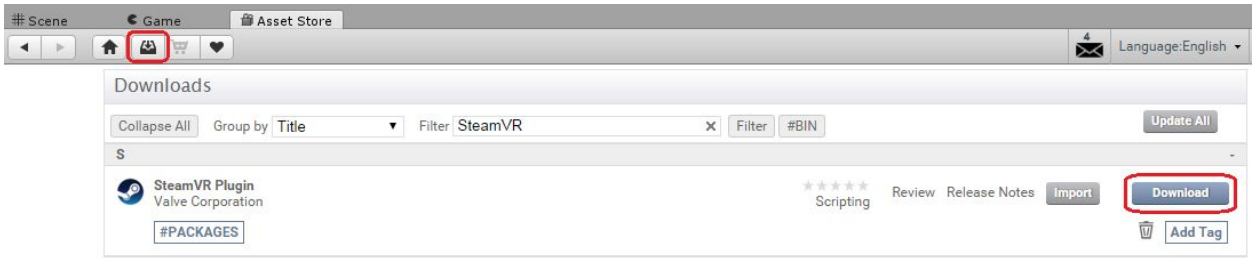

The SteamVR Unity plugin comes in three different versions depending on which version of Unity is used to download it.

1) **v4** - For use with Unity version 4.x (tested going back to 4.6.8f1)
2) **v5** - For use with Unity version 5.x (for versions less than 5.4)
3) **v54** - For use with Unity version 5.4 and newer (including 2017.x)

> *Note: Native OpenVR support was added to Unity starting with version 5.4, so you will see a lot of additional code dealing with rendering in the v5 version of the plugin. This has been stripped out in the v54 version. To make upgrading easier, the v5 version will also work with Unity 5.4.x and will auto-update itself. However, switching back to an older build of Unity requires reimporting the plugin to undo that process.*

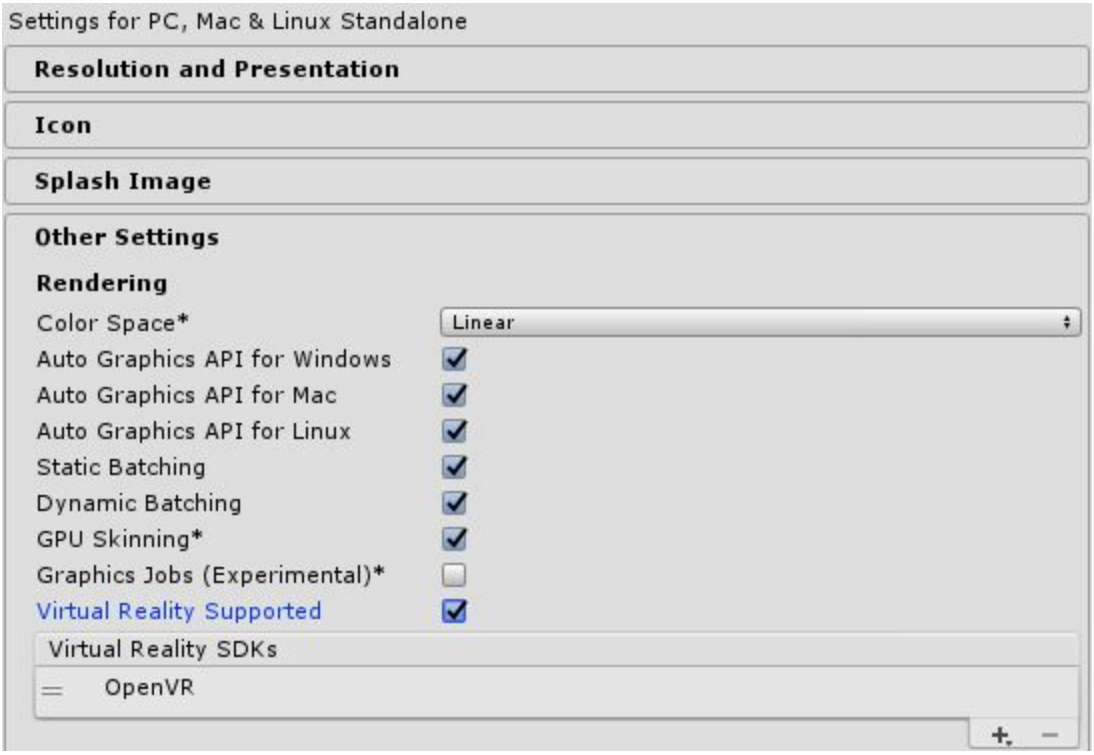
Unity will cache Asset Store downloads. To force redownloading, go to Asset Store, click on the Inbox icon next to Home, find SteamVR Plugin in the list and click Download. Then click Import to re-import into your Project. It is recommended you delete the SteamVR package manually from your project before reimporting in order to ensure old files do not remain.




The Lab's Interaction System is only included in v54 of the plugin. Refer to its own documentation under Asset > SteamVR > InteractionSystem.

## Native OpenVR support

In Unity 5.4 and newer, you can enable native OpenVR support under Edit > Project Settings > Player > Other Settings. Checking Virtual Reality Support will add a list of Virtual Reality SDKs. Use the + dropdown to add OpenVR, and drag it up and down in the list to control the initialization priority.




See more details here: https://docs.unity3d.com/Manual/VROverview.html

When importing the SteamVR Unity plugin, this setup is performed for you automatically.

# Example scene

The plugin includes an example scene which exercises the core functionality of the plugin. This includes the [Status] prefab which demonstrates how to use SteamVR with different legacy versions of UI support in Unity. This prefab is slow and should not be used in production scenes. It is provided for demonstration purposes only.

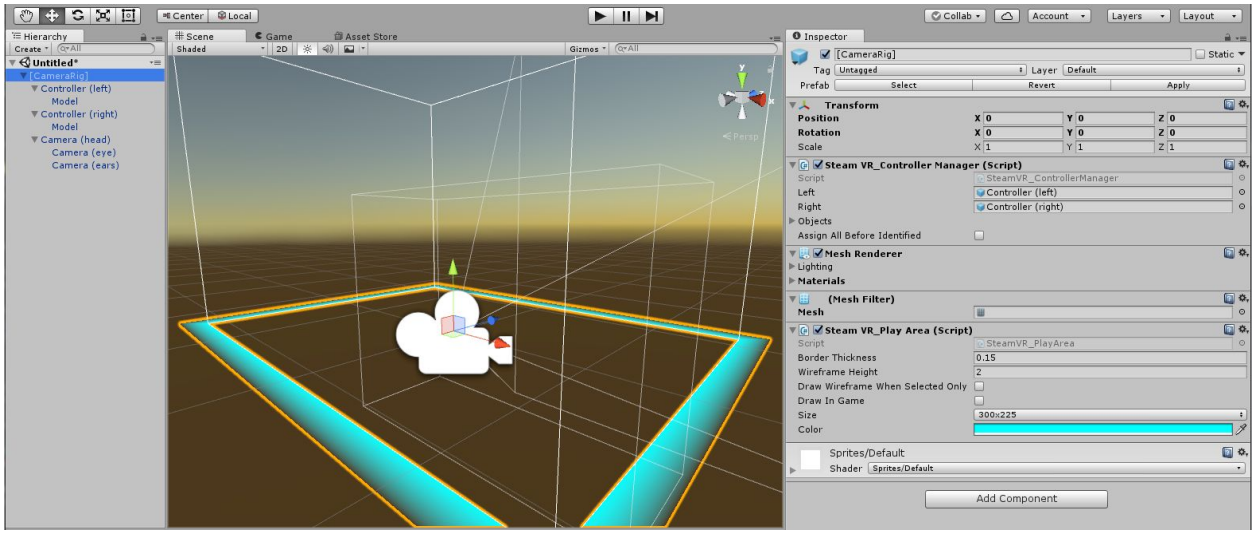
The Main Camera in this scene has also been modified from the [CameraRig] prefab included with the plugin. It has an array of 16 Tacked Devices (with the Render Models for the left and right controller included with the prefab disabled to avoid double rendering). This is useful for displaying all the tracked devices connected to the system, but is likely not what you want in production scenes. This scene is also setup to use Seated tracking (see the [SteamVR] object's Tracking Space setting) rather than Standing (i.e. Room Scale) tracking.

Because of the above issues it is recommended that you ***not*** use the example scene as a starting point for any production scenes. Instead, start from an empty scene (delete the Main Camera) and drag and drop the [CameraRig] prefab into your scene.

If you want to change any values controlled by [SteamVR], you can optionally drag and drop that into your scene as well and change them there. Otherwise, this object will be created for you automatically at runtime using the default values.

**tl;dr - Do *not* use the example scene as a basis for production levels, and in particular do not use the [Status] prefab.**

# [CameraRig] prefab




## Play Area

The [CameraRig] prefab includes a Play Area script which draws a play area representing the floor in the user's space they can walk around in. This is the root object which can be moved (and scaled) to change how the user's room setup maps to the game world. There is a drop-down to select a specific size (large, medium and small room scale spaces) as well as a Calibrated setting to match the value set by the user's room setup. This is intended for development reference and hidden at runtime, but there is a checkbox to enable it at runtime as well. It is recommended you use Calibrated in that case so it always matches each user's room setup.

## Controllers

Controllers are assigned using the Controller Manager script. Each root controller object has only a Tracked Object script. The Tracked Object script updates the transform to match the associated device. SteamVR assigns an index to each tracked device starting with 0 (zero) for the headset. Tracked devices are not just controllers, but also include the basestations (or tracking cameras), tracking pucks, etc.

### Render Models

Each controller has a single child object with a Render Model script. The Render Model script is responsible for dynamically loading the geometry and textures for the assigned tracked device. This is useful for supporting future hardware that does not exist at the time you ship your game.

The Model Override drop down allows you to select any existing device model in the Editor. This will populate the Model object with child objects representing the parts of the selected model. All children under a Render Model object will be disabled and only activated if actually attached. Therefore, you should not add any child objects yourself under a Render Model.

Each Render Model piece has an 'attach' point. This can be used to attach objects similarly across different devices. Again these will be activated only when present. Render Model pieces should have similar names where appropriate to help make handling different devices easier. For example both the Vive wand and Touch controller have a 'tip' component. This can be used for attaching a laser pointer for example.

### Camera (eye)

The Camera (eye) object is the camera used for rendering. It is shared across both eye by default. Any full screen fx will normally be added to this object.

### Camera (head)

In earlier versions of Unity (pre-5.4) the Camera (head) object was used to follow the headset position and rotation, with the Camera (eye) as a child. With native OpenVR integration (5.4 and later) the Camera (eye) is now moved to follow the headset motion, so you'll find the Camera (head) object swap places with the Camera (eye) and gets disabled. The Camera (head) object was also responsible for rendering the companion window (the VR preview that shows up on the desktop) in older versions (view the Game View script), while newer versions handle this automatically using the Camera (eye). This can be overridden by adding a separate camera to your scene and setting its TargetEye to None (Main Display). Just keep in mind that this will be more expensive as it will be re-rendering the scene rather than simply reusing the texture from one of the eyes.