15-869 Project
Raycaster for the Oculus Rift

Brian Fischer
bfischer@andrew.cmu.edu

1 Summary

I modified an existing raycaster to act as a raycaster for the Oculus Rift. Additionally, an analysis in work done(number of rays traced) was done in order to determine the benefits of directly rendering images for the Oculus Rift instead of rendering and shading an image.

2 Background/Approach

This project was developed for and tested using the Oculus Rift Development Kit DK2. The DK2 contains a screen with 1920x1080 resolution. The screen is divided into 2 parts, and displays a separate image for each eye(960x1080 per eye).

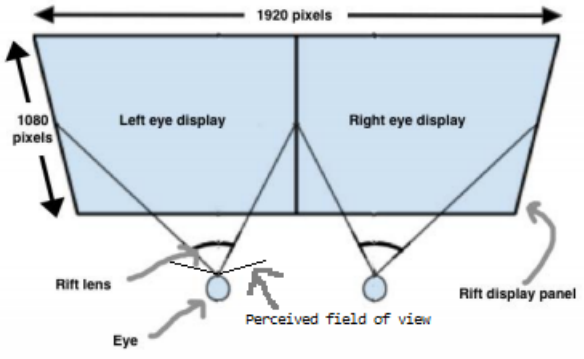

Figure 1: The Oculus Rift Display[1]

The most important feature of the Oculus Rift display is the lens in front of the user's eyes. These lenses serve 2 purposes. The first is to increase the user's perceived field of view. This helps incease the immersion in Virtual Reality by causing images to occupy more of the user's field of view. The second purpose of the lens is that it enables the user to focus on the screen and see images displayed on it clearly. Without the lenses, it's very difficult to focus on the screen since it is very close to the user's eyes.

The lens, however apply distortion to the images that are viewed through them. This distortion is known as pincushion distortion. This kind of distortion pushes pixels in image out towards the edges. Barrel distortion is the inverse of pincushion. Images with barrel distortion have pixels pushed in towards the center of the image.

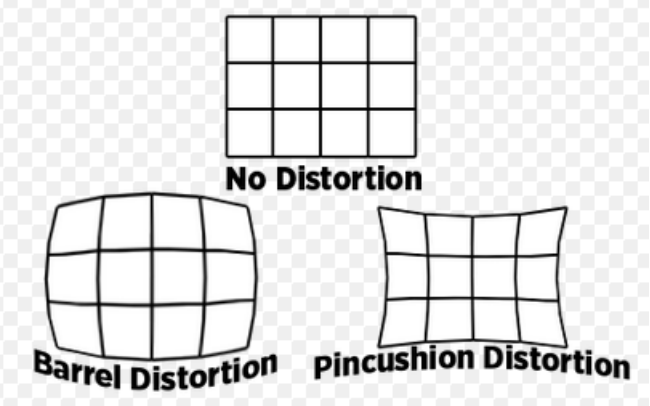

Figure 2: Grid with no distortion, barrel distortion, and pincushion distortion

In the above image, there are examples of both pincushion and barrel distortion. Images for the Oculus Rift all feature the distinctive barrel distortion shape.

The starting point for this project was an existing raycaster written in Javascript. The raycaster had basic functionality, but needed to be modified for this project. The first step to changing the raycaster, was to change how rays were computed. Usually, we'll compute some ray R for some pixel(x,y). However, to calculate a ray R' for the Oculus, we must first calculate a new (x',y'). (x',y') is the pixel value stored at position (x,y) in our image after barrel distortion has been applied. In order to calculate the new (x',y'), we do the following:

```
uniform float BarrelPower;
vec2 Distort(vec2 p)
{
    p = p - (0.5*width, 0.5*height);
    float theta  = atan(p.y, p.x);
    float radius = length(p);
    radius = pow(radius, BarrelPower);
    p.x = radius * cos(theta);
    p.y = radius * sin(theta);
    p = p + (0.5*width, 0.5*height);
    return Vec2(p.x, p.y);
}
```

The above code starts by shifting the origin of the image (0,0) to the center of the image(assuming the framebuffer is indexed with (0,0) in the lower left had corner. This is done because barrel distortion scale quadratically with respect to how far away they are from the center of the image. Next we calculate The angle of our input vector with respect to the x axis and calculate the length of the vector. We can imagine these steps as converting our input pixel coordinates (x,y) from the cartesian coordinates to polar coordinates. Next we'll scale our radius by raising it to some power. In testing, I found that the 1.05 1.10 was an appropriate power that would cause straight lines in our scene to be seen as staight lines when wearing the Oculus headset and looking through the lens. After this, we shift our points back to the cartesian coordinates by the vertical and horizontal components of our new vector. Finally our (x',y') are shifted back to the original image space. This procedure allows us to calculate our rays R' in order to render a barrel distorted image. One remark to make is that if we perform this distortion on framebuffer of size width*height, we'll end

up calculating some rays that fall outside of the framebuffer(i.e. (x' less than width or greater than zero, y' less than zero or greater than height). This makes sense since barrel distortion shrinks the image. In order to fill the entire frame buffer, we'll have to increase the field of view of the camera. If we don't expand the field of view of the camera, we'll have a larger black border around our images.

After this, we had a functioning raycaster that rendered barrel distorted images for the Oculus Rift. The next step of the project was to to add adaptive sampling to our project, such that we take 1 sample for 4x4 boxes of pixels of pixels on the edge of the screen, 2x2 boxes of pixels closer to the center, and only sample with full density in the center of the image.

3 Results

The result of this project is a raycaster that acts as a proof of concept as a raytracer for the Oculus Rift. It is less successful in regards to the performance of this raytracer. It has a low framerate, and reduced functionality(limited geometric and shading capabilities). However, the produced images displayed well on the Oculus Rift headset.

An analysis was done for the amount of rays that are cast by this raycaster compared to the normal method of rending for the Oculus Rift. For rendering a 1920x1080 image, the normal method of rendering will render an image of size ˜2364x1461(taken from Oculus World demo). This image will result in ˜3.5 million rays being cast per image if one ray is cast for each pixel rendered. A larger image is rendered since its size is reduced when barrel distortion is applied to it. The resulting image will fit most of the Oculus Rift's display. For the Oculus Rift raycaster, an image with resolution 1920x1080 is rendered if we expand the field of view. This results in ˜2 million rays being rendered. If we do not expand the field of view, the resuling image has a size of ˜1400x800. This results in roughly 1.1 million rays being cast. If we perform adaptive sampling, we ˜1.2 million rays are cast(for the hardcoded bounderies).

This sort raycaster can significantly reduce the work done(with respect to the number of rays cast into the scene. If adaptive sampling is performed, the work done can be reduced even further at the cost of the quality of the output image.

For demos, see http://www.andrew.cmu.edu/user/bfischer/15869.html

4 Future Work

The raycaster for this project was developed in Javascript and allowed for easy testing. However, a result of this is that the raycaster is not very performant and has a low frame rate. This project served as a proof of concept for a raycaster for the Oculus Rift, and could be ported to a C++/OpenGL Raycaster/tracer for increased performance capabilities. Additionally, chromatic aberration is very apparent when images from this raycaster are viewed on the Oculus Rift. The usual Oculus shading methods both distort the image and correct for chromatic aberration. In order to add functionality like this to this raytracer, individual rays would need to be traced for each color(RGB) per pixel, as opposed to casting only one ray per pixel. Additionally, while the current raycaster perfroms addaptive sampling on fixed regions, changing this raycaster into

foveated render would be a fun project.

5 References

1. Oculus Rift in Action. Davis, P. A., Bryla, K, and Benton, A. 2014. Manning Publications
2. Oculus Developer Guide Version 0.4.4
3. http://github.prideout.net/barrel-distortion/