CeMSIM-ORSim VR Avatar Setup Tutorial

Written by: Douglas Burnside

Table of Contents

Before Getting Started

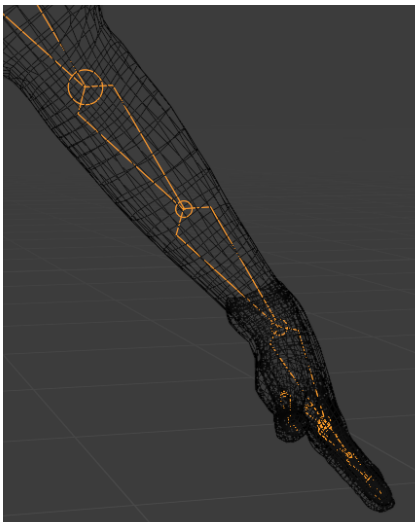
It is recommended to watch these video tutorials before creating an avatar. This document will also cover the steps in the video but there are additional steps that will be included.

- HurricaneVR VRIK Tutorial
- HurricaneVR Grabbable/Hand Pose Tutorial

Setting Up Avatar Models

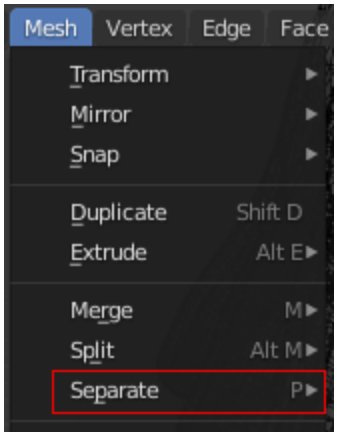
Bone Anatomy

All avatars must have standard humanoid bone anatomy. However, one small change will be necessary to allow for wrist bending improvements. The forearm must be split into two equal halves and skinned accordingly. This can be done for any bone that needs twist improvements.



Mesh Skinning

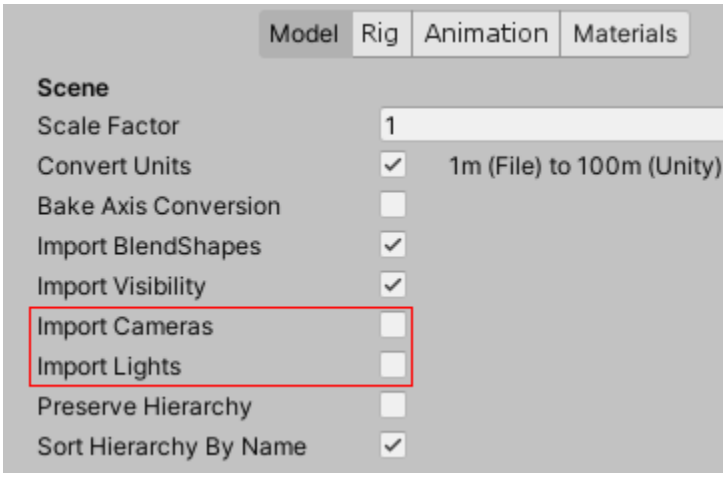
If there are any separate clothing objects that will be placed on top of the model's body, the polygons that are underneath the cloth may need to be hidden when the clothing is active to prevent clipping. To implement this properly, the whole model's body will be skinned first. Then, in edit mode, select which polygons are underneath the cloth and separate them into a new object as a child of the model. This will keep the skin weights from the previous step while allowing that mesh to be disabled as a separate GameObject in Unity.



Importing Avatar Models

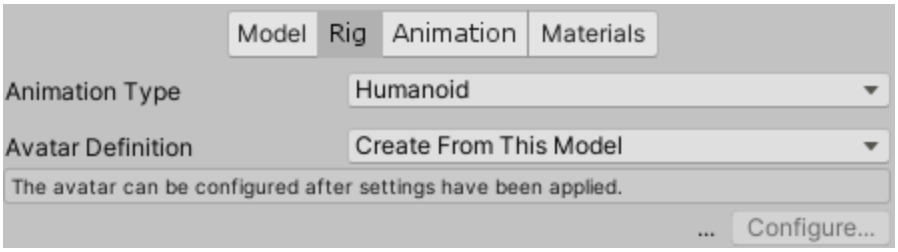
Model Import Settings

To prevent importing cameras and lights from the file, navigate to the model import settings and deselect *“Import Cameras”* and *“Import Lights”*.



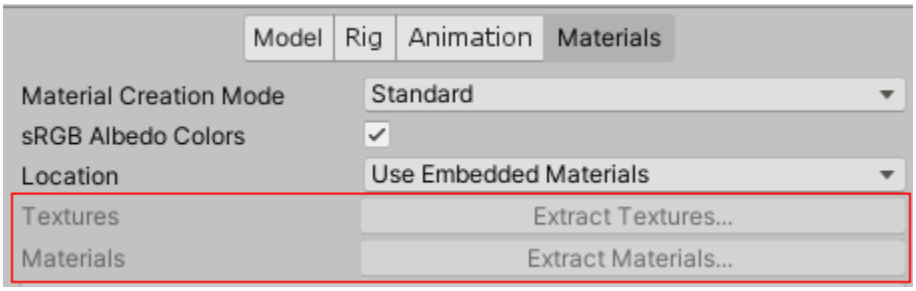
Animation Avatar Definition

If the avatar uses custom animations, it may need an animation avatar to properly animate. Navigate to the rig import settings and choose the *“Humanoid”* animation type, then apply the settings. If Unity does not automatically detect all of the bones, the avatar may have to be configured manually to assign them. For this, Click on *“Configure…”* and add any missing bones to the avatar.

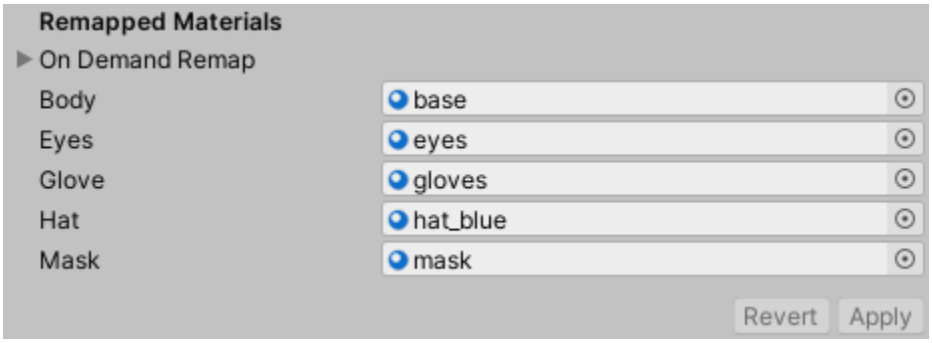


Material and Texture Extraction

If the materials on the avatar will be modified in-engine, they will need to be separated from the blend file in the project. Navigate to the material import settings and click on *"Extract Textures.."* if the textures are contained in the file and *"Extract Materials…"*. Both of these options will open a prompt asking where to save these new assets. This will create new Texture and Material assets based on their name in the blend file. Textures are typically stored in "*Assets* > *Project* > *Textures"* and Materials are stored in "*Assets* > *Project* > *Materials"*. The location where these are stored may change based on which branch you are using.



Afterwards, the materials will need to be remapped in the model's material import settings. Under the *"Remapped Materials"* section, the newly created materials from the previous step can be assigned. After they have been remapped, click the *"Apply"* button to save the changes.



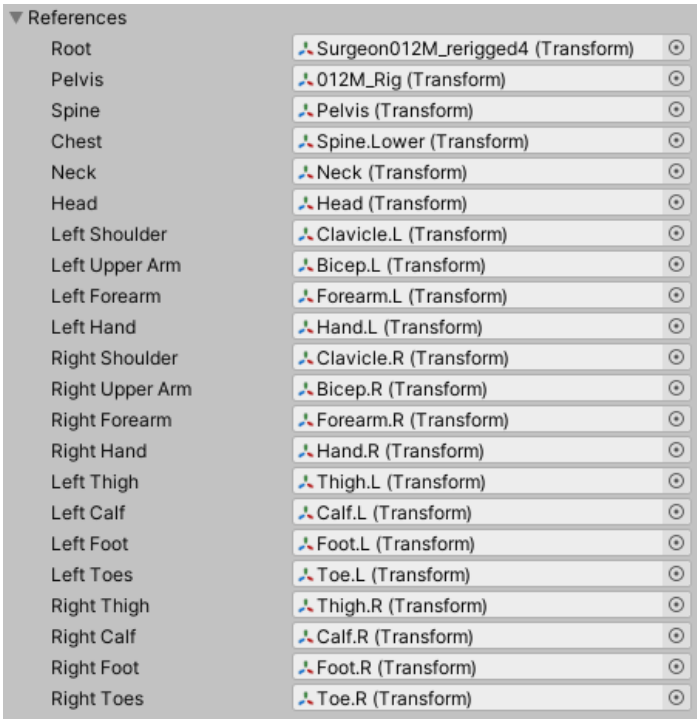
Setting up the Avatar Prefabs

Creating the Avatar Prefab

To create the avatar's prefab, create an empty GameObject and add the avatar model as a child. Name the empty object accordingly and drag it into the asset browser. Avatars are typically stored in *"Assets > Prefabs > Avatars"*.

VR IK

To add IK to the avatar, add the *"VRIK.cs"* script to the base object. If the bone names are not recognized by the script, the bone references may have to be added manually. The *"Root"* transform must be the model object. The *"Pelvis"* is the first bone in the armature.



In the solver, the *"Plant Feet"* setting changes whether the avatar will be forced to touch the ground at all times, regardless of head height. This may be disabled to improve performance slightly. Many other settings can be adjusted to change how the avatar follows its targets. For now, the target fields will be left empty.

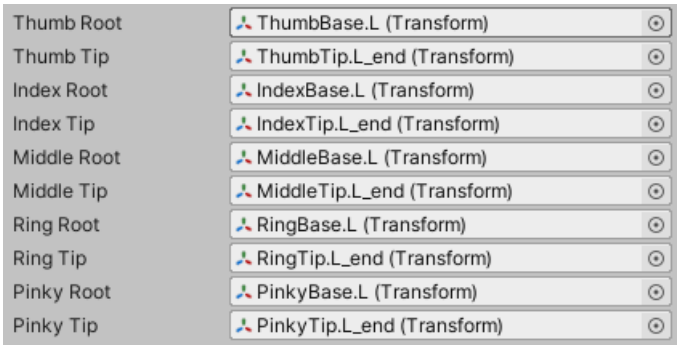
HVR Physics Poser

On both hands, there are four main scripts that will need to be added. The first one is *"HVRPhysicsPoser.cs"*.

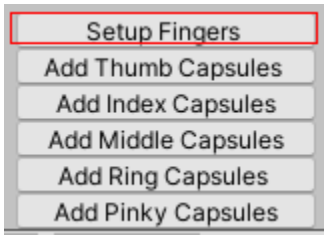
This will need two references. The first one is the *"HVR Posable Hand"* component that will be added next. The second is a palm reference position. To create this, make an empty object as a child of the hand that is located in the center of the palm, pointing outwards.

HVR Posable Hand

The next script that will be added to the hand is *"HVRPosableHand.cs"*. If the hand is the left hand, enable the *"Is Left"* option. The next few references will include the base and tip bone of every finger.



After adding every bone object, click the *"Setup Fingers"* button near the bottom of the inspector section.



The hand mirror settings will also need to be added to the *"Mirror Settings"* reference.

HVR Hand Poser

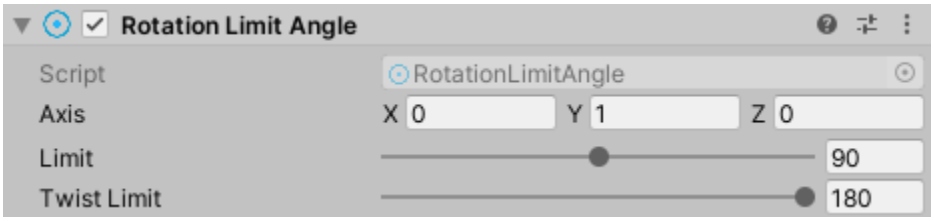
Another script to be added to the hand is *"HVRHandPoser.cs"*. This script will contain any hand poses created in a later step.

HVR Hand Animator

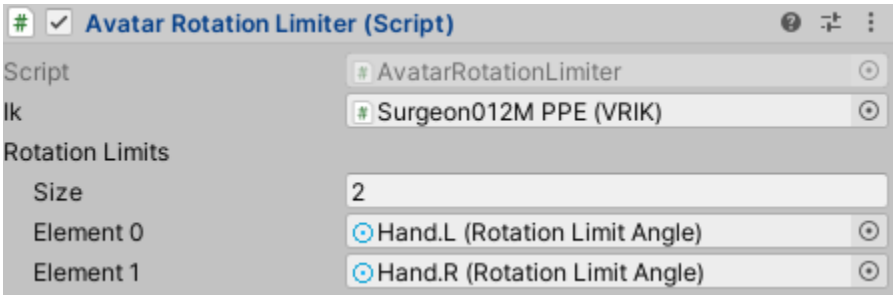
The last HVR script to be added to the hand is *"HVRHandAnimator.cs"*. Unselect the *"Pose Pos and Rot"* setting and add the three other HVR scripts to the components section.

Rotation Limits

Another script that will be added to the hand is *"RotationLimitAngle.cs"*. This script will limit how far the hand can rotate to ensure the hand's rotation looks natural. The axis vector should only have a 1 in the axis that points away from the arm and a 0 for the other 2 axes. The limit restricts how much the hand can rotate from its original rotation. The twist limit restricts how much the hand can twist along the arm's axis. A gizmo will appear when the script is added to help visualize the angle restrictions.



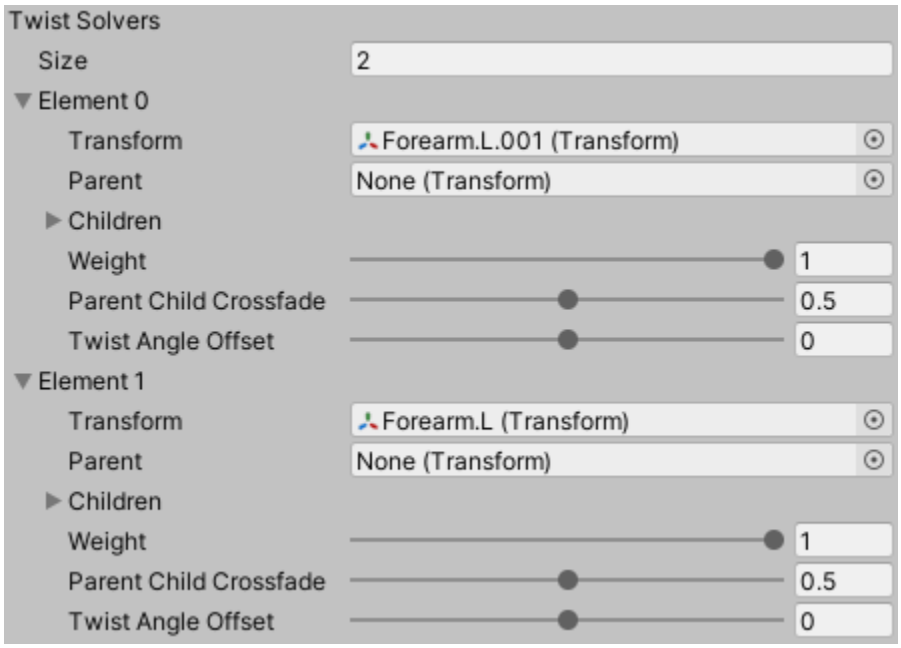
After adding this script to both hands, the *"AvatarRotationLimiter.cs"* script will be added to the base object. The *VRIK* and rotation limit scripts will all need to be added to this script for them to update properly.



Twist Relaxer

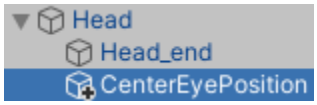
To allow the wrist to rotate properly with the hand, add the *"TwistRelaxer.cs"* script to the second forearm bone. Also, add the *VRIK* script to the IK reference.

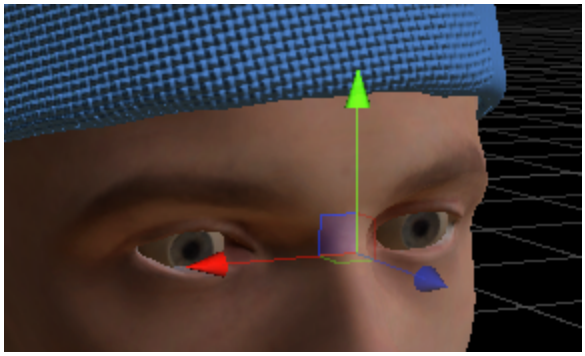
There will need to be two twist solvers added for the wrist to rotate properly. The proper settings can be seen in the image below.



Height Calibration

To scale the avatar according to the user's height, two scripts will be added to the base object: *"Avatar Height Calibration.cs"* and *"AvatarPrefabHeightUtility.cs"*. Under the *Avatar Prefab Height Utility*, the *"Avatar Floor"* reference is the object that is located at the floor level under the avatar's feet, typically the root of the model. The *"Avatar Eyes"* reference is an object that needs to be created as a child of the head bone that marks the center position between the eyes. Once this is done, click *"Calculate Height"* to retrieve the height of the avatar. This also has to be saved on the prefab.



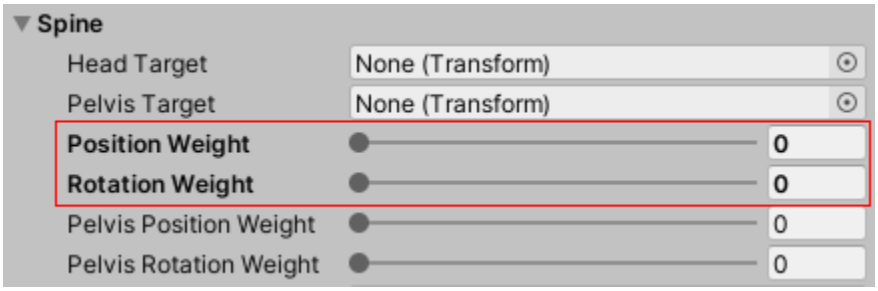


The *Avatar Height Calibration* script will need references to both the *VRIK* and *Avatar Height Utility* scripts. The *"Height Adjustment Multiplier"* is applied after the height is calibrated and is typically increased if the avatar's feet aren't fully planted when standing up straight at runtime. To calibrate the height manually, call the *"Calibrate()"* function from this script. The *"On Avatar Height Changed"* event can be used to call a function when the avatar's height is calibrated.

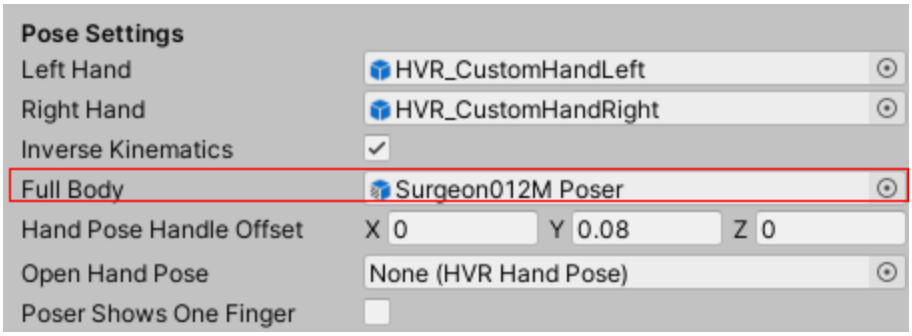
Creating the Poser Prefab

Before creating hand poses, a prefab will need to be made that is spawned when the pose preview is enabled. The poser prefab will be a prefab variant of the avatar prefab, but a few components will need to be removed for it to function properly. After creating the variant, remove the following components from it: *Avatar Height Calibration*, *Avatar Prefab Height Utility*, and *Avatar Rotation Limiter*. Then, add the scripts *"HVREditorIK.cs"* and *"HVRIKTargets.cs"*. Nothing will need to be changed in these scripts.

Since the poser's spine will have no target, the IK needs to be disabled for those bones. Under the spine section, set the position and rotation weight to 0.

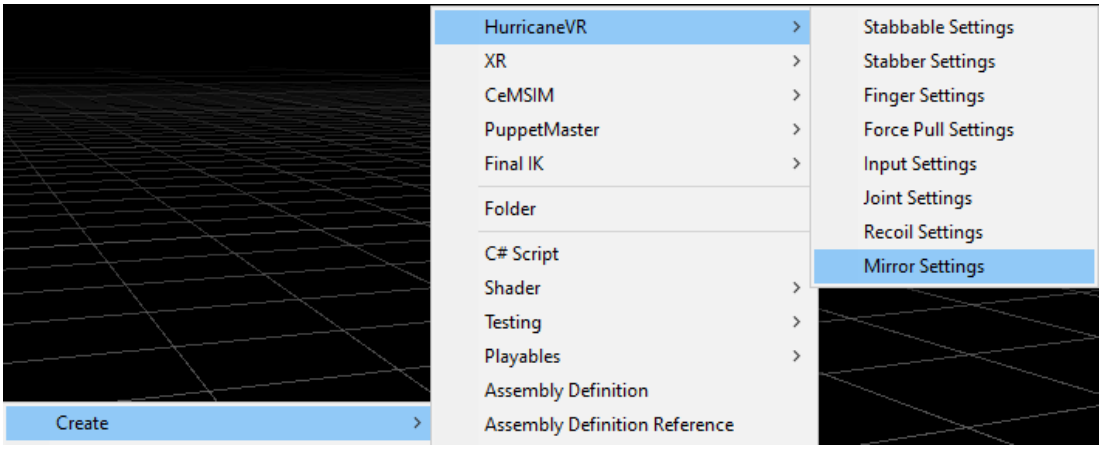


To use the poser, find the *"HVRSettings"* asset in the asset browser and add the poser prefab to the *"Full Body"* reference.

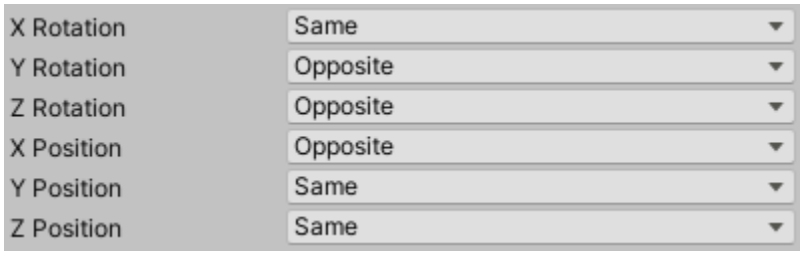


Hand Mirror Settings

While creating hand poses, it may be necessary to mirror the pose from one hand to the other to accelerate the process. To do this, a *"HandMirrorSettings"* asset will need to be created and added to the avatar's hands. This will store the differences between each finger's translation and rotation directions.



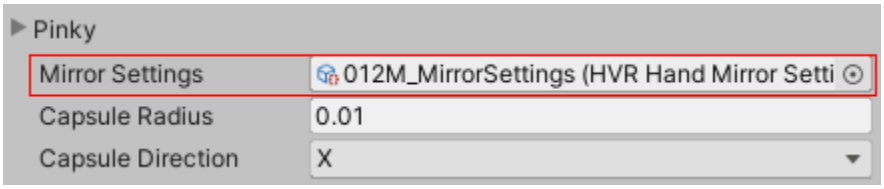
The *"Use [Finger] Setting"* option should only be checked if all 3 bones of that finger have the same rotation settings. The options that correspond to this option is the *"[Finger] Setting"*. Otherwise, use the *"[Finger] Settings"*. For the size of each of these settings, there will be 3 elements for each of the 3 finger bones. The bone index is 0 for the root and 2 for the tip. Usually, the fingers will use the same settings but exceptions can happen. An example of the settings are shown in the image below.



If a bone does not mirror correctly, then it must be changed here. To do so, move and rotate that finger bone from both hands. This must happen in a direction mirrored across the center of the avatar. For *"[Axis] Position"*, move the bones in the same direction. If the positions both increase or decrease, use *"Same"* as the mirror setting. Otherwise, if one position increases and the other decreases, use *"Opposite"*. The *"[Axis] Rotation"* setting can be tested in a similar manner.

The forward and up (Z and Y) axes of the avatar usually have the same translation but opposite rotation. The left/right (X) axis will usually have opposite translation and the same rotation. However it must be noted that the bone must move outwards away from the hand on the X axis to know its mirror setting. Below is an example of when the X axis has the "Opposite" setting. Since both axes point in the same direction, they must be moved in the opposite direction to have a mirrored effect.

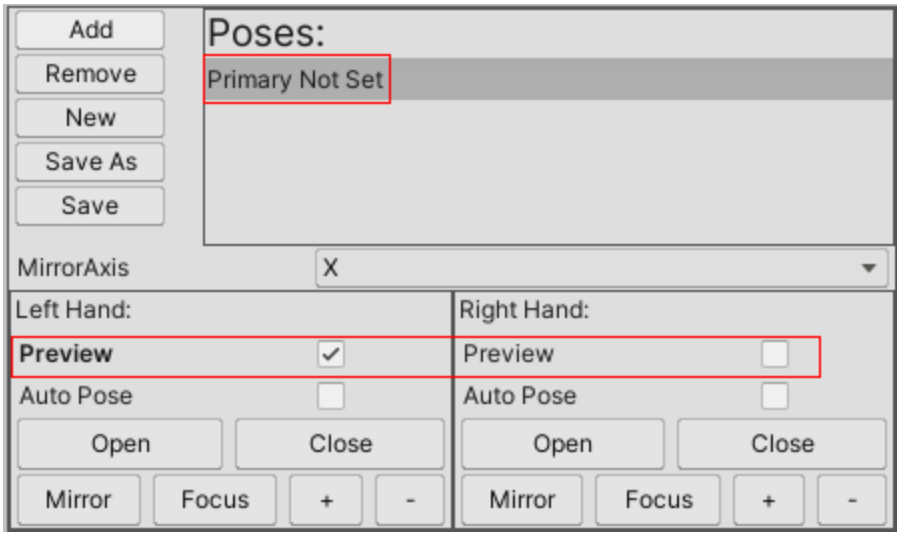
Once the mirror settings are finished, add this asset to the *HVR Posable Hand* component's *"Mirror Settings"* on each hand.



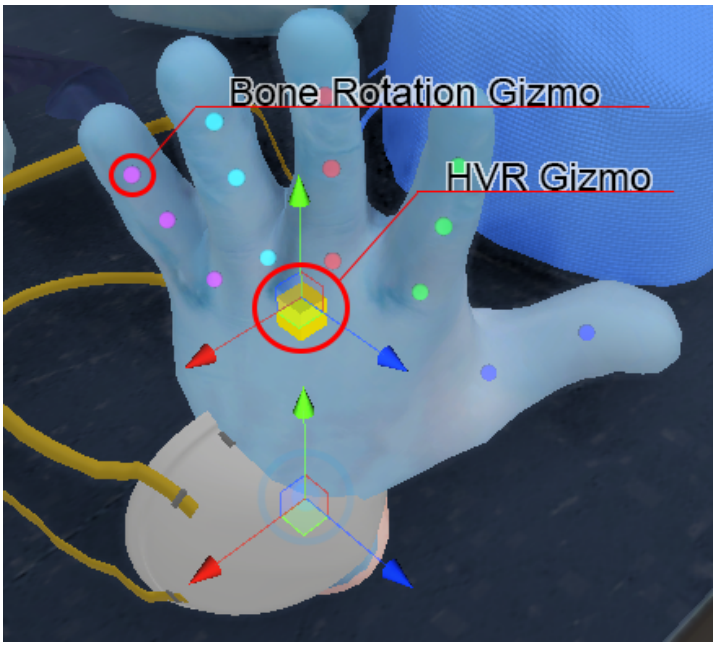
Adding Hand Poses

When creating a posable hand, at least two hand poses will need to be added to the Hand Poser to allow the hand to swap between a resting pose and a grabbing pose when the trigger or grip is pressed. When first created, the primary hand pose will be empty. It is recommended to create an open hand and closed fist pose for this poser.

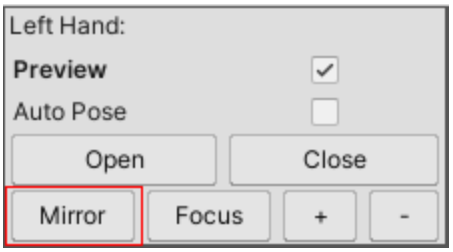
To edit and create new grab poses, enter a scene, create an empty GameObject, and add the *"HVRHandPoser.cs"* script to this object. Then, create an object of any basic shape as a child of that empty object to act as a placeholder for a grabbed object. Click on *"Primary Not Set"* to select the pose that will be edited, then click *"Preview"* under either hand to begin editing the pose.



The avatar poser prefab will appear and the hand will be targeted to the origin of the object by default. The hand position should be primarily modified by the GrabPoint object, since this will be the anchor position. The yellow HVR gizmo can be used for small translation adjustments and rotation. Clicking on the yellow cube toggles between translation and rotation of the hand. To modify the finger bone rotations, click on one of the bone rotation gizmos.

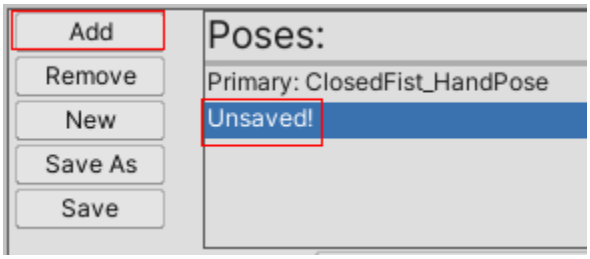


After modifying the bone rotations and hand position, the hand pose can be mirrored to the opposite hand by clicking *"Mirror"* on the hand that was modified. This will mirror the pose based on the *HandMirrorSettings* on the avatar.



The pose will also need to be saved. To do so, click either “Save” or “Save As” and choose the location for the asset to be saved to and its name. Hand poses are typically stored under *“Assets > Project > HandPoses”*.

To add an additional pose to the hand, click *“Add”* and then click *“Unsaved!”* to start posing.



After adding any hand poses, the poses will also need to be added to the original avatar prefab for the poses to be used during runtime. The poses do not have to be made again. To add a pose, insert the saved pose asset under the *“Pose”* reference.

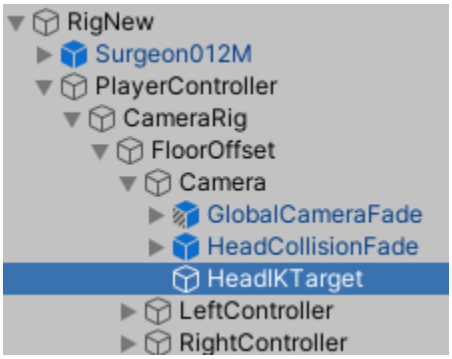
Attaching the Avatar Prefab to an HVR Rig

Creating the VR Rig

The avatar VR rig will use a pre-built rig from HVR but it will be modified to allow the use of a custom model. Find the prefab called *"RigNew"* under *"Assets > HurricaneVR > Framework > Prefabs"* and add it to a scene. Unpack this prefab (not completely) to allow modifications to this rig. Add the avatar prefab as a child of the base object and save it as a new prefab. In order to allow this rig to control the avatar, a few references need to be changed.

IK Targets

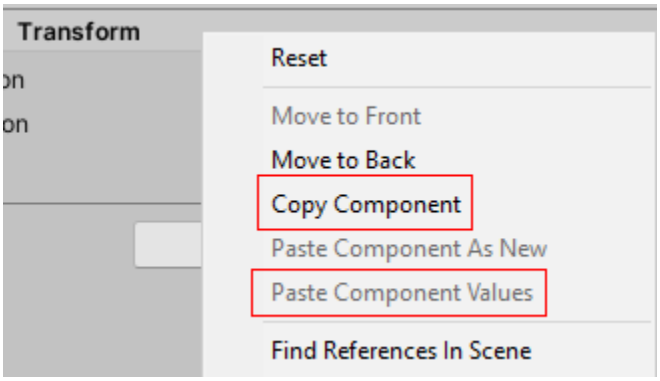
To allow the avatar to follow the HMD and controllers, IK target objects need to be created and added to the *VRIK* references. For the head target, create an empty object under the *"Camera"* object.



For both of the hands, create an empty object under the *"LeftHand"* and *"RightHand"* objects. These three objects will be included as references under the avatar's *VRIK* solver section. Drag the head target into the spine's *"Head Target"* and the two hand targets into the left/right arm target.

These targets can be adjusted to more accurately follow the hand and head positions, since they will be treated as an offset from the tracked position. There are two prefabs under the left and right hand objects that can be used as a reference for where the hand positions should be. These objects can be disabled after this step. To make adjustments, run the program and make

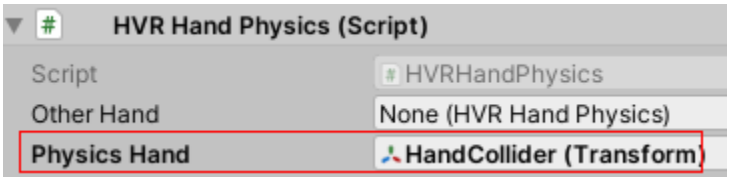
adjustments to the objects at runtime. Copy the transform component values for that specific object and paste the component values in that same object when the program is stopped.



Hand Collider

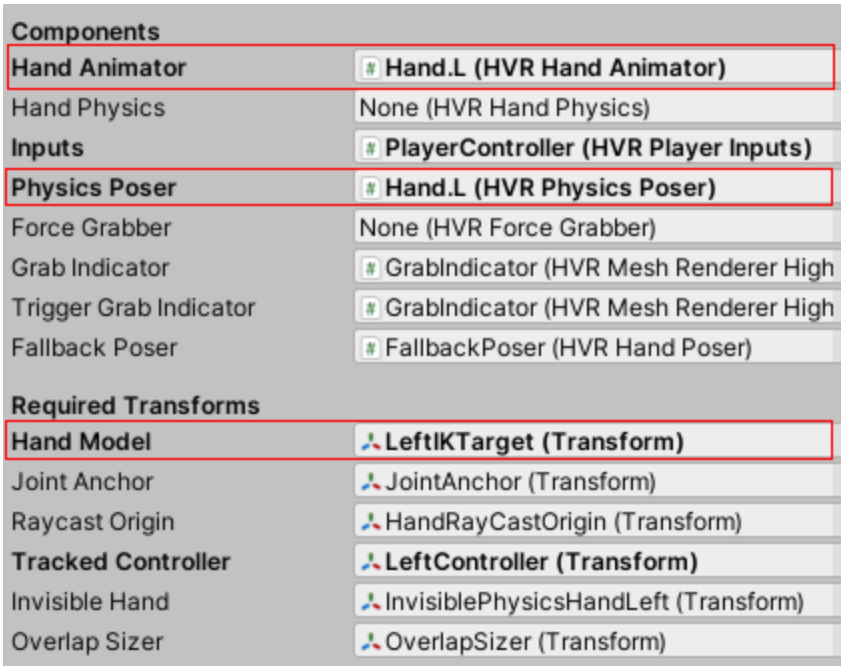
A collider will need to be added to the hand for the *"Hand Physics"* component of each rig hand. For a simple collider, create an empty object as a child of each rig hand and another object with a collider component as a child of that. Resize and align the collider so that it covers the bounds of the hand. If a more complex collection of colliders is desired, add the object that is a parent of every collider. This would be the avatar's hand object if each finger bone has a collider.

This will then be added to the rig hand's *"Hand Physics"* component as a reference for the *"Physics Hand"*.



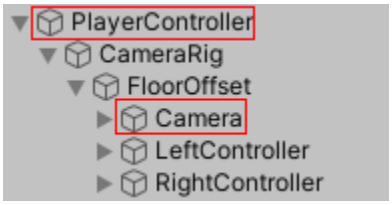
Hand Component References

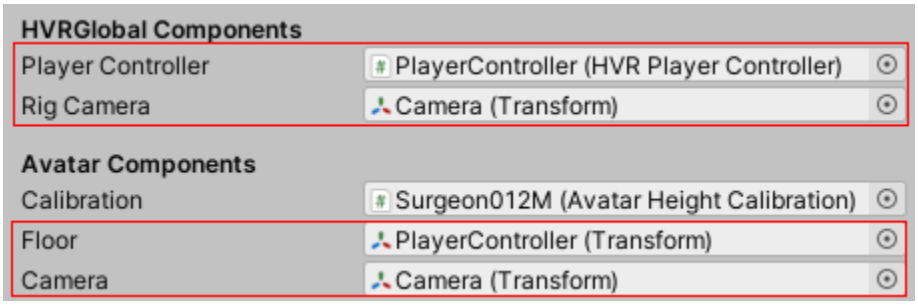
There are a few references that need to be replaced so the hand can be posed with this rig. Under the *"Components"* section of each rig hand's *"Hand Grabber"* component, replace the *"Hand Animator"* and *"Physics Poser"* with the components from the avatar's hands. Then, the *"Hand Model"* will be replaced by the hand's IK target.



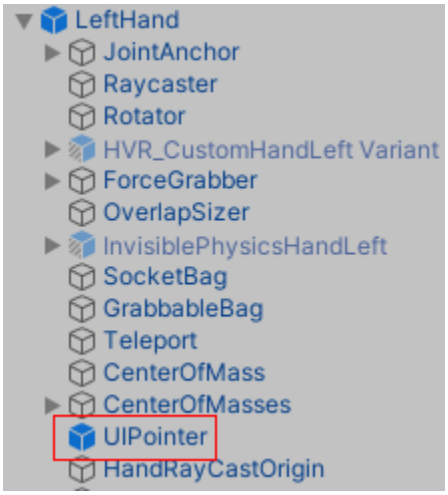
Avatar Components

The *"AvatarComponents.cs"* script will need to be added to the base rig object which contains references to objects and components that need to be modified by the avatar spawner. The *"Player Controller"* is a component of one of the child objects of the rig with the same name and the *"Rig Camera"* and *"Camera"* are contained as a nested child of the *PlayerController* object. The *"Floor"* object will also be set to the PlayerController object. The *"Calibration"* reference must point to the *Avatar Height Calibration* component contained on the avatar prefab.





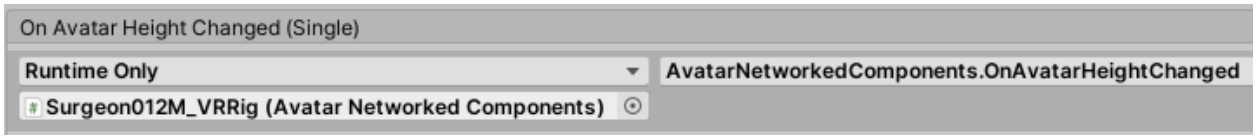
The *"Left/Right Pointer"* components are contained on objects named *"UIPointer"* that are children of the *Left/Right Hand* objects.



Networked Components

If the avatar will be networked, the *"AvatarNetworkedComponents.cs"* script will need to be added, which contains references to the rig's head and hand tracking objects. The *"Left/Right Controller"* references will contain the *Left/Right Hand* objects and the *"Hmd"* reference will contain the camera object. The *"Height Calibration"* reference will contain the *Avatar Height Calibration* component from the avatar.

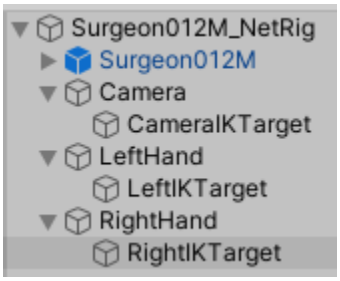
Additionally, the avatar's height calibration script must call the networked components' event when calibration occurs. On the avatar's *Avatar Height Calibration "OnAvatarHeightChanged"* event, add the networked components' *"OnAvatarHeightChanged.Invoke()"* method as a dynamic listener.



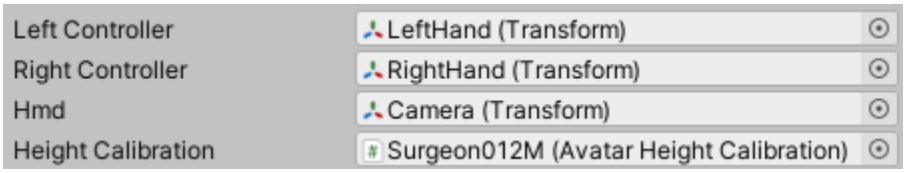
Creating the Networked Rig

If the avatar will be networked, a separate prefab will need to be made to show external users in the local scene. This will be somewhat similar to the VR Rig, but only a few components necessary for synchronization will be added.

To start making this net rig, create an empty GameObject and add the avatar prefab as a child. Then, a few reference IK targets will need to be created since their positions will be synchronized. The hierarchy of these objects is shown in the image below.



Then, to allow the targets to be followed by the avatar, insert the IKTarget objects into their references in the *VRIK* component similarly to how they were added in the VR Rig. The *Avatar Networked Components* script will also need to be added and the references will need to be added similarly to the VR rig. Usually, this avatar will receive a message from the server to update the calibration and it should not be called in other ways. The *"OnAvatarHeightChanged"* event will be called when this avatar is calibrated.

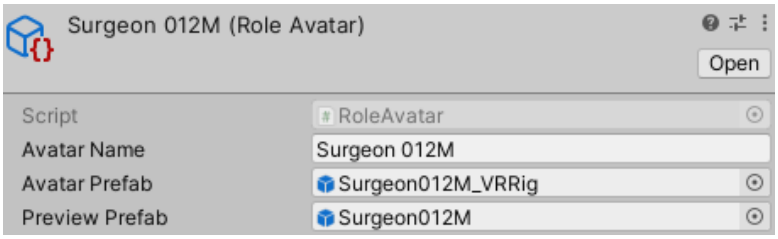


Including the VR Rig

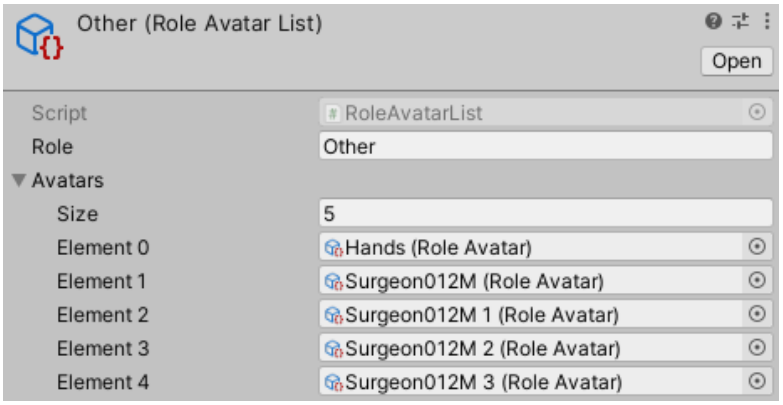
Adding a Role for Role Selection

To add a role to the avatar selection UI, an avatar and role asset will need to be created. Both of these assets will contain some information and objects needed to create them at runtime. These assets are located in the right-click menu under "*Create* > *CeMSIM* > *Role Avatar/Avatar*". These assets are typically stored under *"Project* > *RoleAssets"*.

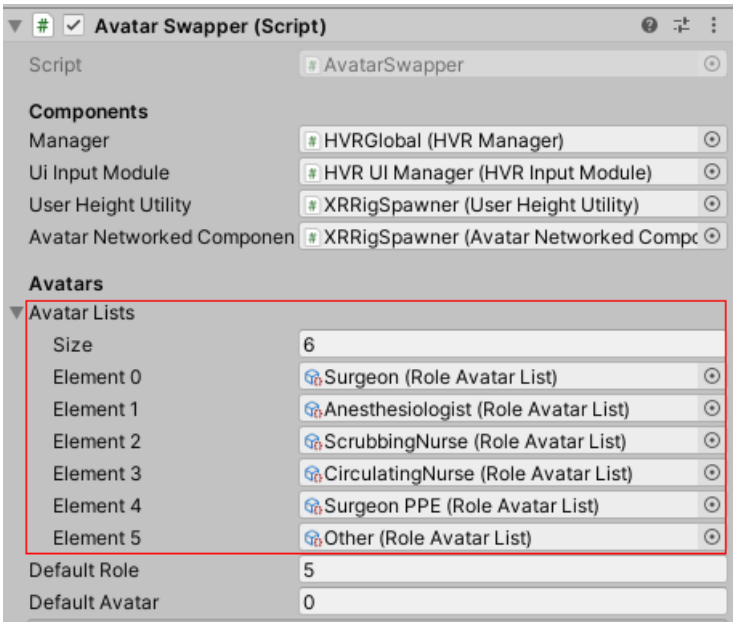
The *Role Avatar* asset will first need the avatar name and VR rig prefab to be included. The preview prefab is only used to show the user how the avatar looks when selected and may be left empty. If it will be included, it must be assigned to the avatar prefab or a separate preview prefab, not the VR rig prefab. The preview prefab will only be necessary if using the main prefab has additional scripts that will conflict with the avatar's scripts. It can be created as a variant of the avatar prefab if desired.



A Role asset will also need to be created that contains a list of all *Role Avatars* that will be an option for the role. The role name will be displayed in the role selection menu and can be used to determine which role the user will be.



Once this avatar list is created, it can be added to the scene's "XRRigSpawner" "Avatar Swapper" component in the avatar lists.


Avatar Swapper (Script)
Script
AvatarSwapper
Components
Manager
HVRGlobal (HVR Manager)
Ui Input Module
HVR UI Manager (HVR Input Module)
User Height Utility
XRRigSpawner (User Height Utility)
Avatar Networked Componen
XRRigSpawner (Avatar Networked Compc
Avatars
Avatar Lists
Size
6
Element 0
Surgeon (Role Avatar List)
Element 1
Anesthesiologist (Role Avatar List)
Element 2
ScrubbingNurse (Role Avatar List)
Element 3
CirculatingNurse (Role Avatar List)
Element 4
Surgeon PPE (Role Avatar List)
Element 5
Other (Role Avatar List)
Default Role
5
Default Avatar
0