Project 3: Image-Based Lighting

Name: Minsuk Kim(m.kim)
Instructor: Dr. Gary Herron
Class: CS562
Semester: Spring 2022
Date: 3/11/2022

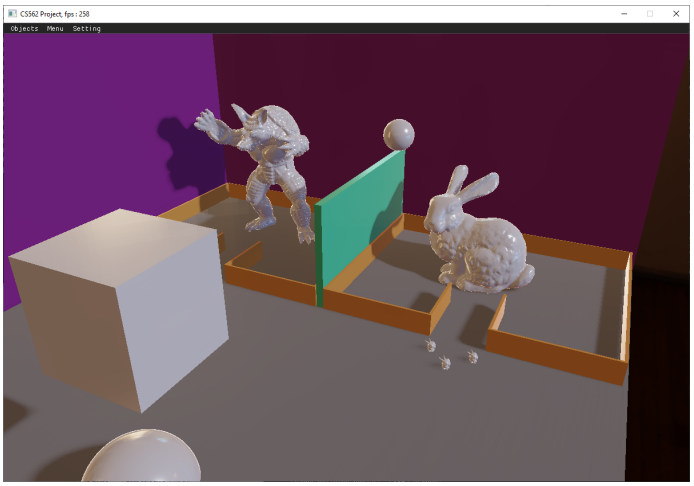


Table of Contents

Introduction

Overview

The main purpose of the project is to implement physically-based lighting and image-based lighting. The physically-based lighting is a more realistic rendering technique. The image-based lighting uses an environment map for providing surround lighting for object lighting calculation. To provide surround lighting, the irradiance mapping for environment texture will be needed.

In this project, I will cover the three different micro-facet BRDF models for lighting calculation and image-based lighting as well. The program will have two passes that involve in this project: irradiance mapping pass and lighting calculation pass. The irradiance mapping uses the environment to generate the irradiance map from it by projecting all pixels onto nine spherical harmonic irradiance coefficients. The irradiance mapped texture will be used in the diffuse calculation for image-based lighting. The lighting calculation pass calculates the physically-based lighting by reading the g-buffer data(albedo color will be needed to convert to sRGB color space). Also, It calculates the image-based lighting with the irradiance texture and environment texture. After all lighting calculations are done, it tone mapped the output color and converts it to linear color space.

I started the project with Vulkan API and GLFW. I also used an external library Assimp that read the object files.

Implementation

The physically-based lighting uses the micro-facet BRDF lighting equation for calculating light. The rendering sequence for this project will be the following: generating an irradiance environment map, calculating the physically-based lighting, calculating the image-based lighting, and tone-mapping and color space conversion.

First, the program will generate an irradiance environment map from the environment map. The program uses compute shader in the initial stage to pre-generate the irradiance texture. The compute shader iterates through all pixels on a full HDR image to generate a low-resolution irradiance map for efficiency. It projects all pixels onto nine spherical harmonic irradiance coefficients and evaluates each pixel's irradiance.

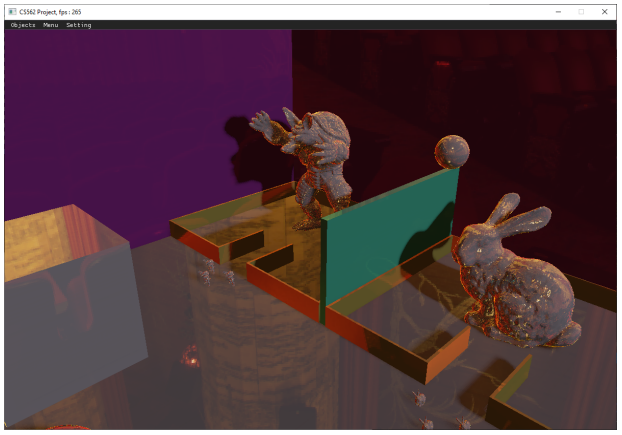
The next step is physically-based lighting. The program uses g-buffer values for calculating the lighting values with BRDF lighting calculation. In this project, I cover the three different models for PBR, Phong BRDF, GGX BRDF, and Beckman BRDF. All models have minor differences in the output of lighting calculation. The program calculates the distribution, geometry, fresnel value and combines the values for getting the final lighting values.

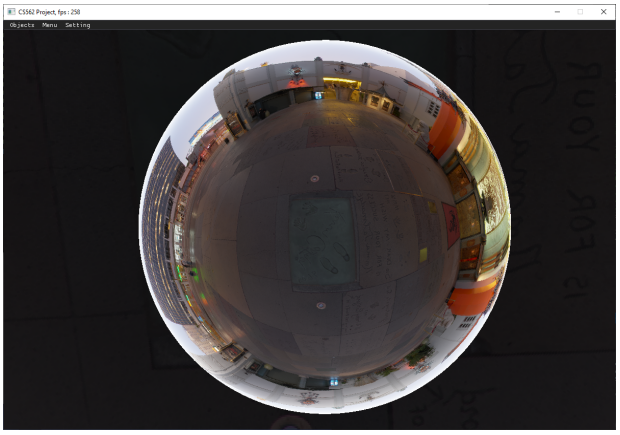
Next, the program calculates the image-based lighting. The image-based lighting handles multiple lightings that come from the environment. It reads two textures(irradiance map and original map) for lighting calculation. The irradiance mapped texture is for a diffuse portion of lighting. The original map is for specular reflection. For efficiency, the program uses mipmap for the original map to reduce the cost of reading the original map(which has high resolution).

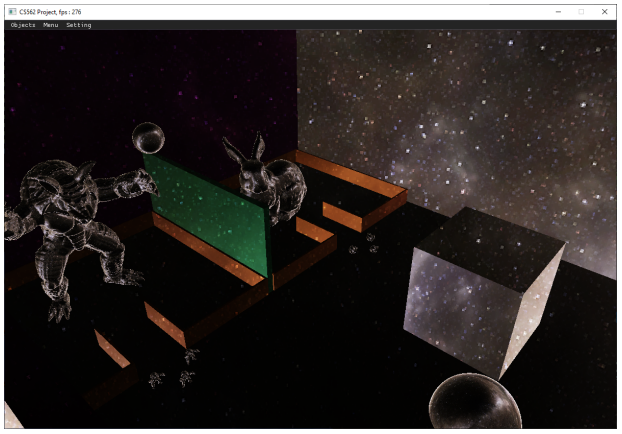
The final step is tone mapping and conversion in color spaces. When the program reads the g-buffer value, it converts the albedo values to linear color spaces from sRGB color space. After the program ends the lighting calculation, It converts the output color to sRGB color space from linear color space. The tone mapping is using a gamma value to adjust the whole brightness of the output color.

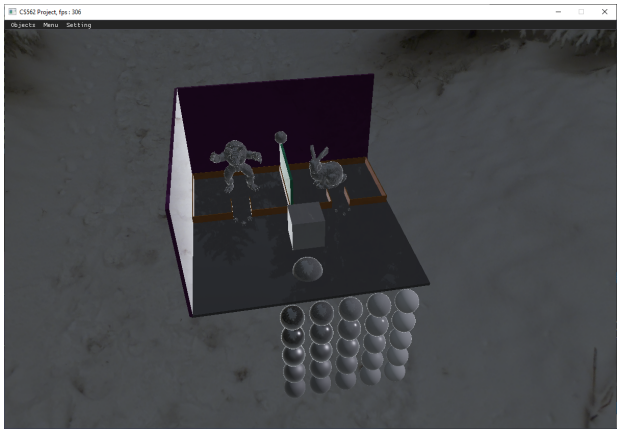
Result Images

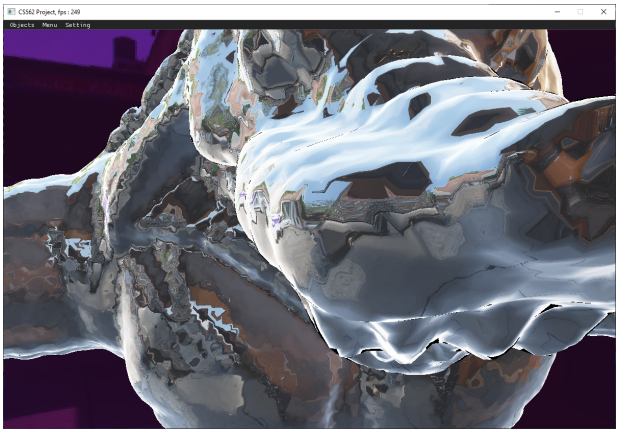
some sample images

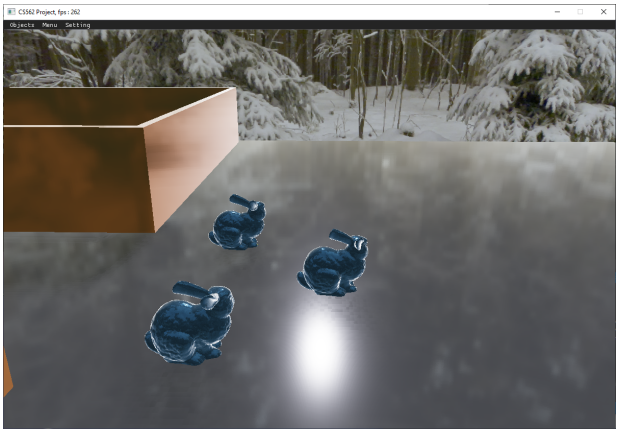






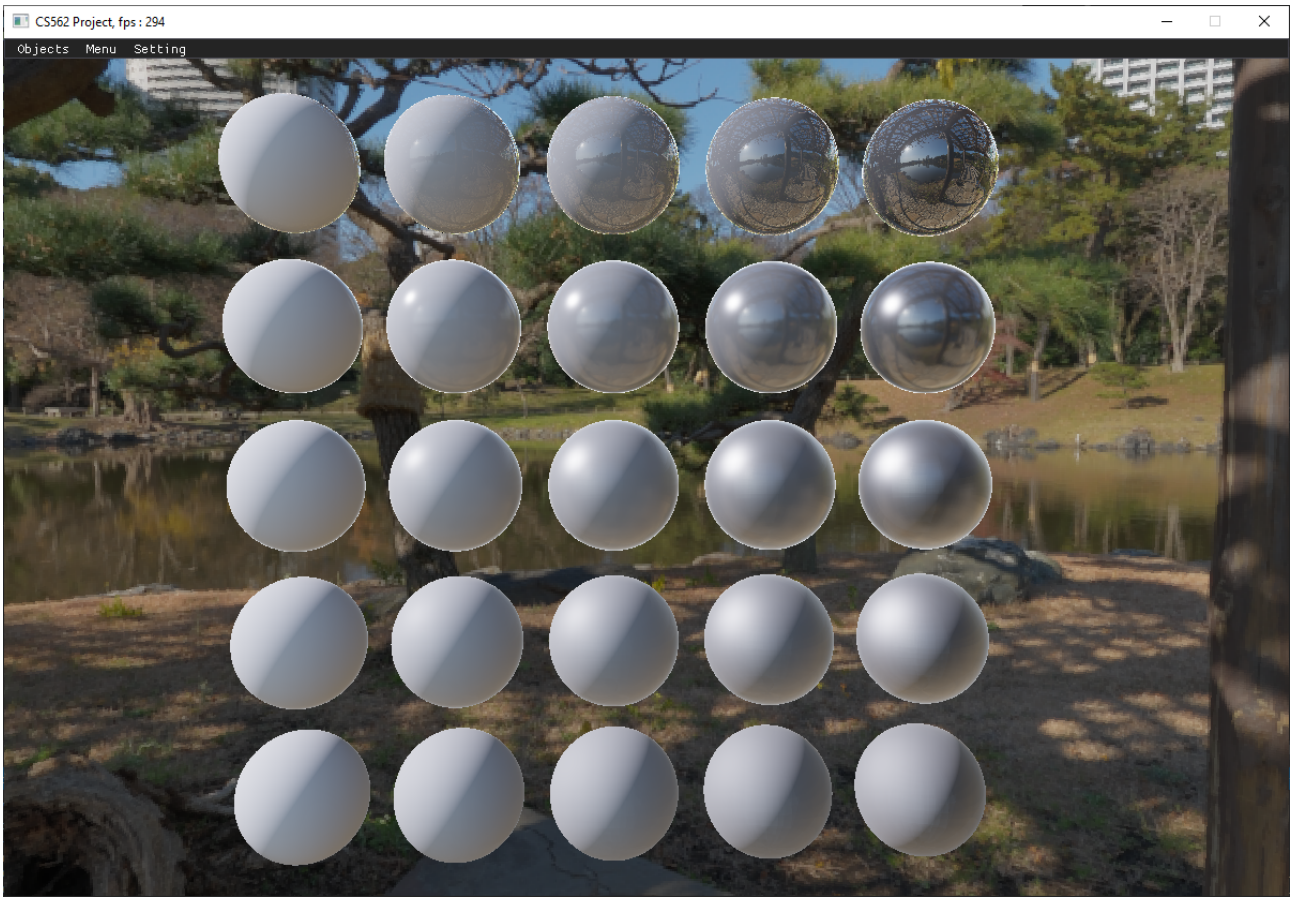


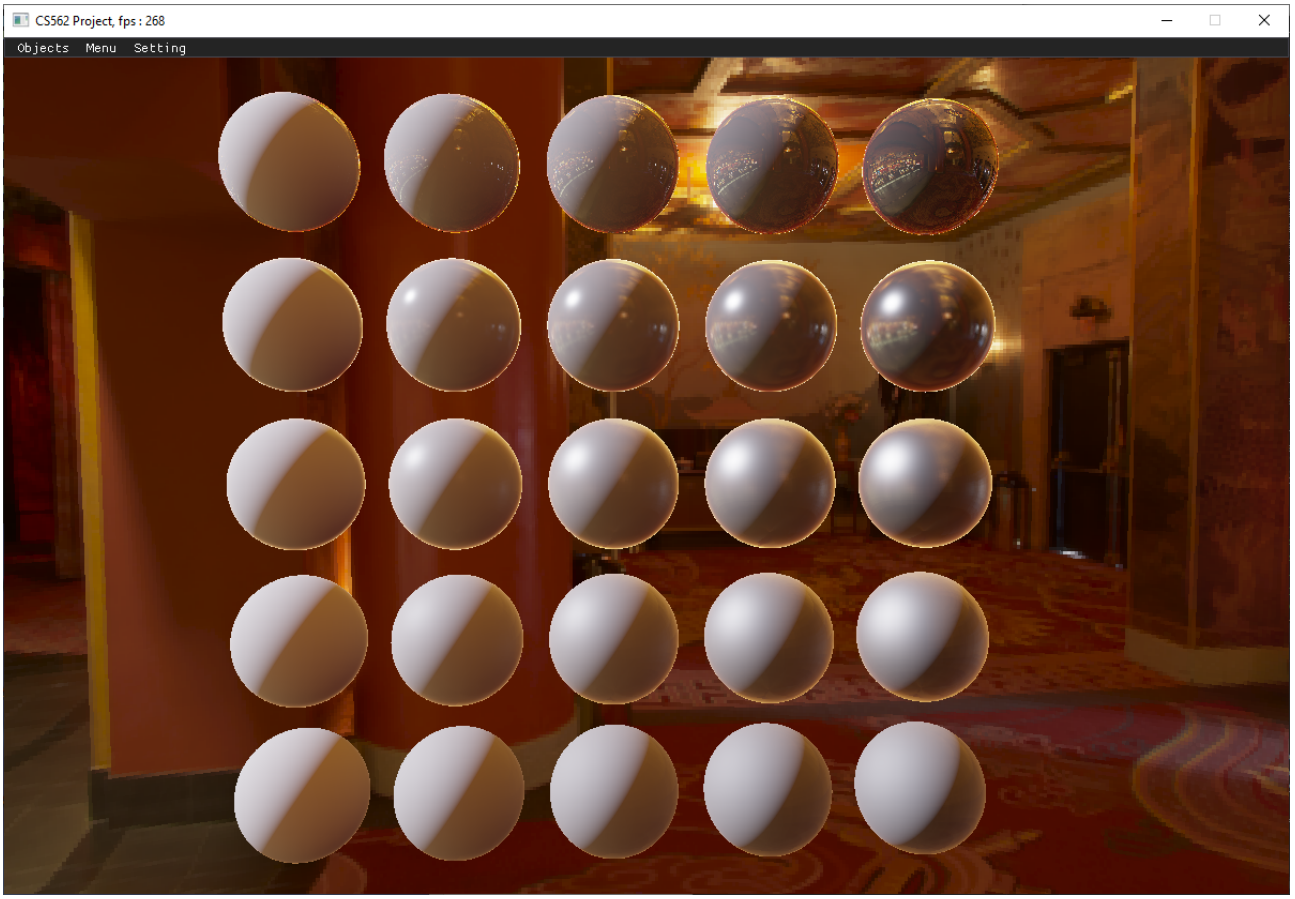


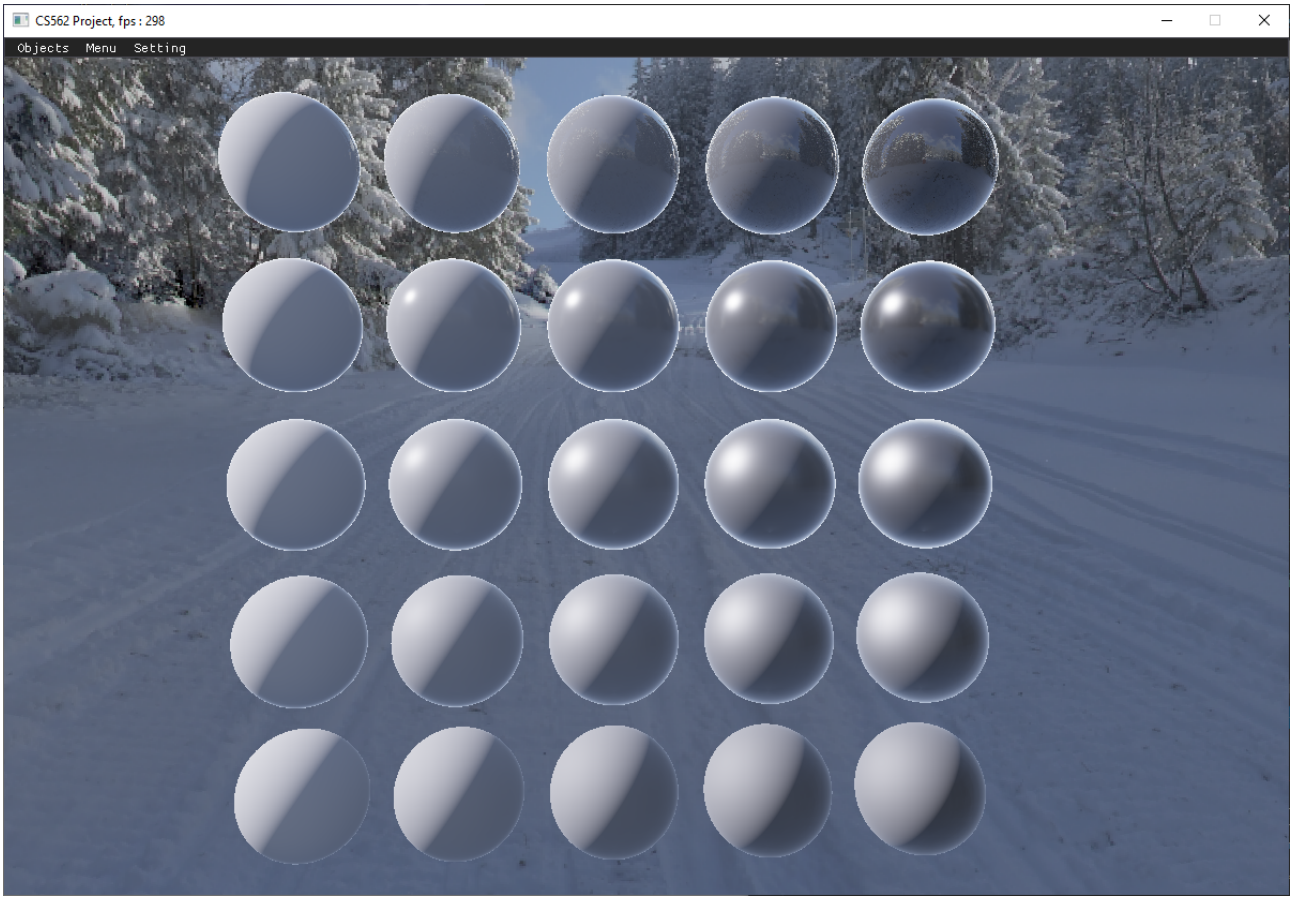


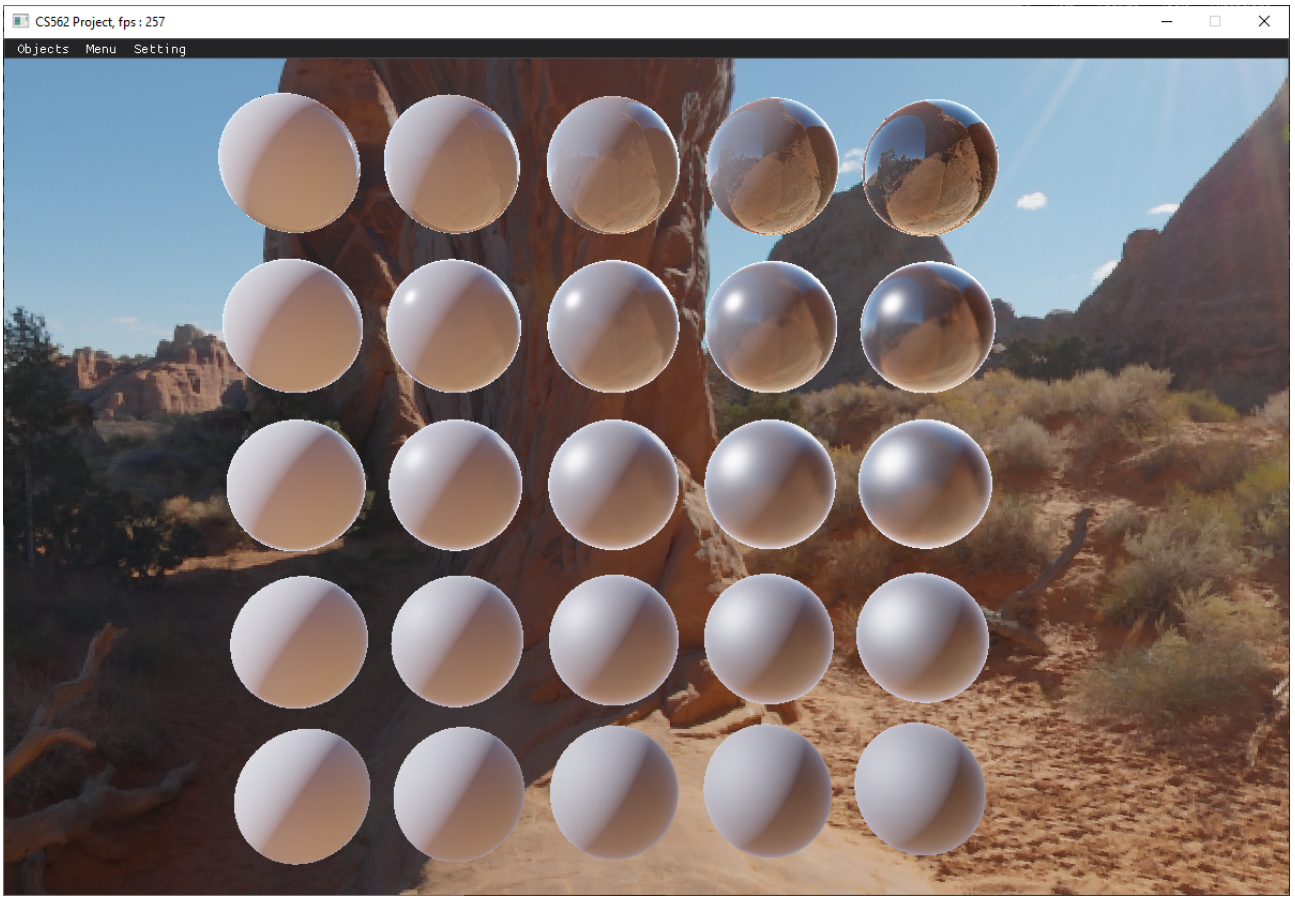
Images with various roughness and metal

the top-right corner sphere is full-reflective






CS562 Project, fps : 298
Objects Menu Setting


CS562 Project, fps : 257
Objects Menu Setting

Notes

PBR

In this project, the program used the micro-facet BRDF lighting equation. The lighting equation will be calculated by

$$D_p(m) = \frac{a_p+2}{2\cdot\pi}(m \cdot N)^{a_p}$$

$$D_b(m) = \frac{1}{\pi a_b^2 (N\cdot m)^4} e^{\frac{-tan^2\theta_m}{a_b^2}}$$

$$D_g(m) = \frac{\alpha^2}{\pi((N\cdot h)^2\cdot(\alpha^2-1)+1)^2}$$

where D_p is the normal distribution for the Phong model

D_b is a normal distribution for the Beckman model

D_g is the normal distribution for the GGX model

$a_p = -2 + 2/(a_g^2)$ a_g and a_b is the roughness value

$$tan\theta_m = \sqrt{1.0 - (m \cdot N)^2}/(m \cdot N)$$

$$H = normalize(V + L),\ F = F_0 + (1 - F_0)\cdot(1 - (H \cdot V))^5$$

F_0 is calculated with $(\frac{1-ns}{1+ns})^2$ where ns is the refractive index of the material

$$G(N,V,L) = G'(V,N)\cdot G'(L,N)$$

$$G_p'(V,N) = \frac{3.535a+2.181a^2}{1+2.276a+2.577a^2}$$ if a >= 1.6, the values will be 1

where $tan\theta_v = \sqrt{1.0 - (v \cdot N)^2}/(v \cdot N)$,

$$a = \sqrt{a_p/2 + 1}/tan\theta_v$$

$$G_b'(V,N) = \frac{3.535a+2.181a^2}{1+2.276a+2.577a^2}$$ if a >= 1.6, the values will be 1

where $a = 1/(a_b \cdot tan\theta_v)$

$$G_g'(V,N) = \frac{2}{1+\sqrt{1+a_g^2+tan^2\theta_v}}$$

$$f_{cook-torrance} = \frac{D*F*G}{4(V\cdot N)\cdot(N\cdot L)}$$

$$f_{lambert} = \frac{c}{\pi}(c\ is\ color)$$

$$final = ((1 - F)\cdot f_{lambert} + F\cdot f_{cook-torrance})\cdot(N \cdot L)$$

each model(Phong, GGX, Beckman) has no big difference in output.

IBL

The image-based lighting is a lighting technique that used the environment texture to calculate the lighting from all surrounded lighting. The IBL is calculated by

$$L_{output} = L_{specular} + \frac{Kd \cdot albedo}{PI} L_{irradiance}$$

, where irradiance color is read from irradiance environment texture. The specular color will be calculated by

$$L_{specular} = \frac{1}{n} \Sigma \frac{G \cdot F}{4(w_k \cdot N)(V \cdot N)} L_i(w_k) cos\theta_k$$

, where w_k is a random distributed directional vector and L_i is the value from environment texture.

The $cos\theta_k$ is calculated by

$$\theta_k = cos^{-1}(\xi_2^{\frac{1}{a+1}})$$ in the Phong model

$$\theta_k = tan^{-1}(\sqrt{-a^2 log(1 - \xi_2)})$$ in Beckman model

$$\theta_k = tan^{-1}(\frac{a\sqrt{\xi_2}}{\sqrt{1-\xi_2}})$$ in the GGX model

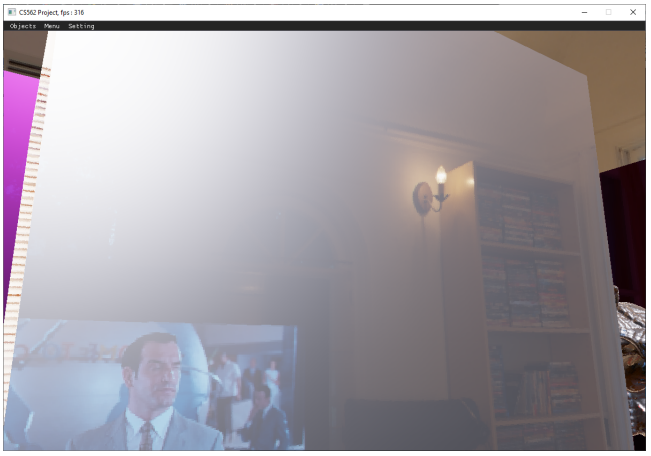
, where a is roughness value on each model.

Low-discrepancy random

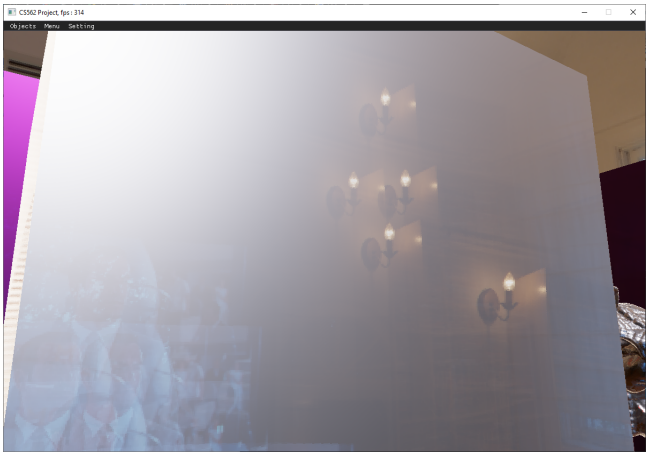
To get the proper uniform distribution of surrounding environment light direction, the program uses a low-discrepancy random generator. The program generates the array of data called Hammersley in the initial stage and passes it into the shader. The shader then uses the pre-generated uniformly distributed random number to compute all surrounding environment light.

The proper size of the random number is 30 - 40. The following images show how low-discrepancy works. (to visibility, the mipmap will be disabled for the following images)

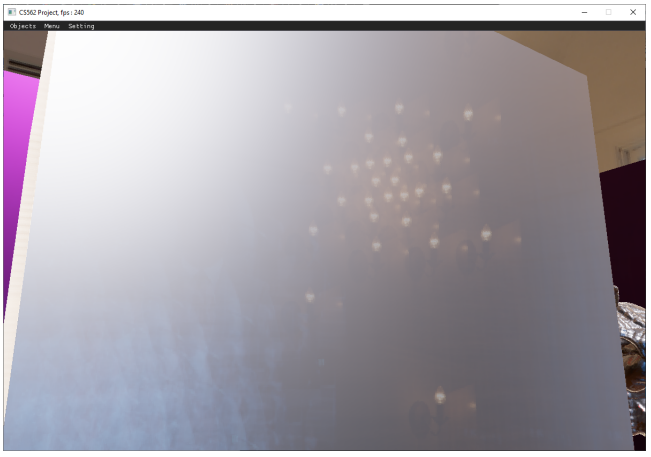
no block size



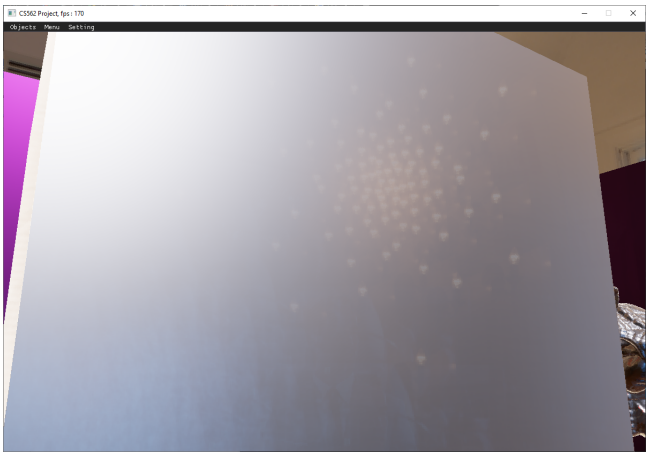
block size 5



block size 30



block size 100



Mipmap

In the image-based lighting calculation, we approach the environment map at different levels for better efficiency. The program will generate the mipmaps when the program loads texture in the initial stage. We calculate which level should the program approach by calculating:

$$level = 0.5 \cdot (log_2(W \cdot H/N) - log_2(D(H))$$

, where N is the number of samples in a Low-discrepancy random paragraph and D(H) is distribution.

This is implemented on line 81 in lighting.frag

Equirectangular

The environment texture is stored in a 2-dimensional texture with equirectangular projection. To convert this 2-dimensional texture to a 3-dimensional environment, we have to convert the texture coordinate to a world coordinate. Assume the texture coordinate is UV and the world coordinate is V, the conversion occurs with the following:

$$UV = (0.5 - \frac{atan(V.x, V.z)}{2\pi}, \frac{acos(v.y)}{\pi})$$

$$V.x = cos(2\pi \cdot (0.5 - UV.x)) \cdot sin(\pi \cdot UV.y)$$

$$V.y = sin(2\pi \cdot (0.5 - UV.x)) \cdot sin(\pi \cdot UV.y)$$

$$V.z = cos(\pi \cdot UV.y)$$

This is implemented on line 114 in light.glsl.

Irradiance mapping

To compute the diffuse portion on image-based lighting, the program needs to read the high-resolution HDR texture to get the diffuse color of the object. However, this is very inefficient to read the high-resolution HDR color. For efficiency, we generate the low-resolution(about 400X200 size) irradiance environment map that will be used in the diffuse calculation. The program uses compute shader that iterates all pixels and projects pixels onto nine spherical harmonic coefficients. After computes all nine spherical harmonic coefficients, the compute shader generates irradiance textures. The pixel data will be

$$pixel = \Sigma EY(x, y, z)$$

where $E$is spherical harmonic coefficients and Y is spherical harmonic basis functions. The spherical harmonic basis functions are calculated with

$$Y_{0,0}(x, y, z) = \frac{1}{2\sqrt{\pi}}$$

$$Y_{1,-1}(x, y, z) = \frac{y\sqrt{3}}{2\sqrt{\pi}} \quad Y_{1,0}(x, y, z) = \frac{z\sqrt{3}}{2\sqrt{\pi}} \quad Y_{1,1}(x, y, z) = \frac{x\sqrt{3}}{2\sqrt{\pi}}$$

$$Y_{2,-2}(x, y, z) = \frac{xy\sqrt{15}}{2\sqrt{\pi}} \quad Y_{2,-1}(x, y, z) = \frac{yz\sqrt{15}}{2\sqrt{\pi}} \quad Y_{2,0}(x, y, z) = \frac{\sqrt{5}}{4\sqrt{\pi}}(3z^2 - 1)$$

$$Y_{2,1}(x, y, z) = \frac{xz\sqrt{15}}{2\sqrt{\pi}} \quad Y_{2,2}(x, y, z) = \frac{\sqrt{15}}{4\sqrt{\pi}}(x^2 - y^2)$$

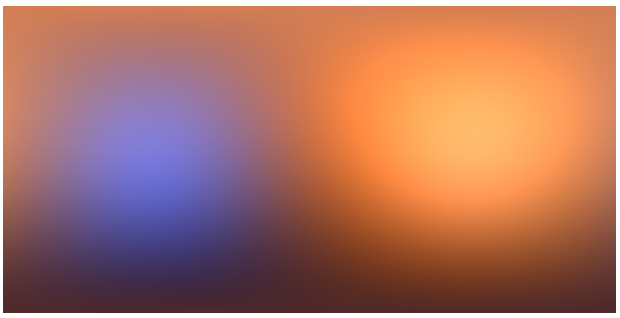
The spherical harmonic coefficients are calculated by

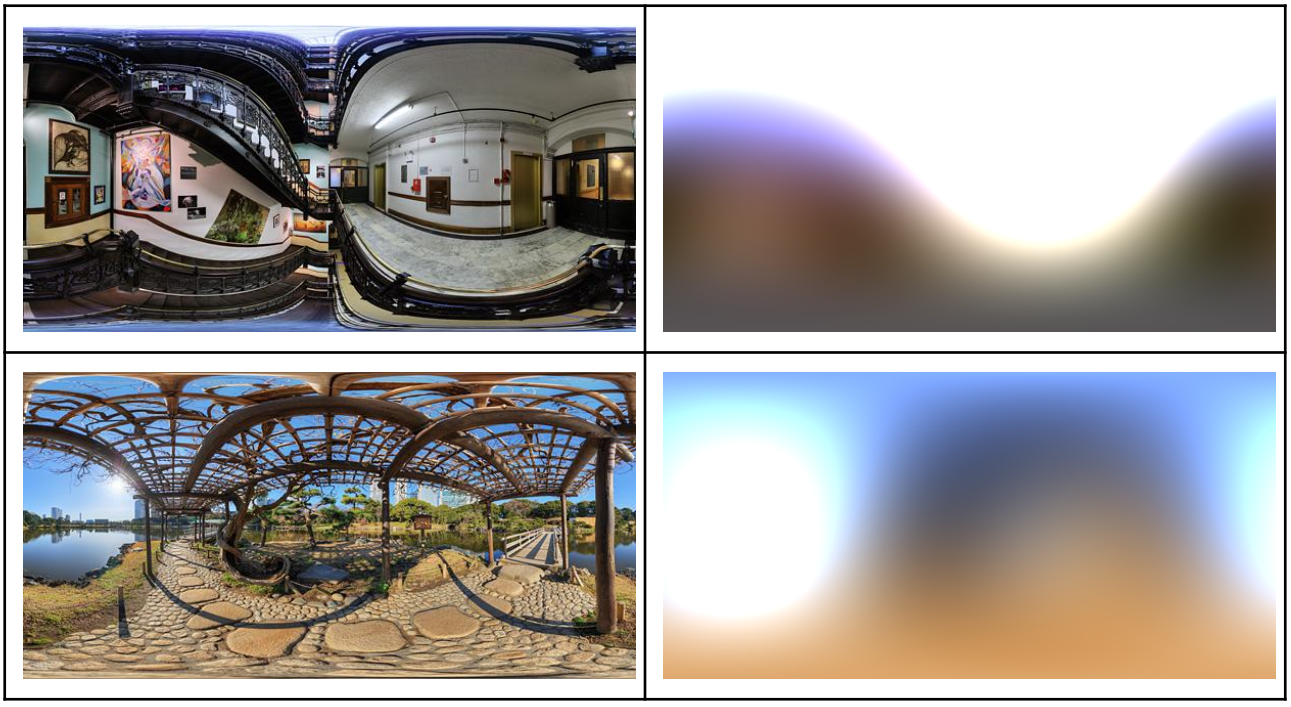
$$E_{lm} = A_l L_{lm}$$

Where A is $A_0 = \pi A_1 = \frac{2}{3}\pi A_2 = \frac{1}{4}\pi$ and L_{lm} is calculated from iterating all original texture pixels.

$$L_{lm} = L(i, j) Y_{lm}(x, y, z) sin\theta \frac{2\pi^2}{HW}$$

where L(i, j) is a pixel of the original texture and θ is an angle in spherical coordinate. i and j are pixel coordinates.

Original Texture	Irradiance Map
	
	



This is implemented in envmapping.comp

Tone mapping and color space conversion

The program uses environment texture with HDR format, which has color values in linear color space. Since the output values should be in sRGB space, the program calculates the lighting values in linear color space and then converts them into sRGB space. In this project, the program reads three color inputs for lighting calculation: albedo color of the object and two environment textures. The environment textures are in linear color space so we don't need to convert them. The program needs to convert albedo color into linear color space before the lighting calculation. We can convert color space with

$$C_{linear} = C_{sRGB}^{2.2}$$
$$C_{sRGB} = C_{linear}^{c/2.2}$$

, where 2.2 is the gamma value for changing the color space between linear color space and sRGB color space and c is the contrast parameter.

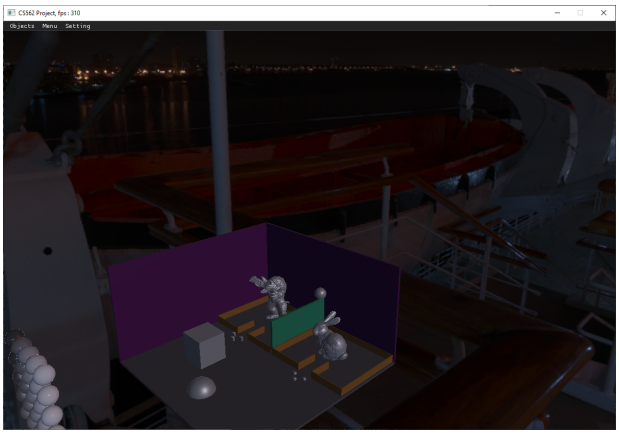
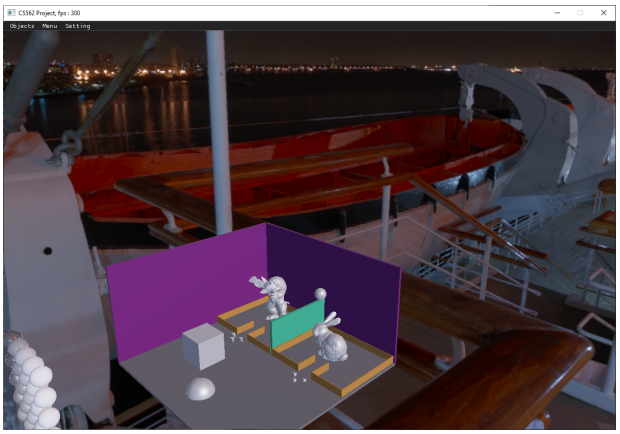
The program needs to set the brightness since the program uses HDR images with a range of 0 to infinity. We want to convert it into a displayable range of 0 to 1. The program uses tone mapping to change the range of HDR values with the following:

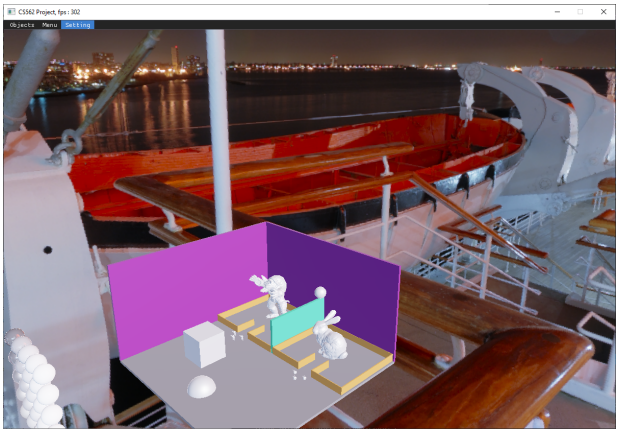
$$C_{SDR} = \left(\frac{eC_{HDR}}{eC_{HDR}+(1,1,1)}\right)$$

, where e is exposure control.

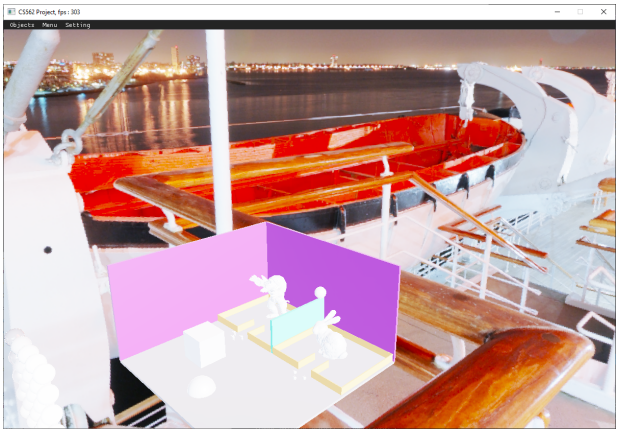
In this project, I combined two functions for adjusting the final output color.

The following table shows the difference between exposure and contrast values. (all objects goes full reflective for visibility)

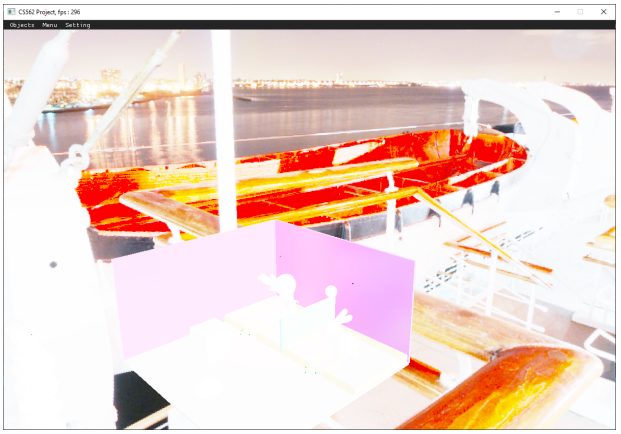
Exposure varies, Contrast = 1	
exposure : 0.1	exposure : 1
	
exposure: 5	exposure: 50

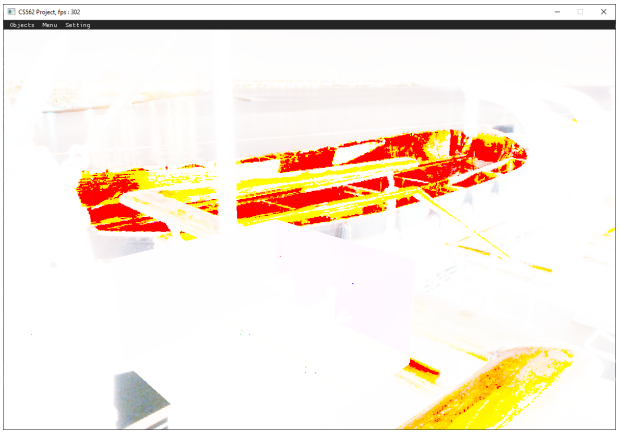


exposure: 500



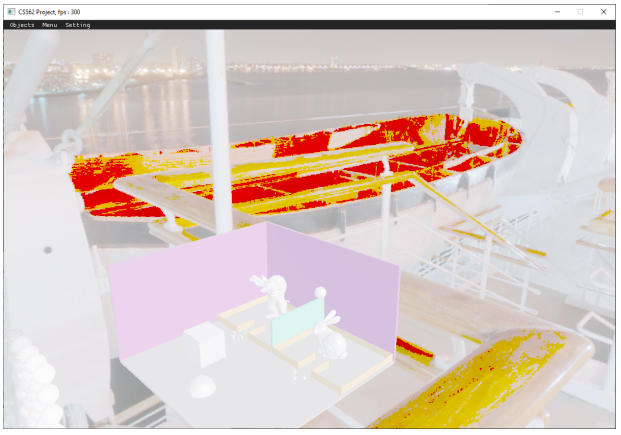
exposure: 10000





Contrast varies, Exposure = 1

contrast : 0.1



contrast : 5

This is implemented on line 158 in light.glsl