浙江大学

本科实验报告

课程名称：计算机图形学

姓　　名：王晨宇

学　　院：计算机科学与技术学院

系：

专　　业：计算机科学与技术

学　　号：3180104919

指导教师：唐敏

目录

浙江大学实验报告

课程名称：	计算机图形学	实验类型：	综合		
实验项目名称：	OpenGL 三维观察				
学生姓名：	王晨宇	专业：	计算机科学与技术	学号：	3180104919
同组学生姓名：	None	指导老师：	唐敏		
实验地点：	紫金港机房	实验时间：	2021-04-15		

一、 实验目的和要求

在模型变换实验的基础上，通过实现下述实验内容，掌握 OpenGL 中三维观察、透视投影、正交投影的参数设置，并能使用键盘移动观察相机，在透视投影和正交投影间切换，验证课程中三维观察的内容；进一步加深对 OpenGL 三维坐标和矩阵变换的理解和应用。

二、 实验环境

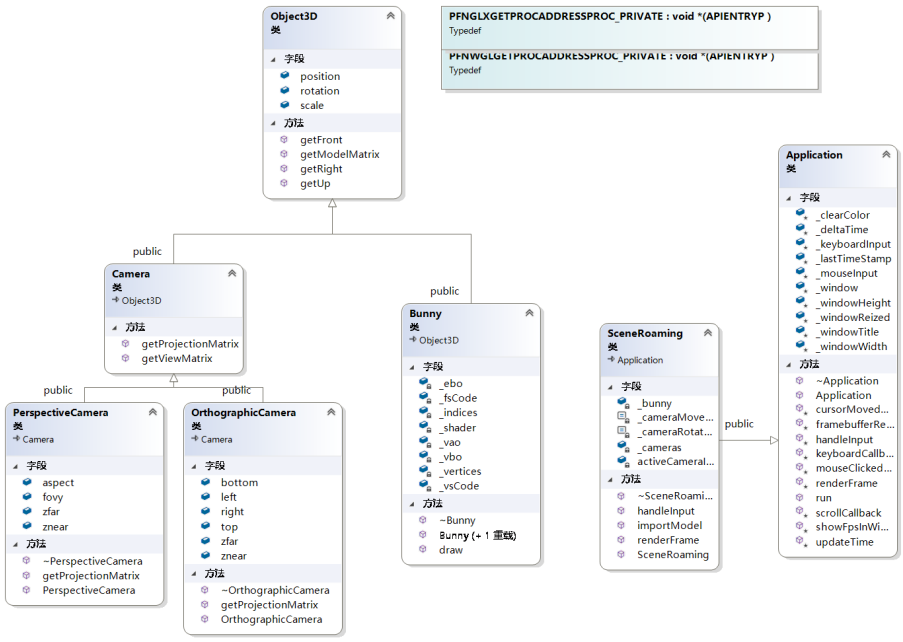
- Visual Studio C++ 2019
- GLFW、GLAD、GLM

三、 实验内容和原理

这个部分主要是对代码框架和实现细节的理解。

3.1 类图

通过类图能够便于理解整个实现的结构



3.2 Object3D 类

主要是维护一个点的三维坐标和这个点对应的三维坐标系。并维护缩放、旋转矩阵。

当发生旋转时，三维坐标系的单位向量也要发生旋转。

同时我们还要获得对应的 Model Matrix，这个过程类似上个实验中的 draw 方法的实现。

3.3 摄像机类

摄像机类的主要功能是获得视图空间的变换矩阵和投影矩阵。

ViewMatrix 的获得

```
1  glm::mat4 Camera::getViewMatrix() const {
2    return glm::lookAt(position, position + getFront(), getUp());
3  }
```

lookAt 函数需要一个位置、目标和上向量。这些都在 Object3D 里写好了。

其中因为实现的是 FPS 相机，所以 target 默认是正前方也就是 getFront() 的向量。如果是 LookAt 相机，把 target 设置成物体的中心点（例如这里是 (0,0,0)），这样物体会一直固定在屏幕中间，相机的移动就变成了绕着物体旋转。

$$LookAt = \begin{bmatrix} R_x & R_y & R_z & 0 \\ U_x & U_y & U_z & 0 \\ D_x & D_y & D_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 & -P_x \\ 0 & 1 & 0 & -P_y \\ 0 & 0 & 1 & -P_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

这个矩阵其实就是原点和标准坐标系变换到相机位置和坐标系的变换矩阵的逆矩阵

正交投影相机

因为我们默认用的是右手系，所以用的是这段

```
template<typename T>
  GLM_FUNC_QUALIFIER mat<4, 4, T, defaultp> orthoRH_NO(T left, T right, T bottom,
      T top, T zNear, T zFar)
  {
    mat<4, 4, T, defaultp> Result(1);
    Result[0][0] = static_cast<T>(2) / (right - left);
    Result[1][1] = static_cast<T>(2) / (top - bottom);
    Result[2][2] = - static_cast<T>(2) / (zFar - zNear);
    Result[3][0] = - (right + left) / (right - left);
    Result[3][1] = - (top + bottom) / (top - bottom);
    Result[3][2] = - (zFar + zNear) / (zFar - zNear);
    return Result;
  }
```

是标准立方体变换的逆矩阵。对角线上是缩放，然后平移

$$M_{\text{ortho}} = \begin{bmatrix} \frac{2}{r-l} & 0 & 0 & 0 \\ 0 & \frac{2}{t-b} & 0 & 0 \\ 0 & 0 & \frac{2}{n-f} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -\frac{r+l}{2} \\ 0 & 1 & 0 & -\frac{t+b}{2} \\ 0 & 0 & 1 & -\frac{n+f}{2} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

透视投影相机

```
template<typename T>
  GLM_FUNC_QUALIFIER mat<4, 4, T, defaultp> perspectiveRH_NO(T fovy, T aspect, T
      zNear, T zFar)
  {
    assert(abs(aspect - std::numeric_limits<T>::epsilon()) > static_cast<T>(0));

    T const tanHalfFovy = tan(fovy / static_cast<T>(2));

    mat<4, 4, T, defaultp> Result(static_cast<T>(0));
    Result[0][0] = static_cast<T>(1) / (aspect * tanHalfFovy);
    Result[1][1] = static_cast<T>(1) / (tanHalfFovy);
    Result[2][2] = - (zFar + zNear) / (zFar - zNear);
    Result[2][3] = - static_cast<T>(1);
    Result[3][2] = - (static_cast<T>(2) * zFar * zNear) / (zFar - zNear);
    return Result;
  }
```

透视变换有两个过程，一个是从视锥变换到长方体，再进行正交变换。

第一个过程利用相似三角形确定 x、y 方向的变换比例，再根据远近平面不变的性质解出 z 方向的变换向量

3.4 四元数的旋转

四元数由 4 个数字 [x y z w] 来表达：

```
x = RotationAxis.x * sin(RotationAngle / 2)
y = RotationAxis.y * sin(RotationAngle / 2)
z = RotationAxis.z * sin(RotationAngle / 2)
w = cos(RotationAngle / 2)
```

它本质上是存储了旋转轴和旋转角度，可以用 gml::angleAxis 生成。

另外的一些运算规则在讲义中有写，就不再赘述。

四、 代码实现

实现的部分主要是摄像机接受输入而做出相关动作的逻辑。

键盘控制的移动（平移变换）

```
  if (_keyboardInput.keyStates[GLFW_KEY_W] != GLFW_RELEASE) {
    std::cout << "W" << std::endl;
    // move front
    camera->position += camera->getFront() * _deltaTime * 5;
  }

  if (_keyboardInput.keyStates[GLFW_KEY_A] != GLFW_RELEASE) {
    std::cout << "A" << std::endl;
    // move left
    camera->position -= camera->getRight() * _deltaTime * 5;
  }

  if (_keyboardInput.keyStates[GLFW_KEY_S] != GLFW_RELEASE) {
    std::cout << "S" << std::endl;
    // move back
    camera->position -= camera->getFront() * _deltaTime * 5;
  }

  if (_keyboardInput.keyStates[GLFW_KEY_D] != GLFW_RELEASE) {
    std::cout << "D" << std::endl;
    // move right
    camera->position += camera->getRight() * _deltaTime * 5;
  }
```

根据按键的不同将相机移动相应单位的距离

4.1 鼠标控制的旋转

```
1   if (_mouseInput.move.xCurrent != _mouseInput.move.xOld) {
2     std::cout << "mouse move in x direction" << std::endl;
3     // rotate around world up: glm::vec3(0.0f, 1.0f, 0.0f)
4     const float angle = -_cameraRotateSpeed * _deltaTime * (_mouseInput.move.
          xCurrent - _mouseInput.move.xOld);
5     const glm::vec3 axis = glm::vec3(0.0f, 1.0f, 0.0f);
6     camera->rotation = glm::angleAxis(angle, axis) * camera->rotation;
7     _mouseInput.move.xOld = _mouseInput.move.xCurrent;
8   }
9
10  if (_mouseInput.move.yCurrent != _mouseInput.move.yOld) {
11    std::cout << "mouse move in y direction" << std::endl;
12    // rotate around local right
13    const float angle = -_cameraRotateSpeed * _deltaTime * (_mouseInput.move.
          yCurrent - _mouseInput.move.yOld);
14    const glm::vec3 axis = camera->getRight();
15
16    camera->rotation = glm::angleAxis(angle, axis) * camera->rotation;
17    _mouseInput.move.yOld = _mouseInput.move.yCurrent;
18  }
```

鼠标在平面向上的移动，相当于绕着当前相机坐标系的 Y 轴 (getRight()) 旋转，而当鼠标在平面横向的移动相当于绕着摄像机的 y 轴旋转（左右摇头）而 glm 的 angleAxis 函数可以将绕轴的旋转转成四元数矩阵的形式

五、 实验结果

旋转的展示比较困难，具体以验收和代码运行情况为准。

5.1 透视相机的运行结果

图 1: W

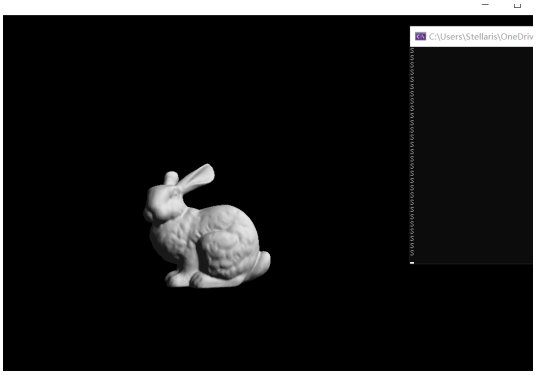


图 2: S

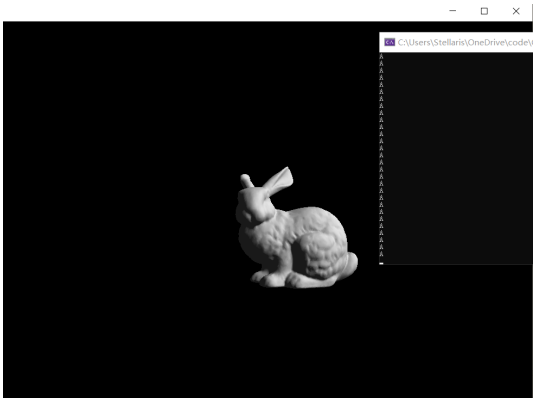


图 3: A

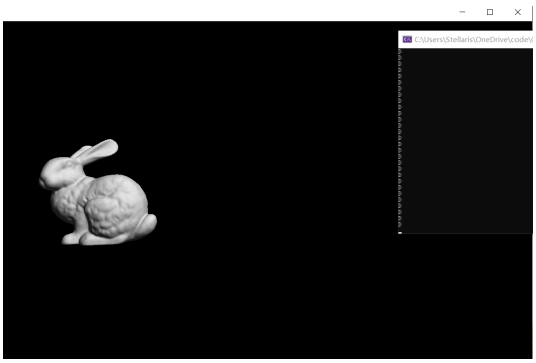


图 4: D

图 5: 鼠标向 y 方向移动

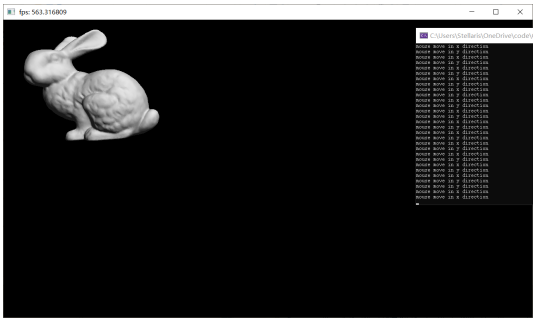


图 6: 鼠标随意移动

5.2 正交投影相机的运行结果

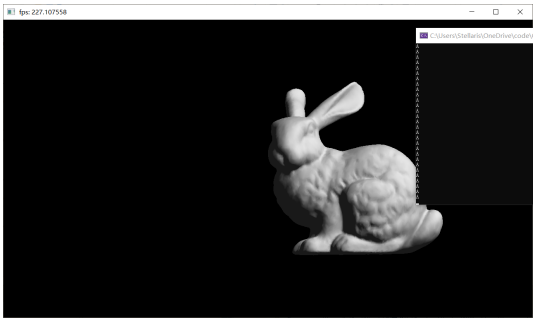


图 7: A

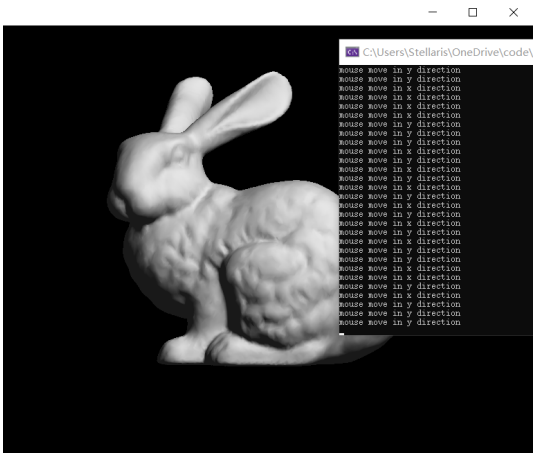


图 8: 鼠标移动

5.3 LookAt 相机的效果

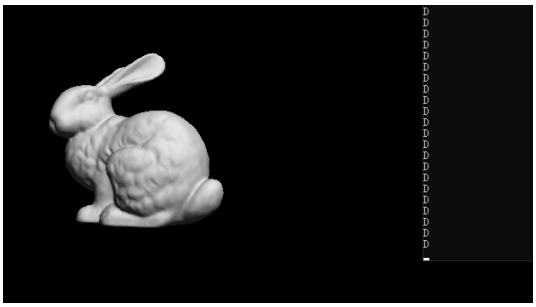


图 9: D

图 10: W

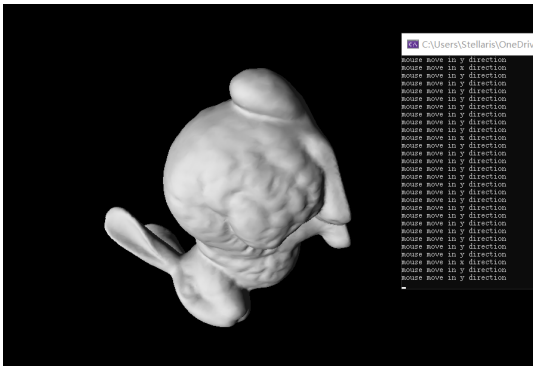


图 11: 鼠标移动

可以看到 LookAt 相机下的物体固定在中间，仅作大小缩放以及不同方向的旋转。

六、 讨论和心得

这次的实验工作量比较小，但是在理论学习的过程中还是学了非常多的知识，关于欧拉角和四元数方式的旋转研究了非常久，同时关于正交投影和透视矩阵的推导也非常有意思。

也花了非常多的时间研究了学长提供的代码，发现这样的模式非常优美，值得在之后的实践中模仿学习。

最后关于 LookAt 相机的讨论一开始想的不太对，在学长的提醒下找到了正确答案，在实现结果的表现也十分有趣。