CSE 167 - Final Project

Shadow Mapping

I. Introduction

Shadow mapping is the main topic of this final project. It is basically a process of adding shadows to 3D scenes. In other words, shadows help viewers observe a great deal of realism or spatial relationships with a greater sense of depth among objects in the 3D scene. From the light source, shadows are added to the scene by mapping out the object's depth while rendering the scene.

This project is mostly implemented based on Professor Albert Chern's instructions for CSE 167 – Fall 2022, the lighting technique and code implementation from HW3, and the shadow mapping technique guidelines in the below sources:

- https://www.opengl-tutorial.org/intermediate-tutorials/tutorial-16-shadow-mapping/
- https://learnopengl.com/Advanced-Lighting/Shadows/Shadow-Mapping

II. Briefly explanation of the related math/physic methods

The shadow mapping is technically based on two rendering passes by calling the function of the drawing scene twice. Each rendering pass is set with a different camera and shader:

1. In the first rendering pass, the shadow map is created from the light's point of view. In other words, the light from the camera is responsible for casting a shadow by performing a per-pixel shading computation following the Blinn – Phong shading equation as we did in HW3

$$\mathbf{R} = \mathbf{E} + \sum_{j=1}^{\text{num of lights}} \left(\mathbf{C}_{\text{ambient}} + \mathbf{C}_{\text{diffuse}} \max(\mathbf{n} \cdot \mathbf{l}_j, 0) + \mathbf{C}_{\text{specular}} \max(\mathbf{n} \cdot \mathbf{h}_j, 0)^{\sigma} \right) \mathbf{L}_j.$$

The camera at this first pass uses a camera matrix $\mathbf{C_{light}}$, an inverse of the camera matrix – a view matrix $\mathbf{V_{light}}$, and a projection matrix $\mathbf{P_{light}}$ to produce an image whose pixel value is the depth of the scene. This depth value is possibly the physical distance between the fragment position and the eye of light.

2. In the second pass, the camera is placed at its actual position and also uses its own matrixes to form the final image such as a camera matrix $\mathbf{C_{cam}}$, a view matrix $\mathbf{V_{cam}}$, and a projection matrix $\mathbf{P_{cam}}$. The color of this final image is evaluated by using the information sampled from the first image. Based on the above shading equation and implementation from HW3, the fragment shader is the main different change in the implementation of this second rendering pass. The surface will face away from the light and the shadow technique will not be invoked if the multiplication between the surface normal vector **n** and the directional light **l** is less than 0. Vice versa, we need to check if the current fragment is in the shadow or not by transforming the fragment position to the light's normalized device coordinate taken from the fragment coordinate in the first pass image, sampling the depth recorded in the texture with the texture coordinates range in [0,1] x [0,1], and then computing the depth value of the current fragment until the sampled depth is much shorter than the new depth value to result in the needed shadow.

III. Algorithm description

As mentioned above, the idea of the shadow map algorithm also consists of two rendering passes. Firstly, only each fragment depth is computed when the scene is rendered from the light's point of view. The depth map is stored from the first pass so that in the second pass,

the scene is rendered as usual from the main camera's point of view with an extra implementation to see if the current fragment is in the shadow.

The core structure of this algorithm is used based on the OpenGL tutorial sources and extending the skeleton code for implementing lighting and rendering scene given from HW3 with the necessary modifications and extra depth shader implementations.

For setting up the depth map and its corresponding frame buffer object, the procedure includes the following steps:

1. A frame buffer is generated which regroups 1:

```
GLuint depthMapFBO;
glGenFramebuffers(1, &depthMapFBO);
```

2. 2D depth texture is created with a 1024x1024 16-bit depth texture to contain the depth map:

```
GLuint depthMap;
glGenTextures(1, &depthMap);
glBindTexture(GL_TEXTURE_2D, depthMap);
glTexImage2D(GL_TEXTURE_2D, 0, GL_DEPTH_COMPONENT,
        SHADOW_WIDTH, SHADOW_HEIGHT, 0, GL_DEPTH_COMPONENT,
        GL_FLOAT, NULL);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
```

3. The depth map is attached to the depth buffer of the depth map frame buffer object:

```
glBindFramebuffer(GL_FRAMEBUFFER, depthMapFBO);
glFramebufferTexture2D(GL_FRAMEBUFFER, GL_DEPTH_ATTACHMENT,
        GL_TEXTURE_2D, depthMap, 0);
```

```
glDrawBuffer(GL_NONE); // Omitting color data
glReadBuffer(GL_NONE); // Omitting color data
glBindFramebuffer(GL_FRAMEBUFFER, 0);
```

IV. Detail implementation

Before implementing the project, using the skeleton code included the scene and lighting implementations from HW3 to initialize the work. Then, creating and performing some modifications based on this skeleton code in the following order:

- In the included folder storing the core structures:
  1. Deriving a structure named DarkShader.h from the Shader.h and mostly based on SurfaceShader.h containing a shader that has uniform variables and functions for setting and initializing view and projection.
  2. Also adding viewShadow and projectionShadow variables for light space, then the functions for setting and initializing viewShadow and projectionShadow to the SurfaceShader.h structure.
  3. Next, augmenting the Light.h structure as a class for camera object to store the texture or buffer object of the shadow map by using the procedure mentioned in the algorithm description.
  4. In Scene.h, adding a pointer attribute darkShader to the class and deleting this pointer later after creating another function void drawDepth().
- In the part storing shaders:
  1. First off, augmenting Lighting.frag by appending a texture sampler as uniform sampler2D shadowMap; to sample the shadow map. To smooth out the problem of shadow acne, adding a slight bias computation to the sampled depth

```
float bias = max(0.05 * (1.0 - dot(normal, normalize(lightpositions[i].xyz / lightpositions[i].w))),
0.00000065);
```

before using a helper function to check whether the fragment is in shadow with the bias value or not. If not, using a per-pixel shading computation following the Blinn – Phong shading equation.

2. Modifying projective.vert to get a new output for positionShadow which is another coordinate of the shader in the light space.
3. Deriving another vertex shader named shadowMap.vert based on the vertex shader of the light projection projective.vert to forward the raw position and normal in the model coordination to frag shader.
4. Creating a simple fragment shader named shadowMap.frag to visualize depth with a coordinate value of a linearDepth computed by far and near variables from the camera's clipping plane.

❖ In the implementation part:

1. Augmenting another draw() method as void Scene::drawDepth() in Scene.cpp based on the function of void Scene::draw(void) but for handling the shader swapping on demand. Also setting the correct light and depth shader for projection and view matrices.
2. Adding a ground plane and model to Scene.inl so that the viewer can observe the effects of the object's shadow with a single light source of "sun", then also initialize a DarkShader.
3. Finally, to forming the final image, adding a depth rendering pass to the function display() in main.cpp heavily based on the first rendering pass that is used to cast shadow.

V. Results and Demo

A demo is attached to this report as a short video (ShadowMapping.mp4).

Below are the screenshot results of our final product:

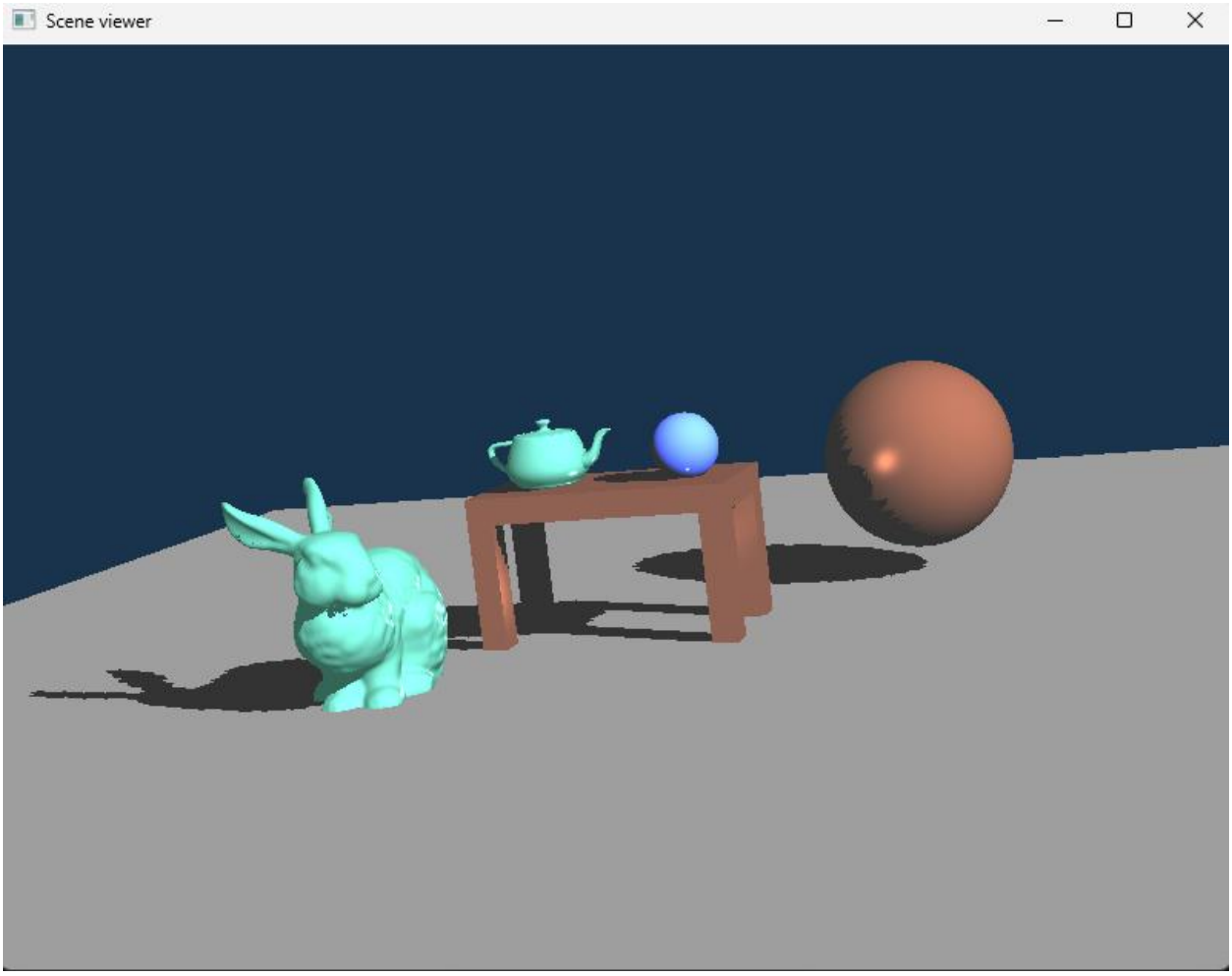

Figure 1: View from the front

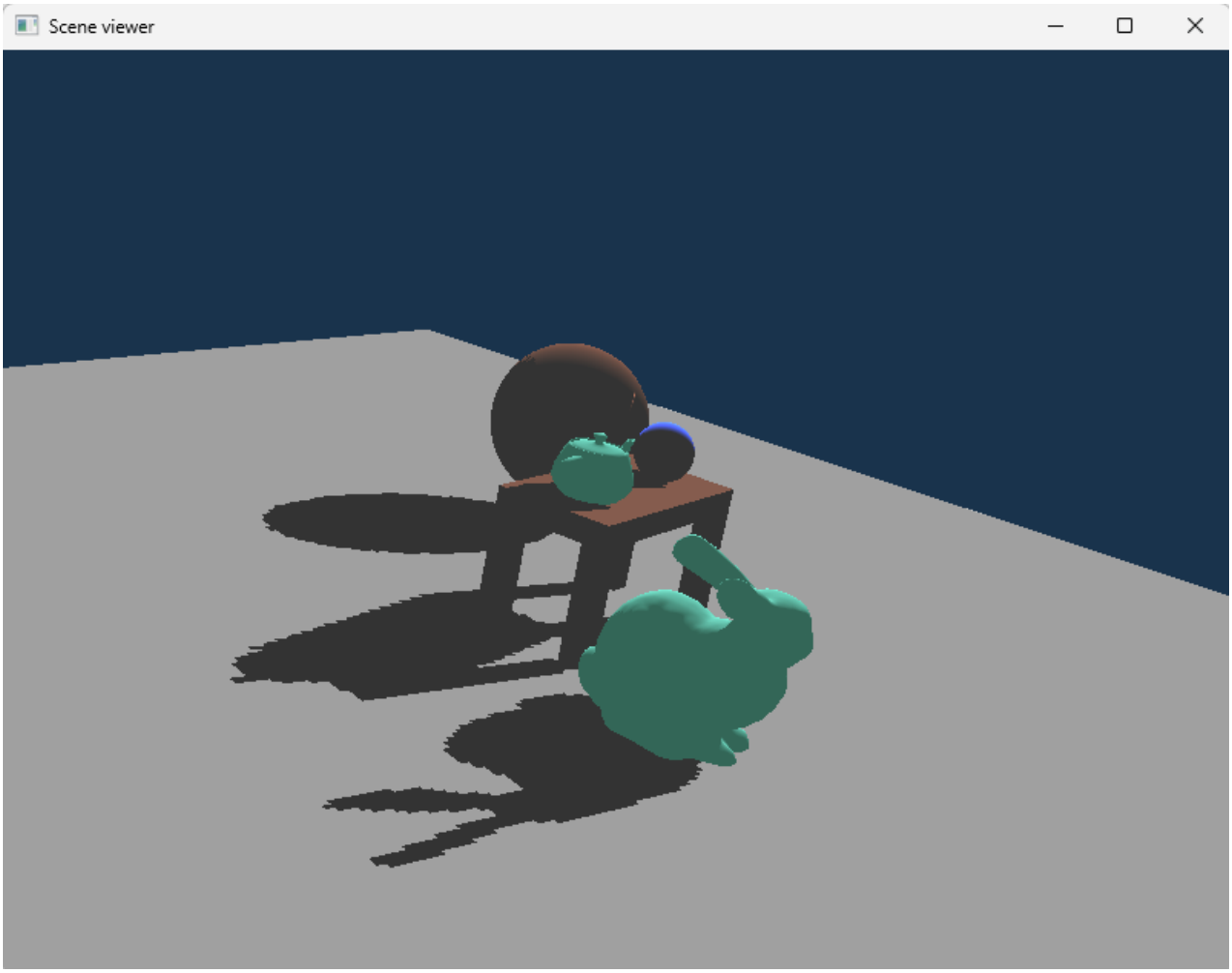

Figure 2: View from the back

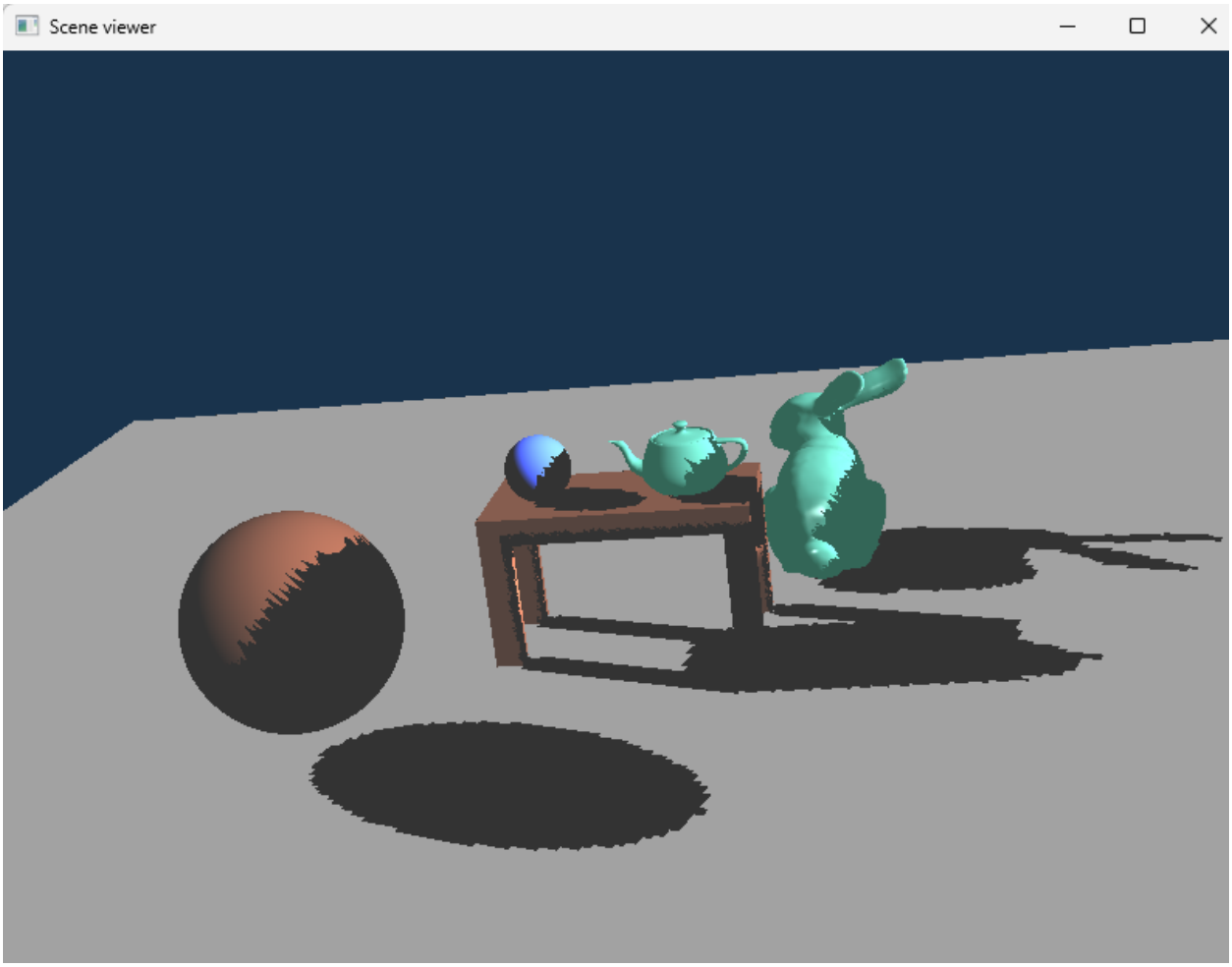

Figure 3: Another view from the back

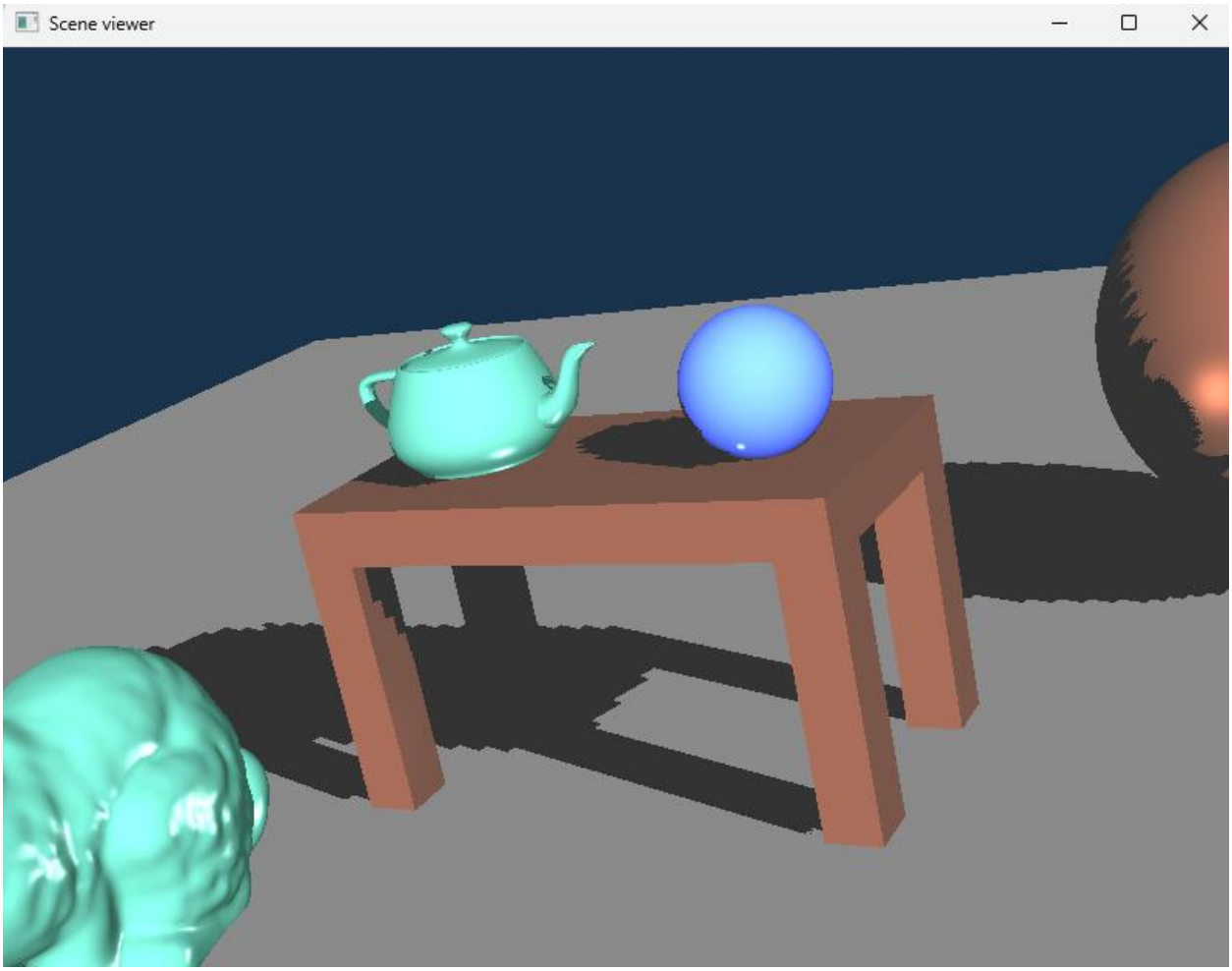

Figure 4: Close view of the table

VI. Contribution

This final project is created by

- Anh Vuong – PID A17201884 (ahvuong@ucsd.edu)
- Huy Nguyen – PID A17160864 (chn019@ucsd.edu)

Finally, we are grateful to Professor Albert Chern and all TAs for helping us go through CSE 167 course during this Fall 2022 quarter.