Project 1: Deferred Shading

Name: Minsuk Kim(m.kim)
Instructor: Dr. Gary Herron
Class: CS562
Semester: Spring 2022
Date: 1/24/2022

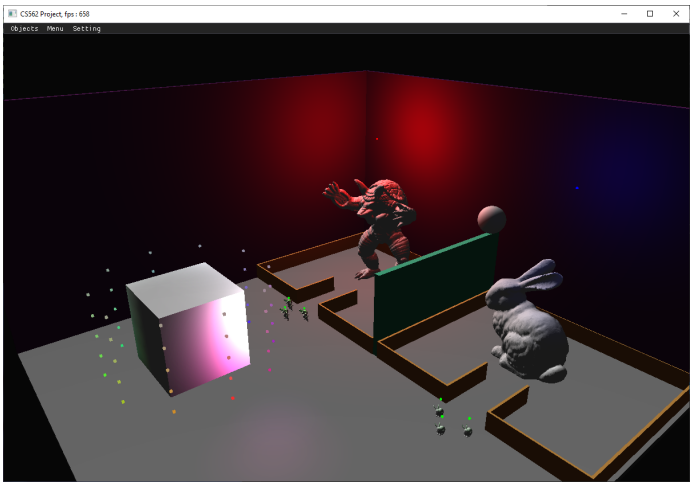


Table of Contents

Introduction

Overview

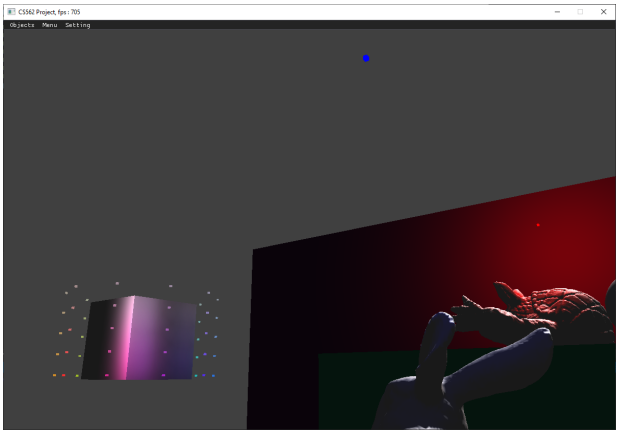
The main purpose of the project is to implement deferred rendering that will cover many local lights. The deferred rendering is an efficient rendering technique for lighting calculation. It first stores the data of each object to images that will be used in lighting calculation. After storing all information, draw the full-sized quad that fits the window size and do a lighting calculation with the stored information. In this project, my program will have three main passes in total: G-buffer pass, lighting pass, and Local lights pass. The G-buffer pass is storing the values that will be used in lighting calculation such as position, normal, albedo, and so on. The lighting pass is calculating the lighting of the global light and gets the pixel color with the global lights. The Local lights pass will draw the area of the local-lights range and then calculate all other lightings with the local lights in that range. I started the project with Vulkan API and GLFW. I also used an external library Assimp that read the object files.

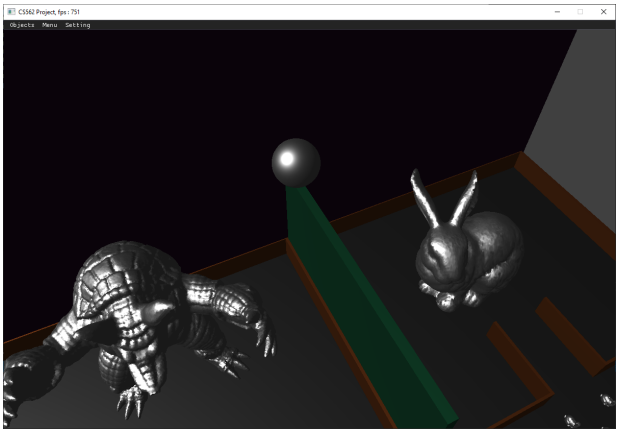
Implementation

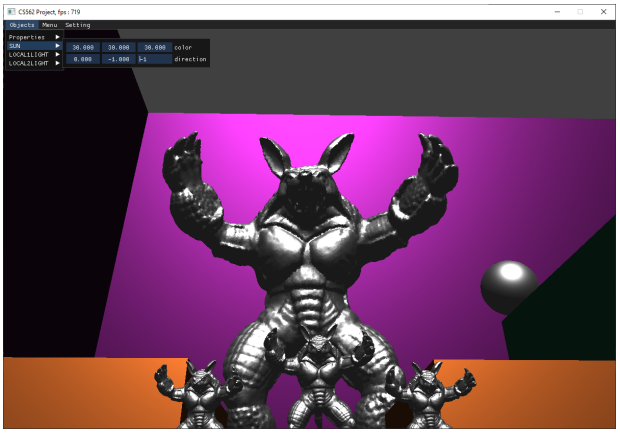
The original way of drawing an object was directly drawing it into the surface once. Otherwise, in the deferred rendering, the drawing sequence needs two steps. The one is drawing on the g-buffer, and the other is restoring the g-buffer and drawing on the surface. The drawing g-buffer step does not involve lighting calculation. Only the drawing surface passes do lighting calculations.
There is a graphics pipeline (called g-buffer pipeline) in the drawing g-buffer step. The vertex shader of the g-buffer pipeline transforms positions and normals to NDC space from local object space. The pixel shader stores the g-buffer data sets. The data will use in lighting calculation in the following pipelines. To define the g-buffer, it stores the position, normal, texture coordinate, metallic, roughness, albedo values on separate images. One of the noticeable disadvantages of deferred rendering is that it takes too much space for storing the g-buffer. To relieve the overtaking memory, I established my own data sets for the g-buffer. Since we need accurate precision on positions, normals, and texture coordinates, I set the size of those images as 64 bits (16 bits for each component). Also, I store texture coordinates and object properties on one image for saving memory. On the other hand, I store it on 32 bits(8 bits for each component) for the albedo data because color only handles a range of 0-255 values.
In the swap chain parts, there are three main pipelines in total. The lighting pipeline is the pipeline that will occur after the g-buffer passes. It disabled depth testing because it is unnecessary. The vertex shader in the lighting pipeline passes the full-sized quad vertices and texture coordinates to the pixel shader. Pixel shader calculates the lighting calculation of the sun(global light) and performs the global ambient light. The next pipeline handles the local lights. It blends the original color with the lighting color of local lights. The vertex shader draws the sphere that represents the range of local lights. The pixel shader restores the g-buffer value and compares the position value whether the position is in the light range or not. If the pixel position is in the light range, then the shader calculates the pixel light. The calculated light blends with the previous image. The last pipeline draws the local lights. It uses the depth value that is used in the g-buffer part. It re-enables depth testing. It draws the local lights with the color of light.

Result Images

some sample images







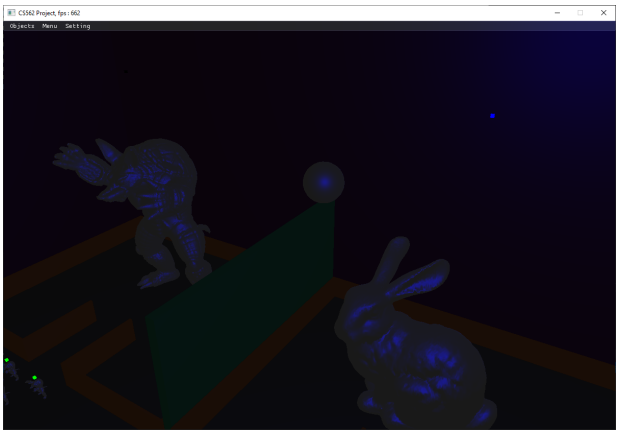
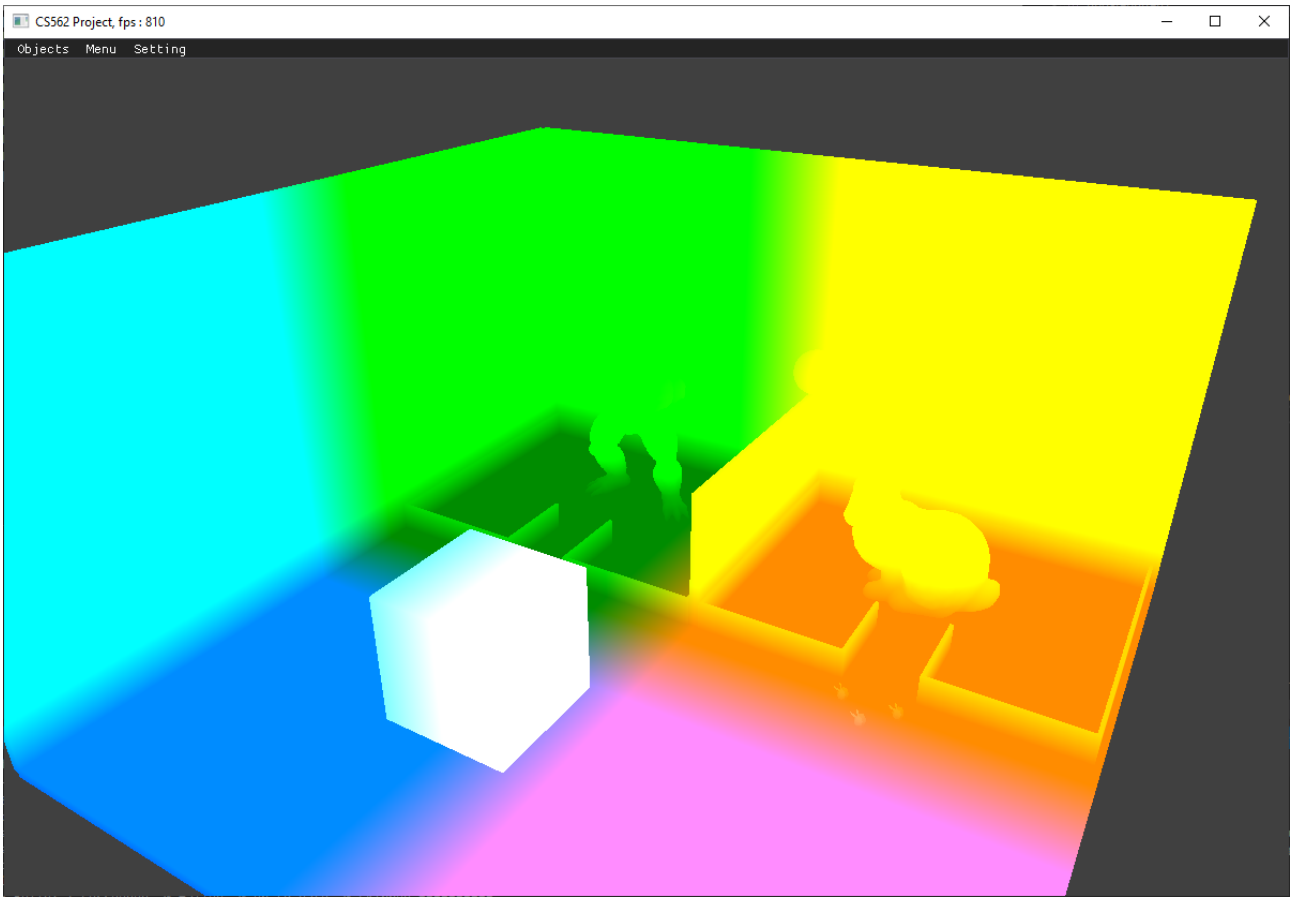
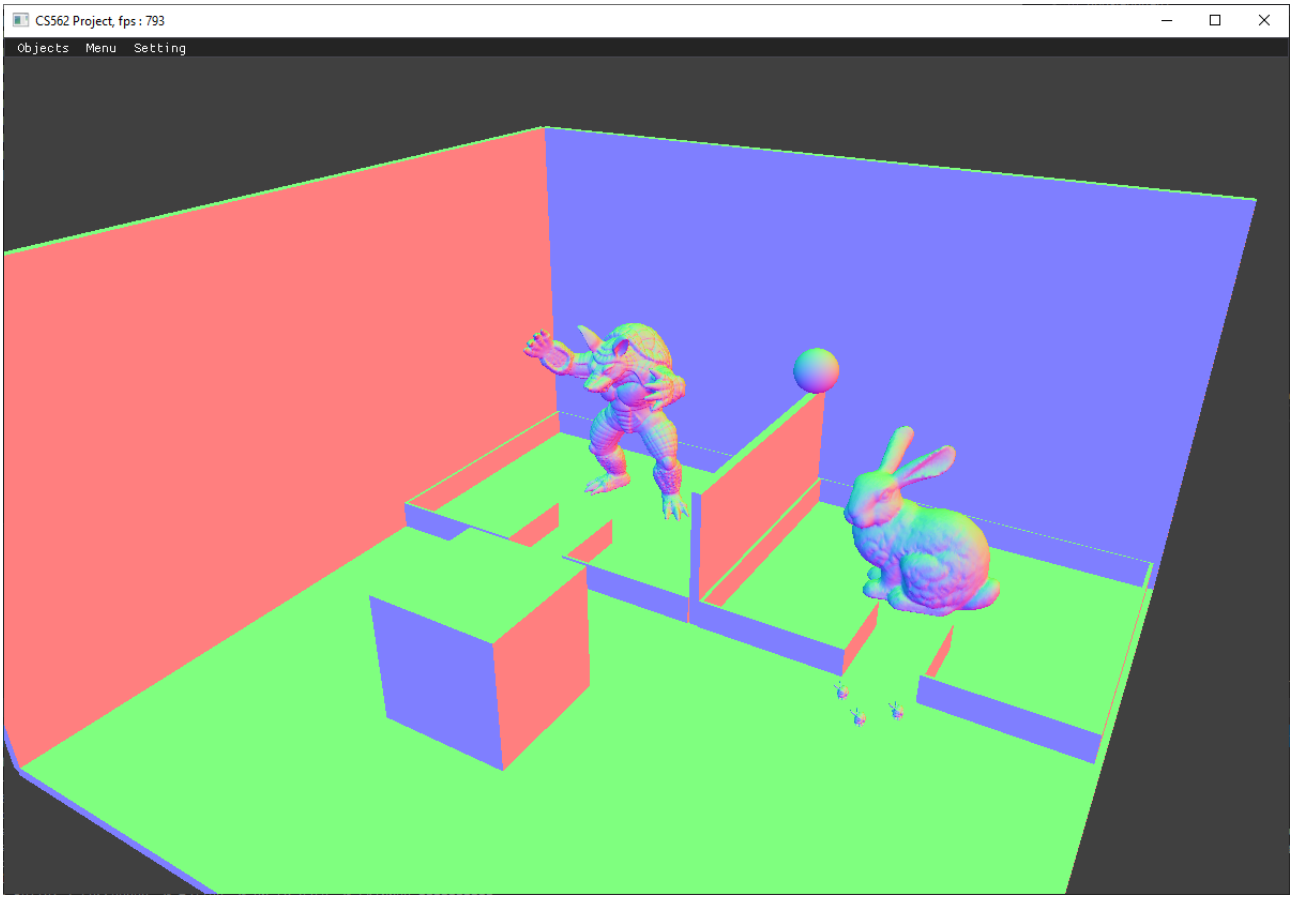


Image with deferred rendering target

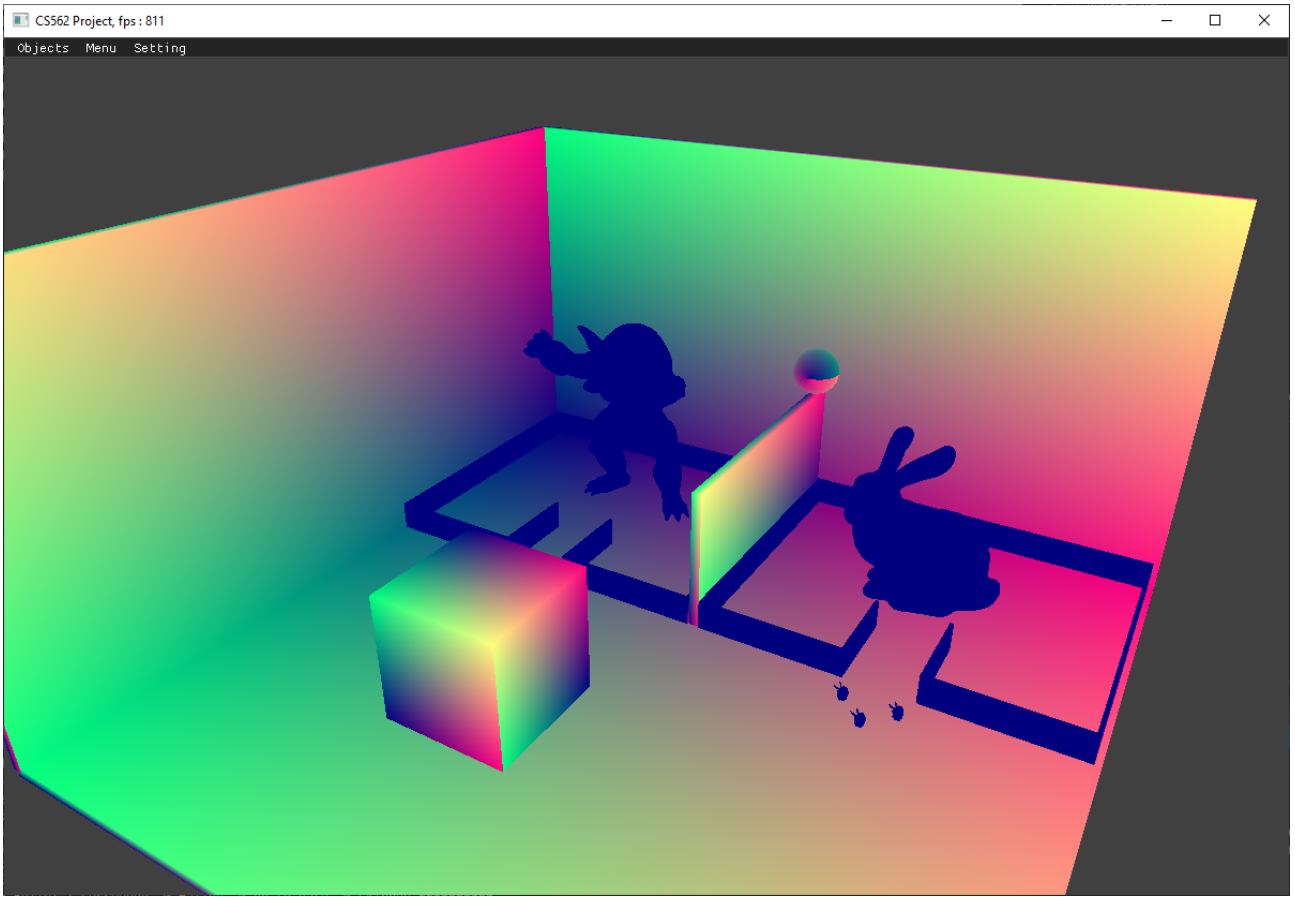
(position + (2,1,6)) / 2 (for better visibe color range)



(normal + (1,1,1)) / 2 (for better visibe color range)



texture coordinate (object files not included texture coordinate)



albedo color

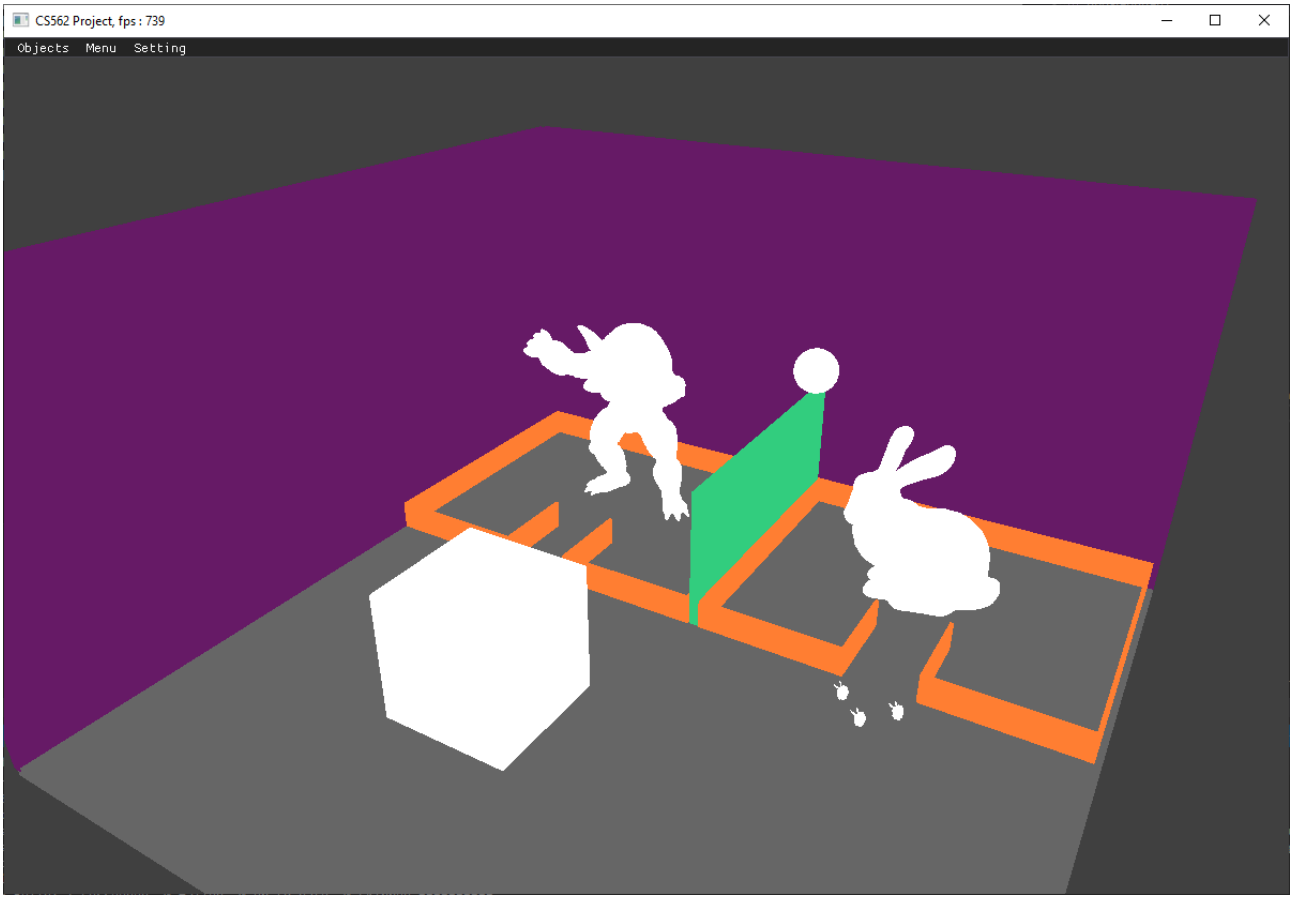


Image with no local light

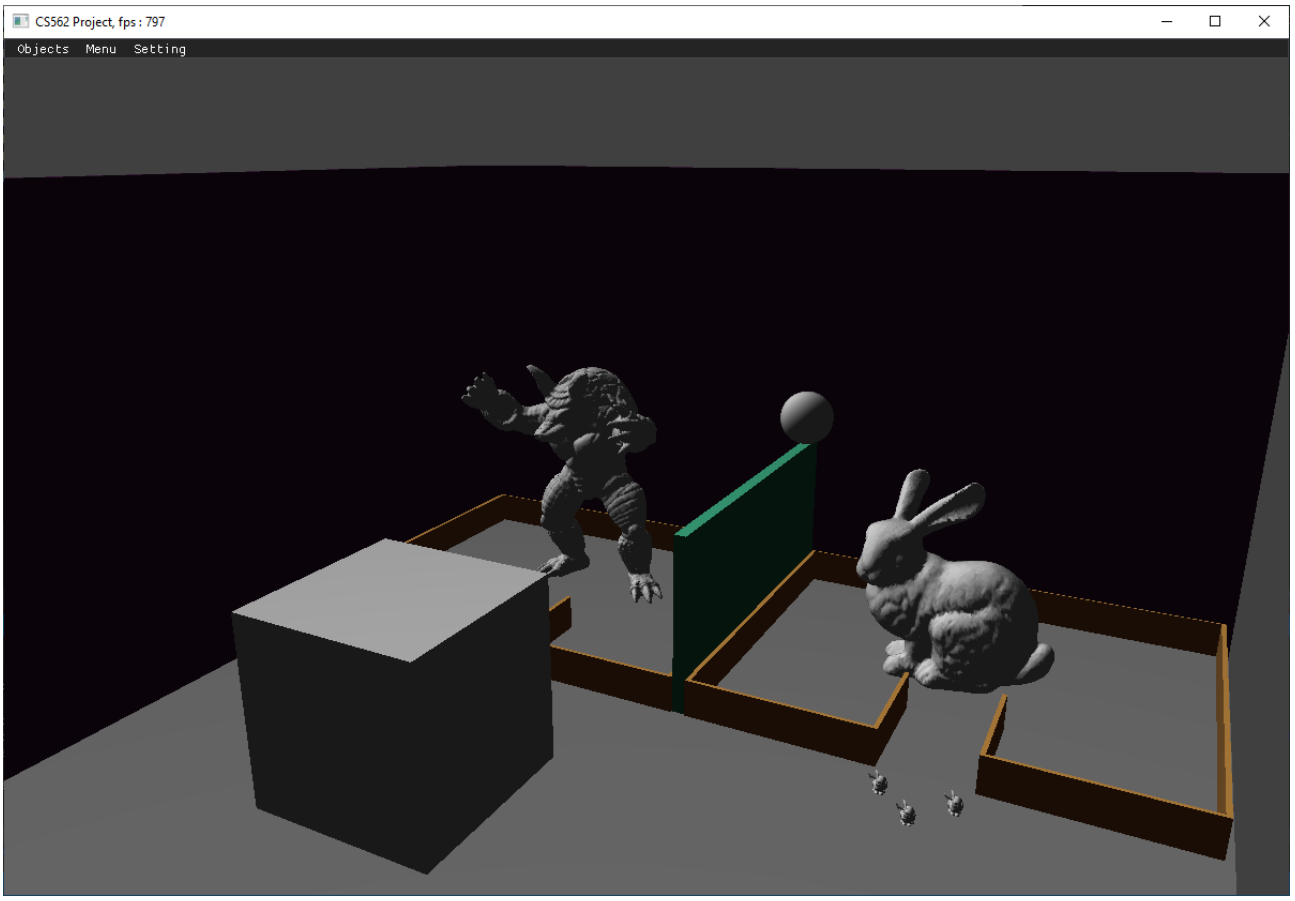
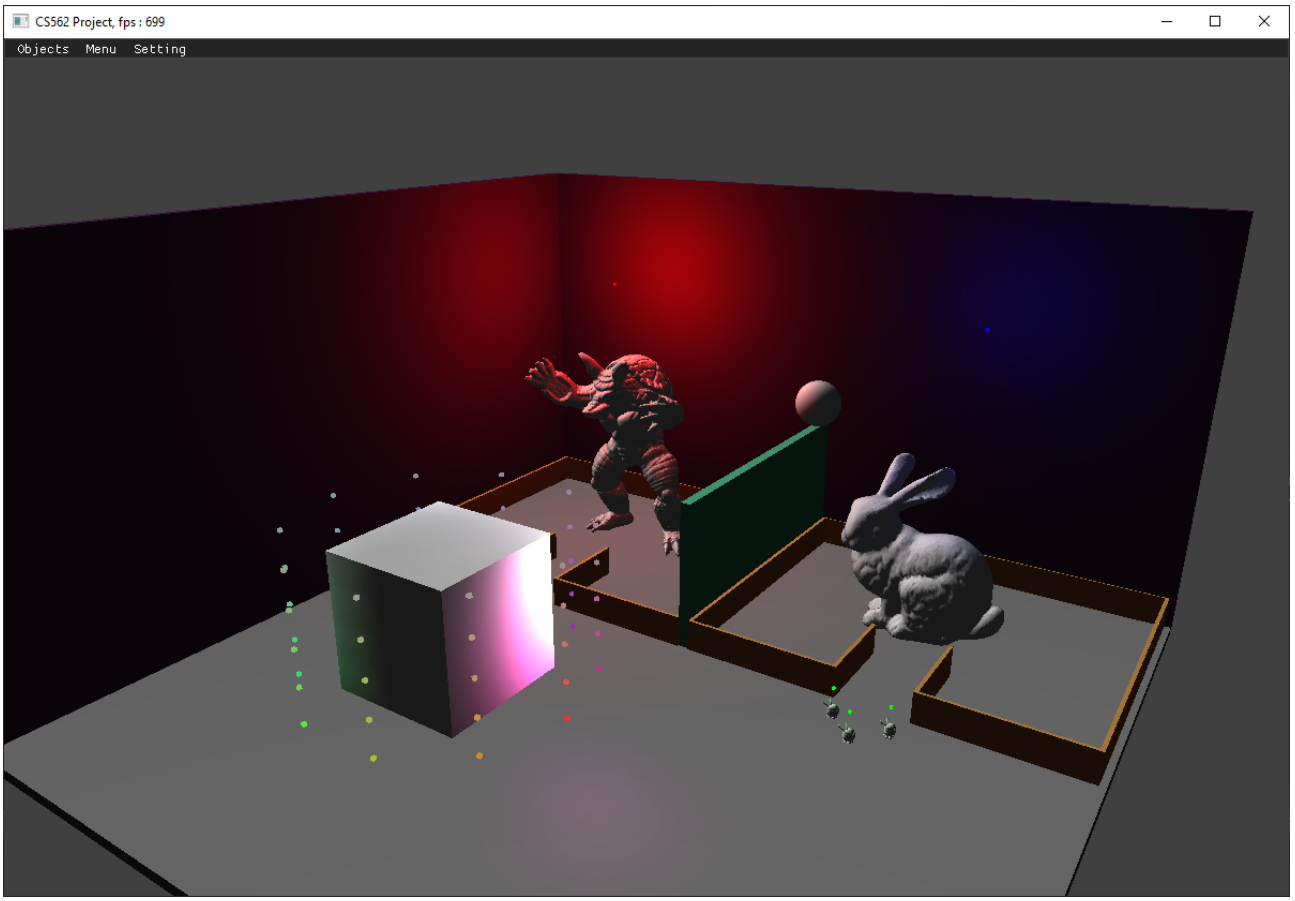
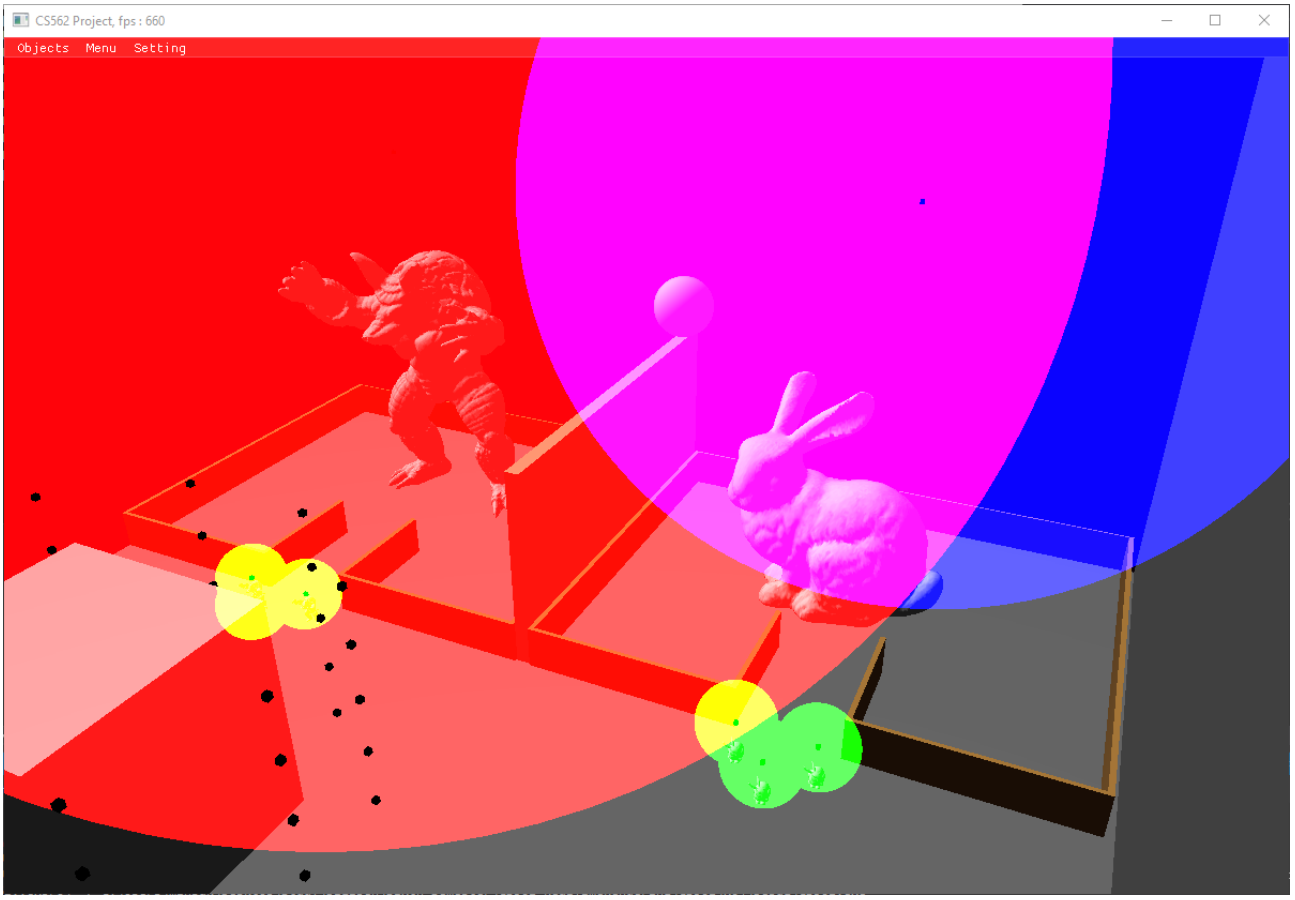


Image with local light

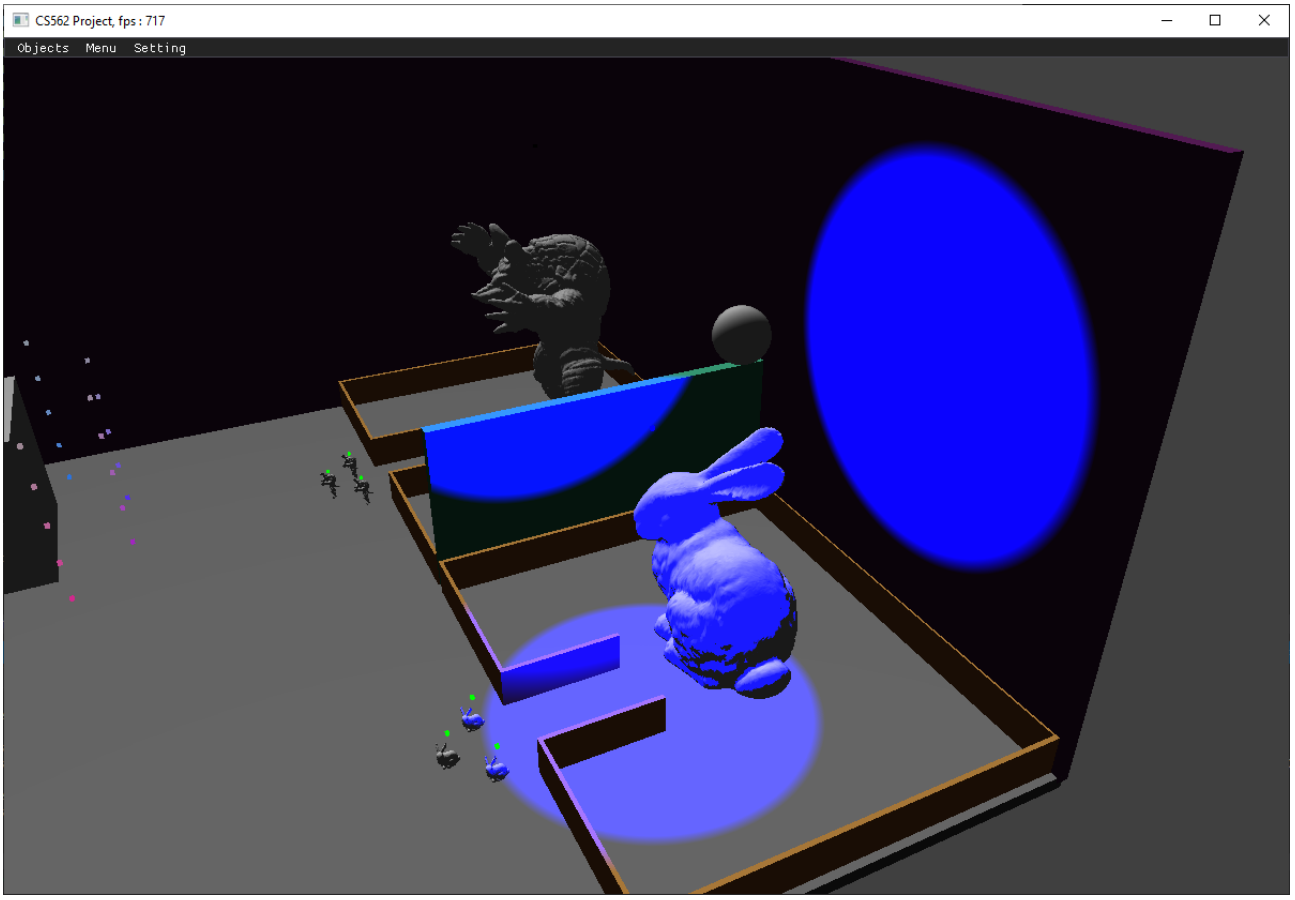
image with local light



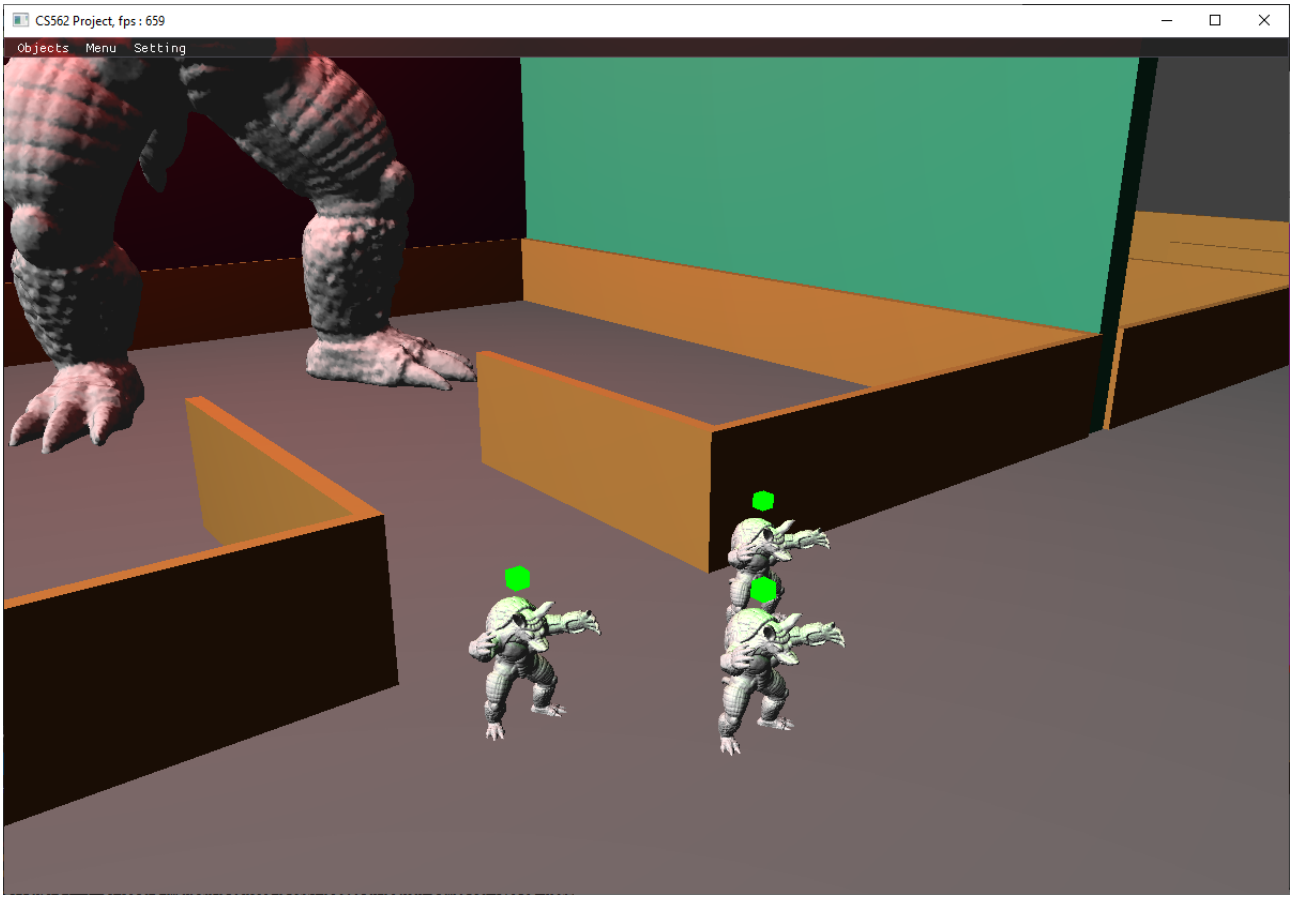
light space visible



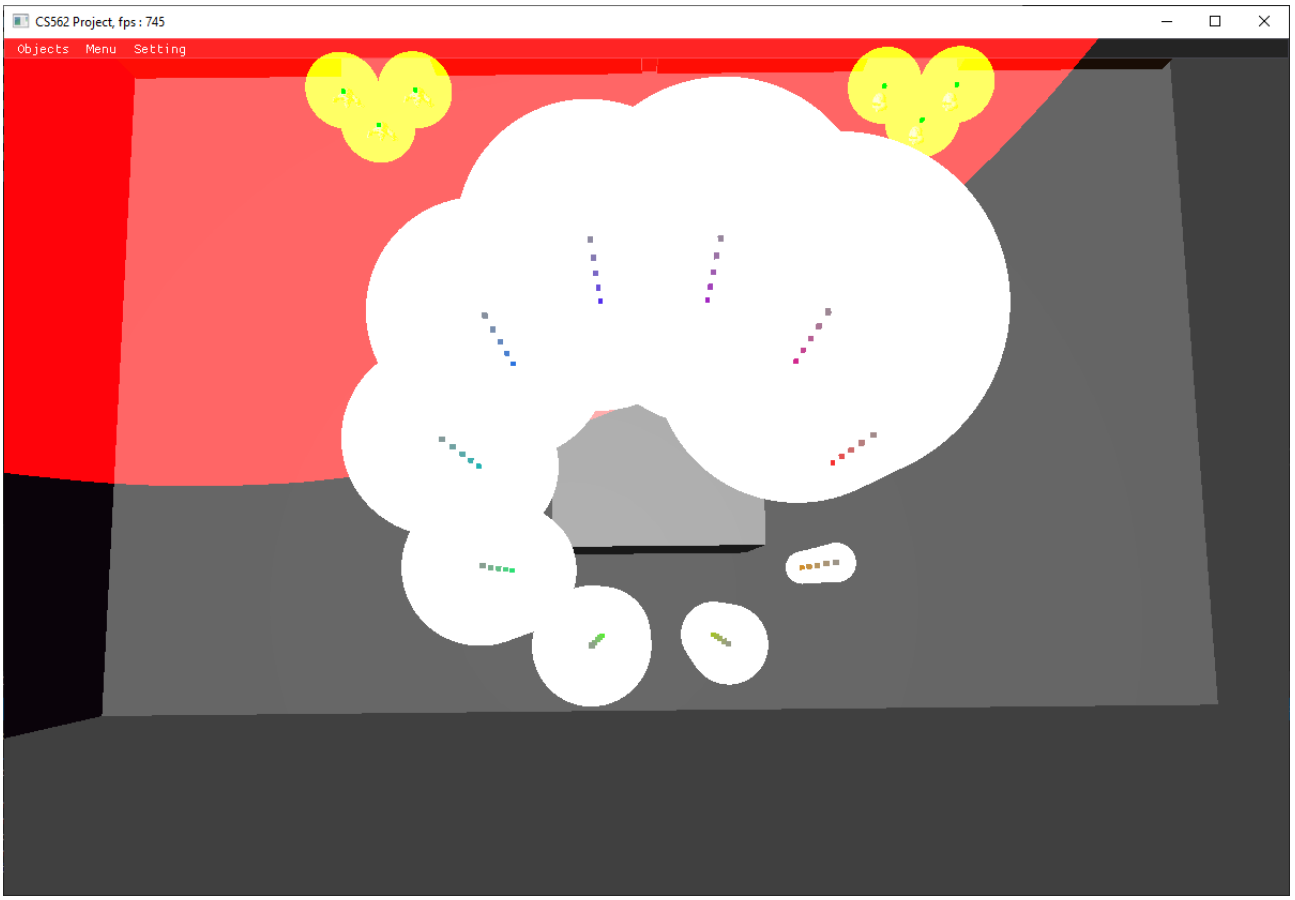
the object only be lit when it overlaps with light space
(the light intensity goes 5000 and radius goes 5)



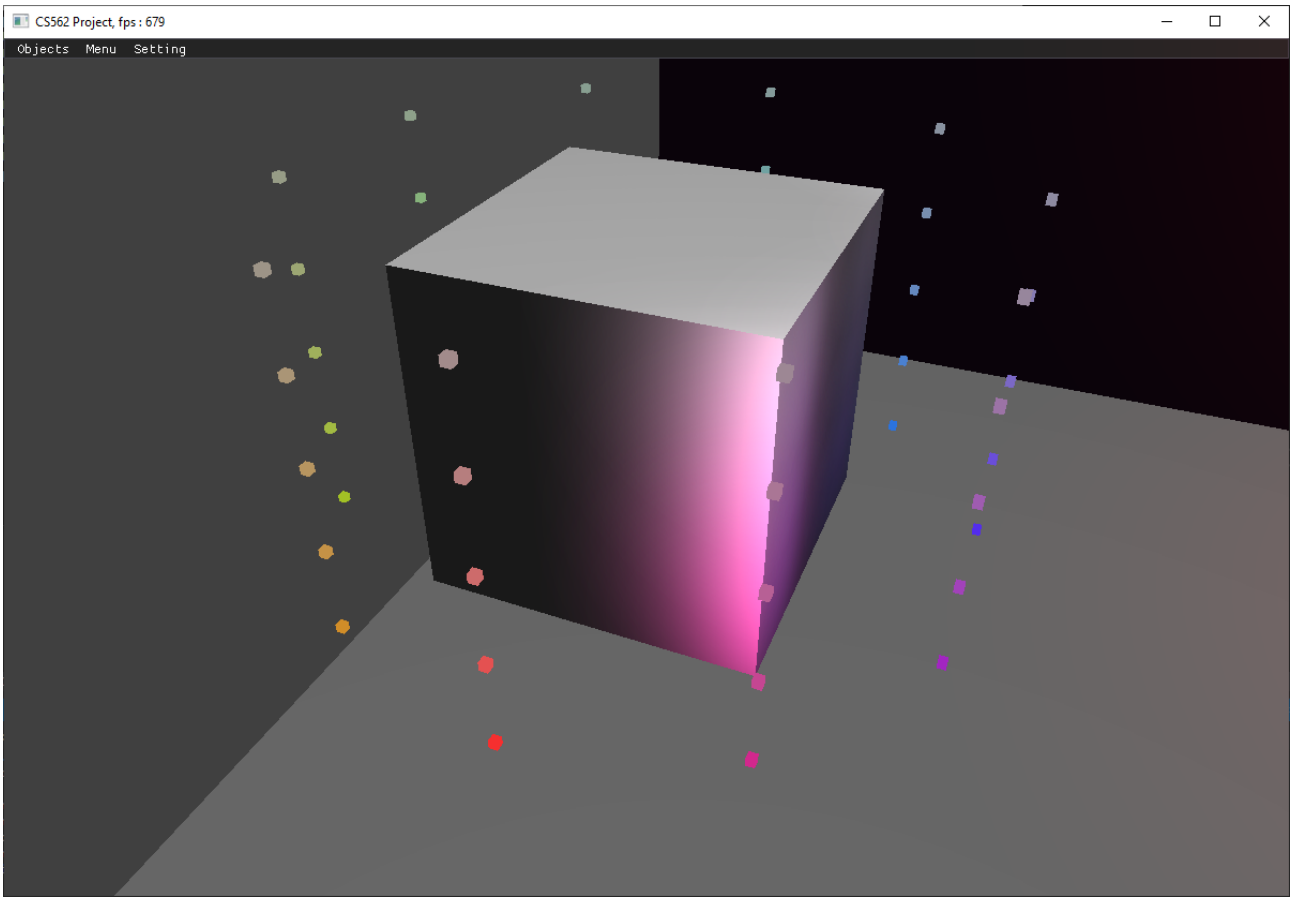
small size local light (space visible, invisible)
color is blended. the color of light space of small range light is yellow because red + green



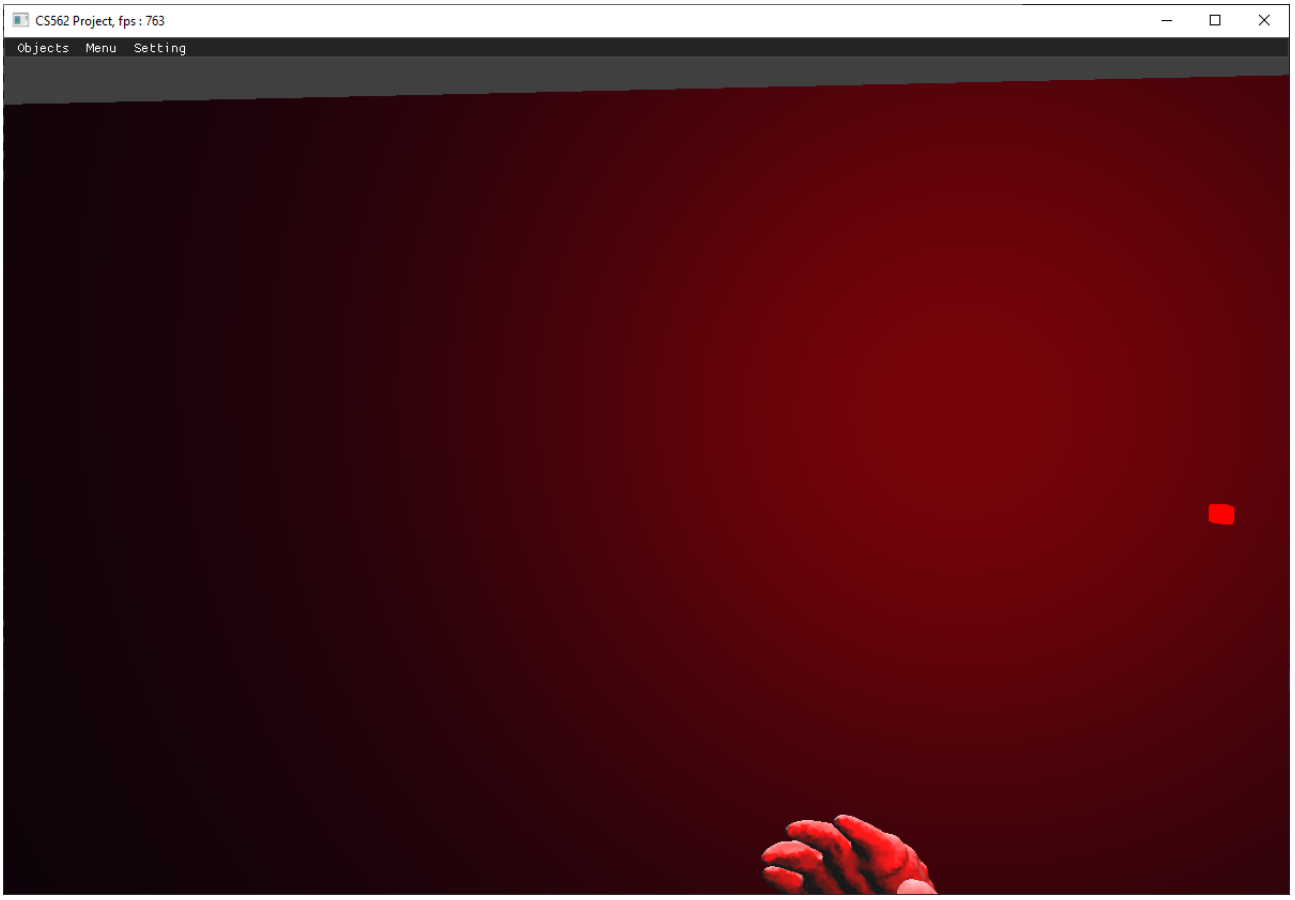
local lights that has various size from the angle



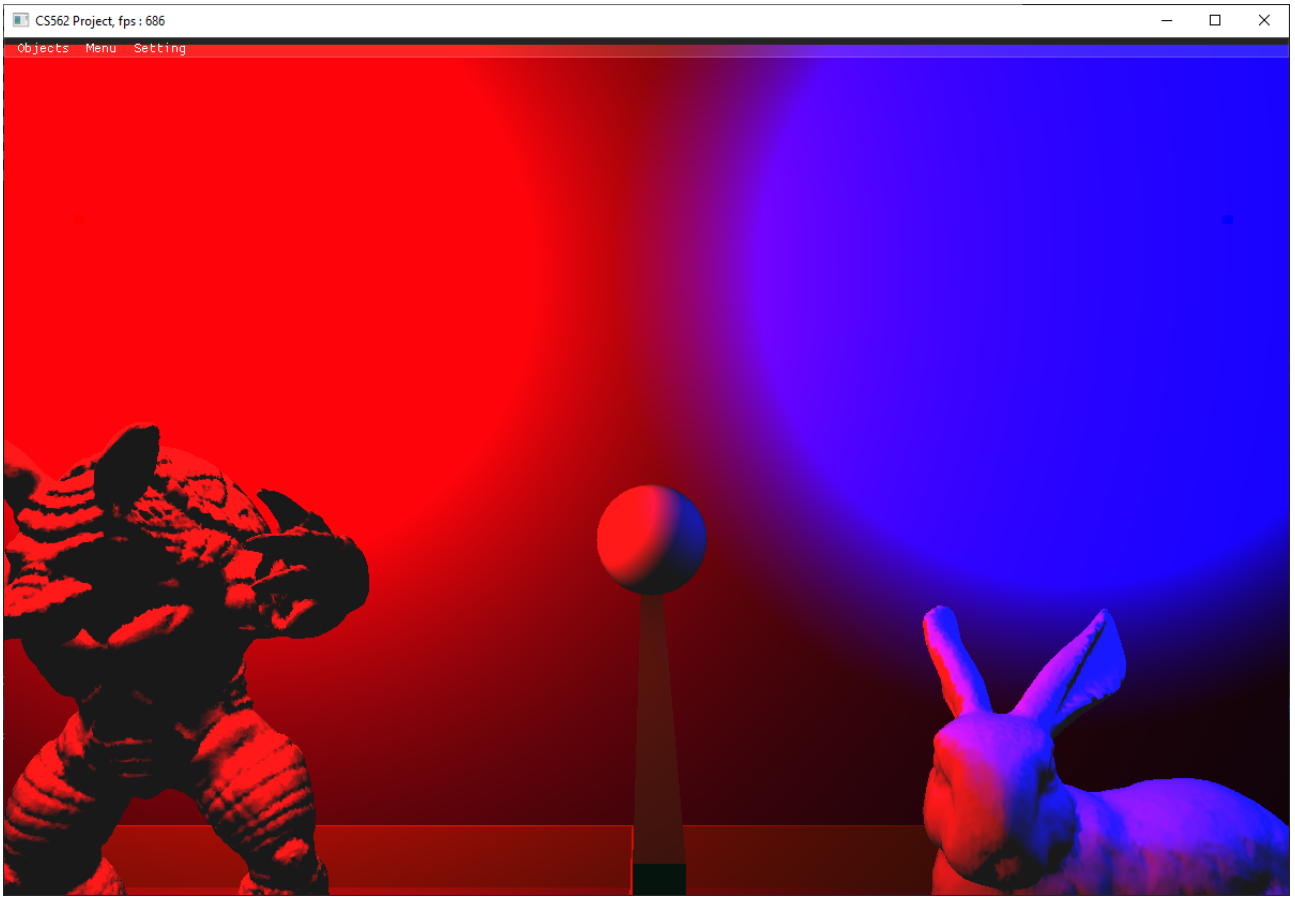
how lights range effect on the object (for the left side of the cube, the light range is small, so there is no light on the cube)



light attenuation with range (lights goes 0 on the maximum range (radius of light space)
light color is faded out on the left side



this image shows how the range of light space determines the light color. right light has 10 times more intensity than left light but the light color on the sphere goes red because of light attenuation.
right light (intensity : 500, range 5), left light (intensity : 50, range 10)



Notes

BRDF

I used the BRDF(bidirectional reflectance) Cook-Torrance model for my lighting. Here is the function of my BRDF function (where V is view vector and L is a light vector, F0 is base reflectivity)

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

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

$$k = \frac{\alpha^2}{2},\ G = \frac{(V \cdot N)}{(V \cdot N) \cdot (1-k) + k} \cdot \frac{(L \cdot N)}{(L \cdot N) \cdot (1-k) + k}$$

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

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

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

This is implemented on light.glsl that includes files for other shader files that calculate the lighting.
image of the model with metallic 1 and roughness 0.2

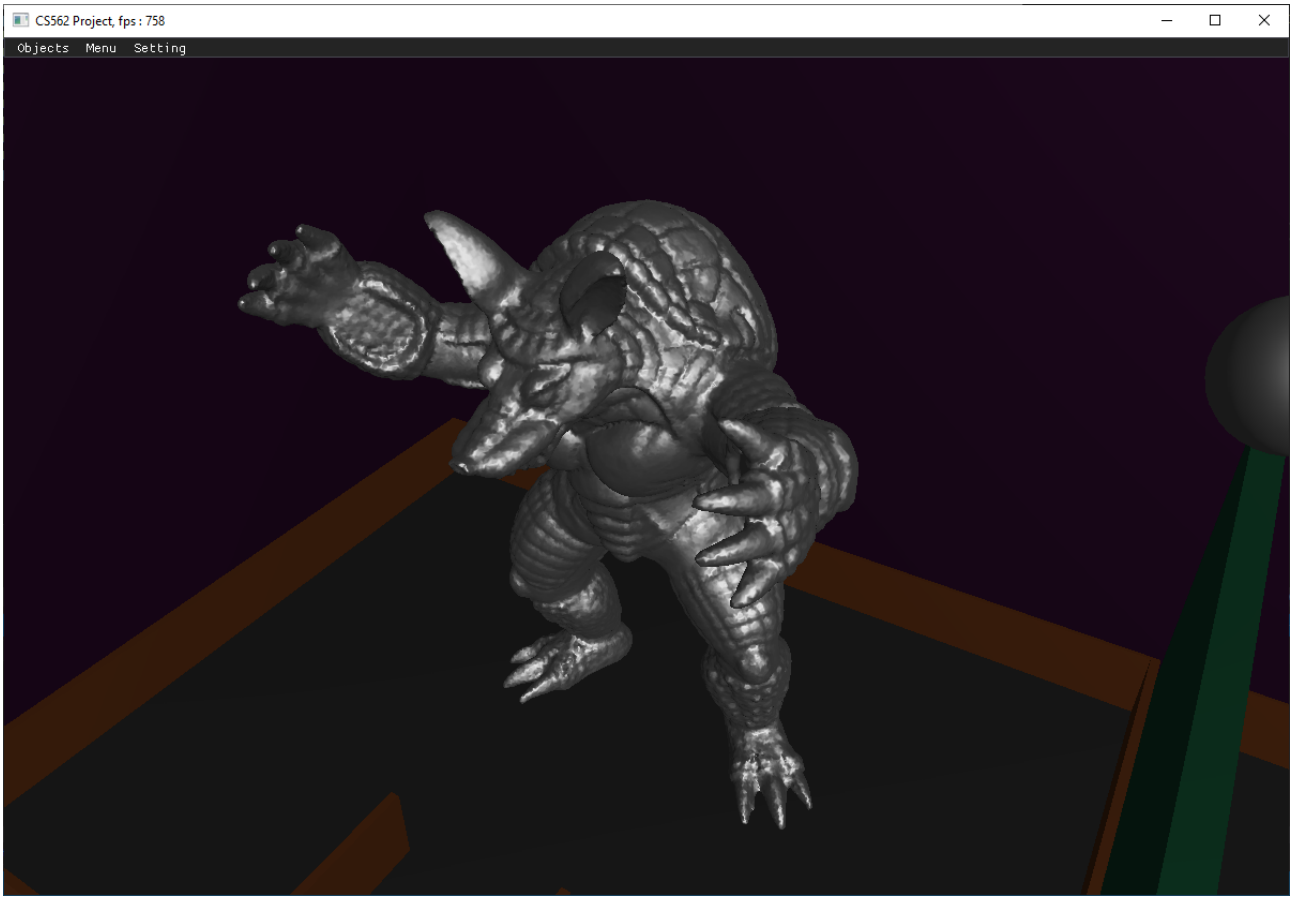
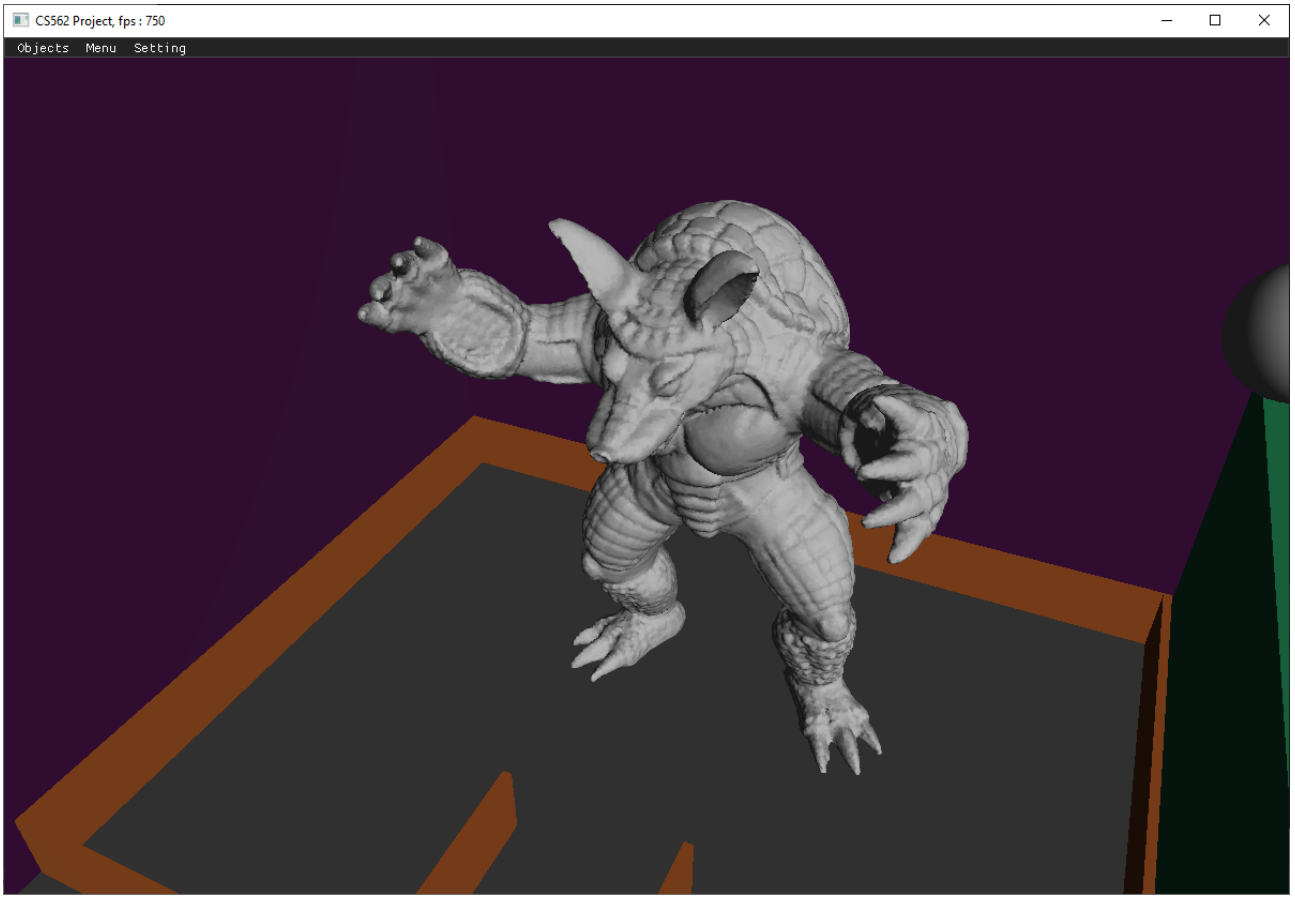


image of the model with metallic 1 roughness 1

image of the model with metallic 0

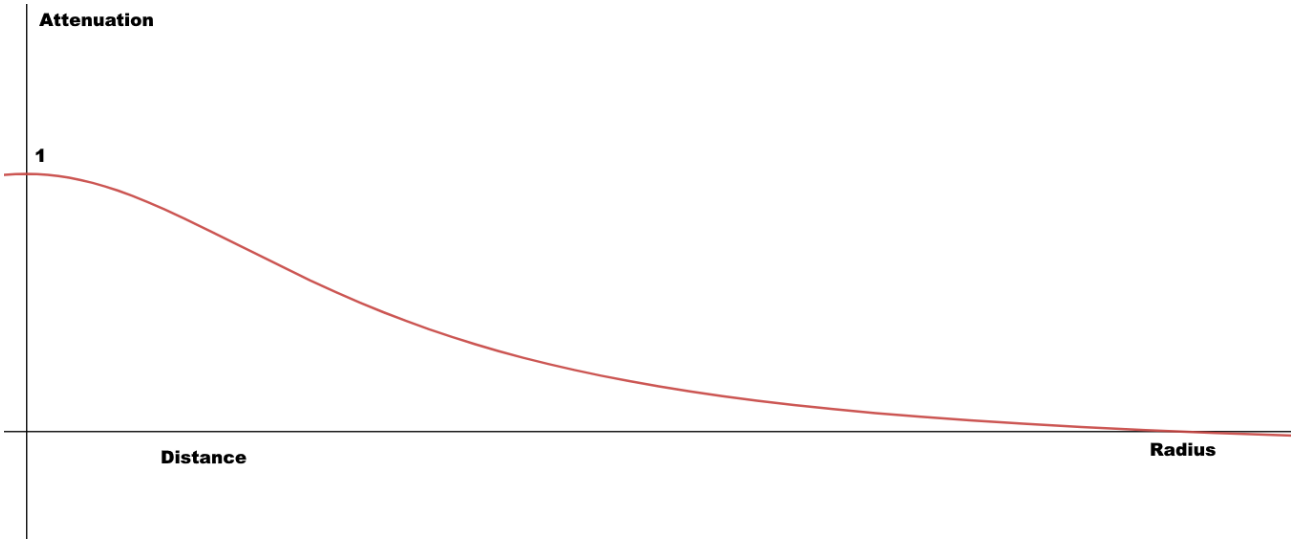


Attenuation

I set the attenuation of light as the function when d is light distance and r is light space radius

$$attenuation = \frac{r^2 - d^2}{C \cdot d^2 + r^2} \, (C \; is \; slope \; constant)$$

that makes attenuation goes 0 when the light distance goes maximum light space range.



This is implemented on locallights.frag

Depth conservation

After finishing all lighting calculations(global and local), I draw the light object that will not be affected by light. Since the first frame buffer(g-buffer pipeline) stores the depth values, I need to conserve the depth texture. I re-use the depth images from the previous render pass. In the Vulkan API, there is a loadOp variable on VkAttachmentDescription that defines whether to clear the frame buffer data or not. I disabled the second render pass so that It did not clear the depth value when entering the second render pass.
This is implemented on render.cpp 211 lines

Sphere Collision

I determine whether the pixel of an object is in the light space or not using a sphere collision. I pass the projection matrix, view space transformation matrix, and model space transformation matrix. Vertex shader transforms the sphere mesh position and sends it to the pixel shader. The pixel shader calculates the relative texture coordinate on the window screen using the gl_FragCoord.xy / vec2(width, height). From the texture coordinate, I get the front-most pixel data. It compares the length between position and sphere center with sphere radius. If the sphere radius is smaller than the length between position and sphere center, the pixel is not in the light space. That means the pixel should not be affected by the local light.
This is implemented on locallights.frag 57-62 lines

Debugging

The user can look around the scene with a camera object while pressing the right mouse button. The user can move forward by pressing W, S and move right by pressing A, D. There is a menu bar in the scene where the user can adjust some variables such as albedo, metallic, roughness of object, light properties of global light and two local lights, and render target mode selection. GUI also told the user about how many times it takes for doing a draw operation.