Final project Write up

Project name: Ray tracer

HU Hengyuan

For me, the most attractive thing in Computer Graphics is to synthesize amazing pictures. There are two main ways to synthesize pictures using computer. One is called rasterization, which is employed by OpenGL, the other one is ray tracing, which is done by a particular program called ray tracer. The main stream in the industry is rasterization because the famous saying “It is fast, and fast is good”. However, I am more interested in the quality of the picture. Therefore, after writing some assignments in OpenGL, I decided to do the same thing in ray tracer to compare the outputs to gain a more clear understanding of both methods. So I wrote a ray tracer from scratch and kept adding new features to it.

To write a ray tracer, the first thing is to fully understand the algorithm behind it. In nature, we can see an object because light hits it and the reflected light goes into our eyes. So the most straightforward way to simulate this process is to keep emitting light beams from the light source and track each light to see whether it can hit some pixel on our screen. If the light beam hits a pixel, set the color of that pixel as the color of the light. If another light beam hits the same pixel, just mix the new color with the old one. Then we are done. Nevertheless, this method has two main problems. One is that only a few light beams can actually goes into our eyes in nature, so we are wasting a lot of computation resource to tracking useless light beams. The other one is that we will never know how many beams are enough to color all the pixels on the screen because we cannot predict light path in advance.

To solve these two problems, people come up with a method called eye tracing. In eye tracing algorithm, each pixel emits a light beam to the scene. If the light hit some object, we note down the hit point and emit a new light beam from

the hit point to the light source to check whether it is in the shadow of some other objects. Then we use these two light beams and other information to calculate the color of the pixel where the light comes from.

To turn this algorithm into program, I employed an object-oriented way to implement it. My program consists of four parts, an abstract base class “Object” providing unified public API for all objects inheriting it, a “Scene” class consisting of a list of Objects and a list of light source, an “Image” class storing the data and writing it out to png file, and a “Ray-tracer” class implementing the rendering methods.

In the process of coding the basic part of the ray tracer, I used lots of knowledge learned from the course assignments. For example, when I run across a bug like Figure, I recalled the similar problem I encountered while doing the rasterization assignment. I then realized that this weird output was caused by the precision error of floating point numbers. So I added an epsilon term in every comparison and then the problem was solved! I also used the phong shading model learn in assignment5 to make the output more realistic (Figure 2, 3). To solve the “black shadow” problem (Figure 3), I add ambient lighting to the environment to make shadow more natural (Figure4). The idea is borrowed from OpenGL.

After finishing the basic part, I decided to add reflection and refraction features to the ray tracer. To accomplish it, we need to revise the algorithm. We shoot a primary ray from the eye and the closest intersection (if any) with objects in the scene. If the ray hits an object which is not a diffuse or opaque object, we must do extra computational work. To compute the resulting color at that point, we need to compute the reflection color and the refraction color and mix them together with the original color of that point according some laws. We need to compute the color of one pixel in four steps. Compute the original color (the basic part of ray tracer only does this step), compute the reflection color, compute the refraction color, and then mix them together [1]. When implementing refraction part, I recalled what we did in the “pbrt” assignment. So

I used the equations derived in that assignment to calculate refraction ray and they were pretty handy. The new function to compute the final color of a pixel is implemented in a recursive way. The pseudo code is like this: [2]

```
Color trace_ray( Ray original_ray )
{
    Color point_color, reflect_color, refract_color
    Object obj

    obj = get_first_intersection( original_ray )
    point_color = get_point_color( obj )

    if ( object is reflective )
       reflect_color
    = trace_ray( get_reflected_ray( original_ray, obj ) )
    if ( object is refractive )
        refract_color
    = trace_ray( get_refracted_ray( original_ray, obj ) )
    return ( combine_colors( point_color, reflect_color,
    refract_color ))
}
```

After adding reflection and refraction features, my ray tracer can produce pictures like Figure5 (only reflection) and Figure6 (reflection and refraction).

The above is how far I have reached till now. And final pictures (Figure5 and Figure6) is still not perfect because I haven't figure out how to properly mix up three colors. However, I really learned a lot during the design and implementation process. I ran across many weird situations that the outputs were entirely different to what I had imagined and gained a deeper understanding of the saying "you have to be creative when debugging graphics programs". I reviewed and took advantage of a lot of knowledge learned in this course and applied the knowledge to improve my ray tracer. Finally, I do thoroughly understand ray tracer and make some pretty awesome pictures. Apart from all of that, I gained a lot of fun!

Reference:

[1]: Adding Reflection and Refraction,

http://scratchapixel.com/lessons/3d-basic-lessons/lesson-1-writing-a-simple-raytracer/adding-reflection-and-refraction/

[2]: Ray Tracing: Graphics for the Masses,

https://www.cs.unc.edu/~rademach/xroads-RT/RTarticle.html

Remark:

In my code, the method "`int Image::writePNG(const char *path)`" is adapted from the website: http://www.lemoda.net/c/write-png/

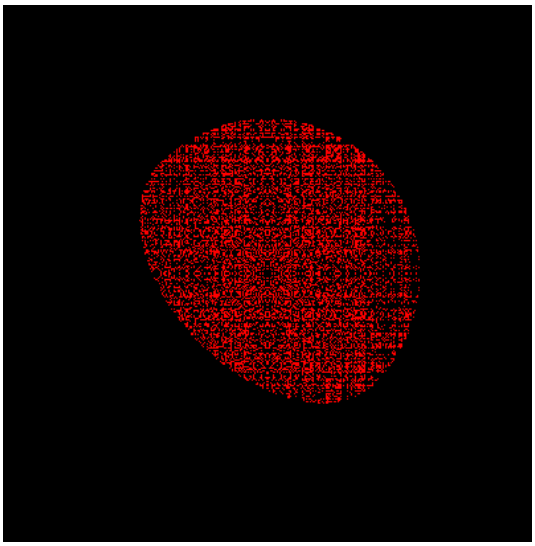

Figure 1

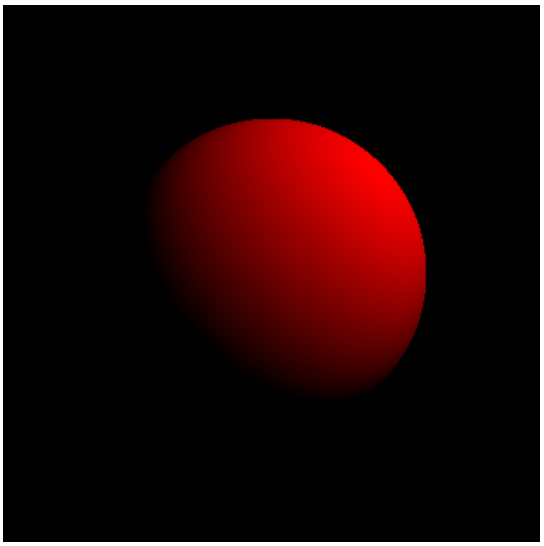

Figure 2

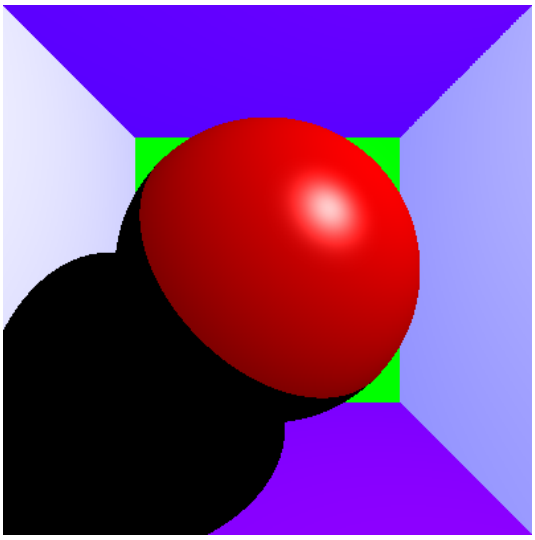

Figure 3

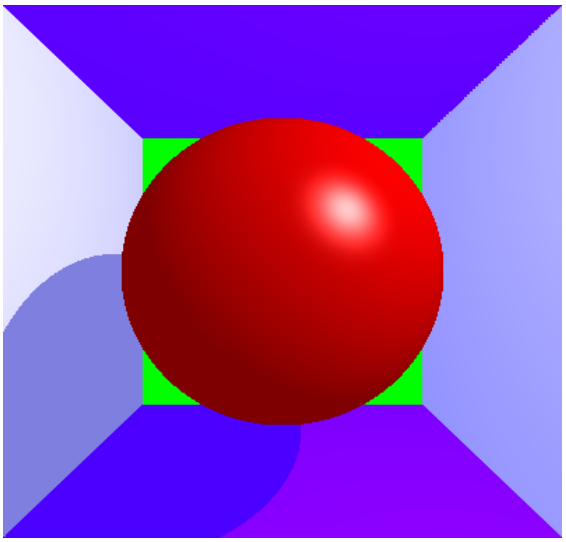

Figure 4

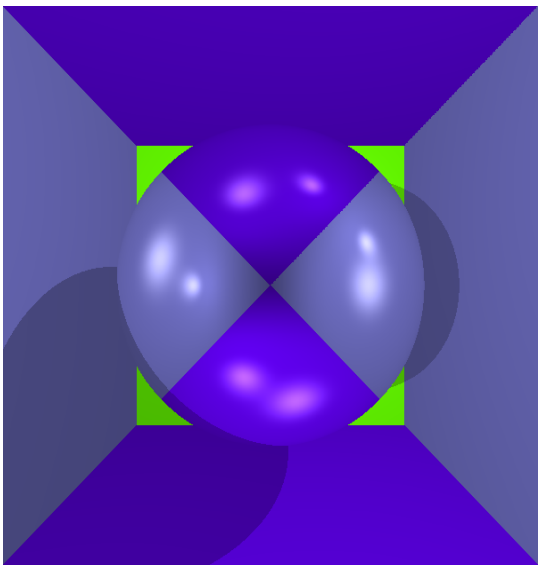

Figure 5

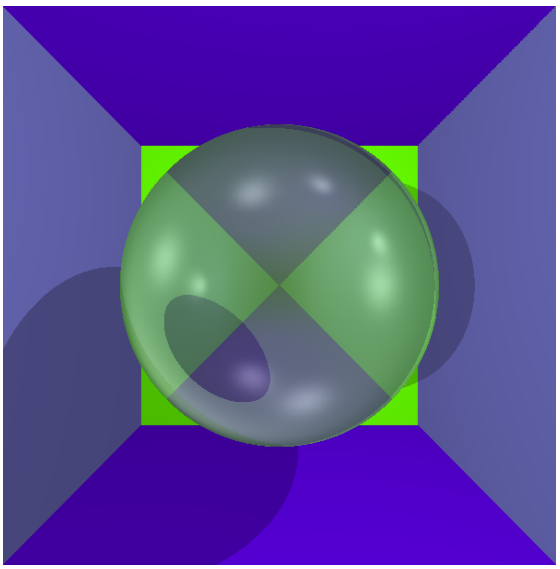

Figure 6