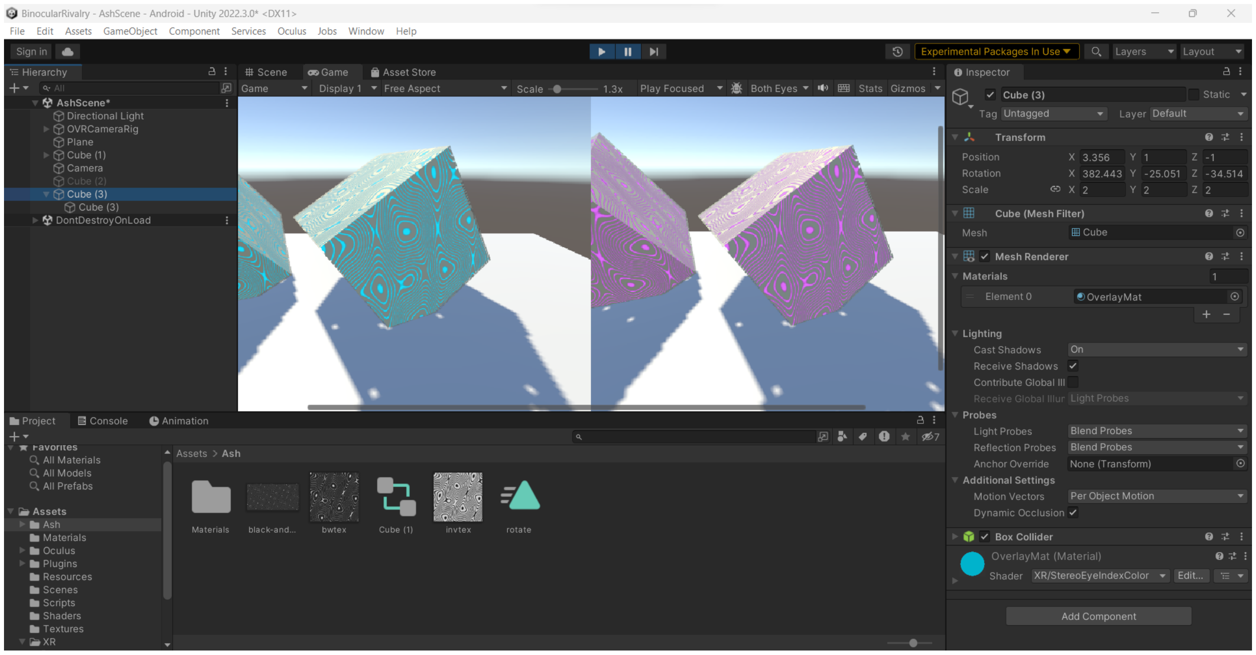
CSE 493V Final Project Report

ASH LUTY, DEEPTI RAMANI, and NIK SMITH, University of Washington



Cube (3) — Tag: Untagged — Layer: Default

Transform
Position X 3.356 Y 1 Z -1
Rotation X 382.443 Y -25.051 Z -34.514
Scale X 2 Y 2 Z 2

Cube (Mesh Filter)
Mesh: Cube

Mesh Renderer
Materials: 1
Element 0: OverlayMat
Lighting
Cast Shadows: On
Receive Shadows: ✓
Contribute Global Ill:
Receive Global Illur: Light Probes
Probes
Light Probes: Blend Probes
Reflection Probes: Blend Probes
Anchor Override: None (Transform)
Additional Settings
Motion Vectors: Per Object Motion
Dynamic Occlusion: ✓

Box Collider

OverlayMat (Material)
Shader: XR/StereoEyeIndexColor

Fig. 1. Generating a complex cube with patterned rivaling colors and a cube rotation animation, to test the effect of more complex binocular rivalry on visual comprehension and comfort. Unity programs were built to Oculus Quest 2

Abstract

What happens when we are given complete control over what images are sent to each eye? We can generate robust, programmatic illusions which harness the splicing of binocular images performed by the brain, and observe the qualities of the resulting perceived binocular images. Previous work in this area has attempted to use rivalry as a stimuli to draw attention to specific objects in the scene (Krekhov et al.). Binocular rivalry and the stitching together of almost identical images has also been used to improve the perceived contrast of images, despite the physical limitations of a headset (Zhong et al.). In this project, we explored different ways to induce binocular rivalry and the effects this had on user-perceived images.

Introduction

The most common reason to send different images to each eye in a virtual reality headset is to give users visual depth cues. Objects are projected differently to give an illusion of depth and occlusion. What if we did more than change the projection of the scene per eye? How would users respond if an object was a different color per eye, or present in one eye or not the other?

Binocular rivalry is the optical effect of applying a very different image to each eye [4]. Instead of the images being superimposed or fused, the two rivalring images can be focused on individually. This is usually not desired, since virtual reality experiences are intended to be fully immersive and seamless. However, some research has been done [1] to explore the usefulness of using binocular rivalry as a visual cue.

We wanted to discover more about the usefulness of binocular rivalry, such as ways to draw attention to objects, novel experiences, and more. It is also important to explore the positive and negative effects on users. It is possible that some effects produce minimal negative effects on users, while larger discrepancies in images might be unpleasant.

Our approach is to create various filters to change the visuals of objects per eye. The effects can be modified to be minor or major discrepancies. This will be useful for user studies where we test the mental and physical effects of binocular rivalry of various types. Users can also explore the usefulness of visual cues in a virtual escape room.

Contributions

Our primary contributions for this project are:

- We created stereo shaders to render objects differently for the left and right eyes. The first shader we implemented was a color change shader, which rendered the object with different color overlays for the left and right eyes. We also implemented a grayscale-color shader, which rendered the object in grayscale for the left eye and in the original color scheme for the right eye. Finally, we made an HSV shader, which renders the object with different (adjustable) HSV values for the left and right eye.

- We ran a user study to determine how the different stereo shaders affected how users focused on and reacted to objects. We created a Unity scene containing an environment with a few objects rendered in grayscale, then added an object that would be rendered differently. We then asked users to switch between stereo shaders and observed how they reacted to the object for each one.

Related Work

A previous research paper by Krekhov, Krüger [1] compared binocular rivalry to other visual cues, such as color, to find objects. Participants were asked to notice if a particular object is present. If the object was present in the image, it is shown in one eye and not the other. The object is displayed in neither eye if it is not there. While this is a minimally-researched area, they found that binocular rivalry is slightly better, or at least comparable to, other visual cues. On its own, binocular rivalry did not cause any discomfort in participants.

Method and Implementation Details

Our approach to this project can be divided into two parts: creating the shaders to be applied to different objects in the scene, and testing the different applications of those shaders and their effects on human perception.

Stereo Shaders

We wanted to implement a variety of different stereo shaders in order to best test how humans respond to different forms of binocular rivalry. Unity provides support for single-pass stereo instanced rendering, which we used to make our custom shaders [3].

When returning the final color to output in each fragment shader, we want to return something different for the left eye and right eye. We did this by using unity's $lerp$ function, which takes in two inputs to interpolate between, and the time value used to interpolate between them. Something to note is that $lerp$ returns the value of the first input when the time value is equal to 0, and returns the value of the second input when the time value is equal to 1. Additionally, Unity provides a built-in shader variable, $unity\_SteroEyeIndex$, which is set to 0 when rendering the left eye and 1 when rendering the right eye.

Using this, we set our fragment shader to return separate values when rendering the left and right eye views by using $lerp(left\_color,\ right\_color,\ unity\_StereoEyeIndex)$, where $left\_color$ and $right\_color$ are the RGBA vectors for the color rendered for the left eye and right eye, respectively.

Grayscale-Color Shader

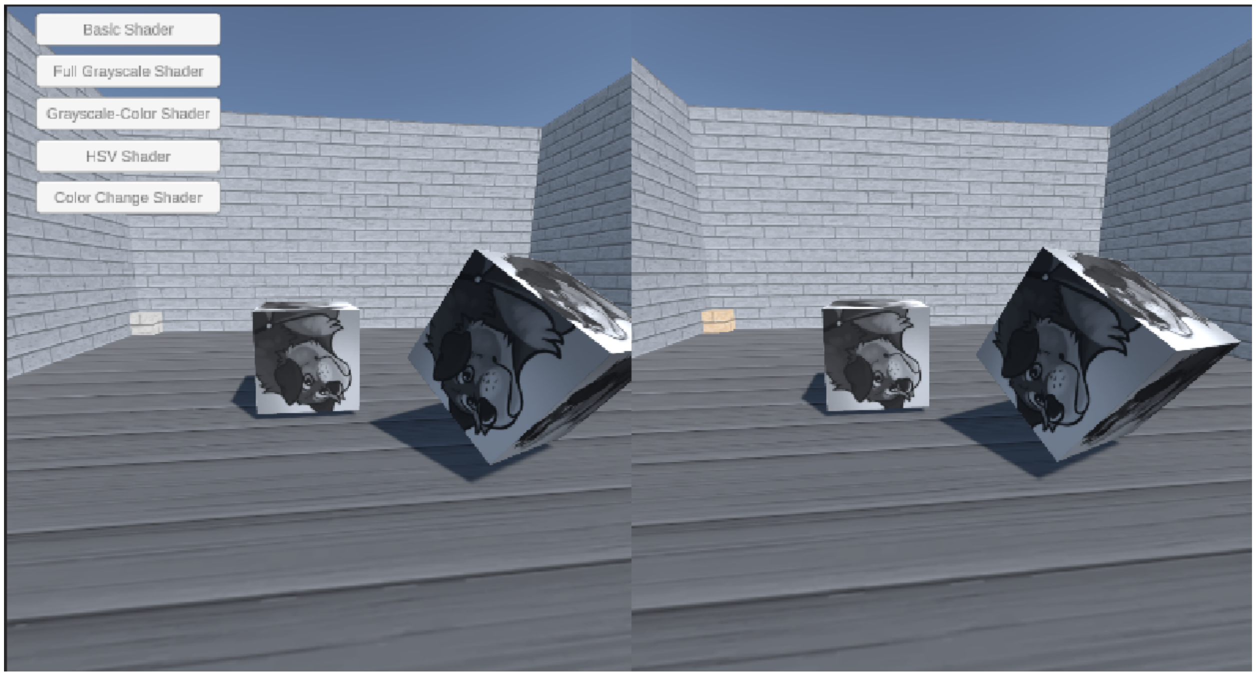


Fig. 2. Rendering a Unity Scene with the right eye seeing an object in color, and the left eye seeing the same object in grayscale.

Our shader for grayscale-color stereo rendering only takes a 2D texture image as input. The vertex shader converts the vertex position from object space to clip space so that the object can be properly rendered, and also passes in the uv coordinates for the passed-in texture, which will be used in the fragment shader.

The fragment shader samples from the texture using the passed in uv coordinates. For the right eye, the final color returned is simply whatever was sampled, but for the left eye, the shader performs a weighted sum of the RGB color values to calculate the final grayscale value, and uses that value for all three of the RGB channels in the final color. The weights used for this sum are: z
$grayscale = (0.299 * red) + (0.587 * green) + (0.114 * blue).$

As seen in Figure 2, applying this shader to an object results in the left eye seeing the object in grayscale, and the right eye seeing the object in full color.

Adjustable HSV Modifier Shader

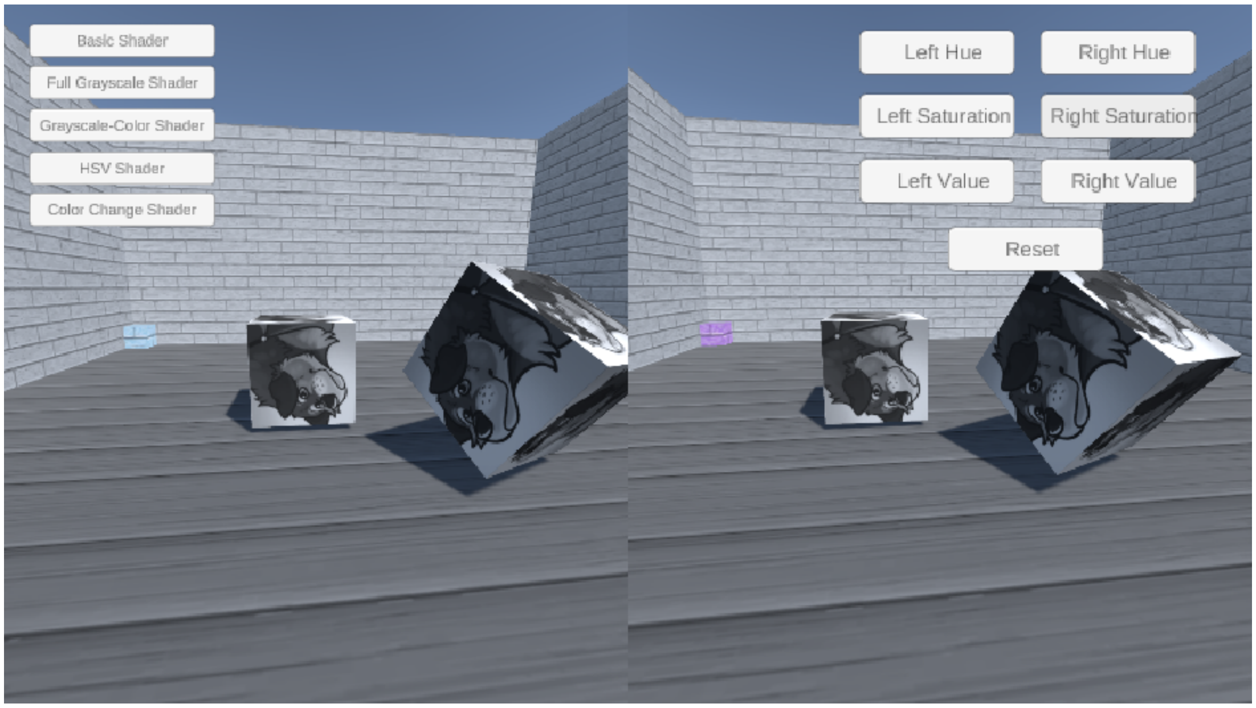


Fig. 3. Rendering a Unity Scene with the right eye and left eye seeing the same object with different hue, saturation, and value adjustments. The scene includes a UI in the top right corner of the screen for modifying each property for the left and right eye.

Our shader for HSV stereo rendering was created as a way to experiment with how extreme we could make the differences between images sent to the left and right eye, and whether certain properties had more of an impact on the effects of binocular rivalry than others.

The HSV shader takes in a 2D texture as input, in addition to modifiers for the hue, saturation, and value for each eye. Similarly to the grayscale-color shader, the HSV shader's vertex shader also converts the vertex position to clip space and returns the uv coordinates for the texture at that position.

The fragment shader, on the other hand, requires a lot more computation, since it needs to convert from RGB to HSV, apply the modifications to each property, then convert the modified values back to RGB before returning them. Unity's shader graph includes a $Unity\_ColorspaceConversion$ node which includes both required conversions for this shader. We initially tried to implement this conversion manually, but found that Unity's provided implementation was more accurate and efficient, and went with that instead.

Something to note is that when applying the HSV property modifiers to the original texture color values, we needed to clamp the output HSV values between 0 and 1, since

Unity’s node uses these bounds for them, rather than the typical 0-360 for hue and 0-100 for saturation and value.

Color Change Shader

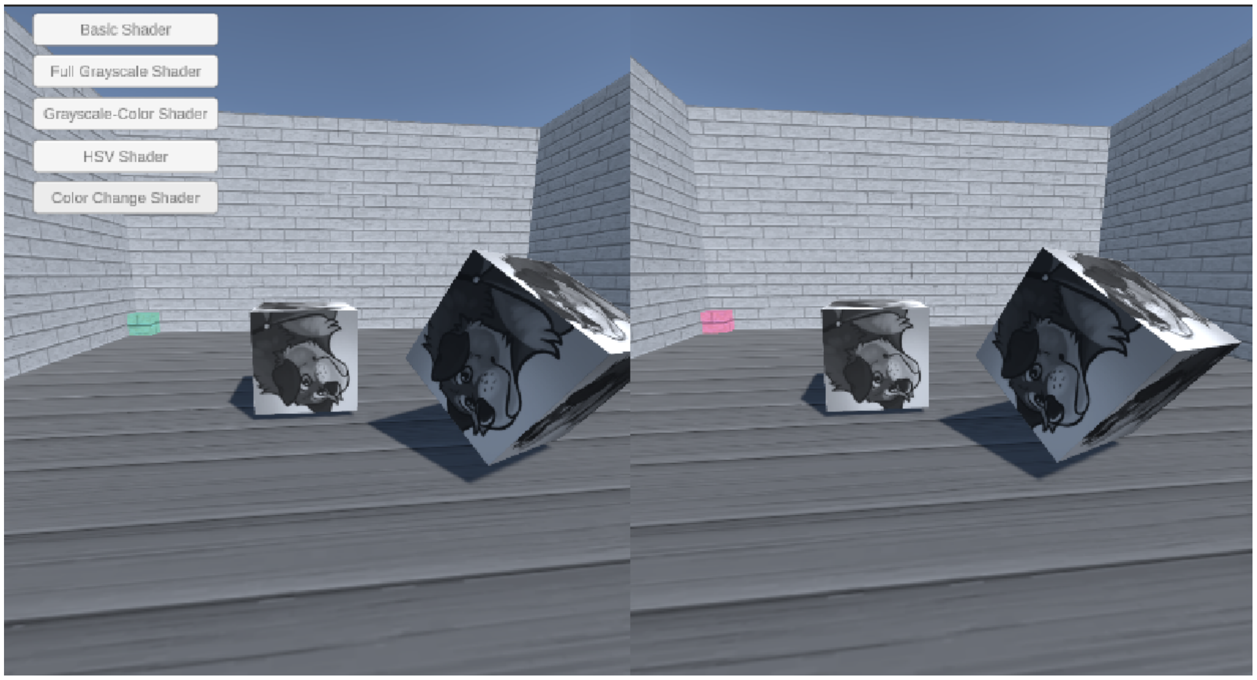


Fig. 4. Rendering a Unity Scene with the right eye seeing an object with a cyan overlay, and the left eye seeing an object with a magenta overlay.

Our shader for color change stereo rendering takes in a 2D texture as input, in addition to the colors to use for the left and right overlays. Similarly to the two previous shaders, the vertex shader simply converts the vertex position to clip space and returns the uv coordinate to use when sampling the texture input.

The fragment shader then samples from the texture input to get the initial RGBA values, then multiplies it by the right eye color input to get the right eye output, and multiplies it by the left eye color input to get the left eye output. This multiplication is element-wise multiplication, so each channel in the original sampled color will be multiplied by the corresponding channel in the overlay color.

As seen in Figure 4, applying this shader to an object results in it being rendered in two different colors for the left and right eye, with the original texture information still being preserved underneath the color overlay.

Applications

The main goal of this project was to test the effects of sending different images to the right and left eye, and to determine if there was any use for those effects in VR applications.

We initially wanted to use the color overlay stereo shader to test whether rendering an object in different colors for the left and right eye would make it stand out to the viewer without any negative effects, or whether it would simply cause disorientation or nausea. After creating an initial implementation of this filter (using bright red and green), we found that it resulted in the object having an almost shiny, reflective appearance, despite not having any specular or metallic properties. After this discovery, we began to experiment with different color combinations to see what different effects we could find, and concluded that most color combinations result in a shiny, almost iridescent appearance- this made sense, since iridescence is dependent on viewing angle and will have the similar effect of reflecting different colors to each eye from the same location.

We also tested the color overlay shader on moving objects, to see if this would remove or change the iridescence effect. We didn't see any noticeable change, but rotating the object did make the texture seem to shift slightly as the colors "flashed", which was an interesting effect.

We also wanted to test the HSV modifier shader to see what properties would make the object stand out the most to users, as well as how much those properties needed to differ between eyes in order for this to take effect. We found that modifying saturation didn't have too much of an impact on the visibility of the object until the difference was from almost grayscale to extremely high saturation. We also found that changing the value of the object made it stand out the most, since it created an almost flashing effect when it was viewed, and changing the hue also had a significant effect on visibility, though it was found to be more disorienting than the other properties in general.

Hardware and Software

For this project, we used Unity to create the application used to test our shaders. We used the Unity XR Plugin Management system to handle the necessary plugins and packages required for Unity VR development. Specifically, we used the Mock HMD plugin to render and display the scene independently for the right and left eye views within the Unity game view in order to better debug and test the shaders without having to build to a VR headset each time. We also used the Oculus plugin in order to run our app on the Oculus Quest and test it in VR, to better experience what effects each shader had on our perception of the object they were applied on. We also downloaded the Unity VR Escape Room package, since we wanted to reference how to create an escape room for VR and then apply our different shaders to objects within that escape room, but ultimately ran out of time for doing so.

The Unity project can be found at https://github.com/AshleyL-02/BinocularRivalryVR.

Evaluation of Results

One limitation of our approach is that we were unable to figure out how to apply certain filters or effects over only one object in the scene. Because shaders are only applied to specific objects (or the whole scene, by using cameras and replacement shaders), this made it difficult to figure out how to implement filters that affect parts of the background, in addition to the object itself (such as blur, glow, luminance, etc.) This made it a lot more difficult to implement our initial vision for the project.

Another limitation of our approach is that we didn't have time to implement a full escape room environment for testing the effectiveness of our shaders on improving object visibility and how much they stand out to users. Because of the simple, empty nature of the "room" we used to test these shaders, the objects weren't difficult to sport from the beginning, which made the results of our experiments less meaningful.

Despite this, we found that we could create some interesting effects using stereo rendering and applying different effects to each eye. The color change effect was the most interesting- we were expecting the colors to clash and the final result to look disorienting or difficult to look at, but it ended up having a more visually appealing effect. On the other hand, the grayscale-color filter didn't have as much of an impact. Although it did make the object stand out more than its surroundings, the resolution between the left and right images wasn't as smooth as that of the color change shader.

Discussion of Benefits and Limitations

The benefits of this work was creating various ways of creating visual cues using binocular rivalry. We also were able to create a simple scene for users to explore and experience the different filters. However, we ran into limitations with Unity and the Universal Render Pipeline. We found it easier to create filters for a specific material, so we were unable to test full-eye filters in this project. We also found it difficult to create filters that modify an object's texture or alpha, like glow or gaussian blur.

Future Work

Future work could examine form-based rivalry, where objects with different forms, like a square and circle, are sent to each eye. Implementation would involve setting up two locations in a Unity scene, with a right-eye camera viewing one form and a left-eye camera viewing the other form. The perception of these forms could further be heightened by using forms with rivaling semantic meaning, such as sending similar pictures of a dog and cat to each eye, and asking the user to determine what animal they are being shown.

Since introducing new images to a given eye will increase the amount of attention the brain gives to a specific eye, we wonder if the frame rate of a given display could be improved by splitting frames not just over time but also alternating frames between eyes. The resulting effect of both mental splicing but also the improved recognition of the most recently updated frame could potentially make this concept work without inducing too much rivalry.

Conclusion

Binocular rivalry creates a novel experience for users by creating two different images they can focus on. This allows developers to include important cues that are harder to miss, and more interesting to see. Binocular rivalry can easily be implemented in modern rendering systems, especially systems that use a separate frame buffer for each eye, or using Unity stereo rendering. Binocular rivalry could be leveraged for many interactive scenarios, from virtual reality games to scientific research to make important information visually unique.

Acknowledgements

[1] (Krekhov et al.) Deadeye: A Novel Preattentive Visualization Technique Based on Dichoptic Presentation,
https://www.cs.rpi.edu/~cutler/classes/visualization/S20/papers/deadeye_2019.pdf

[2] (Zhong et al.) DiCE: Dichoptic Contrast Enhancement for VR and Stereo Displays,
https://www.cl.cam.ac.uk/research/rainbow/projects/dice/

[3] Single-Pass Instanced Rendering,
https://docs.unity3d.com/Manual/SinglePassInstancing.html

[4] Binocular rivalry, https://en.m.wikipedia.org/wiki/Binocular_rivalry