PPT制作+汇报（5）	实验报告撰写（25）	创新性（10）	实验基本要求（60）			小组平均得分
			二维图形绘制与变换（20）	三维图形绘制与变换（20）	投影变换（20）	
5	22	7	20	20	20	94

《计算机图形学》3 组第一次实验报告

何宇航　21301095

钱波　　21301102

黄一圃　21301096

石昊原　21301103

张鲲鹏　21231303

文泽　　21301108

目录

1 实验目的

本次图形学实验旨在加深对二维、三维几何变换的理解，同时探究视图变化和投影变换的差异。具体目的包括：

1. 结合课堂理论知识，通过实际代码渲染加深对二维、三维几何变换的理解。
2. 理解视图变化和投影变换在图形学中的应用和区别。
3. 学习并掌握相关的图形学 API，如通过 OpenGL 编写代码实现二维、三维几何变换，同时比较不同变换的效果和应用场景。

2 实验环境

本次实验使用 OpenGL 完成。OpenGL 接口是由 Khronos 组织维护，其严格的规范了各个方法如何执行，以及对应的输出值。然而，具体 OpenGL 内部各个方法的实现方式由开发者（GPU 开发商）自行实现。下面将会对本次实验中所用的环境库或工具进行介绍。

2.1 GLFW

GLFW 是一个专门针对 OpenGL 的 C 语言库，它提供了一些渲染物体所需的最低限度的接口。它允许用户创建 OpenGL 上下文、定义窗口参数以及处理用户输入。GLFW 可以从 GLFW 处下载。在本次实验中，我们只需要编译生成的库和 include 文件夹两个内容。

2.2 GLAD

GLAD 是一个用于管理 OpenGL 扩展和核心 API 的加载库。在使用 OpenGL 开发时，直接从驱动程序加载所有函数是一个复杂且错误易发的过程，因为 OpenGL 的函数指针需要在运行时获取。GLAD 作为一个加载库，帮助开发者自动加载并管理这些函数指针，简化了 OpenGL 程序的初始化过程。GLAD 可以从 GLAD 进行下载。其中我们需要将 glad 和 KHR 两个头文件放入 include

文件夹中，将 glad.c 放入 src 文件夹中。

2.3 GLM

GLM (OpenGL Mathematics) 是一个为图形软件开发而设计的 C++数学库，专门用于模拟 GLSL（OpenGL Shading Language）的数学指令集，以方便在 C++中进行图形和游戏项目的开发。GLM 提供了矩阵、向量、四元数、数学函数和各种变换的实现，非常适合用于 3D 图形的变换计算，如旋转、缩放和平移等。GLM 可以在 GLM 中下载，将下载的内容放入 include 文件夹中即可。

2.4 Assimp

Assimp（Open Asset Import Library）是一个用于导入和处理各种 3D 模型格式的库。它提供了一个通用的接口来读取大多数流行的 3D 模型格式，如 FBX, COLLADA, 3DS, OBJ 等。Assimp 主要用于游戏开发和其他图形相关应用中，使开发者能够轻松地将 3D 资源集成到他们的项目中。

2.5 CMake

CMake 是一个跨平台的自动化建构系统，它用于管理软件构建过程中的编译器、编译选项、库文件和可执行文件的生成等。CMake 通过读取名为 CMakeLists.txt 的配置文件来生成标准的构建脚本（如 Makefile、Visual Studio 项目文件等），从而实现跨平台构建。在本次实验中，我们使用 CMake 完成相关库的链接，并进行编译等操作。

3 实验原理

3.1 着色器

着色器是一种计算机渲染图形的程序，用于计算每个像素的最终颜色，它主要可以分为顶点着色器和片段着色器两种，分别执行不同的功能。在 OpenGL 中，

着色器使用 GLSL(OpenGL Shading Language)来进行编写。

3.3.1 顶点着色器

顶点着色器（Vertex Shader）负责处理每个顶点的位置、颜色等属性，将顶点转换为裁剪空间的位置。它可以执行顶点坐标的变换操作，比如平移、缩放、旋转等。

顶点着色器接收顶点属性作为输入，可以包含顶点坐标、顶点颜色等等。顶点着色器则输出对应于裁剪空间中的顶点坐标，以及其他顶点属性。

3.3.2 片段着色器

片段着色器 (Fragment Shader) 主要用于处理光栅化阶段产生的每个片段 (像素），计算最终的颜色值。在计算颜色的同时，还会进行光照计算、纹理采样、颜色插值、透明度计算等操作。

片段着色器接收顶点着色器处理后的顶点属性和其他数据，比如光照条件、纹理信息等。最后输出每个像素的颜色值用于最终的渲染。

3.2 二维图形变换原理

3.2.1 位移变换

位移变换是一种基本的几何变换，用于沿着某个方向移动对象。位移不会改变对象的大小或形状，只改变其位置。

位移变换可以使用一个称为位移向量的三维向量来描述。将该三维向量放入其次坐标矩阵的最后一列即可实现位移操作。如下图所示，齐次矩阵中的 tx，ty 分别表示在 X 轴，Y 轴上的移动量。

$$\mathbf{T}(t_x, t_y) = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

图 3-2-1-1 位移矩阵

3.2.2 旋转变换

旋转变换可以改变图形的方向, 而不改变其大小或形状。在计算机图形学中, 平面旋转通常围绕着一个坐标系中的点进行。

旋转变换矩阵通常使用一个旋转角来实现，如下图所示。

$$\mathbf{R}(\alpha) = \begin{bmatrix} \cos\alpha & -sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

图 3-2-2-1 旋转矩阵

3.2.3 缩放变换

平面缩放变换是一种用于改变对象大小的几何变换。在二维空间中，缩放可以沿着 X 轴、Y 轴进行。

齐次坐标形式的缩放矩阵如下图所示，X 轴，Y 轴缩放因子分别为 Sx，Sy。

$$\mathbf{S}(s_x, s_y) = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

图 3-2-3-1 缩放矩阵

3.2.4 反射变换

平面反射变换是一种用于将当前图形轴对称的几何变换。在二维空间中，反射可以沿着 X 轴、Y 轴与原点进行。

- 关于 x 轴对称：$\begin{bmatrix} x^* & y^* & 1 \end{bmatrix} = \begin{bmatrix} x & y & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} x & -y & 1 \end{bmatrix}$
- 关于 y 轴对称：$\begin{bmatrix} x^* & y^* & 1 \end{bmatrix} = \begin{bmatrix} x & y & 1 \end{bmatrix} \cdot \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} -x & y & 1 \end{bmatrix}$
- 关于原点对称：$\begin{bmatrix} x^* & y^* & 1 \end{bmatrix} = \begin{bmatrix} x & y & 1 \end{bmatrix} \cdot \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} -x & -y & 1 \end{bmatrix}$

图 3-2-4-1 反射矩阵

3.2.5 切变变换

切变变换，也被称为剪切变换，是一种线性映射，它会改变物体的形状，但不会改变物体的面积。

$$\mathbf{H}(a_x, a_y) = \begin{bmatrix} 1 & a_x & 0 \\ a_y & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

图 3-2-5-1 切变矩阵

3.2.6 组合变换

二维组合变换是指将多个基本变换（如平移、旋转、缩放、反射、切变）结合在一起，以实现复杂的变换效果，通常从右往左进行，即对于一个点，从右边的齐次矩阵相乘，并执行对应的变换，然后向左继续执行。

3.3 三维图形变换原理

3.3.1 坐标系统

在绘制三维图形过程中，有五个重要的坐标系统：

- **局部空间**(Local Space，或者称为物体空间(Object Space))
- **世界空间**(World Space)
- **视图空间**(View Space，**或者称为视觉空间**(Eye Space))

- **裁剪空间**(Clip Space)
- **屏幕空间**(Screen Space)

在物体顶点最终转化为屏幕坐标之前会经历**模型**(Model)、视图(View)、**投影**(Projection)三个变换到这几个坐标系统，如下是变换的流程：

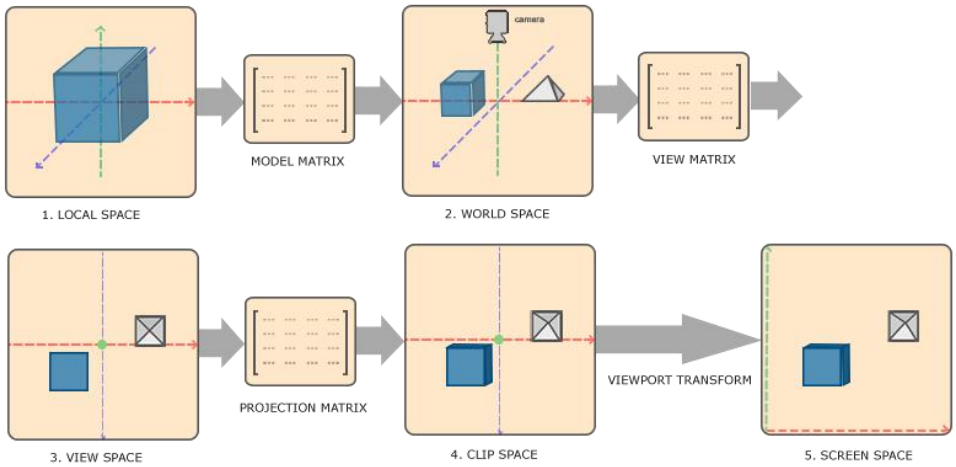

图 3-3-1-1 五种空间

我们首先根据局部空间原点构建一个物体，经历模型变换后根据世界坐标原点将其转换到世界空间，构建的其他所有物体也和该物体一样相对世界坐标原点进行摆放。

接着我们使用观察变换将世界空间坐标转换为视图空间坐标，这里每个坐标都是从摄像机的角度进行观察。

视图空间的坐标点会经过投影变换进入裁剪空间，坐标点会被转化至[-1,1]范围内，并判断哪些顶点会出现在屏幕上。

最后，裁剪空间的坐标会经历视口变换转换至 glViewport 函数所定义的坐标范围内。最后变换出来的坐标将会送到光栅器，将其转化为片段。

3.3.2 模型变换

3.3.2.1 位移变换

位移变换是一种基本的几何变换，用于沿着某个方向移动对象。位移不会改变对象的大小或形状，只改变其位置。

位移变换可以使用一个称为位移向量的三维向量来描述。将该三维向量放入

其次坐标矩阵的最后一列即可实现位移操作。如图 3-3-2-1-1 所示，齐次矩阵中的 t_x, t_y, t_z 分别表示在 X 轴，Y 轴，Z 轴上的移动量。

$$\mathbf{T}(t_x, t_y, t_z) = \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

图 3-3-2-1-1 位移齐次矩阵

3.3.2.2 旋转变换

旋转变换可以改变对象的方向或朝向，而不改变其大小或形状。在计算机图形学中，旋转通常围绕着一个坐标系中的轴进行，可以是围绕 X 轴、Y 轴、Z 轴或任意其他轴。

旋转变换矩阵通常使用一个旋转角来实现，其中绕着 X 轴、Y 轴、Z 轴等特殊的旋转方向的矩阵较容易实现，如图 3-3-2-2-1 ~ 3-3-2-2-3 所示。其中特别需要注意的是绕 Y 轴旋转的写法与绕 X 轴和 Z 轴略有不同，这是因为绕 Y 轴旋转时与右手法则方向相反造成的。

$$\mathbf{R}_x(\alpha) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha & 0 \\ 0 & \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

图 3-3-2-2-1 绕 X 轴旋转的旋转矩阵

$$\mathbf{R}_y(\alpha) = \begin{pmatrix} \cos\alpha & 0 & \sin\alpha & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\alpha & 0 & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

图 3-3-2-2-2 绕 Y 轴旋转的旋转矩阵

$$\mathbf{R}_z(\alpha) = \begin{pmatrix} \cos\alpha & -\sin\alpha & 0 & 0 \\ \sin\alpha & \cos\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

图 3-3-2-2-3 绕 Z 轴旋转的旋转矩阵

3.3.2.3 缩放变换

缩放变换是一种用于改变对象大小的几何变换。在三维空间中，缩放可以沿着 X 轴、Y 轴和 Z 轴分别进行，也可以沿着一个中心点进行均匀缩放。

齐次坐标形式的缩放矩阵如图 3-3-2-3-1 所示，X 轴，Y 轴，Z 轴缩放因子分别为 s_x, s_y, s_z。

$$\mathbf{S}(s_x, s_y, s_z) = \begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

图 3-3-2-3-1 缩放矩阵

3.3.2.4 组合变换

三维组合变换是指将多个基本变换（如平移、旋转、缩放）结合在一起，以实现复杂的变换效果。特别需要注意的是，在三维空间中，组合变换的顺序会影响最终的结果，因为矩阵乘法不满足交换率。

组合变换从右往左进行，即对于一个坐标值，它会从右边的齐次矩阵相乘，并执行对应的变换，然后向左继续执行。

3.3.3 视图变换

视图变换用于将场景从世界坐标系转换到视图坐标系，从而实现相机的观察效果。视图变换包括平移、旋转和缩放等操作，通常是相机固有的属性和行为。

通常是由一系列的位移和旋转的组合来完成，平移/旋转场景从而使得特定的对象

被变换到摄像机的前方。这些组合在一起的变换通常存储在一个视图矩阵中，它被用来将世界坐标变换到观察空间。

一般而言，我们需要通过相机的相关属性来计算视图矩阵。具体来说，我们需要相机的当前位置（position），摄像机方向（direction），摄像机向上方向（up）。由于摄像机方向和向上方向垂直，可以通过此计算出摄像机向右（right）方向。

3.3.4 投影变换

投影变换是计算机图形学中的重要概念，用于将三维场景投影到二维平面上，以便最终显示在屏幕上。投影变换通常包括透视投影和正交投影两种类型。

3.3.4.1 正交投影

正交投影是一种平行投影，远处和近处的物体看起来一样大。在正交投影中，视锥体被一个平行于观察平面的立方体代替，物体被投影到一个平行于观察平面的平面上，不考虑深度，因此没有透视效果，如图 3-3-4-1-1 所示。

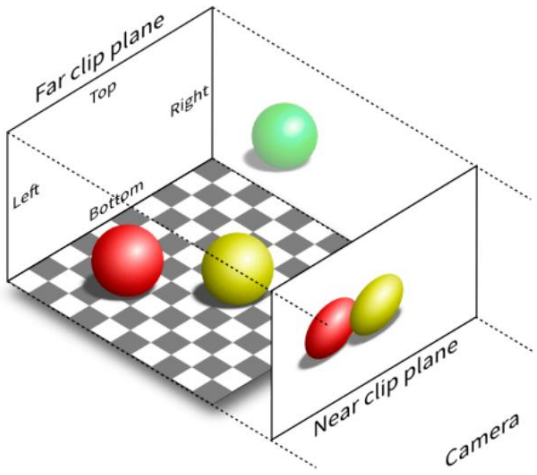

图 3-3-4-1-1 正交投影示意图

计算正交投影矩阵我们需要定义这个立方体六个属性，分别为左平面坐标（l），右平面坐标（r），上平面坐标（t），下平面坐标（b），近平面坐标（n），远平面坐标（f）。

变换矩阵如图 3-3-4-1-2 所示，首先将其移动到原点，然后进行缩放操作。

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

图 3-3-4-1-2 正交变换矩阵

3.3.4.2 透视投影

透视投影是模拟人眼观察物体时的效果，远处的物体看起来比较小，近处的物体看起来比较大。透视投影通过一个视锥体来表示可见空间，视锥体的近端和远端分别是近裁剪平面和远裁剪平面。在投影过程中，物体在视锥体内部被投影到一个平面上，从而产生透视效果，如图 3-3-4-2-1 所示。

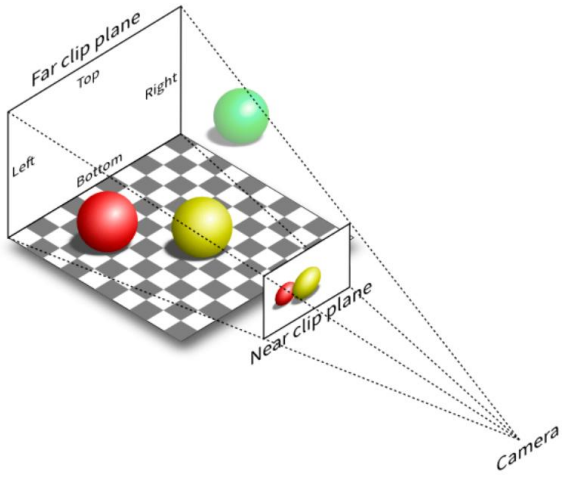

图 3-3-4-2-1 透视投影示意图

计算透视投影需要获得四个量，分别为仰角，裁剪平面的宽高比，近裁剪平面距离摄像机的距离，远裁剪平面距离摄像机的距离。

4 实验内容

4.1 二维图形绘制

在本次实验，我们绘制了一个三角形。首先创建三行六列的顶点矩阵 vertices，每一行前三列表示坐标，后三列表示颜色，由于是二维图形因此 z 轴坐标固定为

0，之后调用 glBufferData 函数将顶点矩阵上传到 gpu 上进行绘制。再创建一个着色器对其进行着色，该着色器由由顶点着色器和片段着色器组成。

4.2 二维变换

实现二维图形绘制之后，我们只需要在模型变换时进行对应的坐标转化即可，通常这个过程是对 Model 矩阵处理而实现的。由于 opengl 只有三维的概念，我们将 z 轴坐标固定为 0，以实现二维的处理。

4.2.1 位移变换

创建 glm::vec3 类型的变量 objectPos 记录当前图形的坐标，通过监听键盘事件更改。之后调用 glm::translate 函数，传入类型为 glm::mat4 的原始四维向量与 objectPos 即可得到位移后的图。

4.2.2 旋转变换

创建 objectRotation 变量记录图形旋转的角度，通过监听键盘事件更改。之后调用 glm::radians 函数，传入 objectRotation 得到旋转矩阵，之后再调用 glm::rotate 函数，传入类型为 glm::mat4 的四维向量与旋转矩阵即可得到旋转后的图。

4.2.3 缩放变换

创建 glm::vec3 类型的 objectScale 变量记录图形 x 轴与 y 轴的缩放比例，z 轴固定为 1，通过监听键盘事件进行修改。之后调用 glm::scale 函数，传入四维向量与 objectScale 即可得到缩放后的图。

4.2.4 反射变换

反射变换的方式按照缩放变换的一种特殊情况进行处理，将 x 轴与 y 的缩放比例设为相反数，之后再调用 glm::scale 函数即可实现反射变换。

4.2.5 切变变换

创建 glm::mat4 类型的原始四维向量 shearMatrix，将 shearMatrix[1][0]设为 x 轴切变比例，将 shearMatrix[0][1]设为 y 轴切变比例，之后将该四维向量与模型的变换向量点乘即可实现切变变换。

4.3 二维组合变换

将 Model 矩阵按照缩放、旋转、切变、位移的顺序进行处理，即可得到组合变换后的图形。

4.4 三维图形绘制

正二十面体是一种正多面体，由 20 个正三角形组成，正二十面体有 20 个面、30 个边和 12 个顶点，其对偶是正十二面体。正二十面体的各个顶点坐标有多种计算方式。其中较为巧妙的方式是使用三个黄金分割比的矩形计算。

黄金分割比的矩形为长宽比为$\frac{\sqrt{5}+1}{2}$的矩形，通过图 4-4-1 所示，我们可以得知三个矩形相互垂直，即分别构成三维坐标系中的三个平面。如果以中心点作为坐标原点，我们可以容易的计算出 12 个点的坐标。假设矩形的宽为 1，那么 12 个点的坐标分别为$(\pm\frac{\sqrt{5}+1}{4},0,\pm\frac{1}{2})$，$(\pm\frac{1}{2},\pm\frac{\sqrt{5}+1}{4},0)$，$(0,\pm\frac{1}{2},\pm\frac{\sqrt{5}+1}{4})$

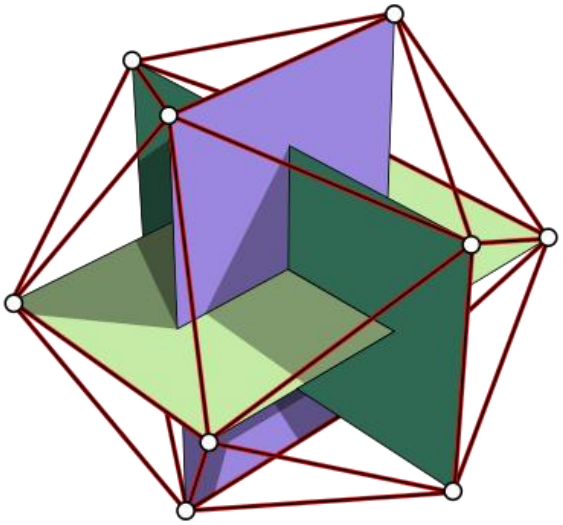

图 4-4-1 正二十面体构造图

在获得各个顶点的坐标后，即可以对正二十面体进行渲染。与二维的类似，渲染依旧是三角形作为基本单元。因此，需要将 12 个顶点依照实际组成三角形的情况进行分类。此时，我们可以完成当前正二十面体在局部空间的工作。

接下来，我们需要将局部空间的坐标依次转移到世界空间、观察空间、裁剪空间、屏幕空间。依据上文关于对应空间的介绍，我们需要构建三个矩阵，分别为 Model、View、Projection。他们依次执行下述任务：Model 负责对局部空间内指定物体的坐标进行转换，可能有旋转、平移、缩放等；View 负责将世界空间所有的物体坐标根据摄影机的位置进行转换；Projection 负责将观察空间的坐标再次进行转换，转换方式依照于投影方式，常见的方式有正交投影、透视投影。

由于在绘制三维图形的过程中，对坐标系统进行了多次转换，因此需要对顶点着色器与片段着色器进行一定的修改。

为了使正二十面的立体感更明显，我们还加入了纹理与光照。

- 纹理通过调用 stb_image 库来实现，stb_image 是一个轻量级的 C/C++ 图像加载库，可以读取 JPEG、PNG、BMP 等各种常见的图像格式。我们利用这个库来读取 JPEG 格式的纹理图片数据 data，结合 OpenGL 库中的函数 glTexImage2D 和 glGenerateMipmap 来实现给模型贴纹理。glTexImage2D 可以创建一个 2D 纹理并将数据加载进入纹理中，而 glGenerateMipmap 可以为当前绑定的纹理生成多个不同分辨率版本的 mipmaps，用于在纹理远处的细节呈现上更加平滑。
- 光照的设置需要首先指定光源的位置与光源的颜色，这里采用冯氏光照模型(Phong Lighting Model)，然后在着色器中分别设置三种不同的关照：漫反射、镜面反射、环境光。最后使用漫反射贴图（Diffuse Map）为纹理添加关照反射。

4.5 三维变换

实现三维图形绘制之后，我们只需要在模型变换时进行对应的坐标转化即可，通常这个过程是对 Model 矩阵处理而实现的。

4.5.1 位移变换

调用 glm::translate 函数，传入类型 glm::mat4 的原始四维向量和类型为 glm::vec3 位移向量 cubePositions 即可得到平移过后的图。在这个实验中，我们选择在 while 循环中利用 glfwGetTime()循环改变 cubePositions 的第三个值，即 z 轴的值，达到动画的效果。

4.5.2 旋转变换

调用 model = glm::rotat，传入类型 glm::mat4 的原始四维向量、类型为 float 的旋转角度和类型为 glm::vec3 旋转的中心轴向量。同样的，我们在 while 循环中利用 glfwGetTime()循环改变旋转角度，达到动画的效果。

4.5.3 缩放变换

调用 glm::scale 函数。我们需要传入类型 glm::mat4 的原始四维向量和类型为 glm::vec3 的三个轴的变换比例。同样的，我们在 while 循环中利用 glfwGetTime()循环改变缩放比例，达到动画的效果。

4.6 三维组合变换

在上述三维变换中已经讲述了 translate、scale、rotate 三个函数的使用，这里不再赘述。在组合变换的调用顺序为：rotate->translate->scale。注意的是，我们首先将其缩放，然后再将其移到固定坐标点，最后绕着(-1,-1,-1)轴旋转。我们对距离、旋转角度、缩放比例同时进行变换，生成了一个较为神奇的效果。

4.7 导入模型

4.7.1 模型加载

Assimp 是一个非常流行的模型导入库，它是 **Open Asset Import Library**（开放的资产导入库）的缩写。Assimp 能够导入很多种不同的模型文件格式（并也能够导出部

分的格式)，它会将所有的模型数据加载至 Assimp 的通用数据结构中。我们调用 Assimp 库尝试加载 pmx 模型。

4.7.2 模型处理

当我们导入模型时，遇见如下问题：

1. 模型贴图丢失
2. pmx 模型导入后脸部出现贴图错误

针对问题 1，我们排查后发现，可能是模型 texture 文件存在中文，而 C++Assimp 在加载含中文路径的模型时可能会出现路径乱码导致材质无法对应的问题。为解决该问题，由于我们要加载的模型时.pmx 格式，所以我们使用了 pmxEditor 软件对材质进行了重命名与重映射，使得 Assimp 成功加载模型。

针对问题 2，我们猜测可能是 Assimp 对 pmx 的支持并不高导致的，所以我们使用 pmxEditor 将 pmx 模型转为 obj 模型，再次使用 Assimp 加载，模型正常导入并且显示正常。

4.7.3 模型渲染

使用 Assimp 导入一个模型的时候，它通常会将整个模型加载进一个场景(Scene)对象，它会包含导入的模型/场景中的所有数据。Assimp 会将场景载入为一系列的节点(Node)，每个节点包含了场景对象中所储存数据的索引，每个节点都可以有任意数量的子节点。

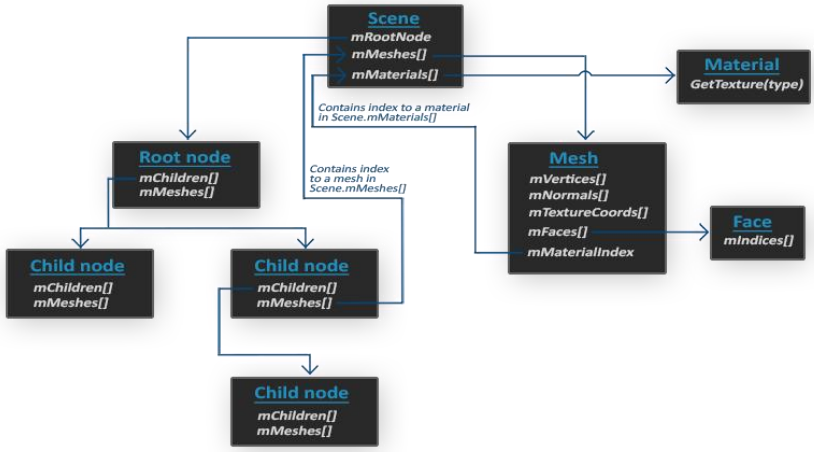

图 4-7-3-1 节点索引示意图

Mesh 对象本身包含了渲染所需要的所有相关数据，像是顶点位置、法向量、纹理坐标、面(Face)和物体的材质。为实现渲染模型，我们将物体加载到 Scene 对象后，遍历 Scene 节点得到多个 Mesh 对象，之后渲染每一个 Mesh 对象即可将模型完整渲染出来。

我们将 Mesh 封装成类，为了渲染一个 Mesh，我们首先使用 glGenVertexArrays、glGenBuffers、glBindVertexArray 等对 VAO、VBO、EBO 数据进行载入绑定，之后在绘制前对 Mesh 的多个纹理进行绑定，最后调用 glDrawElements 实现 Mesh 渲染。

4.7.4 模型光照

由于我们对模型渲染的认知还停留于表面，所以对于模型的光照我们并没有使用法线贴图等高级光照，我们采用最简单的 Phong Lighting Model 仅仅对模型实现了环境光、镜面反射光与漫反射光照。其中，环境光由 ambientStrength * texture(texture_diffuse1, TexCoords).rgb 计算得出，漫反射光照由 diff * light.diffuse * texture(texture_diffuse1, TexCoords).rgb 计算得出，镜面反射光照由 specularStrength * spec * texture(texture_diffuse1, TexCoords).rgb 计算得出。

5 实验结果

5.1 二维图形绘制

编译并运行文件，即可观察到一个三角形被绘制在屏幕中心，三角形有三种不同的颜色。

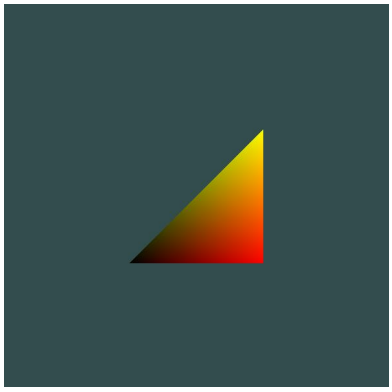

图 5-1-1 三角形示意图

5.2 二维变换

按“wsad”键即可控制图形进行上下左右移动，即位移变换。

图 5-2-1 位移变换示意图

按“qe”键即可控制图形进行逆时针或顺时针旋转，即旋转变换。

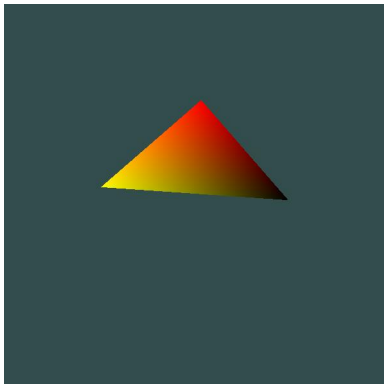

图 5-2-2 旋转变换示意图

按“←→”即可控制图形在 x 轴方向上进行缩放，按“↑↓”即可控制图像在 y 轴方向上进行缩放，即缩放变换。

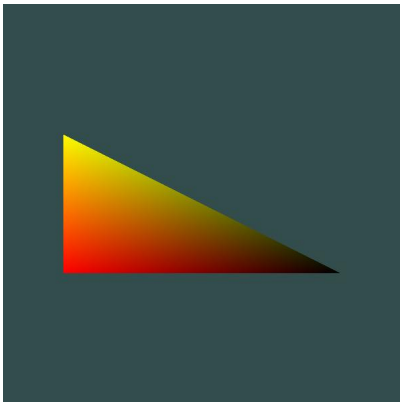

图 5-2-3 缩放变换示意图

按左“Ctrl”键即可控制图形在 x 轴上进行反射变换，按左“ALT”键即可控制图形在 y 轴上进行反射变换。

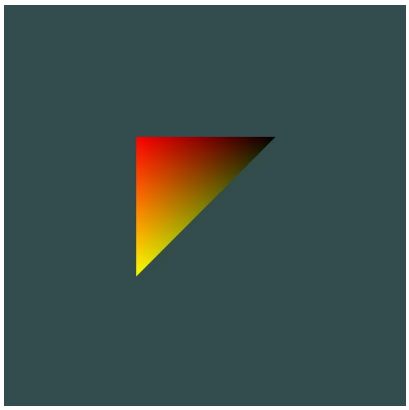

图 5-2-4 反射变换示意图

按“jk”键即可控制图形在 x 轴上进行切变，按“io 键”即可控制图形在 y 轴上进行切变。

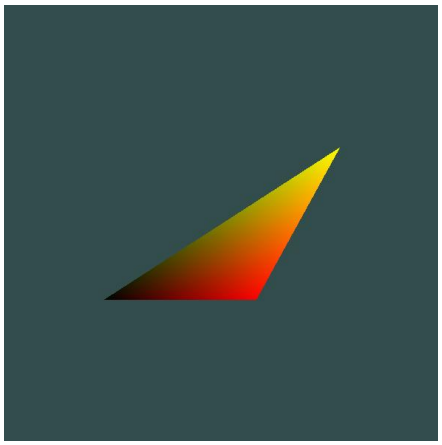

图 5-2-5 切变示意图

5.3 二维组合变换

以上操作均可以随意组合使用，即完成组合变换。

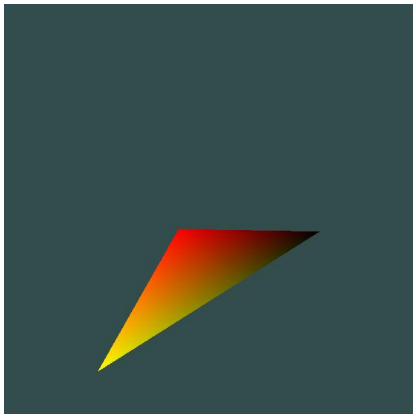

图 5-2-6 组合变换示意图

5.4 三维图形绘制

编译文件，一个有着紫色纹理的正二十面体出现在窗口的中心，二十面体的中心位于坐标系中心。可以根据鼠标和键盘完成对二十面体的移动。同时根据视角的不同，可以看到反光量的不同，如图 5-4-1~5-4-2 所示。

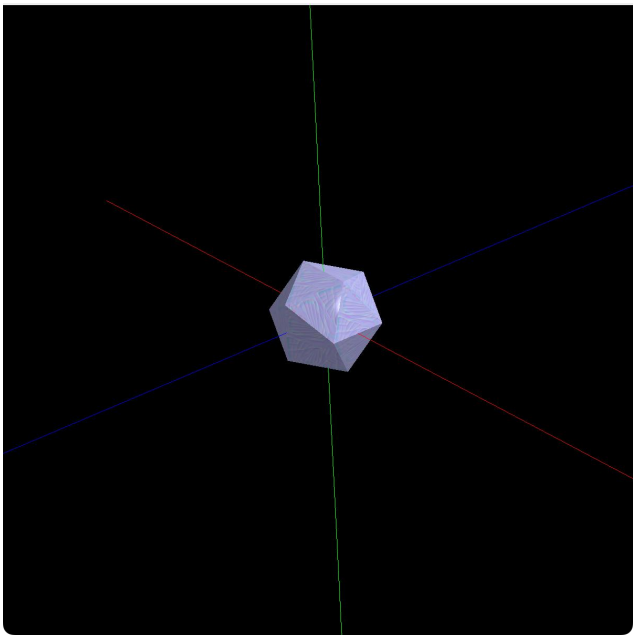

图 5-4-1 正交投影视图

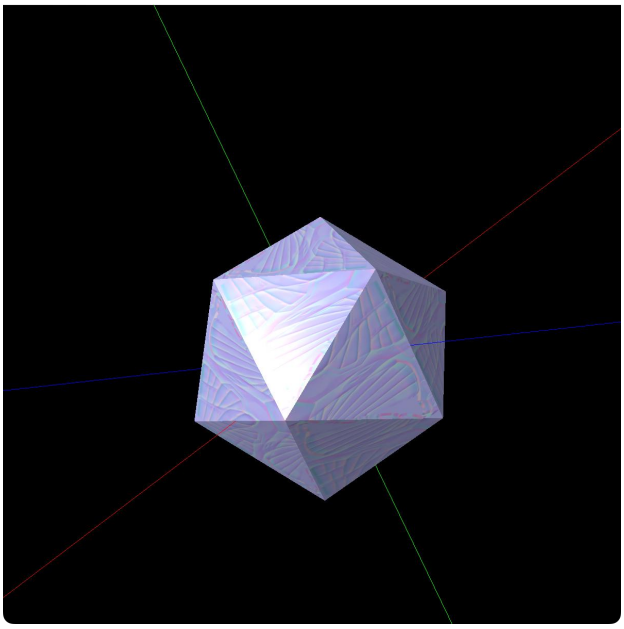

图 5-4-2 透视投影视图

5.5 三维变换

5.5.1 平移变换

编译文件并执行，会观察到一个有着紫色纹理的正二十面体出现在窗口的中心，并沿着(1,1,1)方向做周期运动。可以根据鼠标和键盘完成对二十面体的移动。

同时根据视角的不同，可以看到反光量的不同。如图 5-5-1-1 所示。

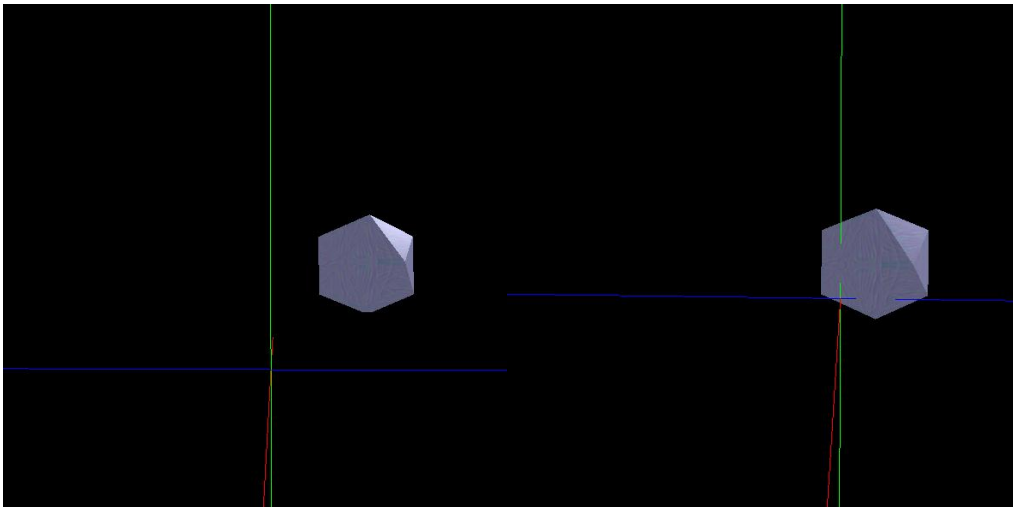

图 5-5-1-1 平移对比图

5.5.2 旋转变换

编译 rotateX、rotateY、rotateZ 三个文件并执行，会分别观察到一个有着紫色纹理的正二十面体出现在窗口的中心，并沿着 x/y/z 轴方向做旋转。可以根据鼠标和键盘完成对二十面体的移动。同时根据视角的不同，可以看到反光量的不同。如图 5-5-2-1~5-5-2-3。

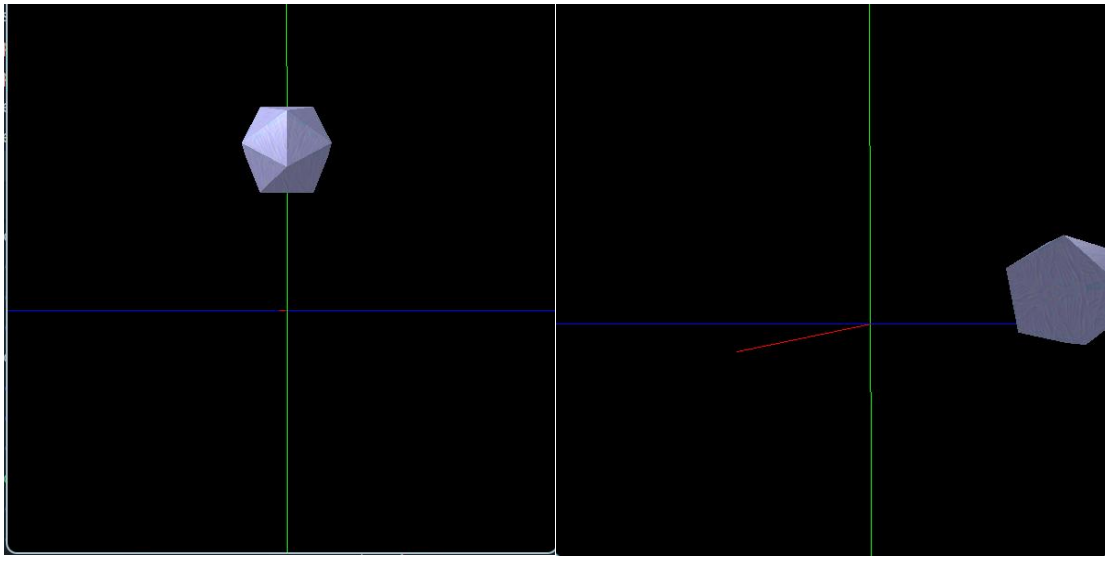

图 5-5-2-1 X 轴旋转图

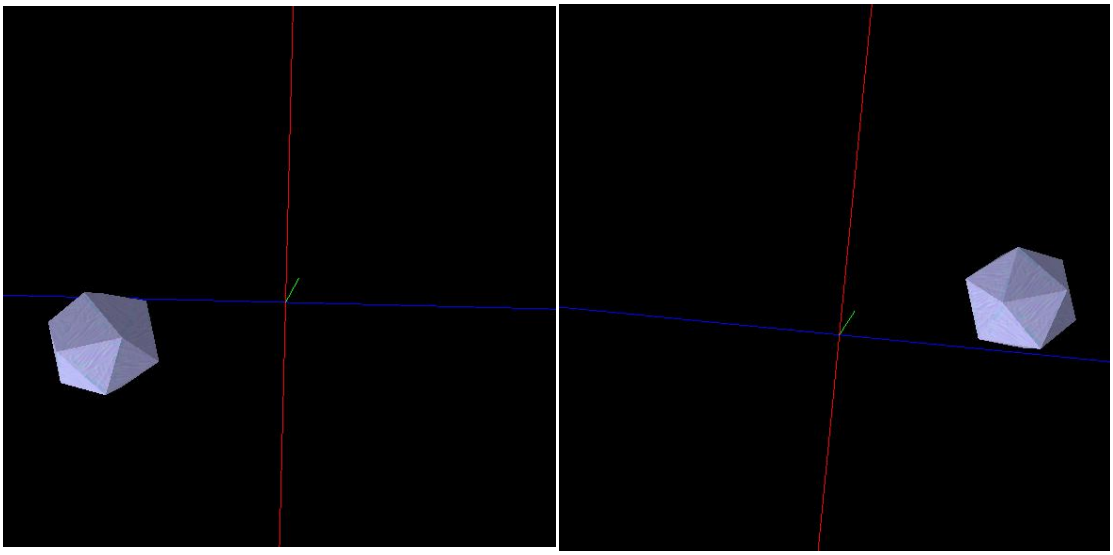

图 5-5-2-2 Y 轴旋转图

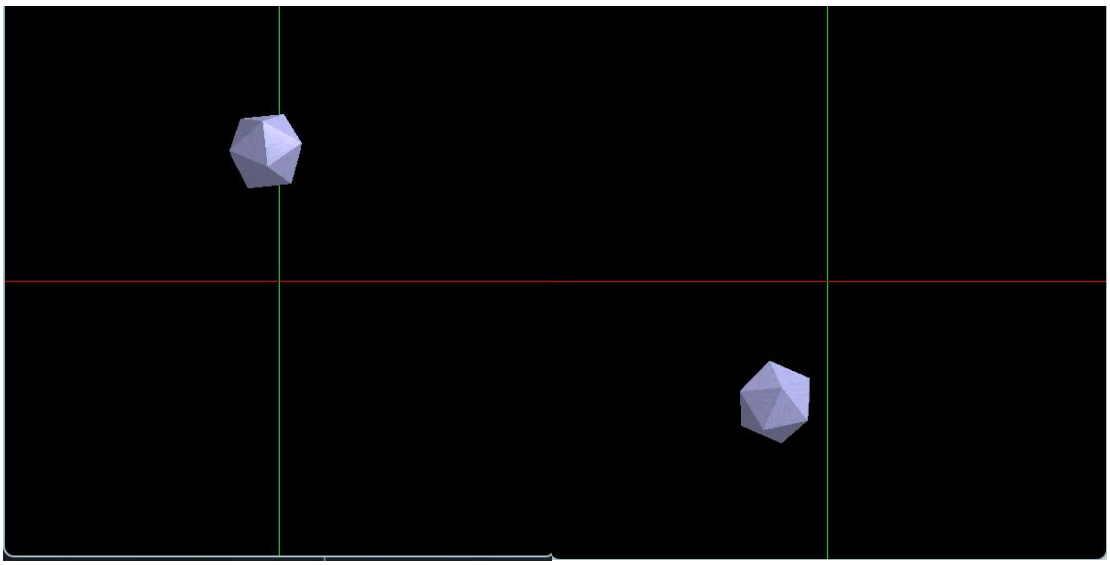

图 5-5-2-3 Z 轴旋转图

5.5.3 缩放变换

编译文件，一个有着紫色纹理的正二十面体出现在窗口的中心，二十面体的中心位于坐标系中心并不断继续缩放。可以根据鼠标和键盘完成对二十面体的移动。同时根据视角的不同，可以看到反光量的不同，如图 5-5-3-1 所示。

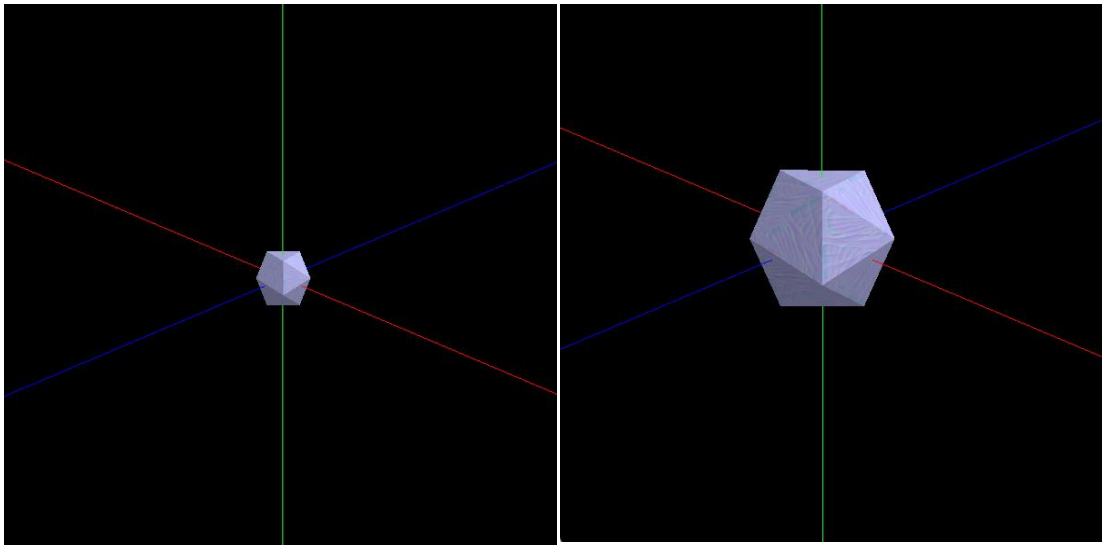

图 5-2-3-1 缩放变换图

5.6 三维组合变换

编译文件，一个有着紫色纹理的正二十面体出现在窗口的中心，绕着 z 轴进行旋转，旋转半径随时间变化，同时也在进行缩放变换，缩放比例也随时间变换。可以根据鼠标和键盘完成对二十面体的移动。同时根据视角的不同，可以看到反光量的不同，如图 5-6-1-1 所示。

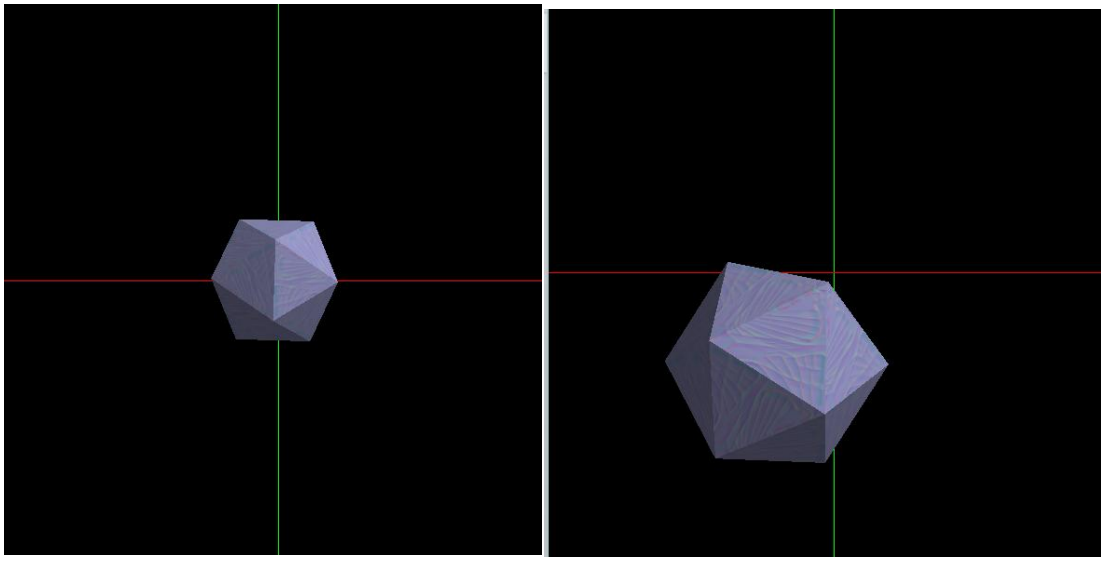

图 5-6-1-1 组合变换图

5.7 投影的切换

编译文件 3d 中的任意文件，按下'B'即可以实现投影的切换，如图 5-7-1~5-7-2。

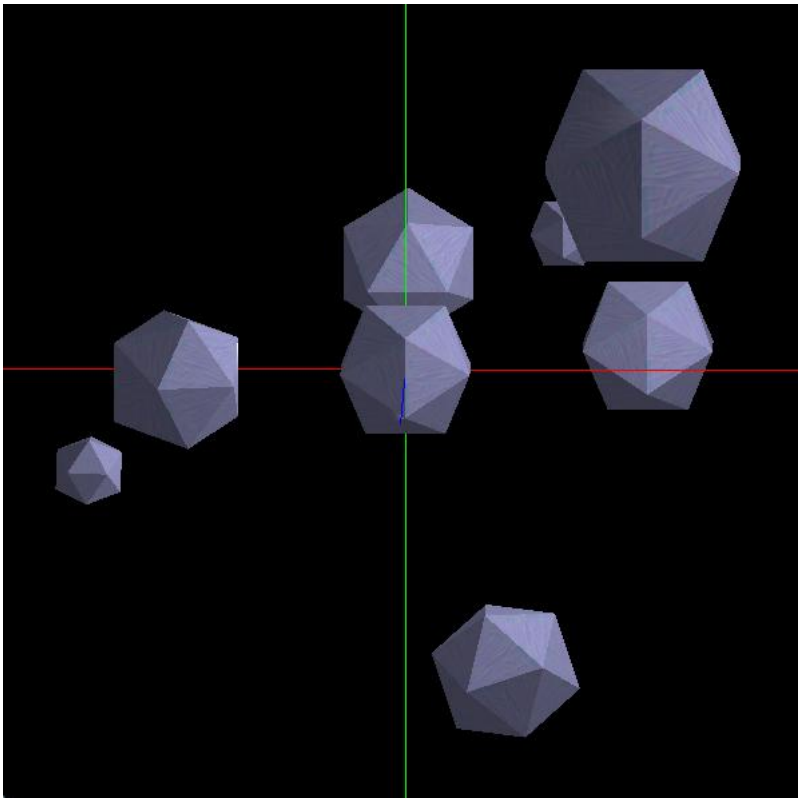

图 5-7-1 正交投影图

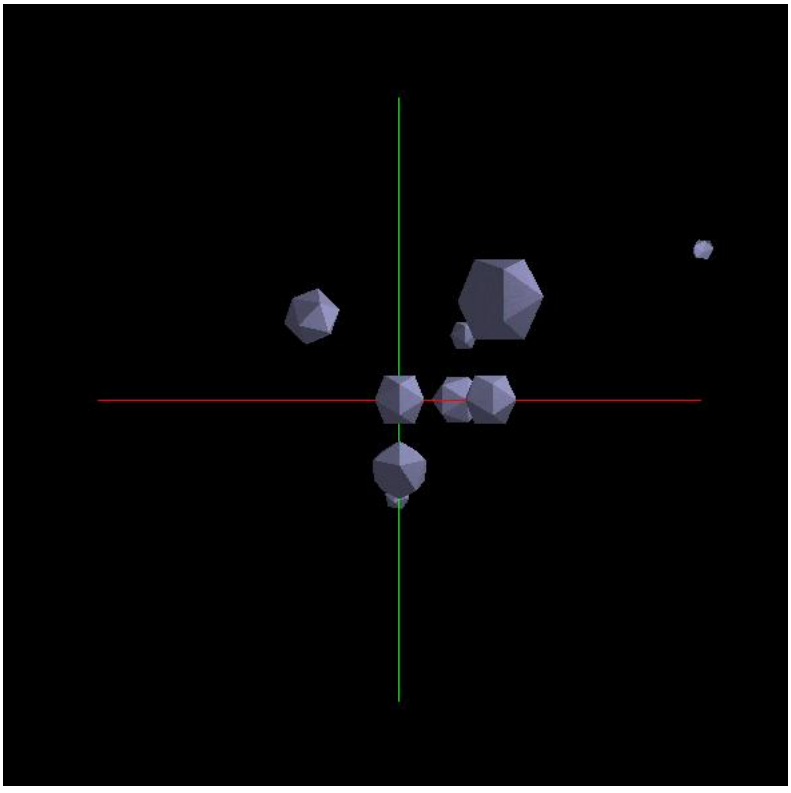

图 5-7-2 透视投影图

5.8 模型导入

编译文件，旋转视角可以看到 3D 模型已经被成功渲染，默认模式为仅漫反射光照的渲染。如下图。

图 5-8-1 模型渲染

使用 WASD 与鼠标可以联合操纵摄像机位置与视角，按下键盘 L 键切换至点光源光照模式，按下键盘 C 键切换回漫反射光照模式。下图为漫反射光照模式与点光源光照模式对比。

图 5-8-2 点光源渲染

6 实验总结与感想

何宇航：通过本次实验，首先学习了如何在本机上使用 OpenGL 的相关接口，通过不同的接口绘制出二维、三维图形。对如何使用矩阵进行模型矩阵的变换有了更深刻的理解，同时也对较为抽象的视图变换、投影变换从 glm 提供的方法角度有了新的理解。在绘制三维图形中，我尝试了正方体、正二十面体。为了使得正二十面体更具有立体感，我又进一步学习并加入了光源。为了计算光源，我需要计算正二十面体每一个面的法向量，这是一个繁琐的过程，或许在深入的学习后，会有更简单的方式。此外，传统绘制立方体的过程中需要计算出每一个点的坐标，但实际上，更复杂的规则立方体需要使用复杂的方程求解，不规则图形则几乎不可能求解，在未来学习中，或许可以学习其他建模软件，如 Blender、Maya 等，进而完成复杂图形的顶点、法向量的计算与绘制。

张鲲鹏：这次图形学实验让我深入学习了 OpenGL 相关库的基本使用。通过实验操作，我更加深入地理解了二维和三维变换。在实验中，我学会了如何使用 OpenGL 进行简单的图形绘制，包括点、线、多边形等基本图元的绘制。我还学习了如何进行基本的坐标变换，包括平移、旋转、缩放等操作，以及如何使用投

影变换将三维物体投影到二维屏幕上。这次实验不仅让我掌握了 OpenGL 的基本操作，还加深了我对图形学中基本概念的理解。

黄一圃：总的来说，这次实验让我对 OpenGL 的基本概念和使用方法有了更深入的理解。我学习了如何创建窗口，如何处理用户输入，如何加载和使用着色器，以及如何配置 OpenGL 的状态。这些知识作为图形编程的基础，帮助我成功入门了这一学科。

石昊原：在本次流程中，我完整走完了 OpenGL 绘制 2D 图形、着色、纹理、变换、坐标系统、摄像机、光照、材质与模型渲染的流程，这极大加深了我对 OpenGL 的理解。并且学习接触了更加复杂光照的相关理论与知识，如阴影与法线贴图等，希望在下一次实验中可以使用这些知识来实现更好的渲染效果。

钱波：本次实验中，我了解了 OpenGL 的相关库的配置。在配置过程中发现并解决了 MingW 版本过低等一些 C++环境问题。在 OpengGL 的学习过程中，我学会了创建基本图形的相关流程以及图形在 OpenGL 上的渲染流程。在二维变换的学习过程中，我学会了着色器的应用以及各种基本变换的简单应用。在三维变换的学习过程中，我学会了纹理的使用，以及 3 维图形的渲染和基础变换，更加加深了我对 glm 库的运用。希望下一次实验可以学习到更多的知识。

文泽：在本次实验中，我了解了 OpenGL 的相关配置及其使用，学会了如何使用 OpenGL 相关库进行图形的绘制，渲染。同时经过本次实验，对图形的二维变换，三维变换，都有了更深刻的体会，并学会了光照的相关知识。希望在下一次实验中，能够运用这些知识实现更好的效果。

7 实验分工

7.1 工作量

何宇航（组长）21301095 软件 2104 **20%**

钱波 21301102 软件 2104 **20%**

石昊原 21301103 软件 2104 **20%**

黄一圃 21301096 软件 2104 **20%**

文泽 21301108 软件 2104 **10%**

张鲲鹏 21231303 软件 2104 **10%**

7.2 具体分工

何宇航：主要负责正二十面体的绘制，三维图形的键盘控制，鼠标控制，正交、透视投影的切换，光照的计算等。文档部分负责实验目的、实验环境、着色器、三维图形变换原理、三维图形绘制等部分内容。

钱波：主要负责三维图形的变换，包括平移、缩放、旋转以及组合变换。负责三维图形的纹理以及最后的整合。文档部分负责实验原理中三维图形变换原理的坐标系统以及实验内容的三维变换部分。

黄一圃：主要负责二维图形的变换，二维展示的键盘、鼠标控制，包括位移、旋转、缩放、反射、切变等。文档部分负责二维变换的实验内容。

石昊原：主要负责模型导入、模型光照，模型的键盘、鼠标控制，光照模式切换等。文档部分负责二维变换的实验原理与模型导入部分的实验原理、实验结果。

文泽：主要负责文档排版与校对，实验结果验证。

张鲲鹏：主要负责文档排版与校对，实验结果验证。