Rigging in AutoDesk Maya: A Compendium

DISC241: Foundations of Animation

Audrey Zeng

Table of Contents

Overview

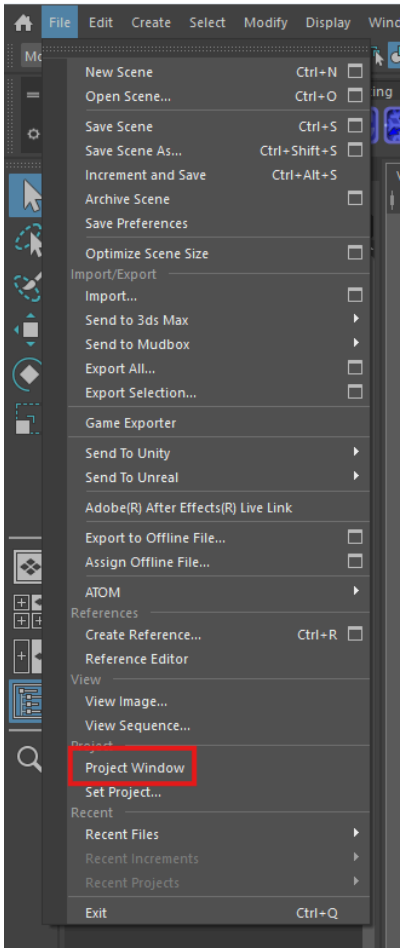
What is rigging?
In 3D animation, rigging involves creating a skeleton, or “rig” to control a 3D character for animation. Creating a rig means (1) creating the bones, or “joints” which make up a skeleton, (2) creating controls by which animators can control the rig, and (3) constraints dictating how moving each control affects the others.

This compendium will cover the rigging process in Autodesk Maya, an industry-standard 3D animation software.

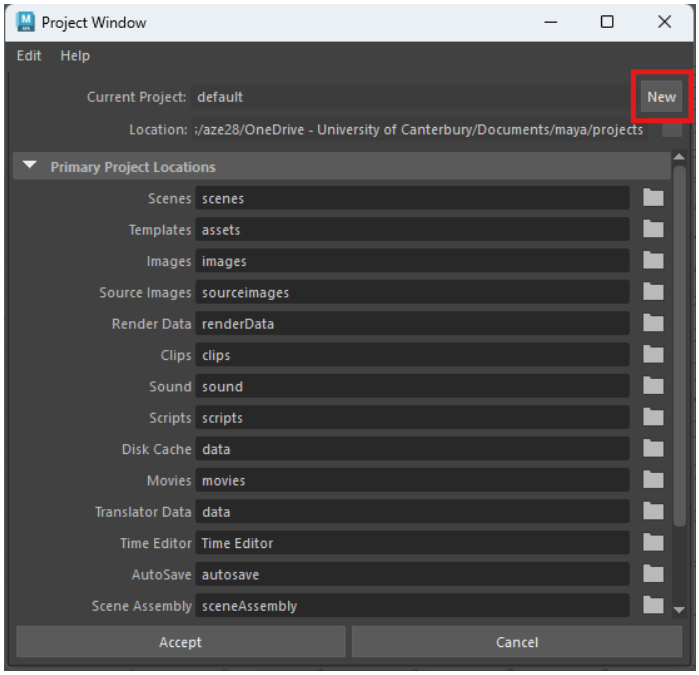
Getting Started

Creating a Maya Project

Open Maya, and click “File” → “Project Window”



Click “New”



Check that all directories are as expected, then click accept.

Maya Directory Structure

Maya identifies and finds files to use (scenes, scripts, skins, etc.) by folder location. It is important to keep files under their respective folders so you can use them from within the Maya GUI! Here are the important folders within a project that Maya uses

Assets:	Your base mesh
Scenes:	Your working rigging files
Scripts:	all the scripts you want to use
Sourceimages:	Textures and maps for Maya to use
Images:	Rendered images (for your own use)

Working with and Saving Files

Opening Scenes:

Every time you work in Maya, make sure you’re working in the right project:

- “File” → “Set Project” → Navigate to and click on your project folder → “Set”

This makes sure Maya is saving files in the correct place.

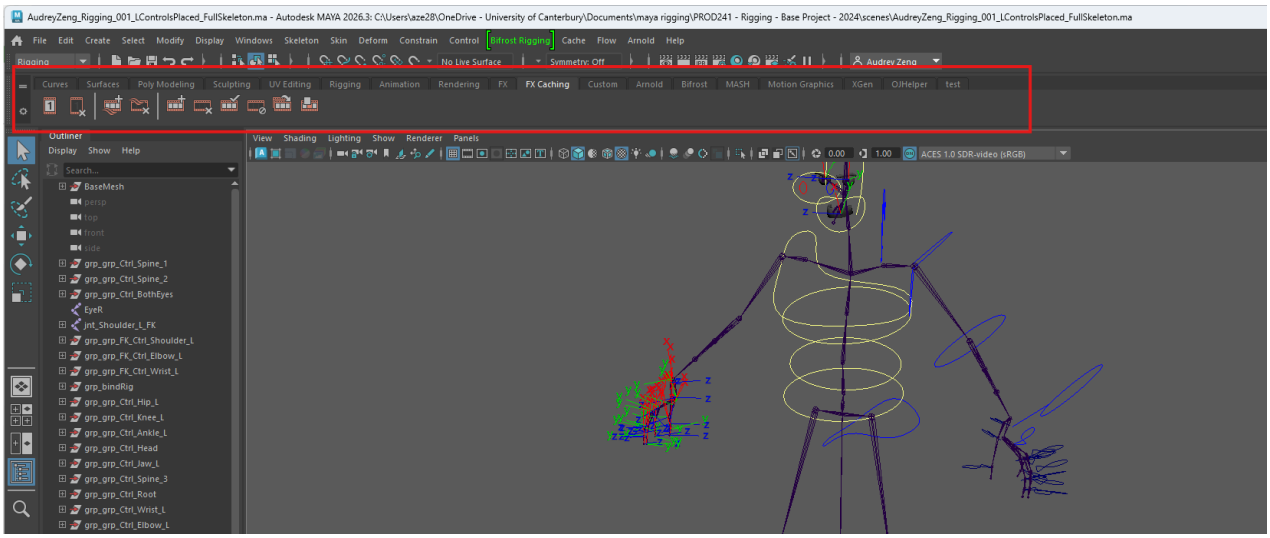
Saving:

- Save your rig incrementally – a new file after every time you work on it
    - If something goes wrong, you have a previous version to return to
    - You can do this manually with the “save as” option (ctrl+shift+s) or use Maya’s default increment and save option (ctrl + alt + s)
- Name saves with 3-digit padding (ex. “Rig_001.ma)
- Save as Maya Ascii (*.ma) instead of Maya Binary (*.mb)
    - Especially for complex, fast-moving projects, Maya Ascii is the safer format
    - Ascii files are human readable, meaning that if something goes wrong, you can go into the Ascii file and identify/fix the issue – you won’t lose your rig

Automation: Maya Scripts and Shelves

Rigging is very repetitive. Shelves (collections of command shortcuts) can help automate this process so you aren't navigating through a sea of menus every other minute to access the same command.

Shelves are the little collections of icons under the top menu.

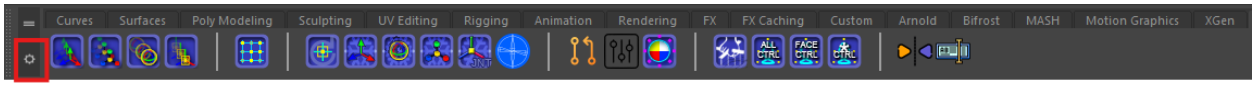


Shelves are a type of script in Maya's embedded programming language (Maya Embedded Language). All Maya scripts (including shelves) have an extension of *.mel

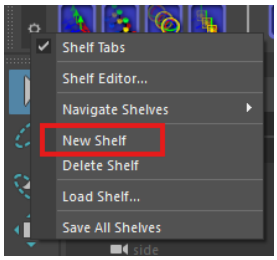
Installing and Creating a Maya Shelf

Making a custom shelf:

1. Click the cog left of the shelves



2. "New Shelf"



3. Give it a descriptive name

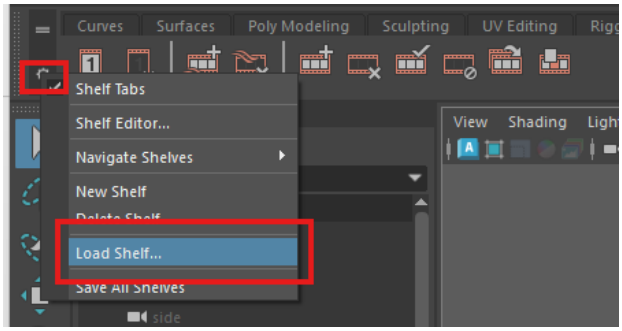
To add a command to your shelf: Ctrl + shift + left click on the button

Note: Maya shelves are stored in local directories

Shelves are stored locally in C:\Users\[user]\Documents\maya\20xx\prefs\shelves. If you are working on a project saved in an online location, your shelf will not be saved to the cloud. If you use a different device to access your project, you will no longer have your shelf. Make sure to save your shelf on the cloud, then load it in every time.

Installing a Maya shelf:

Option 1: Click the cog next to the shelves → "Load shelf" → select your .mel file



Option 2: Drag and drop your shelf from your filesystem into Maya.

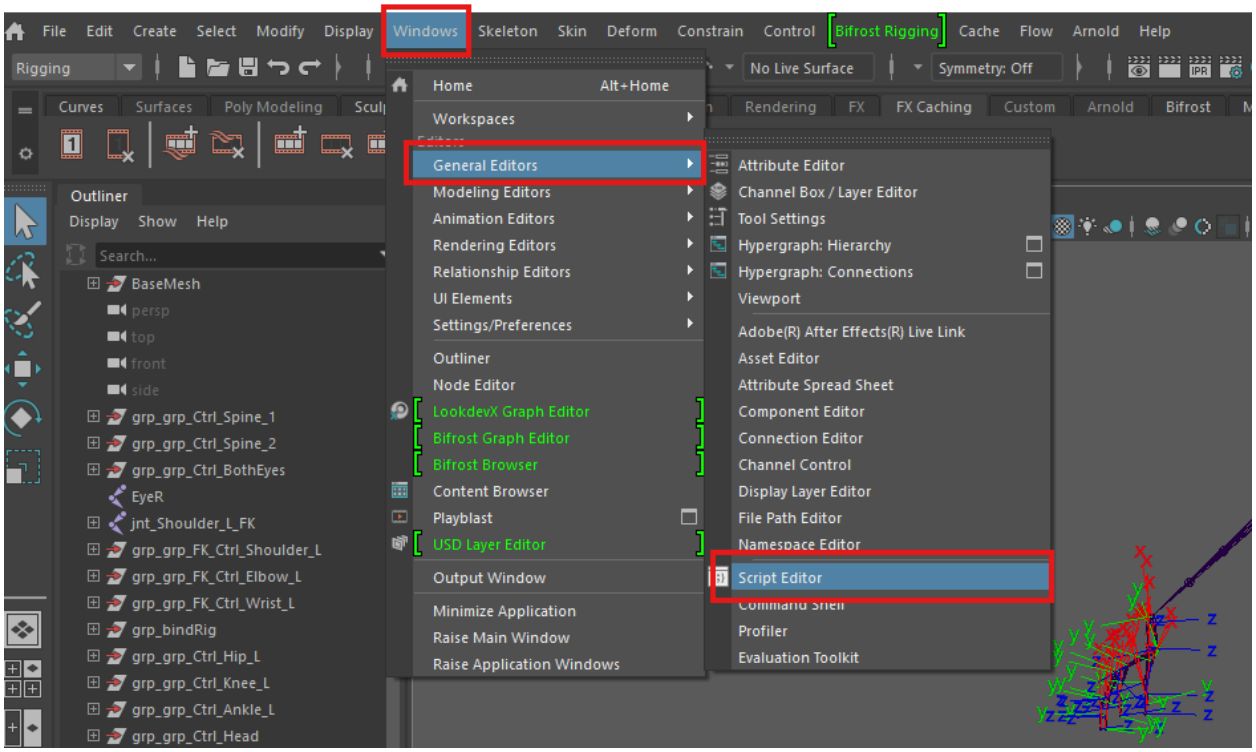
Investigating Maya shelf details:

If you ever forget what an icon on a shelf is, right click on the icon and press "edit". This will show you the exact commands Maya runs when you click on the shelf icon.

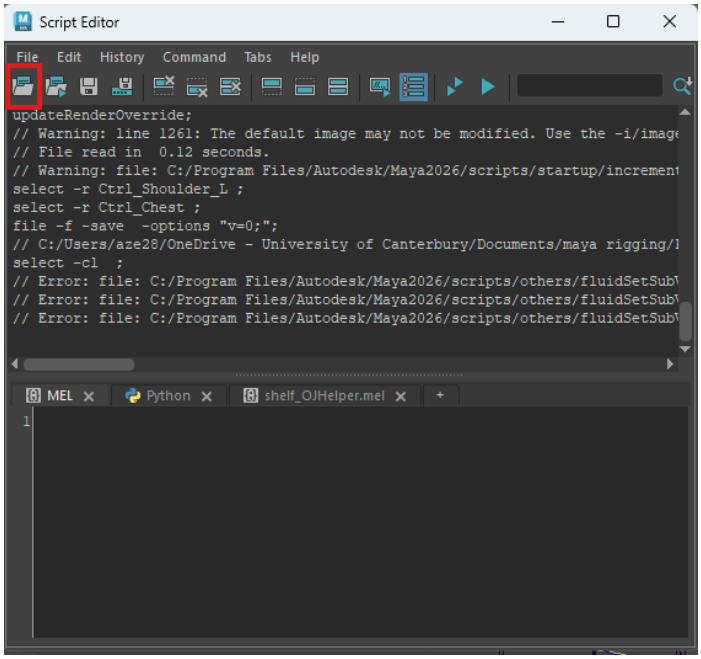
Running a Maya script

More generally, you can run Maya scripts through the *Script Editor*. Maya has many types of editors providing different UIs for different purposes. This is one of them. Maya is also compatible with Python, so you can use that to automate if you're more familiar with Python!

From the top menu, click "Windows" → "General Editors" → "Script Editor"

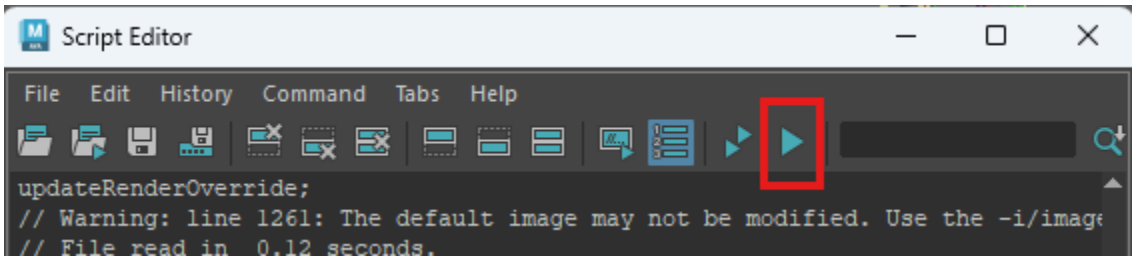


Click the "open script" button



Double click on your *.mel file. It should open as a tab next to the MEL and Python terminals in the bottom half of the script editor.

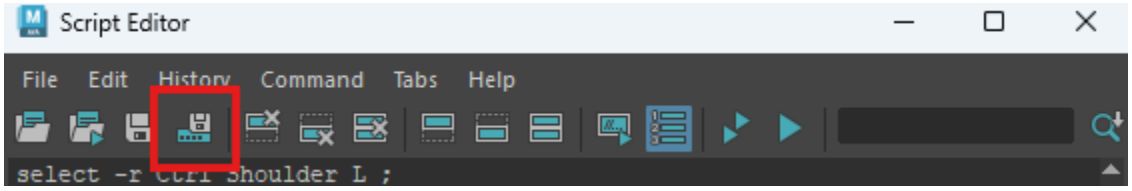
To execute the script, press the right-facing blue arrow:



> *Note:* You can also code your own scripts or run commands from the MEL terminal in the bottom half of the script editor. Type in your desired commands, then press the run button
> *Another Note:* The top half of the script editors shows a running history of every command Maya is running! This makes it convenient to save a series of commands to a shelf, as detailed next.

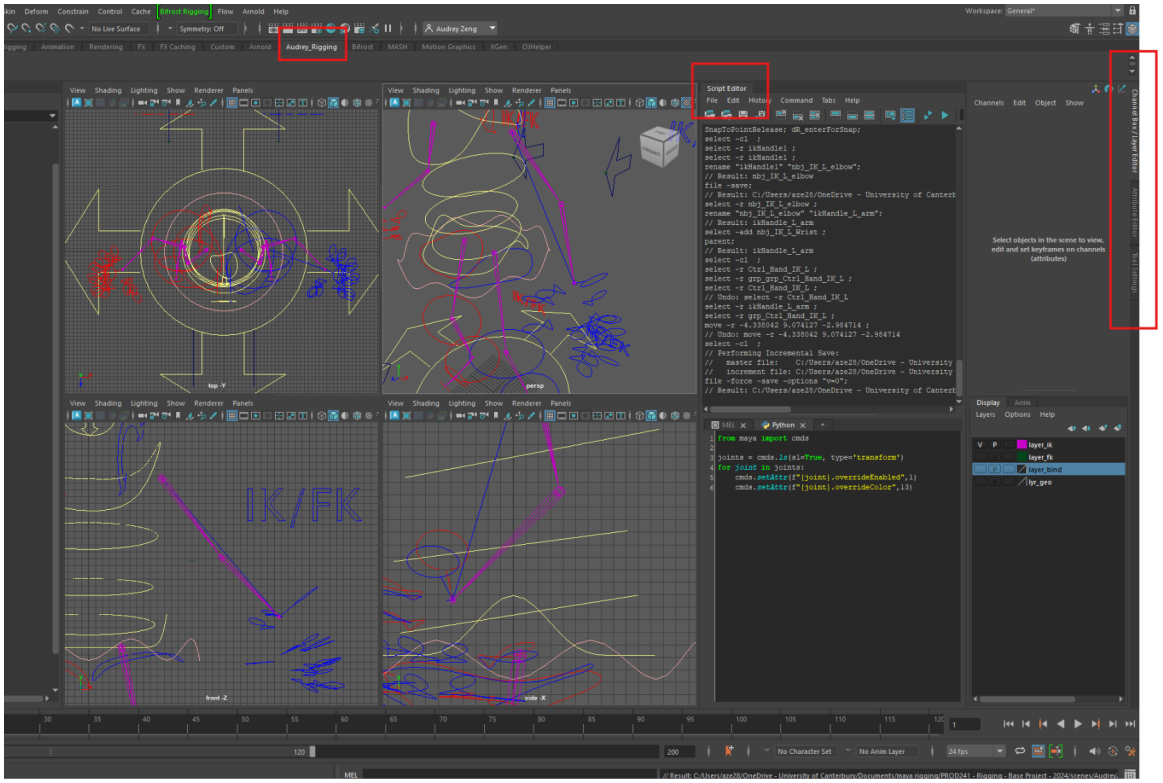
Save a script or history of commands to a shelf:

- Select all the commands in the script editor, and press "save script to shelf" (the icon right under the "history" button)



- Give it a name, then save it as a MEL file. It will save to the shelf you have currently selected

Ideal UI Setup (for Audrey)

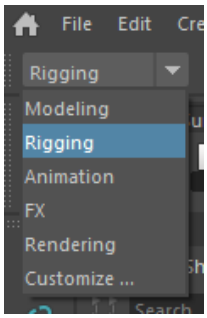


- All 4 viewports open and ready for use (helpful to toggle between)
- Script Editor open next to it
- On the side menu:
    - Channel box/layer editor (to view transforms and layers)
    - Attribute editor (for other attributes)
    - Tool settings (for axis orientation, transform tools, etc.)

Joints

We are going to have roughly one joint for each major bone in the body, or each part of the body that moves as a unit.

Make sure that you are working in the “rigging” workflow – change this in the dropdown menu at the top left of the GUI.

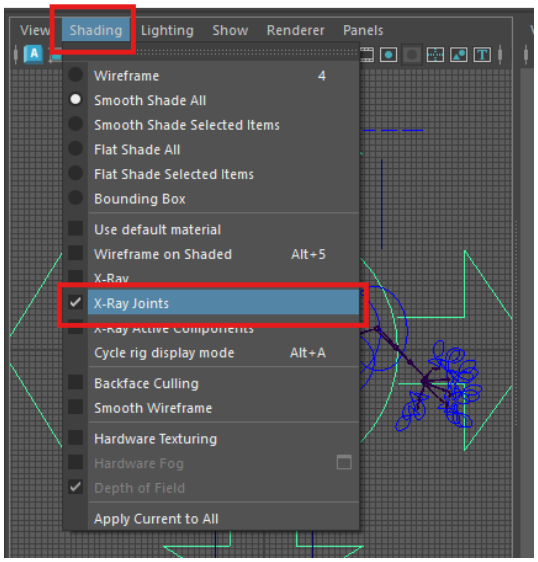


Creating Joints

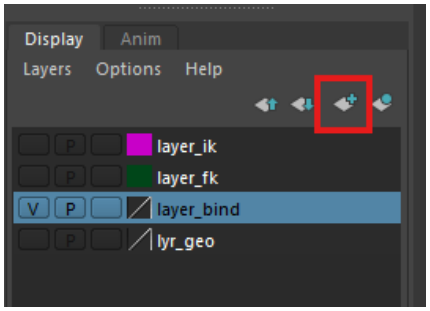
Viewports:
ALWAYS create joints in either the side or front viewpoint to make sure they're actually in line with the mesh.
Press space to see all viewports.

- to click into one, select it then press space again
- Helpful to have all 4 open

Make sure joints will show in the viewport: "Shading" → "X-Ray Joints"



Layers:
These joints will be the bind skeleton. Put them into another display layer so we can easily toggle them on and off: in the channel box, create a new layer for the bind skeleton with the "add layer" button.
To add objects to a layer, select the objects in the outliner, right click the layer, and press "add selected."



Hierarchy:
To create a joint, go to "skeleton" → "Create joint"

Create joints for each major bone in the body, for the LEFT SIDE ONLY. We will orient all the joint axes for the left side first, then mirror it all to the right to avoid unnecessarily doing this twice.

Name your joints well, and according to the same format for each, such as {jnt|nbj}_bodyPart_sideOfBody (ex. jnt_Hip_L or nbj_LowerEyelid_End_L).

- You will have lots of joints. Consistent naming helps keep track of them

For reference, these are common joints in a good hierarchy.

- Nbj ROOT
    - Center of gravity (COG)
        - Spine (for top half of rig) (multiple spine joints here)
            - Multiple neck joints
                - Head
                    - Nbj top of head
                    - Jaw
                        - Nbj chin
                        - Tongue joints (multiple, ending in a nbj)
            - L clav
                - L shoulder
                    - L elbow
                        - L wrist
                            - L thumb
                            - Nbj Palm
                                - L fingers (3 joints per finger)
        - Pelvis (for bottom half)
            - L hip
                - L knee
                    - L ankle
                        - L ball of foot
                            - L toe

The root joint should be at the bottom of your rig, oriented to the world. It's helpful in game engines to have one joint that everything else is grouped under.

Nbj = non-bind joint. These are joints that will not be bound to the mesh (skin).

The Finger Joints

- Put one joint at the wrist, a nbj for the palm, and three joints for each finger (just like how your actual finger is!)

The Eye Joints

We want to create the eye joints in the exact mathematical center of the eye. To do this:

- Isolate the eye mesh by clicking on the shape, then pressing Ctrl+1
- Create a joint for the eye under the head joint
- Create two locators and snap into place on either side of the eye (hold V while moving)
    - Freeze locator transforms
- Open the point constraint menu
    - Click one locator then shift-click the joint, and press apply
        - Joint should snap to locator
    - Click the other locator then shift-click the joint (which is on top of the first locator), and press apply again
        - Joint will snap to exact mathematical middle of both locators
- Locators can now be deleted
    - Note that they must be deleted to freeze transforms
- Add a nbj under this newly placed joint to tell the eye where to point:
    - Use v to lock the nbj to the middle of the front of the eye

To mirror one eye and create the other:

- Select the joint, go “Skeleton” → “Mirror Joints” and MIRROR ORIENTATION

Orienting Joints

We want to point (or “orient”) the axes of each joint so that each joint’s most natural movements occur along the axes, and that these movements don’t create gimbal lock: when two of the rotation axes line up with each other, and you can no longer rotate along these separately.

Preventing gimbal lock means looking at Maya’s axes hierarchy. Rotations occur in a hierarchy – rotating along the axis at the top of the hierarchy will rotate the other two axes, rotating the middle axis will rotate the one below it, and rotating the last will only rotate itself. Gimbal lock can occur if you rotate the middle axes so that the other two line up.

Prevent gimbal lock by making rotation along the middle axes the least common movement. Rotation along the highest hierarchical axis in the positive direction will be the most common movement (so that it rotates all other axes).

Maya writes its axes in the opposite order of their hierarchy – it writes them as xyz (the notation order), when the hierarchy, from highest to lowest, is zyx (the rotation order). Therefore, we want rotation in the +z direction to be the most common movement.

Here are some examples of axis directions:

- Orienting the spine:
    - Most common action: curling forward (rotation axis horizontally sideways) = z
    - Least common action: bending sideways (rotation axis forward) = x
    - Primary points up = y

- Orienting the jaw:

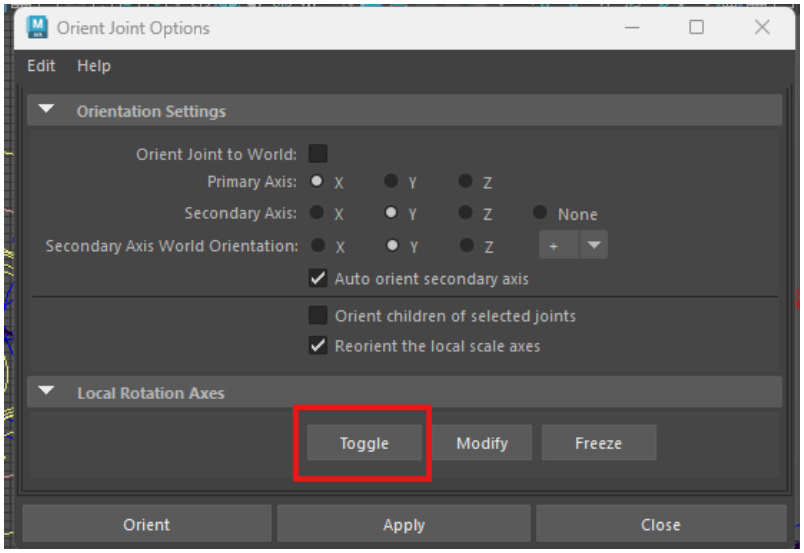
- Most common: opening (axis horizontal sideways) = +z
- Least common aka impossible: rotation around fwd pointing axis = x
- Primary points up = y

To orient a joint: “Skeleton” → “Orient Joint” options (window icon next to "orient joint")

- Turn OFF “orient joint to world” (this orients it along the world axes instead of along the joint)
- Turn OFF “orient children of selected joints”
  - We will orient the children manually
- Primary axis = axis pointing down the joint chain

Orienting joints tends to be trial and error: set your primary and secondary axes, hit apply, then rotate the joint along the most common movement and watch the Channel Box or Attribute editor to see if the rotation is what you expect. If not, switch your primary and/or secondary axes and test again.

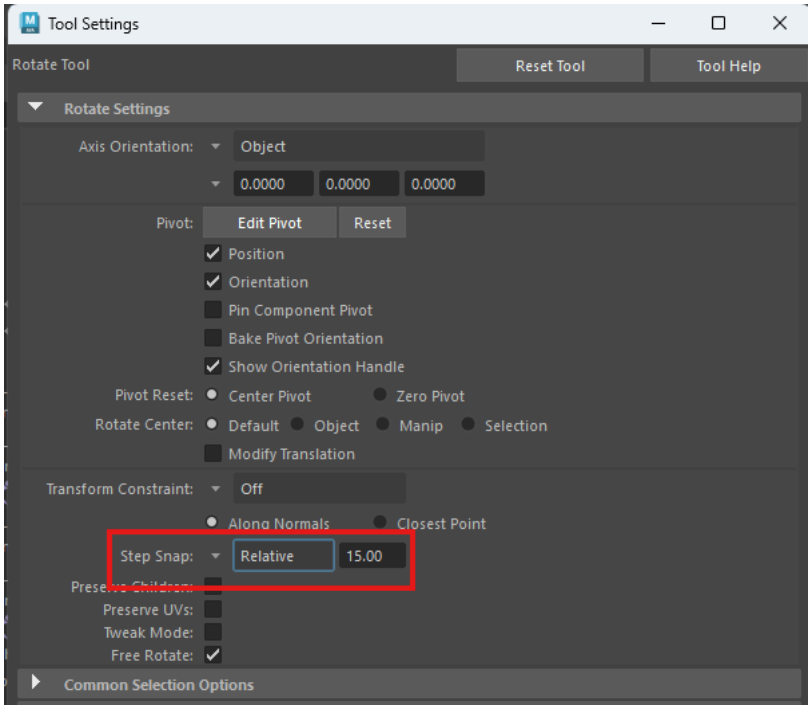
To view rotation axes of each joint: Press “Toggle” under “Local Rotation Axes” in the Orient Joint Options menu.



OR select the joint(s) of interest and go to “Display” → “Transform Display” → “Local Rotation Axes”

If the Orient Joint Options aren't giving you the joint directions you want, you can manually orient them axes:

- Under the orient joints menu: under "local rotation axes" click "modify"
- Modify the rotate tool to rotate in 15º steps so that you have cleanly angled axes
    - "Modify" → "Transformation Tools" → "Rotate Tool" Menu
    - Set "snap set" to relative and 15º
    - Check the rotation degrees under the attribute editor to make sure they're multiples of 90



Freezing Transforms

We ALWAYS want to freeze transforms on our oriented joints (really, on most objects we create) – aka setting the rotate and scale values to 0 (we can't do it to translate values because Maya uses translate values to help determine the relationship between parent and child joints.

- If an animator were to accidentally scale or rotate a joint out of place, frozen transforms make it easy to reset – just set everything back to 0 (for rotate) or 1 (for scale)

To freeze transforms:

1. Delete History: "Edit" → "Delete By Type" → "Delete History"
    a. Keeps file size small and prevents Maya from crashing
2. Freeze Transforms: "Modify" → "Freeze Transforms"
    a. If you only want to freeze certain transforms, you can go into the menu and select or deselect transforms of your choice

Mirroring Joints

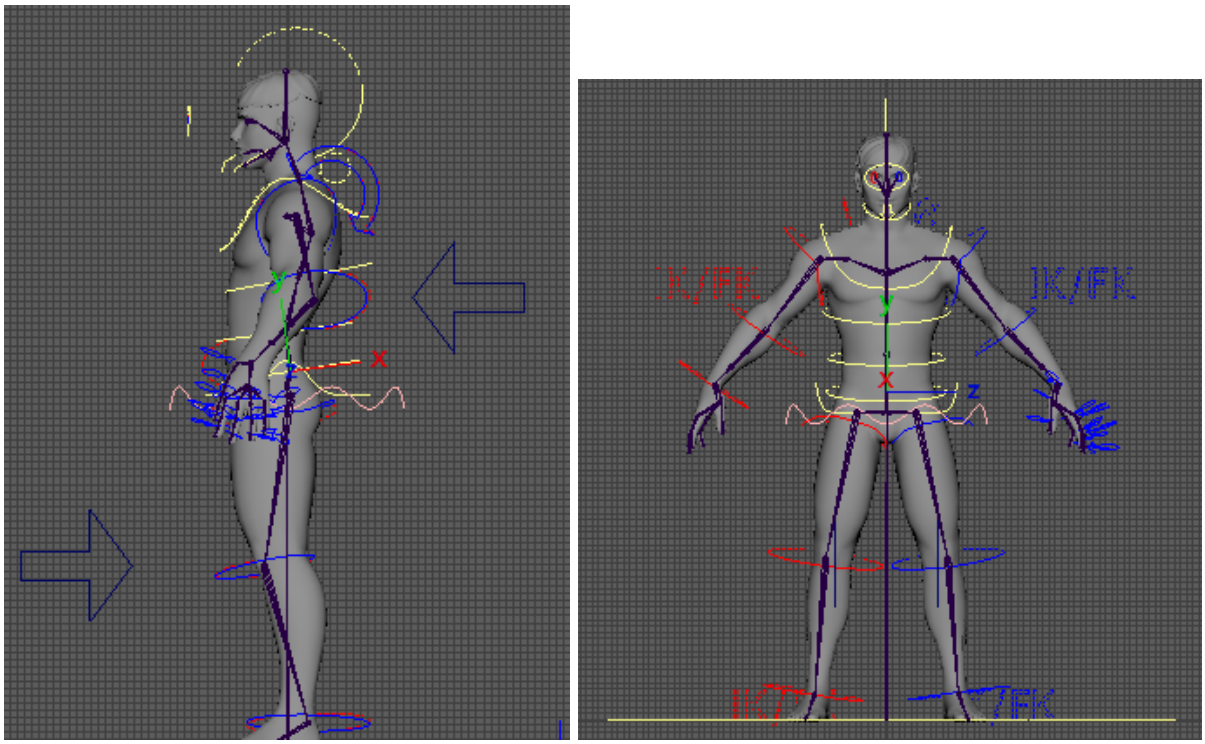
To mirror joints: “Skeleton” → “Mirror Joints”

In the mirror joints options, you’ll notice that you have mirror behavior or orientation. Mirroring behavior will, in short, also mirror the axes. Mirroring orientation will not.

- Mirror behavior for arms and legs so they flap in opposite directions
- Mirror orientation for eyes so they work together)

Controls

Joints are small, hard to reach, and hard to control precisely. Thus, we make separate shapes, called controls, which are easier for animators to reach and use, but will move/rotate/scale joints proportionally (we do this by constraining the joints to the control shapes). All these yellow, blue, and red things are controls:



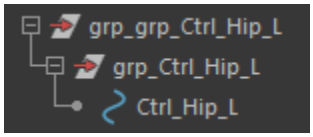
Creating Controls

We use NURBS curves for the joints. NURBS (Non-Uniform Rational B-Spline) curves are a type of Maya geometry whose most important property is that they don't appear in rendering.

> *Note:* The acronym for NURBs describes its mathematical representation. Non-uniform refers to the parameterization of the curve: non-uniform parameterization means each point along the curve is numbered proportionally to where it sits along the curve's length. Rational means the curve's equation is made of rational numbers, and B-spline means it can be represented as a piecewise polynomial with a parametric representation.

Double Grouping

Each control gets double-grouped. For example:



Animators will be moving the actual control when working, so we want to be able to freeze all transforms on the control itself (again, for redundancy – if someone makes a mistake while animating, it's easy to

move the control back to its original spot). However, we also want the control axes to be lined up with the axes of the joint it's controlling so that the joint moves how we configured it to move during all that orienting. Thus, we keep rotation values on the top group (the "group group") of the joint so that this group group's rotation axes line up with the joint. Since child objects have axes dictated by their parent objects, the control itself will have rotation axes in the right direction.
The middle group is there to provide extra flexibility for other transforms you may want to put on the control.

Controls should be easy to reach and manipulate, and ideally not overlap with other controls

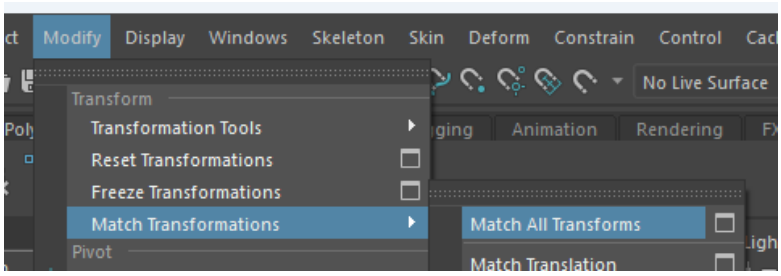
Also, note the naming convention! Largely following the same format as joints, but with "Ctrl" instead. Groups should be named with the prefix "grp_" followed by the top child object's name.

Have one control for each joint (NOT for any nbjs except for the root)

- Shape the control to where it makes sense
    - Ex. spine controls can be simple circles rotated along the joint axes
    - Ex. For the head, do a 3/4 circle around the top of the head
    - Ex. For the jaw, make it jaw-shaped and fit under the jaw
- Skip the eyes for now. We'll cover that later.
- Put all the fingers in a big double group with its pivot point at the wrist (this will be used later for FK/IK stuff)

To make a simple circle control (for example, for the knee).

- "Create" → "NURBS Primitives" → "Circle"
- "Ctrl+g" twice to group it with itself
- Snap the top group to the joint: Click the top group, shift click the joint, then go "Modify" → "Match Transformations" → "Match All Transformations"

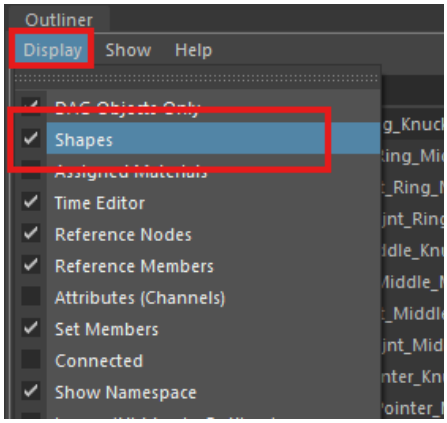


To make a custom shaped control:

- Use NURBS circles for round things
- Use EP curves for straight lines
    - "Create" → "Curves Tools" → "EP Curve Tool" Menu
    - Make sure Curve Degree is "1 Linear" if you want straight lines
    - For clean lines: press X while creating points to snap them to the grid

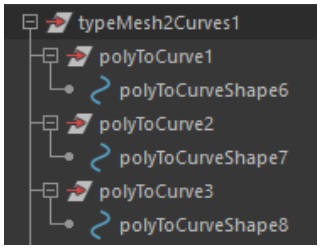
To combine multiple shapes into one:

In the Outliner, show shapes: “Display” → Check “Shapes”

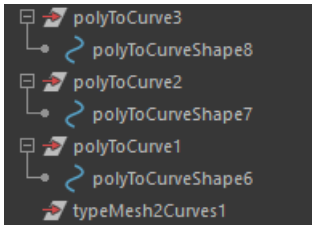


Each curve is now a group with a Shape object under it.

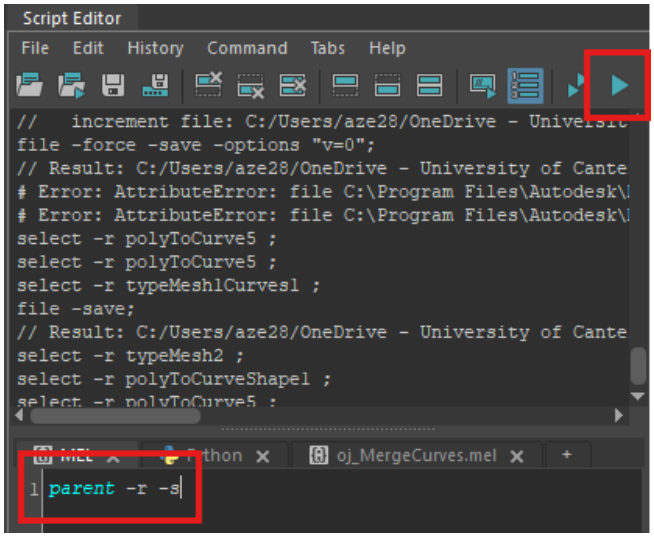
> *Note:* This is because curves are made up of 2 DAG Objects (what the first checkbox in the Display menu is talking about): transforms and shapes. By default, the outliner only shows transforms, which control your object’s translation, rotation, and scale (sound familiar?). The shape stores information about the object’s control vertices, lines and polygons. The shapes are children of the transforms because you want anything you do to the transform also!



We want to move all these shapes under one transform, which is going to require a bit of scripting. First, middle-mouse drag to unparent each curve out from the main transform (in the picture, this is “typeMesh2Curves1”)

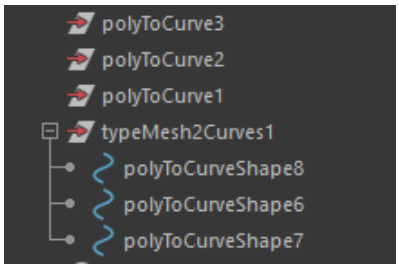


Open the script editor. One by one, select each shape, then the main parent transform. Be sure to delete history and freeze transforms for each shape, and the parent transform before you do this. In the MEL terminal of the script editor, type “parent -r -s,” and press run:



Note: Curious about the flags following the parent command? "-r": relative, preserve local object transformations (this is why you should freeze transforms before running this command!). "-s": shape, allows shapes to be parented under the transform (by default, the command assumes the object is a transform)

At the end, you should have something like this:



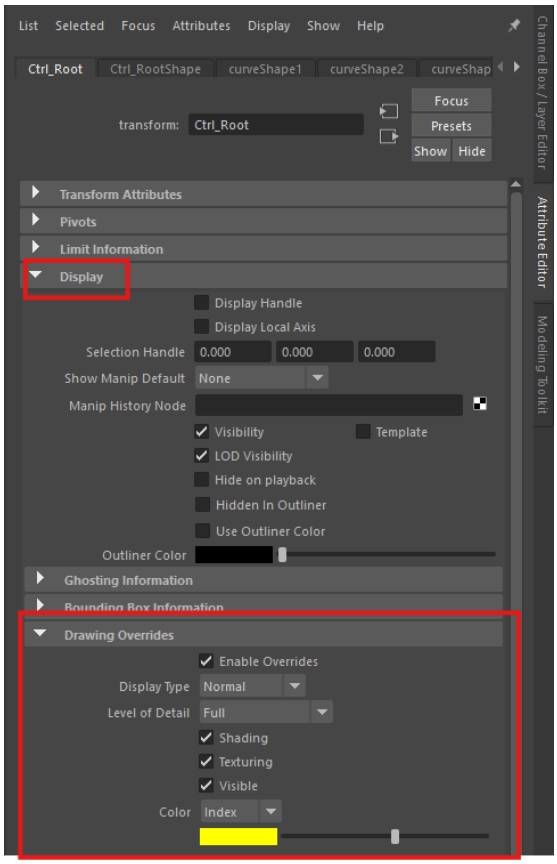
You can now delete the empty transforms (in the picture, these are the top 3 objects). When you select one of your sub-shapes in the viewport, it should select all of them! Remember to double group this control, just like with all the others.

Color Coding

Controls are usually color-coded by which side of the body they control for extra visual clarity. Usually, this is:

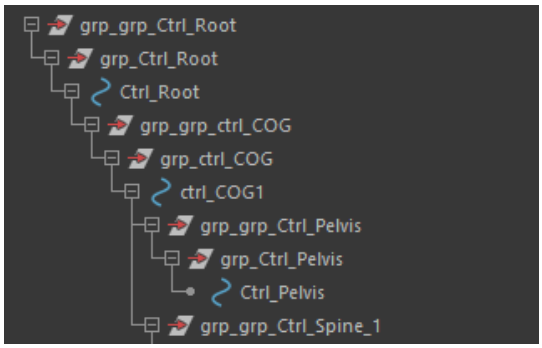
- Red = right
- Blue = left
- Yellow = center

To change the color of a transform object, use the Attribute Editor. Under “Display” then “Drawing Overrides”, check “Enable Overrides.” Use the slider to choose a color.



Hierarchy

After all your controls are done, put them into a separate hierarchy from the bind joints, but with the same hierarchy structure. Make sure that the group group of your each child control is under the CURVE of the parent control. For example:

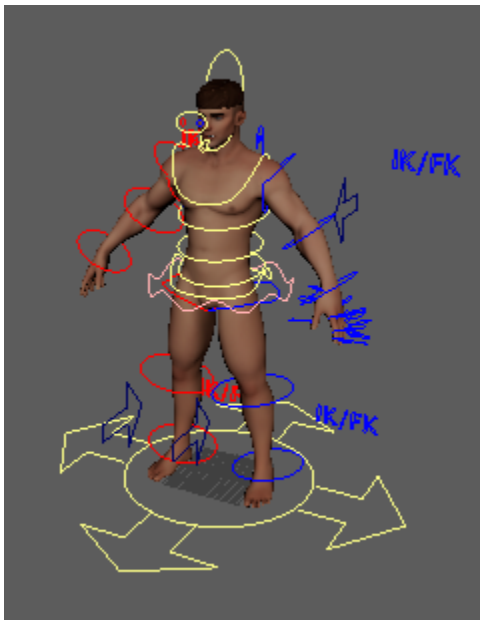


Constraining Controls

- Have it control a joint with parent constraint:
    - "Constrain" → "Parent" menu → click the controller then shift-click the joint → "Apply"
    - Make sure “Maintain offset” is UNCHECKED
- After constraining a joint and control pair, move the control around to make sure the rig moves accordingly and as expected
- If your joint is popping out of place when you apply the constraint, you need to re-orient it

RENAME YOUR CONSTRAINTS TO HAVE THE PREFIX “cnst_” or anything that’s not “jnt_”

Example Controls Setup

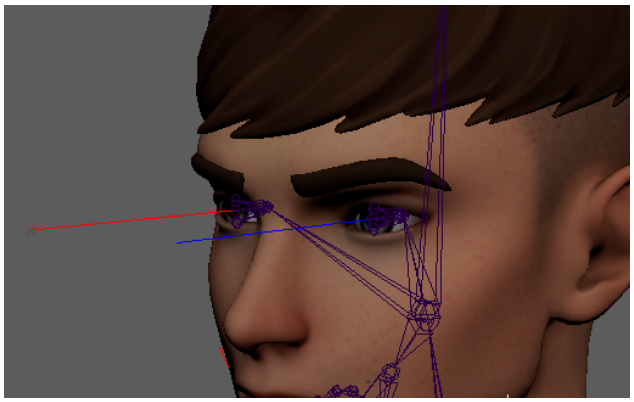


Eye Controls (Aim and Look-At Constraints)

Making the Controls:
For the eyes, we'll have different controls to aim the eyes and make them look at things.

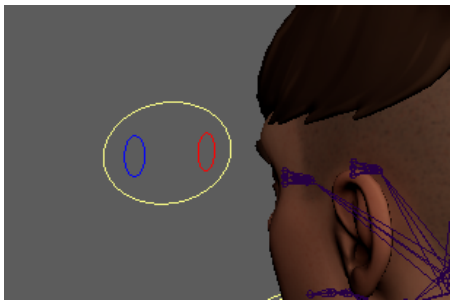
The aim controls:

- Two straight lines sticking out of the eye
- Match transforms to the joint, not the nbj

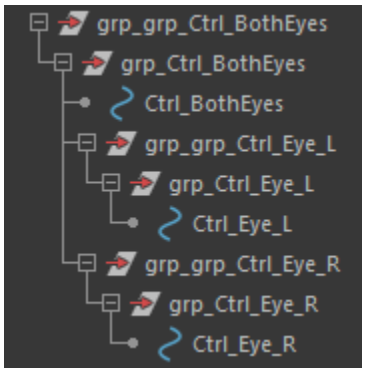


The look-at controls:

- We want one control for each eye alongside a control that can move both at once.

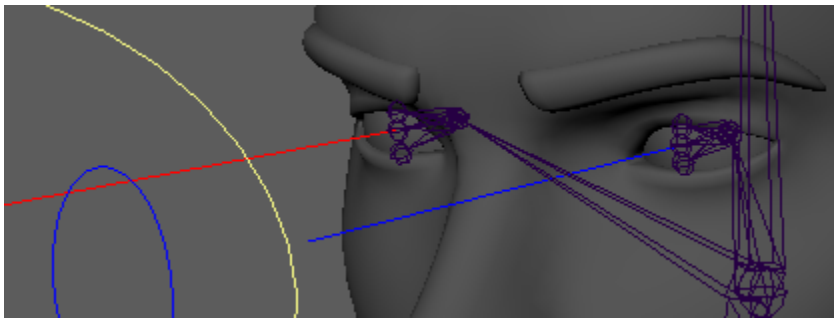


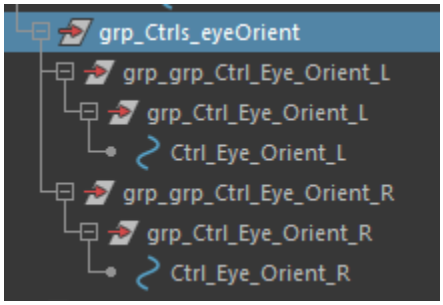
- Put these in one double group:



FK orient controls:

- 1 line sticking out of each eye:





Put ALL the eye controls under one group: “grp_Eye_Ctrls”

Adding Blink Attributes and a Control Center:

For animators to control blinks:

- Select ctrl_head
- Go “Modify” → “Add Attribute”
- Add 2 Float attributes: “L_blink” and “R_blink” (min = 0, max = 1, default = 0)
- Add a third float attribute: “eye_mode” (min = 0, max = 1, default = 0)

Apply the LookAt Constraint:

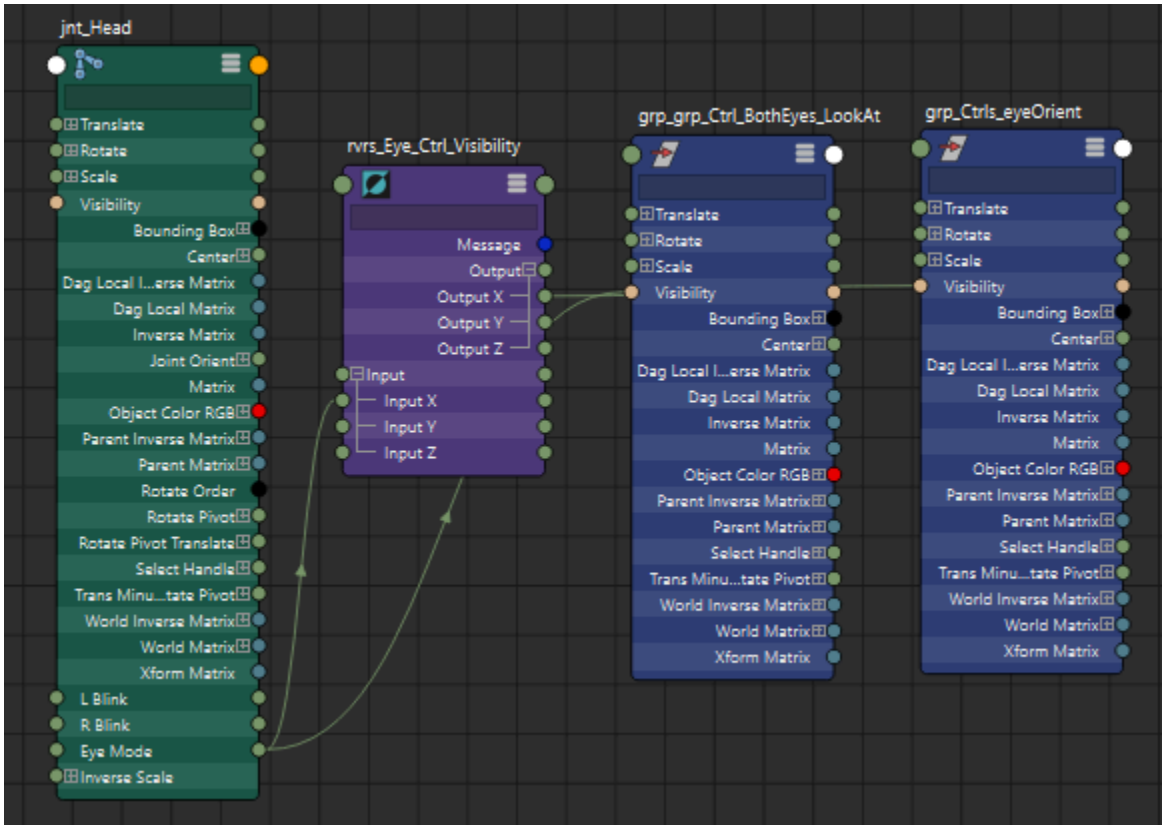
- Select “ctrl_L_LookAt” then “jnt_L_eye”, and apply an aim constraint) MAINTAIN OFFSET ON)
- Do the same with the right
- Do NOT constrain the master LookAt control

Apply the FK Orient Systems:

- Orient constrain the FK controls to the eye joint (maintain offset OFF)

Visibility between the FK (Direct) Controls and the LookAt controls:

- Use the node editor to wire “eye_mode” to the visibility of LookAt and Orient
- 0 for orient, 1 for LookAt



Constraint Weight Switch:

- Use the node editor to connect “eye_mode” to constraint weights on “jnt_L_eye”

Eyelid Setup (with Set Driven Keys):

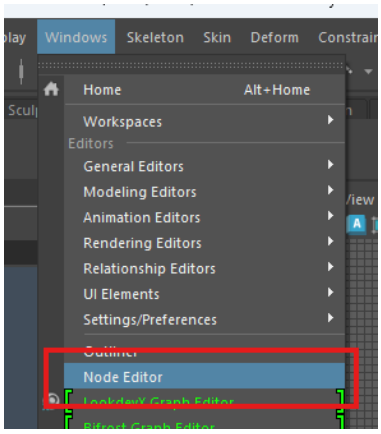
- “Animation Menu” → “Key” → “Set Driven Key” → “Set…”
- Load “ctrl_head” as Driver, select the “L_blink” attribute
- Load left eyelid joints as “Driven” and select their primary rotation axis (usually X)
- With “L_blink”=0, leave eyelids in their default open pose and click “Key”
    - You have to click “Key” for each eyelid separately!
- Set “L_blink” = 1, rotate the eyelid joints so they meet in the middle, and click “Key”
- Repeat for right eye

Forearm Twist

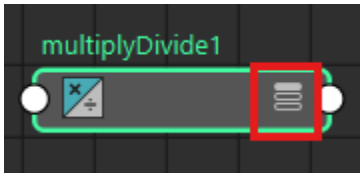
We are making a separate forearm twist mechanism because your forearm (and everything below it, aka your wrist and hand) can twist without your elbow twisting. Therefore, we make two extra forearm joints, and when you rotate the wrist, the lower forearm joint rotates ⅔ of that amount and the upper forewarm joint rotates ⅓ of that amount.

Making the Joints:
Duplicate the elbow joint twice. Move it down the forearm joint (this should be easy as its axes should be oriented down the joint already!); put one ⅓ of the way down, and the other ⅔ of the way down. Name them “jnt_ForearmTwist_1_L” and “jnt_ForearmTwist_2_L”. Now mirror them to the right (temporarily parent them under your root joint before mirroring so that they mirror exactly over).

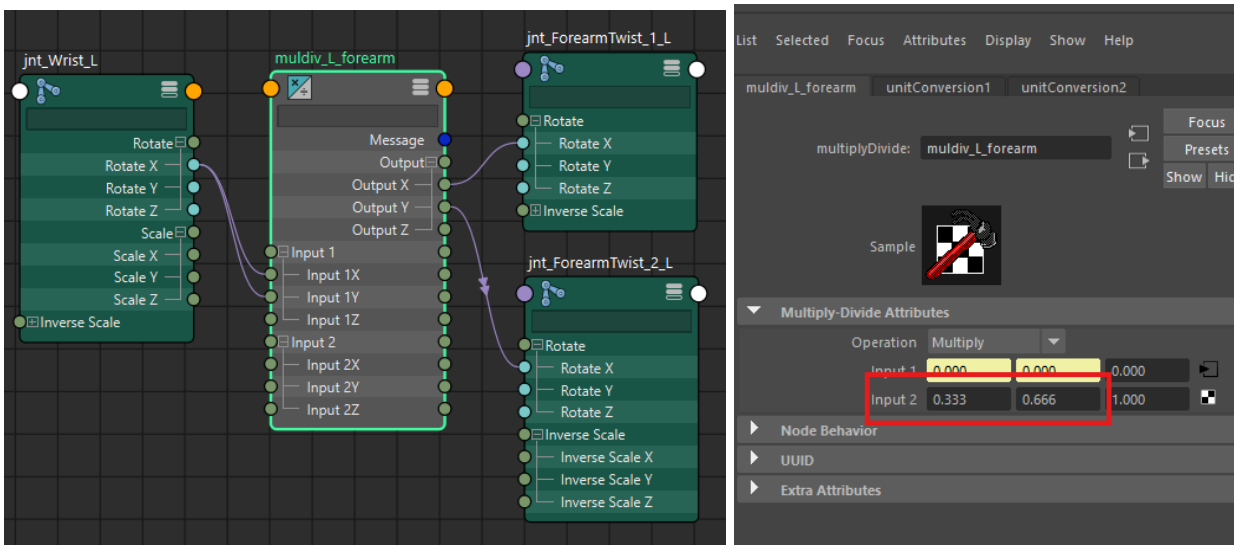
Constraining the Rotation to the Wrist Rotation:
We are going to use the node editor to do this. Open that.



Bring in your two new forearm joints and your wrist joint by selecting them, and middle-mouse-click dragging them into the editor. Create a multiply-divide joint (press “tab” to add a new node, and search for multiply divide), and name it “multiplyDivide1”. To expand the node and see its attributes, press the stacked menu button on the right side of the node.



Connect “Rotate X” and “Rotate Y” under the wrist joint to “input 1X” and input “1Y” of the multiply divide joint by clicking and dragging from the input to the output. Then, in the attribute editor of the multiply divide joint, set Input 2X to 0.33 and Input 2Y to 0.66. Connect Output X of multiply divide into Rotate X of forearm 1, and Output Y into Rotate X of forearm 2.



Check your work by rotating the wrist! The forearms should rotate smoothly accordingly.

IK/FK (Inverse Kinematics and Forward Kinematics)

There are two ways to control joints while animating: Inverse Kinematics (IK) and Forward Kinematics (FK). FK is the more intuitive one: it means moving the parent joint to control the child joint. However, sometimes animators like to use IK, or Inverse Kinematics, in which moving the child joint also moves some of its parents – for example, moving the hand determines how the elbow should bend. This is helpful specifically for limbs: FK allows for natural swinging movements (like a character's arms when they walk), and IK is useful when you want the feet/hands placed somewhere specific, and the elbow to bend accordingly.

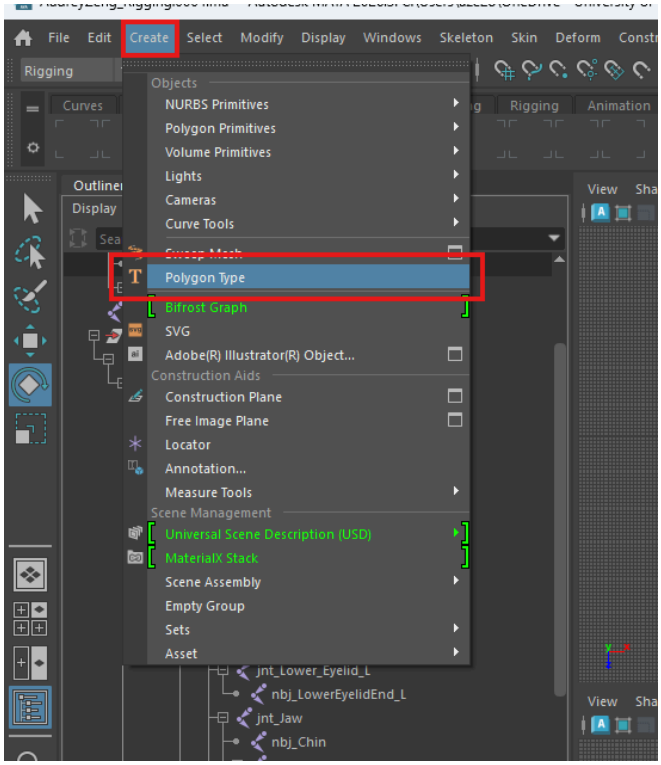
The controls we have made thus far are all FK controls. We will add one more control to accommodate IK: a lollipop handle on each wrist called "ctrl_{L|R}_hand_IK"

The following takes you through how we are going to switch between the two.

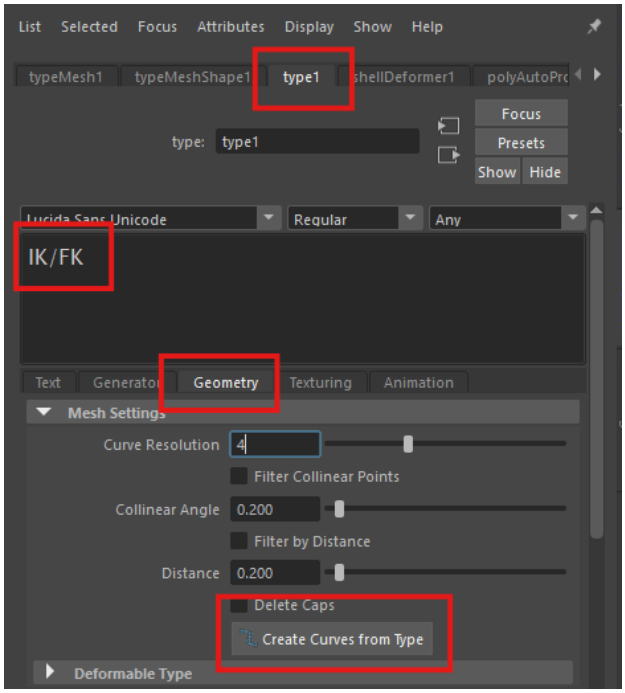
The Switch Control

We want a control shaped as text that says "IK/FK," which the animator can change the properties of to easily switch between IK and FK controls.

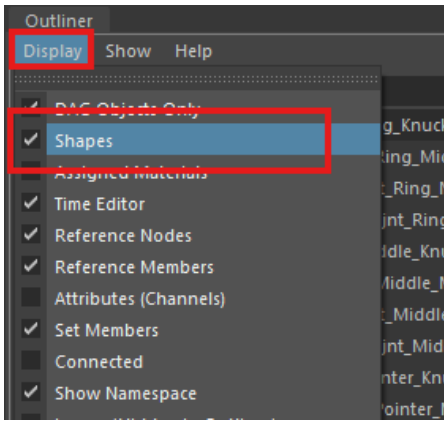
First, we make the text as a polygon type: Under the top menu, go to "Create" then "Polygon Type"



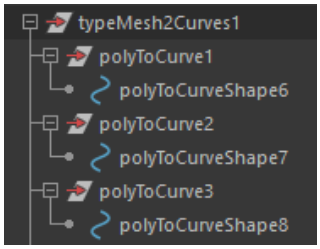
In the Attribute Editor, click on the middle tab of the top half of the editor to change the shape to your desired text. Then, go to the “geometry” tab on the bottom half, and click “Create Curves from Type” to generate curves. You can now delete the original polygon type.



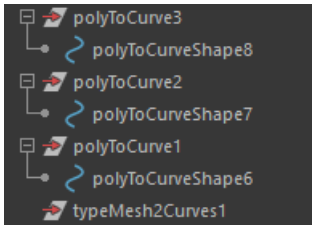
Now, if you click on the text, you’ll notice that each letter is its own separate curve. We want to merge them all into 1, which we do by following the steps to merge shapes in the Controls section above:
In the Outliner, show shapes: “Display” → Check “Shapes”



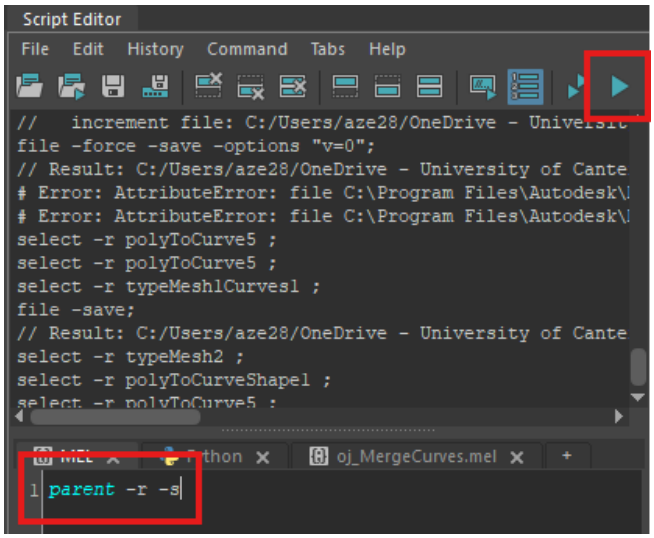
You'll now notice that each curve is now a group with a Shape object under it.



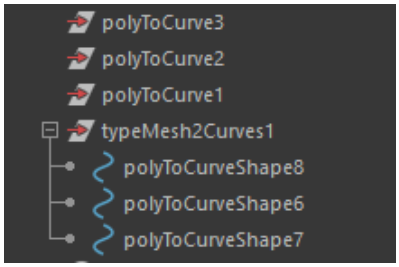
We want to move all these shapes under one transform, which is going to require a bit of scripting. First, middle-mouse drag to unparent each curve out from the main transform (in the picture, this is "typeMesh2Curves1")



Open the script editor. One by one, select each shape, then the main parent transform. Be sure to delete history and freeze transforms for each shape, and the parent transform before you do this. In the MEL terminal of the script editor, type "parent -r -s," and press run:



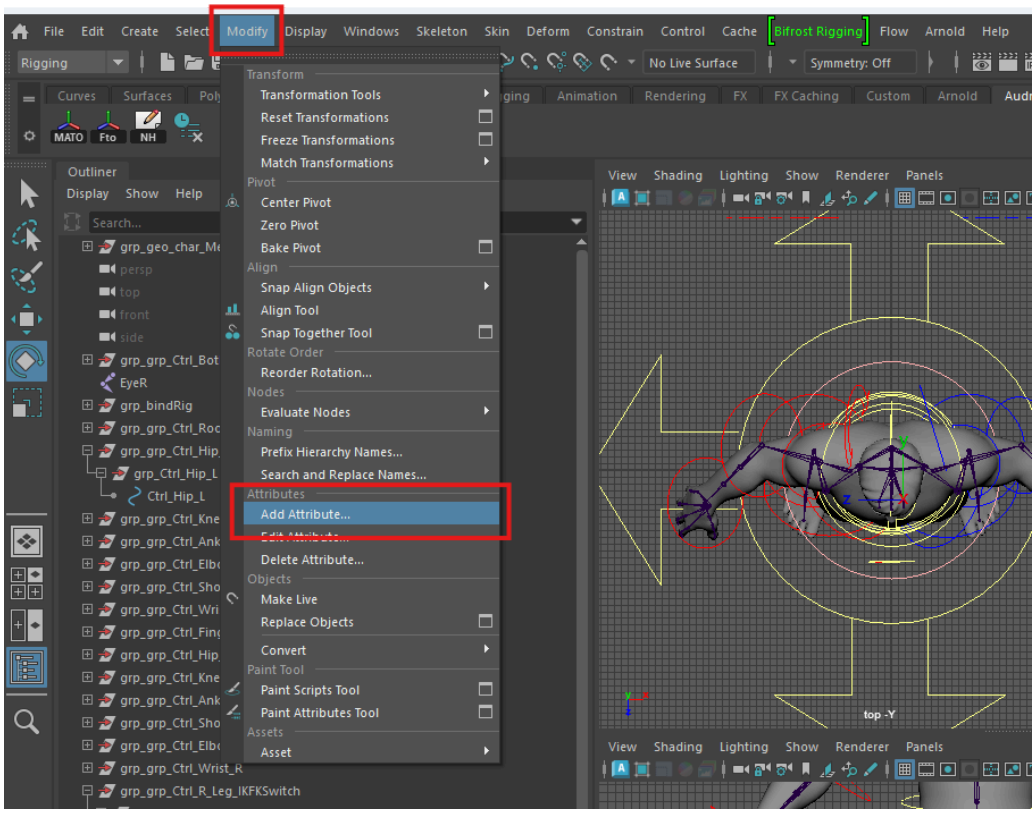
At the end, you should have something like this:



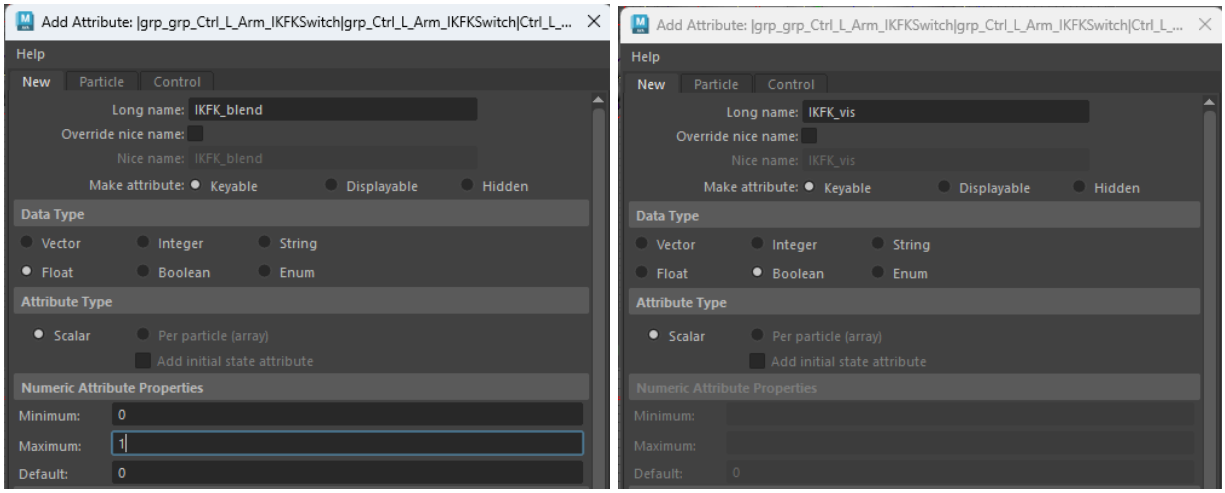
You can now delete the empty transforms (in the picture, these are the top 3 objects). When you select the text in your viewer, it should select all the letters together! Remember to double group this control, just like with all the others.

Make 4 copies of this: we'll need one for each arm and leg. Move each control to behind its respective limb.

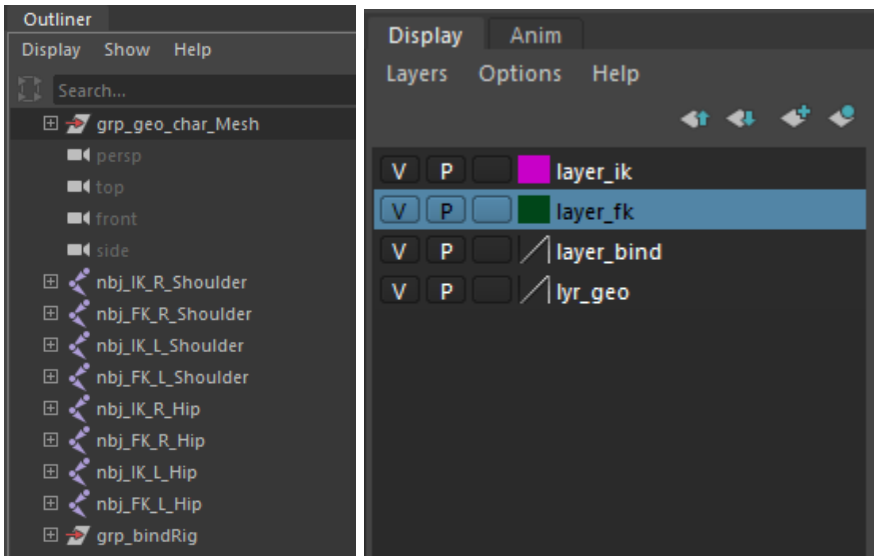
Now, we add attributes that will let us switch between IK and FK:



Add two: one called IKFK_blend (type=float, minimum=0, max=1, default=0), and another called IKFK_vis (type=boolean)



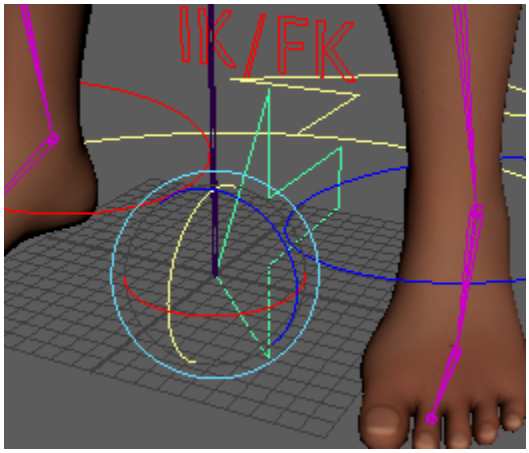
Now, duplicate the left and right arm joints (shoulder, elbow, and wrist only) and the left and right leg joints (hip, knee, ankle, ball, and toe) TWICE. Move them out of the hierarchy so they are separate in the outliner. Name the first copy of each joint with the prefix “nbj_IK” and the second copy of each joint with the prefix “nbj_FK”. Create two display layers called “layer_IK” and “layer_FK”, giving them distinct colors, and add the joint copies to each accordingly. Your outliner and display layer window should now look like this:



Note: We require separate skeletons for bind joints, FK joints, and IK joints so as to not confuse Maya. Maya does not like when you try to control one joint with multiple controls. Thus, we put our IK and FK controls on separate skeletons. We will constrain the bind skeleton to them so that when we change our IKFK_blend attribute, the bind skeleton will constrain to the joints with IK controls (value=0), the joints with FK controls (value=1), or switch smoothly between them.

The Aim (Pole Vector Control)

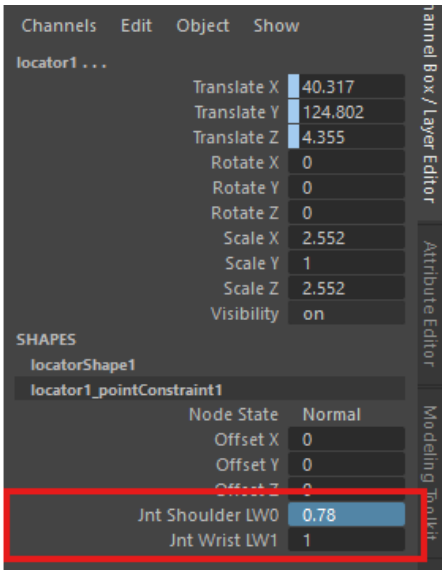
Using the Bezier curve tool, draw an arrow on the floor (press X while making nodes to snap them to the grid). Move the pivot point to the tip of the arrow, and rotate it so that it's sideways and pointing forwards. Scale it up to be a good size, then freeze transforms. Name it "Ctrl_L_Elbow_Aim", and double group it, named accordingly.



Now, we want to snap this arrow so it is *exactly* in the plane of the elbow. We will use a locator to do this, similar to creating the eye controls.
We'll do the left elbow first, then duplicate it to the rest of the 3 IK/FK joints.

- Create a locator
- Shift click the left shoulder joint, the left wrist joint, then the locator
- Apply a Point Constraint with "Maintain Offset" off.

- In the Channel editor, under the pointConstraint shape, adjust the shoulder and wrist values until the locator is lined up with the elbow in the side viewport (don't worry about making it too perfect).

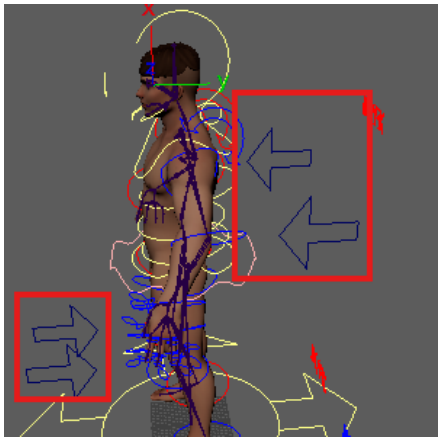


- Select the elbow joint (ex. “nbj_IK_L_elbow”), then the locator, then apply an aim constraint (“Constrain” → “Aim Constraint” with Maintain Offset unchecked)
    - MAKE SURE AXES ARE SET TO OBJECT IN TOOL SETTINGS
- Delete the constraints under the locator

- Move the locator straight backwards ALONG THE ALIGNED AXES until it is a healthy distance behind the model’s arm.
- Move the arrow so that its tip is directly on the locator (press V to snap it to the locator).
- Repeat for the other arm
    - Mirror the locator: group it, duplicate it, and change the duplicated group’s Scale X value to -1

- Repeat for the knees, but put the arrows in front of the knees (still pointing at the model)

It should look like this:



> *Note:* if the arrow is not in the “Pole Vector Plane,” or the plane of the 2D triangle formed by the shoulder, elbow, and wrist joint, Maya gets mad when you activate the IK controls. More specifically, it will look for an aim constraint to point the elbow towards. If your constraint is not in line with the joints (even if it’s only off by a little bit), the entire arm will “pop,” or twist out of place.

Constraints: Visibility Controls

We want the “IKFK blend” attribute on the IK/FK switch to control visibility of each control group. If the value is 0, only the IK controls should show; if it’s 1, only the FK controls should show (this would be flipped if you named it “FK/IK”

Use the Node Editor to do this (“Windows” → “Node Editor”)

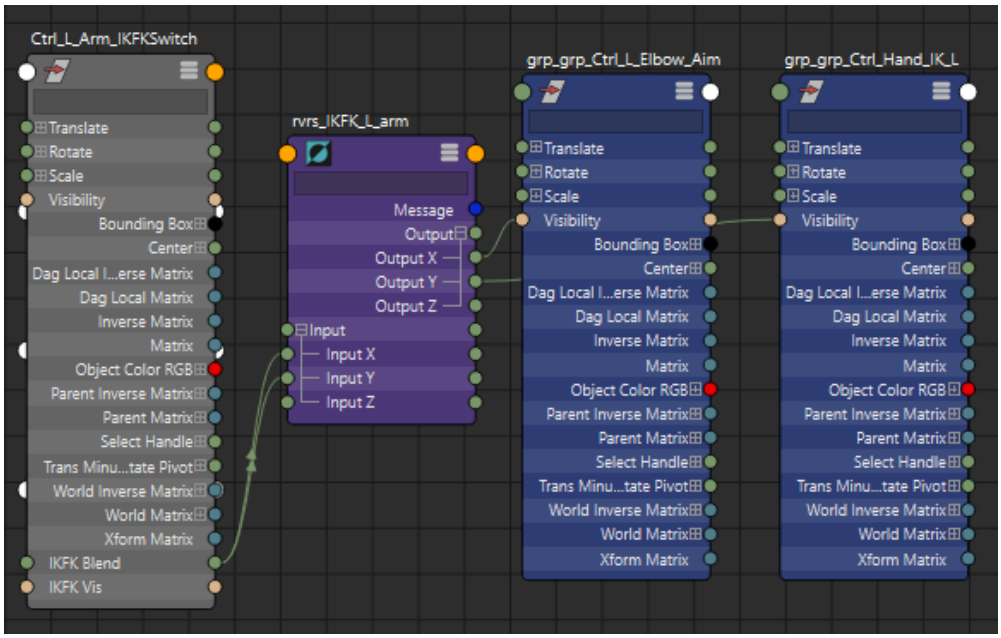
This will walk you through steps for the left arm and can be repeated for all other limbs.

IK Visibility:

To the node editor, bring in “Ctrl_L_Arm_IKFKSwitch”, “grp_grp_Ctrl_L_Elbow_Aim,” and “grp_grp_Ctrl_Hand_IK_L”

Press tab to create a “reverse” node, name it “rvrs_IKFK_L_arm”

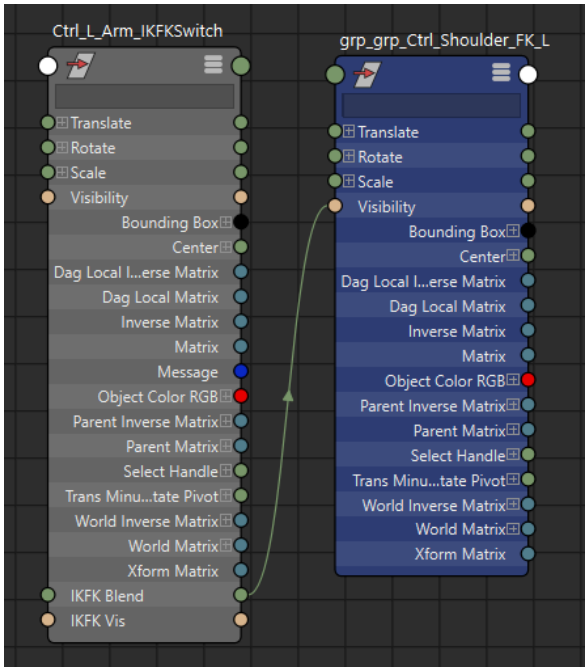
From “Ctrl_L_Arm_IKFKSwitch:” connect “IKFK_blend” to 2 inputs of the reverse node
Connect the corresponding reverse outputs to “Visibility” on the group groups for the elbow aim and IK hand control.
It should look like this at the end:



Now you can clear the node editor

FK Visibility:
Into the node editor, bring in “Ctrl_L_Arm_IKFKSwitch” and “grp_grp_Ctrl_Shoulder_FK_L” (no need to bring in the elbow or wrist as these are parented under the shoulder)
Connected “IKFK_blend” on the switch to “Visibility” on the shoulder

Should look like this:



Constraints: Connecting Controls and Joints

FK:

Use Orient Constraints to connect the FK controls with the nbj FK skeleton

- Select the control, then shift select the nbj FK joint
- Shoulder: “Constrain” → “Parent Constrain” – make sure “Maintain Offset” is OFF
- Elbow and Wrist: Orient Constrain

  Note: Orient constraints only allow the controls to rotate the joints, no translation. Arm joints only rotate – no need to give the animator unnecessary degrees of freedom

  We parent constrain to allow the root control to move the FK controls

IK:

Create an IK handle (“ikHandle_L_arm”):

- “Skeleton” → “Create IK Handle” Settings → make sure “Current Solver” is “Rotate-Plane Solver”
  - Use RP for limbs: RP calculates rotations in 3D, SC only does so in 2D
- To create the handle: shift-click the IK shoulder joint then the IK wrist joint
  - For legs, only connect handle from hip to ankle (will connect the foot with reverse foot)
  - A line connecting the shoulder and wrist should appear

Pole Vector Constraint:

- Shift select elbow aim control (“ctrl_L_elbow_aim”) then the IK handle
- “Constrain” → “Pole Vector”
  - MAKE SURE THE ELBOW JOINT DOESN’T MOVE/”POP”

For arms: Parent the IK handle under the wrist **control**
For legs: Will deal with the handle when we do the reverse foot

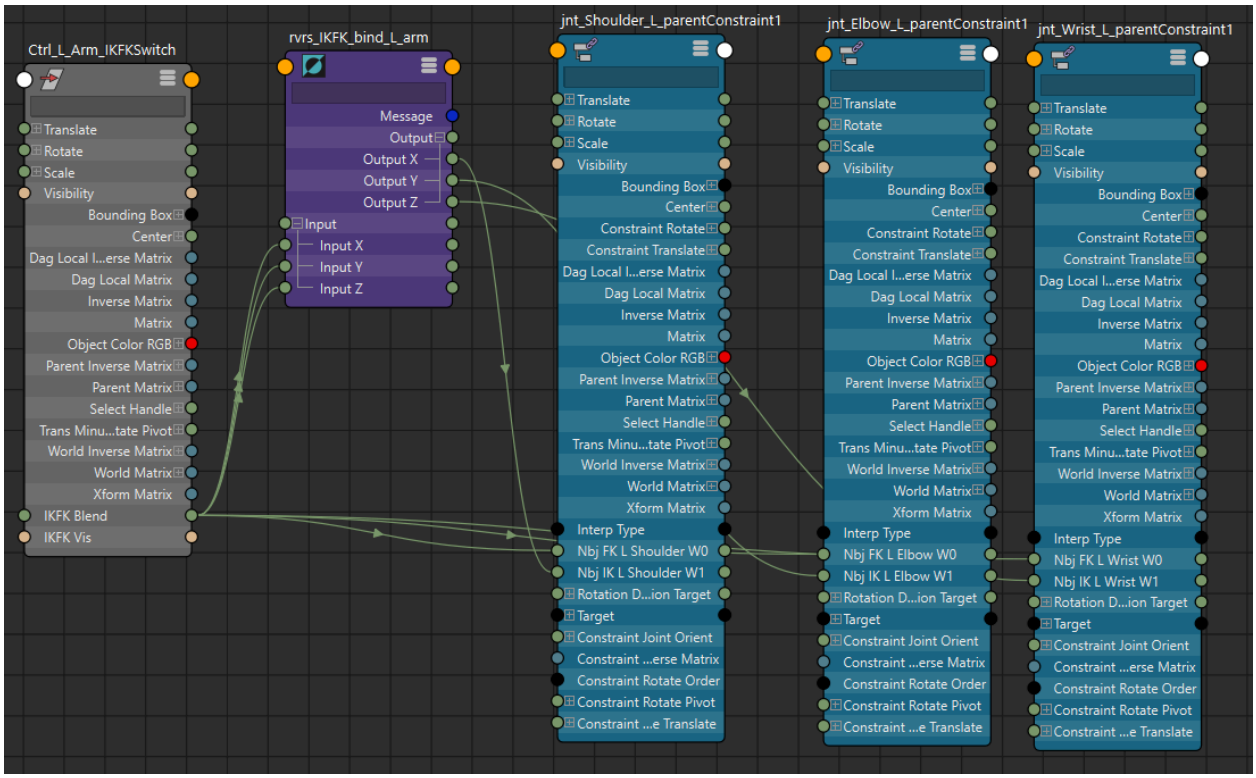
Constraining to Bind Skeleton:
We want the value of the "IKFK_blend" attribute to control how much weight each control system has

First, connect each skeleton to the bind skeleton:

- First, Parent Constrain the bind shoulder to the FK shoulder (select the FK first, then the bind)
- Then parent constrain the IK shoulder to the bind shoulder (select IK then bind)
- Repeat for elbow, wrist, and other limbs – including all joints for leg

Bind the Constraints:

- Using the node editor, connect the "IKFK_Blend" attribute to the FK and IK weights of each parent constraint on the bind skeleton (routing IK through a reverse node – or FK if you've named it FKIK)
- Should look like this:



Connecting the hand to the rest of the arm for FKIK:

- Want to connect the hand fish controls to the wrist control
- Group all your finger controls together twice
    - Make sure to change the pivot point to the wrist: hold d to access the pivot point, move it to the wrist (can hold "v" while doing so to snap it to other points)

- Parent constrain the topmost finger control group to the bind wrist joint

Connecting the shoulders/hips to the bind skeleton (so the IK skeletons move with the COG joint)

- Parent constrain the IK shoulders/hips to the next parent bind skeleton joint (either clavs or pelvis)

Testing:

Do this *after* doing the reverse foot (so you can test FK/IK controls for the legs too)

1. Set IKFK_blend to 0. The IK controls should be invisible, and the bind skeleton should follow the FK controls perfectly.
2. Set IKFK_blend to 1. The bind skeleton should follow the IK skeleton when you move either the IK wrist control or the pole vector.
3. Slide IKFK_blend between 0 and 1: the bind skeleton should transition smoothly.

Reverse Foot

We will create a heel, toe, ball, and ankle joint for the foot in a hierarchy that's the reverse of our bind skeleton. This is to help animate "rolling" actions of feet (ex. When you jump, walk, etc.)

This is the IK foot control!

Joints

- Place a new joint below and behind the left ankle for the heel ("nbj_rvrsFoot_L_Heel")
- Place new Toe, Ball, and Ankle joints exactly on top of the bind skeleton joints (press "v" to snap them exactly)
    - Name then with the prefixes "nbj_rvrsFoot_L_" then the joint name
- Orient the heel (most common movement along +z)
- Mirror the joint to the right (mirror behavior)
    - "Skeleton" → "Mirror Joints" Options → Check that "Mirror function" is set to "Behavior"
- Add both to a new layer

Controls

- Make a control from a NURBS Circle that looks like a footprint ("ctrl_L_Foot")
    - Remember to double group it!!
- Match transforms of the grp grp to the Heel
    - Make sure that the grp grp has all transforms zeroed except rotations
    - & the grp and control have all transforms zeroed
- Duplicate the control to the right side:
    - Duplicate the grp grp and rename everything to _R_
    - Unparent the control
    - Snap the grp grp to the R heel
    - Group the control, and set the scale to -1 so it goes to the right foot
    - Reparent the control into the group group

Constraints

Add IKSC (single chain) solvers to the foot:
Use SC because we don't need rotations like in the arms

- "Skeleton" → "Create IK Handle" Options
- Change Current solver to "Single-Chain Solver"
    - Click the IK Ankle, then shift click the IK ball ("ikHandle_L_Ball")
    - Press "Y" to repeat last tool:
        - Click IK ball, then IK toe ("ikHandle_L_Toe")

Parenting the IK handles:

- Parent Leg IK handle (“ikHandle_L_Leg”) under rvrsFoot ankle
- Parent Ball IK handle (“ikHandle_L_Ball”) under rvrsFoot ball
- Parent Toe IK handle (“ikHandle_L_Toe”) under rvrsFoot toe

Attributes and Wiring:

- Select the IK foot control, go to “Modify” → “Add Attribute”
    - Add the following as Float attributes (can leave min/max blank for unlimited rotation, or set rotation angle limits)
        - Ball_Roll
        - Toe_Roll
        - Toe_Pivot
- Open the connection editor: “Windows” → “General Editors” → “Connection Editor”
    - Select “ctrl_L_foot” and click “Reload Left”
    - Ball Roll
        - Select rvrsFoot ball joint, click “reload right”
        - Click “Ball_Roll” on left
        - Click “Rotate <axis>” on the right for the axis pointing sideways
    - Toe Roll
        - Select rvrsFoot toe joint, click “reload right”
        - Click “Toe_Roll” on left
        - Click “Rotate <axis>” on right
    - Toe Pivot (“squish”)
        - Keep rvrsFoot toe on right
        - Click “Toe_Pivot” on left
        - Click rotate <up/down axis>”
        - (animator can twist heel left/right while grinding toe into floor)

Final Parenting:

- Parent the reverse foot root joint (rvrsFoot heel) under the IK foot ctrl (ctrl_L_foot)

Cleanliness

Hide reverse foot joints and IK handles!

- Option 1: hide with Ctrl+H
- Option 2: put them in a “Do_Not_Touch” layer with visibility off

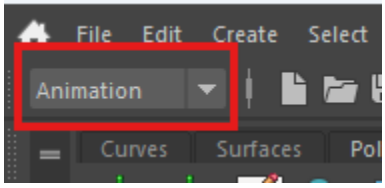
Set Driven Keys

Rather than animating an object over time, a set-driven key will animate an object based on the attributes of another object. For example, animating a door to open when a car comes near it, or setting the FK controls to disappear for a certain "FKIK blend" value.

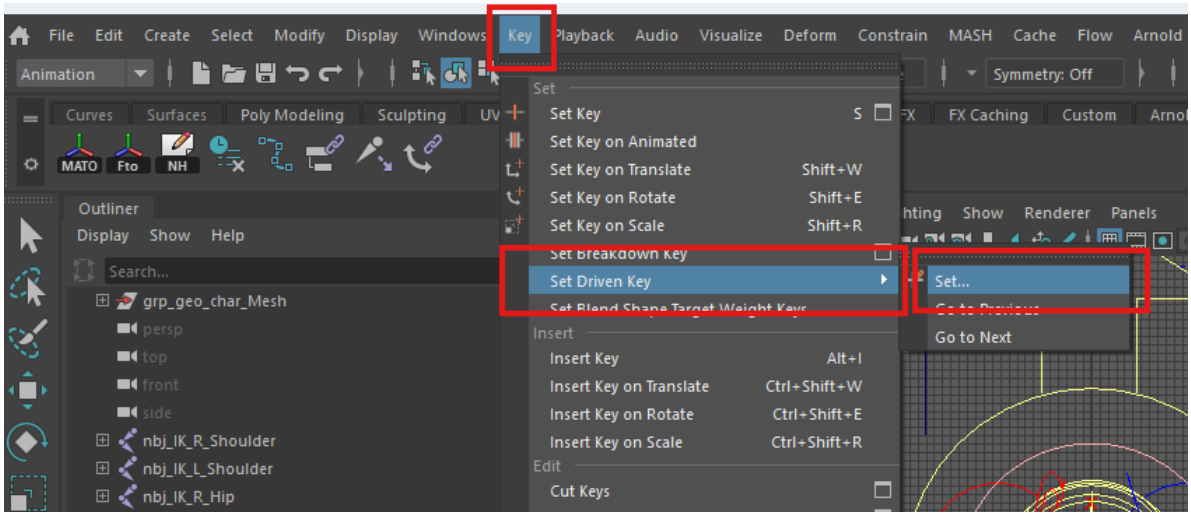
The "driver" is the object whose attribute causes the other to change. The "driven" is the object which changes based on the driver's attributes.

Creating a set-driven key:

- Make sure you're working in "animation" mode (top left of Maya GUI)

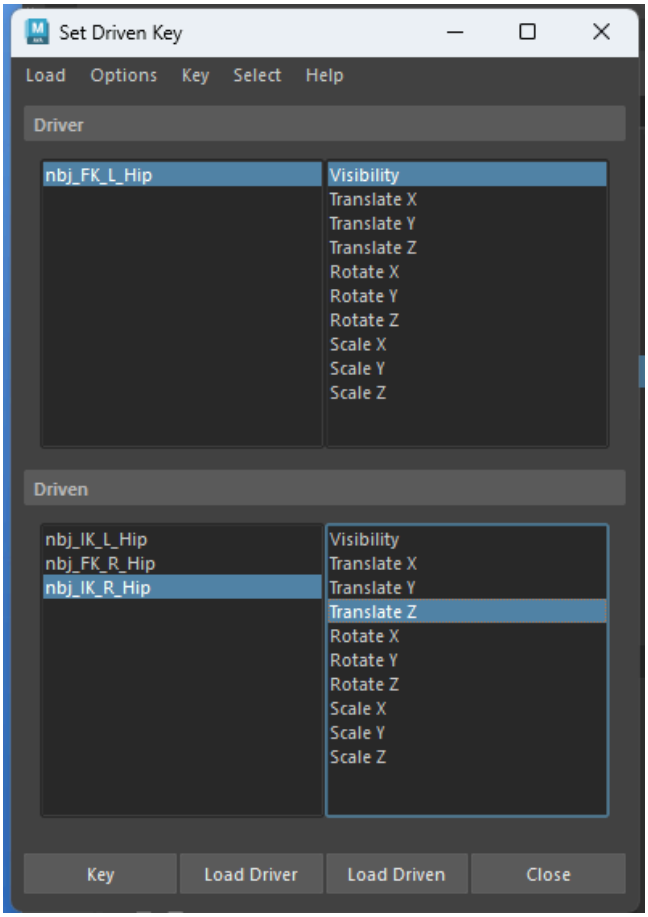


- "Key" → "Set Driven Key" → "Set…"



- Set the driver and driven:
    - Select your desired object, and press "Load" → "Selected as Driver" or "Selected as Driven", depending on what you want to load it in as

- Choose which attribute of the driver will drive the driven object: in the image below, the visibility of “nbj_FK_L_Hip” is driving the “Translate Z” or “nbj_IK_R_Hip”. Note that you can have multiple objects driving/being driven



- Set your first key:
  - Set your driver attribute (ex. The car is far from the door), then your driven attribute (ex. The door is closed), and press “key”
- Set your second key:
  - Set your second driver attribute (ex. The car is close to the door), then your reactionary driven attribute (ex. The door opens), and press key
- You now have an animation with set driven keys!

Skinning and Weight Painting

What is skinning?

- Looking at each vertex on the geometry & telling it how to constrain itself to the joints
- Like a simpler constraint: this geometry is going to follow x joint x amount, and y joint y amount

How does the computer do it?

- It guesses, then we have to edit it and fix it to what we want

Using skinning vs constraints:

- For when a shape is only controlled by one joint
- Skinning usually better with scaling

Option 1: Using "Bind Skin"

- Attach geometry vertices to joints
    - Need to think of the bone (rather than joints) because joints influence bones
- Select joints, shift select geometry
    - "Skin" → "bind skin" options
        - "Bind to: selected joints"
            - "Joint hierarchy" binds to all children of the selected joints
            - Only bind to selected joints (we don't want to bind to nbjs!)
        - "Bind method"
            - Closest distance: map the closest vertices to the joint
            - Closest hierarchy: find hierarchy
            - Recommends "geodesic voxel"
        - "Skinning method"
            - Use "classic linear"
            - "Dual quaternion" better for preserving volumes: bad for game engines
        - "Normalizing weights" IMPORTANT IMPORTANT
            - Each vertex has a number of joints influencing it
                - Each joint can have diff weights
                - Maya can let you specify joint weights in whatever units you want (is not in %)
            - Set as "interactive:" automatically normalize total weights to 1 live
                - "Post" normalizes afterward
            - Can have rounding errors here b/c maya only goes to 3 decimal places
            - Some people prefer to normalize randomly so maya won't change weights of other joints when you change one
        - "Weight distribution" distance
        - Uncheck "allow multiple bind poses"
        - Max influences: max number of joints influencing one vertex
            - Game engines struggle if it's above 4: use 4
        - Check "maintain max influences"
        - Keep all else as default

    - "Apply and close"
- "Windows" → "general editors" → "Component editor"
    - Lists relationship and values between subobjects
    - Go to "smooth skins" tab
    - Hold right click on the mesh, go into "vertex mode"
        - Select a vertex, you can see how much weight each joint has on the vertex
    - Can manually adjust weights all in component editor
        - Be careful: even if it says 0.000, if the joint is in there it has nonzero influence
    - Good for big, obvious adjustments, or fine tuning
- Do most of work in painting:
    - Paint per joint
    - "Skin" → "paint skin weights"
    - "Tool settings"
        - "Paint operation": "smoothing smooths it out
            - "Adding" adds it (change opacity and value)
            - "Replace" remove
    - "Mirror skin weights"
        - Weight influence association 1 as "one to one"
            - Makes it even
- If still wonky:
    - "Skin" → "Hammer skin weights"
        - Averages between weights of selected vertices
    - If hard to select vertices:
        - Go to "show/hide selection mask values" and turn off all except vertices

"Skin" → "edit influences" → "Add influence" options

- When adding influences, you are MOSTLY THERE and want to add additional influences from the geometry
- Turn "weight locking" on, set "default weight" as 0
    - Adds joints but sets weight as 0 so you can manually edit
    - If you trust the mesh to guess correctly, you can uncheck "weight locking"

Option 2: Using "Interactive Bind Skin"

Optional different way that oana likes to skin (requires more fiddling before weight-painting):

- Select joints, shift-select geometry
- Use "interactive bind skin" instead of "bind skin"
    - Gives more freedom at the beginning
    - Can't work with normalizing weights, requires extra step at the end:
        - Export weights to texture map (your UVs have to be right)
        - Unbind, rebind using normal bind weights
    - All settings that are the same should stay the same
        - "Bind to: selected joints"
        - "Volume type": "capsule"
            - Red: 100% influence

            - blue : 0% influence
        - "Skinning method": "classic linear"
        - Max influences: 4
        - Check "maintain max influences"
- To do after adjusting all capsules to renormalize weights:
    - Select geometry → "skin" → "export weight map"
    - Suggest higher "map sizes" for more complex meshes (recommend 1024 for both as base)
    - Chuck into assets: "leg_skinWeights"
    - Then, "skin" → "unbind skin"
    - Now, select the same joints
        - Note that you can use a mel script to select joints influencing a skin
    - Skin → bind skin
    - Skin → import weight maps, import your prev saved file

- After you've done it the first time, have to use "interactive bind skin tool" instead of "interactive bind skin"

Adding Extra SDK Joints:
Add extra joints with set driven keys to emulate stuff in body

- Ex. knuckles and elbow bones sticking out when you bend your arm
- Snap to middle of elbow
    - Name "sdk_elbow_pop" or "jnt_sdk_elbow_pop"
    - Maybe not "jnt" because you don't want it to skin automatically
    - Parent it to elbow
- Go into set driven keys:
    - Elbow as driver, sdk as driven
    - Elbow bend rotate direction → sdk translate x, y, z
    - Flat: sdk in elbow
    - Elbow bent: sdk pops out a bit
    - Key

Additional Tips for Skinning a Character's Head:
For additional geometries after your first one:

- CHECK "allow multiple bind poses" otherwise it won't skin

On the character:

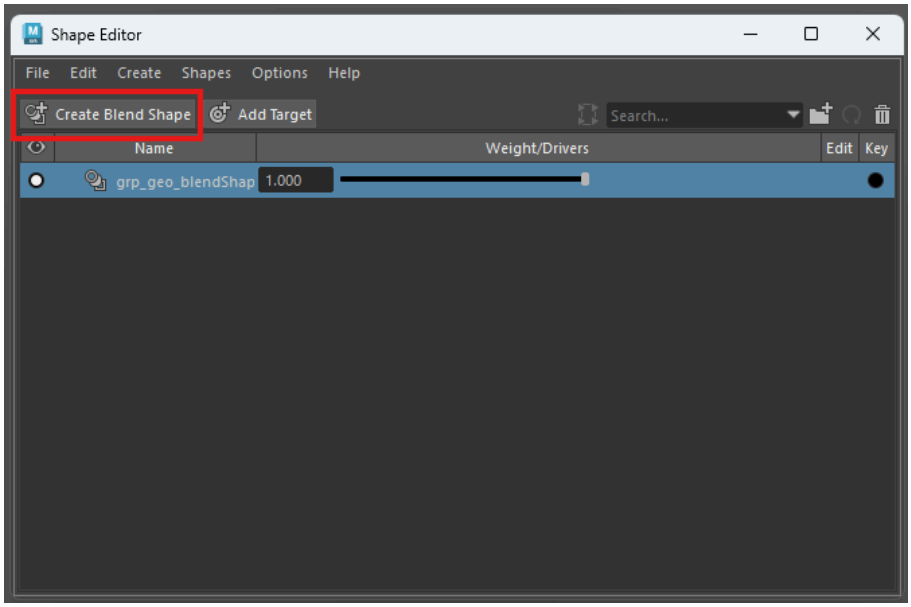
- Click on the rollout immediately to the right of "symmetry: off" next to the little icons: the grid icon is a drop down → "select by name"
    - "jnt_*"
    - Make sure ur constraints are not selected lols
- When u rotate the head, make sure that the mouth and eyes and stuff follow
    - Select the hair, then the head joint: → bind skin

    - Top teeth: attach to just head
    - Bottom teeth: jaw
    - Eyes: since they are one piece of geometry:
        - Bind the one geometry to both main eye joints
        - Use component editor to set all the vertex weights for each eye to their respective joint
        - MAKE SURE THE OTHER COLUMN DISAPPEARS
        - Add eyeballs as sdk joint to skin at the end (bc eyelids move when your eyes move)
    - Tongue: bind to all tongue joints
    - Go through each geometry element and make sure they're good
- Weight painting
    - Make it so when u try to open the jaw, it works
- Shift + greater than/less than keys → increase vertex selection
- Binding head w interactive weights
    - Lowkey don't include jaw – do that separately

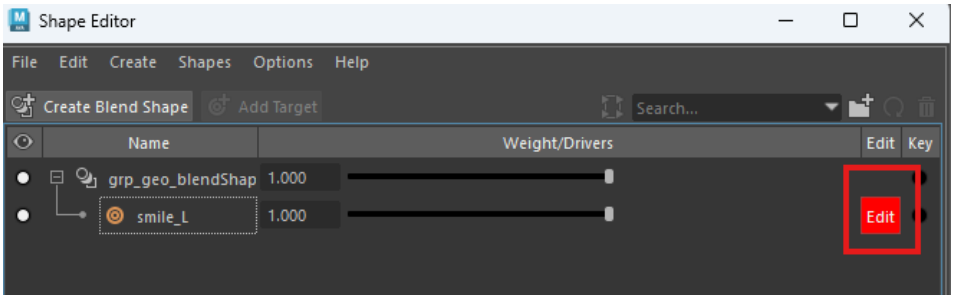
Facial Blend Shapes

What are blend shapes? Another way of manipulating the mesh – other than joints. Predetermine how the mesh will deform, control it with a slider from 0-1.

- Windows → shape editor
  - Select desired geometry → "create blend shape"



  - Will create target to specifically move mouth: "add target"
  - Name it "smile_L" (will be in charge of making left side of mouth smile)
  - Edit button is red: means that every move we make it recorded



  - Click "b" for soft selection mode → manipulates multiple vertices
    - Hold "b" and click and drag → increases or decreases selection
  - Make the desired edit, click the red "edit" button, which locks it
  - Mirror to other side: right click, "duplicate target"
    - Name "smile_R"
    - Right click, flip target
  - Group the blend shapes
    - Select desired target → right click → "group"

Now, dragging the slider for each target will induce movement in the mesh

- “Deform” → “blend shape” options
    - Create a blend shape
    - You can drag this into the node editor and edit visibility/attributes (like the FK/IK switch)
        - Make a switch with a float attribute, connect it to each action under the blend shape

Some common blend shapes

- Smile
- Eyebrow raise
    - Note that for this, we want the skin under the eyebrows to follow as they raise – move both the eyebrow geometry and the skin under it

Appendix

Troubleshooting (Common Issues)

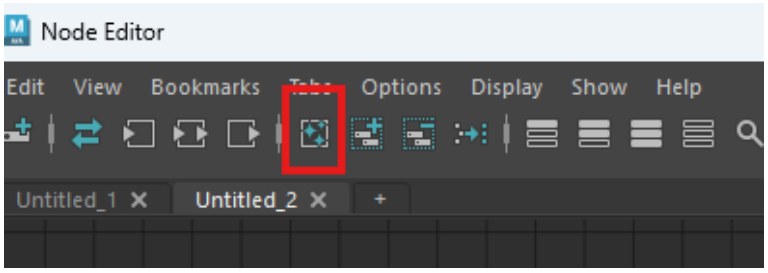
ALWAYS DELETE HISTORY AND FREEZE TRANSFORMS WHEN YOU WANT TO FREEZE SOMETHING'S POSITION

Elbow Joint "Popping" when Pole Constraint is Applied

Check that axes are oriented to "object" in Tool Settings

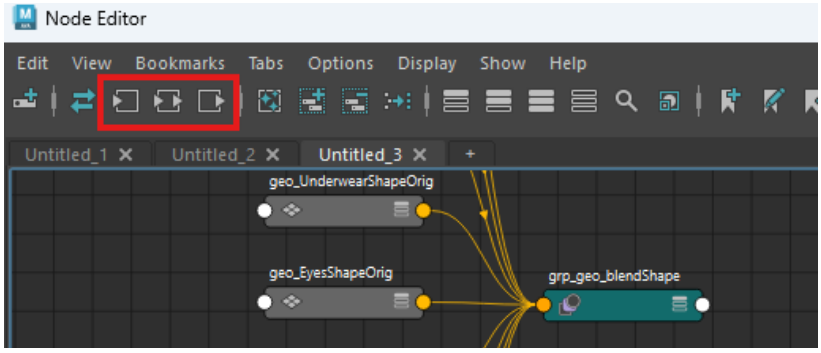
Clearing the Node Editor

Selecting notes and deleting them deletes the actual object. To clear the editor, use the "Clear" button at the top under "Tabs":



Finding Incoming Object Connections

Helpful for determining what is influence an object
Open node editor, click one of these buttons to see either incoming, incoming+outgoing, or outgoing connections



Finding Non-Object Nodes

In the Outliner: “Display” → Uncheck “DAG Objects Only”
Search for the node

Objects not Disappearing when Layer Turned Off

I had an issue where objects that I KNEW for sure to be in a layer were still visible even when layer visibility was turned off. Solution:

- If the object is in a group, check the group attribute editor → go to the tab for the layer
- Ensure enable overrides is on

Object Not Changing Color with Drawing Overrides

I had an issue where I had an fk control that WOULD NOT CHANGE COLORS. Drawing overrides in the attribute editor was off on the control and both of its groups, and no matter how many times i turned it on and off, the color WOULD NOT CHANGE

- Solution: the SHAPE under the nurbs curve had drawing overrides on and set to one color, which was overriding the curve and its groups.
- Explanation: in the viewport, the lowest child node’s display properties take precedent over any parent nodes
    - More specific rules override general ones

Keyboard Shortcuts

- G: repeat last action
- Y: repeat last tool
- P: parent
    - Select child first, then parent
- V: snap into place while moving
- F: make selected object fit full screen
- B: (in vertex mode) soft selection mode
    - Edit vertices around the selected one
    - Hold B and left-click drag:
- Ctrl+1: remove visibility of everything except what is selected
- D (hold): move parent joint without moving children
- Shift + P: unparent
- Ctrl+A: show attribute editor
- Ctrl+H: hide
- Move around: Alt + middle mouse drag
- Pan: alt + left click drag
- Zoom into a specific section:
    - Ctrl + alt + click and drag to select the section
- Make a keyframe
    - Select all, press s
- Delete a keyframe:
    - Right click on keyframe --> "cut"
- Timeline settings: guy running w cog in the lower right corner
- Change the perspective of your viewport:
    - Hold space + left click

Python Code!

Whenever running anything in Python, first import the “cmds” item from the “Maya” module:

```
from maya import cmds
```

Now, here is some code that I came up with because I love Python:

1. Renaming

```
1   joints = cmds.ls(sl=True, type='joint')
2   for joint in joints:
3       name = joint;
4       for i in range(len(joint)):
5           if joint[i] == "|":
6               name = joint[i+1:]
7       new_name = "nbj_FK_L_" + name[4:-2]
8       cmds.rename(joint, new_name);
```

Line by line explanation:

1. Create a list called “joints.” Populate this list with all (cmds.ls) your selected (sl=True) joints (type=’joint’)
    a. If you want to rename other objects (ex. Transforms or shapes) change the “type” argument to your desired object. You can see which object each item is if you hover over it in the outliner
2. A for loop to iterate through every item in “joints”
3. Assign the variable “name” to the item in “joints”
4. An if statement to parse through Maya’s joint naming scheme, which includes the names of all the parent joints (parent|parent|parent|joint)
5. Parsing continued
6. Parsing continue
7. Assign the variable new_name to your desired new name
    a. Concatenate whatever strings you want to create the new name. This code adds the prefix “nbj_FK_L_” to the 5th to second-to-last characters in the joint name
8. Use the maya rename command (cmds.rename) to rename “joint” to “new_name”

With some extra if statements, you can edit this code to add suffixes, fix capitalization, really anything. Note that this code only currently works for joints selected that are not in a hierarchy – otherwise, renaming the parent means that the script’s saved names for the child joints are wrong, and Maya can no longer find the child joints. But it’s still faster to apply this to one joint at a time instead of going through and manually renaming!

2. Setting Override Colors for Controls

```
1   controls = cmds.ls(sl=True, type='transform')
2   for control in controls:
```

```
3        cmds.setAttr(f"{control}.overrideEnabled",1)
4        cmds.setAttr(f"{control}.overrideColor",6)
```

3. From the selected objects, select only joints with “jnt” in the name

```
1    from maya import cmds
2    joints = cmds.ls(sl=True,type='joint')
3    select_joints = [j for j in joints if "jnt" not in j]
4    cmds.select(joints)
5    cmds.select(select_joints,d=True)
```