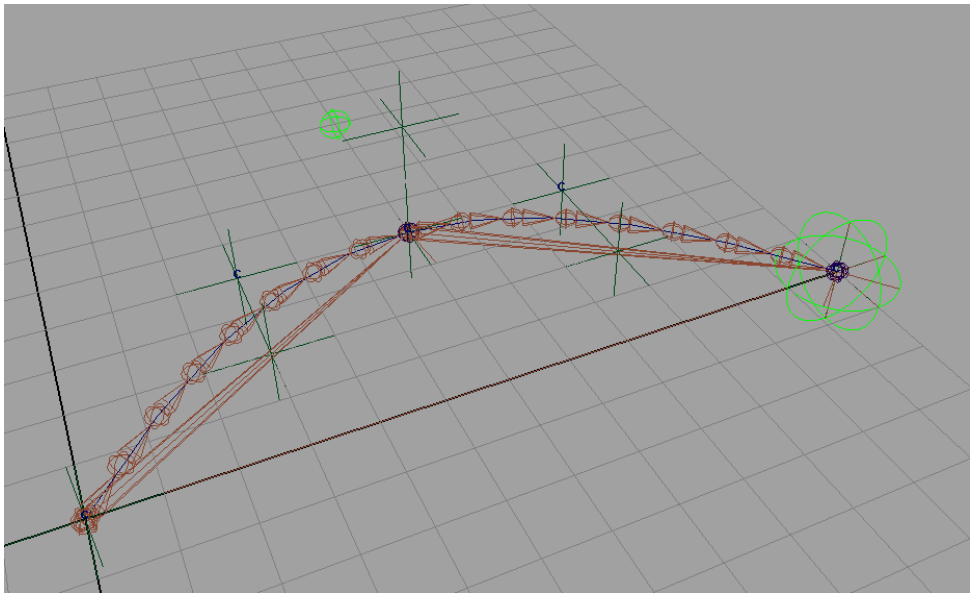
Creating a Stretchy, Bendy Arm Setup in Maya

Written by Duncan Skertchly
duncanskertchly@yahoo.co.uk

Introduction

In this document I will present a method for achieving an arm rig in Maya that has the ability to bend and stretch, creating a similar effect to early animated cartoons where characters had “pipe cleaner” arms. This setup is ideally suited for rigging cartoony characters, however the setup could be quite easily used / adapted to other rigging tasks (creating a robots telescopic legs / arms for instance).



The rigging process involves working with splineIk, node networks and expressions (including a bit of simple vector maths).

This is the first tutorial type document I’ve written (outside of work anyway). Hopefully it’s clear and concise and easy to follow. But if you have trouble with any part of it for whatever reason please get in touch via the email address above and I’ll try and help out. Also if you come up with any additions / improvements please let me know about them.

Script Installation

Before you start there is a script provided with this document that you will need to install. It’s a small script that creates a joint chain made of a specified number of joints between two selected objects. You can install this script by copying it to your usual Maya script path, something like.

C:\Documents and Settings\User\My Documents\maya\7.0\scripts

Once you've installed it, restart Maya and you're ready to go. How to use this script is detailed later in the doc. That's it really. Let's get going.

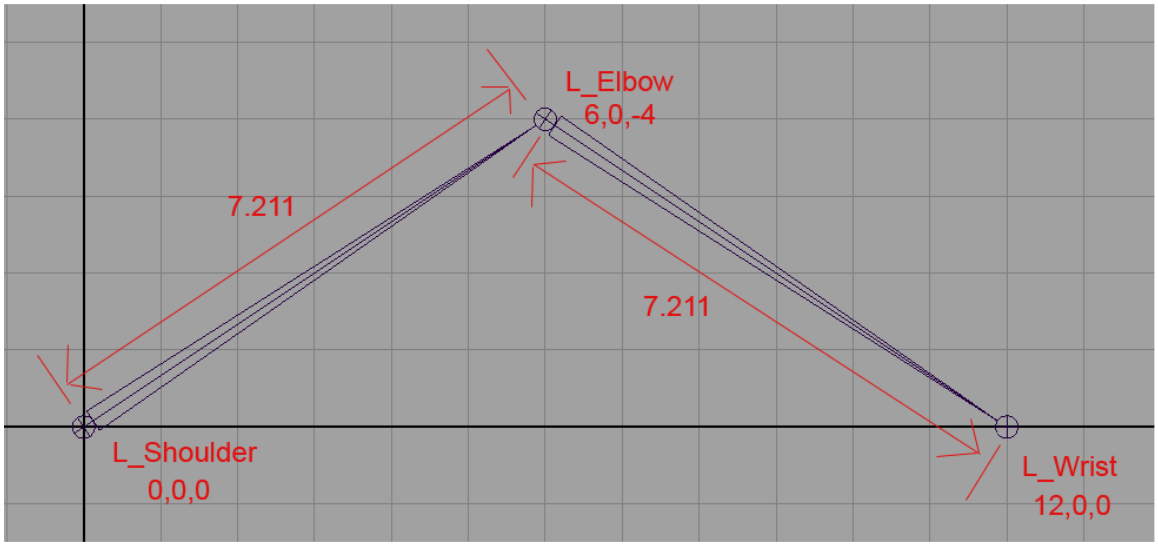
First Steps

Creating the Main Joints

First, in the top viewport create a normal 3 joint arm setup. Use **Grid Snapping** to draw the joints with equal length.

Before you draw them set your joints to **XYZ** orientation. You can use other orientations; however this may change some of the small details later on in the document. Once you've learned how the setup works you can go back and re-create the rig using your preferred orientation. The only orientation to always avoid is the **None** option. Set the **Second Axis World Orientation** to **+Y**.

Note that for this tutorial it is important that you make the upper and lower arm joints equal length as the rest of this tutorial assumes that you have. You can use this setup with arms that don't have equal length; however there is a very slight difference in the rig. Instead of complicating the main body of this document I have put an explanation of this alteration in the notes section. So once you've run through it once, have a look at them.



Rename the joints **L_Shoulder**, **L_Elbow** and **L_Wrist**.

Next we create a chain of joints that will eventually become our "Bendy" joints, controlled by splineIk. We'll just create the joint chains for the moment and leave the other stuff until later.

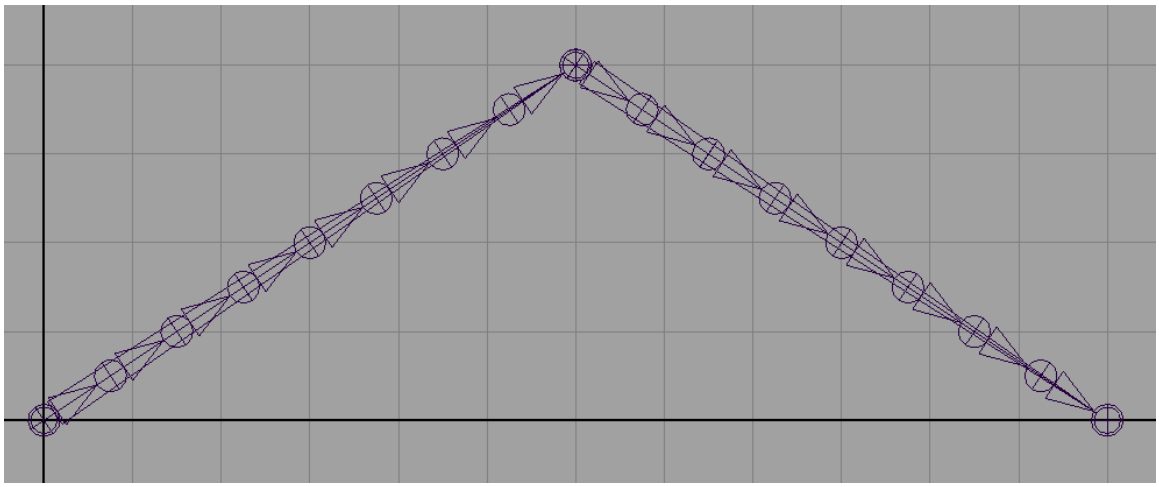
Creating the Bendy Joints

Select the **L_Shoulder** joint and then shift select **L_Elbow**. If you installed the script correctly type this line in to Maya's command line or script editor.

```
dsCreateInterpolatedJoints(9);
```

This will create a new chain of 9 joints between **L_Shoulder** and **L_Elbow**. If you use this script it's important to pick an odd number of joints for the new chain. That way we get a nice joint exactly half way between our main joints.

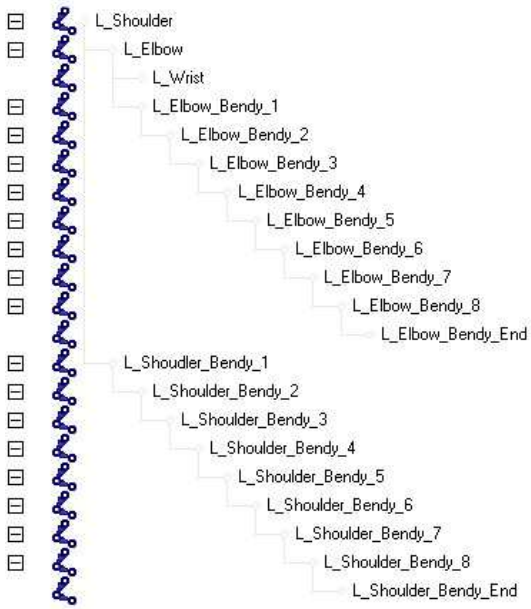
Run the script again. But this time select **L_Elbow** and **L_Wrist**. You should have something similar to the image below.



Note that it is very important that you create two separate chains, one running from the shoulder position to the elbow position, the other running from the elbow position to the wrist position. This method will not work correctly using one long chain.

Rename the new chains. Call them **L_Shoulder_Bendy_1**.. **L_Shoulder_Bendy_2**.. etc. The last joint in the chain I call **L_Shoulder_Bendy_End**. Do the same for the elbow chain, but change the names accordingly.

Now parent the root joint of the **L_Shoulder_Bendy** chain to the main **L_Shoulder** joint (select **L_Shoulder_1**, shift select **L_Shoudler** and hit *P*). Parent **L_Elbow_Bendy_1** underneath **L_Elbow**. If all is going to plan you should have a hierarchy that looks something like this.



Now we have all the joints that we need we're ready to create the stretch effect on the main part of the arm.

Adding the Stretchy Setup

Preparation

Create a standard **ikRP** solver between **L_Shoulder** and **L_Wrist**. Once you've created it rename it to **Main_ikHandle**.

Create a control that we can use to move the ikHandle and have somewhere to put all the custom attributes we'll eventually be adding to the rig. I often use an implicit sphere for this. You can create one by typing the line below in to the script editor or command line, or you can use whatever you prefer (nurbs shape etc).

```
createNode "implicitSphere";
```

Once you've created your control rename it **Wrist_Ctrl**. Move it to the position of the wrist joint using point snapping and then point constrain the ikHandle to it (select **Wrist_Ctrl**, shift select the **Main_ikHandle** and go *Constrain>>Point*). Now when you move the control around the bones should move with it.

This setup involves creating quite a few locators. These are the first ones we'll need.

Create three locators. Using point snapping, place one at **L_Shoulder**, one at **L_Elbow** and one at **L_Wrist**. Scale them down to a sensible size if you need to. Rename them **L_ShoulderLoc**, **L_ElbowLoc** and **L_WristLoc**.

Point constrain **L_ShoulderLoc** to the **L_Shoulder** joint (select **L_Shoulder**, shift select **L_ShoulderLoc**, *Constrain >> Point*).

Point constrain **L_ElbowLoc** to the **L_Elbow** joint.

Point constrain **L_WristLoc** to the **L_Wrist** joint.

Now we're ready to create the stretchy Ik arm.

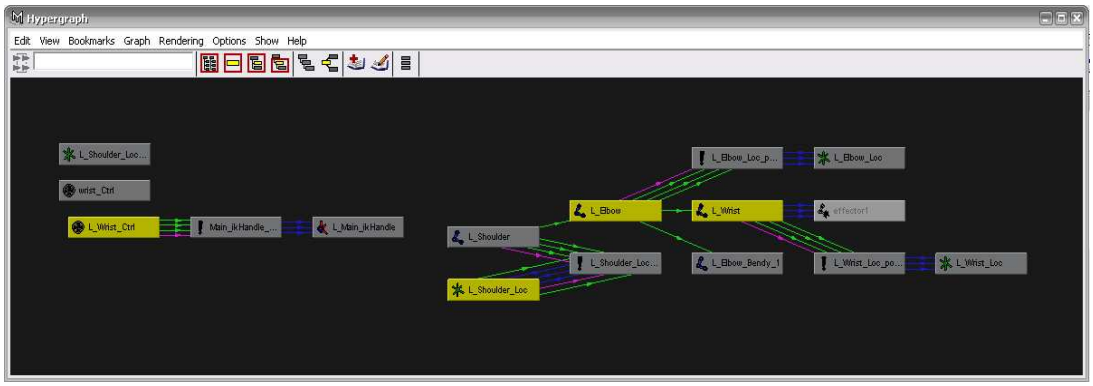
Making It Stretchy

First create a Distance Tool. *Create >> Mesaure Tools >> Distance Tool*. Click in the viewport in two places somewhere below the arm to create the tool. This should create

two locators with a distance measurement in between them. Next move the first locator to the shoulder position and the second locator to the elbow position (use point snapping) Make a note of the distance displayed (two decimal places should be fine)
Multiply this number by two (the lower part of the arm should be the same length so we don't need to do another measurement). Keep a note of the result. Delete the distance tool, we won't need it anymore.

Now we need to add an attribute to the **L_Wrist_Ctrl** that we can use to switch the stretchiness on and off. Select **L_Wrist_Ctrl** and go *Modify >> Add Attribute*. Set the attribute to **Boolean** and name it **Stretchy**.

Now we're going to create a small node network that will deal with controlling the stretch effect. In the viewport shift select **L_Wrist_Ctrl**, **L_ShoulderLoc**, **L_Elbow** and **L_Wrist** and open up the hypergraph and go *Graph >> Input and Output connections*. This should display everything we need to create the stretchy IK arm.

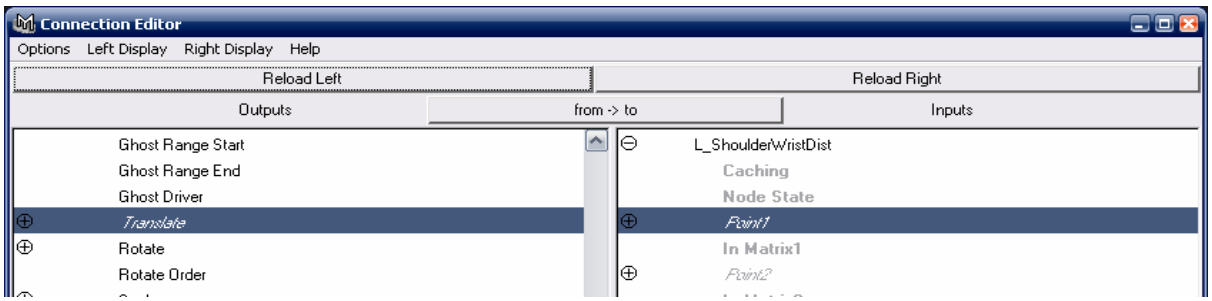


Create a **distanceBetween** node. *Rendering >> Create Node >> Utilities Tab >> Distance Between*.

We'll use this to measure the distance between **L_ShoulderLoc** and **L_Wrist_Ctrl**. The node should appear somewhere in the window. Right click to re-name it. Call it **L_ShoulderWristDist**. Position it somewhere convenient in the window.

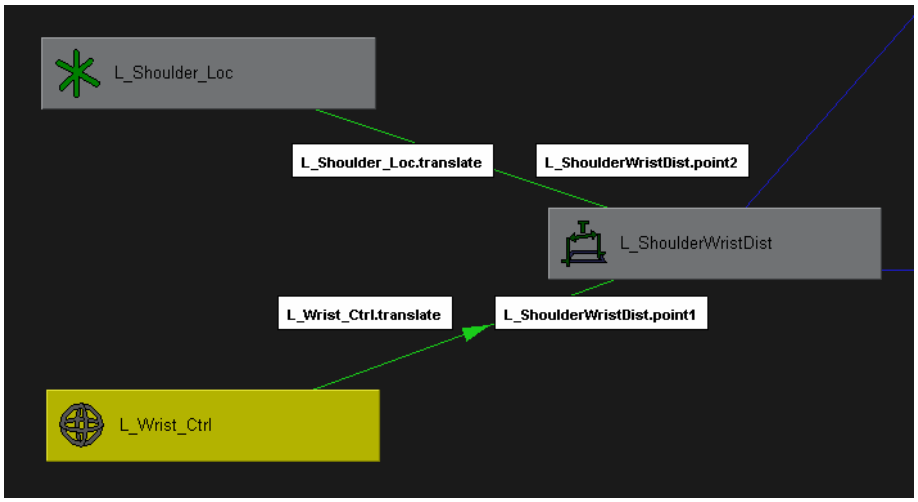
Connect **L_Wrist_Ctrl.translate** to **L_ShoulderWristDist**.**Point1** attribute.

Do this by right clicking on the edge of the **L_Wrist_Ctrl** node, and selecting *Translate >> Translate*, then dragging over to the **L_ShoulderWristDist** node, right click on it and select **Other**. This should bring up the connection editor where you can connect the attributes.



Connect **L_ShoulderLoc.translate** attribute to **L_ShoulderWristDist**'s **Point2** attribute using the same method as above.

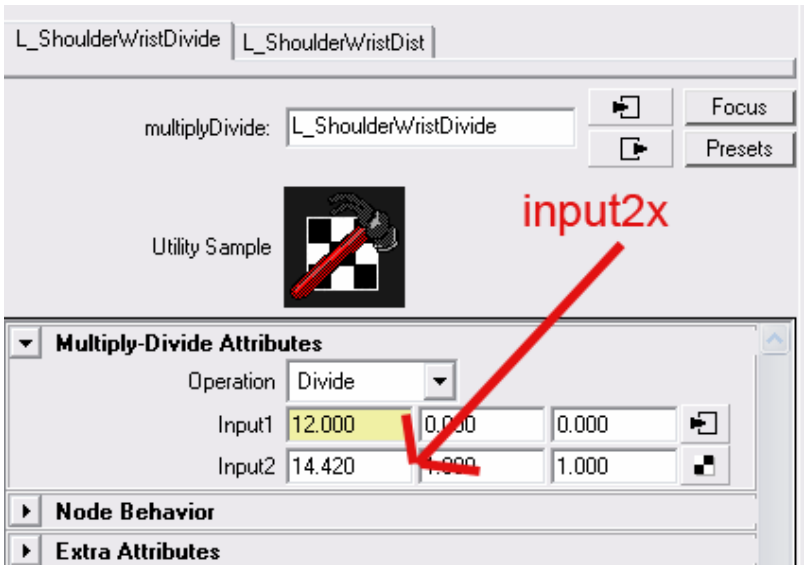
The distance node should now be measuring the distance between **L_Wrist_Ctrl** and **L_ShoulderLoc**, sweet.



Go back in to the Create Node panel. Create a **multiplyDivideNode**. Rename it, **L_ShoulderWristDivide**.

Connect **L_ShoulderWristDist.Distance** to **L_ShoulderWristDivide.input1x** (you shouldn't have to use the connection editor this time).

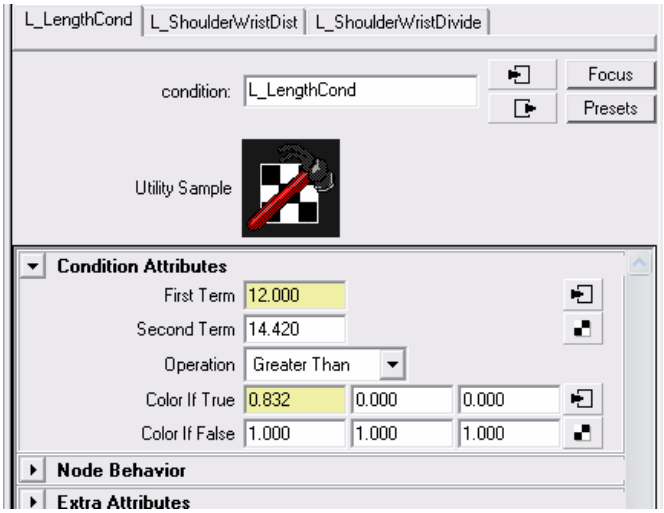
Select the **L_ShoulderWristDivide** and open the attribute editor. Remember the number you wrote down earlier ? Type that number in to **input2x**. Set the operation to **Divide**.



What this does is divide the current distance between the wrist and the shoulder by the length of the arm when fully extended. We can then use this figure out how much to stretch our joints. However first we need to make some additional nodes that will make sure the stretching will only take effect when we extend the arm beyond its full reach. Otherwise we'll end up with a crazy constantly scaling arm.

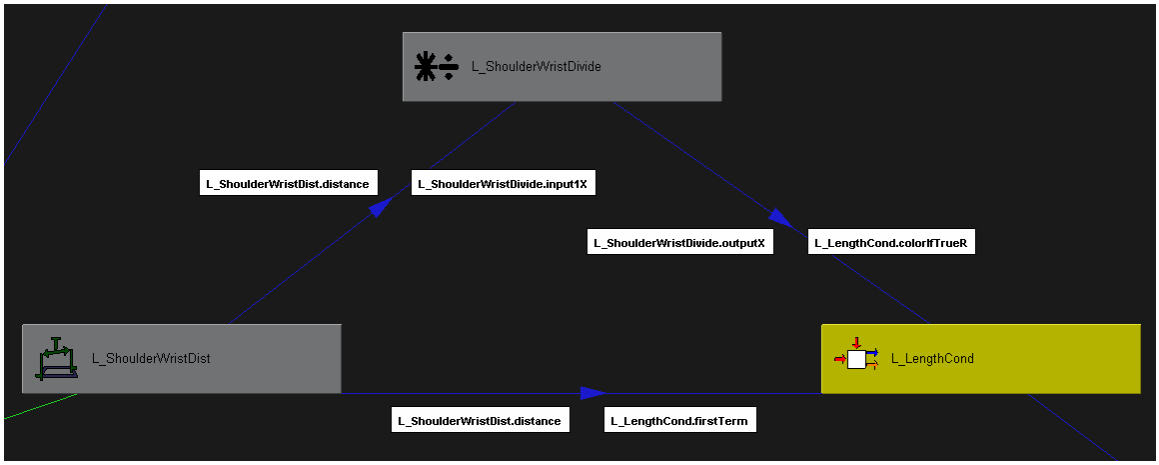
Create a **Condition** node. Name it **L_LengthCond**.

Open the attribute editor and copy the number you wrote down earlier in to the nodes **Second Term** attribute. Set the operation to **Greater Than**.



Connect **L_ShoulderWristDists.distance** to **L_LengthCond.firstTerm**.

Connect **L_ShoulderWristDivide.outPutX** to **L_LengthCond.colourIfTrueR**.



This set of nodes will make sure that only when the arm is extended to its full reach and beyond will the **colourIfTrueR** (divided arm length) result be passed along further in the network. Otherwise **colourIfFalseR** will be passed along the network, by default this should be 1.00. Thus stopping the bones from stretching.

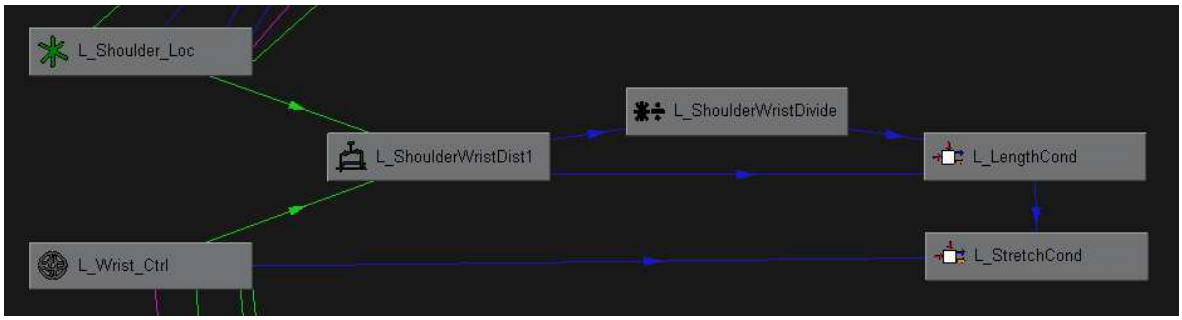
Not much further to go now. The last addition we need to make to this network is to take into account whether the **Stretch** attribute we created earlier is on or off. For this we'll need another **condition** node.

Go ahead and create one. Rename it **L_StretchCond**.

Connect **Wrist_Ctrl.Stretchy** to **L_StretchCond.firstTerm**. Set the **Second Term** to 1.00.

Connect **L_LengthCond.outColorR** to **L_StretchCond.colorIfTrueR**.

Set the nodes operation to **Equal**. And that's about it for this network, its pretty much ready to connect the output of **L_StretchControl** to the bones in order to make them scale. You're network should look something like the one below.

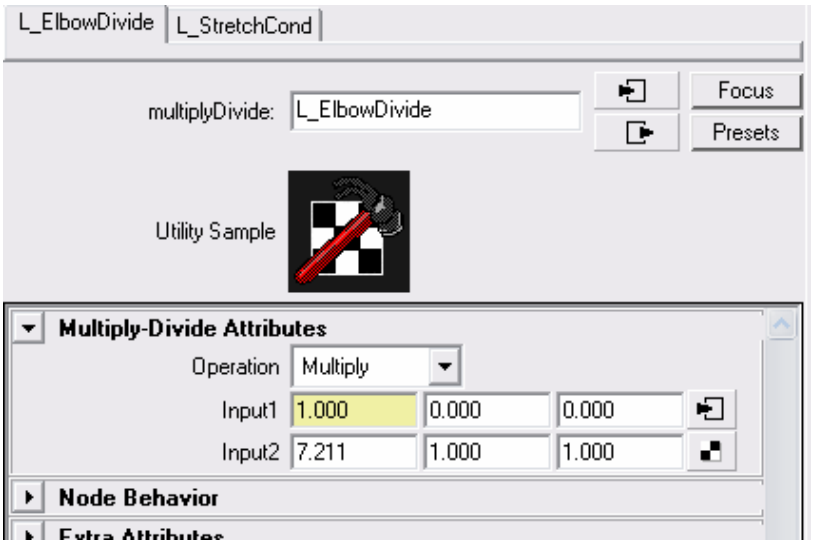


We could take the quick route and connect the output of **L_StretchCond** to the bones **scaleX** attributes, however scaling bones is not always a good idea, its better to translate them. This will require a small amount of extra work.

Still in the hypergraph, find the **L_Elbow** node. Create a new **multiplyDivide** node, re-name it **L_ElbowMultiply**.

Connect **L_StretchCond.outColorR** to **L_ElbowMultiply.input1X**.

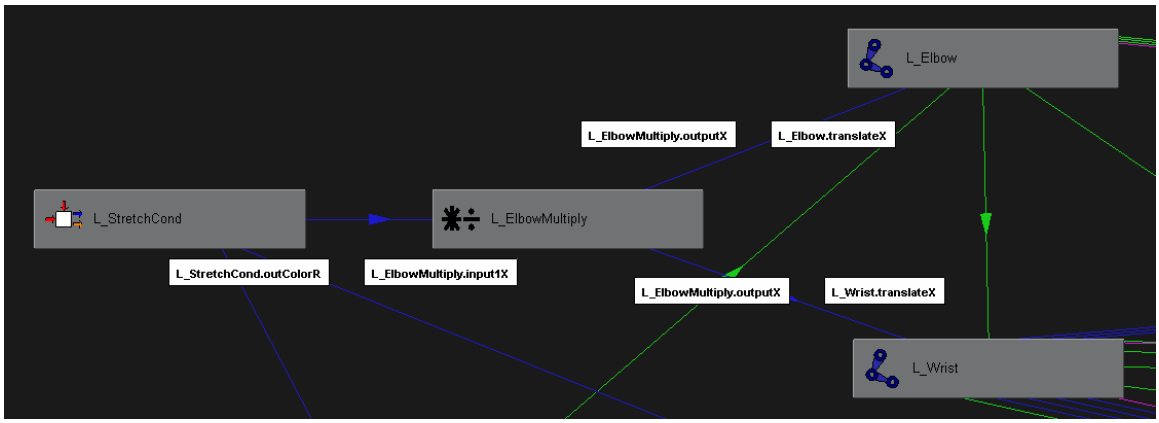
Take a note of the elbow joints **translateX** value. Open the attribute editor and change **L_ElbowDivides.input2X** attribute to that value. Make sure the operation is set to **Multiply**.



Finally connect **L_ElbowMultiply.outputX** to **L_Elbow.translateX**.

Your arms upper and lower arm joints should be of equal length, so you can connect **L_ElbowMultiply.outputX** to **L_Wrist.translateX** aswell.

And that's it. Now when you set the **Stretchy** attribute to on and pull the arm beyond its normal extension it should stretch. The "**Bendy**" bones won't stretch correctly yet, we'll deal with that in the next section.

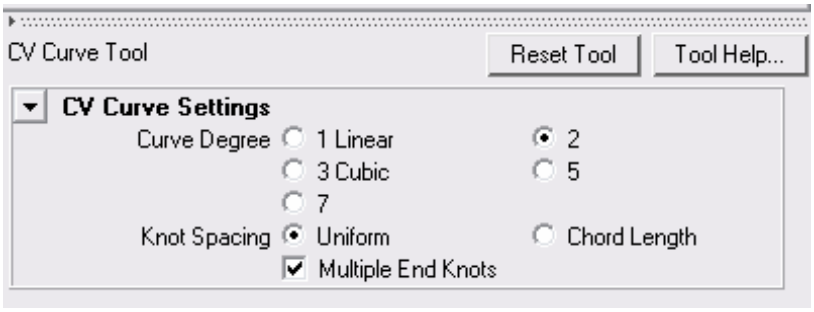


Adding the Bendy Setup

Creating Curves and splineIk

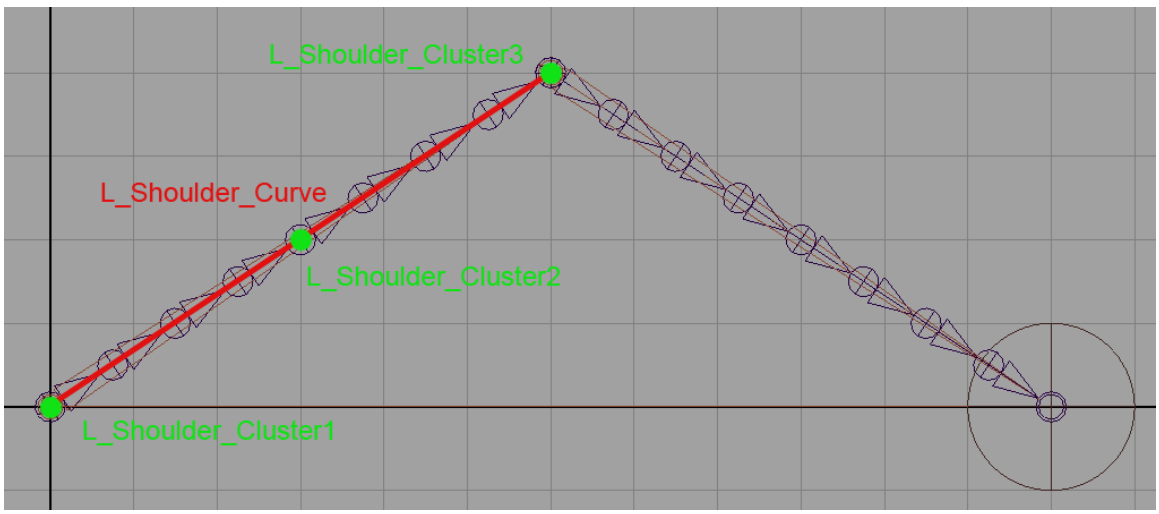
Now we're ready to create the set up that will control the bending.

Create a CV curve. Use the option box and set the **Curve Degre**e to **2** before you draw it.

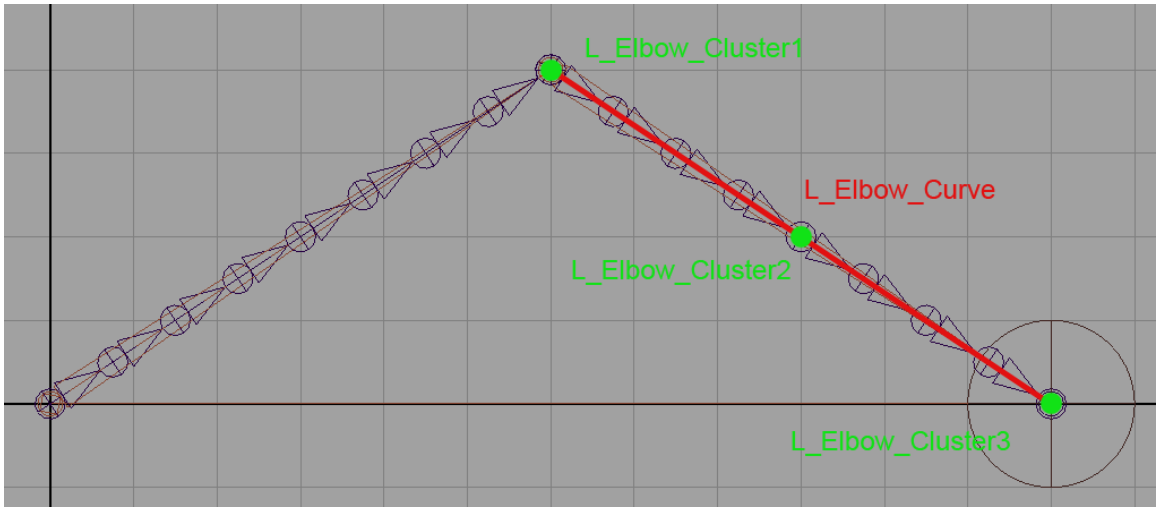


Use point snapping to draw the curve. Put one point at the shoulder joint, one at the mid point (between the shoulder and the elbow, point snapping it to one of the "**Bendy**" joints is probably the best way) and the last point at the elbow position. Rename the curve **L_Shoulder_Curve**.

Go in to component mode and create clusters for each of the curves 3 CV's. Do this by selecting a CV and going *Deform >> Create Cluster*. Rename the clusters **Shoulder_Cluster1**, **Shoulder_Cluster2** and **Shoulder_Cluster3**.



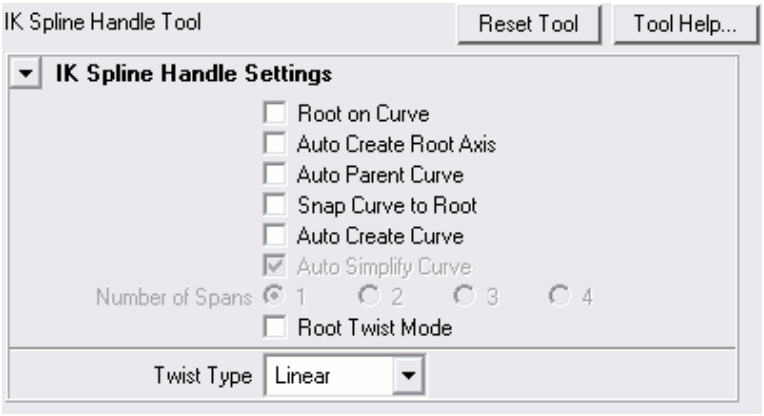
Repeat the above procedure to create a curve and some clusters going from the elbow to wrist positions.



That's the curves created, now we can use them to create a splineIk solver to control the joints.

Go *Skeleton >> Ik Spline Handle Tool*. In the option box turn off **all** the options.

Go in to the outliner and whilst holding *Ctrl* select in order **L_Shoulder_Bendy_1**, **L_Shoulder_Bendy_End** and **L_Shoulder_Curve**. This will create a splineIk system using the curve you draw earlier.



Repeat the above to create a system for the **L_Elbow_Bendy** joints using **L_Elbow_Curve**. Rename the two ikHandles **L_ShoulderSpline_ikHandle** and **L_ElbowSpline_ikHandle**.

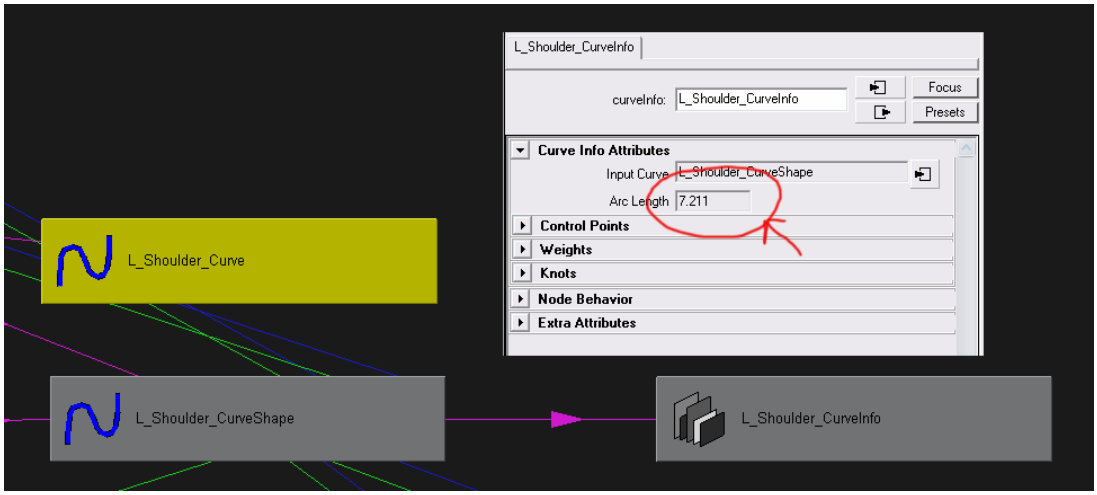
Making it Bendy

Before we get in to the main business of giving the splines the ability to bend we need to give the bones they control the ability to scale as the curve scales, otherwise the arm will break. The way we set this up shares some similarities to the way we created the stretch set up earlier, however it is much simpler.

First select **L_Shoulder_Curve** and type this into the command line.

```
arclen -ch 1;
```

With your curve still selected, open the hypergraph. Re-graph it. If you look at the curves shape node it should now have a **curveInfo** node attached to it. This node gives us the length of the curve which we need to work out how to scale the bones. Rename it to **L_Shoulder_CurveInfo**.

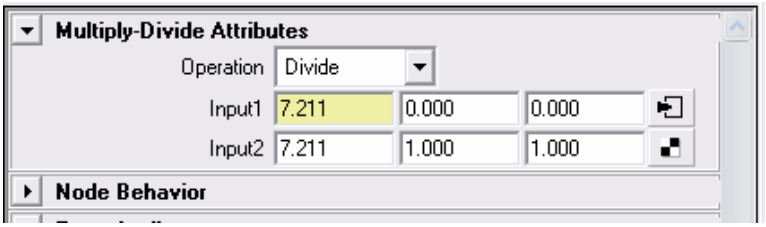


With the node selected and open the attribute editor. Make a note of the nodes arc length attribute.

Create a **multiplyDivideNode**. Rename it **L_Shoulder_CurveDivide**. Make sure the operation is set to **Divide**.

Plug **L_Shoudler_CurveInfo.arcLength** into **L_Shoulder_CurveDivide.input1x**

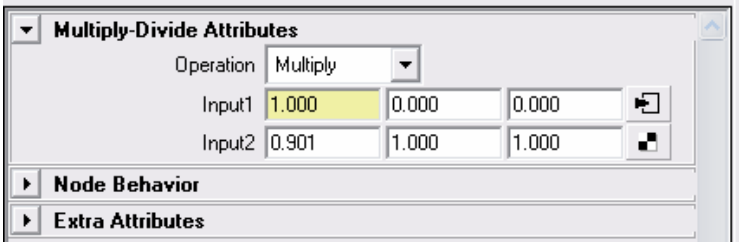
In the attribute editor change **L_Shoulder_CurveDivide.input2x** to the value you noted a little while ago. It should be the same as **input1x**.



Create another **multiplyDivideNode**. Rename it **L_Shoulder_CurveMultiply**. Set the operation to **Multiply**.

Connect **L_Shoulder_CurveDivide.output1x** to **L_Shoulder_CurveMultiply.input1x**.

Select one of the **Shoulder_Bendy** joints. Make a note of its **translateX** attribute. As long as you created them using the script provided they should all be of equal length. Set **L_Shoulder_CurveMultiply.input2x** this value.



With **L_Shoulder_CurveMultiply** selected, in the viewport shift select all the **L_Shoulder_Bendy** joints.

Refresh the hypergraph (*Graph >> Input and Ouput Connections*).

Plug **L_Shoulder_CurveMultiply.outputX** in to each of the **L_Shoulder_Bendy** joints **translateX** attributes starting from **L_Shoulder_Bendy_2** (**DO** include **L_Shoulder_Bendy_End**).

Of course this will only work if the bendy joints do have equal lengths, which is why it's important not to eyeball it when you create them.

And that's it. Repeat the above steps to make the **L_Elbow_Bendy** joints stretchy. Drag some of the clusters around and you should see that the bones scale to fit the curve.

Constraining Clusters and Creating Locators

Now we can set about attaching the clusters to the arm and creating the bend effect. For the start and end clusters on each curve this is simple, you point constrain them to the joints they're lying on top of like so.

Point constrain **L_Shoulder_Cluster1** to **L_Shoulder**.
Point constrain **L_ Shoulder_Cluster3** to **L_Elbow**.
Point constrain **L_Elbow_Cluster1** to **L_Elbow**.
Point constrain **L_Elbow_Cluster3** to **L_Wrist**.

For the middle clusters on each curve things get a bit more complex. The real problem that needs to be solved with this kind of set up lies in finding a method that will position the middle clusters of our curves correctly as the arm moves around in order to create a nice smooth bend in our spline curves. This will involve creating some more locators, a bit of vector maths and an expression or two. It's really the meat of the setup so let's get going.

Create four new locators. Name them **L_Shoulder_ProjectLoc**, **L_Shoulder_ResultLoc**, **L_Elbow_ProjectLoc** and **L_ElbowResultLoc**.

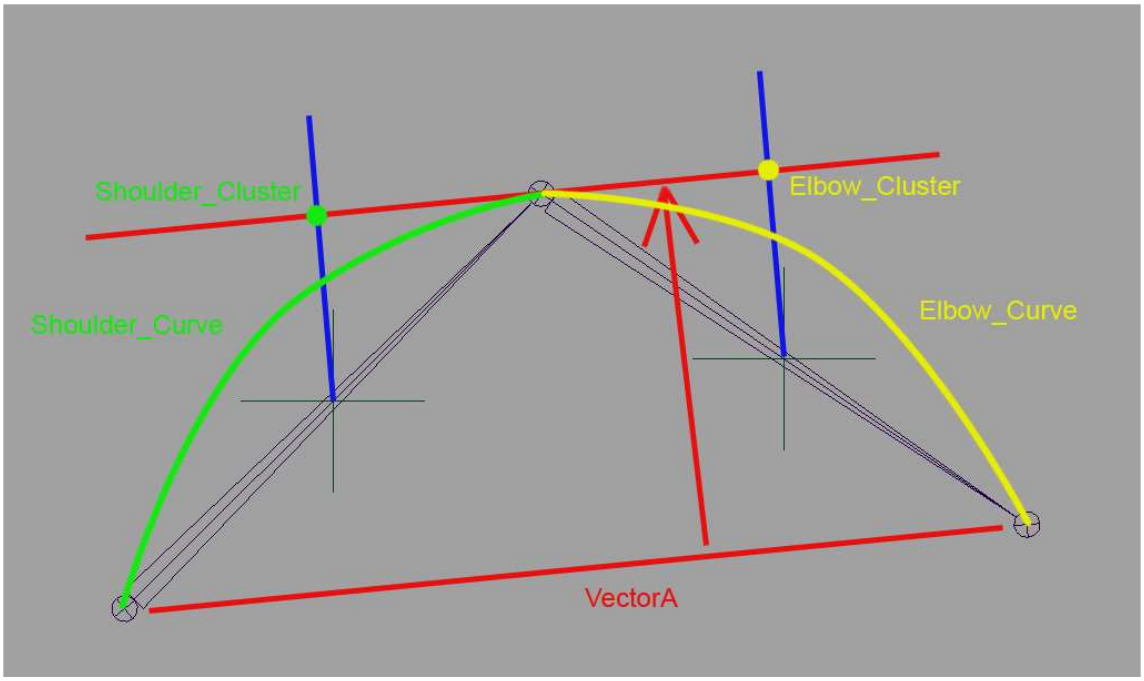
In this order shift select (or Ctrl + Select in the outliner) **L_Shoulder**, **L_Elbow** and **L_Shoulder_ProjectLoc** and go *Constrain >> Point* (make sure maintain offset is not ticked).

Now in this order select **L_Wrist**, **L_Elbow** and **L_Elbow_ProjectLoc** and go *Constrain >> Point*.

This should create a weighted constraint that initially places the project locators half way between the joints they are constrained to. This is fine for now, but later we'll use the weighting of these constraints to animate the amount of bend in our arm. First we need to deal with the other two locators, these will be controlled by expressions.

The Expressions

The idea behind the following expression is explained in the diagram below.



The basic idea is we take a vector (**VectorA**) between the shoulder and the wrist and transform it so that it sits at the elbow. Then we project our two "Projection" locators on to this vector. The position where the two vectors intersect (the projection) is where we need to place our clusters in order to create a nice smooth curve (in our case we'll use the "**Result**" locators and point constrain the middle clusters to them).

A detailed explanation of the maths involved here is a bit beyond the scope of this document, but if you are interested have a look at Mike Isners website (www.isner.com) and follow the link to the **Thinking Tactics from The Guild Of Scripted Operators** section of Softimages site. The videos and samples on this site are an excellent resource for learning about vector maths for use in rigging. In particular have a look at the *Spline Through Points* and *Poly Loft Muscle videos*.

Let's get working on our expression.

Open up the expression editor. Here's the expression you'll need to type in. If you aren't too worried about understanding the code you could just copy this out of the finished scene file provided.

```
vector $A = unit(<<(L_WristLoc.translateX - L_ShoulderLoc.translateX),
                   (L_WristLoc.translateY - L_ShoulderLoc.translateY),
                   (L_WristLoc.translateZ - L_ShoulderLoc.translateZ)>>);

vector $B = <<(L_Shoulder_ProjectLoc.translateX - L_ElbowLoc.translateX),
               (L_Shoulder_ProjectLoc.translateY - L_ElbowLoc.translateY),
               (L_Shoulder_ProjectLoc.translateZ - L_ElbowLoc.translateZ)>>;

vector $C = <<(L_Elbow_ProjectLoc.translateX - L_ElbowLoc.translateX),
               (L_Elbow_ProjectLoc.translateY - L_ElbowLoc.translateY),
               (L_Elbow_ProjectLoc.translateZ - L_ElbowLoc.translateZ)>>;

float $dotProd = dot($A, $B);
float $length = mag($A);
float $scale = $dotProd / ($length * $length);

L_Shoulder_ResultLoc.translateX = L_ElbowLoc.translateX + ($A.x * $scale);
L_Shoulder_ResultLoc.translateY = L_ElbowLoc.translateY + ($A.y * $scale);
L_Shoulder_ResultLoc.translateZ = L_ElbowLoc.translateZ + ($A.z * $scale);

float $dotProd = dot($A, $C);
float $length = mag($A);
float $scale = $dotProd / ($length * $length);

L_Elbow_ResultLoc.translateX = L_ElbowLoc.translateX + ($A.x * $scale);
L_Elbow_ResultLoc.translateY = L_ElbowLoc.translateY + ($A.y * $scale);
L_Elbow_ResultLoc.translateZ = L_ElbowLoc.translateZ + ($A.z * $scale);
```

Expression Explanation

In the first section we get three vectors.

$A is the vector between the **L_WristLoc** and **L_ShoulderLoc**.

$B is the vector between **L_ShoulderLoc** and **L_ElbowLoc**.

$C is the vector between **L_Elbow_ProjectLoc** and **L_ElbowLoc**.

The next block of code solves the maths explained previously.

The next block positions the **L_Shoulder_ResultLoc** at the vector that gets spat out of this calculation.

The next blocks of code are a repeat of the previous two (just solving for the **L_Elbow_ResultLoc**).

Once you have the expression working the **L_Shoulder_ResultLoc** and **L_Elbow_ResultLoc** should pop in to their proper places. Now you just constrain the middle clusters to them.

Select in order **L_Shoulder_ResultLoc** and **L_Shoulder_Cluster2** and go *Constrain >> Point*.

Select in order **L_Elbow_ResultLoc** and **L_Elbow_Cluster2** and go *Constrain >> Point*.

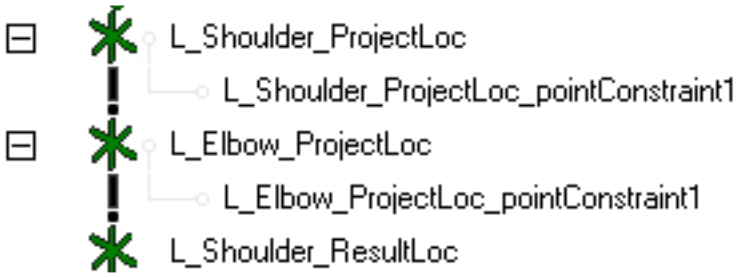
You should now have a pretty much fully working bendy arm. Try moving **Wrist_Ctrl** around to see it work. The only thing left to do is to add some controls that will let you alter the amount of bend that your arm displays. This is quite simple to get working and doesn't involve any more maths. We'll do this next.

Adding Bendy Controls

Select **L_Wrist_Ctrl** and open the Add Attribute window (*Modify >> Add Attribute*).

Create two new attributes. Make them both **Float** type, **Min 0**, **Max 1**, **Default 0**. Name them **Sh_Bendy** and **El_Bendy** (Shoulder Bendy and Elbow Bendy).

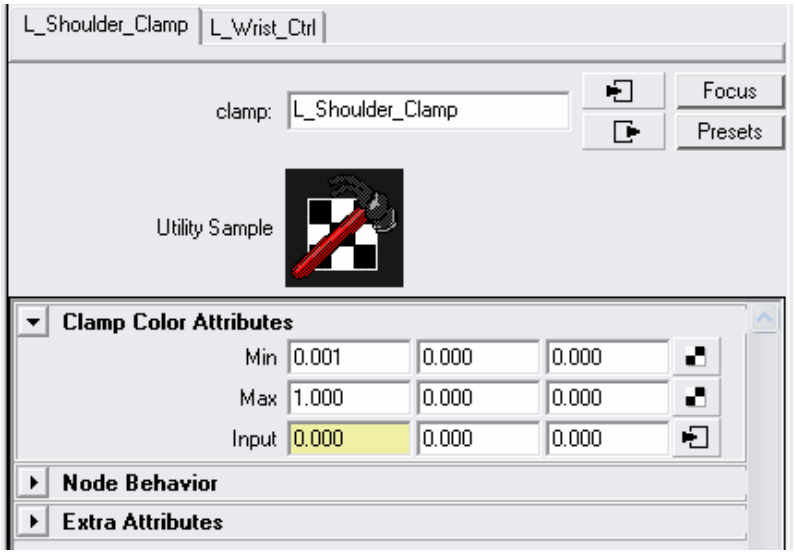
In the outliner expand **L_Shoulder_ProjectLoc** and **L_Elbow_ProjectLoc** so that you can see there constraints.



Still in the outliner Ctrl + Select **L_Wrist_Ctrl**, **L_Shoulder_ProjectLoc_pointConstraint1** and **L_Elbow_ProjectLoc_pointConstraint1**. Open the hypergraph and re-graph it.

Create a **Clamp** node. Rename it **L_Shoulder_Clamp**.

Set **L_Shoulder_Clamps MinR** to 0.001, **MaxR** to **1.00**.



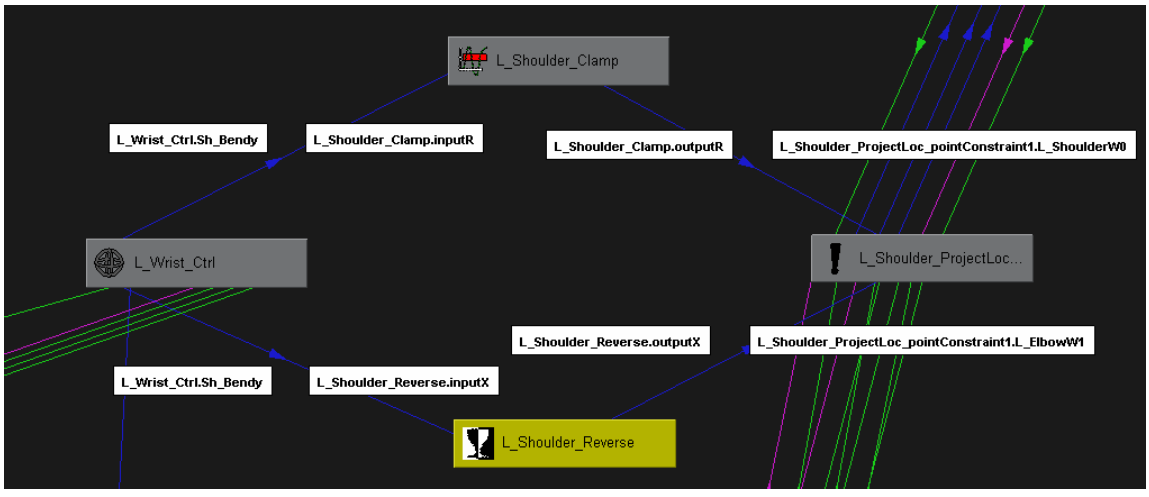
Connect **L_Wrist_Ctrl.Sh_Bendy** to **L_Shoulder_Clamp.inputR**.

Connect **L_Shoulder_Clamp.outputR** to **L_Shoulder_ProjectLoc_pointConstraint1.L_ShoulderW0**.

Open the Create Render Node panel. Create a **Reverse** node. Name it **L_Shoulder_Reverse**.

Connect **L_Wrist_Ctrl.Sh_Bendy** to **L_Shoulder_Reverse.inputX**.

Connect **L_Shoulder_Reverse.outputX** to **L_Shoulder_ProjectLoc_pointConstraint1.L_ElbowW1**.



This network is really simple. All we're doing is connecting the **S_Bendy** attribute into one of the constraint weights and for the other of the two weights we put the Reverse node in between (0 becomes 1 vice versa) so as you drag your bendy attribute the constraint weights start at 1, 0 and smoothly interpolate to end up at 0, 1.

The reason we have the clamp node between wrist ctrl and one of the constraint weights is that we don't want **L_Shoulder_ProjectLoc** to move right on top of the elbow joint (at completely 0). This is for two reasons, we're later going to use it as part of an aim constraint setup and having the locator move like this would break it. I have also that I have found the stretching effect on the arm to be much more stable this way.

Now have a play with the **Sh_Bendy** attribute in the channel box. You should see the bend moving from completely straight at 0 to a nice bendy curve at 1.

Using the steps above you should easily be able to work out how to connect the **El_Bendy** attributes to **L_Elbow_ProjectLoc_pointConstraint1** correctly.

Final Steps

Adding Wrist Twist

One cool thing with using this kind of set up is that we already have a nice long chain of joints we can use to create a forearm twist set up. And even better, we're using splineIk, so there is automatically a **Twist** attribute on the ikHandles that we can use to do this. Here's how.

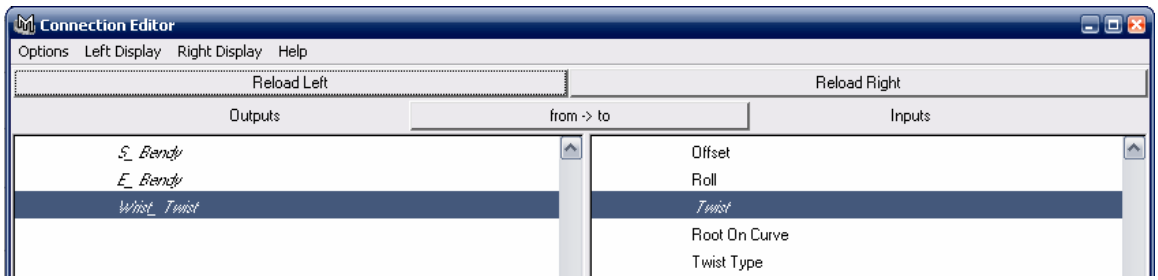
Create a new attribute on **Wrist_Ctrl**.

Call it **Wrist_Twist**. Make it a **Float** type, **min -90**, **max 90**, **default 0**.

Open the connection editor (*Window >> General Editors >> Connection Editor*).

Load **Wrist_Ctrl** in to the left pane, **L_ElbowSpline_ikHandle** into the right pane.

Connect **Wrist_Ctrl's Wrist_Twist** attribute to the ikHandles **Twist** attribute.



That's all we need to do. You could add another attribute to control the upper arm twist if you wanted. You could also add some sort of connection between the rotation of the characters wrist and the amount of twist, instead of using an attribute. It's up to you.

Adding a Skinning Elbow

One problem with the setup as it stands is that it will be very difficult to get nice deformation around the elbow area. What we need is a joint at the elbow that will basically stay flat relative to the vector between the wrist and the shoulder (the one we're using in our expression to create the bend effect). We also need the joint to move back and forth slightly to compensate for the joints either side moving as the arm bends. For various reasons we can't use any of the joints we have so far, so we'll have to create a new one.

Create a new single joint. Name it **L_Elbow_Skin**.

Shift select **L_Shoulder_Bendy_8** and **L_Elbow_Bendy_2** and go *Cons rain >> Point*.

Parent **L_Elbow_Skin** to **L_Shoulder**.

Create a locator. Point snap it to the **L_Elbow** and then move it a few units up from the elbow in **Y**. Rename it to **L_Elbow_SkinUpLoc**.

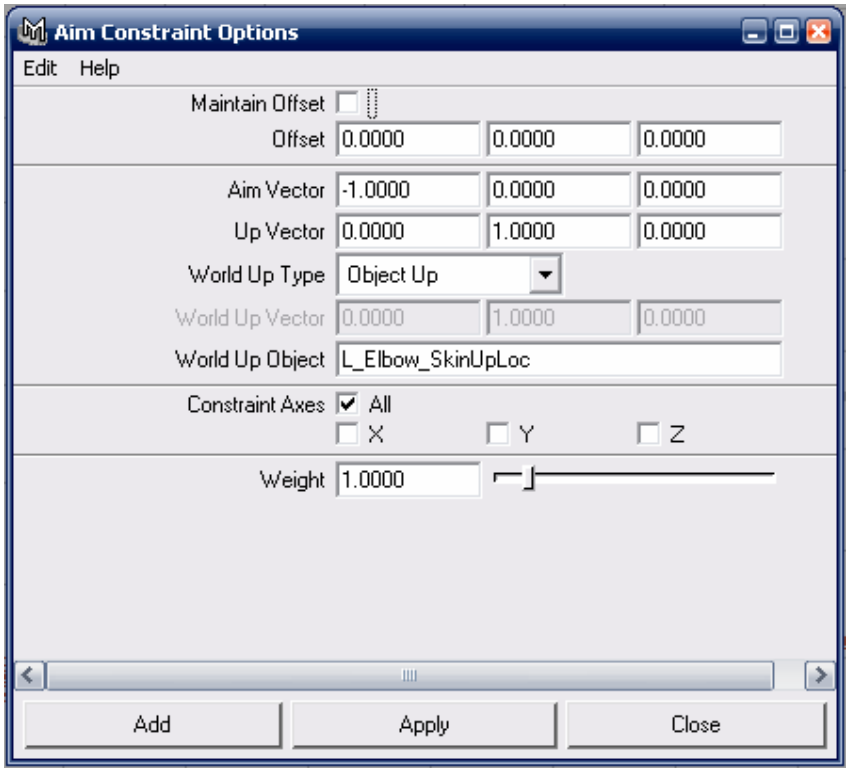
Parent **L_Elbow_SkinUpLoc** to **L_Elbow**. Zero out its rotations if you want (it doesn't really matter).

Turn up the bend controls slightly so you can get to **L_Shoulder_ResultLoc** easily.

Open the **Aim Constraint** option box.

We want **L_ElbowLocs** up vector to always point at the **L_Elbow_SkinUpLoc**. So set the **World Up Type** to **Object Up** and enter **L_Elbow_SkinUpLoc** in the World Up Object field. Don't hit add just yet.

Select you're **L_Elbow_Skin** joint and set the axis display type to Object (hold down *W* and use the hotbox that appears). Look at you're joint and find out what its Up Vector axis is. On mine its +Y (0, 1, 0). Also look at what you're aim axis needs to be. I want to keep the current orientation of my joint so it will be –X (-1, 0, 0).



Enter the figures in the X, Y and Z fields according to you're joints orientation and add the constraint.

Select **L_Shoulder_ResultLoc**, Shift + Select **L_ElbowLoc**, and hit **Add**. **L_ElbowLoc** should point at **L_Shoulder_ResultLoc**, with its Y axis always pointing at **L_Elbow_SkinUpLoc**.

Finally Shift + Select in order **L_ElbowLoc** and **L_Elbow_Skin** and go *Constrain >> Orient*.

Adding a pole vector

Just one last small thing to do.

Create another implicit sphere (use a locator or nurbs shape if you wish). Rename it **L_Arm_Pv**.

Snap **L_Arm_Pv** to the elbow position and move it back in space a few units.

Select **L_Arm_Pv**, Shift + Select **L_Main_ikHandle** and go *Constrain >> Pole Vector*.

And that's the setup pretty much finished. All you need to do now is lock all the necessary attributes and clean up the outliner by grouping everything together. I'll trust you to do this ☺

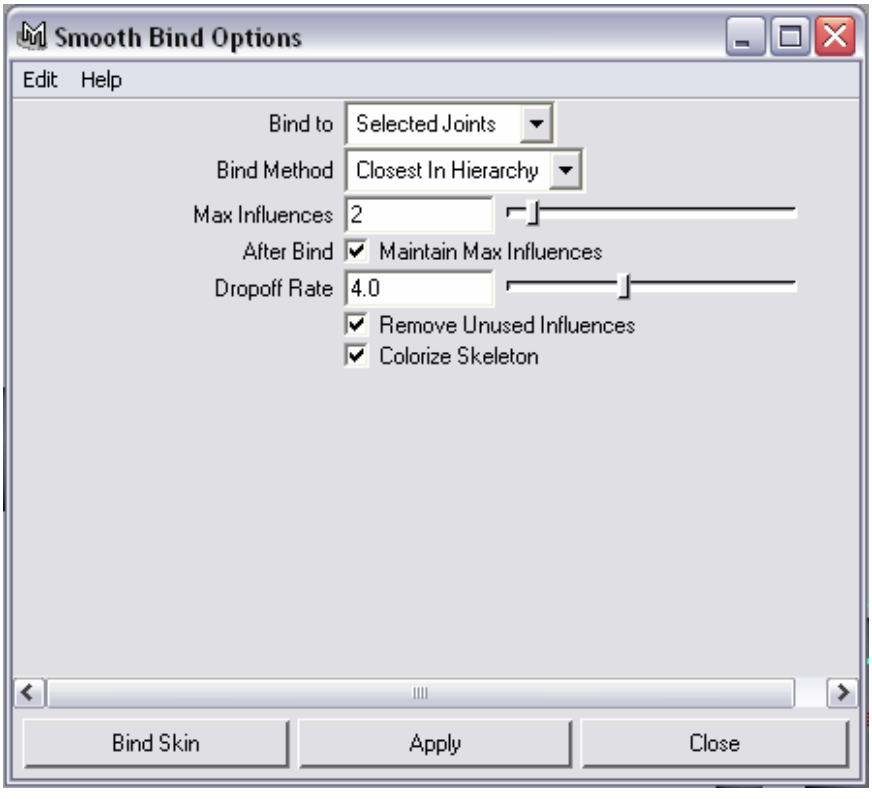
Notes

Skinning

The joints you should be skinning to are.

L_Shoudler_Bendy_1 to **L_Shoudler_Bendy_8**
L_Elbow_Skin
L_Elbow_Bendy_1 to **L_Elbow_Bendy_8**
L_Wrist (and on to your hand rig)

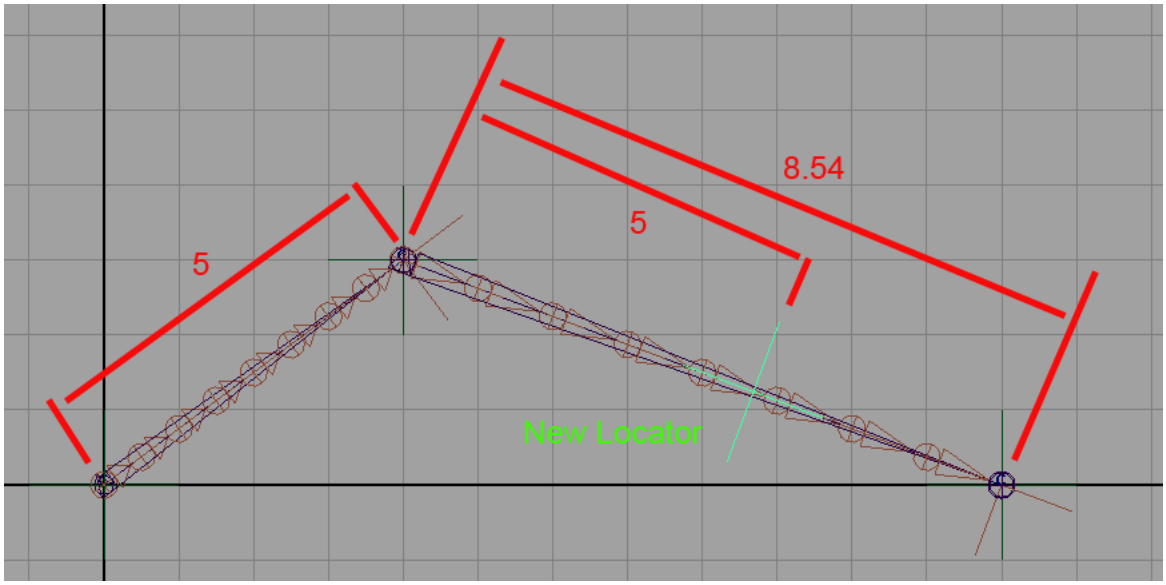
If you want to do a test mesh the best way is to select all these joints (in order), shift select your mesh and go *Skin >> Bind Skin + Option Box*. Then set the **Bind To** option box to **Selected Joints**. Make sure you use a decent number of influences.



This will skin your mesh to the bones you have selected. Once it's skinned I've found that you will usually have to go in and make sure that the vertices at the elbow area are heavily influenced by **L_Elbow_Skin**. For some reason Maya's default skinning doesn't give it nearly enough influence. It usually takes a little bit of playing around with the weights, but it shouldn't take too long to get a quite nice result.

Arms With Non Equal Length

It's quite easy to adapt this setup for use with arms where the upper arm and forearm are different lengths. The basic idea is that you place a locator along the forearm that matches the length of your upper arm. Then parent this to the elbow joint. The only bits of the rig you should have to alter are the node networks that concern the arm stretching. So for instance instead of measuring from the **Wrist_Ctrl** to the Shoulder, you measure from the new locator to the shoulder and so forth.



I might write a proper little addition to the document that explains how to do this in detail at some point. But if you understand the principles behind the setup it should be fairly easy for you to figure out yourself ☺