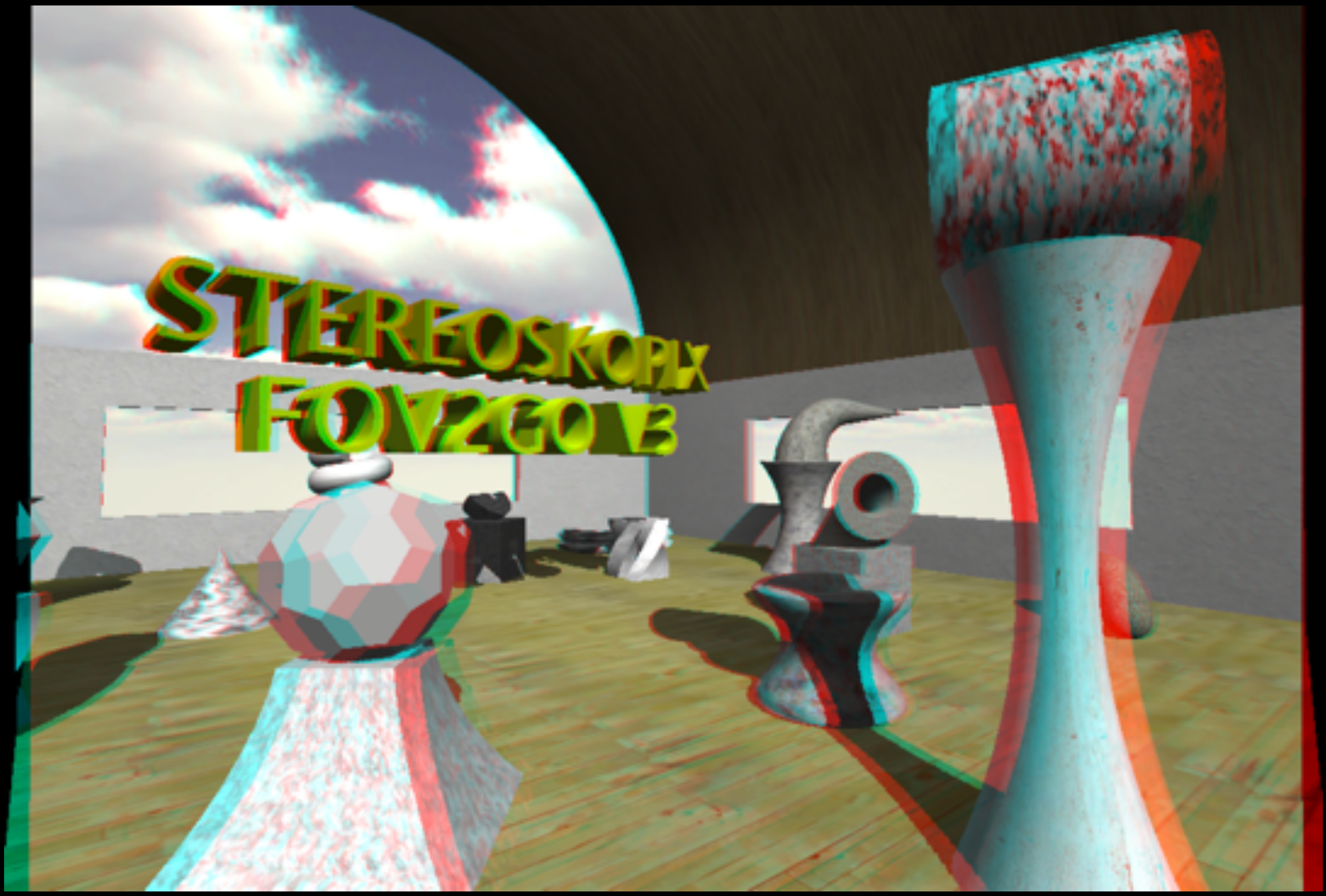
Stereoskopix FOV2GO for Unity

version 3.04



Stereoskopix FOV2GO for Unity version 3.0

Table of Contents

Introduction

Stereoskopix FOV2GO for Unity allows you to create and play Unity games in **full stereo 3D**. It supports nearly any kind of stereo 3D display system (with the exception of *time-interleaved* [field-sequential] display), including *side-by-side* and *anaglyph* display. It works on desktop, web and mobile. Now making a stereoscopic game can be as simple as dropping a script onto your camera and setting a few parameters.

Stereoskopix FOV2GO for Unity also has extensive capabilities as a component of a *virtual reality* system, using smartphones and tablets enclosed in a **FOV2GO** (or other) stereoscope.

Please direct any bugs/comments/suggestions to Perry Hoberman <hoberman@usc.edu>

FOV2GO is a hardware and software kit for the creation of immersive virtual reality experiences using smartphones, tablets and other mobile devices. It's an emerging platform for **dirt-cheap, world-class** virtual reality. The smart phone in your pocket (and the tablet in your backpack) has everything it needs to become a **low cost, state-of-the-art** virtual reality device. All the components for creating **fully immersive virtual worlds** have suddenly become ubiquitous and cheap. The only thing lacking is a kit that puts all the pieces together - so we built one. We want to see virtual reality finally develop to its true potential as an artistic medium - and so we're making these tools available to artists and designers everywhere. The **MxR Lab at USC** is making everything you need to get started freely available: plans, instructions, links & software. Visit **http://diy.mxrlab.com** for more information, documentation, etc.

FOV2GO viewers and applications are currently being developed for games, training, visualization, art and social media. You can find **FOV2GO** apps at the iTunes App store and the Android Market. Just search for **FOV2GO**,The **MxR Lab at USC** provides templates and instructions for making your own FOV2GO viewer for your iOS or Android phone or tablet. Templates and instructions are included in this package (inside the Documentation folder), and are also available at http://diy.mxrlab.com. For support and questions, you can contact us at: fov2go@ict.usc.edu

Important: Please include the following credit on your FOV2GO projects:

- *This project was developed using the FOV2GO toolkit created by the MxR Lab at USC.*
- and a link to: *http://diy.mxrlab.com*

FOV2GO is a project of the **MxR Lab at USC.**

Thanks to USC Institute for Creative Technologies, USC School of Cinematic Arts, Microsoft Research, Fakespace Labs and Phasespace.

Copyright Notice

Stereoskopix FOV2GO for Unity Copyright (c) 2011-12 Perry Hoberman.
All rights reserved.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Main 3D Camera Scripts

s3dSmoothMouseLook.js

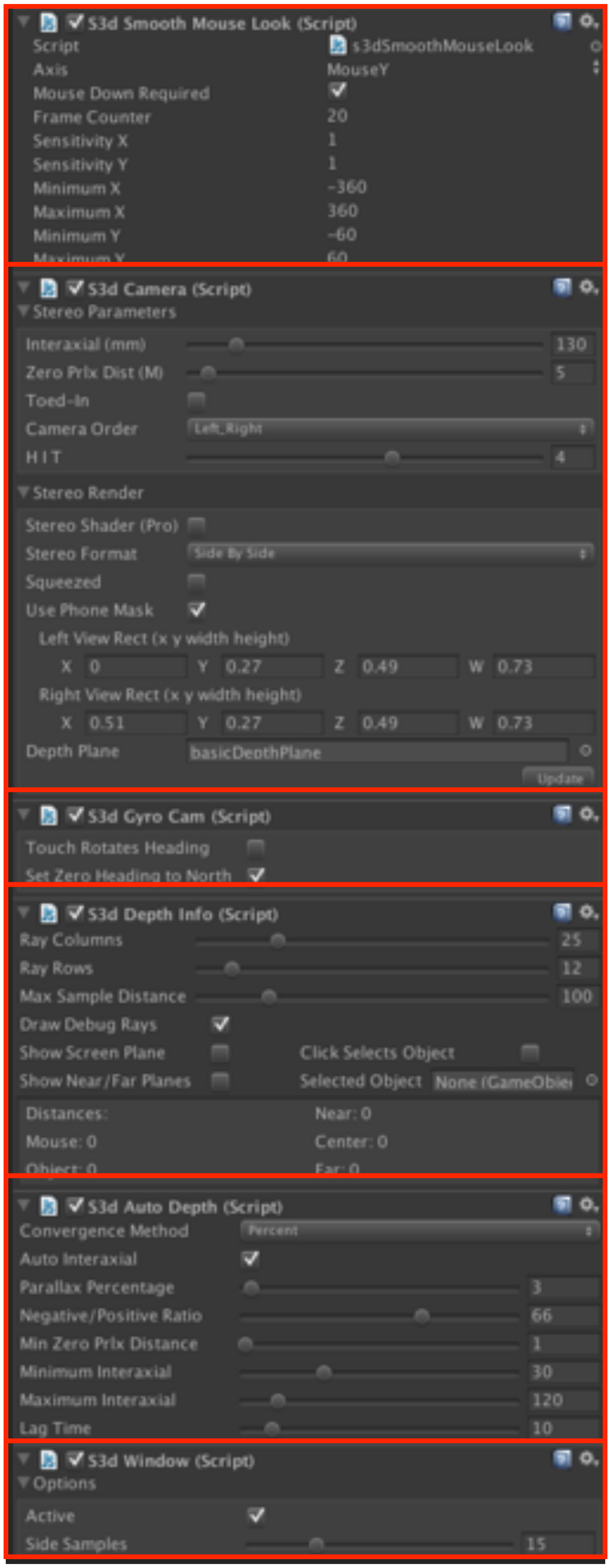
s3dCamera.js

s3dGyroCam.js

s3dDepthInfo.js

s3dAutoDepth.js

s3dWindow.js



S3d Smooth Mouse Look (Script)

Script	s3dSmoothMouseLook
Axis	MouseY
Mouse Down Required	✓
Frame Counter	20
Sensitivity X	1
Sensitivity Y	1
Minimum X	-360
Maximum X	360
Minimum Y	-60
Maximum Y	60

S3d Camera (Script)

Stereo Parameters

Interaxial (mm)	130
Zero Prlx Dist (M)	5
Toed-In	
Camera Order	Left_Right
H I T	4

Stereo Render

Stereo Shader (Pro)	
Stereo Format	Side By Side
Squeezed	
Use Phone Mask	✓

Left View Rect (x y width height)

X 0 Y 0.27 Z 0.49 W 0.73

Right View Rect (x y width height)

X 0.51 Y 0.27 Z 0.49 W 0.73

Depth Plane basicDepthPlane

Update

S3d Gyro Cam (Script)

Touch Rotates Heading	
Set Zero Heading to North	✓

S3d Depth Info (Script)

Ray Columns	25
Ray Rows	12
Max Sample Distance	100
Draw Debug Rays	✓
Show Screen Plane	
Show Near/Far Planes	
Click Selects Object	
Selected Object	None (GameObje

Distances: Near: 0

Mouse: 0 Center: 0

Object: 0 Far: 0

S3d Auto Depth (Script)

Convergence Method	Percent
Auto Interaxial	✓
Parallax Percentage	3
Negative/Positive Ratio	66
Min Zero Prlx Distance	1
Minimum Interaxial	30
Maximum Interaxial	120
Lag Time	10

S3d Window (Script)

Options

Active	✓
Side Samples	15

Section 1: Setup & Quickstart

Setup for FOV2GO Projects

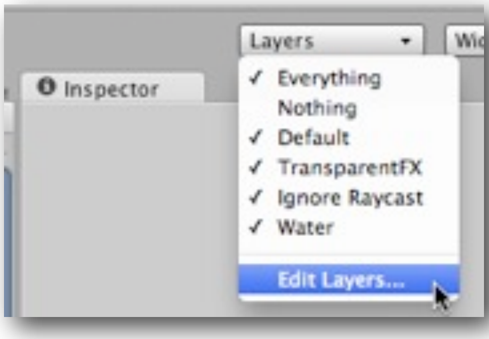
Installation

To install the package and run the tutorials:

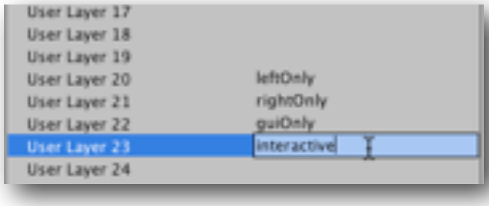
- Create an empty New Project (**File/New Project**) in Unity
- Import the **FOV2GO** package (**Assets/Import Package/Custom Package**)
- Create required custom named layers (described below)
- In the **Project** pane, navigate to the Scenes folder (**FOV2GO/Scenes/**)
- Double-click on **1_Anaglyph_FPS** to load it, and you're all set.

Creating Custom Layers

FOV2GO projects require certain **Layers** to be named for full functionality. Unfortunately, it's currently impossible in Unity to automatically set custom names for Layers when a package is imported, or even to set custom names via a script. So the first thing you should do after importing the **FOV2GO** package into your project is to manually name the necessary Layers.



1. Above the **Inspector** panel, there's a **Layers** pulldown menu. Select **Edit Layers** (the last entry).
2. This will bring up the **TagManager**, which handles **Tags** and **Layers**.
3. Layers 8 through 31 can be given user-defined names. We will be naming four of these layers.
4. Enter the following names for layers 20 through 23:
   - User Layer 20: **leftOnly**
   - User Layer 21: **rightOnly**
   - User Layer 22: **guiOnly**
   - User Layer23: **interactive**



User Layer 17	
User Layer 18	
User Layer 19	
User Layer 20	leftOnly
User Layer 21	rightOnly
User Layer 22	guiOnly
User Layer 23	interactive
User Layer 24	

Requirements

- For **3D TV** in *side-by-side squeezed* (frame-compatible) format (also *over-under stretched* format*)*, only **Unity Free** is required.
- **Unity Pro** is required to use **anaglyph** mode, due to the use of **RenderTextures** (a Pro-only feature).
- **Unity iOS** (or **Unity iOS Pro**) is required to build and run on **iOS** devices.
- **Unity Android** (or **Unity Android Pro**) is required to build and run on **Android** devices.

Quickstart: Demo Scenes

Quickstart: Anaglyph (Red/Blue) Demo Scene (Unity Pro Only)

Navigate to **Assets/FOV2GO/Scenes/**, and open **1_Demo_Anaglyph.**
Put on your anaglyph (red/blue) glasses.

The scene uses a standard first-person controller camera - look around by holding down the left mouse button, and navigate with the wasd or arrow keys.

Quickstart: Side-by-Side Demo Scene for 3DTV

Navigate to **Assets/FOV2GO/Scenes/**, and open **1_Demo_3DTV.**
Plug in your **3DTV**, and make sure that its resolution is either 1280x720 or 1920x1080.
Turn on **3D** on the **3DTV**, and make sure that **3D Settings** are set to **Side-by-Side** display
The scene uses a standard first-person controller camera - look around by holding down the left mouse button, and navigate with the wasd or arrow keys.

Quickstart: Side-by-Side Demo Scene for FOV2GO Stereoscope (iOS or Android)

1. Navigate to **Assets/FOV2GO/Scenes/**, and open **1_Demo_Stereoscope.**
2. In the **Project** view, select the **Device Manager**.
3. In the **Inspector**, select the device that you want to build for.
4. Open **File/Build Settings** (Shift-Command-B) and make sure that **Platform** is set to the correct setting.
   - If not, choose either **iOS** or **Android**, then press the **Switch Platform** button.
5. Connect a **USB** cable to your device.
6. If you are compiling for an **iOS** device, start **XCode**.
7. Select **Project Settings/Quality** and set the appropriate **QualitySettings:**
   - Most mobile devices should have **Anti Aliasing** set to **Disabled.**
   - You'll get a warning if you set **Pixel Light Count** to more than 1.
8. Set all **Player Settings** appropriately.
   - Under **Other Settings**, you'll need to specify the **Bundle Identifier**.
   - You'll also need to set the **Product Name** under **Cross-Platform Settings**.
   - Also, currently, **Default Orientation** should be set **Auto Rotation** for correct gyroscope operation.
9. Press **Build and Run** to build, load and run the app on your device.
10. Put the device into an **FOV2GO** stereoscope and get immersed.

Quickstart: Making a new stereo camera from scratch

For a 3DTV:

To create a stereo camera in an existing scene:
1. Navigate to **Assets/FOV2GO/Scripts/core**.
2. Drag the **s3dCamera.js** onto your **Main Camera.**
3. A stereoscopic camera will be created, displaying a **side-by-side** (left/right) image.
4. To format the image for a **3DTV**, make sure that **Squeezed** is checked in the **s3dCamera Inspector**.
5. Make sure that **Use Phone Mask** is unchecked.
6. Make sure that you've sized the Unity Editor to fill the screen, and select **Maximize on Play** in the **Game** window.
7. Plug in your **3DTV**, and make sure that its resolution is either 1280x720 or 1920x1080.
8. Turn on **3D** on the **3DTV**, and make sure that **3D Settings** are set to **Side-by-Side** display.
9. Press the **play** button, and you should be able to see your game in glorious **stereoscopic 3D**.

For Anaglyph (Red/Blue) Display (Unity Pro Only):

To create a stereo camera in an existing scene:

1. Navigate to **Assets/FOV2GO/Scripts/core**.
2. Drag the **s3dCamera.js** onto your **Main Camera.**
3. A stereoscopic camera will be created, displaying a **side-by-side** (left/right) image.
4. In the **s3dCamera Inspector**, click on the **Stereo Shader (Pro)** checkbox so that it's checked.
5. Navigate to **Assets/FOV2GO/Materials.**
6. Drag **stereoMat** onto the **Stereo Material** field in the **s3dCamera Inspector**.
7. For **Stereo Format,** select **Anaglyph.**
8. Put on your 3D (red/blue) glasses, press **Play**, and see your game in glorious **stereoscopic 3D.**
9. Try out the various **Anaglyph Modes** (hit **Update** each time to go to the new setting).

Adjusting Stereo Parameters:

The only stereo parameters that you need to pay attention to at first are **Interaxial** and **Zero Parallax Distance** (often referred to as *convergence).*

Interaxial: This is 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.

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.

Quickstart: Dynamic control of stereoscopic parameters

This scene uses three new scripts - **s3dDepthInfo**, **s3dAutoDepth**, and **s3dWindow** - to dynamically control all stereoscopic parameters, so that the stereoscopic depth automatically adjusts to scene content.

1. Navigate to **Assets/FOV2GO/Scenes/**, and open **1_Demo_Anaglyph_Plus.**
2. Put on your **anaglyph** (red/blue) glasses, and press **Play.**
3. As you navigate around the scene, keep an eye on the **Interaxial** and **Zero Prlx Dist** settings in the **s3dCamera Inspector** - you'll see them change dynamically as your view changes.
4. Also notice that the masking at the left and right edges of the screen, which dynamically updates to prevent what are known as **stereo window violations**. These occur when something that is located in 3D *in front* of the screen plane is cut off by the screen edge, which is *behind* it, creating a contradiction that destroys the stereo illusion.

s3dDepthInfo

This script analyzes the contents of the scene and calculates distances to the nearest and farthest objects or surfaces, to the center of the scene, and to any selected object in the scene. This information can be used by the **s3dAutoDepth** script to automatically adjust **interaxial** and **convergence** to keep them within certain limits, making the scene viewable even when the viewpoint changes radically (from a closeup to a wide shot, for instance). The script has options to visualize the **near**, **zero parallax**, and **far** planes.

Usage: Drop this script onto a **s3dCamera**. Refer to the **Reference** section to learn about the various parameters.

s3dAutoDepth

This script uses information from the **s3DdepthInfo** to dynamically set the **interaxial** and/or the **convergence** (zero parallax distance) to keep screen **parallax** within a specified percentage of the screen width. You have a choice between a number of strategies.

For **convergence**, you can choose to leave it alone (**none**), base it on a ratio between **negative** and **positive parallax** (**percent**), keep convergence on the center of the screen at all times (**center**), converge on mouse clicks (**click**), converge on mouse position (**mouse**), or converge on a selected object (**object**).

Note: To track convergence on a selected object (**object**), **Click Selects Object** has to be **checked** (on) in the **s3dDepthInfo Inspector**. Also, objects can only be clicked on if they have a **collider** component.

Note: The only convergence strategies that currently work with **Side-by-Side** mode are **percent** and **center.**

For **interaxial**, you can choose to leave it alone (**Auto Interaxial** unchecked) or keep it within a specified percentage of screen width (**Auto Interaxial** on).

Usage: Drop this script onto a **s3dCamera**. Refer to the **Reference** section to learn about the various parameters.

s3dWindow

This script provides dynamic masking on the left and right sides of the screen to prevent **stereo window violations.**

Usage: Drop this script onto a **s3dCamera**. Refer to the **Reference** section to learn about the various parameters.

Section 2: Tutorials

To help you get up to speed with **FOV2GO**, we've included a folder of demo scenes designed for **FOV2GO** stereoscopes (labeled **SideBySide**) as well as for **Anaglyph** (red/blue) display (**FOV2GO/Scenes**). These scenes are designed to lead you step by step through **FOV2GO**'s features, starting with the most basic capabilities and progressing to more advanced ones. Each scene uses the same basic setting: a very simple room with a few doors and windows.

Note: The essential components of the **FOV2GO** system (a **stereoscopic camera** and **sensor-based navigation**) are all covered in the **first three tutorial scenes**. Everything else that follows demonstrates various methods of user interaction for **FOV2GO** stereoscopes and desktop displays; but it would be perfectly possible to build compelling, fully immersive experiences using only the techniques demonstrated in the first three scenes.

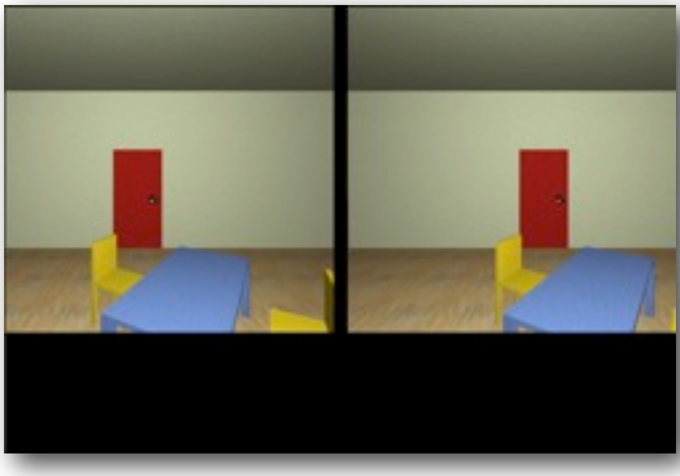
- **1_SideBySide_lookAround:** Demonstrates the most basic project concepts, including setting up a stereoscopic camera and the use of the gyroscope for looking around the scene (camera rotation).
- **1_Anaglyph_lookAround:** Demonstrates setting up a red/blue stereoscopic camera for desktop.
- **2a_SideBySide_plusNavigate**: Adds two virtual touchpads for navigation (movement) and manually rotating the camera.

- **2b_Anaglyph_lookAndNavigate:** Demonstrates basic lookaround and navigation for desktop input
- **3a_SideBySide_plus3dPointer:** Adds a third touchpad to control a 3D cursor that can be moved around in the scene
- **3b_Anaglyph_plus3dPointer:** Adds a 3D cursor that can be moved around in the scene
- **4a_SideBySide_plusInteractiveObjects:** Adds interactive furniture, doors, light switches to the scene. The chairs have interactive behaviors attached, which can be triggered by tapping and dragging with the 3D cursor.
- **4b_Anaglyph_plusInteractiveObjects:** Same as 4a but in anaglyph mode.
- **5a_SideBySide_plusPointingObjects:** Adds two first-person objects: a flashlight and a gun, along with two buttons for toggling the objects on and off
- **5b_Anaglyph_plusPointingObjects:** Same as 5a but in anaglyph mode.
- **6a_SideBySide_plusButtons:** Adds two buttons: one for loading a new scene (level), and another that brings up three touchpads for changing basic stereo settings
- **6b_Anaglyph_plusButtons:** Same as 6a but in anaglyph mode.

In the **FOV2GO/Prefabs** folder, you'll find folders for each scene with **prefabs** so that you can easily add **FOV2GO** elements to your own scenes.

Tutorial 1: LookAround (camera rotation)

Scene: 1a_SideBySide_lookAround



This scene introduces the three key scripts that drive the entire **FOV2GO** system:

- **s3dCamera.js**: the stereo camera
- **s3dGyroCam.js**: the gyroscope manager
- **s3dDeviceManager.js**: the device manager

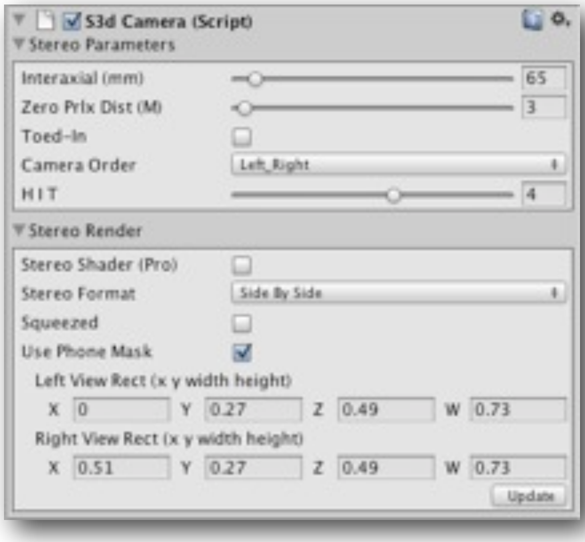
These scripts are in the **FOV2GO/Scripts/Core** folder, and each has a companion **Editor Script** (in the **FOV2GO/Editor** folder) that provides a custom Inspector.

There is also an **anaglyph** (red/blue) version of this scene (**1a_Anaglyph_lookAround**). For camera rotation (instead of **s3dGyroCam.js**), it uses **s3dSmoothMouseLook.js** (a variant of the standard **MouseLook.cs** script). You can test this scene by running it in the editor. Look around by clicking and dragging the mouse. *Please note that anaglyph mode requires **Unity Pro**, as it makes use of **render textures**, which is a pro-only feature.*

Script: s3dCamera.js

This script handles the creation of the stereo 3D camera, as well as rendering in a number of different stereo formats. We are primarily concerned here with the *side-by-side* format used with the **FOV2GO** viewer, but the script also supports *anaglyph* (red/blue) and a number of other viewing formats. To create a stereo 3D camera, just drag the **s3dCamera.js** script onto the **Main Camera** in your scene (it's already set up for you in this tutorial scene). Its default setting is *side-by-side* format, which will work in any version of Unity (some other formats require Unity Pro).

Note: In most cases, you will want to have exactly **one s3dCamera** in your scene (and that will generally be the **Main Camera**). While it is possible to have multiple **s3dCameras** in a scene, certain components of the basic **FOV2GO** package will not function correctly if there's more than one.



S3d Camera (Script) [✓]

Stereo Parameters

Interaxial (mm)	65
Zero Prlx Dist (M)	3
Toed-In	☐
Camera Order	Left_Right
H I T	4

Stereo Render

Stereo Shader (Pro)	☐
Stereo Format	Side By Side
Squeezed	☐
Use Phone Mask	✓

Left View Rect (x y width height)

X 0 Y 0.27 Z 0.49 W 0.73

Right View Rect (x y width height)

X 0.51 Y 0.27 Z 0.49 W 0.73

The inspector is divided into two panels: **Stereo Parameters** and **Stereo Render.**

Stereo Parameters

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.

Toed In: This is an esoteric setting that will only be of interest to stereoscopic specialists. Its default setting is **off**, and it should be left that way. If interested, see http://www.binocularity.org/page3.php for an explanation.

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. See http://www.angelfire.com/ca/erker/freeview.html for details.

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. While a H I T shift can be very useful for viewer-based formats (such as the **FOV2GO** viewer), it should normally be set to 0 for screen-based formats such as anaglyph. 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 an **FOV2GO** viewer, with the parts of the image that don't overlap being perceived as *peripheral vision,* and increasing the overall sense of immersion.

Stereo Render

Stereo Shader (Pro): For the **FOV2GO** viewer, this should be left **off** (unchecked)**,** since the **FOV2GO** viewer uses **side-by-side** format, which doesn't require the use of the stereo shader. Other formats (notably **anaglyph**) require Unity Pro, due to the fact that they make use of *render textures* (described below).

Stereo Format: For the **FOV2GO** viewer, this will always be set to **Side By Side.** Other options are described below.

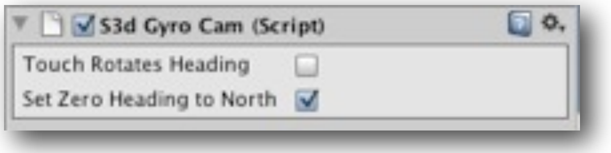
Squeezed: For the **FOV2GO** viewer, this should be left **off (**unchecked). **Squeezed** should be enabled if you are outputting to a 3D TV that uses 50% horizontal (frame-compatible) format.

Use Phone Mask: For the **FOV2GO** viewer, this should be **on** (checked). This allows you to render the left and right camera views to a specific portion of the screen that matches the optics of the **FOV2GO** viewer, leaving the remainder of the screen black for interface elements such as **touchpads** (described below). The layout is controlled by the next settings, **Left View Rect** and **Right View Rect.**

Left View Rect, Right View Rect: Although it is possible to set the these manually, they will normally be controlled by the **s3dDeviceManager.js** script, described below.

Update button: After you've changed any Render options, you may have to press this button to update the game view. If the scene still doesn't update, just press Play and then Stop, and everything should update to the current settings.

Script: s3dGyroCam.js


S3d Gyro Cam (Script)
Touch Rotates Heading ☐
Set Zero Heading to North ✓

This script manages a gyroscope-controlled camera for **FOV2GO** (iOS and Android platforms). Just drag **s3dGyroCam.js** onto the Main Camera in your scene (along with the **s3dCamera.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.

s3dGyroCam.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 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.

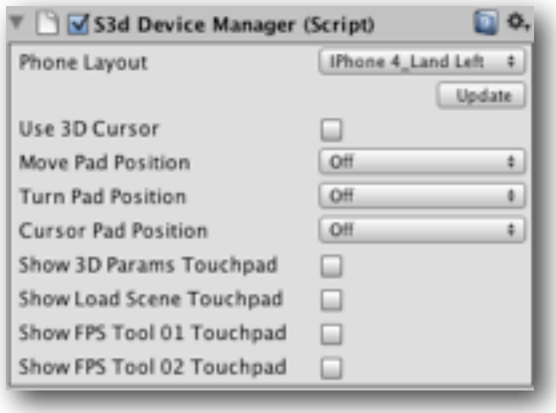
Touch Rotates Heading: Changing the camera's heading would normally be controlled via a **s3dTouchpad** (described below), but it can also be controlled via the **Touch Rotates Heading** checkbox. When checked (as it is in this scene), this allows the camera to be rotated via horizontal swipes, or via 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: The only other inspector setting is **Set Zero Heading To North**, which will use 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: For **Unity 3.x**, **Default Orientation** should be set **Auto Rotation** for correct gyroscope operation, and **Check For Auto Rotation** should be checked. This is *not* necessary for **Unity 4.x**, therefore **Default Orientation** should be set to a single desired orientation (normally **Landscape Left**), and **Check For Auto Rotation** should be left unchecked.

Script: s3dDeviceManager.js

This script handles the layout for **FOV2GO** viewer apps. In this first scene, it is used only to set the left and right camera **view rectangles** to the correct positions for a specified mobile device. All other options are unchecked in this scene, and will be described as we make use of them in further tutorial scenes.



S3d Device Manager (Script)

Phone Layout	iPhone 4_Land Left
	Update
Use 3D Cursor	☐
Move Pad Position	Off
Turn Pad Position	Off
Cursor Pad Position	Off
Show 3D Params Touchpad	☐
Show Load Scene Touchpad	☐
Show FPS Tool 01 Touchpad	☐
Show FPS Tool 02 Touchpad	☐

Pick your phone or tablet in the **Phone Layout** pulldown menu. If your device isn't listed, pick the setting that is closest to your device (alternately, you can disable **s3dDeviceManager.js**, and manually change any of the settings on the various game objects themselves).

Note: Unless the Game View is sized to the exact pixel dimensions of your device screen, the layout of your scene may not match the layout on your device, especially regarding camera view rectangles and GUI elements. Some devices are listed in the Game View Aspect drop-down menu, and if your device is listed, you should select it there. If your device isn't listed, you should pick the one that's closest to it in pixel dimensions. But for certain devices, matching the Game View to the device screen resolution is going to be next to impossible - for example, the iPad 3 (with Retina display) will be too large (2048 x 1536) to fit on most laptop or desktop screens. Since the iPad 2's pixel dimensions (1024 x 768) are exactly 1/2 of the iPad 3's, you can set the Device Manager to iPad 2 to test in the editor - but don't forget to switch it back to iPad 3 before you build.

Note that the **anaglyph** scenes do not include a **s3d_Device_Manager** object - because the **s3dDeviceManager.js** script is designed specifically to deal with layout on mobile devices, not the desktop.

Supported Devices

Here are the specs for the devices that are currently listed in the s3d Device Manager:

- Apple iPad 2
  - resolution: 768 x 1024 pixels
  - screen dimensions: 149 x 198.1mm
- Apple iPad 3
  - resolution: 1536 x 2048 pixels
  - screen dimensions: 149 x 198.1mm
- Apple iPhone 4/4S
  - resolution: 640 x 960 pixels
  - screen dimensions: 49.2 x 74.88mm
- HTC One S
  - screen resolution: 540 x 960 pixels
  - screen dimensions: 53mm x 95mm
- HTC Rezound
  - screen resolution: 720 x 1280 pixels
  - screen dimensions: 53 x 95mm
- LG Thrill 4G
  - screen resolution: 480 x 800 pixels
  - screen dimensions: 56 x 93mm
- Samsung Galaxy Nexus
  - resolution: 720 x 1280 pixels
  - screen dimensions: 58 x 96mm
- Samsung Galaxy Note
  - screen resolution: 800 x 1280 pixels
  - screen dimensions: 71 x 113.5mm

Note: The iPad has settings for both **portrait** (vertical) and **landscape** (horizontal) configurations - pick one or the other depending on which **FOV2GO** viewer you are using.

Hasbro my3D Viewer

In 2011, the toy company **Hasbro** introduced a stereoscope designed to hold an iPhone or iPod Touch, called **my3D**. Originally priced at $35 (and available exclusively at **Target**), as of mid-2012, you can pick one up (if you can find one) for about $15.

It certainly isn't an ideal device for FOV2GO content - the lenses have a long focal length, and touchscreen interaction is limited by two small thumb holes in the base of the viewer. But it does work, and if you don't feel like constructing your own FOV2GO viewer (and you have an iPhone or an iPod touch), then it's well worth getting your hands on one.

s3dDeviceManager.js has a Phone Layout setting called **My3D_LandLeft,** which will select the correct screen layout for the **my3D** viewer.

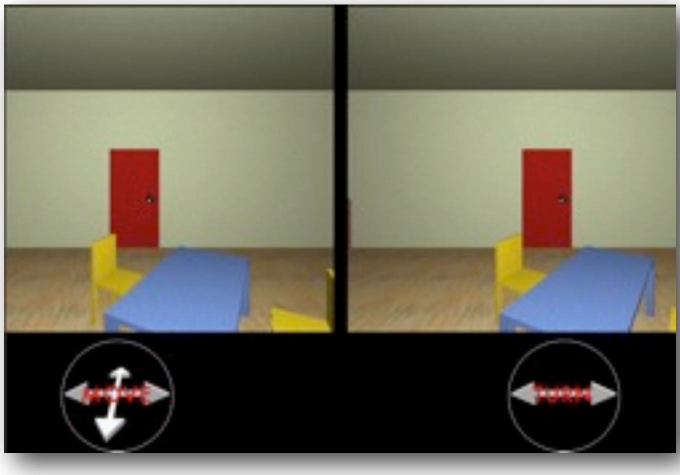
To Compile and Test App on Mobile Device

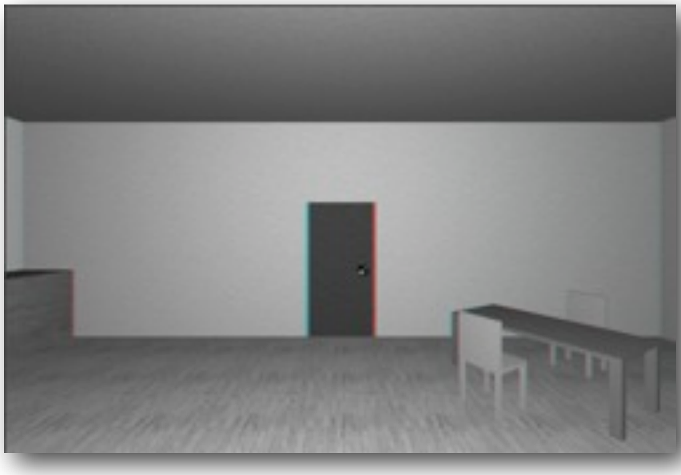
Detailed instructions for building and deploying scenes on Android or iOS mobile devices is covered in the Unity documentation, and is beyond the scope of this document, but here's a quick overview of the steps involved:

1. Open the scene in the Unity Editor.
2. Connect a USB cable to your device.
3. If you are compiling for an iOS device, start **XCode**.
4. Select **Project Settings/Quality** and set the appropriate **QualitySettings:**
   - Many mobile devices should have **Anti Aliasing** set to **Disabled.**

    - You'll get a warning if you set **Pixel Light Count** to more than 1.
5. Select **File/Build** Settings (Shift-Command-B)
6. Make sure that **Platform** is set to the correct setting
    - Click **Switch Platform** if necessary.
7. Set all **Player Settings** appropriately.
    - Under **Other Settings**, you'll need to specify the **Bundle Identifier**.
    - You'll also need to set the **Product Name** under **Cross-Platform Settings**.
    - For **Unity 3.x**, **Default Orientation** should be set **Auto Rotation** for correct gyroscope operation. This is *not* necessary for **Unity 4.x**, therefore **Default Orientation** should be set to a single desired orientation (normally **Landscape Left**).
8. Press **Build and Run** to build, load and run the app on your device.

Tutorial 2: Navigation (camera movement)





Scene: 2a_SideBySide_plusNavigation

This scene adds two virtual **touchpads**: one for navigation (movement), and the other to control the camera's heading (rotation). The scene also adds a FirstPersonController object (called **s3d_FirstPersonController**).

The following scripts are introduced in this scene:

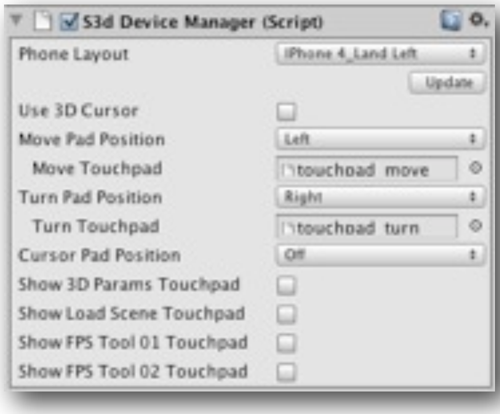
- **s3dTouchpad.js:** Used along with a **GUI Texture** to create a virtual touchpad
- **s3dFirstPersonController.js**: A first-person controller; requires a **Character Controller** component and a **Character Motor** script to provide first-person navigation. Controlled by a **s3dTouchpad**.
- **s3dRotateHeading.js**: Script to manually rotate camera heading. Controlled by a **s3dTouchpad.**

These scripts are in the **FOV2GO/Scripts/Core** folder, and **s3dTouchpad.js** has a companion **Editor Script** (in the **FOV2GO/Editor** folder) that provides a custom Inspector.

Like the previous scene, this scene contains a **s3d_Device_Manager** object with a **s3dDeviceManager.js** script. The touchpads for navigation and turning need to be entered here so that they appear in the correct positions for a particular mobile device. If you look at the **s3d_Device_Manager** inspector, you'll see that **Move Pad Position** has been changed from **Off** to **Left,** exposing a field for a **s3dTouchpad**, to which **touchpad_move** (a child of **s3d_Touchpads** in the scene) has been assigned. Next, **Turn Pad Position** has been set to **Right**, and **touchpad_turn** (also a child of **s3d_Touchpads**) has been assigned.

Because **s3dTouchpad.js** accepts mouse input in the Unity Editor, you can run this scene in the Editor to simulate running it on a mobile device, using the mouse to simulate touch input.

Note about included s3dTouchpad textures: The visible part of the textures for the Move, Turn, and Cursor touchpads are square (256 x 256 pixels), but the image dimensions are actually twice as wide (512 x 256 pixels), with an alpha channel to make the background drop away. This is to make it easier to find the touchpads with your thumbs, since the touchpads will usually be out of your visual field as you operate them.



S3d Device Manager (Script)

Phone Layout	iPhone 4_Land Left
	Update
Use 3D Cursor	☐
Move Pad Position	Left
Move Touchpad	touchpad move
Turn Pad Position	Right
Turn Touchpad	touchpad turn
Cursor Pad Position	Off
Show 3D Params Touchpad	☐
Show Load Scene Touchpad	☐
Show FPS Tool 01 Touchpad	☐
Show FPS Tool 02 Touchpad	☐

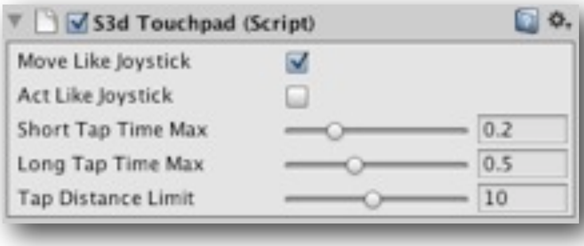
Scene: 2b_Anaglyph_lookAndNavigate

There is also an **anaglyph** (red/blue) version of this scene (**2b_Anaglyph_lookAndNavigate**). For camera movement, it uses the standard **FPSInputController.js** script (instead of **s3dFirstPersonController.js**), and for camera rotation (instead of **s3dGyroCam.js**), it uses two instances of **s3dSmoothMouseLook.js** (a variant of the standard **MouseLook.cs** script). You can test this scene by running it in the editor. Navigate by using the WASD keys (or the arrow keys), and look around by clicking and dragging the mouse.

Script: s3dTouchpad.js

s3dTouchpad.js creates a virtual touchpad from a **GUI Textur**e that can be used to control an element in the virtual world, such as the camera or an object in the world. It responds just like a real touchpad to taps and swipes, and when used in the Unity editor, it accepts mouse clicks and drags so that you can simulate touchpad behavior on a mobile device.

FOV2GO viewers are designed to be open at the bottom, so that the touchscreen is still accessible via the user's thumbs. The lenses are placed slightly higher than the center of the screen, leaving the lower portion of the screen available for thumb-based user input. For this purpose, one or more virtual **touchpads** are created. These function just like the physical touchpad on a laptop. The virtual touchpads are placed just outside the visual field, and can be used to manipulate elements inside the visual field. In this sense, they are more like the standard interface devices used on a laptop or desktop system. With a touchscreen, the display screen and the input device are one and the same, but with both desktop and virtual reality systems, the display screen is of necessity discrete from the input system.



S3d Touchpad (Script)

Move Like Joystick	✓
Act Like Joystick	☐
Short Tap Time Max	0.2
Long Tap Time Max	0.5
Tap Distance Limit	10

s3dTouchpad.js is based on the **Joystick.js** script included with Unity Penelope iOS Tutorial, but it's been substantially stripped down and rewritten to provide the specific functionality required by **FOV2GO**. Unlike **Joystick.js**, which can be set to either mimic a joystick or a touchpad, **s3dTouchpad.js** can combine elements of either (touchpad behavior with optional joystick visual movement, or joystick behavior without visual movement). **Joystick.js** handles multitouch, but because we never need to handle more than one touch per touchpad, all that complexity has been stripped away. Also, **s3dTouchpad.js** only deals with single taps (but it can distinguish between long and short taps when desired).

Unlike many other elements in the **FOV2GO** system, which automatically clone themselves to provide left and right eye elements, **s3dTouchpads** are singular - they appear in one location on the screen. When used with an **FOV2GO** viewer, they are designed to be rendered outside of the visual field. For other 3D formats, such as anaglyph, they can appear in or out of the camera view.

Move Like Joystick: When checked, the **GUI Texture** will follow touch and mouse drag movements, snapping back to the origin when the touch is lifted. When unchecked, the GUI Texture doesn't move at all. This has no effect on the touchpad's behavior.

Act Like Joystick: When checked, the touchpad x and y values will jump to the initial touch/click position (in the range -1.0 to 1.0), and then change *relative to this value* when dragged. When unchecked, the touchpad x and y values will start at 0.0 (no matter where the initial touch/click occured) and then change *relative to the initial position* when dragged.

ShortTapTimeMax: This sets the maximum amount of time for a short tap - if the touch lasts longer than this, it's considered a long tap. The default setting is 0.2 seconds.

LongTapTimeMax: This sets the maximum amount of time for a long tap - if the touch lasts longer than this, it's not considered a tap. The default setting is 1.0 seconds.

Tap Distance Limit: This sets the maximum amount of movement for a tap - if the touch moves more than this, it's no longer considered a tap. The default setting is 10 pixels.

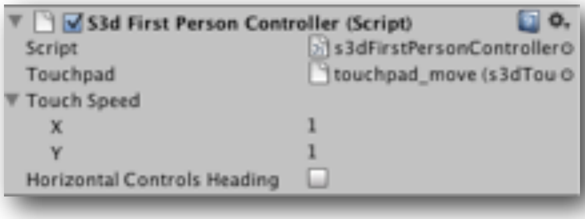
How to create a new s3d Touchpad from scratch

Here are the steps you would need to create a new **s3dTouchpad** (for this scene, the two **s3dTouchpads** are already set up):

1. Create a **GUI Texture** game object.
2. Drag a **s3dTouchpad.js** script onto the object, or onto its Inspector.
3. Drag a texture into the **GUI Texture** Inspector's **Texture** field.
4. Set the object's **Layer** to **guiOnly.**
5. Set all fields in the **Transform** of the object (including **Scale**) to 0. (You can change the **Position X** and **Y** fields later, but for now they should both be set to 0).
6. Set the **GUITexture**'s **Pixel Inset** to **X** = 0, **Y** = 0, **Width** = desired width in pixels, **Height** = desired height in pixels.
7. The object should now appear in the lower lefthand corner of the game view.
8. Since the origin for **GUI Texture** coordinates is the lower left corner, setting **Transform Position X** = 0 and **Y** = 0 puts the lower left hand corner of the **GUI Texture** right at the origin. Now you can reposition the **GUI Texture** anywhere in relation to the lower left hand corner by changing the **Pixel Inset X** and **Y** fields to positive values.
9. However, what happens if you want to position the **GUI Texture** in relation to one of the other corners? Unless you know the exact pixel dimensions of your device, it's impossible to do this using the **Pixel Inset** fields.
10. Let's say you want to set the **GUI Texture** in relation to the upper right hand corner.
11. First, set **Transform Position X** = 1 and **Y** = 1. This places the lower left hand corner of the **GUI Texture** in the upper right hand corner of the display - which unfortunately means it's entirely out of view.

12. To fix this, set the **Pixel Inset X** and **Y** fields to the **Width** and **Height** values multiplied by –1. (For example: **X** = –320 **Y** = –240 **Width** = 320 **Height** = 240).
13. Your **GUI Texture** should now appear in the upper right hand corner of the display.
14. Finally, to set a **GUI Texture** in relation to the center of the display, just set **Transform Position X** and/or **Y** to 0.5, and set the **Pixel Inset X** and/or **Y** to –1/2 the **Width** and/or **Height** values.

Script: s3dFirstPersonController.js



S3d First Person Controller (Script)	
Script	s3dFirstPersonController
Touchpad	touchpad_move (s3dTou
Touch Speed	
X	1
Y	1
Horizontal Controls Heading	☐

This script provides the same functionality as the standard **FPSInputController.js** script (included in the standard **Character Controller** package). However, instead of using the standard **Horizontal** and **Vertical** axes provided by the **Input Manager**, this script uses a **s3dTouchpad** (called **touchpad_move** in this scene) for input. Vertical touchpad input moves the player forward and back, and horizontal touchpad movement moves the player left and right (default; see below)

The script is attached to a First Person Controller object (called **s3d_FirstPersonController**), which parents the **Main Camera** object. This object has a **Character Motor** script (added via **Component/Character/Character Motor**) and a **Character Controller** component (added automatically when the **Character Motor** script is added).

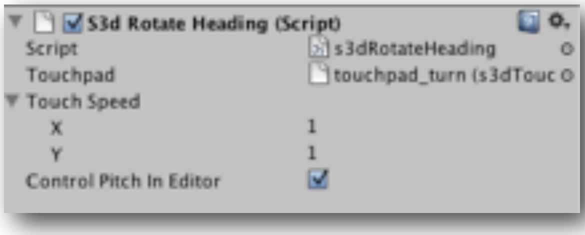
Touchpad: Drag a **s3dTouchpad** from your scene into this field, or click on the Object Picker (the little target icon) and pick an available **s3dTouchpad**.

Touch Speed: Scales the touchpad's output. You can set different values for x and y, although normally these would be identical. Default is 1 1.

Horizontal Controls Heading: In the default **off** (unchecked) setting, horizontal input moves the player left and right. When checked, horizontal movement instead rotates the player; that is, it controls the camera heading. For FOV2GO applications, camera rotation (including heading) would normally controlled by the gyroscope, so this setting would be left unchecked.

Script: s3dRotateHeading.js

For FOV2GO applications, full lookaround capability is normally provided by the gyroscope. However, it is often convenient to be able to rotate the camera heading manually (for instance if the user is seated), and this functionality is provided by **s3dRotateHeading.js**, which accepts input from a **s3dTouchpad**.



S3d Rotate Heading (Script)	
Script	s3dRotateHeading
Touchpad	touchpad_turn (s3dTouc
Touch Speed	
X	1
Y	1
Control Pitch In Editor	✓

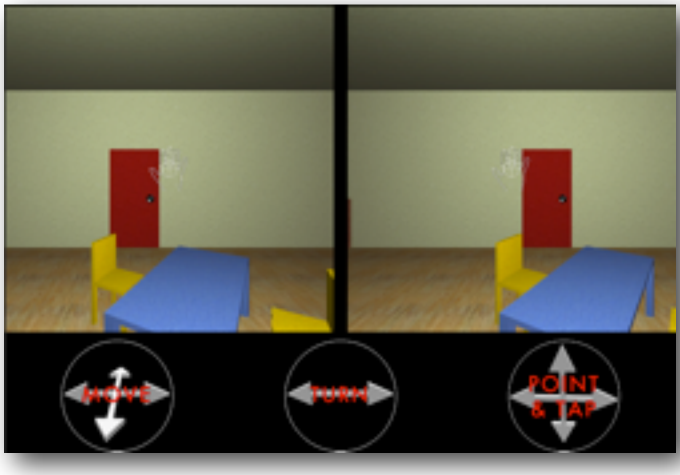
Here it is controlled by the righthand touchpad (called **touchpad_turn** in this scene), operated by the righthand thumb. It is, however, not indispensable, and in later tutorial scenes (in which the righthand thumb controls a 3D cursor) **touchpad_turn** is moved to a center position, where it is somewhat less accessible.

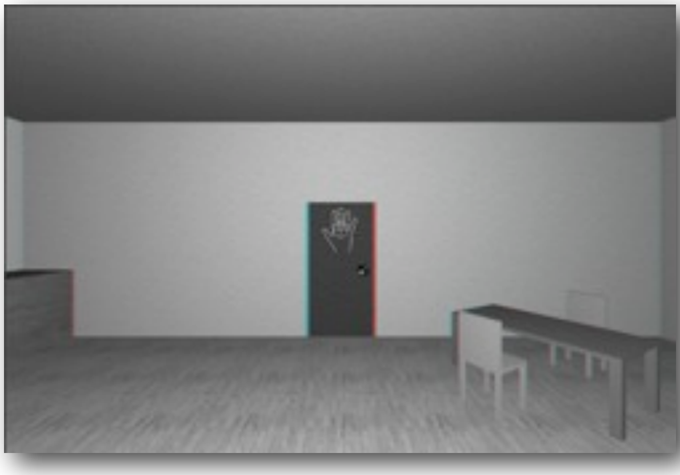
Touchpad: Drag a **s3dTouchpad** from your scene into this field, or click on the Object Picker (the little target icon) and pick an available **s3dTouchpad**.

Touch Speed: Scales the touchpad's output. You can set different values for x and y, although normally these would be identical. Default is 1 1.

Control Pitch in Editor: Because the gyroscope is unavailable in the Unity Editor (and because the gyroscope is unsupported in Unity Remote), there is normally no convenient way to simulate full camera lookaround in the Editor. However, when **Control Pitch in Editor** is checked (default = on), full camera lookaround can be simulated using the assigned **s3dTouchpad.** Then, when run on the mobile device, **s3dRotateHeading.js** controls heading only (and pitch is controlled via the gyroscope).

Tutorial 3: 3D Cursor





Scene: 3a_SideBySide_plus3dPointer

This scene adds a third touchpad to control a 3D cursor that can be moved around in the scene.

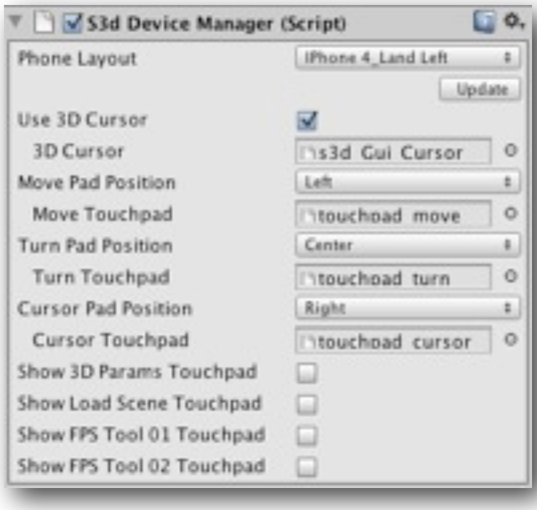
The following scripts are introduced in this scene:

- **s3dGuiTexture.js:** used with a GUI Texture to create a left & right stereo pair from the texture.
- **s3dGuiCursor.js:** used with a s3dGuiTexture.js to create a 3D cursor in the scene.

Like the previous scenes, this scene contains a **s3d_Device_Manager** object with a **s3dDeviceManager.js** script.

In this scene, **Use 3D Cursor** has been checked, exposing a field for a 3D cursor (a **s3dGuiCursor**), to which **s3d_Gui_Cursor** has been entered. Also, the touchpad for the 3D cursor needs to be entered here so that it appears in the correct position for a particular mobile device. **Cursor Pad Position** has been changed from **Off** to **Right,** exposing a field for another **s3dTouchpad**, to which **touchpad_point** (a child of **s3d_Touchpads** in the scene) has been assigned. Also, **Turn Pad Position** has been change to **Center**, so that **touchpad_turn** now appears in the center position.

Again, because **s3dTouchpad.js** accepts mouse input in the Unity Editor, you can run this scene in the Editor to simulate running it on a mobile device, using the mouse to simulate touch input. The third **s3dTouchpad (Point & Tap)** moves the 3D cursor around in the scene.



S3d Device Manager (Script)

Phone Layout	iPhone 4_Land Left
	Update
Use 3D Cursor	✓
3D Cursor	s3d Gui Cursor
Move Pad Position	Left
Move Touchpad	touchpad move
Turn Pad Position	Center
Turn Touchpad	touchpad turn
Cursor Pad Position	Right
Cursor Touchpad	touchpad cursor
Show 3D Params Touchpad	☐
Show Load Scene Touchpad	☐
Show FPS Tool 01 Touchpad	☐
Show FPS Tool 02 Touchpad	☐

Special note for Hasbro my3D: Because the my3D's thumbholes severely restrict touchscreen interaction, it's impossible to make more than two touchpads accessible (one per thumb) - so in the **My3D_LandLeft** settings of **s3dDeviceManager.js**, all other touchpads have been moved to the center strip between the left and right views, where they can only be accessed by temporarily snapping open the viewer.

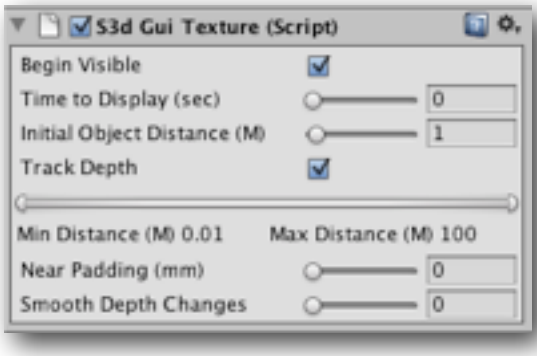
Scene: 3b_Anaglyph_plus3dPointer

There is also an **anaglyph** (red/blue) version of this scene (**3b_Anaglyph_plus3dPointer**). It also uses **s3dGuiTexture.js** and **s3dGuiCursor.js** to create a stereo cursor (described below), but instead of using a **s3dTouchpad** to control the cursor, the cursor is controlled directly by mouse or touchpad input (the option **Track Mouse Position** has been checked, and **Use Touchpad** is unchecked).

Script: s3dGuiTexture.js

Since a **GUI Texture** is a 2D element that is rendered on top of camera view, it does not automatically support stereoscopic imaging, for which each element has to be assigned a position in stereoscopic depth - an operation that normally requires two copies of the element that can be positioned in a precise relationship to each other.

s3dGuiTexture.js is a versatile script that can be added to a **GUI Texture** to create left and right copies of the texture so that it can be used as a stereoscopic element. It can adjust the **GUI Texture**'s parallax automatically to keep the element closer than anything it occludes (this is necessary because the **GUI Texture** will always be rendered on top of the camera view - and if it were to appear to be stereoscopically behind any element in the view, a depth paradox would result).



S3d Gui Texture (Script)

Begin Visible	✓
Time to Display (sec)	0
Initial Object Distance (M)	1
Track Depth	✓
Min Distance (M) 0.01	Max Distance (M) 100
Near Padding (mm)	0
Smooth Depth Changes	0

s3dGuiTexture objects are automatically placed in **leftOnly** Layer and **rightOnly** Layer (which you should have named when you installed the **FOV2GO** package, above. These layers are set in **s3dCamera.js**, and the defaults are Layer 20 for **leftOnly** and Layer 21 for **rightOnly**.

Begin Visible: Default **on** (checked). If unchecked, can be toggled by calling **toggleVisible**() function.

Time To Display (sec): Default 0 = leave on forever. If set to any other value, will disappear after specified interval.

Initial Object Distance (M): Default 1 meter. Only has effect if **Track Depth** is left unchecked.

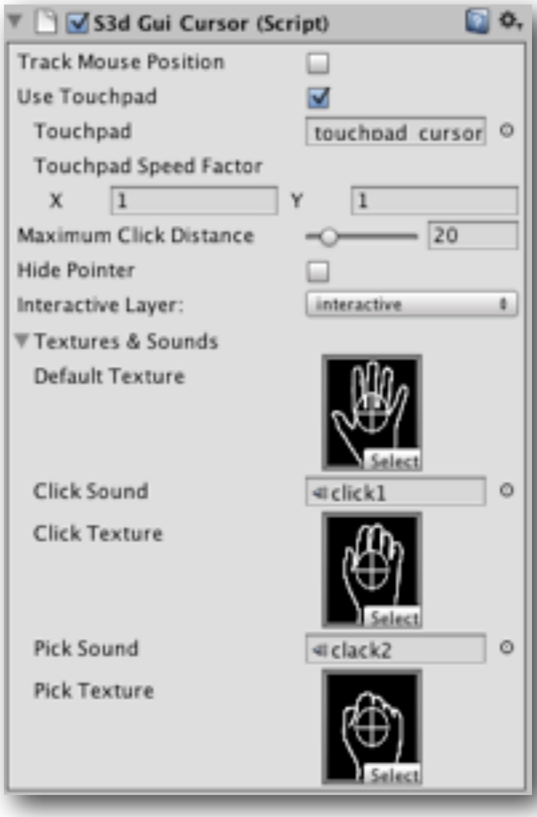
Track Depth: Default **on** (checked) Automatically updates stereo depth (parallax) to keep texture closer than scene objects behind it.

Min Distance/Max Distance: Defaults 0.01 and 100 meters. Minimum and maximum allowed stereoscopic depth.

Near Padding (mm): Default 0. Bring forward slightly so that texture is that much closer than objects directly behind it.

Smooth Depth Changes: Default 0. Change depth more gradually (reasonable values are up to 25).

Script: s3dGuiCursor.js


S3d Gui Cursor (Script)
Track Mouse Position ☐
Use Touchpad ✓
Touchpad: touchpad cursor
Touchpad Speed Factor X 1 Y 1
Maximum Click Distance 20
Hide Pointer ☐
Interactive Layer: interactive
Textures & Sounds
Default Texture
Click Sound: click1
Click Texture
Pick Sound: clack2
Pick Texture

This script works in conjunction with **s3dGuiTexture.js** (which it requires to be attached to the same game object), and transforms the stereo 3D texture into a stereo 3D cursor.

For a stereoscopic cursor, 3 textures are assigned: one for its default state, one for clicking on an object, and one for when an object is selected (**default**, **click** and **pick**). The cursor can also be assigned sounds for clicking and picking. In this scene, an image of an open hand is used for the default cursor texture, with the fingers progressively curved inward for the **click** and **pick** textures.

A 3D cursor created with **s3dGuiCursor.js** can be used to trigger actions on specified objects by tapping and dragging on them, which is demonstrated in the next tutorial scene. The current scene simply demonstrates the cursor itself - the cursor doesn't have any effect on the objects in the scene.

Track Mouse Position: Default **off** (unchecked). Track mouse position on desktop. Leave off for use with s3dTouchpad.

Use Touchpad: Default **off** (unchecked), but should be **on** (checked) to control via **s3dTouchpad** for FOV2GO applications.

Touchpad: Drag a **s3dTouchpad** from your scene into this field, or click on the Object Picker (the little target icon) and pick an available **s3dTouchpad**.

Touchpad Speed Factor X Y: Scales the touchpad's output. You can set different values for x and y, although normally these would be identical. Default is 1 1.

Maximum Click Distance: Default 20. Only objects closer than this distance will register taps or clicks.

Hide Pointer: Default **off** (unchecked). For FOV2GO apps, there is no mouse pointer, so setting is irrelevant. For desktop applications, where **s3dGuiTexture.js** is used to create a 3D cursor, this can be toggled **on** (checked) to hide the default cursor.

Interactive Layer: Default 23 (Layer named **interactive**). Objects placed in this layer can be tapped/clicked on. Objects placed in other layers will be used to determine cursor depth, but will not register taps/clicks.

Default Texture: Assign Texture for default state.

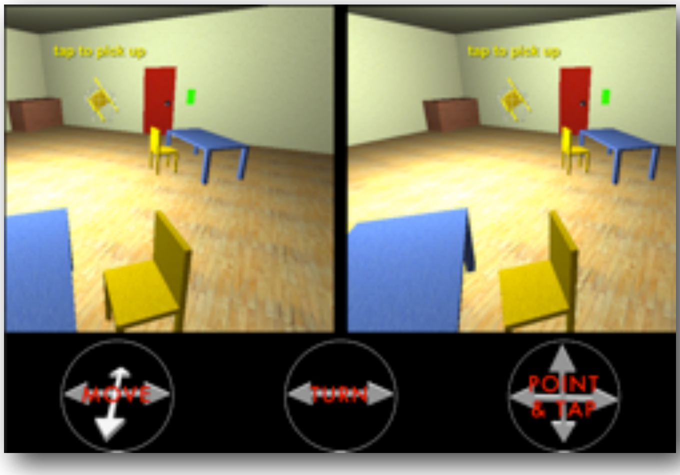
Click Sound: Assign AudioClip for tap/click actions.

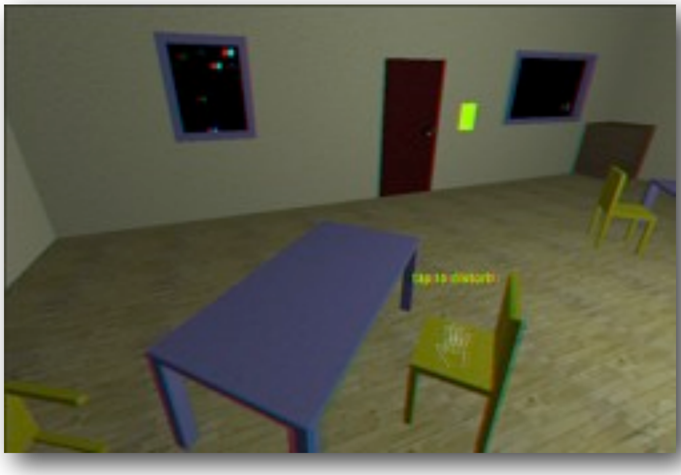
Click Texture: Assign Texture for momentary click state.

Pick Sound: Assign AudioClip to play when object is picked.

Pick Texture: Assign Texture for object pick state.

Tutorial 4: Interactive Objects





Scene: 4a_SideBySide_plusInteractiveObjects

This scene demonstrates FOV2GO's interaction system. Additional furniture has been added to the scene, and each chair has one or more interactive behaviors attached, which can be triggered by tapping and dragging with the 3D cursor. Also, the four doors can be opening and shut by tapping on them; and there are four light switches (on the walls next to the doors) that, when tapped/clicked on, will raise and lower the room lighting.

The following scripts are introduced in this scene:

- **s3dInteractor.js**
- **s3dGuiText.js**

• **Interaction** scripts (located in **FOV2GO/Scripts/interactions** folder)

- **pickUpPutDown.js**
- **jumpInTheAir.js**
- **rolloverText.js**
- **rotateAndSpin.js**
- **pivotObject.js**
- **pushAway.js**
- **centerToCamera.js**
- **toggleLighting.js**

Like the previous scenes, this scene contains a **s3d_Device_Manager** object with a **s3dDeviceManager.js** script. In this scene, the settings for **s3dDeviceManager** remain unchanged from the previous scene.

Scene: 4b_Anaglyph_plusInteractiveObjects

There is also an **anaglyph** (red/blue) version of this scene (**4b_Anaglyph_plusInteractiveObjects**). Again, instead of using a **s3dTouchpad** to control the cursor, the cursor is controlled directly by mouse or touchpad input (the option **Track Mouse Position** has been checked, and **Use Touchpad** is unchecked). Otherwise, the functionality of the interactive system is identical to the description that follows.

As in the previous scene, the 3D cursor is actually controlled by dragging and tapping on the right hand **s3dTouchpad**. If you run the scene, you'll notice that moving the 3D cursor over one of the chairs brings up a text message that describes what will happen when you tap (or click) on the chair. Tap on the various chairs to try out the various interactive behaviors (There's also a light switch on the wall that toggles the room lighting up and down). Most of the interactive scripts implement momentary behaviors that respond the same way each time you tap/click. However several of them (**pickUpPutDown.js** and **centerToCamera.js**) work as toggles: the first tap/click selects the object, and the object remains selected until the next tap/click.

Rules for interactive objects & scripts

These are the basic rules for creating interactive objects, and for writing additional **interaction** scripts.

- All interactive objects should be placed in the **interactive** layer (default = Layer 23)
- All interactive objects need to have **colliders** (**Box, Sphere, Capsule, Mesh**, etc)
- Any of the Interaction scripts included in the **FOV2GO/Scripts/interactions** folder can be added.
- Other interaction scripts can be written following the examples in the **FOV2GO/Scripts/interactions** folder.
- All **interaction** scripts should include the line: **@script RequireComponent(s3dInteractor)**. This line ensures that when the script is added to an object, **s3dInteractor.js** is automatically added as well.
- Some **interaction** scripts require a **RigidBody** component, which can be added automatically by including the line **@script RequireComponent(RigidBody)**
- Some **interaction** scripts require a **AudioSource** component, which can be added automatically by including the line **@script RequireComponent(AudioSource)**

Program flow for interactive objects

When an interactive object is rolled over or tapped on with the 3D Cursor:

- First, the 3D Cursor's **s3dGuiCursor.js** script does a **GetComponent** to find the object's **s3dInteractor.js** script.
- Next, the 3D Cursor's **s3dGuiCursor.js** script runs one of these functions in the object's **s3dInteractor.js** script:
    - **tapAction**()
    - **upDatePosition()**
    - **deactivateObject()**
    - **rolloverText()**

- The function (in the object's **s3dInteractor.js** script) does a **SendMessage** on the object
- The **SendMessage** is picked up by one or more **interaction** scripts attached to the object, running one of these functions contained in the interaction script.
    - **NewTap**()
    - **NewPosition**()
    - **Deactivate**()
    - **ShowText**()
    - **HideText**()

Most interaction scripts will contain (at minimum) a **NewTap** function that processes taps. The **NewTap** function accepts an argument (**params**) that includes the **RayCastHit** from the 3D Cursor's raycast (**params.hit**) and an **integer** (**params.tap**) that specifies whether the tap was short (1) or long (2). Most interaction scripts include a variable called **tapType** that can be set to respond to short taps (1) long taps (2) or any tap (3).

Interaction Script Example

Here’s the basic structure of an interaction script.

```
@script RequireComponent(s3dInteractor);
var tapType : int = 3; // 1 = short tap 2 = long tap 3 = any tap

function NewTap(params: TapParams) {
   if (params.tap == tapType || tapType == 3) {
       // do something!
   }
}
```

You can examine any of the scripts in the **FOV2GO/Scripts/interactions** to see how various other functions are implemented.

Script: s3dGuiText.js

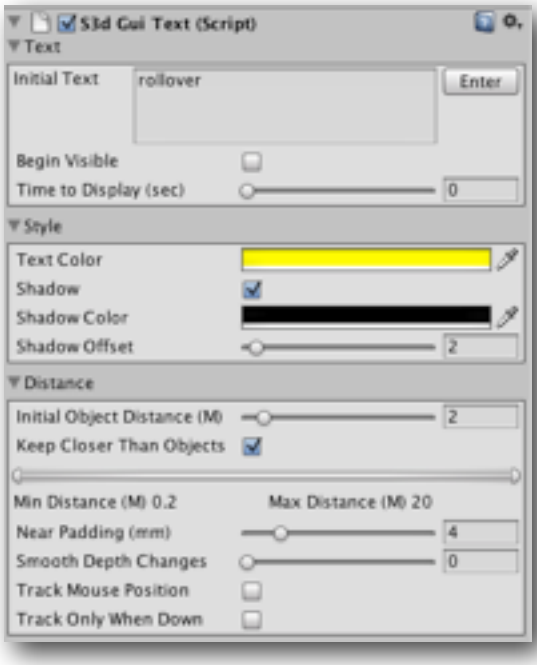
rolloverText.js, which has been added to each of the interactive objects, makes use of a **GUI Text** object that has an attached **s3dGuiText.js** script. **s3dGuiText.js** is very similar to **s3dGuiTexture.js** - it can be added to a **GUI Text** to create left and right copies of the text so that it can be used as a stereoscopic element. Like **s3dGuiTexture.js**, it can adjust the **GUI Text**’s parallax automatically to keep the element closer than anything it occludes.

__Text__

Initial Text: Enter the text that should be displayed when the scene starts.

Begin Visible: Default **on** (checked). If unchecked, can be toggled by calling **toggleVisible**() function.

Time To Display (sec): Default 0 = leave on forever. If set to any other value, will disappear after specified interval.



S3d Gui Text (Script) [✓]

Text

Initial Text	rollover	Enter
Begin Visible	☐	
Time to Display (sec)		0

Style

Text Color	🟨
Shadow	✓
Shadow Color	⬛
Shadow Offset	2

Distance

Initial Object Distance (M)	2
Keep Closer Than Objects	✓
Min Distance (M) 0.2	Max Distance (M) 20
Near Padding (mm)	4
Smooth Depth Changes	0
Track Mouse Position	☐
Track Only When Down	☐

__Style__

Text Color: Default White. Set the text color

Shadow: Default **on** (checked). Creates a second copy of the text slightly offset from the first, typically set to a darker color so that it appears to be the shadow of the text. Increases legibility.

Shadow Color: Default Black. Set the text shadow color.

Shadow Offset: Default 5. Determines how far (to the right and down) the text shadow should appear.

__Distance__

Initial Object Distance (M): Default 1 meter. Only has effect if **Track Depth** is left unchecked.

Track Depth: Default **on** (checked) Automatically updates stereo depth (parallax) to keep texture closer than scene objects behind it.

Min Distance/Max Distance: Defaults 0.01 and 100 meters.

Minimum and maximum allowed stereoscopic depth.

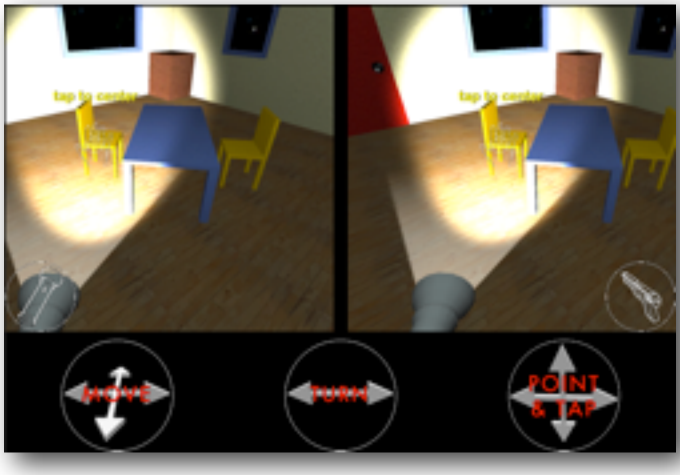
Near Padding (mm): Default 0. Bring forward slightly so that texture is that much closer than objects directly behind it.

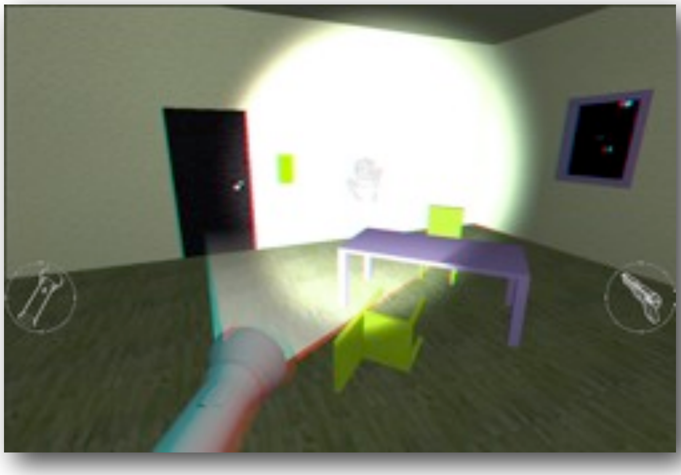
Smooth Depth Changes: Default 0. Change depth more gradually (reasonable values are up to 25).

Track Mouse Position: Default **off** (unchecked). When **on,** track mouse position on desktop.

Track Only When Down: Default **on** (checked). When **off** (unchecked), text will follow mouse position at all times.

Tutorial 5: First Person Pointing Objects



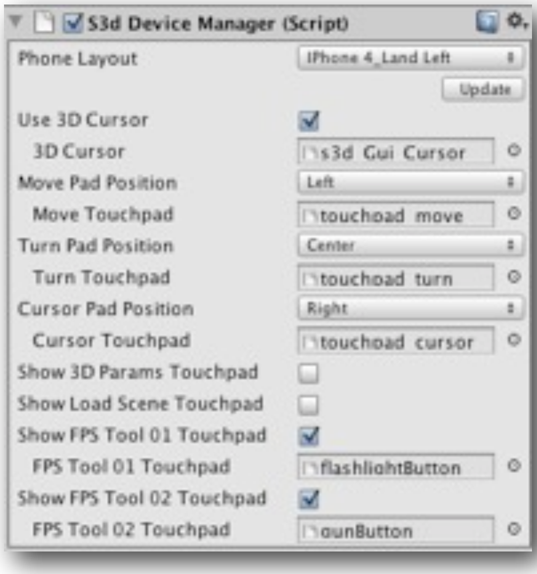


Scene: 5a_SideBySide_plusPointingObjects

This scene adds two first-person objects: a flashlight and a gun, along with two **s3dTouchpads** for toggling the objects on and off (and firing the gun).

This scene includes the following new scripts:

- **triggerObjectButton.js**
- **aimObject.js**
- **pointAtPoint.js**
- **fireGun.js**



S3d Device Manager (Script)

Phone Layout	IPhone 4_Land Left
	Update
Use 3D Cursor	✓
3D Cursor	s3d Gui Cursor
Move Pad Position	Left
Move Touchpad	touchpad move
Turn Pad Position	Center
Turn Touchpad	touchpad turn
Cursor Pad Position	Right
Cursor Touchpad	touchpad cursor
Show 3D Params Touchpad	☐
Show Load Scene Touchpad	☐
Show FPS Tool 01 Touchpad	✓
FPS Tool 01 Touchpad	flashlightButton
Show FPS Tool 02 Touchpad	✓
FPS Tool 02 Touchpad	gunButton

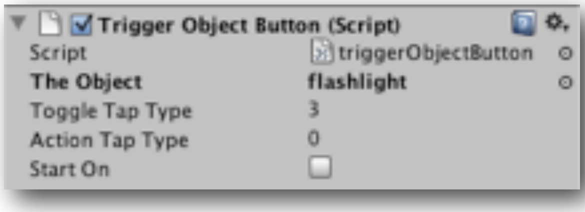
Like the previous scenes, this scene contains a **s3d_Device_Manager** object with a **s3dDeviceManager.js** script. In this scene, **Show FPS Tool 01** has been checked, exposing a field for a **s3dTouchpad,** for which **flashlightButton** (a child of **s3d_Touchpads** in the scene) has been entered. **Show FPS Tool 02** has also been checked, exposing a field for a **s3dTouchpad,** for which **gunButton** (a child of **s3d_Touchpads** in the scene) has been entered.

This scene adds two small **s3dTouchpads**: one on the left for toggling the flashlight on and off, and one on the right for toggling a gun on and off. Both the gun and the flashlight automatically aim at the current position of the 3D cursor. The touchpad for the gun is set to accept only short taps to toggle the gun on and off; long taps trigger a separate function: when the gun is toggled on, a long tap will cause it to fire. You can run the scene in the editor to test all of these various functions.

Scene: 5b_Anaglyph_plusPointingObjects

There is also an **anaglyph** (red/blue) version of this scene (**5b_Anaglyph_plusPointingObjects**). Once more, instead of using a **s3dTouchpad** to control the cursor, the cursor is controlled directly by mouse or touchpad input. Otherwise, the functionality of the interactive system is identical to the description that follows - including the two small **s3dTouchpads** (for toggling the flashlight and gun) which work the same way in both versions.

Scripts: triggerObjectButton.js & fireGun.js



Trigger Object Button (Script)	
Script	triggerObjectButton
The Object	flashlight
Toggle Tap Type	3
Action Tap Type	0
Start On	☐

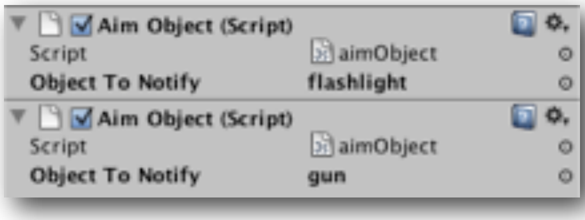
The two new **s3dTouchpads** each have an additional script: **triggerObjectButton.js** (contained in **FOV2GO/Scripts/ ButtonBehaviors).**

Set **The Object** to the object that you want to control (in this case, both objects - the flashlight and the gun - are children of **Main Camera**, so that they stay with the camera wherever it moves or turns to).

ToggleTapType can be set to 0 (ignore) 1 (short tap), 2 (long tap) or 3 (any tap). The specified tap will toggle the object (and any children) on and off. **ActionTapType** can also be set to ignore, short, long or any tap, and will trigger a function called **Go()** in any script attached to the receiving object.

The **triggerObjectButton.js** attached to the **gunButton** touchpad is set so that short taps toggle the gun object on and off, and long taps trigger a **SendMessage** that runs a **Go**() function in the target object. For the gun object, this triggers the **Go**() function in **fireGun.js.**

Script: aimObject.js

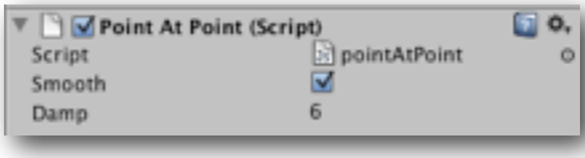


Aim Object (Script)	
Script	aimObject
Object To Notify	flashlight

Aim Object (Script)	
Script	aimObject
Object To Notify	gun

The 3D Cursor has two instances of a new script called **aimObject.js.** This script takes one parameter, called **Object To Notify**, which is set to the object to be aimed at the 3D Cursor's position. These are set to the flashlight and gun objects. **aimObject.js** uses the same technique as **s3dGuiTexture.js**, finding the target object's **s3dInteractor.js** script and triggering a function in that script (in this case the **updatePosition** function)

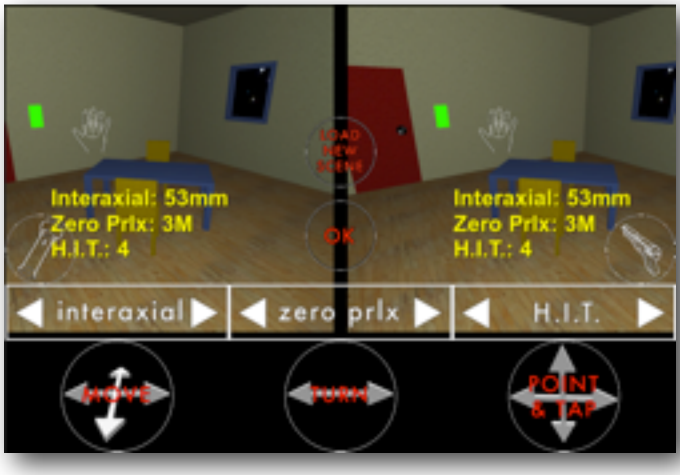
Script: pointAtPoint.js

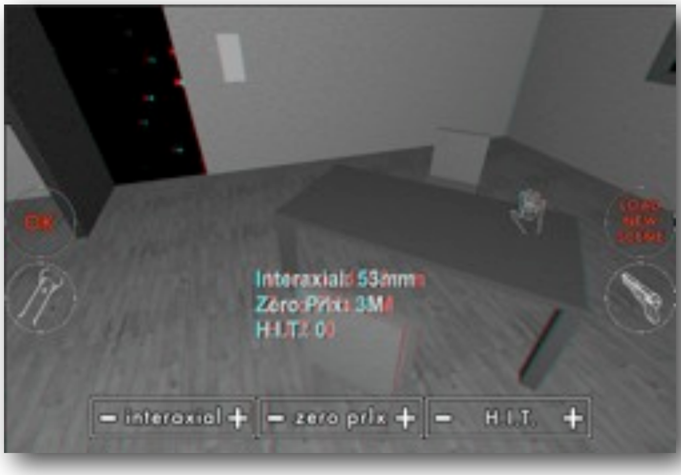


Point At Point (Script)	
Script	pointAtPoint
Smooth	✓
Damp	6

The two objects being controlled - the flashlight and the gun objects - each have a **pointAtPoint.js** script. This is an interaction script (requiring an **s3dInteractor.js** script). When **updatePosition** is triggered in the **s3dInteractor.js** script, it runs a **SendMessage** that triggers the **NewPosition** function in **pointAtPoint.js,** aiming the object at the current position of the 3D Cursor.

Tutorial 6: Button Functions





Scene: 6a_SideBySide_plusButtons

This scene adds two **s3dTouchpads**: one for loading a new scene (level), and another that brings up three additional **s3dTouchpads** for changing basic stereo settings.

This scene includes the following new scripts:

- **triggerSceneChange.js**
- **s3dStereoParameters.js**

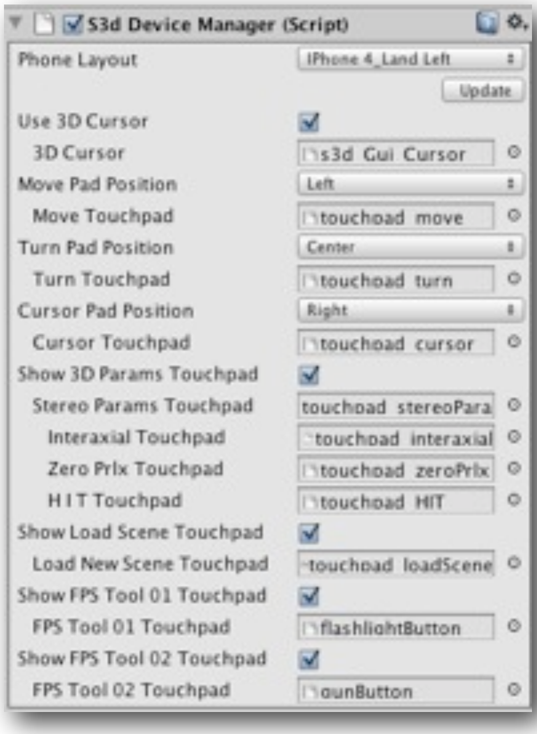
Like the previous scenes, this scene contains a **s3d_Device_Manager** object with a **s3dDeviceManager.js** script. The **s3d_Device_Manager** also has a new script attached, called **s3dStereoParameters.js**. This script works in conjunction with **s3dDeviceManager.js** to provide an interface for adjusting key stereo parameters.

In **s3dDeviceManager.js**' inspector, you'll notice that **Show 3D Params Touchpad** has been checked, exposing a number of new fields:

- **Stereo Params Touchpad** (field for **s3dTouchpad):** set to **touchpad_stereoParams**
- **Interaxial Touchpad** (field for **s3dTouchpad):** set to **touchpad_interaxial**
- **Zero Prlx Touchpad** (field for **s3dTouchpad):** set to **touchpad_zeroPrlx**
- **H I T Touchpad** (field for **s3dTouchpad):** set to **touchpad_HIT**

These touchpads provide an interface for adjusting key stereo parameters, which is controlled by **s3dStereoParameters.js** (explained below).

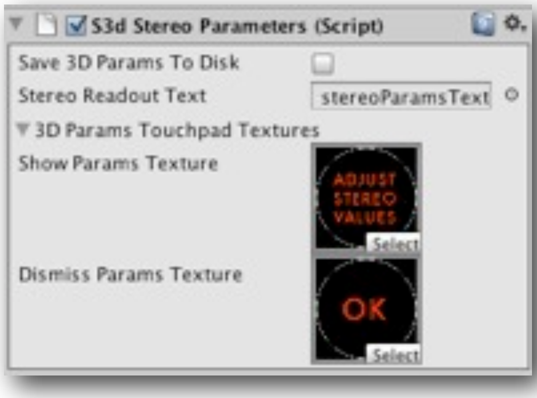
Show Load Scene Touchpad has also been checked, exposing a field for **Load New Scene Touchpad** (another **s3dTouchpad),** set to **touchpad_loadScene.**



S3d Device Manager (Script)

Phone Layout	iPhone 4_Land Left
	Update
Use 3D Cursor	✓
3D Cursor	s3d Gui Cursor
Move Pad Position	Left
Move Touchpad	touchpad move
Turn Pad Position	Center
Turn Touchpad	touchpad turn
Cursor Pad Position	Right
Cursor Touchpad	touchpad cursor
Show 3D Params Touchpad	✓
Stereo Params Touchpad	touchpad stereoPara
Interaxial Touchpad	touchpad interaxial
Zero Prlx Touchpad	touchpad zeroPrlx
H I T Touchpad	touchpad HIT
Show Load Scene Touchpad	✓
Load New Scene Touchpad	touchpad loadScene
Show FPS Tool 01 Touchpad	✓
FPS Tool 01 Touchpad	flashlightButton
Show FPS Tool 02 Touchpad	✓
FPS Tool 02 Touchpad	gunButton

Script: s3dStereoParameters.js



S3d Stereo Parameters (Script)

Save 3D Params To Disk	☐
Stereo Readout Text	stereoParamsText

▼ 3D Params Touchpad Textures

Show Params Texture	
Dismiss Params Texture	

s3dStereoParameters.js contains functions for adjusting the key stereo parameters (interaxial, zero parallax distance, and horizontal image transform). As always, you can run the scene in the Unity editor to test all functions. Try running the scene and clicking on the **Adjust Stereo Values** button. This will bring up three **s3dTouchpads** for adjusting stereo parameters. Click in the left half of each touchpad to reduce the value; click in the right half of the touchpad to increase the value. Click the **OK** button to dismiss the stereo parameter touchpads.

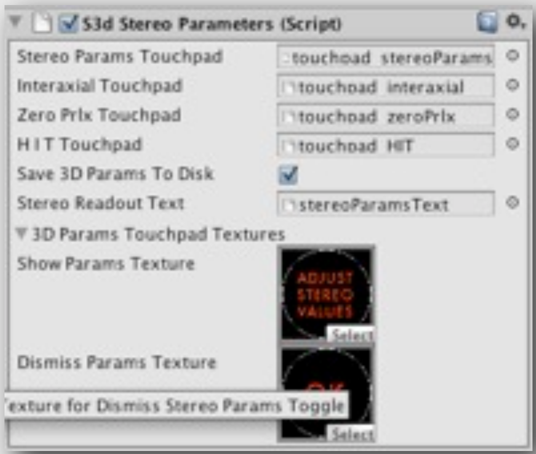
- **Save 3D Params To Disk:** Default **on** (checked): when checked, saves user settings for stereo parameters (**Interaxial**, **Zero Prlx** and **H I T**) to disk (to the PlayerPrefs file), and loads these updated settings next time app is run. When unchecked, settings will revert to the values that were set at build time.
- **Stereo Params Texture** and **Dismiss Params Texture**: these two fields are set to two Textures to be toggled depending on whether the three **s3dTouchpads** for Stereo Parameters (**Interaxial**, **Zero Prlx** and **H I T**) are visible or not.
- **Stereo Readout Text** (field for **s3dText**): set to **stereoParamsText**

If **Save 3D Params To Disk** is checked, the values set by these three **s3dTouchpads** will be saved to disk (via PlayerPrefs) when this app is built and run on a mobile device (and then will be loaded from disk next time the app is run).

Scene: 6b_Anaglyph_plusButtons

There is also an **anaglyph** (red/blue) version of this scene (**6b_Anaglyph_plusButtons**). Other than differences already noted, everything functions identically to the viewer version of the scene.

Note that, however, since there is no **s3dDeviceManager.js** script, the various **s3dTouchpads** are each positioned via their individual inspectors. And **s3dStereoParameters.js** is designed so that, when it's attached to an object that doesn't also have a **s3dDeviceManager.js** script attached**,** fields for the various **s3dTouchpads** are exposed in the inspector of **s3dStereoParameters.js**.



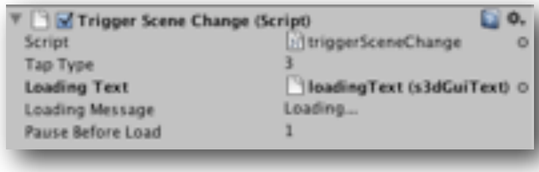
S3d Stereo Parameters (Script)

Stereo Params Touchpad	touchpad_stereoParams
Interaxial Touchpad	touchpad_interaxial
Zero Prlx Touchpad	touchpad_zeroPrlx
H I T Touchpad	touchpad_HIT
Save 3D Params To Disk	✓
Stereo Readout Text	stereoParamsText

▼ 3D Params Touchpad Textures

Show Params Texture	
Dismiss Params Texture	

Script: triggerSceneChange.js



Trigger Scene Change (Script)

Script	triggerSceneChange
Tap Type	3
Loading Text	loadingText (s3dGuiText)
Loading Message	Loading...
Pause Before Load	1

This script (found in the **FOV2GO/Scripts/ButtonBehaviors** folder) is attached to the **touchpad_loadScene** object. When clicked/tapped, the next scene (in Build Settings) will be loaded. **TapType** can be set to 1 (short tap), 2 (long tap) or 3 (any tap). **Loading Text** should be set to a **s3dText** object (in this example it's set to **loadingText**). **Loading Message** (default “Loading…”) determines the message that will be displayed just before the next level is loaded. **Pause Before Level** (default = 1 second) introduces a short delay before the next level is loaded for **Loading Text** to be displayed.

Automatic Selection of Device Layout

Scripts: androidDeviceSelector.js & iosDeviceSelector.js

Included in the **FOV2GO** package are two scripts - one for Android devices (**androidDeviceSelector.js**) and the other for iOS devices (**iosDeviceSelector.js**)- that can automatically update the screen layout based on what device the app is running on. This allows you to build a single app that can run on all supported devices on a given platform.

On startup, these scripts check the screen resolution of the device - and then, if that resolution matches one of the presets contained in **s3dDeviceManager.js** - the display will automatically switch to that screen layout.

Just add the appropriate script (depending on your build platform) to the **s3d_Device_Manager** object (that is, the same object that has the **s3dDeviceManager.js** script attached) and the screen layout will be selected automatically when the app runs on the device.

Setting up Anaglyph mode from scratch (Unity Pro Only)

- Toggle **Stereo Shader (Pro)** on.
- Fields for **Stereo Material, Render Texture L** and **Render Texture R** will appear.
- Drag the material "**stereoMat**" (from **FOV2GO/Materials**/) into the **Stereo Material** field.
- For **Stereo Format**, select **Anaglyph**.
- Press **Play**. The Game View will now render in **anaglyph** (red/blue) mode.
- Stop the game. The scene will continue to be displayed in stereo 3D.
- Under **Options**, select a **Color Mode.**
- If there are saturated colors in your scene, **Full Color** will be difficult to view.
- **Monochrome** (black and white) always works, but **Half Color** and **Optimized** are usually fine.
- After you've changed any Render options, you may have to press Update (or press Play) to update the game view.

Section 3: Reference

Scripts/Core: Camera & Controller Scripts

s3dCamera.js

This script handles the creation of the stereoscopic 3D camera, as well as rendering in a number of different stereo formats. FOV2GO supports *side-by-side* format used with stereoscopes, 3DTVs and free-viewing, as well as *anaglyph* (red/blue) and a number of other viewing formats. To create a stereo 3D camera, just drag the **s3dCamera.js** script onto the **Main Camera** in your scene (it's already set up for you in this tutorial scene). Its default setting is *side-by-side* format, which will work in any version of Unity (anaglyph and other formats require Unity Pro).

Note: In most cases, you will want to have exactly **one s3dCamera** in your scene (and that will generally be the **Main Camera**). It is possible to have multiple **s3dCameras** in a scene (a technique known as *multi-rigging)*, but this is an advanced topic that is not fully documented at present.

The inspector is divided into two panels: **Stereo Parameters** and **Stereo Render.**

Stereo Parameters

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 stereoscope-based media. 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.

Toed In: This is an esoteric setting that will only be of interest to stereoscopic specialists. Its default setting is **off**, and it should be left that way. If interested, see http://www.binocularity.org/page3.php for an explanation.

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. See http://www.angelfire.com/ca/erker/freeview.html for details.

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. While a H I T shift can be very useful for stereoscope-based formats viewer), it should normally be set to 0 for screen-based formats such as anaglyph. 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.

Stereo Render

Stereo Shader (Unity Pro Only)**:** For **Unity Free**, this will need to be set to **Off** (unchecked), which allows use of the **Side by Side**format only. Other formats (notably **Anaglyph**) require **Unity Pro**, due to the fact that they make use of *render textures*.

Stereo Format:
The various available stereoscopic formats: **Side By Side**, **Anaglyph, Over Under**, **Interlace**, and **Checkerboard**. In **Unity Free**, the only available format is **Side by Side.**

Side By Side: Squeezed (checkbox):
Off: For stereoscopes and for free-viewing.
On: For outputting to a 3D TV that uses 50% horizontal (frame-compatible) format.

Side By Side: Use Phone Mask (checkbox) **(Unity Free** Only) :
For a stereoscope, this can be left **on** (checked). This allows you to render the left and right camera views to a specific portion of the screen that matches the optics of the the stereoscope, leaving the remainder of the screen black for interface elements such as dividers or touchpads. The layout is controlled by the next settings, **Left View Rect** and **Right View Rect.**

Side by Side: Left View Rect, Right View Rect (Unity Free Only)
These set the viewports for the left and right views in **Side By Side** mode. The four numbers represent in order, *x, y, width*, and *height*, calculated from the lower left-hand corner). The settings to fill the entire window are: Left: 0.0, 0.0, 0.5, 1.0, and Right: 0.5, 0.0, 0.5, 1.0. (These would be the settings for 3D TVs). You can set the these manually, or they can be controlled by the **s3dDeviceManager.js** script.

Anaglyph: Anaglyph Mode (Unity Pro Only)**:**
Monochrome: Black and white. Uses red channel for left eye, green and blue channels (cyan) for right eye.
Half Color: Half-saturation (muted colors).
Full Color: Full saturation.
Optimized: Colors optimized for anaglyph format.
Purple: Uses red channel for left eye, blue channel for right eye. Blue channel off.

Interlace: Rows (Unity Pro Only)**:**
Number of horizontal lines making up the stereo image (left and right images are each half of this number).

Checkerboard (Unity Pro Only):
Rows: Horizontal resolution of stereo image
Columns: Vertical resolution of stereo image

Stereo Material (Unity Pro Only)**:**
Drag the material **stereoMat** (from **FOV2GO/Materials/**) into the **Stereo Material** field.

Depth Plane:
A prefab that is used to display the near distance, zero parallax distance, and far distance. Accessed via s3dDepthInfo.js. Drag a prefab from the **Prefabs/DepthPlanes** folder into this field.

Update button: After you've changed any Render options, you may have to press this button to update the game view. If the scene still doesn't update, just press **Play** and then **Stop**, and everything should update to the current settings.

s3dDepthInfo.js

This script sends an array of raycasts into the view frustum and reports:
- the distance of the nearest object in the scene
- the distance to the object or surface at the center of the view
- the distance of the farthest object or surface in the scene
- the distance to the object or surface currently under the mouse position in the scene
- the distance to a selected object (set either by script or by mouse click) in the scene

Ray Columns: Number of columns (arrayed along the x axis) of rays. More is better, less is faster. If you want to make sure you are calculating the exact center of the view, pick an odd number.

Ray Rows: Number of rays (arrayed along the y axis) of rays. More is better, less is faster. If you want to make sure you are calculating the exact center of the view, pick an odd number.

Max Sample Distance: Distance to cast rays

Draw Debug Rays: Show visible representation of rays in the Scene view.

Show Screen Plane: Show visible representation of the screen plane (zero parallax distance)

Show Near/Far Planes: Show visible planes at the distance of the nearest point and at the distance of the farthest point in the scene.

Click Selects Object: When checked, a mouse click will report the current distance to that object.

Selected Object: When **Click Selects Object** is checked, the name of the selected object appears here.

Distances: Reports near, center and far distances, as well as distance to object/surface currently under mouse pointer, and distance to selected object.

s3dAutoDepth.js

Uses information from the **s3DdepthInfo.js** to dynamically set the **interaxial** and/or the **convergence** (zero parallax distance) to keep screen parallax within a specified percentage of the screen width. The interaxial can be clamped between a minimum and a maximum, and a minimum for zero parallax distance can be specified. Objects can be tagged **Ignore Raycast** to hide them from raycasts (this might be be done with ground planes). Convergence can be controlled by a number of strategies, the most useful of which are **percent** (set convergence by a ratio of negative to positive parallax) and **center** (keep convergence set to the center of the camera view).

Convergence Method
none: leave convergence (zero parallax distance) fixed at initial setting
percent: maintain ratio of negative to positive parallax as set by "percentageNegativeParallax".
center: keep convergence at center of scene
click: click mouse button to set zero parallax distance to point on object
mouse: zero parallax distance set continuously to mouse position
object: continuously keep zero parallax distance at clicked object point

Auto Interaxial: Control interaxial.
True: keep within "parallaxPercentageOfWidth".
False: leaves interaxial alone (control with s3Dcamera or other means)

Parallax Percentage: Automatically calculates interaxial based on a total parallax value expressed as a percentage of image width.

Negative/Positive Ratio: When **convergenceMethod** is set to **percent**, automatically calculates zero parallax distance (convergence) based on a negative parallax value expressed as a percentage of total parallax.

Min Zero Prlx Distance: Set zero parallax distance minimum

Minimum Interaxial: Limit minimum allowed interaxial; overrides parallaxPercentageOfWidth

Maximum Interaxial: Limit maximum allowed interaxial; overrides parallaxPercentageOfWidth

Lag Time: How gradually to change interaxial and zero parallax (bigger numbers are slower - more than 25 is very slow);

s3dFirstPersonController.js

This script provides the same functionality as the standard **FPSInputController.js** script (included in the standard **Character Controller** package). However, instead of using the standard **Horizontal** and **Vertical** axes provided by the **Input Manager**, this script uses a **s3dTouchpad** (called **touchpad_move** in this scene) for input. Vertical touchpad input moves the player forward and back, and horizontal touchpad movement moves the player left and right (default; see below)

The script is attached to a First Person Controller object (called **s3d_FirstPersonController**), which parents the **Main Camera** object. This object has a **Character Motor** script (added via **Component/Character/ Character Motor**) and a **Character Controller** component (added automatically when the **Character Motor** script is added).

Touchpad: Drag a **s3dTouchpad** from your scene into this field, or click on the Object Picker (the little target icon) and pick an available **s3dTouchpad**.

Touch Speed: Scales the touchpad's output. You can set different values for x and y, although normally these would be identical. Default is 1 1.

Horizontal Controls Heading: In the default **off** (unchecked) setting, horizontal input moves the player left and right. When checked, horizontal movement instead rotates the player; that is, it controls the camera heading. For FOV2GO applications, camera rotation (including heading) would normally controlled by the gyroscope, so this setting would be left unchecked.

s3dGyroCam.js

This script manages a gyroscope-controlled camera for the iOS and Android platforms. Just drag **s3dGyroCam.js** onto the **Main Camera** in your scene (along with the **s3dCamera.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.

s3dGyroCam.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 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: Manually changing the camera's heading would normally be controlled via a **s3dTouchpad**, but it can also 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). So when running the scene in the editor, you will be able to fully control the camera's rotation via the mouse.

Set Zero Heading To North: The only other inspector setting is **Set Zero Heading To North**, which will use 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: For **Unity 3.x**, **Default Orientation** should be set **Auto Rotation** for correct gyroscope operation, and **Check For Auto Rotation** should be checked. This is *not* necessary for **Unity 4.x**, therefore **Default Orientation** should be set to a single desired orientation (normally **Landscape Left**), and **Check For Auto Rotation** should be left unchecked.

s3dWindow.js

Provides dynamic masking on the left and right sides of the screen to prevent **stereo window violations.**

Active: Turns masking on or off.

Side Samples: Number of rays sampling the left and right edges of the image. More rays will be more accurate; less will be faster.

Mask Limit: Normally this should be set to the **Screen Plane,** since stereo window violations are only a problem with negative parallax (theatre space), but it can also be set to a fixed **Maximum Distance** (set with next parameter) or **Far Frustum** (far clipping plane).

Maximum Distance: Fixed distance for masking. Only operates when **Mask Limit** is set to **Maximum Distance**.

s3dRotateHeading.js

For tablet and phone-based applications, full lookaround capability is normally provided by the gyroscope. However, it is often convenient to be able to rotate the camera heading manually (for instance if the user is seated), and this functionality is provided by **s3dRotateHeading.js**, which accepts input from a **s3dTouchpad**.

Touchpad: Drag a **s3dTouchpad** from your scene into this field, or click on the Object Picker (the little target icon) and pick an available **s3dTouchpad**.

Touch Speed: Scales the touchpad's output. You can set different values for x and y, although normally these would be identical. Default is 1 1.

Control Pitch in Editor: Because the gyroscope is unavailable in the Unity Editor (and because the gyroscope is unsupported in Unity Remote), there is normally no convenient way to simulate full camera lookaround in the Editor. However, when **Control Pitch in Editor** is checked (default = on), full camera lookaround can be simulated using the assigned **s3dTouchpad.** Then, when run on the mobile device, **s3dRotateHeading.js** controls heading only (and pitch is controlled via the gyroscope).

s3dSmoothMouseLook.js

This is basically a javascript version of the **MouseLook.cs** script provided with Unity. You can replace the standard **MouseLook.cs** character controller script with this script. It provides smoother mouse movement, and by default is only active when mouse button is down. You can uncheck **Mouse Down Required** to make it always active, like the standard script. The script is only active on desktop, and is automatically disabled on iOS and Android, where lookaround is provided by the device gyroscope. Please note that this script conflicts with the **s3dTouchpad** system, because this script uses entire screen for input, so touchpad movement triggers it.

Scripts/Core: Other Scripts

s3dEnums.js

Contains all enumerations used in the Stereoskopix package.

s3dInteractor.js

Rules for interactive objects & scripts

These are the basic rules for creating interactive objects, and for writing additional **interaction** scripts.

- All interactive objects should be placed in the **interactive** layer (default = Layer 23)
- All interactive objects need to have **colliders** (**Box, Sphere, Capsule, Mesh**, etc)
- Any of the Interaction scripts included in the **FOV2GO/Scripts/interactions** folder can be added.
- Other interaction scripts can be written following the examples in the **FOV2GO/Scripts/interactions** folder.
- All **interaction** scripts should include the line: **@script RequireComponent(s3dInteractor)**. This line ensures that when the script is added to an object, **s3dInteractor.js** is automatically added as well.

- Some **interaction** scripts require a **RigidBody** component, which can be added automatically by including the line **@script RequireComponent(RigidBody)**
- Some **interaction** scripts require a **AudioSource** component, which can be added automatically by including the line **@script RequireComponent(AudioSource)**

Program flow for interactive objects

When an interactive object is rolled over or tapped on with the 3D Cursor:

- First, the 3D Cursor's **s3dGuiCursor.js** script does a **GetComponent** to find the object's **s3dInteractor.js** script.
- Next, the 3D Cursor's **s3dGuiCursor.js** script runs one of these functions in the object's **s3dInteractor.js** script:
    - **tapAction**()
    - **upDatePosition()**
    - **deactivateObject()**
    - **rolloverText()**

- The function (in the object's **s3dInteractor.js** script) does a **SendMessage** on the object
- The **SendMessage** is picked up by one or more **interaction** scripts attached to the object, running one of these functions contained in the interaction script.

    - **NewTap**()
    - **NewPosition**()
    - **Deactivate**()
    - **ShowText**()
    - **HideText**()

Most interaction scripts will contain (at minimum) a **NewTap** function that processes taps. The **NewTap** function accepts an argument (**params**) that includes the **RayCastHit** from the 3D Cursor's raycast (**params.hit**) and an **integer** (**params.tap**) that specifies whether the tap was short (1) or long (2). Most interaction scripts include a variable called **tapType** that can be set to respond to short taps (1) long taps (2) or any tap (3).

Interaction Script Example

Here's the basic structure of an interaction script.

```
@script RequireComponent(s3dInteractor);
var tapType : int = 3; // 1 = short tap 2 = long tap 3 = any tap

function NewTap(params: TapParams) {
   if (params.tap == tapType || tapType == 3) {
        // do something!
   }
}
```

You can examine any of the scripts in the **FOV2GO/Scripts/interactions** to see how various other functions are implemented.

s3dStereoParameters.js

s3dStereoParameters.js contains functions for adjusting the key stereo parameters (interaxial, zero parallax distance, and horizontal image transform). As always, you can run the scene in the Unity editor to test all functions. Try running the scene and clicking on the **Adjust Stereo Values** button. This will bring up three **s3dTouchpads** for adjusting stereo parameters. Click in the left half of each touchpad to reduce the value; click in the right half of the touchpad to increase the value. Click the **OK** button to dismiss the stereo parameter touchpads.

- **Save 3D Params To Disk:** Default **on** (checked): when checked, saves user settings for stereo parameters (**Interaxial**, **Zero Prlx** and **H I T**) to disk (to the PlayerPrefs file), and loads these updated settings next time app is run. When unchecked, settings will revert to the values that were set at build time.
- **Stereo Params Texture** and **Dismiss Params Texture**: these two fields are set to two Textures to be toggled depending on whether the three **s3dTouchpads** for Stereo Parameters (**Interaxial**, **Zero Prlx** and **H I T**) are visible or not.
- **Stereo Readout Text** (field for **s3dText**): set to **stereoParamsText**

If **Save 3D Params To Disk** is checked, the values set by these three **s3dTouchpads** will be saved to disk (via PlayerPrefs) when this app is built and run on a mobile device (and then will be loaded from disk next time the app is run).

s3dTouchpad.js

s3dTouchpad.js creates a virtual touchpad from a **GUI Textur**e that can be used to control an element in the virtual world, such as the camera or an object in the world. It responds just like a real touchpad to taps and swipes, and when used in the Unity editor, it accepts mouse clicks and drags so that you can simulate touchpad behavior on a mobile device.

FOV2GO viewers are designed to be open at the bottom, so that the touchscreen is still accessible via the user's thumbs. The lenses are placed slightly higher than the center of the screen, leaving the lower portion of the screen available for thumb-based user input. For this purpose, one or more virtual **touchpads** are created. These function just like the physical touchpad on a laptop. The virtual touchpads are placed just outside the visual field, and can be used to manipulate elements inside the visual field. In this sense, they are more like the standard interface devices used on a laptop or desktop system. With a touchscreen, the display screen and the input device are one and the same, but with both desktop and virtual reality systems, the display screen is of necessity discrete from the input system.

s3dTouchpad.js is based on the **Joystick.js** script included with Unity Penelope iOS Tutorial, but it's been substantially stripped down and rewritten to provide the specific functionality required by **FOV2GO**. Unlike **Joystick.js**, which can be set to either mimic a joystick or a touchpad, **s3dTouchpad.js** can combine elements of either (touchpad behavior with optional joystick visual movement, or joystick behavior without visual movement). **Joystick.js** handles multitouch, but because we never need to handle more than one touch per touchpad, all that complexity has been stripped away. Also, **s3dTouchpad.js** only deals with single taps (but it can distinguish between long and short taps when desired).

Unlike many other elements in the **FOV2GO** system, which automatically clone themselves to provide left and right eye elements, **s3dTouchpads** are singular - they appear in one location on the screen. When used with an

FOV2GO viewer, they are designed to be rendered outside of the visual field. For other 3D formats, such as anaglyph, they can appear in or out of the camera view.

Move Like Joystick: When checked, the **GUI Texture** will follow touch and mouse drag movements, snapping back to the origin when the touch is lifted. When unchecked, the GUI Texture doesn't move at all. This has no effect on the touchpad's behavior.

Act Like Joystick: When checked, the touchpad x and y values will jump to the initial touch/click position (in the range -1.0 to 1.0), and then change *relative to this value* when dragged. When unchecked, the touchpad x and y values will start at 0.0 (no matter where the initial touch/click occured) and then change *relative to the initial position* when dragged.

ShortTapTimeMax: This sets the maximum amount of time for a short tap - if the touch lasts longer than this, it's considered a long tap. The default setting is 0.2 seconds.

LongTapTimeMax: This sets the maximum amount of time for a long tap - if the touch lasts longer than this, it's not considered a tap. The default setting is 1.0 seconds.

Tap Distance Limit: This sets the maximum amount of movement for a tap - if the touch moves more than this, it's no longer considered a tap. The default setting is 10 pixels.

Creating a new s3d Touchpad from scratch

Here are the steps you would need to create a new **s3dTouchpad** (for this scene, the two **s3dTouchpads** are already set up):

1. Create a **GUI Texture** game object.
2. Drag a **s3dTouchpad.js** script onto the object, or onto its Inspector.
3. Drag a texture into the **GUI Texture** Inspector's **Texture** field.
4. Set the object's **Layer** to **guiOnly.**
5. Set all fields in the **Transform** of the object (including **Scale**) to 0. (You can change the **Position X** and **Y** fields later, but for now they should both be set to 0).
6. Set the **GUITexture**'s **Pixel Inset** to **X** = 0, **Y** = 0, **Width** = desired width in pixels, **Height** = desired height in pixels.
7. The object should now appear in the lower lefthand corner of the game view.
8. Since the origin for **GUI Texture** coordinates is the lower left corner, setting **Transform Position X** = 0 and **Y** = 0 puts the lower left hand corner of the **GUI Texture** right at the origin. Now you can reposition the **GUI Texture** anywhere in relation to the lower left hand corner by changing the **Pixel Inset X** and **Y** fields to positive values.
9. However, what happens if you want to position the **GUI Texture** in relation to one of the other corners? Unless you know the exact pixel dimensions of your device, it's impossible to do this using the **Pixel Inset** fields.
10. Let's say you want to set the **GUI Texture** in relation to the upper right hand corner.
11. First, set **Transform Position X** = 1 and **Y** = 1. This places the lower left hand corner of the **GUI Texture** in the upper right hand corner of the display - which unfortunately means it's entirely out of view.
12. To fix this, set the **Pixel Inset X** and **Y** fields to the **Width** and **Height** values multiplied by –1. (For example: **X** = –320 **Y** = –240 **Width** = 320 **Height** = 240).
13. Your **GUI Texture** should now appear in the upper right hand corner of the display.

14. Finally, to set a **GUI Texture** in relation to the center of the display, just set **Transform Position X** and/or **Y** to 0.5, and set the **Pixel Inset X** and/or **Y** to –1/2 the **Width** and/or **Height** values.

s3dGuiCursor.js

Since a **GUI Texture** is a 2D element that is rendered on top of camera view, it does not automatically support stereoscopic imaging, for which each element has to be assigned a position in stereoscopic depth - an operation that normally requires two copies of the element that can be positioned in a precise relationship to each other.

s3dGuiTexture.js is a versatile script that can be added to a **GUI Texture** to create left and right copies of the texture so that it can be used as a stereoscopic element. It can adjust the **GUI Texture**'s parallax automatically to keep the element closer than anything it occludes (this is necessary because the **GUI Texture** will always be rendered on top of the camera view - and if it were to appear to be stereoscopically behind any element in the view, a depth paradox would result).

s3dGuiTexture objects are automatically placed in **leftOnly** Layer and **rightOnly** Layer (which you should have named when you installed the **FOV2GO** package, above. These layers are set in **s3dCamera.js**, and the defaults are Layer 20 for **leftOnly** and Layer 21 for **rightOnly**.

Begin Visible: Default **on** (checked). If unchecked, can be toggled by calling **toggleVisible**() function.

Time To Display (sec): Default 0 = leave on forever. If set to any other value, will disappear after specified interval.

Initial Object Distance (M): Default 1 meter. Only has effect if **Track Depth** is left unchecked.

Track Depth: Default **on** (checked) Automatically updates stereo depth (parallax) to keep texture closer than scene objects behind it.

Min Distance/Max Distance: Defaults 0.01 and 100 meters. Minimum and maximum allowed stereoscopic depth.

Near Padding (mm): Default 0. Bring forward slightly so that texture is that much closer than objects directly behind it.

Smooth Depth Changes: Default 0. Change depth more gradually (reasonable values are up to 25).

s3dGuiText.js

rolloverText.js, which has been added to each of the interactive objects, makes use of a **GUI Text** object that has an attached **s3dGuiText.js** script. **s3dGuiText.js** is very similar to **s3dGuiTexture.js** - it can be added to a **GUI Text** to create left and right copies of the text so that it can be used as a stereoscopic element. Like **s3dGuiTexture.js**, it can adjust the **GUI Text**'s parallax automatically to keep the element closer than anything it occludes.

Text

Initial Text: Enter the text that should be displayed when the scene starts.

Begin Visible: Default **on** (checked). If unchecked, can be toggled by calling **toggleVisible**() function.

Time To Display (sec): Default 0 = leave on forever. If set to any other value, will disappear after specified interval.

Style

Text Color: Default White. Set the text color

Shadow: Default **on** (checked). Creates a second copy of the text slightly offset from the first, typically set to a darker color so that it appears to be the shadow of the text. Increases legibility.

Shadow Color: Default Black. Set the text shadow color.

Shadow Offset: Default 5. Determines how far (to the right and down) the text shadow should appear.

<u>Distance</u>

Initial Object Distance (M): Default 1 meter. Only has effect if **Track Depth** is left unchecked.

Track Depth: Default **on** (checked) Automatically updates stereo depth (parallax) to keep texture closer than scene objects behind it.

Min Distance/Max Distance: Defaults 0.01 and 100 meters. Minimum and maximum allowed stereoscopic depth.

Near Padding (mm): Default 0. Bring forward slightly so that texture is that much closer than objects directly behind it.

Smooth Depth Changes: Default 0. Change depth more gradually (reasonable values are up to 25).

Track Mouse Position: Default **off** (unchecked). When **on,** track mouse position on desktop.

Track Only When Down: Default **on** (checked). When **off** (unchecked), text will follow mouse position at all times.

s3dGuiTexture.js

Since a **GUI Texture** is a 2D element that is rendered on top of camera view, it does not automatically support stereoscopic imaging, for which each element has to be assigned a position in stereoscopic depth - an operation that normally requires two copies of the element that can be positioned in a precise relationship to each other.

s3dGuiTexture.js is a versatile script that can be added to a **GUI Texture** to create left and right copies of the texture so that it can be used as a stereoscopic element. It can adjust the **GUI Texture**'s parallax automatically to keep the element closer than anything it occludes (this is necessary because the **GUI Texture** will always be rendered on top of the camera view - and if it were to appear to be stereoscopically behind any element in the view, a depth paradox would result).

s3dGuiTexture objects are automatically placed in **leftOnly** Layer and **rightOnly** Layer (which you should have named when you installed the **FOV2GO** package, above. These layers are set in **s3dCamera.js**, and the defaults are Layer 20 for **leftOnly** and Layer 21 for **rightOnly**.

Begin Visible: Default **on** (checked). If unchecked, can be toggled by calling **toggleVisible**() function.

Time To Display (sec): Default 0 = leave on forever. If set to any other value, will disappear after specified interval.

Initial Object Distance (M): Default 1 meter. Only has effect if **Track Depth** is left unchecked.

Track Depth: Default **on** (checked) Automatically updates stereo depth (parallax) to keep texture closer than scene objects behind it.

Min Distance/Max Distance: Defaults 0.01 and 100 meters. Minimum and maximum allowed stereoscopic depth.

Near Padding (mm): Default 0. Bring forward slightly so that texture is that much closer than objects directly behind it.

Smooth Depth Changes: Default 0. Change depth more gradually (reasonable values are up to 25).

Scripts: Mobile

s3dDeviceManager.js

This script handles the layout for **FOV2GO** viewer apps. In this first scene, it is used only to set the left and right camera **view rectangles** to the correct positions for a specified mobile device. All other options are unchecked in this scene, and will be described as we make use of them in further tutorial scenes.

Pick your phone or tablet in the **Phone Layout** pulldown menu. If your device isn't listed, pick the setting that is closest to your device (alternately, you can disable **s3dDeviceManager.js**, and manually change any of the settings on the various game objects themselves).

Note: Unless the Game View is sized to the exact pixel dimensions of your device screen, the layout of your scene may not match the layout on your device, especially regarding camera view rectangles and GUI elements. Some devices are listed in the Game View Aspect drop-down menu, and if your device is listed, you should select it there. If your device isn't listed, you should pick the one that's closest to it in pixel dimensions. But for certain devices, matching the Game View to the device screen resolution is going to be next to impossible - for example, the iPad 3 (with Retina display) will be too large (2048 x 1536) to fit on most laptop or desktop screens. Since the iPad 2's pixel dimensions (1024 x 768) are exactly 1/2 of the iPad 3's, you can set the Device Manager to iPad 2 to test in the editor - but don't forget to switch it back to iPad 3 before you build.

Supported Devices
Here are the specs for the devices that are currently listed in the s3d Device Manager:

- Apple iPad 2
  - resolution: 768 x 1024 pixels
  - screen dimensions: 149 x 198.1mm
- Apple iPad 3
  - resolution: 1536 x 2048 pixels
  - screen dimensions: 149 x 198.1mm
- Apple iPhone 4/4S
  - resolution: 640 x 960 pixels
  - screen dimensions: 49.2 x 74.88mm
- HTC One S
  - screen resolution: 540 x 960 pixels
  - screen dimensions: 53mm x 95mm
- HTC Rezound
  - screen resolution: 720 x 1280 pixels
  - screen dimensions: 53 x 95mm
- LG Thrill 4G
  - screen resolution: 480 x 800 pixels
  - screen dimensions: 56 x 93mm
- Samsung Galaxy Nexus
  - resolution: 720 x 1280 pixels
  - screen dimensions: 58 x 96mm
- Samsung Galaxy Note
  - screen resolution: 800 x 1280 pixels
  - screen dimensions: 71 x 113.5mm

Note: The iPad has settings for both **portrait** (vertical) and **landscape** (horizontal) configurations - pick one or the other depending on which **FOV2GO** viewer you are using.

s3dDeviceManager.js has a Phone Layout setting called **My3D_LandLeft,** which will select the correct screen layout for the **my3D** viewer.

androidDeviceSelector.js & iosDeviceSelector.js

Included in the **FOV2GO** package are two scripts - one for Android devices (**androidDeviceSelector.js**) and the other for iOS devices (**iosDeviceSelector.js**)- that can automatically update the screen layout based on what device the app is running on. This allows you to build a single app that can run on all supported devices on a given platform.

On startup, these scripts check the screen resolution of the device - and then, if that resolution matches one of the presets contained in **s3dDeviceManager.js** - the display will automatically switch to that screen layout.

Just add the appropriate script (depending on your build platform) to the **s3d_Device_Manager** object (that is, the same object that has the **s3dDeviceManager.js** script attached) and the screen layout will be selected automatically when the app runs on the device.

Scripts: Other

Scripts/ButtonBehaviors

triggerObjectButton.js

Used with **s3dTouchpads.**

Set **The Object** to the object that you want to control

ToggleTapType can be set to 0 (ignore) 1 (short tap), 2 (long tap) or 3 (any tap). The specified tap will toggle the object (and any children) on and off. **ActionTapType** can also be set to ignore, short, long or any tap, and will trigger a function called **Go()** in any script attached to the receiving object.

triggerSceneChange.js

When clicked/tapped, the next scene (in Build Settings) will be loaded. **TapType** can be set to 1 (short tap), 2 (long tap) or 3 (any tap). **Loading Text** should be set to a **s3dText** object (in this example it's set to **loadingText**). **Loading Message** (default "Loading…") determines the message that will be displayed just before the next level is loaded. **Pause Before Level** (default = 1 second) introduces a short delay before the next level is loaded for **Loading Text** to be displayed.

Scripts/Interactions

See Tutorial 4.

centerToCamera.js

fireGun.js

jumpInTheAir.js

pickUpPutDown.js

pivotObject.js

playSoundSimple.js

pointAtPoint.js

pointAtPoint.js is an interaction script (requiring an **s3dInteractor.js** script). When **updatePosition** is triggered in the **s3dInteractor.js** script, it runs a **SendMessage** that triggers the **NewPosition** function in **pointAtPoint.js,** aiming the object at the current position of the 3D Cursor.

pushAway.js

rolloverText.js

rotateAndSpin.js

toggleLighting.js

toggleText.js

Scripts/Miscellaneous

aimObject.js
Used with 3D Cursor. **aimObject.js.** takes one parameter, called **Object To Notify**, which is set to the object to be aimed at the 3D Cursor's position.

playCollisionAudio.js

triggerSoundByProximity.js

Editor Scripts

s3dCameraEditor.js
s3dDepthInfoEditor.js
s3dDeviceManagerEditor.js
s3dGuiCursorEditor.js
s3dGuiTextEditor.js
s3dGuiTextureEditor.js
s3dGyroCamEditor.js
s3dStereoParametersEditor.js
s3dTouchpadEditor.js
s3dWindowEditor.js