Programming Assignment #3

Raytracer

CMPSC 458 Computer Graphics, Fall 2021

Release Date: Thursday, October 21, 2021
1st Intermediate Submission: Friday, November 5, 2021 (**10 points**)
2st Intermediate Submission: Friday, November 12, 2021 (**30 points**)
Final Submission: Friday, November 19, 2021 (**60 points**)

1 Introduction

This assignment will require you to build a raytracer; a program that takes a high-level description of a scene (where the objects, lights, and camera are) and produces the image seen through that camera. You will do this without using glm to render the image: you will use mathematical operations to calculate the RGB value that should be at each pixel in the desired image. According to wikipedia: In computer graphics, ray tracing is a technique for generating an image by tracing the path of light through pixels in an image plane and simulating the effects of its encounters with virtual objects. The technique is capable of producing a very high degree of visual realism, usually higher than that of typical scanline rendering methods, but at a greater computational cost. This makes ray tracing best suited for applications where the image can be rendered slowly ahead of time, such as in still images and film and television special effects, and more poorly suited for real-time applications like computer games where speed is critical. Ray tracing is capable of simulating a wide variety of optical effects, such as reflection and refraction, scattering, and chromatic aberration.

Start this project early, it might seem easy at first but many people have struggled with this before.

2 Grading Details

Your project will be graded by:

1. Intermediate submission 1 (10 points)
    - Ray intersection (10):
        - Triangle (5)
        - Sphere (5)

2. Intermediate submission 2 (30 points)
    - Texture mapping (10):
        - Triangle (5)
        - Sphere (5)
    - Reflection (10)
    - Refraction (10)
3. Final submission (60 points)
    - Diffuse and specular shading (10)
    - Shadows (10)
    - Recursive reflection/refraction (10)
    - Animation (10)
    - Two (or more) original scenes with some new materials or with some creativity in the scenes (10)
    - Video of Animation (5)
    - Readme (5)
    - **ExtraCredit** (Up to 20)

3 Requirements

Your raytracer must read a description of a file of the .ray format (see your starter code). An example .ray file is shown in Figure 1. This file format includes descriptions of camera, background color, ambient light, diffuse/specular lights, and spheres & triangles with material properties and texture images. Your raytracer must handle all of these accurately to receive full credit. With your starter code comes a set of sample scenes useful to test that your raytracer can handle all requirements. You will need to make two new .ray files of your own, and you will need to submit renderings and animations of your new scenes (as well as renderings of the sample scenes) along with your code and readme.

For help, see the lecture notes and project slides on CANVAS. Chapter 4 (Ray Tracing) and Chapter 13 (More Ray Tracing) in the book are also very helpful.

For the animation part, you should render an image, then move something (whatever you want to animate) a very small amount, and render again. Repeat until you have all the frames of the video (usually about 30 frames per second). What you move is up to you. You could even just move the camera. In essence, you will create the frames individually and then put them together into a movie after everything is rendered.

FFmpeg is a good tool which is free online for putting together the frames. You can also use Matlab if you are familiar with that.

```
# author: Kyle

Background {
        # Color of the background
        color 0.15 0.15 0.25 # blue-gray
        # low intensity white light
        ambientLight 1 1 1
}

Camera {
        eye 0 0 0
        lookAt 10 0 0
        up 0 0 1
        fovy 45
}

Lights {
        Light {
                position 10 0 5
                color 1 1 1
        }
}

Materials {
        Material {
                textureFilename NULL
                diffuseColor 0.5 0 0
                specularColor 0.3 0.3 0.3
                reflectiveColor 0.3 0.3 0.3
                shininess 50
                transparentColor 0 0 0
                indexOfRefraction 0.6
        }
}

Group {
        Sphere {
                materialIndex 0
                center 10 0 0
                radius 2
        }
}
```

Figure 1: A Sample Ray File.

4 Intermediate Submission 1 (10 Points)

For intermediate submission 1, you need to have the triangle and sphere intersections, as well as the lighting and basic coloring completed. Your submission should contain ray-tracing result based on sample scenes of 'singleTriangleNoLight.ray' and 'singleSphereLighted.ray'. The file structure of your intermediate submission 1 should be like:

```
Project3_Intermediate_1_YourName
├── Project_3
│   ├── Headers
│   ├── Media
│   ├── Shaders
│   ├── Sources
│   └── Project_3 executable
├── RenderingScenes
│   ├── your_singleTriangleNoLight.png
│   └── your_singleSphereLighted.png
└── Readme.md
```

5 Intermediate Submission 2 (30 Points)

For intermediate submission 2, you need to have the reflection and refraction done, as well as the texture mapping completed. Your submission should contain ray-tracing result based on sample scenes of 'reflectionTest.ray', 'refractionTest.ray', 'reflection&refraction.ray', 'textureTriTest.ray', 'textureSphereTest.ray' and 'reflectiveSpheres&Tris.ray'. The file structure of your intermediate submission 2 should be like:

```
Project3_Intermediate_2_YourName
├── Project_3
│   ├── Headers
│   ├── Media
│   ├── Shaders
│   ├── Sources
│   └── Project_3 executable
├── RenderingScenes
│   ├── your_reflectionTest.png
│   ├── your_refractionTest.png
│   ├── your_reflection&refraction.png
│   ├── your_textureTriTest.png
│   ├── your_textureSphereTest.png
│   └── your_reflectiveSpheres&Tris.png
└── Readme.md
```

6 Final Submission (60 Points)

For final submission, you need to have everything done. You **MUST** have a rendering for each of the example ray files in your final submission, and make at least two new .ray files of your own. Finally you will need to have animations of your new scenes. As a minimum requirement for animation, you could let the camera moving a bit by each frame, to give a surround view of your

scene. The file structure of your final submission should be like:

```
Project3_Intermediate_2_YourName
├── Project_3
│   ├── Headers
│   ├── Media
│   ├── Shaders
│   ├── Sources
│   └── Project_3 executable
├── RenderingScenes
│   └── list of rendering results
├── your_animation.avi/mp4
└── Readme.md
```

7 Extra Credit (Up to 20 Points-)

As usual, be creative, describe everything you do in the readme file, and make things look nice to get more points. Some ideas:
- Good antialiasing.
- Animation with motion blur.
- Depth of field.
- Blinn blobs.
- Multiple object groups with bounding spheres.
- BSP Tree for speed up.
- Up to two additional object types (e.g. Procedural Object like a fractal or a Mesh).
- Procedural texture mapping.
- Loading an obj file and showing it in your work. This would require a new type of object (You will want to also use BSP and make sure your code is efficient before attempting this).

8 Additional Notes

The mathematic library is GLM but you cannot use the functions like 'reflect' or 'refract' within the library. You must code them yourself and any other mathematics (except basic functions like cross product, normalization, or anything else in the starter code).

The rows are parallelized using OpenMP so the code will run in parallel which should give a significant speed increase. You can turn this off by commenting it out if, like an animal, you debug using print statements.

9 Working with the Clusters

The access to the CyberLamp clusters is **currently to be decided**. We will let you how to access to the cluster later. Kindly pay attention to our course announcements on Canvas. You may use

these resources run multiple ray files at the same time (which will be useful for animation and for debugging).

10 Common Issues

- The code take a very long time to run.

  Use a tool to measure how long each function takes to run to find what is taking the longest and then figure out how to make it faster. You should be good enough coders to code efficiently by now.

- The colors/textures have lots of dots in the reflection.

  There is some machine imprecision when dealing with intersections. Conceptually, you are finding the intersection between the ray and the wall but rounding slightly inside of it and then hitting the wall again when you reflect. You need to define a minimum intersection distance (normally very small such as 1e-5) to deal with this imprecision (like when you compare floats) which is typically called epsilon (see Figure 2).

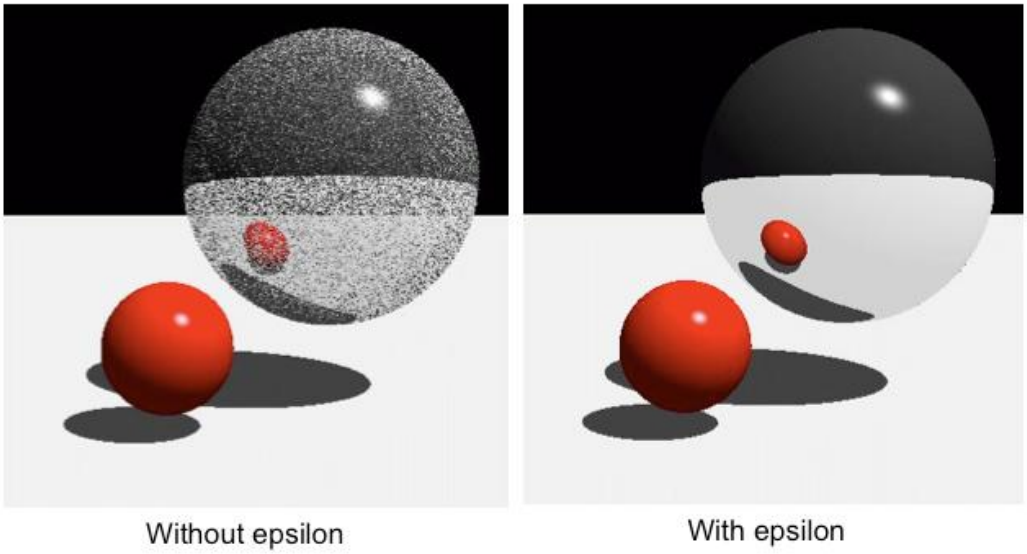

Figure 2: Example image showing with and without a minimum distance (epsilon) for intersections.