IK INVERSE KINEMATIC LEG AND REVERSE FOOT SET UP

How to set up an IK leg and reverse foot control Rig:

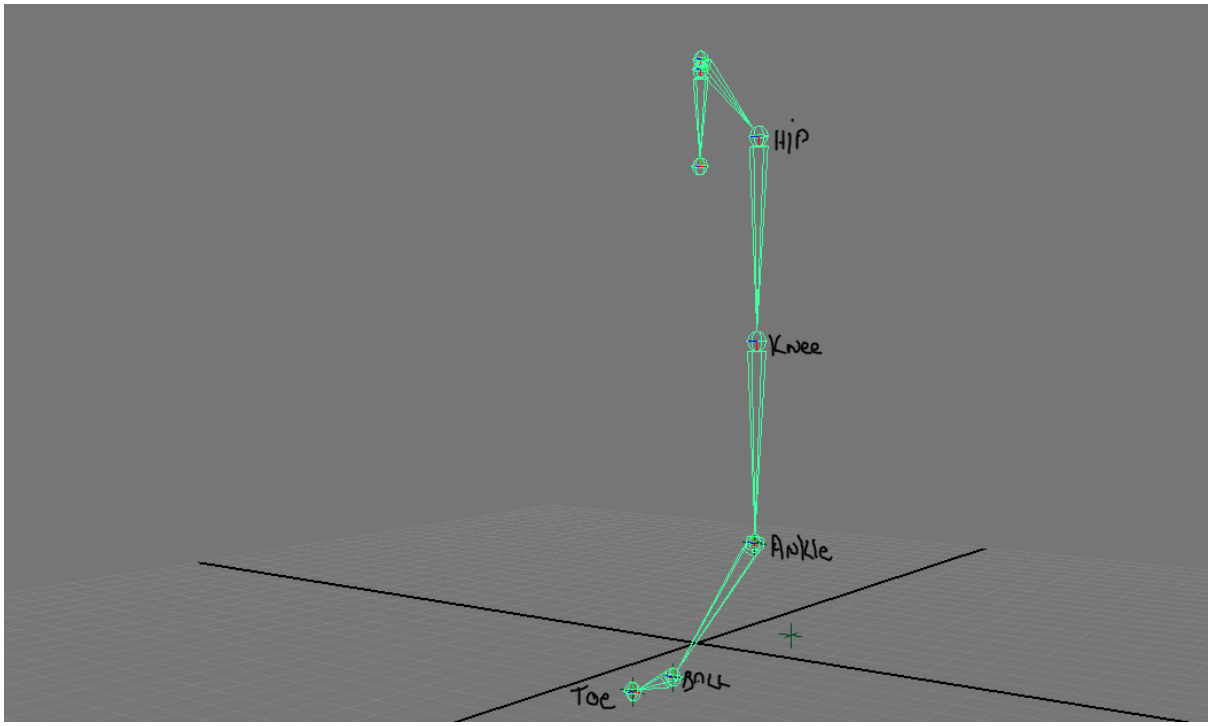
The theory covered in this tutorial is about using the IK Handle tool with its different solvers to solve and achieve the desired results for the leg and foot rig along with additional connections, constraints and parenting to finish off the rig.

Before I start anything I would have to ensure that I have the proper joints set for a leg and a foot ready to be rigged:

Leg Joints set up

Create joints for the leg set up, with the joint tool settings set to default. Starting at the hip, ending at the toe joint.

- Rename joints accordingly as the left leg bind joints.
- Parent the l_hip_bind_joint to the the root_bind_joint to link the leg to the root. (Other parts of the character will also be incorporated separately to this root joint).



Inverse Foot Joints set up

From this point I can start to set up the joints for the foot, however this would need to be done separately.

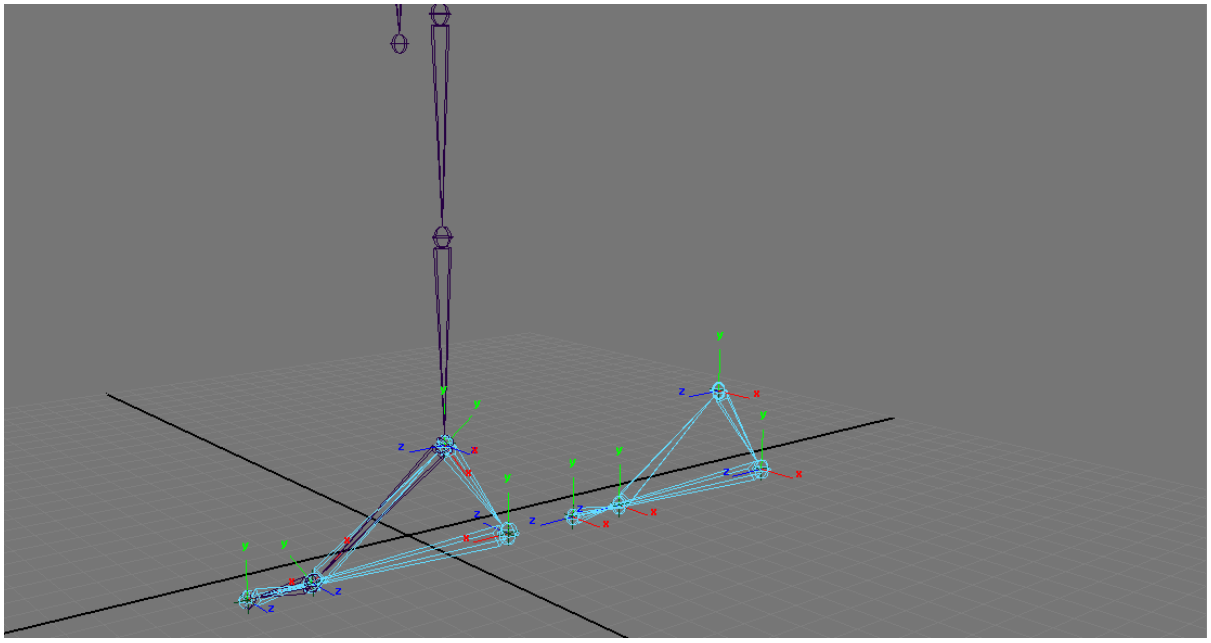
Make sure to hide the leg joints on another layer. In this example I have locators which mark exactly where the joints for the foot need to be laid in the scene in relation to the joints of leg.

It is important to always name joints accordingly before moving on. Keep workflow and naming conventions consistent, because it is crucial for the purpose of trouble shooting and problem solving later down the track.

-How to proceed:

First I have to understand the settings for the joint tool I have to use next and why I would choose to use it in that particular way. Below, I have an example of the results from the two options I can use to build the joints for the Inverse Foot.

- The first example on the left, the joint tool is used at default, as the same with the leg joints, for local orientation so as to have the x always pointing down the bone.
- In the second example to the right, the tool is set to make the joints orientations behave according to world space orientation.



Based on the image above, thinking ahead I would have to decide which result would suit my solution better. Ideally I would later like to incorporate control curves for the foot which would allow me to control the rig without needing to touch the ik Handles or joints or anything else of the rig from this base level of construction.

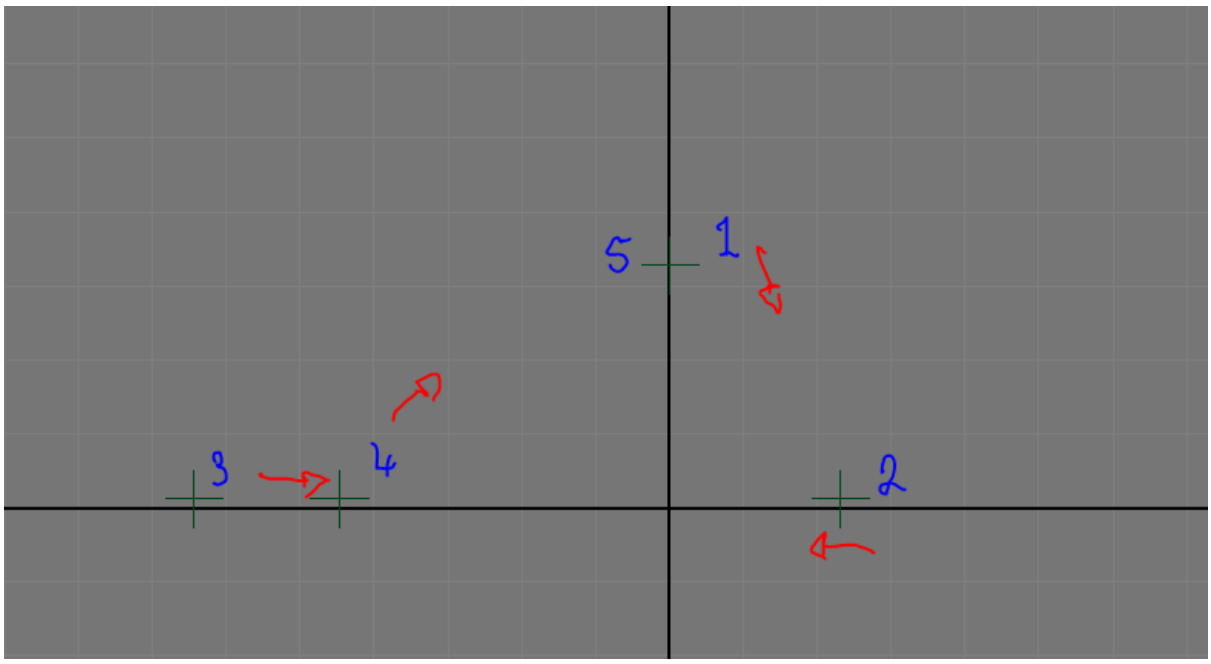
For that to be a success I need to remember that control curves are created to behave in accordance to world space orientation.

The problem is, when I would try to connect an object with local orientation to an object with world orientation, an offset would occur. This means that the mathematical calculation between the two objects do not match up and an additional calculation would force an average result between the two. Causing a visual representation of that calculation, the object that is further away from the average result would flip to match the maths of the object with the lesser gap in the calculation. This is something I should avoid because I don't want the rig to break and behave in strange unpredictable ways after the rig is finished as a character for animation.

Therefore, I have my answer to ensure my solution.

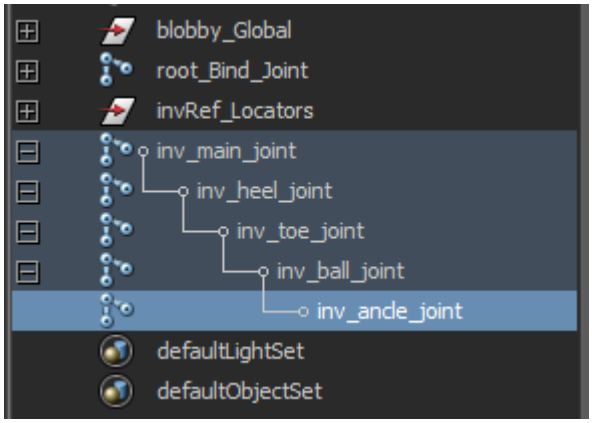
Foot Joints set up:

Understanding now where I'm heading with my next steps, I can build the joints for the inverse foot with the joint tool set to "Orient to World".



On a new layer, lay down the points for the joints by following the diagram, finally completing the operation on the start point.

It is important to keep track with the naming of objects as I am building. Because this is an inverse foot set up, the foot joints should read as follow:



Based on joint numbers from previous diagram the naming for the joints should be accordingly:

1- l_inv_main_joint
2- l_inv_heel_joint
3- l_inv_toe_joint
4- l_inv_ball_joint
5- l_inv_ancle_joint

(Because this is a set up for the left side of a character, there needs to be a prefix "l" in the naming convention to clarify what side of the character these joints are associated with)

The leg and the foot are almost ready to be rigged. Double checking orientations.

- As a precaution to make sure the leg would bend in the way I would like it to, I can set a preferred angle on the joint from where I would like the leg to bend.
- So, back on the layer for the leg joints, rotating the knee joint I can set the preferred angle to let the leg bend backwards as anticipated naturally. When doing this, because im working with joints that functions in local orientation, I need to ensure that the rotation tool is also set to rotate local orientation when completing this step.

IK Handles

Ik handles are what is used to drive and manipulate the joints system. This is also the actual "rigging" part of the whole process.

"IK solvers are the mathematical algorithms behind the IK handles. IK solvers calculate the rotations of all the joints in a joint chain controlled by an IK handle. The effect an IK handle has on a joint chain depends on the type of IK solver used by the IK handle". – **Autodesk Maya Help files.**

For this tutorial, I used two different ik solvers to solve the rigging problem of the leg and the inverse foot separately.

- RP iKsolver- Rotation plane Solver - used for rotation of the shoulder or hip .
- SC iKsolver- Single Chain Solver - used for single chain solutions.

ikRPsolver - This reads as Inverse Kinematic Rotation Plane Solver.

This solver is ideal for posing joint chains, such as arms and legs, because you would want them to stay in the same plane.

So if you think about it, the leg functions in a rotational manner, therefore the tool option setting for RPsolver would do the job just right.

ikSCsolver – this reads as Inverse Kinematic Single Chain Solver.

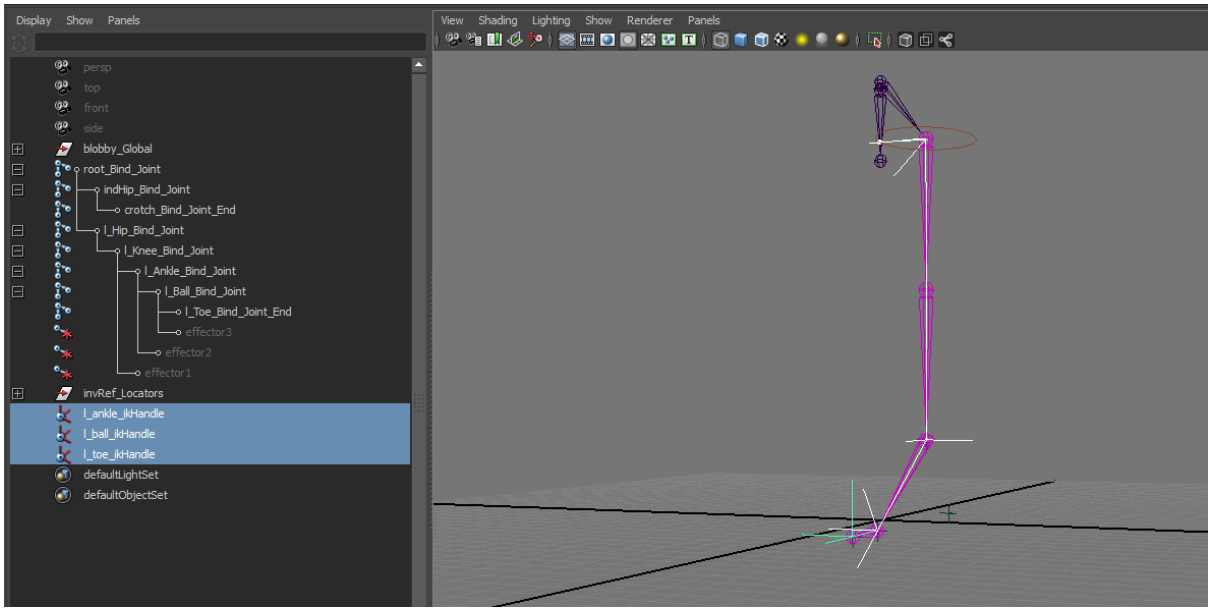
A single chain IK handle uses the single chain solver to calculate the rotations of all the joints in the IK chain. Also, the overall orientation of the joint chain is calculated directly by the single chain solver unlike the rotate plane IK handle.

In that respects, a SCsolver will be used for rigging the foot.

RIGGING THE LEG AND THE FOOT:

Having all that said, I can now know what settings to use when applying the IK Handle to the joints. (Still just working with the Leg joints layer).

Remembering I named my joints properly I should now apply the IK Handles accordingly:



- With the ikHandle tool setting set to **ikRPsolver**, insert a handle by clicking from:
- **inv_hip__bind_joint** to **inv_ankle_bind_joint.** (To allow the knee to bend, also it is how ikHandles work, with one joint skipped between the start and end of the handle)
- Change the tool solver settings **ikSCsolver** and insert a handle by clicking from:
- **Inv_ankle_bind_joint** to **inv_ball_bind_joint**
- **Inv_ball_bind_joint** to **Inv_toebind_joint.**

(Remember to name the ikHandles appropriately after finished with applying them).

- l_ankle_ikHandle
- l_ball_ikHandle
- l_toe_ikHandle

Hierachy:

It is now time to bring the foot joints layer in view so I can start combining the leg joint system with the foot joint system.

Parent IK's (previously placed on the leg) to respective joints of the foot *(straight parenting)*

- l_ankle_ikHandle to l_inv_ankle_joint
- l_ball_ikHandle to l_inv_ball_joint
- l_toe_ikHandle to l_toe_joint

Pole vector Constraints and Control Curves:

A pole vector constraint is used with an ikRPsolver for additional control over the behaviour of the middle joint in the IK chain.

SETTING UP CONTROL CURVES:

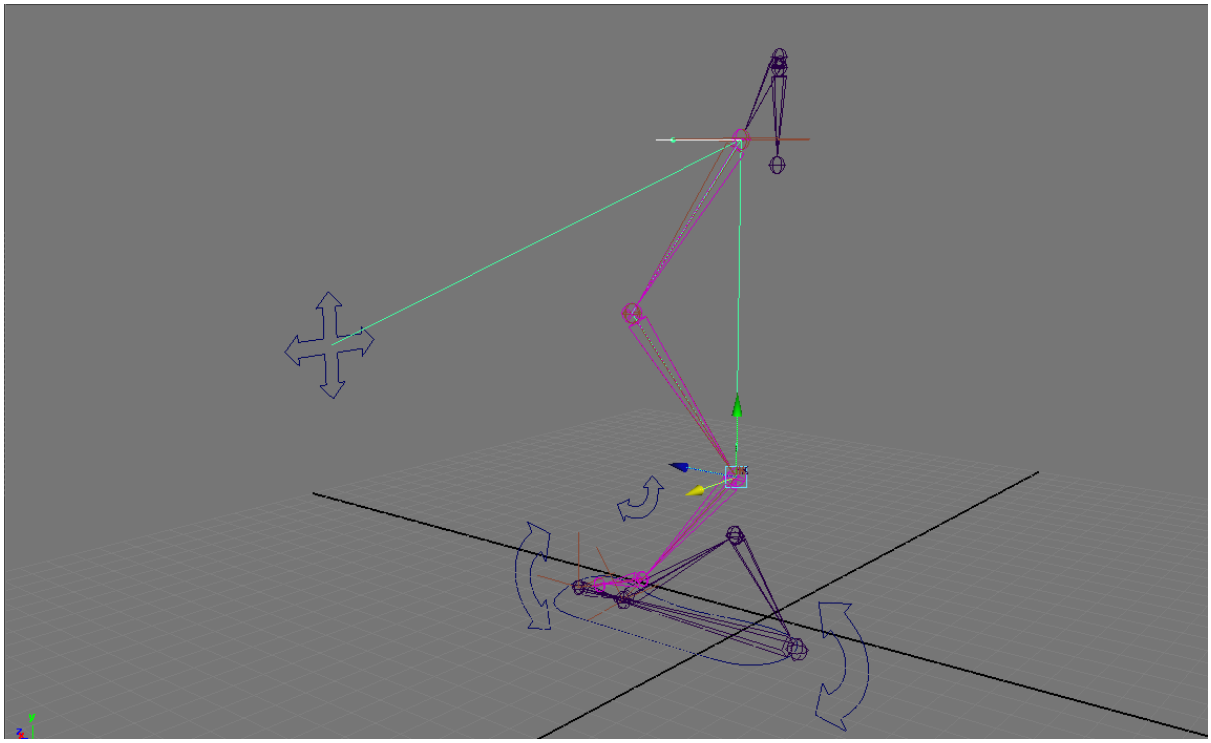
- Snap in place, foot control curves and a knee polevector control curve.
  (The curves would have been pre-made separately).
- Freeze Transformation and centre pivot the knee's pole vector control curve.

Simple knowledge to follow in order to complete the set up for the control curves:

The knee control curve can be v-snapped in place on the knee joint and simply moved off in the z-axis to be positioned a distance in front of the knee.

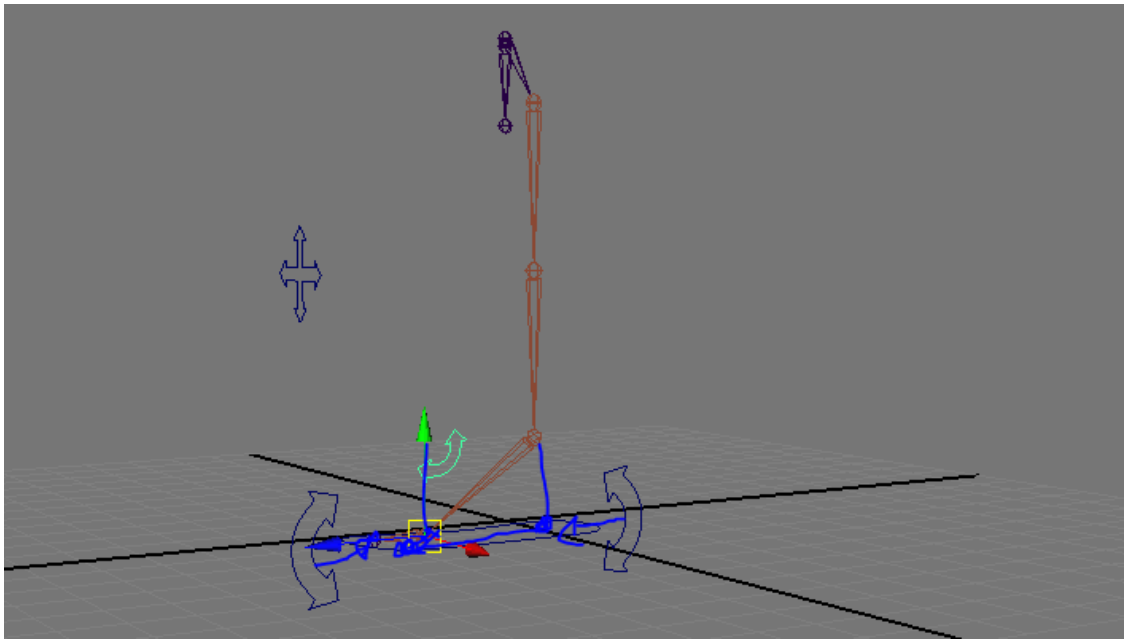
It is important to freeze the transformation and delete the constructions history of the control curve to ensure that there will be no history information, rotates or translates values remaining to cause later issues.

- Apply pole vector constrain for the knee (At default pose, select curve then l_ikHandle, apply the pole vector constraint and test to see if it works, always test the rig as you go because if a problem is discovered too late, often you would have to start again from the point where the mistake was made.



By applying this knowledge before incorporating the controls into the rig, it will allow for a clean control curve with zero rotation and zero translates which means all the garbage values are discarded. After including these control curves in the rig to drive the ikHandles, it means there will be no unexpected calculation issues occurring, enabling it to act as the perfect control with a zero starting point with which the animator will always be able to reset a character to its default pose by simply resetting the controls to their zero values.

(Visual reference of where I am heading)

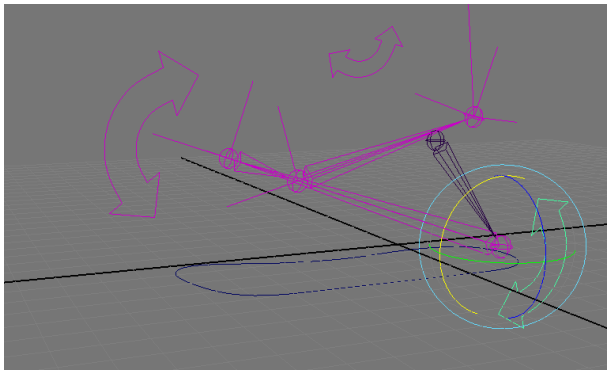
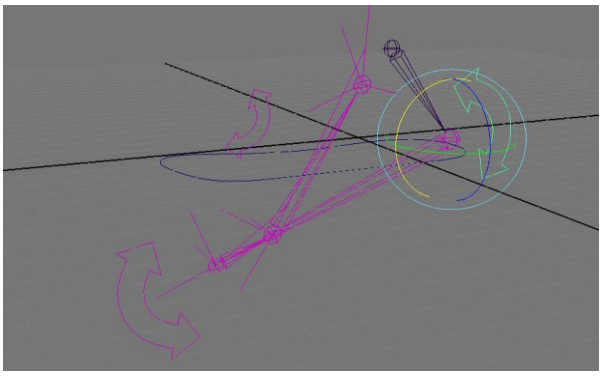


The amount of curves that are needed is usually dependent on the rigger's style, how the rig is set up and of course how much control you would like to have over the ability to manipulate the rig.

In this reverse foot control rig as a particular example, it can give me a lot of control over the various pressure points in how a foot functions:

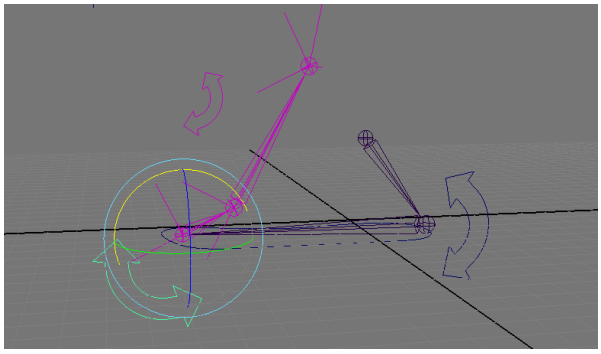
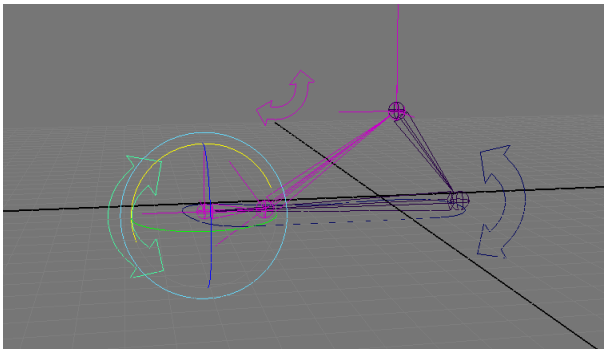
(Sticking point – heel):

The whole foot pivoting from the heel, up and down, an attribute to the main foot control can be added later to also manipulate side to side rotation.

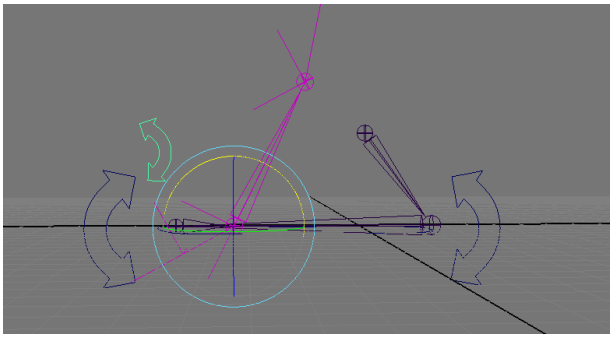
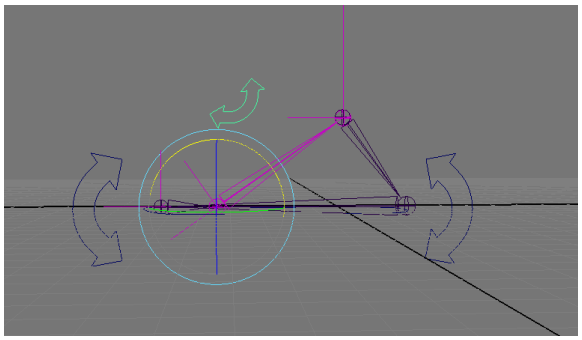
(Sticking point – toe):

The foot pivots up and down from the tip of the toe.

(Sticking poiny – ball):

The foot pivots from the ball of the foot up and down. Can also with an added attribute to the foot control, pivot on ball from side to side, like squashing a bug.

CURVES' PIVOT POINTS:

- l_foot_control – place and shape around the foot, snap the control curve's pivot to the l_inv_main_joint. This will be the main/global foot control which would ultimately be in control the whole foot with all its control following.

1. l_heel_control – Snap to l_inv_heel_joint and move off slightly to position the curve behind the heel.
2. l_toe_control – Snap to l_inv_toe_joint and move off slightly to position the curve in front of the toe.
3. l_ball_control – Snap to l_inv_ball_joint move off slightly to position the curve above the ball joint and rotate to follow the shape of the foot. (see above diagrams)

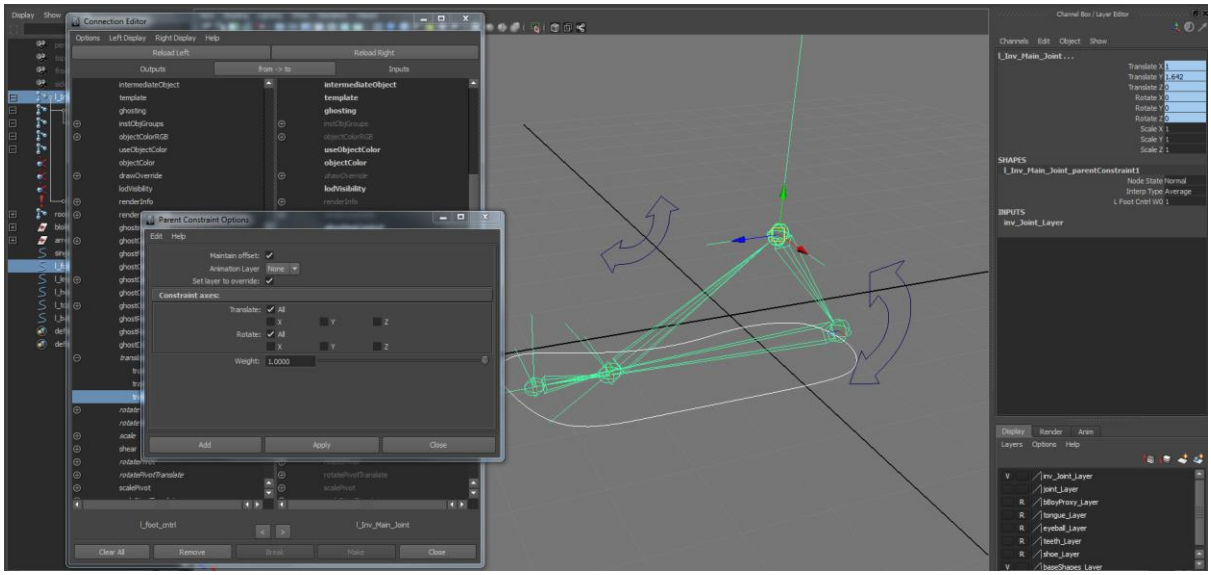
As a final touch to setting up the curves, it is important to point snap the curves' pivot points to the joints they are meant to control:

- This allows for the controls to properly rotate the joints from the point of where they are meant to rotate naturally instead of from the original centre pivot point of the curve which would not be correct and would cause the rig to break.
- When everything is done, ready in place, and the control curves pivots are snapped to the correct joints, finish this step again by deleting the construction history and freezing its transformations as explained before. Just to be sure everything is clean.

Making curve controls manipulation of the foot. Understanding where I am going before attempting to solve the problem by testing possible solutions.

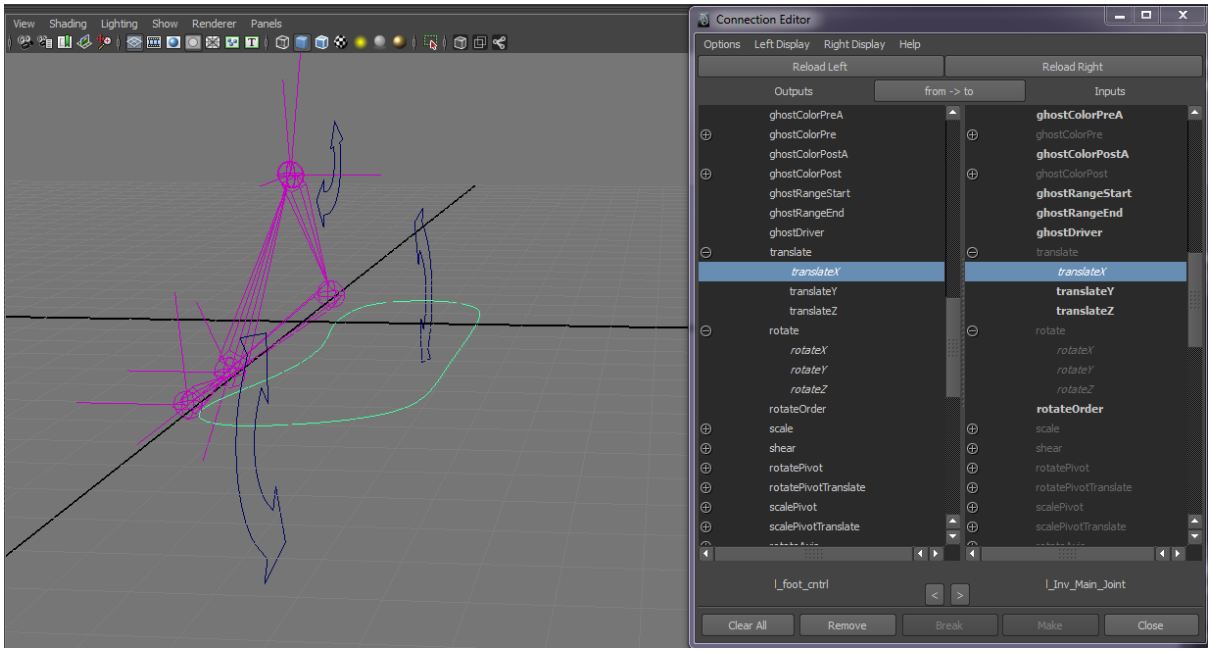
CONNECTING THE CONTROLS INTO THE RIG.

Connection editor or a Constraint? Or both? – Lets test it:



When trying to straight connect the l_foot_cntrl curve to l_inv_main_joint to control the rotations and translates of the foot, the connection makes the foot joints flip out.

Connecting up the rotations worked fine however as soon as I attempted to connect the translates as well, **the calculation in the joints did not match that of the control curve and so tried to find an average solution between the two, causing an undesired result:**



Why?

Even though the curve itself has no construction inputs on it due to Freeze Transforms and deleted history, as explained at the start, the joints actually do still have their own local translate values to mark their constructed location in relation with world space. So trying to make a straight connection for the translate values through the connection editor, it forces the maths of l_Inv_Main_Joint and the control curve to match up, which it can't because the calculation doesn't correspond to each other. The rotation connection works fine because there is no rotate value information on the joint I'm trying to make the connection to, to cause an offset.

Removing construction history and freezing an object's transformation, resetting the math to zero, makes it look in reference to numbers like the object should be in the default construction location value (00;00;00 in X Y and Z on the grid) while the object itself remains where it was last manipulated and placed. So during the process of trying to make numbers match up, the joints get confused when forced to connect with the control curve and so it jumps back to the 00;00;00 location where it thinks it is matching up with the control curve according to its mathematical values "frozen" to default. (Refer to image above)

So this approach for the first step in connecting the curves is not going to work in my favour and therefore I need to explore another option.

Solution:

Using a Parent constraint (a constraint allowing for translates and rotations to be controlled) with maintain offset on, would allow you to bypass that clash in calculation which is achieved through the "maintain offset" option to make both objects stick to their mathematical locations, leaving those numbers unaffected while achieving the desired effect I was after without further problems.

Often the above step would explain reason. But I found at this particular step in the tutorial that even though I have maintain offset left off, the effect of a constraint allowed the result I needed, compared to the connection editor which caused an offset in the translates.

To explain this, I think because these two options both achieve results through a different way for calculating the task at hand. The connection editor makes for a direct connection like going straight to the core of the objects and forcing them to make that connection to each other, bypassing all else. Whereas with a constraint, the path is not so direct, allowing for other things to be taken in consideration; The joints for the inverse foot setup were created to be set in world space (for this particular reason): the control curve is world oriented; The pivot point for the control curve has been snapped in place with the inv_main_joint making the two objects which I'm trying to form a relation in connection with, to have the same pivot points in world space, avoiding an offset when leaving "maintain offset" off when applying the constraint.

In general this is also a very good way to test yourself and see if you can understand your workflow in progress, if your work is clean and planned ahead for in order to not have to rely on an option such as the "maintain offset" setting to fix your calculation mistakes automatically.

With all the above mentioned considered, for this particular step, it turns out I have no problems when I constraint the control curve to the joint, without needing to tell it to maintain offset. I got the result I was after and the control curve's input values remain untouched.

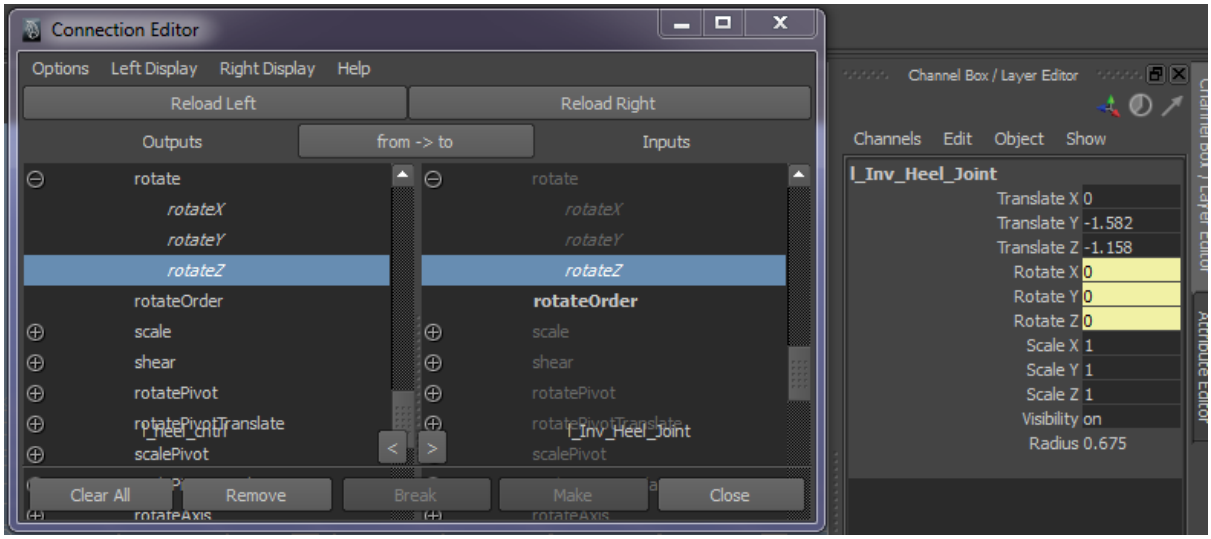
CONNECTING CONTROLLERS

Controls step 1: Foot Control – a control to manipulate the root of the IK reverse foot set up Rig, by translates and rotational values.

- Parent constraint l_foot_control to l_inv_main_joint

Controls step 2: Individual rotation controls – a control for each sticking point of the foot to create a natural walk cycle.

For this step I know that I need to make a connection in only rotation between the controller rotations and the respective joint I want to control. Already tested and knowing that it works, **this step becomes a simple direct connection, bringing me back to the connection editor.**



The control curves are loaded as an output master – Select, Reload Left.

The joints to be controlled are to be the input values connected to the controls – Select, Reload Right.

1. **l_heel_control (Reload Left) to l_inv_heel_joint (Reload Right)**
   - Find and connect:
     - rotateX to rotateX
     - rotateY to rotateY
     - rotateZ to rotateZ
2. **l_toe_control (Reload Left) to l_inv_toe_joint (Reload Right)**
   - Find and connect:
     - rotateX to rotateX
     - rotateY to rotateY
     - rotateZ to rotateZ
3. **l_ball_control (Reload Left) to l_inv_ball_joint (Reload Right)**
   - Find and connect:
     - rotateX to rotateX
     - rotateY to rotateY
     - rotateZ to rotateZ

Tidying Up, the relation of the curves to each other.

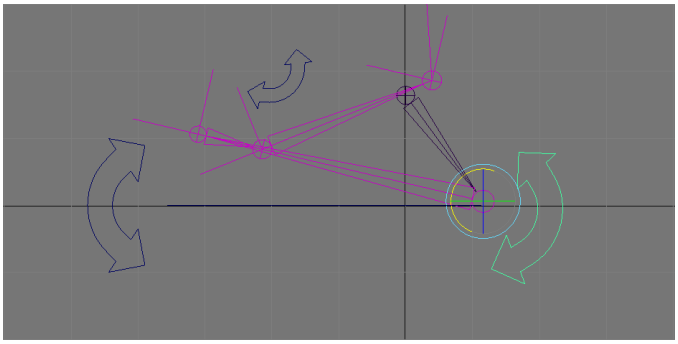
I now need to make sure that I wrap up all the curves together so they behave in a way that is visually pleasing and easy to control.

Now that ive got each control curve driving their assigned joint, I can parent them in such a way so as to not have any curves look like they are “left behind visually” while im controlling another part of the foot.

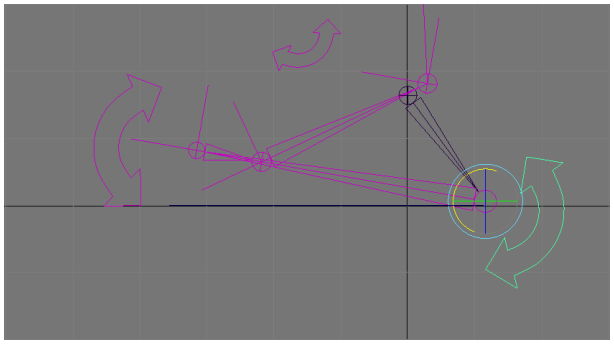
- Keeping in mind that I would like to have the l_foot_control to be the master/driver whole foot. When this control is manipulated, everything else needs to follow.

Controls Step 3 – Clean control:

No Relation between curves:



Hierarchy relation between curves:



Breakdown:

- Rotating l_heel_control ~ l_toe_control and l_ball_control needs to follow.
- Rotating l_toe_control ~ l_ball_controll needs to follow.
- Rotating l_ball control ~ last one in the hierarchy, rotating by itself.

- Moving or Rotating foot_control ~ all the other curves follow.

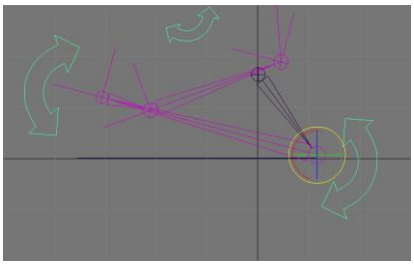
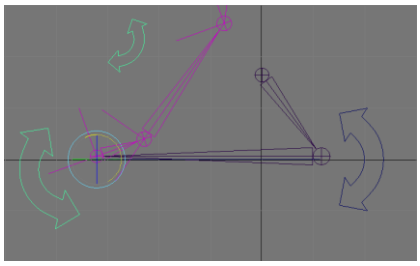
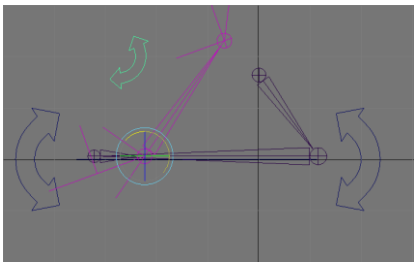
How to resolve: (Solution Version 1)

Creating a hierachy:

Select l_toe_control (child) then l_heel_control and apply parent.

Select l_ball_control (child) then l_toe_control (child of l_heel_control) and apply parent.

- **The child follows the child that follows the parent. The parent controls all the children.**

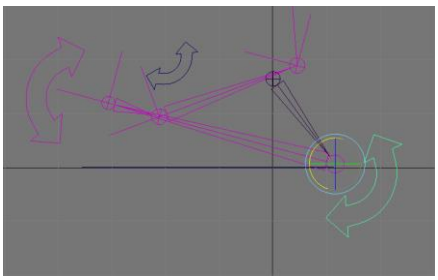
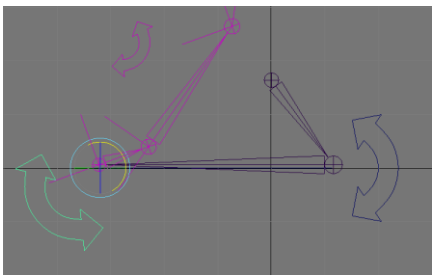
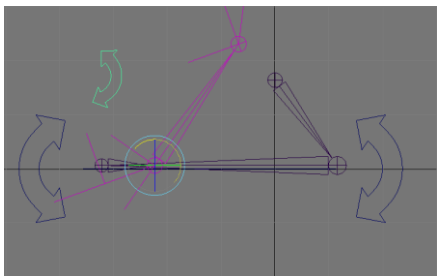




How to resolve: (Solution Version 2)

Thinking about the concept of direct connection and how hierachy works at the same time:

Still at the same result, the rotation controls driving the joint rotation.

- The rotation of the controls are directly connected to the rotation of the joints through the previous step in the connection editor.
- As a result, when the control rotates, the joint rotates.
- So im thinking how I can make the l_toe_control follow when I rotate the l_heel_control.
- So if I directly parent the l_toe_control to the l_inverse_toe_joint (which in world orientation and hierachy up the chain follow the rotation of the l_heel_joint) in theory, that will make the control stick and follow when the l_heel_control is rotated.
- This approach worked really well with the correct reults that I was trying to achieve.

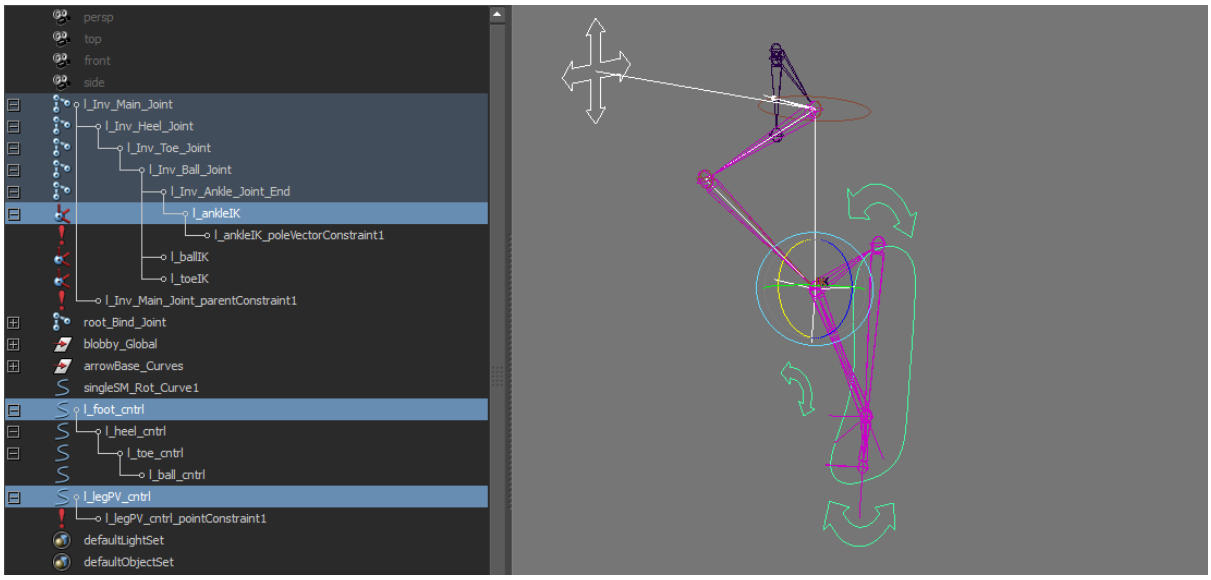




1) **Select l_toe_control; parent to l_inv_toe_joint.**
   (result, parented control follows the top rotating control in the hierachy: above left image)
2) **Select l_ball_control; parent to l_inv_ball_joint.**
   (result, parented control follows the next rotating control that followes the main parent rotating control in the hierachy: above middle image)

Side Note:

- The bottom curve in the hiearchy still rotates individually as there is nothing next to it down the hierachy chain.
- The rotation curves inherited the joint's local orientation translate values in world space. But this is not such a big problem because as polished cleanup the translate values of the foot rotation controls will be locked and hidden from the animator since these controls are not made animating the translates.
- In conclusion, the results I got from this solution is visually more pleasing having only the active curve highlted showing exactly which control is being manipulated.

Final Control:



1) Select the l_heel_control (parent of the foot rotation curve children) and apply parent to the foot control to make it the ultimate parent over all the Inverse Foot Control Curves.

2) **Include the Pole Vector control for clean manipulation**: point constrained to the foot control to make the knee control follow the translates of the foot control so it doesn't get left behind during animation process.

THE END…