VR3DMediaViewer

Manual

V1.0.1

for Unity 5.6+

Contents:

1. Quick "How to" for known devices/SDKs

The setup process to use this asset can be broken down into 3 major steps:

- **Camera Setup**
- **Image/Video Setup**
- **Canvas Setup**

General Preparation:

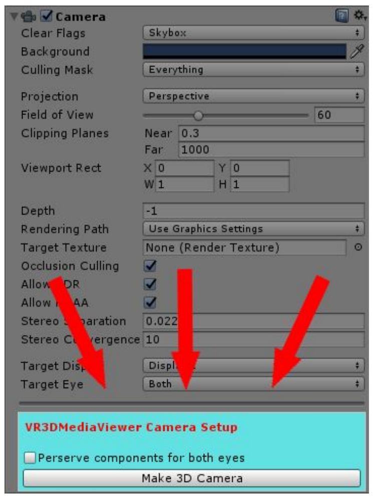
Some SDKs may require it, but not automatically enable the following. So you may need to do it yourself.

Edit > Project Settings > Player > Other Settings > Virtual Reality Supported

Camera Setup:

1. Make sure you have the SDK for your target VR Device imported, and add its Camera Prefab to your scene.

2. Locate the Camera **component** in the prefab/MainCamera object in your scene, locate the "VR3DMediaViewer Camera Setup" area on the camera, and select "Make 3D Camera". (Alternatively, you can Right click on the Camera **component** and select "VR3DMediaViewer Camera Setup".)



✓ Camera	
Clear Flags	Skybox
Background	
Culling Mask	Everything
Projection	Perspective
Field of View	60
Clipping Planes	Near 0.3
	Far 1000
Viewport Rect	X 0 Y 0
	W 1 H 1
Depth	-1
Rendering Path	Use Graphics Settings
Target Texture	None (Render Texture)
Occlusion Culling	✓
Allow HDR	✓
Allow MSAA	✓
Stereo Separation	0.022
Stereo Convergence	10
Target Display	Display 1
Target Eye	Both

VR3DMediaViewer Camera Setup

☐ Perserve components for both eyes

Figure A.

For reference, these are the hierarchy paths to the Camera object in common VR SDK prefabs at this time.

OculusVR SDK Camera:

OVRCameraRig/TrackingSpace/CenterEyeAnchor

SteamVR SDK Camera:

[CameraRig]/Camera (head)/Camera (eye)

Daydream/Cardboard Camera (Post Unity v5.6):

Just add a normal camera to the scene.

Daydream/Cardboard Camera (Pre Unity v5.6/Legacy):

Add a normal camera to the scene.

Extra steps:

1. Create a new GameObject, drag the "****-Left" and "****-Right" camera objects under it and zero their positions/rotations.

2. Add a "StereoController.cs" from the GoogleVR SDK folders to it.

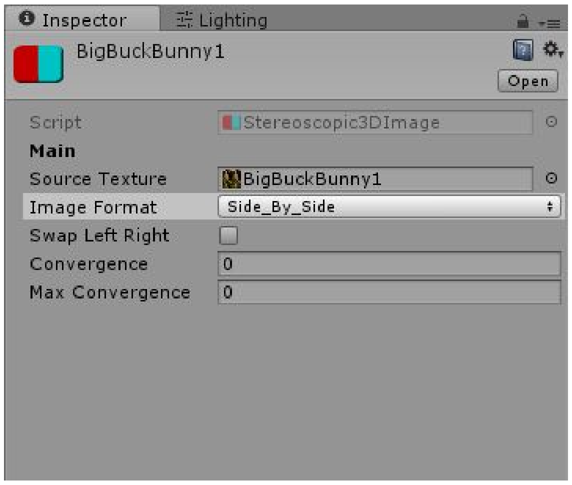
3. Add an "GvrEye.cs" from the GoogleVR SDK folders to each of the "****-Left" and "****-Right" camera objects, and on each GvrEye component select its corresponding eye and opposite eye Layer.

Image/Video Setup:

1. Import a 3D image or video. Once imported this will be referred to as a Texture or Video.

2. Locate the Texture(s) or Video you want to be displayed in 3D in the Project panel, Right Click on it and select "**Create > Stereoscopic 3D Image**".

3. Select the newly created file and choose the correct **Image Format** for the Texture/Video. I.E. If the content of the Texture has 2 slightly different images Side-by-Side like L R, it's of the "Side-by-Side" format.



BigBuckBunny1

Script	Stereoscopic3DImage
Main	
Source Texture	BigBuckBunny1
Image Format	Side_By_Side
Swap Left Right	☐
Convergence	0
Max Convergence	0

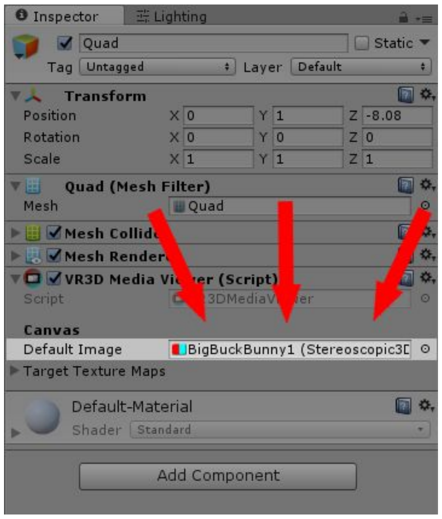
Figure B.

Canvas Setup:

1. In the Hierarchy panel of the current scene, create or select a GameObject with a MeshRenderer (and a single Material), or a UI.RawImage component. If you're uncertain about this step, just Create a "Quad" for now and select that. This is our Canvas.

2. With the Canvas GameObject selected, in the Inspector panel add a component called "VR3DMediaViewer".

3. In the "Default Image" field of the "VR3DMediaViewer" component, select the Stereoscopic 3D Image asset you created in #2-#3 of the "Image/Video Setup".



```
Quad                                  Static
Tag Untagged            Layer Default
Transform
Position    X 0    Y 1    Z -8.08
Rotation    X 0    Y 0    Z 0
Scale       X 1    Y 1    Z 1
Quad (Mesh Filter)
Mesh                 Quad
Mesh Collid
Mesh Render
VR3D Media Vi   er (Script)
Script               3DMediaV   er
Canvas
Default Image        BigBuckBunny1 (Stereoscopic3D
Target Texture Maps
Default-Material
Shader   Standard
```

Figure C.

Other Devices/SDKs:

This asset can't predict the existence of every possible VR device or SDK. But it should work with most that I haven't been able to list in this Manual.

2. Demos

Several Demo scenes have been provided with this Asset.

All demos have been created with a purpose of conveying this script on a standard 2D monitor, using a custom demo Camera Rig. Different VR HMDs use their own SDK's and Camera Rigs, which of course I can't include. So all I can do is provide my own as an example to help illustrate the idea.

If you want to try using the demo scenes with your VR HMD/SDK, refer to the "How to" section above to get the gist of how to set things up for that. There's a blank demo scene provided just for that purpose.

BONUS!!!

Stereoscopic 3D Screenshot Demos:

General Overview:

There are 2 provided demo scenes for the Stereoscopic3DScreenshot script.

The first, original scene is designed for developer usage. It's an example for developers to set up a scene and take a screenshot in Editor for use with their projects various needs.

The second, newer scene is designed to illustrate how to allow use of the script out of the editor by normal users at runtime.

Both scenes use an Example3DScreenshotController script. This is just an example implementation of how to access the main Stereoscopic3DScreenshot script to produce a screenshot with it. Though the first scene uses the controller in a very minimal fashion, the second scene shows greater usage!

Naturally, such a controllers primary goal should be to call `Stereoscopic3DScreenshot.TakeScreenshot();` based on someone intentionally triggering it to do so, in this case via Key/Mouse button.

Helpful Advice:

Generally you should only take screenshots using the Side-by-Side, Top/Bottom or Two Images formats. Avoid the Interlaced/Checkerboard formats for typical use (those are just available for the sake of completeness).

You can control the full/squashed size of the output image, but just remember. You can always squash a full image later in a paint program, but can't "un-squash" an image, without quality loss. Top/Bottom+Full is recommended.

Generally you should leave the projection setting set to "OffCenter". The other 2 have little to no benefits but do have drawbacks (again, I just included them for the sake of completeness).

While the Example3DScreenshotController script provides a framing/depth plane for use in game, in Editor there is one too, in the Scene view. These planes represent the resulting images frame and point in the scene where your focus will match the border/frame of the resulting image. Objects that are in between the focal plane and the camera will appear to "pop out" of the image when viewed as 3D.

Learn from the example script for your own implementation, or straight up use it with some customizations. Don't choose to just not provide players any control over framing/focal depth at all. Take pride in your game and allow players to take as beautiful of screenshots as can be done!

If you want to provide your users simple instructions on how to take **High Quality** 3D screenshots, like via an in game tooltip, you could describe it as such:

Think about the resulting 3D image as like an open window.

Generally, most of the content inside the 4 edges of the image should appear as if behind the window and its frame. Anything in front of the window will appear to viewers as if it's in front of the image itself, or "popping out". The trick in most cases is to only have such content pop out if it is not in contact with the edges of the picture. Such contact can ruin the effect, make it harder to focus on, and makes it seem as if the popping out content is cut off unnaturally.

Also consider that past a short distance, scenery shows little to no difference for a 3D effect, so try to focus on content nearby.

Or course, use your own words as needed.

3. Additional Notes

General:

Reflection Probes will only reflect the image from one eye. Thankfully most 3D images aren't very different per eye, so viewers won't notice the difference in the reflection as much, if at all.

The Stereoscopic 3D image/video ***can*** use transparency.

You can use any shader you want for the canvas. Standard, Unlit, Custom, etc. The exception being the panoramic CanvaFormats.

While not officially supported, this can be used with Stereograms by adjusting the convergence values.

Video:

Typically with Panoramic 3D video you will want to use a higher resolution video. Keep in mind that on Windows 7 there's a limit of 1920x1088 resolution thanks to Unity using Media Foundation to transcode videos for it's VideoClip format. On Windows 10 the limit is higher, and you may not notice an issue, but your Windows 7 users will. If you transcode the video (outside of Unity) to .webm, and import it you can get around the limits.

Panorama:

For Panoramic media, you have 2 types of canvas format options for display. A Standard canvas and custom canvas format, depending on the if its 360 or 180.

Standard:

When using this, the Mesh objects shape is what's most important to wrapping the content around the viewer. The texture appears on the surface of the mesh, so in order for the texture to be displayed around the viewer as a sphere/dome, the Mesh object itself must be a sphere/dome. Example sphere/dome mesh models are provided in the "VR3DMediaViewer/Demo/Models" folder.

360:

This uses a shader to wrap the texture around the viewer in 360. The Mesh object used for the canvas is more forgiving as the texture is not projected onto the meshes surface, but instead is more that the 360 media is like a skybox and the meshes surface is simply a window that lets you see that skybox. For example, you can use a cube with inverted normals, and it will still wrap around the viewer like a sphere.

You can rotate the media in the canvas by the Rotation option.

180:

This is like 360, except for 180 Panoramic content. Of note, the texture “Wrap Mode” determines how the other 180 degrees of the canvas is drawn. All black for “Clamp”, or tiled a second time for “Repeat”. This can be accessed in the textures Import Settings, or via the override option in the Stereoscopic3DImage object.

Why would you want to choose one or the other?

It of course depends on your projects needs. But a Standard Mesh Sphere/Hemi-Sphere canvas is in a fixed position. When a viewer moves forward/back, they move closer to/further from the texture. Such movement can also break the perspective effect as the viewers camera is meant to remain stationary relative to the content. The 360/180 canvas types are more like a skybox. They appear to be at the same distance regardless of the viewers camera position, provided the viewer has not passed/clipped through the canvas object itself.

Say you have no other content in the scene. You are just making a image/video viewer app. The 360/180 canvas format may be best for your needs.

If you have a physical gallery scene, somewhere your player controls an avatar and walks around. Then for this a Standard Mesh Sphere/Hemi-Sphere may make the most sense. To them it would be like being inside a room with the media projected on the walls.

You may also want to combine the 360/180 canvas with a flat quad type of canvas normally used for a 2D picture/video, for a neat effect. The effect would be similar to looking through a window. It’s kind of like using traditional non-Panoramic 3D media, but getting a better sense of perspective due to a wider FOV image behind the frame.

4. Stereoscopic 3D formats and standards

Through my research of the various stereoscopic 3D formats that I've tried to support with this asset, I've come to the conclusion that... there just aren't any "Standards".

What I mean by this is, for example, to some people Top/Bottom images have the image meant for the left eye on top, for others the Right eye is on top.

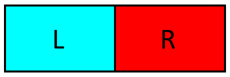
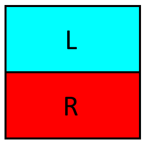
Even the terms used for the formats don't have a single standard. Top/Bottom can also be called Over/Under, Above/Below, etc.

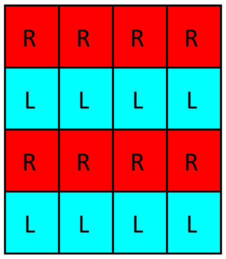
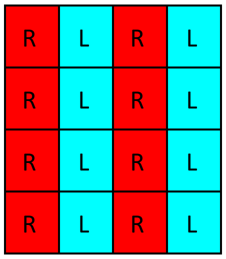
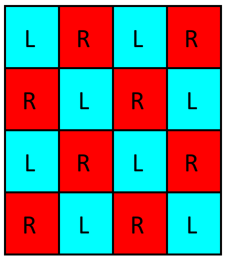
Ideally I would support the "Standards" as the defaults, and require swapping the left and right eye images only for Cross-eyed SBS images and fringe cases. But that's just not how things are.

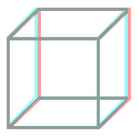
So I'm going to do my best to support the common implementations of the standards as I can see, based off of images/videos I find online.

Because of that, all I can do is declare how this asset supports Stereoscopic 3D as "My Standards", or "VR3DMediaViewer Standards".

For reference, here are the standards this asset acknowledges as follows:

Format	Variations	Representation	Notes
Side-by-Side (A.K.A. SBS, L/R)	Squashed/Half Size Full* Cross-eyed		This supports Full* or Squashed*. This supports Cross-eyed by use of the Swap Left/Right option.
Top-Bottom (A.K.A. Over-Under, Above-Below)	Squashed/Half Size* Full*		This supports Full* or Squashed*. Use the Swap Left/Right option if your image has the right on top.
Two Images	N/A	N/A	This is simply using a different image file/texture for each eye.

Horizontal Interlaced (A.K.A. Interlaced, Row Interlaced)	Squashed/Half Size* Full*		This supports Full* or Squashed*, though generally such images are squashed. All odd rows are for the right image, even rows are for the left. Internally, this converts to squashed Top-Bottom as described above. Use the Swap Left/Right option if your image has the left as odd/right as even.
Vertical Interlaced (A.K.A. Column Interlaced)	Squashed/Half Size* Full*		This supports Full* or Squashed*, though generally such images are squashed. All odd columns are for the right image, even columns are for the left. Internally, this converts to squashed Side-by-Side as described above. Use the Swap Left/Right option if your image has the left as odd/right as even.
Checkerboard (A.K.A. Tiled)	Squashed/Half Size* Full*		This supports Full* or Squashed*, though generally such images are squashed. All odd pixels on odd rows are for the left image, even pixels are for the right. All even pixels on odd rows are for the left, odd pixels are for the right. Internally, this converts to squashed Side-by-Side as described above. Use the Swap Left/Right option if your image has the right as odd/left as even.

Anaglyph (A.K.A. Red/Cyan, Red/Blue, R/B, etc)	Red/Cyan Magenta/Green Yellow/Blue		Both images exist in the same space. This displays a very different image to each eye with a similar effect to wearing colored glasses, but only having the image tinted instead of the whole world. Internally, this converts to squashed Side-by-Side as described above. There is an option for trying to show each image as monochrome, but it has to be enabled by editing the script.
Frame Sequential (A.K.A. Alternating Frames)	N/A	N/A	This is just listed for completeness. It's **UNSUPPORTED**! This is a video only format. Proper, full video support in this case would be nice, but it is just to hard to actively support for an asset like this. If you want to use 3D video, it's recommended to use SBS formatted 3D video.

*** Full or Squashed/Half-sized means, if both images are full resolution, or cut in half so both fit into a full resolution ratio. Example 3840x1080 = Full, 1920x1080 = Squashed/Half sized.**

5. Scripting

The main scripts in this asset use the "VR3D" namespace.

To access these methods and variables, naturally you will first need to get a reference to the component in question via your own script. Something like:

Code:

```
VR3D.VR3DMediaViewer myCanvas = gameObject.GetComponent<VR3D.VR3DMediaViewer>();
```

VR3DMediaViewer.cs:

SetNewImage();

You can call this at any time after the scripts Start() method has finished, to display a new image in the canvas.

Generally you should use the "Stereoscopic3DImage" overload so you can supply how the image should be formatted, but overloads for just a Texture/VideoClip/URL exist too. Using those uses the current 3D settings with the new image.

SetImageFormat();

You can call this to change the current images format to something new.

SetCanvasFormat();

You can call this to change the current images canvas format to something new.

The following are public assessors you can set directly and see an immediate change:

- SwapLeftRight;
- ConvergenceMode;
- Convergence; / MaxConvergence;
- Rotation; (Used with Panoramic canvas formats.)
- VerticalFlip;

- LeftEyeColor; / RightEyeColor;

Public read only assessors:

- currentImage;
- ImageFormat;
- CanvasFormat;

Events:

NewImageLoaded(Stereoscopic3DImage newImage);

Subscribe to this to know when a new image is loaded. (The "defaultImage" being loaded will trigger this too.)

FormatChanged(ImageFormat imageFormat);

Subscribe to this to know when the format for the currentImage has changed. This will also fire when an image is loaded.

CanvasFormatChanged(CanvasFormat canvasFormat);

Subscribe to this to know when the canvas format for the currentImage has changed.

Stereoscopic3DScreenshot.cs:

GetScreenshot(Texture2D texture, Texture2D texture2 = null);

Call this to take a screenshot.

Screenshots are returned as a Texture2D (or 2 in the case of the TwoImages format). To save a screenshot from a texture to disc, you can refer to the Example3DScreenshotController.

Get3DTextures(Texture2D texture, Texture2D texture2 = null);

When Interlaced/CheckerBoard/Anaglyph is set, this returns the screenshot texture(s) in a more RAW related format.

Notes:

You can view an example of setting images to a canvas via script in the demo script named "ImageCycler".

6. FAQ

Issue:

My image looks more pixelated or blurry then I would expect, since it's of a "High Resolution".

Answer:

This is likely because Stereoscopic 3D images tend to be in monitor resolutions (I.E. 1920x1080), and as such commonly exceed the default size that Unity imports them as. When that happens Unity lowers the resolution to more common texture resolutions.

So after importing a image, select it and ensure it's of the size that you want. If it is not, try turning up the max size past the larger of the base images Width/Height.

Also Unity by default rounds a textures resolution to the nearest "Power of 2". So you may want to set the textures "Advanced > Non Power of 2" property to "None".

Issue:

I create my texture/video at runtime (perhaps like by downloading it from a WebHost), but the setup seems geared only towards pre-existing textures/video. Can I still use this?

Answer:

Yes! Though naturally you would need to use scripting. Refer to the "**6. Scripting**" section of this manual for more. Of note though is that dynamically created textures/video wont come attached with 3D formating data like "Is it Side-by-Side, Top-Bottom, etc?". You will need to figure out and set this info on your own.

Issue:

I can tell my 3D Image/Video is displaying differently than a 2D image. But it's an underwhelming or difficult to look at effect. Something seems off about it.

Answer:

A common cause of this is the Stereoscopic 3D Image object needing its "**SwapLeftRight**" property changed. Some images may be incorrectly labeled from what you would expect, or were meant for what is known as "Cross-eyed" viewing.

For example, the Demo Scenes included with this asset are meant for Cross-eyed viewing, as the demos need to convey the purpose of the asset to people using a 2D monitor, and the Cross-eye technique can be done with a 2D monitor by anyone.

This is just something you will have to test and set for each image yourself.

Issue:

I have to strain to look at my image. It's as if what I'm trying to look at in the image is REALLY close to my eyes and I just can't converge my eyes together enough to focus on it.

Answer:

Sometimes 3D images were taken incorrectly, and the center of each eye's image is too far apart. It's up to the artists flavor, but generally the bulk of a 3D images content should appear as if its behind the borders/frame of the image, with maybe 1 or so foreground objects in the center jumping out as an exception. Images that cause a strain such as this, don't appear behind the border but instead in front.

I've added an option to the Stereoscopic 3D Image asset called "Convergence". Using this you can still use such incorrectly taken 3D images by sacrificing some pixels from the width of the image. How many pixels you need to sacrifice with "Max Convergence", and what "Convergence" value you need is something you will have to play with and figure out yourself per image.

Issue:

Interlaced/Checkerboard media looks like it isn't working. Or maybe made worse by being more blocky/pixelated.

Answer:

While I can't dismiss the possibility of a bug, this is likely not the case. What is probably the reason is that the media you are trying to display has experienced some data loss due to compression. The JPG format for example is not a "loss-less" format, and is particularly terrible for interlaced/checkerboard 3D.

Likewise, videos uploaded to common websites like YouTube, get recompressed and as such the alternating interlacing patterns get disrupted in the final video.

However in the case of images, Unitys own default compression and settings can distort the image enough to disrupt the interlacing pattern. If that's the case, you can try this…

Change the following texture import settings to:

Non Power of 2:	None
Generate Mip Maps:	false
Filter Mode:	Point Filter
Compression:	None

Issue:

Videos don't play when using the Unity 5.6 VideoPlayer. No texture is ever set to the canvases Mesh Renderer material.

Answer:

Try selecting the video clip file and selecting H264 for a Codec. This has fixed such an issue several times for me. This may get better over time as the VideoPlayer component is improved.

Issue:

There’s a really obvious jolting vertical seam/line in my 360 video. I don’t see such a seam/line when I view the 360 video in another app. I’m using Unity’s VideoPlayer/MovieTexture for my video.

Answer:

This unfortunately can happen due to the way Unity transcodes and stores videos. You may open the directory in your project and see the video look normal and unchanged from the original copy, but Unity likes to create and store an altered copy elsewhere. It’s that copy that you are seeing at runtime. You may have to play around with the Import Settings for the video, and/or manually convert it yourself in some instances to reduce loss in quality.

Why this happens with the transcoding process, I believe is because of how neighboring pixels/blocks are used to calculate a pixel/block. At the edges of a video, the encoder would need to see the pixels/blocks on the other side of the image as neighboring. I don’t think any encoder does such a thing.

Trying a third party video player asset instead may help.