital model based on previously created concept drawing references.
Texturing
Creation of 2D images which will be patterned onto the surface of the 3D model in order to add a further layer of detail.
Rigging
A step for models that will be animated. Establishes the mechanics of how a 3D model will be able to move.
Animation
The controls produced in the rigging phase are manipulated to make the model move in a (hopefully) lifelike and interesting fashion.
Rendering
The finalized production of the imagery. In a film, lighting for shots would have to be perfected, the final look of the characters could be tweaked (the way a furry character's hair appears, for example), etc. This is the stage that the computer calculates and "draws" the end result.
In games, this step occurs in real-time, as the player plays the game. Therefore the tasks needed for rendering in a game have to be predicted, designed, and programmed before the actual event of viewing the art takes place.
Editing / Compositing
In film, this is where finalized CG shots can be integrated with live action shots of living actors, a technique called "compositing". There are many other uses for compositing.
Compositor post-processing effects can be added in games as well, as an extra layer of sophistication to the render phase. A very simple example effect: a power-up makes the player see everything in hues of red.

Figure 3.1: A typical CG pipeline.

3.2 Maya – A Node-based Application

In order to understand the concepts presented in this document, it is important to first become acquainted with how Maya works internally. Essentially this 3D application is composed of three things – commands, nodes and attributes. A node is an object present in the scene which has its own specific function or purpose. A "mesh" node, for example, is responsible for generating and displaying polygonal meshes. A "joint" node connects to mesh nodes and deforms them. Apart from functionality types, nodes also have names by which they can be referenced. These names are defined by the user.

Each node has certain attributes associated with it. These attributes define various parameters that can be changed to manipulate the behaviour of the node they are attached to. Connections between attributes can be established to transfer data between nodes and form various complex relationships. Just like nodes, attributes also have names by which they are accessed.

Finally, commands are used for the manipulation of nodes and attributes. Creation of a node happens via a relevant command. Adding, deleting and editing attributes is also performed by associated commands. Some secondary commands also exist which, instead of actually changing something, simply provide the user with information, such as a list of nodes present in the current scene.

3.3 Skinning

Once a skeleton fitted to the anatomy of a model has been built, the mesh can be attached to that skeleton via the process of skinning. Essentially it involves defining how much the transformations of a certain bone affect a certain part or vertex of the model. Apart from bones, virtual muscles may also be added to create more realistic characters. The muscles are attached to the bones and then linked to the model in the same way – by defining which muscles affect which areas.

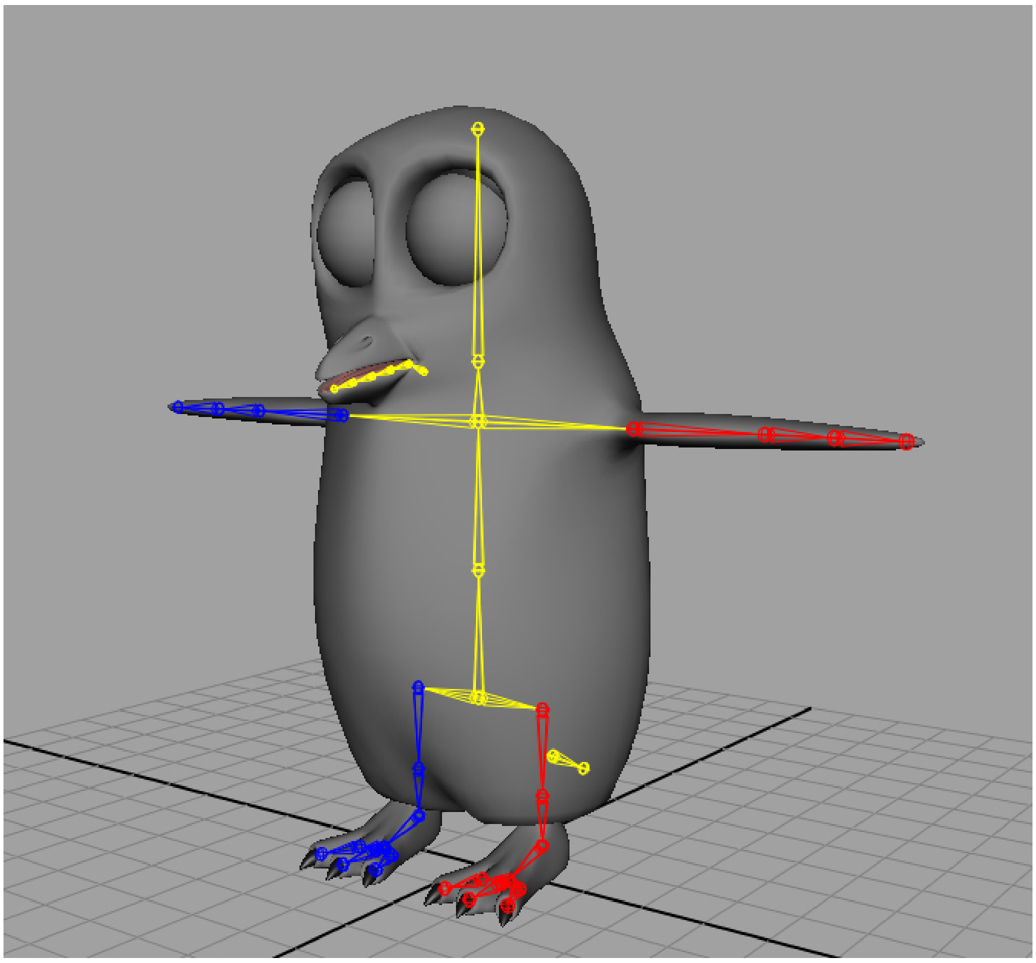


Figure 3.2: A character at the skinning stage.

3.4 Rigging

Rigging is a process in which a model is prepared for animation. Bones are fitted into the model, matching the anatomy, and controls for the created skeleton are then attached. Because animation is a rather slow process, it is important that the animation controls are made as intuitive and efficient as possible. The stage of creating animation controls usually takes up most of the time in a rigging process. Complex behaviours often need to be established to define how the animation controls interact with the skeleton.

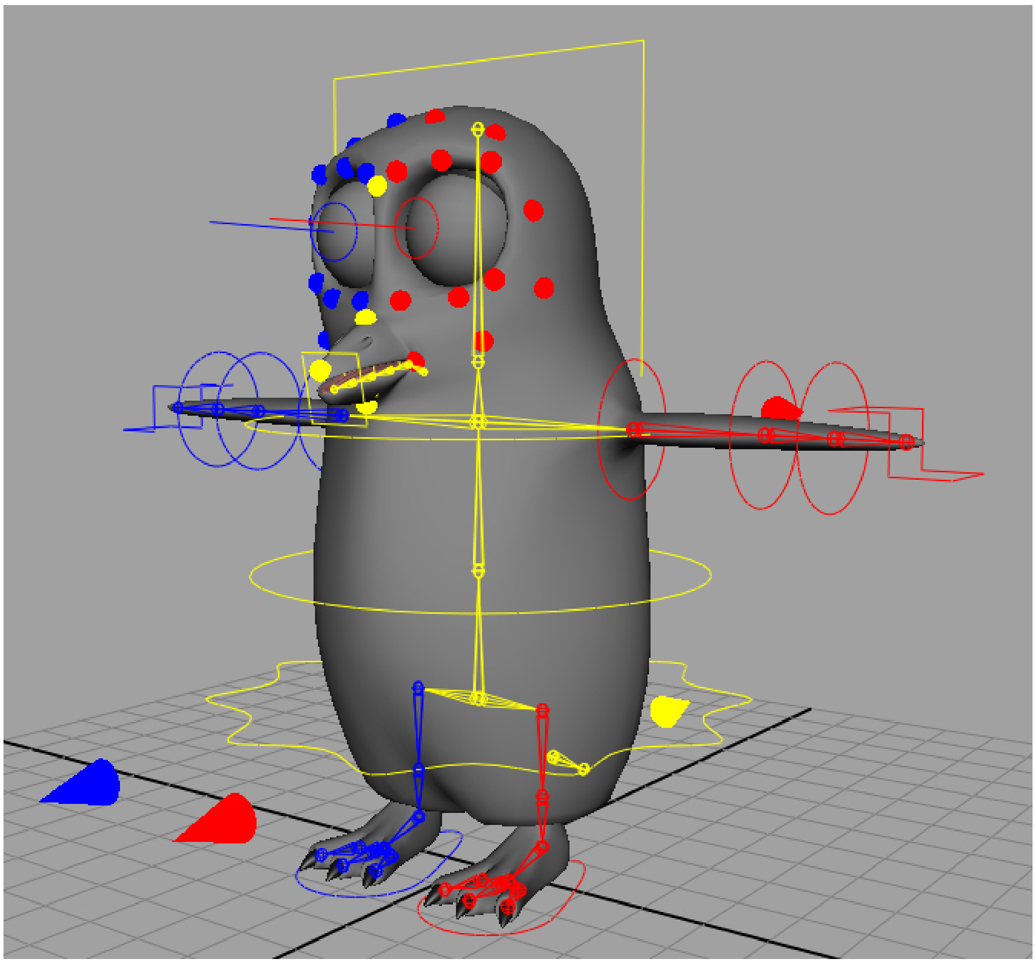


Figure 3.3: A rigged character.

3.5 Procedural Rigging

While manual rigging may sometimes be fairly intuitive, when dealing with a production-level rig thousands of nodes may need to be created and interconnected before the rig begins to function as it should. This means that rigging can quickly become a complicated process. Furthermore, the rig may have to be modified at later stages of production. This can be very difficult to do, because rigs are easy to break and a modified rig may not fit within the general pipeline.

Such issues are good reasons for using procedural rigging. This concept involves a different approach to constructing rigs. Instead of manually building rig components every time, various scripts and tools are used to automate the process. Rather than creating nodes and connecting them manually, the rigger defines in more general terms how the rig should function using the procedural rigging tools. Automation algorithms and scripted actions then perform the steps necessary to build the rig according to the rigger's specifications.

This approach to the rigging process has some distinct benefits. First of all, a highly technical process becomes simpler by leaving the finer details to the rigging tools. Also, a lot of time is saved because a large part of the process becomes automated. And finally a better structured pipeline is created as a result, allowing numerous issues to be avoided. Modifying the rig at any stage of the animation pipeline, for example, becomes much easier because the rigger can simply change the instructions provided to the rigging tools and the rig is instantly rebuilt with the new modifications implemented.

4. Procedural Rigging Implementation

4.1 Overview

In order to be efficient and useful, any procedural rigging implementation must introduce a certain structured workflow into the rigging process. As stated previously, it is difficult to build general-purpose tools which would integrate into any animation pipeline. An attempt to create such a set of tools has been made nevertheless, resulting in the development of four interconnected tools within a unified, general-purpose rigging toolkit. There are two tools for the rigger and two tools for the animator. They are all built using the same code base and, despite functioning independently of each-other, are part of a single workflow. The code itself has been developed with extensibility in mind, so that the toolkit can be extended as necessary to accommodate the needs of any pipeline.

4.2 Workflow

The rigging process, when using this toolkit, takes on a certain workflow which forms the base of the animation pipeline. First of all the rigger uses the Rig Group Editor and the Rig Editor in order to construct a rig. Once this has been completed, the rig is saved as a template. The template file holds information about the skeleton and the actions which need to be performed in order to build and attach animation controls to that skeleton. The rigger's tools can be used to modify such templates at any time. The template files are stored on a server to which animators have access. They use the Rig Loader to select a template and load it from the server. This tool reads the information contained in the template file and uses it to construct the defined skeleton and rig. Once this is done skinning information can be imported to connect the skeleton to the mesh. Now the animators can start their part of the job.

Probably the most powerful feature of this toolkit comes into play at this point. Because rigs often need to be modified after the animation process has already started, the procedural nature of rig construction helps immensely. In the case of manual rigging, the entire rig would have to be broken, reconstructed and the animator would have to wait for all this to happen before animation can resume. This is usually a waste of time, since the required modifications can often be quite small, but due to the nature of manual rigging, the whole animation process must be put on hold regardless of what changes need to be made to the rig. To overcome this problem, this rigging toolkit allows animators to continue working on an older version of the rig, while the rigger incorporates necessary changes into the rig's template. When that is done, the updated template is uploaded to the server and the animators can simply use the Rig Updater to rebuild the rig they were working on using the new template, while keeping the animation they have already done. It is a very simple process, especially for the animator who just needs to click a single button and wait a few seconds for the update to load. Using this workflow, the rigs can be constructed and then passed back and forth through the animation pipeline without any substantial delays in the production process.

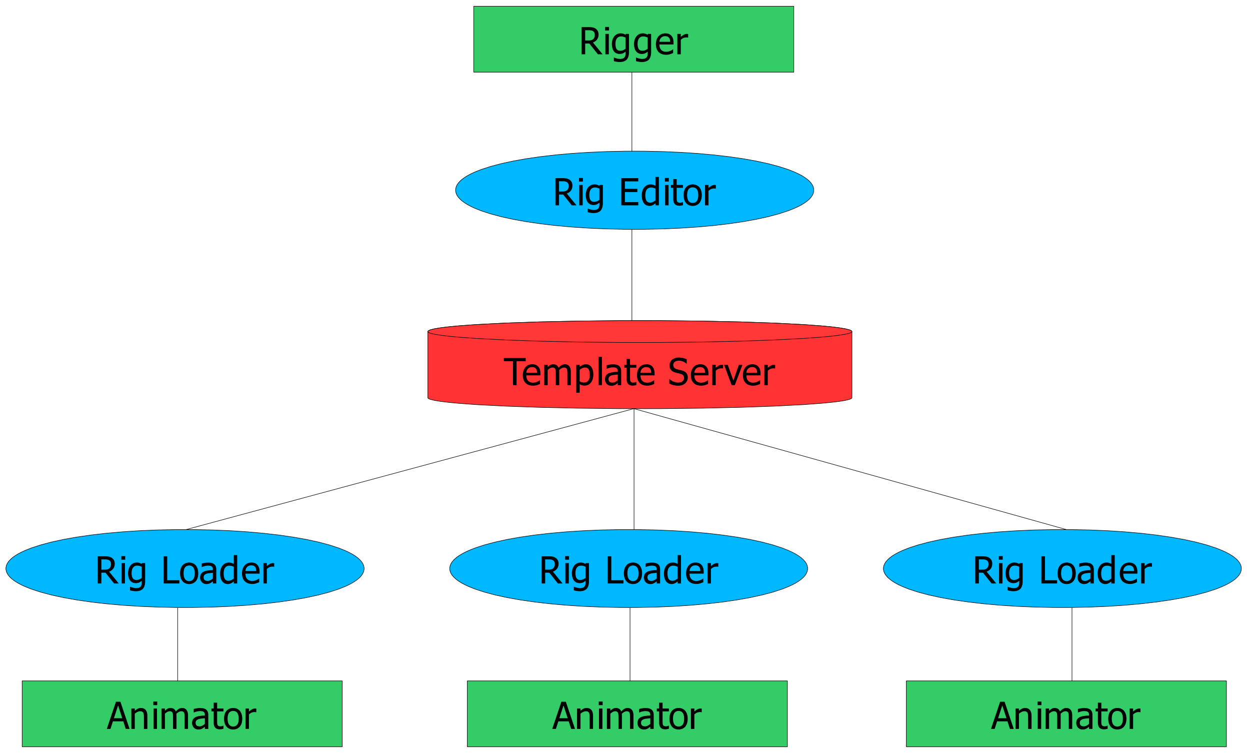


Figure 4.1: Procedural rigging toolkit workflow.

4.3 Development Considerations

To develop this solution several key decisions had to be made: which programming language to use, how to design the user interface, how to make the tools useful in any pipeline and last but not least – how to implement the concepts of procedural rigging efficiently. Although some similar solutions exist, most of them are proprietary, in-house applications which are not available for public use, hence this implementation was based almost entirely on personal experience and ideas of the author.

Programming language: MEL was chosen as the programming language because of how it allows rapid prototyping and generally short tool development cycles. Also, Maya is the application most widely used for CG in general and for rigging in particular. Naturally, since MEL is a rather slow interpreted language, the plan is to continue the development of this toolkit and gradually convert it to C++ code using the Maya API. Nevertheless, because code execution speed is not that important in this solution, MEL was sufficient for developing a working prototype.

User interface: The UI of any application has to be clean and intuitive first and foremost, especially when it comes to tools built with artists as users in mind. In this prototype the animator's toolset has been kept as simple as possible. The rigger's main tool – the Rig Editor, was designed with a graph-based interface in mind, but that can only be implemented successfully after the functionality of the underlying code has been finalised. At the moment the rig construction process takes place using a stack-based interface which was easier and faster to implement in order to develop a fully functioning toolkit on time. The Rig Editor interface has a structured workflow which guides the user from the rig construction stage to the template exporting stage in the correct order, while allowing to quickly switch between the different stages of the process. The Rig Group Editor is the secondary rigger's tool and, just as the animator's tools, has also been kept simple.

Interoperability: When considering the overall implementation, a decision was made to stay away from the need for custom nodes and plug-ins as much as possible. Because of this, the toolkit works entirely using default Maya functionality and the rigs built with it do not require the toolkit to be installed in order to be used. This allows easy integration into just about any pipeline, as well as interoperability with other tools. Furthermore, cleaner scenes are produced as a result because fewer nodes are needed within rig node networks.

Proceduralism: Rigging is essentially a set of steps that the rigger takes in order to construct a finished rig. By examining these steps, they were split up into several modules which the user can employ to procedurally construct rigs. The toolkit allows the user to add as many or as few modules as necessary and arrange them in any required order to achieve the desired result. This is an implementation of a procedural process which was adapted in the development of this solution.

Extensibility: Another important decision was made about the extensibility of the tools. Because rigging is such a varied process, the toolkit can be extended with custom-built modules that perform certain rigging functions. As requirements change, or new requirements arise, modules can either be modified or added without the need for dealing with the base code of the tools themselves. In other words, module code can be added independently of the main toolkit code.

Error-resilience: Since rigging is a complex process, it is easy to make mistakes when building a rig. It is also easy to break one. Breaking of rigs is something that happens quite often at the animation stage due to human errors. To prevent such issues the tools were built with workarounds in mind. First of all, the Rig Editor essentially enforces a certain workflow during the rig construction process and halts the process when errors are encountered, allowing the user to go back, fix the mistakes and then continue working on the rig. Secondly, the most common issue that occurs with rigs is accidental renaming. Quite often rig components need to reference each-other to function correctly. If this is done by using node names, then renaming even a single node can potentially break a rig. That is why a concept the author called "Nameless Scene Traversal" has been implemented to avoid using node names completely. This concept is described later on in this document.

4.4 Pipeline integration

Just about any company that deals with computer graphics has a certain animation pipeline in place. That means the toolkit needs to be integrated into that existing pipeline if it is to be used at all. Because Maya is used as the basis of most pipelines, implementing the solution as a set of tools for this application already formed a solid base for efficient integration. Furthermore, the fact that rigs built with this toolkit can be used on any computer with Maya installed provides for even easier integration, as well as compatibility with other tools that the company might have. If desired, the toolkit can also be used as the base for an animation pipeline, taking on the primary role. In such a case some existing tools may have to either be modified or rebuilt to work based on the principles forming this procedural rigging solution, so that the pipeline workflow stays seamless. Whatever the case, the toolkit was developed with both possibilities in mind.

4.5 Tags and Containers

One of the most important principles behind procedural rigging is the ability to execute commands in a specific order to build rig components. These commands have to be stored in a script of some sort. To make the whole rigging process seamless and more intuitive, a decision was made to avoid using separate scripts that go along with the rig. Instead, joints of the skeleton built for a certain rig act as containers that store these commands. Because the joints always go wherever the rig file is moved, this approach ensures that the commands that build the rig move along as well. The commands themselves are stored in the form of "tags", which are essentially hidden string attributes attached to the joints. The strings contain command parameters. These commands are read and executed by passing the parameters to modules with matching names.

For example, the "createIkHandle" command may have two parameters – start joint and end joint. These would be stored in a string attribute called "createIkHandle". The toolkit's command processor steps through various joints in hierarchic order and looks for such hidden attributes. When it comes across this attribute, it calls the "createIkHandle" module and passes the start joint and end joint parameters to it. The module itself contains a script which builds the required IK handle using those parameters. By chaining a list of such commands together an entire rig can be constructed. The toolkit allows attaching any tags to any joints, but to keep things organised it is best to attach tags to relevant joints, so that navigating through all these tags becomes more intuitive.

There are four types of tags involved in this implementation – system tags, identifier tags, command tags and feature tags. System tags are hidden internal attributes which the user does not have access to and which are used by the system to internally identify various rig components. Identifier tags, on the other hand, can be added and modified by the user to identify various rig components without resorting to node names. These identifier tags are immediately converted to system tags which are the ones actually used for such identification purposes. Command tags, as described earlier, are the ones that store parameters for module commands which build up the rig when executed. Finally, feature tags can be used to group command tags. Thus, by turning feature tags on and off, commands are also turned on and off, allowing for the rig to be built in different ways by forcing the command processor to take varying execution paths.

4.6 Nameless Scene Traversal

This is another important concept behind this procedural rigging solution. To avoid various errors that arise due to accidentally renamed nodes, a system has been implemented which allows avoiding node names completely. Identifier tags, described in the previous section, can be added by the user to various joints of the skeleton. By doing so the skeleton can be split up into groups – global and local groups. A global group unifies the entire rig and separates it from other rigs in the scene. A local group separates various parts of an individual rig – arms, legs etc. Furthermore, node labels can be added to identify individual joints. By examining these identifiers any component of the rig can be found. For instance, the system can find the fourth joint in local group "X" and global group "Y" by getting a list of all the joints in the scene with the matching identifiers attached and then looking up the fourth index in the resulting array.

The same concept applies to nodes created by executed commands. Each node is automatically assigned a set of identifiers depending on which joint the command is attached to (ie: the joint's identifiers are duplicated). Commands are also assigned unique ID numbers (long random number strings) which are then assigned to nodes created by those commands. This allows the nodes created by a specific command to easily be found by looking up objects in the scene with the appropriate ID number. Therefore, to find a specific node of the rig, the system first looks for the appropriate global group, then the local group, then joint number, then executed command name, then command ID and finally the index within the created nodes array.

Note that this object referencing system can be used to reference objects which have not yet been created. Since the rig is built procedurally, most commands reference objects which will only exist after they have been created by another command.

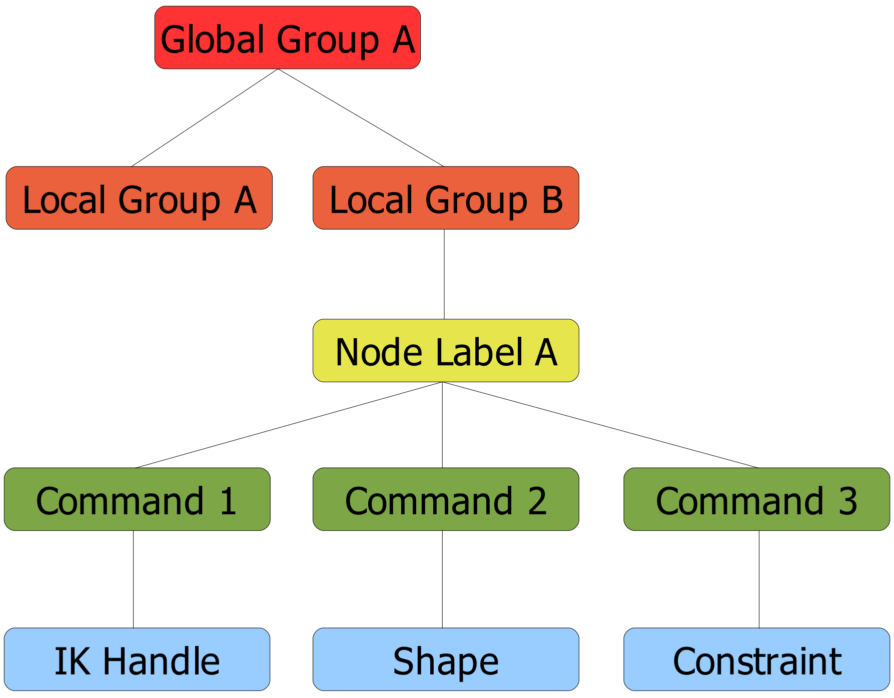


Figure 4.2: Nameless scene traversal.

4.7 Modules and UI Templates

A module is essentially a MEL command which performs certain rigging-related actions, such as building an IK handle. What a module does is left entirely up to the module developer – there are no limitations. In order to install a module the written MEL script must be assigned a specific name, which must also be added (declared) within a list of module names contained in the code. The module is a command and has certain parameters. The system must know what parameters are available and how to present them to the user in the form of options within the UI. This is carried out by building a template which uses a basic scripting language that the system understands. When the options of a module need to be displayed in the UI, the system looks up the template which has the same name as the module and uses the script contained in it to build an interface window which gives the user access to the module's parameters. An icon must also be created for each module in order for it to appear correctly in the UI.

Modules appear in the system in the form of command tags. Feature tags can be added in the same manner, with the only difference being that they do not have underlying modules – only UI templates.

4.8 Rig Templates

These are scripts which store all the information related to a certain rig. First and foremost, a rig template contains information about the joints – positions, rotations, orientations etc. Secondly, it contains the tags attached to each joint. When a rig template is loaded and processed, the system first looks at the joint information and builds the skeleton, then adds all the tags and finally executes commands contained within them. Essentially a rig template is the custom file format of the rigging toolkit, just as Maya ASCII and Maya binary are the custom file formats of Maya. These templates are created and modified by the Rig Editor, while the Rig Loader can only read them.

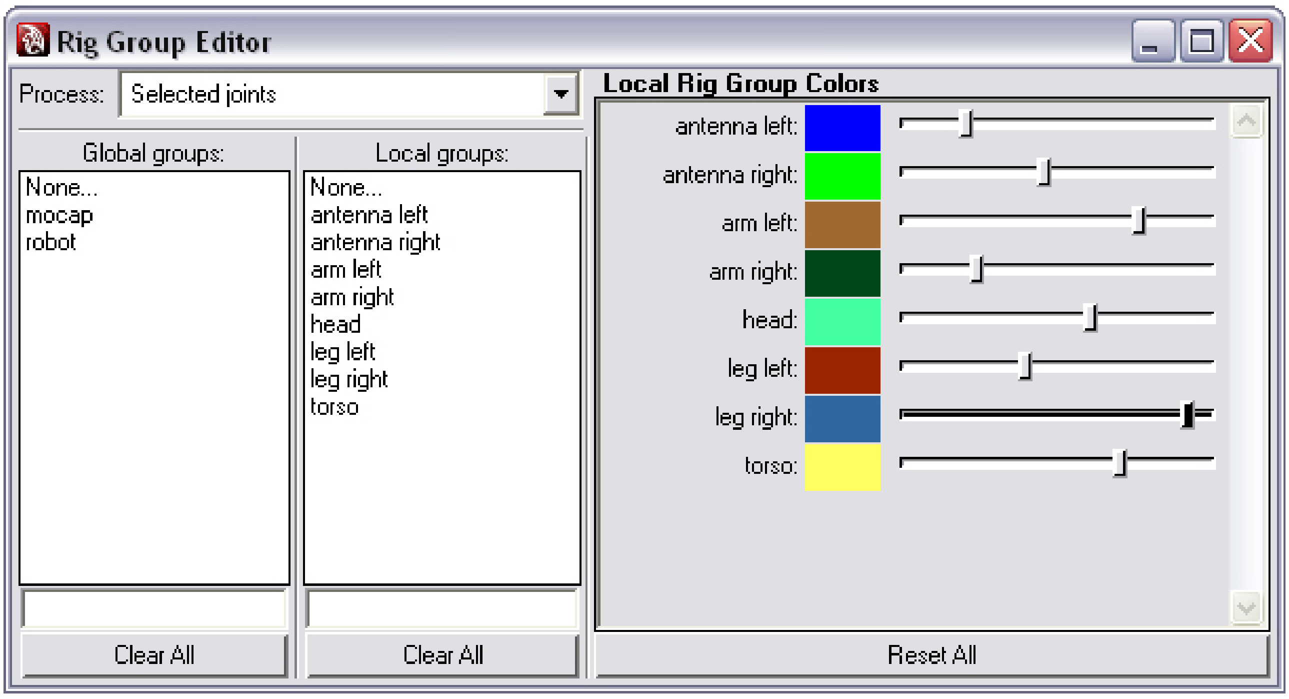
5. Rigging Tools

The rigging toolkit is composed of four tools. Each tool is independent and has its own separate purpose within the rigging process. These tools are illustrated and described below. A detailed explanation of how to use them is contained in Appendix A.

5.1 Rig Group Editor

This is a simple tool which can be used to quickly split up a skeleton into global and local groups. This procedure can also be done in the Rig Editor, but this specialised tool makes it faster and more intuitive.

The tool window is composed of two sections. The left section allows the user to define which joints are to be used for the group assignment process and specify the names of the groups. The right section can be used to test how the skeleton has been split up into local groups by changing colours associated with different groups and looking at how those colour changes are reflected on the skeleton.



Process: Selected joints

Global groups:
None...
mocap
robot

Local groups:
None...
antenna left
antenna right
arm left
arm right
head
leg left
leg right
torso

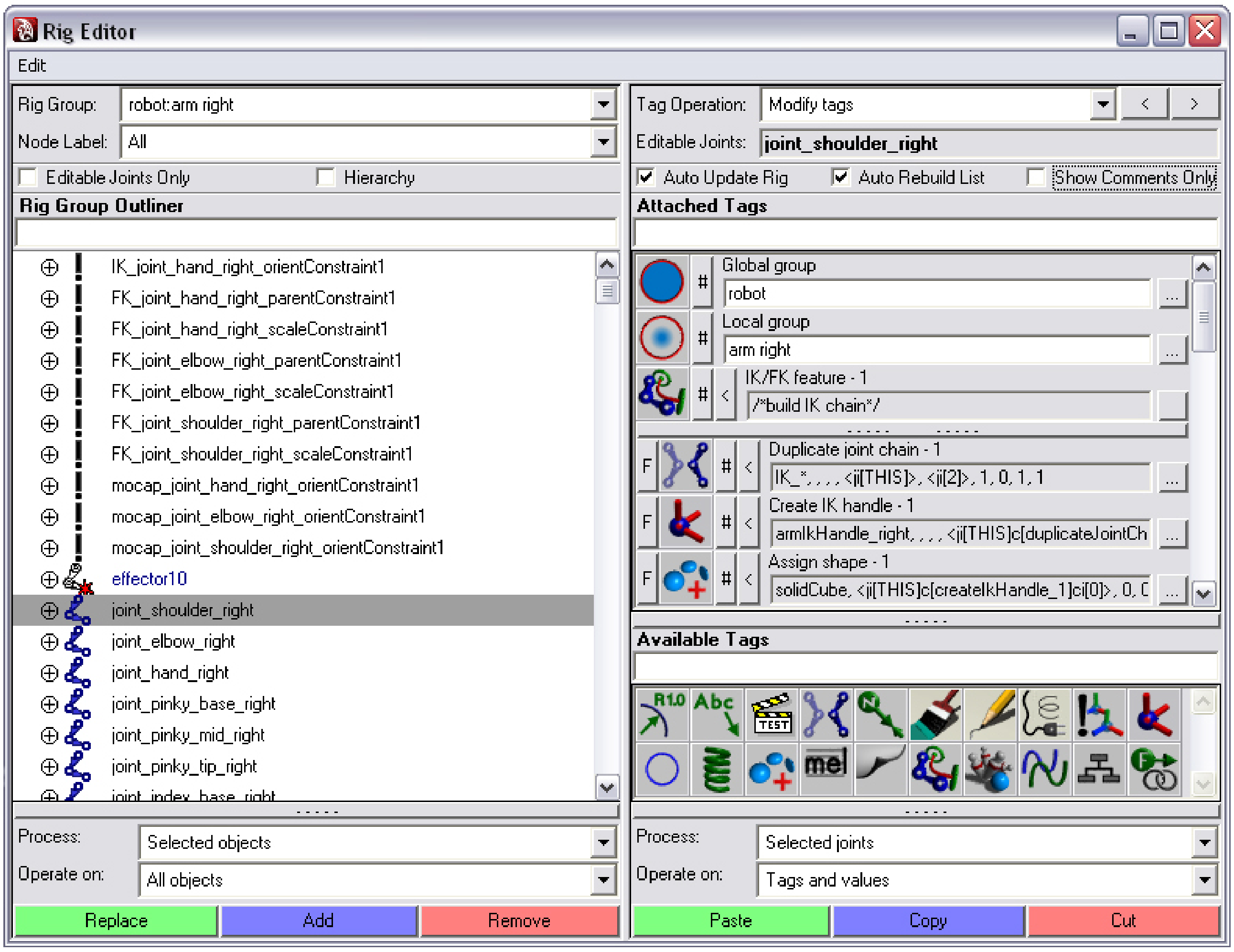
Local Rig Group Colors
antenna left:
antenna right:
arm left:
arm right:
head:
leg left:
leg right:
torso:

Figure 5.1: Rig Group Editor window.

5.2 Rig Editor

The biggest and most important tool is the Rig Editor. It allows rigs to be constructed procedurally by working with tags. Using this tool tags can be attached to joints and modified, command tag parameters can be specified, commands can then be processed and rig templates can be exported. It even allows things such as copying and pasting of tags in order to quickly mirror a rig across from one side to the other.

This tool's window has two sections. On the left is the rig group outliner. It works in a similar way to the default Maya outliner, with the main difference being its ability to only display nodes related to a specific rig group, as well as other useful node filtering and selection functions. On the right of the window is the tag parser which allows the user to work with tags attached to joints. This section is split into three subsections or rig building steps – modify tags, process tags and edit templates. First comes the "modify tags" section. This is used to add, edit and delete tags from joint containers. By selecting a joint or set of joints, any tags attached are displayed in the form of a stack. The tags can be dragged and dropped to change their order in the stack, thus changing the order of command execution. The "process tags" section is used to build a list of commands (by examining the command tags in a rig) and executing them. It allows the user to step back and forth through commands much like using a VCR player. This approach makes building and debugging rigs an intuitive process. Finally, the "edit templates" section can be used to export rigs as templates. It can also be used to load entire templates into the scene or to load specific tags contained within the template.


Rig Editor

Edit

Rig Group: robot:arm right

Node Label: All

☐ Editable Joints Only ☐ Hierarchy

Rig Group Outliner

IK_joint_hand_right_orientConstraint1
FK_joint_hand_right_parentConstraint1
FK_joint_hand_right_scaleConstraint1
FK_joint_elbow_right_parentConstraint1
FK_joint_elbow_right_scaleConstraint1
FK_joint_shoulder_right_parentConstraint1
FK_joint_shoulder_right_scaleConstraint1
mocap_joint_hand_right_orientConstraint1
mocap_joint_elbow_right_orientConstraint1
mocap_joint_shoulder_right_orientConstraint1
effector10
joint_shoulder_right
joint_elbow_right
joint_hand_right
joint_pinky_base_right
joint_pinky_mid_right
joint_pinky_tip_right
joint_index_base_right

Process: Selected objects

Operate on: All objects

Replace | Add | Remove

Tag Operation: Modify tags

Editable Joints: **joint_shoulder_right**

☑ Auto Update Rig ☑ Auto Rebuild List ☐ Show Comments Only

Attached Tags

Global group
robot

Local group
arm right

IK/FK feature - 1
/*build IK chain*/

Duplicate joint chain - 1
IK_*, , , , <j[THIS]>, <j[2]>, 1, 0, 1, 1

Create IK handle - 1
armIkHandle_right, , , , <j[THIS]c[duplicateJointCh

Assign shape - 1
solidCube, <j[THIS]c[createIkHandle_1]c[0]>, 0, C

Available Tags

Process: Selected joints

Operate on: Tags and values

Paste | Copy | Cut

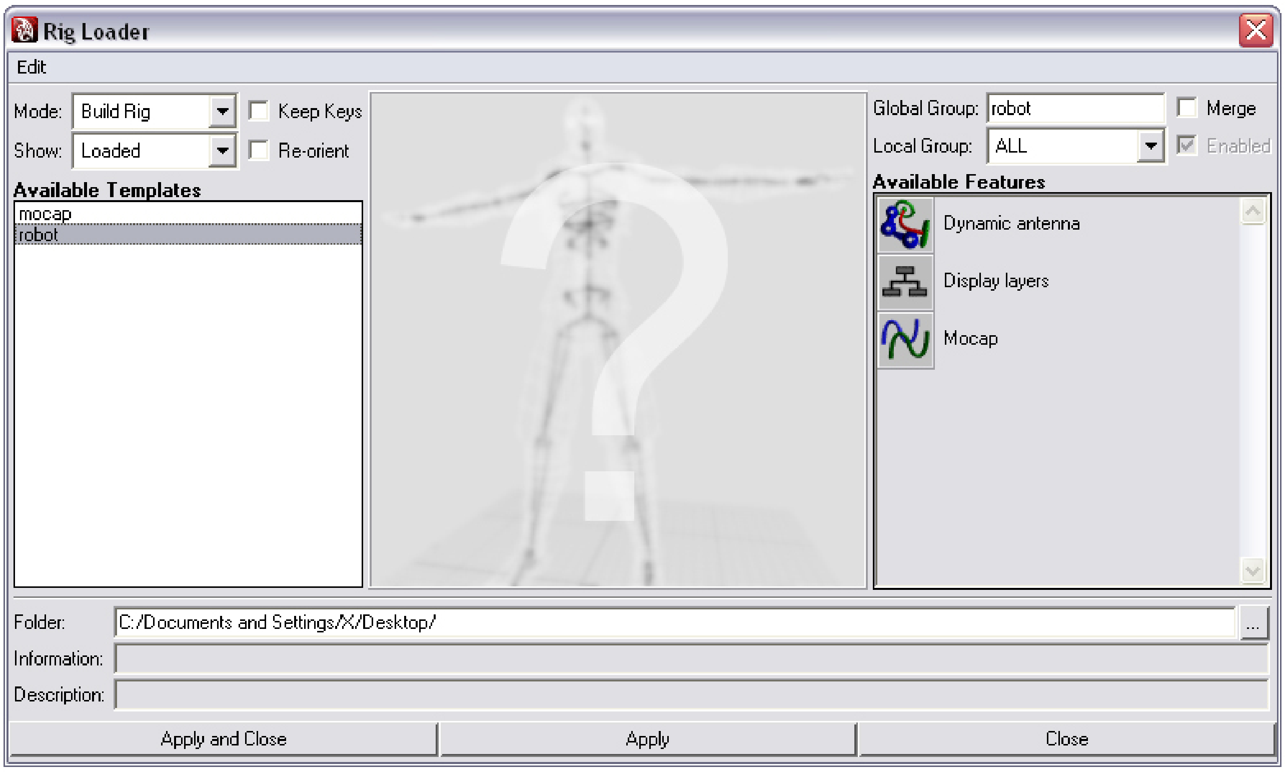
Figure 5.2: Rig Editor window.

5.3 Rig Loader

This tool is intended to be used primarily by animators and any other users of the created rigs. It features a simple interface for loading rig templates. Furthermore, it allows the user to modify the shape of a loaded rig's skeleton, as well as providing a clean way of deleting an entire rig. Also it gives access to features defined in the template so that the user can select which features of the rig are to be enabled, if there are any.

The Rig Loader window is split up into three parts. On the left a rig, or several rigs, can be selected in a list and one of four operations to be performed on those rigs can be chosen – load skeleton, edit skeleton, build rig or delete rig. It is possible to go back and forth between these operations at any time. Also, the user can seamlessly select either a template which has not yet been loaded or one which is already present in the scene. In the middle of the window is a preview image of the rig. If there is a JPEG image stored in the same folder as the template and that image has the same name, then the system uses it as a preview image. The size of the image has to be 300 by 300 pixels in order for it to display correctly. On the right side of the window is a section which shows the available features of a rig and allows the user to turn them on and off. Different parts of the rig (arms, legs etc.) can also be turned on and off here.

It is important to note that if a rig is present in the scene, but has not yet been exported as a template, it can still be accessed in the Rig Loader. Its global group will be used in place of the template – information will be read directly from the rig, as opposed to reading it from the associated template file.


Rig Loader

Mode: Build Rig
Show: Loaded
Global Group: robot
Local Group: ALL
Folder: C:/Documents and Settings/X/Desktop/

Figure 5.3: Rig Loader window.

5.4 Rig Updater

This basic tool allows the user to quickly check if any rigs loaded in the scene have updates in their source template files. When a template is exported a version number is stored in it. This number can then be modified when the template is edited. The Rig Updater compares numbers of loaded rigs to numbers in the template files. If an update is found the user is notified of this and presented with the option to load those updates.

6. Code Overview

The rigging toolkit is built on a substantially large code base. This section presents some of the more important aspects of the code.

6.1 Organisation and Naming Conventions

Each function of the code is contained in its own separate file. The filename is identical to the name of the function. There are several types of functions – core, UI, shape, module and module SDK. Core functions form the main part of the system's work structure, while UI functions are responsible for building the interface and responding to user interaction with the UI. Module functions are the custom modules described earlier in this document. Module SDK functions are some common useful procedures that can be used within modules. Finally, shape functions create different shapes which can be attached to rig components.

Function names follow a certain convention. They all have a prefix "rigging", which is followed by either "internalFunction" or "responseFunction". Internal functions are called by the code itself, while response functions are called when the user interacts with the interface. User interface functions are denoted by "UI". Finally, the actual name of the function follows. Here is an example of this naming convention...

rigging_responseFunction_UI_copyPasteTag

This is a function which is called whenever the user performs a copy-paste operation on tags using the Rig Editor interface. The use of such a naming convention makes the source code easier to manage and navigate.

6.2 Code Flow Outline

Below is a description of code execution steps in some key scenarios.

Defining Rig Groups

1. Add identifier tag
2. Convert user-accessible identifier to system identifier
3. Remove existing system identifier and replace with new one
4. Add rig group to global array if it is not already included

Adding and Editing Command Tags

1. Build commands list using existing tags
2. Roll back commands to destroy the rig
3. Add or edit tag
4. Build commands list using updated tags
5. Execute commands to build updated rig

Loading Rig Template

1. Read next line in template and build joint
2. Add tags to joint
3. Process identifier tags
4. Go to step 1 until entire template is loaded
5. Perform joint parenting
6. Build list of commands based on tags added to the joints
7. Execute list of commands to build the rig

6.3 Nameless Scene Traversal Implementation

This concept is based on the Maya "ls" command. This command can be used to list all objects in the scene which match certain criteria. By using it to find all objects that have a certain attribute attached, nodes can be found without resorting to lookup by name. For example, the following command returns a list of all joints in the scene which have the attribute "attrABC" attached...

```
ls -objectsOnly -type "joint" "*.attrABC"
```

6.4 Rig Template Scripts

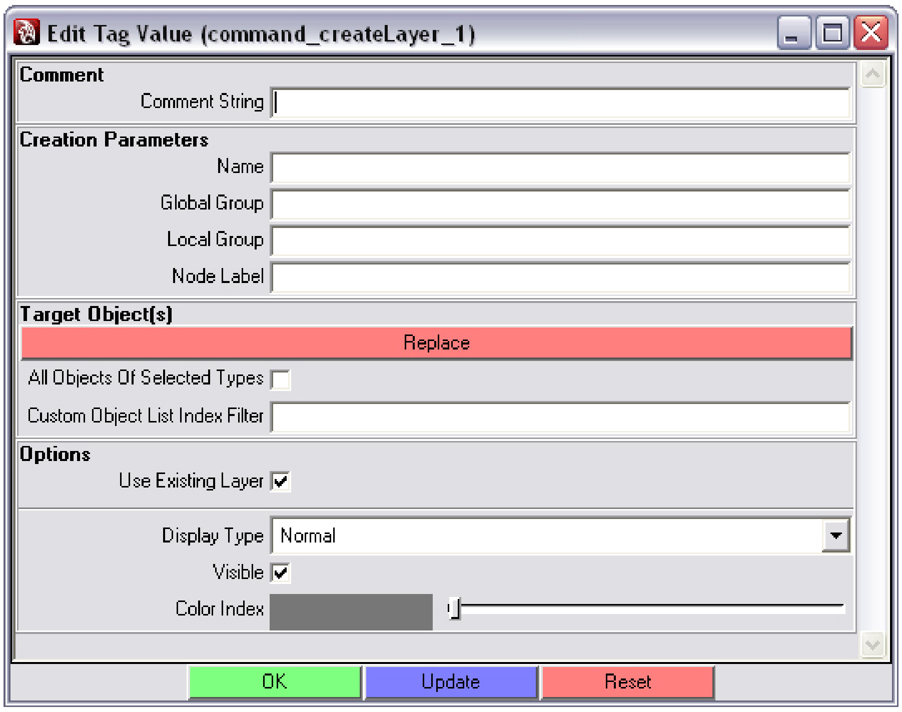
These files start with a header which contains overall information about the rig contained in the template – how many joints there are, what features are available etc. The header is read by the Rig Loader to quickly display information about a specific template selected in the list. What follows is a line-by-line representation of the rig. On each line a joint is first specified, followed by tags attached to that joint. By stepping through each line the rig is gradually loaded.

An important thing to note here is "unique identifiers". These are long random numbers generated for each joint whenever a rig template is exported. They allow the system to link joints present in the scene with joints stored in a template. This is necessary in order to perform update operations. To make sure that tags from a template are being attached to the correct joints in the scene, unique identifiers are examined and compared. This is especially useful when dealing with rigs constructed from several templates. For example, an arm template can be imported twice, along with a leg template. These can then be attached to an existing torso and head, thus forming a complete skeleton. Unique identifiers allow the system to trace every joint in the combined rig to its source template in such cases.

6.5 UI Template Scripts

In order to accommodate the needs of a module-based system, a simple scripting language was devised for defining UI layouts of parameters associated with a particular module. These scripts are parsed whenever a window is opened with options of a certain tag in the Rig Editor. The scripts consist of layout commands and parameter commands. An example of such a script is presented below. This is the script used to build an interface for the "createLayer" command tag.

```
frameLayout (Comment)
commentField (Comment String)

frameLayout (Creation Parameters)
textField (Name)
textField (Global Group)
textField (Local Group)
textField (Node Label)

frameLayout (Target Object(s))
identifierField

frameLayout (Options)
checkBox (Use Existing Layer): 1

separator

optionMenu (Display Type): Normal, Template, Reference
checkBox (Visible): 1
colorIndexSlider (Color Index)
```


Edit Tag Value (command_createLayer_1)

Comment
Comment String

Creation Parameters
Name
Global Group
Local Group
Node Label

Target Object(s)
Replace
All Objects Of Selected Types
Custom Object List Index Filter

Options
Use Existing Layer ✓

Display Type: Normal
Visible ✓
Color Index

OK | Update | Reset

Figure 6.1: Resulting window built by the above UI template script.

6.6 Code Evaluation

Overall, the code is sufficiently well structured and efficient for the given purpose. The approach taken when developing the tools allows for easy extensibility, as well as debugging of existing code. One issue to note with regards to how the code was written is the inclusion of some workarounds to avoid certain issues with how MEL commands work under Windows. These workarounds would have been unnecessary if it wasn't for some bugs in the core Maya code. In terms of portability there are issues which arise under Linux-based operating systems. These issues are only related to UI code, which works differently under different operating systems. Once again, this is an issue that arises due to errors within Maya itself. Also, an object-oriented approach to developing this toolkit would have been beneficial, but MEL does not have that capability and using the C++ API was too risky due to time constraints.

7. Conclusion

The intention of this project was to implement a fully functional prototype of a flexible procedural rigging solution. This goal was achieved with satisfying results. The developed tools work as intended and are almost ready to be used in a real-life production environment. The rigging process introduced by this solution is innovative and well thought out. The decision to split up the rigging task among several tools was correct, because otherwise, if a single tool was developed, it would have been too complex. Also, the module-based structure works well and allows for easy further development.

Along with benefits offered by this solution, there are also certain limitations that arose mostly due to the lack of development time. First of all, MEL code is inherently slow and the execution of certain functions could be speeded up dramatically if they were written in C++. Since the toolkit features interactive updates of rigs such speed improvements could be extremely beneficial for the usability of these tools. Also, the UI still needs some improvements, the most beneficial of which would be the addition of a graph-based interface for the process of rig construction. This approach would make the task much more intuitive and manageable than the current stack-based interface. Furthermore, the interactive rig updates could be improved dramatically by implementing a dynamic approach to the rig reconstruction process. Currently the entire rig is rebuilt during each update. A dynamic approach to this would be the possibility to rebuild only those parts of the rig that need to be changed. This could be implemented by tracking inputs and outputs of commands and only executing those commands whose data has changed somehow, rather than processing the entire list. This functionality combined with C++ code could potentially make the rig update process almost real-time.

Future work on this project would involve implementing the improvements described above, as well as the gradual conversion of all the code to C++ with the aid of Maya API. An additional extension to the functionality could be an intuitive way of defining and using templates for entire parts of a rig. This could be implemented by adding an additional menu with a list of commonly-used templates that can quickly be loaded onto a set of selected joints, such as an IK/FK arm setup. Such improvements would make this toolkit a truly production-ready tool, ready to form the base of a solid animation pipeline.

Appendix A. User Guide

This section presents a more detailed overview of how the procedural rigging toolkit works and how to use it.

A.1 General Procedure

As mentioned previously, the rigging toolkit consists of four tools which perform separate functions. When using the toolkit to its full extent, the procedure followed involving all those tools would be similar to the following steps...

Build skeleton: Using default Maya functionality or other custom tools, a skeleton is constructed.

Assign rig groups: The Rig Group editor can now be used to assign a global group to the skeleton. This will isolate the attached rig from other rigs in the scene. Local groups are also assigned to separate different parts of the skeleton, such as arms and legs.

Add rig construction tags: Using the Rig Editor, the skeleton can now be filled with tags which build up the rig. Command tags are the ones that build the actual rig components. Feature tags can be used to combine commands into groups that can be switched on and off to determine which parts of the rig are built and which are not.

Export rig template: Once the rig has been built and works as intended, it is exported as a rig template, also via the Rig Editor. This template file is usually stored on a server accessible by animators (in a production environment).

Load rig template: The Rig Loader is used to load templates and manipulate the resulting rigs. Any features incorporated into the rig, for example, can be turned on and off using this tool.

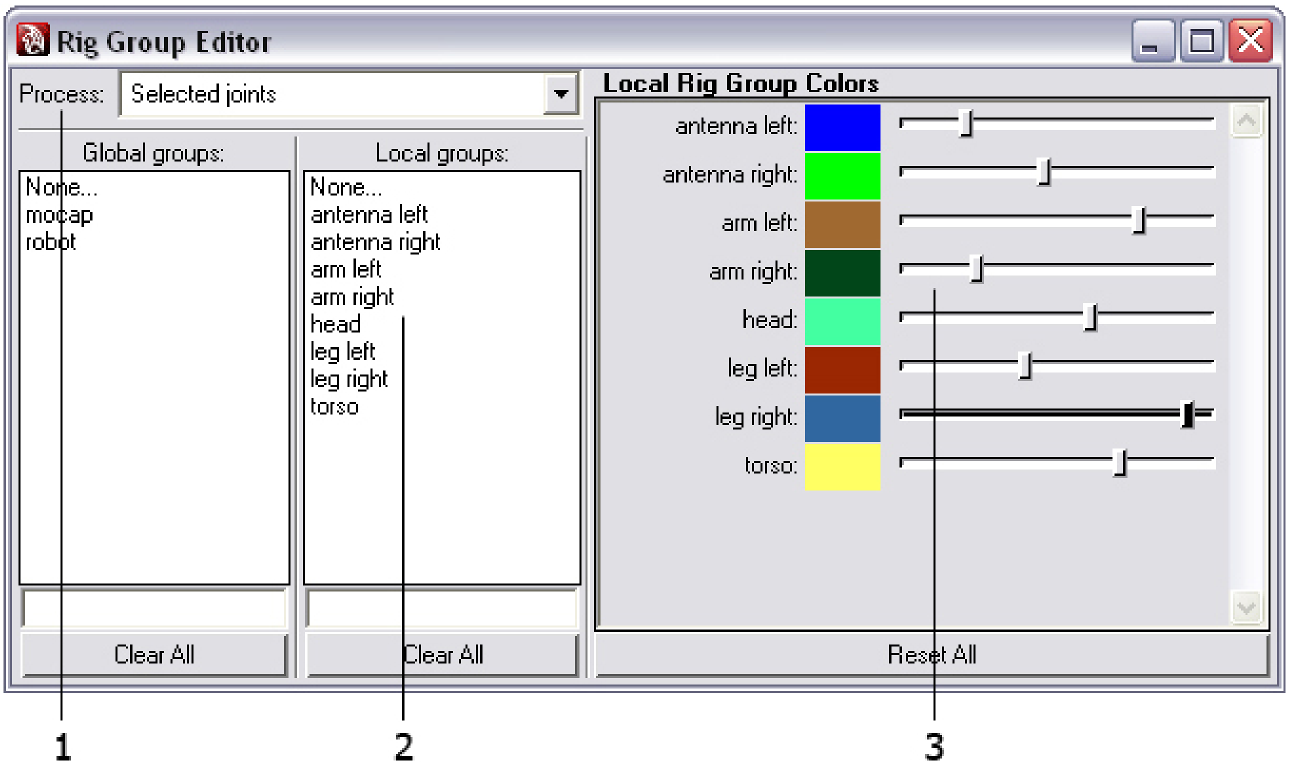
Perform animation: The rig is now ready to be animated.

Distribute update: The original template may be modified using the Rig Editor at this stage. Once modifications have been made, the template is uploaded back on the server, replacing the existing one.

Load update: Modified templates can be loaded using the Rig Updater. It automatically looks at all the rigs in the scene and checks relevant template files to see if changes have been made. These changes can then be loaded, resulting in updated rigs being rebuilt.

Continue animation: Once updates have been loaded, animation can be resumed from the point at which it was left off. If rig modifications were integrated correctly, then existing keyframes are not lost during the rig update process.

A.2 Rig Group Editor

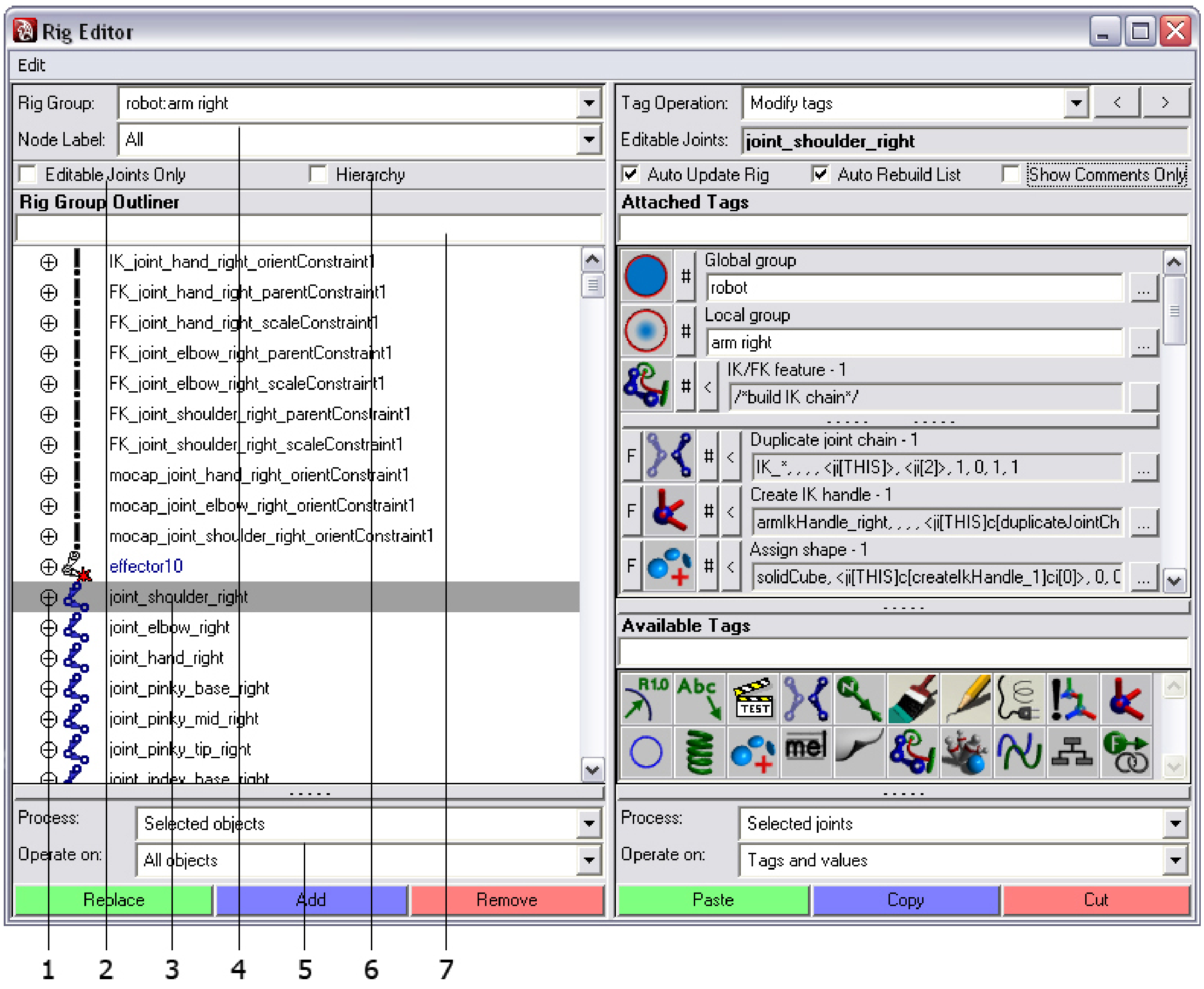


(1) Selection processing option: This drop-down menu allows the user to specify which joints are to be used for the rig group assignment operation. There are three options: selected joints, selection hierarchy and all joints in scene.

(2) Group assignment panel: Once the desired joint selection has been made, this panel can be used to assign groups to the selection. There are two lists displaying groups which are present in the scene. By clicking on an item in the list that group will be assigned. Alternatively, using the text input boxes below the lists, new group names can be specified. The "Clear All" buttons strip all group identifiers from all joints in the scene.

(3) Group colour selection panel: A list of all local groups in the scene is presented in this panel. By dragging the sliders next to the colour preview boxes, joints assigned to the relevant local groups will change colours to match the slider settings. This is used for testing purposes to see if skeletons were split up into local groups correctly. The "Reset All" button is used to reset all local group colours to the default setting.

A.3 Rig Editor - Rig Group Outliner



Rig Group: robot:arm right
Node Label: All
☐ Editable Joints Only ☐ Hierarchy

IK_joint_hand_right_orientConstraint1
FK_joint_hand_right_parentConstraint1
FK_joint_hand_right_scaleConstraint1
FK_joint_elbow_right_parentConstraint1
FK_joint_elbow_right_scaleConstraint1
FK_joint_shoulder_right_parentConstraint1
FK_joint_shoulder_right_scaleConstraint1
mocap_joint_hand_right_orientConstraint1
mocap_joint_elbow_right_orientConstraint1
mocap_joint_shoulder_right_orientConstraint1
effector10
joint_shoulder_right
joint_elbow_right
joint_hand_right
joint_pinky_base_right
joint_pinky_mid_right
joint_pinky_tip_right
joint_index_base_right

Process: Selected objects
Operate on: All objects

Tag Operation: Modify tags
Editable Joints: joint_shoulder_right
✓ Auto Update Rig ✓ Auto Rebuild List ☐ Show Comments Only

Global group: robot
Local group: arm right
IK/FK feature - 1: /*build IK chain*/
Duplicate joint chain - 1: IK_*, , , , <j[THIS]>, <j[2]>, 1, 0, 1, 1
Create IK handle - 1: armIkHandle_right, , , , <j[THIS]c[duplicateJointCh
Assign shape - 1: solidCube, <j[THIS]c[createIkHandle_1]c[0]>, 0, C

Process: Selected joints
Operate on: Tags and values

(1) Show attached tags button: This button can be used to show or hide the list of tags attached to a node.

(2) Editable joints option: Editable joints are those to which tags can be attached. All joints created by commands while the rig is built are non-editable, since they are not originally present in the scene. This option box is used to specify whether all joints should be displayed in the outliner, or only editable joints.

(3) Outliner nodes list: The rig group outliner displays nodes contained in a certain rig group. By selecting nodes in the outliner, they will also be selected in the scene. If an editable joint is selected its list of attached tags will be displayed in the tag parser which is located on the right half of the window. Multiple editable joints can be selected, which results in a compacted list of all their tags being displayed, allowing the user to edit all the tags at once.

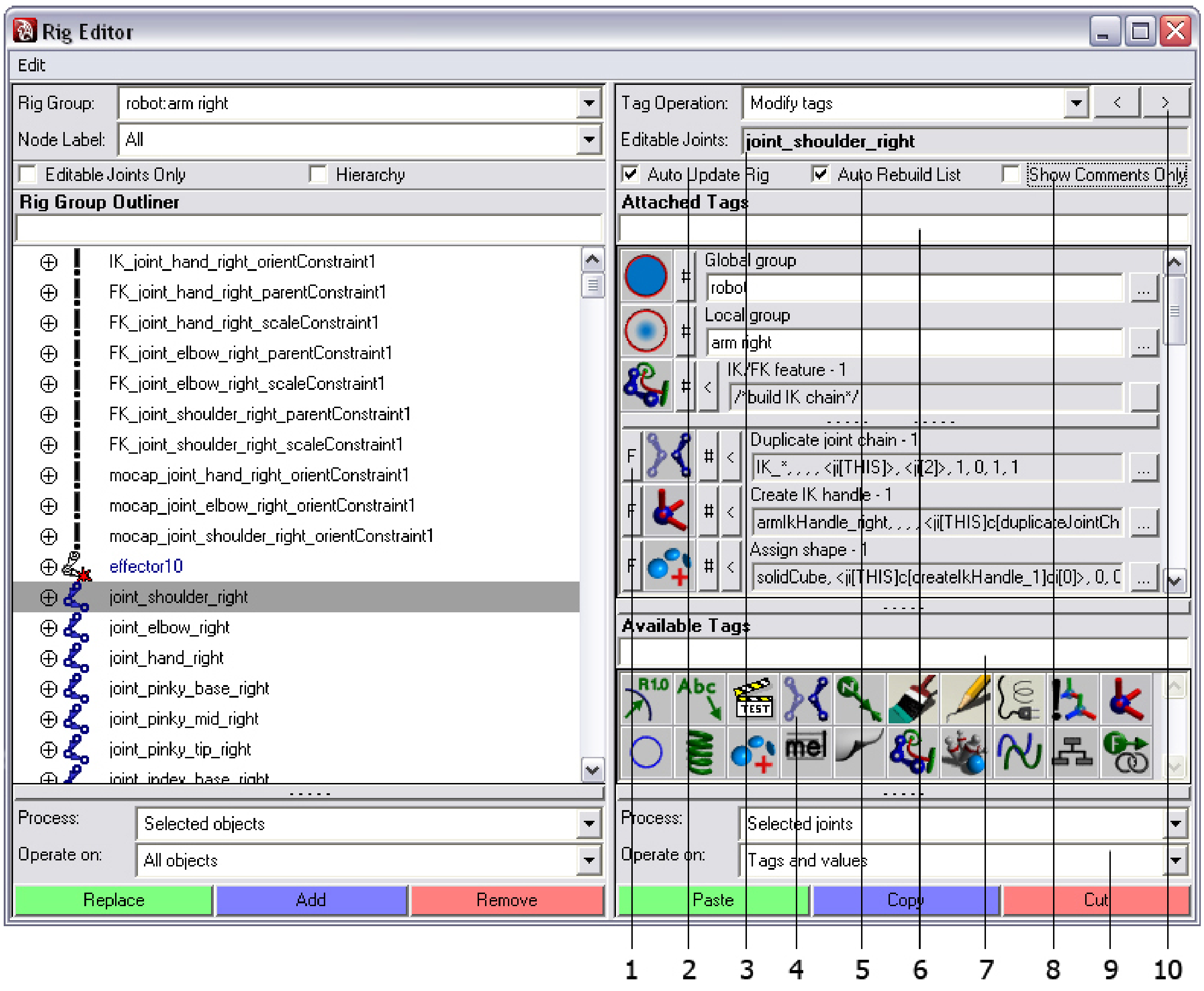
(4) Rig group selection panel: The outliner displays nodes belonging to a rig group selected in this panel. When a certain rig group is selected a list of all node labels available in that group is placed into the drop-down menu below. Selection of a specific node label causes the outliner to only show nodes which have that node label assigned.

(5) Selection operations panel: A set of controls for performing various advanced selection operations is provided in this panel. It allows the selection of editable joints, non-editable joints, objects other than joints and even joints that contain the command which was executed to create certain nodes (this is done by switching the "operate on" drop-down menu to "source editable joints").

(6) Hierarchy option: The outliner can be instructed to display nodes in a hierarchical way or as a simple list. IMPORTANT: if a node is not visible in the list because it is hidden within the hierarchy, then it cannot be selected in the scene. This is why it is recommended to always turn this option off, so that a simple list is displayed.

(7) Displayed nodes list filter: This text field accepts node type names (full or partial) and filters the list displayed in the outliner by matching the input strings to types of nodes contained in the selected rig group. For example, entering "joint cons expr" in this text box will cause the outliner to only show joints, constraints and expression nodes.

A.4 Rig Editor - Modify Tags Section



Rig Group: robot:arm right
Node Label: All
Editable Joints Only
Hierarchy
Rig Group Outliner

IK_joint_hand_right_orientConstraint1
FK_joint_hand_right_parentConstraint1
FK_joint_hand_right_scaleConstraint1
FK_joint_elbow_right_parentConstraint1
FK_joint_elbow_right_scaleConstraint1
FK_joint_shoulder_right_parentConstraint1
FK_joint_shoulder_right_scaleConstraint1
mocap_joint_hand_right_orientConstraint1
mocap_joint_elbow_right_orientConstraint1
mocap_joint_shoulder_right_orientConstraint1
effector10
joint_shoulder_right
joint_elbow_right
joint_hand_right
joint_pinky_base_right
joint_pinky_mid_right
joint_pinky_tip_right
joint_index_base_right

Process: Selected objects
Operate on: All objects

Tag Operation: Modify tags
Editable Joints: joint_shoulder_right
[✓ Auto Update Rig] [✓ Auto Rebuild List] Show Comments Only
Attached Tags

Global group
robot
Local group
arm right
IK/FK feature - 1
/*build IK chain*/
Duplicate joint chain - 1
IK_*, , , , <j[THIS]>, <j[2]>, 1, 0, 1, 1
Create IK handle - 1
armIkHandle_right, , , , <j[THIS]c[duplicateJointCh
Assign shape - 1
solidCube, <j[THIS]c[createIkHandle_1]d[0]>, 0, 0

Available Tags

Process: Selected joints
Operate on: Tags and values

(1) Attached tag controls: Any tags attached to a joint are presented in the Rig Editor as shown in this example. The tag controls consist of several items which can be used to perform all the necessary modifications on any specific tag.

The "F" button appears next to command tags which are attached to feature tags. By clicking this button command tags are detached from features.

The main icon is a distinct visual representation of a tag. Clicking this icon deletes the tag. Dragging and dropping it onto another icon in the list shifts the tag position – this way execution orders can be modified. Command tags are executed in a top-to-bottom order. Dragging and dropping a command onto a feature performs a connection to that feature tag. The feature connection can also be performed by dragging and dropping onto a command which is connected to a feature. IMPORTANT: drag and drop operations are done using the middle mouse button.

The "#" button copies its associated tag, allowing it to be pasted later on.

The "<" button can be clicked to turn a tag on and off. This determines whether commands are executed or not. Turning feature tags on and off also affects all the command tags attached to them.

Across the top is a text label displaying the current tag's type (eg: Global group, IK/FK feature etc.) along with an index – if multiple tags of the same type are attached to a joint, they are all assigned a unique index so that the system can differentiate between them. Identifier tags can only be attached once, hence they do not have an index.

Across the middle is a text field which shows the contents of the string stored in a tag. Because tags are essentially hidden string attributes, the data they contain is stored in a string. The system reads these strings and parses them using a specialised function to extract the necessary data from each tag. When this string cannot be edited manually the text field is disabled. In such cases the tag value button on the left side is used.

Editing of tag values is done by clicking the button on the right side. It brings up a separate window which provides access to all the options relevant to that specific tag (determined by the UI template described earlier). This part of the interface is described in section A.7 – Tag Value Editor.

A long bar-like button appears across the bottom on feature tags. Because they essentially group command tags together, this bar button can be clicked to show and hide the attached command tags. This has no effect on command execution and is only used for managing the visual layout of tags in the interface window. It is useful when there are too many tags displayed at once.

(2) Auto update rig option: When this option is enabled the rig is automatically rebuilt whenever a tag modification occurs. This can be the deletion of a tag or simply a change of some parameter.

(3) Editable joints selection field: This text field shows the name of the currently selected editable joint (its contents are being displayed in the tag parser).

(4) Available tags section: Tags which can be attached to a currently selected joint are displayed in this section. Identifier tags can only be attached once to every joint, hence they disappear from this box after attachment. In order to attach a tag the icon can either be clicked, or it can be dragged and dropped onto one of the tags which are already attached.

(5) Auto rebuild list option: This option determines whether the list of commands is rebuilt during each automatic rig update, or whether the existing list is used. Normally this option should always be enabled, unless only tag parameters are being modified.

(6) Attached tags list filter: This text field is used for filtering what is displayed in the attached tags section. Just as the filter works for the outliner, strings entered in this field are matched to tag types in the list. Entering "feat com", for example, would display only features and command tags.

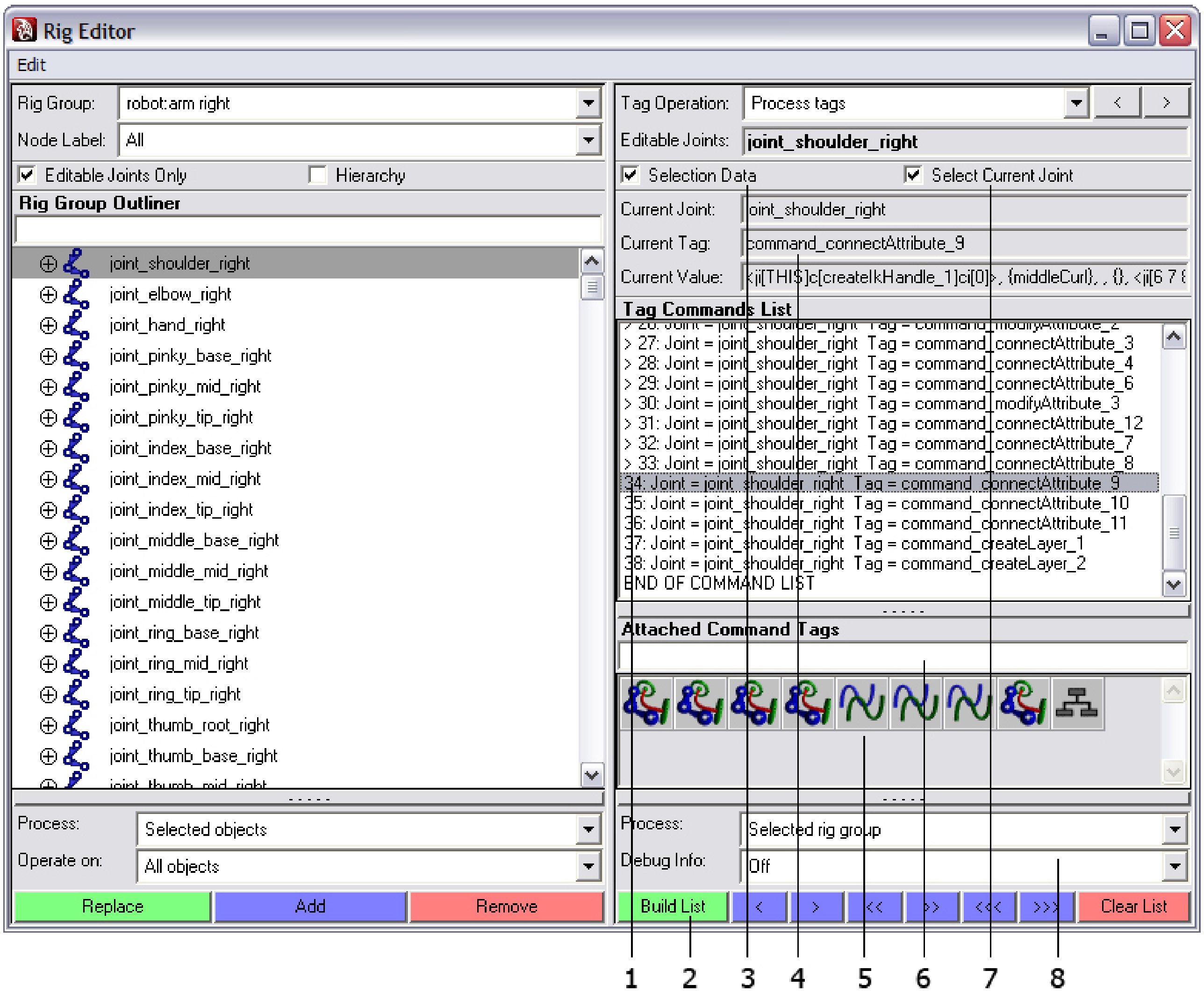
(7) Available tags list filter: Just as the filter described above, this one filters what tags are displayed in the available tags section. Note that apart from entering generic tag types, they can also be matched by specific parts of the tag type name – if you only wish to diplay tags related to IK, for example, then enter the text "ik" in the filter.

(8) Show comments only option: This option can be used to show only comments in the tag value text fields of attached tags, instead of displaying the entire raw string. Comments are useful when numerous tags are attached – they can be used to describe what is going on in the rig building process more clearly.

(9) Copy-paste operations panel: Tags can be copied and pasted just as files in an operating system or text in a word processor. This panel allows copying and pasting either tags, tag values (the raw strings) or both. Also, you can specify whether to perform the copy-paste operation on the selected joints only, on the selection hierarchy or on the entire list of joints contained in the rig group displayed in the outliner. IMPORTANT: to copy or cut only certain tags, you can use the filter of the attached tags section (6). Only tags which are currently being displayed will be copied.

(10) Tag parser section switch: The drop-down menu and left/right buttons can be used to switch between the different sections of the tag parser.

A.5 Rig Editor - Process Tags Section


```
Rig Editor
Edit
Rig Group: robot:arm right
Node Label: All
[✓] Editable Joints Only    [ ] Hierarchy
Rig Group Outliner
joint_shoulder_right
joint_elbow_right
joint_hand_right
joint_pinky_base_right
joint_pinky_mid_right
joint_pinky_tip_right
joint_index_base_right
joint_index_mid_right
joint_index_tip_right
joint_middle_base_right
joint_middle_mid_right
joint_middle_tip_right
joint_ring_base_right
joint_ring_mid_right
joint_ring_tip_right
joint_thumb_root_right
joint_thumb_base_right
joint_thumb_mid_right
Process: Selected objects
Operate on: All objects
Replace   Add   Remove

Tag Operation: Process tags
Editable Joints: joint_shoulder_right
[✓] Selection Data    [✓] Select Current Joint
Current Joint: joint_shoulder_right
Current Tag: command_connectAttribute_9
Current Value: <ji[THIS]c[createIkHandle_1]ci[0]>, {middleCurl}, , {}, <ji[6 7 8
Tag Commands List
> 26: Joint = joint_shoulder_right  Tag = command_modifyAttribute_2
> 27: Joint = joint_shoulder_right  Tag = command_connectAttribute_3
> 28: Joint = joint_shoulder_right  Tag = command_connectAttribute_4
> 29: Joint = joint_shoulder_right  Tag = command_connectAttribute_6
> 30: Joint = joint_shoulder_right  Tag = command_modifyAttribute_3
> 31: Joint = joint_shoulder_right  Tag = command_connectAttribute_12
> 32: Joint = joint_shoulder_right  Tag = command_connectAttribute_7
> 33: Joint = joint_shoulder_right  Tag = command_connectAttribute_8
34: Joint = joint_shoulder_right  Tag = command_connectAttribute_9
35: Joint = joint_shoulder_right  Tag = command_connectAttribute_10
36: Joint = joint_shoulder_right  Tag = command_connectAttribute_11
37: Joint = joint_shoulder_right  Tag = command_createLayer_1
38: Joint = joint_shoulder_right  Tag = command_createLayer_2
END OF COMMAND LIST
Attached Command Tags
Process: Selected rig group
Debug Info: Off
Build List  <  >  <<  >>  <<<  >>>  Clear List
```

(1) Commands list: Rigs are built by executing this list. It shows the name of the joint along with the name of the tag that is being executed. Tags which have already been executed are defined by an arrow ">" prefix in the list. Commands can be executed backwards to undo their actions, thus destroying the rig.

(2) Execution controls: These buttons allow the user to manually build the list of commands and step through it using VCR-like arrow controls. From left to right they allow executing: a single command, all commands attached to a single joint and the entire list. There is also a button on the right which clears the list.

(3) Selection data option: This option box shows and hides the panel showing the data of the currently selected command.

(4) Selection data panel: Information about the currently selected command is presented in this panel. It shows the name of the joint that the command tag is attached to, the name of the command tag itself and the raw string contained in the tag.

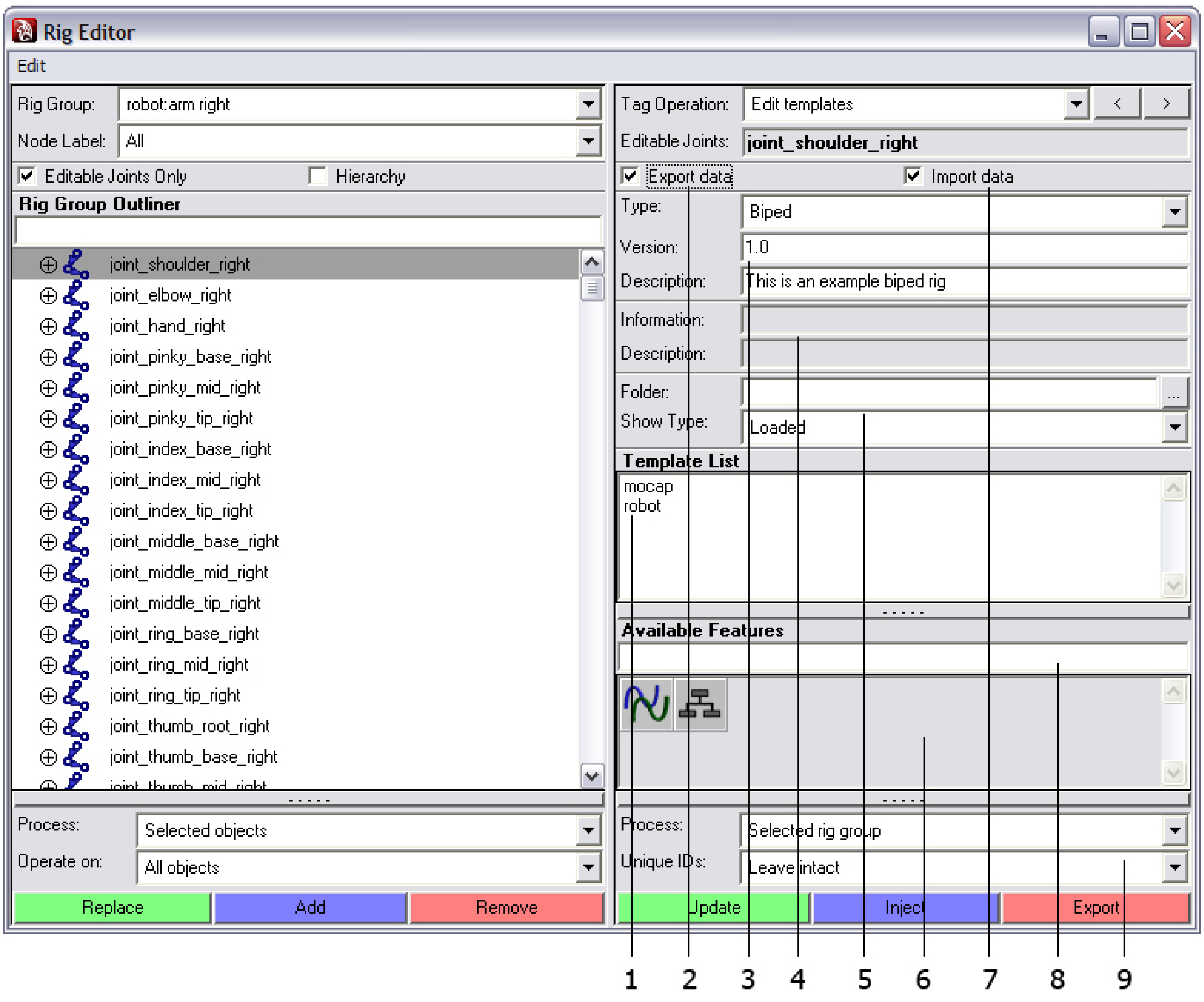
(5) Attached command tags section: Feature and command tags attached to the selected joint(s) are displayed here. By clicking on their icons they can be turned on and off (thus defining whether they will be executed or not).

(6) Attached command tags list filter: Works just like the filters described previously.

(7) Select current joint option: As the command processor steps through the list, the joint that contains the currently executed command can be selected by enabling this option.

(8) Command list operations panel: The top drop-down menu defines which joints to use when building a commands list. IMPORTANT: This option applies to both the "Build List" button and the automatic rig rebuild feature. That feature rebuilds the list based on the selection in this drop-down menu. Located at the bottom, the "Debug" menu can be used to turn on printouts of debug data during the execution of commands. This data shows useful information, such as the time taken to execute a specific command.

A.6 Rig Editor - Edit Templates Section



(1) Template list: Templates of currently selected type are displayed in this list. They can be selected to show information about them in the import data panel.

(2) Export data option: This shows and hides the export data panel – used for organising the layout of the UI.

(3) Export data panel: The text fields in this panel can be used to specify data related to a template when exporting it.

(4) Import data panel: Information about a template selected in the template list will be displayed in this panel.

(5) Template file selection panel: Here a folder in which to look for templates can be specified, along with the template types to look for. The "Loaded" template type refers to templates which are already loaded in the scene. All other types refer to templates stored in files.

(6) Available features section: Features attached to the selected joints and available for export in templates are displayed in this section. In order for features to be exportable they must be given a name. Unnamed features are simply used to group command tags together within the Rig Editor – they will not be available in the Rig Loader. This is sometimes desirable if the features are used only to organise large numbers of tags. Note that in this section all features with the same type and name will be grouped into one icon. Clicking on the icon turns the entire group of features on and off. This can be used to test how features will work before exporting them in the template.

(7) Import data option: Used for showing and hiding the import data panel.

(8) Available features list filter: Works in the same way as other filters described earlier.

(9) Template operations panel: The "Process" menu determines what list of joints is to be used in the operation.

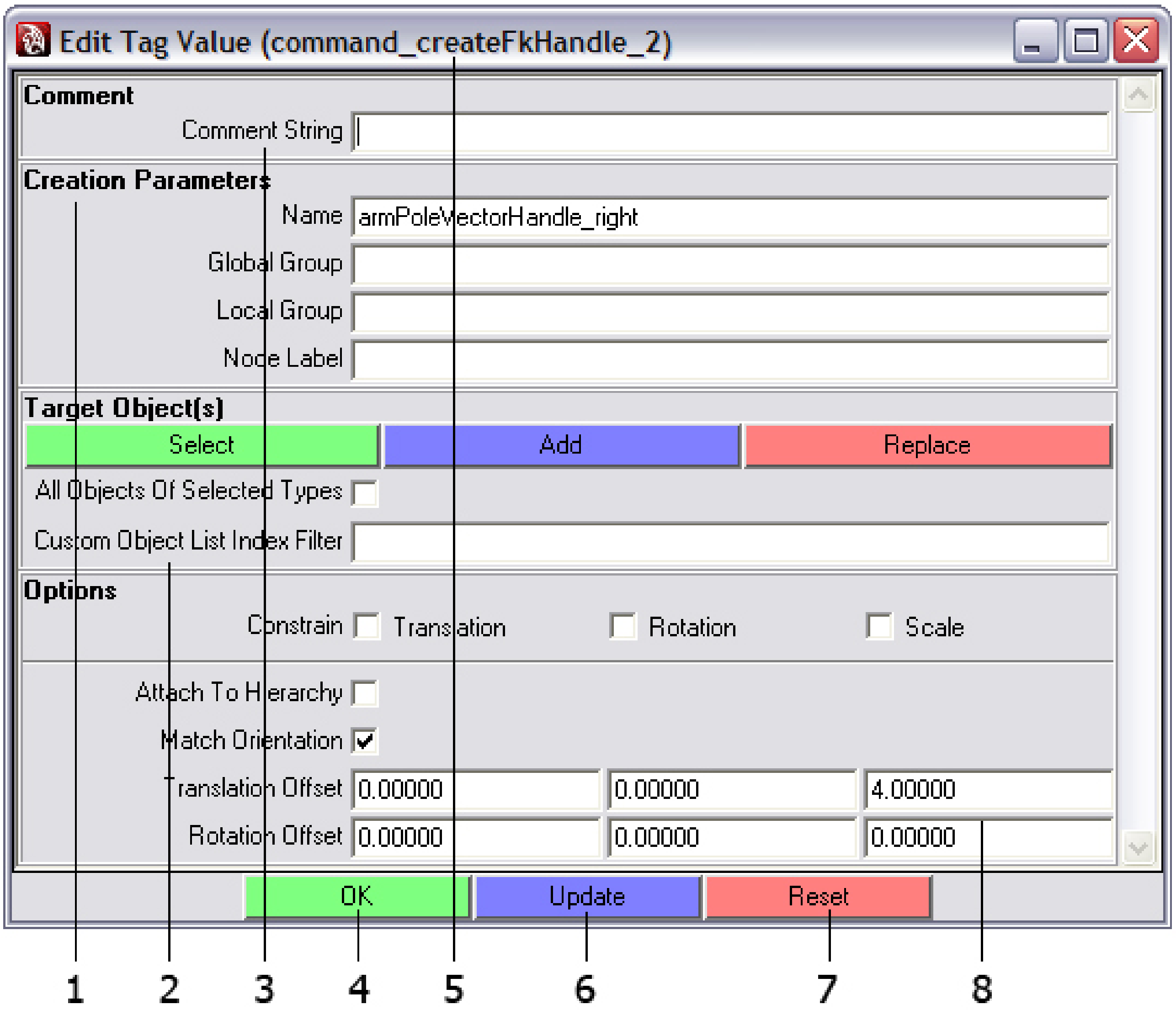
The "Unique Ids" menu is used to define whether unique identifiers are generated for joints when exporting a template. These identifiers are used by the system to keep track of which joints in the scene relate to which template file.

The "Update" button involves these identifiers. By selecting a template in the list and clicking this button, unique identifiers of selected joints will be compared with those in the template file. Whenever a match is encountered, tags on the selected joint will be replaced by tags in the template. This is essentially how the Rig Updater tool works.

The "Inject" button ignores unique identifiers and loads tags from the template into selected joints. If more joints are present in the template file than in the scene selection, then the system will keep injecting tags until the end of the selection list is reached.

Exporting of template files is done via the "Export" button.

A.7 Tag Value Editor


Edit Tag Value (command_createFkHandle_2)

Comment

Comment String: |

Creation Parameters

Name: armPoleVectorHandle_right

Global Group:

Local Group:

Node Label:

Target Object(s)

All Objects Of Selected Types ☐

Custom Object List Index Filter:

Options

Constrain ☐ Translation ☐ Rotation ☐ Scale

Attach To Hierarchy ☐

Match Orientation ☑

Translation Offset: 0.00000 0.00000 4.00000

Rotation Offset: 0.00000 0.00000 0.00000

(1) Creation parameters: These parameters are present in all command tags which create nodes. A name for the created node(s) can be specified here, along with custom identifiers. If identifiers are not specified, then those of the joint that contains this command are used.

(2) Identifier field: This is a set of controls which can often be encountered in command tags. By using these controls, nodes can be specified using the Nameless Scene Traversal principle. Whenever the actual command behind a tag is evaluated, this notation is resolved into actual node names which are passed to the command. "All Objects Of Selected Types" is an option which specifies that, within the scope of any selected objects, all other nodes of the same type are to be used. This scope can either be a rig group or a set of nodes created by a single command. The custom index filter accepts numbers and keywords to specify which indices of a resolved array of nodes should actually be passed to the command. Numbers can be specified here, separated by spaces. Also there are three keywords – START, MIDDLE and END. These can be used when array sizes vary.

(3) Comment string: Just like code comments, these can be used to make ambiguous parts of the rig tag network easier to understand.

(4) OK button: When pressed, this button stores the specified parameters in the related tag and closes the window.

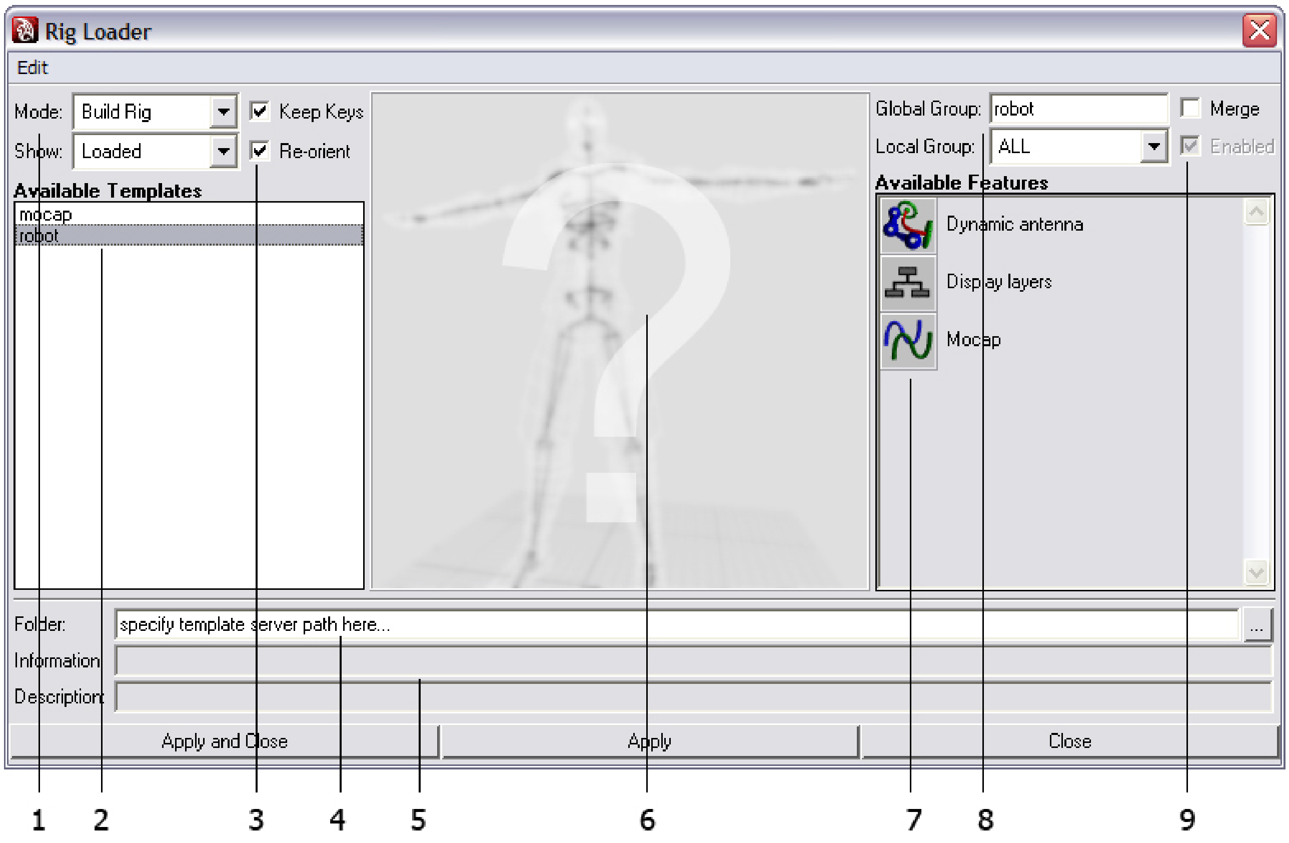
(5) Tag name: This is the name of the tag currently being edited.

(6) Update button: This button stores parameters in the tag and forces a rebuild of the rig.

(7) Reset button: Every parameter has a default value which is defined in the UI template of the given module. This button reloads the editor window with those default parameters.

(8) Miscellaneous tag parameters: Different tags have different parameters, according to what is specified in their respective UI templates.

A.8 Rig Loader



(1) Rig template operation options: The "Mode" menu is used for stepping between four rig operations. The first mode loads a skeleton without building the rig. The second mode allows editing the joint position of that skeleton. The third and fourth modes build and delete a rig respectively. These operations can be carried out on both loaded rigs and those contained in template files. Furthermore, it is possible to switch to any of these modes at any time. The mode comes into effect after the "Apply" button is pressed.

The second option menu defines which template types to show in the list. "Loaded" refers to templates which are currently present in the scene in the form of built rigs. All other types refer to external template files.

(2) Template list: Templates of currently selected type are displayed in this list. By selecting a template the UI updates to reflect data related to the selected template.

(3) Additional template operation options: These are two useful options which come into effect during a template mode switch. "Keep Keys" is an option which attempts to maintain created keyframes after a rig is rebuilt. This is done by disconnecting keyframe nodes, rebuilding the rig and connecting them back to the relevant nodes. Those nodes have to be defined by node labels. The system looks for node labels that match before and after the rig is rebuilt when attempting to reconnect keyframes. Any keyframe nodes that have not been reconnected are automatically deleted. "Re-Orient" is an option that sets the skeleton to default Maya joint orientation settings before building the rig.

(4) Template files folder: A folder in which to look for template files can be specified here. By default the predefined server path is used. This path can be specified in a special file contained in the "data" folder of the toolkit's installation directory.

(5) Template information panel: Information about a template file is stored in its header. When a template is selected in the list that header is read and information extracted from it is displayed in this panel. A rig which has been loaded from a template has hidden attributes which specify the path to its source template file. That path is used when looking for the header information.

(6) Preview image: If there is a JPEG image in the same folder as the selected template, with a name identical to that of the template file, then it will be used as a preview image. In order for it to display in the interface correctly its size must be 300 by 300 pixels.

(7) Available features list: This list displays features available within all or a specific local group of the selected template's rig. By clicking on the feature's icon that feature can be turned on and off. Also, features can have switches. The selected switch name will be displayed in brackets next to the feature name. Switches can be changed by right clicking on the feature icon and selecting one of the options in the menu that pops up.

(8) Rig group options: Using this set of controls the global group of a rig can be renamed. Also, by switching through the list of local groups, features contained in that group can be displayed in the list.

(9) Additional rig group options: The "Merge" option allows merging several selected templates into one global group. This is useful when importing several rig parts (arms, legs etc.) with the intention of combining them into a single rig. Such combinations can be done by simply parenting joints, then building the rig. The "Enabled" option box specifies which local groups should be enabled. Disabling a local group means that the joints in that group will be permanently deleted in the scene.

A.9 Rig Updater

This simple tool checks for rig template updates. First of all, it examines the rigs loaded in the scene, looking at hidden attributes which define the names of template files that the rigs were loaded from, as well as the version numbers those templates had. Then the system looks for those files to see if their version numbers have changed. If that is the case, then the rig is destroyed, tags from the updated template are loaded, replacing the existing ones, and the rig is built again. This is done optionally – if an update is found the user is presented with a window that prompts whether the update should be loaded or not. Note that the system tries to keep animation keyframes using the technique described in the previous section (Rig Loader description, UI element 3).

Appendix B. Tag Types

This section briefly describes the various tag types available for rig construction with the Rig Editor.

B.1 Identifier Tags

These tags define which rig group an object belongs to. They only have one parameter – the group name, or node label.

B.2 Feature Tags

In order to make rigs more flexible, feature tags are used. They combine command tags into groups. Manipulations performed on the feature tag are propagated to all attached command tags. These tags have two parameters – feature name and switch.

By specifying a feature name, that feature becomes visible in the Rig Loader interface (without a name it can still be used for tag organisation purposes). Features of the same type and name found within a single local rig group are combined into one within the Rig Loader, allowing the user to turn all those features on and off at the same time.

Switches are secondary names which also show up in the Rig Loader via a right-click menu. They allow certain parts of a combined feature to be turned on and off.

Although feature tags of different types exist, they all function identically to each-other. The only reason behind using different types of features is to make rig interfaces in the Rig Loader more organised and visually pleasing.

B.3 Command Tags

These are the main tags responsible for procedurally constructing a rig. Behind every command tag is a module which gets executed when the command tag is processed. Parameters of these tags vary, but are mostly straight-forward and understandable by anyone acquainted with rigging.

The commands themselves have two modes – constructor and destructor. In constructor mode the command performs rig construction actions. In destructor mode those actions are undone (for example, created nodes are deleted).

Because one command may require the output of another to function correctly, it is important to keep track of the order of execution. When building the list of commands, joints are examined in a parent-to-child order. The commands attached to joints are added to the list in a top-to-bottom order (as displayed in the Rig Editor).

References

Maya Documentation. *Autodesk 2000 - 2008*

HUNT, D. **Modular Procedural Rigging.** *Game Developers Conference 2009*

RITCHIE, K., CALLERY, J., AND BIRI, K. **The Art of Rigging: Volume 1, 2, 3.** *CgToolkit 2005 - 2007*

BARAN, I., AND POPOVIC, J. **Automatic Rigging and Animation of 3D Characters.** *ACM SIGGRAPH Transactions on Graphics 2007*

CAPELL, S., BURKHART, M., CURLESS, B., DUCHAMP, T., AND POPOVIC, Z. **Physically Based Rigging for Deformable Characters.** *Eurographics/ACM SIGGRAPH Symposium on Computer Animation 2005*