Computer Graphics HW1 Report

310551067 吳岱容

- **Key Event Explanation :**
  - Key 1~9 : Rotate the layer along a specific axis.
  - Key E, R, T : Rotate the entire cube along a specific axis.
- **Implementation :**
  - Cube :: draw()

```
// Blue, bottom
glBegin(GL_TRIANGLE_STRIP);
glColor3f(0.0f, 0.0f, 1.0f);
glNormal3f(0.0f, -1.0f, 0.0f);

glVertex3f(-1.0f, -1.0f, -1.0f);
glVertex3f(1.0f, -1.0f, -1.0f);
glVertex3f(-1.0f, -1.0f, 1.0f);
glVertex3f(1.0f, -1.0f, 1.0f);
glEnd();
```

I just did what the hint did. We need to draw a square, so we call glBegin (GL_TRIANGLE_STRIP) to start. GL_TRIANGLE_STRIP can help us draw a strip of triangles which can form a square. Then I defined the normal vector of the surface. I had the y value equal -1 because it is the bottom surface, and different surfaces would have different values. After defining the normal vector, I started to define the four vertices. To be noticed, be careful when defining the vertices. The order will affect the result, which may cause a face culling problem.

I did similar jobs when drawing other faces, but I adjusted some parameter values (e.g. color, normal vector,and so on).

  - Cube :: setUpModel ()

```
glm::mat4 rotationMattrix = glm::mat4_cast(this->rotation);
const float *ptrTrans = glm::value_ptr(this->translation);
const float *ptrRotate = glm::value_ptr(rotationMattrix);
glMultMatrixf(ptrRotate);
glMultMatrixf(ptrTrans);
```

We need to update the model with the new transformation and rotation matrix. Since the type of the rotation information saved in Cube class is quat, we need to transform the quaternion to a matrix. We can just call API to finish this job. Then we need to multiply the matrices to have the correct model. Be careful of the multiplication order, the matrix should be first translated, and then be rotated.

- QuanternionCamera :: updateProjection (float aspectRatio)

```
projectionMatrix = glm::identity<glm::mat4>();
projectionMatrix = glm::perspective(FOV, aspectRatio, zNear, zFar);
```

As the spec requirement, we need to use perspective projection, and the parameters we need are all given, so we only have to call the API to calculate.

- QuanternionCamera :: updateView ()

```
viewMatrix = glm::identity<glm::mat4>();
front = original_front*this->rotation;
up = original_up*this->rotation;
right = cross(front,up);
viewMatrix = glm::lookAt(position, position+front, up);
```

We need to update the camera view when we detect key or mouse events (The information will be saved in rotation and position.). We need to rotate the camera's axis by using rotation, and then we can get the new right vector by cross product front and up vectors. Finally, we can calculate the viewMatrix by calling the API.

- main()

```
for (int i = 0; i < cubes.size(); i++) {
  glPushMatrix();
  cubes[i]->setupModel();
  cubes[i]->draw();
  glPopMatrix();
}
```

Here, we traverse all the cubes and set up models, and then draw them. We need to use glPushMatrix() and glPopMatrix() to cache the current matrix, or after drawing the current matrix will be changed, and when we draw the next one, it won't be in the right position.

- keyCallBack (GLFWindow *window, int key, int, int action, int)

```
case GLFW_KEY_1:
  for (int i = 0; i < cubes.size(); i++) {
    float cubePos = cubes[i]->getPosition(Axis::Z);
    if (cubePos == -1.0f) cubes[i]->rotate(Axis::Z);
  }
  break;
```

```
case GLFW_KEY_E:
  for (int i = 0; i < cubes.size(); i++) cubes[i]->rotate(Axis::Z);
  break;
```

I implemented key events in this function. If you press a specific key, the cube will do the corresponding rotation. To rotate the whole

cube, We just need to traverse all the small ones and let them rotate along with an axis. When we need to rotate a specific layer, we need to set a condition to detect if the layer is in the position we want. We can get the layer position by using getPosition().

- **Problems :**
  - Face culling

    In the cube drawing part, we need to define the faces' vertices, and I did encounter some problems in this section. OpenGL will do the face culling task to lower the resources the program uses. Therefore, we should be very careful about the vertex order, or the face will be detected back facing and be eliminated.
  - Mouse controll

    I had some problems with the mouse control. I tried to do the same thing as the hint says, but it turned out to be a weird result. Not like the example, it seemed like I was rotating the cube with my mouse. Later, I discovered that I should have taken the camera position into account. After correcting the parameters, it works well now.