Portals and Gravitational Lensing Using Ray-Tracing

Sean Dunigan
Rensselaer Polytechnic Institute
dunigs@rpi.edu

Mira Pianta
Rensselaer Polytechnic Institute
piantm@rpi.edu

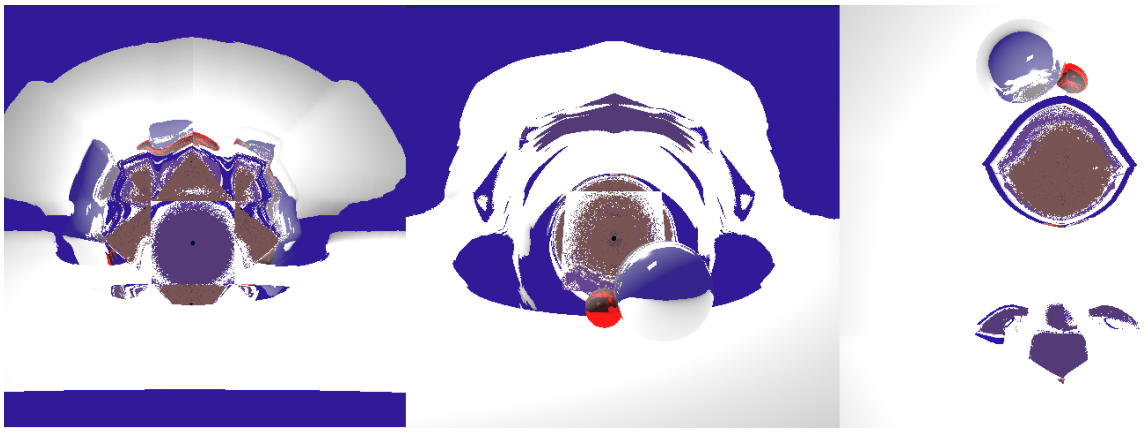

Figure 1: Three views of a pair of portals facing opposite directions, one with mass and one without, along with two reflective spheres which are in front of the massive portal. On the left is the view of the non-massive portal, in the middle is the view of the massive portal, and on the right is the bird's-eye view of both portals. The visible light effects are that of light being sucked into the non-massive portal, light orbiting the massive portal, and light from the backside of the non-massive portal taking the shortest route to the front side. These took 2 hours to render altogether.

1. Abstract

In this paper, we implement gravitational lensing and portals, and we investigate how the two can affect one another. As each of these effects deals with spacetime, we have combined them into a single simulation using a modified ray-tracing system. If we assume that spacetime remains continuous through a portal, then it stands to reason that the curvature of spacetime on one side of a portal can be affected by a large amount of mass on the other side of the portal. Thus, we can expect a black hole on one side of a portal to affect and bend the light on the other side of the portal. Our work achieves this effect through a one-sided distance minimization method.

2. Introduction

Gravitational lensing is an effect that is formed when light passes by a large amount of mass, causing it to bend as it follows the curvature of spacetime. On the other hand, portals are an effect that act as a “bridge” between two points in space. These are essentially wormholes, in which the curvature of spacetime links two points and allows for direct travel between them. The former is a known phenomenon in space, whereas the latter is a hypothetical structure that has not been observed in reality. In this paper, we present the following contributions: a calculation of ray-based gravitational lensing in three-dimensional space, the transformation of rays through the use of portals or wormholes, and the integration of the two using distance minimizing formulas.

3. Related Work

Although we struggled to find related work that investigated the goal we have attempted to achieve, given that our problem combines two different phenomena, we did find work done on the individual problems being approached. Subileau, Vanderhaeghe, and Paulin approach the portal problem by solving the question of how light, as a ray, might travel through a portal from its input to its output plane[1]. They developed their

algorithm in hopes of creating a simpler way for artists to add lighting effects to a scene without the need to perfectly align them to the desired location, using portals to transport the effects instead. Their diagrams and methods to calculate light ray transport alteration were referenced in the creation of portals for this paper.

Additionally, Thomas Rinsma developed a recursive portal algorithm using the OpenGL stencil buffer[2]. His method took inspiration from the video game *Portal*, which we also hoped to mimic. Though we did not use the stencil buffer for our own approach, his approach to portals as a reflection problem was the inspiration for our own approach, just through using ray-tracing rather than OpenGL's stencil.

Much work has been done with regards to approximating gravitational lensing. Killedar et al. developed code to perform a ray-traced simulation of gravitational lensing without utilizing the multiple-lens approximation used in previous works[3]. Instead, they attempt a direct numerical integration of the null geodesic equations, and they compare it with results that use the multiple-lens approach. Their method is computationally heavy due to its use of fast Fourier transforms when calculating the gravitational potentials at each point in space. While they attempt a more physically-accurate simulation of gravitational lensing, our method will avoid directly solving the null geodesics by approximating them instead.

Finally, we look at a previous gravitational lensing project done by Mack Qian that closely follows the method we have attempted[4]. Qian's method approximated black hole renderings by tracing the path of photons as they are affected by Newtonian gravitational forces. Qian also simulated the image distortion of the accretion disk that can form around a black hole. While our method does not render an accretion disk, it will also approximate the null geodesic equations by treating photons as a low-mass particle that can be affected by Newton's law of universal gravitation.

4. Portals

In this paper, the term "portal" refers to a wormhole-like pair of one-sided planes that are connected to each other through space. Portals behave visually as if they are each other's backsides, transporting light directly from one finite plane to its pair, and continuing as the light normally would. This behavior can be seen in Figure 2.

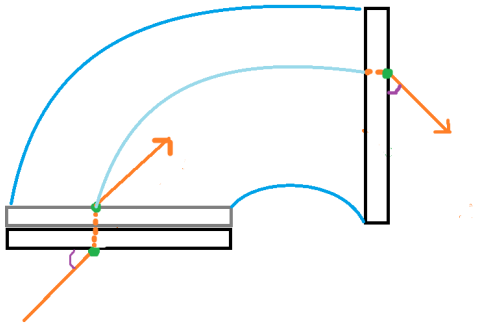

Figure 2: A top-down view of portal behavior. The right plane is transformed from the left plane's back side, retaining position and ray information through the transformation.

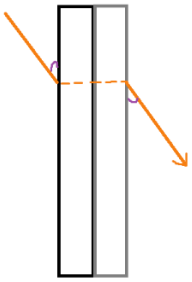

Figure 3: A side view of a ray passing through a finite plane as if its front and back sides were separate faces. This is how rays pass through a portal, with the faces being spatially separated from each other as well.

In this approach, the method has been implemented on top of an existing implementation of a ray-tracer, including ray-face intersections. Portals are treated as a special type of one-sided face, which has a connected "Pair" face. The pair connection is determined through the input .obj file, which takes in an integer as a pair value when creating a portal face. By storing meshes in this format, portals can be arbitrarily paired to one another while also allowing for more than one pair of portals to be rendered in a scene. By using the existing intersection method, which determines whether a ray has intersected with a plane, and then whether the point of intersection is within the

face's area, the intersection point can be stored for use in calculation.

The existing implementation renders reflections with a maximum depth value, so the cast ray reflects off of surfaces for a given limited number of recursions. In our approach, ray-portal interaction behaves in the same way, with one ray-portal intersection acting as a transformed ray-reflection intersection: rather than reflecting off the same point, the ray "reflects" off the location it would be on the paired portal face, retaining its direction according to the normal of the paired face. This is done by retrieving the local translation of the point with respect to the intersected plane's centroid, multiplying that translation by the rotation matrix corresponding to the transformation from the intersected plane to its pair, and then adding the resulting vector to the paired plane's centroid, as seen in Figure 4. Similarly, the direction of the incoming ray is also multiplied by the corresponding rotation matrix to get the direction that the outgoing ray should come out of the paired portal in order to keep continuity. On its own, this calculation is insufficient. A directional ray only calculated by the rotation matrix may return the incorrect angle when rotating between portals at a 180 degree angle across the y-axis, for example, due to the x or z location coordinates being reflected, but not the x or z direction coordinates, which would cause the portal to act instead like a mirror. Due to this case, before computing the rotated direction vector, the vector is first reflected across the intersecting plane's normal vector to get the direction that the rotation matrix should be applied to. The rotation matrix is calculated outside of the program from the angle of rotation that would be required for the faces to be back-to-back if they were at the same center coordinate.

With these calculations, our program is able to continuously trace a ray through a portal, retaining the position and direction across the pair of portals. This behaves correctly in both directions, meaning that portals in this implementation are two-directional rather than one-directional, and can continue recursively between portals that face each other (Figure 5). To stop these instances from creating an infinite loop, the "reflection" algorithm recurses depending upon a maximum depth value, determining the number of times a single ray can be moved and retraced. In order to differentiate between the two "sides" of the portals, one side is assigned a blue-colored material, and the other is assigned an orange-colored material, in order to help visualize which portal is receiving which hits when looking at recursively bouncing images.

```
Vec3f translate =
 signed_difference(intersection,
 portal1_centroid);
translate =
 MatrixMultiply(rotation_matrix,
 translate);
Vec3f transformed_pos =
 portal2_centroid + translate;

Vec3f ray_dir = incom_ray_dir -
 (2.0 *
 (incom_ray_dir . portal1_normal) *
 portal1_normal);

ray_dir =
 MatrixMultiply(rotation_matrix,
 ray_dir);
```

Figure 4: Pseudocode example of math required to transform a position vector (`intersection`) from the intersected portal (its centroid denoted by `portal1_centroid`) to its paired portal (its centroid denoted by `portal2_centroid`) according to the pair's inputted rotation matrix. Rotates incoming ray's direction (`incom_ray_dir`) across the reflection, then by the rotation matrix.

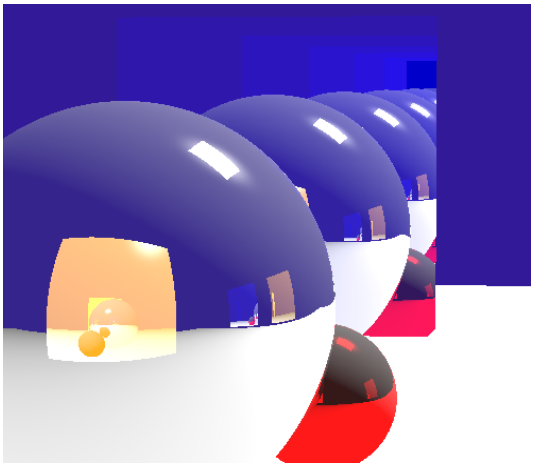

Figure 5: A pair of portals facing each other with two reflective spheres in between them and a second pair of portals in the scene to the right. The camera is facing the blue side of the recursive portals, and the orange side can be seen reflected in the larger sphere.

5. Gravitational Lensing

Our implementation begins by recognizing that black holes are not the sole reason for gravitational lensing: rather, it is a consequence of having a large amount of mass near a point in space. Thus, we extend a primitive class to include information regarding the mass (in kilograms) and the center of mass of a primitive object. This allows us to use arbitrary shapes, such as cubes, cones, and rings, although we focus mainly on spheres for the purposes of this work. Primitives also keep track of two special variables, which indicate the object's radius of influence and its Schwarzschild radius.

The radius of influence, also known as the sphere of influence, is an idea used in prior works to limited success. We built upon this idea by choosing the size of the influence radius dynamically, such that the radius equals the distance at which the force exerted on a photon of light becomes equal to a certain Newtonian force value. For our simulations, we chose within the range from 0.001N to 1.0N for the minimum force value. Setting a lower value allows for more accurate renderings, while higher values tend to be faster to generate. Although Qian chose to not use this technique due to its inaccuracy[4], we kept it for the other benefits it provides: such as the early termination of rays that exit the radius of influence and are sent into empty space. We calculate the radius of influence as shown, where G is the gravitational constant, M is the mass, and f is the minimum force that must be applied to a photon before we should consider accelerating the photon:

$$ir = \sqrt{\frac{GM}{f}}$$

The Schwarzschild radius is an incredibly vital piece of our implementation. When a photon of light is within this distance of an object, we terminate the operation and consider the photon "absorbed". Like the radius of influence, this provides an early termination for a subset of rays in the scene. It is important to note that the Schwarzschild radius and the size of an object are independent of each other. In our implementation, a sphere could end up having a radius greater than its Schwarzschild radius. In this situation, while the simulated photon will be attracted to the center of mass, it will not be absorbed, and thus a black hole will not be rendered. This allows us to simulate gravitational lensing around arbitrary objects (such as galaxy clusters) rather than being limited to black holes. We calculate the Schwarzschild radius as follows, where G is the gravitational constant, M is the mass, and c is the speed of light:

$$sr = \frac{2GM}{c^2}$$

As these radii make use of the gravitational constant and the speed of light, it is worth noting that we used unrealistic values in our simulation. Instead of using 6.6743 x 10^{-11} $m^3kg^{-1}s^{-2}$ and 299,792,458 m/s respectively, we use 1.0 $m^3kg^{-1}s^{-2}$ and 45.0 m/s. As the ratio of the speed of light to the gravitational constant is approximately 4.5 x 10^{19}, we chose these specific values to make the difference in scale between our values and the real values roughly 10^{18}, on the assumption that this would help to provide more-realistic results.

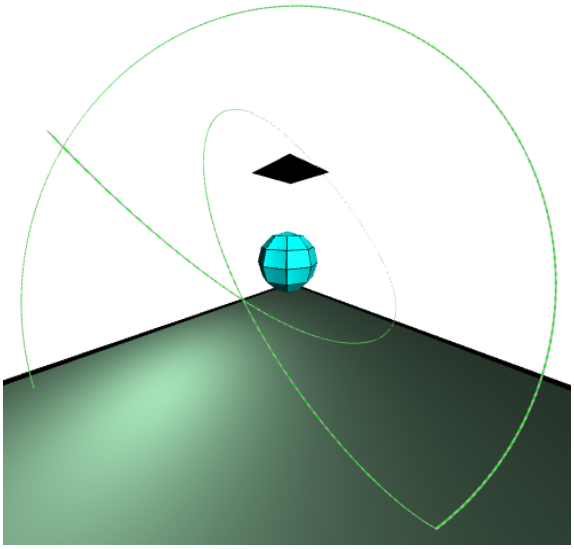

Figure 6: A photon's path as it circles around a dense object, eventually terminating upon hitting a surface after a single bounce. Notice that the path is made up of alternating green and gray segments.

At a high level, our implementation works by splitting rays of light into smaller segments, and by treating each ray of light as a single photon traveling through space (as shown in Figure 6). We

calculate the acceleration on each photon due to masses using Newton's law of universal gravitation, while assuming that each photon has a mass of 1 kilogram. This is unrealistic as a photon is considered a massless particle, but for the sake of our implementation, this allows us to use Newtonian physics as a good approximation. This is preferable to solving the null geodesic equations as done in [3]. When photons are influenced, the calculated acceleration is "fixed" before being applied to the photon's velocity. We implement the step detailed by Orban[5]. As Orban explains, because a photon moves at the speed of light, its velocity vector must be constrained to a circle around it with a radius equal to the speed of light. However, our acceleration vector may have a component parallel to the direction of the velocity vector. This would unnecessarily cause the photon to speed up or slow down. To help mitigate this issue, we project the acceleration vector onto the velocity vector, determine its parallel component, and use that to find the vector component of acceleration that is perpendicular to the velocity. At each timestep, we update the velocity using only this perpendicular component. While this does result in the velocity becoming "faster than light", the error is smaller and easily mitigated by normalizing the resulting velocity and then rescaling it to be equal to the speed of light.

We have experimented with using both Euler's method and Runge Kutta (RK4) for improving the accuracy of our resulting images. We found that while RK4 tended to create smoother arcs as the light rays bent in space, it also took significantly longer to render. While Euler's method requires one check of the gravitational influence at a point in space, RK4 requires four separate checks. This is very slow, and the generated images seemed to indicate very little improvement in overall quality. As a result, we made the choice to stick with Euler as the default option for our implementation.

After a photon's position and velocity have been updated, we check to see if it has fallen within the Schwarzschild radii of any objects in the scene. If so, we terminate the ray immediately and render a black pixel. If it was not terminated, we continue to trace the ray until we fall into a black hole, intersect with an object, or run out of iterations. The final method *CastGravitationRay* is a near-replacement for our original *CastRay* method, with the simple addition of a few more arguments and the consideration of a "photon termination" return value.

6. Gravitationally-Linked Portals

The final (and perhaps most interesting) piece of our work is our attempt to combine portals with gravitational lensing. Because our methods each rely on raytracing, this appears to be an easy problem to solve. In our method, when we calculate the net gravitational acceleration at a point in space due to masses in the scene, we also iterate over every portal in the scene. At each portal, we "enter" at the point on its face closest to the point in space, and then do the same acceleration calculations at the other side of the portal. This continues recursively for a specified depth. What makes our method interesting is that we always look for the closest point on the front face of the portal. This means that if a point is behind the portal, it will attempt to wrap around by finding the closest point on one of the edges of the portal instead. Our expected result was to see the light around the back edges of a portal appear to "squish" as the light attempted this bending process. As shown in Figure 7, our first rendering that combined portals with gravitational lensing, we were relatively successful.

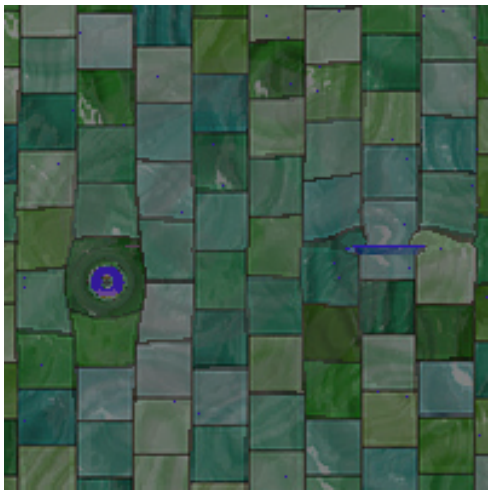

Figure 7: Pictured are a dense object and orange portal (left) and a blue portal (right). Gravitational lensing is seen around the mass, while gravitational forces appear to bend light above the blue portal around it towards the front face.

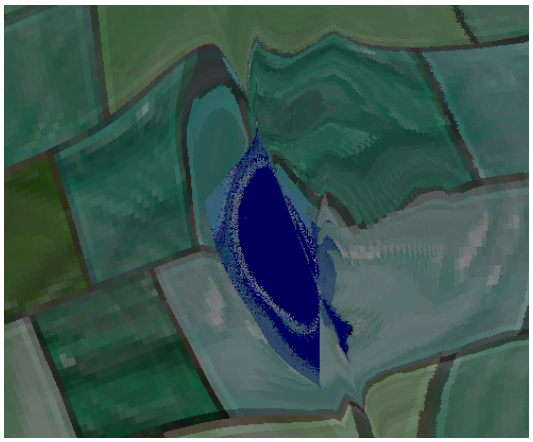

Figure 8: A closer image of the blue portal shown in Figure 7, shown from a bird's-eye view. On the front face of the portal, the light can be seen bending according to the mass on the other side of the orange portal, and on the back face of the portal, the light can be seen attempting to take the shortest path to the front face as it gets pulled in.

In order to get the closest point on the front of the paired portal for each point around it, we implemented David Eberly's "Distance Between Point and Triangle in 3D" algorithm in the function *GetClosestPoint*[6]. The face of the portal is split into two triangles and, using the closest of the two triangles, Eberly's algorithm is applied. First, we set the triangle onto an st-plane with 7 different sections so that the minimization equation can be applied to the corresponding section correctly. These sections are the triangle itself, for which the closest point is the point from a straight line to the triangle; the 3 edges, for which the closest point is the point from a straight line to its closest edge; and the 3 vertices, for which the closest point is the vertex of the triangle itself. Once the section containing the requesting point is found, the algorithm minimizes the s and t values according to this section, multiples these values by edges 1 and 2 of the triangle, and then adds these results to the vertex connecting these two edges of the triangle to get the closest point. The resulting paths of least distance that the points take to get to the portal from this algorithm can be seen in Figures 9 and 10, in which the green lines denote the shortest path.

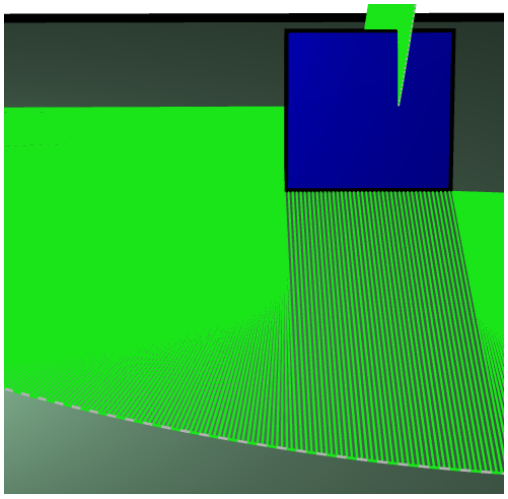

Figure 9: Points taking the shortest path to reach the front face of a portal.

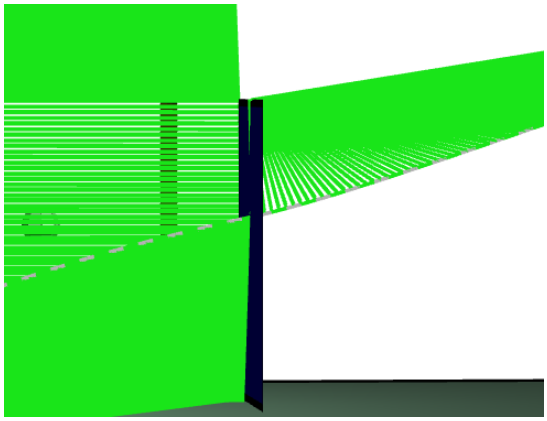

Figure 10: Points on the back side of the portal taking the shortest path to the edge of the front face, rather than the shortest path to the back face.

7. Results

Figure 7 took 579 seconds to render, which was significantly longer than the same scene when rendered with portals and gravitational lensing separately. We suspect that the main bottleneck of our implementation was the use of matrix multiplication in ray-portal transformation. At each timestep, a photon must determine its gravitational acceleration in space. This requires finding the closest points on every portal and using matrix multiplication to determine the corresponding point before repeating the gravitational force calculations. Other renderings include those shown in Figure 1 and Figure 18.

8. Future Work

We came up with a few more methods for determining the gravitational force through portals that could yield results better than our own.

First, the portals themselves could be given an interpolated mass value based on the masses of all nearby sources of mass. Then, when determining the net gravitational force, one could simply include the masses of the portals themselves into computation. This method would likely be the fastest (requiring a single precomputation, and recomputation in the event of scene changes), but it would also be a gross approximation. Due to the nature of the gravitational forces, we would expect the magnitudes of the forces to be radially-related and their directions to be based on the directions of the associated objects. However, applying a mass value to the portal would remove directional information and result in inaccuracies in the force magnitudes at certain positions on the portal face.

Our second method seems much more feasible. We propose a photon-mapping-like solution that we have nicknamed “graviton mapping”. Gravitons are a hypothetical quantum particle that carries the gravitational field. Rather than applying a single interpolated mass value to each portal face, one could scatter a uniform array of gravitons to the face of each portal at the start of simulation. Then, for each graviton, we could determine and store the gravitational force that would affect a particle at that point in space. This can be precomputed, and recomputed in the event of substantial sources of masses being moved, created, and destroyed within the scene. As the simulation runs, we would find the closest point on the face of a portal as normal. However, instead of entering the portal, we would find the 4 closest gravitons and bilinearly interpolate their forces. This would give us an approximate gravitational force at our specific position. Increasing the number of gravitons on each portal face would linearly increase the precomputation step, but result in more-accurate force computations.

Alongside these improvements, there are a number of bugs that we would like to look into in the future. As shown in Figure 14, we encountered an issue where light would enter a circular orbit around a dense object. This is a realistic phenomenon that takes place on the photon sphere, at a distance of 1.5 times the Schwarzschild radius. Although realistic, this takes up processing time as the ray runs through all allocated iterations. A new method to terminate these rays early, or to prevent photons from getting trapped in this radius would be ideal for further work.

9. Conclusion

We have presented methods for rendering portals and gravitational lensing. Due to the expense of our ray-tracing implementation, our approach is only feasible for generating static images. We also found success in integrating these effects together to create a variety of interesting scenes. While such renderings may not be useful in realistic simulation (as portals are hypothetical), our techniques may find a use in popular media.

10. References

[1] Subileau, T., Mellado, N., Vanderhaeghe, D., and Paulin, M. 2015. Light Transport Editing with Ray Portals. Computer Graphics International 2015.

[2] Rinsma, Thomas. Rendering Recursive Portals with OpenGL, 19 May 2013.

[3] Killedar, M., et al. Gravitational Lensing with Three-Dimensional Ray Tracing. ArXiv.org, 15 Oct. 2011, https://arxiv.org/pdf/1110.4894.pdf.

[4] Qian, Mack. Approximate Black Hole Renderings with Ray Tracing. May 2021.

[5] Chris Orban. Black hole derivation. https://www.asc.ohio-state.edu/orban.14/stemcoding/blackhole_derivation_slide2.png, 2019.

[6] Eberly, David. Distance Between Point and Triangle in 3D. Geometric Tools, September 11, 2020.

11. Work Division

Sean Dunigan - Approximately 30 hours
- Gravitational Lensing
- Gravitation/Portal Force Computation

Mira Pianta - Approximately 30 hours
- Portal Implementation
- Ray-Portal Matrix Transformations
- Distance Minimization Algorithm

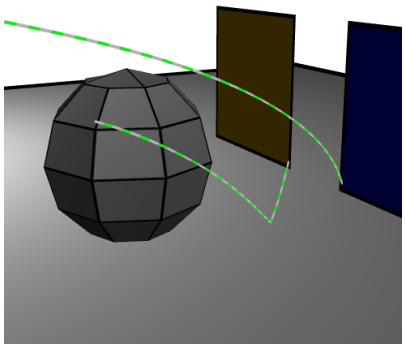

Figure 11: A ray traced from the top left corner of the image being attracted to a mass, entering the blue portal, coming out of the orange portal, bouncing off the floor, and being bent again around the mass.

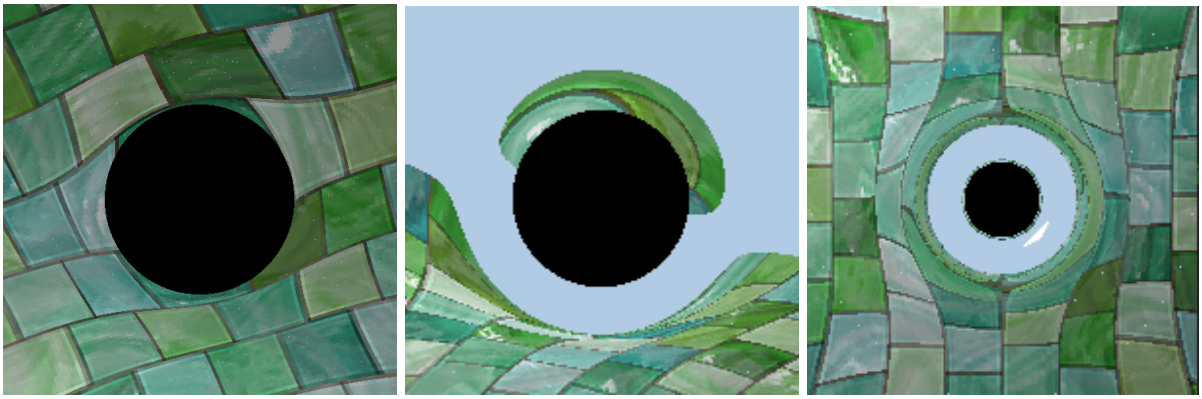

Figure 12: Several renderings of a black hole with different viewing angles and parameters. In the left image, the black hole has a relatively weak force and is close to the ground plane, causing a clean gravitational lensing effect. In the middle image, a black hole is being viewed from the side with the ground plane below it. As rays travel over the top of the black hole, some are pulled down to hit the ground plane, creating a visual texture "flip" effect. In the right image, the black hole is viewed from top-down with the textured plane directly below it. Because the black hole is far enough from the plane, rays that are nearby bend completely around the black hole and return in the direction of the camera.

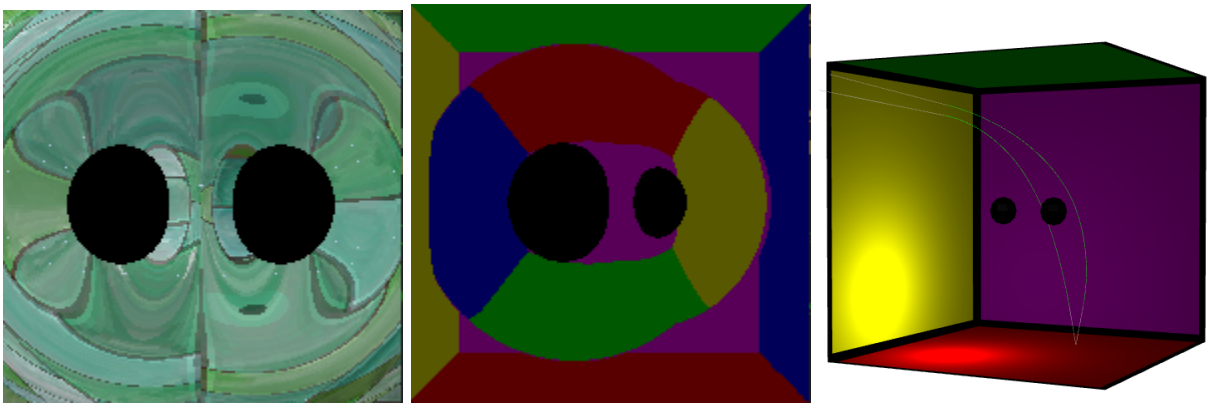

Figure 13: Several images showing the interaction between multiple dense objects in the scene. In the left image, two equally-dense black holes cause a symmetrical distortion in the texture. In the middle image, two unequally-dense black holes cause an asymmetrical lensing effect in our "color box" environment. In the right image, the path of a photon in the color box environment is visualized as it creates the "flip" effect.

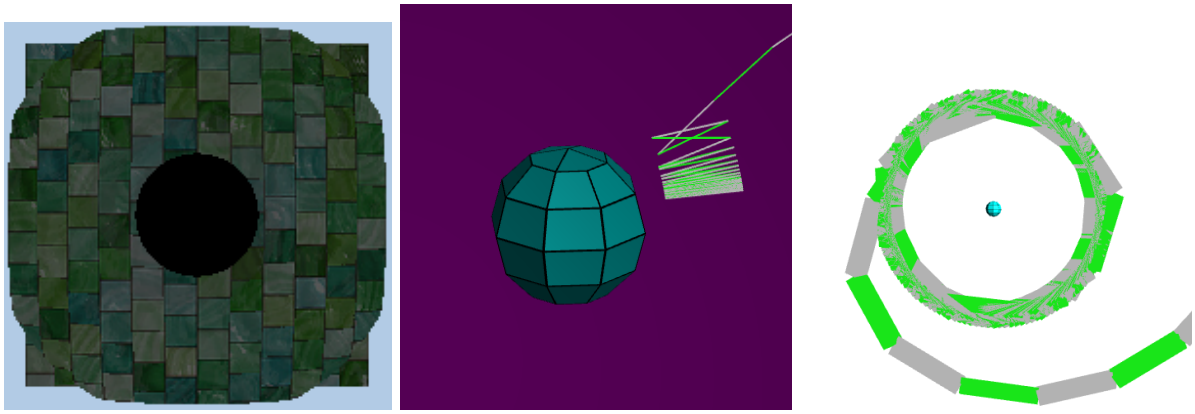

Figure 14: Images showing various issues encountered while implementing gravitational lensing. In the left image, our "sphere of influence" was improperly tuned, causing a "magnification" effect rather than the lensing effect we hoped for. In the middle image, the combination of a higher timestep and the omission of parallel-acceleration removal causes a photon to bounce back and forth when near a dense object. In the right image, a photon appears to enter an orbit around a black hole. This may simulate the real-world phenomenon in which light can enter a circular orbit at a radius equal to 1.5 times the Schwarzschild radius of a black hole, found at the "photon sphere".

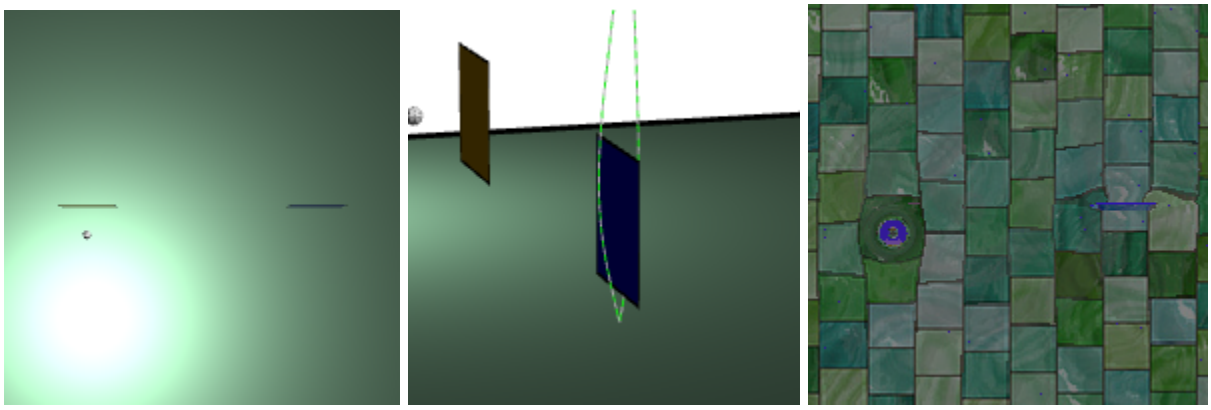

Figure 15: Images showing how our scene appeared for our first attempt at rendering light being affected by gravitational forces through the face of a portal due to a dense object.

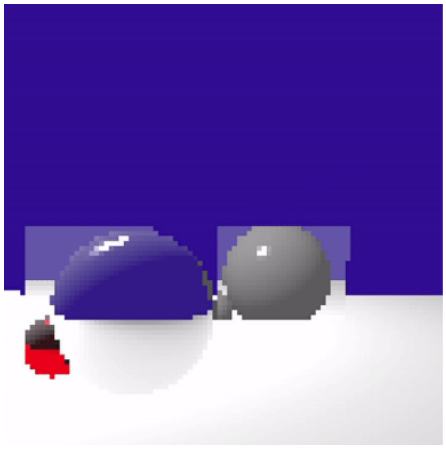

Figure 16: Image showing how a portal with an incorrectly-calculated ray direction would appear. In this image, the reflection ray was not yet precalculated in getting the proper portal ray, so the image in the rightmost portal appeared as a mirror reflection of the spheres rather than as a transformed image.

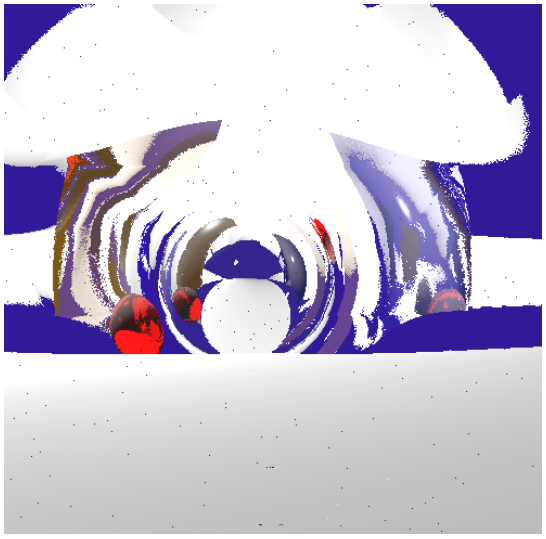

Figure 17: First attempt at giving a portal mass. Since the portal's center was not explicitly determined for faces as it was for spheres, the center of mass was instead positioned at (0,0,0), resulting in the entire image being warped around the origin.

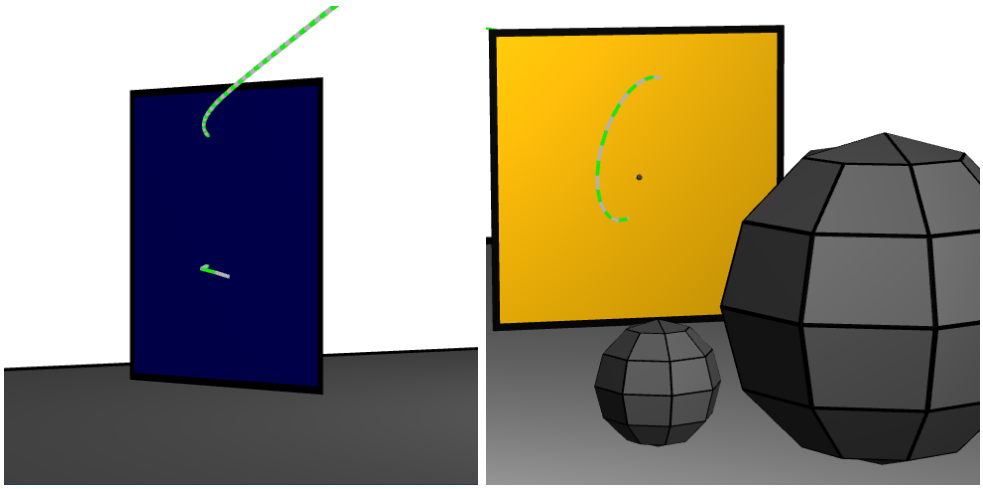

Figure 18: A single visualized ray traced from the top right corner of the blue face of the portal. The ray can be seen curving into the blue portal due to the mass of the orange portal. It then begins orbiting around the orange portal's center when it comes out of the other side, going back into the orange portal and popping out on the blue portal again. The ray then curves a final time due to the force of the paired face's mass, heading back into the portal and stopping due to the maximum depth value of 2.