CSC 305 Assignment 1 – Ray Tracer Report

Dora Ou
V00835123

Explanation:

Standard

- Output an image: Set a list to store rgb colors of each pixel of image and write the png image by stb_image_write.h
- Make your own vec3 class: Set vec3 object with 3 float number and identify +/-/*/div/cross/dot between 2 vectors or between float number and vectors
- Render a background: The ray function linearly blends white and blue depending on the up/downess of the y coordinate. I first made it a unit vector so -1.0<y<1.0. Then i did a standard graphics trick of scaling that to 0.0<t<1.0. When t=1.0, I want blue. When t=0.0 Iwant white. In between, I want a blend. This form a “linear blend”, or “linear interpolation” by the form “blended_value = (1-t) *start_value+t*end_value”
- Render a sphere: the equation of sphere is x*x+y*y+z*z=r*r. Let p be any point (x,y,z) and c be the center. We get dot((p-c),(p-c))=r*r. Then set p(t)=A+t*B as a ray. To check if the ray hit the sphere by dot(p(t)-c, p(t)-c)=r*r
- Render a plane (eg. floor under the sphere): Render a rectangle as xz-rectangle
- Render multiple spheres: Set a sphere class. Render multiple spheres by calling it with different center and radius.
- Implement anti-aliasing: get a random number between 0-1 to average a bunch of samples inside each pixel.
- Diffuse material: diffuse object that do not emit light but reflect light. It can either scatter always and attenuate by its reflectance R, or it can scatter with no attenuation but absorb the fraction 1-R of the rays. Or it could be a mixture of those strategies. So the ray can be randomly scattered.
- Positional camera: Set the camera position as lookfrom(x,y,z) and the point we look at lookat. We can rotate the camera by change lookfrom(x,y,z)

Advanced

- Metal material: The ray will get reflected from a metal mirror. The reflected ray direction is (v+2B), v is vector of the ray, N is the normal vector, and B is dot(v,N)*N.
- Dielectric material: When a light ray hits then, it splits into a reflected ray and a refracted ray. We can handle that by randomly choosing between reflection or refraction and only generating one scattered ray per interaction.
- Defocus blur: 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_distance). The aperture of the lens determine defocus blur. Let lens

radius=aperture/2. Rays offset depends on lens radius. If radius become bigger, then the offset becomes bigger and ray directions become wide. Hence, we get a defocus blur.

- Rectangles and Rectangular Lights: We need to add an emitted function, which just tells the ray what color it is and performs no reflection. Then make the background black in our color function and pay attention to emitted. Rectangle can be made in xy, yz, and xz direction to form 3D vision. For example, to form a xy plane, set z=k and an axis-aligned rectangle is defined by lines x=x0, x=x1,y=y0,y=y1. To determine whether a ray hits such a rectangle, we first determine where the ray hits the plane. Recall that a ray p(t) = a+t*b has its z component defined by z(t) = az+t*bz.
- Instances: firstly, create a box by 6 rectangles. Then rotate them a bit by move the rays in the opposite direction by rotating object in x, y, z-axis. For example in x-axis: y' = cos(theta)*y-sin(theta)*z , z'=sin(theta)*y+cos(theta)*z

Bonus

- Motion blur: To generate rays at random times while the shutter is open and intersect the model at that one time. Store a time in ray and modify the camera to generate rays at a random time between time1 and time2. Then creating a moving sphere object random the time between time1 and time2, and get the real time center.
- Solid Textures: Make textured material by replacing the vec3 color with a texture pointer. We can create a checker texture by noting that the sign of sine and cosine just alternates in a regular way and if we multiply trig functions in all 3D, the sign of that product forms a 3D checker pattern.
- Perlin Noise: it takes a 3D point as input and always returns the same randomish number. Nearby points return similar numbers. We could just tile all of space with a 3D array of random numbers and use them in blocks.
- Image Texture Mapping: we can load the image by stb_image.h. Then we need to add u,v, to hitable structure and get attenuation from albedo->value(rec.u,rec.v,rec.p). In the image_texture, we get each color from load image and convert it to our sphere object.
- Volumes: A ray going through there can either scatter inside the volume, or it can make it all the way through like the middle ray in the figure. The probability that the ray scatters in any small distance dL is: probability = C*dL, where C is proportional to the optical density of the volume. If you go through all the differential equations, for a random number you get a distance where the scattering occurs. If that distance is outside the volume, then there is no "hit". For a constant volume we just need the idensity C and the boundary. The scattering function of isotropic picks a uniform random direction.

Questions:

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

The design of a ray tracer involves implementing shading, shadow, reflection, refraction and materials appearance when rays hit objects. A ray tracer is to send rays through pixels and compute what color is seen in the direction of those rays. In object-oriented programming, the factors of ray tracing, such as material and hitabe, can be set as objects (super classes) with several subclasses. The subclasses are different situations of the super classes. For example, there are metal, lambertian, and dielectric in class material. The parameter and subclass can affect the output of ray tracer.

The hit_record is a structure of an object which contains the material of object and it is in class hitable. When a ray hits a surface, the material pointer in the hit_record will be set to point at the material pointer the object(sphere) was given when it was set up in main. The material is to produce a scattered ray and say how much the say should be attenuated, so it need the parameter of hit_record and include hitable.h. Camera can choose the position of lookfrom and lookat. Hitable or not will determine what the position looks like.

Explain how a ray tracer implements the transport of light.

- Talk about how your rays bounce off objects.
- Explain how diffusion of light is modeled by a Lambertian (diffuse) material.

A ray as a function p(t)=A+t*B. p is a 3D position along a line in 3D. A is the ray origin and B is the ray direction. The ray parameter t is a real number. Plug in a different t and p(t) moves the point along the ray. If t is negative, it can transport to anywhere. If t is positive, the ray transport to in front of A.

Different materials of objects get different ray bounce.
For diffuse material case, it can either scatter always and attenuate by its reflectance R, or it can scatter with no attenuation but absorb the fraction 1-R of the rays. Or it could be a mixture of those strategies. So the ray can be randomly scattered.

For metal material, the ray will get reflected from a metal mirror. The reflected ray direction is (v+2B), v is vector of the ray, N is the normal vector, and B is dot(v,N)*N.
For dielectric material, when a light ray hits then, it splits into a reflected ray and a refracted ray. We can handle that by randomly choosing between reflection or refraction and only generating one scattered ray per interaction.

Because diffuse material is not smooth, we need to random the directions of diffusion. Letting the vector of ray A + normal vector + a unit random vector - the vector of ray, then we get a random vector B. Then diffuse scattered is a ray with the origin of A and the direction of the random vector, which is A+t*B.

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

The rendering equation is

$$I(x,x') = g(x,x')\left[\epsilon(x,x') + \int_S \rho(x,x',x'')I(x',x'')dx''\right]$$

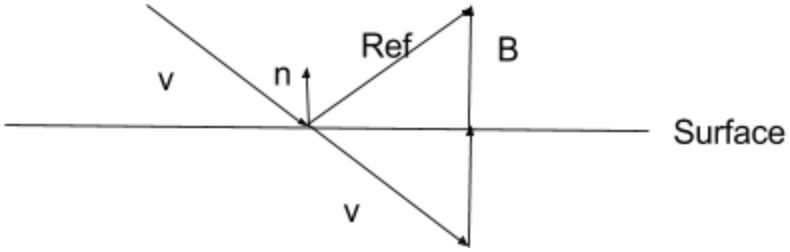
where:

$I(x,x')$	is the related to the intensity of light passing from point x' to point x
$g(x,x')$	is a "geometry" term
$\epsilon(x,x')$	is related to the intensity of emitted light from x' to x
$\rho(x,x'x'')$	is related to the intensity of light scattered from x'' tox by a patch of surface at x'

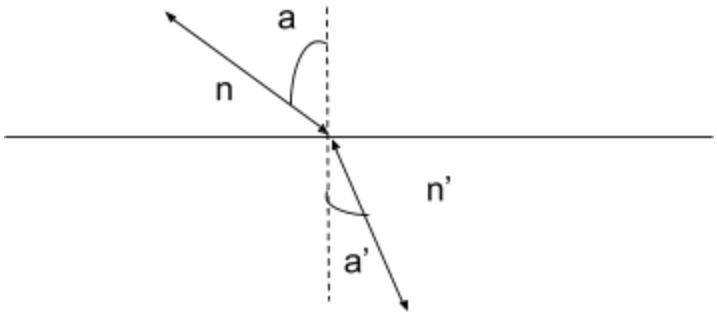
I(x,x') is the ray tracing from object x' to the direction x'x. g(x,x') is objects term. Epsilon(x,x') is the diffuse light material which can emit light. p(x,x',x'') is the scattered system in material. ∫p(x,x',x'')*i(x',x'') compute the ray tracing between different material in total. So the total equation implements ray tracing between objects with light object epsilon(x,x').

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

Metal material reflects the ray by the equation reflection = v-2*dot(v,n)*n. As the graph shown below, v is ray vector, n is normal, Ref is reflected ray. reflection = v-2B. B=n*dot(n,v), that dot(n,v) is the length of B. Hence, we can get the equation reflection = v-2*dot(v,n)*n



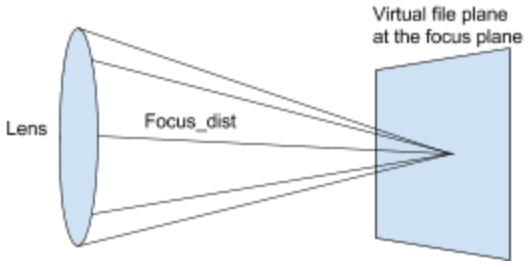
Dielectric material refracts by Snell's law:
n*sin(a) = n'*sin(a'), which as graph shown below.



If the ray hit the object from outside, then the degree of ray and normal is over 90 degree, so the dot product of them is less than 0.
In the refraction function, we should check if the ray can be refracted by a discriminant equation 1.0-(ni/nt)*(ni/nt)*(1-dt*dt). The dt is the dot product of unit ray direction and normal. The product is between 0 and 1. ni/nt is n/n' as shown in graph if the ray is into the object. Then calculate refracted equation by refracted = ni/nt*(uv-normal*dt)-n*sqrt(discriminant).

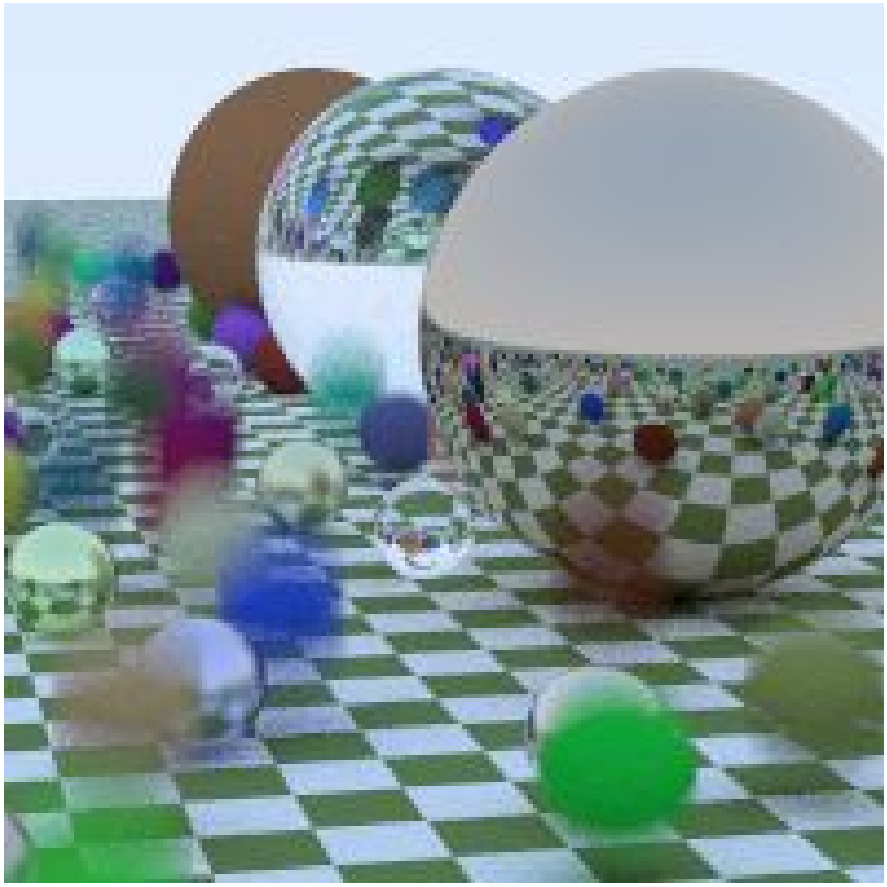
- Explain how a ray tracer can be used to model a lens.

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_distance), shown below.



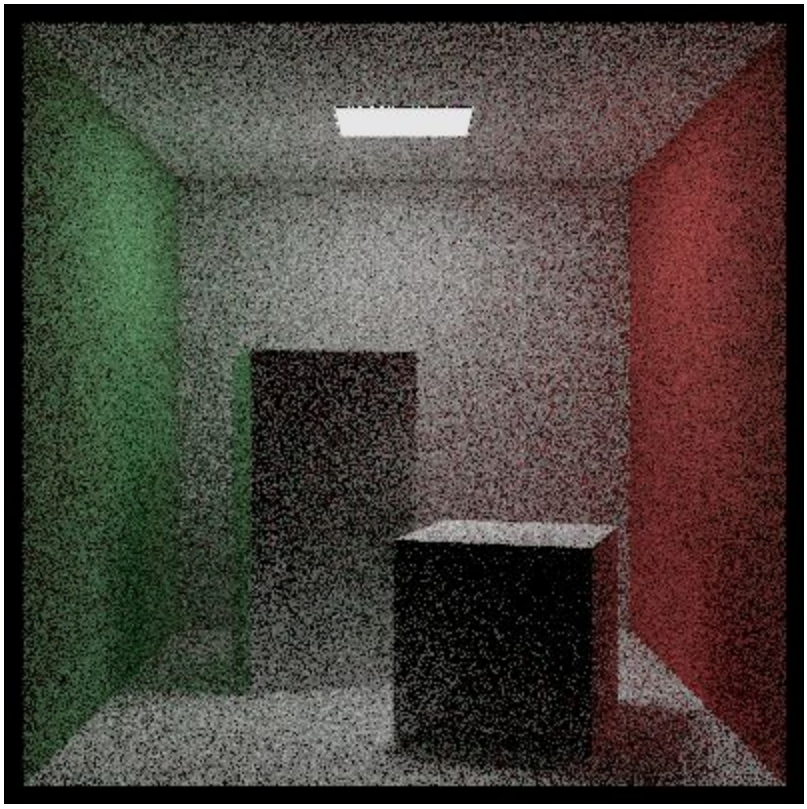
The aperture of the lens determine defocus blur. Let lens radius=aperture/2. Rays offset depends on lens radius. If radius become bigger, then the offset becomes bigger and ray directions become wide. Hence, we get a defocus blur.

The following image's function can be found in the main.h:



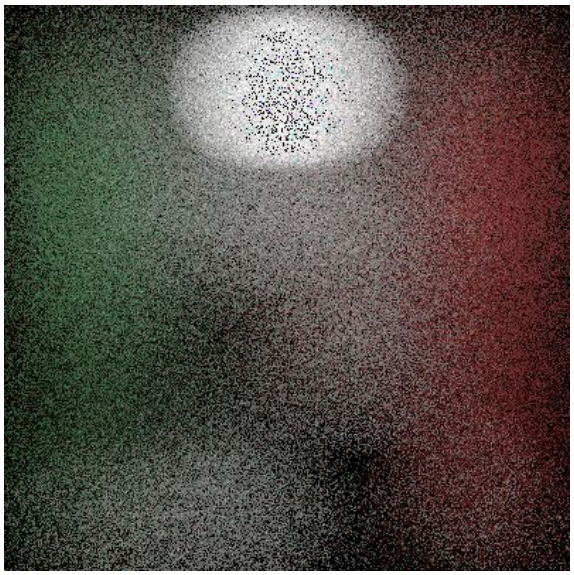
This image implement:

- a background
- 200 random spheres
- 3 different materials
- set a position of camera
- motion blur
- Anti-aliasing (in main function)
- Solid texture

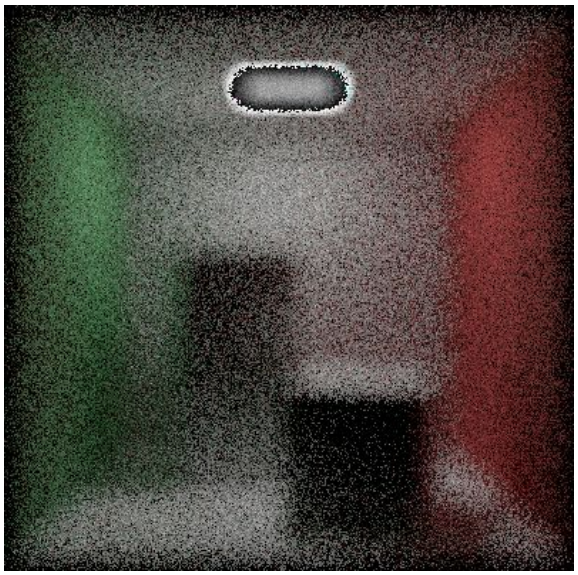


This image implements:

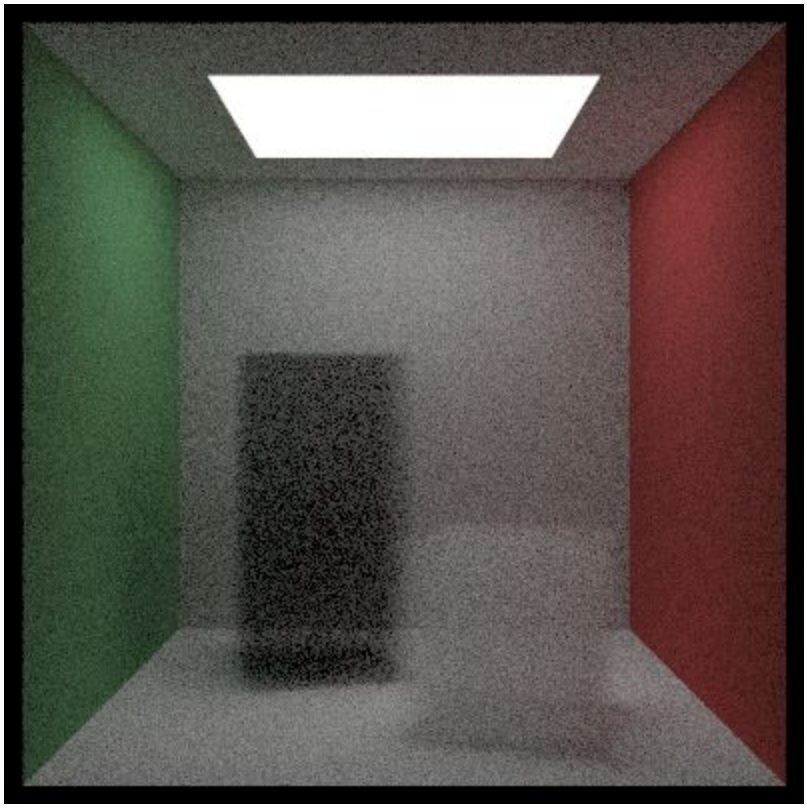
- Instances
- Rotation



This image implement defocus blur
with aperture=2.0

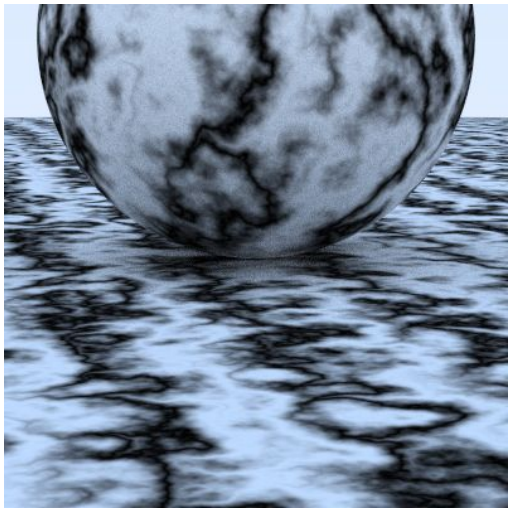


This image implement defocus blur
With aperture = 0.5



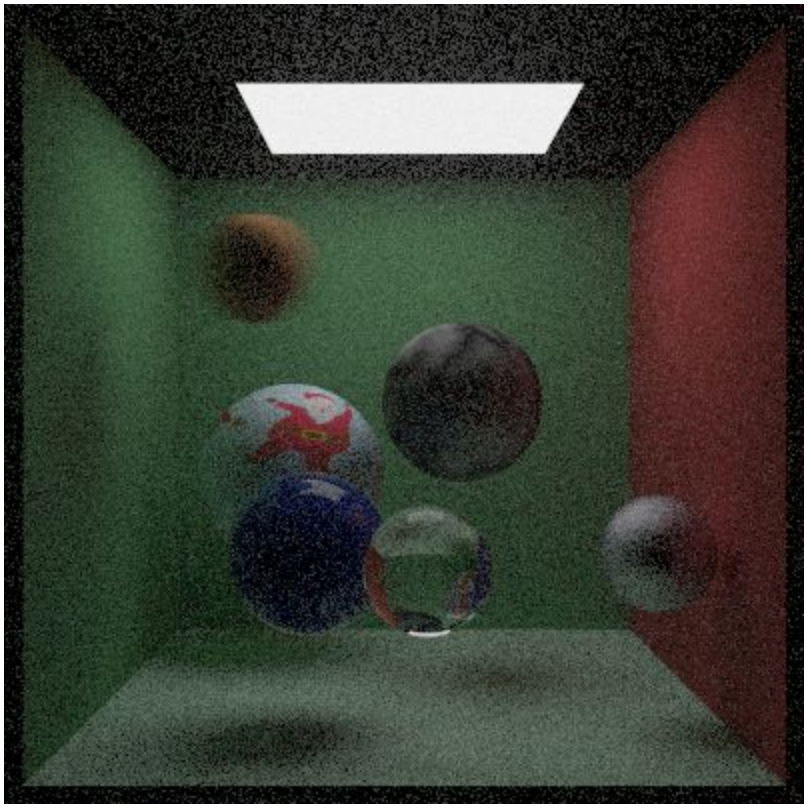
This image implements:

- Rectangle and lights
- Render a plane (floor)
- volumes



This image implements:

- Perlin Noise



This image implements:

- Image texture mapping
- Perlin Noise