csc305_A01 report

V00823808 Xinyue Liu

Standard Requirements

Explain the design of a ray tracer from the perspective of object-oriented programming.

- Explain the relationships between the hitable, the material, the hit_record, camera (Translate these names to equivalent concepts if you wrote your ray tracer differently.)

In our assignment, the hitable class is for checking if an object hitted by rays. The material class is checking if a ray scattered and calculate how much the ray should be attenuated. Hitable and materials need to know each other. Under hitable class, the hit_reorcd includes the normal vector of the ray hit, the vector ray hits in and the material ray hits on.The material class is also setted up in hitable.h file to tell how rays interact with the surface. When a ray hits a surface, the material pointer in the hit_record will be set to point at the material pointer the sphere was given when it was set up in main() when we start. Camera class represents virtual scene.

Explain how a ray tracer implements the transport of light.

- Talk about how your rays bounce off objects.

The way of how a ray bounce off an object is related to the material of that object. For lambertian material, the reflected ray is randomly scattered. For metal case, the reflected ray v direction is v+2b, where b is the dot of v and normal vector. For dielectircs case, the ray will both reflect and refract.

- Explain how diffusion of light is modeled by a Lambertian (diffuse) material.

Lambertian material reflects light in random direction. First, pick a random point p by executing random_in_unit_sphere(). Then, adding p with normal vector and hitpoint to get a target vector. Then minus target vector to get a reflected ray vector.

Advanced Requirements

Further explain how a ray tracer implements the transport of light.

- Relate your ray tracer to Kajiya's Rendering Equation: https://en.wikipedia.org/wiki/Rendering_equation

Kajiya rendering equation is: outgoing light = emitted light+reflected light. And in my code the rendering equation is emitted+attenuation*color(scattered,world,depth+1), which looks similarly as Kajiya's.

- Explain how reflection and refraction of light are modeled by metal and dielectric materials.

metal: The equation for compute reflected ray is (v-2*dot(v,n)*n). The angle of reflection is equal to the angle of incident ray. Then adding fuzz*random_in_unit_sphere() with reflected ray to make it looks more real. The bigger the sphere, the fuzzier the reflections will be.

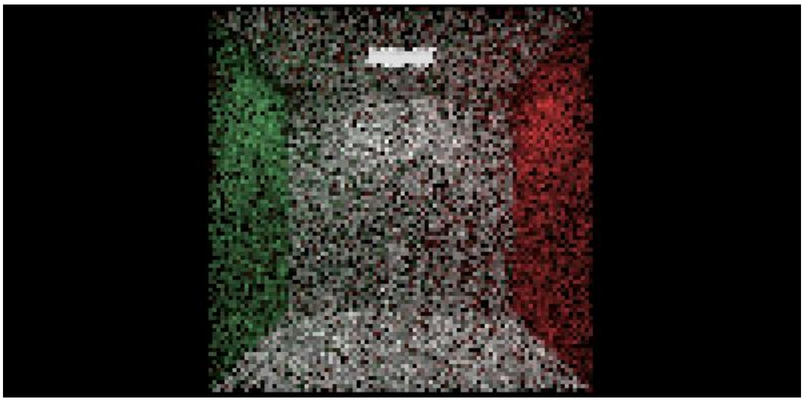
dielectric: The reflected ray is the same as the one used in metal material. The refraction is described by Snell's law: n*sin(theta) = n'sin(theta). n and n' are both the refractive indices. Calculating incident ray to get n and then get n'.

- Explain how a ray tracer can be used to model a lens.

In the code, simulate a real camera following the order: sensor, then lens, then aperture, and figure out where to send the rays and flip the image once computed. First, start rays from the surface of the lens, and send them toward a virtual film plane, by finding the projection of the film on the plane that is in focus at the distance focus_dist.

- **Render a background**
- **Render a sphere**
- **Render multiple spheres**
- **Implement anti-aliasing**
- **Diffuse material**
- **Positional camera**
- **Metal material**
- **Dielectric material**
- **Defocus blur**
- **Motion blur**

- **Image Texture mapping**
- **Solid Textures**
- **Perlin Noise**



- **Rectangles and Rectangular Lights**
- **Render a plane (eg. floor under the sphere) o Do not do this by rendering a big sphere as the floor.**

in additional:

Please comment

35. return attenuation*color(scattered,world,depth+1);
38. return vec3(0,0,0);
42. vec3 unit_direction = unit_vector(r.direction());
43. float t = 0.5*(unit_direction.y()+1.0);
44. return(1.0-t)*vec3(1.0,1.0,1.0)+t*vec3(0.5,0.7,1.0);

in main.cpp and

49. scattered = ray(rec.p, target-rec.p);

in material.h.

And then remove the comment lines that in front of

32. vec3 emitted = rec.mat_ptr->emitted(rec.u,rec.v,rec.p);
34. return emitted+attenuation*color(scattered,world,depth+1);
37. return emitted;
45. return vec3(0,0,0);

in main.cpp and

```
48. scattered = ray(rec.p, target-rec.p,r_in.time());
```

in material.h.

Finally, chang the position of camera to

```
vec3 lookfrom(278,278,-800);
vec3 lookat(278,278,0);
```

Before you run two_perlin_spheres(); simple_light(); cornell_box(); cornell_smoke() and final().