Real-Time God Rays*

with a Github Implementation Link†

Nataly Moreno
University of California, Santa Barbara
github.com/Kaikat/GraphicsProject
nataly_moreno@umail.ucsb.edu

Figure 1: *Rays of light emanate from the source and pass through a participating medium.*

ABSTRACT

Volume caustics are light patterns that occur when light enters and exits a participating medium scattering light in various directions. These illumination patterns that resemble god rays can greatly add to the visual effect of a rendered scene. However, adding such effects is tedious and non-trivial requiring multiple passes on the scene and the object serving as the participating medium. The results of the multiple passes are the composited into one final image. Since most of these calculations are done on the GPU, these effects can be done in real time.

ACM Reference format:
Nataly Moreno. 2017. Real-Time God Rays. In *Proceedings of CS280: Computer Graphics Final Project conference, El Santa Barbara, California USA, March 2017 (CS280 2017),* 5 pages.
DOI: 10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Caustics refers to the the refraction and reflection of light through a participating medium. A participating medium is a volume in which light can scatter through such as smoke, clouds, water, glass, etc. Calculating these visual effects can be computationally expensive so approximations are made to run in real-time. Adding caustic effects alongside other lighting effects can give a rendered scene a more realistic feel.

This short paper explains a partial implementation of light refracting through an object made from water and shows the intermediary results to achieve a scene like the one in Figure 1. This implementation is based off of (Hu et al. 2010). The scene was created using the base code distributed in CS280 and uses the Sponza model for the scene and the dragon model as the participating medium.

Creating visually pleasing effects often requires mixing various lighting models to achieve various results. In addition to the caustics, this scene uses HDR lighting, the Cook-Torrance BRDF model, shadow maps, and screen space ambient occlusion. A skybox was added to give a more realistic feel when wandering through the scene. There are three point lights in the scene, however the caustics are only done from the perspective of one light while the shadow maps are done for all three lights.

2 OVERVIEW

In order to make this effect various textures need to be rendered from the point of view of the light. The first step in creating volumetric caustics is to render the front face and back face normals and depths to a texture for the object in question. The second step is to render the front positions of the object, the back positions are also needed in a texture but they need to be calculated. The geometry shader takes the results from all of these steps and uses them to create line primitives representing the rays of light. This implementation does not go further than this but the next step is to pick an image to use as a "light texture" and use it with the results

*Produces the permission block, and copyright information
†The full version of the author's guide is available as `acmart.pdf` document

CS280 2017, El Santa Barbara, California USA

DOI: 10.1145/nnnnnnn.nnnnnnn

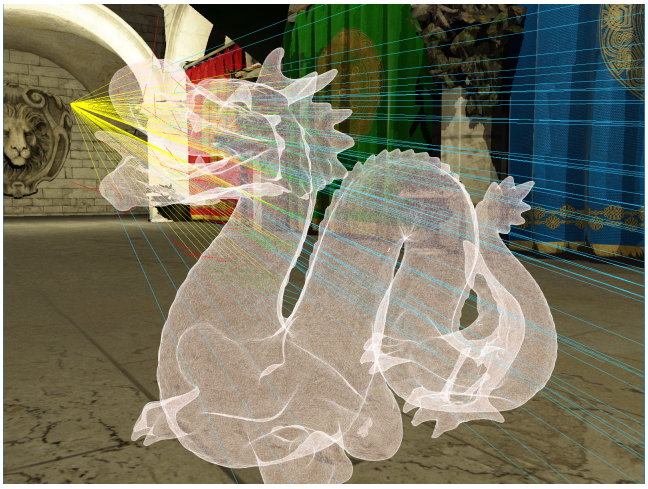

Figure 2: *Yellow rays are rays that originate from the light source, green represents the first refraction that begins from the first intersection point in the object to the back intersection point of the object, blue is the refraction the light takes on exiting the object and finally ending in some place in the scene. Red lines represent the normals of vertices that make up the front face of the object from the light's point of view.*

of the geometry shader to color the rays of light. The results from this step are then used with the scene to create a composite which will put the god rays in the scene. This only gives the god rays and not the patterns that form on the floor.

3 IMPLEMENTATION

In this section the implementation details to achieve the results shown on this paper will be given for each step along with images of the intermediary results. Unforeseen errors that were encountered during implementation will also be discussed per section.

3.1 Creating Position, Normal, and Depth Buffers

This is a multiple pass effect done mostly from the perspective of the light in screen-space. The front face of the object are the parts of the object that are "seen" directly by the light, the back face are the parts of the object that are behind the front faces. We need to create textures holding the positions, normals, and depth for later use. Creating the textures for the front face positions and normals is simple as is saving the depth buffer to a texture. In order to be able to create the back face positions, normals, and depth textures, we need to enable cull face and choose the appropriate front and back while culling. Culling means that part of the object will be cut off. The resulting depths can be seen in Figure 4, the positions in 6, and the normals in Figure 5. Additionally, the scene depth buffer as seen from the light can also be created here for use later, Figure 3.

3.2 The p1 and p2 Textures

The p1 and p2 textures refer to Figure 8. Creating the texture for p1 is not trivial. P1 is the exit point of the first refraction. This

Figure 3: *The texture for the depth of the scene as seen by the light.*

Figure 4: *The texture for the front depth on the left and for the back depth on the right.*

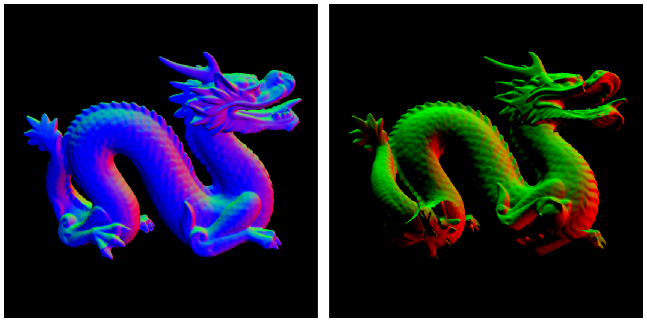

Figure 5: *The texture for the front normals is on the left and the back normals are on the right.*

calculation for the refraction is explained in De Greve (2004) which can also be used to calculate P2, the point where the light finally hits the scene. Once the direction of the refraction is calculated, the exit point position is found using the textures calculated before through ray-marching. This ray marching method is explained in

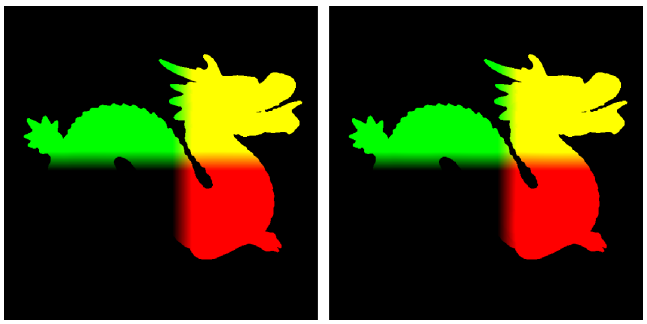

Figure 6: *The texture for the front positions on the left and back positions on the right.*

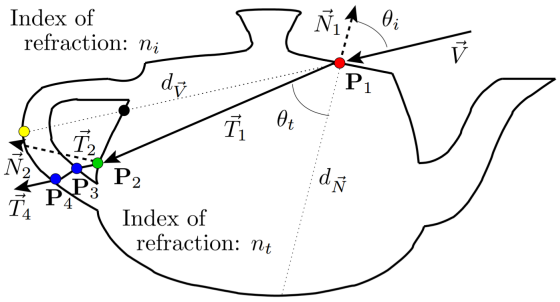

Figure 7: *This image shows a visual that helps calculate the direction in which the exit point in the object is as seen in (Wyman 2005). The direction that needs to be calculated is T1.*

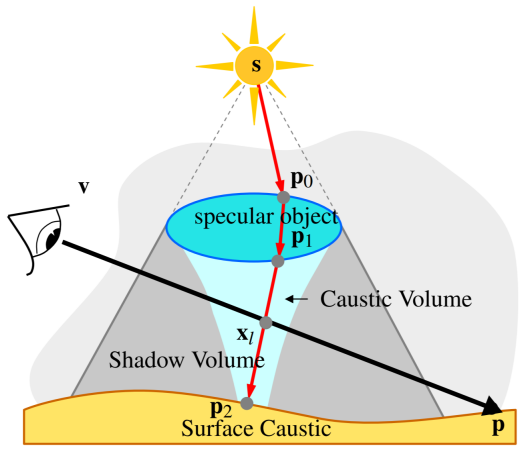

Figure 8: *This image shows p1 which is the point where the light leaves the medium and p2 is the point where the light hits the surface as seen in (Hu et al. 2010).*

(Wyman 2005). The direction ray is used along with the starting point and the ray is marched until the calculated depth at the step matches or surpasses the depth in the depth buffer. For p0 to p1 the back depth buffer is used to compare against and the scene depth

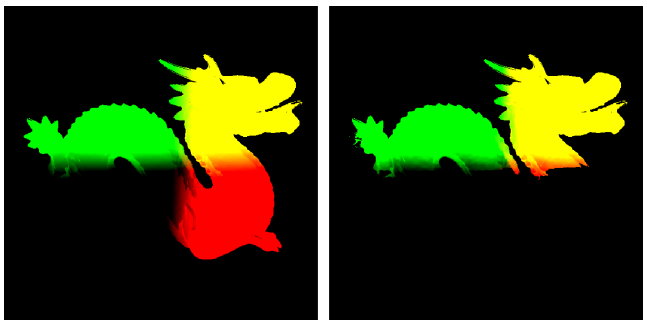

Figure 9: *The texture for p1 on the left and for p2 on the right.*

buffer is used for p1 to p2. The results of p1 and p2 can be seen in Figure 9.

This section was particularly challenging because the refractions kept going in the wrong direction. Furthermore, once the refraction directions were working, some rays that did not hit any depth ended up in strange places so they had to be removed. (Hu et al. 2010) mentions that they have the ray hit an imaginary sphere instead. For simplicity, in this implementation, if they were invalid they were removed as there are enough rays to make up for it without it being noticeable.

3.3 The Geometry Shader

Once all the information is saved in their respective textures, the results are used in order to create new geometry. The geometry shader is after the vertex shader but before the fragment shader. We need to pass all the results into this shader. Using the p1 and p2 textures we are able to create new vertices for them and draw lines between them. These lines resulting from the geometry shader can be seen in Figure 10.

Debugging the correctness of the lines is challenging without visuals. Therefore, in order to aid the process, the dragon was drawn as a clear object in the scene along with its wireframe to make it easy to see if the normals were showing up properly. Being able to render the normals properly helps to know if the refractions were calculated properly.

4 CONCLUSIONS AND FUTURE WORK

Graphics is a very challenging field that requires a strong math, physics, and programming foundation. Even though a full implementation was not possible in two weeks due to the difficulty of graphics programming and the tricky math invovled, the setup for adding the final caustics touch was completed. One of the most challenging parts of this assignment was debugging since a lot of the things that happen within the shader are unreachable. One way to debug was to draw the contents of temporary textures to the screen. Another helpful way to debug was to use Nsight to see the textures being created which is how all the texture images were obtained. Nsight does not help debug completely and since OpenGL has a lot of hidden state apitrace was also helpful in finding broken state within the code.

Future work would be to add the corresponding colors to the lights to make them look like actual god rays as well as doing the

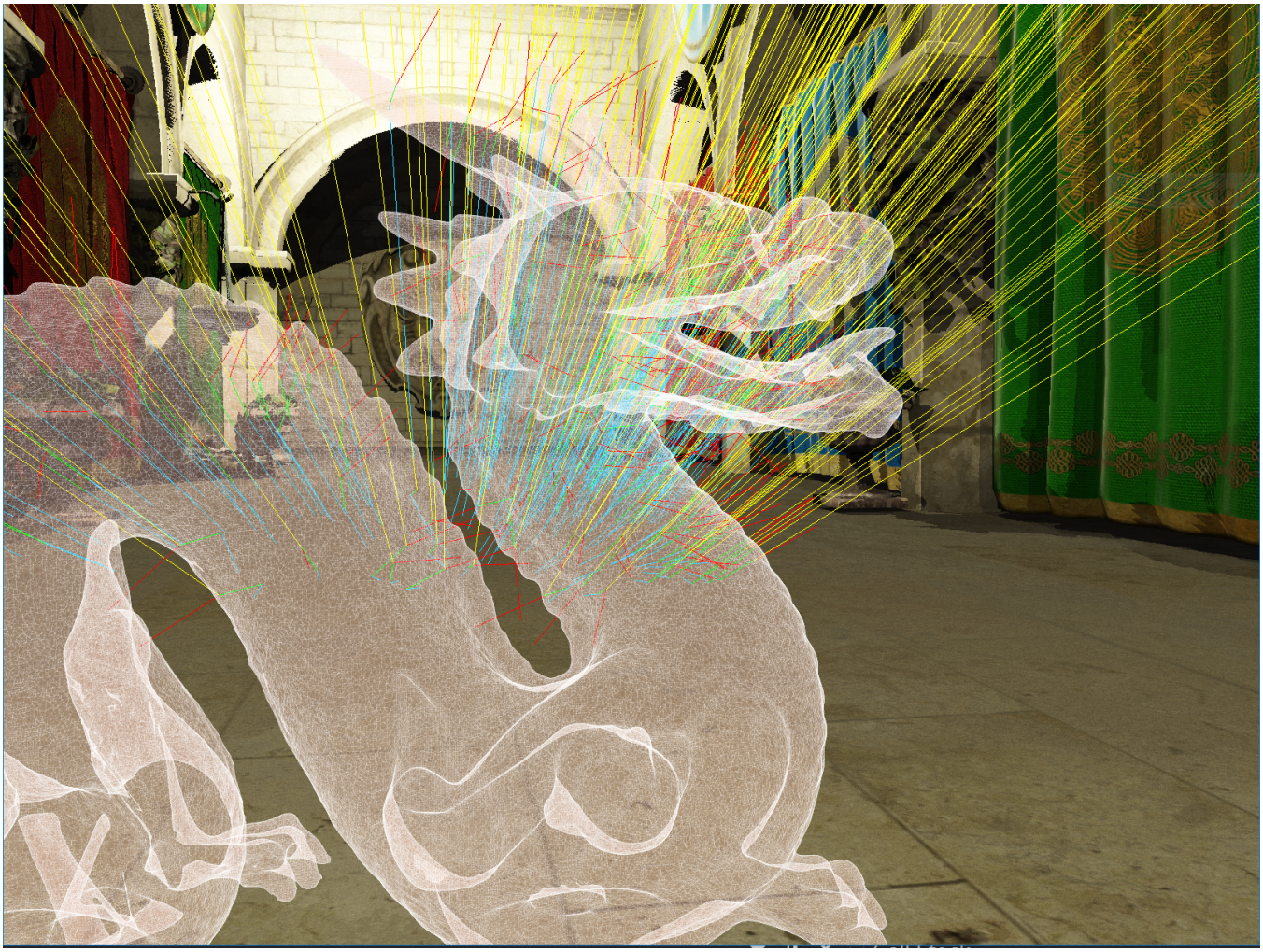

Figure 10: *Results from the geometry shaders. The wireframe over the clear transparent dragon and the colored light primitives representing the rays of light were created using geometry shaders.*

floor caustics for the full visual effect. The final steps were not clear from the paper and were not reached in time, but they will be implemented in the future.

REFERENCES

Bram De Greve. 2004. Reflections and refractions in ray tracing. *on line at http://graphics. stanford. edu/courses/cs148-10-summer/docs/2006–degreve–reflection_refraction. pdf* (2004).

Wei Hu, Zhao Dong, Ivo Ihrke, Thorsten Grosch, Guodong Yuan, and Hans-Peter Seidel. 2010. Interactive volume caustics in single-scattering media. In *Proceedings of the 2010 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games*. ACM, 109–117.

Chris Wyman. 2005. An approximate image-space approach for interactive refraction. In *ACM Transactions on Graphics (TOG)*, Vol. 24. ACM, 1050–1053.

Figure 11: *The clear dragon in a scene using the Cook-Torrance BRDF model showing the skybox. The lights were given a yellow color to give the scene a warm effect. HDR was used to give the scene a little more contrast with the gamma and contrast.*