# CS419 Final Project

Nishant Bhushan(bhushan3), Richard Gunter(rgunter3), Huizi Hu(huizihu2)

**Abstract**— Features implemented for the ray tracer are: Motion Blur, Volumetric Rendering and Image-based Texture Mapping. Volumetric Rendering is a set of techniques used to render a 2-D projection of a 3-D discretely sampled data set . Motion Blur enables one to render images of objects in motion. Texture Mapping is a technique used to map a texture onto an image using uv-coordinates for that image.

**Index Terms**—Ray Tracer, Volumetric Rendering, Motion Blur, Texture Mapping

**Repo Link**—https://github-dev.cs.illinois.edu/rgunter3/419-Final

## Implementation

The Ray Tracer is implemented in C/C++. The hardware information of the machine used to render each of the final images is:

i7-7700K LGA1151 intel core.

GPU (not used): Nvidia GTX 1080 GamerX, 48GB RAM

A Windows Operating System (OS) was used to render each of the final images.

## 1 Motion Blur

Motion Blur is an effect that occurs when the image being rendered changes during the recording of a single exposure. This can occur if the object or camera is in motion and if the length of time of the exposure is sufficiently long. It gives a sense of realism and high speed.

Motion blur for a physical camera happens because the "shutter" (exposure) is open for a certain amount of time. Therefore, when a camera creates an image, that image represents a scene over a period of time, not at a single instance of time. The exposure period is usually small enough so that the image rendered appears to capture an instantaneous moment. However, a fast-moving object or a longer exposure time can result in artifacts that are blurred along the direction of relative motion. As objects in a scene move, an image of that scene must represent an integration of all positions of those objects, as well as the camera's viewpoint, over the period of exposure determined by the shutter speed. This was implemented by allowing the ray class to accommodate rays that exist at a specific time and by passing the value of time in the constructor. After that, we updated the camera class to store information about the shutter open and shutter close time. This allowed us to create rays for a random time in that interval and then cast those rays into the scene. A new *moving_sphere* class was added to be able to calculate the hit point of a moving sphere. The difference between this class and the sphere class is that the center of the moving sphere is dependent on the time the ray exists whereas the center of the sphere remains fixed.

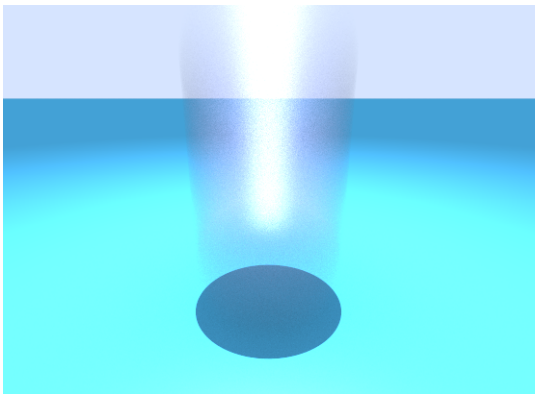

Fig. 1. Image demonstrating Motion Blur.

In Fig 1, The Blurred object in the image is a single sphere that is shown to be moving upward over 10 units of time, but all in 1 frame.

Table 1: Motion Blur Performance

| Number of Spheres | Time (s) - (Albedo) | Time (s) - (Texture) | Speedup |
|---|---|---|---|
| 1 | 14.546 | 15.0259 | 0.9681 |
| 10 | 28.8348 | 29.8377 | 0.9664 |
| 100 | 62.9044 | 63.4172 | 0.9919 |
| 1000 | 304.589 | 305.25 | 0.9978 |
| 10000 | 2788.12 | 1970 | 1.4153 |

The speedup is less than 1 for n <= 1000 spheres indicating that the ray tracer is slower while implementing Motion Blur. however it is still quite close to 1. The speedup changes significantly in this case for a large number of spheres in the sense that it becomes faster for a large number of spheres (n = 10000).

## 2 Volumetric Rendering

Volumetric Rendering involves a set of techniques used for rendering high quality 2-D images from a 3-D volumetric data set. The technique does not spawn a secondary ray, instead, the ray simply "pushes" through the surface of the object and computation is resumed along the ray. This is how volumetric data is gathered. Weather phenomena such as smoke and fog can be rendered this way. We can model them as surfaces and have a probability function tell us whether the surface exists at that point or not. As the ray passes through the volume, it may scatter at any point. A denser volume increases the probability of scattering. Our implementation assumes the shape of the boundary of the surface is convex such as a box or sphere. It does not work with a torus or other non-convex shapes. We also assume that the participating media have constant density.

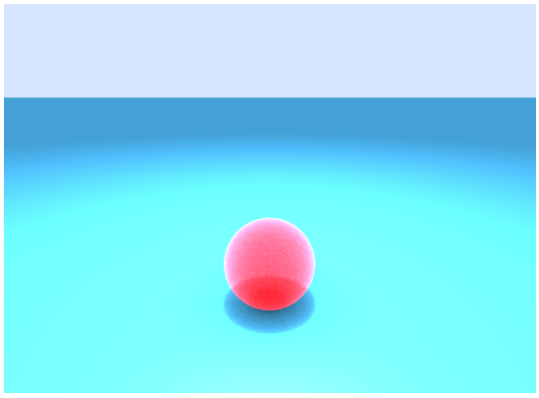

Fig. 2. Image demonstrating Volumetric Rendering.

Fig 2 shows a single sphere that is rendered as if it were made out of a thick fog or smoke using a constant density medium algorithm for checking if rays bounce or pass through

Table 2: Volumetric Rendering Performance

| Number of Spheres | Time (s) - (Albedo) | Time (s) - (Volume) | Speedup |
|---|---|---|---|
| 1 | 14.546 | 14.9646 | 0.9720 |
| 10 | 28.8348 | 24.4697 | 1.1784 |
| 100 | 62.9044 | 49.8276 | 1.2624 |
| 1000 | 304.589 | 235.808 | 1.2917 |
| 10000 | 2788.12 | 776.86 | 3.5890 |

The speed up increases significantly for n >= 10 spheres. From the table, we can extrapolate that our method will be quite fast for a large number of spheres.

## 3 Image-based Texture Mapping

Image-based Texture Mapping is a technique used to generate colors for a surface by producing a view-independent texture map from a set of images taken from different viewpoints. From the hit point, we compute the surface coordinates (u,v). We then use these to index into our procedural solid texture. We can also read in an image and use the 2D (u,v) texture coordinate to index into the image. The *image_texture* class holds an image and contains a function that, given a hit point and (u,v) coordinates, will return the color of the surface at that hit point by converting into pixel (x,y) coordinates and then grabbing the color of the image at that coordinate.

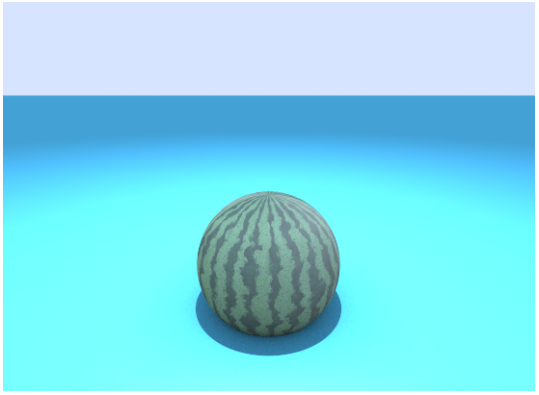

Fig. 3. Image demonstrating Texture Mapping.

Fig 3 shows a single sphere that has had a rectangular image of a watermelon texture UV mapped onto its surface.

Table 3: Image-based Texture Mapping Performance

| Number of Spheres | Time (s) - (Albedo) | Time (s) - (Texture) | Speedup |
|---|---|---|---|
| 1 | 14.546 | 14.7587 | 0.9856 |
| 10 | 28.8348 | 30.6681 | 0.9402 |
| 100 | 62.9044 | 96.5956 | 0.6512 |
| 1000 | 304.589 | 344.733 | 0.8836 |
| 10000 | 2788.12 | 2791.06 | 0.9989 |

The speedup is less than 1 for n = 10000 spheres indicating that the ray tracer is slower while implementing Texture Mapping. However it is still quite close to 1. The speedup does not change significantly in this case for a large number of spheres.

## 4 Conclusion

The team was very interested in implementing a ray tracer in CUDA and/or including Photon Mapping in our implementation. However, due to the nature of the programming language we chose and time constraints, we were not able to get a proper render. If we had more time, we could perhaps try to render a photo with caustics and include our motion blur/texture mapped spheres in the photo as well.

## References


[1] Shirley, Peter. *Ray Tracing in One Weekend Series*, raytracing.github.io/books/RayTracingInOneWeekend.html.

[2] Shirley, Peter. "Ray Tracing in The Next Week." *Ray Tracing in One Weekend Series*, raytracing.github.io/books/RayTracingTheNextWeek.html.