HTC Vive SRWorks Portal Effects Guidelines for Unity Developers

To guide you in developing VR experiences, this document describes the settings of the portal effects in HTC Vive SRWorks plugin, as well as its asset contents.

About portal effects

The portal effects in the HTC Vive SRWorks plugin are a collection of rendering techniques. Prefabs, scripts, and shaders are provided in the plugin, and serve as fundamental elements to apply in the VR experiences you're developing.

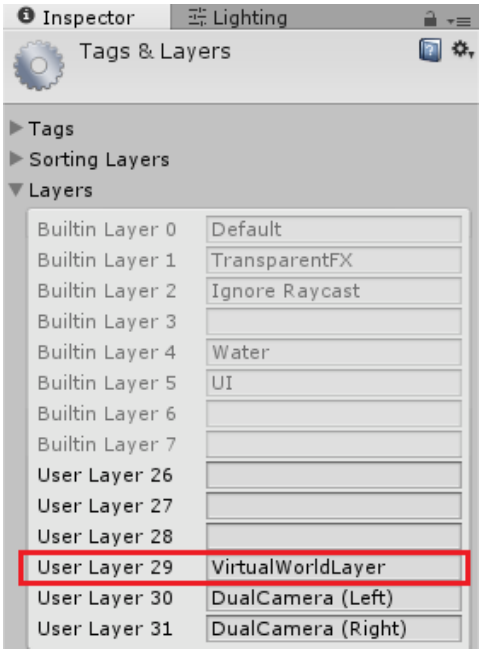
You may need to read, modify or add new scripts or shaders if you want to use portal effects in a more complete VR experience. For example, the shaders included in the plugin may not meet the desired shading effect for the VR experience, so as a developer, you may prepare your own customized shaders. In this case, it's helpful to note that shaders should follow the stencil test rule, like the ones provided in the plugin.

Preparation settings

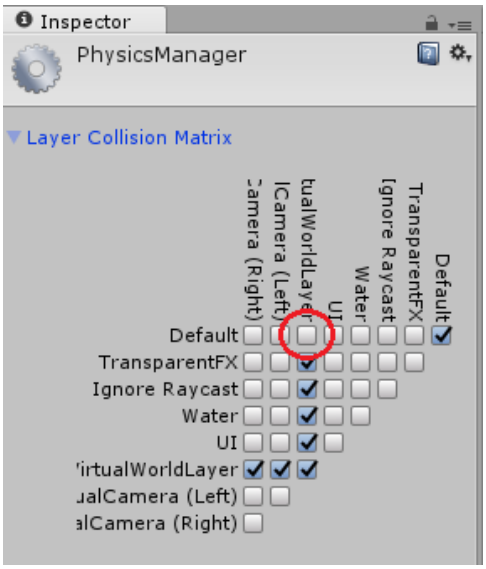
The settings described below are implemented by default on the SRWorks plugin.

- New layer added for the portal world
  Since the portal world is separated from the original world (see-through/real world), it needs to be rendered with another camera targeting at a different layer. For this, a new layer is added on the layer manager.

  Note: If you want to render more than two worlds at the same time, you may need to add another layer (for example, two portal worlds + real world).



- Collision matrix
  Portal effects isolate collision detection from each world. In the collision mask, remember not to check the "Default-VirtualWorldLayer" option.

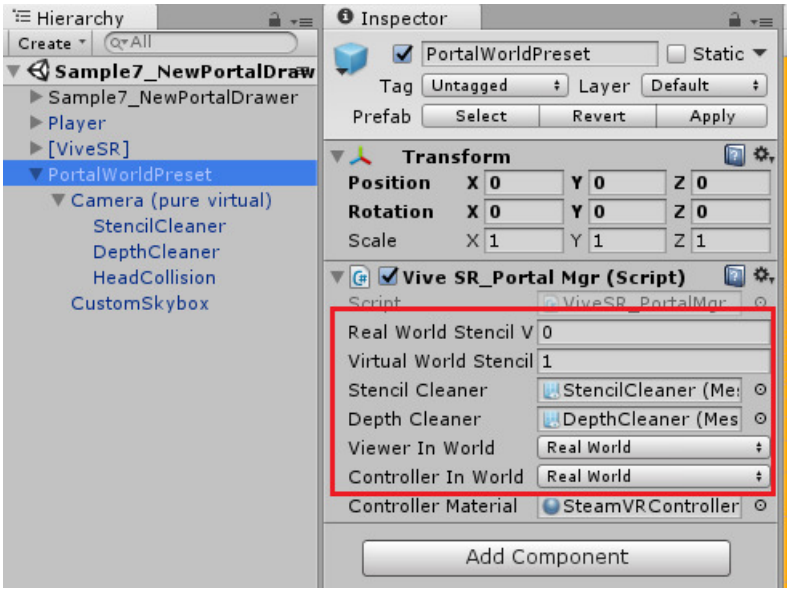


Asset contents

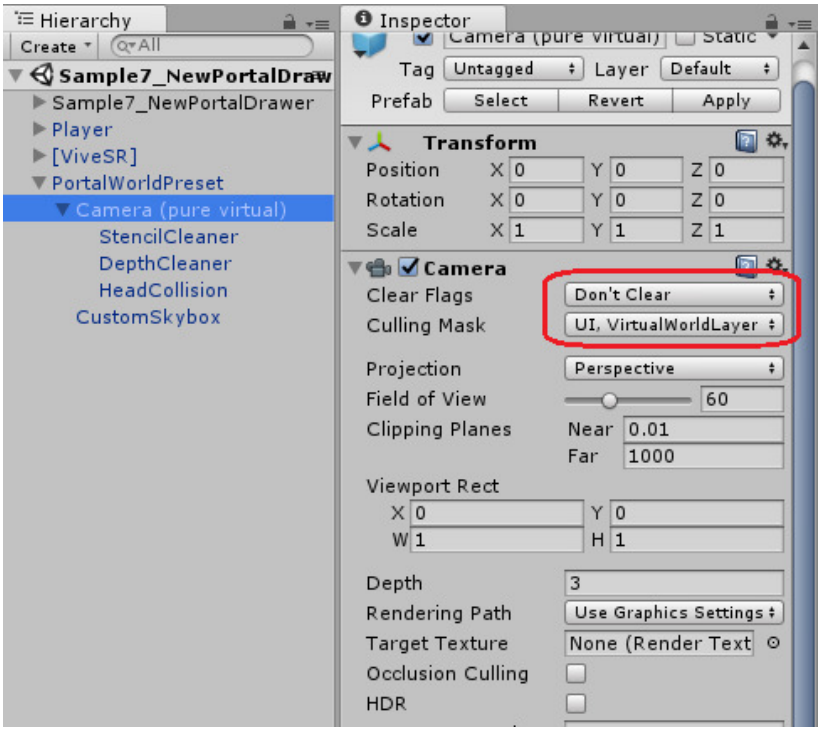
Asset contents included in the SRWorks plugin and how to work with their settings are described below.

PortalWorldPreset

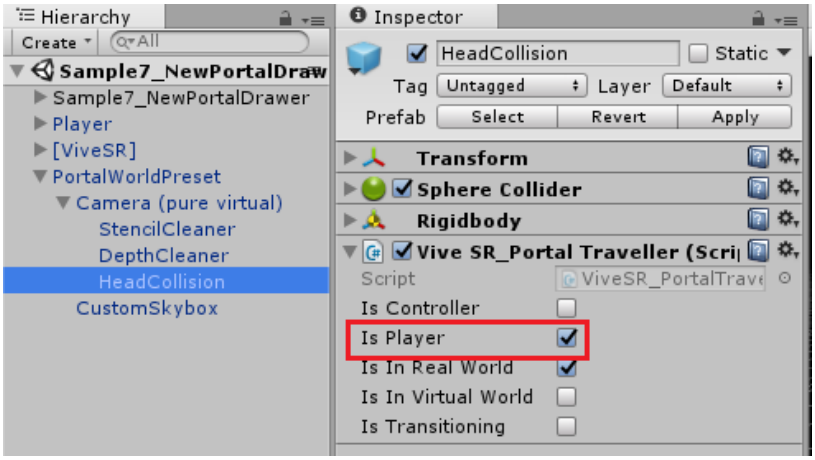
- **Root Script: ViveSR_PortalMgr**. This manager script controls the rendering pipeline as well as set the materials' properties and the render queue of the portal objects (the portal gates).
  1. Except for "Controller Material", you don't need to change any setting in the prefab, which just show the information and the current state.
  2. Optionally, set **Controller Material** if you are using SteamVR's interaction system to draw the controller. It will change the controller's material to a stencil test shader. If you are not using it, just leave it blank.



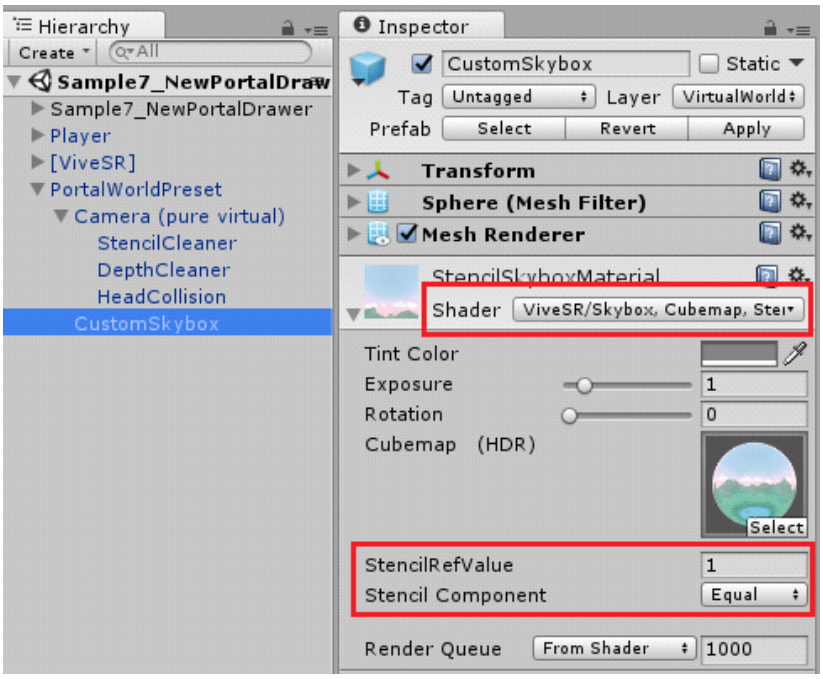
- **Camera (pure virtual)**: This is a camera rendering for the virtual world.
  1. Set the **CullingMask** to "VirtualWorldLayer". For UI or other purposes, the setting should depend on your own game design.
  2. Set **ClearFlags** to "Don't Clear", since Unity will always clear the stencil buffer in any other setting. This is not a desired situation, so StencilCleaner and DepthCleaner are attached, which only clear stencil and depth and are controlled in script.



- **HeadCollision**: This is a collision sphere representing the player.
  1. Any GameObject that is allowed to travel between different worlds through portals should add the **ViveSR_PortalTraveller** script. This script checks which world is the current location, and then changes all the renderers' layers and material properties.
  2. To make the script work, you need a trigger collider.
  3. The last three checkboxes indicate important information. Do not change these settings through the Inspector.
  4. Only the first two checkboxes are set from the Inspector by developers. If the traveler represents the player (like in this HeadCollision example), then "**IsPlayer**" should be checked. This is an important element that tests whether the player is viewing from the real or virtual world.
  5. For an example of when to use the **Is Controller** flag, refer to *SteamVR controller (optional)* later in the document.



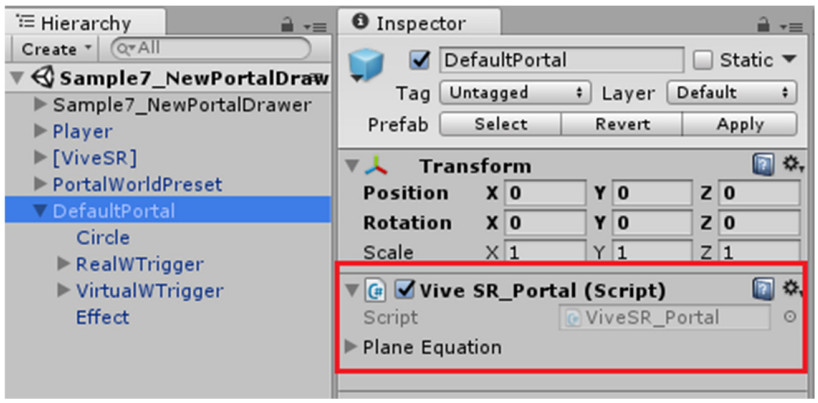
- **CustomSkybox**: This is a skybox box renderer. This mesh renderer replaces the built-in skybox rendering pipeline.
  There are two reasons why the built-in skybox rendering pipeline cannot be used:
  1. The skybox should enable the stencil test since it is inside the portal. Developers are provided with a skybox shader that supports the stencil test. This way, the mesh will be rendered only on the stencil value = virtual world stencil.
  2. ClearFlags shouldn't be set to DepthOnly, SolidColor, or Skybox since it will clear the stencil buffer as well. A mesh renderer is put in place to replace the built-in skybox.



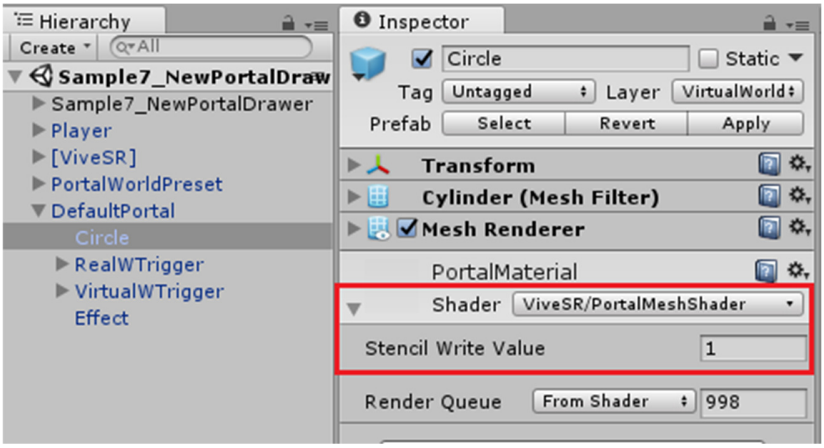
Default Portal Prefab

This default prefab is a sample portal GameObject. Developers can also design their own portal prefab by following these rules:

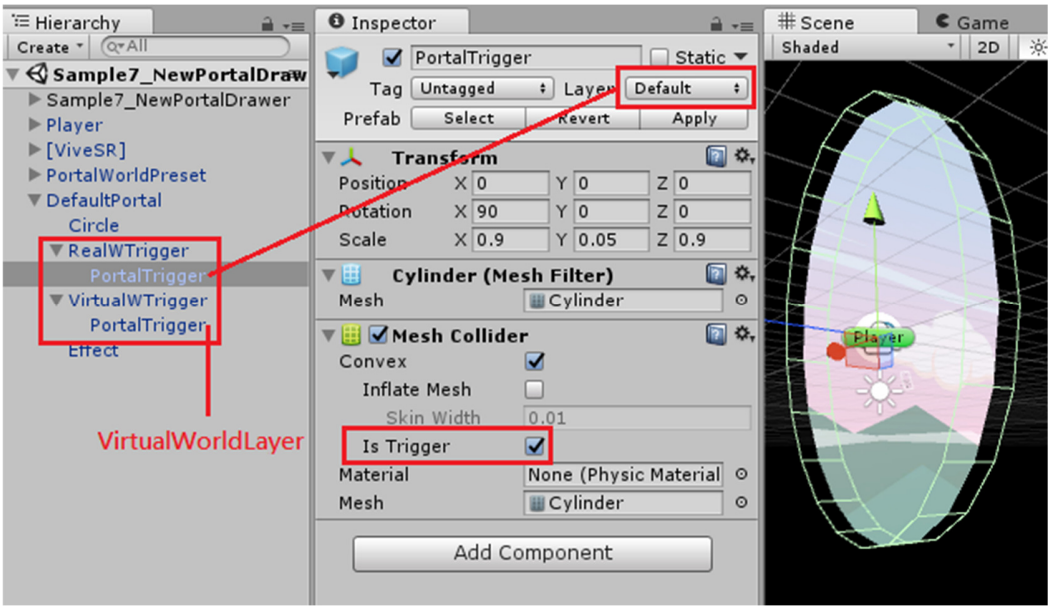
- **Root Script**: attach the **ViveSR_Portal** component to collect the portal mesh renderers and particle system renderers. When the player (viewer) is in a different world, this script will set the rendering rule of the portal mesh automatically.



- **Circle**: This is a basic mesh renderer. Portal mesh (portal gate) should use the “PortalMeshShader”, which is a stencil-writer. The written value should be equal to the inside-world stencil value.



- **Triggers**: These are a range of transitions. A portal traveler is indicated as “Is Transitioning” when it overlaps with this trigger. The transitioning period is an important period for changing the rendering and collision rule.
  1. The trigger box should not to be too small. Make sure that **Is Trigger** is selected.
  2. You need to have two identical trigger boxes, except for the layers (one is default, and the other is VirtualWorldLayer.) Since the traveler is going between different worlds, “Default-VirtualWorldLayer” should be unchecked in the collision matrix.



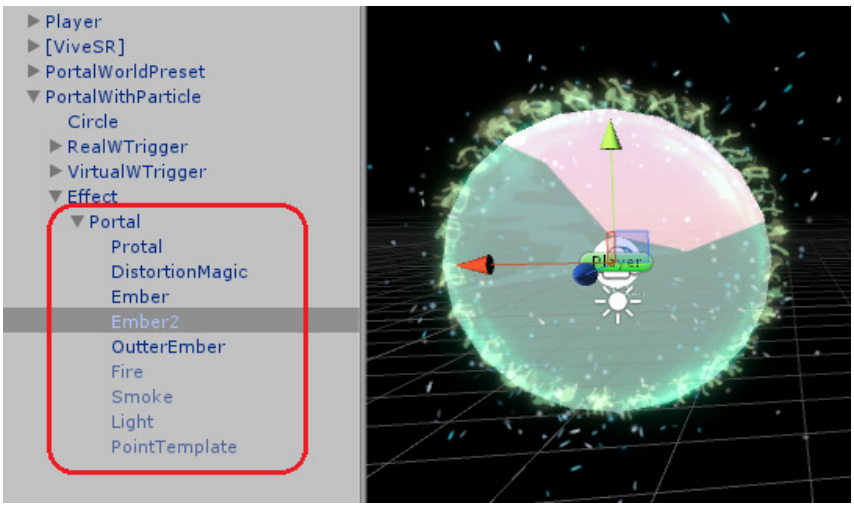
- **Effect**: A reserved group for developers to put the additional effect (particles or others).

Settings for sample assets

This section describes how you can use the provided scripts and prefabs in your own VR content. You can run the "Sample7_Portal.unity" to see the demo results.

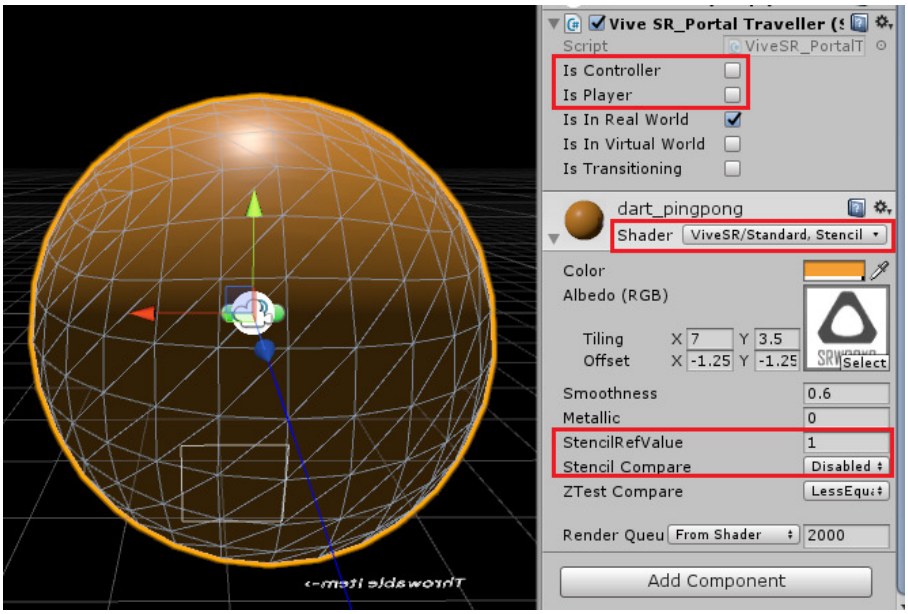
Portal with applied effects

An extension prefab from "Default Portal Prefab", some sample particles are attached around the portal gate to make it look more vivid.



Dart traveler

In the ViveSR_Experience portal demo, you are able to throw darts, spheres, or other objects through portals. To do this, the "Portal Traveller" script should be added to it. In addition, change the shader to "Standard, Stencil", which supports the stencil test.



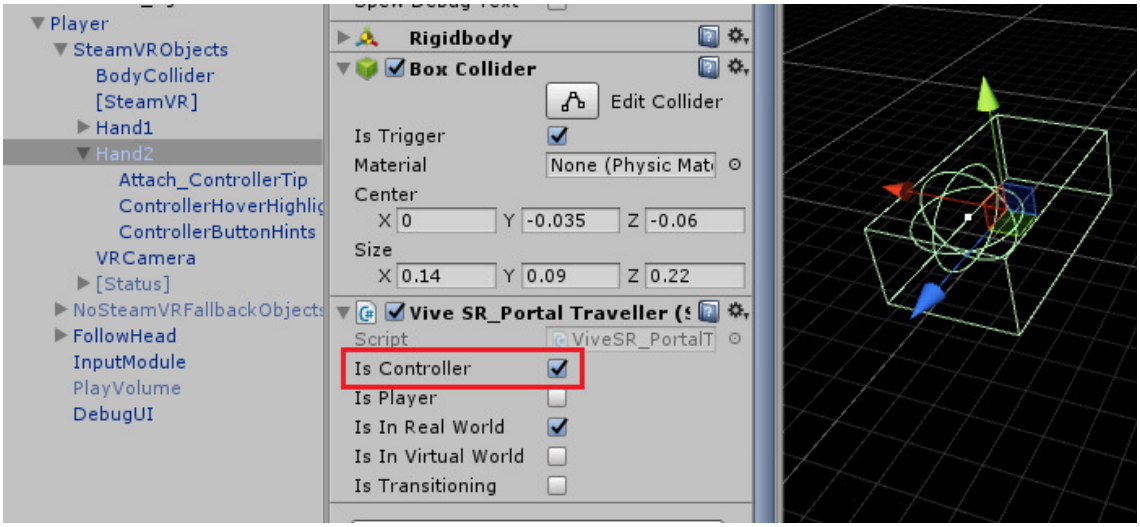
SteamVR controller (optional)

In PortalWorldPreset, the “HeadCollision” GameObject is introduced and is noted to be set as “Is Player”. Another flag, “Is Controller”, is explained below, which you can opt to use.

- **Is Controller**: Set this optional prefab if you are using SteamVR’s interaction system.
  If you choose not to use this prefab, you can follow the next steps to modify your own controller prefab. In many content demos, players would spawn new GameObjects in a game, and they would also be travelers at the same time (such as in the throwable dart example). Thus, a GameObject that represents the controller is needed to check the initial world of the spawned traveler.

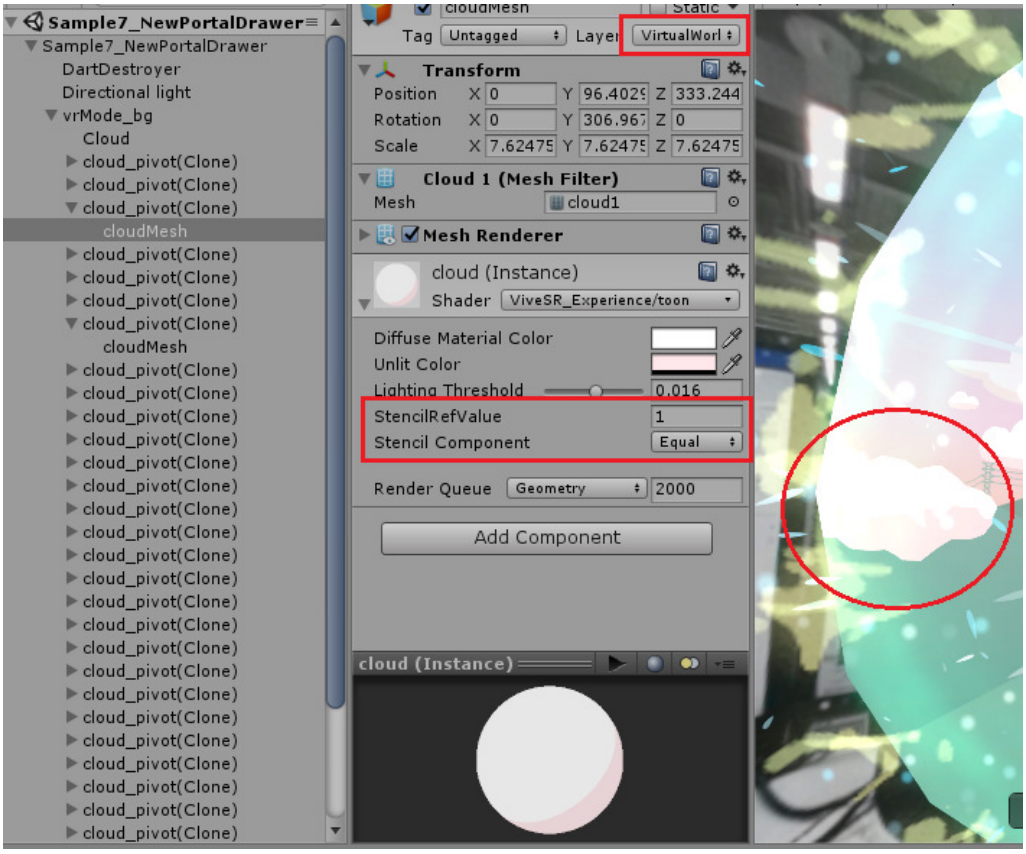
  For example, if the controller is in real-world, then the dart would be set to “Is In Real World” when it’s spawned, and vice versa.

- **Box Collider**: Since a traveller script was added to it, remember to give it a fitting trigger collider as well.



Virtual world object settings

Make sure that all the objects in the virtual world are put in the “VirtualWorldLayer” and that their shaders should support the stencil test, such as the cloudMesh and its toon shader (as shown below).



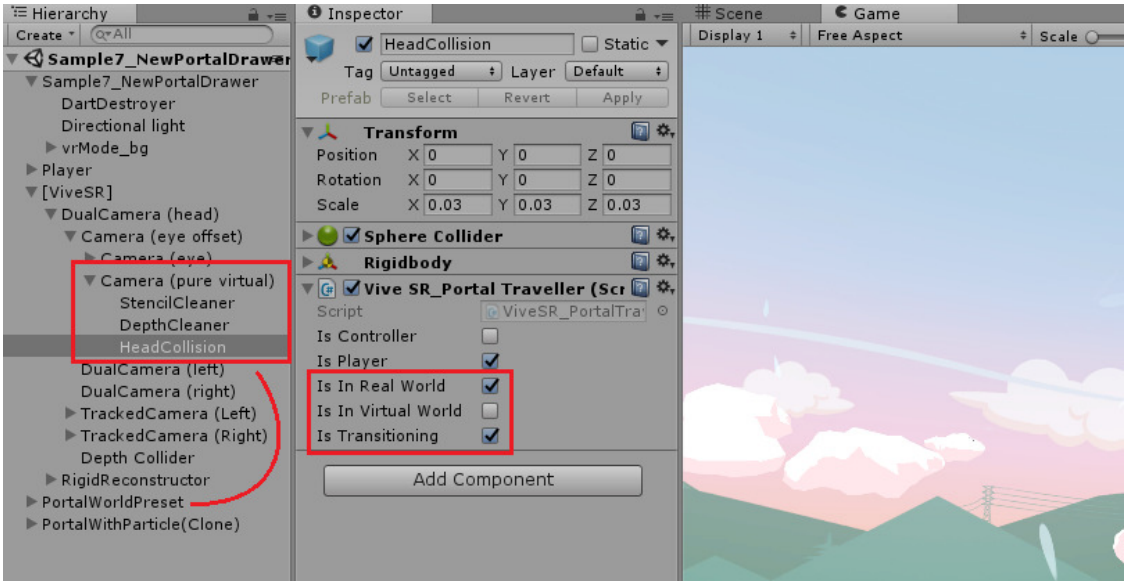
Run-time check

During runtime, the simplest way to check if the plugin works correctly is to review the properties of "ViveSR_PortalTraveller": in real-world, in virtual-world, and is transitioning. Another setting to check is the properties on "ViveSR_PortalMgr".

Traveller World State

Shown as an example below is the HeadCollision traveller. When playing, the GameObjects under PortalWorldPreset will be moved to [ViveSR] prefab.

- **Contact with portal**: Developers can try to move the HMD toward the portal, and then pass through the portal to the virtual world completely. During the process, observe the world state in the traveller inspector.



Information in ViveSR_PortalMgr

Do the same test by moving the HMD around (with player or viewer). Try to put the HMD in different sides of the portal, and check how the **Viewer in World** state under ViveSR_PortalMgr toggles between “Real World” and “VR World”.

You can also do the same test for the controller, and review the **Controller in World** state.