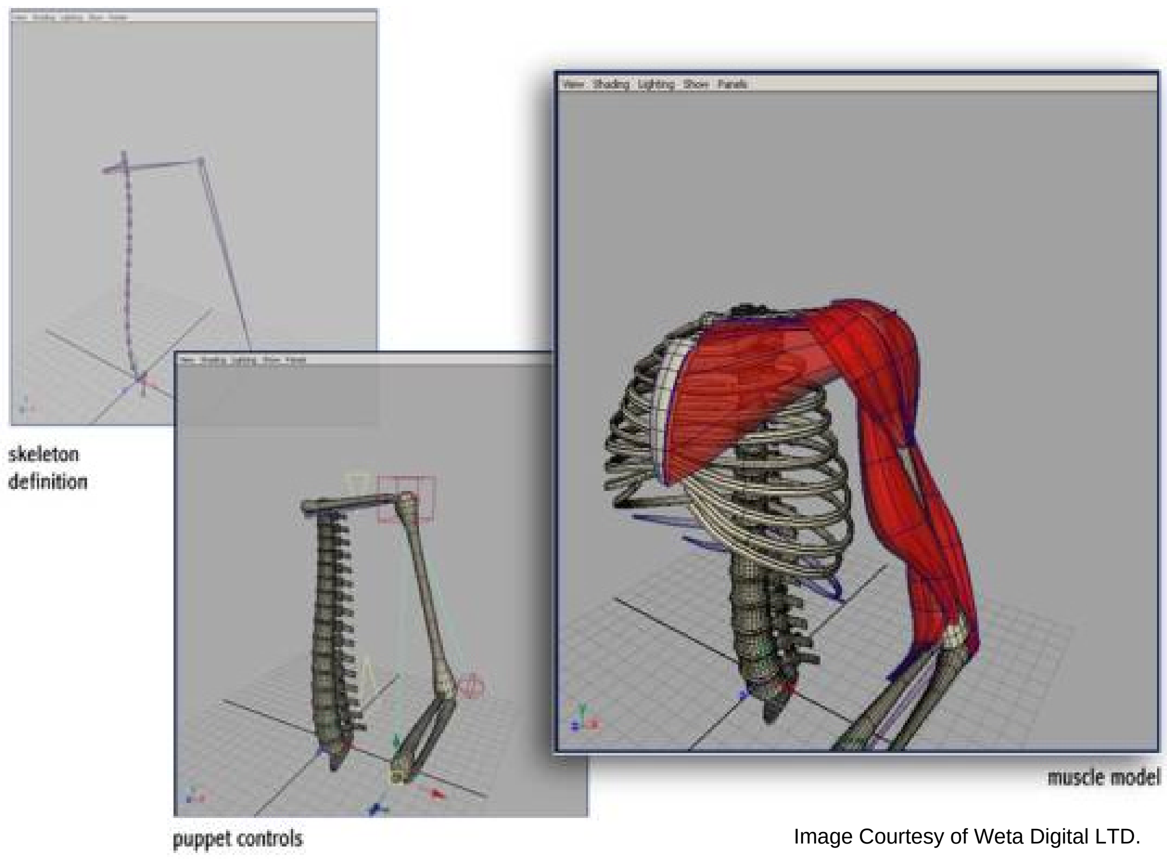


Image Courtesy of Weta Digital LTD.

INTEGRATING AN ANIMATION RIG INTO A CREATURE PIPELINE

AWGUA Maya Master Class August 2001
Integrating an animation rig into a creature pipeline: Presentation Outline

Presenter: Jason Schleifer
Weta Digital, Wellington, NZ

I. Developing the Pipeline

- What is a pipeline?
- How does the animation rig fit into the pipeline?

II. Defining the Creature

Copyright Alias|Wavefront, Jason Schleifer and Weta Digital LTD.

- What is a creature?
- Creating and locating creatures.

III. Orienting Joints

- Manually Re-orienting joints.
- Re-orienting joints based on a script.

IV. Animation Control Concepts

- Iconic Representation.
- Limiting Selection and keyability.
- Rotation Order.
- Extra Gimbal Control
- Customized pickWalking.

V. Automatic/Manual shoulder control

- History
- Creating the control structure
- Cleaning the controls

VI. Realistic Forearm Twist

- Radius and Ulna
- Creating the control structure

VII. FK/IK Back Control

- History
- Creating the spline ik
- Adding stretching
- Creating ik control structure
- Adding fk control structure
- Adding twist
- Adding stretch warning color

VIII. Procedural Animation Rig

- What is a procedural animation rig?
- Creating the rig
- Updating the rig

Copyright Alias|Wavefront, Jason Schleifer and Weta Digital LTD.

IX. Recap and Questions

These data files are included with the coureware data:

MEL Scripts:

endName.mel
jsChannelCtrl.mel
jsConstObj.mel
jsCreateCreature.mel
jsDefineCreature.mel
jsGetShape.mel
jsListCreatures.mel
jsMovIn.mel
jsMovOut.mel
jsOrientJoint.mel
jsOrientJointUI.mel
jsPickWalk.mel
jsRenameWindow.mel
jsRotateOrder.mel
jsScaleJointsByCurve.mel
jsUnlockTransforms.mel

Scenes

backSolver_start.mb*
boneStructure_done.mb*
boneStructure_start.mb*
forearm_end.mb*
forearm_start.mb*
lrKeithAnim.mb*
lrKeithGimbal_done.mb*
lrKeithGimbal_start.mb*
mrKeithSkel.ma*
rings.mb*

Other

web pages and sample movies
backSolver_end.mb
boneStructure_start.mb
boneStructure_done.mb

Copyright Alias|Wavefront, Jason Schlieifer, and Weta Digital LTD.

I. Developing the Pipeline

What is a pipeline?

A pipeline is the basic path that a shot, or a portion of a shot can take from start to finish.

Example 1: Film I/O Pipeline:

- ○ Scan film into computer using film scanner.
- ○ Convert images to all resolutions/formats for production (TV res for animation, 1k and 2k for comp).
- ○ Put images in logical location.
- ○ Notify appropriate parties.

Example 2: Modeling Pipeline:

- ○ Take physical sculpture and trace contour lines around model
- ○ Digitize countours into computer.
- ○ Build rough surface based on mesh.
- ○ Refine, and deliver final subdivision or nurbs surfaces.

Example 3: Shot pipeline.

- ○ Bring scanned plates online
- ○ Matchmove a camera to the plates
- ○ Build relevant 3D data (floor plane, interaction geometry).
- ○ Animate creatures to the scene
- ○ Add FX
- ○ Render separate elements
- ○ Composite all elements together.
- ○ Write final frames to film.

Obviously building a facility which runs well requires smaller pipelines which fit into the broader structure of creating a shot. In this course, were going to focus on the character animation, or creature rig aspect of the pipeline:

- Building a skeletal structure that makes sense.
- Developing controls for that skeletal structure.
- Delivering the motion to the skinning system in a consistant way.

Where Does the Animation Rig fit in the Pipeline?

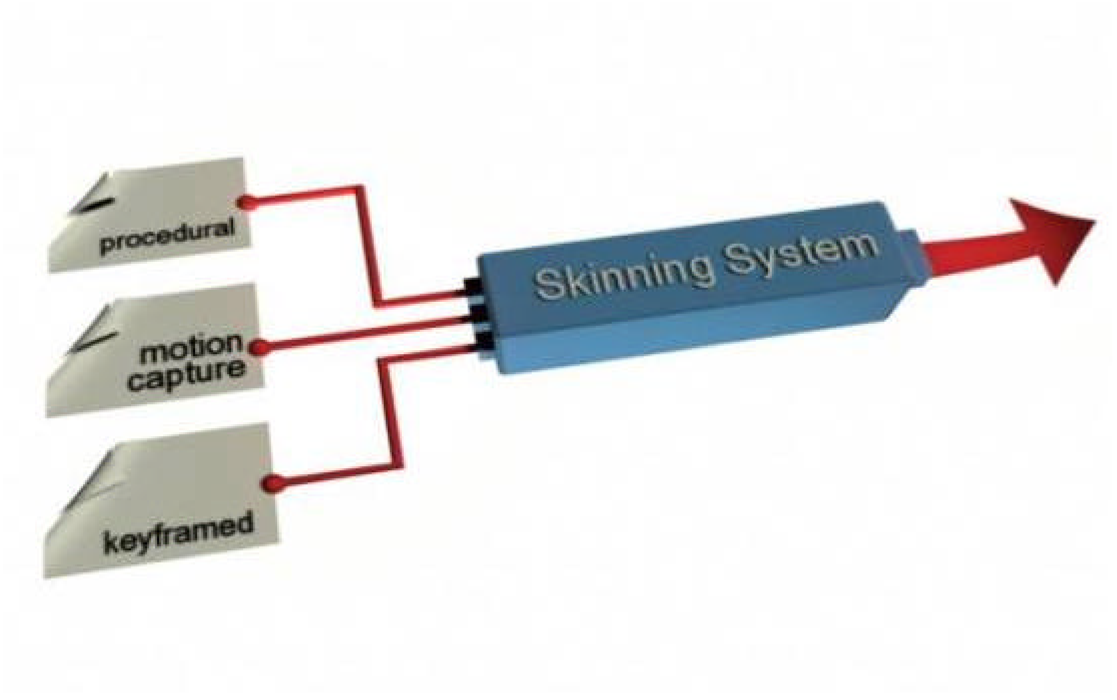
In order to determine how were going to build our puppets to fit into the production pipeline, its important to know what part of the pipe they are, and what their relationship is with other aspects of the pipe. To do this, you need to determine the following things:

Copyright Alias|Wavefront, Jason Schleifer and Weta Digital LTD.

- What type of motion are you going to be dealing with? Hand keyed animation? Motion capture? Procedural? A combination of those?

- What packages are involved in the process? Are they all Maya based, or do you have to integrate with Houdini, Softimage, Mirai, etc.

Basically you want to ask yourself, How am I going to get this motion to that skinned creature?.



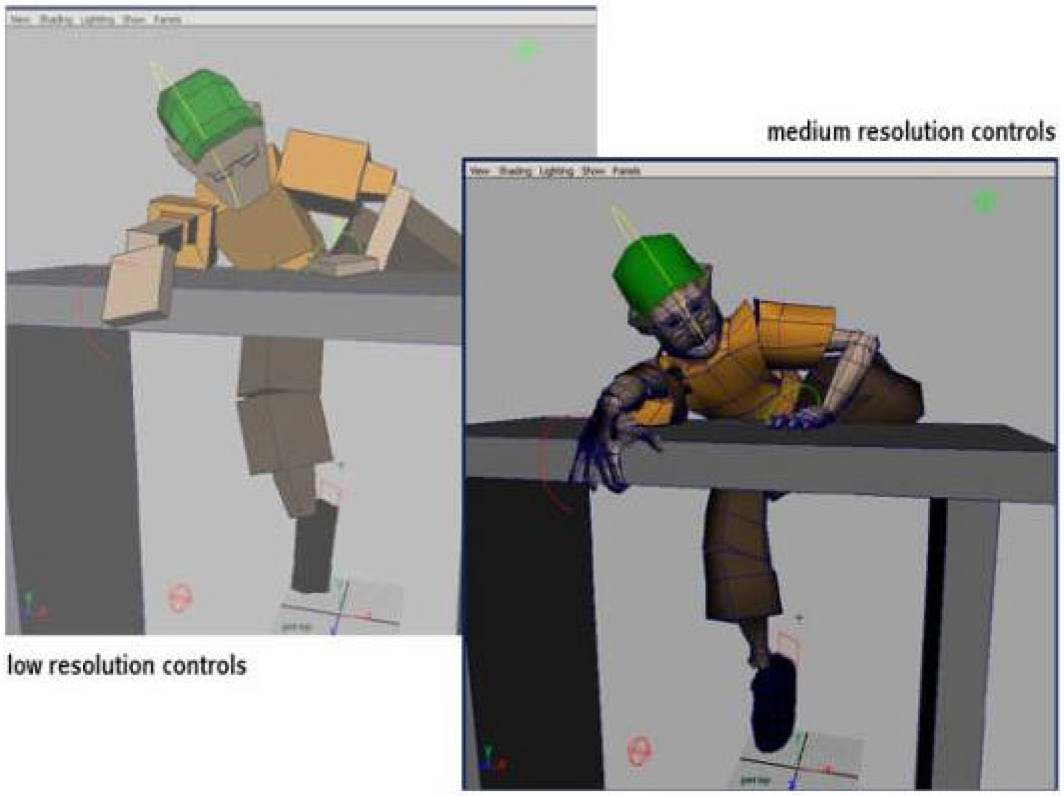
The image above shows a simplified flow of motion through the skinning system at Weta Digital. We had at least three different types of motion being delivered through the system: proceduraly generated, motion capture, and keyframe animated.

In order to keep things consistant and easy to keep track of, we knew that we would need one method of transferring animation from the motion capture, the rigs, and the procedural system to our skinning system.

There are basically two methods for transferring animation from a creature rig to a skinned creature:

- Control based
- Skeletal based

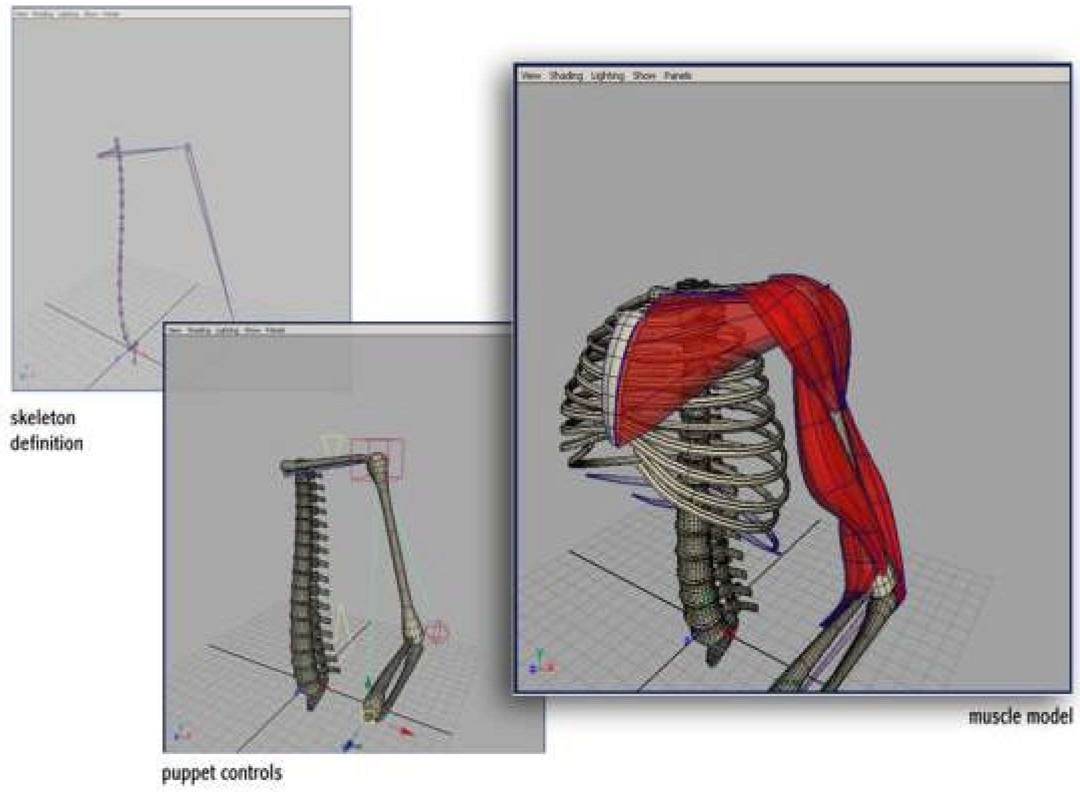
Control Based Animation Transfer



- Animation gets copied from the animator controls, and applied to those same controls on a more complicated skeleton.
- This setup usually involves a few resolutions of a creature:
    - **Low resolution**, easy and fast to animate.. usually a body control, arm and leg controls, and a head control.
    - **Medium resolution**, still somewhat quick to animate, but has more detail.. fingers, toes, eyes, facial, etc. Basically everything you need about 90% of the time
    - **High resolution**. This has everything on the creature you could possibly animate maybe even all the skinning is applied. This is a slow creature to move, but you can control every aspect of it.

Skeletal Based Animation Transfer

Copyright Alias|Wavefront, Jason Schleifer and Weta Digital LTD.



- Animation gets copied from the bones themselves, to a creature rig which is driven by the same bone structure.

- Any control can be applied to the skeleton, as long as the enveloping and puppet models have the same controls the animation will come across.

Which to use?

- Both method work fine, and have been used sucessfully in many productions. Which method you choose depends on what the pipeline is you need to use.

- A benefit to the control based animation transfer is that the animator can tweak the final skin of their creature with the same controls they use to animate with. They can at any point make those fine adjustments, always knowing exactly what theyre going to get.

- With the skeletal based animation system, once the skeleton is defined the animator and td can do whatever they want to the control structure, and the animation will always come across correctly. This is extremely important if you have a creature that needs different setups depending on the situation theyre in. You only have to build ONE muscle model, but you can have 15 different control structures depending on whats needed. Converseley, you can have 47 different muscle models with fixes for certain angles (if necessary), and as long as the skeleton is the same you can take the animation across.

II. Defining the "Creature" (step by step)

What is a creature?

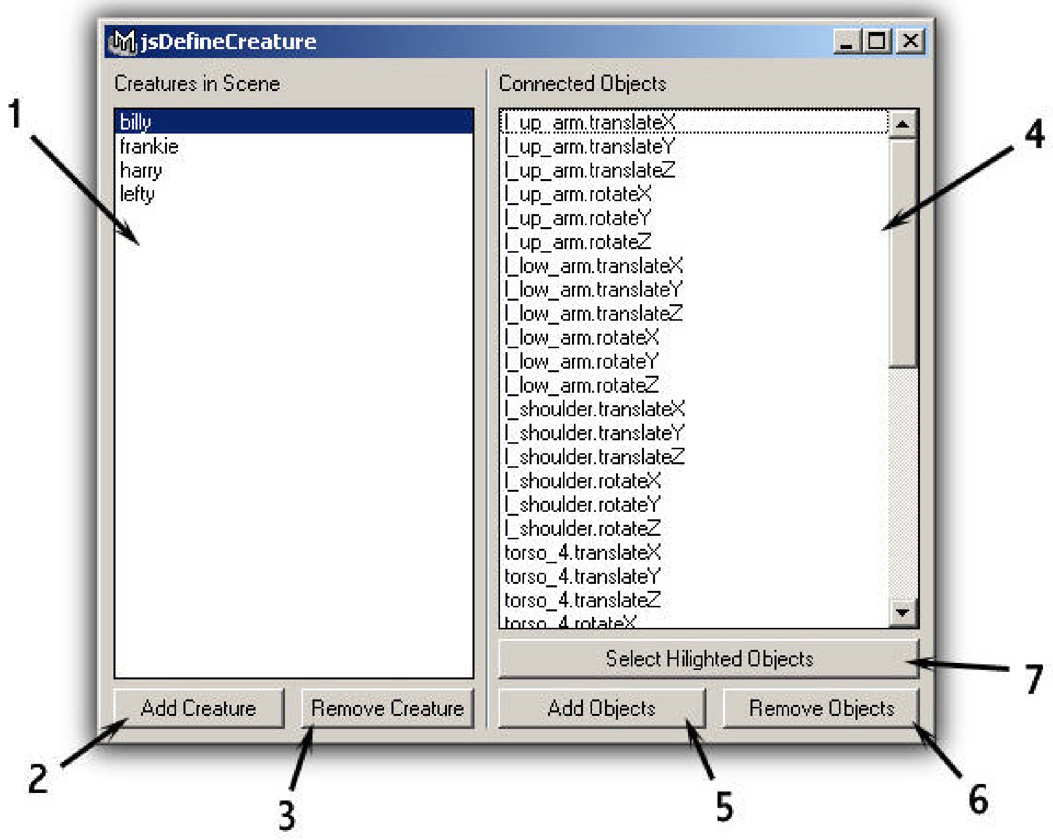
- In order to be able to consistenly work with your animation rig and muscle model, it's important to define what the "creature" is.
- The "creature" can be anything, really, as long as it's easily accessable and you can keep track of it. It's a handy place to keep notes about the setup, keep track of version numbers, even animation tips.
- At Weta we had a specific node which was our "creature" node. That was the parent of all the animation controls in the rig, and of the entire muscle model in the muscle setup. It allowed us to keep things neat and tidy when working with 50 characters in the scene.
- The creature node was also used to point to the ASF which was represented by that character. Because all our animation is being transferred using ASF/AMC, we need to know what ASF is being used (sometimes asf's would change for a creature).

jsDefineCreature.mel script

This script allows you to define which controls are going to be exported using **jsMovOut**, or imported using **jsMovIn**.

- ○ Bring up the interface by typing "**jsDefineCreature**" in the script editor.

Copyright Alias|Wavefront, Jason Schleifer and Weta Digital LTD.



1. All the "creatures" in the scene.

2. Add a new creature to the scene.

3. Remove a creature from the scene.

4. All the connected objects for the selected creature.

5. Add selected objects to the current creature

6. Remove highlighted objects from the current creature

7. Select the highlighted objects.

Example:

- Load lrKeithAnim.mb
- Bring up jsDefineCreature.
- Notice how there's a lrKeith creature in the scene.
- Now we're going to add a "creature" for the platform.
- Click **Add Creature** to create a new creature.
- Call it "*platform*".
- Highlight **platform**.
- In the outliner, select *platform_twist, platform_tilt1, platform_tilt2.*
- Click **Add Objects**.

Notice how the platform attributes are all shown in the UI. It will add all keyable attributes.

- Select the attributes we don't want and remove them by clicking **Remove Objects**.

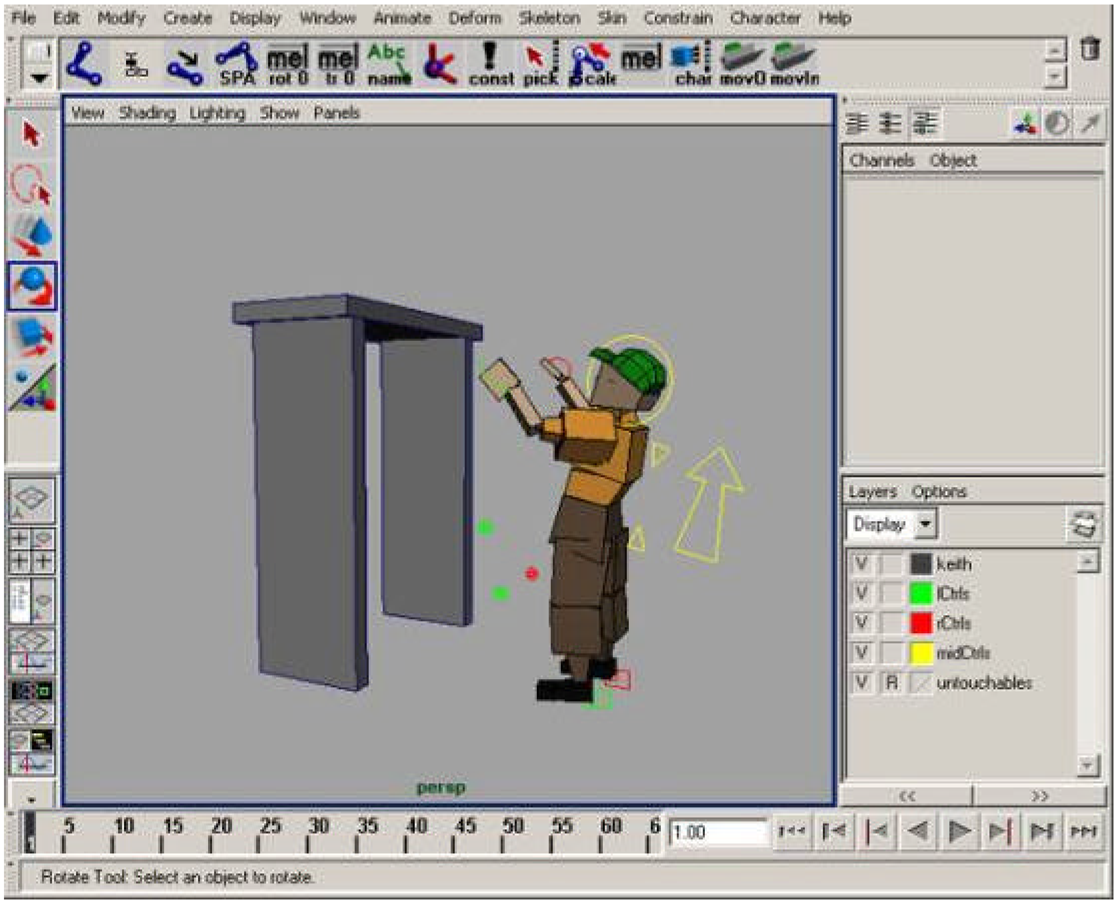
Now we have a platform creature!

jsMovOut.mel script

This script allows you to take the animation from a creature and write it out as mov data.

Use this script whenever you want to export frame-by-frame animation from a creature to a **mov** file.

Example:


File Edit Modify Create Display Window Animate Deform Skeleton Skin Constrain Character Help
SPA
rot 0
tr 0
name
const
pick
scale
char
mov0
movIn
View Shading Lighting Show Panels
persp
Channels Object
Layers Options
Display
keith
lCtrls
rCtrls
midCtrls
untouchables
1.00

Rotate Tool: Select an object to rotate.

- Load **lrKeithAnim.mb**

  This is blocking motion of **keith** attempting to jump onto a pillar of blocks. (view the *lrKeithJumping.avi*)

- Bring up **jsMovOut**.
- Select the lrKeith.

- Click **Export MOV**
- Enter the name of a **mov** file.

  Maya will now export the information on those controls frame by frame.

jsMovIn.mel script

This script allows you to import the animation from a mov file onto a creature.

Use it when you've exported animation from a creature and you want to import it onto another creautre.

Example:

- Open **mrKeithSkel.mb**

  This is the medium resolution of the Keith puppet.

- Run **jsMovIn**.
- Pick **mrKeith**.
- Click **Import Mov**.
- Choose the **mov** file you generated from above.

  This will now import the motion onto this medium resolution version of keith!

III. Defining a "type" of node (step by step)

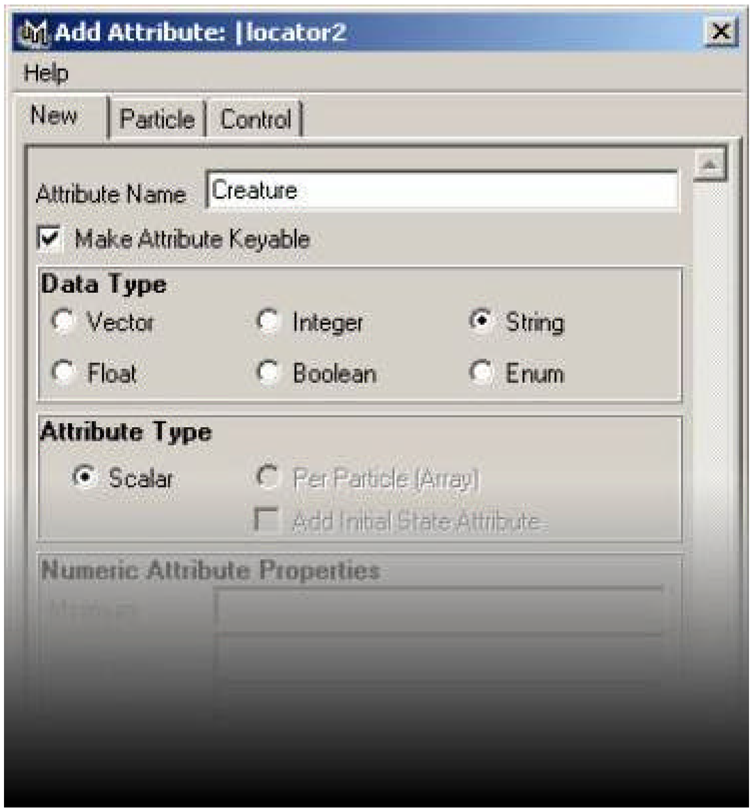
Creating a "type" of node

The example of using **jsDefineCreature**, **jsMovIn**, and **jsMovOut** works because we've defined what the creature is, and we can search our scene for all the nodes of that type.

This is done by simply adding an attribute which we can search for.

- Create a locator, add an attribute called "**Creature**" of type string.
- Select **Create** > **Locator**
- Select **Modify** > **Add Attribute**;
- Name the attribute **Creature**, and make it type **String**;

Copyright Alias|Wavefront, Jason Schleifer and Weta Digital LTD.



Add Attribute: |locator2

Help

New | Particle | Control

Attribute Name: Creature

[✓] Make Attribute Keyable

Data Type

Vector | Integer | [✓ String]
Float | Boolean | Enum

Attribute Type

[✓ Scalar] | Per Particle (Array)
Add Initial State Attribute

Numeric Attribute Properties

Finding a creature

Now that we know that our "creatures" are actually locators with an attribute on them called "Creature", it becomes really easy to find. Using mel, we can type the following command:

```
ls -type spaceLocator;
```

This will return all the locators in the scene.

You can use the **attributeQuery** command to find out of an attribute exists on a certain node:

```
attributeQuery -exists -node <node> <attribute>;
```

Throw this whole thing into a loop, and you can easily find all the locators in the scene with the attribute of name **Creature**:

```
string $locators[0];
string $creatures[0];
int $c = 0;

// get the list of locators
$locators = `ls -type spaceLocator`;

// cycle through each locator
for ($loc in $locators)
```

```
        {

                // check and see if the attribute exists
                if (`attributeQuery -exists -node $loc "Creature"`)
                {

                        // if the attribute exists, add $loc to the list of creatures
                        $creatures[$c] = $loc;

                        // increment $c
                        $c++;

                }

        }

        print $creatures;
```

If you want to get really tricky, you can throw the whole thing into a procedure called "findCreature", and add an option for a "type" of creature.

```
        global proc string [ ] findCreature (string $type)
        {

                string $locators[0];
                string $creatures[0];
                int $c = 0;

                // get the list of locators
                $locators = `ls -type spaceLocator`;

                // cycle through each locator
                for ($loc in $locators)
                {

                        // check and see if the attribute exists
                        if (`attributeQuery -exists -node $loc "Creature"`)
                        {

                        // if the attribute exists, check and see if
                        // the creature is of the right type
                        $attrType = `getAttr ($loc + ".Creature")`;
                        if ($type == "")
                        {

                                $creatures[$c] = $loc;

                                // increment $c
                                $c++;

                        }
                        else if ($type == $attrType)
                        {

                                $creatures[$c] = $loc;

                                // increment $c
                                $c++;

                        }

                }

        }

        return $creatures;

}
```

Finding nodes which relate to the creature

Quite frequently, you'll need to execute scripts based on certain nodes of a creature. For example, at Weta, we had scripts to automatically switch between fk and ik using a single (or a few) button clicks. To do that, we had to know which controls were being used for fk, which for ik, and what attribute was dealing with the constraints. In the case of **jsMovIn** and **jsMovOut**, we need to know which nodes and attributes we're going to be exporting.

This can be done using names, but 99.9% of the time, the name you determine for a node is going to be changed, either through importing, or another object being named something similar.

Using connections to derive names from your scripts is really the only way to insure that you're not going have your scripts break each time they're executed.

To do this, add an attribute which is easily found by you, and is of type message. It can be multi, or not. It depends on what you're using it for.

You are going to Add the attribute, make the connection, and use listConnections to determine what the names are.

- **Adding the attribute**:

```
// add a message attribute to a node
// addAttr -ln <attribute name> -at "message" <nodeName>;
addAttr -ln leftHand -at "message" troll;

// add a multi message attribute to a node
// addAttr -ln <attributeName> -at "message" -multi -im false <nodeName>;
addAttr -ln exportBones -at "message" -multi -im false troll;
```

Copyright Alias|Wavefront, Jason Schleifer and Weta Digital LTD.

- **Connecting to the attribute**

```
// connect to a single message attribute
// connectAttr -force <node.message> <nodeName.attribute>
connectAttr -force leftHandCtrl.message troll.leftHand;

// connect to a multi message attribute
// connectAttr -force <node.message> -nextAvailable <nodeName.attribute>
connectAttr -force -nextAvailable torso_1.message troll.exportBones;
connectAttr -force -nextAvailable torso_2.message troll.exportBones;
connectAttr -force -nextAvailable torso_3.message troll.exportBones;
```

- **Listing the connections**

```
// list the connection without the attribute (just return the node name)
// listConnections -plug false -destination false -source true
//     <nodeName.attribute>;
listConnections -plug false -destination false -source true troll.leftHand;
// Result: leftHandCtrl //

// list the connection WITH the attribute
// listConnections -plug true -destination false -source true
//     <nodeName.attribute>;
listConnections -plug true -destination false -source true
troll.exportBones;
// Result: torso_1.message torso_2.message torso_3.message //
```

Note: When you use listConnections, the result always comes back as a string array.. even if there's only one element being returned.

Copyright Alias|Wavefront, Jason Schleifer and Weta Digital LTD.

III. Orienting Joints

Joint orientation

Joint orientation doesn't seem to get a lot of attention, but it's extremely important.

Maya's joint structure is defined different from other programs, it's not segment based, it's joint based, meaning that a skeleton segment is actually made up of two locations, as apposed to an orientation and a length.

The auto-align tool is handy when first creating joints for making sure they're somewhat aiming at their children, but they don't actually fit the bill when really looking to control a joint structure.

For example, if you're building a characters back and you use the auto-orientation tool, maya re-orients the joints each time you click the mouse, and doesn't actually draw them all in the same orientation. Also, if you move the joints around the orientation doesn't change.

You can always use the **joint -e -oj xyz** command to re-orient the joints, but it doesn't give you quite the control which is necessary when building complicated characters.

Building a back with standard Maya Joint Orientation

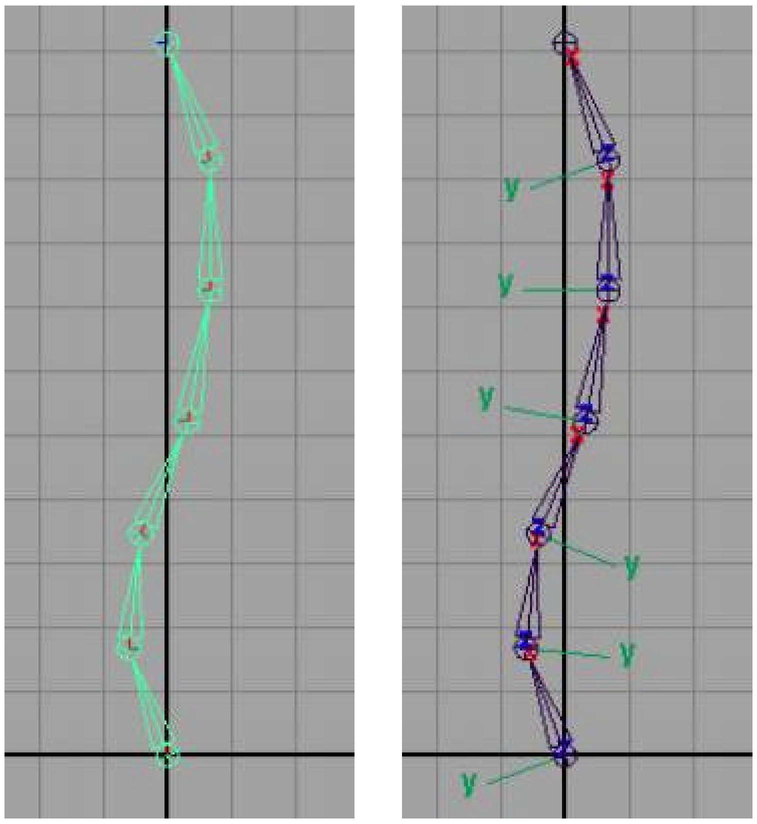
- Select the joint tool.

    In the settings, make sure **Auto Joint Orient** is **on**.

- Create a series of joints which represent the s-curve of a back (6 to 8 joints).
- Display the local rotation axes for each of the joints.

    You can see that the joints are indeed aiming at their children, there's no consistancy as to how the joints are oriented.
    (**Display > Component Display > Local Rotation Axis**).

Copyright Alias|Wavefront, Jason Schleifer and Weta Digital LTD.



This can be a problem in two areas:

1. If you modify the joint positions, the joints no longer are aiming at their children. (fixable with the **joint -e -orientJoint** command).

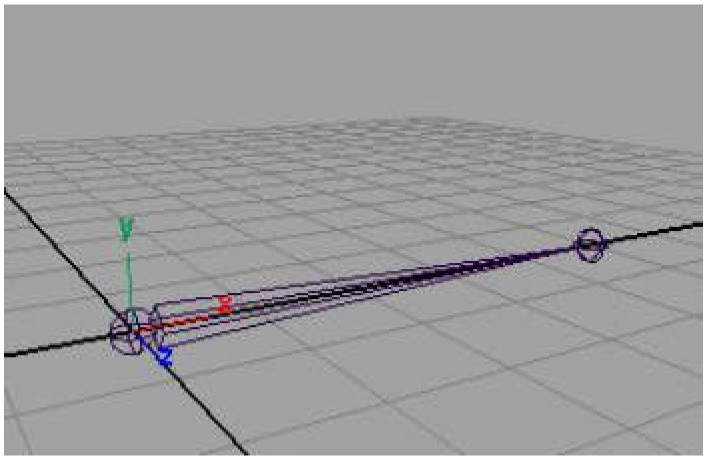
2. If you rotate all the joints in Z, some joints rotate forward, others rotate backwards. This can be confusing for animators.

The solution is to come up with a way of re-orienting the joints with control as to what the joint is aiming at, and what it's up vector is.

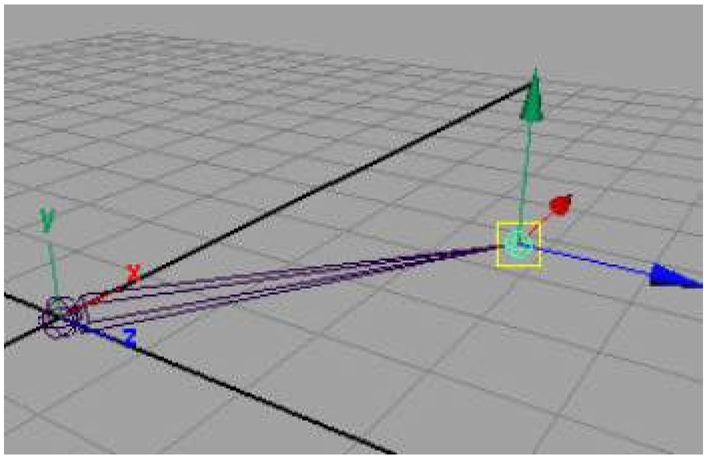
Manually Re-orienting a Joint in Maya.

- Select the joint tool.
- Create two joints, one at **0, 0, 0**, and the other at **5, 0, 0**;
- Display the local rotation axis for the joint.

Copyright Alias|Wavefront, Jason Schleifer and Weta Digital LTD.



- Move the end joint somewhere off to the side.

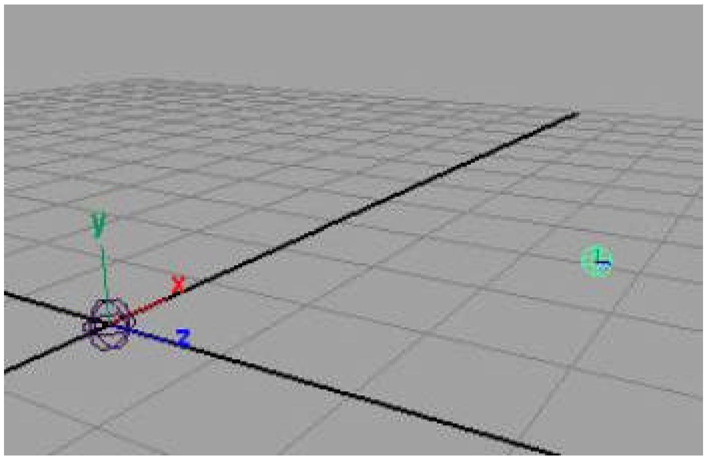


Notice when the first joint is selected, it no longer aims at it's child, and in fact if you rotate it in X, whichis what used to cause the joint to spin around it's axis, it no longer does so. This joint has become extremely difficult to control.

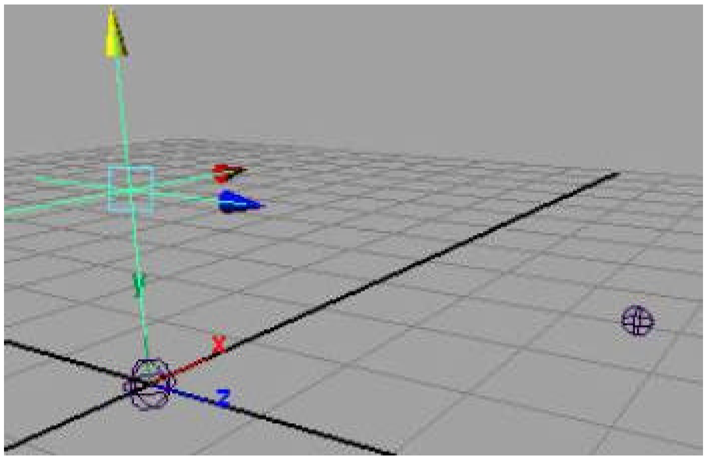
To solve this, we're going to manually force joint1 to aim at it's child. Note, in some cases using the joint -e -orientJoint command may work fine. In many cases, however, the technical director will want more control as to the up vector of the joint. The following technique will provide exact control.

- Unparent **joint2** from **joint1.**

Copyright Alias|Wavefront, Jason Schleifer and Weta Digital LTD.



- Open the attribute editor for joint1 and make sure the **jointOrient** is set to **0 0 0**, and that rotate is **0 0 0**;
- Create a locator to be used as the upVector. Place is somewhere above joint1.



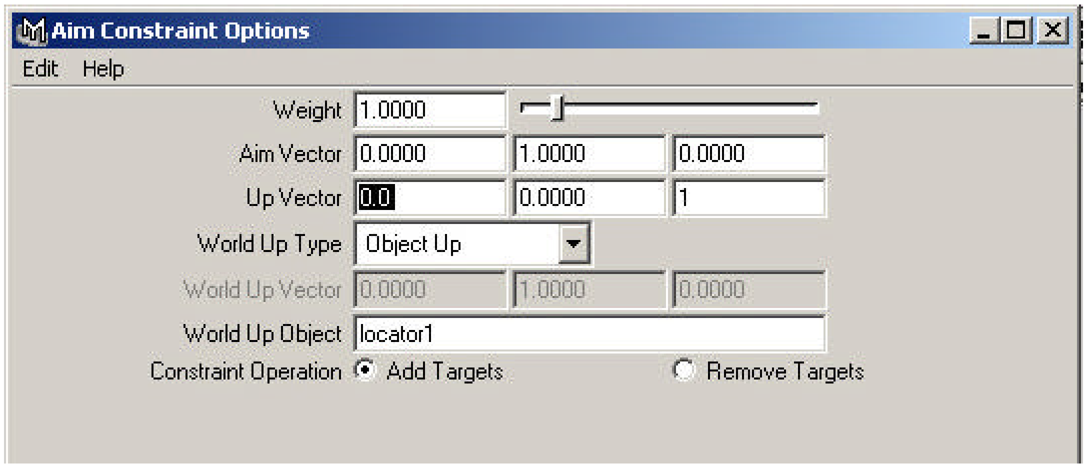
Now you are going to make an aimConstraint.

- Select **joint2** then **joint1.**
- Select **Constrain > Aim > Option**.

Set the following values:

Aim Vector: 0 1 0
Up Vector: 0 0 1
World Up Type: Object Up
World Up Object: locator1

- Click **Add/Remove**.


Aim Constraint Options

Edit Help

Weight 1.0000

Aim Vector 0.0000 1.0000 0.0000

Up Vector 0.0 0.0000 1

World Up Type Object Up

World Up Vector 0.0000 1.0000 0.0000

World Up Object locator1

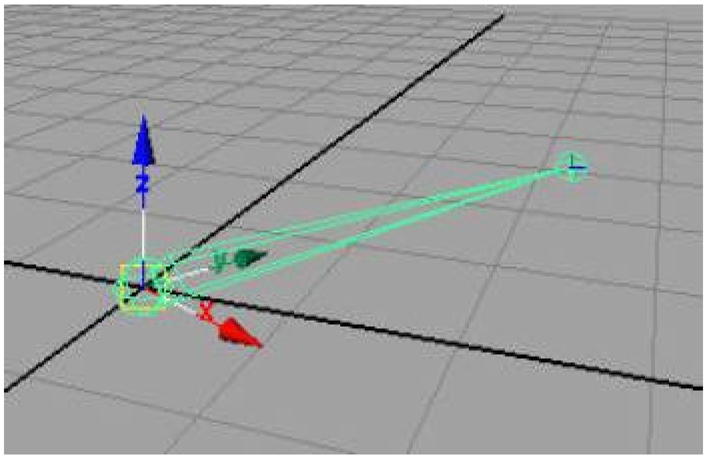
Constraint Operation [✓ Add Targets] Remove Targets

Notice the joint is aiming at the child. If you want to change the upVector for the joint, you can move it wherever you like, until you're happy with the orientation.

- Delete the aimConstraint and the locator.
- Copy the rotation values from joint1 to the jointOrient values.

```
setAttr joint1.jox `getAttr joint1.rx`;
setAttr joint1.joy `getAttr joint1.ry`;
setAttr joint1.joz `getAttr joint1.rz`;
```

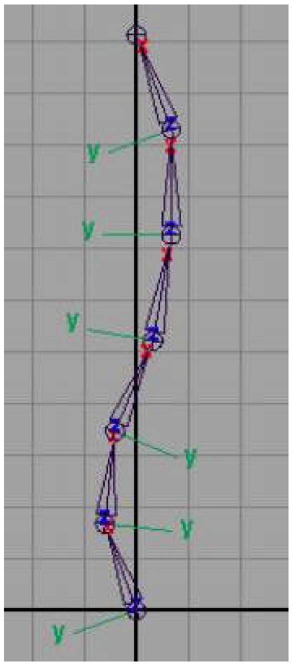
- Now set all the rotations to 0.
- Parent joint2 to joint1 again. The joint has now been re-oriented to aim at the child joint.



Automatic re-orienting of joint(s)
(jsOrientJoint.mel)

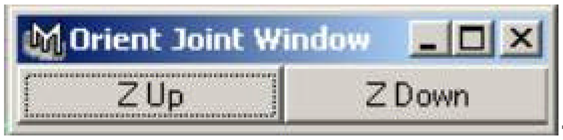
Obviously the previous process is too time consuming to perform all the time. Thus, we can use jsOrientJoint for 95% of the cases where it's necessary to re-orient the joints.

- Recreate the 6 to 10 back joints as shown above.
- Display the local rotation axis for each of the joints.



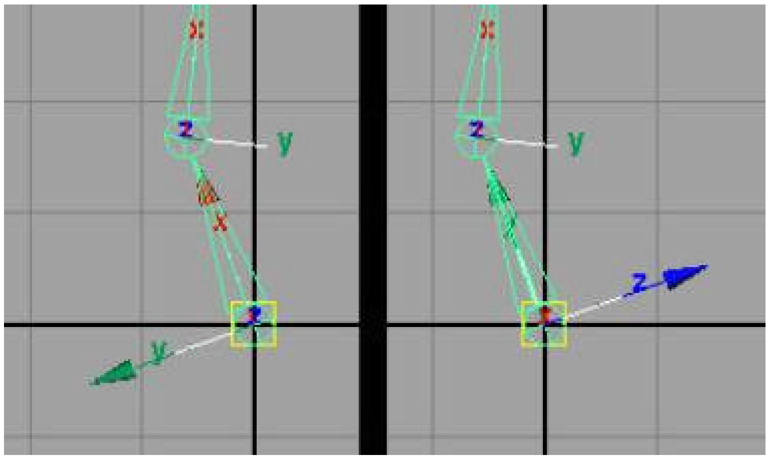
- Bring up the **jsOrientJoint** window by typing jsOrientJoint in the scriptEditor.

  This interface has two buttons, one will align the selected joints with their Z axis pointing up, the other with the Z axis pointing down.



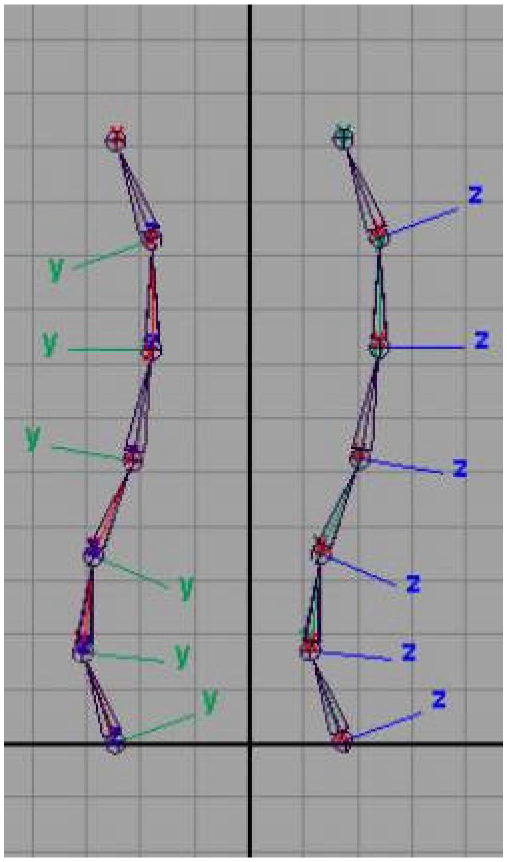
.

- Select the first joint (joint1).
- Click **Z Up**. The joint will re-orient so the Z is aiming up



- Select the next joint and this time click **Z Down** to keep the Z axis going the same direction.

- Continue this, choosing **Z Up** or **Z Down** until all the joints are aligned the same direction. In the image below, you can see the difference between the orig joints and the newly oriented joints.



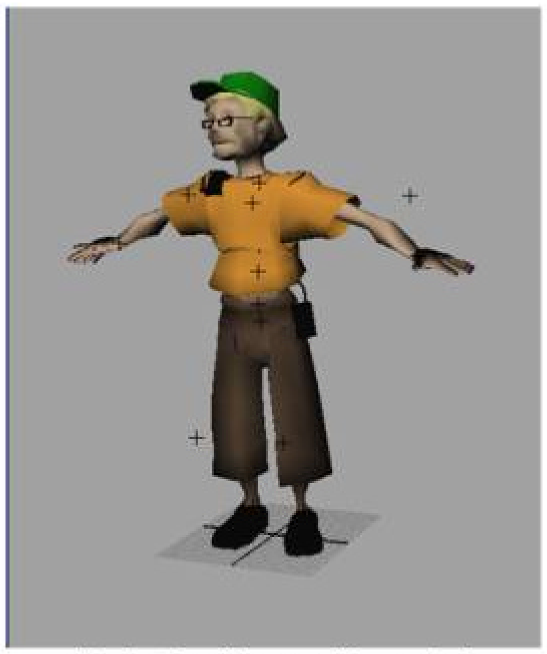
Copyright Alias|Wavefront, Jason Schleifer and Weta Digital LTD.

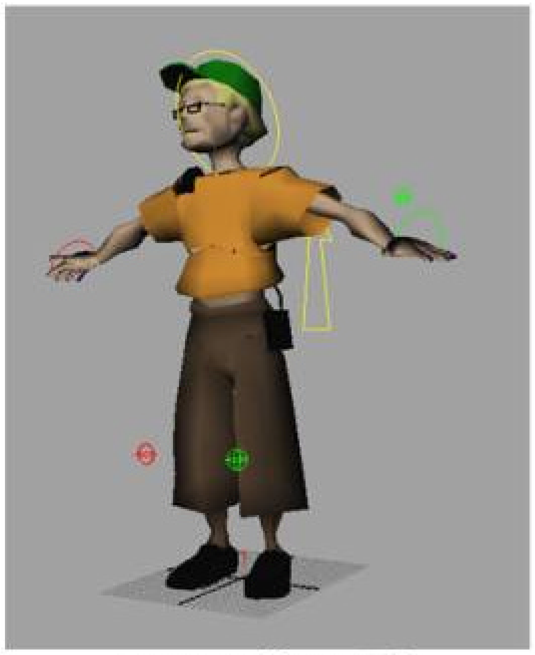
IV. Animation Control Concepts

Quite frequently you will be setting up animation rigs for other people to animate, not for yourself. Because of this, it is imperative that you control what the animator can and cannot touch, and give them easily recognizable controls. It's hard enough to animate a convincing creature, let alone spend extra brain power trying to figure out what to pick, where all those extra keys are coming from, and why in the world your creature's left arm is flying off into never-never land.

Iconic Representation

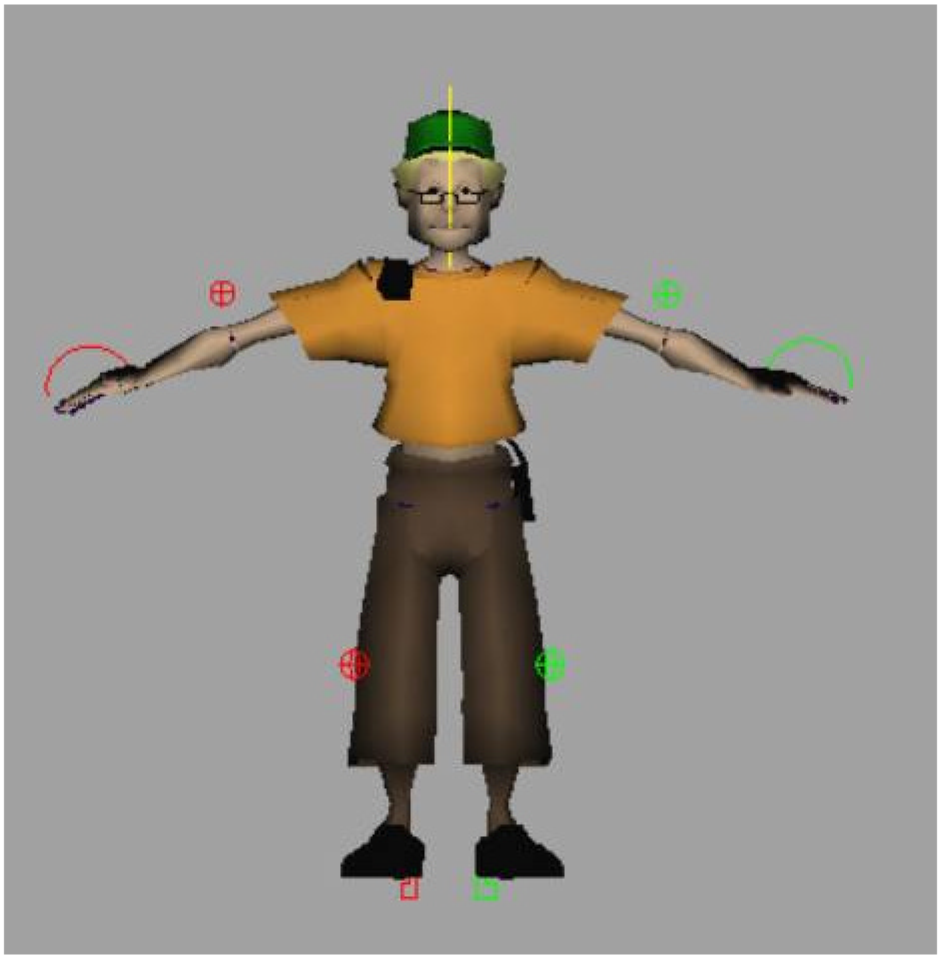
The first step in making things easy to select is giving making the controls easy to understand. If you have a creature that has all the controls visible just as IK handles or selection handles, the screen can become extremely confusing.


display handles used for controls


curves used for controls

In the images above, you can see that in the setup on the left is a little bit more difficult to understand what's going on. If an animator were asked to pick the full body control, which one would they choose' In the image on the right, we've clearly defined which controls are which. If we keep consistent in making these controls the animator always knows what control is what.

Copyright Alias|Wavefront, Jason Schleifer and Weta Digital LTD.



In addition, you'll also notice that the two sides of the controls are different colors. This again helps the animator differentiate between left and right sides of the character.

To change the color of a control, use display layers.

Limiting Selection

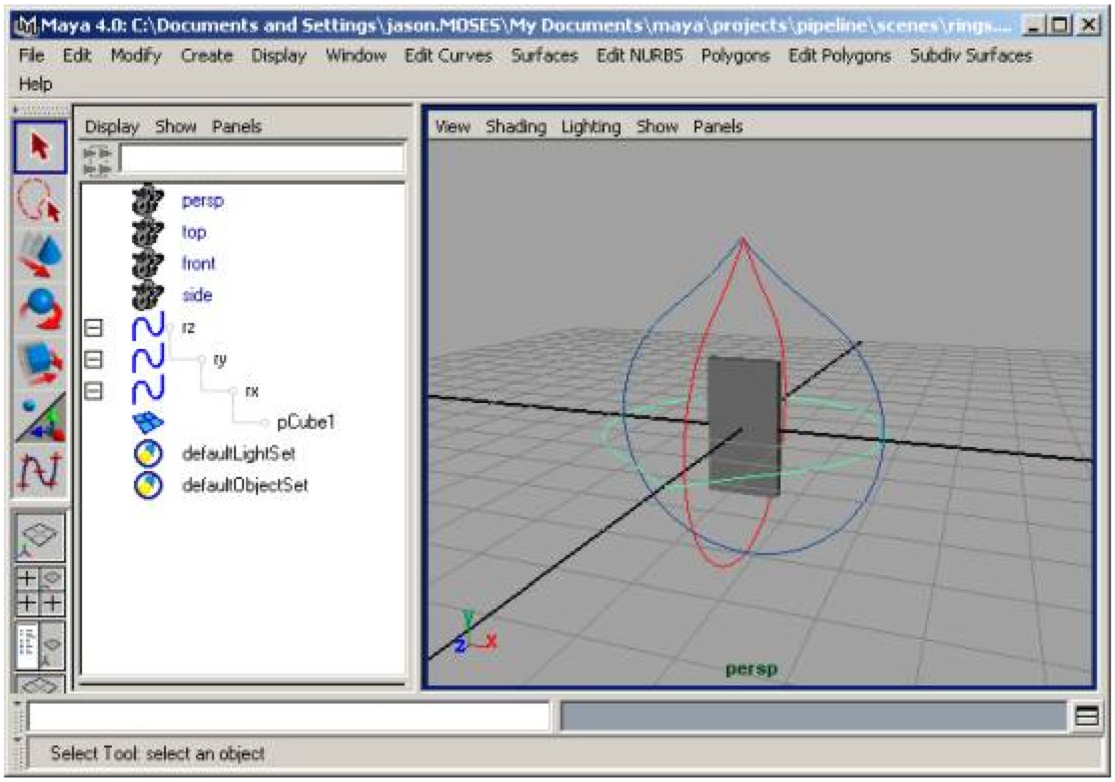
The next step in making sure an animator doesn't grab the wrong controls is limiting what they can select. Take all the objects you don't want the animators touching, and put them in layers which are set to reference mode. This means they're visible, and they show up in shaded mode (unlike templated objects), but you can't select them.

Limiting Keyability

For all the nodes they're animating, it's important to set any attributes you don't want them keying to be non-keyable. (**Window > General Editors > Channel Control**). It does the animators no good to be dealing with five times the number of anim curves they need to.

Rotation Order

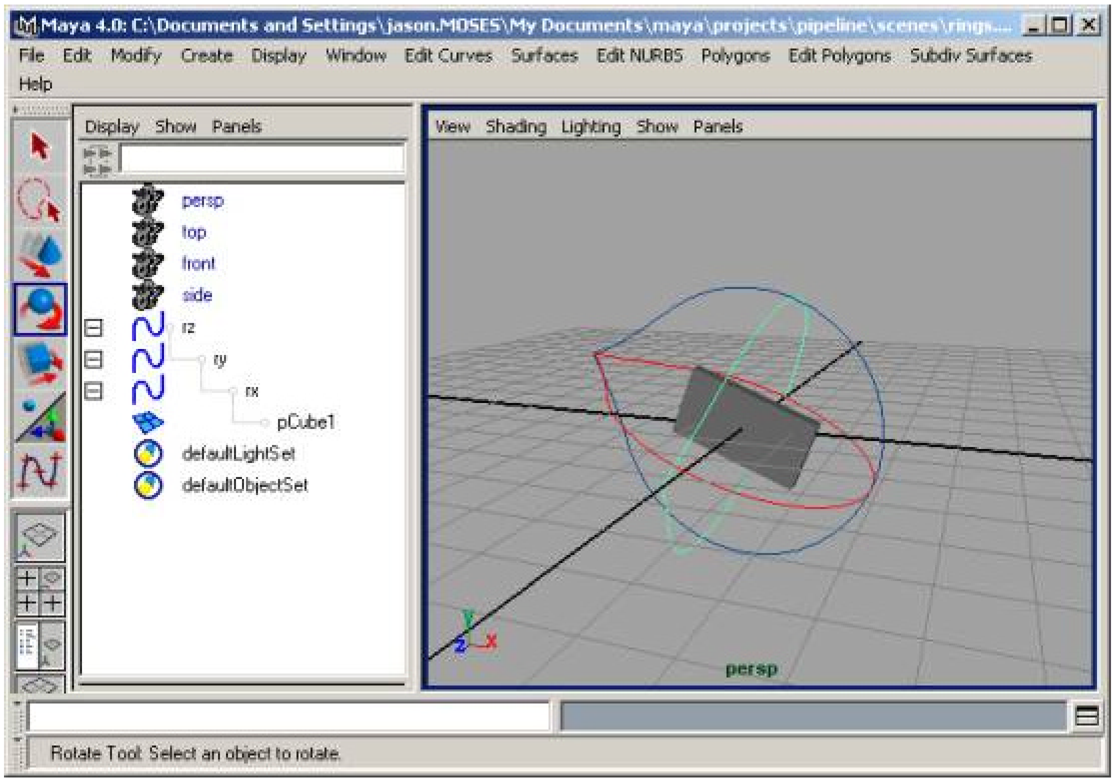
The next step in making sure the creature can move around is to check the rotation orders on the objects they're animating. This is the order that the rotations get evaluated on a node. An easy way to understand how this works is to actually use separate nodes to control the rotation of an object.


Maya 4.0: C:\Documents and Settings\jason.MOSES\My Documents\maya\projects\pipeline\scenes\rings....
File Edit Modify Create Display Window Edit Curves Surfaces Edit NURBS Polygons Edit Polygons Subdiv Surfaces Help
Display Show Panels
persp
top
front
side
rz
ry
rx
pCube1
defaultLightSet
defaultObjectSet
View Shading Lighting Show Panels
persp
Select Tool: select an object

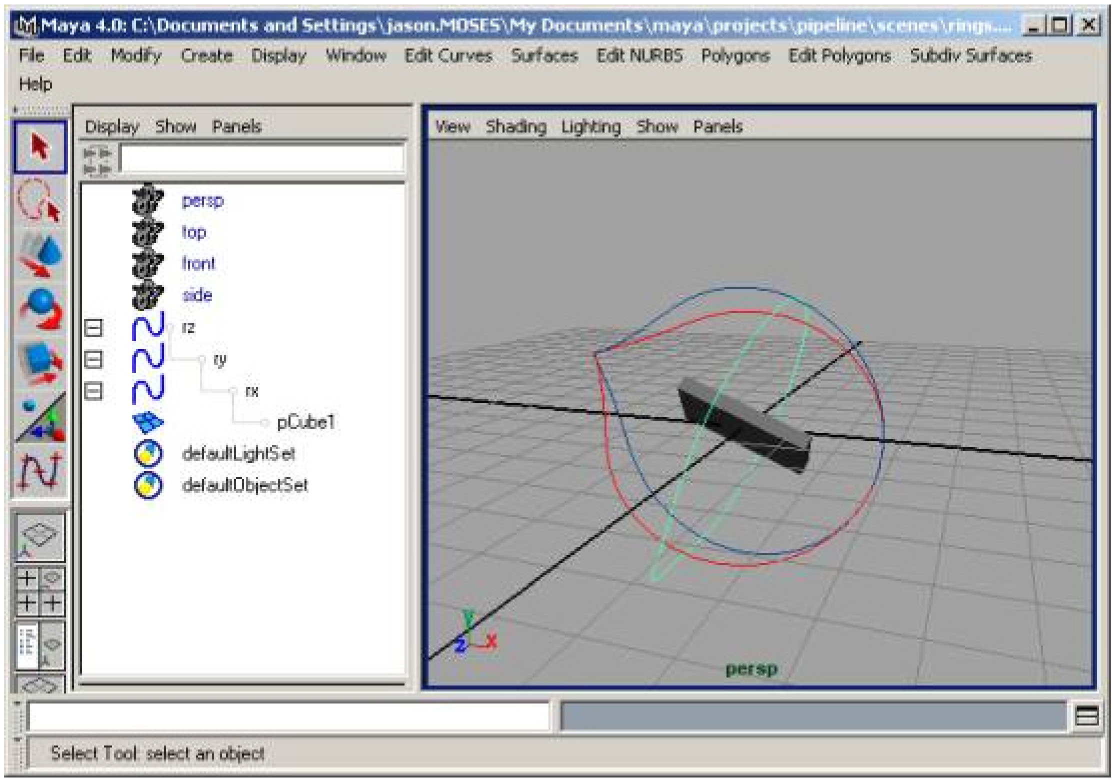
In the image above you can see that we have three different objects available for rotation, rx, ry, and rz. They are currently in the default evaluation order that maya works in (xyz).

- If we rotate the **rz** you will see the other rings rotate with it.

Copyright Alias|Wavefront, Jason Schleifer and Weta Digital LTD.

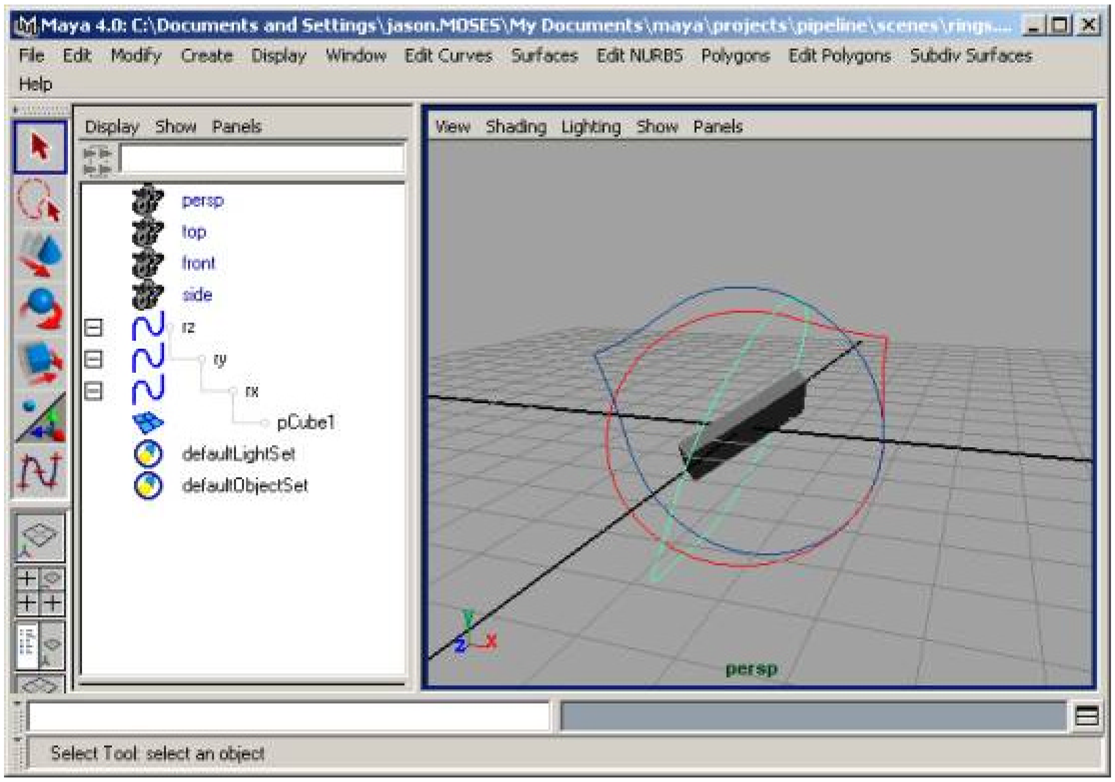

Maya 4.0: C:\Documents and Settings\jason.MOSES\My Documents\maya\projects\pipeline\scenes\rings....
File Edit Modify Create Display Window Edit Curves Surfaces Edit NURBS Polygons Edit Polygons Subdiv Surfaces Help
Display Show Panels
persp
top
front
side
rz
ry
rx
pCube1
defaultLightSet
defaultObjectSet
View Shading Lighting Show Panels
persp
Rotate Tool: Select an object to rotate.

- Now, if we rotate **ry** you will see that **rz** stays where it is, and **rx** moves.


Maya 4.0: C:\Documents and Settings\jason.MOSES\My Documents\maya\projects\pipeline\scenes\rings....
File Edit Modify Create Display Window Edit Curves Surfaces Edit NURBS Polygons Edit Polygons Subdiv Surfaces Help
Display Show Panels
persp
top
front
side
rz
ry
rx
pCube1
defaultLightSet
defaultObjectSet
View Shading Lighting Show Panels
persp
Select Tool: select an object

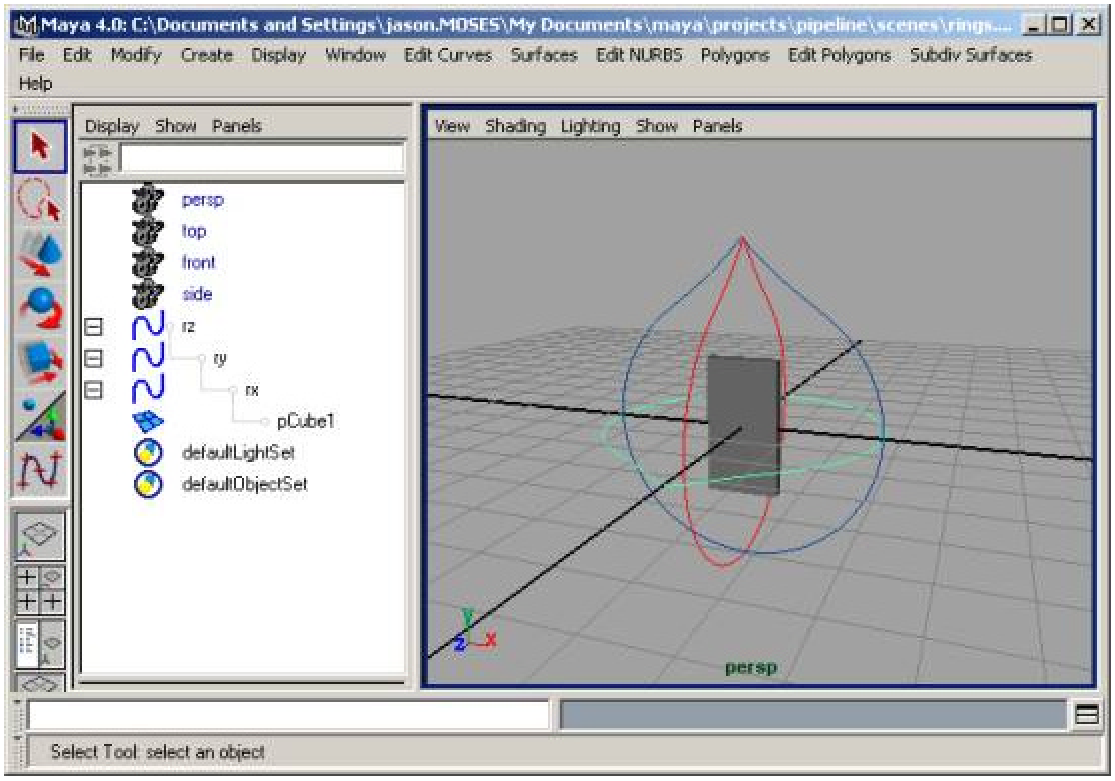
- If we rotate **rx**, both **ry** and **rz** will stay where they are.

Copyright Alias|Wavefront, Jason Schleifer and Weta Digital LTD.

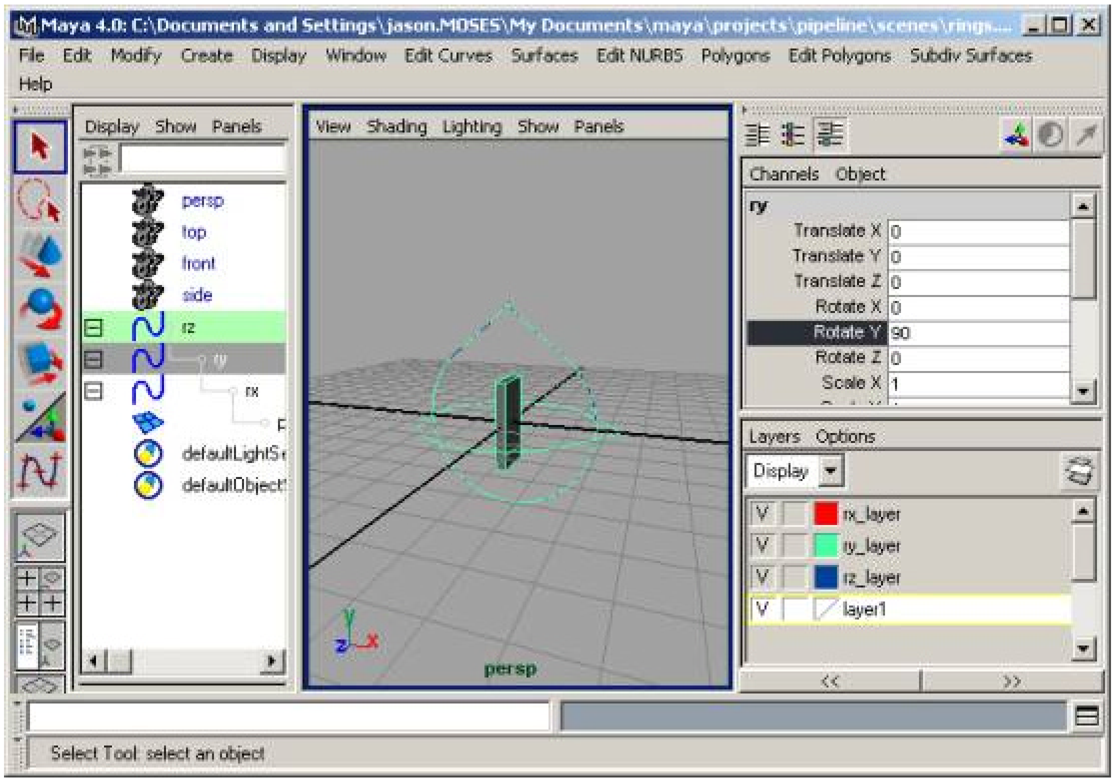

Maya 4.0: C:\Documents and Settings\jason.MOSES\My Documents\maya\projects\pipeline\scenes\rings....
File Edit Modify Create Display Window Edit Curves Surfaces Edit NURBS Polygons Edit Polygons Subdiv Surfaces Help
Display Show Panels
persp
top
front
side
rz
ry
rx
pCube1
defaultLightSet
defaultObjectSet
View Shading Lighting Show Panels
persp
Select Tool: select an object

You can see the same result if you use the gimbal mode when selecting an object and rotating it.

If we take a look at a common example of where the default rotation order can screw up a character, let's set the rotation values back to **0 0 0**.


Maya 4.0: C:\Documents and Settings\jason.MOSES\My Documents\maya\projects\pipeline\scenes\rings....
File Edit Modify Create Display Window Edit Curves Surfaces Edit NURBS Polygons Edit Polygons Subdiv Surfaces Help
Display Show Panels
persp
top
front
side
rz
ry
rx
pCube1
defaultLightSet
defaultObjectSet
View Shading Lighting Show Panels
persp
Select Tool: select an object

- Now we want to rotate the character around in Y so he's facing the a different direction.


File Edit Modify Create Display Window Edit Curves Surfaces Edit NURBS Polygons Edit Polygons Subdiv Surfaces Help
Display Show Panels
persp
top
front
side
rz
ry
rx
defaultLightS
defaultObject
View Shading Lighting Show Panels
persp
Channels Object
Layers Options
Display
V rx_layer
V ry_layer
V rz_layer
V layer1

Maya 4.0: C:\Documents and Settings\jason.MOSES\My Documents\maya\projects\pipeline\scenes\rings....

ry

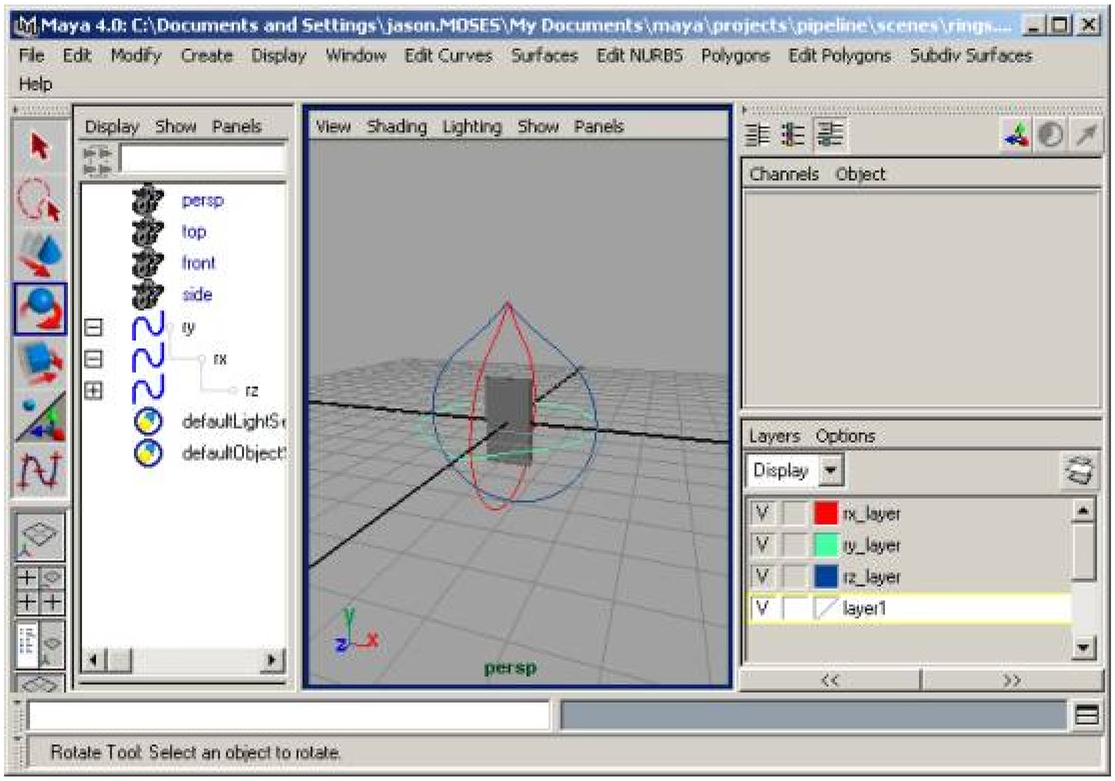
Translate X	0
Translate Y	0
Translate Z	0
Rotate X	0
Rotate Y	90
Rotate Z	0
Scale X	1

Select Tool: select an object

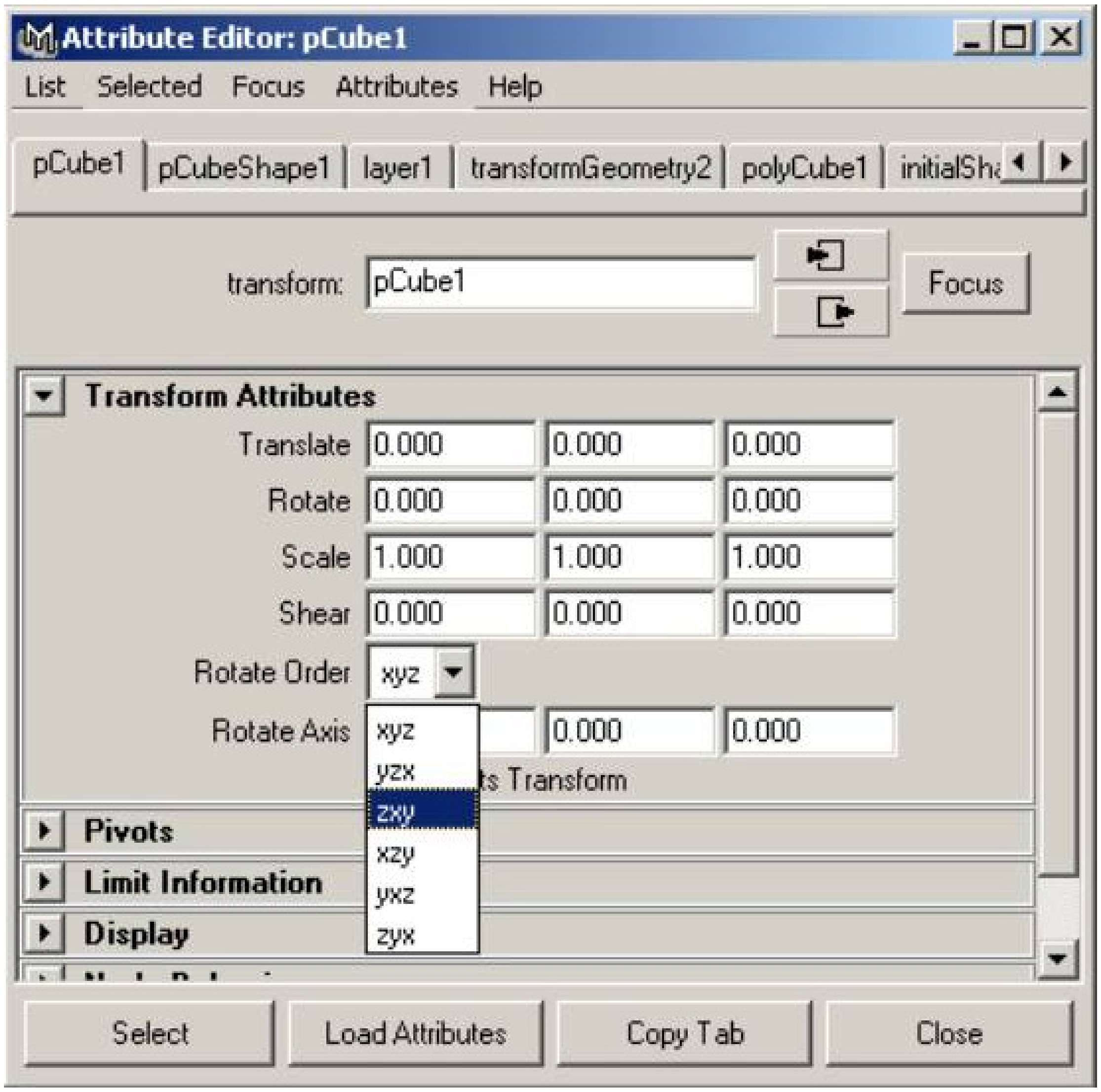
- We just rotated **ry** 90 degrees. You can tell pretty quickly that we're in trouble with rotation orders. Right now the character can keep twisting in Y, and it can bend forward, but it can't lean to the side.

  If we decide to change the rotation orders instead so ry is evaluated first, then we eliminate the problem of not allowing our creatures to turn around.

Copyright Alias|Wavefront, Jason Schleifer and Weta Digital LTD.


Maya 4.0: C:\Documents and Settings\jason.MOSES\My Documents\maya\projects\pipeline\scenes\rings....
File Edit Modify Create Display Window Edit Curves Surfaces Edit NURBS Polygons Edit Polygons Subdiv Surfaces Help
Display Show Panels
persp
top
front
side
ry
rx
rz
defaultLightS
defaultObject
View Shading Lighting Show Panels
persp
Channels Object
Layers Options
Display
rx_layer
ry_layer
rz_layer
layer1
Rotate Tool: Select an object to rotate.

- You can do the same thing with changing the rotation order for an object in Maya. In this case, you would change the rotation order in the Attribute editor for the cube to zxy.


Attribute Editor: pCube1

List Selected Focus Attributes Help

pCube1 | pCubeShape1 | layer1 | transformGeometry2 | polyCube1 | initialSh

transform: pCube1

Focus

Transform Attributes

Translate	0.000	0.000	0.000
Rotate	0.000	0.000	0.000
Scale	1.000	1.000	1.000
Shear	0.000	0.000	0.000

Rotate Order: xyz

Rotate Axis: 0.000 0.000

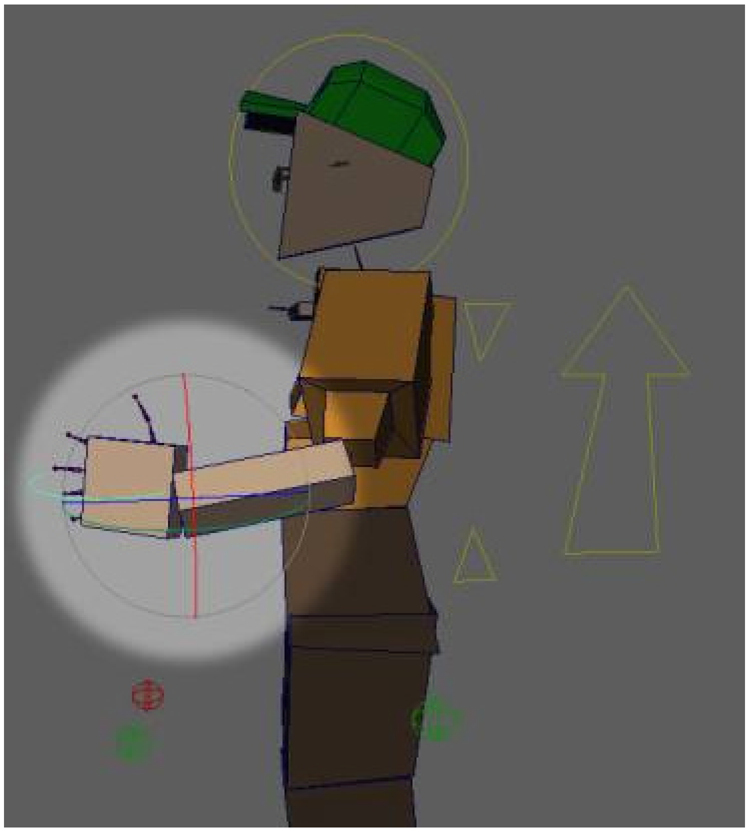
Extra gimbal control

While changing the rotation orders can certainly help remove some of the problems with gimal locking, it can't remove them 100%. No matter what rotation order you choose for a control, you may run into a gimbal lock problem at some point. (Imagine a character falling from a great height.. if he has to tuble, twist, and spin.. at some point the animator may want him to turn in some direction you haven't planned on.)

Due to this, it's sometimes necessary to add an extra "gimbal" control for the animators to work with.

- Load **lrKeithGimbalHand.ma**

Copyright Alias|Wavefront, Jason Schleifer and Weta Digital LTD.

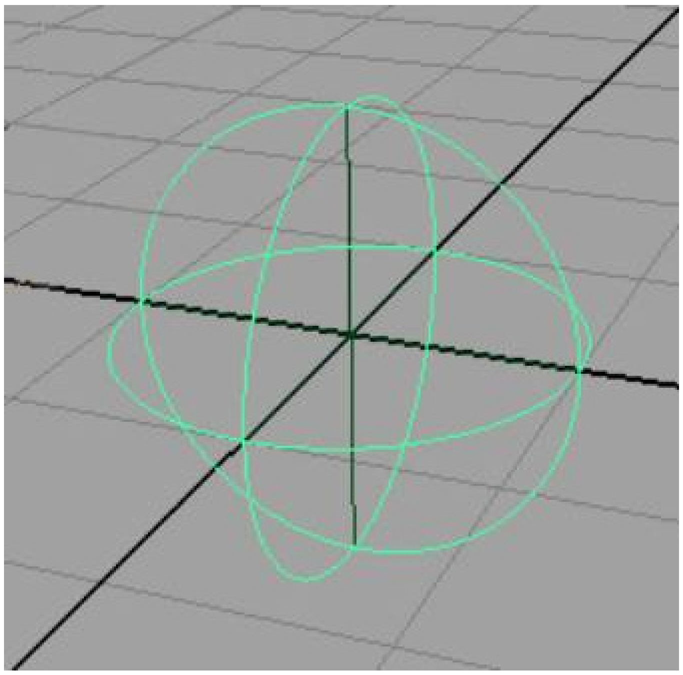


As you can see, we can't rotate Keith's hand up towards his face due to running out of available rotations.

- We will create an object which the animator can recognize as an extra "gimbal" control. An implicit sphere works well for this.

```
createNode implicitSphere;
```

Copyright Alias|Wavefront, Jason Schleifer and Weta Digital LTD.

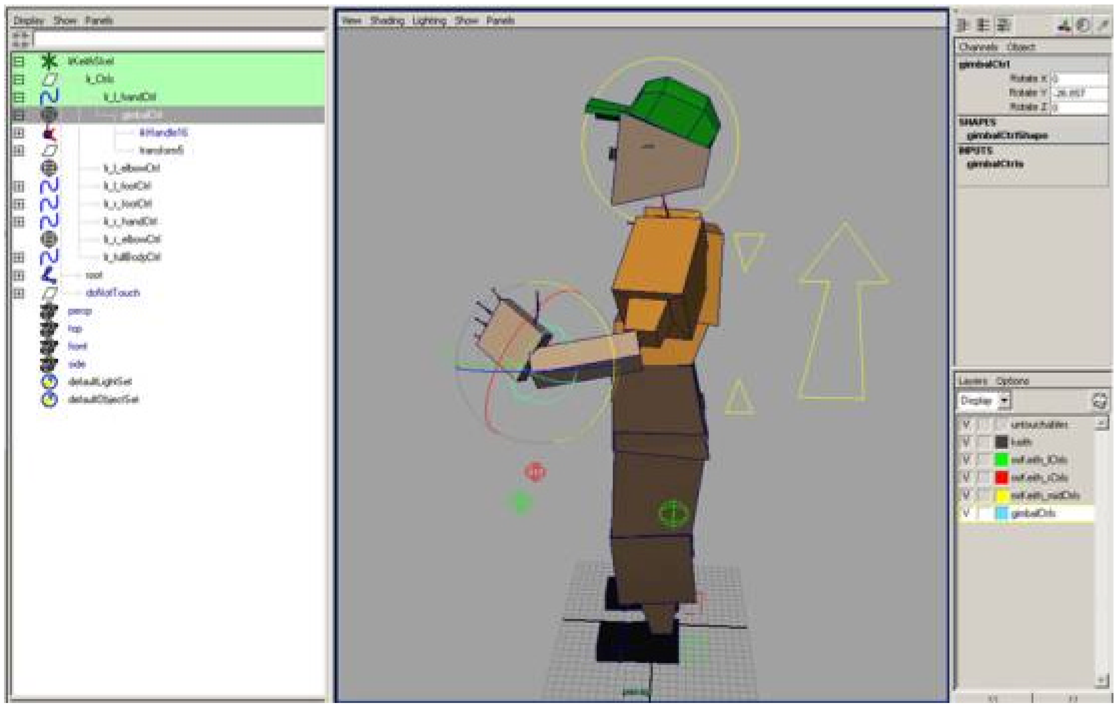


When you create the implicitSphere in this manner, it selects the shape, not the transform.

- Pickwalk up to get the transform.
- Rename it **gimbalCtrl**.
- Parent **gimbalCtrl** under **lr_l_handCtrl.**
- Point and orient constrain the **gimbalCtrl** to **lr_l_handCtrl** to get it in the right position.
- Apply a freezeTransformations.
- Change the gimbalCtrl's radius to make it more visible.

This is accessable through the attribute editor.

- Parent all the objects which were under the orig control to the new **gimbalCtrl**.
- Lock and make unkeyable all translate, scale, and visibility attributes on the gimbalControl.
- If desired, you can make a gimbalCtrl display layer for all the gimbal controls.. that way the animator only sees them when they need to.



It's also highly useful to create a script to generate the gimbal control. This will allow you to add the control to any node the animator has selected, giving them the opportunity to add them only where needed.

```
// find out what objects are selected
string $objs[0];
$objs = `ls -sl`;

// for each object selected, greate a gimbal control
for ($ob in $objs)
{

        // create a gimbal control
        $gimbalShape = `createNode implicitSphere`;

        // get the transform
        $parents = `listRelatives -fullPath -parent $gimbalShape`;
        $gimbalTransform = $parents[0];

        // rename the gimbal node to match the object
        $gimbalTransform = `rename $gimbalTransform ($ob + "_gimbal")`;

        // add a "radius" attribute to the transform and attach it to
        // the shape's radius attribute. This will allow us to
        // scale the control
        addAttr -ln "radius" -at double $gimbalTransform;
        setAttr -k 1 ($gimbalTransform + ".radius");
        connectAttr ($gimbalTransform + ".radius") ($gimbalTransform +
```

Copyright Alias|Wavefront, Jason Schleifer and Weta Digital LTD.

```
"Shape.radius");

        // move the gimbalTransform to the same place as the control
        // using point and orient constraints
        select $ob $gimbalTransform;

        $cmd = ("pointConstraint");
        $pc = `evalEcho $cmd`;

        $cmd = "orientConstraint";
        $oc = `evalEcho $cmd`;

        // get the children of the orig control
        $children = `listRelatives -f -c -type transform -type joint $ob`;

        // parent the children under the gimbal control
        parent $children $gimbalTransform;

        // parent the $gimbalTransform under the gimbal control, in
        // order to make sure we still get the right name for
        // $gimbalTransform, select it, parent it,
        // and then grab the name of the selected object
        select $gimbalTransform;
        parent $gimbalTransform $ob;
        $sel = `ls -sl`;
        $gimbalTransform = $sel[0];

        // delete the constraints
        delete $oc $pc;

        // lock translate, scale, and visibility of the gimbal control
        string $attrs[ ] = {"tx", "ty", "tz", "sx", "sy", "sz", "v"};
        for ($at in $attrs)
        {

                setAttr -l 1 ($gimbalTransform + "." + $at);
                setAttr -k 0 ($gimbalTransform + "." + $at);

        }

}

// select the orig objects
select $objs;
```

Add Customized PickWalking Tools

Because there is so much to think about when animating, it can be extremely

Copyright Alias|Wavefront, Jason Schleifer and Weta Digital LTD.

helpful for the animator to have tools to easily move between controls without having to hunt and peck for what they're going to select, and without having to go to a separate interface.

PickWalking is a great way of doing this (hitting the arrow keys to step to the different controls), however, it is extremely rare that the next node the animator will want is going to be anywhere near the one they have selected in the hierarchy. Thus, you can use **jsPickWalk** to provide that fucntionality.

jsPickWalk does performs it's navigation through connections as opposed to hierarchy, so there is no limit as to what the next object the animator is going to select.

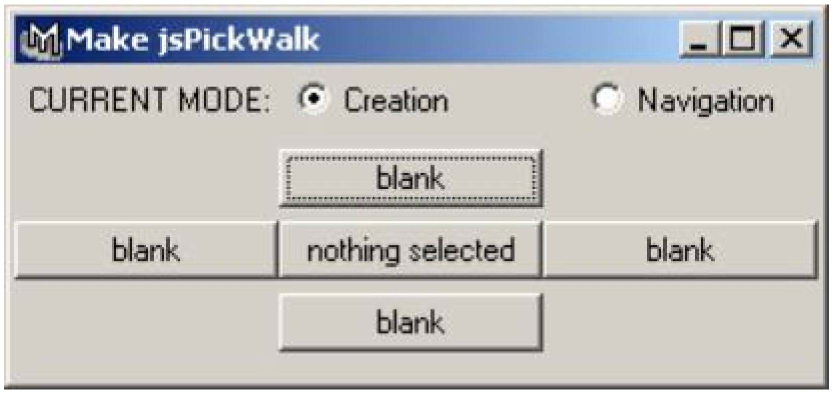
jsPickWalk comes with two different methods of use, definition and navigation. When defining what the animator is going to pickWalk to, you can use the **jsMakePickWalkUI** to easily define what is going to be selected. To navigate, the command **jsPickWalk <dir>** to navigate to either up, down, left, or right, based on the current selection.

Defining the navigation

- Bring up the interface by sourcing **jsPickWalk**, and then executing **jsMakePIckWalkUI**

```
source jsPickWalk; jsMakePickWalkUI;
```

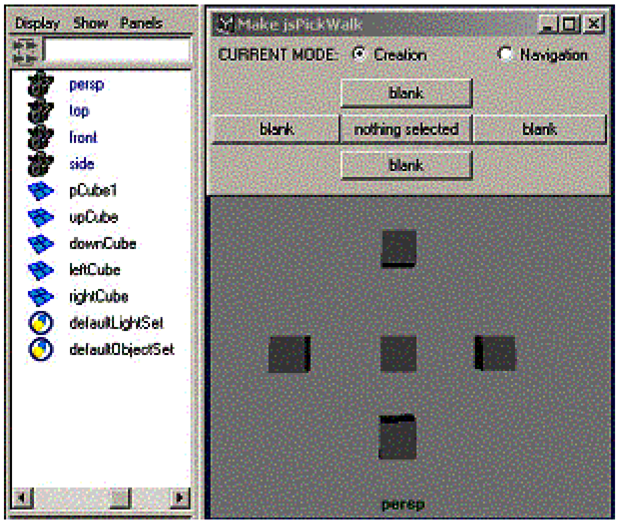
- The interface will come up looking like the following image:

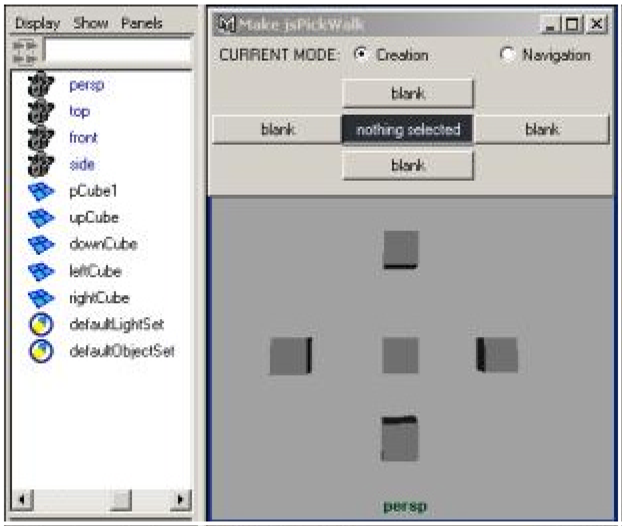

Make jsPickWalk

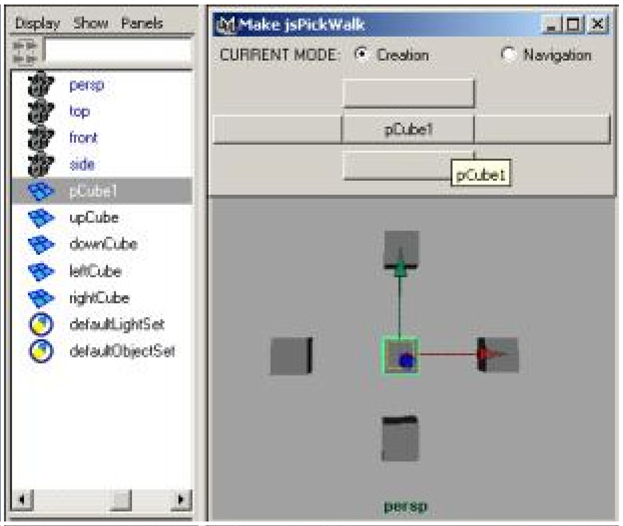
CURRENT MODE: (● Creation) ○ Navigation

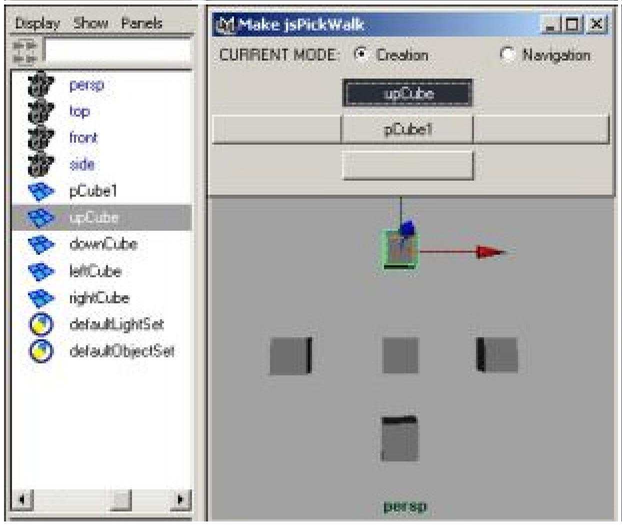
The **Current Mode** section defines what mode the interface is in.

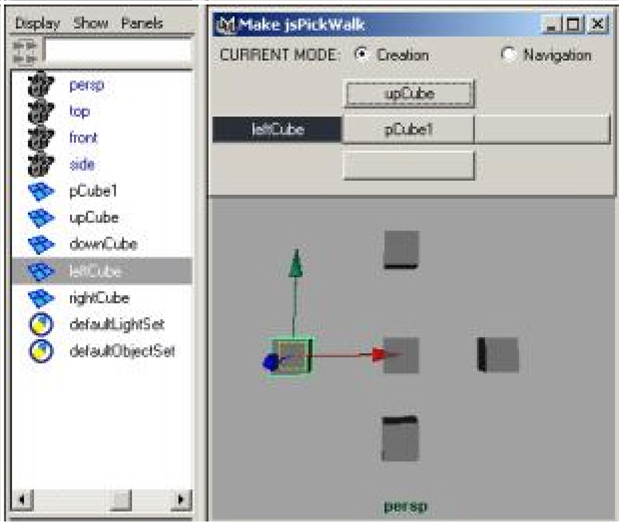
In **Creation** mode, clicking on the **middle** button will put the selected object in the middle, and show the relationships already defined in the **up, down, left**, and **right** buttons. Selecting another object and clicking on one of those buttons will define that object as the target for pickWalking in that direction.



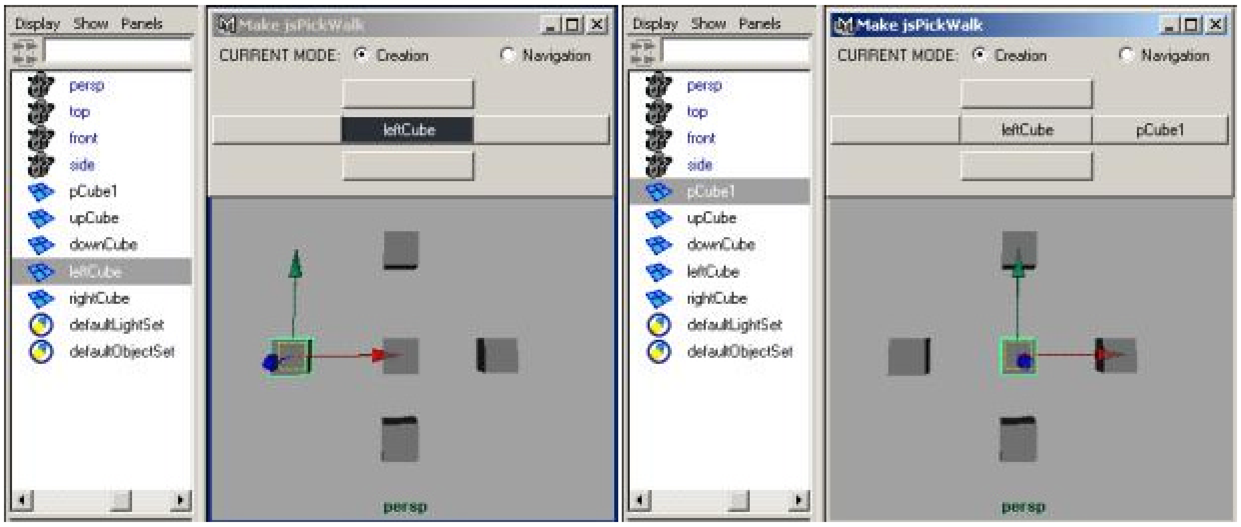








Copyright Alias|Wavefront, Jason Schleifer and Weta Digital LTD.



In **Navigation** mode, clicking on one of the up, down, left or right buttons will place that object in the middle and re-draw the layout based on the newly selected object. You can easily check your navigation this way

Navigating

To navigate, you can either use the interface shown above, or map the navigation to hotkeys using the following commands:

```
jsPickWalk "up";
jsPickWalk "down";
jsPickWalk "left";
jsPickWalk "right";
```

Copyright Alias|Wavefront, Jason Schleifer and Weta Digital LTD.

V. Automatic/Keyable Shoulder Solution

Why need one?

- The clavical is an extremely important aspect of the character, which quite often gets overlooked. It's motion is integral to creating realistic characters.
- Because of the tight deadlines, and the amount of revisions that are getting done, reducing the amount of animating that the animator needs to do is extremely benifitial - an automated solution provides an answer to this problem.
- Animators need to have utmost control over what it is they're animating, thus a keyable solution is absolutely necessary.
- Coming up with a solution which works for FK is easy, simply rotate the shoulder joint based on the rotation of the upper arm. However, when you try and do something like that for an IK arm, it doesn't quite work as well. You end up with a cycle. (load shoulderNotWorking.mb to see an example of how this doesn't work)

The following solution provides a control which gives the animator the ability to key the automation on and off, animate on TOP of the automation, and even provide a percentage of influence.

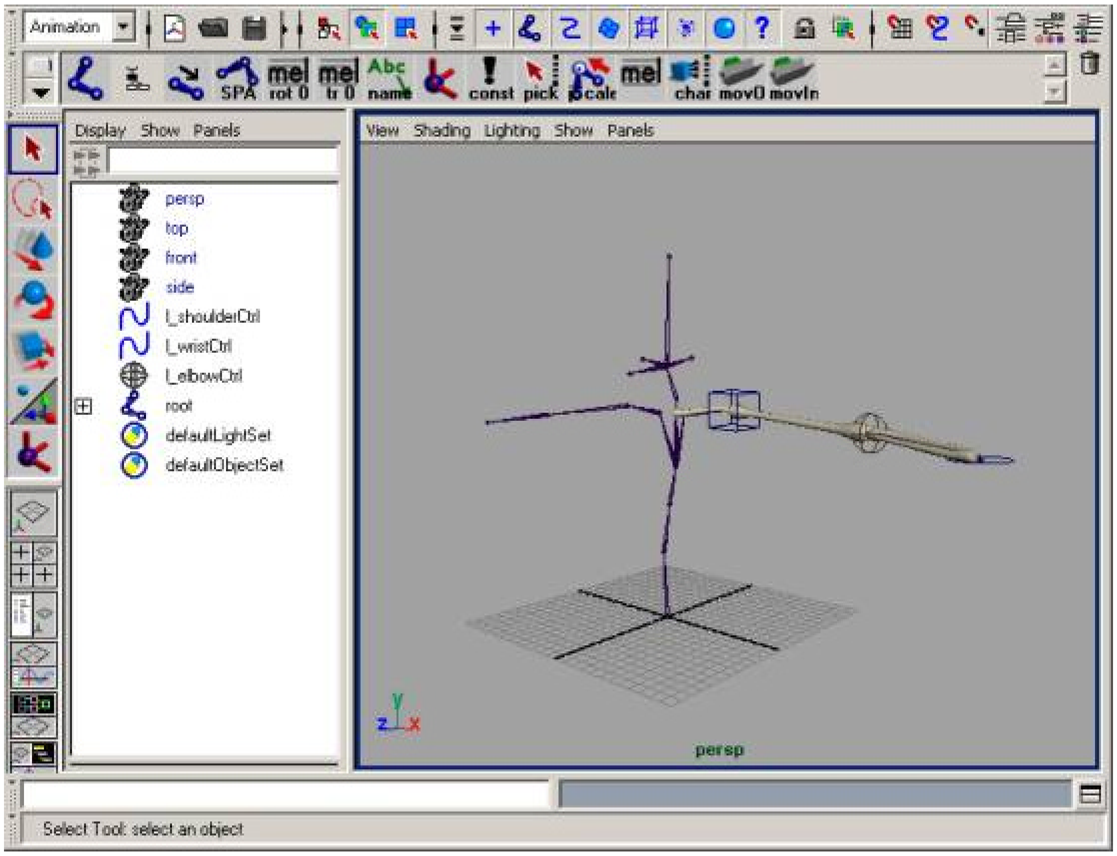
Creating the Control Structure

Given a simple torso, clavical, arm structure, create the necessary additional controls.

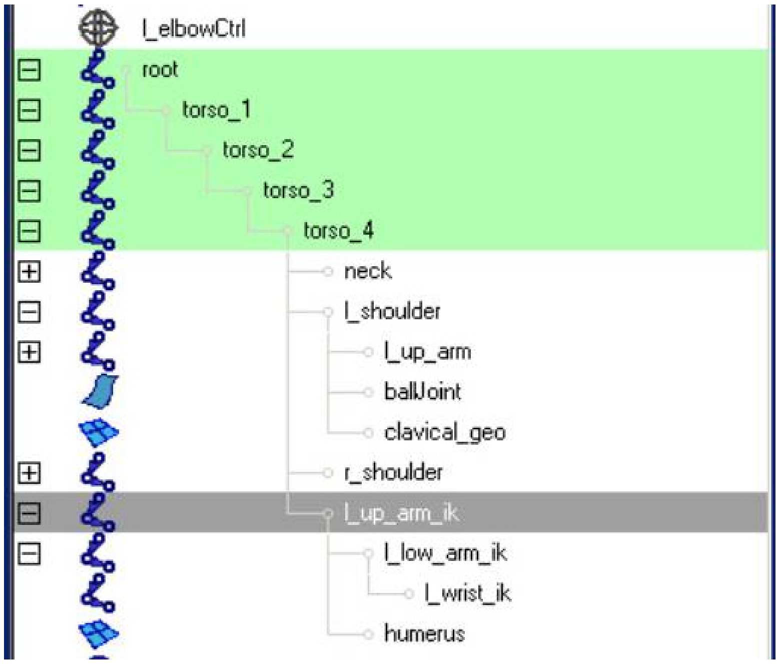
boneStructure_done.mb
boneStructure_start.mb

Load **boneStructure_start.mb**

Copyright Alias|Wavefront, Jason Schleifer and Weta Digital LTD.



- Duplicate **l_up_arm** and parent the duplicate under **torso_4.**
- Rename the duplicate hieararchy **l_up_arm_ik**, **l_low_arm_ik**, **l_wrist_ik**.
- Delete all the extra geometry under the l_up_arm_ik hierarchy.

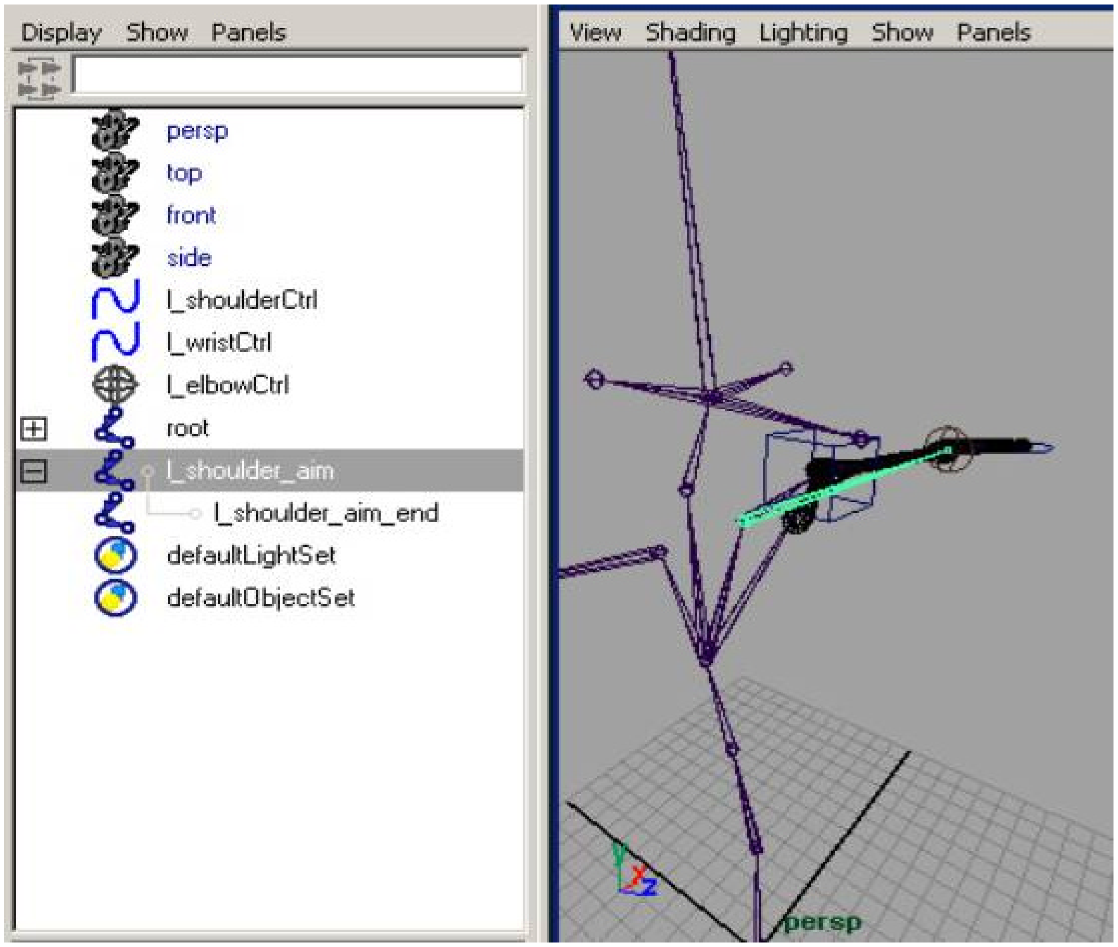


- Create a joint chain which will be used to aim at the **l_low_arm_ik** elbow. It

Copyright Alias|Wavefront, Jason Schleifer and Weta Digital LTD.

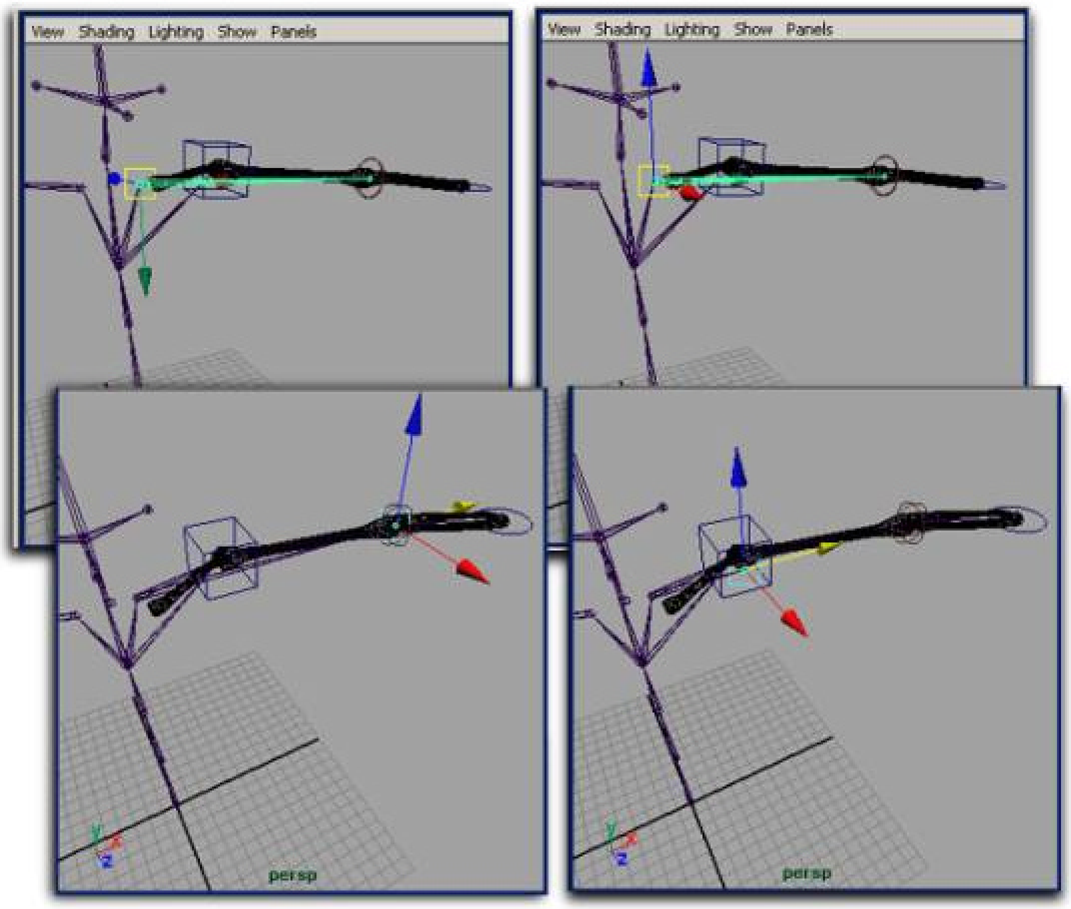
should start at **l_shoulder**, and end at **l_low_arm_ik**.

- Name these joints **l_shoulder_aim** and **l_shoulder_aim_end**.



- Orient the joint so it's **Z** is pointing up using **jsOrientJointUI**, then decrease the ty value of **l_shoulder_aim_end** until it's located roughly around the same location as **l_up_arm** (about 5).

Copyright Alias|Wavefront, Jason Schleifer and Weta Digital LTD.


View Shading Lighting Show Panels
View Shading Lighting Show Panels
persp
persp

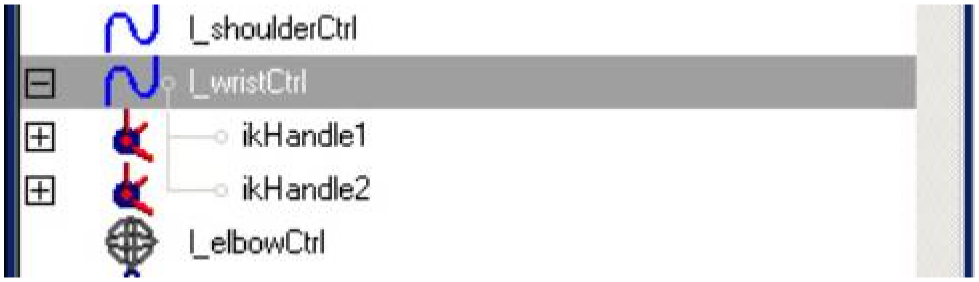
- Parent **l_shoulder_aim** under **torso_4.**

Generate ikHandles for the joints

- Choose the IK Handle tool and bring up the option box.

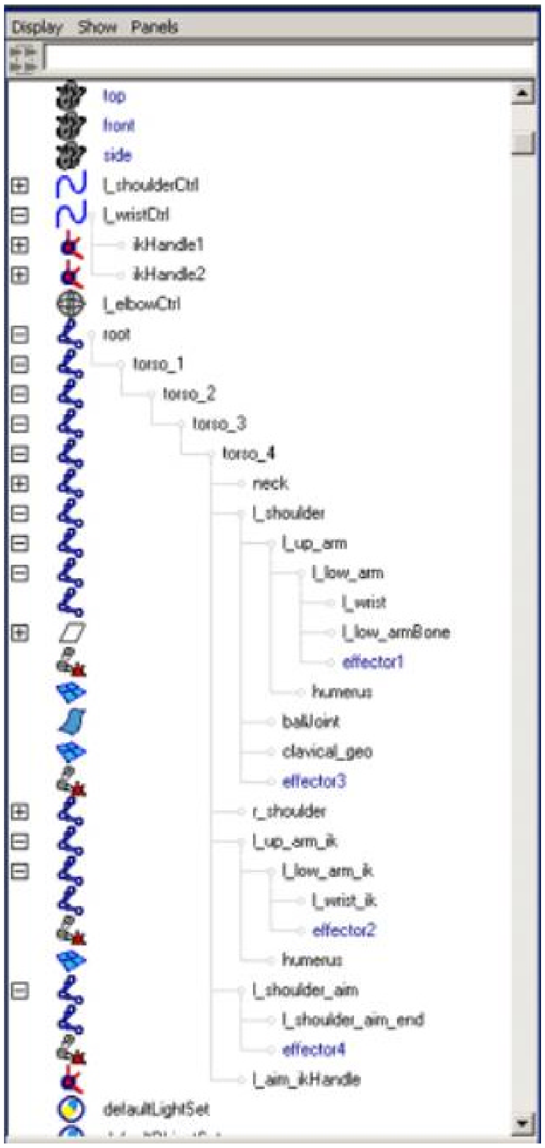
  Set the solver type to ikRPsolver.

- Create an ikHandle for **l_up_arm** to **l_wrist**, and another one for **l_up_arm_ik** to **l_wrist_ik**.
- Parent both ikHandles under **l_wristCtrl**.
- Make the **l_elbowCtrl** a poleVectorConstraint for each of the ikHandles.



Copyright Alias|Wavefront, Jason Schleifer and Weta Digital LTD.

- Bring up the option box for the ikHandle tool again and change the **Solver Type** to **ikSCsolver**.
- Create an ikHandle from **l_shoulder** to **l_up_arm**.
- Parent the ikHandle under **l_shoulderCtrl**.
- Create an ikHandle from **l_shoulder_aim** to **l_shoulder_aim_end**.
- Parent this ikHandle under **torso_4,** and name it ***l_aim_ikHandle***.

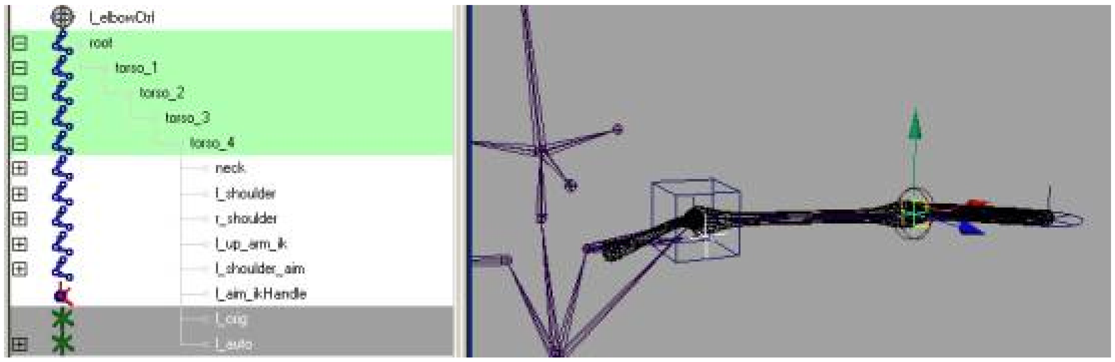


```
top
front
side
l_shoulderCtrl
l_wristCtrl
    ikHandle1
    ikHandle2
l_elbowCtrl
root
    torso_1
        torso_2
            torso_3
                torso_4
                    neck
                    l_shoulder
                        l_up_arm
                            l_low_arm
                                l_wrist
                                l_low_armBone
                                effector1
                            humerus
                        ballJoint
                        clavical_geo
                        effector3
                    r_shoulder
                    l_up_arm_ik
                        l_low_arm_ik
                            l_wrist_ik
                            effector2
                        humerus
                    l_shoulder_aim
                        l_shoulder_aim_end
                        effector4
                    l_aim_ikHandle
defaultLightSet
```

Create the constraints for the l_aim_ikHandle.

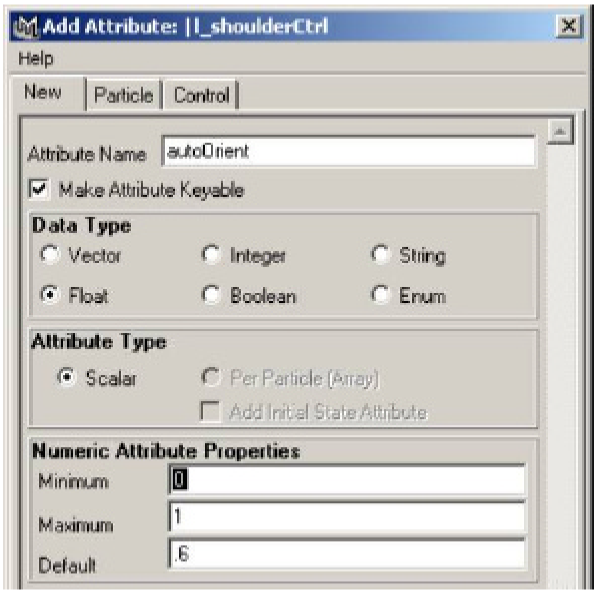
- Create a locator, name it ***l_auto***, and constrain it to **l_low_arm_ik**
- Create another locator, name it ***l_orig.*** Put it in the same position as **l_shoulder_aim_end** but do not constrain it.
- Parent both locators under **torso_4.**

Copyright Alias|Wavefront, Jason Schleifer and Weta Digital LTD.



- Select both locators, and select the **l_aim_ikHandle**, and create a pointConstraint.
- Add an attribute to **l_shoulderCtrl** called **"autoOrient".**

  Give it a **Min** value of **0**,
  Max of **1**,
  Default of **.6**.


Add Attribute: |l_shoulderCtrl

Help

New | Particle | Control

Attribute Name: autoOrient

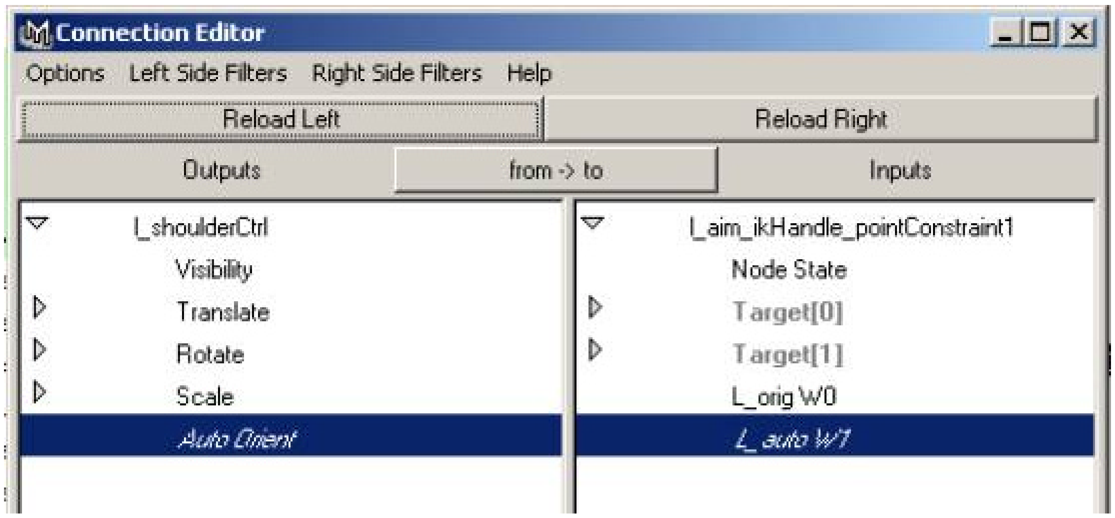
[✓] Make Attribute Keyable

Data Type
Vector | Integer | String
[✓ Float] | Boolean | Enum

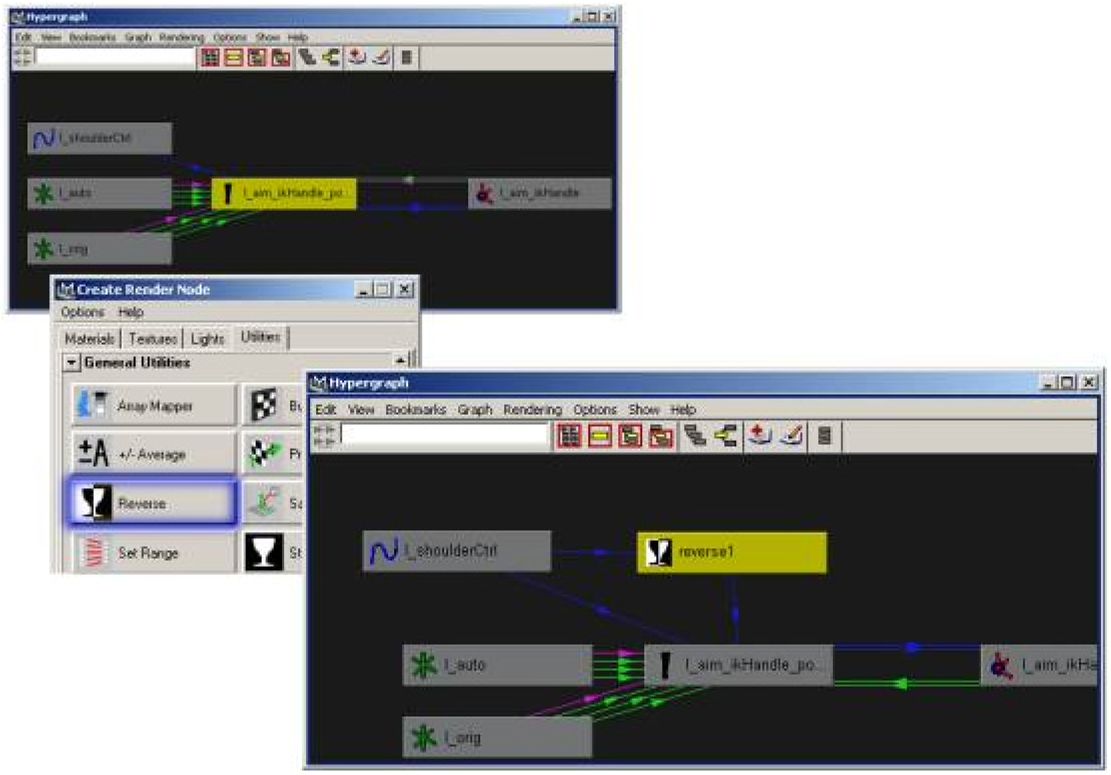
Attribute Type
[✓ Scalar] | Per Particle (Array)
Add Initial State Attribute

Numeric Attribute Properties
Minimum: 0
Maximum: 1
Default: .6

- Using the connectionEditor, connect **l_shoulderCtrl.autoOrient** to the pointConstraint's **l_autoW#** attribute.



- In the Hypergraph, create a reverse node, and connect **l_shoulderCtrl.autoOrient** to the **inputX** of that node, then connect the **outputX** to the pointConstraint's **l_origW#** attribute.



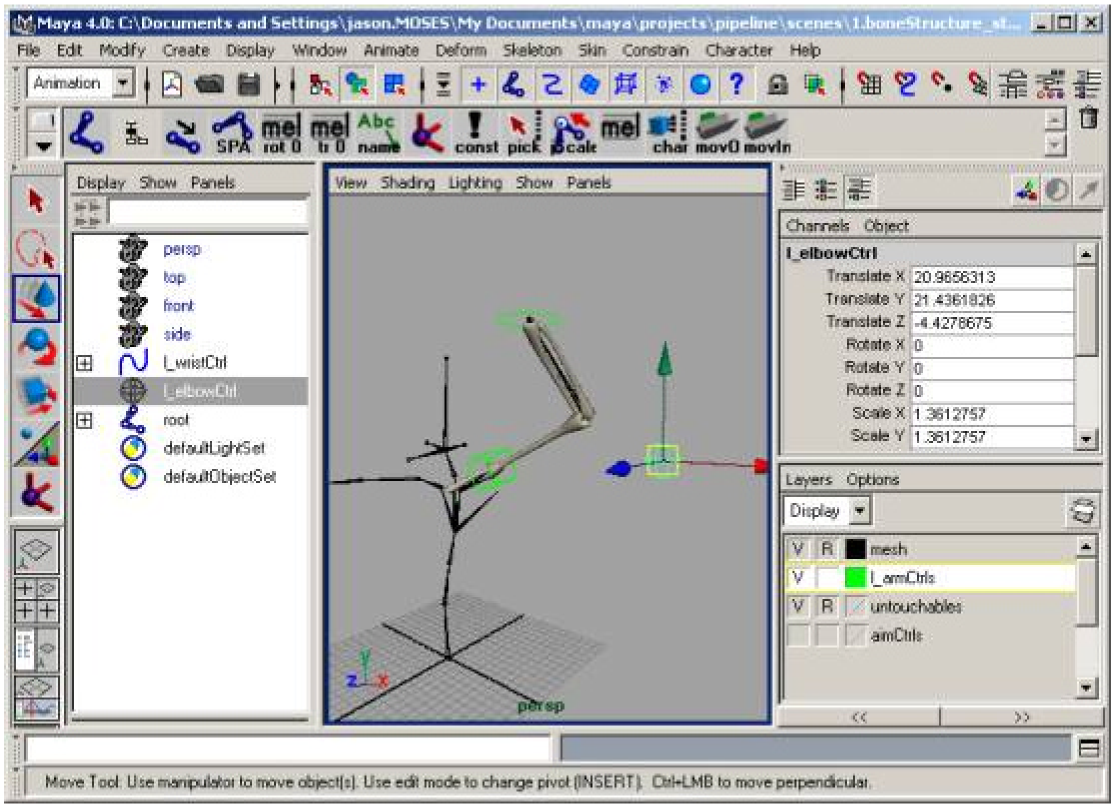
- Parent **l_shoulderCtrl** to **l_shoulder_aim**
- Unparent the ikHandle which is under **l_shoulderCtrl**,
- Perform a freezeTransformations on **l_shoulderCtrl**, and re-parent the ikHandle.

  Now when you rotate the handle wrist control around, you can see how the shoulder will follow the motion of the elbow. You can control how much this following action happens by animating the **autoOrient** attribute on **l_shoulderCtrl**.

Finishing up the Controls

Hide unecessary nodes & Create appropriate display layers.

- Hide all the ikHandles.
- Hide the two locators.
- Create **l_armCtrls** display layer.
- Create untouchables display layer.
- Create **aimCtrls** display layer.
- Select all the joints that the animator shouldn't be touching, and put them in untouchables.
- Select **l_shoulder_aim**, **l_shoulder_aim_end**, **l_up_arm_ik**, **l_low_arm_ik**, **l_wrist_ik** and put them in **ikCtrls**.
- Select **l_wristCtrl, l_elbowCtrl,** and **l_shoulderCtr**l and put them in **l_armCtrls**.
- Set **untouchables** and **ikCtrls** to reference.
- Set a nice color for **l_armCtrls**.


Maya 4.0: C:\Documents and Settings\jason.MOSES\My Documents\maya\projects\pipeline\scenes\1.boneStructure_st...

l_elbowCtrl

Translate X	20.9656313
Translate Y	21.4361826
Translate Z	-4.4278675
Rotate X	0
Rotate Y	0
Rotate Z	0
Scale X	1.3612757
Scale Y	1.3612757

V	R	■	mesh
V		🟩	l_armCtrls
V	R		untouchables
			aimCtrls

Move Tool: Use manipulator to move object(s). Use edit mode to change pivot (INSERT). Ctrl+LMB to move perpendicular.

File Edit Modify Create Display Window Animate Deform Skeleton Skin Constrain Character Help
Animation
SPA rot 0 tr 0 name const pick scale char mov0 movIn
Display Show Panels
persp
top
front
side
l_wristCtrl
l_elbowCtrl
root
defaultLightSet
defaultObjectSet
View Shading Lighting Show Panels
persp
Channels Object
Layers Options
Display

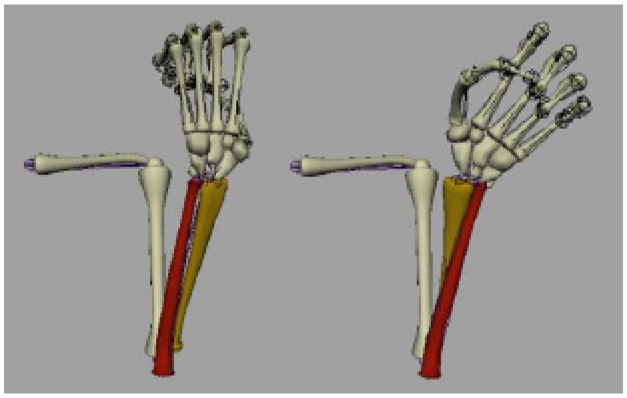
Get rid of Unecessary Attributes

- Lock and make unkeyable all rotate, scale, and the visibility attribute for **l_elbowCtrl**.

- Lock and make unkeyable scale and visibility for both **l_shoulderCtrl** and **l_wristCtrl**.

Copyright Alias|Wavefront, Jason Schleifer and Weta Digital LTD.

VI. REALISTIC FOREARM TWIST



What is a "realistic forearm"

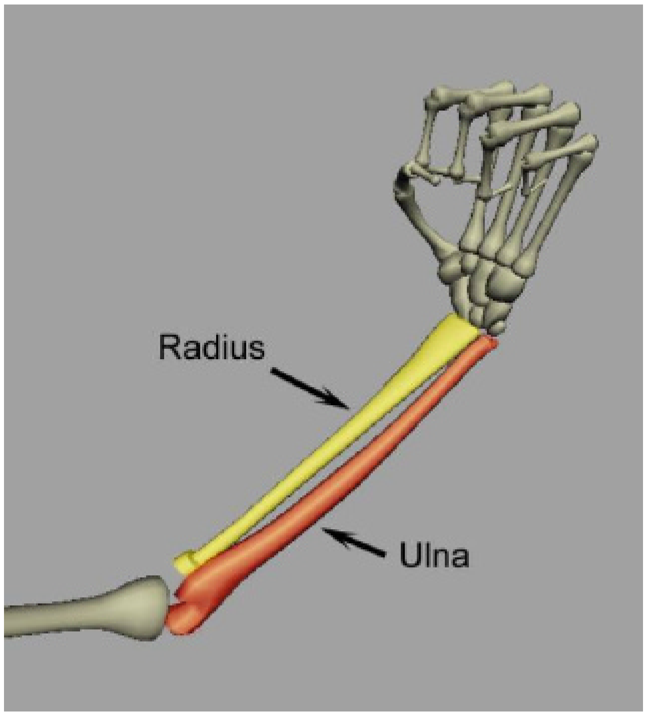
When most people set up a forearm/wrist control in Maya, they do one of two things:

- Forget about doing anything special for their character's forearm and end up with "candy-wrapper" hands.
- Create a joint in the middle of their lower arm and put the twist there, using a lattice deformer to deform the skin.

While the second option will work in many cases, if you're creating a realistic character, it's important to follow what's really happening under the skin.

In actuality, the twist which happens in your forearm is caused by the movement of two bones, the radius and the ulna.

Copyright Alias|Wavefront, Jason Schleifer and Weta Digital LTD.



- The **ulna** starts at your elbow, and travels down to the top inside of your wrist (it's that little bump on your wrist where your pinky is).
- The **radius** also starts up near your elbow and reaches down to the other side of your wrist.

When you twist your wrist, the ulna moves slightly to keep itself connected to the end your wrist, and the radius rotates over the ulna, causing the twisting of the muscles. This is how your elbow stays in one place while you wrist can twist around.

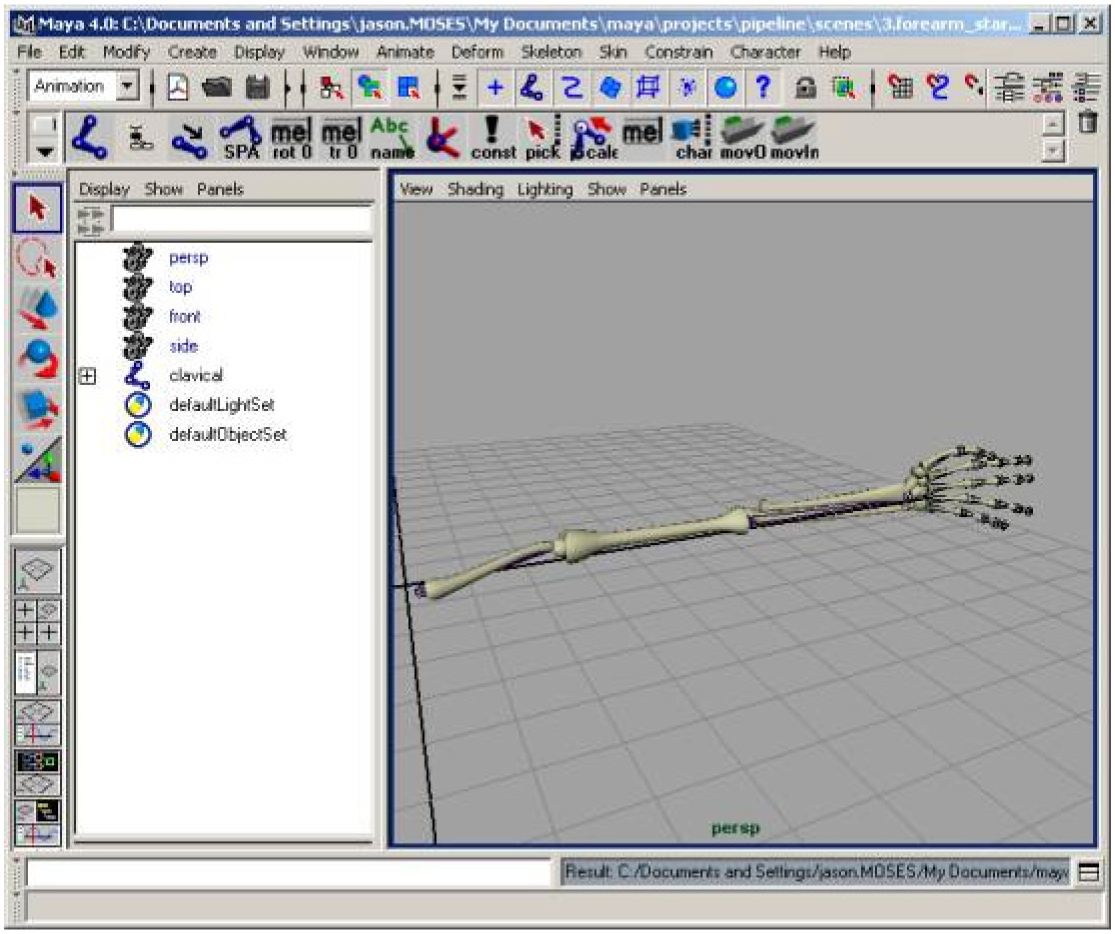
Creating the Control Structure

Moving the raidus and ulna correctly based on wrist rotation

forearm_start.mb

- Load **forearm_start.mb**

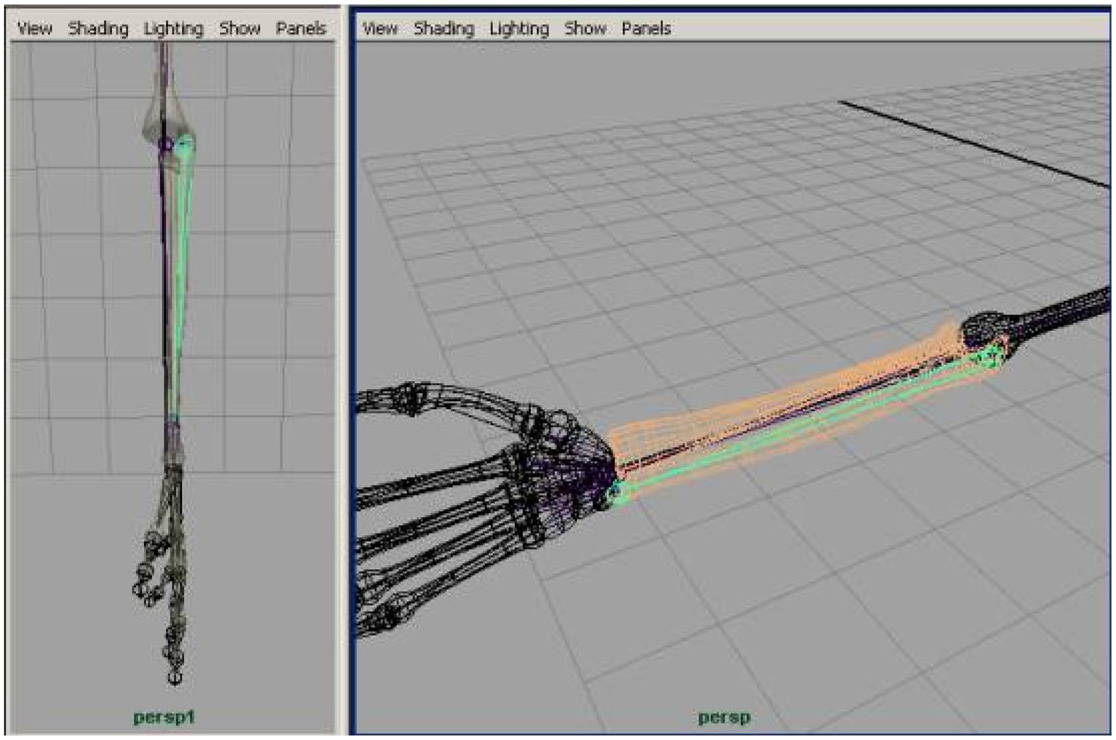
Copyright Alias|Wavefront, Jason Schleifer and Weta Digital LTD.


Maya 4.0: C:\Documents and Settings\jason.MOSES\My Documents\maya\projects\pipeline\scenes\3.forearm_star...
File Edit Modify Create Display Window Animate Deform Skeleton Skin Constrain Character Help
Animation
SPA rot 0 tr 0 name const pick scale char mov0 movIn
Display Show Panels
persp
top
front
side
clavical
defaultLightSet
defaultObjectSet
View Shading Lighting Show Panels
persp

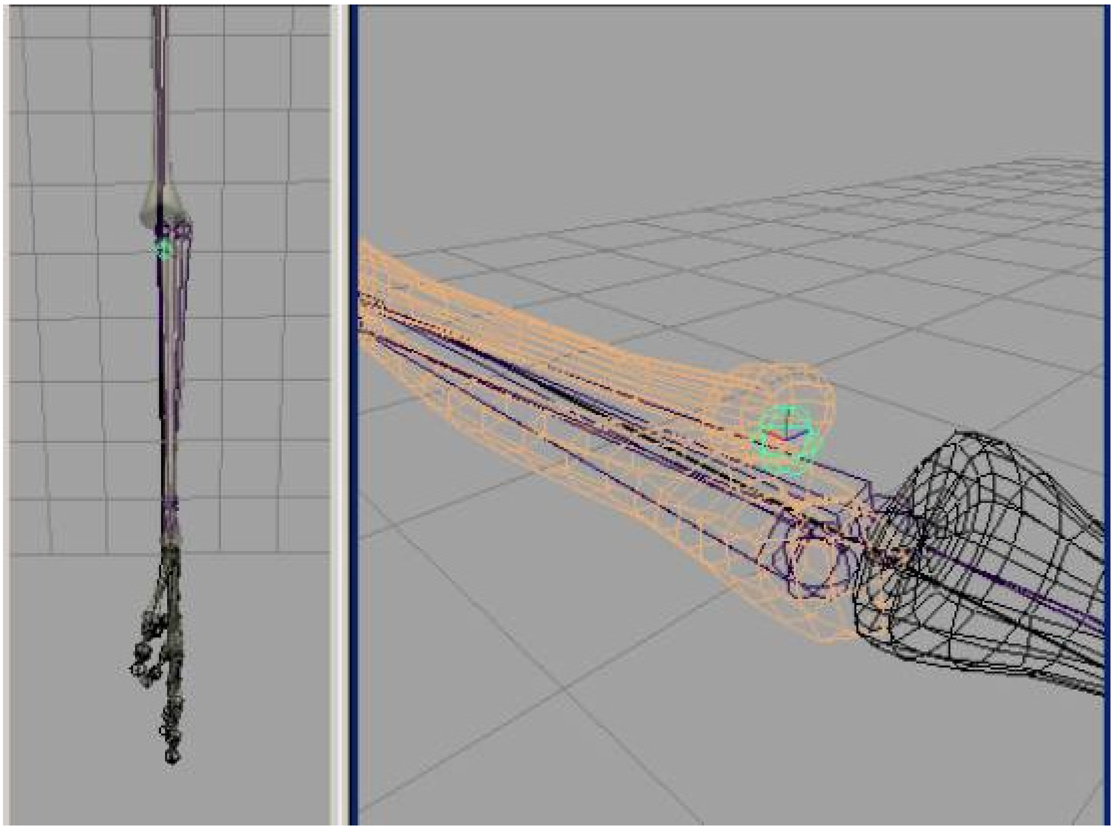
Result: C:/Documents and Settings/jason.MOSES/My Documents/maya

- Create joints for the radius and ulna.
    - Draw a joint chain going from the elbow of the ulna down to the end.
    - Name it **ulna_joint**, and parent the ulna geometry under it.

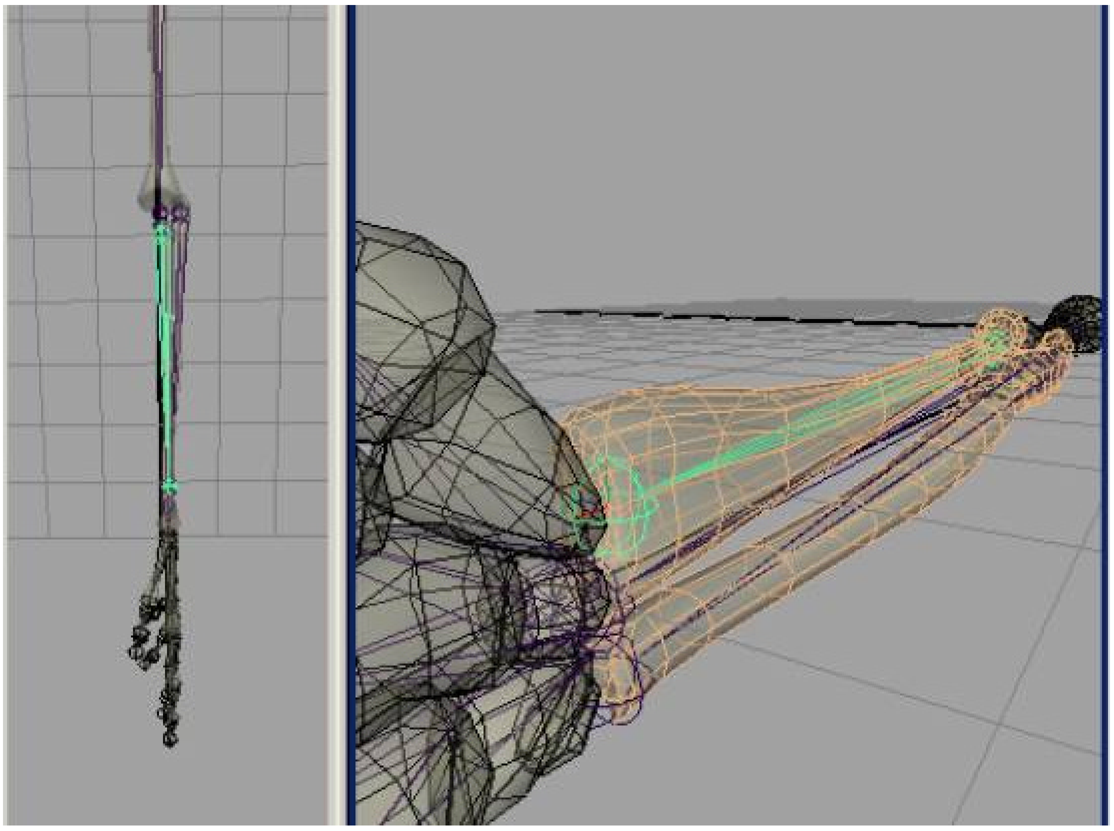
Copyright Alias|Wavefront, Jason Schleifer and Weta Digital LTD.


View Shading Lighting Show Panels
persp1
View Shading Lighting Show Panels
persp

- Draw a chain going from the place where the radius touches the ulna (near the elbow) to the end of the radius.
- Name it **radius_joint**, and parent the radius geometry under it.



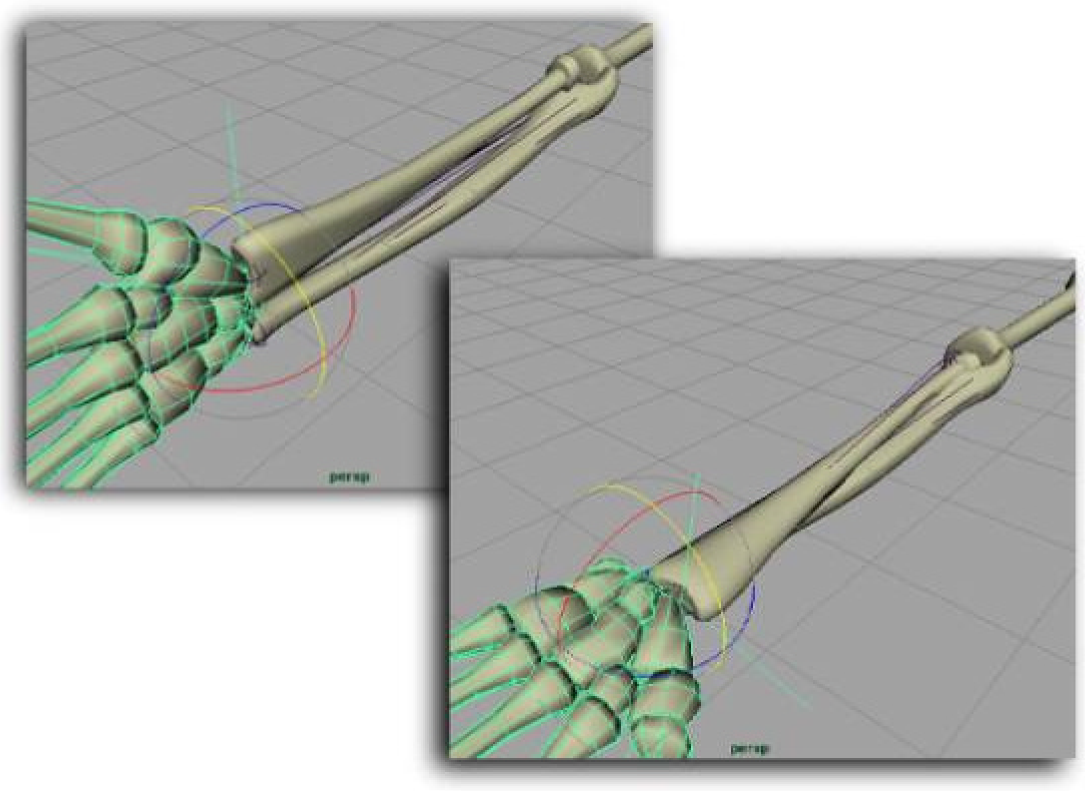
Copyright Alias|Wavefront, Jason Schleifer and Weta Digital LTD.



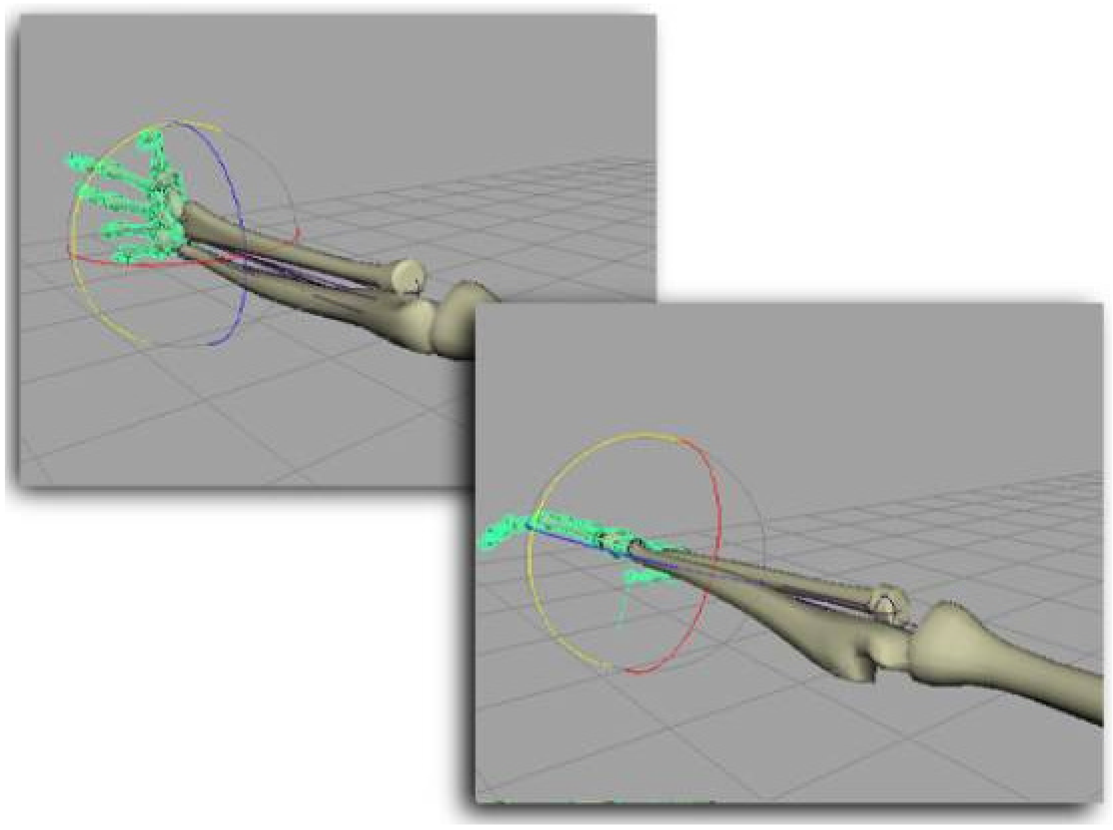
  - Move the joints if you need to to get them in the desired places.
  - Use **jsOrientJoint** to make sure the joints are all oriented correctly
- Create an ikHandle for the radius joint.
- Parent the ikHandle under wrist.

Now when the wrist rotates, the radius will rotate with it.

Copyright Alias|Wavefront, Jason Schleifer and Weta Digital LTD.



- Hide the ikHandle.
- Create an ikHandle for the ulna joint.
- Parent the ikHandle under the wrist.



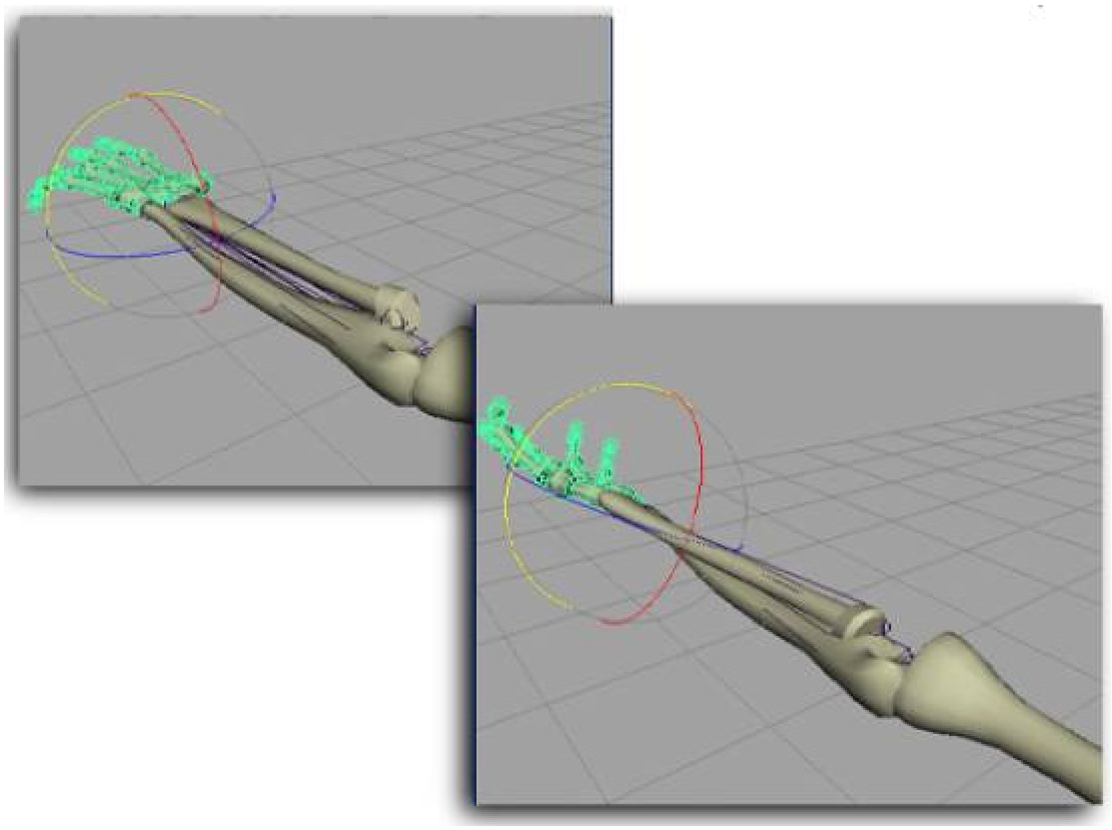
Notice when the wrist rotates, the elbow breaks. This is because the ulna doesn't actually twist, it only moves a bit to keep the radius attached.

To keep this from breaking, instead of parenting the ikHandle, we'll point constrain it to a locator in the same place, and then parent the ikHandle under low_arm.

- Create a locator.
- Put it in the same location as the ikHandle
- PointConstrain the ikHandle to the locator.
- Parent the locator under wrist.
- Parent the ikHandle under **low_arm**.
- Hide the locator.

Now rotate the wrist and see how the arm behaves much better.

Copyright Alias|Wavefront, Jason Schleifer and Weta Digital LTD.

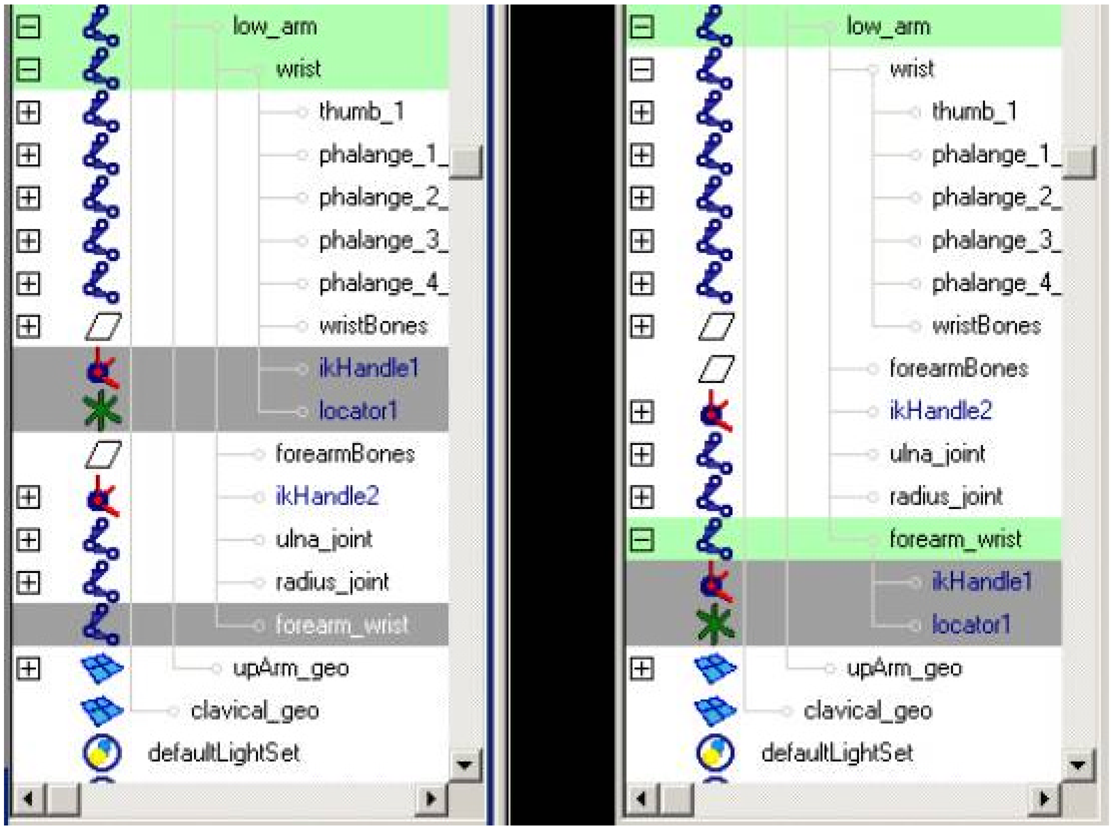


- The next step is to parent the joints under low_arm. Now you can rotate low_arm and the wrist and everything will behave correctly.

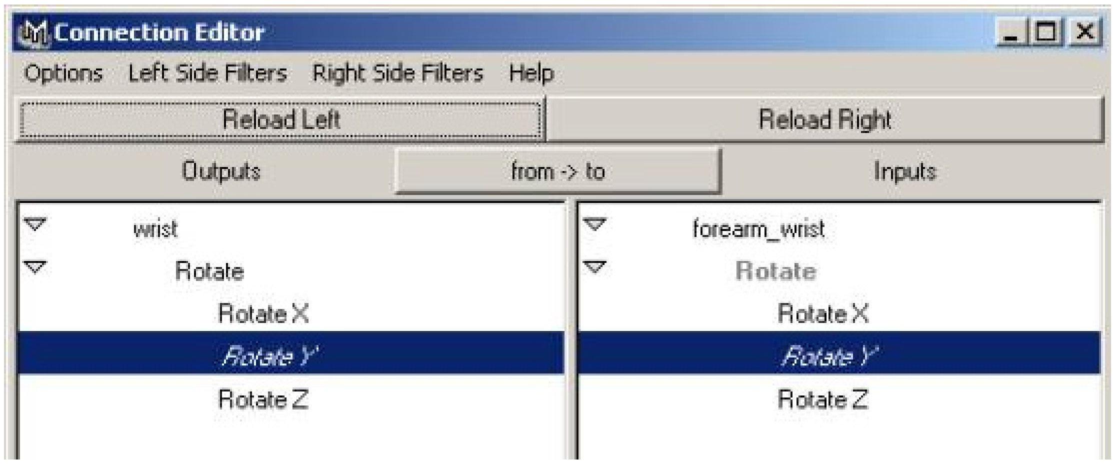
  Notice the wrist is working correctly for twisting in **Y**, however, if you twist it in **X** or **Z**, you get a little bit of movement in the forearms. In a real arm, there would be little to no motion here. If you want to replicate that behavior, you need to create another "wrist" to parent the radius and ulna controls under. Then connect the rotateY of your animated wrist to the rotateY of that new wrist.

- Duplicate **wrist**.
- Name it ***forearm_wrist***.
- Delete all the children of the duplicated wrist.
- Parent **ikHandle1** and **locator1** to the **forearm_wrist**.

Copyright Alias|Wavefront, Jason Schleifer and Weta Digital LTD.



- Using the conneciton editor, connect the **rotateY** of wrist to the **rotateY** of **forearm_wrist**.



- Hide **forearm_wrist**.

VII. FK/IK BACK SOLUTION

WHY COME UP WITH A BACK SOLUTION?

What does the back need to do?
Some animators like to work in an FK mode of using the back, others prefer IK.
The animator should be able to move the hips and shoulders independently of eachother.

- Example: Animate a character turning around in a circle as if he's going to wrestle. After the blocking is done, you want to be able to keep the shoulders somewhat steady, while allowing for the hips to move independently.
To make sure the back is easy to manipulate, it should stretch to meet the root and shoulder controls.

CREATING THE BACK

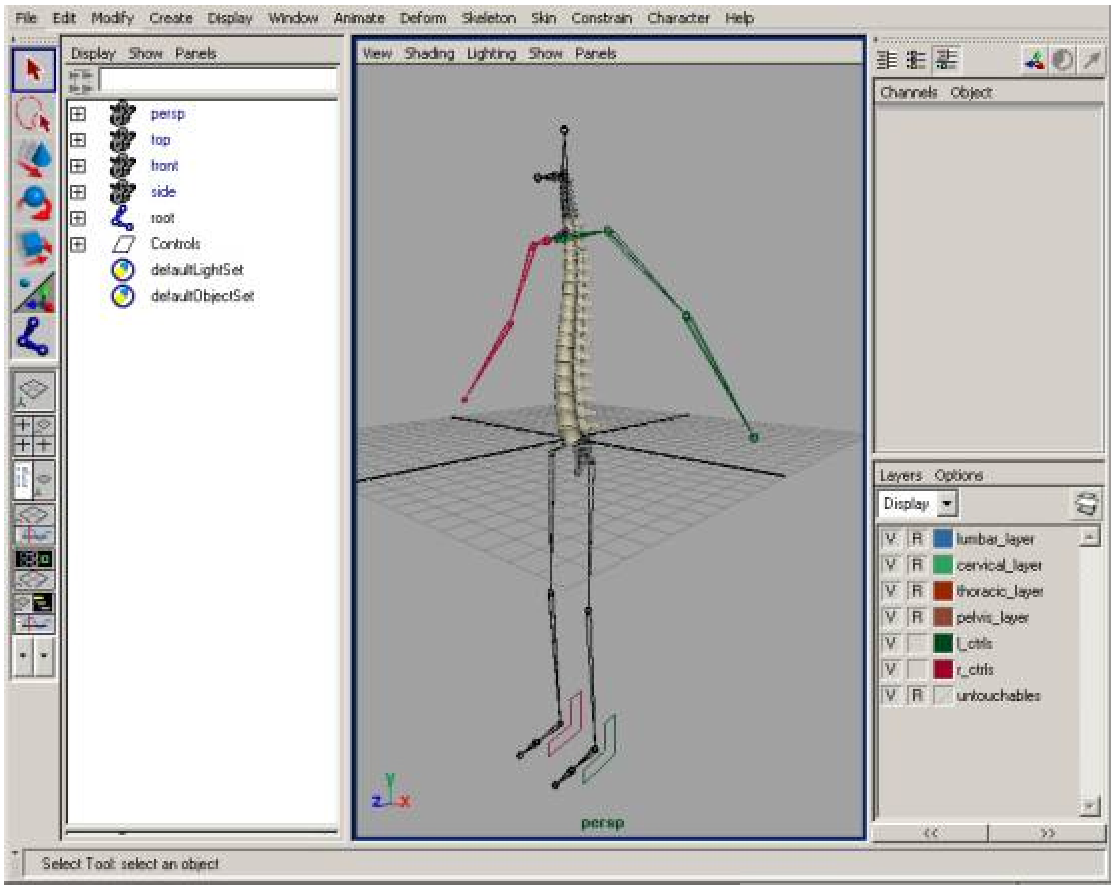
Notes on joints created for the back

- Use the joint tool to create a common s-curve back shape. The more joints you use, the better, as these joints are merely going to be a representation of the curve used to manipulate them.
- Orient the joints correctly using **jsOrientJoint**. This script will allow you to make sure the joints all are oriented towards eachother, and facing the correct direction. It's extremely important for the joints to ONLY have translations in Y, as this is the axis we're going to be scaling them in.

Generate splineIk for the back joints

backSolver_start.mb

Copyright Alias|Wavefront, Jason Schleifer and Weta Digital LTD.



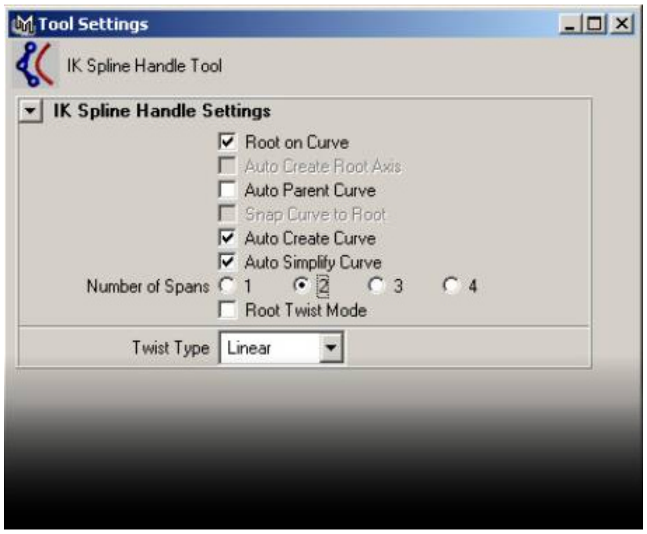
- Open the file **backSolver_start.mb** .

  This file has a back already generated using the above method, with geometry parented underneath so you can see what the back is doing when it's rotated.

- Choose the IK Spline Handle tool and bring up the option box.

  Auto create Curve to **On**,
  Root on Curve to **On**
  Auto Simplify Curve to **On**.
  Auto parent curve to **off**.
  Set Auto simplify curve to **2**.

Copyright Alias|Wavefront, Jason Schleifer and Weta Digital LTD.


Tool Settings

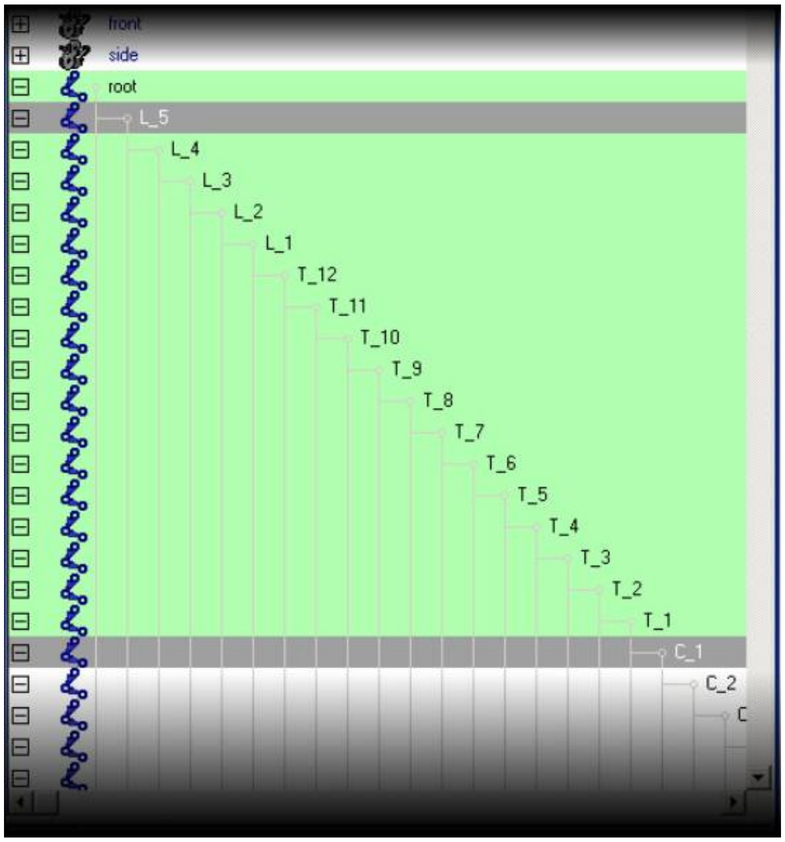
IK Spline Handle Tool

IK Spline Handle Settings

[✓] Root on Curve
[ ] Auto Create Root Axis
[ ] Auto Parent Curve
[ ] Snap Curve to Root
[✓] Auto Create Curve
[✓] Auto Simplify Curve
Number of Spans: 1 [✓ 2] 3 4
[ ] Root Twist Mode

Twist Type: Linear

Choose **L_5** and **C_1**. This will create an ikHandle and a curve from the lumbar joint to the cervical. Name the curve "**back_curve**", and the ikHandle "**back_ikHandle**".

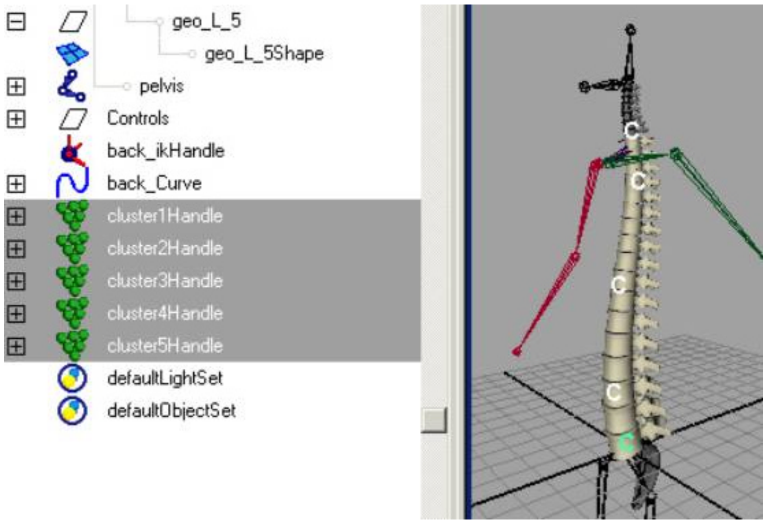


- Hide the ikHandle.

Create clusters for the curve.

In order to move the back around, we're going to create two main controls. A shoulder control and a hip control. These controls will move the curve through the use of clusters.

- Create a cluster for each cv on the curve.



Create "stretchyness" on the joints.

We need to allow the joints to stretch in order to make the control work correctly. To do this, we measure the arcLength of the curve. Divide the original arcLength by the current arcLength, and multiply that result by each of the joint's original **translateY** attributes. This value will be what the joint's new **translateY** will be.

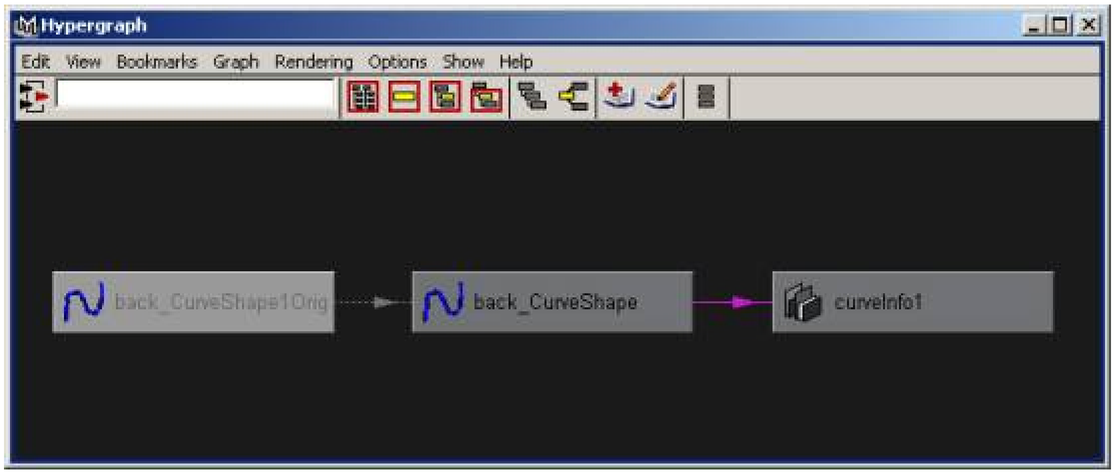
- Select the curve and type:

```
arclen -ch 1;
```

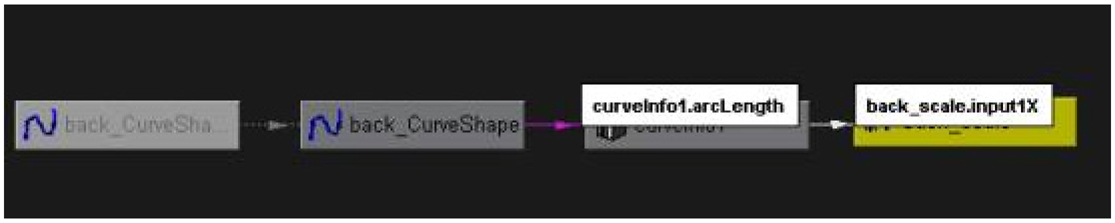
This adds a node called curveInfo1. This node contains the arclength of the curve (a aproximation of the true length of the curve).

Note: If you want to simplify the display of the hypergraph as shown below, you can use the ***Show > Show Selected Type*** *option with nodes that are selected*.

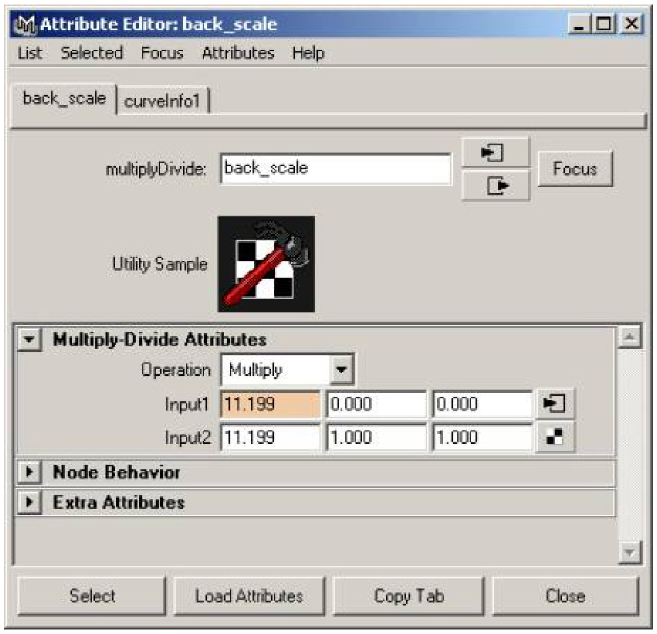
Copyright Alias|Wavefront, Jason Schleifer and Weta Digital LTD.



- Create a **multiplyDivide** node which will measure the length of the curve and output a scale value (the scale of the curve relative to it's initial length).
- Name the multiplyDivide node "**back_scale**".
- Connect the arcLength value of the curveInfo node and put it in the **input1X** of the **multiplyDivide** node.



- Take the current arcLength value of the curveInfo node and put it in the **input2X** of the **multiplyDivide** node.



multiplyDivide: back_scale

Multiply-Divide Attributes

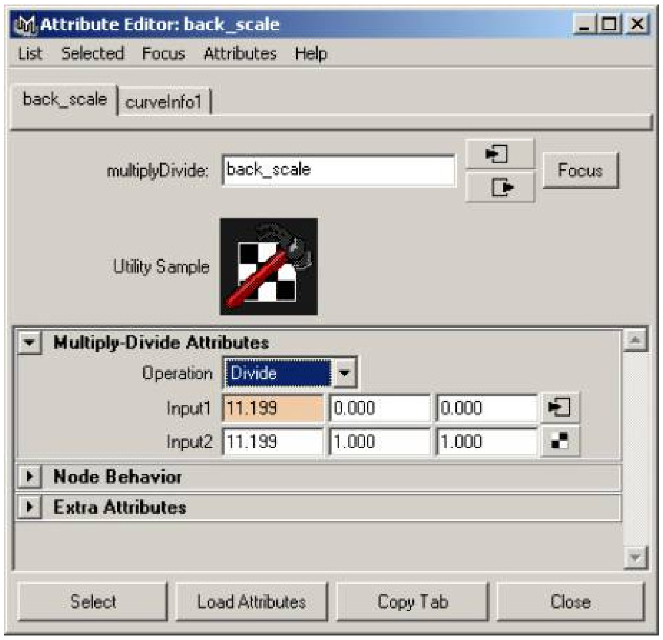
Operation: Multiply

Input1	11.199	0.000	0.000
Input2	11.199	1.000	1.000

Copyright Alias|Wavefront, Jason Schleifer and Weta Digital LTD.

- Set the Operation attribute to "**Divide**".

  This will give you the scale. When the two values are the same, the "scale value" will be 1. When the curve is half it's length, the scale will be .5. When it's twice it's length it will be 2. We're going to use this scale value to adjust the torso joint **translateY** attributes.


Attribute Editor: back_scale
List Selected Focus Attributes Help
back_scale | curveInfo1
multiplyDivide: back_scale
Utility Sample
Multiply-Divide Attributes
Operation: Divide

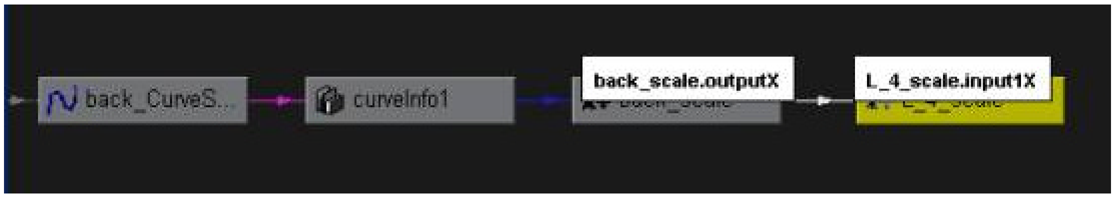
Input1	11.199	0.000	0.000
Input2	11.199	1.000	1.000

We start stretching from **L_4**, because you don't want to change the translateY on the first joint (it will cause the joint to go off the curve).

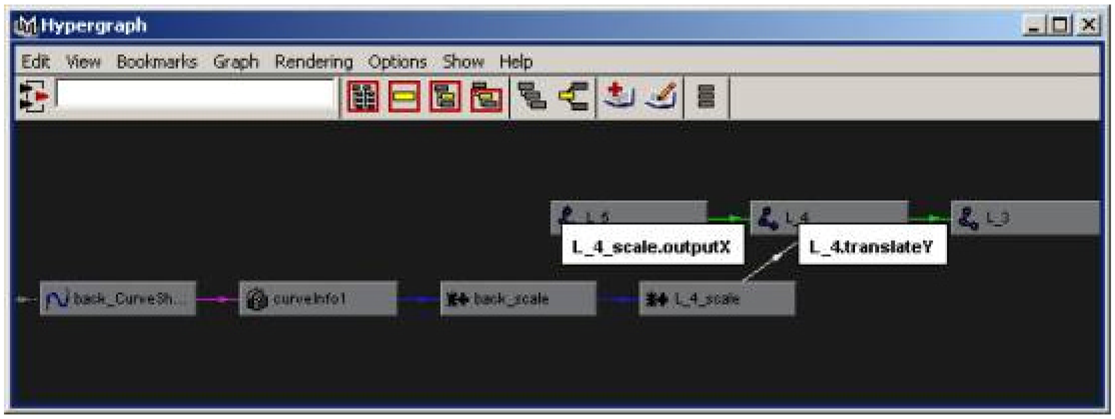
- Create another **multiplyDivide** node.

  This one will be used on **L_4** (call it **L_4_scale**).

- Copy the **L_4.ty** value and put it in **L_4_scale.input2X**.
- Connect the **back_scale.outputX** attribute to **L_4_scale.input1X**.



- Connect **L_4_scale.outputX** to **L_4.ty**;

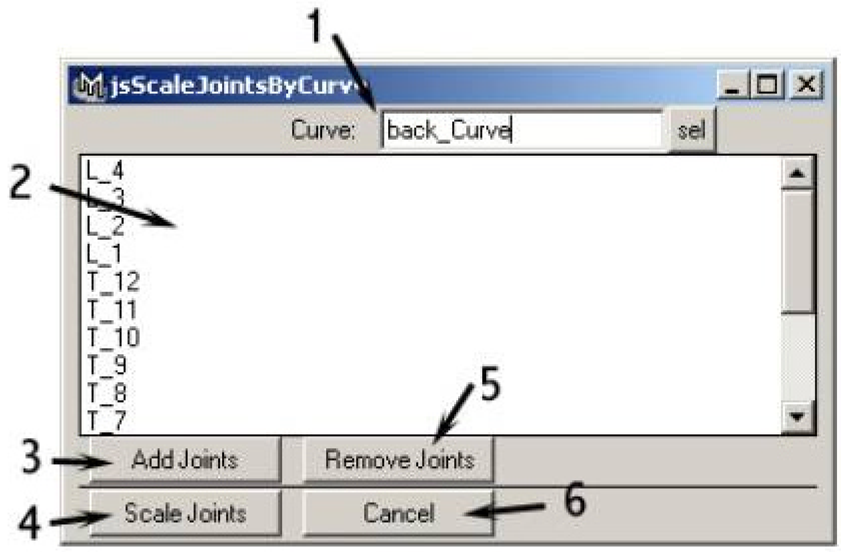


Now when you scale the curve, you should see L_4.ty changing.

Using jsScaleJointsByCurve to Automate Back Scaling

To make this process easier, you can use the jsScaleJointsByCurve script which will hook up the given chosen joints to the given curve.

- Bring up the **jsScaleJointsByCurve** interface by typing **jsScaleJointsByCurve** in the script editor.



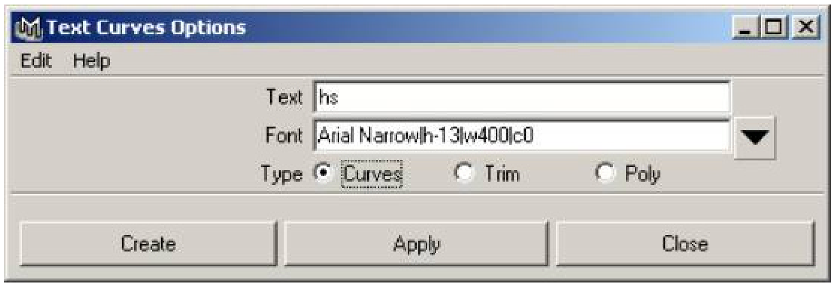
1. Curve to measure.
2. All the joints that are going to be scaled.
3. Add a joint to the list.
4. Apply the scale joint command.
5. Remove joints from the list of joints.
6. Close the interface.

- Select **back_curve** and click **sel** in the "curve" area of the **jsScaleJointsByCurve** interface.
- Select all the joints from **L_4** through **C_1**.
- Click **Add Joints** In the **jsScaleJointsByCurve** interface.
- Translate the clusters around to see the back scale.

Create hip and shoulder controls.

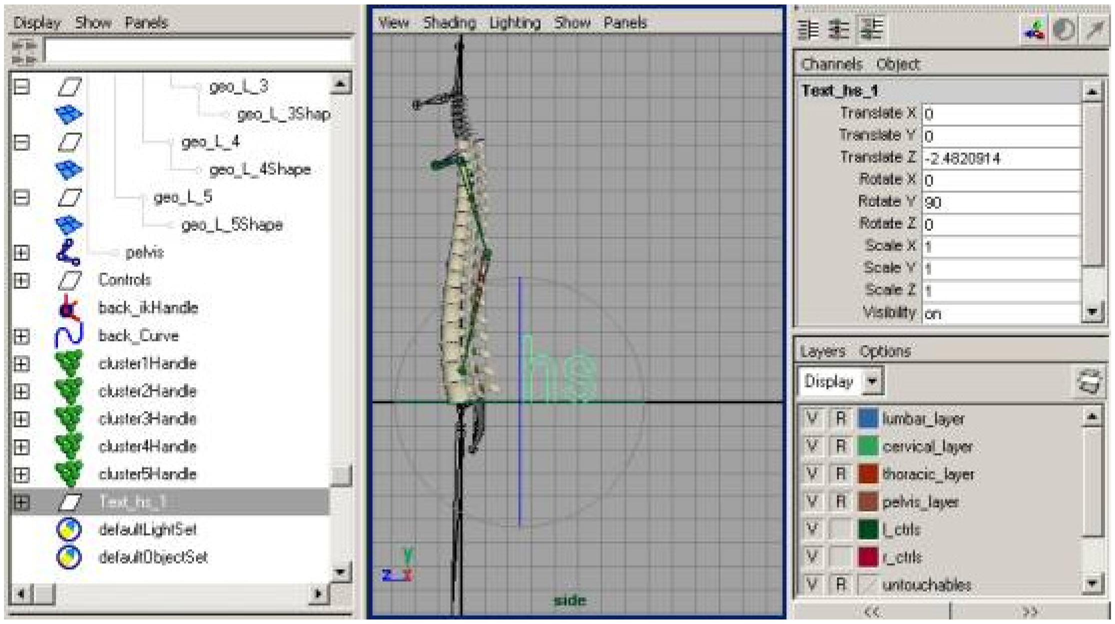
To control the back, we're going to use a hip and shoulder control setup.

- Create two text curves, one "**h**" for hip and one "**s**" for shoulder.



Text: hs
Font: Arial Narrow|h-13|w400|c0
Type: [✓ Curves] Trim Poly

- Rotate the controls **90** deg in **Y** so they face the correct direction.



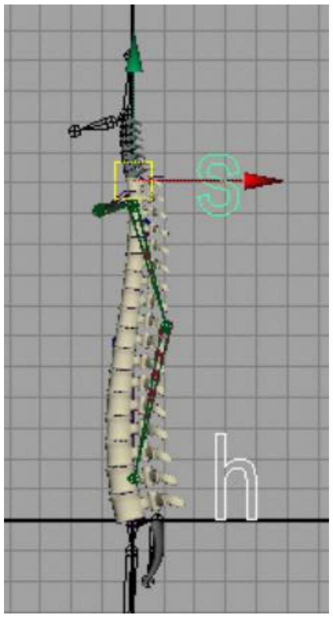
Text_hs_1

Translate X	0
Translate Y	0
Translate Z	-2.4820914
Rotate X	0
Rotate Y	90
Rotate Z	0
Scale X	1
Scale Y	1
Scale Z	1
Visibility	on

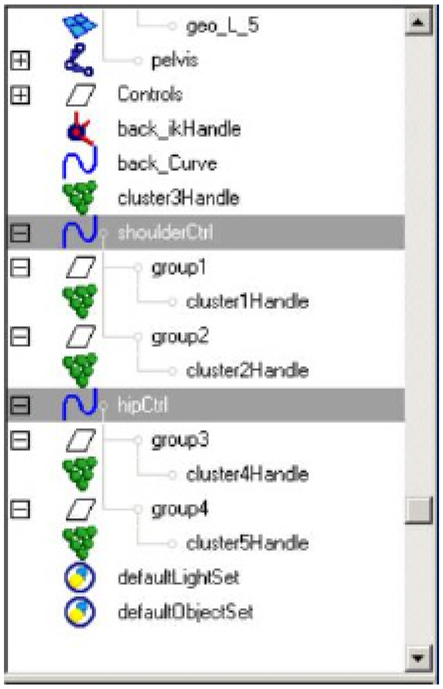
V	R	🟦	lumbar_layer
V	R	🟩	cervical_layer
V	R	🟥	thoracic_layer
V	R	🟫	pelvis_layer
V		🟩	l_ctrls
V		🟥	r_ctrls
V	R		untouchables

- Label the "**s**" "**shoulderCtrl**".
- Label the "**h**" "**hipCtrl**".
- Move the pivot for **shoulderCtrl** to **C_1**.
- Move the pivot for **hipCtrl** to **root**.

Copyright Alias|Wavefront, Jason Schleifer and Weta Digital LTD.



- Parent the top two clusters to the **shoulderCtrl**.
- Parent the bottom two custers to the **hipCtrl**.

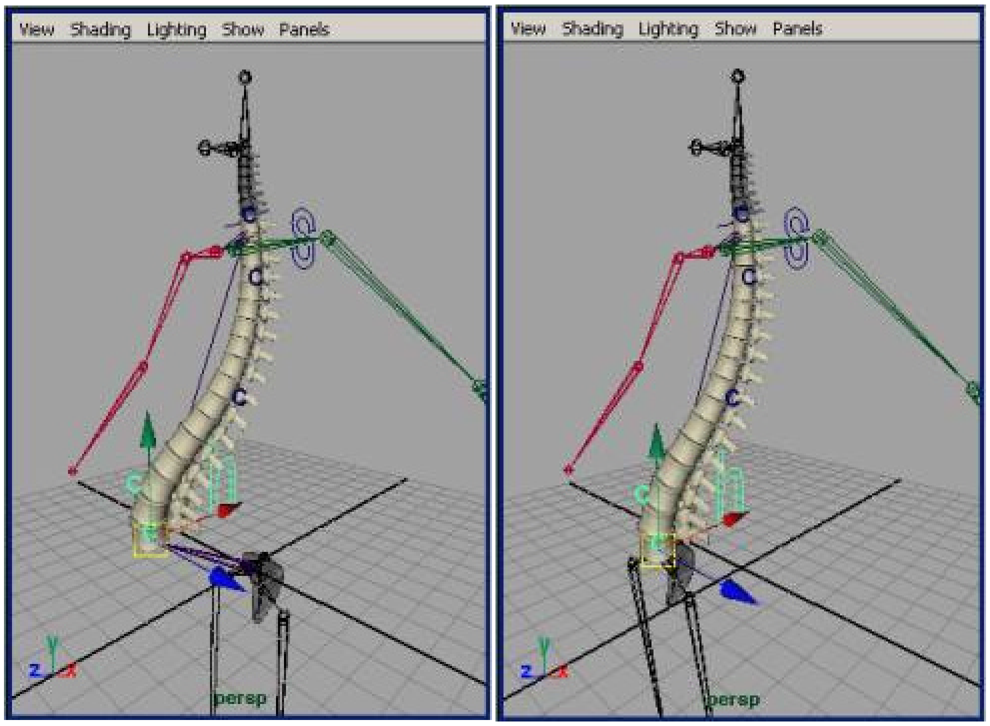


```
        geo_L_5
      pelvis
Controls
back_ikHandle
back_Curve
cluster3Handle
shoulderCtrl
    group1
        cluster1Handle
    group2
        cluster2Handle
hipCtrl
    group3
        cluster4Handle
    group4
        cluster5Handle
defaultLightSet
defaultObjectSet
```

Move the hip control and notice how the root doesn't move with it.

- Use **jsConstObj** to constrain the root to the hip control. (Select **hipCtrl**, select **root**, type **jsConstObj**. Or, you can point and orientConstrain the root to the **hipCtrl**).

Copyright Alias|Wavefront, Jason Schleifer and Weta Digital LTD.


View Shading Lighting Show Panels
persp
View Shading Lighting Show Panels
persp

Now when you move the top two controls, you'll notice the back moves around, aside from the middle cluster. This middle cluster is a stabalizing control used mainly to make the FK part of the control more stable.

Create Stabalizing and FK controls

The stabalizing cluster is going to be constrained between the shoulder and hip controls.

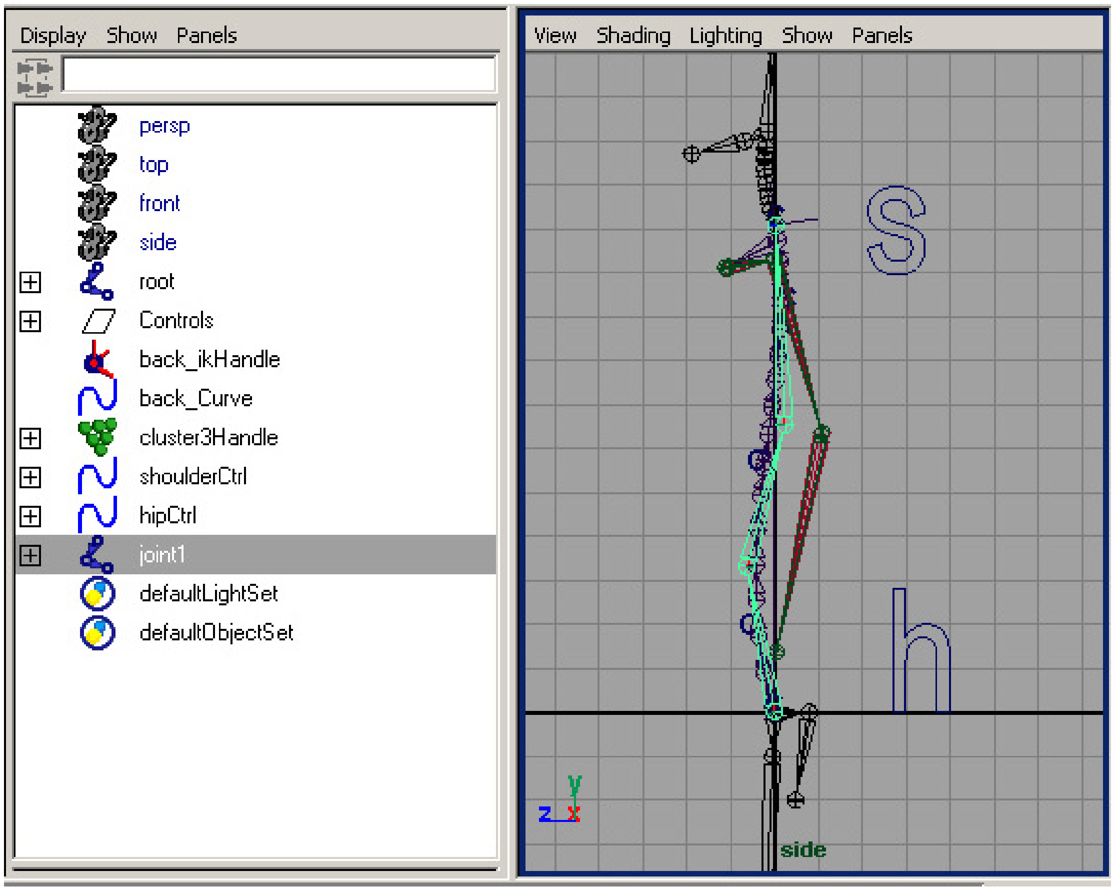
- Select the **hipControl** and the **shoulderControl** and the middle cluster.
- Use **jsConstObj** to constrain it to both of them.

> jsConstObj is a mel script which will automatically create a point and orient constraint for an object. It also ties the point and orient constraints together, and adds attributes to the object being constrained to allow for easier manipulation of the constraint weights.

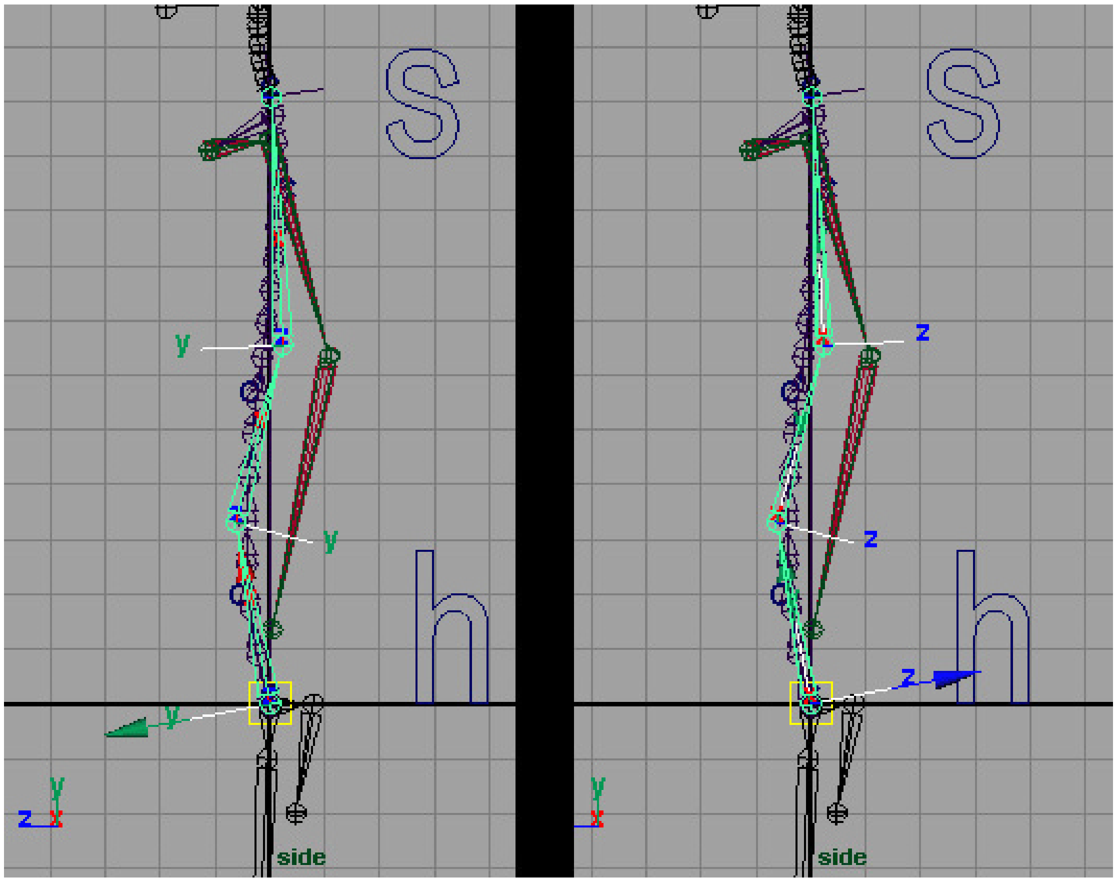
Now create the joints which we're going to use to animate the back.

- Create three skeleton segments, one which starts at the root and goes 1/3 of the way up the back, then another which goes 2/3 up, and the last which ends at the neck (C_1).

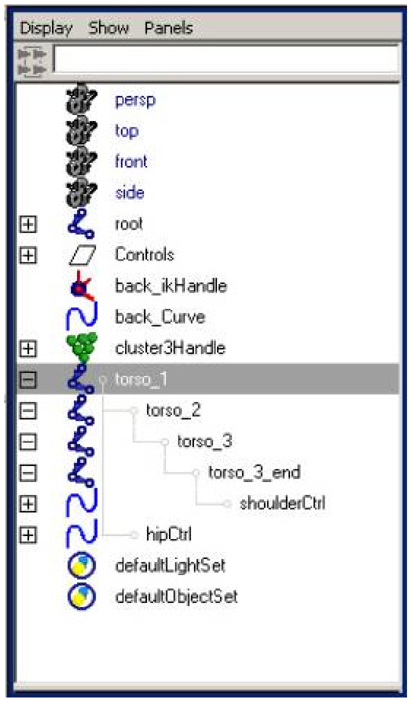
Copyright Alias|Wavefront, Jason Schleifer and Weta Digital LTD.



- Use **jsOrientJoint** to orient the joints correctly.



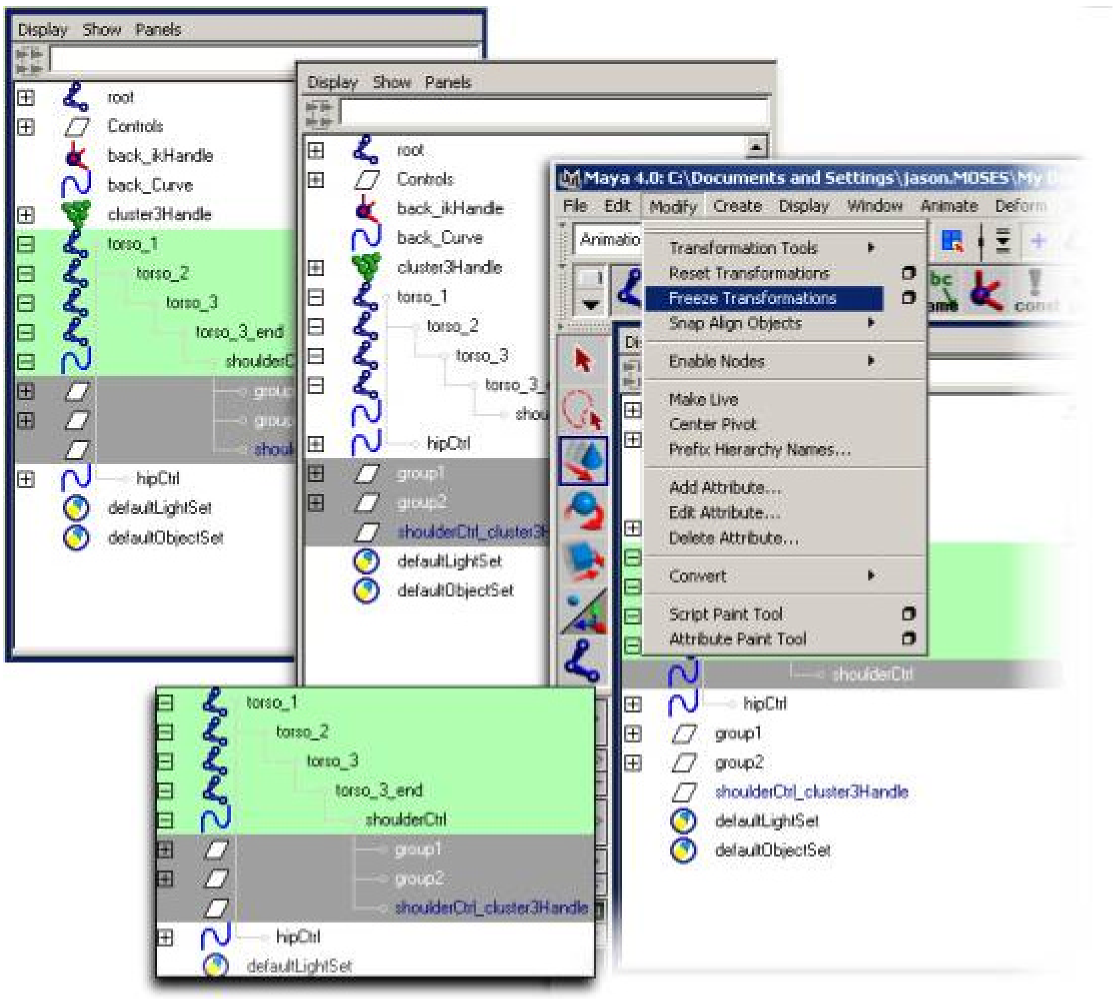
- Name the joints **torso_1, torso_2, torso_3,** and **torso_3_end**.
- Parent **shoulderCtrl** under **torso_3_end**.
- Parent **hipCtrl** under **torso_1.**



```
persp
top
front
side
root
Controls
back_ikHandle
back_Curve
cluster3Handle
torso_1
    torso_2
        torso_3
            torso_3_end
                shoulderCtrl
    hipCtrl
defaultLightSet
defaultObjectSet
```

Copyright Alias|Wavefront, Jason Schleifer and Weta Digital LTD.

- Make sure **hipCtrl** and **shoulderCtrls** are **0 0 0.**
    - For **shoulderCtrls**, unparent the children, select **shoulderCtrl**, perform Freeze Transformations, and re-parent the children.



For **hipCtrls** we need to do something a little bit different. If you perform a freeze transformations the same as done for the shoulder controls, then the orientation of the hip control won't be aligned correctly with the world.

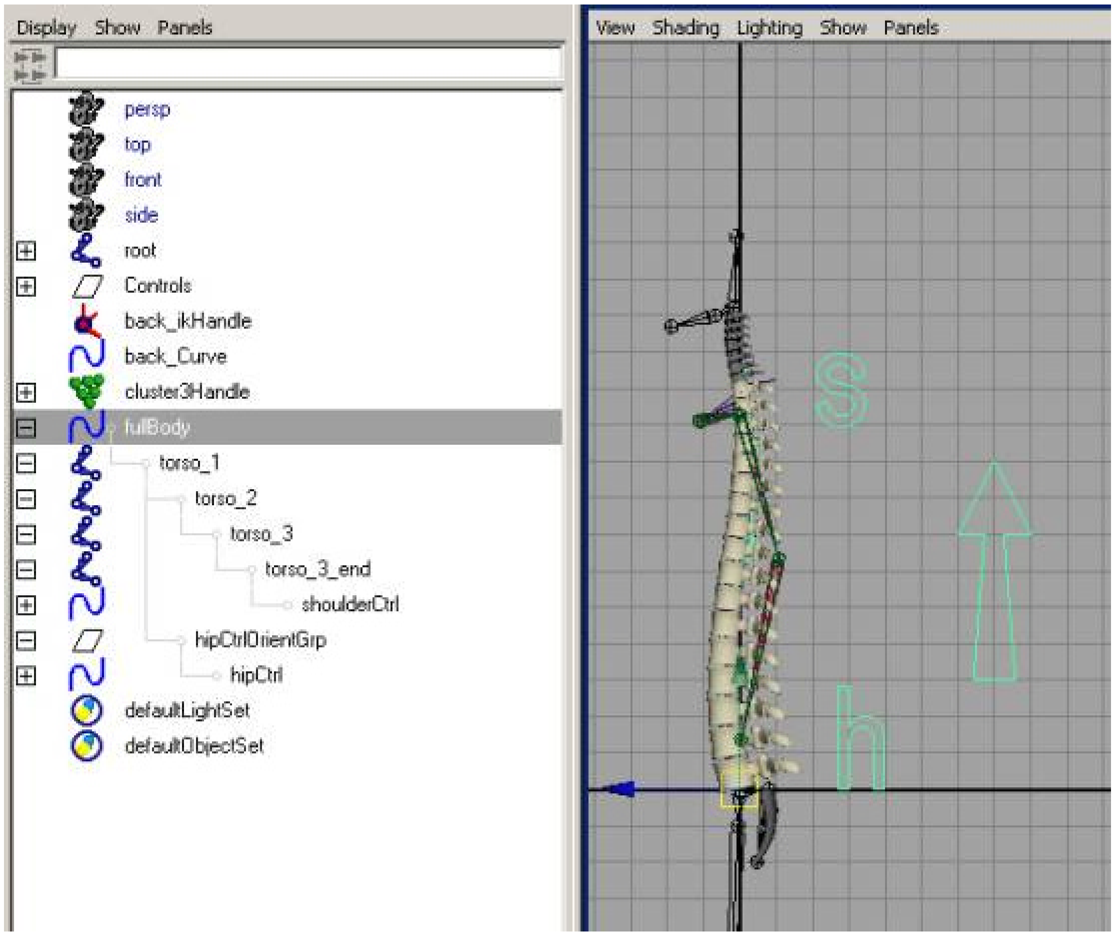
- To fix this, unparent the **hipCtrl** children as you did with the **shoulderCtrl** children.
- Group **hipCtrl** to itself to make a group above it, and rename that group **hipCtrlOrientGrp**.
- Measure the **jointOrientX** attribute of the **torso_1** joint and subtract that value from **180.**

This will give you the amount that you need to rotate hipCtrlOrientGrp to get the hipCtrl to align correctly. For example, use the following mel command:

setAttr hipCtrlOrientGrp.rx (180-`getAttr torso_1.jointOrientX`);

  - Now freeze transformations on hipCtrl, and re-parent the children.

- Create a "**fullBody**" control that can be used to animate the entire back, fk, ik, and all. This can be a curve which is easily recognizable.
- Parent **torso_1** to **fullBody**.

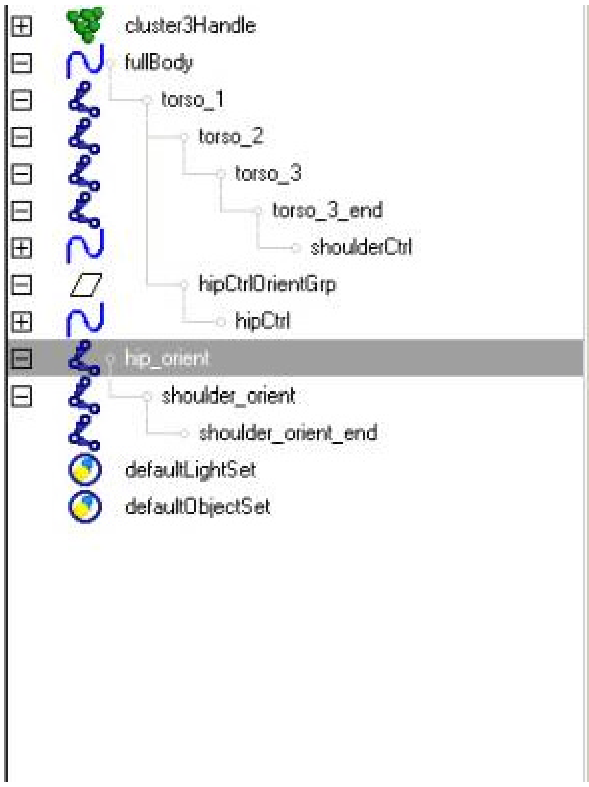


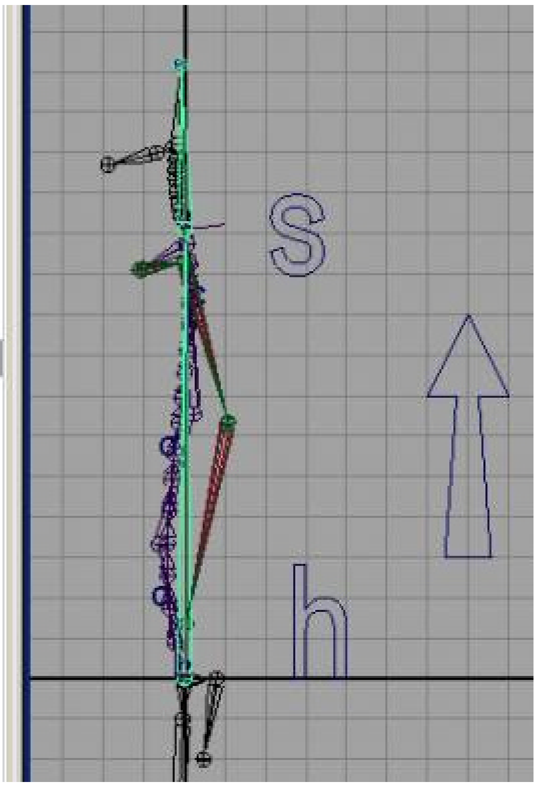
Add twist to the back

Currently the back will move around with the controls, but not when you try and twist it. This next step will create the control for twisting the back in any direction. Note: This technique will work with rotations up to 180 degrees. If the back rotates past it, it will flip. However, It's not physically possible to rotate your back past 180 degrees anyway, so this shouldn't be too much of an issue.

- Create a joint structure which goes from the root to the neck, and then from the neck to the top of the head.
- Name the joints "**hip_orient**", "**shoulder_orient**", and

"**shoulder_orient_end**".





- Use **jsOrientJoint** to orient the joints correctly. Set the rotation order on all of them to **xzy**.

  We're going to use the y rotation of "shoulder_orient" to drive the twist value of the ikHandle.

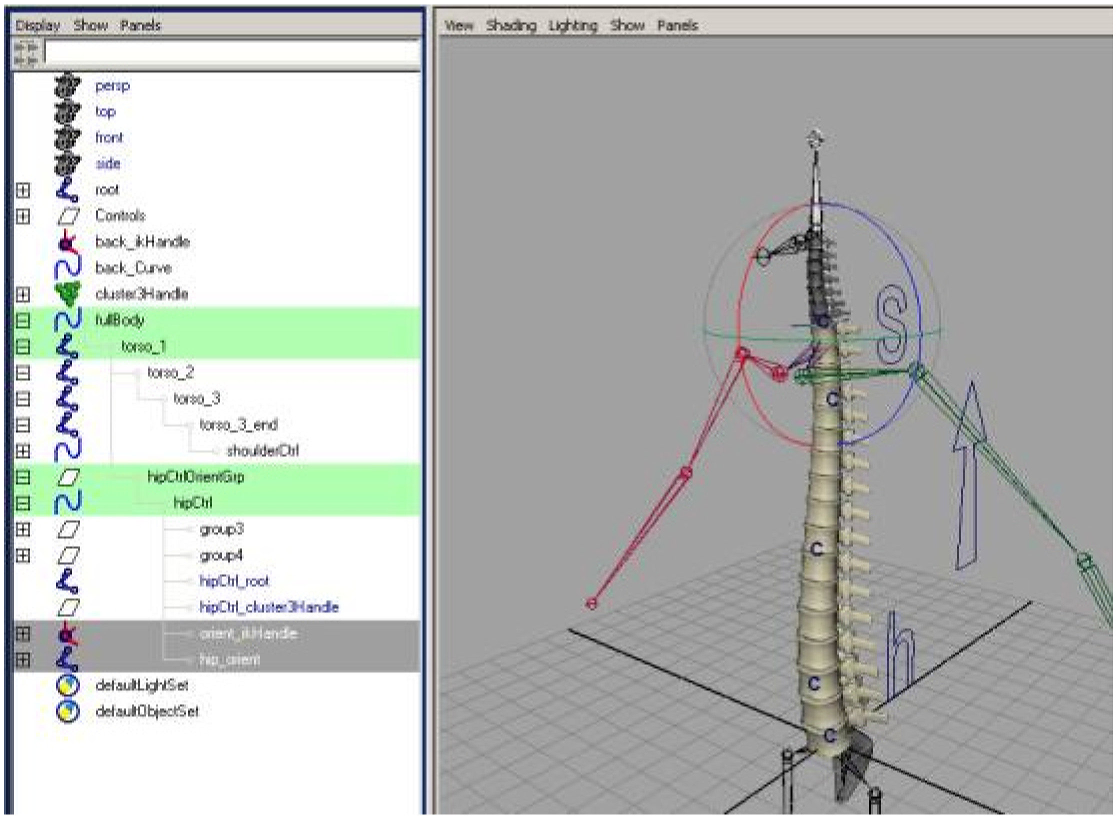
- Create a **ikSCsolver** ikHandle from "**hip_orient**" to "**shoulder_orient**", name it **orient_ikHandle**.
- OrientConstrain **shoulder_orient** to **shoulderCtrl** (use **jsMakeOrient** and delete the pointConstraint).
- Make an expression on back_ikHandle that connects **shoulder_orient.ry** to **back_ikHandle.twist**

  (Note: you MUST do an expression for this. A direct connection breaks it).

  back_ikHandle.twist = shoulder_orient.ry;

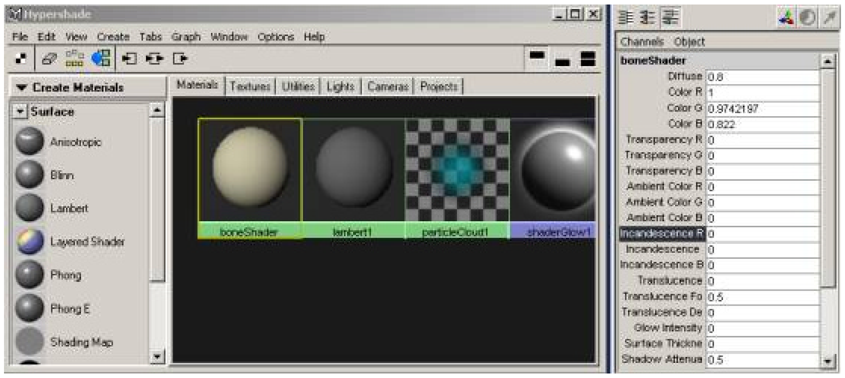
- pointConstrain the **orient_ikHandle** to **shoulderCtrl**.
- parent the **orient_ikHandle** and **hip_orient** to **hipCtrl**.

Copyright Alias|Wavefront, Jason Schleifer and Weta Digital LTD.



Add Stretch Warning Color

While this has lots of control to move things around, it's good to warn the animator if they're taking things too far. You can do this using a "stretch" warning color.



boneShader	
Diffuse	0.8
Color R	1
Color G	0.9742197
Color B	0.822
Transparency R	0
Transparency G	0
Transparency B	0
Ambient Color R	0
Ambient Color G	0
Ambient Color B	0
Incandescence R	0
Incandescence	0
Incandescence B	0
Translucence	0
Translucence Fo	0.5
Translucence De	0
Glow Intensity	0
Surface Thickne	0
Shadow Attenua	0.5

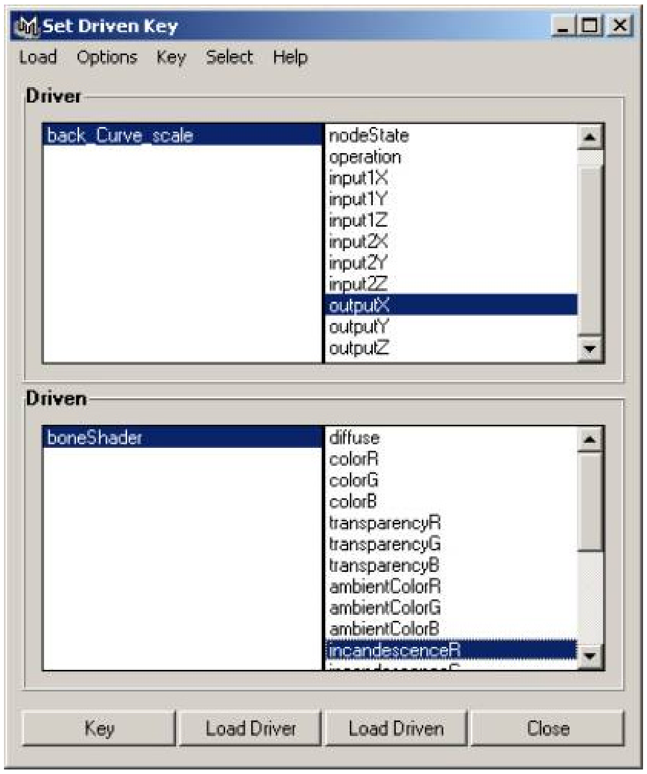
- Bring up the hypershade and select the **boneShader**.

  We're going to use the incandescenceR channel to let the animator know if they're stretching things too far.

- Select the **incandescenceR** channel in the channel box and click the **RMB** to bring up the setDrivenKey dialogue.

Copyright Alias|Wavefront, Jason Schleifer and Weta Digital LTD.

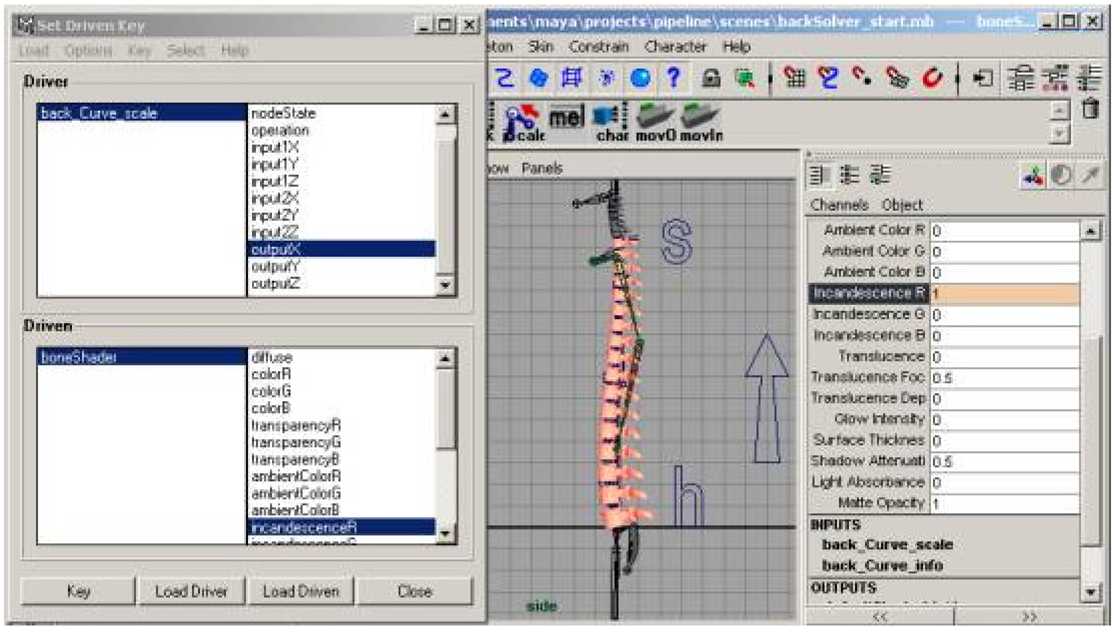
- Select the **backCurve** and bring up the hypergraph.
- Graph the dependency graph for the curveShape. Pick the back_curve_scale and put it in the drivers column of the setDrivenKey window.



outputX is the item which is driving the scale. We'll use it to drive the incandescenceR of the shader.

- Select those channels, and hit **Key**.
- Move **shoulderCtrl** up a bit until the back is stretched as far as you want to allow it.
- Select the shader in the setDrivenKey window and in the channel box change **incandescenceR** to **1.**

Copyright Alias|Wavefront, Jason Schleifer and Weta Digital LTD.



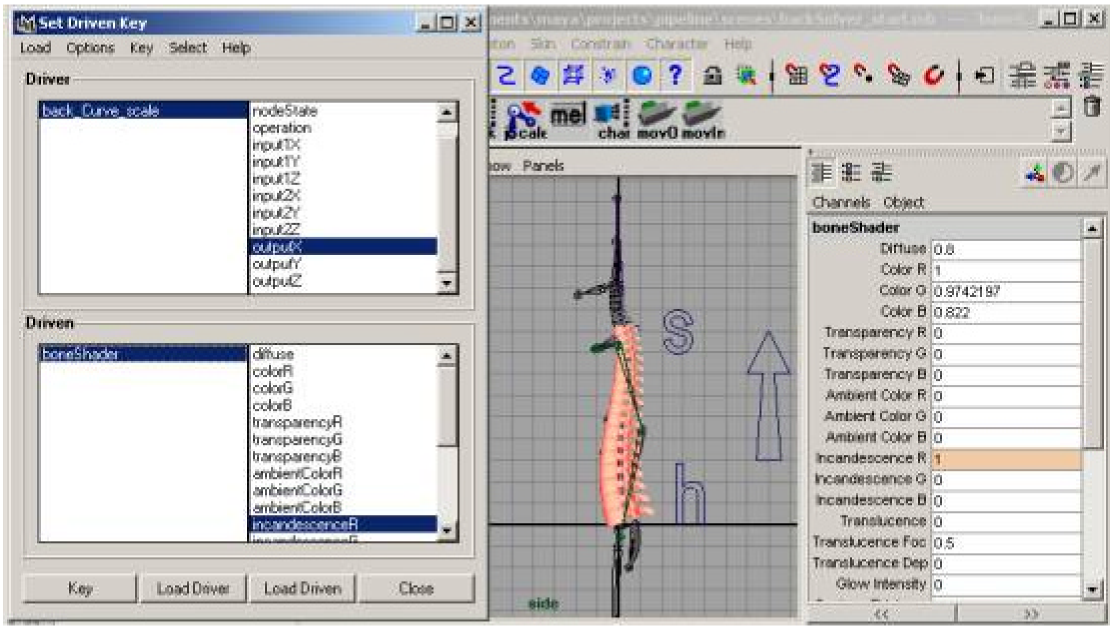
Driver: back_Curve_scale — nodeState, operation, input1X, input1Y, input1Z, input2X, input2Y, input2Z, outputX, outputY, outputZ

Driven: boneShader — diffuse, colorR, colorG, colorB, transparencyR, transparencyG, transparencyB, ambientColorR, ambientColorG, ambientColorB, incandescenceR

Ambient Color R	0
Ambient Color G	0
Ambient Color B	0
Incandescence R	1
Incandescence G	0
Incandescence B	0
Translucence	0
Translucence Foc	0.5
Translucence Dep	0
Glow Intensity	0
Surface Thicknes	0
Shadow Attenuati	0.5
Light Absorbance	0
Matte Opacity	1

INPUTS
back_Curve_scale
back_Curve_info
OUTPUTS

- Press **Key**.
- Move the control down so the back is compressed as much as you want it to be.
- Change the **incandescenceR** to **1**



Driver: back_Curve_scale — nodeState, operation, input1X, input1Y, input1Z, input2X, input2Y, input2Z, outputX, outputY, outputZ

Driven: boneShader — diffuse, colorR, colorG, colorB, transparencyR, transparencyG, transparencyB, ambientColorR, ambientColorG, ambientColorB, incandescenceR

boneShader

Diffuse	0.8
Color R	1
Color G	0.9742197
Color B	0.822
Transparency R	0
Transparency G	0
Transparency B	0
Ambient Color R	0
Ambient Color G	0
Ambient Color B	0
Incandescence R	1
Incandescence G	0
Incandescence B	0
Translucence	0
Translucence Foc	0.5
Translucence Dep	0
Glow Intensity	0

- Press **Key**.
- Now alter the curves in the graph editor to give you just as much stretching as you want to allow.

Clean everything up for the animator

- Assign all joints and controls which you don't want the animator to have access to a display layer called "untouchables". These will be in reference mode.

- Assign the torso controls to a displayLayer called "**torsoCtrls**". Color it in a way that would make it easy to see.
- Assign the back cuve controls to a displayLayer called **backCtrls**;
- Turn on display handles for the joint-based torso controls.
- Check all rotation orders so they behave how you'd expect. (e.g. the ik controls should be **xzy** so the body can rotate around in any direction).

Copyright Alias|Wavefront, Jason Schleifer and Weta Digital LTD.

VIII. Procedural Animation Rig

What is a "procedural animation rig"?

The idea behind the procedural animation rig is to provide automated and consistent ways of creating and updating rigs. If the directory structure for genering rigs is consistent, and you have mel scripts written to build controls for the character, it's possible to make scripts which will build an animation rig from the ground up in a matter of seconds.

Creating a procedural animation rig

Here are the basic steps one could take in building a rig for their character:

- Import the skeleton
- Import the geometry and parent it to the appropriate joints.
- Define the "creature"
- Add arm controls
- Add leg controls
- Add back control
- Add head control
- Save File.

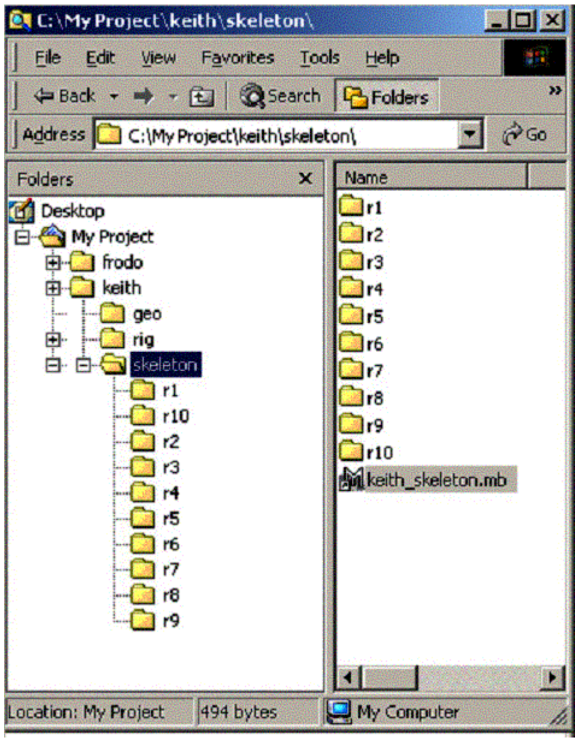
These steps can easily be broken down into mel scripts and commands if the locations for the files and the naming scheme for geometry and controls is consistent.

For example, if you have previously defined in your environment the location of all skeleton files as a global string **$gSkelLoc**, all geometry files as **$gGeoLoc**, and the location to save the rigs as **$gRigLoc**, all you need to do is determine which creature you're building the rig for and the script can easily import the skeleton and geometry, combine them, and then save the file.

```
// get the creature
string $creature = "keith";

// define the global variables
global string $gBaseProject;
global string $gSkelLoc;
global string $gGeoLoc;
global string $gRigLoc;

$gBaseProject = "C:/myProject/";
$gSkelLoc = ($gBaseProject + $creature + "/skeleton/");
$gGeoLoc = ($gBaseProject + $creature + "/geo/");
$gRigLoc = ($gBaseProject + $creature + "/rig/");
```

Copyright Alias|Wavefront, Jason Schleifer and Weta Digital LTD.

Once you've defined where the files will sit, it's up to you how you locate them. The easiest thing to do is to have the file always named the same, and linked to the latest revision. For example, if you have 10 versions of the skeleton, you should have 10 directories in the skeleton directory called "r#", and one file in the directory called creature_skeleton.mb which is linked to the most recent version. That way it's easy to find the most recent version, for humans and for the script.



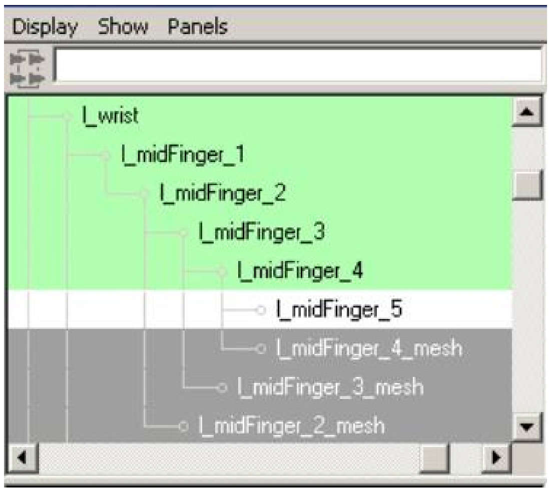
If the pipeline is set up correctly, you can just assume that the files are going to be named as you're expecting and not need to perform a search for it. But it's always good to check. To do that you can use the fileTest command:

```
// IMPORT THE SKELETON
$skeleton = ($creature + "_skeleton.mb");

// check and see if the file exists and is readable
If (!`fileTest -r ($gSkelLoc + $skeleton)`)
{

    // it doesn't exist! Error out.
    error ($skeleton + " in " + $gSkelLoc + " is not readable.\n");

}
else
{

    // it does exist.. import it.
```

Copyright Alias|Wavefront, Jason Schleifer and Weta Digital LTD.

```
        file -import ($gSkelLoc + $skeleton);

}
```

Next we import the geometry using the same technique as above. Again, if the geometry is created of individual pieces which have the same names as the joints it makes it extremely easy to place them correctly on the skeleton. In the case below, all the geometry is named "joint_mesh". So the geometry for the root joint would be "root_mesh", and the geometry for the fingers would be "finger_mesh".



Then if your geometry is named correctly you can run a script like the following to parent it under the joints:

```
// PARENT THE GEOMETRY UNDER THE SKELETON

// find all the geometry
string $geos[0];
string $geo;
$geos = `ls -type transform ("*_mesh")`;

// for each bit of geometry, find the appropriate joint name and parent it.
for ($geo in $geos)
{

        // use the substitute command to find the correct joint.
        string $jointName;
        $jointName = `substitute "_mesh" $geo ""`;

        // check and make sure the joint exists
        if (!`objExists $jointName`)
        {

                warning ($jointName + " doesn't exist.. skipping..\n");
        }
        else
        {

                // parent the geo under the joint
                parent $geo $jointName;

        }

}
```

Now that all the geometry is parented correctly, you can define the creature using the methods shown in the previous lesson. The steps are as follows:

- Create a locator
- Add a "Creature" attribute to the locator.
- Add a multi attribute to connect the joints you want.
- Connect the joints/controls necessary to the attribute.

Once the creature is defined you have a good low-resolution version of your creature. It can be used for lighting, quick playblasts, rotoscoping, etc. It doesn't have all the nifty controls your final animation rig will have, but it does have the joints which will be driving the skinned version, so it can take animation just as the muscle model can take animation. It's a good idea to save this version of the file as a low-resolution lighting rig.

Now we start to add the controls. You can easily use mel commands to perform this, and if you have a consistant setup for how your creatures are generated, they can all take the same scripts. For example, if you have twenty bipedal characters in your scene, you only have to write the script for generating arm controls once, then you can apply it to each of your creatures.

After the controls are added, simply save the file as the most recent version (make a new revision number, and save the file in that directory, then make a link to it in the $gRigLoc directory).

Updating an animation rig

Creating the rig is only first step in having a procedural system. While it's nice to be able to generate rigs by executing only a few commands, it's in updating the rigs that using mel becomes extremely handy.

If you have 12 animators all working on similar characters, and you need to make an update as to how some of the creatures are working, instead of going to each animator's computer and manually updating the scenes, it's much more time-effective to write a mel script to do it.

The trick to doing this is that joints and structures may not be named the same

Copyright Alias|Wavefront, Jason Schleifer and Weta Digital LTD.

as they were when you created the character (due to multiple creatures in the scene). So a script which would work with one character in the scene:

```
ikHandle -sj l_up_arm -ee -l_wrist;
```

suddenly doesn't work anymore because now there are two creatures in the scene, and maya doesn't know which l_up_arm you're talking about.

The fix for this is to know what creature it is you're wanting to update.

You can use the **ls** command with wildcards to find out all the instances of **l_up_arm**:

```
string $l_up_arms[0];
$l_up_arms = `ls -long "*l_up_arm"`;
```

This will return the entire path of all the "l_up_arm"'s in the scene.

Next we get a full path for the creature that we want in the scene. We can use the **findCreature** script created earlier to do this.

```
string $creatureFullPath = `findCreature "keith"`;
```

This will return the full path for the "keith" creature.

Note: If your production is going to have multiple creatures of the same type in the scene, you can modify the script to perform whatever update your making to each creature.

Once we have the full path for that creature, we can determine the top node in the hierarchy. For example, if the keith creature's fulll path is:

"|topNode|runAround|keith"

we want to grab "topNode" from this. You can do that using the tokenize command:

```
string $breakItUp[0];
tokenize ($creatureFullPath, "|", $breakItUp);
string $topNode = $breakItUp[0];
```

Note: The best thing to do here would be to define the above commands as a procedure called "findTopNode" which takes a string and returns the top node.

Now that we have the top node saved as $topNode, we can iterate through the $l_up_arms and find the one which has the same top node.

Copyright Alias|Wavefront, Jason Schleifer and Weta Digital LTD.

```
string $node;
string $l_up_arm; // this will be what we save l_up_arm as
for ($node in $l_up_arms)
{

        string $breakItUp[0];
        tokenize ($node, "|", $breakItUp);

        // check and see if $breakItUp[0] is the same as $topNode
        if ($breakItUp[0] == $topNode)
        {

                // we have a match! Put $l_up_arm as $node
                $l_up_arm = $node;

        }

}
```

Now we've found the correct $l_up_arm that we were looking for.

If you wrap all those commands into a single script called something like "**findSpecificNode**" and took two arguments, the name of the node you're looking for and the creature you looking for it under, you could then write something which would look like this:

```
$l_up_arm = `findSpecificNode "l_up_arm" "keith"`;
$l_wrist = `findSpecificNode "l_wrist" "keith"`;

ikHandle -sj $l_up_arm -ee $l_wrist;
```

While this may seem like a pretty simple example, you can extrapolate this idea in to replacing anything on a creature. Not only replacing or adding controls, but modifying the current controls, copying animation from one control to another, even performing a "creature check" to make sure the animators haven't accidentally deleted anything.

By making sure you've build your rig in a consistant and procedural manner, you're providing yourself with a flexible method of generating and updating your creatures.

Copyright Alias|Wavefront, Jason Schleifer and Weta Digital LTD.