Project 4: Ambient Occlusion

Name: Minsuk Kim(m.kim)
Instructor: Dr. Gary Herron
Class: CS562
Semester: Spring 2022
Date: 4/2/2022

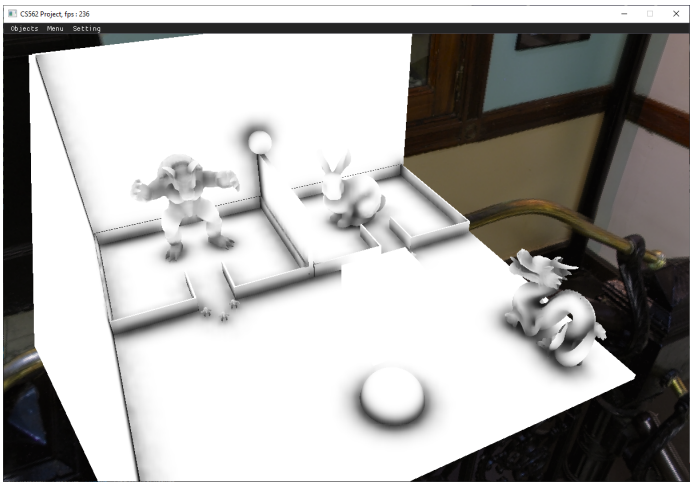


Table of Contents

Introduction

Overview

The main purpose of the project is to implement ambient occlusion. The ambient occlusion produces illumination effects like darkened corners, cracks and wrinkles. It investigates the screen space to measure the obscurance of the scene to get the ambient occlusion. It reads the depth and normal buffers to estimate the obscurance.

In this project, I will cover the Alchemy Screen-Space ambient occlusion algorithm. The program will have three passes that involve this project: ambient occlusion mapping pass, blurring ambient occlusion map pass and lighting calculation pass. The ambient occlusion mapping pass reads the screen-space factor to calculate the ambient occlusion factor and maps it into ambient occlusion texture. The blurring ambient occlusion map blurs the ambient occlusion texture with a bilateral blur filter. The lighting calculation pass uses the final ambient occlusion value to final lighting results.

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

Implementation

The project will use the alchemy algorithm for the ambient occlusion calculation. The rendering sequence for this project will be the following: generating an ambient occlusion map, blurring(filtering) the ambient occlusion map and using ambient occlusion factor to apply it in the final lighting result.

First, the program will generate an ambient occlusion map from the gbuffer. The program reads position and normal from the scene to calculate the ambient occlusion for the scene. Then the ambient occlusion value will be stored in the texture. For the performance, the program uses distributed samples when calculating the ambient occlusion value.

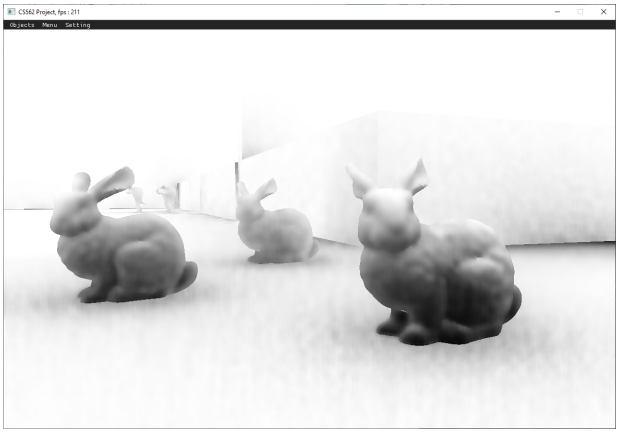
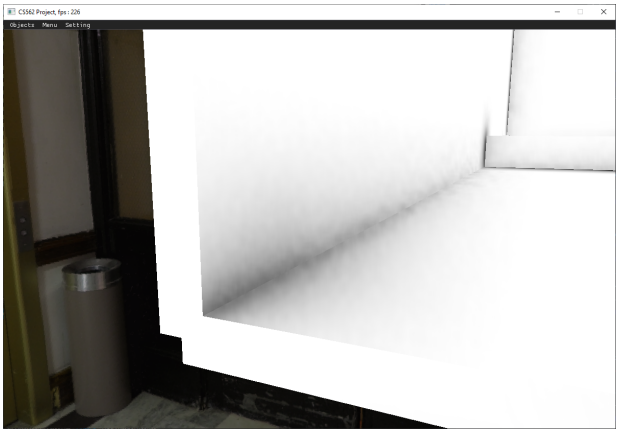
The next step is filtering the ambient occlusion map. The program uses compute shaders for calculating blur-filtered ambient occlusion map. The program uses threaded group shared memory to calculate the vertical bilateral filter of ambient occlusion and horizontal as well. To filter the texture with a bilateral method, the program needs to read the depth and normal texture from the g-buffer.

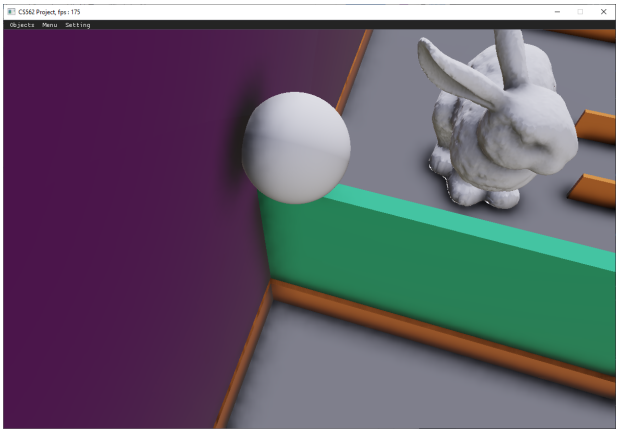
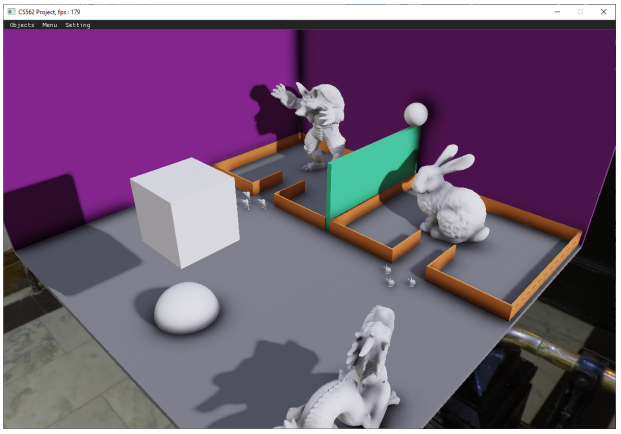
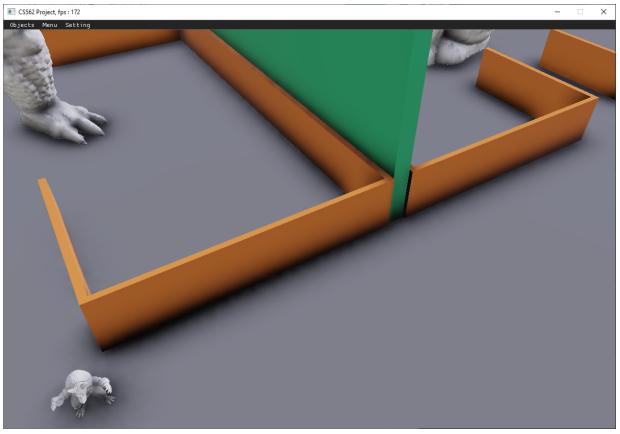
The final step is applying the ambient occlusion on the final lighting result. The ambient occlusion will multiply on ambient light. It cannot apply to any direct light such as specular or diffuse light.

Result Images

some sample images

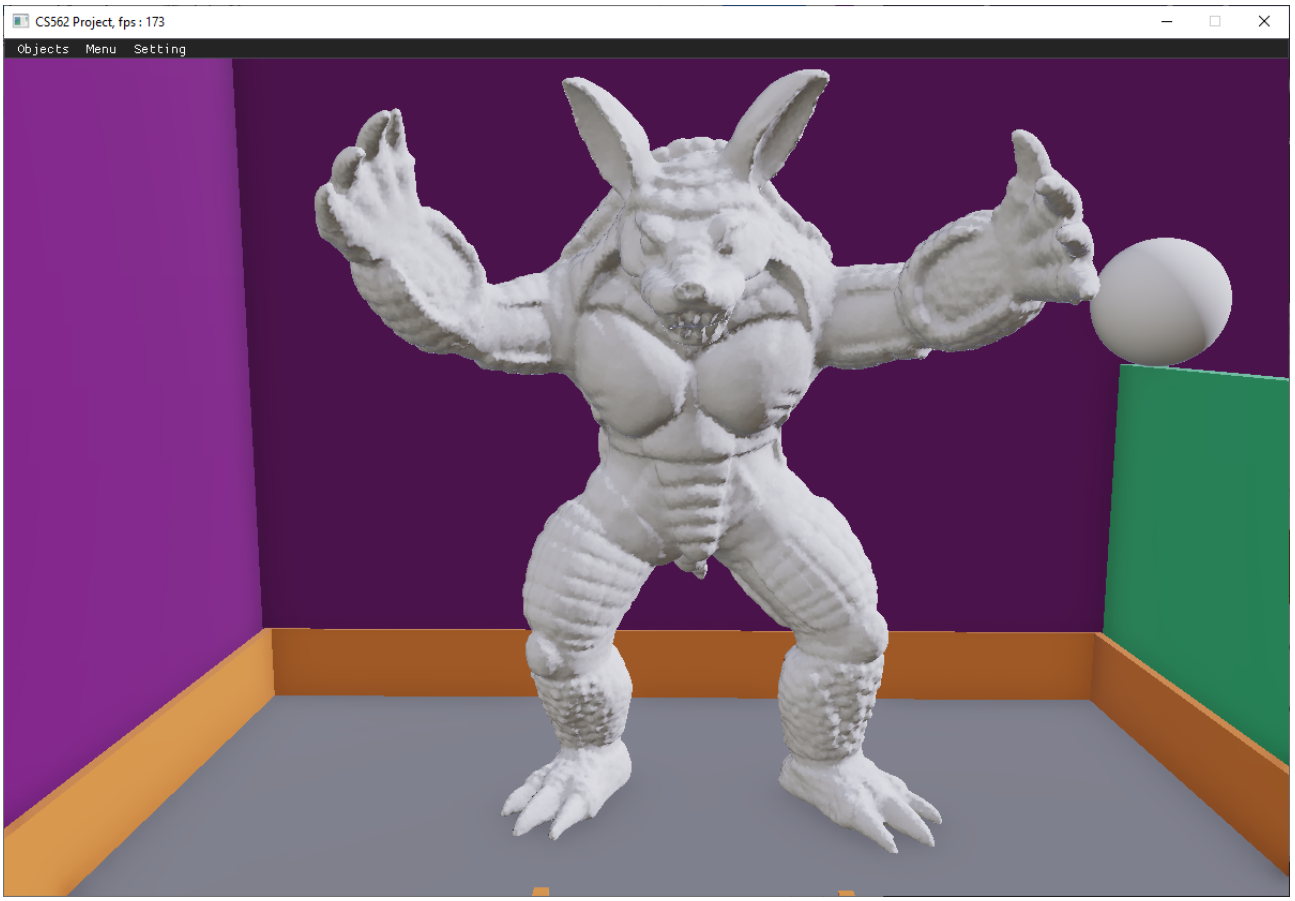


Images with various ambient occlusion constant

s=1,k=1,r=1



s=1,k=10,r=1

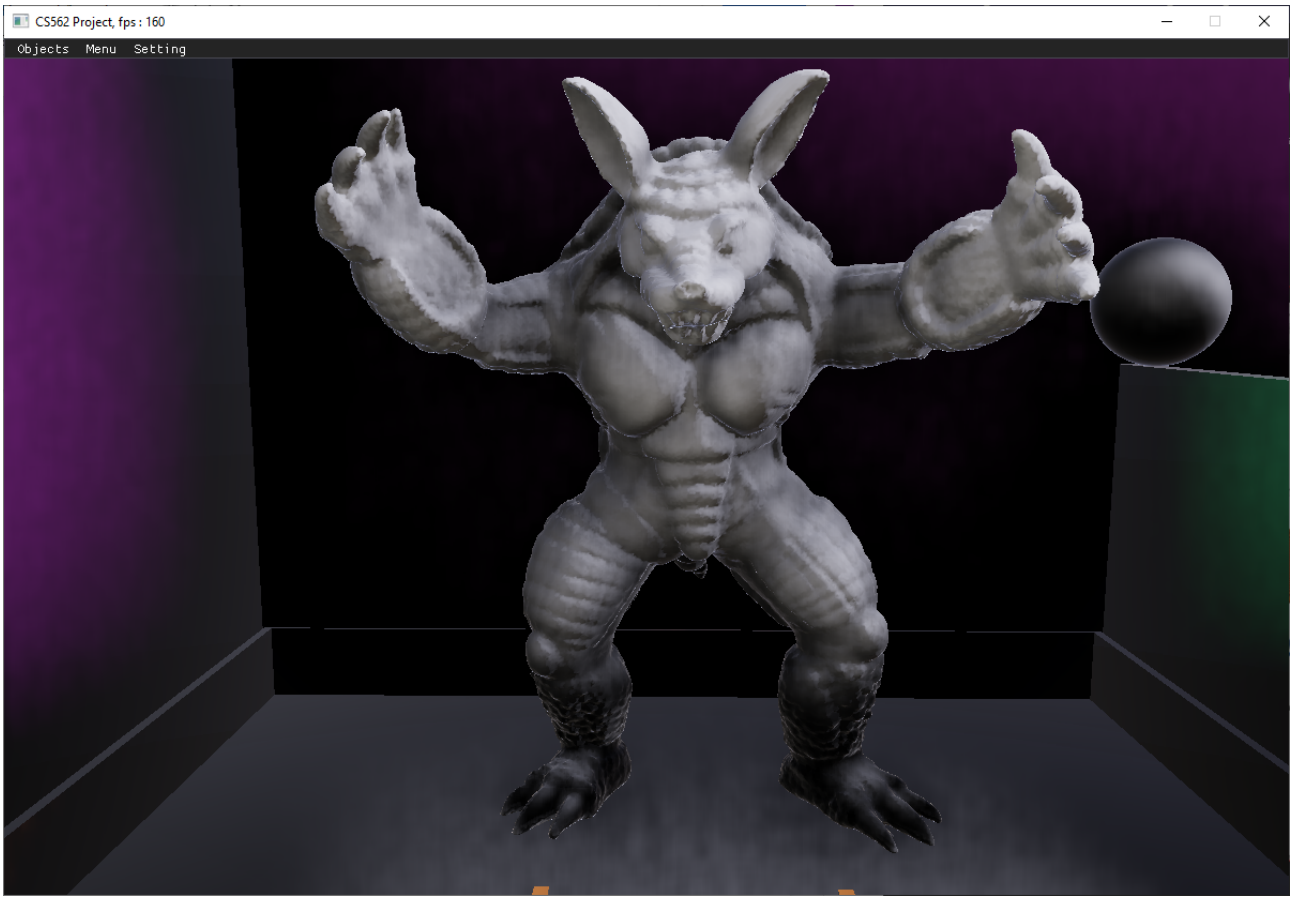
s=10,k=1,r=1

s=3,k=3,r=1

s=10,k=10,r=1

s=3,k=3,r=1.5

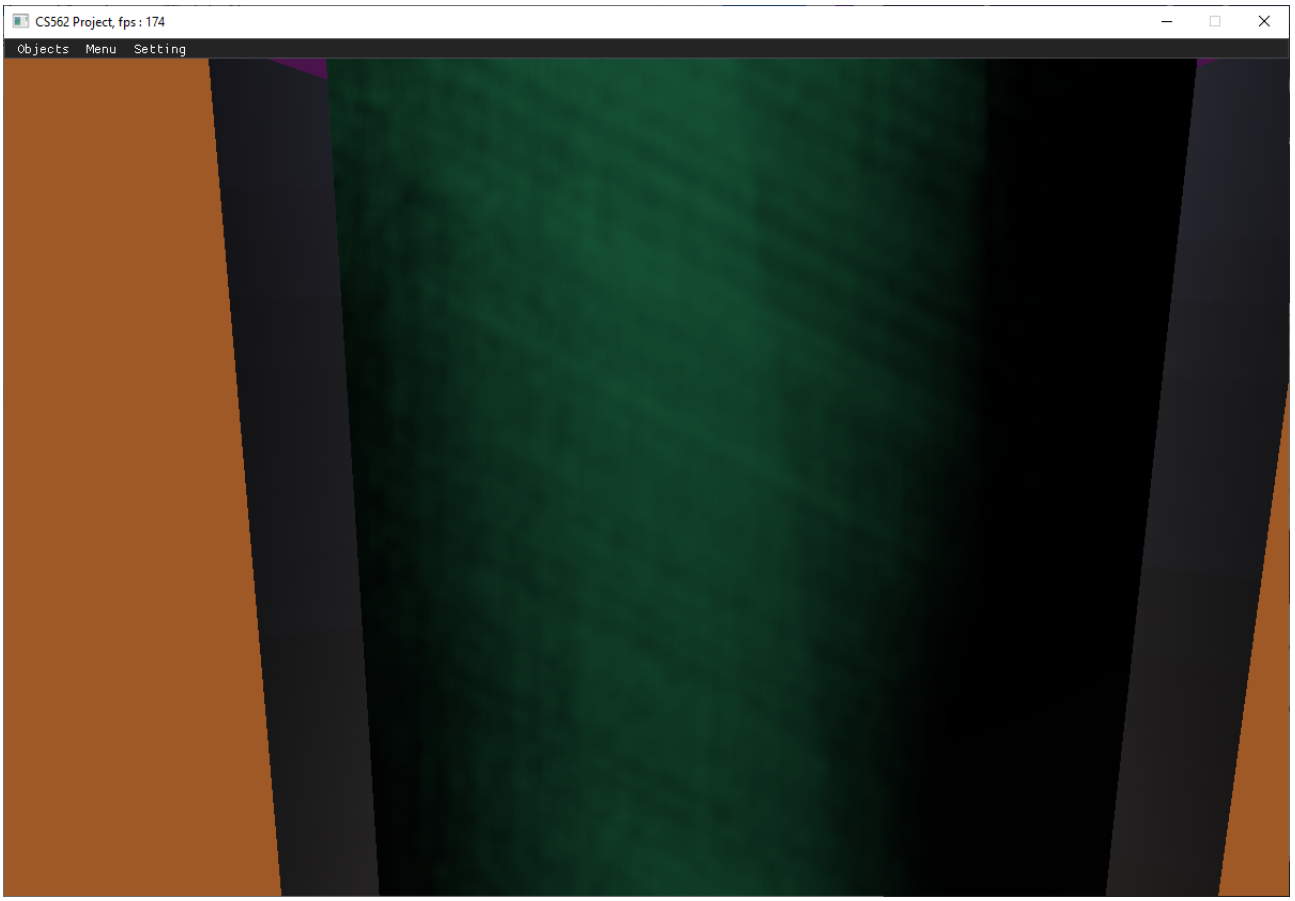
s=3,k=3,r=5



Images with error from the algorithm

on the right-bottom corner, the ambient occlusion does not apply for the area that camera does not contain (because it only calculates the screen-space)

because of the screen-space, the ambient occlusion will not apply properly when camera is too close to the object



Notes

Uniform Distribution

In this project, the program used the uniform distribution when the program calculates the ambient occlusion. The reason to use such samples is because of the performance. It needs fewer samples but produces accurate results. The uniform distributed vector will be calculated with:

$$\phi = (30 \cdot x \wedge y) + 10xy$$

, where x and y are fragment coordinate

$$\alpha = (i + 0.5)/n$$
$$h = \alpha R/d$$
$$\theta = 2\pi\alpha(7n/9) + \phi$$
$$vector = h \cdot (cos\theta, sin\theta)$$

, where n is the number of samples and i is the variable from 0 to n-1. The R is the range of influence of ambient occlusion. d is scene depth of the current position.

Ambient Occlusion

The ambient occlusion is calculated with:

$$S = \frac{2\pi c}{n}\sum_{i=1}^{n}\frac{max(0,N\cdot w_i - \delta d_i)\cdot H(R-|w_i|)}{max(c^2, w_i\cdot w_i)}$$
$$A = (1 - sS)_+^k$$
$$w_i = P_i - P$$

, where P_i is the position in texture coordinate + uniformed distributed vector. N is the normal vector and n is number of samples and δ is around 0.001. c is 0.1 times R and $H(R - |w_i|)$ is the Heaviside step function that returns 0 for negative values and returns 1 for not-negative values. The s and k are ambient occlusion variables that can be adjusted.

Bilateral blur

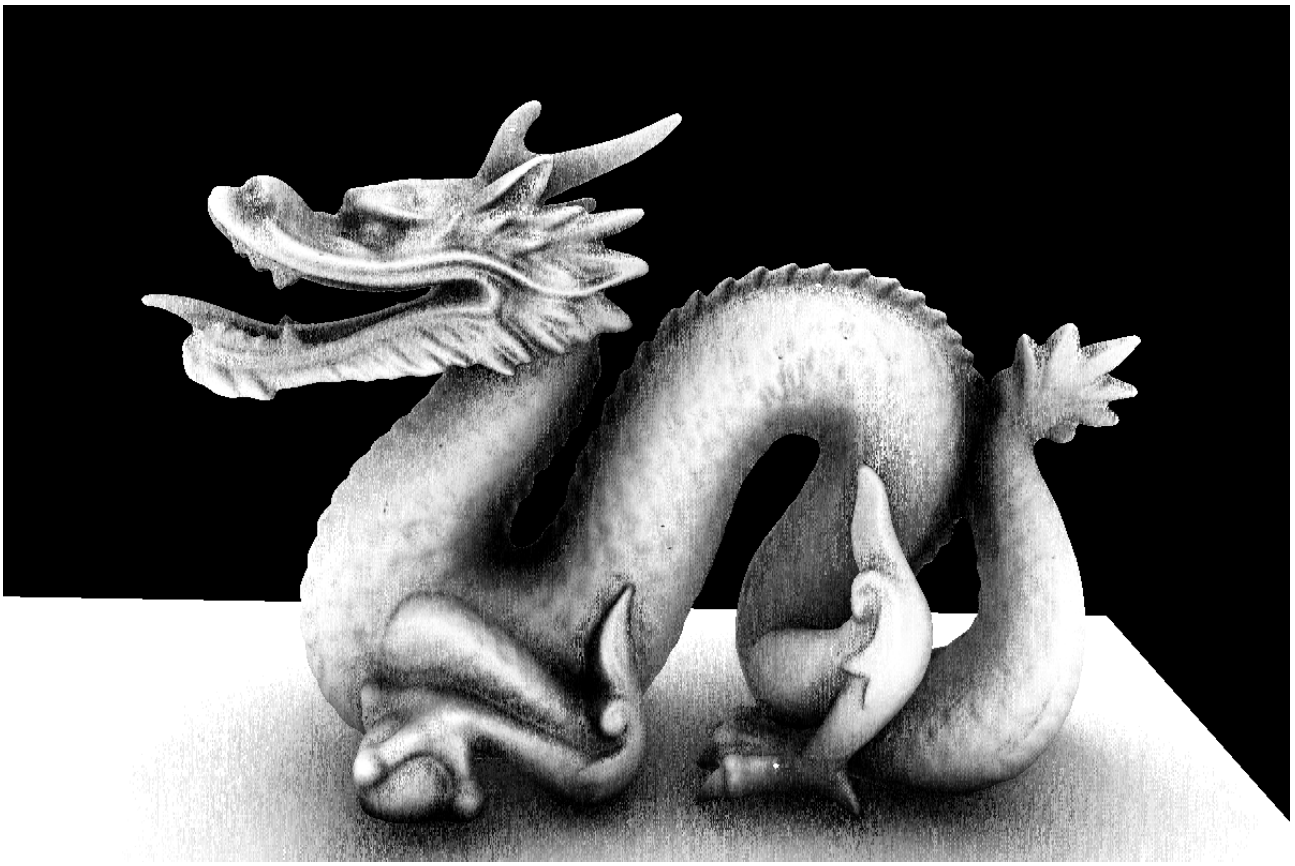
The program filtered the ambient occlusion map using the bilateral blur.

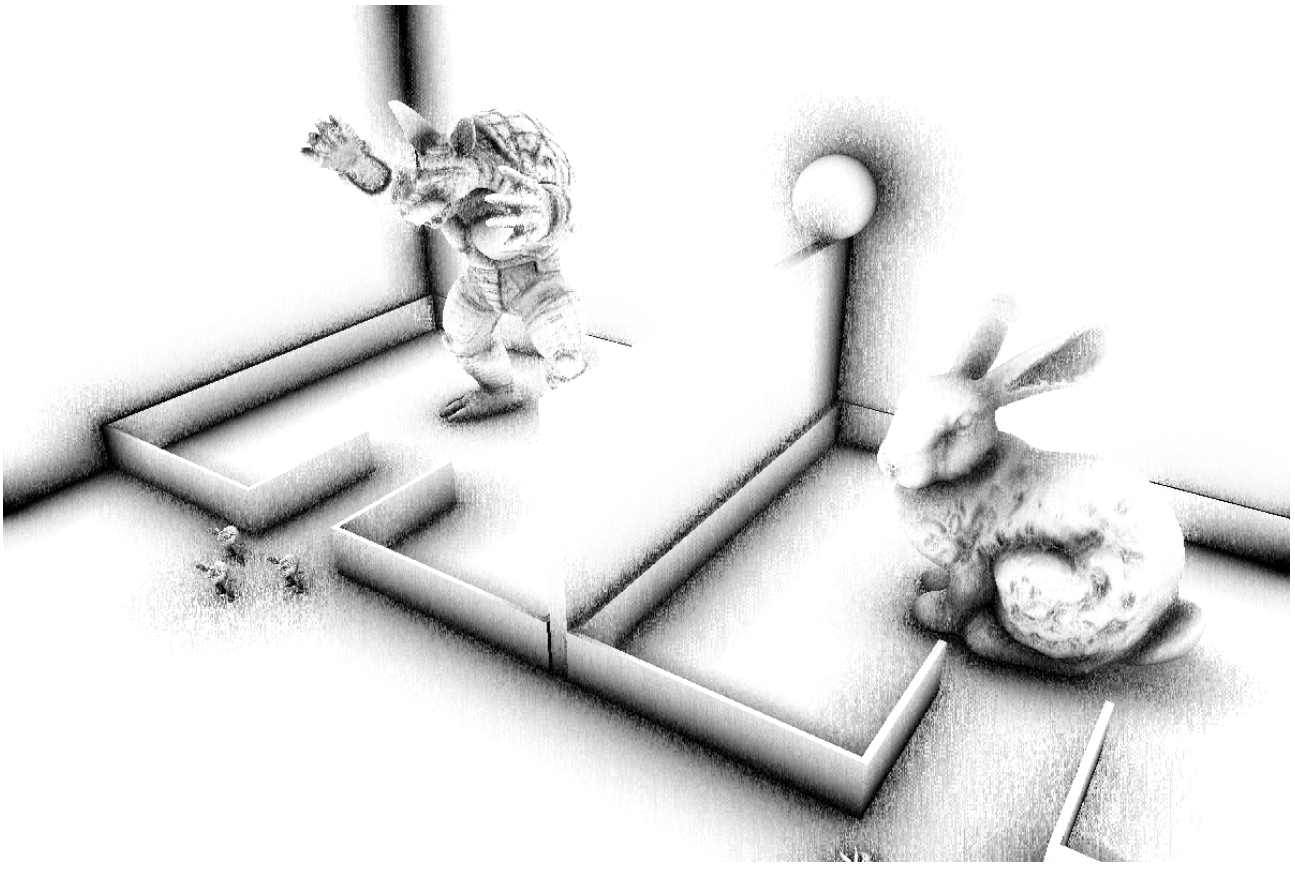
$$R = (N_i \cdot N)_+ \frac{1}{\sqrt{2\pi s}} e^{-\frac{(d_i - d)^2}{2s}}$$
$$W = R \cdot S$$

, where N_i and N are normal at certain fragment coordinates and di and d is depth from the camera. S is Gaussian weights and s is 0.01(variance of the Gaussian). Note that the program needs to normalize the weight before filtering. The program will filter the ambient occlusion map vertically and then filter the ambient occlusion map horizontally. Since the program uses a compute shader and there are some repeated loads for pixels, the compute shader utilizes the threaded group shared memory. Even though the size of gbuffer is not a multiplication of 128, it shows improvements when using group shared memory with size of near 128(In this project, the g-buffer size is fix to 1200X800).

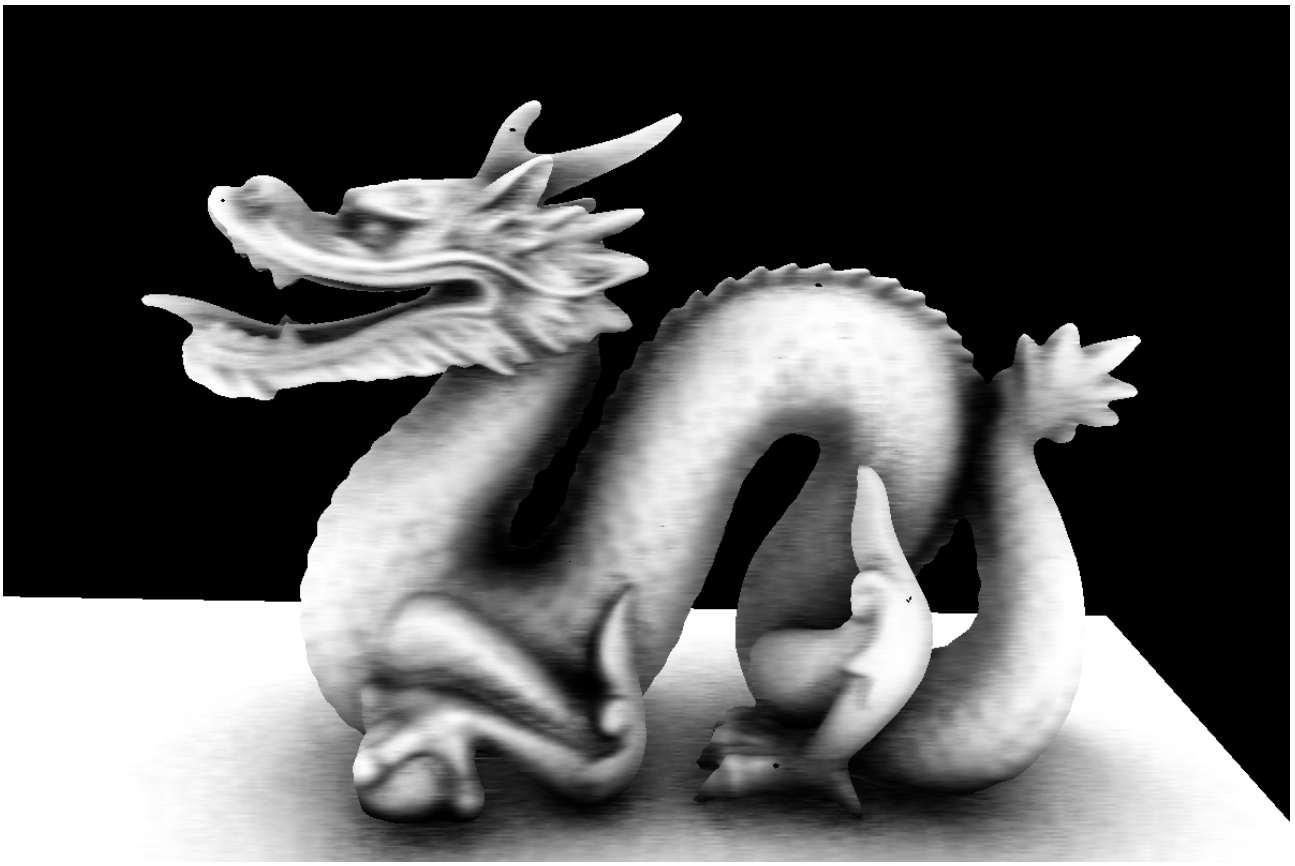
Following is sample images for filtering ambient occlusion map:

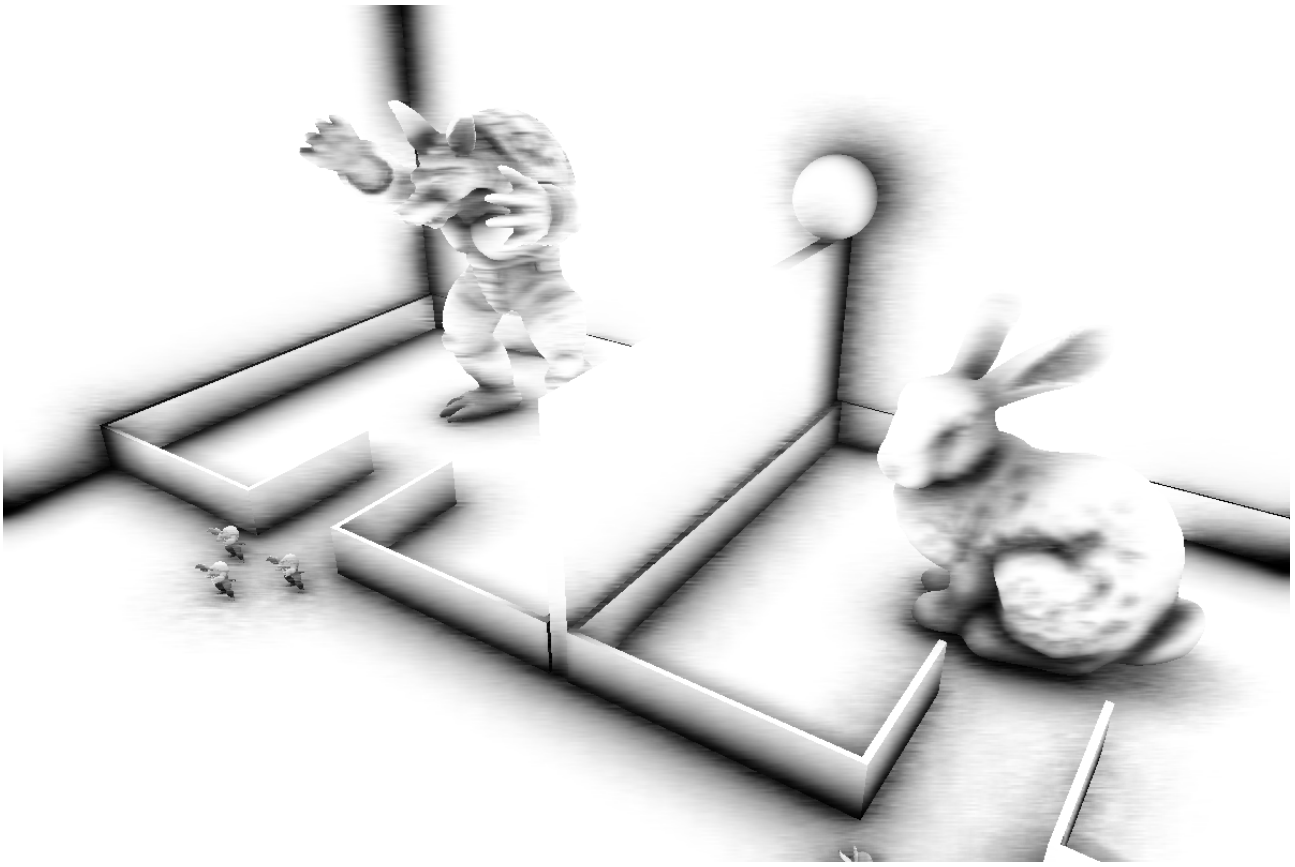
ambient occlusion map before filtering



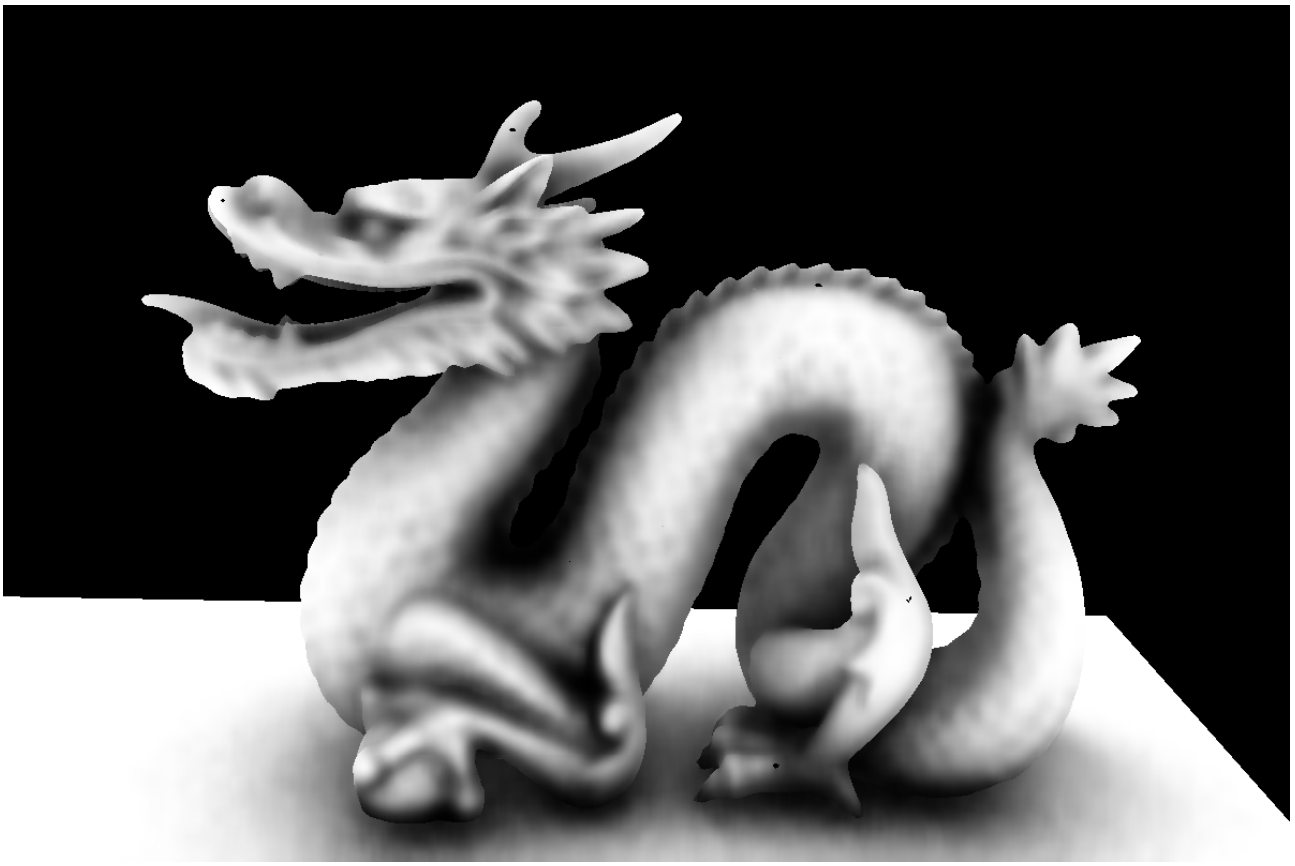


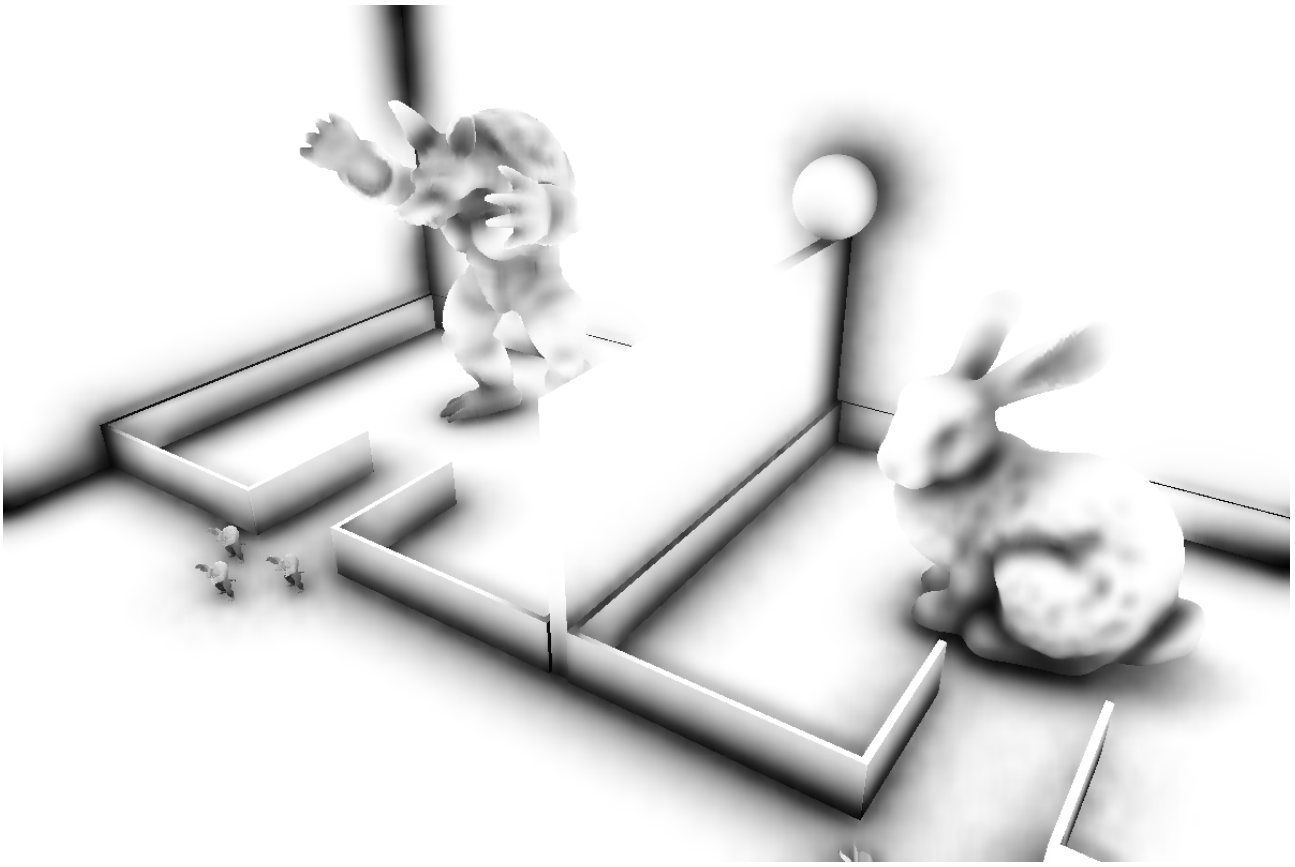
ambient occlusion map after horizontally blur texture



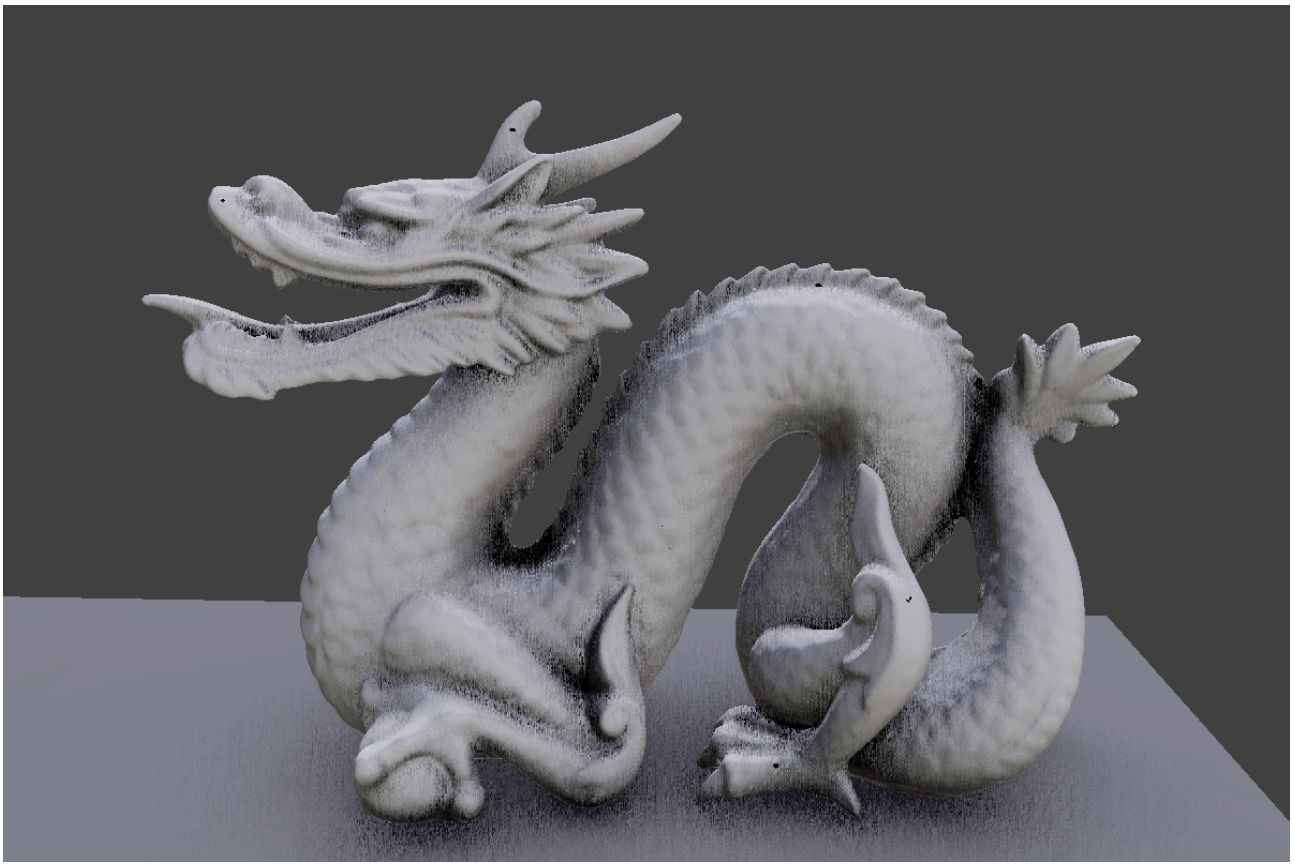


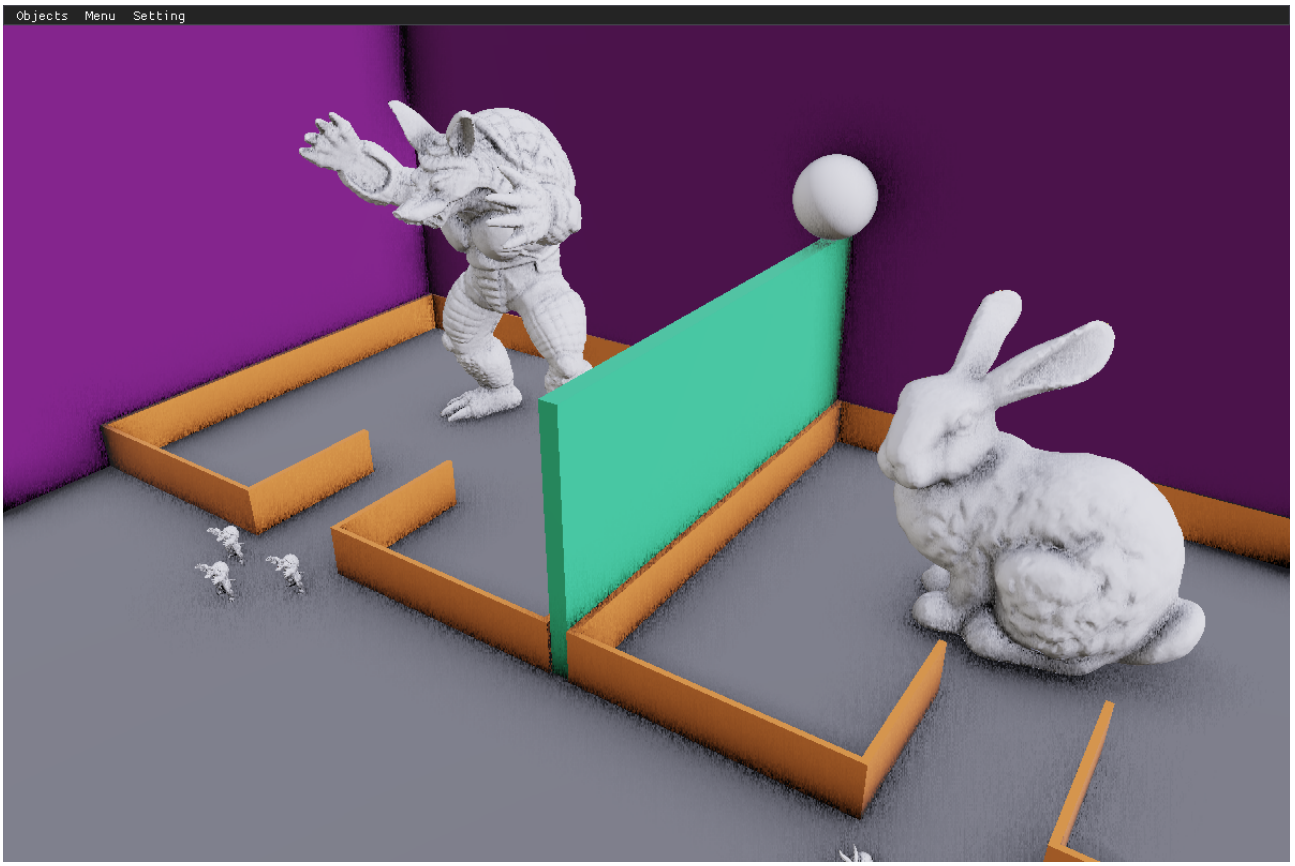
ambient occlusion map after filtering the texture





final scene without filtering ambient occlusion map





final scene with filtering ambient occlusion map