The CS488 Project Report

Student id:20850175

Ningzhi Hou

Report for Champion

Purpose:

The propose of this project is to explore the interesting topics I learned from CS 488 class and give my own implementations to further empower my raytracer from Assignment 4.

Technical Outlines:

1. **Implementations of cylinder and cone:**

We've already got a class called Primitives in our ray tracer, so that we can create classes for Cone and Cylinders deriving from Primitive class and override hit function. Since primitives can transform after placing into a hieratical structure, LUA scripts don't need to specify the size of the cones and cylinders. We can do 3D affine transformations to draw cylinders and cones of any size.

The most interesting thing about this part is to discuss when the light ray intersects with bottoms of cylinder and cones. After we solve the equations for elliptic cylinder, we can find the intersection point between the ray and cylinder's elliptic curve. We can divide multiple cases for discussions on which range y-value lies.

2. **Anti-Aliasing:**

Anti-Aliasing was the terminology to remove jaggies on the boundaries of two models. Ray tracer works to shoot a ray for each pixel and find its corresponding colors in an image. Each pixel is a tiny square and generates jaggies when there's a curve.

The method implemented in the project for anti-aliasing is called super-sampling: Instead of just shooting one ray, ray tracer shoot another 8 rays around the initial ray and compute the average color. If the difference pass a threshold, indicating a boundary, we use the average color to replace the initial color.

3. **Refraction:**

Refraction can be implemented to simulate Snell's law: When the light travel from one material to another, the quotient of the sine value of incident angle and refraction angle is only related to the two mediums.

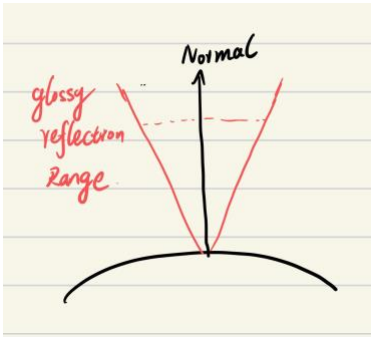
Since we can assume that light always comes from air, so ray tracer suppose to add one more variable to Phongmaterial class to indicate the refraction constant(m_refraction_factor) in my code.

4. **Soft Shadows:**

In Assignment 4, the light in model was considered as a point. That's not true in real case: for areas under shadows, the light intensity increases gradually while points are away from the shadow center and expose more to light source(not a point). So in code, multiple points around light was picked to simulate spherical light and take the mean value to indicate light intensity.

5. **Glossy reflection:**

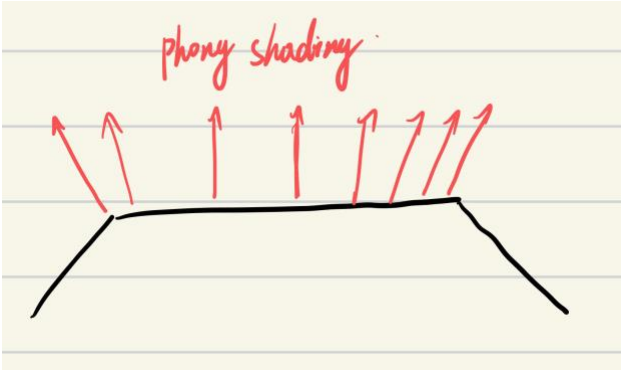
Unlike mirror reflection where light is purely specular, more materials also diffused some light. In my code, I define a constant called glossy radius. Based on normal, two orthonormals perpendicular to our normal can be found:



To simulate the effect of diffusion, multiple reflected rays was generated from the glossy reflection range depending on glossy radius. When glossy radius increases, glossy reflection range increases, more light rays diffuses.

6. Phong Shading on Triangle Meshes:

For flat shadings, ray intersect with triangular meshes and thus, normal's will be fixed if rays intersect with same triangles. However, in Phong shading, normal interpolates differently as the intersection point varies on same triangle due to a different barycentric coordinates. In my ray tracer, I use $\frac{|\vec{v}\times\vec{u}|}{2}$ to compute the areas of triangles and get corresponding alpha, beta and gamma for barycentric coordinates.



Interpolation in different intersection point

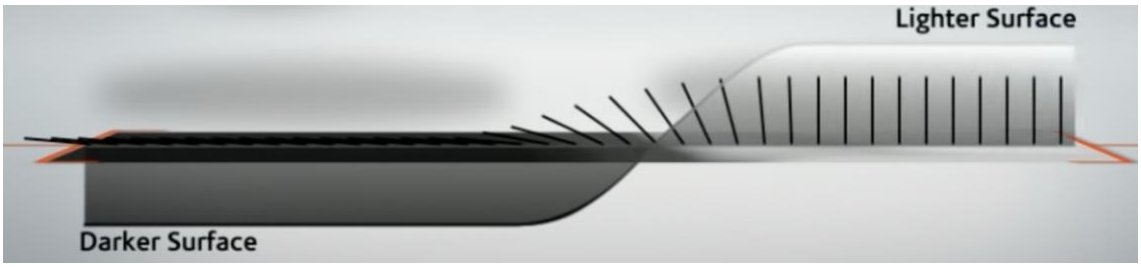
7. 8. 3D texture and Perlin Noise:

My ray tracer create Texture as a base class with 2 virtual function: Get_color(u, v) and Get_color_3D(u,v,w) for both 2D and 3D textures. I used external library glm::perlin(vec3(u,v,w)) to generate a perlin noise, where u, v, w must be decimal value between 0 to 1. 3D perlin noise selected random vector for each vertex with integer x,y,z. For any vertex lies in (x, y, z), (x + 1, y, z), (x, y + 1, z), (x, y, z + 1), (x + 1, y + 1, z), (x + 1, y, z + 1), (x, y + 1, z + 1), (x + 1, y + 1, z + 1) cube, perlin(u,v,w) generate a random linear combination of 8 vectors affliated to 8 vertices. So, there's a clear boundary between 0 to 1 and 1 to 2 as the random vector used are different.

To resolve this issue, I calculate the maximum x, y, z and minimum x, y, z. Use the intersection points' coordinates to divide the size of range of x, y, z. So all vertices are mapped to a vector between 0 to 1.

9.Bump Mapping:

The idea of bump mapping is that we interpolate the normal of a surface due to a pixel we map onto an image:[1]



In this example, the darker the pixel is, the more normel get more perturbed.

Maps of Code:

1. Implementations of cylinder and cones: Go to primitives.cpp for implementations
2. Anti-aliasing: Go to A4.cpp
3. Refraction: Go to A4.cpp, search for //refraction, all codes are listed below(mirror refraction is enabled by default)
4. Soft shadows: Go to A4.cpp, search for #ifdef ENABLE_SOFT_SHADOW
5. Glossy reflection:Go to A4.cpp, search for #ifdef ENABLE_GLOSSY_REFLECTION
6. Phongshading on triangular meshes: Go to Mesh.cpp, all search for ENABLE_PHONG_SHADING
7. Solid Texture: Go to texture.cpp
8. Perlin Noise: Go to texture.cpp and take a look are PerlinNoise Class
9. Bump Mapping: Go to GeometryNode.cpp, search for bumpmapping

Bibliography:

1. YouTube. (2014). *YouTube*. Retrieved December 6, 2022, from https://www.youtube.com/watch?v=D7uK4WWW-Rk.
2. *Adrian's soapbox*. Understanding Perlin Noise. (n.d.). Retrieved December 6, 2022, from https://adrianb.io/2014/08/09/perlinnoise.html