Assignment 1: Exploring OpenGL and Phong Lighting

NAME: 王鹏豪
STUDENT NUMBER: 2021533138
EMAIL: WANGPH1@SHANGHAITECH.EDU.CN

1 INTRODUCTION

- Task 1: Load Mesh from File has been down
- Task 2: Phong Lighting has been down
- Task 3: Camera Control has been down
- Bonus1: Play With Light has been down
- Bonus2: Geometry Shader has been down

2 IMPLEMENTATION DETAILS

2.1 Load Mesh From File

In this section, we need to load mesh and draw it onto the screen.

2.1.1 Load Part. In this section, we need to load mesh of .obj format, the .obj file is consist of three parts:

- v is the position of vertices of 3 floats
- n is the normal vector of vertices of 3 floats
- f is the indices of each face of 6 ints

As for v, we can directly push it into *vertices* as *Vertex* of *position*. As for n, because we haven't known the corresponding relationship with v, we can store it in a *temp* vector. As for f, we need first store the indices of *Vertex* into *indices*, then as we have the corresponding relationship of v and n, we need to push the normal stored in *tmep* vector into corresponding *Vertex* of *normal*.

2.1.2 Draw Part. In this section, we need to draw the loaded mesh to the screen. First we need to define:

- VAO Vertex Array Object
- VBO Vertex Buffer Object
- EBO Element Buffer Object

then make corresponding binding, then we use **glDrawElements()** and pass *indices.size()* to it to draw all the triangles of the mesh. Meanwhile, vertexshader and fragment shader is also needed to draw the mesh correctly.

- vertexshader
  we need to define it as a single file like *vertexshader.glsl* first we set aPos of vertex location and aNormal of vertex Normal to attribute location 0,1 then as mesh is 3D model, we need matrix to transform the vertices location, so we define three uniform matrix *model*, *view*, *proj*,
  - *model* matrix includes shifting, scaling, rotating transforms
  - *view* matrix is used to move the scene to make it visible

© 2022
Computer Graphics I Course Assignment

  – *proj* matrix has 2 types: *Orthographic Projection Matrix* and *Perspective Projection Matrix* , it is used to transform coordinate from **view space** to **clip space**.

First our model vertex location coordinate is located at **local space**, we use *model* matrix to transform it into **world space**, Second as all the output pictures is viewed from a specific location of camera, we need *view* matrix to transform the coordinate into **view space**. Third we needed to transform the coordinate into **clip space**, where vertex out of a specific frustum should be clipped. We needed to use *proj* matrix. Fourth we needed to perform a viewport transform to transform it into screen space. Lastly we needed to combine these matrix together, we have $V_{clip} = M_{projection} \cdot M_{view} \cdot M_{model} \cdot V_{local}$

- fragementshader
  we need to define it as a single file like *fragementshader.glsl* as no light is demanded, we can directly output the color as an constant.

2.2 Phong Lighting

In this section we needed to implment three types of lighting and add them together

- Ambient Lighting, even there is no lighting, we also has light in the environment, we could simply set a constant of ambient strength and multiply it with object color.
- Diffuse Lighting, to calculate this lighting, we need normal and direct lighting. we already have the normal stored in the **Vertex.normal**, then we needed to get the direct light, we need the position of light and fragment which is **lightpos** and **FragPos**, then we make a minus of them to get direct light which is **lightDir**, then make a dot operation of **norm** and **lightDir**, then muptiply it with light color.
- Specular Lighting, this comes from the reflection of light on the surface of objects, we needed the viewer's position which is the camera's position. Then make a minus of **viewPos** and **FragPos**, and because of the reflection, we need reflect function to reflect it. Then make a dot operation with **viewDir** and **reflectDir**, and make a pow operation. The power exponent, is just the shininess.

2.3 Camera Control

In this section, we need to response to keyboard's input and mouse's input.

2.3.1 keyboard input. In this section, we process keyboard input. Keyboard will infect the camera's position, to deal with this, we need to define three vector of the camera. Up, Forward, Right vectors. We need to mapping WASD to Forward and Right vectors' add or minus. I also mapping Space and C to Up vector to make the camera able to move in three axis.

2.3.2 mouse input. In this section, we process mouse input. Mouse input includes horizontal mouse movement affects the yaw and vertical mouse movement affects the pitch. We need tell GLFW to hide cursor and capture the mouse input. After define callback functions, we need to register them. We need to calculate the offset perframe and add or minus them to pitch. Also, a restrict need to add to the operation.

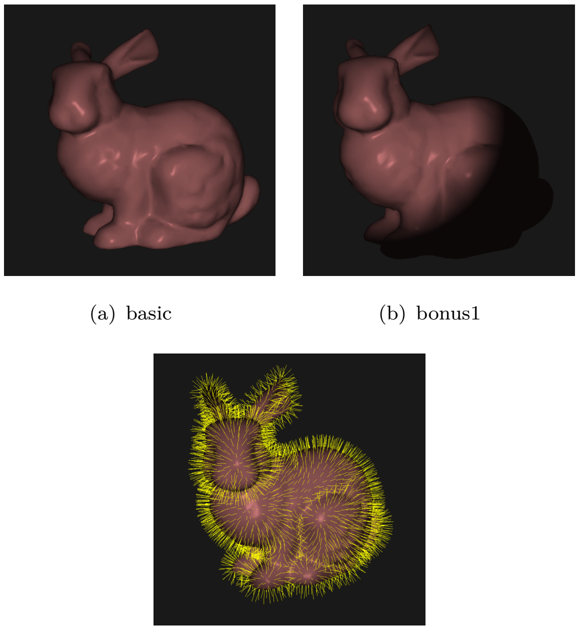
2.4 Bonus 1

In this section, I try to add a flashlight to the scene. First the same with point light, it contains ambient, diffuse, specular lights. As for flashlight, it is bright in the center, and dark if out of a specific area. We always want a little light even out of range, so the ambient will not be effected. We define a intensity and multiply diffuse and specular with it. The representition of I is $I = \frac{\theta - \gamma}{\epsilon}$ ϵ is the cosine difference between the inner ϕ and the outer cone γ that is to say $\epsilon = \phi - \gamma$. The θ is used to decide if the object is in the flashlight's inner area.

2.5 Bonus 2

In this section, we need to add fur to model, first we can make normal visible, that is, add a set of vs, gs, fs. where using gs to make line to present normal. First use vs to get normal vector, then in gs, we make a line of one endpoint is on the surface of the triangles, another one is the point add the normal vector. We get a line, and then simply draw it out using gs.

3 RESULTS


(a) basic (b) bonus1

(c) bonus2

Fig. 1. Results