CS488 Final Project Report

Kabutack and Bathhouse

Zijian Feng

20819676

z63feng

1 Purpose

The purpose of this project is to implement an efficient ray tracer with advanced reflection and refraction, and create the scene *B-Robo Kabutack*.

Topics

- Add reflections and refractions for different materials.
- Make an efficient render using intel multi-threading library.
- Have a great understanding in texture mapping.
- Anti-aliasing and depth blur are properly implemented.
- Have a great understanding in Volume scattering and render smoke effects.

2 Statement

My extra feature in A4 is mirror reflection.

The rendered scene is describing Kabutack resting in a Japanese bathhouse. The global illumination will be shown in this bathhouse and also decorations on the ground. ~~The photon mapping will give the water real reflections. A Roof window will be given as the planar light source.~~

Compared with previous assignments, this project will achieve better light effects since different types of refractions and reflections, and better efficiency as Intel Thread Building Blocks technique will be used and also Monte Carlo method when we simulate the glossy reflections.

Volume scattering might be the hardest objective. From this project, I will have a great understanding of Monte Carlo application in ray tracing, anti-aliasing technique, and texture mapping in geometry.

3 General Implementation

3.1 Lua Command Add-ons and Object Loading

Two Lua commands are changed. ‘gr.material’ and ‘gr.texture’.
Reflection index and refraction index goes after their strings.
If ‘glossy’ is given, fuzziness is defined and with reflection, refraction after.

```
gr.material({0.65, 0.05, 0.05}, {0.1, 0.1, 0.1}, 1, "reflect", 0.4)
gr.material({0.65, 0.05, 0.05}, {0.1, 0.1, 0.1}, 1, "refract", 0.6)
gr.material({0.3, 0.6, 0.6}, {0.5, 0.7, 0.5}, 35, "glossy", 0.1, "reflect", 0, "refract", 0.6)
gr.texture(japan_wood_material, "texture/JapanWood.jpg")
```

gr.texture will bind the image texture to the material. Note that Class Material has a default nullptr of Texture.

For an Object file of format .obj, I implemented full support for loading these files.

A5 > Assets > plane.obj

```
o plane

v -1 0 -1
v 1 0 -1
v 1 0 1
v -1 0 1

vt 0 0
vt 0 1
vt 1 0
vt 1 1

vn 0 1 0

f 1/2/1 3/3/1 2/4/1
f 1/2/1 4/1/1 3/3/1
```

Mesh.cpp handles the obj loading.
For a face ‘f v1/vt1/vn1’, v1, vt1, vn1 donate the index of vertices, texture vertices, and normals.

3.2 Material Class Interface

In this project, three types of materials are given, a Phong Model(glass), Glossy, Lambertian. They all inherited from ‘Material’.

```
virtual bool scatter(const Ray& in, const HitRecord &rec, vec3& attenuation, Ray& scattered) = 0;
virtual bool refract(const Ray &in, const HitRecord &rec, Ray &refracted, bool flip) = 0;
```

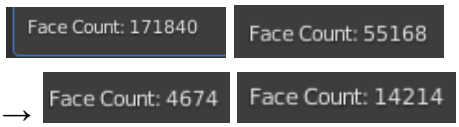
‘scatter’ will calculate the hit and provide the reflected ray.
‘refract’ will calculate the hit and provide the refracted ray.

4 Technical Outline

4.1 Complex Modeling on scene objects

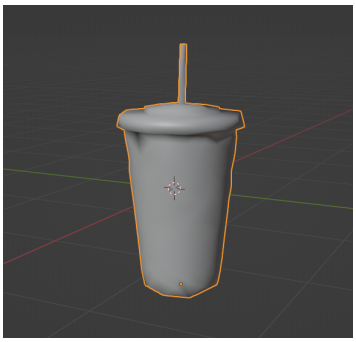
For a modeling of complex objects, primitives can be too time consuming, and also too tricky to connect parts into entirety. To handle this, we may use third-party modeling tools such as blender to generate an obj file for each individual.

Although I made a Kabutack model in A3, it is completely unusable. The reason is that the model has too many faces such that the ray tracer may take several hours rendering. For example, a torso model has 171840 faces and a helmet has 55168 faces. By using blender, I decimated most of the models from A3.



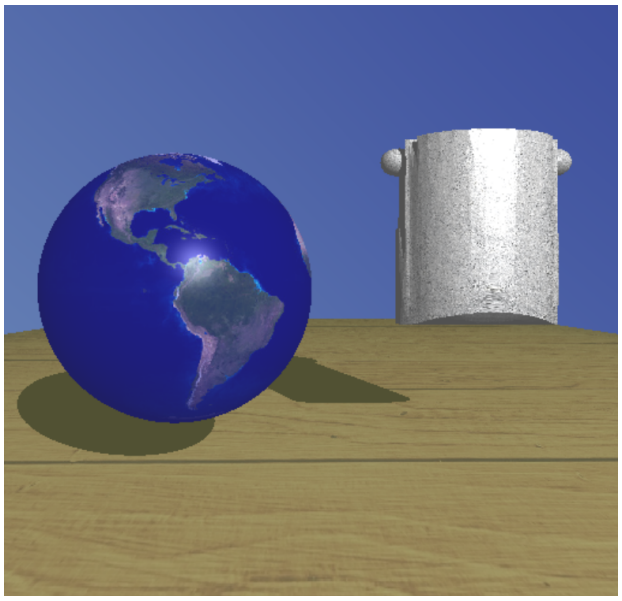
(B-Robo Kabutack)

Other objects are pretty built on my own as well. They are also decimated if too many faces are created during curving.



4.2 Texture mapping on environmental objects

The importance of texture in graphics is undisputed. C++ std_image library will be used for loading. Then, two types of texture mapping will be implemented. One for the sphere, the other for the mesh model.



For Sphere, every coordinate has a corresponding (direction angle θ, zenith angle ϕ)

$u=\phi/(2\pi)$

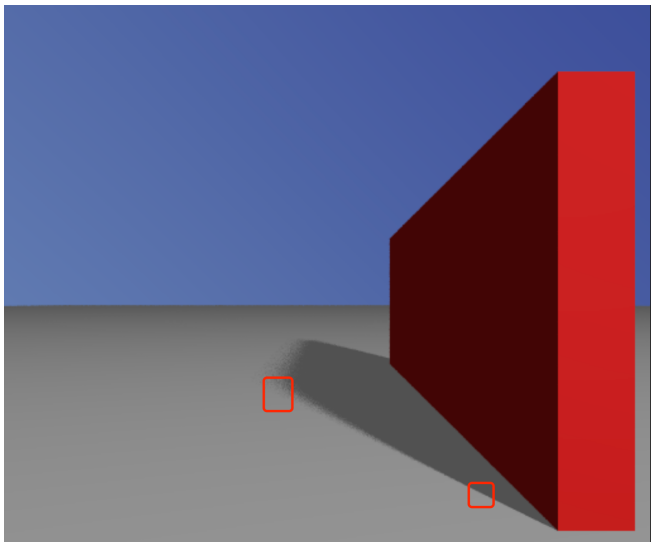
$v=\theta/\pi$

By using it, we get the texture coordinate (u, v).

For mesh, we use Blender to generate the texture coordinates.

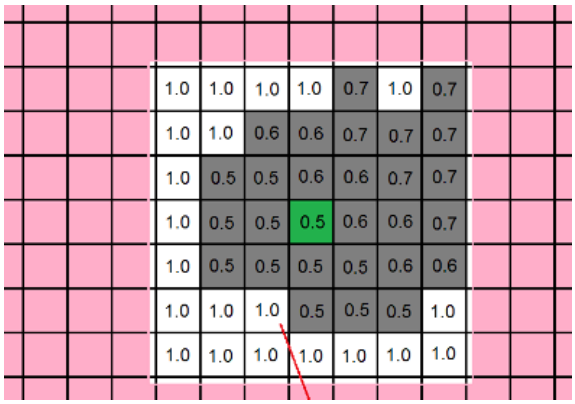
4.3 Soft shadow

The idea for soft shadow is that we calculate weight summation for each pixel. For each ray, the weight can be the ratio of the distance to the occluding object and the distance to the light source.



A shadow coefficient is obtained by combining two factors:

First, the ratio of occlusion distance divided by distance to light origin;



A sampling area is given for each pixel. The ratio of shadow in this area is the second factor.

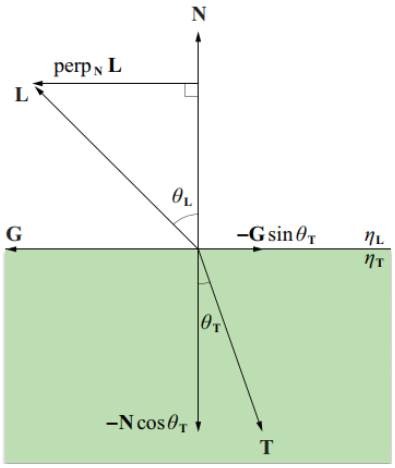
4.4 Refraction

We will involve emitting secondary refracted rays for objects with a transparency coefficient and an index of refraction as clarified in A4 extra feature. The ray refraction is described by Snell's law:

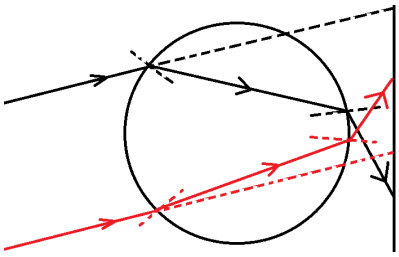
$$\eta \cdot \sin\theta = \eta' \cdot \sin\theta'$$

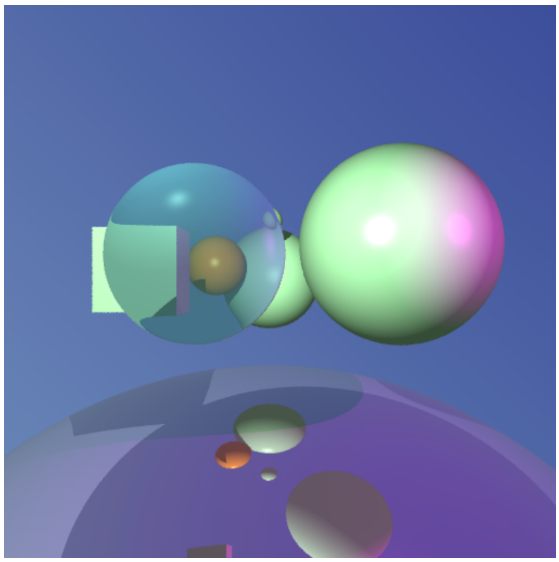
(from "Ray Tracing in One Weekend", Chapter 10.2)

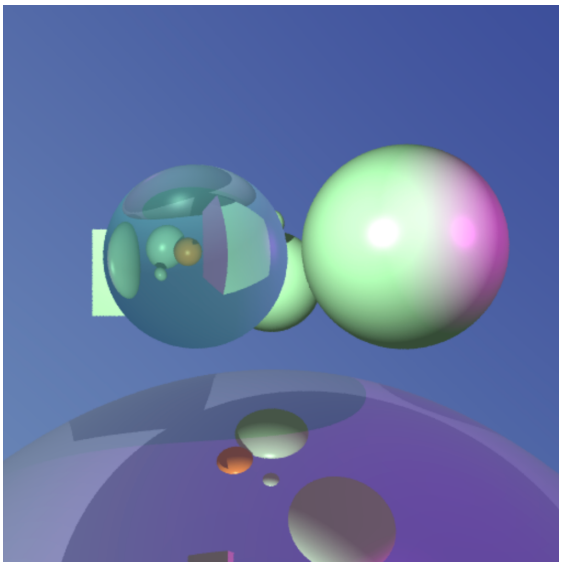
To combine refraction and reflection of glass, a coefficient is also defined but only as constant for the purpose of testing. Note that real glass has reflectivity that varies with angle, Fresnel function is also introduced. But I will use Schlick's Approximation to achieve this in a cheaper way. (from "Ray Tracing in One Weekend", Chapter 10.4) The Schlick function is in PhongModel.cpp.



From the above equation and the figure on the left, the refracted ray can be calculated.
When the index of refraction is increasing, the image is reversed. Note that this is correct due to the ray refraction.







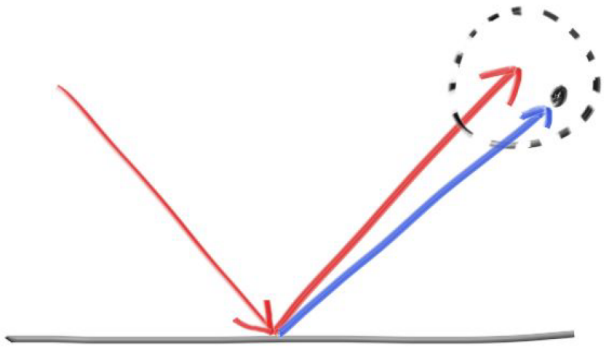
4.5 Glossy refraction/reflection

For a hit on a glossy lambertian, theoretically, there should be numbers of rays uniformly reflected from the hit point. For our glossy model, we will have a cosine distribution replacing the reflection probability. Similar to lambertian, it is not realistic to have many rays reflected and traced. To solve this efficiency, Monte Carlo method is introduced and will simulate the result.

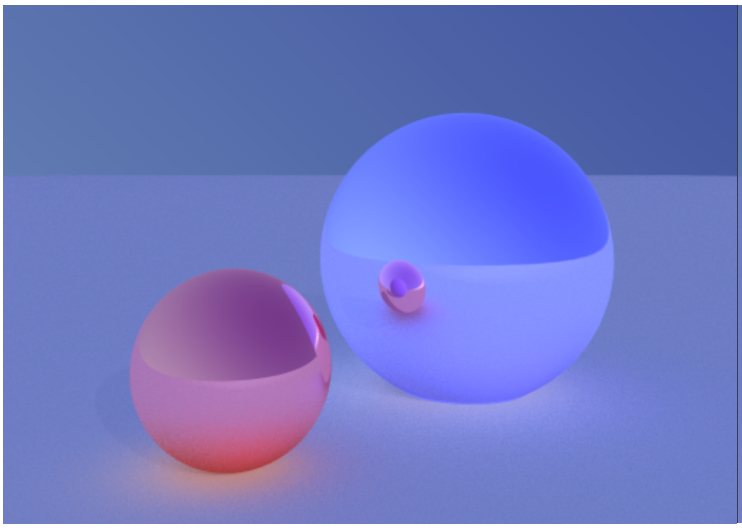
The reflection of a lambertian model referred from "Ray Tracing in One Weekend", Chapter 8. A simple importance sampling, Random Hemisphere Sampling is introduced from "Ray Tracing: the rest of your life", Chapter 6.2.

The Glossy reflection is implemented based on the mirror reflection.

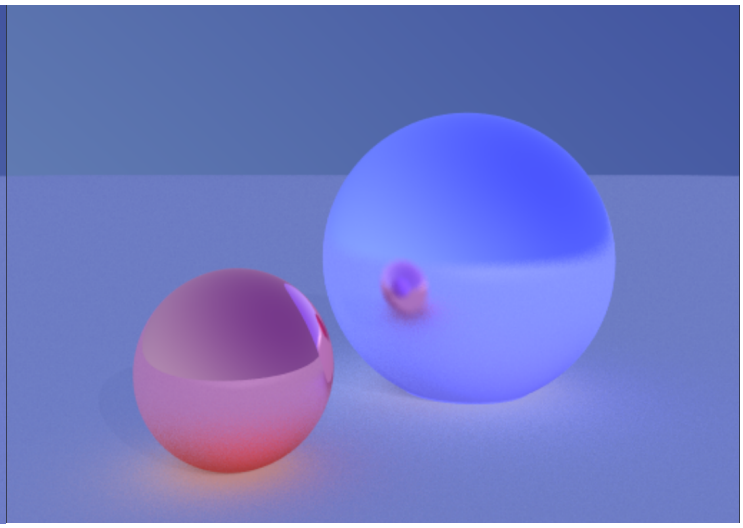
First, we have a config parameter **m_fuzz,** which controls the glossiness of the reflection.



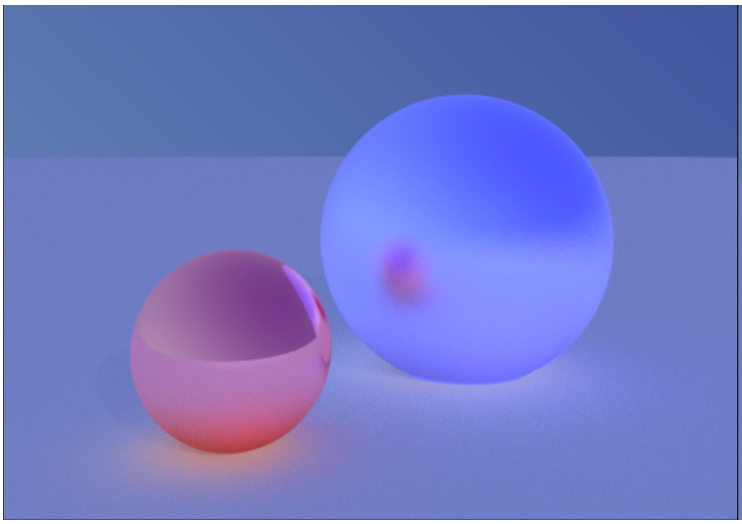
At the end of the mirror reflection vector, we give a sphere with a radius of **m_fuzz.** Then we randomly choose a point inside the sphere as the end point of the reflection ray. The glossy effect is done.



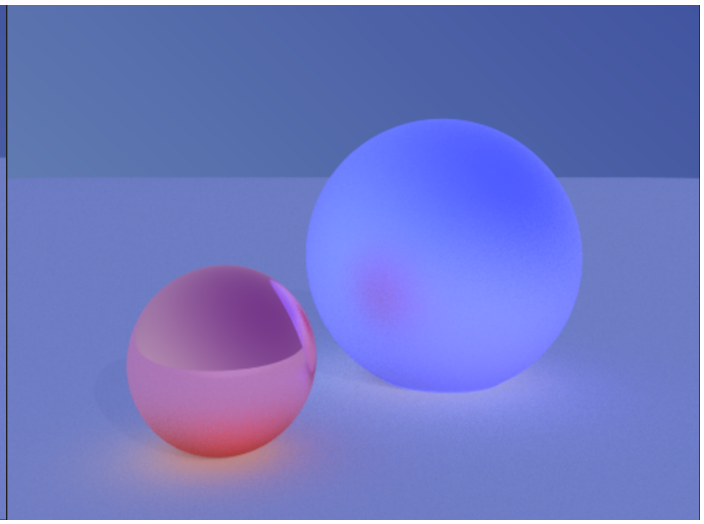
With **m_fuzz = 0.01**



With **m_fuzz = 0.1**

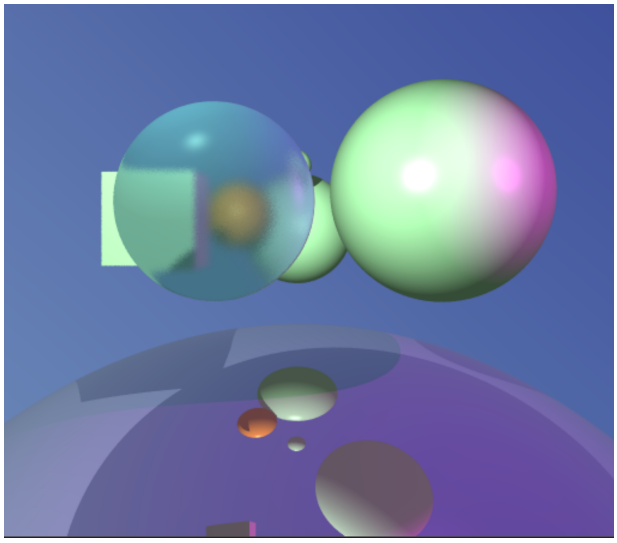


With **m_fuzz = 0.3**

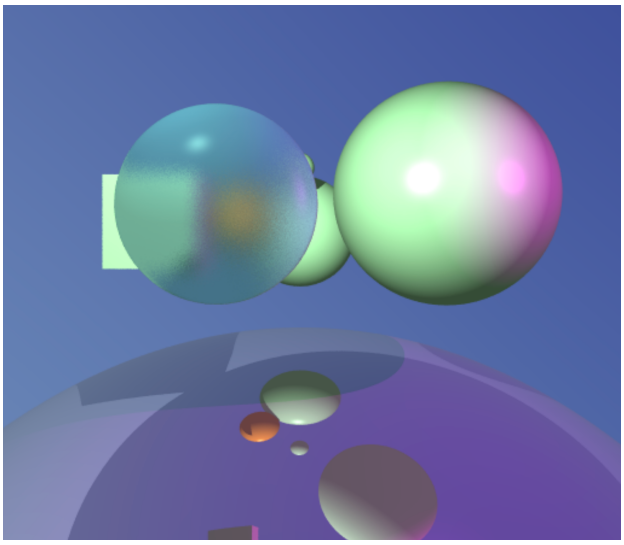


With **m_fuzz = 0.8**

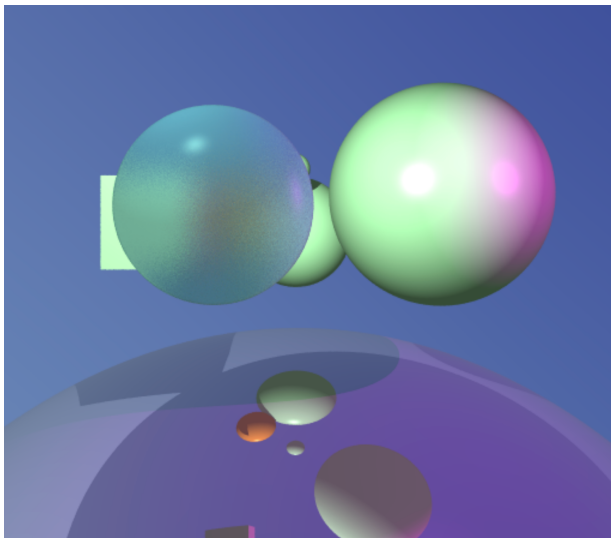
Note that in refraction, a ray will refract twice to display the actual scene. The m_fuzz will be more sensitive compared to the reflections.



m_fuzz = 0.01



m_fuzz = 0.05

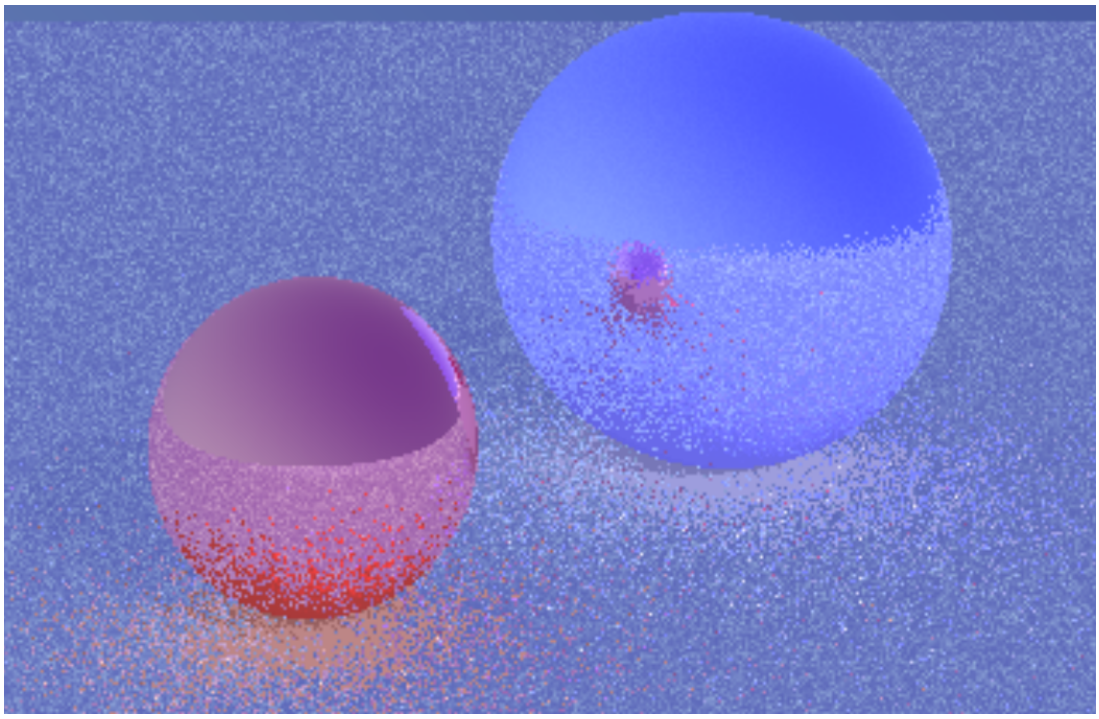


m_fuzz = 0.1

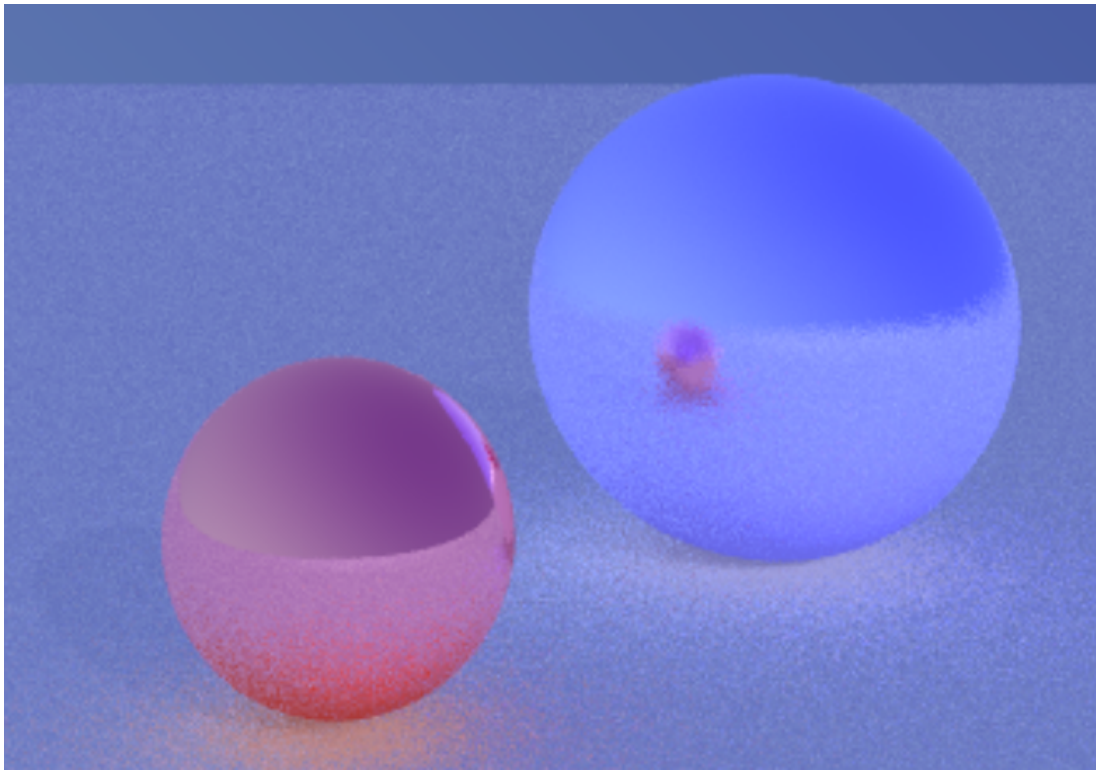
4.6 Super-sampling for anti-aliasing

The super-sampling is generating pixels with an average result from multiple rays emitted. To illustrate the effect of reducing noisy points, we use Lambertian Material to mock the matte surface such as paper because, in the Lambertian Model, the ray is generated by a diffuse reflection. It has the uniform probability in every direction. If sampling is not enough, the noise will be more apparent than other materials.

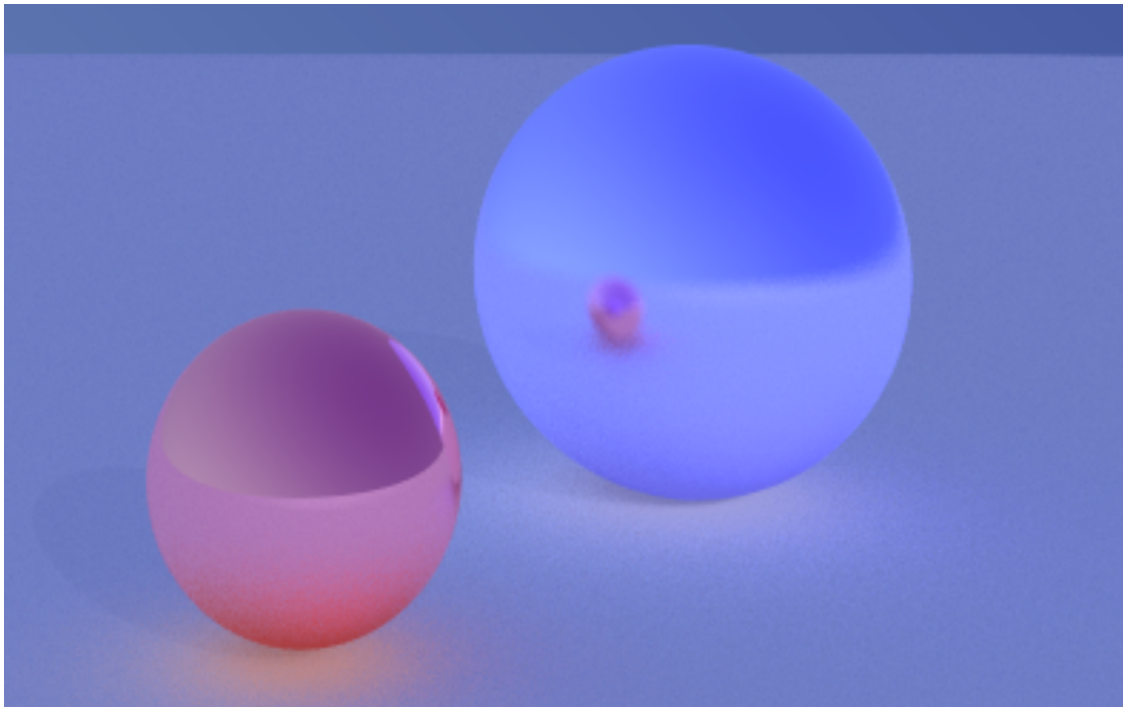
The implementation is pretty similar to previous objective 4.5 glossy. We randomly emit rays for each pixel with the end point in a unit circle.



No super-sampling anti-aliasing



10x super-sampling anti-aliasing



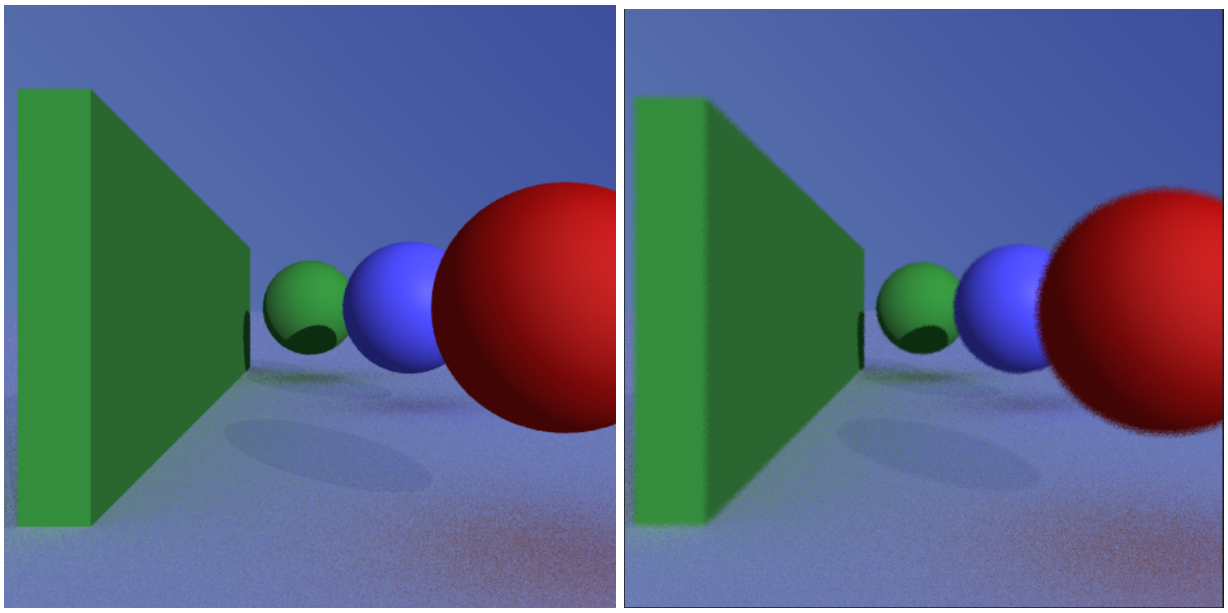
100x super-sampling anti-aliasing

4.7 Camera depth of field

To simulate aperture from an actual camera, we will define a defocus disk to replace the eye. We could understand that our original image has a len disk of zero radius, which means every ray is perfected emitted from a single eye pos. (no blur at all).

Also, to have a real camera effect, we introduce focus distance, the distance between the defocus disk and render image, such that the blur effect can be adjusted.

focusDist = length(target - cameraPos)



4.8 Photon Mapping

I want to achieve global illumination by using basic photon mapping.
The idea is coming from the lab of Zack Waters.
There will be two passes.

Pass 1: build a photon map.

The rule is to emit rays from light and let them reflect with power decay for a depth. Every hit will be recorded and final power will be stored in the map.

Pass 2: Rendering.

For rendering, we can use density estimation to estimate the illumination by querying for the nearest N photons from the photon map. Then by using the estimation, each light will be calculated.

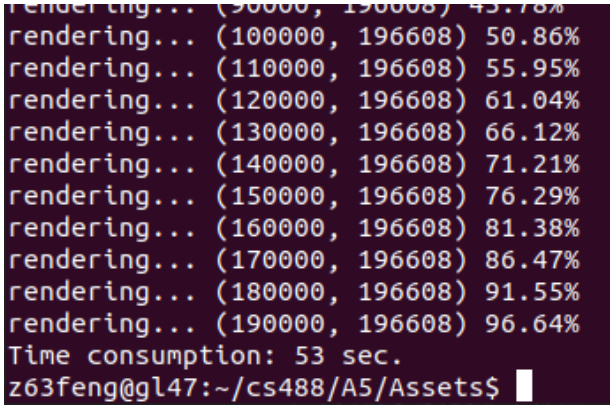
This objective has to be aborted. The reason is that the investigation before the proposal is far from enough. After a further investigation, I found the workload and difficulty is impossible for me to finish in the remaining days. First, it is a completely different ray tracing method. I cannot reuse the ray tracer mechanism in A4 to build the photon map. Second, even if I had done the photon map, I have to write an efficient kb-tree structure to store and query the photons. Third, a mathematical method has to be come up to give the approximation of photons during the rendering. I tried to understand one of them but failed. Lastly, the final effect is not as good as a direct ray tracer if I failed one of the requirements: a large amount of photons which means the structure has to be efficient, the approximation and correction during the rendering.

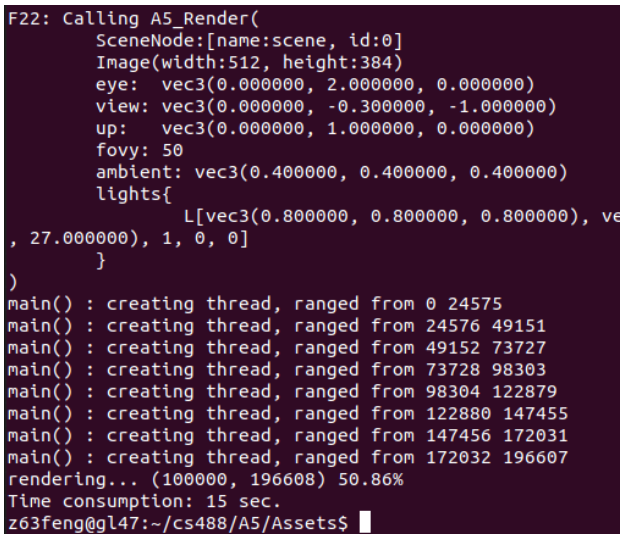
As a consequence, I have to abort this objective but still I want everyone to know that I tried.

4.9 Intel Thread Building Blocks (tbb) multithreading rendering

Currently, we are using a single thread to run our rendering program. It can really be a time killer. Here is the Intel tbb library. By using `parallel_for` interface, we can partitionate it our rendering jobs into multiple threads. For more details, see UserGuide for reference.

During the work of using the third-party library, I had a really bad cross-platform experience using tbb. I had three developing environments. A Mac laptop in school. A PC with wsl2 at home. A linux in the lab. The problem is that on my Mac laptop, it will not be able to link the local library. Although tbb multithreading works well in Ubuntu, I do not have a native linux environment to develop my other work. Most of my work is on Mac. Therefore, I wrote a complex version using pthread library, and that is what I actually use to render all the scenes during the development. To show that I had done Intel TBB, I kept the tbb code in A5.cpp and I will also give a result from the lab machine.

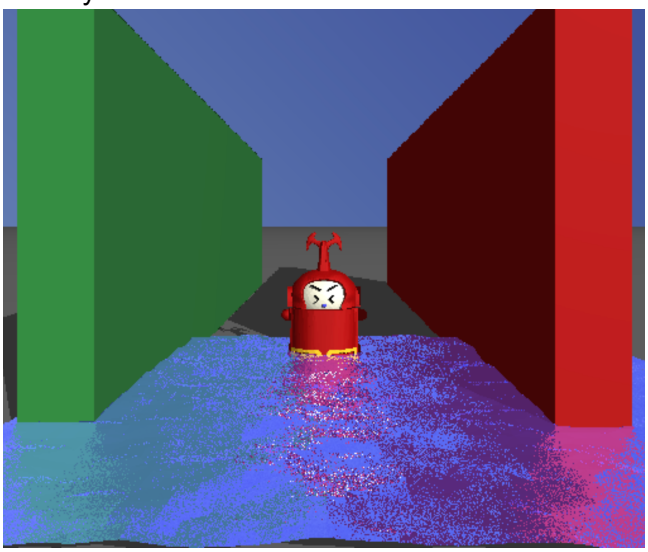




4.10 Final scene

The final scene should give the viewer a feeling of magic and warmth by using lighting and texture effects. The model should be properly coloured. Environmental texture should be reasonably mapped. Roof window should be a light emitting object or a planar light source.

First try to render water.

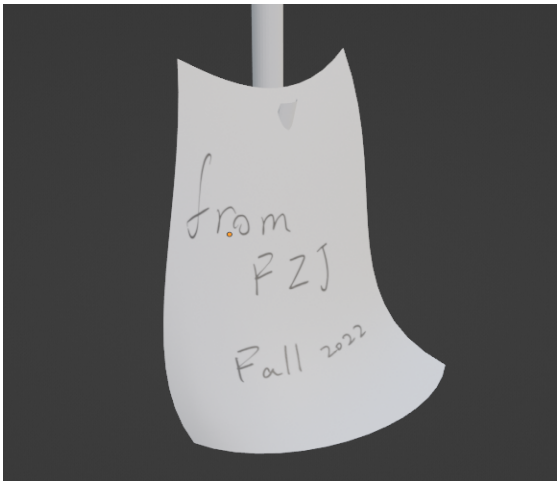


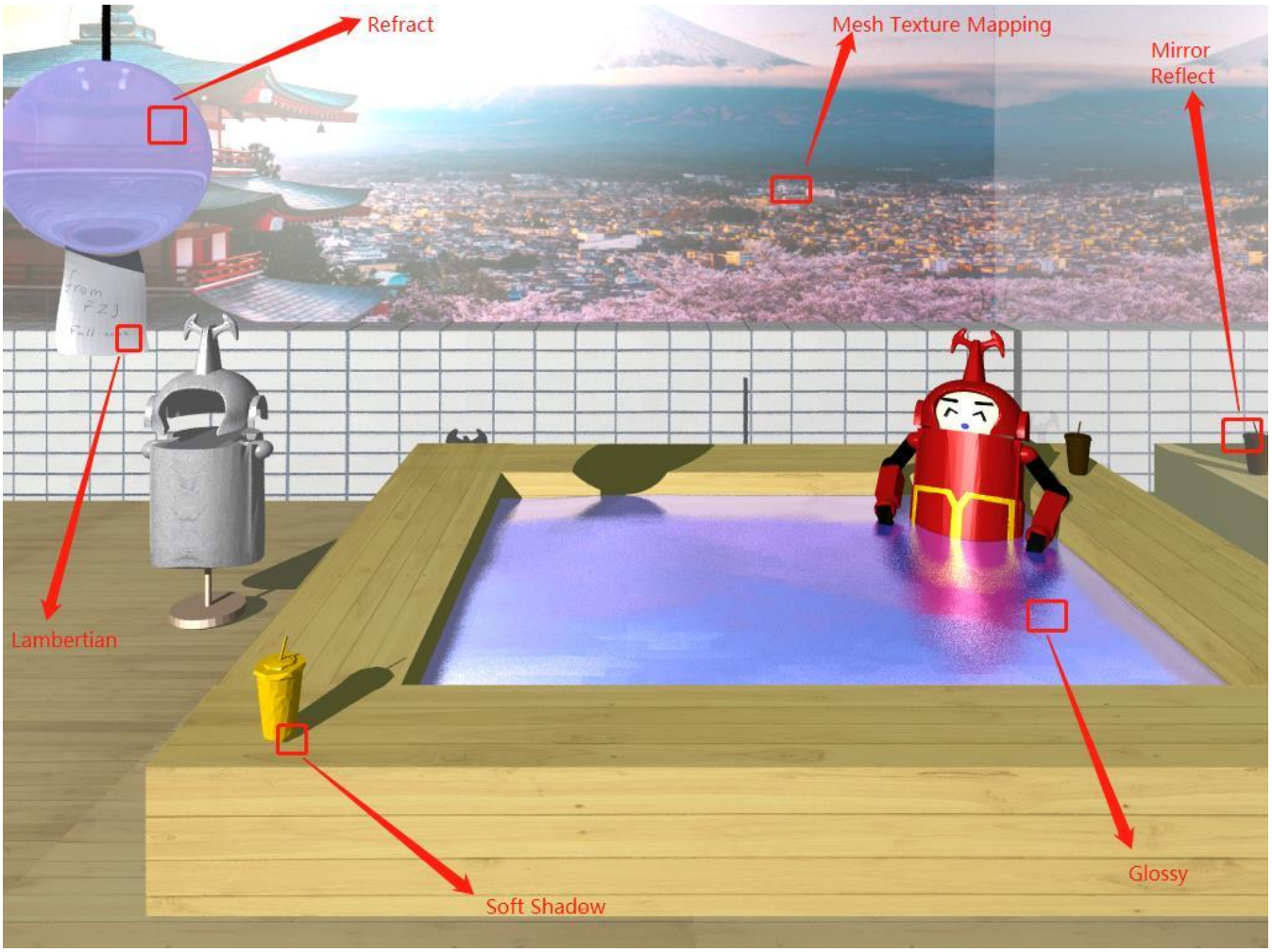
First draft

With environments added.

Final scene done.

An Easter egg is in the pendant.





4 Bibliography

Intel, UserGuide, 2019
https://www.intel.com/content/www/us/en/develop/documentation/tbb-tutorial/top/tutorial-developing-applications-using-parallelfor/develop-an-application-using-parallelfor.html

M, Jakob W, Humphreys G. Physically based rendering: From theory to implementation[M]. Morgan Kaufmann, 2021. Chapter 11: VOLUME SCATTERING
https://pbr-book.org/3ed-2018/Volume_Scattering/Volume_Scattering_Processes

Shirley, Peter. Ray Tracing in One Weekend, 2018.
https://www.realtimerendering.com/raytracing/Ray%20Tracing%20in%20a%20Weekend.pdf

Shirley, Peter. Ray Tracing: The Next Week, 2018.
https://raytracing.github.io/books/RayTracingTheNextWeek.html

Shirley, Peter. Ray Tracing: The Rest of Your Life, 2018.
https://www.realtimerendering.com/raytracing/Ray%20Tracing_%20the%20Rest%20of%20Your%20Life.pdf

Yang, YangWC's Blog, 2019
https://yangwc.com/2019/05/23/RayTracer-Advance/

5 Objectives:

Full UserID: Student ID: 20819676

1. Objective one.
   Complex modeling on Kabutack avatar and bathhouse.
2. Objective two.
   Texture mapping on environmental objects such as grass and lanes.
3. Objective three.
   Soft shadow
4. Objective four.
   refraction
5. Objective five.
   Glossy reflection/refraction
6. Objective six.
   Super-sampling for anti-aliasing
7. Objective seven.
   Simulation of camera focus blur, depth of field.
8. Objective eight.
   Photon mapping
9. Objective nine.
   Intel Thread Building Blocks multithreading rendering
10. Objective ten.
    Final Scene