Assignment 3

Qiming Cao ID:qc690

A demo video is made stored in the directory.

Keyboard Function

1-Insert a unit cube
2-Insert a unit bunny
3-Insert a unit bumpy cube
W-Forward the camera
S-Backward the camera
Z-Move the camera to left
C-Move the camera to right
X-Move the camera to up
V-Move the camera to down
A-Move the camera to left in a ball
D-Move the camera to right in a ball
Q-Move the camera to up in a ball
E-Move the camera to down in a ball
P-Delete the object
U- Wireframe:
I-Flat Shading
O-Phong Shading

Keypad

4-Move the object to left
6-Move the object to right
1-Backward the object
3-Forward the object
7-Rotate the object to left
9-Rotate the object to right
2-Move the object to up
8-Move the object to down
- Shrink the object
+ Enlarge the object

1.1 Scene Editor

First, I need to implement a basic program to show simple object in 3D. After that, read the mesh to memory and upload it to GPU memory as well as detect the keyboard. As task requires me not to upload the data of mesh to GPU twice, so I need to keep track of the model matrix of each object. So I build a struct to store the information of each copy. Once “1” is pressed, a new my_copy object will be pushed into vector.

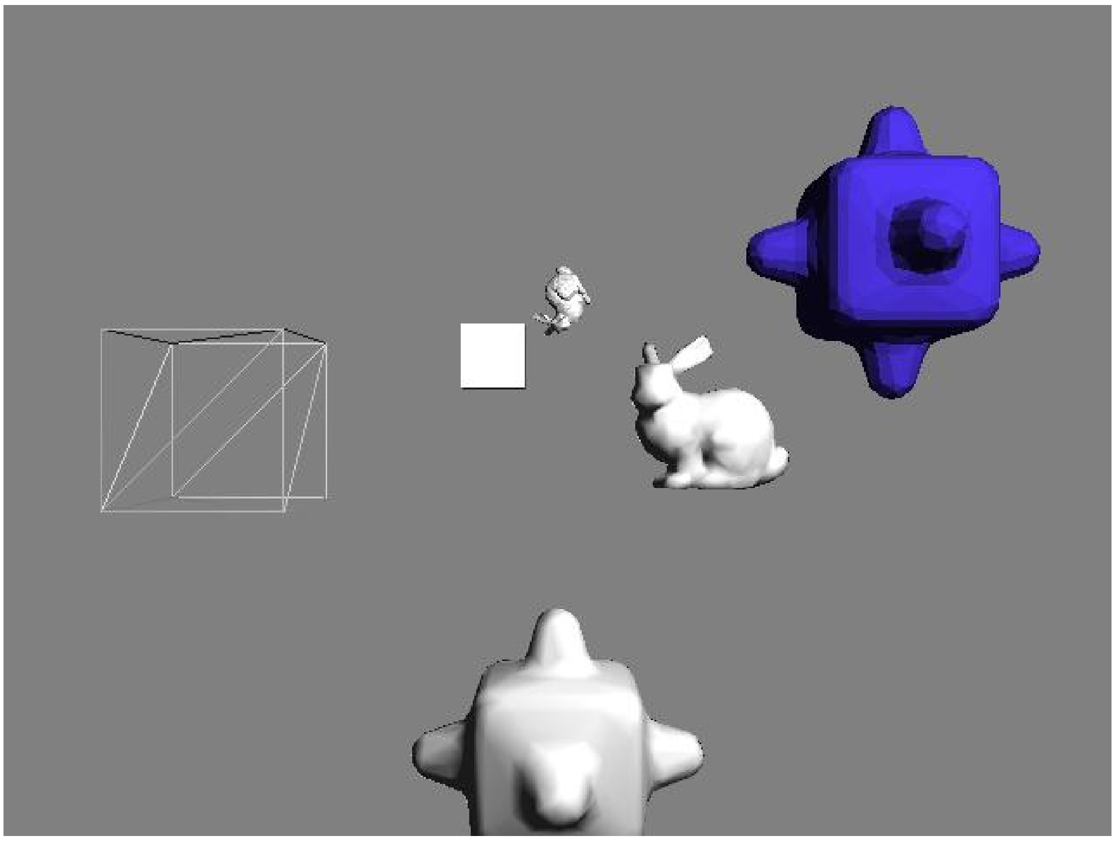
```
typedef struct object_copy
{
	unsigned int type;
	vec3 object_color=vec3(1.0f);
	mat4 model=mat4(1.0f);
	mat4 test=mat4(1.0f);
	unsigned int draw_type;
}my_copy;
vector<my_copy>copys;
vector<my_copy>::iterator iter_global;
```

1.2 Object Control

There are three difficulties in this part.

1. To implement the function of select object, I need to iterate over the copys to find the corresponding object which is closest to the camera.
2. To implement the Phong Shading, I need to compute the average vertices normals while reading the mesh files. For each vertex, know how many points are there by V_mesh, and calculate all the faces related to them by E_mesh(the index of vertices).
3. The lighting should be implemented in the shader, so the formula of calculating reflection like the Assignment 1 should be implemented in GPU. And a judgment is need to add to change the direction of norm.

```
"	if(dot(norm, ray_V)<0)"
"	{"
"		norm=-norm;"
"	}"
```



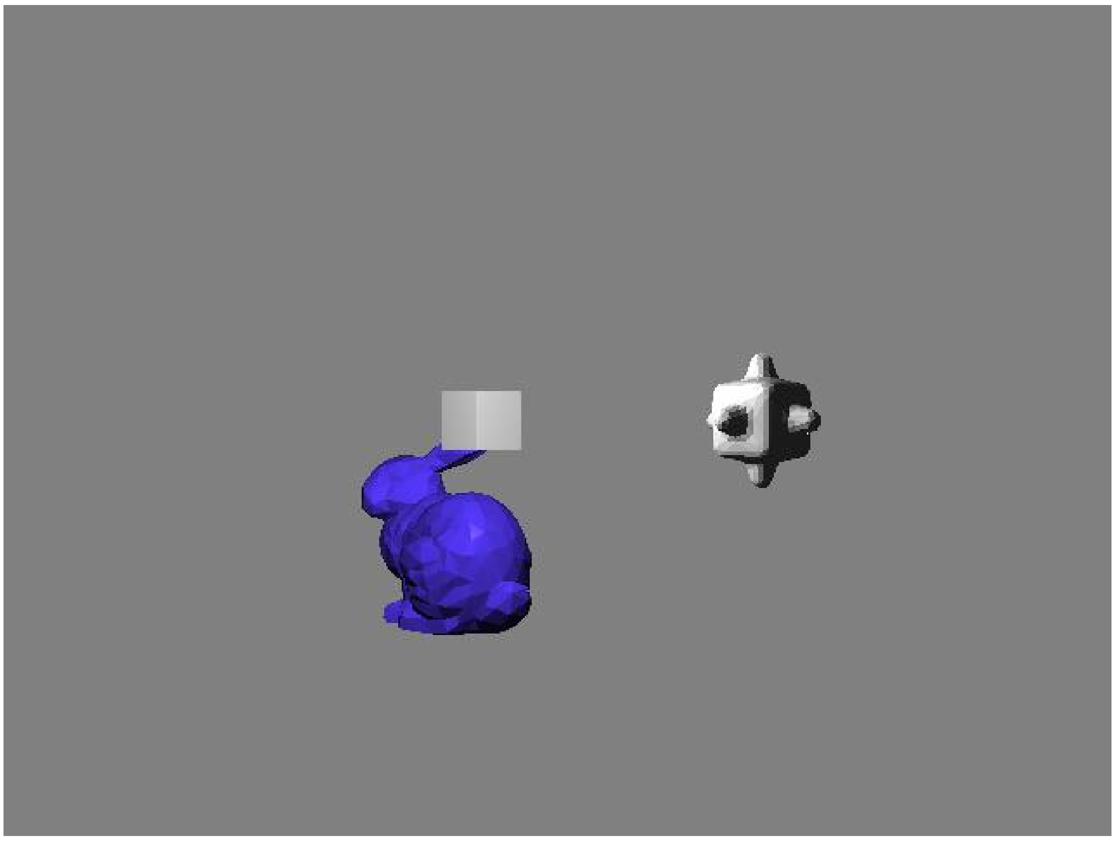
1.3 Camera Control

The main purpose of this task is to change the position of camera and try the take into account the size of the window to not distort the image. Set a camera matrix to main the size of object whenever the window is resized as the following code shows.

```
// Get size of the window
        int width, height;
        glfwGetWindowSize(window, &width, &height);
        float aspect_ratio = 1.0/float(width); // corresponds to the nece
ssary width scaling
        float aspect_ratio2=1.0/float(height);
        camera[0][0]=camera[0][0]*(1.0/aspect_ratio_pre)*aspect_ratio;
        camera[1][1]=camera[1][1]*(1.0/aspect_ratio2_pre)*aspect_ratio2;
        aspect_ratio_pre = 1.0/float(width); // corresponds to the neces
sary width scaling
        aspect_ratio2_pre=1.0/float(height);
        program.setMat4("camera", camera);
```

As to changing the position of camera, like left translation, just to change the variable of cameraPos like the below code.

```
cameraPos+=vec3(-1,0,0);
```



1.6 Track Ball

The main purpose of this task is to change the position of camera by a ball.

As to the camera position, just to rotate the origin camera position by the axis and then the result is the new camera position.

We need to make a judge if the rotation is larger than 90, if so, we have to change to direction of cameraUp.

```
if(cameraPos[2]<=0)
 {
      cameraUp=vec3(0,-1,0);
 }
 else
 {
      cameraUp=vec3(0,1,0);
 }
```