# Mxr Unity Software

# Table of Contents

# Software Overview

The software contained in this Unity package will allow you to develop stereo applications for mobile devices and head mounted displays.

The file structure of this package is:
/MxrOpenSource/Scripts
/MxrOpenSource/Scenes
/MxrOpenSource/Prefabs
/MxrOpenSource/Materials

## Scripts

| | |
|---|---|
| S3dCameraSBS.js | Creates and manages stereo camera pair for desktop or mobile device |
| S3dCameraSBSEditor.js | Provides custom inspector for s3dCameraSBS.js |
| S3dGyroCamSBS.js | Accesses rotation data from gyroscope/compass on mobile device |
| S3dGyroCamSBSEditor.js | Provides custom inspector for s3dGyroCamSBS.js |
| MxrLeepHmd.cs | Sets up stereoscopic view inside the socket |
| MxrPredistortionMesh.cs | Corrects distortion in the rendered view |

## Scenes

| | |
|---|---|
| MonoFPS.unity | Sample scene demonstrating monoscopic view for HMDs |
| StereoFPS.unity | Sample scene demonstrating stereoscopic view for HMDs |
| S3dMobileDevice.unity | Sample scene demonstrating stereoscopic view for mobile devices |

## Prefabs

| | |
|---|---|
| RenderMono | Monoscopic renderer for HMDs |
| RenderStereo | Stereoscopic renderer for HMDs |
| VirtualCameraMono | Complete first-person camera keyboard/mouse control system for the monoscopic renderer. |
| VirtualCameraStereo | Complete first-person camera and keyboard/mouse control system for the stereo renderer. |
| S3dCameraMobile | Stereoscopic camera for mobile devices. |

## Materials

LeftEyeMaterial
RightEyeMaterial

## Developing in Unity

Please visit Unity's License Comparison page for a complete break down of features:
http://unity3d.com/unity/licenses

Please visit Unity's software store for up-to-date prices on all of their licensing options:
https://store.unity3d.com/

### Unity Pro 30-day Free Trial

You can download a 30-day free trial of Unity Pro, which will revert to Unity Free after the 30 days are up.

### Unity Free

If you are using Unity Free, you will need to purchase the Unity iOS add-on ($400) in order to build apps for your iOS device. Similarly, you will need the Unity Android add-on $(400) in order to build apps for your Android device. If you want to render in stereo for HMDs such as the Socket, you will need Unity Pro ($1,500).

### Unity Pro

If you are using Unity Pro ($1,500), you will need to purchase either the iOS add-on ($400) or the iOS Pro add-on ($1,500) in order to build apps for your iOS device. Similarly, you will need to purchase either the Android add-on ($400) or the Android Pro add-on ($1,500) in order to build apps for your Android device.

# Mobile Development

## iOS Requirements

You'll need to enroll in the iOS developer Program ($99/year):
https://developer.apple.com/programs/ios/

You'll need to download the latest version of XCode from Apple.

Minimum iOS version on iPhone, iPod or iPad : 4.0

Minimum Mac OS version: Mountain Lion

Documentation for setting up Unity for iOS development is located at:
http://docs.unity3d.com/Documentation/Manual/iphone-GettingStarted.html

## Android Requirements

You'll need to download and install the Android SDK and set up your development environment. Follow the instructions on the Android Developer Portal: http://developer.android.com/sdk/index.html

Documentation for setting up Unity for Android development is located at:
http://docs.unity3d.com/Documentation/Manual/android-GettingStarted.html

## Developing MXR Unity applications:

### Installation

1. Import these four scripts into your Unity project:

- **s3dCameraSBS.js**
- **s3dGyroCamSBS.js**
- **s3dCameraSBSEditor.js**
- **s3dGyroCamSBSEditor.js**).

2. Drag **s3dCameraSBSEditor.js** and **s3dGyroCamSBSEditor.js** into the **Editor** folder (or, if there isn't already an **Editor** folder, create one, and then drag the scripts into it). These scripts provide custom inspectors for **s3dCameraSBS.js** and **s3dGyroCamSBS.js.**

3. Drag **s3dCameraSBS.js** and **s3dGyroCam.js** onto the Main Camera in your scene.

### Script Reference

#### *s3dCameraSBS.js*

This script handles the creation of the stereo 3D camera, as well as rendering in the *side-by-side* format used with virtual reality headsets. To create a stereo 3D camera, just drag the **s3dCameraSBS.js** script onto the **Main Camera** in your scene.

**Interaxial**: The horizontal separation between the left and right cameras, measured in millimeters. The interaxial setting controls the overall *amount* of stereoscopic depth in the scene. 65mm corresponds to the average interocular (the distance between the left and right eyes) for most adults, and so this is the default setting. If you build your scene in real world units (one unit = one meter), then 65mm will provide the most "realistic" view of your scene. A higher value will exaggerate stereoscopic depth (known as *hyperstereo*), and increasing the value too much will result in eyestrain. (Higher values generate more *parallax*, which is a measure of the difference between the left and right views). A lower value will minimize stereoscopic depth (known as *hypostereo*), and decreasing the value too much will result in a less exciting image.

**Zero Prlx Dist** (Zero Parallax Distance): The distance at which the left and right camera views of the scene converge. The zero parallax distance sets the overall *position* of the stereoscopic depth of the scene. This is analogous to the way that our two eyes converge on a particular object at a particular distance when we focus on it. This setting turns out to be much more critical for screen-based media (when we wear 3D glasses) than for viewer-based media (such as the **FOV2GO** viewer). With screen-based media, the zero parallax distance sets the division between *theatre space* (where objects appear to be *in front of* the screen surface) and *screen space* (where objects appear to be *behind* the screen surface). When the zero parallax distance is set incorrectly with screen-based media, it can create excessive *negative* or *positive parallax,* making the image difficult or impossible to view. For viewer-based media, it can generally be left at its default setting (3 meters), and image position can be controlled with the H I T setting (horizontal image transform), described below.

**Camera Order:** Normally this should be left on its default setting, which is (not surprisingly) **Left_Right.** You can change it to **Right_Left** to *free view* the image using the crossed-eye method.

**H I T** (Horizontal Image Transform): This slider provides a way to shift the left and right camera views horizontally inside of their respective view rectangles. Applying a **horizontal image transform** to the stereo image changes the views of the cameras so that they no longer entirely overlap with each other. As mentioned above, the *interocular distance* for most adults is about 65mm. Therefore, a standard stereoscope would use a display size of about 130mm, with two images of 65mm placed next to each other, resulting in a distance of 65mm from the center of the one image to the other. But the screen widths of devices used by **FOV2GO** viewers (smartphones and tablets) vary widely - some are smaller than 130mm, and some are a good deal larger. Although the view rects (set by **Left View Rect** and **Right View Rect**, above) can be shifted to match the interocular distance, this doesn't take advantage of the fact that we are used to seeing different areas with our two eyes, with the parts of the visual field that don't overlap being perceived as *peripheral vision.* Instead, a horizontal image transform can be used to shift the relationship of the two images so that they can be viewed comfortably in a VR headset, with the parts of the image that don't overlap being perceived as *peripheral vision,* and increasing the overall sense of immersion.

**Squeezed:** For virtual reality headsets, this should be left **off (**unchecked). **Squeezed** should be enabled if you are outputting to a 3D TV that uses 50% horizontal (frame-compatible) format.

**Use Phone Mask:** Turn this **on** (checked) if you want to render the left and right camera views to a specific portion of the screen that matches the optics of your VR viewer, leaving the remainder of the screen black. The layout is controlled by the next settings, **Left View Rect** and **Right View Rect.**

**Left View Rect, Right View Rect:** These settings allow you to specify where on the screen you want your left and right views to be drawn. Each rectangle is specified by four coordinates: left, bottom, width, height.

**Examples:**

- To fill the entire screen:
    - Left View Rect: 0.0, 0.0, 0.5, 1.0
    - Right View Rect: 0.5, 0.0, 0.5, 1.0
- To leave a black bar between the left and right views, otherwise filling the entire screen:
    - Left View Rect 0.0, 0.0, 0.49, 1.0
    - Right View Rect 0.51, 0.0, 0.49, 1.0
- Samsung Galaxy Nexus:
    - Left View Rect 0.0, 0.19, 0.49, 0.81
    - Right View Rect 0.51, 0.19, 0.49, 0.81
- Samsung Galaxy Note:
    - Left View Rect 0.0, 0.22, 0.49, 0.78
    - Right View Rect 0.51, 0.22, 0.49, 0.78
- iPad in landscape mode:
    - Left View Rect 0, 0.33, 0.5, 0.67
    - Right View Rect 0.5, 0.33, 0.5, 0.67
- iPad in portrait mode:
    - Left View Rect 0.08, 0.6875, 0.42, 0.3125
    - Right View Rect 0.5, 0.6875, 0.42, 0.3125
- iPhone 4 or 5:
    - Left View Rect 0.0, 0.27, 0.49, 0.73
    - Right View Rect 0.51, 0.27, 0.49, 0.73
- HTC OneS
    - Left View Rect 0.0, 0.18, 0.49, 0.82
    - Right View Rect 0.51, 0.18, 0.49, 0.82
- HTC Rezound
    - Left View Rect 0.0, 0.27, 0.49, 0.73
    - Right View Rect 0.51, 0.27, 0.49, 0.73
- LG Thrill
    - Left View Rect 0.0, 0.19, 0.49, 0.81
    - Right View Rect 0.51, 0.19, 0.49, 0.81
- Hasbro my3D
    - Left View Rect 0.0, 0.075, 0.4675, 0.925

- Right View Rect 0.529, 0.075, 0.4675, 0.925

### *s3dGyroCamSBS.js*

This script manages a gyroscope-controlled camera on iOS and Android platforms. Just drag **s3dGyroCamSBS.js** onto the Main Camera in your scene (along with the **s3dCameraSBS.js** script, above). Note that Unity Remote does not currently support gyroscope input, so to test the script you'll have to compile your project and run it on an iOS or Android device.

**s3dGyroCamSBS.js** uses three techniques to get the correct orientation out of the gyroscope attitude: First, it creates a parent transform (named **camParent**) and rotates it with **transform.eulerAngles.** Next (for Android devices and Unity 3.5 only) it remaps the **gyro.attitude** quaternion values from xyzw to wxyz (**quatMap**). Finally, it multiplies the gyro values by another quaternion (**quatMult**) that rotates the orientation in increments of 90 degrees. The script also creates a grandparent (**camGrandparent**) that allows an arbitrary heading to be added to the gyroscope reading so that the virtual camera's heading can be set to face any direction in the scene, no matter what the phone's actual heading.

**Touch Rotates Heading:** Changing the camera's heading can be controlled via the **Touch Rotates Heading** checkbox. When checked (as it is in this scene), this allows the camera to be rotated via horizontal swipes, or via the mouse in the Unity Editor. For convenience, in the Unity Editor, the mouse also controls pitch (up/down movement). You can try running the scene in the editor, and you will be able to control the camera's rotation via the mouse.

**Set Zero Heading To North:** This uses the device's **compass** reading to ensure that, upon startup, the forward Z direction in the scene is aligned to North.

**Check For Auto Rotation:** If you have selected **Auto** in **Player Settings/Default Orientation,** then this should be checked. If you have set **Default Orientation** to a single desired orientation (normally **Landscape Left**), then **Check For Auto Rotation** should be left unchecked.

# HMD Development

Included in this package are Unity C# scripts and prefabs for correcting optical distortion in the Socket HMD (which can be generalized for other flat panel-based HMDs). A computer with a modern GPU is recommended when using distortion correction, but can also run on mobile devices as their computational power improve over time.

## Monitor Settings

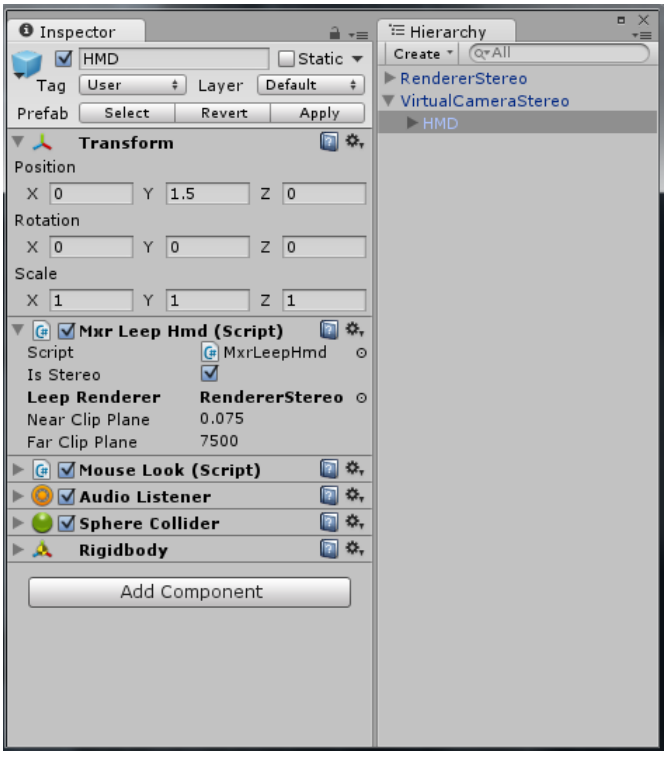
The Socket HMD will be recognized as an independent monitor when you plug it into the display port of your computer. The Socket HMD supports 1280 by 800 pixel resolution natively. It is recommended to duplicate your desktop between your primary monitor and the secondary Socket display. Applications built in Unity should run in fullscreen mode at 1280 by 800.

## Quickstart Guide

'StereoFps' is a complete scene containing everything that is necessary to render a stereoscopic view for the Socket HMD. You will be able to use keyboard (WASD) and mouse controls to navigate the environment. 'MonoFps' is similarly set up, but renders a monoscopic view instead.

Alternatively, you can drag and drop these two prefabs into an existing scene: VirtualCameraStereo and RendererStereo (or VirtualCameraMono and RendererMono)

After doing so, expand the child tree of VirtualCameraStereo in the Hierarchy panel, then highlight 'HMD'. Drag the RendererStereo game object from the Hierarchy to the exposed 'Leep Renderer' variable inside the 'Mxr Leep Hmd' component inside the Inspector panel. The result should look like the screenshot adjacent.




Include the hmd-config.cfg in the root folder of your project (or the same folder as your built executable). This config file contains distortion-correction settings that correspond with the Socket HMD.

## Adjusting Distortion Correction Settings

Optical distortion is corrected in Unity by rendering the left and right eye views to two textures. The textures are applied to two meshes which represent the left and right eye viewports. These meshes are deformed to compensate for optical distortion.

During execution of your Unity build, pressing 'G' will display a checkered grid for each eye. With the right amount of mesh deformation, the corner-angles of each square should appear to be right angles.

You can adjust the amount of deformation, as well as other settings, by editing the hmd-config file. Inside, you can adjust:

- interaxial distance between the two virtual cameras
- vertical field-of-view for each eye
- aspect ratio of each eye's viewport
- predistortion constant K (the amount of deformation for each mesh)

To test and tweak these settings during runtime, you will want to enable keyboard hotkeys inside these two scripts: MxrPredistortionMesh.cs and MxrLeepHmd.cs

MxrPredistortionMesh.cs

Uncomment lines 36 through 39 and lines 52 through 76. This enables these hotkeys:

[ - Pressing the left bracket increases the predistortion constant K

] – Pressing the right bracket decreases the predistortion constant K

MxrLeepHmd.cs

At line 74, change the value of variable "displayValues" to be true. This displays the distortion settings on the screen.

At line 178, uncomment KeyboardAdjust(); This enables these hotkeys:

HOME – Increase vertical FOV

END – Decrease vertical FOV

PAGEUP – Increase interaxial distance

PAGEDOWN – Decrease interaxial distance

| | | | |
|---|---|---|---|
| A | W<br>X | D | Moves the left viewport on the screen Up, Left, Right, or Down |
| 4 | 8<br>2 | 6 | On the keypad, moves the right viewport on the screen Up, Left, Right, or Down |

When you are able to adjust the distortion settings during runtime, their values should appear in the upper-left corner of the screen. There is currently no file-write feature to save these values to the hmd-config.cfg file. But you can update hmd-config.cfg with a text editor, and the saved settings take effect after you restart your Unity application.

## Other Settings

The deformable meshes which have the render textures applied to them are assigned to Layer 8 in Unity (by default). If you need to assign these meshes to a different layer, here are some changes you'll need to make:

- Set the culling mask for the Camera game object inside RendererStereo (or RendererMono) to render only the layer the deformable meshes reside in.
- Exclude that same layer from being rendered in the culling mask settings for the LeftEyeCamera and RightEyeCamera game objects inside VirtualCameraStereo -> HMD (or MonoCamera game object inside VirtualCameraMono -> HMD)