# Ray Tracing with Texture Mapping and Motion Blur

Ian Rudnick (itr2), Fiza Goyal (fgoyal2), Jacob Elling (jelling2)

**Abstract**—Our project expanded our base ray tracer with three different features: image textures, Perlin textures, and motion blur. We added both image-based texture mapping and procedural texture mapping support for all given objects. Additionally, spheres allow motion blur, which renders the object as if it were moving.

## 1 Resources

All of our extended features were implemented using the book *Ray Tracing: The Next Week* [2] by Peter Shirley.

## 2 Implementation

Our starter code was Fiza Goyal's Ray Tracer from MP3, combined with some code snippets from Ian Rudnick and Jacob Elling's MP3. The ray tracer was implemented in C++. The code was run on both Linux and MacOS systems. The MacOS is a 2015 Macbook with a 2.7 GHz Intel Core i5 Processor. The Linux machine has an AMD ryzen 5 3600xt processor running at 4.1 GHz.

## 3 Features

### 3.1 Texture Mapping

A texture is some function that maps colors to the surface coordinates of a rendered object. For this project, we implemented two types of textures: simple Perlin noise-based textures, and image-based textures.

#### 3.1.1 Procedural

Perlin noise is a noise technique commonly used in computer graphics for its pseudo-random appearance that closely models the controlled randomness seen in nature. It can be used to create a wide variety of different textures by layering multiple scaled copies together into mathematical functions. In this project, we used a simple sine function with layered Perlin noise to produce a marbled texture.

Fig. 1 shows a rendering of spheres using our marbled Perlin noise texture.

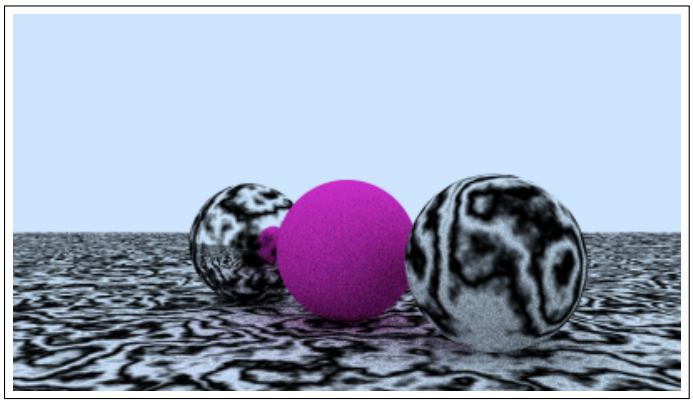

Fig. 1. Marble-like Perlin Noise texture.

The appearance of the texture can be altered by changing the number of layers of noise or the mathematical function used to produce the final color value. Increasing the number of layers of noise makes the lines in the pattern more well-defined. Changing the constants in the sine function can change the frequency of lines in the marble pattern, making them closer together or farther apart. Changing the function to some different combination of mathematical functions can result in completely different texture patterns. We are interested in exploring these possibilities further in the future.

#### 3.1.2 Image-Based

The other type of textures we implemented are image-based textures: the color at each point on the rendered object is directly mapped to some point on an image in memory. Fig. 2 below shows a rendering of a globe using a JPEG image mapped onto a sphere.

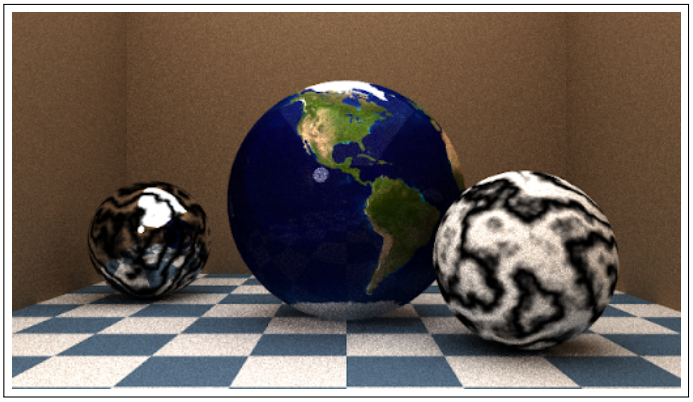

Fig. 2. Texture-mapped globe.

There is a significant performance cost to using our implementation of Perlin textures, because the noise is computed at runtime, adding many calculations to each ray sample that hits a textured object. The performance could be improved by pre-computing the texture and storing it as an image, and then using image texture mapping to render the final scene, because image texture mapping has a simpler per-ray calculation. Table 1 shows the rendering times for the above image using both Perlin textures and image textures, and using only image textures.

Table 1. Rendering times for spheres with texture mapping.

| Image Dimensions | Samples per pixel | Render time for Perlin and Image textures | Render time for Image textures only |
|---|---|---|---|
| 640x360 | 256 | 156s | 132s |

### 3.2 Motion Blur

Motion blur was adapted from Shirley's book and applies to spheres only. The spheres take in a start time and an end time and simulate the motion of a sphere and the associated blur that one would expect from an actual camera due to shutter speed. They also take an original center and a final center to indicate the direction of motion. The result is the average of random samples over the period of the shutter time. Fig. 3

and Fig. 4 show a simple rendering of some spheres, with and without motion blur.

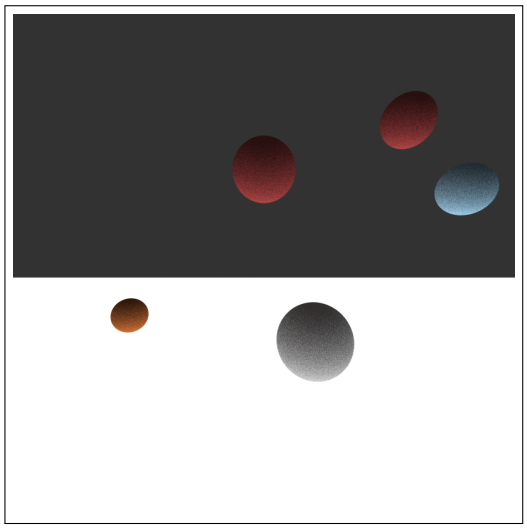

Fig. 3. Spheres without motion blur.

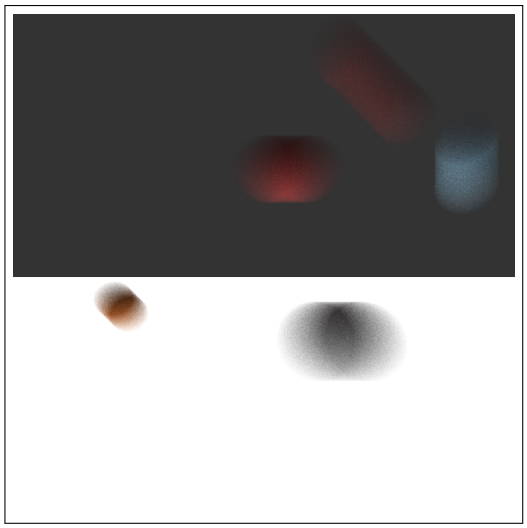

Fig. 4. Spheres with motion blur.

Table 2. Rendering times for spheres with and without motion blur.

| Image Resolution | Samples per pixel | Render Time Without Blur | Render Time With Blur |
|---|---|---|---|
| 800x800 | 64 | 103.862s | 104.217s |
| 800x800 | 100 | 195.597s | 191.082s |
| 800x800 | 144 | 308.289s | 294.981s |

Our implementation of motion blur adds no significant amount to render times of regular spheres and in some cases is more performant.

## 4 Conclusion

Early on, we planned on implementing fluid animations and did research on a few different methods of doing so. We even began implementing a particle-based fluid animation based off a paper by Caio José dos Santos Brito [1]. However, our renderer architecture so far made it difficult to implement realistic-looking fluids without significantly changing our rendering engine.

The texture features we implemented can be extended in many interesting directions. One feature we would like to try is using texture mapping for 3D meshes. The .obj file format has an option to store texture coordinates for each vertex in addition to their positions and normals. This makes it possible to design a mesh and a texture together in a program like blender, export them as a 3D object file (.obj) and an image file (.png), and load them into another renderer while keeping the correct texture mapping at each triangle.

Another feature we want to work on is implementing a variety of different Perlin noise textures. We could also produce a more colorful marble pattern by layering Perlin noise textures with different colors– maybe one with brown lines and one with gray, to make a more realistic multicolored marble surface. The Perlin noise doesn't have to be used for only textures either – we could use it to create bump maps or normal maps to create objects with both natural textures and contours.

A link to our github repo is here: **https://github.com/fgoyal/CS419-Final**

## References


[1] C. J. dos Santos Brito. *A Ray Traced Rendering Solution for Particle-Based Fluid Simulation*. PhD thesis, Federal University of Pernambuco, Brazil, 2015.

[2] P. Shirley. *Ray Tracing in One Weekend*. 3.2.3 ed., 2020.