Trigger

Release 1

Arda Kutlu

Aug 08, 2023

CONTENTS

Note: This project is under active development.

CHAPTER

ONE

CONTENTS

1.1 Overview

Trigger is a modular rigging and automation tool for Autodesk Maya.

It is designed to streamline and simplify the rigging process for animators and riggers.

With Trigger, users can quickly and easily create custom rigs with a wide variety of features and options.

The tool offers an intuitive user interface and a range of modules that can be combined to create unique rigs tailored to specific project requirements.

Whether you are a seasoned professional or just starting out in rigging, Trigger can help you create high-quality rigs quickly it would take with traditional methods.

Besides rigging, trigger can be used to automate and template tasks, such as assembling shots/assets, quality checks, etc.

It allows TDs and developers build their own actions and modules as plug-ins and easily integrate as plug-ins

1.2 Installation

1. Download the repository zip file: https://github.com/masqu3rad3/trigger/archive/refs/tags/rigging-0.2.8.zip
2. Extract the contents into a folder
3. locate the `trigger/python/dragAndDropMe.py`
4. Drag and drop this file into a Maya viewport.

That's all!

The repository can be cloned or forked for advanced users as well

```
git clone https://github.com/masqu3rad3/trigger.git
```

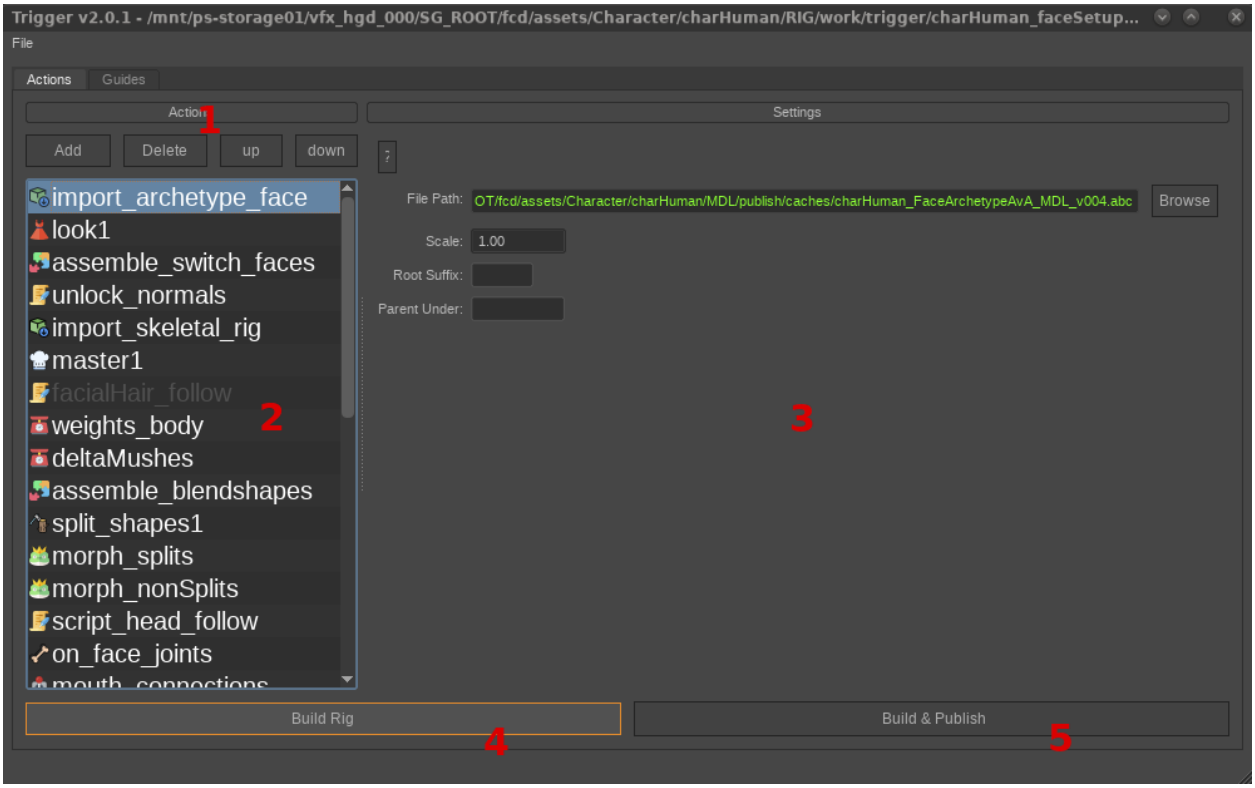
1.3 Getting Started

1.4 Interface

Trigger UI contains two main tabs

1.4.1 Actions Tab

Actions tab is the most important part of the UI and pretty much everything is happening in this section.

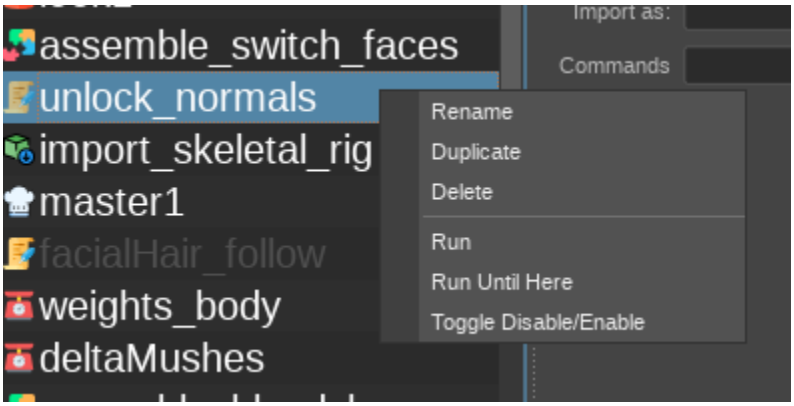


1. Action Buttons

- **Add**: Drops down a floating menu containing the list of available actions
- **Delete**: Removes the selected action from the list
- **Up**: Moves the selected action up in the layer hierarchy
- **Down**: Moves the selected action down in the layer hierarchy

2. Actions list

List of available actions in the current Trigger Session Right click any action will display the following menu:



- **Rename**: pops up a edit dialog to rename the current action
- **Duplicate**: Duplicates the selected action to below
- **Delete**: Removes the selected action
- **Run**: Runs only the selected action. Does not reset the scene
- **Run Until Here**: Starts running the actions from the begining of list and stops on selected action
- **Toggle Disable/Enable**: Enables or disables selected action. Disables actions displayed as grayed out

3. Settings Panel

Displays settings for selected action. There are detailed eplanations for each action in *Actions* section

4. Build Rig Button

Resets the scene and runs everything in the actions list. The action items change color during the run:

- **yellow**: means the action is processing
- **green**: means that action successfully completed
- **red**: means that the action cannot be completed due to an error. Check the script editor for details.

> **Danger:** Build Rig button will RESET the scene. If you have unsaved work or unsaved action related changes ALL will be lost. Be sure you saved your scene and/or your actions and guides are up-to-date

5. Build And Publish Button

Builds the rig and publishes maya scene*[0]

1.4.2 Guides Tab

Guides section is more similar to traditional modular rigging tools. In guides section, we will be creating the 'Guide Joints' which will work with 'Kinematics' action in Actions panel.

Unlike Actions panels, joints created in Guides section is not going to be saved with trigger session. This panel is only for manipulating maya joints and adding them the correct attributes which will be used by trigger when creating the actual rig.

The guides must exported in order to be used in Kinematic actions. The guides created from this section can be exported using File => Export guides menu item.

Tip: Guides can be exported to be used in Kinematics action from File -> Export Guides menu

---

1. Alignment Panel

Defines the alignment of the next limb module that will be created

- **C**: Centered. The modules created with Center alignment are yellow and does not have a mirrored pair
- **L**: Left. Left modules are indicated with blue color and represents the left side of the character/rig.
- **R**: Right. Right modules are red and represents the right side of the character/rig
- **Both**: Creates both Left and Right Modules at the same time and dynamically mirror links them together. In mirrored pairs Right sided (Red) modules cannot be manipulated and always mirror the Left side
- **Auto**: This option is useful if you want to create paired limbs which does not share the same parent. For example, Right and Left thumbs of a character should be mirrored but they dont getting mirrored using the same joint as center. When Auto, trigger tries to find the corresponding parent pair joint instead of creating both limbs connected into the same joint.

[0] This feature is not yet implemented

2. Modules List

This area contains all available limb modules in Trigger & pre-defined presets (Essentially, collection of modules with a hierarchy)

The numbers next to the modules means that how many joints will be drawn with that module. If the numbers are disabled and not editable, that means that module can only be created with fixed number of joints. For example, the Arm module can only have 4 guide joints defining shoulder, upper arm, elbow and hand

Warning: The guide joints drawn into viewport is not going to be saved with the Trigger session! You need to export your guides to a .trg file from File -> Export Guides

Guide Creation

- **Normal Mode**
  When clicked each module button, the corresponding limb will be created using the selected joint in the scene as parent. If nothing selected, the limb (or limb pairs) will be created on their default locations. Depending on the set alignment the pairs will be created automatically

- **Define Mode**
  Holding down *Ctrl* when the trigger window is activated will turn the outlines of the module buttons to green, activating 'Define' mode. In define mode, you can convert any joint hierarchy into Trigger guides.

3. Guides List

List of guides available in the Scene. Please note that, the export guides command will export all these guides listed in this list.

4. Guide Properties

Guide properties have some shared properties for each guide and some other properties unique to each module. More detail on modules section.

Common properties are:

- **Module Name**: name of the selected module. Although it is not mandatory, it is advised to provide a unique name for each module in the hierarchy
- **Up Axis**: Defines the up-axis of the module. Default is +Y
- **Mirror Axis**: defines in which axis the module will be mirrored. Default +X
- **Look Axis**: Modules facing direction. Default +Z
- **Inherit Orientation**: If this is checked, Trigger will assume the existing orientation of the guide joint as correct. Otherwise the orientations will be calculated automatically and only positional values will inherited. This is especially important for the modules using IK chains.

Although these values are common for all modules, some or all may not be affecting anything depending on the complexity of the module

5. Test Build Selected Branch Button

This button will imitiate the Kinematics action in actions panel and creates a kinematic rig from the selected guide branch. In order to work, a root joint of a module must be selected.

All hierarchy starting from the selected root will be rigged. For example, in humanoid preset. If one of the LegRoot joints selected, it will only rig the selected leg, but if the Base joint selected it will rig the entire character.

Tip: 'Test Build Selected Branch' button - as the name suggests - is for testing purposes only. Don't use it on an actual rig. Always export your guides into a file and use them with a 'Kinematics' Action

---

1.5 Actions

Trigger Actions are core of the whole system. They can be anything from simple model imports to complex network creations. Some can work without any dependency, some would require certain elements, some may require inputs, some can save and store iterations or any combination of those. The list of current available Actions in Trigger (October 2021)

1.5.1 Assemble

Assemble Action is for combining multiple of alembic files into the scene at the same time.

Acting as kind of a stage manager, it only updates the existing hierarchy, so the order of Alembic files actually matters.

If grouping hierarchy matches between alembics, instead of creating copies of the same structure over and over, it uses the existing one. This makes it very useful for combining different assets which will be united in a single rig without the need of extra clean-up scripts.

Button Functions

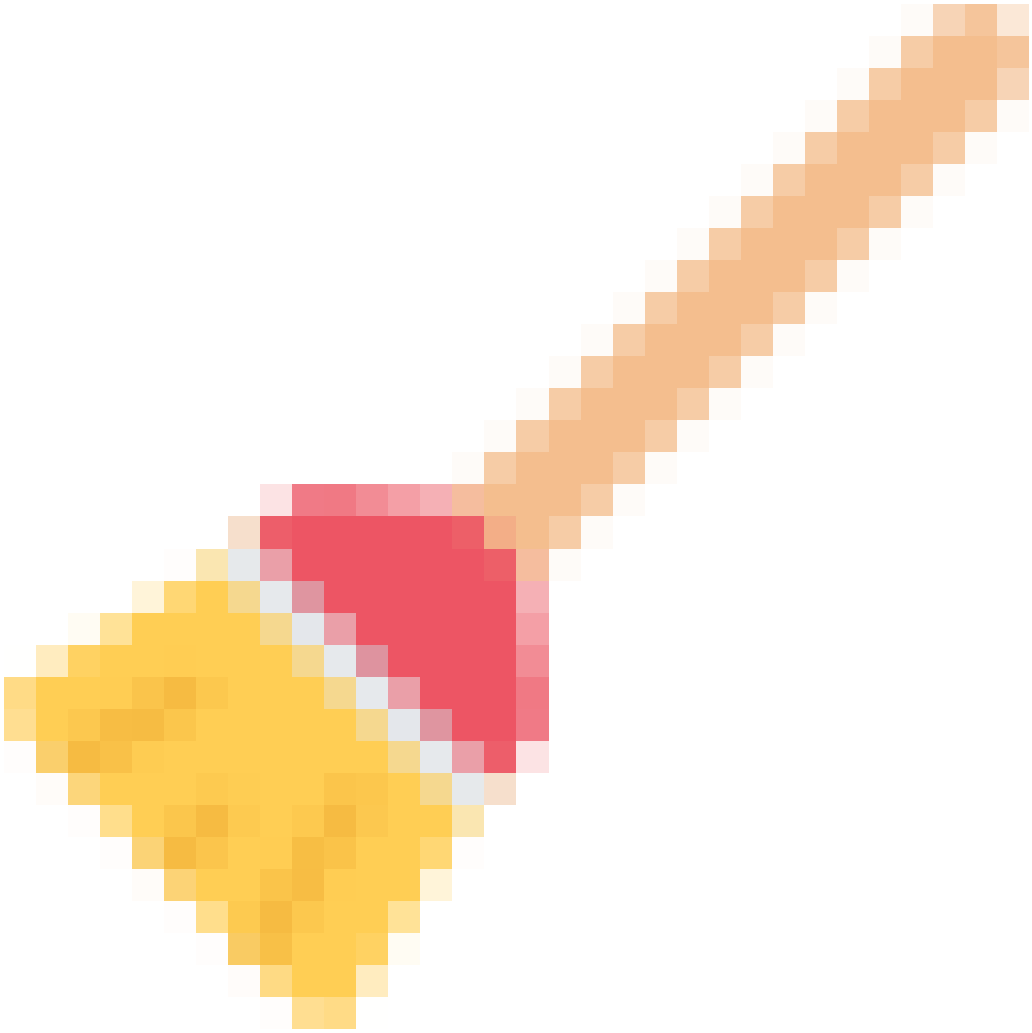
- **Add**: Brings up the browser menu to add a new alembic cache file
- **Next Version**: Switches to the next version of selected alembic file, if there is any. Yellow means that either the file name does not have a version info or it is not the latest version.
- **Previous Version**: Switches to the previous version of selected alembic file
- **Remove**: Removes the selected alembic file from the list. This does not delete the actual file in the file system. Just removes the Trigger definition
- **Clear**: Removes everything from the list.

Tip: *Ctrl + Up Arrow* and *Ctrl + Down Arrow* Keys move between versions if v<digit> format extension used for

version file names.

Note that the all parts of except from the proxyAvA are the latest versions.

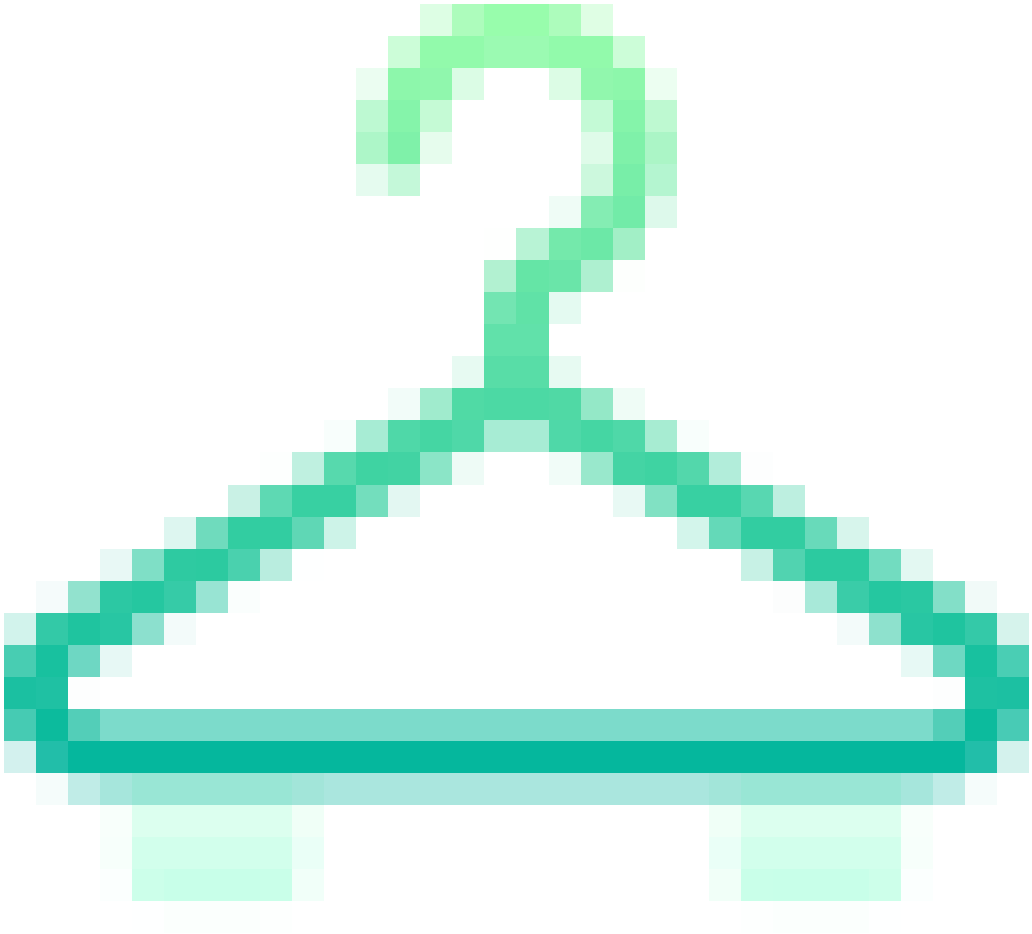
In the example above, we are combining body, cloth, face, hands and proxy parts of a single character coming from different alembic caches. Since they share the same group hierarchy, they will simply assembled into a single hierarchy chain.



1.5.2 Cleanup

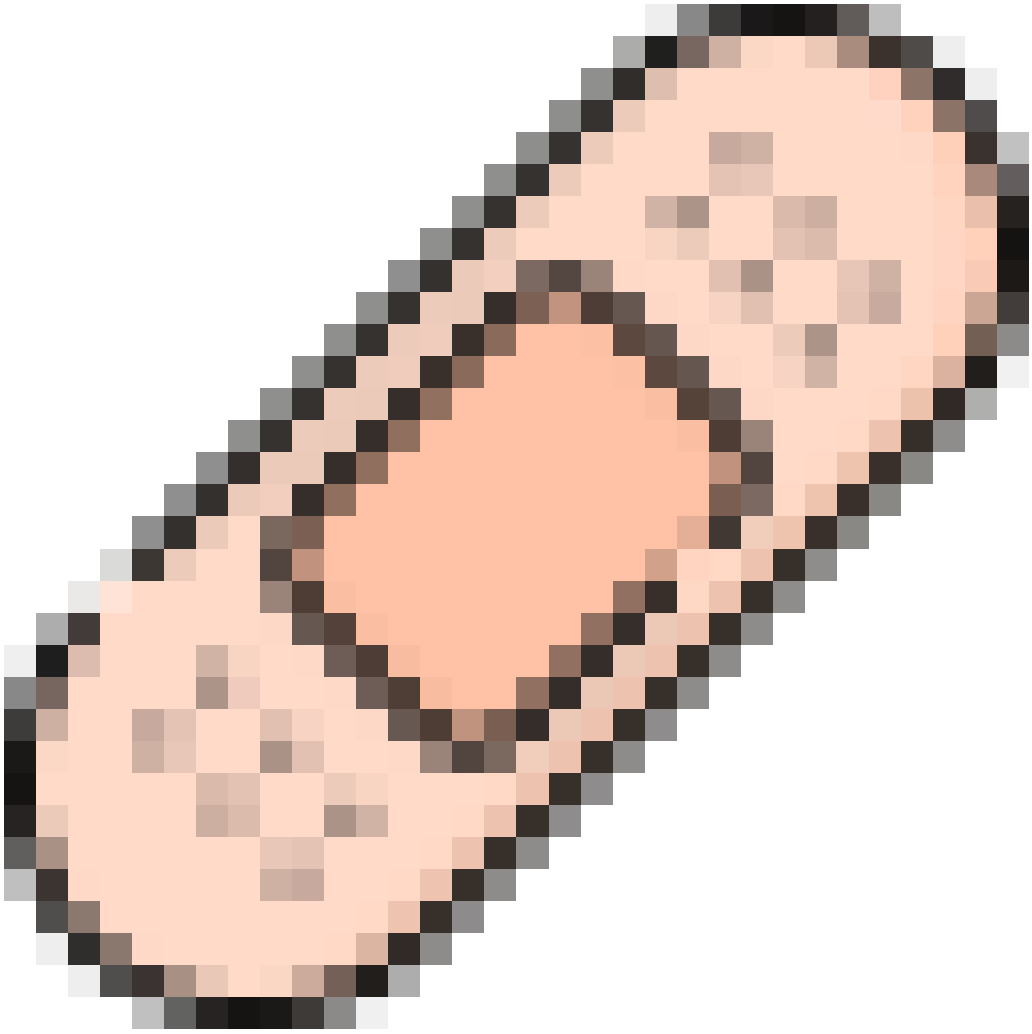
Cleanup action is an action to help keep the rig file as clean as possible. It validates the current state of the scene with checked items and tries to fix those issues automatically.

- **Unknown nodes:** Deletes any unknown nodes in the scene, usually caused by missing plugins
- **Blind Data:** Deletes the blind data nodes which is frequently comes from the game assets
- **Unused Shading Nodes:** Prunes the scene from unused shaders
- **Display Layers:** Deletes all display layers
- **Animation Layers:** Deletes all animation layers
- **Merge Similar File Nodes:** If two or more file nodes share the same file input, it keeps only the first one and makes re-connections.
- **Merge Similar Shaders:** If two or more shaders are identical, uses only first one and makes re-connections.



1.5.3 Cloth Setup

Work in progress



1.5.4 Correctives

Correctives Actions creates deltas from sculpted corrective blendshapes and implements them into the rig.

It can contain as much as definitions as you require in a single Action item. Hitting the Add New Definition button creates a new definition.

Each definition contains a set of properties:

- **Mode:** Switches between Vector and Single axis modes. Vector is useful for multi axis moving joints like shoulders. Single Axis is less resource consuming and suitable for joints moving on a single axis.
- **Driver Transform:** This is usually the bone object that we put the PSD (pose space deformer) on
- **Controller:** object which is getting used to acquire the pose that we want to put corrective shape. This must be the currently active controller. For example if there are switchable IK and FK controllers for the defined Driver Transform and the default mode is IK, then the IK controller needsto be selected. Since the PSD is getting the angle information from the joint, the same corrective will be triggered with both IK and FK once created. This is only for initial creation of PSDs and delta shapes
- **Target Transform:** Holds and displays the captured values of Controller.
- **CAPTURE button:** captures the current transform values of Controller. Move and/or rotate the controller object to the pose that you want to activate the corrective 100% and hit this button to capture the pose to be used later during rig creation Up O ject: Only available in Vector mode. Defines the up vector to align the PSD. Usually any parent node of the Driver Transform works.

- **Corrected Shape:** The sculpted shape of the corrected pose. This mesh MUST be sculpted from the captured state of target transform. This line can be left empty. In that case only the PSD will be created.
- **Skinned Mesh:** the mesh that has the skincluster applied.

> **Warning:** This Action requires the extractDeltas.py plugin to work

Sample workflow where we are putting a corrective on shoulder area:

1. Run the trigger actions until this action. Most of the rig should be functional before adding correctives. Click 'Add Definition' button for the first corrective
2. Select the upper arm joint and click the "<" icon next to the 'Driver Transform' section
3. Select the currently active controller that moves the upper arm. It should be the hand controller if the default mode is IK and upper Arm controller if FK.
4. Hit "<" next to the Controller section.
5. Pose the shoulder area into the pose that you want to apply correctives. and Hit "CAPTURE" button.
6. Select the skinned mesh. First click "<" next to the skinned Mesh section then go to the RigHelpers shelf and click the copy() item from next to the External section. This will duplicate the skinned mesh, unlock the transforms and delete the excess intermediate objects.
7. sculpt & export the mesh to an alembic cache.
8. We need to make sure the sculpted mesh is getting imported on the next run. So add an import asset or assemble action above the correctives action and define the mesh you just exported in here.
9. Lastly while the mesh you exported is selected, hit '<' in Corrected Shape row to add it into correctives definition
10. To add more correctives, hit 'Add New Definition' and repeat steps 2-9

The next time Build Rig command executed, the PSD and corrective deltas will be created on the fly.

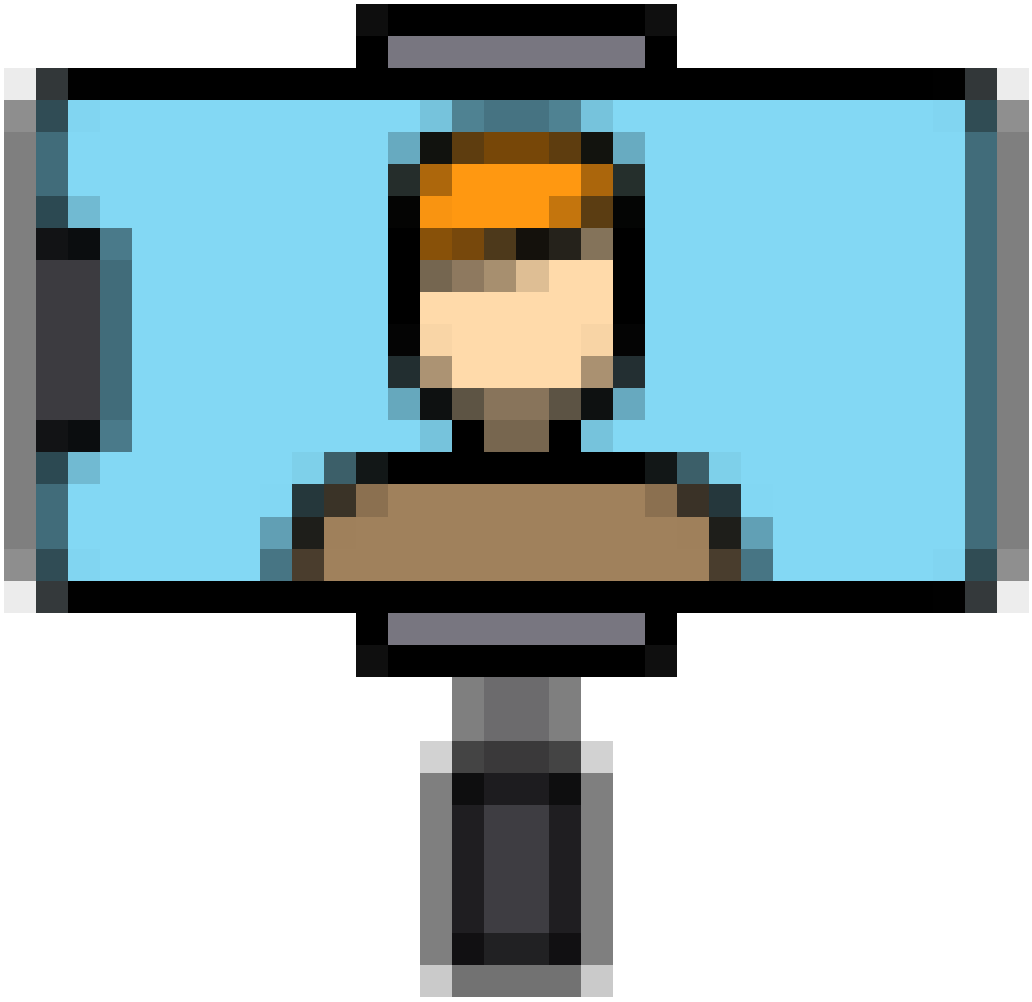
To-do's

- implement jointify to convert corrective blendshapes into joint based deformations
- A helper tool to manage & prepare corrective shape stages for sculptors which uses the correctives defined here (Will use the *.tr file)

1.5.5 Driver 

Driver action creates ranged connection between attributes and creates the missing attributes. Useful to control rig values with custom attributes and adding extra features which is not defined by kinematics.

The result is very similar to what you would get with set driven keys but instead of using keyframes this uses ranged mapped connections. Mapping row creates new mappings, changes their order, removes or duplicates them and clears the whole table. Once the 'New' button pressed, a new row appears in the table.



1.5.6 Face Cam

Face Cam action creates a camera fixed on a part of the rig. It can be rotated freely but the translation is limited within the boundaries of given mesh. Since faces are the most common use, it is named after that but can be used for any part of the rig.

Face Cam action has following properties:

- **Camera Name**: Nothing fancy, just a name for the camera. Default 'faceCam'
- **Face Mesh**: The mesh object that the camera will be focused on.
- **Parent Node**: Parent object that the camera will be constrained
- **Focal Length**: Focal length of the camera. Default 50
- **Initial Distance**: The distance between camera and mesh
- **Limit Multiplier**: The multiplier value which will define the extra transformation boundaries. For example if this set 2, the translation boundaries limits the camera will be doubled

Tip: The boundary limits of face cam is calculated by defined face mesh * limit multiplier value

1.5.7 Import Asset

Import Asset action is used to dynamically import any kind of asset into the build session. This is probably the most commonly used action in all Trigger sessions.

Any valid format can be brought in into the scene. Please note that, the import method is not using any namespaces and changes the names in case of clashes. If you want to collect assets together sharing a similar hierarchy use Assemble action instead.

Valid formats are:

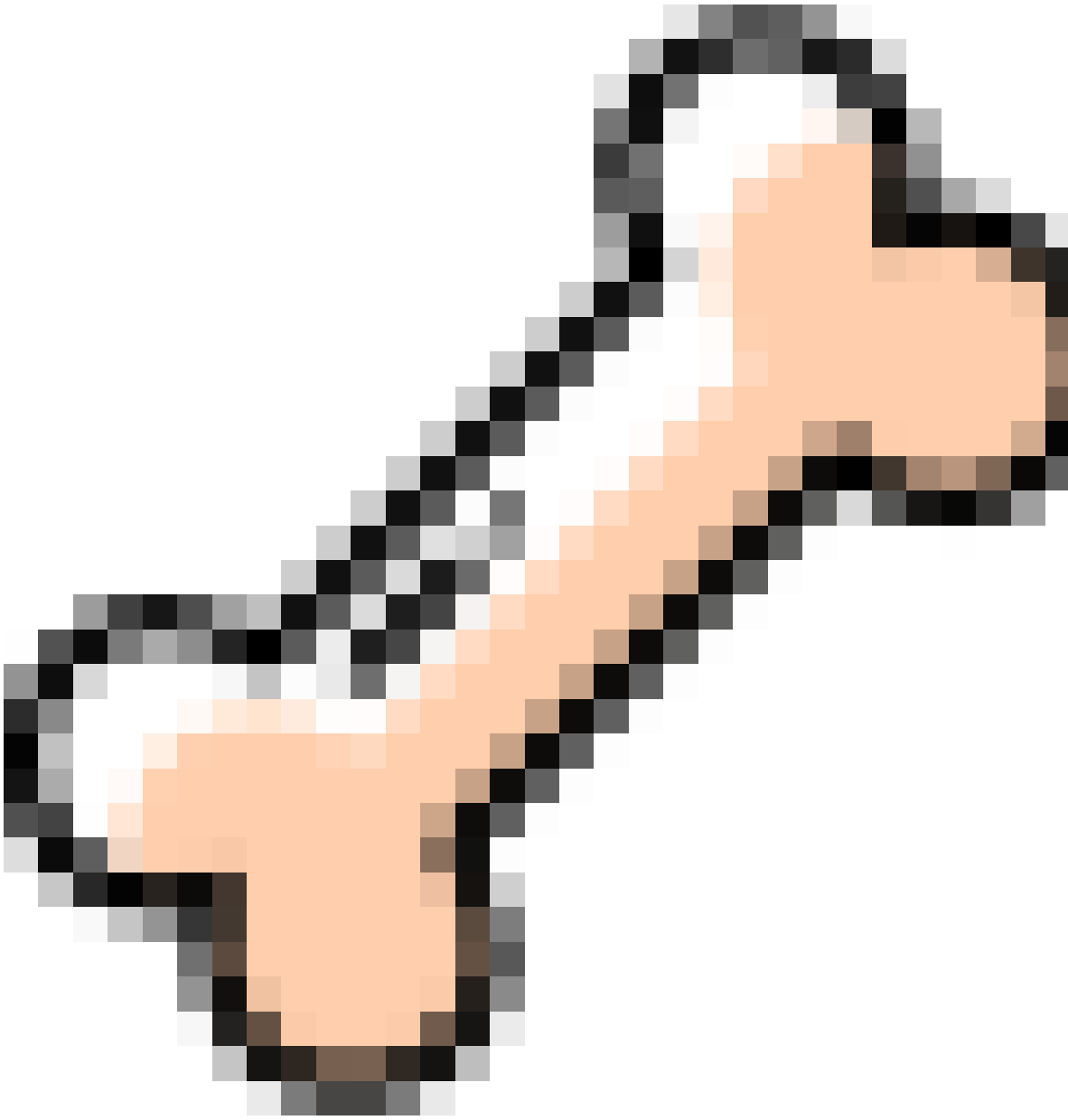
- Maya ASCII (.ma)
- Maya Binary (.mb)
- USD (.usd)
- Alembic (.abc)
- FBX (.fbx)
- OBJ (.obj)

This action has the following features:

- **File Path**: Absolute path of the asset that is going to be brought in
- **Scale**: Post Scale compensation of the asset. Always uses the world origin as scale pivot.

- **Root Suffix**: Adds the value in here as suffix to the root group of imported asset
- **Parent Under**: The imported asset will be parented under the node defined here if defined.

Tip: Import Asset Action does not use namespaces and changes names if a node already exists in the scene



1.5.8 Kinematics

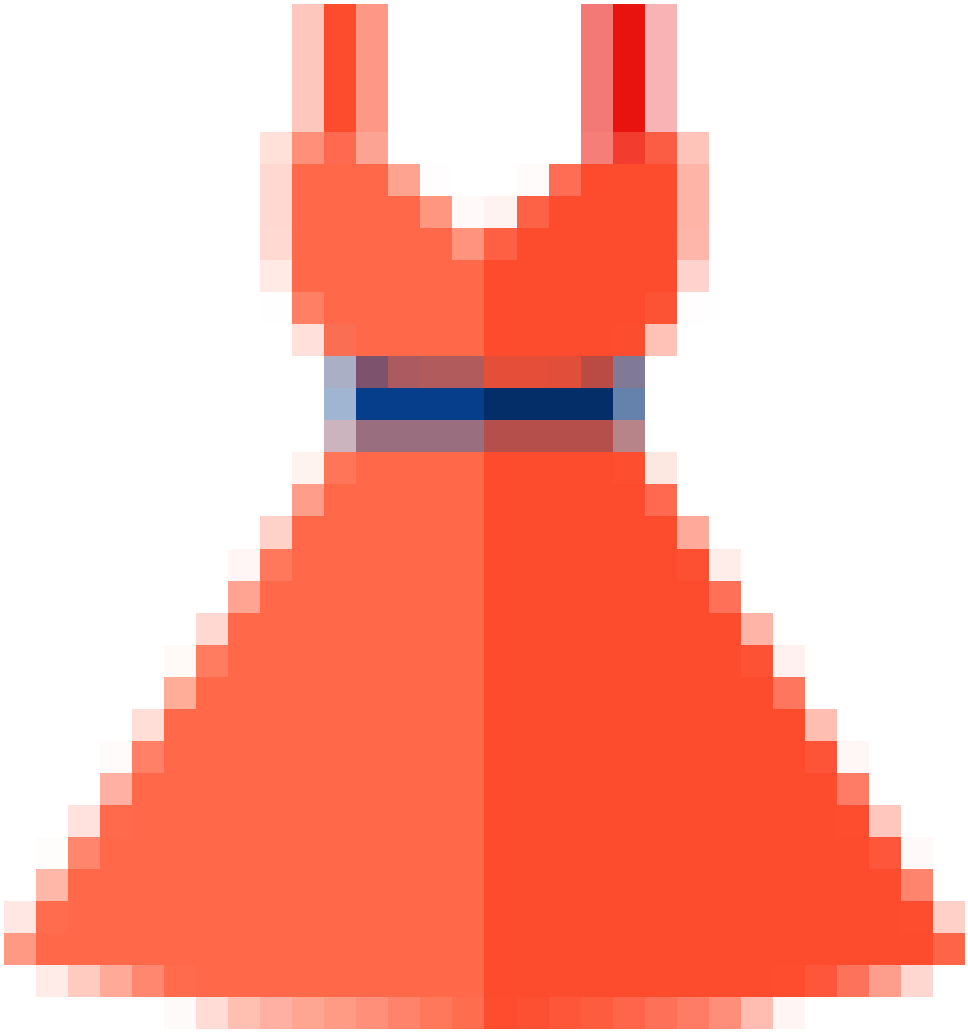
Kinematics Action is the core of rigging workflow. It uses the Trigger Guides created and exported from the Guides Tab in the main menu.

Essentially, it rigs the modular hierarchy starting from the defined root(s)

- **File Path**: The absolute path to guide file (.trg). The guides must be either created or defined from the Guides tab and exported to a file.

- **Guide Roots**: Defines where the modular hiearchy starts from. The kinematic rig will be created starting from the given root(s) and branch out to all children recursively. Multiple roots in a single guide can be defined separeted with ";" (eg. => 'jInit_LegRoot_left; jInit_LegRoot_right')
- **Get Button**: This button is an easy way to define the roots. It looks up to the trg file and drops down a list for all possible root guides in the file.
- **Create Auto Switchers**: If checked, it tries to create the space switchers automatically as defined per module. Default is on.
- **After Action**: Defines what will happen to the guide joints after Action runs successfully. Valid options are 'Do Nothing', 'Hide Guides' and 'Delete Guides'. Default state is 'Delete Guides'
- **Selection Sets**: If checked, creates the selection sets as defined per kinematic module. Default is off.

Tip: Defining a Guide Root is not mandatory. In special cases that you want to only import guide joints for later use you can leave it empty and choose "Do Nothing" from After Action modes.



1.5.9 Look

Look action stores shaders and their matching geometry information and allows it to be loaded back at any time.

The usage of this action is very simple but requires attention because it directly uses scene information. A file can easily be overwritten with wrong information.

To prevent that, always make sure that the maya outliner hiearchy matches the same state to the look action is going to run.

Essentially, the most fool-proof way is to put and keep the Look action right after the import Asset or assemble action where we bring in the geometries.

- **File Path**: The absolute path where the trigger look file (.trl) will be stored AND saved with 'Save' button
- **Save Button**: Saves all shaders and shader-mesh pair information in the CURRENT scene to the file defined in File Path. Think twice before hitting that. Use Save as or increment if you have doubts.
- **Save As**: Brings up the file browser to specify a save as location and saves it
- **Increment**: Versions up the file defined in File Path.

Warning: Be careful when hitting the save button. The current scene will be used to overwrite the file.

1.5.10 Master

Master action has no properties. Its only purpose is that to create the rig group hierarchy and preferences controller (pref_cont) which are necessary for rig publishes. If any of these already exists in the scene or have been created by

other actions (e.g. Kinematics) these won't be created again.

1.5.11 Morph

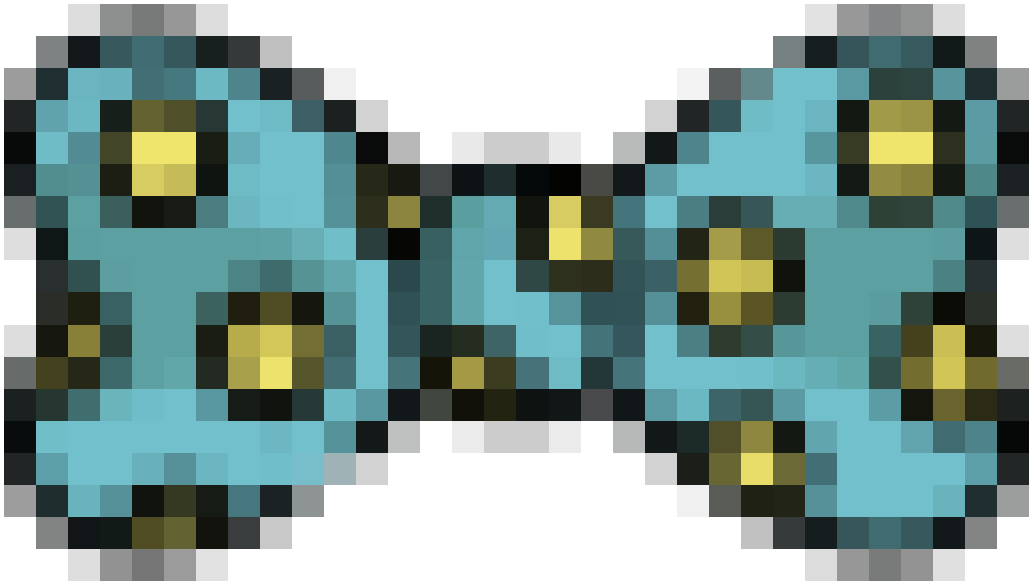
Morph Action can be used in a broad range from creating a couple of simple blendshapes to creating a complex FACS based facial rig with dozens of inbetweens and combinations.

The action strictly depends on naming conventions descriped in Blendshapes Naming Conventions.

Morph Action creates all blendshapes, inbetweens and combinations (even combinations of combinations) based on this naming convention rules.

- **Blendshapes Group**: The root group for all the involving shapes. Morph action will automatically figure out how to use which shape under this group based on their naming convention tags.
- **Neutral Mesh**: The neutral state of the mesh. This will be used to calculate deltas and a duplicate of this will be created during the process if the morph mesh is not defined.

- **Hook Node**: The controller node that will hold the connections. If the object is not exists in the scene, one group with the same name will be created.
- **Morph Mesh**: If defined, the blendshape deformer will be applied to this mesh. Must share the same topology with Neutral Mesh.



1.5.12 Node Presets

Node Presets Action can store and load any property of any node in maya.

Similar to the other save-enabled nodes (e.g look, weights) Node Presets action interacts with the currently open maya scene to store the information into files.

Press the 'Get' button to add the selected nodes into the list. When saved or incremented, all possible values of the nodes listed in there will be stored in a file.

- **File Path**: The absolute path where the trigger presets file (.trp) will be stored.
- **Nodes List**: All properties of the nodes listed here will be stored
- **Rename**: Pops up an edit line to rename the selected node in the list
- **Get**: Gets the selected nodes and puts them into list. The selection type can either be a DAG or non-DAG node.
- **Remove**: Removes selected node from the list
- **Clear**: Removes everything from the list.

- **Save**: Saves the current states of the nodes to the file
- **Save As**: Pops up a file browser to define the save as location
- **Increment**: Versions up the file and save it.

1.5.13 Reference Session

Reference Session Action references another trigger file into the current trigger session. The contents of a referenced session cannot be edited.

Example use cases:

- Separating the longer trigger builds into smaller chunks.
- Combining different parts of the same rig into one. For example putting together Face and Body trigger files into a full character rig.

1.5.14 Script

Script action is for running custom python scripts and commands inside Trigger context.

The modules imported and variables defined (even the global ones) are not accessible other than the individual action module itself.

- **File Path**: The python module or python script file (.py) which will be loaded. This can be left empty for the non-dependant simple cmds commands.
- **Edit**: This button either opens the default code editor assigned for .py files by os or pops a browser windows to select a save location for a new .py file if nothing defined in file path section.
- **Import As**: If the file defined in file path is a module, it will be imported as the word defined here
- **Commands**: multiple commands can be entered separated with ';'

Tip: simple commands may not require a file path defined. Cmds module can be imported and the com-

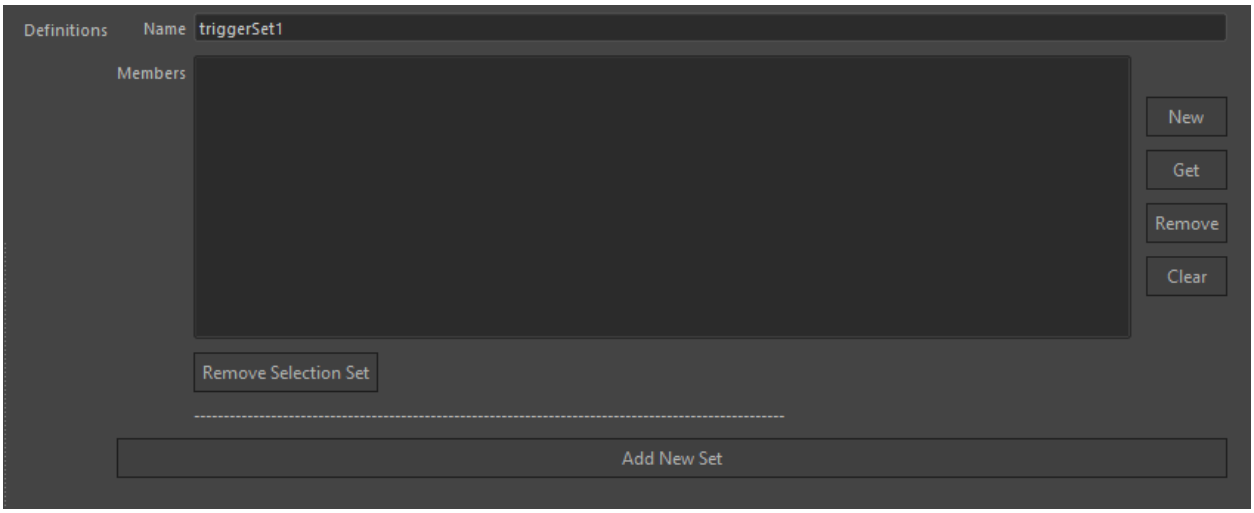
mand can run in commands section without the need of any external file like this:: from maya import cmds; cmds.delete("trigger_refGuides")

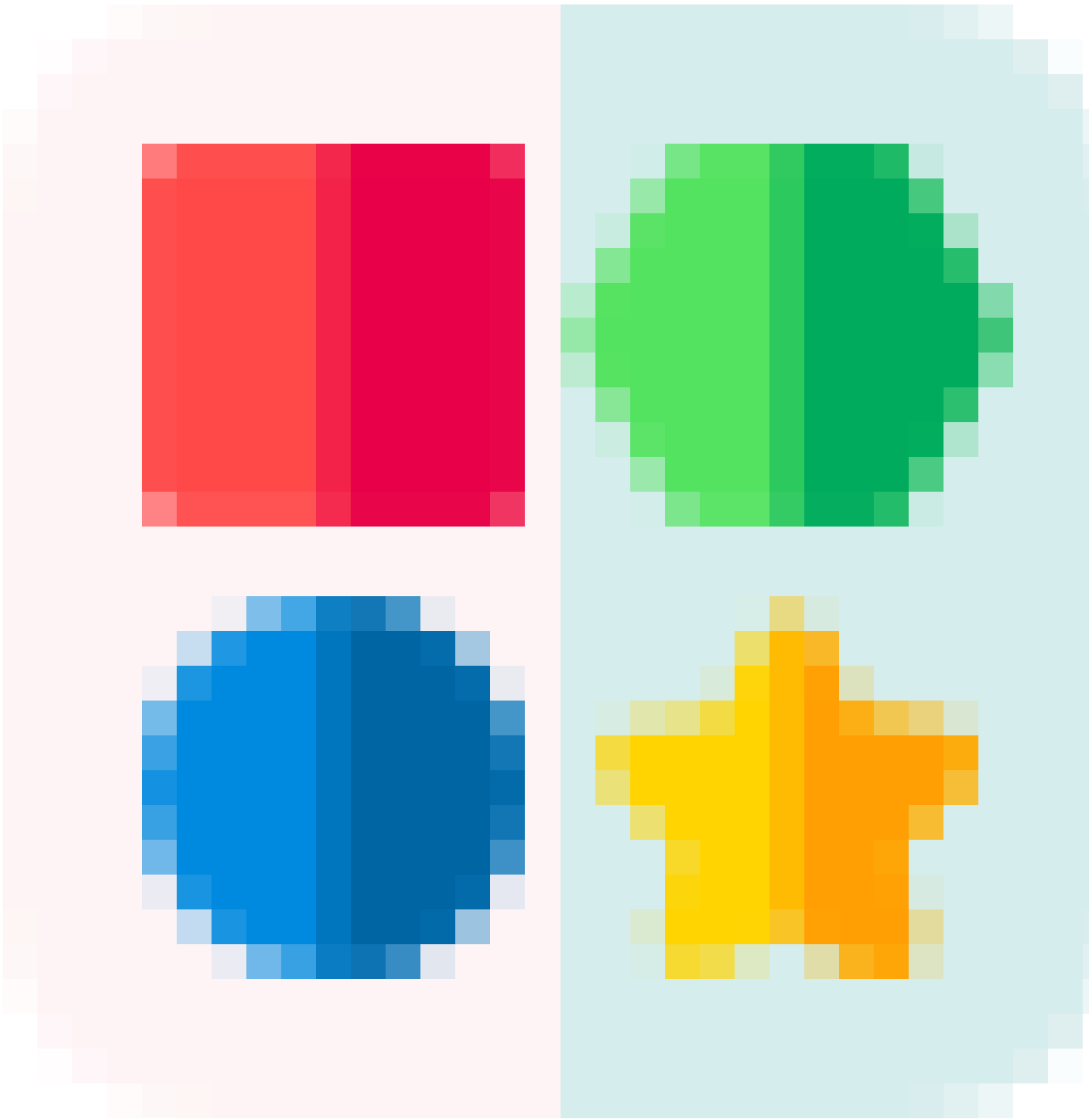
1.5.15 Selection Sets

Creates a selection set based on the defined values.

'Add New Set' button adds a new area to define a new selection set with following options:

- **Name Field**: The name for new selection set.
- **New Button**: Add a new member to the selection set by manually entering.
- **Get Button**: Adds the selected objects in the maya scene to the member list.
- **Remove Button**: Remove the selected item from the members list.
- **Clear**: Remove all items from the members list.
- **Remove Selection Set**: Removes the entire selection set from the definitions.


Definitions
Name
triggerSet1
Members
New
Get
Remove
Clear
Remove Selection Set
Add New Set



1.5.16 Shapes

Shapes Action stores and loads the edits on controller curves.

When the action runs, it only replaces the curves already existing in the scene, ignoring the rest. Similar to Look Action, it collects the current maya scene information during save. Therefore it is advised to be careful when hitting save button since it will replace the already existing Trigger Shapes file (.trs) with whatever the scene contains.

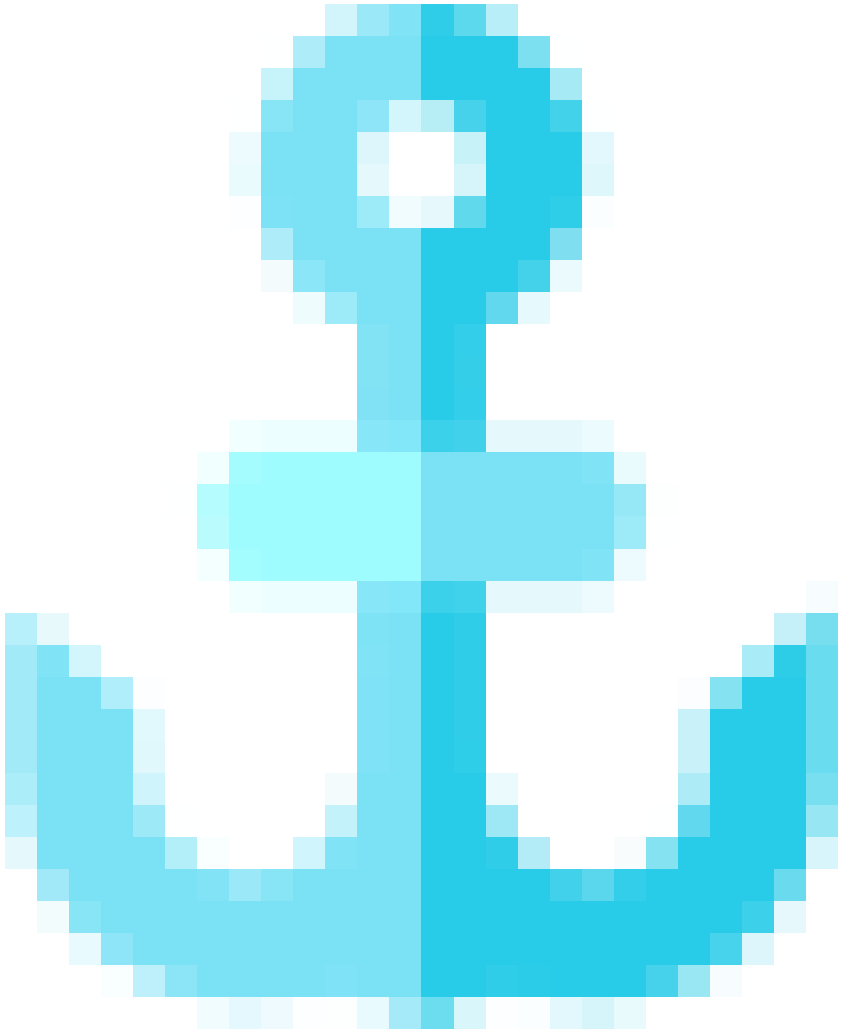
Common usage of shapes action is to edit default controller shapes created with Kinematics Action.

1. Simply drop a Shapes action after the kinematics.
2. Run the session until Shapes action.
3. Edit the controller shapes to match the rig better
4. save the Shapes file.

With the next build, these shapes will be replaced with the default Kinematics shapes.

- **File Path**: The absolute path where the Trigger Shapes file (.trs) will be stored AND saved with 'Save' button
- **Save Button**: Saves all shaders and shader-mesh pair information in the CURRENT scene to the file defined in File Path. Think twice before hitting that. Use Save as or increment if you have doubts.
- **Save As**: Brings up the file browser to specify a save as location and saves it
- **Increment**: Versions up the file defined in File Path.

> **Warning:** Be careful when hitting the save button. The current scene will be used to overwrite the file.



1.5.17 Space Switchers

Creates space switches on the rig. Any number of space switches can be defined with a single Space Switcher Action.

- **Add New Definition**: Adds a new space switcher definition.
- **Anchor**: The object that is going to be 'anchored' to different specified locations. The switch controller will be created on this object
- **Locations**: Any number of objects which the anchor object can follow.
- **Mode: Type of space switch.**
  - **Parent**: Anchor follows both translation and rotation.

  - **Point**: Anchor only follows translation of objects.
  - **Orient**: Anchor only follows rotation of objects.
- **Remove**: Removes the Definition

1.5.18 Split Shapes

Split Shapes is a blendshape action which splits the given shapes into different parts with customizable masks.

Split Shapes Action usually prepares the scene for Morph Actions. The most common usage is to gather symmetrically sculpted shapes and split them into vertical and horizontal splits to provide localized and asymmetrical controls.

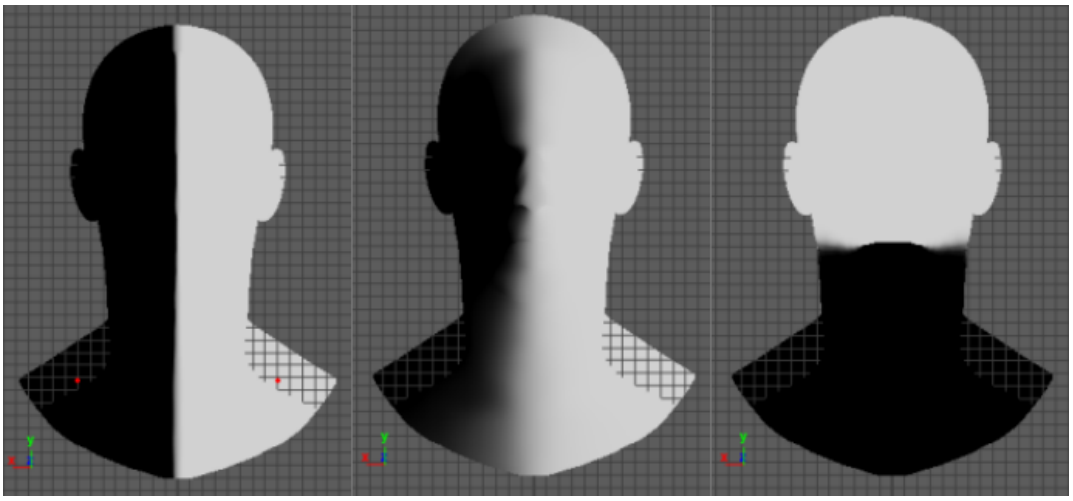
Sample Workflow:

1. Import the sculpted blendshapes pack with an Import Asset Action or Assemble Action
2. Select the neutral mesh from the Maya outliner and hit **prepare** button in the Split Shapes action. A temporary blendshape will be created on neutral mesh with 3 preset split maps. Vertical Sharp, Vertical Smooth and

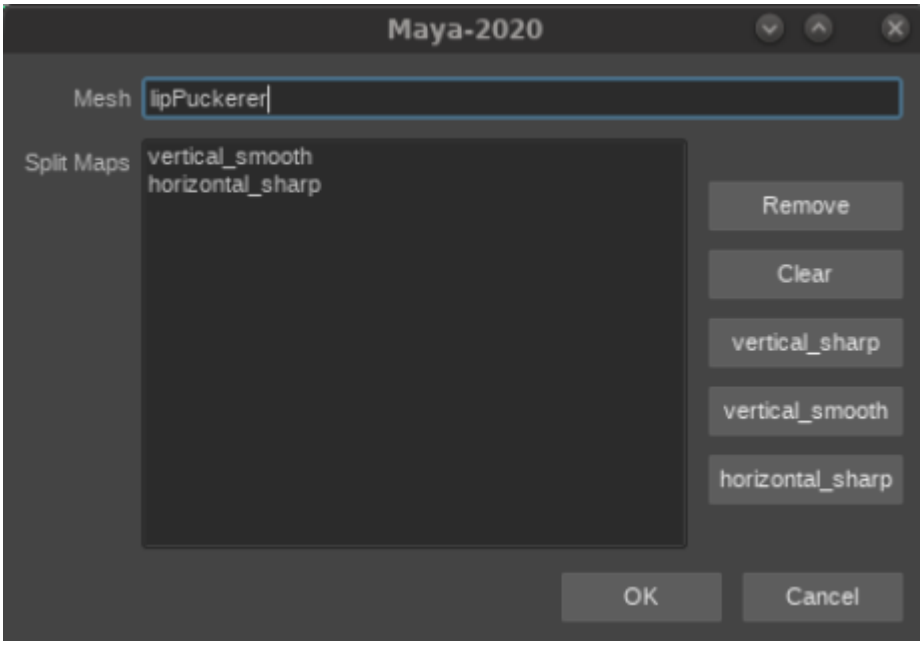
Horizontal Sharp.

3. Using Maya's paint blendshape weights tool, paint each target separately.

   In vertical maps, Left side of the character is represented by white. In horizontal maps upper sides of the character should be white. The results for Vertical Sharp, Vertical Smooth and Horizontal Sharp maps should be similar to these:



4. If more split maps are required, duplicate the Neutral mesh, rename it as the extra split map that you want (e.g horizontal_smooth) and add it as an additional blendshape target to splitMaps_blendshape node and paint it as you like.
5. Hit **Save** button and browse a filepath to save the split maps. Next time you want to edit a map, selecting the neutral mesh and hitting prepare button will bring your painted maps from this file instead of just the blank template
6. Fill the **Blendshapes Root Group** section with the name of the group which holds all the blendshapes.
7. Fill the **Neutral Mesh** section with the name of the neutral mesh in the scene. As a convention, neutral mesh is part of the blendshape pack. Even though it is in the Blendshapes root group, the neutral must be specified here.
8. For each shape in the pack, we need to make a definition. Hit **New** button and a new pop up dialog should appear.
9. In the new menu, write down the shape name that is going to be splitted into the **Mesh** section. Then from the right hand side, select the split maps that you want to apply to this shape.



In this case, we are instructiong the lipPuckerer shape to be split into 4 parts using two split maps.

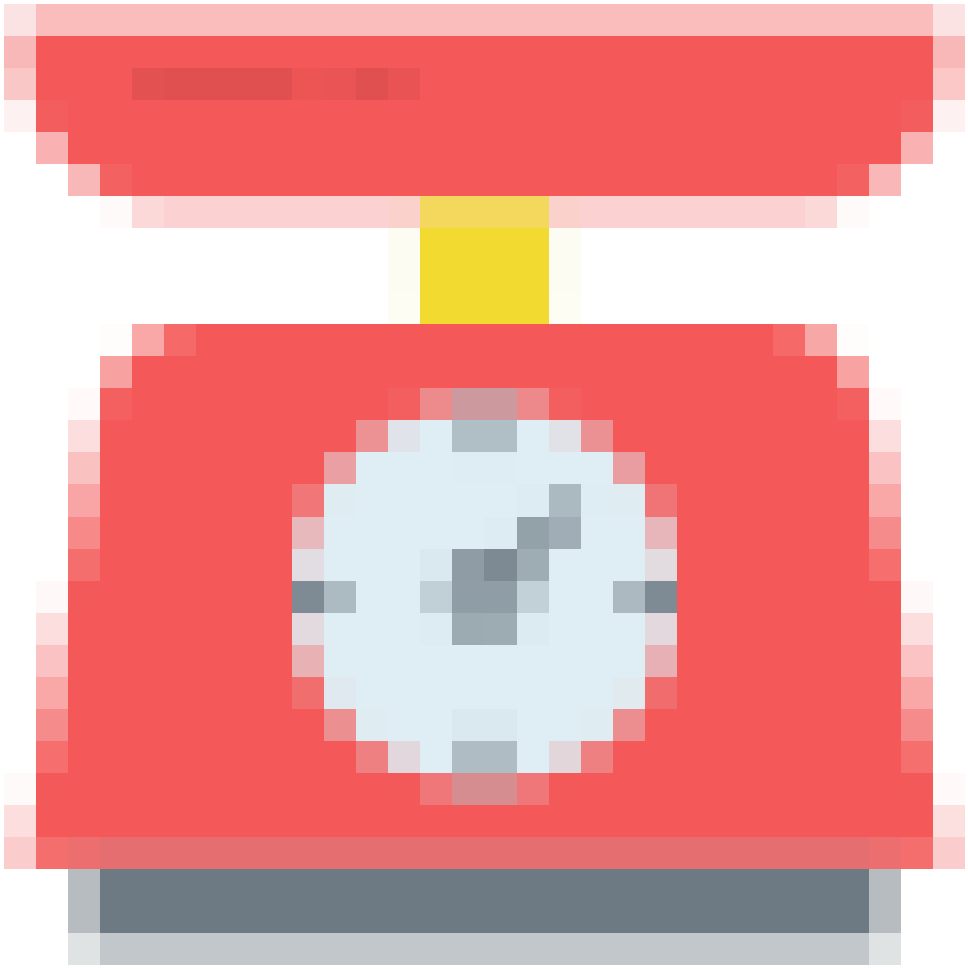
10. Hit **Ok** and repeat the same step for each shape.

Use the preset split map names (vertical_smooth, vertical_sharp, horizontal_sharp, horizontal_smooth) as it is as long as they serve the purpose and do no substitute them with custom named split maps unless you know what you are doing. The reason for that is, these will always split the shapes with the correct naming convention in a way that the Morph Action Action will understand.

For example, if we are splitting the lipPuckerer both horizontally and vertically, when the preset names are used, it will split the shape as

- *ULlipPuckerer (upper left)*
- *URlipPuckerer (upper right)*
- *DLlipPuckerer (lower left)*
- *LRlipPuckerer (lower right)*

After Split Shapes Actions execution complete during building, the splitted shapes will be gathered in a new group called SPLITTES_SHAPES_grp and the unsplit version will be deleted. Any shape not defined in the Split Definitions will stay untouched where it was.



1.5.19 Weights

Weights Action stores and restores weight values to any deformer that is defined in its list of deformers.

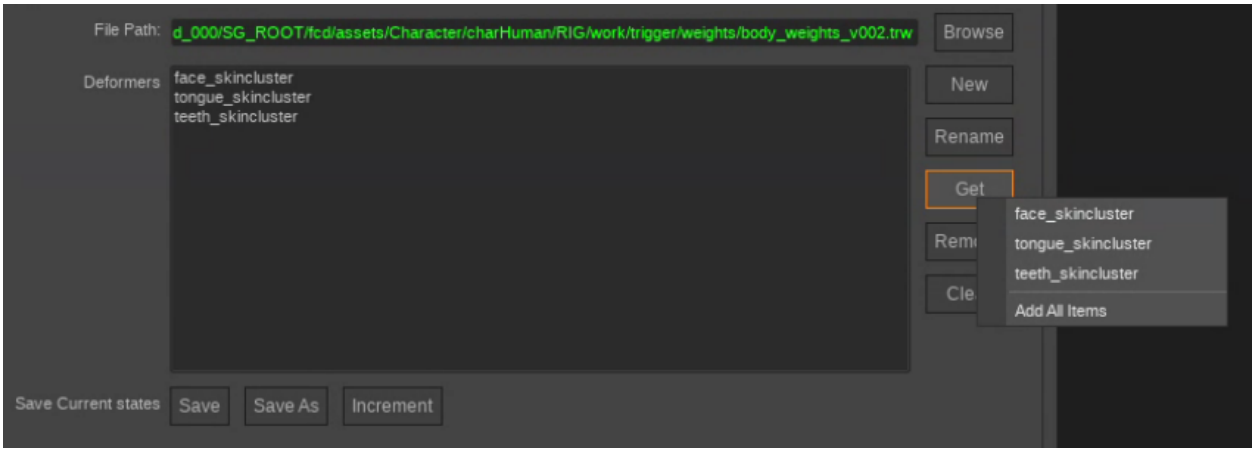
In case the deformer does not exist, it will create it.

Usage of Weights action is similar to other actions that works with Maya scene directly. Remember that all influences that will affect the deformer must be already present in the scene when this action run. For example, you cannot add weights to a character before the kinematics action because at the time the Weights Action runs, there won't be any joints to affect the skinclusters.

Ideally, the session should be executed until the Weights Action. Then you can regularly skin cluster the objects to the joints and paint the weights. Then you can define the deformers on the Weights Action and save it into a file.

The easiest way to do that is to select the mesh and hit **Get** button from the right side menu. This will drop down all valid modifiers applied on the mesh.

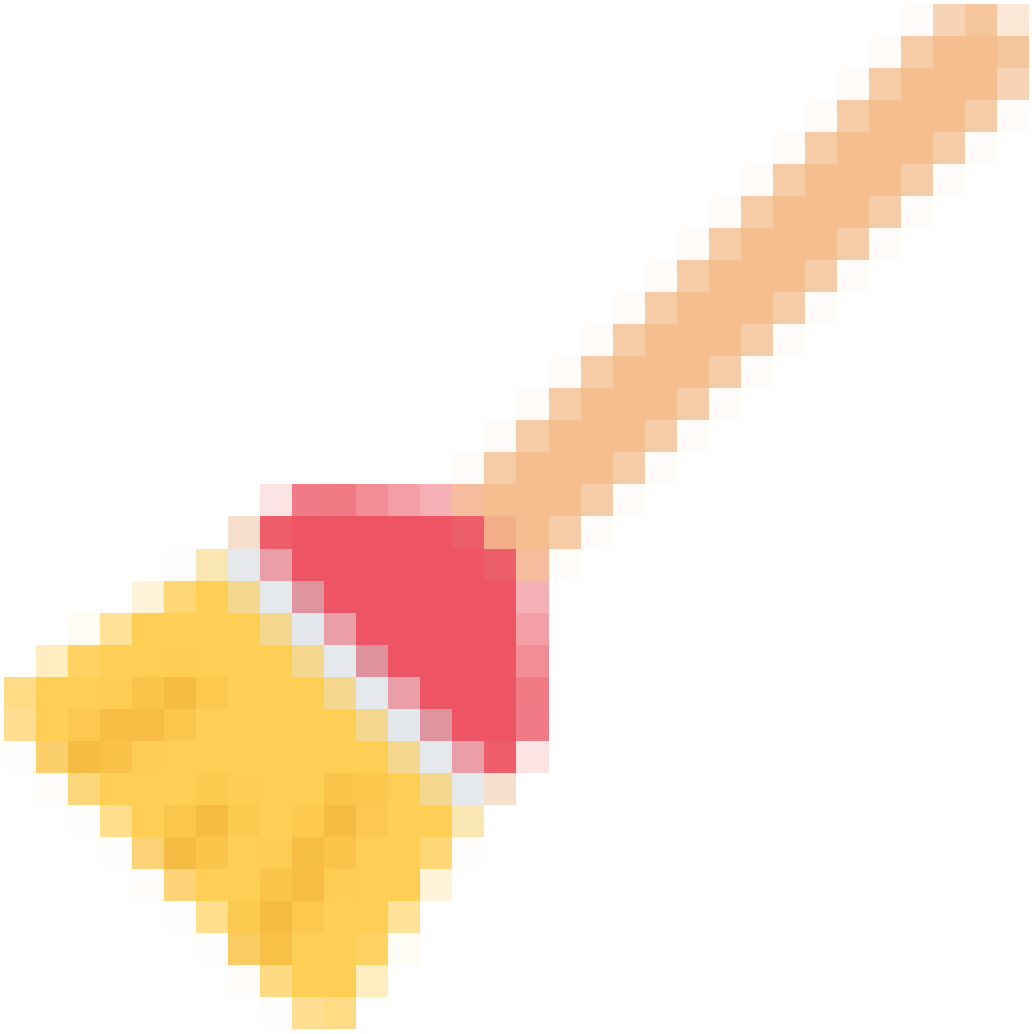
Multiple meshes can be selected and **Add All Items** command can be used from the end of the list to define multiple deformers at the same time



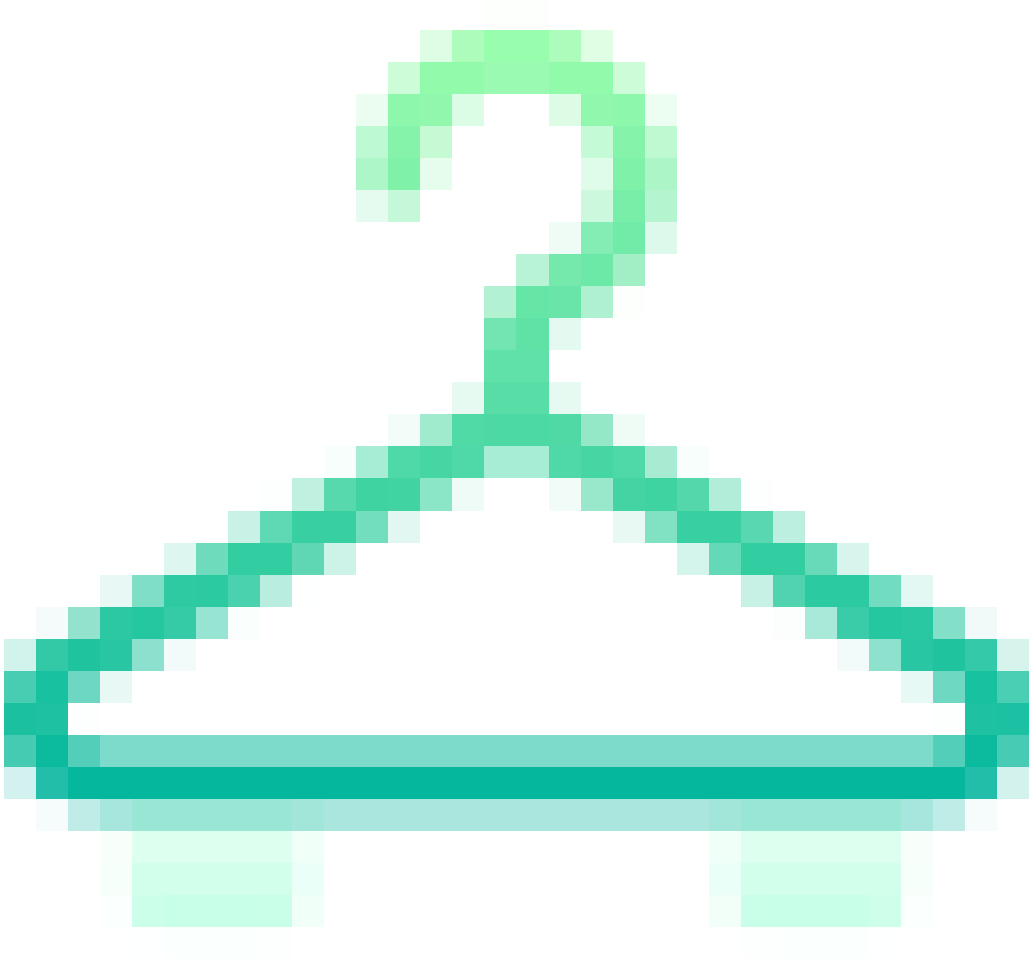
- **File Path**: The absolute path where the trigger weight file (.trw) will be stored AND saved with 'Save' button
- **Deformers**: The list of deformers currently defined in the action
- **New**: Pops up an input window which you can enter the name of the new deformer manually.
- **Rename**: Lets you edit the name of the currently selected deformer from the list
- **Remove**: Removes the selected deformer from the action definition list
- **Clear**: Removes everything from the action definition list
- **Save Button**: Saves all shaders and shader-mesh pair information in the CURRENT scene to the file defined in File Path. Think twice before hitting that. Use Save as or increment if you have doubts.
- **Save As**: Brings up the file browser to specify a save as location and saves it
- **Increment**: Versions up the file defined in File Path

Tip: Deformers are not limited with skin clusters. Currently skincluster, shrinkwrap, deltamush and blendshape deformers are supported.

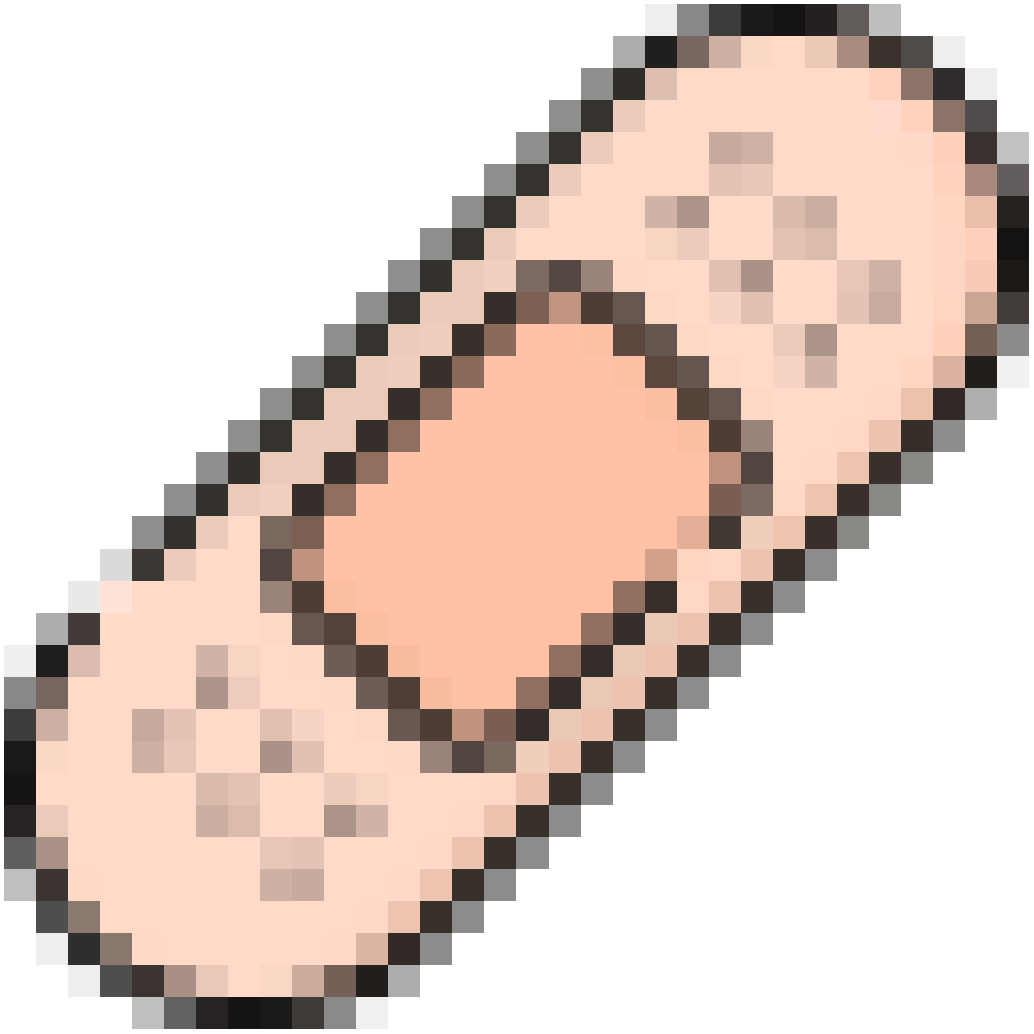
- *Assemble*



- *Cleanup*

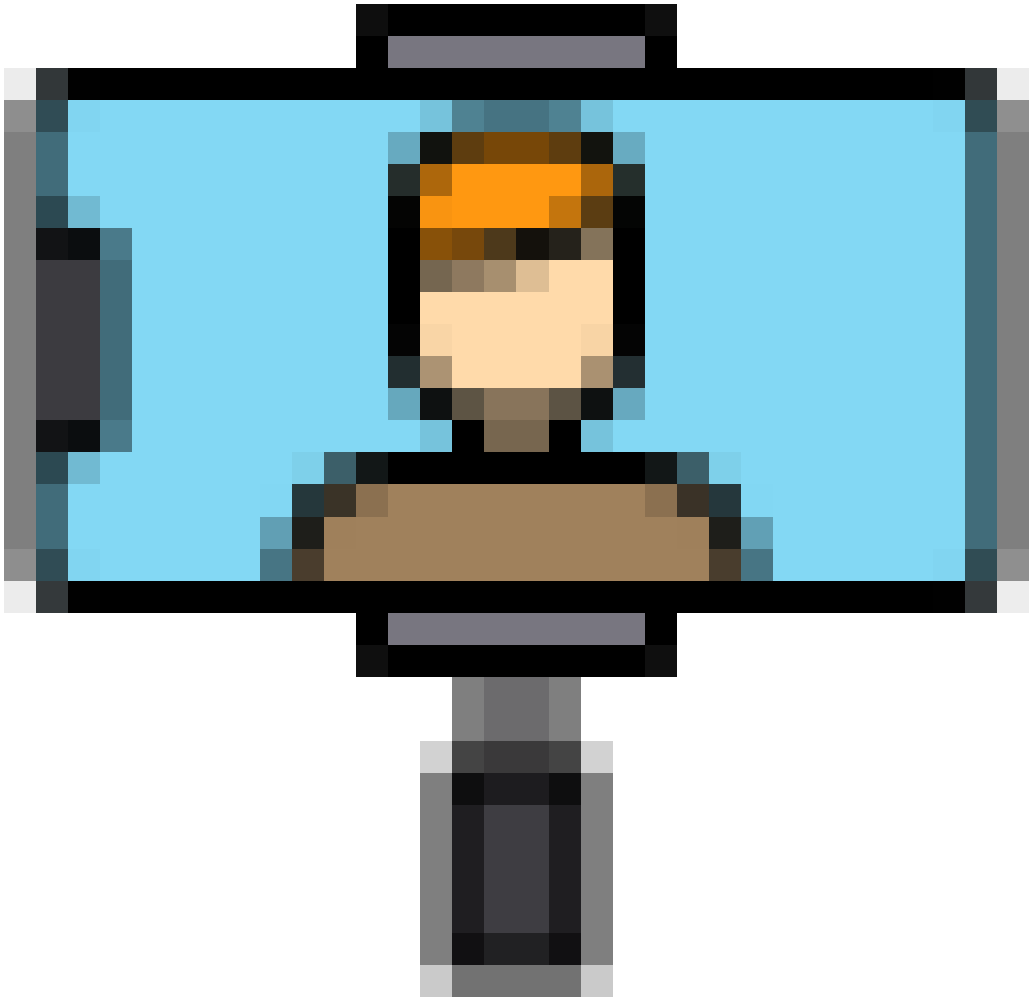


- *Cloth Setup*



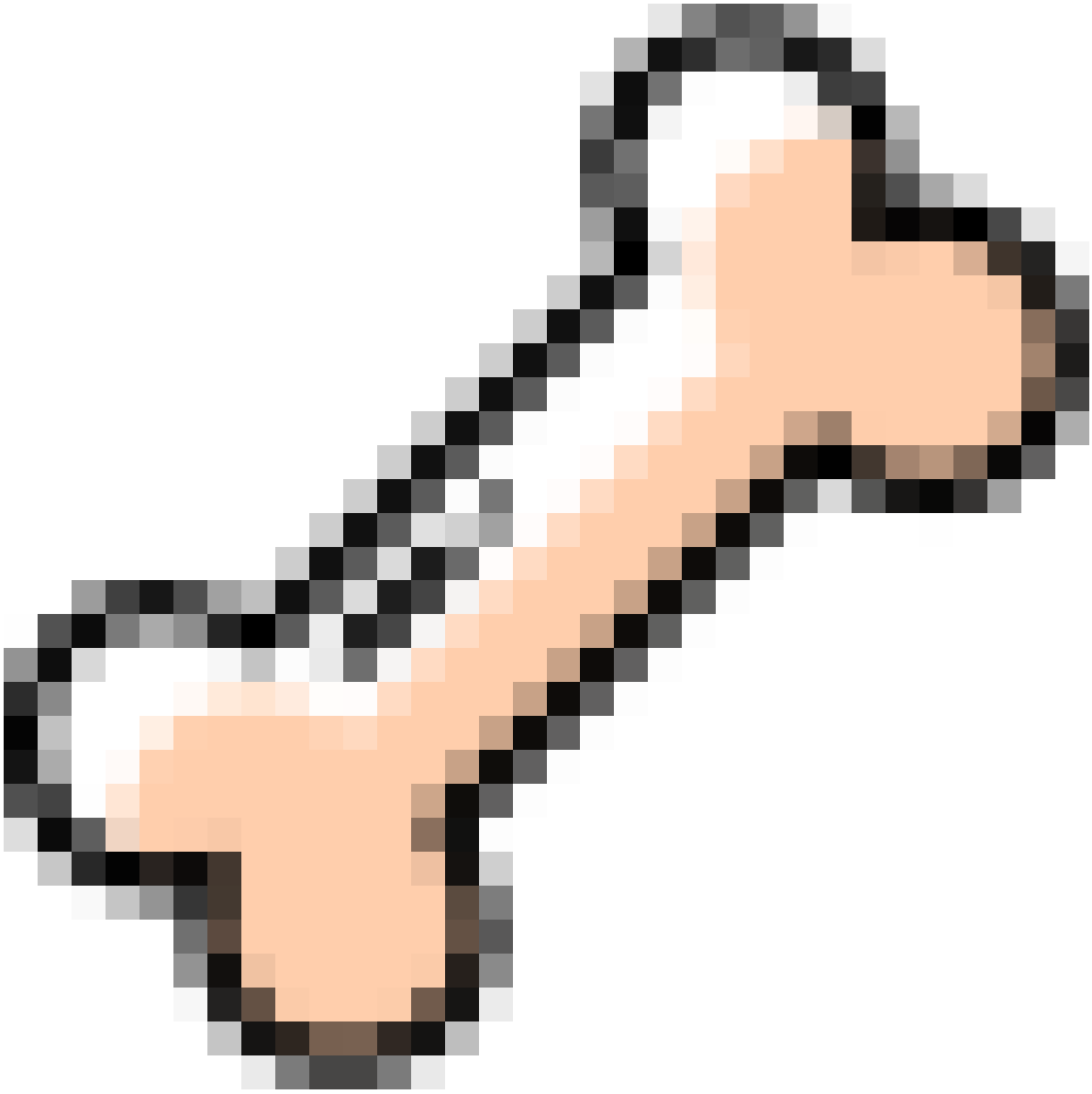
- *Correctives*

- *Driver*

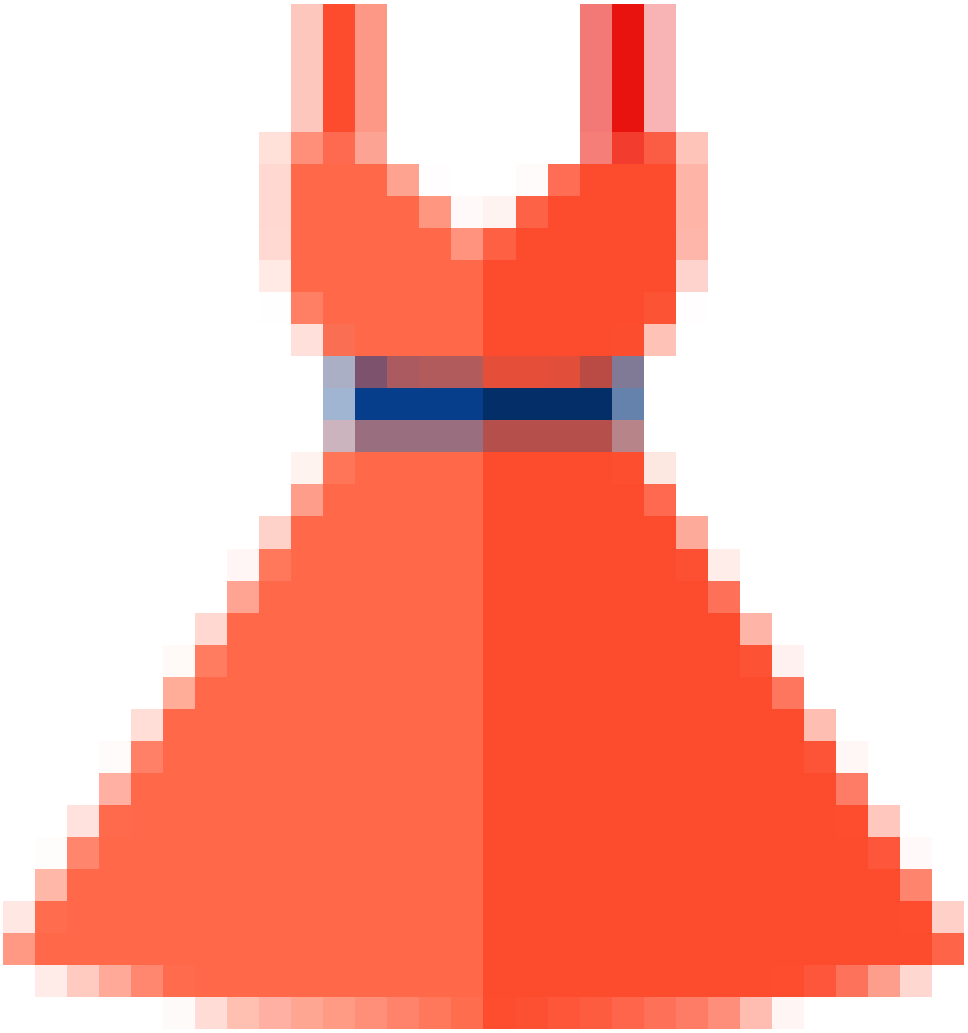


- *Face Cam*

- *Import Asset*

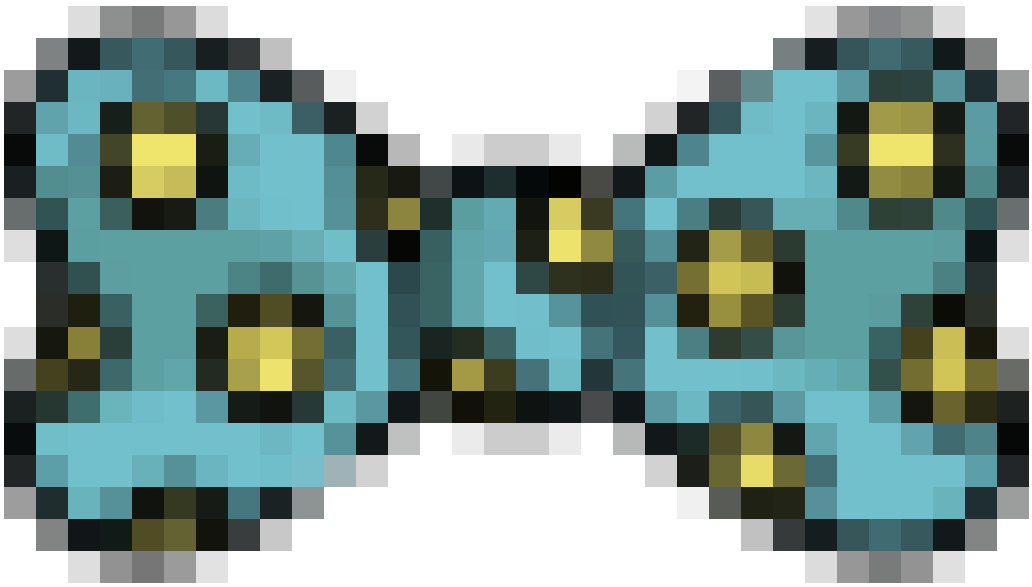


- *Kinematics*

- 
Look

-  *Master*

- *Morph*

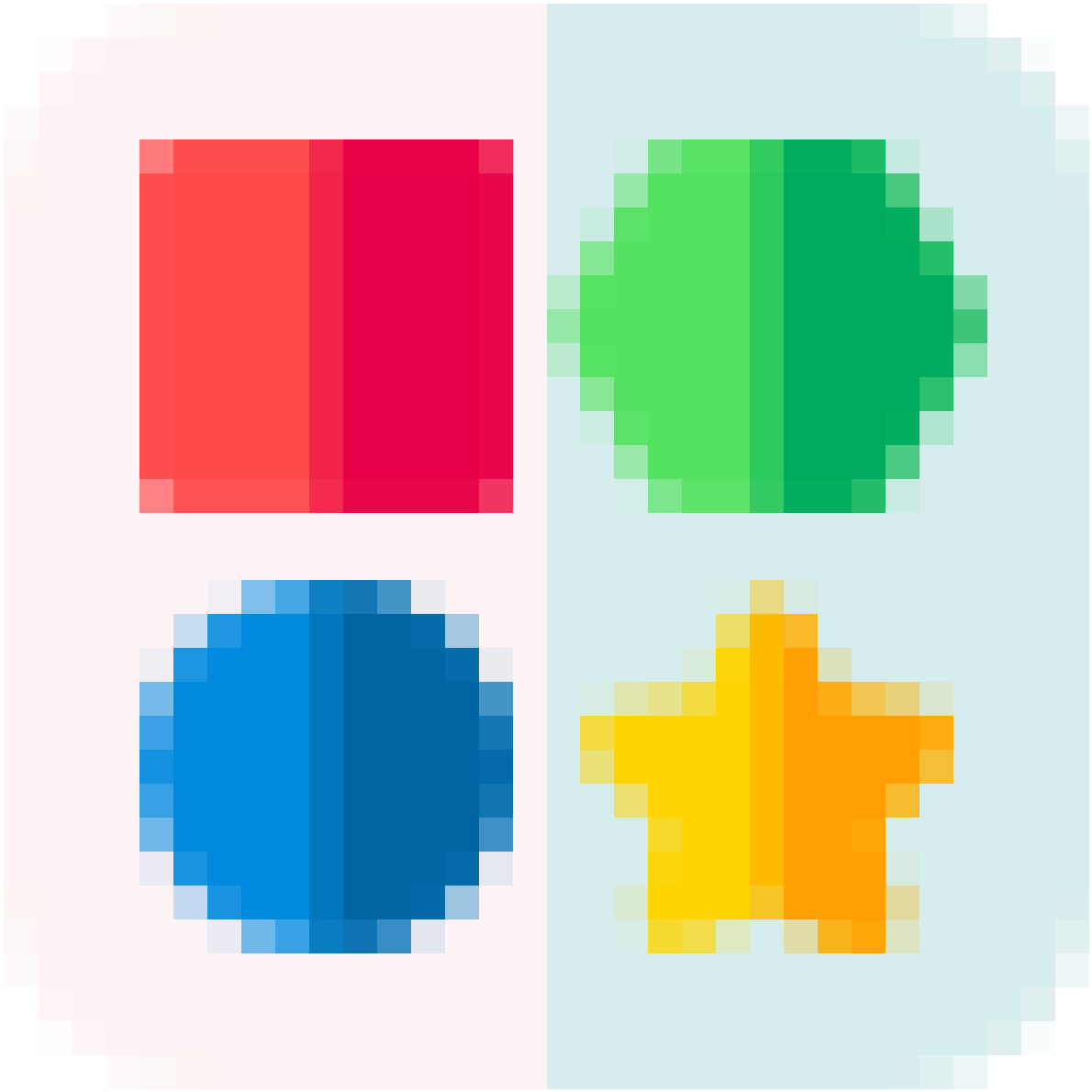


- *Node Presets*

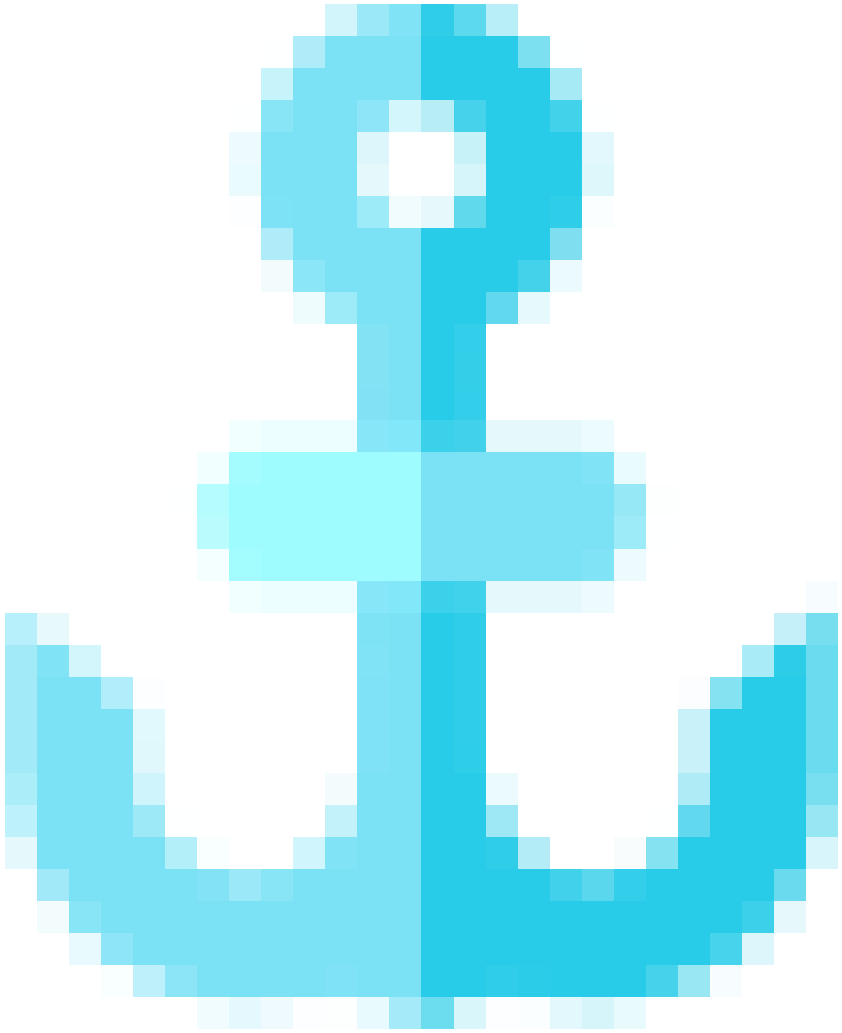
- *Reference Session*

- *Script*

- *Selection Sets*

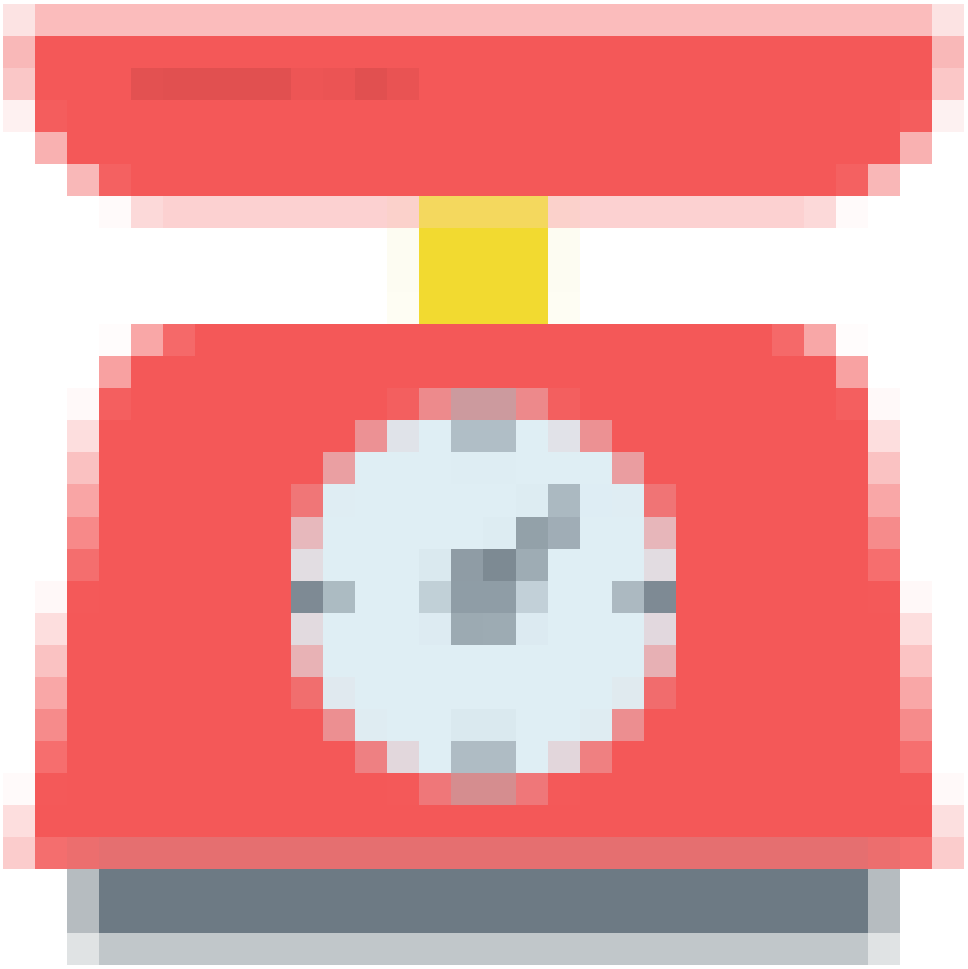


- *Shapes*



- *Space Switchers*

- *Split Shapes*



- *Weights*

1.6 Limb Modules

1.6.1 Arm

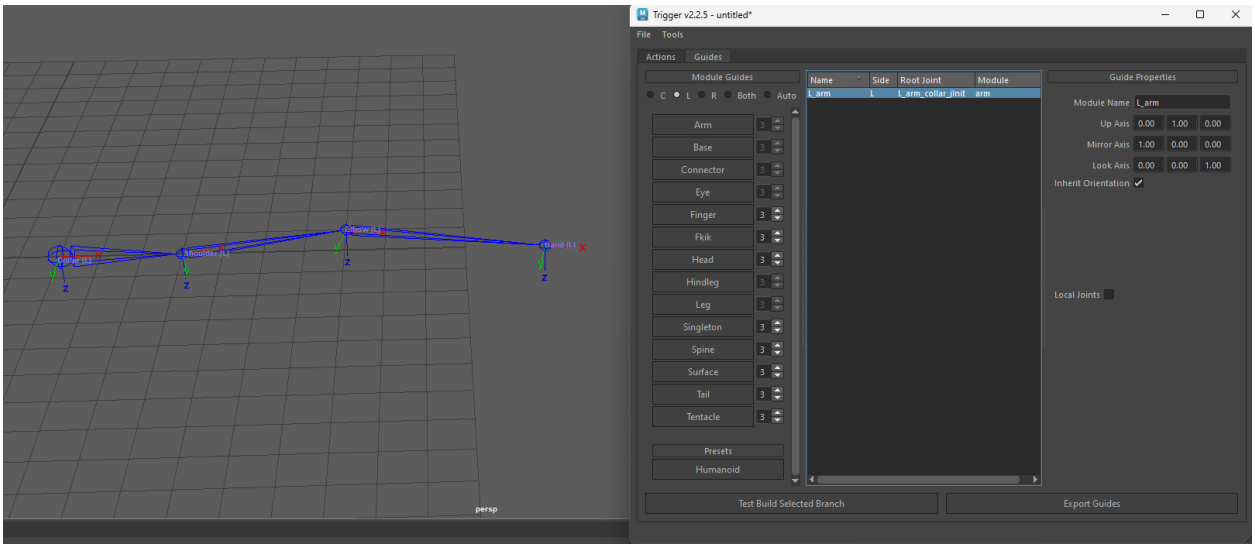
Guides

Arm Module guides needs to have exactly **4 joints**:

- Collar
- Shoulder
- Elbow
- Hand

Additional Guide Properties:

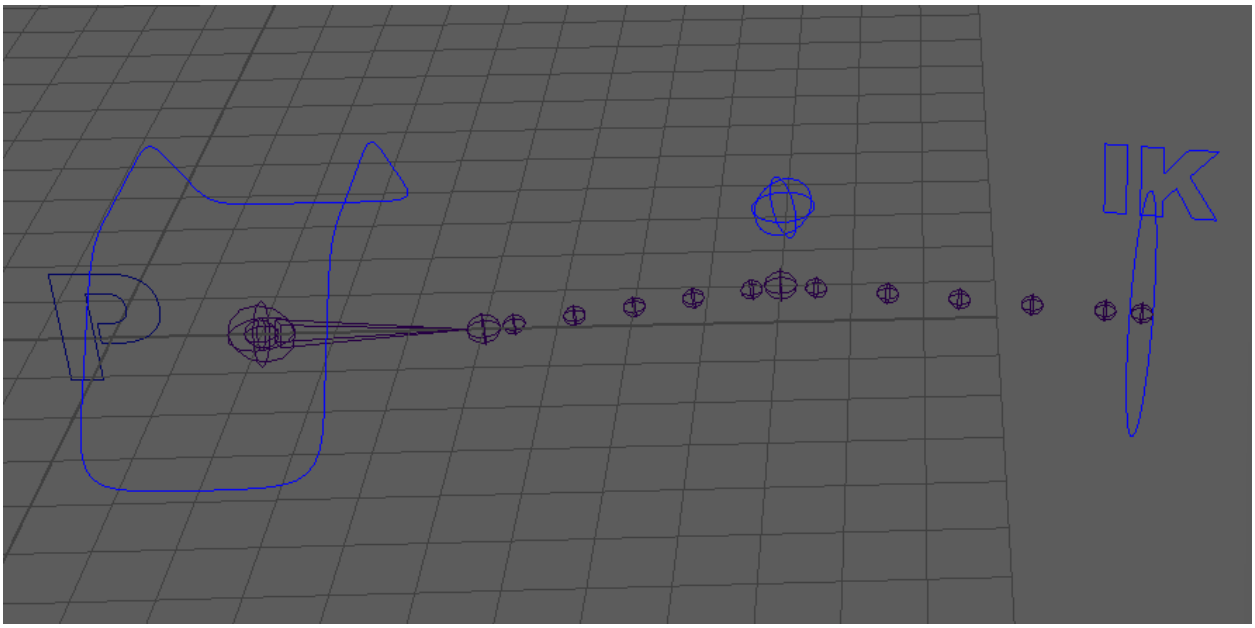
- **Local Joints**: If checked, the deformation joints wont follow the plug, keeping the rig localized. This function is useful where you want to move the controllers with the character but deform a separate geometry locally.



Rig

The rigged module contains 13 deformation joints (10 x ribbon joints, Hand, Collar, Elbow) It has the following features:

- IK/FK switch
- Stretchy (and squashy) IK with volume preservation option
- Soft IK
- Switchable between Rotate Plane Solver and Single Chain Solvers (Pole Vector Attribute on hand controller)
- Elbow pinning
- Upper and Lower arm can be independently scaleable
- Auto Shoulder movement
- Animateable shoulder alignment
- Auto/Manual Twists for Hand and Shoulder
- Extra Tweak Controls

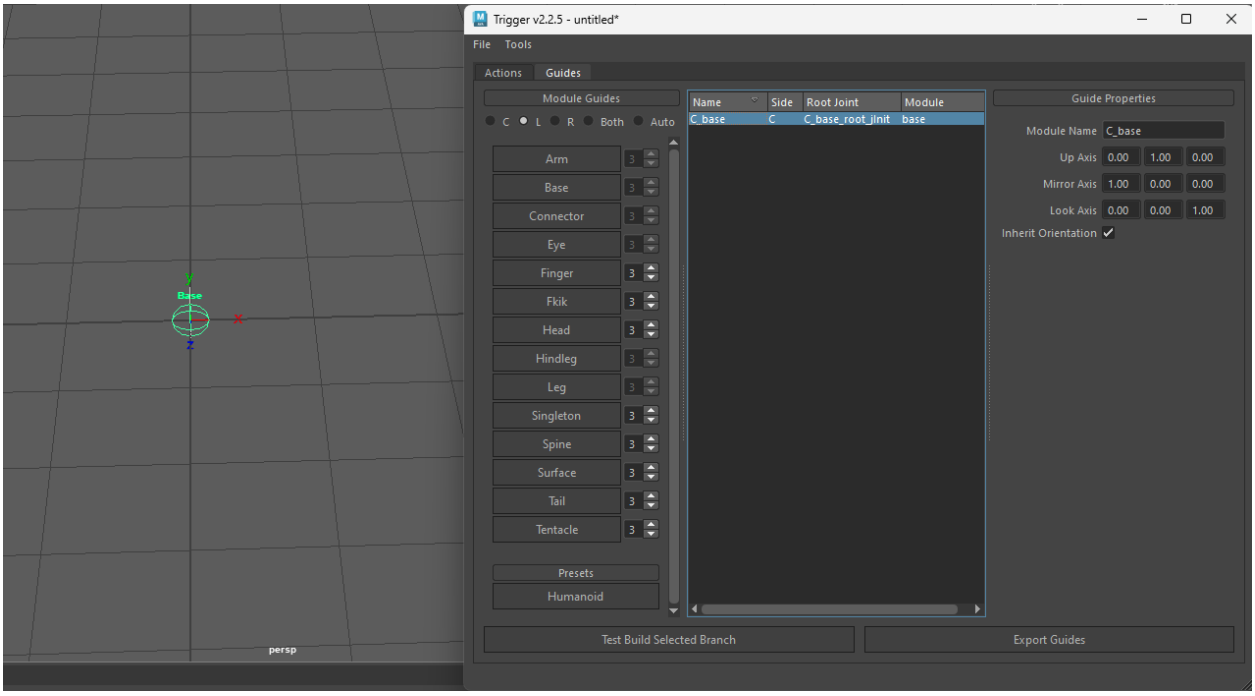


1.6.2 Base

Base module provides two extra nested controls for all child limbs below i.e 'placement' and 'master' controls of a character. As the name suggests, although not mandatory, this module is useful as a base of any rig.

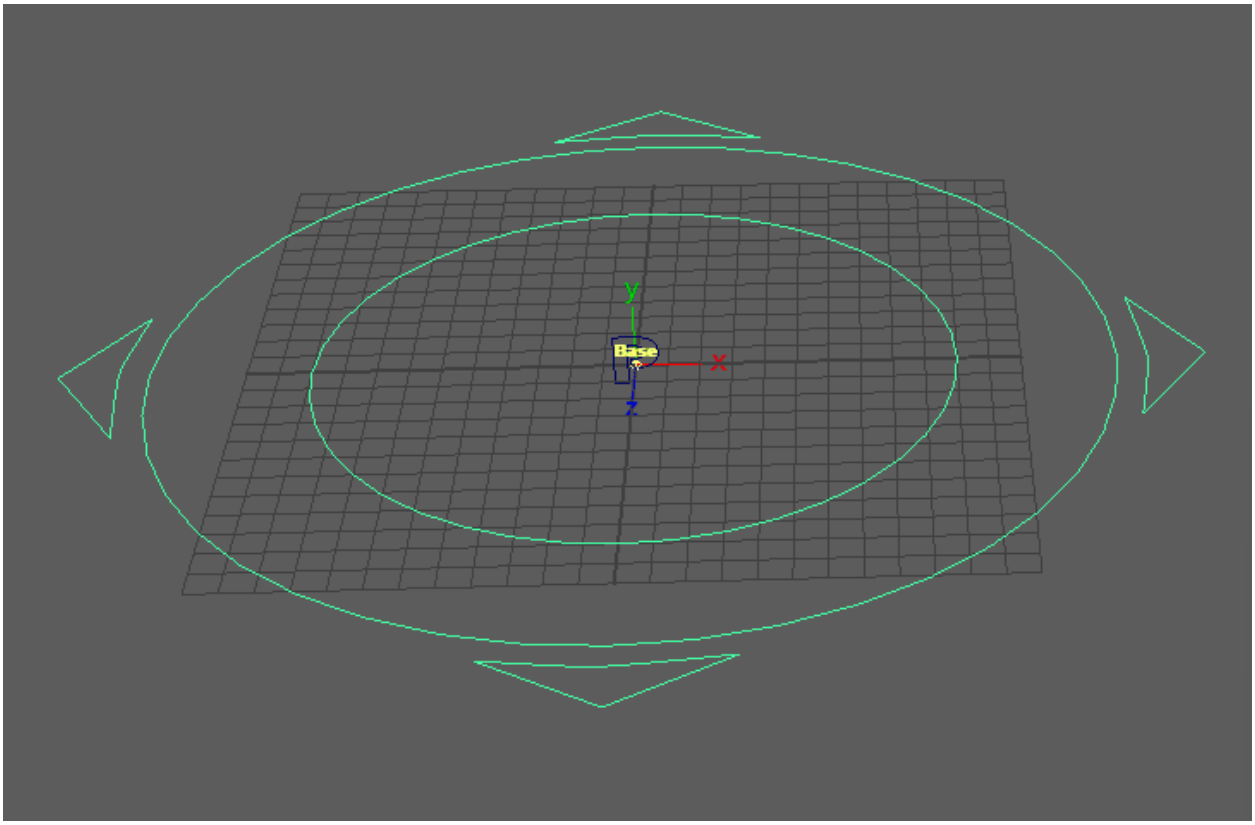
Guides

Base module needs a single joint as guide.



Rig

When rigged, Base module contains 2 nested controllers to drive rest of the limb modules underneath



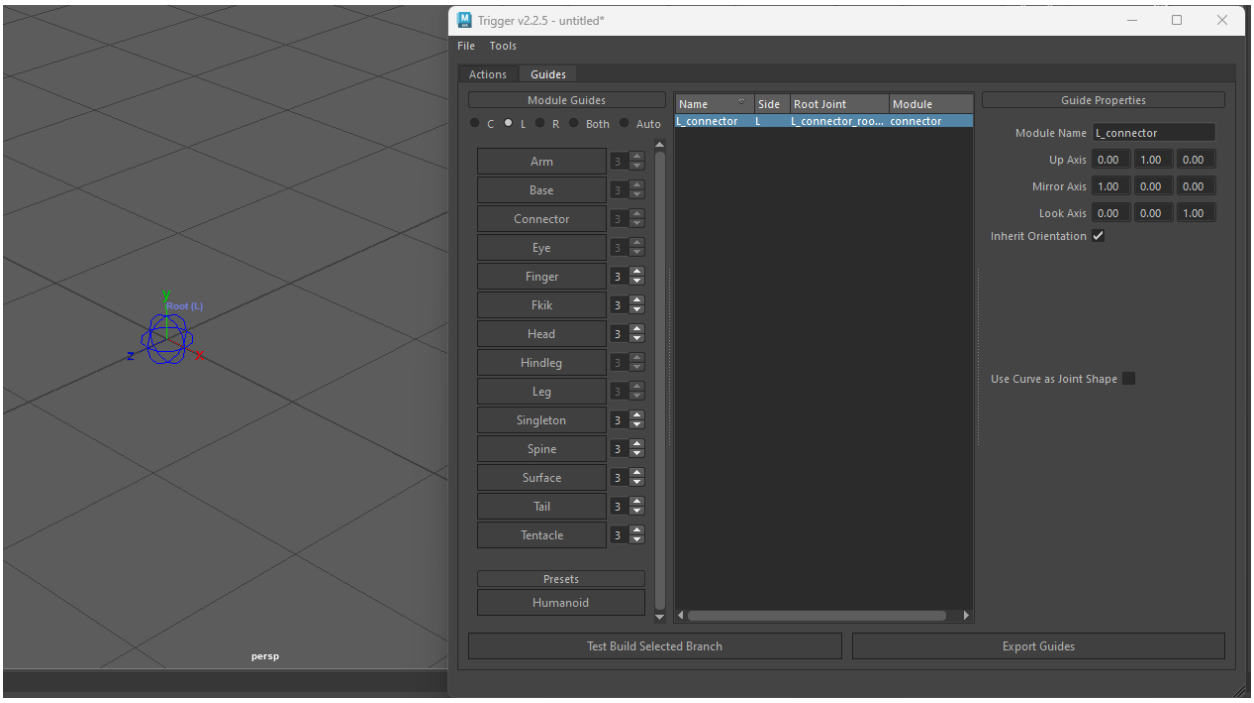
Tip: Since Base module essentially an utility module it contains no deformation joints (jDef) by default.

1.6.3 Connector

Connector is essentially a bridge module. It creates a mid-connection. This single joint will act as a socket for other limbs to connect to. Additionally, the shape can be defined as a curve if wanted to be used as a simple controller

Guides

Can only have a single guide joint.

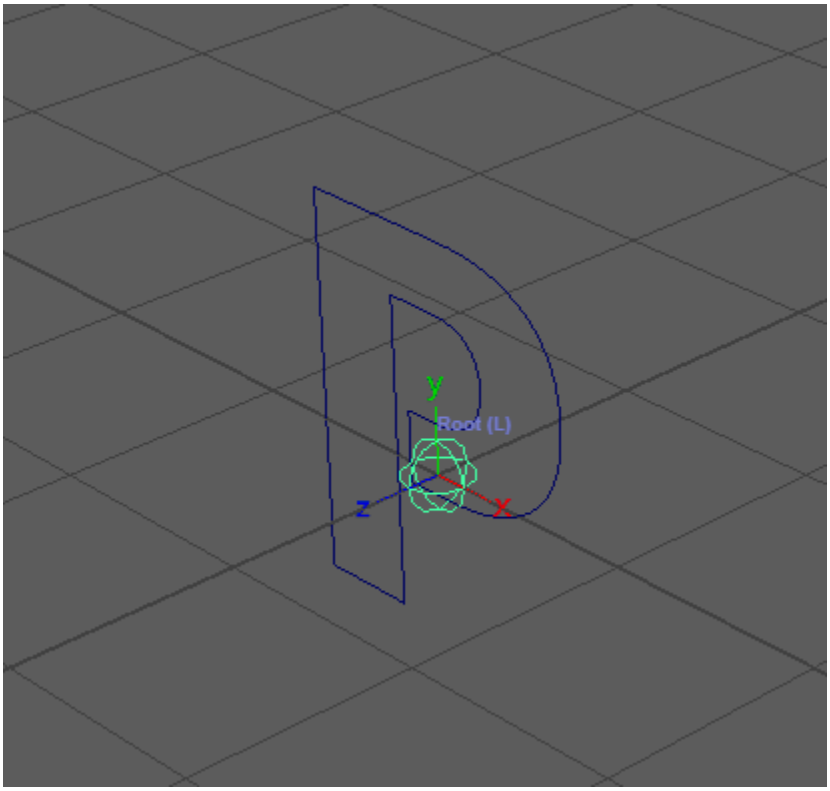


Additional Properties:

- **Use Curve as Joint Shape**: If checked, a curve shape (box) will be used as shape to be used as a controller

Rig

The rig doesn't have any features.



1.6.4 Eye

Eye module is a neighbour-aware module. This means, in rigging time, the module will look for sibling eye modules sharing the same group id and will create a single master controller for all of them in addition to individual controllers.

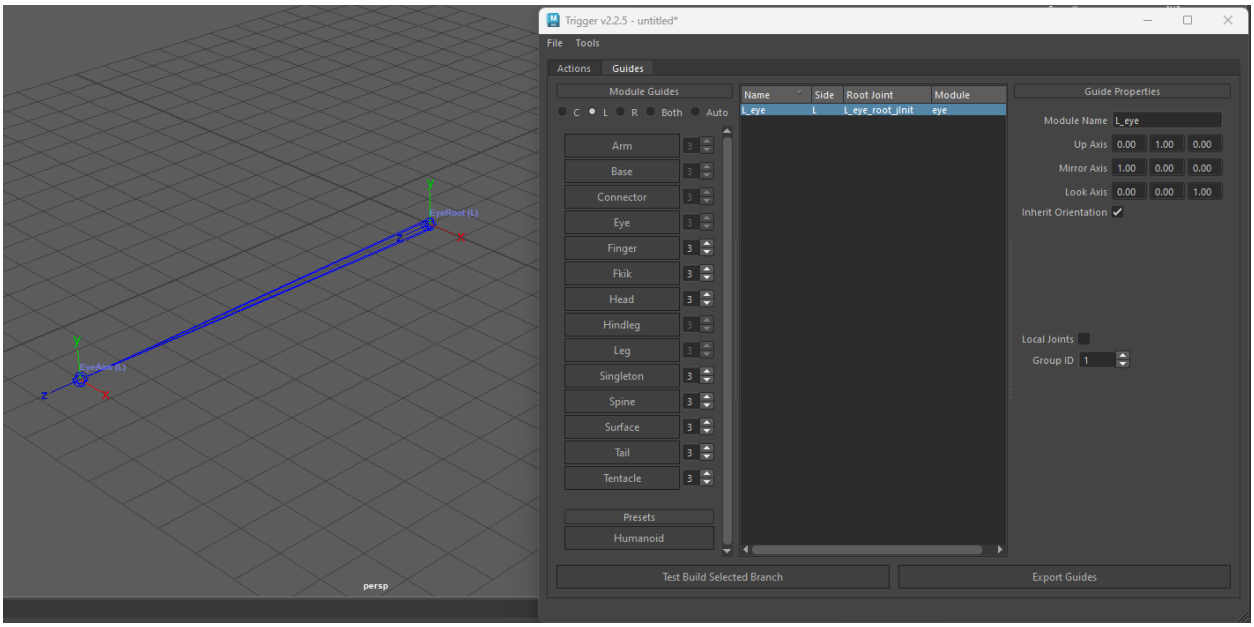
This means, as long as the guides parented to the same limb-end any number of eyes or eye-like rigs can be created

Guides

Eye Modules must contain exactly 2 guide joints. First joint defines the position of the eye and the second one defines the initial aim orientation.

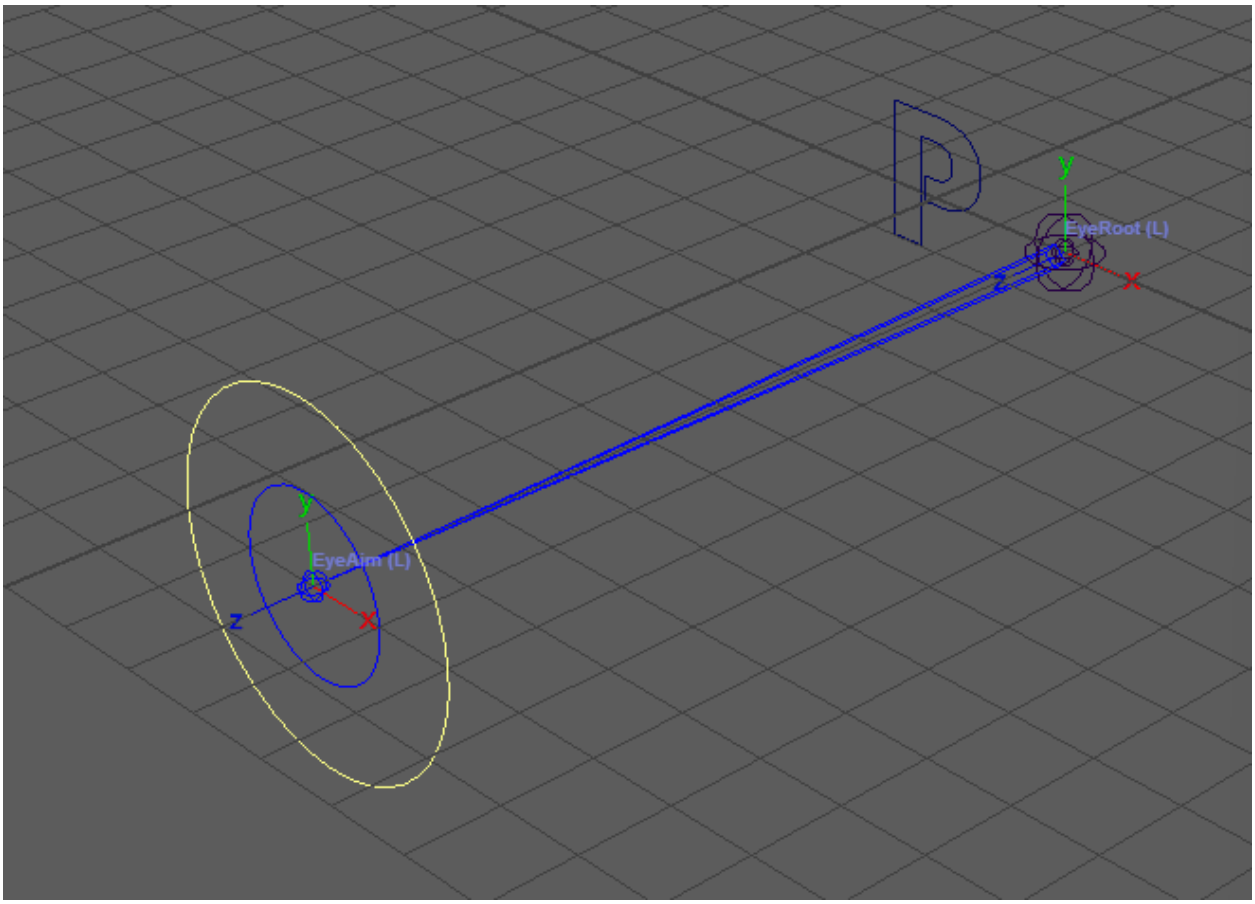
Additional properties:

- **Local Joints**: If checked, the deformation joints wont follow the plug, keeping the rig localized. This function is useful where you want to move the controllers with the character but deform a separate geometry locally.
- **Group ID**: When there are multiple eyes in the rig, the ones have the same Group ID, will share the same outer controller.



Rig

Each rigged eye module contains a single deformation joint



1.6.5 Finger

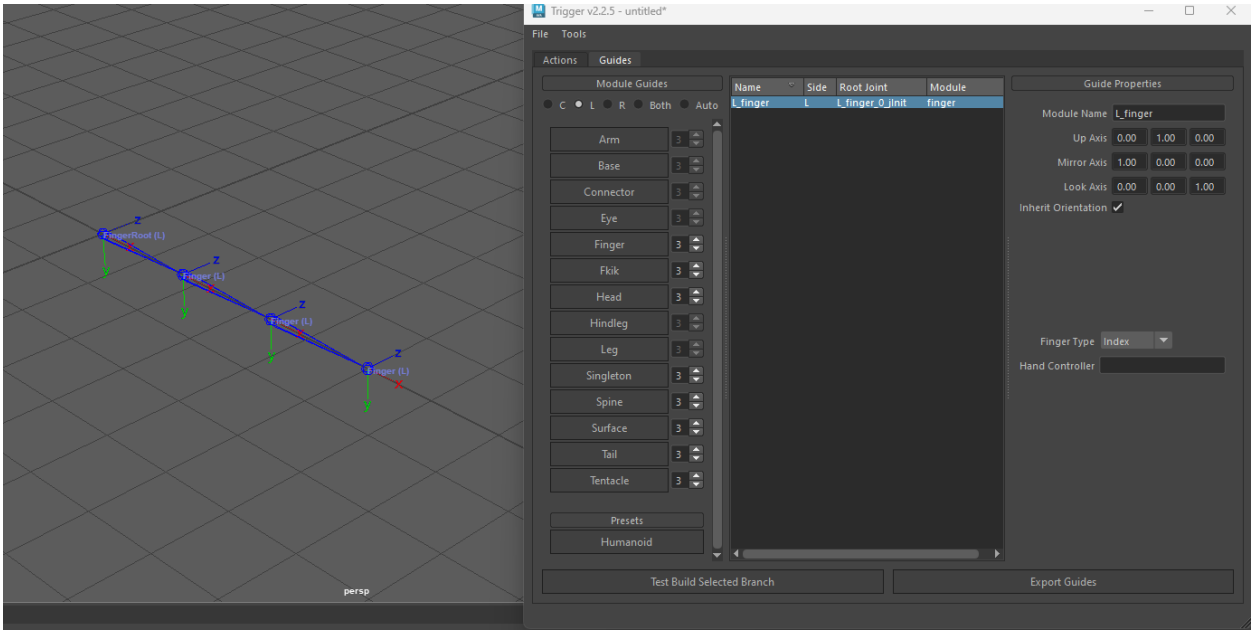
Finger module is a neighbour-aware module. The fingers who share the same hand-controller will have attributes created on the same area and will be considered as a group.

Guides

The minimum number of guide joints required for finger joint is 3 The last joint is the tip of the finger and won't have controller. Essentially a 3 joint finger module has 2 segments.

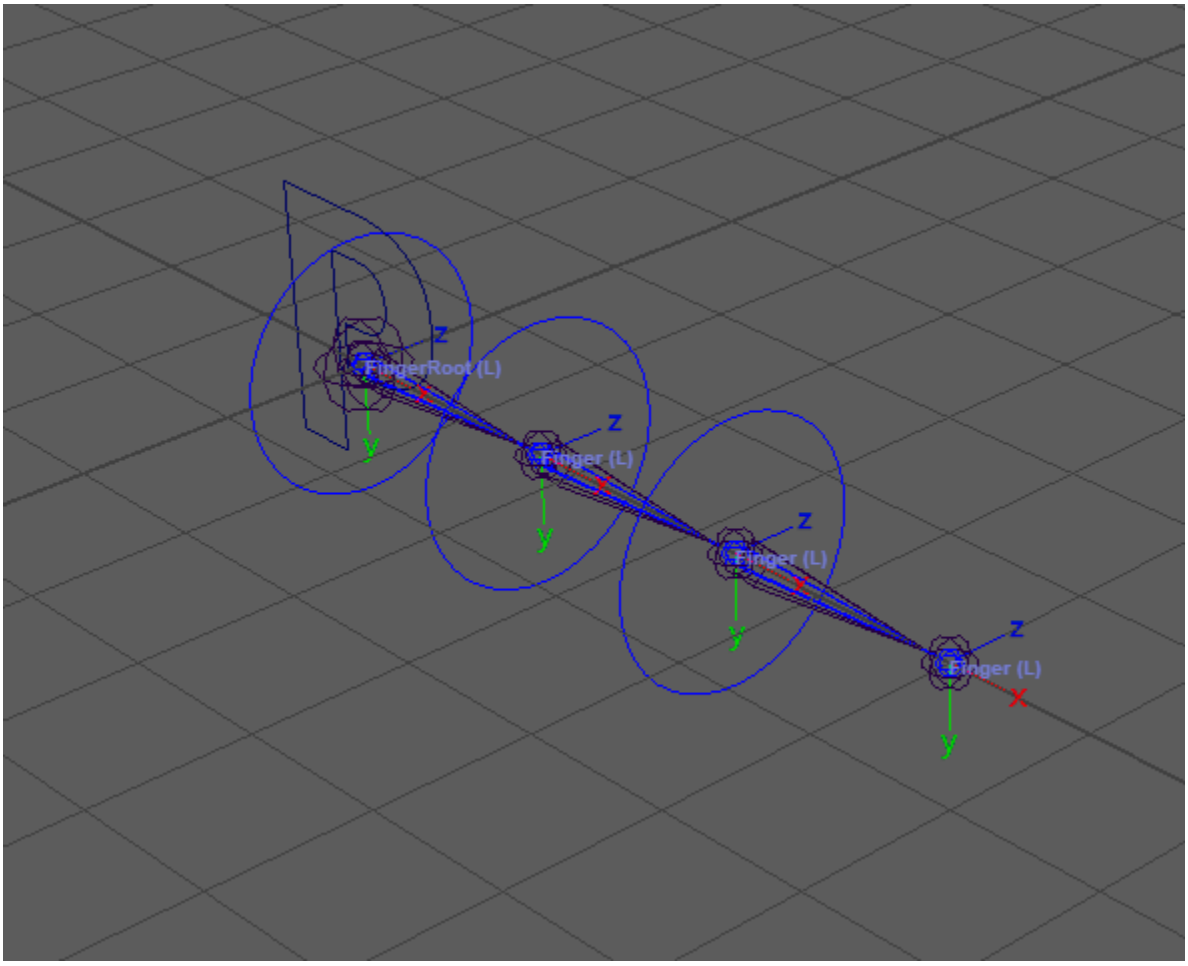
Additional Properties:

- **Finger Type**: Will define the type of finger.
- **Hand Controller**: The controller defined here will hold the bend and spread attribute for the finger.



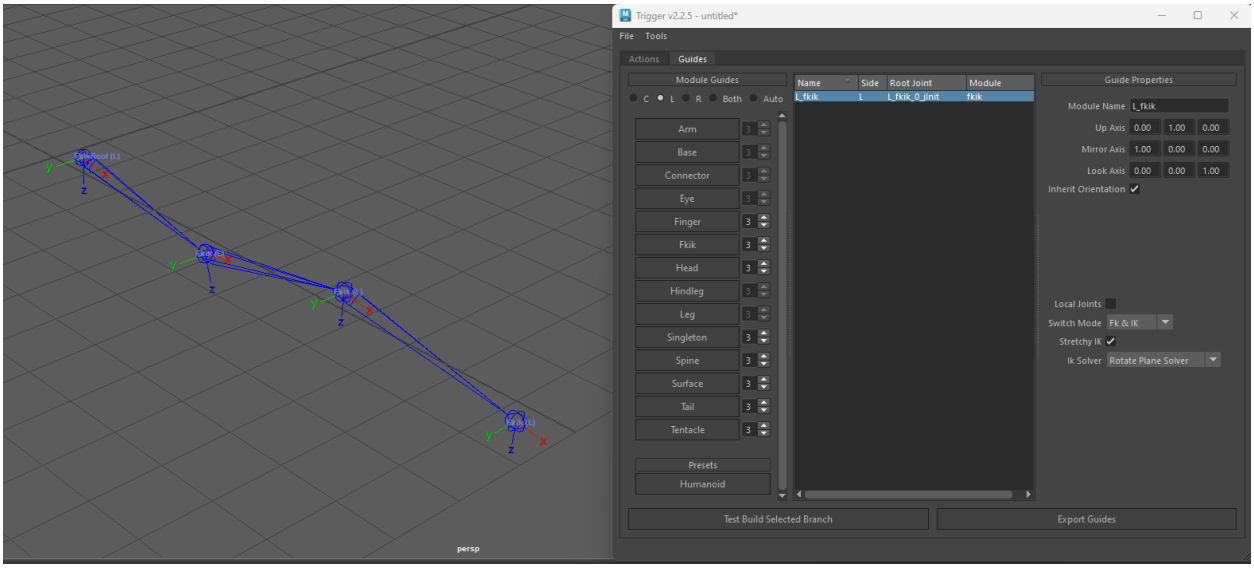
Note: The spinner in the UI, next to the guide creation buttons are for segments, not joint count

Rig

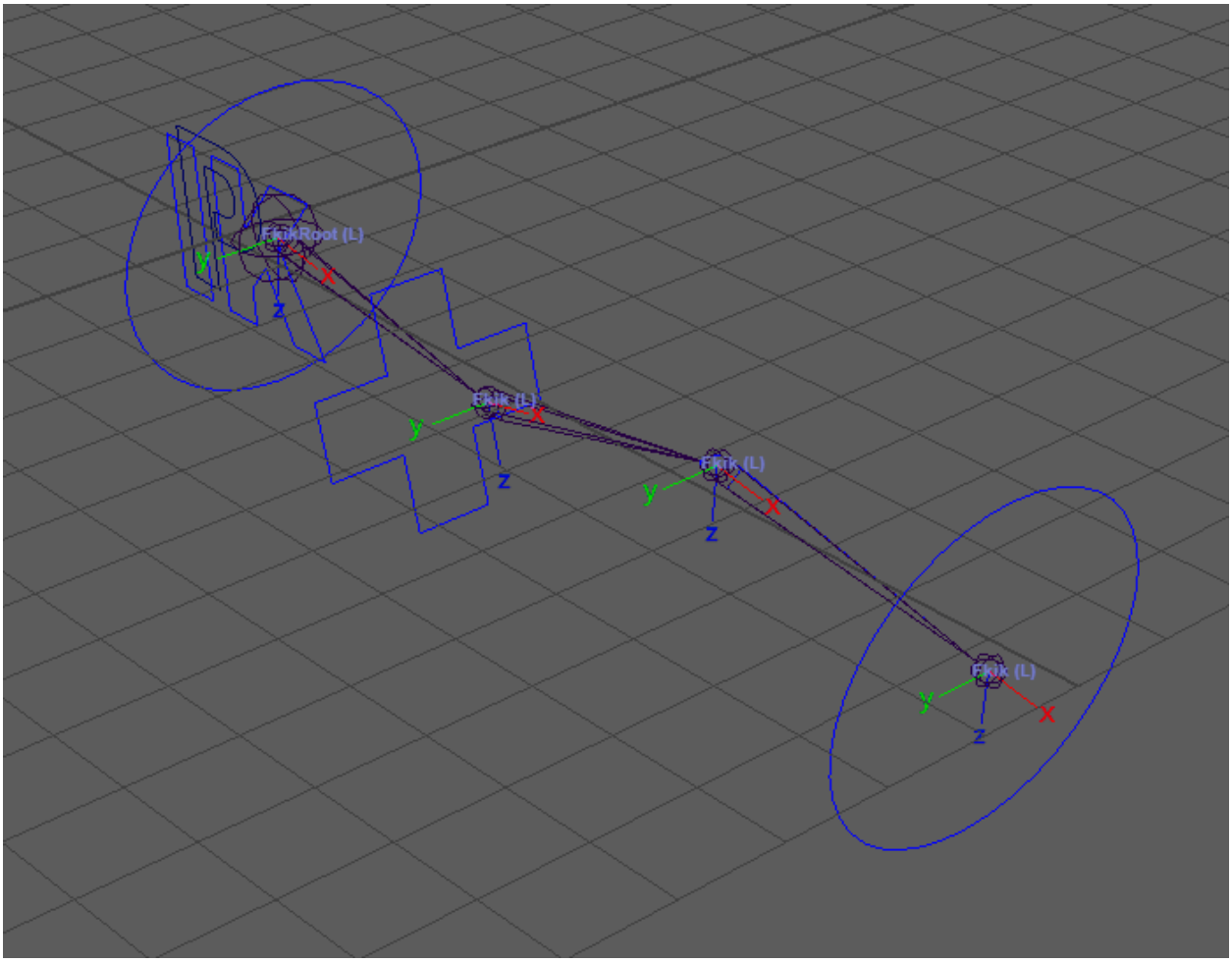


1.6.6 FKIK

Guides

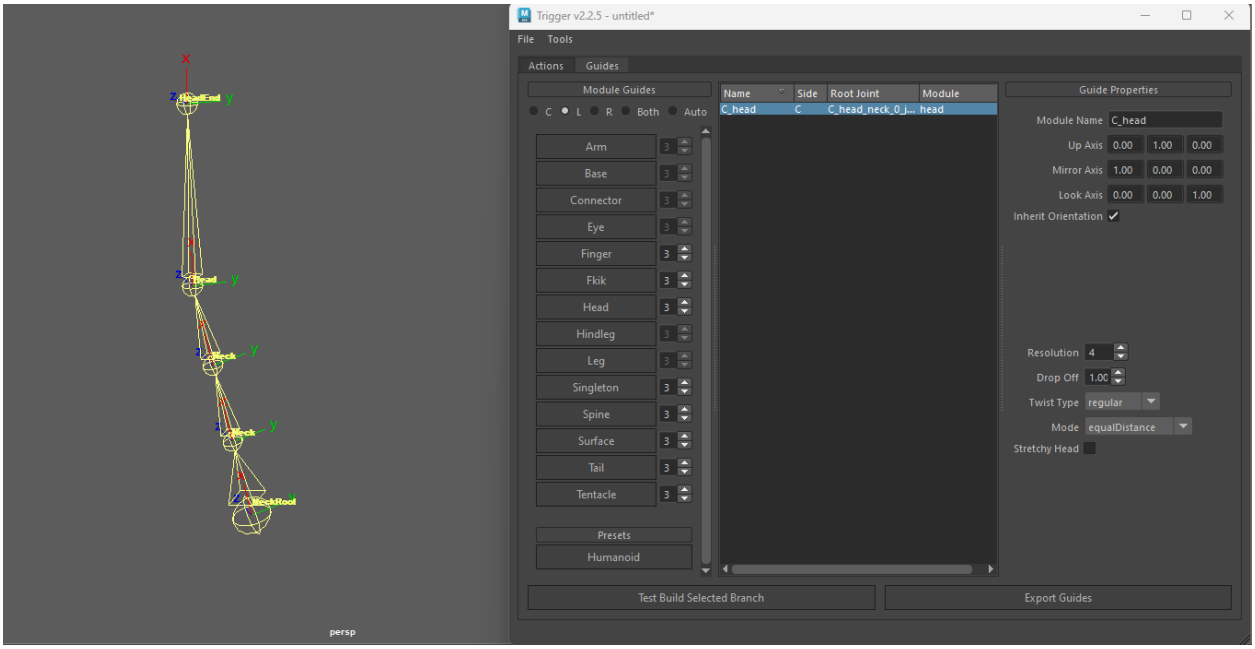

Trigger v2.2.5 - untitled*
File
Tools
Actions
Guides
Module Guides
C L R Both Auto
Arm
Base
Connector
Eye
Finger
Fkik
Head
Hindleg
Leg
Singleton
Spine
Surface
Tail
Tentacle
Presets
Humanoid
Name
Side
Root Joint
Module
L_fkik
L
L_fkik_0_jinit
fkik
Guide Properties
Module Name
L_fkik
Up Axis
Mirror Axis
Look Axis
Inherit Orientation
Local Joints
Switch Mode
Fk & IK
Stretchy IK
Ik Solver
Rotate Plane Solver
Test Build Selected Branch
Export Guides
persp

Rig

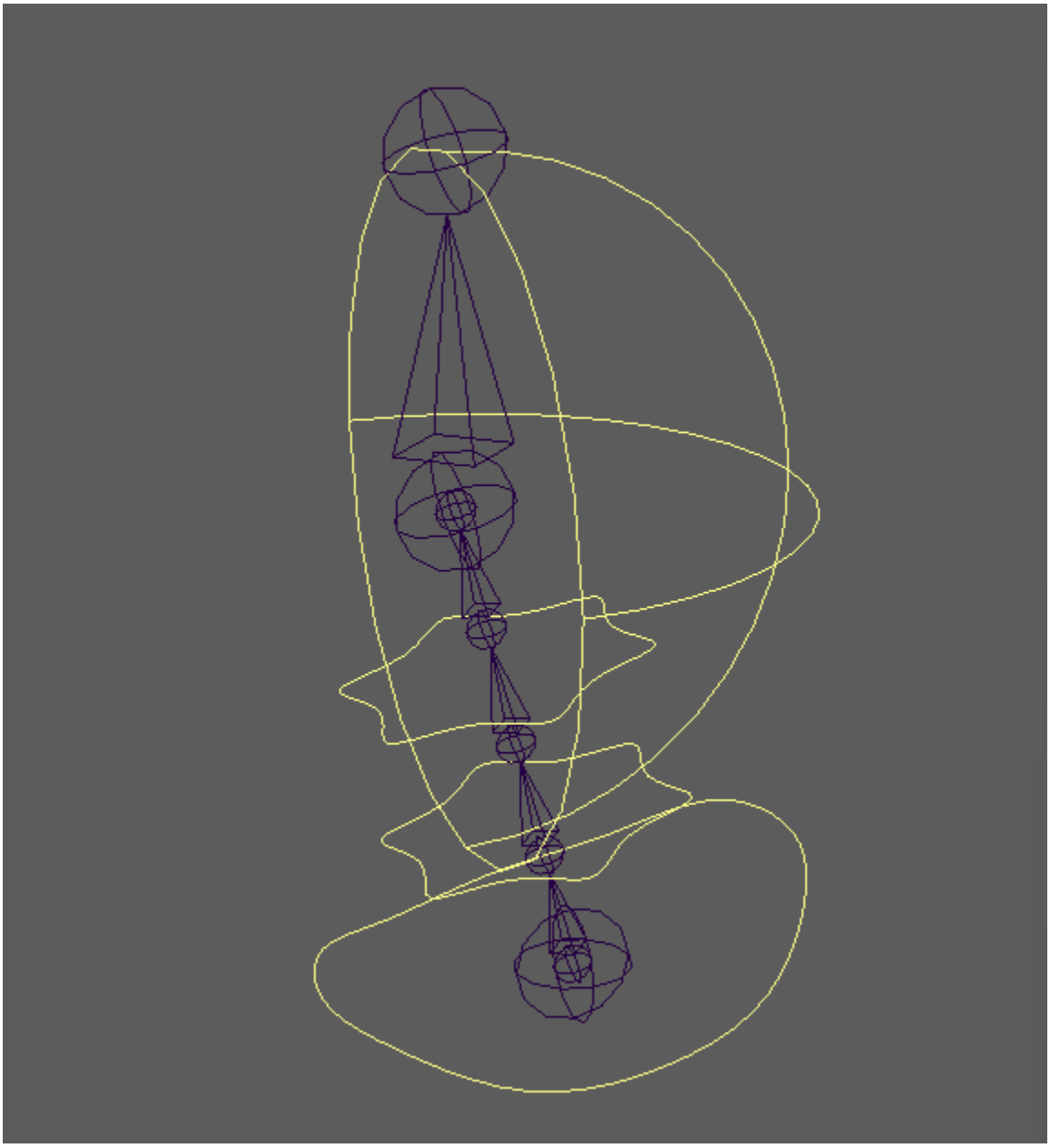


1.6.7 Head

Guides

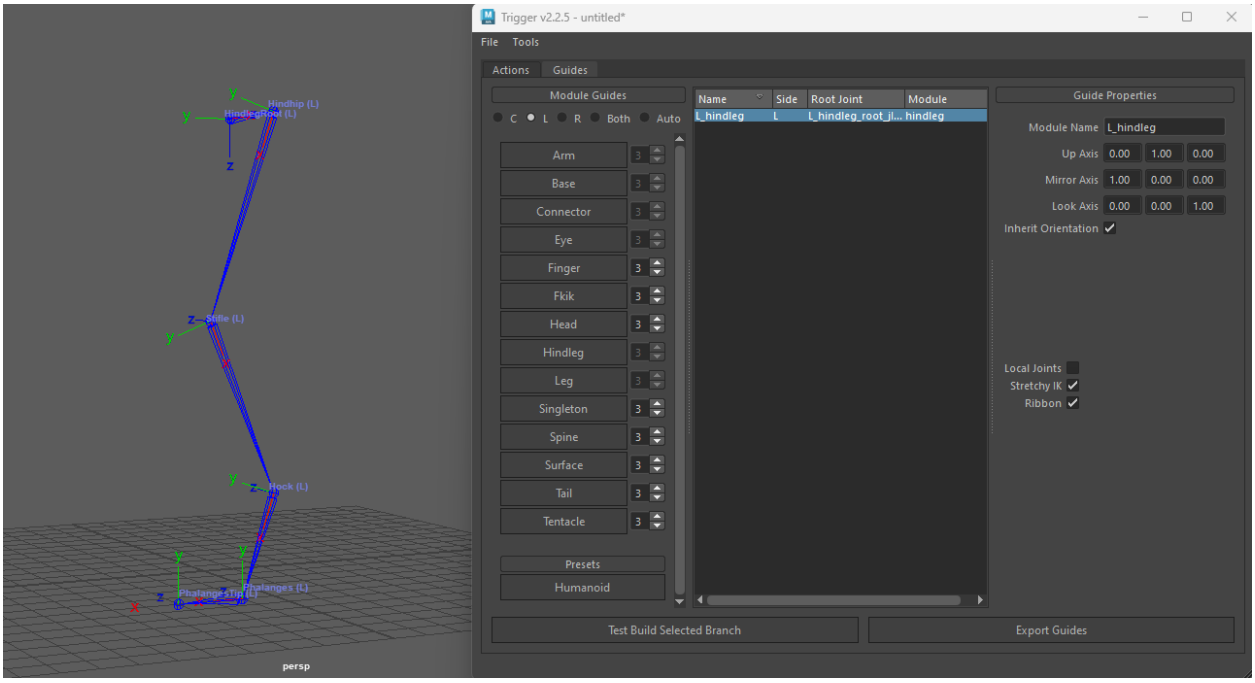

Trigger v2.2.5 - untitled*
File
Tools
Actions
Guides
Module Guides
C
L
R
Both
Auto
Arm
Base
Connector
Eye
Finger
Fkik
Head
Hindleg
Leg
Singleton
Spine
Surface
Tail
Tentacle
Presets
Humanoid
Name
Side
Root Joint
Module
C_head
C
C_head_neck_0_j...
head
Guide Properties
Module Name
C_head
Up Axis
Mirror Axis
Look Axis
Inherit Orientation
Resolution
Drop Off
Twist Type
regular
Mode
equalDistance
Stretchy Head
Test Build Selected Branch
Export Guides
persp

Rig

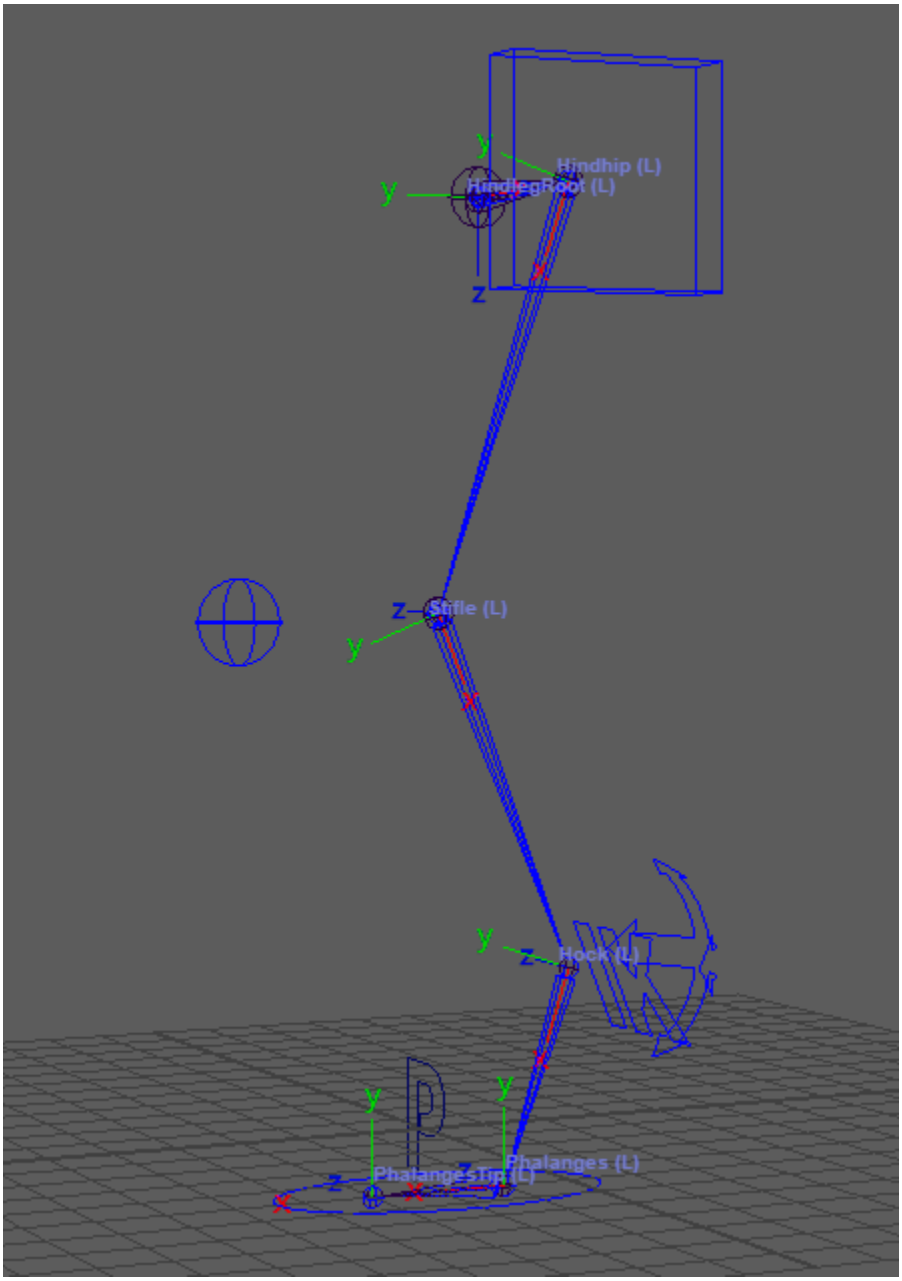


1.6.8 Hindleg

Guides

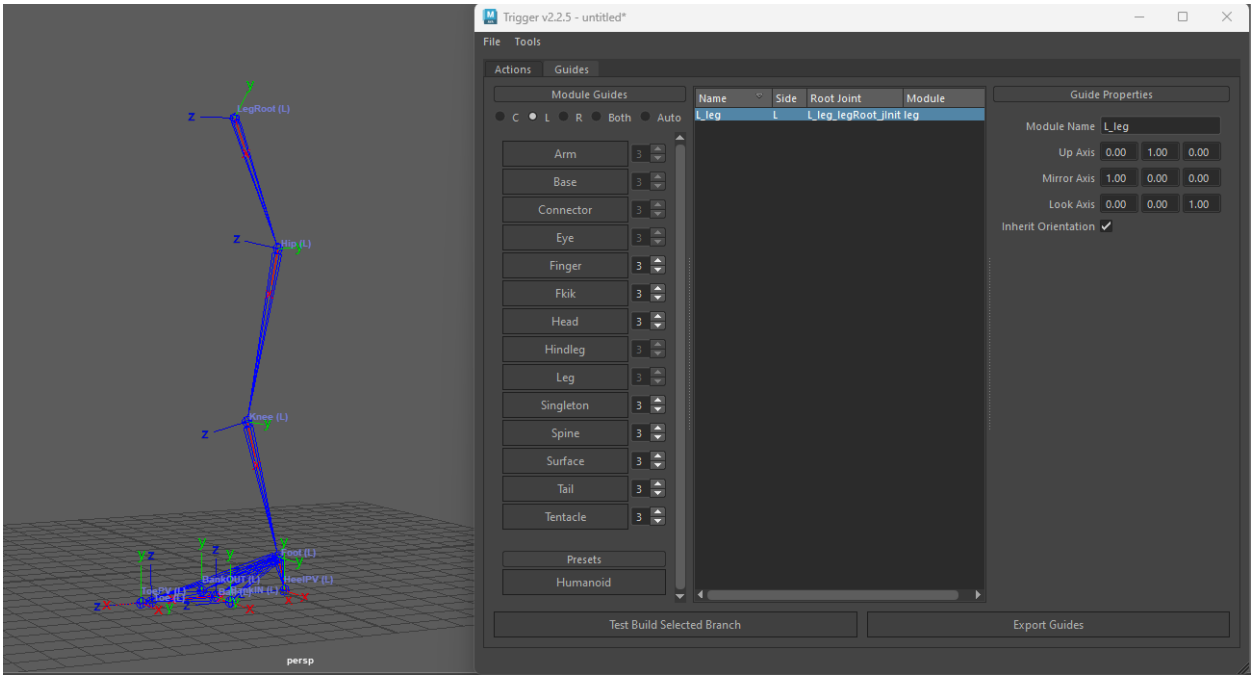


Rig

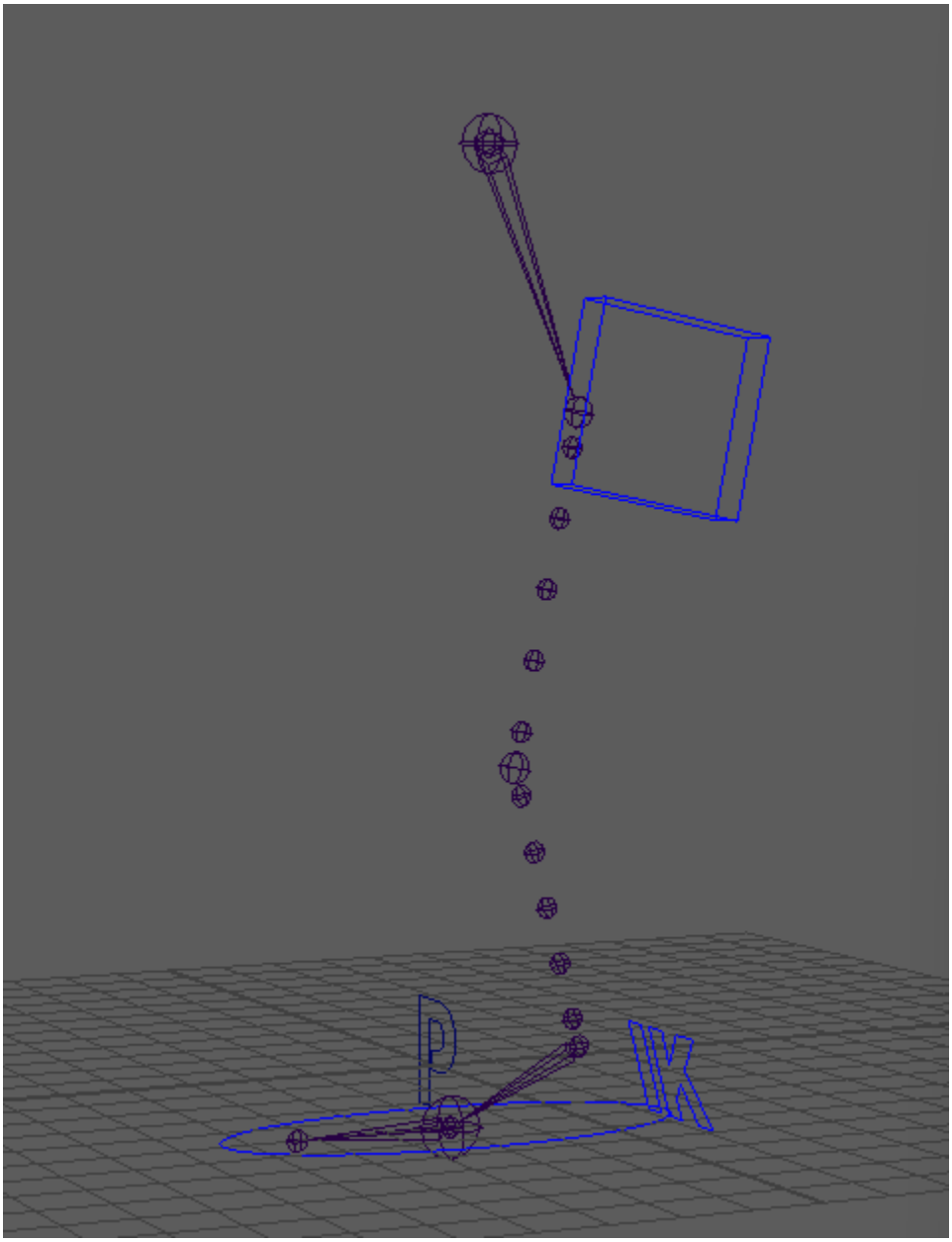

Hindhip (L)
HindlegRoot (L)
Stifle (L)
Hock (L)
Phalanges (L)
PhalangesTip (L)

1.6.9 Leg

Guides

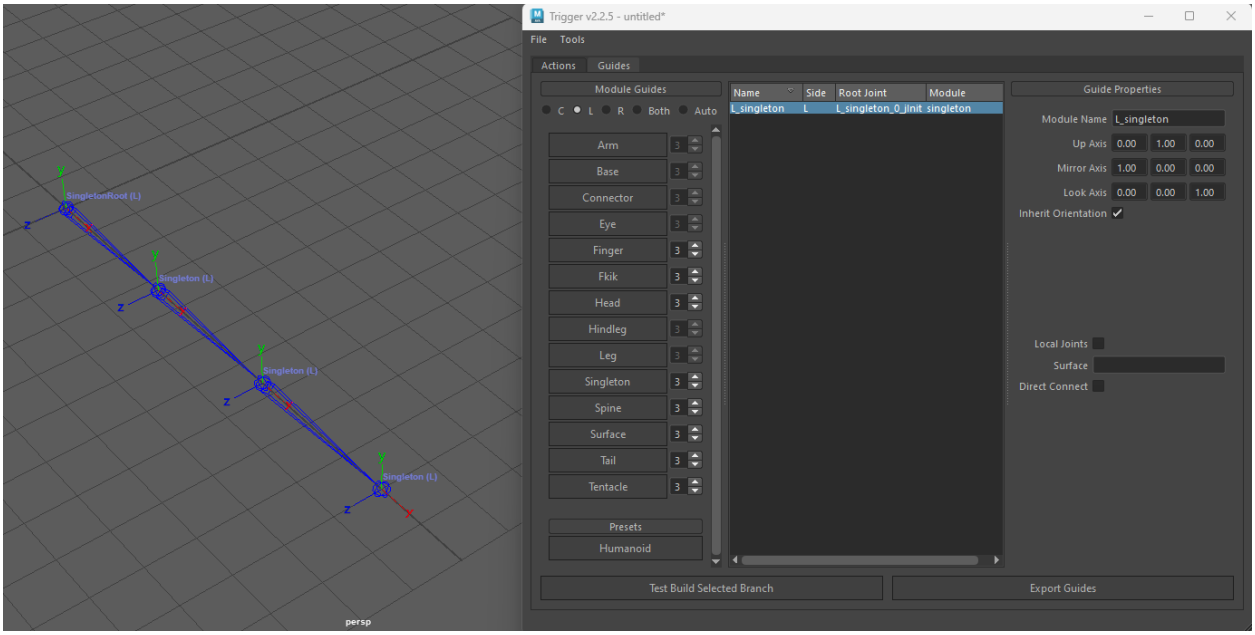


Rig



1.6.10 Singleton

Guides

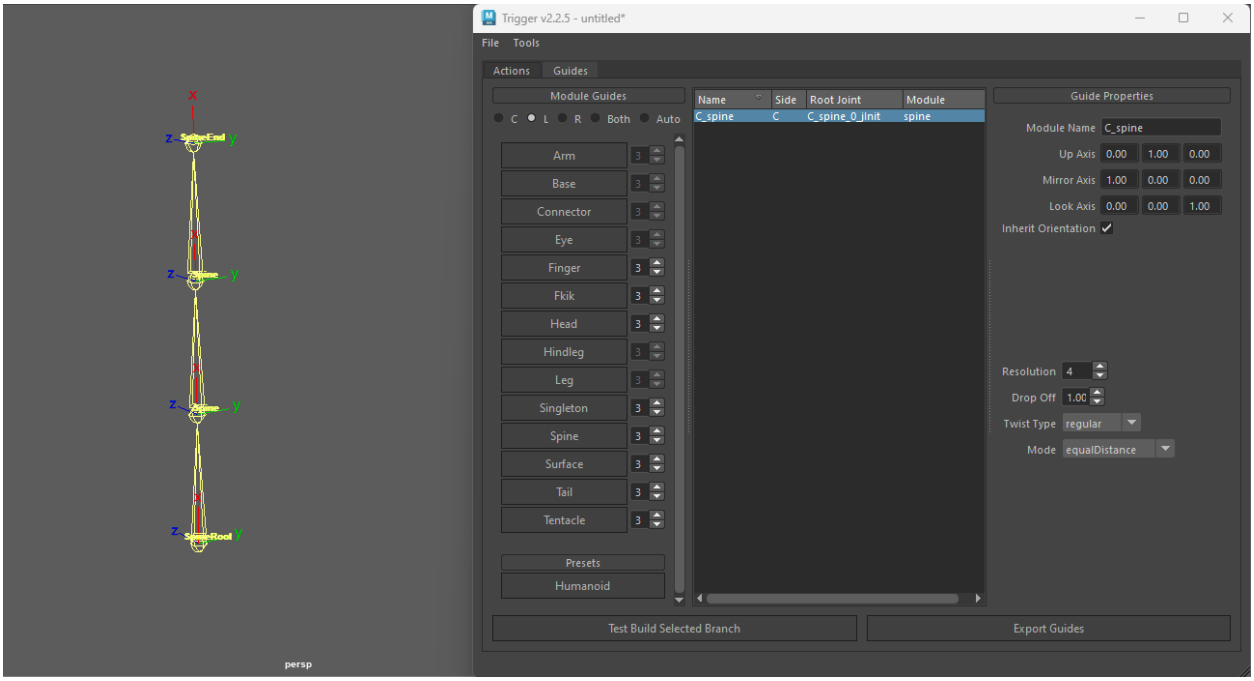


Rig

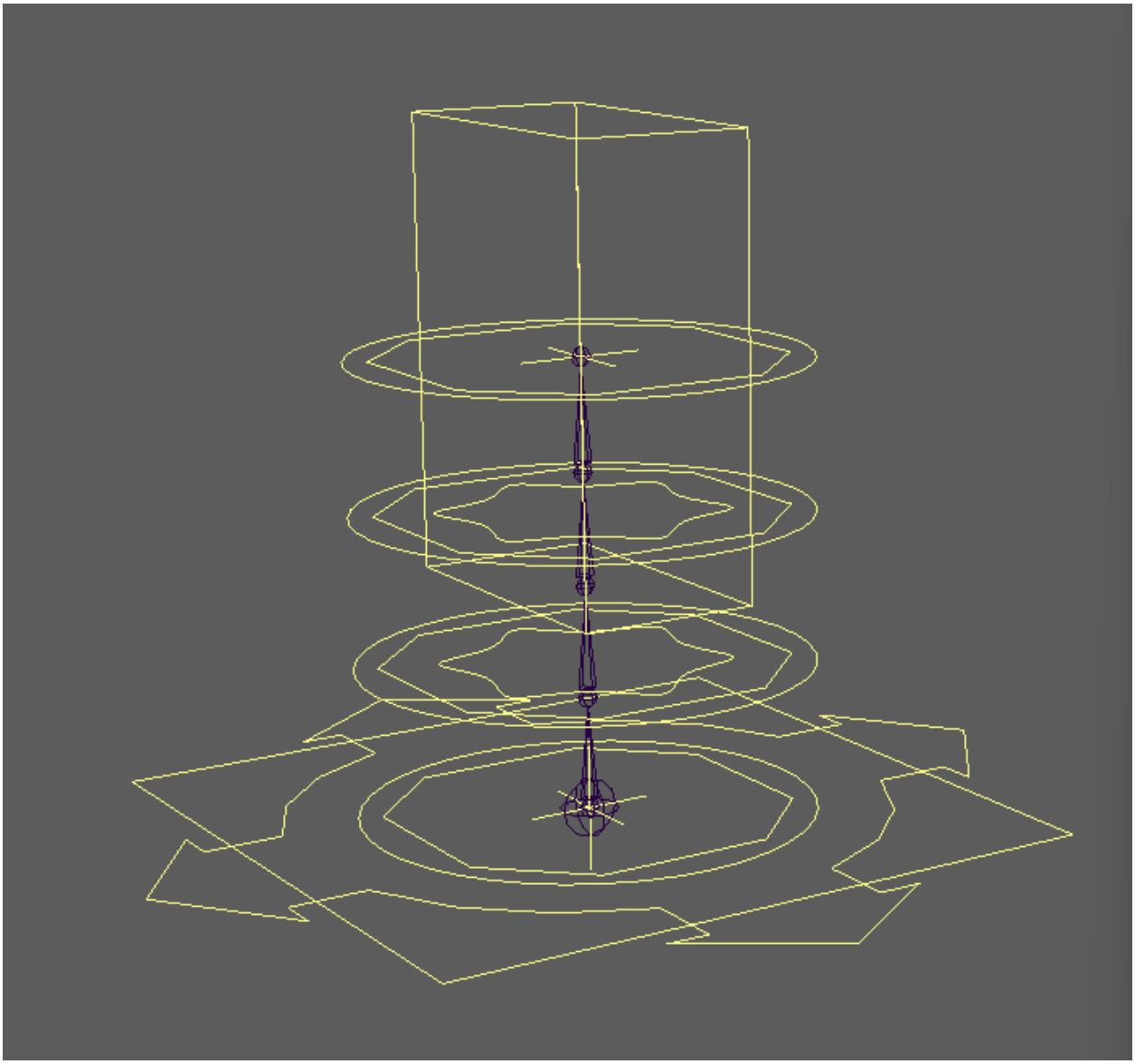
_images/singleton_rig.png

1.6.11 Spine

Guides

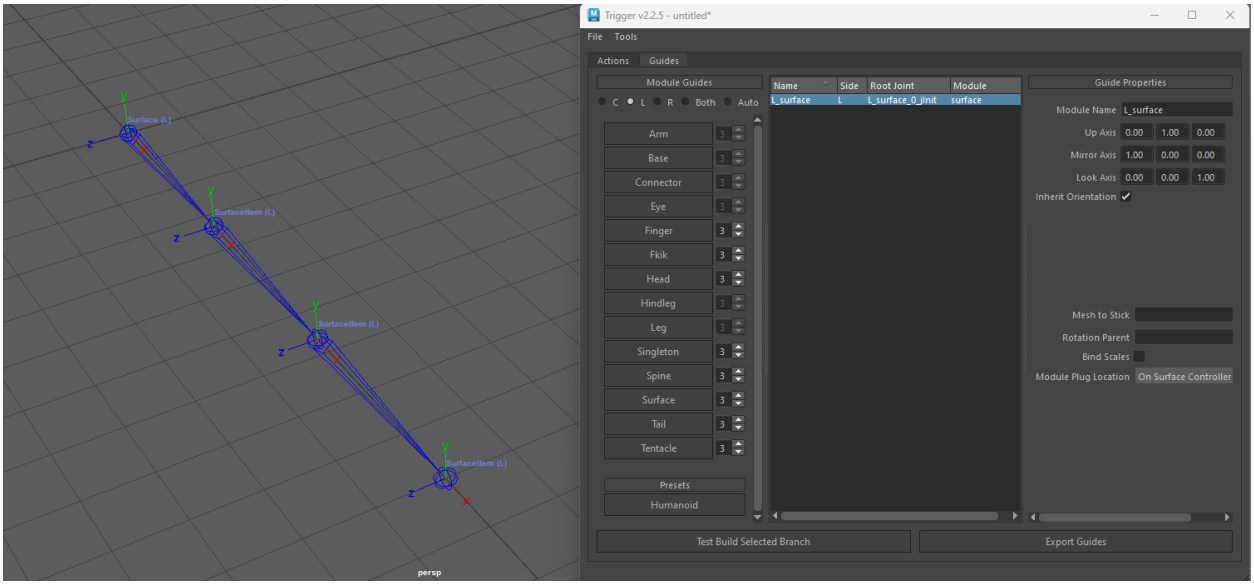

Trigger v2.2.5 - untitled*
File
Tools
Actions
Guides
Module Guides
C
L
R
Both
Auto
Arm
Base
Connector
Eye
Finger
Fkik
Head
Hindleg
Leg
Singleton
Spine
Surface
Tail
Tentacle
Presets
Humanoid
Name
Side
Root Joint
Module
C_spine
C
C_spine_0_jInit
spine
Guide Properties
Module Name
C_spine
Up Axis
Mirror Axis
Look Axis
Inherit Orientation
Resolution
Drop Off
Twist Type
regular
Mode
equalDistance
Test Build Selected Branch
Export Guides
persp

Rig

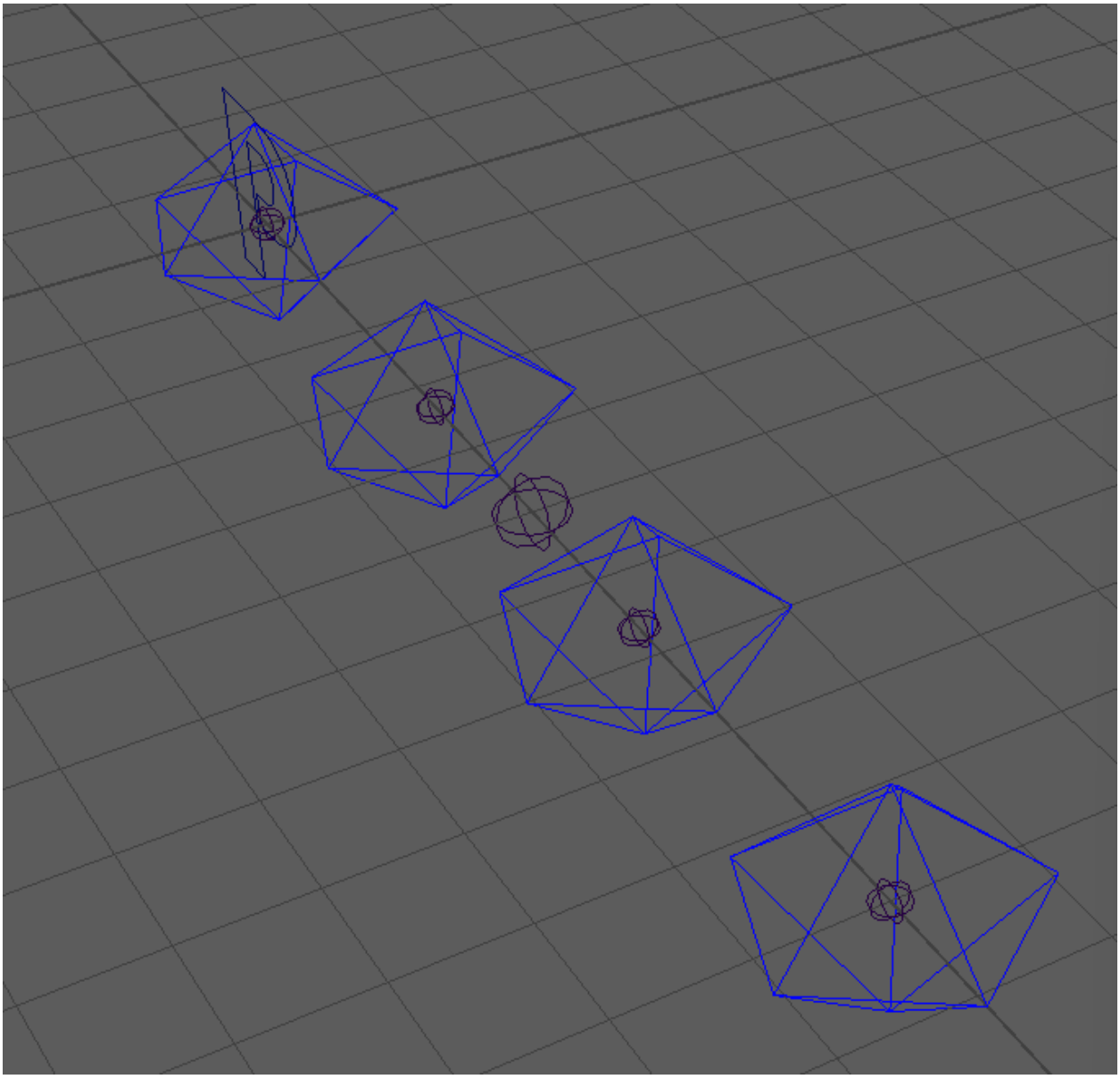


1.6.12 Surface

Guides

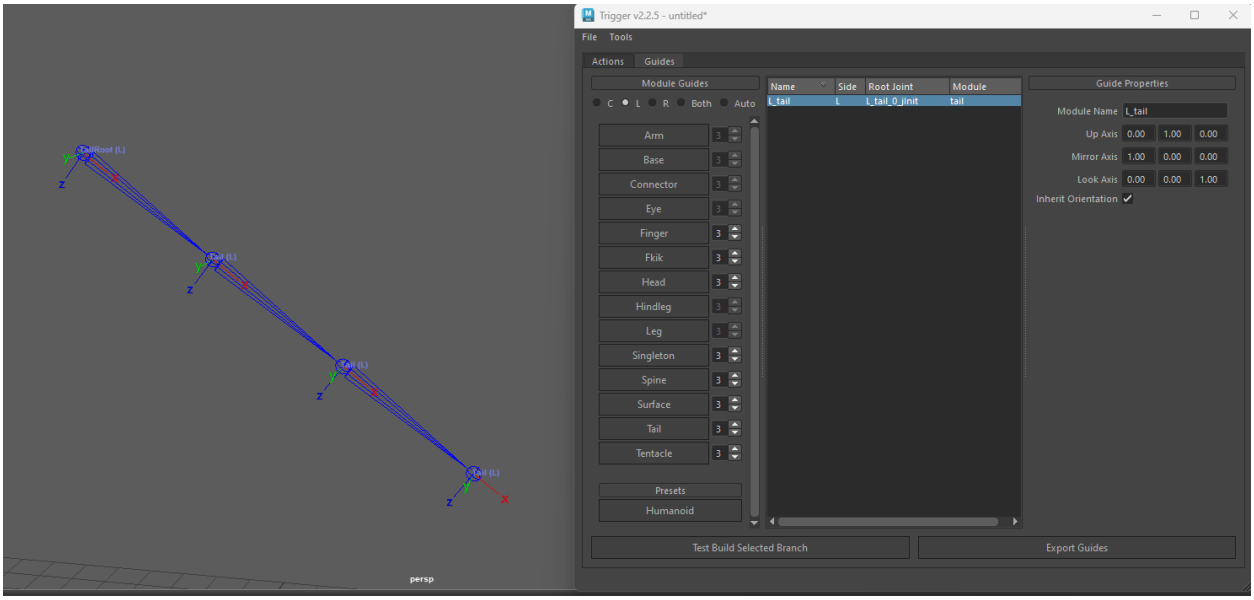

Trigger v2.2.5 - untitled*
File
Tools
Actions
Guides
Module Guides
C
L
R
Both
Auto
Arm
Base
Connector
Eye
Finger
Fkik
Head
Hindleg
Leg
Singleton
Spine
Surface
Tail
Tentacle
Presets
Humanoid
Name
Side
Root Joint
Module
L_surface
L
L_surface_0_jinit
surface
Guide Properties
Module Name
L_surface
Up Axis
Mirror Axis
Look Axis
Inherit Orientation
Mesh to Stick
Rotation Parent
Bind Scales
Module Plug Location
On Surface Controller
Test Build Selected Branch
Export Guides
persp

Rig

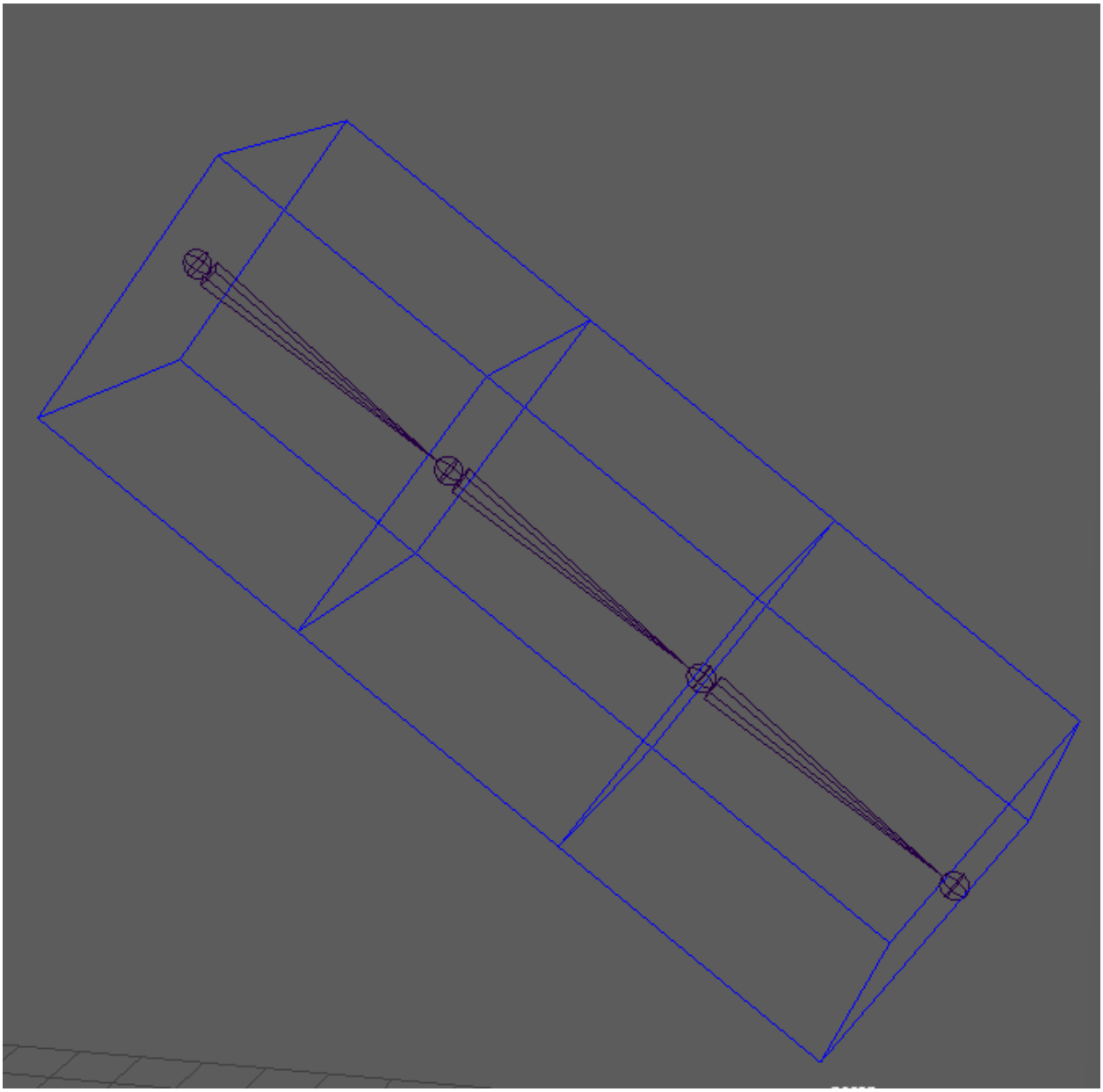


1.6.13 Tail

Guides

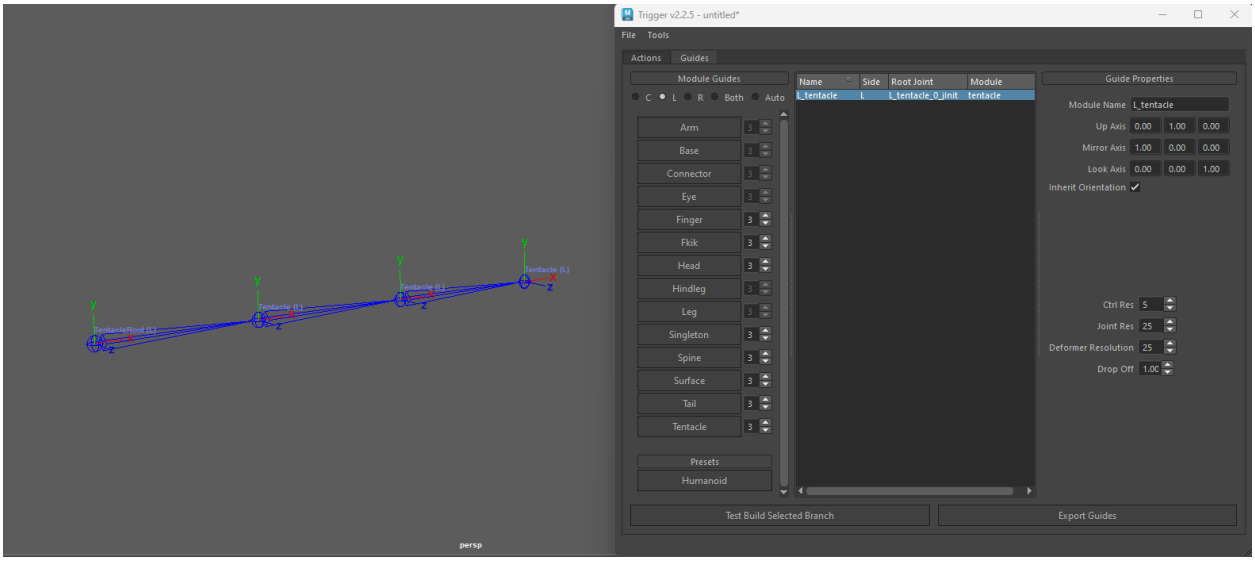


Rig

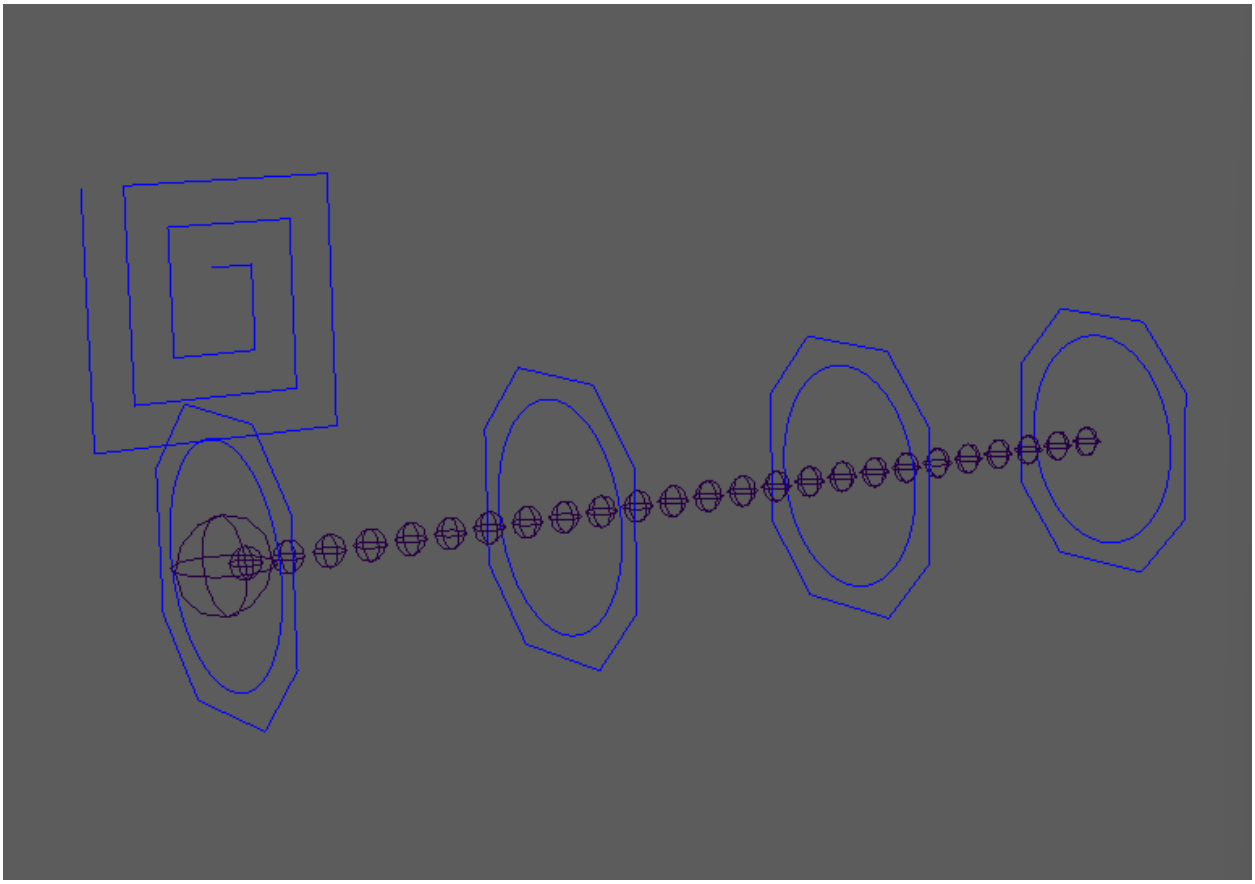


1.6.14 Tentacle

Guides



Rig



- *Arm*
- *Base*
- *Connector*
- *Eye*
- *Finger*
- *FKIK*
- *Head*
- *Hindleg*
- *Leg*
- *Singleton*
- *Spine*
- *Surface*
- *Tail*
- *Tentacle*

1.7 API

Note: Work in progress