Project Final Report for ITCS 5235 2019

Proposers: Matthew Mcquaigue, University of North Carolina at Charlotte, {mmcquaig}@uncc.edu

Title: Path Trace Rendering Capable of Non-Euclidean Geometries in WebGL

This project was a great challenge for me in this last month. Before the beginning of this project, I knew I wanted to do something different from what I have done before. I wanted to experiment with techniques and advanced topics I have never read or seen. I wanted to learn something new while being challenged in the process. I decided on a Path Trace Renderer for WebGL to try and create a Non-Euclidean rendering.

Before I talk about the Path Tracer created in this project, I must explain some background information. I started this project wanting to build a Ray Tracer rather than a Path Tracer. I have never used either rendering techniques before this project, so I knew this would be a struggle. I began researching Ray Tracers and how it works/functions. Due to the limited time on this project and couldn’t take my time practicing concepts (I believed you warned me of this difficulty). I decided to try and find a tutorial for WebGL Ray Tracing. I found a lack of tutorials on WebGL Ray Tracing which didn’t teach enough concepts needed in this project. For some reason, where ray tracing tutorials for WebGL lacked, resources for learning Path Tracing emerged.

Throughout my searching on the web, I found an abundance of path tracing help (I don’t know why). Most importantly, I wanted to know if I could create some non-Euclidean-like structure for a Path tracer. I studied the cube intersection algorithm for path tracing and ray tracing and found an extreme amount a similarity. Instead of detecting a ray in a ‘clear’ aberration and stretching the ray about the z axis and setting it back to normal once exiting the aberration, I can detect the intersection of cube walls in the path tracer and stretch the far cube plane in the z direction by a factor.

Following some tutorials and guides, I have created a Path tracer that allows for non-Euclidean-like structures. I first created a 3DObject class that is a Cube or Platform. Since the cubes are going to help me create a stretched tunnel, I had separate classes for a same object (platform is a cube). Cubes for the tunnel are placed in the world at start in a designated location (not by scene Graph). These cubes (3) are arranged in a shape of a tunnel for a user to walk in [Figure 1]. Viewing from the front half of the tunnel on either the right or left side will show an extremely long tunnel touching the wall [Figure 2]. Once the user is near the middle of the tunnel, it will shrink and appear to be small [Figure 3]. These properties give a non-Euclidean-like feel as the user walks through the tunnel. The tunnel will appear long and shrink as you walk through it. This will seem like you walked a long way in a short amount of time. I know that this isn’t true non-Euclidean geometry, but it is as close as I could get.

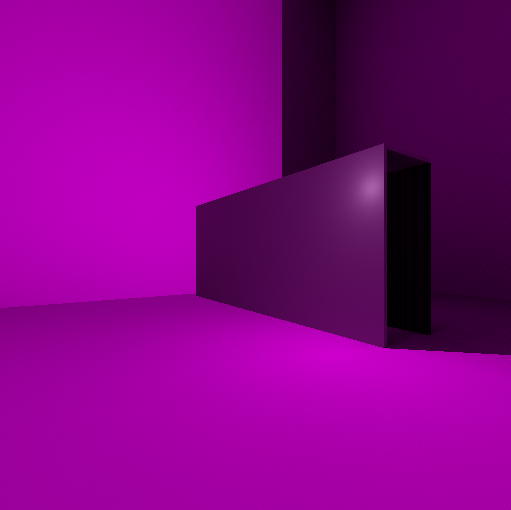
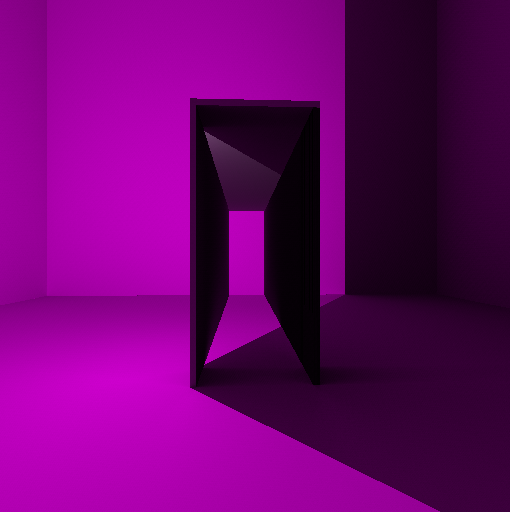


Figure [1] – Front facing Tunnel Using 3 Cubes Figure [2] - Tunnel from left/front side

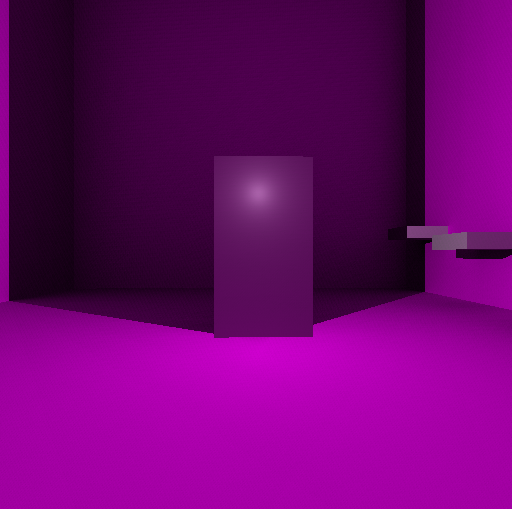


Figure [3] – Looking a tunnel orthogonal to z axis

Platforms are placed on the back wall in a stair like pattern [Figure 4]. I wanted the user to interact with the world, so the user and jump on the platforms to get higher in the room. The platforms positions are determined by the loading of a scene graph. The graph has the first node as the first step and children as the next step. Each step has a local coordinate that is added to the world coordinate of its parent to get its new position relative to its parent. This was the beginning of a game where the user would have to climb to the top of the room to load a new level [figure 5] (levels.js). Collision with the eye of the camera and the platform is computed by checking if the eye is less than the max X of cube, greater than the min X of cube, and repeated for z, when the user jumps.

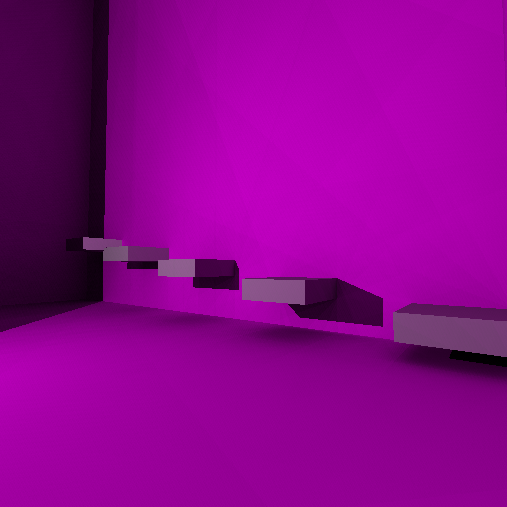


Figure [4] – Looking a platforms on wall Figure [5] – Standing on tallest platform

The camera is a fps style camera that is controlled with the arrow/wasd keys. Up arrow/w moves forward, down/s moves back, left arrow/a rotate about the xz axis by yaw factor, right/d rotate about the xz axis by a factor of yaw. The camera can also jump by translating the camera in the positive y direction when spacebar is pressed.

Users can also click on the shapes to select the shape. When the user clicks a ray is calculated from the eye position and detects an intersection with a shape [Figure 6]. Once this intersection is detected a white outline is casted around the shape in the form of lines. The user then can drag the shape to perform translations and press ‘backspace’ to delete the shape.

Shadows are achieved by getting the shadow intersection for every object. For this calculation, a ray is traced from the hit/bounce \* its normal to the direction of the light. That directional ray calculates which side the intersection is on and casts the float 1.0 from behind the object. Also, for every object a diffuse lighting is calculated by taking the max between 0 and the dot prod of the light direction vector and normal to the surface. This is then multiplied against the light Value and shadow intensity to create the accumulated color.

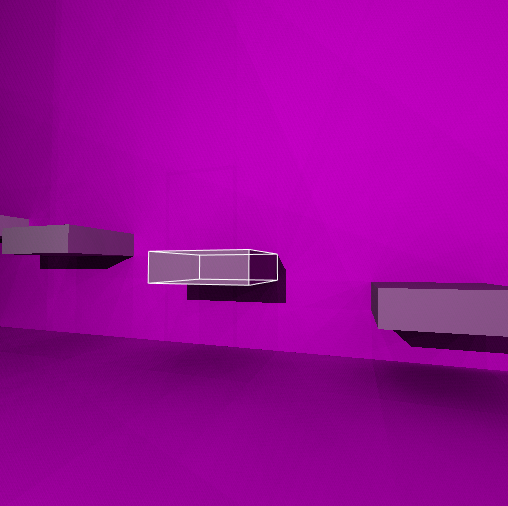


Figure [6] – Selecting a shape

I had a lot of trouble trying to get my own texture mapping and objects rendered within a path tracer (not enough experience). If it helps at all, I added a button on the main page that takes the user to a Cat rendered from an obj file with a texture using traditional rendering techniques [Figure 7] .

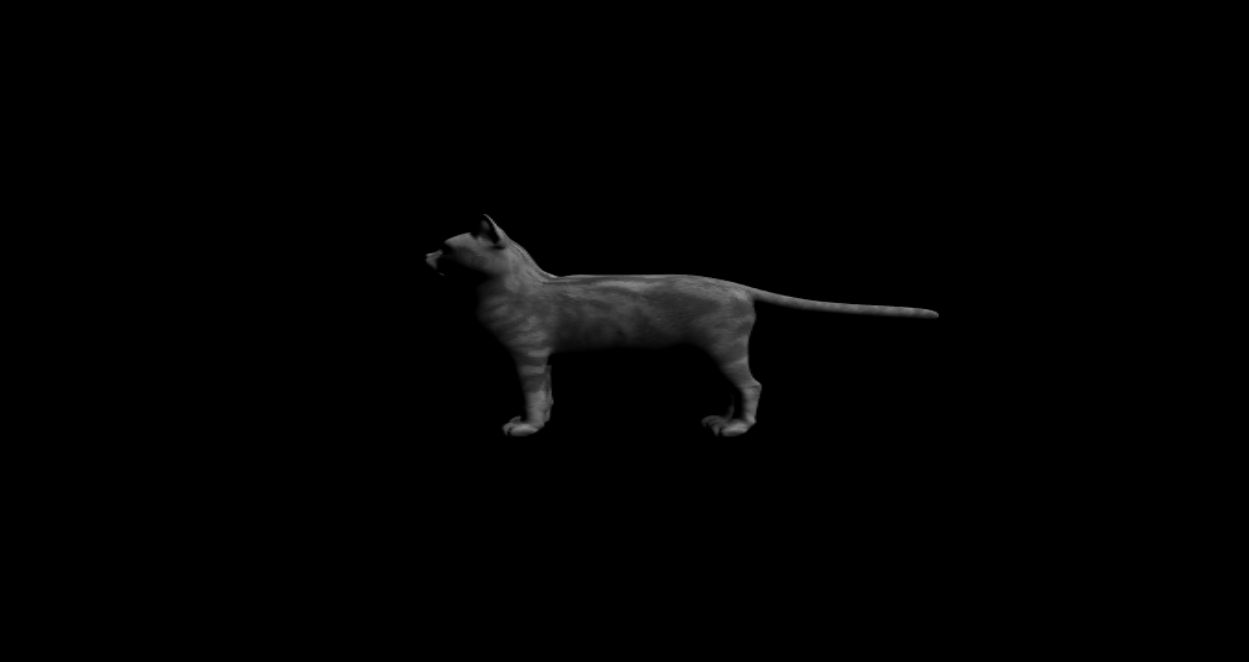


Figure [7] – cat rendering from obj with traditional rendering.

Overall, this was a very fun and difficult project. This is one of the most difficult projects I have worked on because of the lack of knowledge in the area to begin with. I will continue to expand upon this project to create a small parkour game using platforms. I will add more physics and collisions for more realism and continue to research this project to improve on the methods used. I would like to add more non-Euclidean-like features, for instance, rooms that alter based on where the camera stands etc.

References:

[1] P. Rademacher, "Ray Tracing: Graphics for the Masses", ​Cs.unc.edu. [Online]. Available: https://www.cs.unc.edu/~rademach/xroads-RT/RTarticle.html. [Accessed: 16- Mar- 2019].

[2] T. Akenine-Möller, E. Haines and N. Hoffman, Real-Time Rendering, Fourth Edition, 4th ed. Boca Raton Florida: CRC Press

[3] J. Cannon, W. Floyd, R. Kenyon and W. Parry, ​Hyperbolic Geometry, 1st ed. MSRI Publications, 1997.

[4] "Hyperbolic Geometry, Section 5", ​Pi.math.cornell.edu, 2019. [Online]. Available: http://pi.math.cornell.edu/~mec/Winter2009/Mihai/section5.html. [Accessed: 18- Mar- 2019].

[5]”WebGL Path Tracing” , http://madebyevan.com/webgl-path-tracing/, 2010, Evan Wallace

[6]”Caffeine path tracing demo & tutorial”, https://wwwtyro.net/2018/02/25/caffeine.html, 2018, Rye Terrell

[7]”Tesserace 3D Path Tracing Test”, http://jonathan-olson.com/tesserace/tests/3d.html, Jonathan Olson

[8]”WebGL Path Tracing”, http://wulinjiansheng.github.io/WebGL\_PathTracer/, Bo Zhang, Ying, Li

[9] Lafortune, E, Mathematical Models and Monte Carlo Algorithms for Physically Based Rendering, (PhD thesis), 1996.