CSC418 A3 Report

Name: Chengchen Yang, Student ID: 1000285880
Name: Haoyuan Li, Student ID: 1000275766

All pictures for Part A are in folder raytracer/Part A Images
All pictures for Part B are in folder raytracer/Part B Images

For part A, We implemented all the feature required:

1. Ray casting
2. Ray-sphere intersection
3. Ray-square intersection
4. Phong illumination (ambient, diffuse, and specular components)

For Part B, We implemented Advanced ray tracing, hard shadow and 5 other features:

a. Advanced ray tracing: I implemented this feature in shadeRay function at raytracer.cpp. If it hit an object, then I create a reflection ray and call the shadeRay again to update the color. I added a new parameter for the function in order to keep track of the current depth and avoid unstoppable recursive ray tracing.

b. Hard shadow: I implement this feature in computeShading function in raytracer.cpp. First cast the ray from intersection point, if it have another intersection with an object then it is in shadow.

1. Anti Aliasing: I implement this feature in render function in raytracer.cpp. Computing the average colour for the 16 subsections of the whole pixel rather than the centre point. To turn on Anti aliasing, change the value of 'antiAliasing' to ture.

2. Depth of field: I implement this feature in render function in raytracer.cpp. I find the new origin and cast direction from the focal point and aperture. To turn on depth of field, first turn on anti aliasing, then set the 'FOCALRANGE' . A good focal range for view 1 is 4.5.

3. Texture Mapping: I implemented this feature in scene_object.cpp with multiple functions. I added extra cube ray tracing function as well in scene_object.cpp. It calculate the t value based on the intersection with 6 different planes and choose the smallest t value and set the corresponding normal vector and cast the ray. First, I read the bmp image file and store the width height and RGB color in the variables. I added updatePlaneTextureMapping / updateSphereTextureMapping / updateCubeTextureMapping in order to provide mapping for three different objects. Each of them calculate the x and y value and map the color to the pixels.

Then, in the shade function of light_source.cpp, it checks if the hasTexture flag is set and map the texture afterwards.

4. Refraction: I implemented this feature in shadeRay function at raytracer.cpp. If it hit an object, I check it enters or leaves an object and calculate the refraction coefficient based on the results. Then, I compute the refraction vector with the formula and then recursively call shadeRay to get refraction color and use fresnel equations to calculate the ratio of refraction light.

5. Compound object - Cylinder: I Implemented it in UnitCylinder::intersect at scene_object.cpp.To calculate the cylinder first check if the ray intersected with the top or bottom bases, and checks if the intersection point is within the radius. If intersects, then it check if intersects with top or bottom based on the t value. And calculate the corresponding normal vector based on the intersection location. If it intersects with the cylinder body, it decides based on the top and bottom coordinates and cast the ray.

Role of each member:
Haoyuan Li: Ray casting, Ray-sphere intersection, Phong illumination, Hard shadow, Anti Aliasing, Depth of field.

Chengchen Yang: Ray-square intersection, Advanced ray tracing, reflection, Texture Mapping, Refraction, Compound object - Cylinder.

External resources:
Cylinder: http://woo4.me/wootracer/cylinder-intersection/
Cube Ray Tracing:
https://www.scratchapixel.com/lessons/3d-basic-rendering/minimal-ray-tracer-rendering-simple-shapes/ray-box-intersection
Texture Mapping for Cube and Sphere:
http://bentonian.com/teaching/AdvGraph1314/3.%20Ray%20tracing%20-%20color%20and%20texture.pdf