DOI: 10.2352/ISSN.2470-1173.2016.5.SDA-447

# An efficient approach to playback of stereoscopic videos using a wide field-of-view

***Chris Larkee and John LaDisa; Marquette University; Milwaukee, Wisconsin/USA***

## Abstract

*The MARquette Visualization Lab (MARVL) contains a cluster-based large-scale immersive visualization system to display and interact with stereoscopic content that has been filmed or computer-generated. MARVL uses head-mounted and augmented reality devices as portable sources for displaying this unique content. Traditional approaches to video playback using a plane fall short with larger immersive field-of-view (FOV) systems such as those used by MARVL. We therefore developed an approach to playback of stereoscopic videos in a 3D world where depth is determined by the video content. Objects in the 3D world receive the same video texture but computational efficiency is derived using UV texture offsets as opposing halves of a frame-packed 3D video. Left and right cameras are configured in Unity via culling masks so that they only uniquely show the texture for the corresponding eye. The camera configuration is then constructed through code at runtime using MiddleVR for Unity 4, and natively in Unity 5. This approach becomes more difficult with multiple cameras and maintaining stereo alignment for the full FOV, but has been used successfully in MARVL for applications including employee wellness initiatives, interactivity with high-performance computing results, and navigation within the physical world.*

## Introduction

The MARquette Visualization Lab (MARVL) is a shared resource dedicated to the creation of stereoscopic wide field-of-view (FOV) content that has been filmed or computer-generated for use with multiple exhibition methods, from large-scale systems such as CAVEs, to head-mounted displays. The advent of head-mounted displays and affordability of high-resolution cameras has prompted the need for efficient playback of stereoscopic video and photography using a wide FOV. The large-scale immersive environment within MARVL has several unique constraints dictated by its system attributes (10 Christie Digital Mirage WU7K-M projectors with Christie Twist connected to 6 HP Z820 workstations with Xeon E5-2670 CPUs, 32GB RAM and 2 NVidia Quadro K5000 GPUs each). Moreover, the applications of our end-users require the ability to display and interact with forward motion video obtained while moving at constant speed at 4K resolution and high frame rates.

There are several approaches to integrating video based content into VR, but the traditional approach has generally involved mapping a video source onto a flat plane as a 2D texture. This technique can create a sufficiently realistic effect if the video does not dominate the screen coverage but is more problematic for immersive FOVs. However, when the texture increases in size, up to a maximum size of a full sphere around the viewer, a different approach is required in order to create stereoscopic depth. Our objective was to develop an approach to playback of stereoscopic photos and videos in a 3D world where depth is determined by the video content, while also retaining 4K resolution, high frame rate and the possibility for interactions.

## Methods

The general approached developed to achieve our objective involves independently tagging two identically positioned meshes to be used as movie screens so they appear in only one eye. During this approach left and right cameras are configured via culling masks so that they only uniquely show the corresponding object for each eye. Computational efficiency is derived from the use of UV texture offsets as opposing halves of a frame-packed 3D movie (Figure 1) to share a single decoding component instead of having separate decoders for each eye.

*Figure 1: A sample frame from a stereoscopic, frame-packed movie with a wide FOV. This movie uses a custom cylindrical mapping, instead of a fully spherical equirectangular mapping.*

First a mesh is created and assigned UV coordinates to match our content, usually a full sphere, spherical section, or a cylinder. We use Blender for this, but any 3-D modelling program that provides control over UV coordinates and the orientation of normal would work well. The simplest case would be a UV sphere with its normal flipped, facing inwards. For our scene, we truncated the UV sphere (Figure 22) in order to match the shape of the output of our 6-camera GoPro rig (Figure 1).

Once the mesh is imported into Unity, two copies of it are made, one for each eye. Then the objects are tagged on separate layers, labelled “Left Only” and “Right Only,” as shown in Figure 23. The hierarchy of the entire scene is straightforward: two objects for each eye’s screen, two cameras for each eye, and a start node that contains scripts pertaining to the user’s inputs (Figure 4-left). For our scene, the start node contains a script to read a configuration file containing a link to the movie file, allowing us to make a single build of the program that would be reusable for multiple video files, instead of having the video file embedded into the program.

DOI: 10.2352/ISSN.2470-1173.2016.5.SDA-447

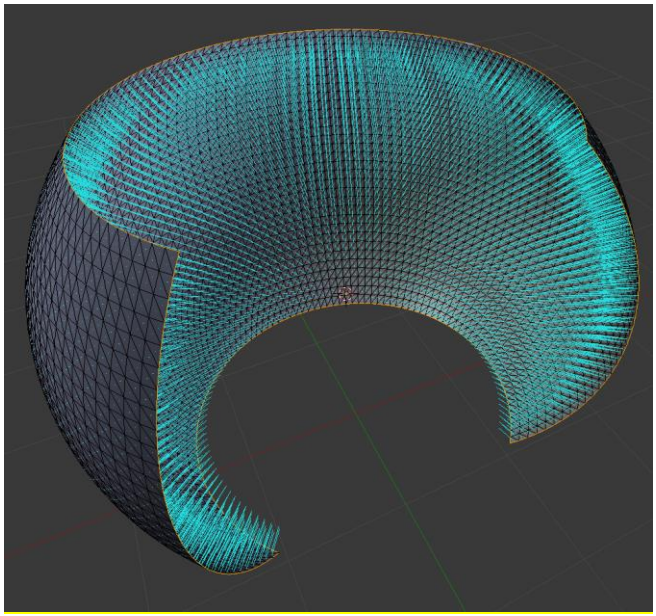

*Figure 2: Example of the custom mesh used for the screen, based on a UV sphere, made in Blender. The cyan lines show that the faces' normals are reversed to face inwards.*

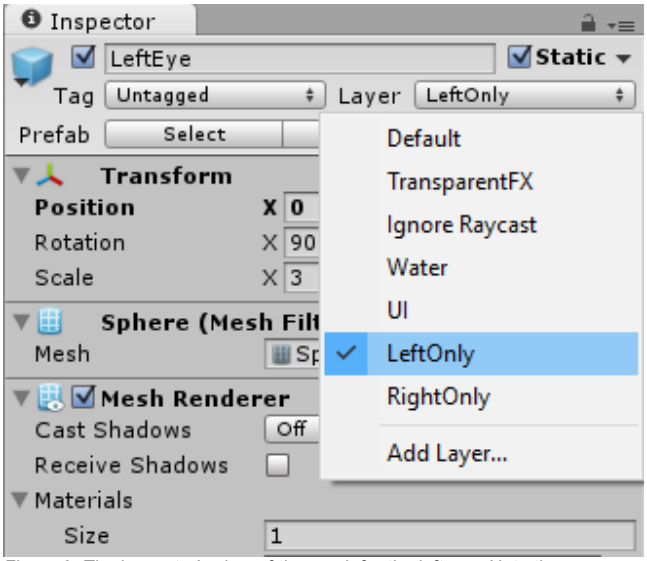


*Figure 3: The inspector's view of the mesh for the left eye. Note the configuration of custom layers.*

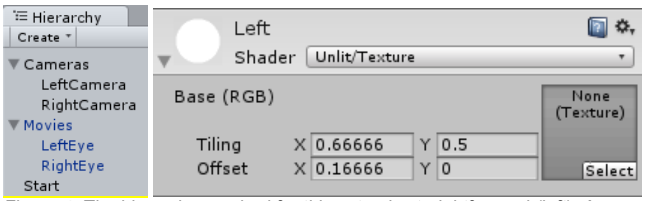


*Figure 4: The hierarchy required for this setup is straightforward (left). An example of the shader configuration for both meshes (right).*

The shader is set up to be as fast and simple as possible (**Error! Reference source not found.**4-right.) We use the "unlit/texture" shader for maximum performance, since we do not want any lights or shadows to affect this mesh. With the Y tiling set to 0.5 and the Y offset set to 0, only the top half of the texture will be stretched to fill the entire mesh. The right object's Y offset is set to 0.5, causing it to map the bottom half of the texture image to the mesh. The image input for the shader is left blank here, because when the program is running, it will be replaced with a new frame of video at every frame update by AVPro (Figure 9). If a texture is added here, it will be visible only for a brief moment while the movie is loading, or if the movie is unable to load. The X tiling and offset values here are changed slightly in order to perform some aspect ratio correction on the source material, which is not applicable to all movies.

The next step is to configure the camera to set up culling masks so that each camera will not be able to see objects for the alternate eye. Unity 5.1 and greater now features built-in VR support, making this step far simpler. Two cameras must be created, one for each eye. For each camera, the target eye setting must be set to left or right, and the culling mask must be set to exclude the alternate eye. When using the Oculus utilities, the included camera rig prefab already contains the correct hierarchy of objects and cameras, making finding the correct camera simpler. Figure 5 below illustrates the sample settings for the left camera, which shows "LeftOnly" enabled, and "RightOnly" disabled.

To run this content in our large-scale immersive environment, we found it was necessary to use a proprietary plugin for Unity called MiddleVR [1]. Although Unity is strongly supportive of VR, their support mostly covers head-mounted displays, and there are still some very significant features that are not yet available to run the game engine in a CAVE, namely multi-machine clustering, nonplanar camera alignment, and infrared tracker support. MiddleVR changes the way all cameras in the scene operate, so the process to set up these stereoscopic textures is slightly different. The configuration of the meshes and objects is the same, but the camera settings must now be done with a simple C# script that executes at runtime (Figure 6). This is because MiddleVR creates the necessary stereoscopic camera rig when the program starts, so our script must be configured to run after MiddleVR has finished creating the cameras.

©2016 Society for Imaging Science and Technology
DOI: 10.2352/ISSN.2470-1173.2016.5.SDA-447

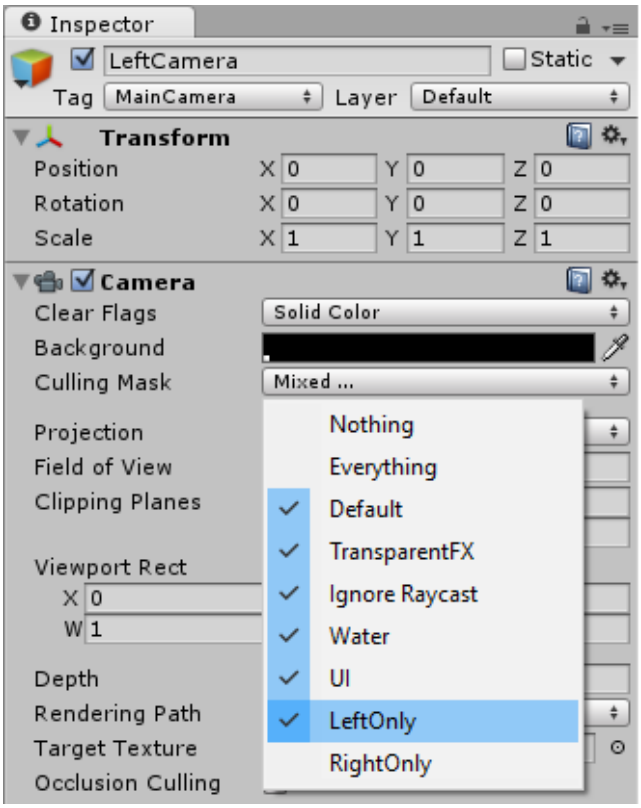


*Figure 5: The inspector's settings for the left eye camera, showing the settings on the culling mask.*

```
foreach (Camera currentcamera in Camera.allCameras) {
      if (currentcamera.name.Contains(".Right"))
            currentcamera.cullingMask = ~(1 <<
LayerMask.NameToLayer("LeftOnly"));
      if (currentcamera.name.Contains(".Left"))
            currentcamera.cullingMask = ~(1 <<
LayerMask.NameToLayer("RightOnly"));
      if (currentcamera.name.Contains ("Overview"))
            currentcamera.cullingMask = ~(1 <<
LayerMask.NameToLayer("RightOnly"));
}
```

*Figure 6: Code sample activating camera culling masks for all cameras.*

The code itself simply performs a search for all active cameras, and if the camera name contains "Left" or "Right," it unchecks the culling mask for the alternate eye. The final conditional statement looks for the word "Overview," which is used as the monoscopic overview camera on the control desk. If a specific layer mask is not selected for the overview camera, it will see both eyes' objects on top of each other, creating a flicker known as z-fighting. Figure 6 also demonstrates the somewhat confusing syntax to uncheck an option from a list in C#. "Camera.allcameras" is an array containing all cameras provided by the MiddleVR API.

## Results

Initial implementation proved difficult to achieve the correct amount of stereo separation because our method runs on top of the pre-existing stereoscopic rendering method. If the plane with the video texture is in a position in 3D space where parallax is not neutral, the user will perceive the video at inconsistent depths, because both the video texture and the pre-existing stereo camera will be creating parallax independently of each other. To combat this conflict, the interpupillary distance for the 3D cameras is set to zero (Figure 7) which cancels out the separation effect that they contribute while maintaining the quad-buffer, stereo rendering pipeline.

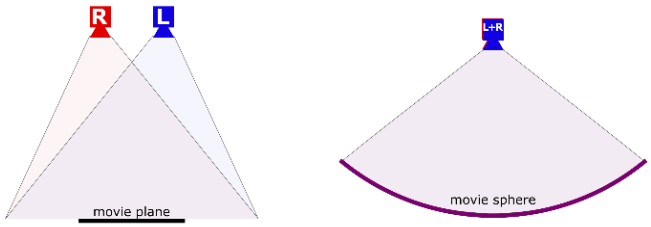


*Figure 7: Comparison of a monoscopic texture in 3D space, where 3D parallax comes from the camera separation (left), and a 360-degree implementation, where 3D parallax is determined by the texture (right).*

This approach has subsequently been used successfully in MARVL for applications including employee wellness initiatives, interactivity with high-performance computing simulation results, and navigation within virtual versions of the physical world (Figure 8).

It is worth noting that we discovered playback of high-resolution video textures in Unity also requires use of the additional third-party plugin AVPro Windows Media [2] made by Renderheads, because Unity's built-in video decoding system (the MovieTexture class) is not sufficient for use in VR. More specifically, Unity's Movietexture class is not optimized enough to provide adequate quality and frame rates when using HD and 4K video textures, while AVPro utilizes hardware acceleration to make the decoding process efficient enough to be satisfying. Several additional features are also needed for our applications, such as variable speed playback and the ability to seek within a video clip.

Despite the efficiency of the plugin, it is still important that the video rendering pipeline is simplified as much as possible in order to maintain high framerates. For this reason, the stereoscopic video is frame-packed, with the left eye's image on top, and the right eye's image on the bottom (Figure 1). AVPro is a modular component based system, so the image decoding script is separated from the script that applies the texture to the object. The sample documentation generally places the decoding script and the texture update script on the same object, so it may make sense to have the two components on each screen object. However, we have found a significant performance improvement by only having a single decoding script and two texture update nodes that both reference the same decoding script (Figure 9). This way, the most computationally intensive task in the pipeline is only done once.

©2016 Society for Imaging Science and Technology
DOI: 10.2352/ISSN.2470-1173.2016.5.SDA-447

*Figure 8: Resulting uses of immersive, wide FOV, video rendering for wellness initiatives such as bicycling and yoga.*

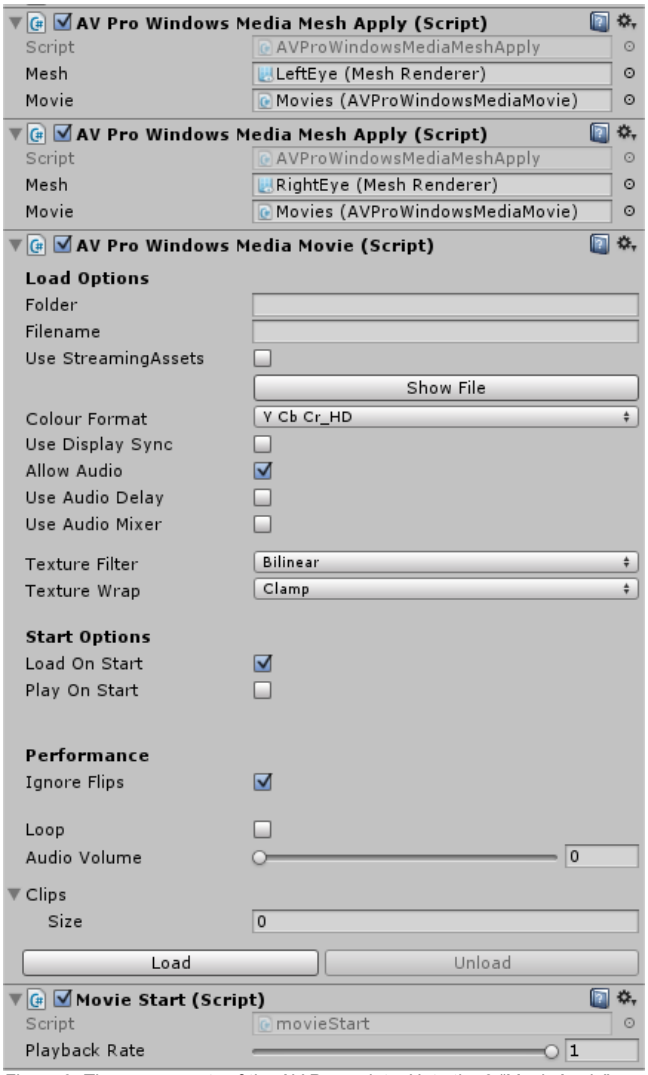


*Figure 9. The components of the AV Pro scripts. Note the 2 "Mesh Apply" scripts, but only 1 movie decoding script. The bottom component, "Movie Start," implements the user's controls.*

In addition, despite the efficiency of this setup, we have found that the specifications of the movie file itself play a big factor in achieving smooth movie playback inside Unity. We have found that playback movies of HD quality (1920x1080 or lower) consistently plays back very smoothly when using common encoder settings and reasonable bitrates, such as h264 at 20Mbps. However, achieving smooth playback at 4K resolution requires a much narrow set of specifications, due the significantly increased size of the frame, and even though the decoding is GPU accelerated, some additional configuration may be required in order to enable the most efficient settings for 4K video. To encode the video, we use the open source video processing tool ffmpeg [3]. Even when the video is made using a program like Adobe Adobe Effects, we export to an intermediate file and then re-encode that file with ffmpeg, because ffmpeg has some additional parameters that make a difference for high-performance decoding.

The two codecs that work the best for 4K in Unity are Hap and Xvid. Hap's performance benefits are no surprise because it is a

DOI: 10.2352/ISSN.2470-1173.2016.5.SDA-447

recently developed codec designed specifically for high-performance realtime playback. It is able to decode 4K frames extremely quickly because its compression method is nearly identical to the texture compression method used internally on the GPU. Its downside is that its bitrate is extremely high, even higher than most intermediate codecs. Bitrates vary according to the content, but our tests measured an average bitrate of 750 Mbps, meaning that a sixty minute video would exceed 320 GB. Although the performance was satisyfing, we still wanted an improvement in file size, encoding times, transfer times, and we found an alternative codec solution with Xvid. Xvid is an unusual choice, because its generally regarded as an obsolete predecessor to the much more advanced H264 and H265 codec family. However, like Hap, its simplicity becomes an advantage when there is a need to decode large frames as fast as possible. In order to achieve sufficient quality at 4K resolution, it is necessary to use a bitrate much higher than one would need for a more efficient codec like H264. We standardized on 90Mbps because higher rates have caused playback hiccups with no additional gain in quality (Table 1).

| | **H264** | **Xvid** | **Hap** |
|---|---|---|---|
| Compression type | Mpeg4 | Mpeg4 | Interframe |
| Encoding performance | Fast | Very Fast | Slow |
| Decoding performance (4K) | Slow | Fast | Very Fast |
| Decoding performance (1080p) | OK | Fast | Very Fast |
| Typical 4K bitrate (Mbps) | 30-60 | 90 | ~750 |
| Supports alpha channel | No | No | Yes |
| Supports random seeks | No | No | Yes |
| Variable frame rate | Limited | Limited | Yes |
| AVPro decoding method | External | External | Internal |

*Table 1: Comparison of codec performance and features for VR usage.*

Table 1 also shows how Hap still has some unique qualities that are useful in special cases. For example, because every frame is encoded independently, seeking within the video is instantaneous, while Xvid and H264 require seeking to a keyframe. Also, the alpha channel support is very useful for creating animated overlays and additional effects. The final row, decoding method, refers to the detail that AVPro requires a separate codec installation and configuration to decode H264 and Xvid [3]. For fixed installations such as ours, this is not an issue, but it does create some difficulty when deploying commerical products. The decoder we use is LAV Filters, a version of ffmpeg's open source decoders packaged for Microsoft's DirectShow framework.

On mobile VR devices such as the Samsung Gear VR, neither MovieTexture nor AVPro are available for animating textures, but the same mask and layer setup has been used with static textures to achieve stereoscopic 360 degree still images on mobile devices very efficiently, even with 4K x 4K textures. An example related to archetectureal pre-visualization from MARVL is shown in Figure 10.

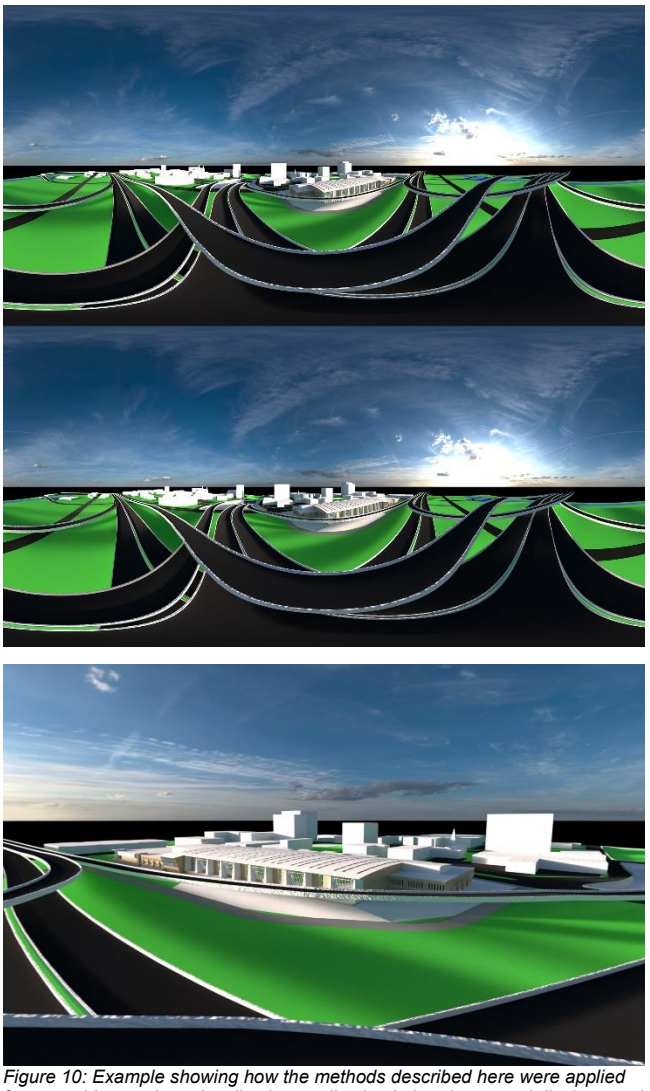

*Figure 10: Example showing how the methods described here were applied for an architectural previsualization application in head-mounted displays such as the Samsung GearVR and Oculus Rift. The stereoscopic spherical version of the content for the L and R eyes is shown in the top two images while a cropped version simulating view from a head-mounted display is shown in the bottom image.*

## Discussion

MARVL contains a cluster-based large-scale immersive visualization system that focuses on displaying and interacting with stereoscopic content that has been filmed or computer-generated. To achieve the goals of the lab and meet the needs of our end-users, we found it was necessary for our projects to utilize Unity instead of any other spherical movie player software in order to bring a degree of interactivity to our video content. By displaying our content in this way, we gain a lot of new possibilities, such as allowing the user to trigger scene changes at their own pace, adding in overlay elements that take advantage of the direction the user is looking, modulating playback speed according to external controls, data gathering on user attention, and more.

When most 3D engines render active stereo for CAVEs, they use a quad-buffered rendering pipeline, which is effectively separate pipelines for each eye. Most users of the rendering engine always

DOI: 10.2352/ISSN.2470-1173.2016.5.SDA-447

want their world to appear spatially consistent, so there is usually very little need to target each eye individually. Initial attempts at stereoscopic textures did not access the quad-buffer pipeline, but instead involved toggling the textures' state at a rate synchronized with v-sync. This was reliable for simple scenes, but synchronization and phase failed during complex scenes and stress tests. This method implements a system of displaying stereoscopic textures in a way that is reliable enough for use with active stereo, without having to modify low-level code manipulating the quad-buffer rendering pipeline.

## Conclusions

In summary, the methods described above utilize layers and masks to create unusual objects that appear in one eye and not the other, allowing us to use differing textures for each eye. The steps to create these objects are not obvious or readily documented, because in most use cases, the rendering system always wants to draw a world that is consistent for each eye. The details above related to the current approach are provided for use and iteration by other visualization facilities with similar constraints and user applications.

## Author Biography

*Chris Larkee is the visualization technology specialist for Marquette University. His work combines a background in video production, motion graphics, broadcast engineering, and graphics programming. Currently, he maintains and develops content for the Marquette University Visualization Lab, whose centerpiece is a 6 node, 10 projector VR Cave. Chris has led the production of over 30 VR projects since it opened in 2014, in topics ranging from healthcare, fitness, engineering, and architecture.*

*John LaDisa was a postdoctoral scholar at Stanford University for 2.5 years after earning his Ph.D. in Biomedical Engineering. He now directs MARVL and the Laboratory for Translational, Experimental and Computational Cardiovascular Research. These labs are supported by grants to study cardiovascular disease, engineer treatments, and uniquely visualize results, while training the next generation of scientists. Together with colleagues, Dr. LaDisa has published >40 peer-reviewed articles and raised ~$2M in extramural funding.*

## References

[1] MiddleVR, "MiddleVR For Unity," [Online]. Available: http://www.middlevr.com/middlevr-for-unity/.

[2] Renderheads, "AVPro Windows Media," [Online]. Available: http://renderheads.com/product/av-pro-windows-media/.

[3] FFmpeg, "FFmpeg," [Online]. Available: http://ffmpeg.org/about.html.