Spring 2021

CSE 190

VR Technologies

Discussion 7

Guowei Yang
UCSDCSE

ANNOUNCEMENTS

- Homework 3 Deadline Extended for A WEEK
  - Due **Sunday (5/23)**
  - Come to Office Hours
- Homework 4 Released
  - Due **Sunday (5/30)**
  - START EARLY

UCSDCSE

AGENDA

- Homework 4 Overview
- Homework 4 Getting Started
- Homework 3 Q&A

UCSDCSE

Homework 4 Overview

Homework 4 Overview

- Simulate the CAVE (Cave Automatic Virtual Environment) in VR (A virtual environment within a virtual environment...?)
- Simulate the tracked user
- Simulate an observer

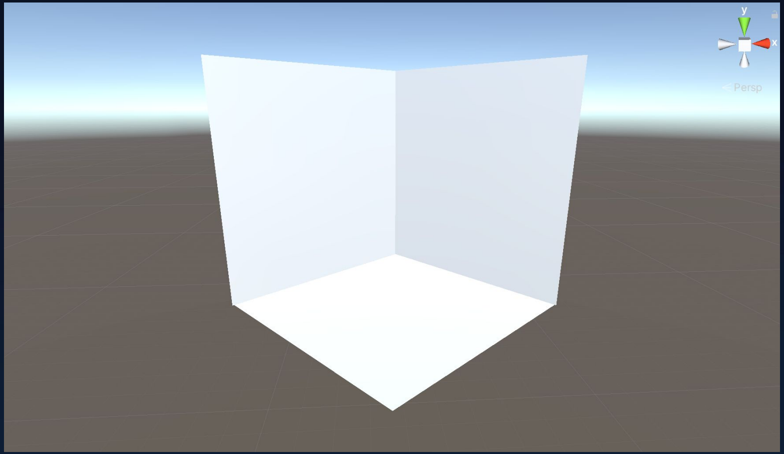
Homework 4 Overview - View Point Switch

- HMD View & Controller View
  - Able to switch from HMD view to Controller View
    - HMD view: view point same as headset
    - Controller view: view point same as right controller
- Freeze View
  - Freeze the view point that is used to render to CAVE wall
- Debug View

Homework 4
Getting Started

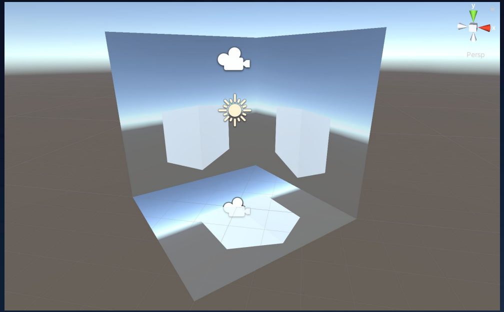
Homework 4 Getting Started: Create CAVE

- Create CAVE Room (Three 3D Planes)
    - Can use GameObject -> Plane

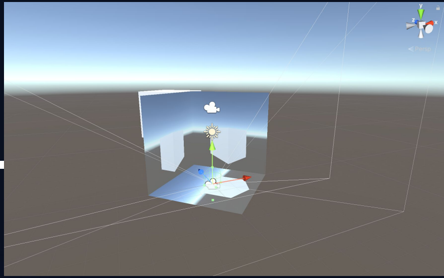


Render to CAVE Walls

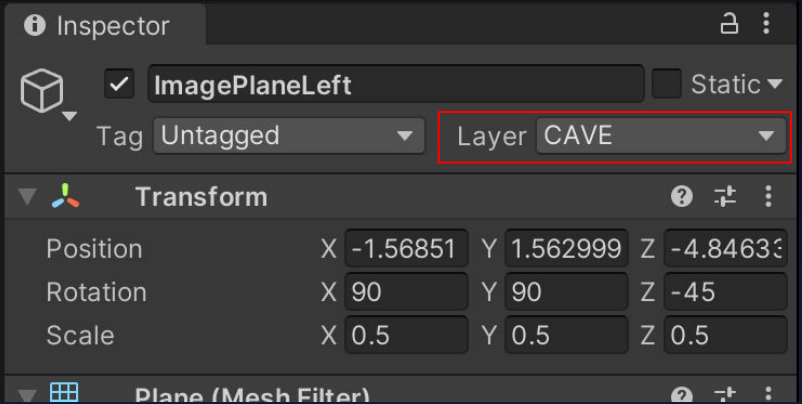
- You need two pairs of cameras (each pair for left/right eyes)
    - One pair looks at the CAVE wall (OVRCameraRig)
    - The other pair serves as the view point to the virtual scene (Custom Camera Rig)
    - CAVE walls should be **invisible** to the **custom rig**
- Render the image from the view point (seen by the custom rig) to the CAVE walls use off-screen rendering
- Needs a custom material and a custom shader for the walls in order to display off-screen rendering images

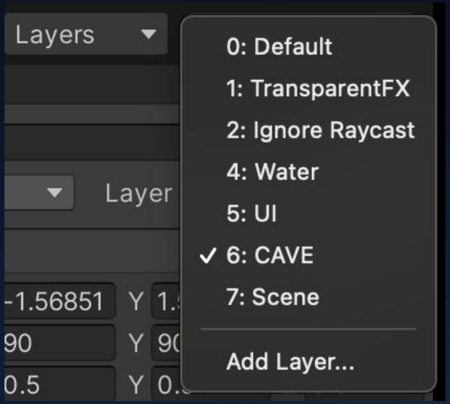


Render to CAVE Walls: Invisibility



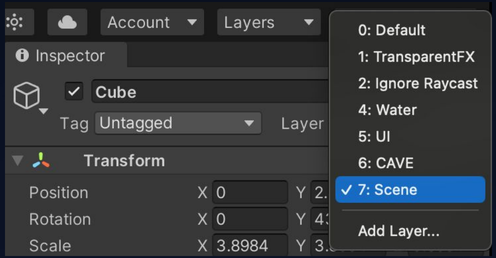
- Don't want the view point camera see the CAVE walls
- Set the Render Layer of the CAVE walls

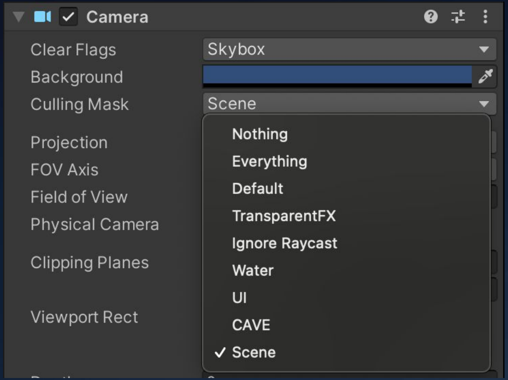




Render to CAVE Walls: Invisibility

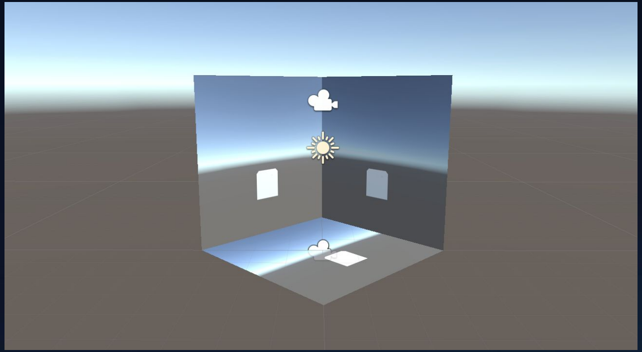
- Set the render layer for the virtual scene objects
- Set the Culling Mask for both pair of cameras
  - The CAVE-looking cameras should see all CAVE walls, but should not be able to see scene objects (OVRCameraRig -> Culling Mask -> Exclude Scene)
  - The scene-looking cameras should NOT see any CAVE walls, but can see all scene objects (CustomCameraRig -> Culling Mask -> Only Include Scene)





Render to CAVE Walls: Off Screen Rendering

- Need to render the view from the custom camera rig to the CAVE walls
- Need to create material with custom shader
- Modify the material shader from Homework 3 (`material.shader`)



Render to CAVE Walls: Off Screen Rendering

```
Properties
{
    _Color ("Color", Color) = (1,1,1,1)
    _MainTex ("Albedo (RGB)", 2D) = "white" {}
    _Glossiness ("Smoothness", Range(0,1)) = 0.5
    _Metallic ("Metallic", Range(0,1)) = 0.0
}
```

```
Properties
{
    _Color ("Color", Color) = (1,1,1,1)
    _MainTexLeft ("Left Texture", 2D) = "white" {}
    _MainTexRight ("Right Texture", 2D) = "white" {}
    _Glossiness ("Smoothness", Range(0,1)) = 0.0
    _Metallic ("Metallic", Range(0,1)) = 0.0
}
```

```
// Use shader model 3.0 target, to get nicer looking lighting
#pragma target 3.0

sampler2D _MainTex;

struct Input
{
    float2 uv_MainTex;
};
```

```
// Use shader model 3.0 target, to get nicer looking lighting
#pragma target 3.0

sampler2D _MainTexLeft;
sampler2D _MainTexRight;

struct Input
{
    float2 uv_MainTexLeft;
    float2 uv_MainTexRight;
};
```

Render to CAVE Walls: Off Screen Rendering

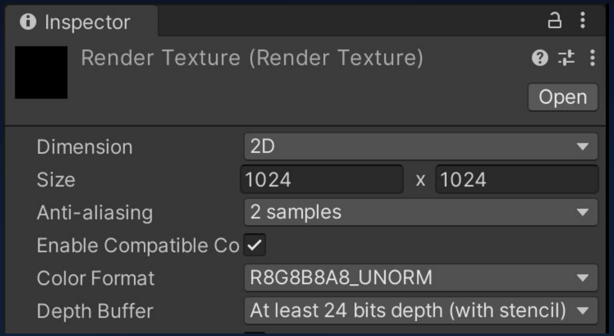
```
void surf (Input IN, inout SurfaceOutputStandard o)
{
    // Albedo comes from a texture tinted by color
    fixed4 c = tex2D (_MainTex, IN.uv_MainTex) * _Color;
    o.Albedo = c.rgb;
    // Metallic and smoothness come from slider variables
    o.Metallic = _Metallic;
    o.Smoothness = _Glossiness;
    o.Alpha = c.a;
}
```

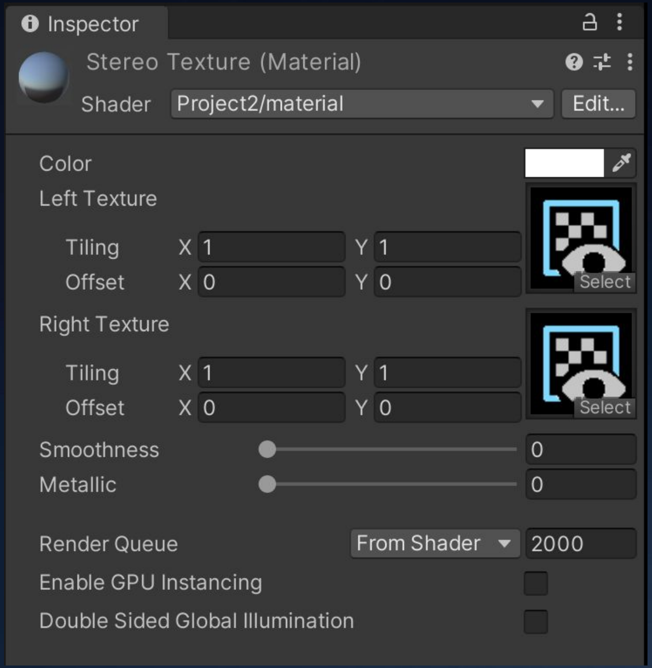
```
void surf (Input IN, inout SurfaceOutputStandard o)
{
    // Albedo comes from a texture tinted by color
    fixed4 c;
    if (unity_StereoEyeIndex == 0) { // Left
        c = tex2D(_MainTexLeft, IN.uv_MainTexLeft) * _Color;
    }
    else { // Right
        c = tex2D(_MainTexRight, IN.uv_MainTexRight) * _Color;
    }
    o.Albedo = c.rgb;
    // Metallic and smoothness come from slider variables
    o.Metallic = _Metallic;
    o.Smoothness = _Glossiness;
    o.Alpha = c.a;
}
```

- This shader ensures that different set of images are rendered to the CAVE walls, corresponding to left and right eyes
- Create a new material and attach this shader

Render to CAVE Walls: Off Screen Rendering

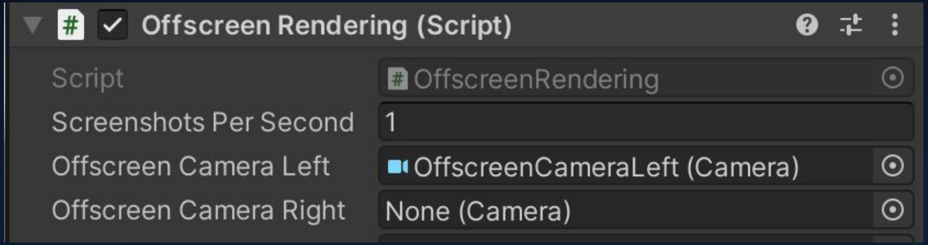
- Create two Render Textures for the material
  - Assets -> Create -> Render Texture
  - Create 2, one for left eye, one for right eye
- Select higher size for the texture (e.g. 1024x1024)
- Attach this material to all three CAVE planes





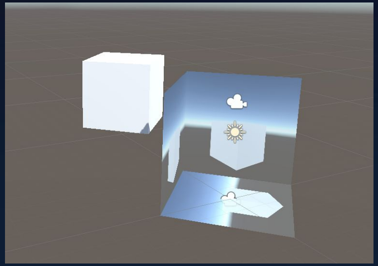
Render to CAVE Walls: Off Screen Rendering

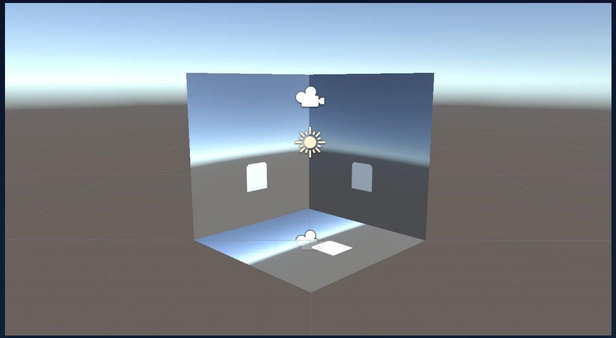
- Download the Off-Screen Rendering Script
  - https://gist.github.com/danielbierwirth/10965844fecc38243007f0cd21843d90
- Create an empty GameObject, and attach the script to it
- Set correct camera correspondences (remember to select the cameras from the custom camera rig, which looks to the scene)

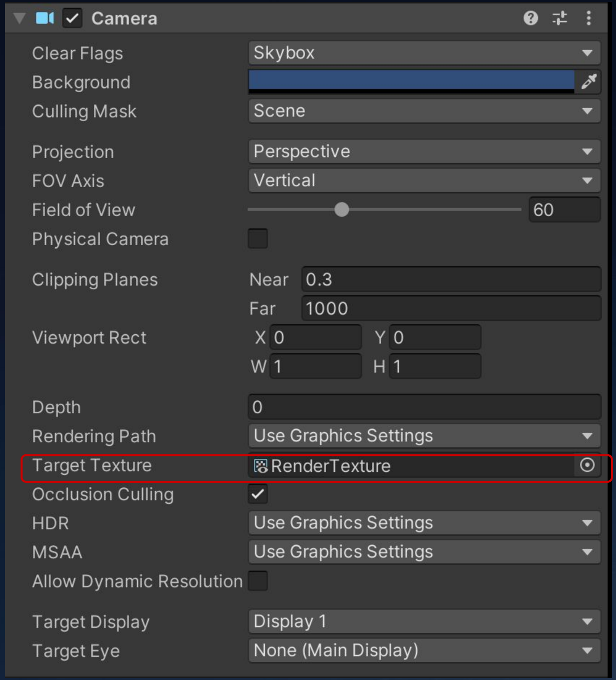


Render to CAVE Walls: Off Screen Rendering

- Set the scene-looking cameras' target texture to the newly created textures
- Now you should be able to see something like this:







Render to CAVE Walls: Off Screen Rendering

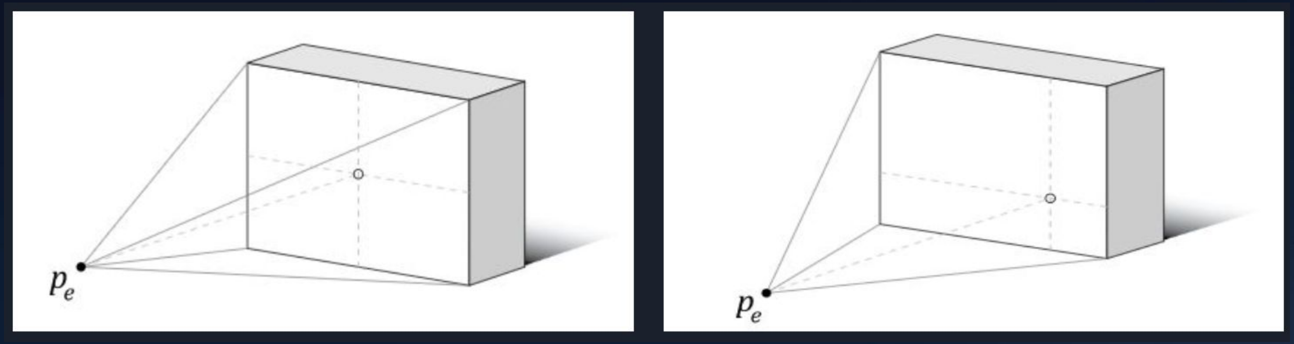
```
RenderTexture currentRT = RenderTexture.active;
RenderTexture.active = texture;
camera.targetTexture = texture;
Matrix4x4 origP = camera.projectionMatrix;
camera.projectionMatrix = P;
camera.Render();
// Read offscreen texture
Texture2D offscreenTexture = new Texture2D(
    texture.width,
    texture.height,
    TextureFormat.RGB24,
    false
);

offscreenTexture.ReadPixels(new Rect(
    0,
    0,
    texture.width,
    texture.height
), 0, 0, false);

offscreenTexture.Apply();
RenderTexture.active = currentRT;
```

Render to CAVE Walls: Projections

- Reminder a typical projective matrix assumes we are right in front of the screen
- We need to be able to render **off-center**



Render to CAVE Walls: Projections

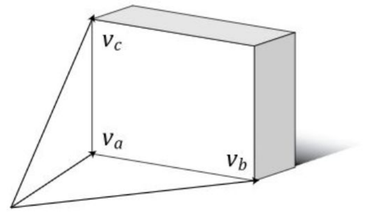
- Review of the projection matrices $$P' = PM^T T$$

Render to CAVE Walls: Projections - P

$$P' = PM^T T$$

1. Calculate vectors from eye position to the screen corners
   - `plane.GetComponent<Renderer>().bounds.max;`
   - `plane.GetComponent<Renderer>().bounds.min;`
2. Calculate distance from eye position to screen space origin

1.



$v_a = p_a - p_e$ $\quad v_b = p_b - p_e$ $\quad v_c = p_c - p_e$

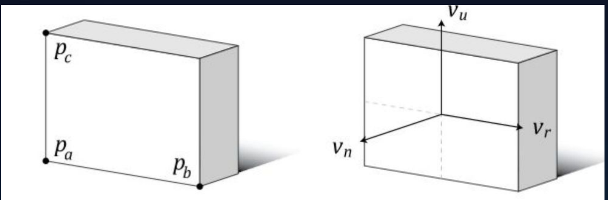
2. $d = -(v_n \cdot v_a)$

Render to CAVE Walls: Projections - P

3. Calculate the frustum extents at the near plane

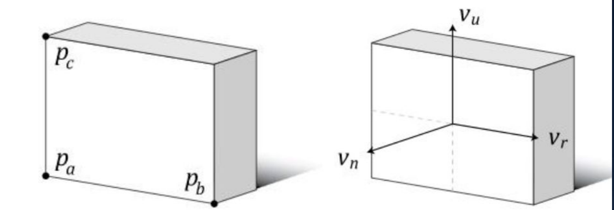
$$P' = PM^T T$$

- `P = Matrix4x4.Frustum(float left, float right, float bottom, float top, float zNear, float zFar);`
- Near and far define the near/far clipping plane
  - Depends on how you want to clip user's view



$$l = (v_r \cdot v_a)\, n/d \quad r = (v_r \cdot v_b)\, n/d$$

$$b = (v_u \cdot v_a)\, n/d \quad t = (v_u \cdot v_c)\, n/d$$



$$v_r = \frac{p_b - p_a}{||p_b - p_a||} \quad v_u = \frac{p_c - p_a}{||p_c - p_a||} \quad v_n = \frac{v_r \times v_u}{||v_r \times v_u||}$$

Render to CAVE Walls: Projections - M

- We want to transform the screens XY plane to be aligned with the viewer XY plane
- M: maps into screen coordinates
- Want to go from screen coordinates to viewer so we take the inverse of M and get $M^{-1} = M^T$
- Note that Unity Matrix is **COLUMN MAJOR**

$$P' = PM^T T$$

$$M^T = \begin{bmatrix} v_{rx} & v_{ry} & v_{rz} & 0 \\ v_{ux} & v_{uy} & v_{uz} & 0 \\ v_{nx} & v_{ny} & v_{nz} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Render to CAVE Walls: Projections - T

- Pe: Position of scene-looking camera

$$P' = PM^T T$$

$$T = \begin{bmatrix} 1 & 0 & 0 & -p_{ex} \\ 0 & 1 & 0 & -p_{ey} \\ 0 & 0 & 1 & -p_{ez} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

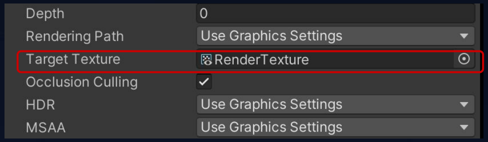
Render to CAVE Walls: Projections

- With P’, you can set the projection matrix of the scene-looking cameras
  - `camera.projectionMatrix = pPrime;`
- The off-screen render script will handle the rest!
- Remember to track the headset pose for your custom camera rig!
  - `camParent.transform.localPosition = UnityEngine.XR.InputTracking.GetLocalPosition(UnityEngine.XR.XRNode.LeftEye);`

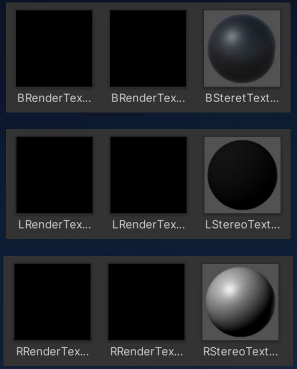
$$P' = PM^T T$$

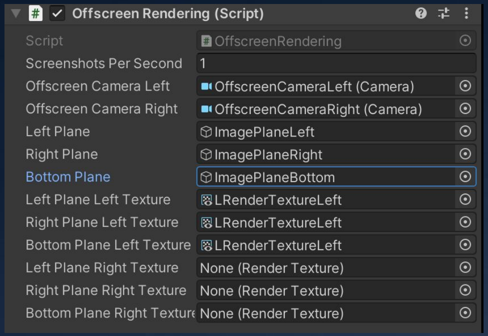
Render to CAVE Walls: Three Walls

- Now you should be able to render the same image to all three walls
- However we want to render different images to the three walls, as we have three different off-center projections
- Need the following modifications:
  - No need to set camera target texture any more, set it in code, point to the following textures
  - Create two more materials with stereo textures
    - LeftWallMaterial (Already have)
    - RightWallMaterial
      - RightWallLeftEyeTexture
      - RightWallRightEyeTexture
    - BottomWallMaterial
      - BottomWallLeftEyeTexture
      - BottomWallLeftEyeTexture



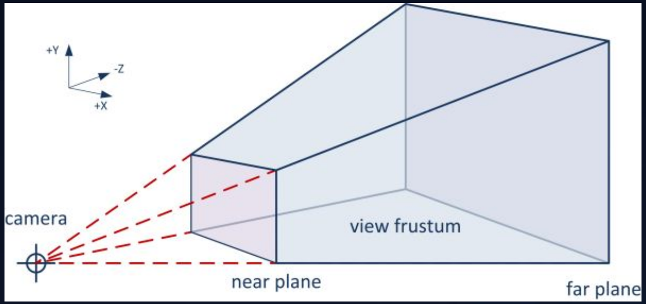

```
public RenderTexture leftPlaneLeftTexture;
public RenderTexture rightPlaneLeftTexture;
public RenderTexture bottomPlaneLeftTexture;
public RenderTexture leftPlaneRightTexture;
public RenderTexture rightPlaneRightTexture;
public RenderTexture bottomPlaneRightTexture;
```





Debug Mode

- Debug Mode to assist you with “head-in-hand” mode
- Visualize the “eye positions” of the controller
- Visualize the pyramids
- You need to draw 6 pyramids to both eyes
    - NOT 3 pyramids for each eye
    - Meaning you should see all 6 pyramids in both eyes
-

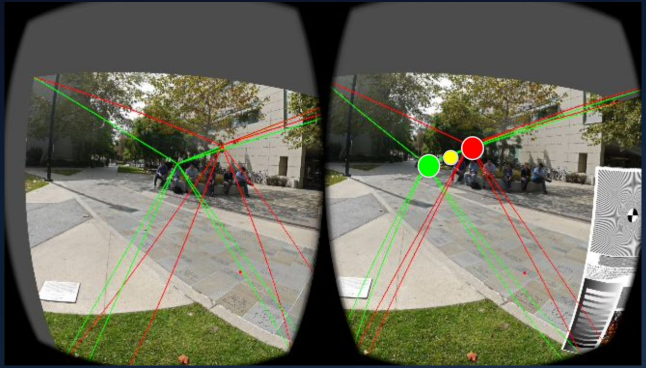


Green Dot: Left Eye Position (On the controller)

Red Dot: Right Eye Position (On the controller)

Yellow Dot: Controller Position (Just for your understanding)

P.S. Those dots don’t need to be rendered



Extra Resources

(Can also be found on course website)

Homework 4 Extra Resources

- **Offscreen Rendering in Unity** (Required to render camera views to the CAVE walls)
  - https://gist.github.com/danielbierwirth/10965844fecc38243007f0cd21843d90
- **Off-Center Projection Matrix Calculation**
  - https://web.archive.org/web/20190219024806/http://csc.lsu.edu/~kooima/articles/genperspective/
- **Original CAVE Paper**
  - http://www.cs.utah.edu/~thompson/vissim-seminar/on-line/CruzNeiraSig93.pdf

Homework 3 Q&A