Vive Stereo Rendering Toolkit – Developer's Guide

vivesoftware@htc.com

Introduction

Vive Stereo Rendering Toolkit provides drag-and-drop components for developers to create stereoscopic rendering effects in a few minutes.

With this toolkit, effects such as mirrors or portal doors can be easily achieved in your VR application.

System Requirement

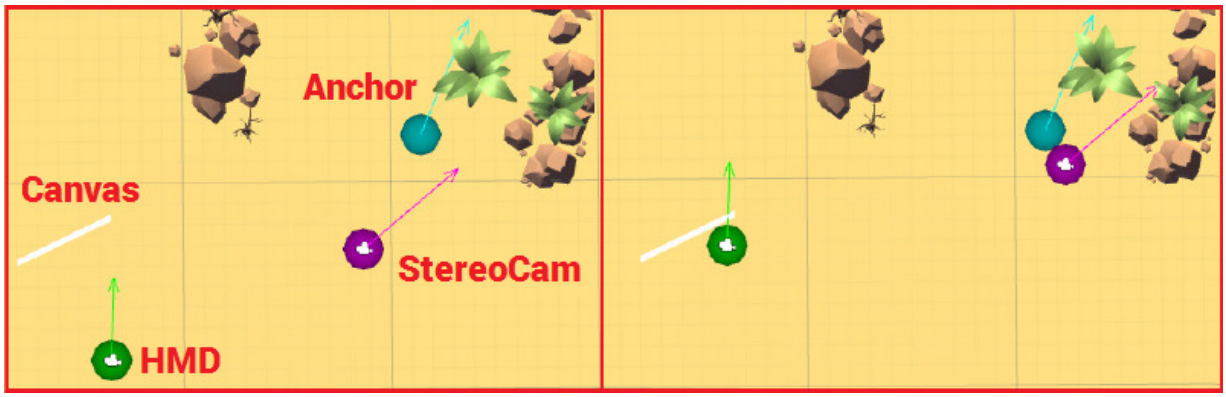
- Unity 5.3.5 or higher
  Compatible with Native VR rendering and Single-pass Stereo introduced in Unity 5.4.0.
- SteamVR Unity plugin, **version 1.2.0** or higher

Terminologies

A camera for performing stereo rendering is called a **stereo camera**. Its rendering result is applied to a **canvas**, which can be any planar GameObject with a Renderer.

Stereoscopic effect, such as parallax, is achieved by synchronizing the motion of HMD (both head and eyes) to the stereo camera object. This is done by assigning a **canvas-anchor** pair to a stereo camera.

The **anchor**'s pose defines the pose of associated stereo camera when the HMD overlaps with the canvas. Using this definition, our code retargets the relative motion between HMD and canvas to the motion between stereo camera and anchor (see the figures below).



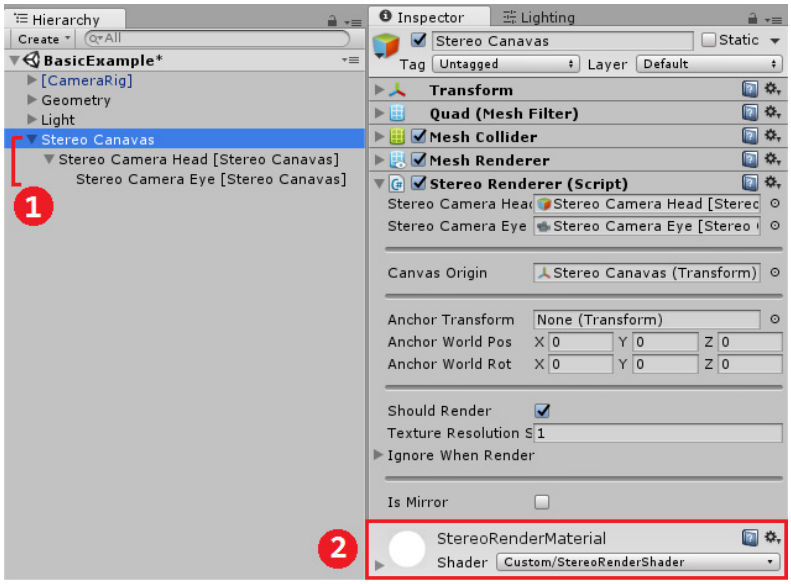
Quick Start Guide

Step 1

- Follow SteamVR Unity plugin quick start guide to setup a basic VR application

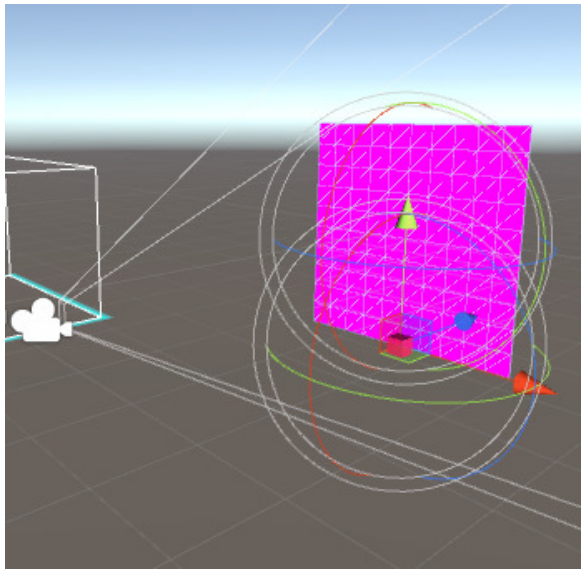
Step 2

- Add script **StereoRenderer** to your **Stereo Canvas**.
  The canvas should be a **planer object** with a Renderer.
- Once added, a Stereo Camera will be automatically added to your hierarchy. (1)
- If your renderer uses only 1 material, it will also be automatically replaced by our **StereoRenderMaterial**. (2)
- If your renderer uses multiple materials, manually change one of them to our **StereoRenderMaterial**.



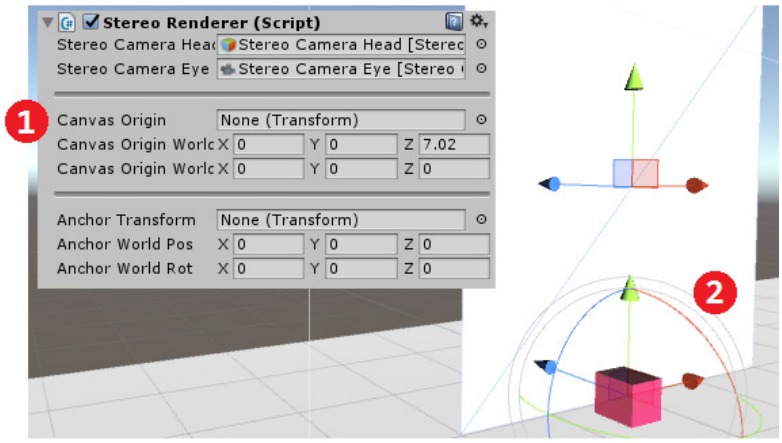
Step 2-1

- If you are using **Unity 5.3.x**, the canvas object will become **pink** after step 2. This is because the default stereo shader we use is written for Unity 5.4.x or newer.
- The stereo shader will be automatically swapped to 5.3.x version during execution.
- Or, you can change the shader of StereoRenderMaterial to Custom/StereoRenderShader_5_3, which also solves the version conflict problem.



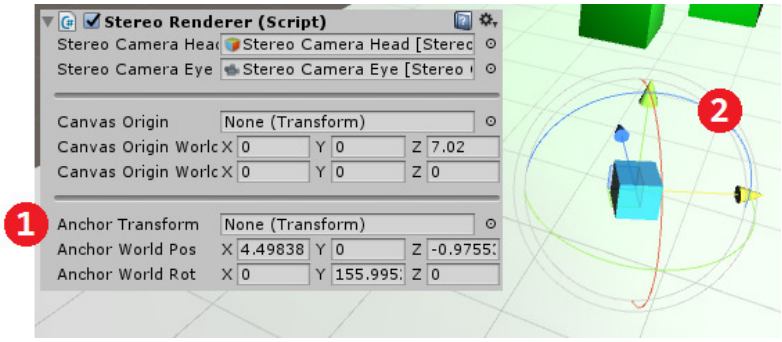
Step 3

- Set the **Canvas Origin** property if your canvas does not lie at the origin of local coordinate system.
- To change its pose, you can either assign a transform to it (1), or use the handles attached to the pink cube in scene (2).



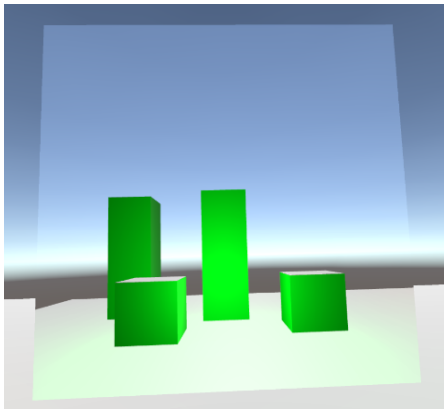
Step 4

- Set the pose of **Anchor**.
  See the "terminologies" section if you don't know its meaning.
- To change its pose, you can either assign a transform to it (1), or use the handles attached to the blue cube in scene (2).



Done!

- Click play, and you should see the result in HMD!
- You can find this tutorial scene in Assets/HTC.UnityPlugin/StereoRendering/Examples/1. BasicExample



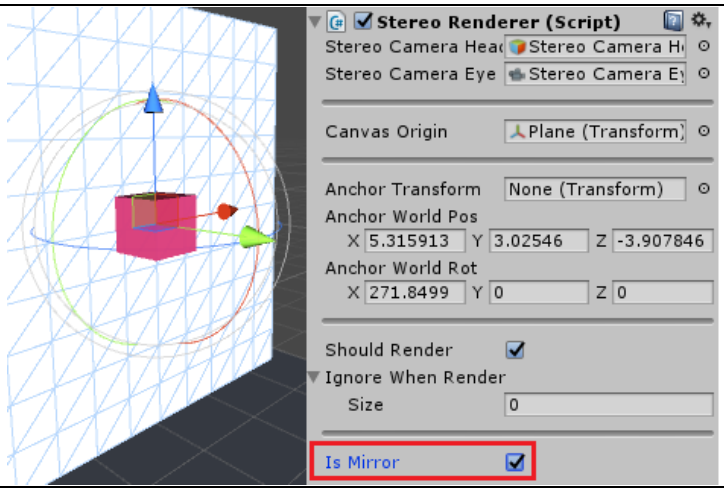
Tutorial - Mirror

Step 1-3

- Follow Steps 1-3 of previous quick start guide.

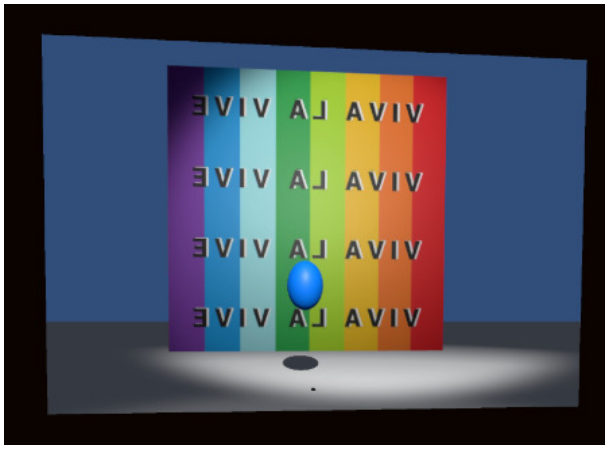
Step 4

- Check the **IsMirror** option of StereoRenderer.
- Rotate the **Canvas Origin** so that its Y axis (green arrow) aligned with the mirror's normal vector (the vector that is perpendicular to mirror plane).



Done!

- Click play, and you should see the result in HMD!
- You can find this tutorial scene in Assets/HTC.UnityPlugin/StereoRendering/Examples/2. Mirror



Tutorial – Double-Sided StereoRenderer

By default, meshes in Unity are rendered with back-face culling enabled. This means, the polygons face away from viewer are not rendered. However, sometimes we may also want to render back faces (for example, some kind of portal doors).

To render both sides of StereoRenderers, **uncomment** line 22 of StereoRenderShader.shader (or line 20 of StereoRenderShader_5_3.shader). The image below shows the effect of this modification.

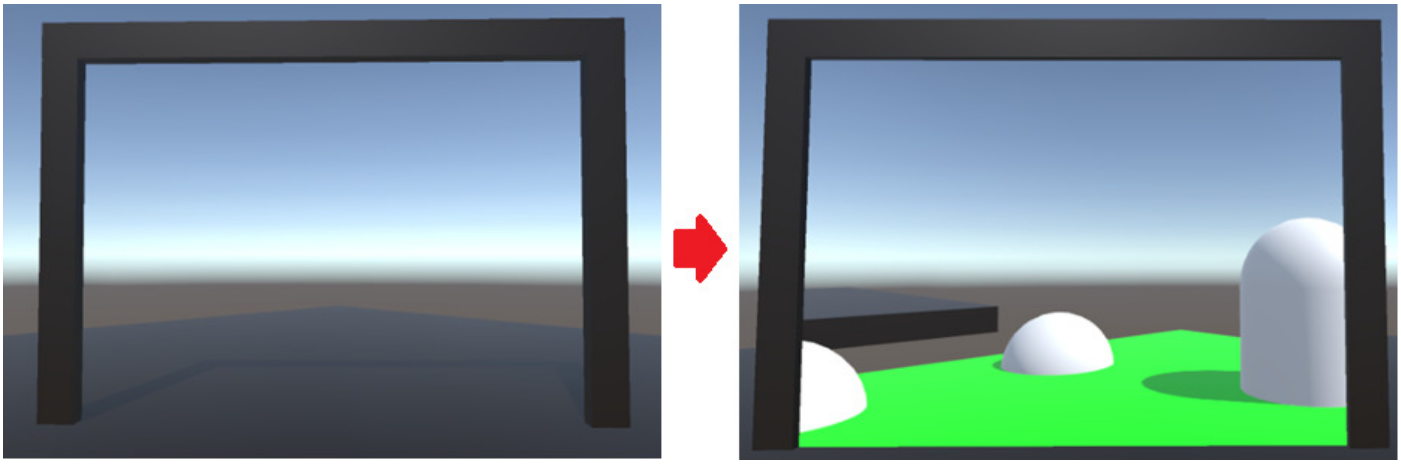


Fig. Viewing a portal door from its back. In the left image, the mesh is culled so nothing is rendered. After back-face culling is disabled, the portal door is rendered (right image).

Note that, disabling back-face culling may have an impact on the performance of your application, depends on the complexity of your target mesh.

Tutorial – Callbacks

We provide callbacks for developers to manipulate objects in the scene at two specific moments – (1) before the first StereoRenderer start rendering, and (2) after the last StereoRenderer finished rendering. Some special effects can be achieved with these callbacks.

In this example, we will use the callbacks to give a green tint to the reflection in mirror.
You can find finished scene at Assets/HTC.UnityPlugin/StereoRendering/Examples/4.Callbacks

Step 1

- Setup a mirror scene using the steps introduced in previous turorial.

Step 2

- Add an empty C# script "IlluminationController.cs" to the scene.
- Add a public variable "Light light" to the script, and drag the light whose color we want to dynamically change to this field in the inspector.

Step 3

- Add 2 functions "OnBeforeRender()" and "OnAfterRender()" to this script.
- Register these functions as callbacks to specific events in Awake()

```
void Awake()
{
    StereoRenderManager.Instance.AddPreRenderListener(OnBeforeRender);
    StereoRenderManager.Instance.AddPostRenderListener(OnAfterRender);
}

void OnBeforeRender()
{
}

void OnAfterRender()
{
}
```

Step 4

- Make sure to unregister the callbacks when the game object is destroyed (scene unloaded)

```
void OnDestroy()
{
    StereoRenderManager.Instance.RemovePreRenderListener(OnBeforeRender);
    StereoRenderManager.Instance.RemovePostRenderListener(OnAfterRender);
}

void OnBeforeRender()
{
}

void OnAfterRender()
{
}
```

Step 4

- Change the light's color in callback functions.

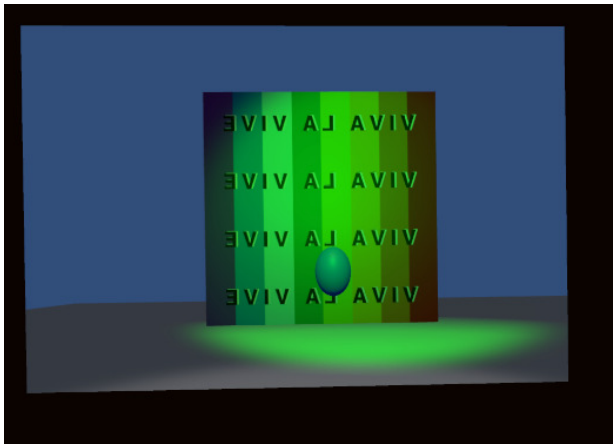
```
private Color normalColor = new Color(1.0f, 1.0f, 1.0f);
private Color scaryColor = new Color(0.0f, 1.0f, 0.0f);

void OnBeforeRender()
{
    light.color = scaryColor;
}

void OnAfterRender()
{
    light.color = normalColor;
}
```

Step 5

- Click play and you can see the special effect achieved!



Scripting Reference

- **StereoRenderManager**

  This script should be attached to the HMD camera object.
  If not attached manually, we will attempt to attach the script at run time.

  The manager keeps reference to all StereoRenderers in current scene, and evokes rendering functions of these renderers each frame (just before the rendering of HMD camera starts).

  - `static StereoRenderManager StereoRenderManager.Instance`

    Get instance of this singleton.

  - `void StereoRenderManager.AddToManager(StereoRenderer stereoRenderer)`

    Register the stereoRenderer to manager so it will be evoked.

  - `void StereoRenderManager.RemoveFromManager(StereoRenderer stereoRenderer)`

    Unregister the stereoRenderer from manager.

  - `void StereoRenderManager.AddPreRenderListener(System.Action Listener)`

    Add a callback function that will be evoked just before the first SteroRenderer starts rendering.

  - `void StereoRenderManager.AddPostRenderListener(System.Action Listener)`

    Add a callback function that will be evoked after the last SteroRenderer finished rendering.

  - `void StereoRenderManager.RemovePreRenderListener(System.Action Listener)`

    Remove specified callback.

  - `void StereoRenderManager.RemovePostRenderListener(System.Action Listener)`

    Remove specified callback.

- **StereoRenderer**

  This script should be attached to the canvas, which is the object that displays stereo rendering result. The canvas should be a planar object with a single Renderer. The stereo camera and anchor associated with the canvas are also accessible via this class.

  Since the canvas mesh is not necessarily at the origin of its local coordinates system, you should access the Canvas' pose via CanvasOrigin instead of Canvas.transform.

  - `bool shouldRender`

    If set to false, StereoRenderManager will not invoke rendering of this object.

  - `Vector3 canvasOriginPos`

    The position of CanvaOrigin in world coordinate system.

  - `Vector3 canvasOriginEuler`

    The orientation of CanvaOrigin in world coordinate system, in the form of Euler angles.

  - `Quaternion canvasOriginRot`

    The orientation of CanvaOrigin in world coordinate system, in the form of quaternion.

- **`Vector3 canvasOriginForward`**

  The forward vector from the orientation of CanvasOrigin in world coordinate system. (Read Only)

- **`Vector3 canvasOriginUp`**

  The up vector from the orientation of CanvasOrigin in world coordinate system. (Read Only)

- **`Vector3 canvasOriginRight`**

  The right vector from the orientation of CanvasOrigin in world coordinate system. (Read Only)

- **`Vector3 localCanvasOriginPos`**

  The position of CanvaOrigin in Canvas' local coordinate system.

- **`Vector3 localCanvasOriginEuler`**

  The orientation of CanvaOrigin in Canvas' local coordinate system, in the form of Euler angles.

- **`Quaternion localCanvasOriginRot`**

  The orientation of CanvaOrigin in Canvas' local coordinate system, in the form of quaternion.

- **`Vector3 anchorPos`**

  The position of Anchor in world coordinate system.

- **`Vector3 anchorEuler`**

  The orientation of Anchor in world coordinate system, in the form of Euler angles.

- **`Quaternion anchorRot`**

  The orientation of Anchor in world coordinate system, in the form of quaternion.

- **`GameObject stereoCameraHead`**

  The camera rig object used to move StereoCamera.

- **`Camera stereoCameraEye`**

  The camera used to perform stereo rendering.

- **`void StereoRenderer.AddPreRenderListener(System.Action Listener)`**

  Add a callback function that will be evoked just before the this SteroRenderer starts rendering.

- **`void StereoRenderer.AddPostRenderListener(System.Action Listener)`**

  Add a callback function that will be evoked after this SteroRenderer finished rendering.

- **`void StereoRenderer.RemovePreRenderListener(System.Action Listener)`**

  Remove specified callback.

- **`void StereoRenderer.RemovePostRenderListener(System.Action Listener)`**

  Remove specified callback.