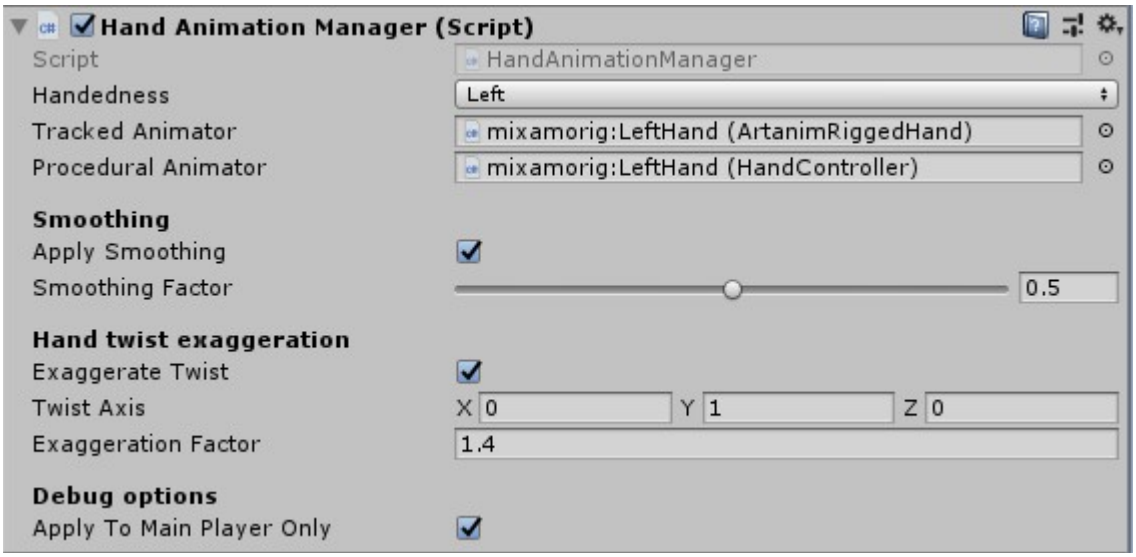
Setting up hand management and streaming

NOTE: The documentation below still mentions hand offset calibration. This has currently been disabled and can therefore be ignored. The documentation will soon be updated.

Setting up the HandAnimationManager

Hands can be animated using data from multiple sources. Below we'll discuss both Leap Motion hand animation, as well as procedurally animated hands. To manage the various sources, we use a HandAnimationManager.



Add the component to both hands. Drag a TrackedAnimator components (for a source like the Leap Motion) into the TrackedAnimator field, and do the same for the ProceduralAnimator field. Make sure to apply the correct handedness for a hand. Currently two HandAnimators are available. The ArtanimRiggedHand for Leap Motion animation, and a HandController for procedural animation.

Smoothing can be applied where a previous hand animation result is partially updated with new animation data. To enable it, check "Apply Smoothing", and set the appropriate "Smoothing Factor". If it's 0, new animation data is ignored, and if it's set to 1, new data is directly applied as is (giving the same result as one would get with "Apply Smoothing" disabled). This smoothing can be useful in cases where animation data causes minor high-frequency jitters.

When using the leap motion for hand animation, the markers a user is wearing are likely to be placed on the forearms, rather than on the hands themselves. The effect this has when rotating the palms, is that the markers rotate less than the hands. The effect this has is that an avatar's hands underrotate with respect to he user. To compensate for this, we can exaggerate the captured twist. To do so, enable "Exaggerate Twist", provide the appropriate axis, and give an exaggeration factor. A factor of 1 would leave the twist unmodified. A factor higher than 1 would exaggerate the twist, and conversely a factor below 1 would dampen the twist, but is generally not what we want.

> Note that while underrotation is an issue when using markers on the lower arms, if only Leap Motion hand animation is used, the twist exaggeration is not necessarily required. The current implementation of ArtanimRiggedHand will use the global hand orientation as provided by the Leap Motion tracking. Exaggeration is primarily useful in the case where there are addition hand animators such as the procedural animation.

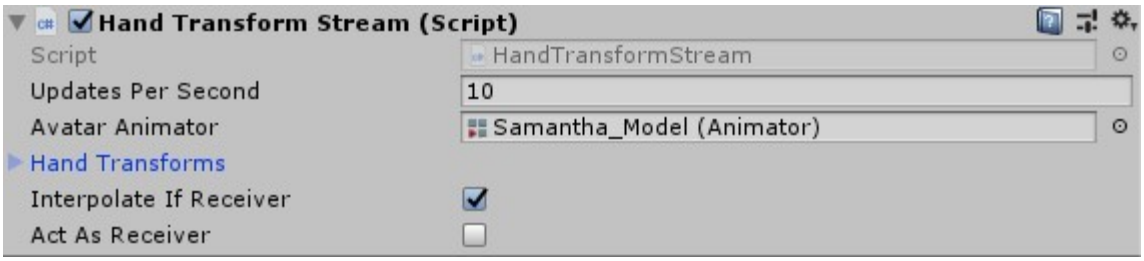
By default the HandAnimationManager makes sure to only apply hand animation to the main avatar on a client. That is, the avatar representing the users themselves. This can be problematic when trying to debug in the editor in cases where an avatar is not the main user. To override this behaviour, disable "Apply To Main Player Only".

Streaming hand animation data to other players

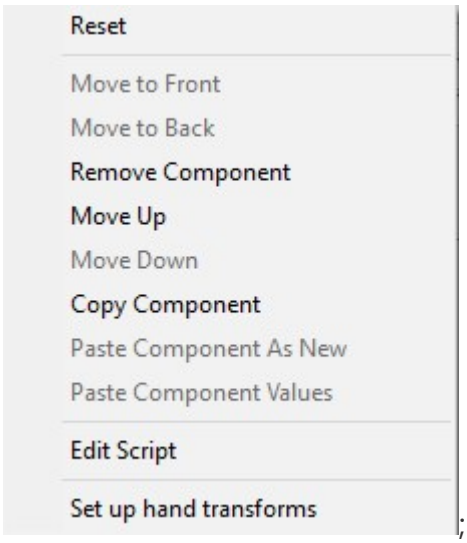
As stated above, hand animations are calculated locally. That is, each client computes its own hand animation for immediate display. That does however mean that other players would not see the results on avatars other than their own. To resolve this, we can stream the hand animation results to other clients.

To do so, add a HandTransformStream component to your avatar. This component, as the name indicates, streams hand transform data to other clients. But in addition it acts as the receiver of hand transform data for data from non-main-player avatars.

Add the avatar's Animator to the "Avatar Animator" field.

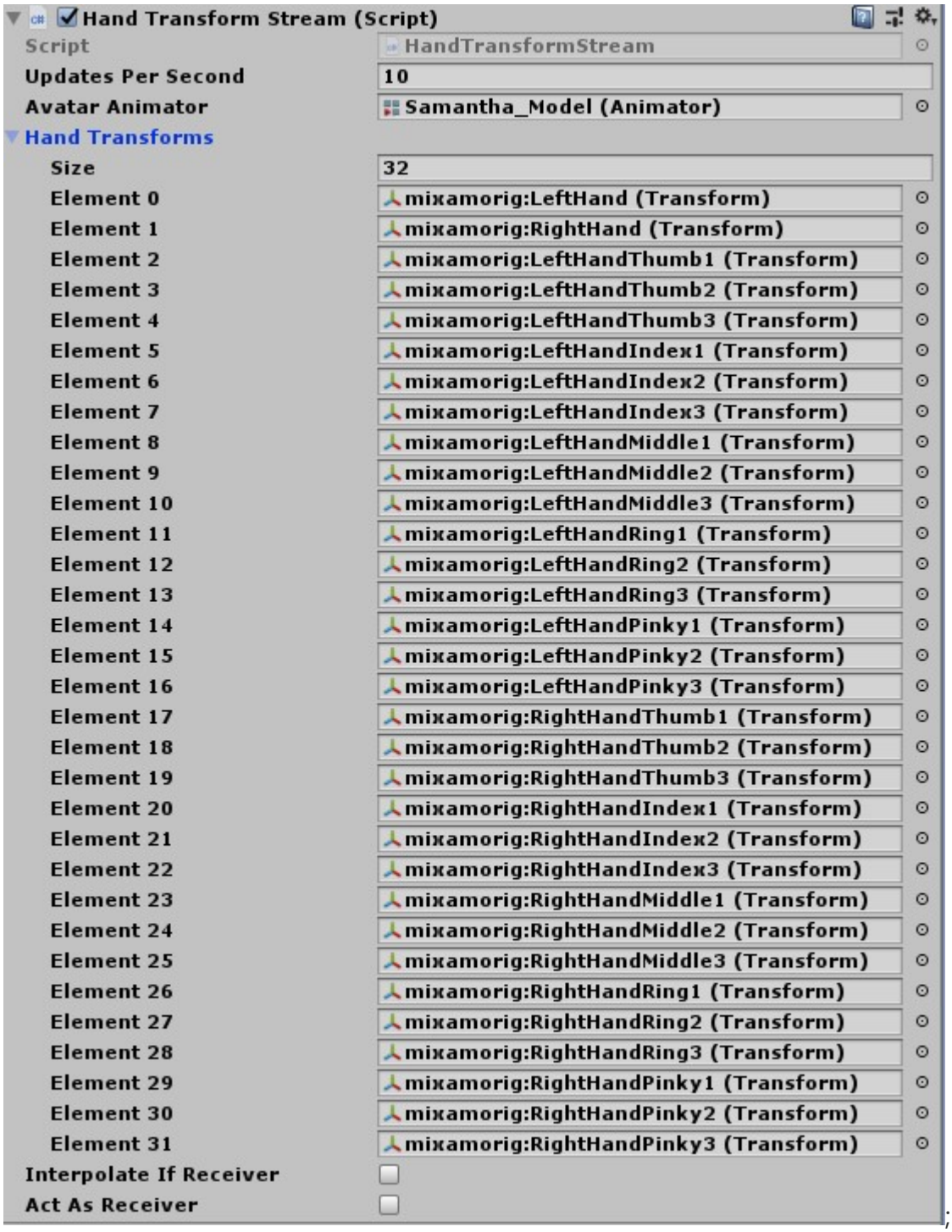


From here we can set up the transform data (rotations) we want to stream in the Hand Transforms array. We can do so manually if needed, but the easier option is to right-click the component and select "Set up hand transforms".



;

If succesful, this automatically add all the relevant transforms to the Hand Transforms list.



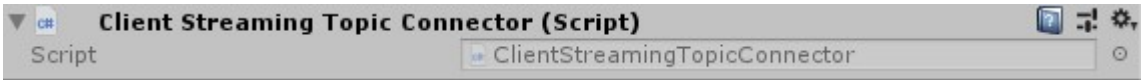
The "Updates Per Second" field allows us to specify how many "frames" per second of hand animation data we want to send out. It is important to keep in mind that there's quite a lot of data to send, and if we do so very frequently, that may cause network issues. So it's best to keep this value as low as we can get away with.

If we send data at 30 "frames" per second or higher, the animation when applied to receiving avatars will look acceptable as is. However, at lower update counts, animation would look jittery. To mitigate this, enable the "Interpolate If Receiver" option. When enabled, several animation frames are buffered, and animation is

interpolated between frames. This ensures animation looks smooth, but comes at the cost of a small delay.

The "Act As Receiver" option is best left unchecked. It only serves a purpose if we want to test in-editor whether streaming works, by allowing an avatar to explicitly act as a receiver of data that got sent.

With that, the streaming setup is complete. The only thing left to do is to add a ClientStreamingTopicConnector component to a GameObject in our scene.



This connector allows us to send and receive hand animation data. Note that this component should only be used once. It is best to add it to an object which is present in the Construct Scene, and to ensure it doesn't get destroyed on scene load. To achieve this, one could add a DontDestroyOnLoadBehaviour to the same GameObject.

Setting up Leap Motion controlled hands for avatars

Prerequisites

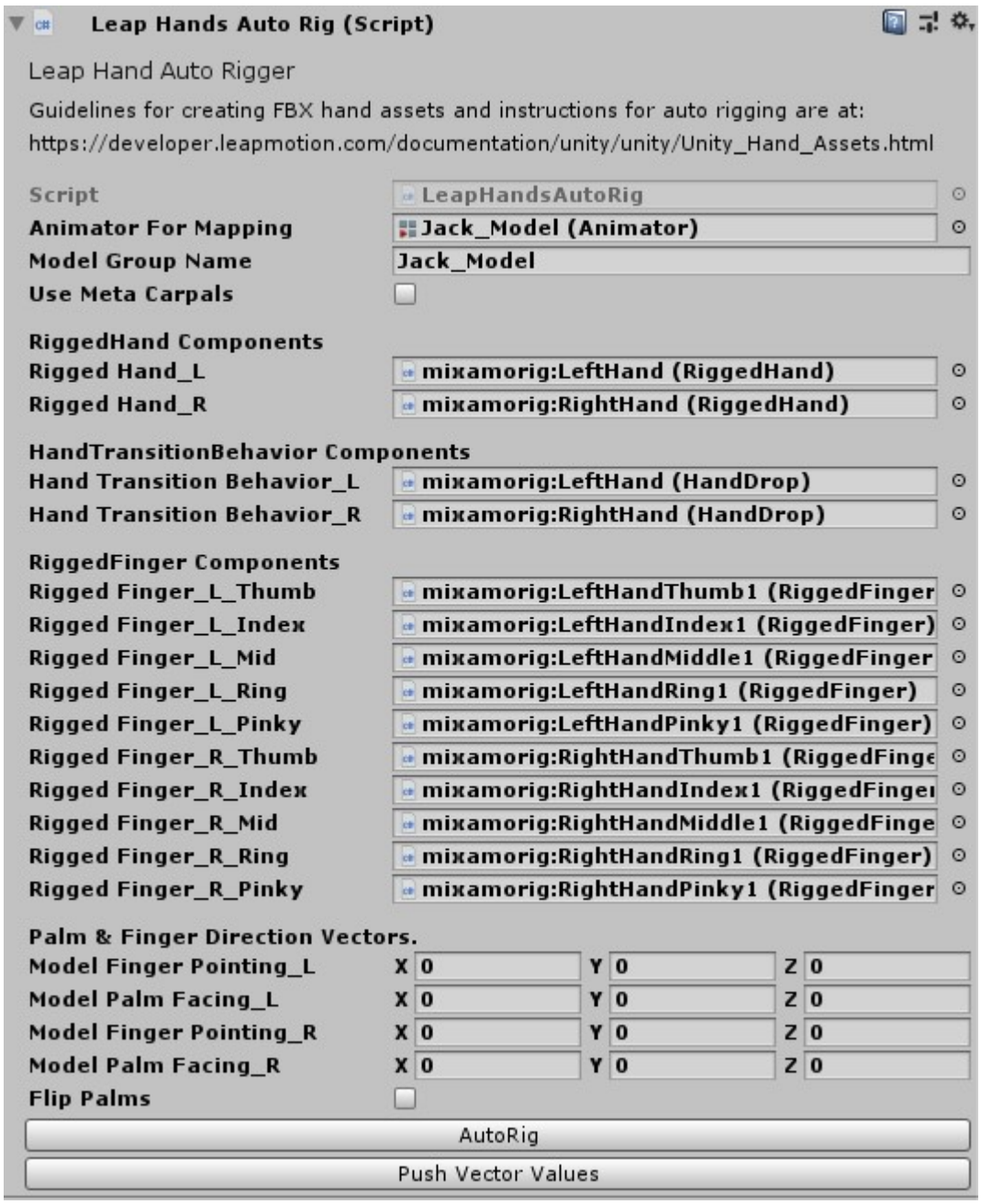
> Note: depending on how you obtained the HandAnimation library, you may already have the prerequisites below

- Download and install the Leap Motion development kit (v4.0+), which can be found at https://developer.leapmotion.com/windows-vr
- For your Unity project download the following packages and add them to your project
  - Unity Core Assets (v4.4.0+)
  - Hands Module (v2.1.4+)

Setting up the hands

To set up hands to be animated by the Leap Motion, it is easiest to make use of the Auto Rig Hands component supplied in the Hands Module. (see https://leapmotion.github.io/UnityModules/hands-module.html)

Add the component to an avatar at he level where the Animator component can be found. Add the Animator to the "Animator For Mapping" field, and click on the "AutoRig" button. The various RiggedHand and RiggedFinger components should appear in the fields as illustrated in the image below.



You can discard the rigging component once setup has been completed, as well as added HandDrop components.

> An alternative option is to add RiggedHand components to both hands, and right-click them to bring up the context menu. Select "Setup Rigged Hand" to setup the specific hand.

In our specific application scenario, we want to apply the hand tracking results to the hands of an avatar whose wrist positions come from Vicon tracking. To do this, we can't directly use the RiggedHand, but we need and additional component; an ArtanimRiggedHand to apply Leap Motion tracking results to the RiggedHand at the correct wrist location.

Setting up the ArtanimRiggedHand

First, add the ArtanimRiggedHand component to both hands.

Drag the RiggedHand component we've set up for a hand into the "Rigged Hand" field, and you're done.

There are however a number of parameters to play with, which are stored in an external json file name `leap_config.json` within the `StreamingAssets` folder. If this file is not yet in your project, you can generate it by going to `"Artanim > Create Leap Config File"` in the editor menu. The content of this file will look something like this:

```
{
    "ConfidenceThreshold": 0.5,
    "CalibrationConfidenceThreshold": 0.8, //NO LONGER USED
    "MaxHandOffset": 0.15,
    "RecalibrationThreshold": 0.03 //NO LONGER USED
}
```

> Note that if you were to forget to create this file or to include it with your experience, the file will still be created from the default values the first time the application runs.

The Leap Motion, in addition to handtracking data, gives us a confidence value between 0 and 1, informing us how confident it is about the tracking data it supplied. Tracking results with a low confidence value can give poor visual results. The "ConfidenceThreshold" is used to ignore all results with a confidence value below this threshold.

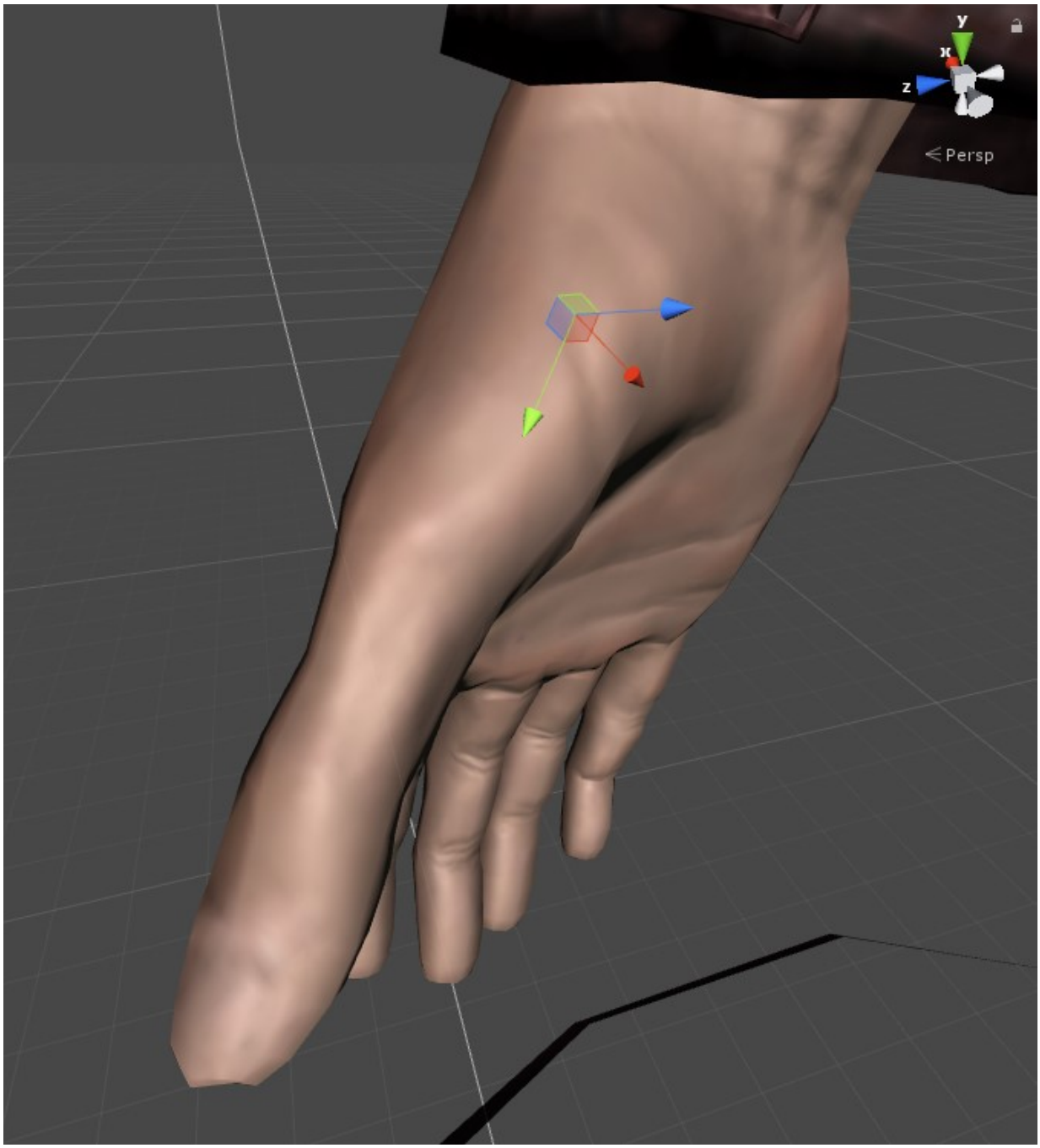
The Leap Motion gets confused by larger markers being located on the top of the hand or on the wrist. To avoid this, markers are best placed on top of the lower arm, as

close to the wrist as possible while still obtaining good tracking results. ~~This means markers are offset with respect to the wrist. To account for his, the ArtanimRiggedHand automtically calibrates the offset at runtime, either the first time it is seen, or when it notices the marker has significantly shifted during use. To avoid using bad data in this calibration, a "CalibrationConfidenceThreshold" is set. It is best to set this value to something relatively high, and generally at or above the regular "Confidence Threshold".~~

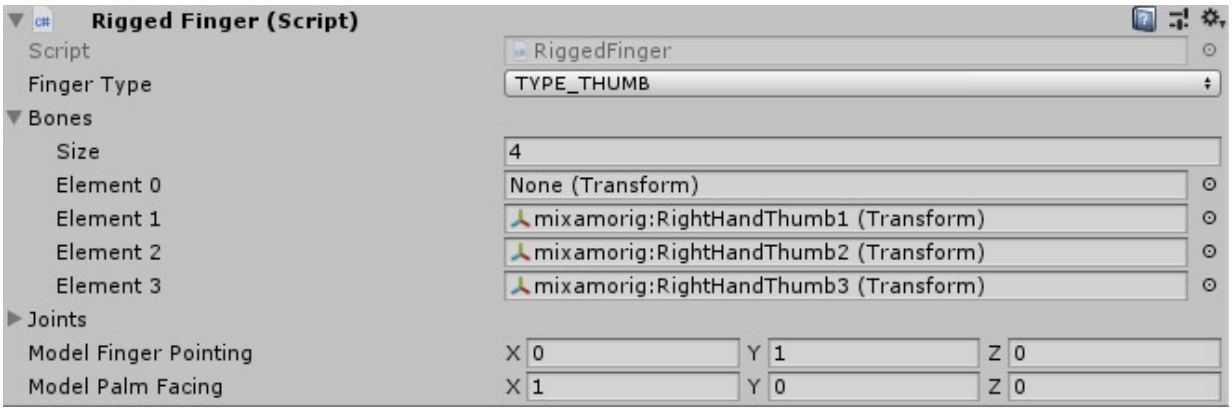
~~To avoid really bad calibrations from happening, the "MaxHandOffset" specifies the distance limit between the marker, and where it thinks the wrist is actually located. In the example we limit this distance to 15cm. Further, at runtime, the application will occasionally check whether or not the offset is still correct. If it isn't, a recalibration will take place. To prevent calibrations from happening too often, a "RecalibrationThreshold" specifies by how much the offset should have changed before we attempt to recalibrate. In the example above this is set to 3cm.~~

A special remark on thumbs

While the auto rigging tool does properly set up the various axes for most fingers, it does seem to fail on thumbs on some of the mixamo-rig characters such as Jack and Samantha. Particularly so for the palm-facing direction. If this stays uncorrected, the thumb will seems to bend in a strange manner.



As can be seen in the figure below, it is the red (positive) x-axis which points towards the palm. The values entered in the Thumb's RiggedFinger component should match that.

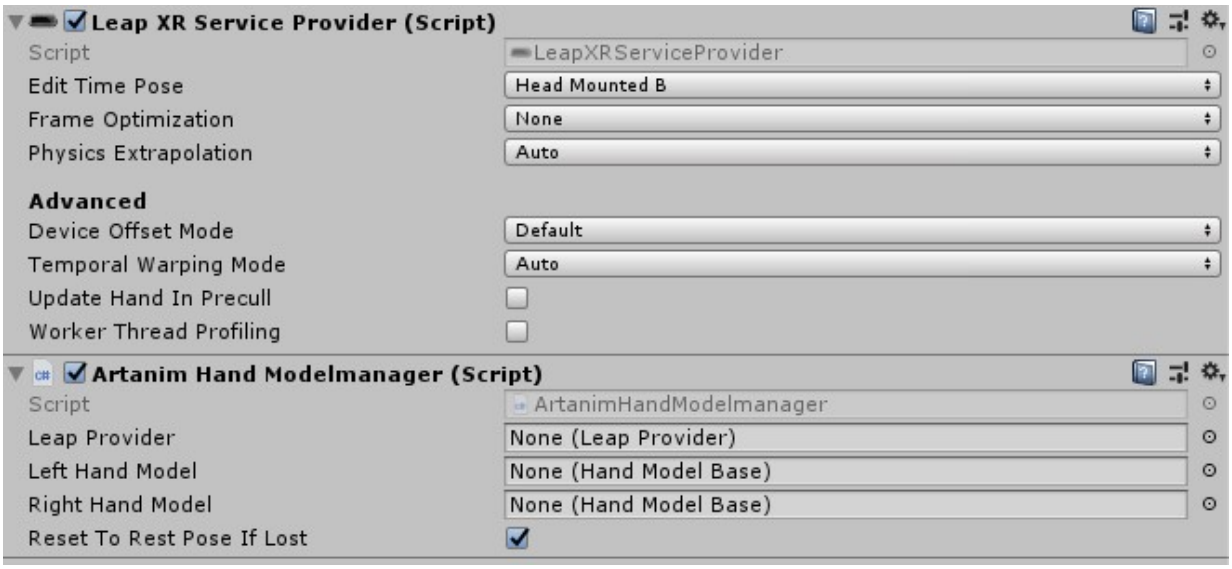


Keep in mind that these factors will most likely be different for the left and right hand.

Using the hand tracking with Leap Motion

The Leap motion and its Unity software assume that there is a fixed set of hands to be instantiated whenever a hand is detected. In our case however, we want the hands we have set up on an avatar to be used if the avatar is that of the main player.

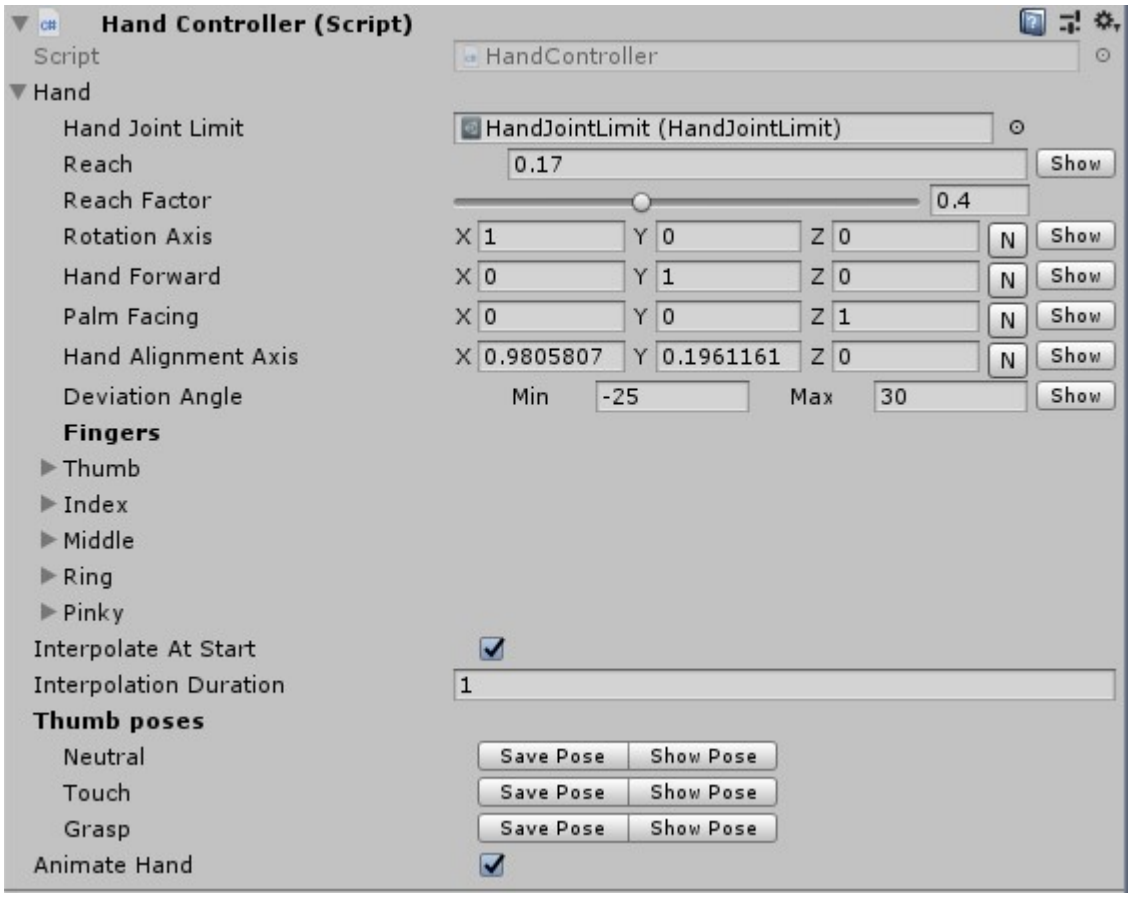
In order to achieve this we have a custom ArtanimHandModelManager. To use it, make sure the camera prefab which is used on the client for the player has both a LeapXRServiceProvider as well as the ArtanimHandModelManger set up on the camera. (You may need to create a custom prefab and adjust your experience settings to use this one rather than the default camera).



Once this is set up, as soon as an avatar is instantiated and is the main player, the hands will register themselves with the ArtanimHandModelManger. When done, whenever the leap detects hands, they should now control wrist and finger movement for your avatar.

Setting up procedurally animated hands for avatars

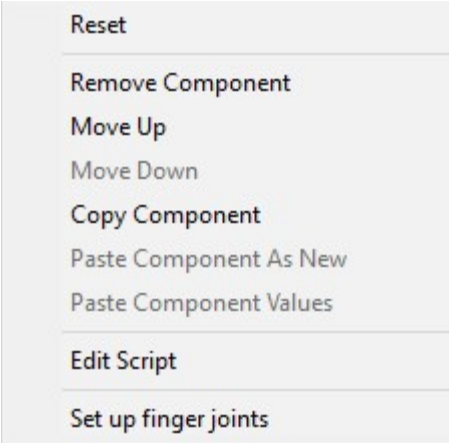
In cases where hand tracking is likely to perform poorly, or is simply not available, we can use procedural hand animation to have avatar hands interact with the environment. To use procedural hand animation, a hand needs to have a HandController.



Setting up finger joints

To set up a HandController we first need to set up all joints. The HandController will be on the GameObject which contains the wrist transform, but it still needs to know about the relevant Transforms for the fingers. If your finger transform contain the finger names ("Thumb", "Index", "Middle", "Ring" and "Pinky") as part of their object names, the assignment can be finished automatically.

Right-click on the HandController component, and select "Set up finger joints" from the context menu.

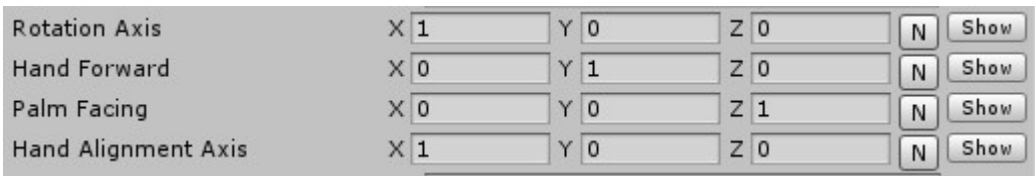


If all worked correctly, you should see the relevant transforms under "Joint Transforms" for each finger, such as the thumb illustrated below.

If transforms haven't been assigned, you can still manually add them to the list. Just make sure you do so in hierarchical order (i.e. from the root down).

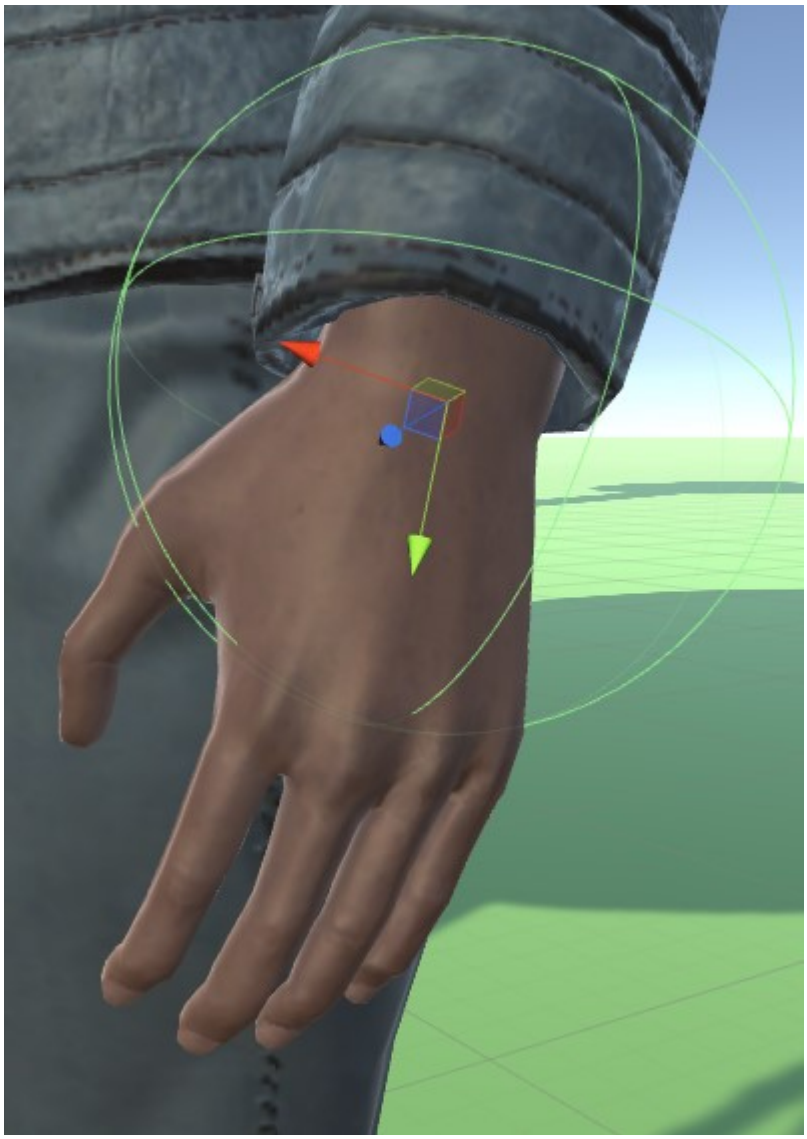
Setting up hand axes and reach

The hand and fingers have various axes which describe how a hand is oriented and along which axes its joints can rotate. As a visual aid, the various axes can be shown in the scene view by selecting the "Show" toggle to the right of each vector. If axes which have been set up in non-axis-aligned directions no longer are of unit length, they can be corrected by clicking on the "N" button.

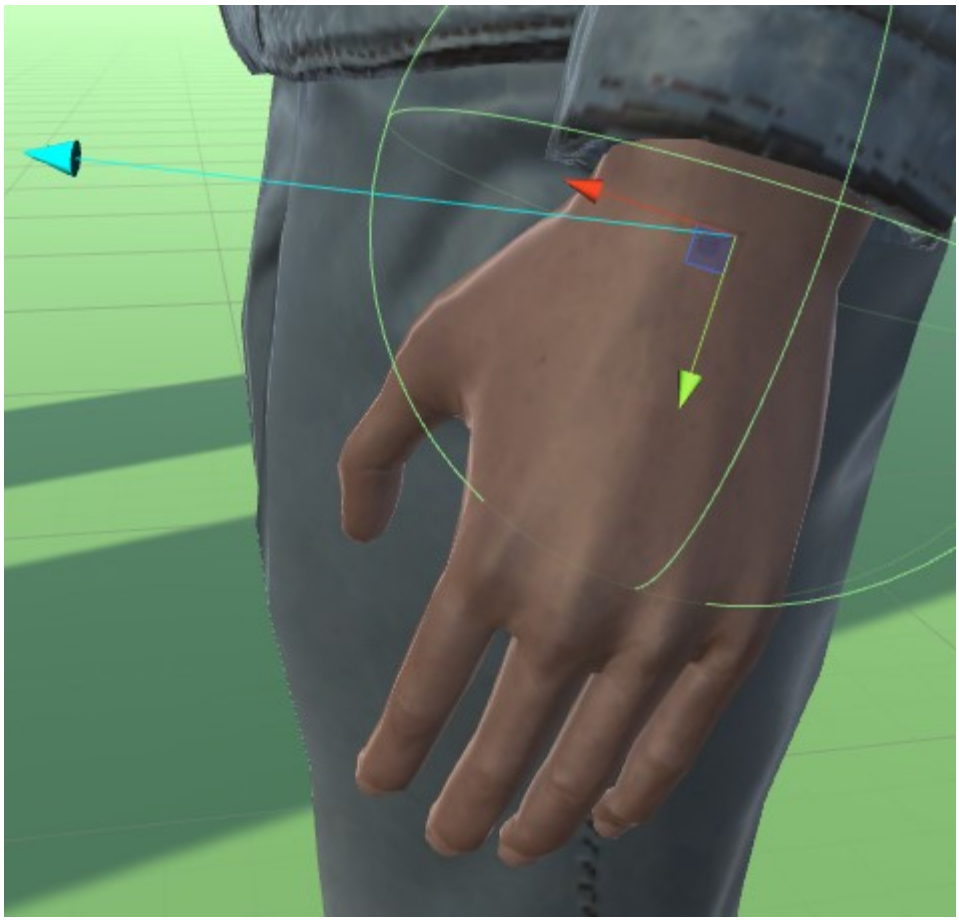


If we look at the hand shown below, and the axes of its transform, we can see that the x-axis is that hand's rotation axis, and that the hand's forward direction points along

the y-axis. And the palm would rest on a flat surface, if it's z-axis is pointing towards that surface, making it the palm-facing axis.

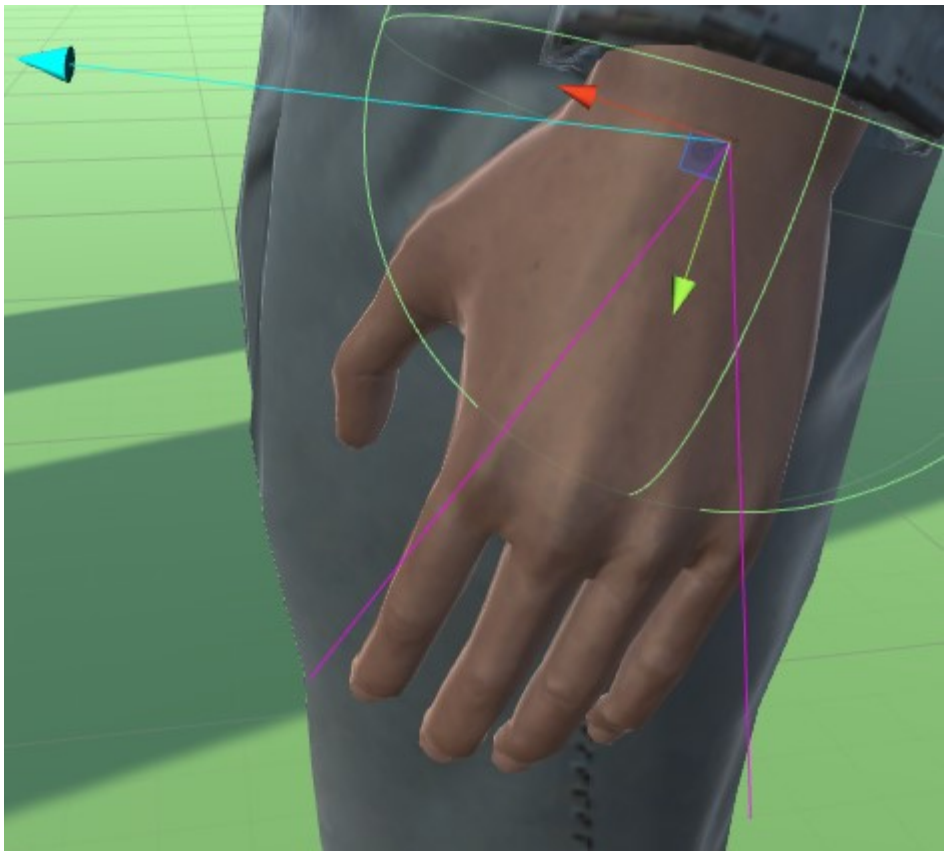


Where the previous axes are generally aligned with the coordinate system of the hand, the Hand Alignment Axis is somewhat different. This axis determines how a hand is likely to align with an object it's graping, along a fundamental edge of this object. To illustrate, think of holding a long thing tube, or a bottle in your closed hand. If we use the hand example above, that object would be mostly align with the z-axis, but at a slight angle. This direction is the direction of the Hand Alignment Axis.



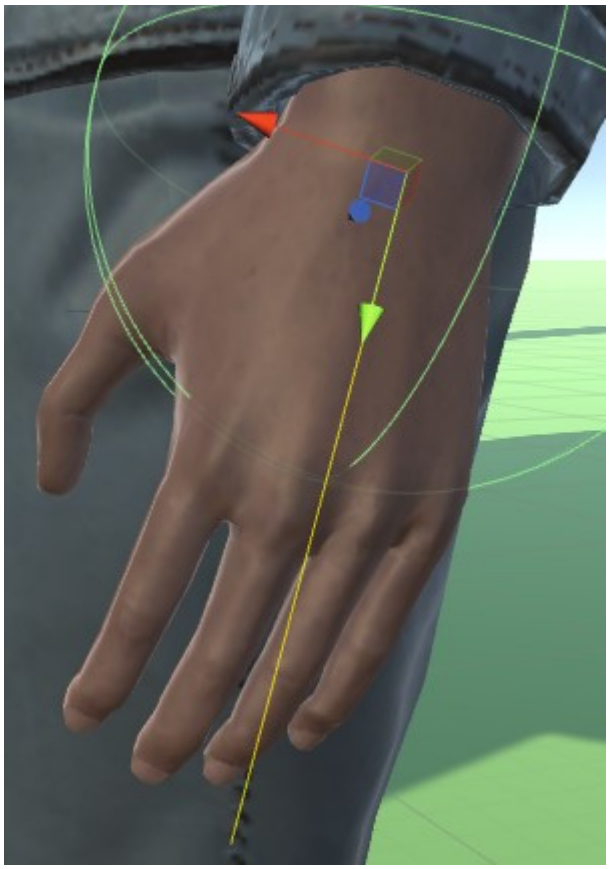
The procedural animation of the hand will rotate the wrist along the Palm Facing Axis, to best align the hand with the object it's interacting with. It uses the Hand Alignment Axis to determine the alignment.

The side-to-side deviation of the wrist is limited by the Deviation Angle. It limits the range of motion for hand alignment with an object.



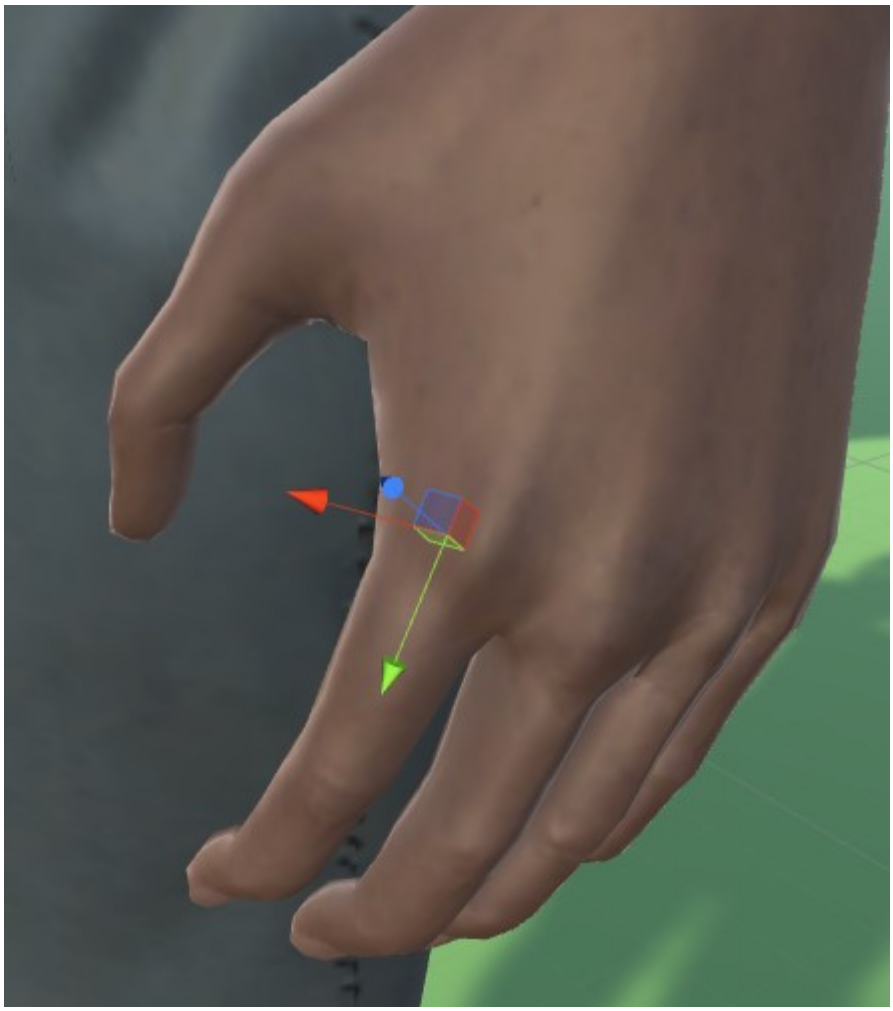
The Reach of a hand describes the length of the hand. It is used to determine what objects are within reach of the hand to be interacted with. Objects beyond this reach will be ignored.

The reach can be visualized by clicking on the "Show" toggle, which displays a vector in the forward direction, of the length of the set reach.

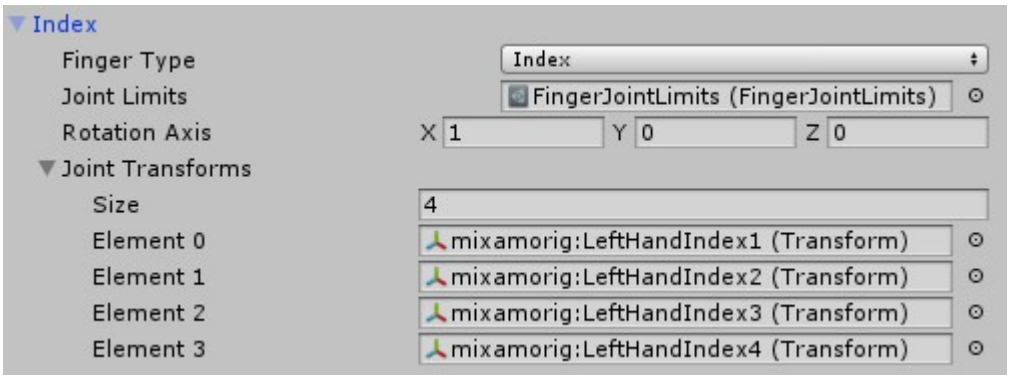


Setting up finger axes

Fingers are set up with a single axis of rotation, which is expected to be the same for each joint. In the illustration below, that would be the x-axis.



Set up the axis by setting the approprate values in the Rotation Axis field for each finger. Note that while the axis will generally be the same for most fingers, it may differ for the thumb.



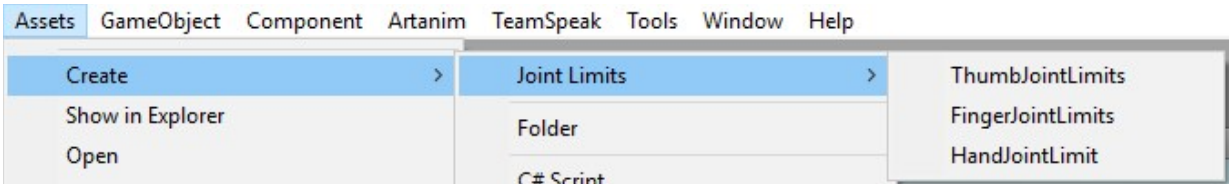
Hand and Finger Joint limits

The hand, as well as the fingers, have joint limits which determine the range of motion for each joint. There are 3 types of joint limits:

- ThumbJointLimits

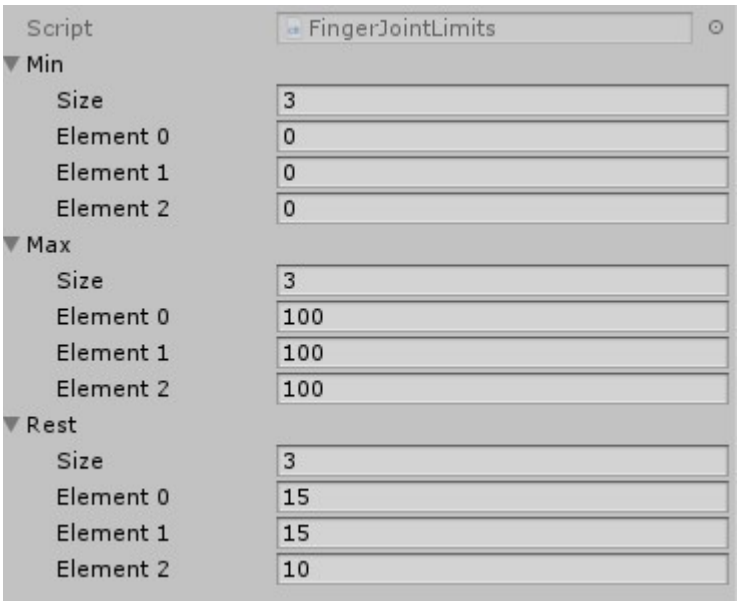
- FingersJointLimits
- HandJointLimit

These limits are stored as assets in your project, and can be created via the menu, by going to `Assets > Create > Joint Limits`.



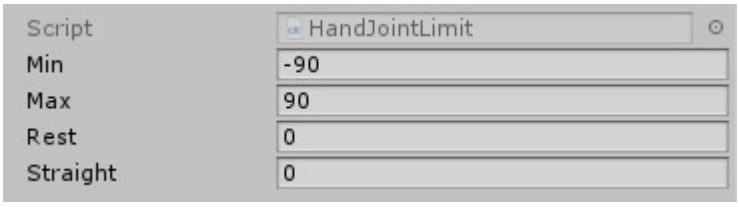
In general a single joint limit asset for all fingers suffices, as well as a single joint limit asset for the hands. You do however need an individual asset for both the left and right thumb, due to stored settings for the various thumb poses (see below).

Finger joint limits store values for all joints; that is, all but the last joint transform which generally indicates the finger tips.



The values indicate minimum and maximum range of motion in degrees for each joint along a given rotation axis, as well as an angle for the joints at rest.

The setup for the hand joint is similar, but of course with values for a single joint. In addition it separates a rest angle out from a straight angle value.



Thumb joint limits and poses

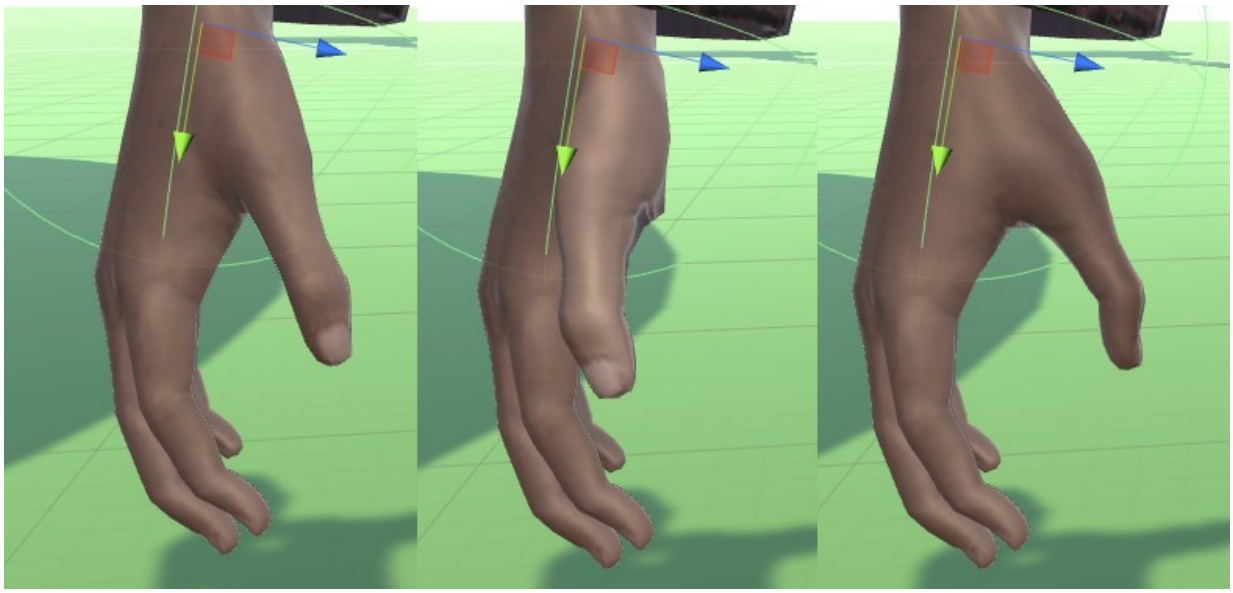
Thumb joint limits follow the same logic as those for other fingers. However, thumb joint limits store additional pose info. That is, the orientation of the first joint with respect to various actions: Neutral, Touch and Grasp.

The procedural hand animation, when interacting with objects, determines what the most likely interaction is that a user is having with that object. Is it just touching it, is it grasping it (i.e. is the user holding on to the object) or is there no interaction at all. These interactions require different start poses for the first joint of the thumb. To get technical, a grasp pose requires abduction of the thumb, and a touch pose requires adduction of the thumb.

When looking at a ThumbJointLimit asset, you'll see these poses expressed as 3 quaternion values. Don't worry though, you don't need to manually enter quaternion values. To aide in setting up these poses, the HandController's inspector allows you to store a new orientation or display an existing orientation.

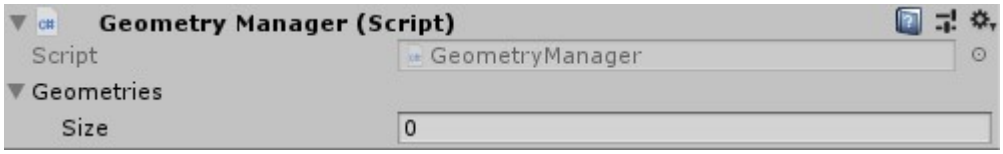
Make sure your thumb has a ThumbJointLimit asset assigned, and set the first joint of the thumb in the correct pose for an interaction. Click on the "Save Pose" button for the appropriate pose, and it will be stored within the ThumbJointLimit asset. The "Show Pose" button will appy a stored value to the thumb to illustrate what has been stored.

A illustration of thumb poses, from neutral to touch and grasp, can be found in the image below.



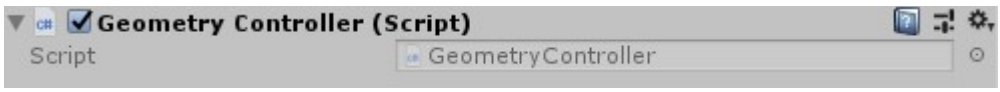
Setting up geometry to interact with

For procedural hand animation to interact with its environment, meshes will need to be set up for this interaction. In effect, we'll add a custom collider to those meshes. But first of all we need to make sure our scene has a GeometryManager.

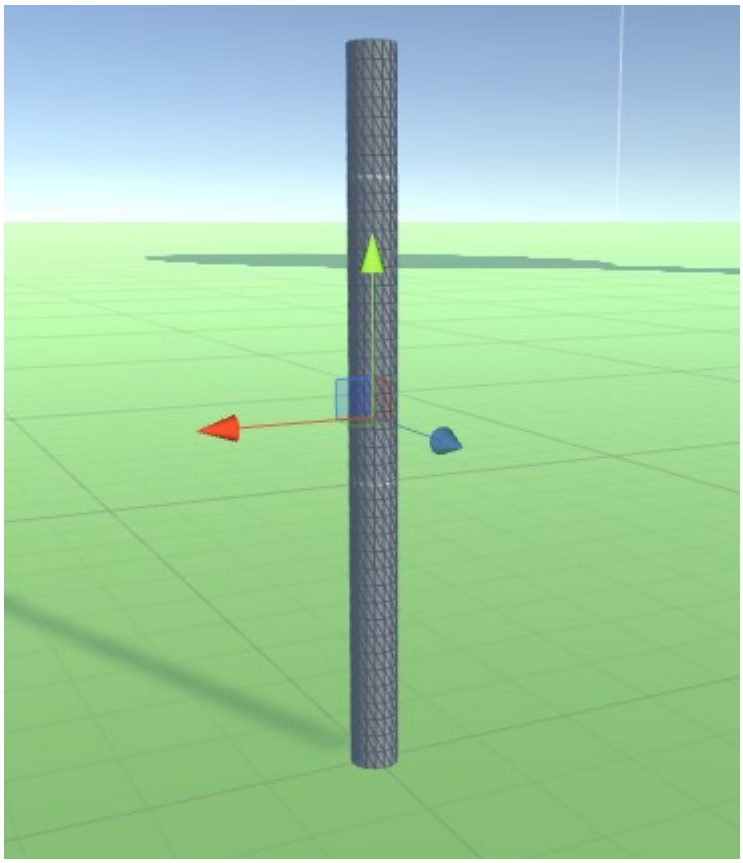


Add it to an object in your scene, and you're done. Each object with which a HandController can interact, will register itself with this GeometryManager whenever it's enabled.

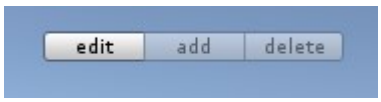
To set up objects for interaction, we need to add a GeometryController component to it on the GameObject which has a MeshFilter set up.



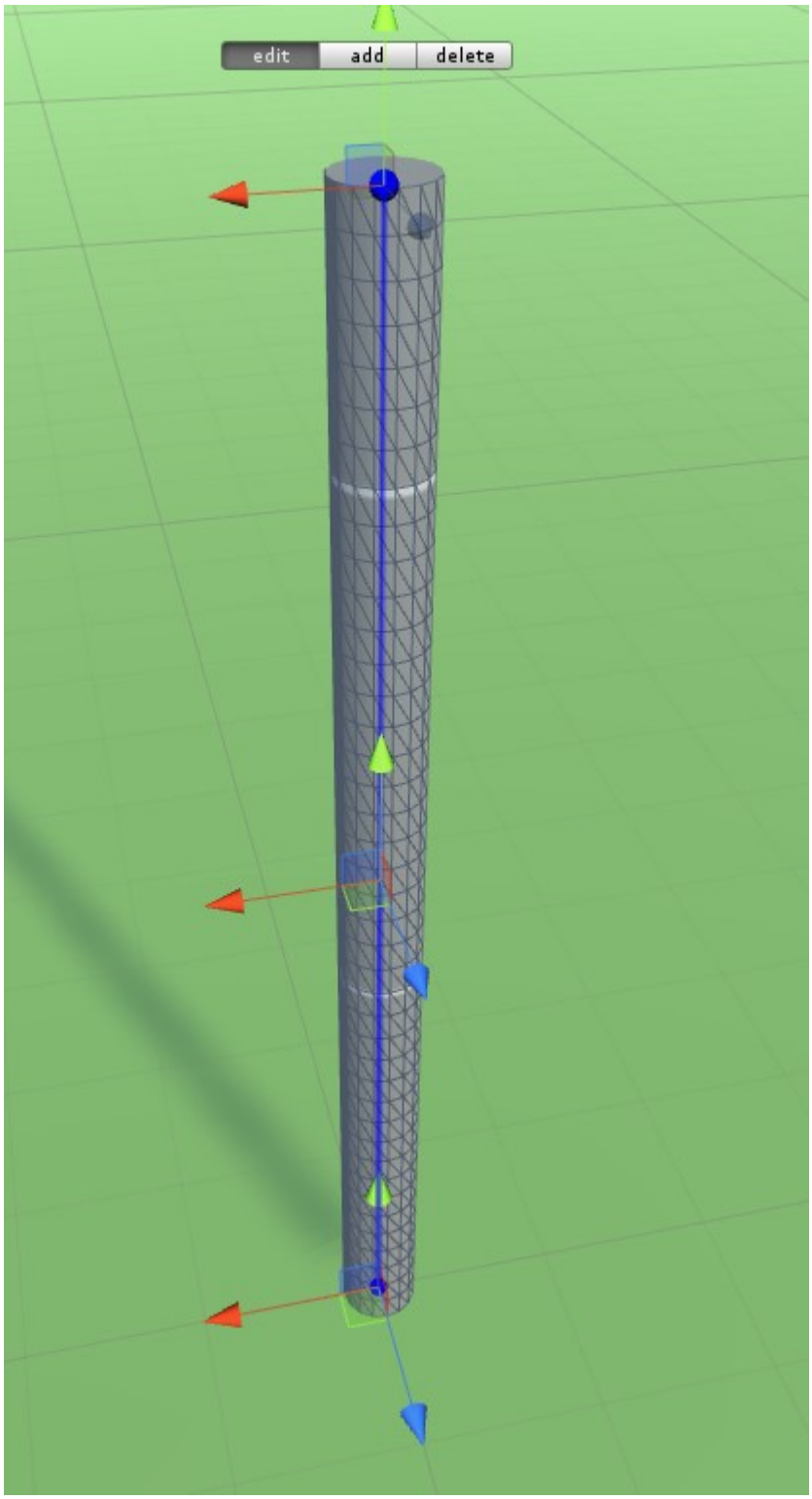
For basic interaction, that's all there is to it. However, recall that when setting up hand axes we discussed setting up a Hand Aligment Axis. For this to function we need to annotate an object with what its fundamental alignment edges are. Let's take for example a rod as illustrated below.



If we think of holding this rod in our hands, it becomes obvious that its fundamental edge would have to align with the model in its longest direction. So let's add the edge. When selecting the object containing a GeometryController, you'll see 3 buttons appear in the scene view.



To start editing the controller, click on the "edit" toggle. If any alignment edges have been set up, they will appear. A new edge can be added by clicking on the "add" button, and an existing selected edge can be removed by clicking on the "delete" button.



An edge can be moved in the right place by selecting it, and moving its handles into the correct location. Once set up, a hand will try to align itself with this edge if it's the closest edge in range, and if it's within the limits of the deviation angles which have been set up.