实验3.1 相机定位和投影

1. 实验内容
2. 了解OpenGL中相机的模型视图变换的基本原理
3. 掌握OpenGL中相机观察变换矩阵的推导
4. 掌握在OpenGL中实现相机观察变换
5. 了解OpenGL中正交投影和透视投影变换
6. 了解在OpenGL中实现正交投影和透视投影变换。
7. 理论背景
8. 齐次坐标

首先回顾一下齐次坐标的概念，详细内容请参考实验2.3。

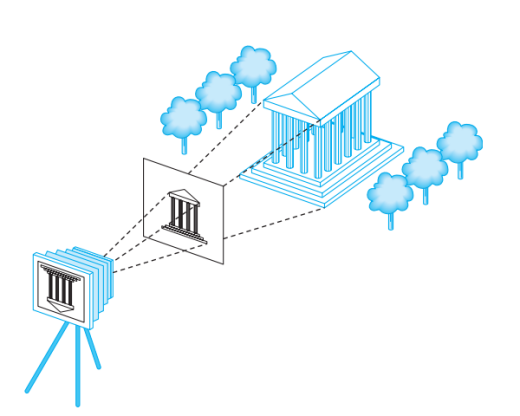
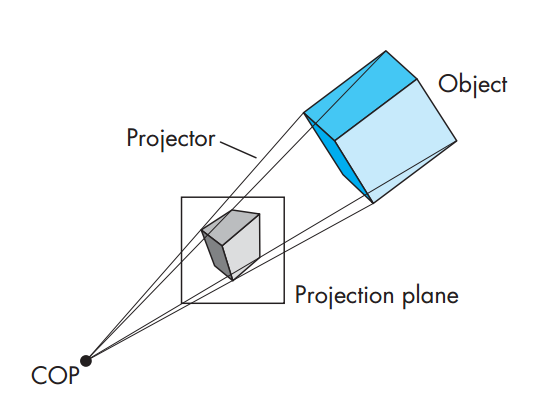
3维数据可以通过3维向量与3×3矩阵的乘法操作，来完成缩放和旋转的线性变换，也就是说三维空间的缩放矩阵和旋转矩阵都可以采用3维方阵表示。但是对3维坐标的平移是无法通过这样类似的操作完成，此时我们还需要额外的一个向量，将三维空间中的点移动到另外的位置，如下表示。



这里将原始点加上第四个维度，就是齐次坐标的表示。三维空间中的点的坐标通过将齐次坐标中前三维度的坐标值除以最后一个维度值得到。在后面章节中将会看到，齐次坐标多出的第四分量是用来实现透视投影变换的

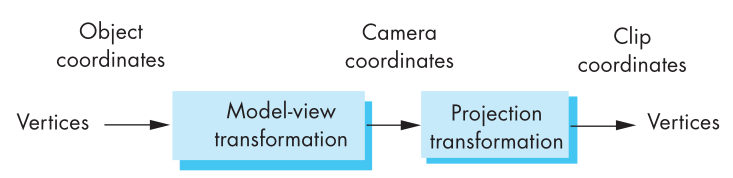
1. OpenGL观察变换

计算机图形学中的观察是建立在虚拟照相机模型的基础上，如下左图所示。而下右图则是对虚拟照相机模型的抽象表示，其本质上都包含有相同的元素：对象、观察者、投影线和投影平面。投影线相交于投影中心（COP：Center of Projection），而COP对应于人眼或者相机镜头，从COP点发出投影光线将物体投影在投影平面上而成像。所以在虚拟照相机模型中，如果需要正确的表现出投影对象在投影平面上的成像效果，我们必须首先确定相机的位置和方向。

在前面的章节中，确定了这样一个事实：只要顶点着色器输出的顶点位于裁剪体（或视见体）内部，那么这些顶点会被送入绘制流水线后面的光栅化模块。如果我们加入自定义的相机姿态，那么对相机的控制则尤为重要，因为相机的位置和朝向将决定怎么发出投影射线而将物体投影到投影平面上。

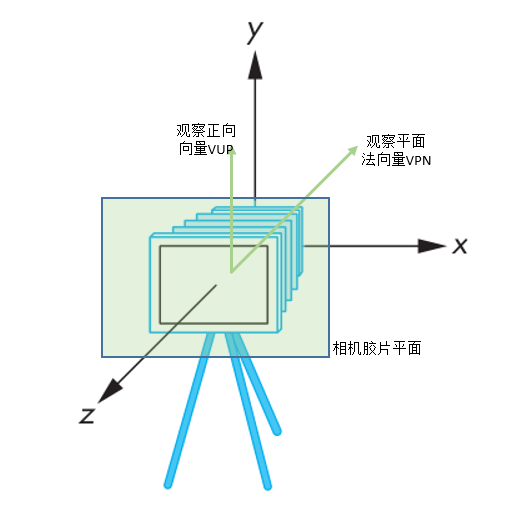
为了使得观察过程更加灵活，我们把它分解成两个基本的操作。首先，必须设置照相机的位置和方向，该操作由模-视变换来完成，当顶点经过该变换之后将位于相机坐标系中。然后是使用投影变换，也就是说把指定的投影（正交投影或透视投影）应用到顶点上。并将默认的视见体内部对象变换到指定的裁剪立方体的内部，整个过程如下图所示。



这一节中我们主要考虑相机的观察变换。所谓变换就是设置一个矩阵，然后乘以三维顶点的坐标（考虑齐次坐标系），得到变换之后的坐标值。正如上述，模-视变换将顶点变换到相机坐标系，所以相机的观察变换必须要包含相机的位置和朝向等信息。

模-视变换是建模变换和观察变换的级联，通常建模变换矩阵是将对象变换到世界坐标，而观察变换将世界坐标转换到相机坐标。一般来说，建模变换矩阵是单位阵，我们不需要额外处理。即：

而对于照相机来说，其初始方向通常指向z轴负方向，这样才能看见位于相机前方的模型。考虑一个位于原点的对象，由于照相机初始位置也在原点，方向指向z轴负方向，如果该对象需要在z轴正方向的投影平面（在OpenGL中通常是近裁剪平面）上成像，那么必须要将相机沿着z轴正方向往后移动一定距离。相机移动之后的位置称为**观察参考点（VRP，View Reference Pint）**。此时，我们考虑采用**规范化变换（Normalization Transformation）**来表示相机坐标系并指定相机的位置，可以分为两个部分：**观察平面法向量（VPN：View-plane Normal）和观察正向向量（VUP：View-up Vector）**。其中VPN指定了相机的胶片平面，而平面是由其法向量决定的。如果只是指定了VPN，相机还可以绕VPN旋转，此时我们再加上VUP，就能完全固定一个相机的位置了。



所以通过指定VRP，VPN和VUP可以定位照相机。但通常在OpenGL中，我们更倾向于在对象坐标系下，采用如下图所示的定位方法，将相机放置于的位置上称为视点，相机指向的另一个点称为参考点。这两个点就确定了VPN和VRP，即：

归一化得到：

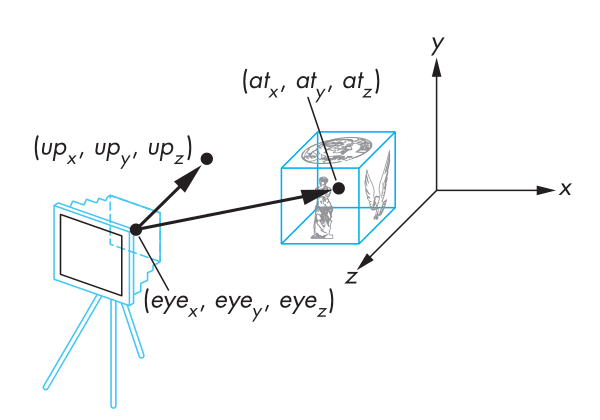
然后通过VUP和VPN生成与它们都垂直的方向向量：

最后再计算得到VUP在照相机胶片平面上的投影：

这样通过采用u,n,v即可定义出相机的观察变换矩阵（其实就是相机的局部坐标系）如下：

由于在最开始，我们还需要将相机从坐标原点移动到视点，所以还需要一个平移矩阵如下：

所以最终的相机观察矩阵viewMatrix为：

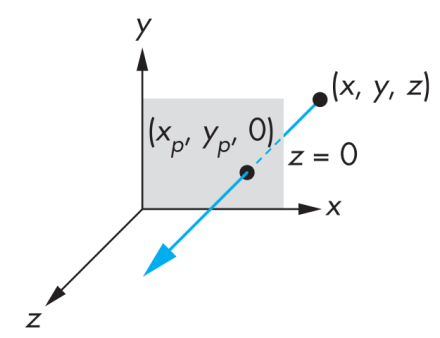


1. 投影变换

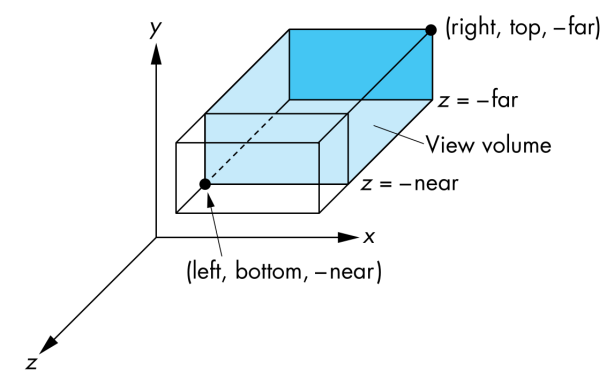
在经过三维物体的模-视变换后，场景中的三维物体即被放在了相机能够观察到的位置。而投影变换的目的则是定义一个视景体（View Volume），使得视景体外多余的部分被裁减掉，最终进入到投影平面上的只是视景体内的部分。投影包含正交投影（Orthographic Projection）和透视投影（Perspective Project）两种。

* 1. 正交投影

正交投影的投影线垂直于观察平面，如果用照相机实现正交投影，那么这个照相机的胶片平面应该平行于镜头。如下图所示，投影平面是。

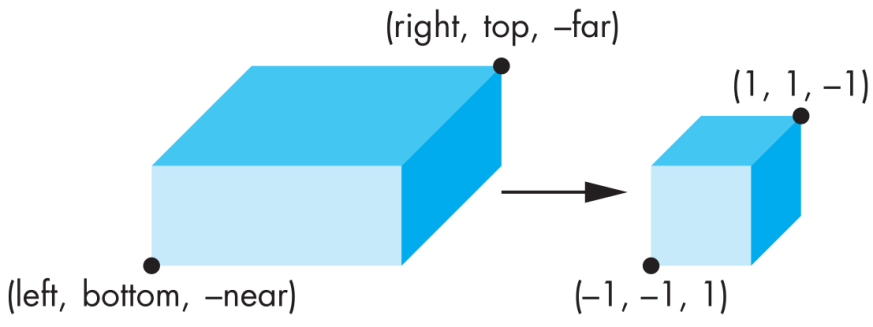


在OpenGL中，通常使用的正交投影是定义在一个平行六面体的视景体（或者说是裁剪体）中，如下图所示，该六面体由六个参数决定，分别为左右裁剪平面（left和right），上下裁剪平面（top和bottom），远近裁剪平面（near和far）。需要注意的是，这些参数的定义都是在相机坐标系下。举例来说，远近裁剪平面相当于是在z轴方向离相机的距离。



但是，在OpenGL的渲染管线中定义了一个标准视景体如下，

也就是说，在渲染的最后过程中，我们需要将上述定义的正交投影视景体变换到该标准视景体中，使用的方法是通过平移和旋转变换将相机坐标系下经过裁剪的顶点变换到默认的标准视景体下，这个处理过程称为**投影规范化（Projection Normalization）**，如下图所示。

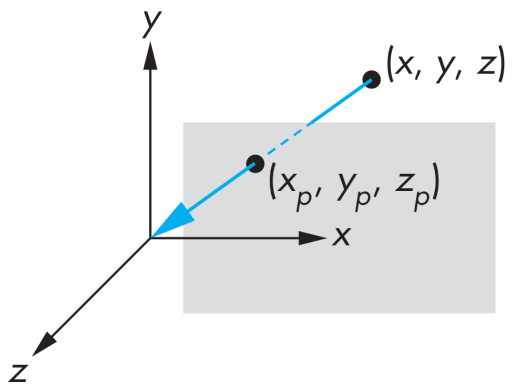


而使用的投影矩阵如下，具体推导过程请参考书本相应章节。

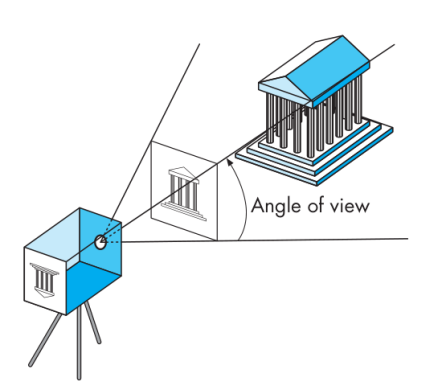
****

* 1. 透视投影

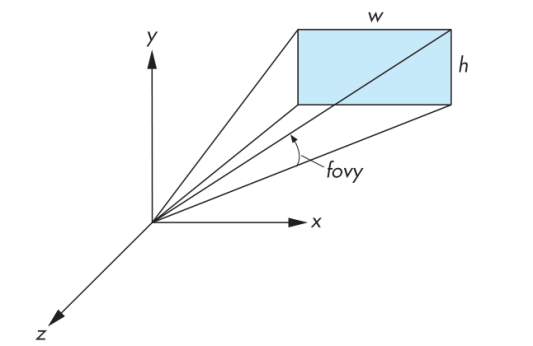
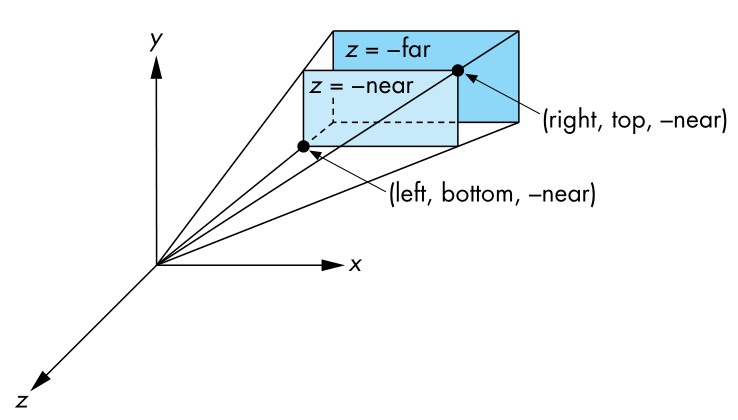
在真实世界中的投影方式是透视投影，即满足人类视觉基本感知原理：近大远小。如下图所示，将坐标原点作为投影中心，任意顶点 经过投影平面 之后均相交于坐标原点，那么基于比例关系即可求出在投影平面上的投影点坐标。



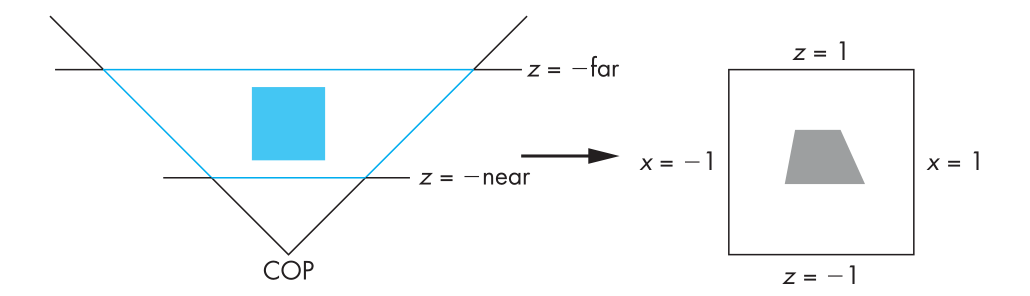
同样，对于透视投影，我们也需要设置一个视景体来裁剪三维物体如下，也就是说在视景体内部的三维物体才能被投影到投影平面上，剩下超出的部分则被裁减掉。在透视投影中，由于投影中心在理想情况下是一个点，所以形成的视景体构成一个截头椎体（Frustum）。



在OpenGL中，通常有两种定义透视投影视见体的方法，如下左图的类似于正交投影的棱台视见体和下右图利用视域（FoV，Field of View）定义的视见体。其中棱台视见体是由左右裁剪平面（left和right），上下裁剪平面（top和bottom），远近裁剪平面（near和far）。而视域视见体则由视角（Field of View），投影平面长宽比（aspect）和远近裁剪平面（near和far）决定。



另外，在透视投影中，同样需要执行投影规范化过程，如下图所示，将视景体内部的顶点变换到标准视景体范围内：



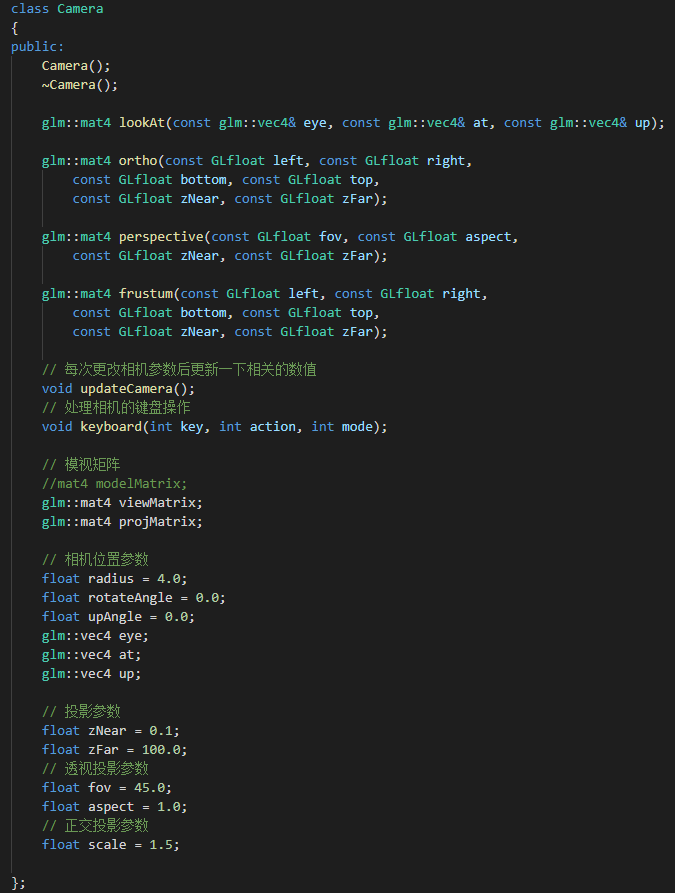
以透视投影的棱台视见体为例，其透视投影矩阵如下，具体推导过程请参考书本相应章节，这里不再赘述。**（请尝试推导出视域视景体的投影变换矩阵，并理解这两种视景体定义方式的等价性。）**

****

（注意和教材上第166页公式不一样，第三行第三列数值教材上的写错了。）

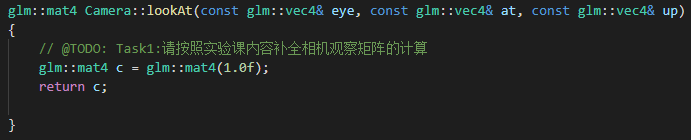
1. 代码讲解
2. Camera类

我们在include文件夹内创建了一个Camera.h文件，并定义了一个相机Camera类，里面声明了一些函数与相机参数，然后在Camera.cpp中实现这些函数。本次实验将要把未实现的投影矩阵函数完成，然后使用这些投影矩阵达到不同的物体观察效果。注意Camera类中的已经定义了投影的一些参数变量，在调用投影矩阵计算函数的时候可以使用这些变量。

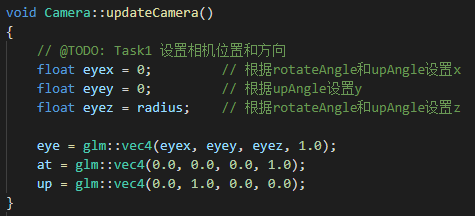


1. lookAt函数补全

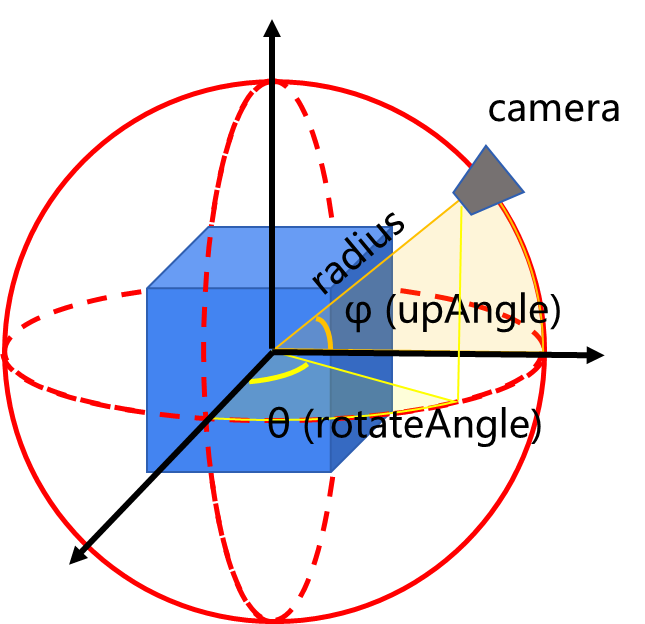
将camera.cpp中的lookAt函数补全，这个函数的计算涉及到归一化和叉乘计算，可以使用glm中自带的normalize和cross函数来实现。



在 updateCamera() 中根据角度与距离半径更新计算相机eye的位置，用于lookAt函数：



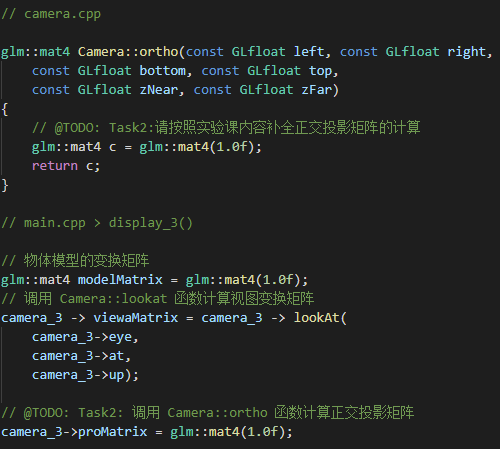
* 建议设置相机的方式: 球形轨迹
* 相机的参考点（at）:指向被观察物体的中心
* 相机的VUP方向：与世界坐标系y方向相同(方向朝上)



1. 实现正交投影

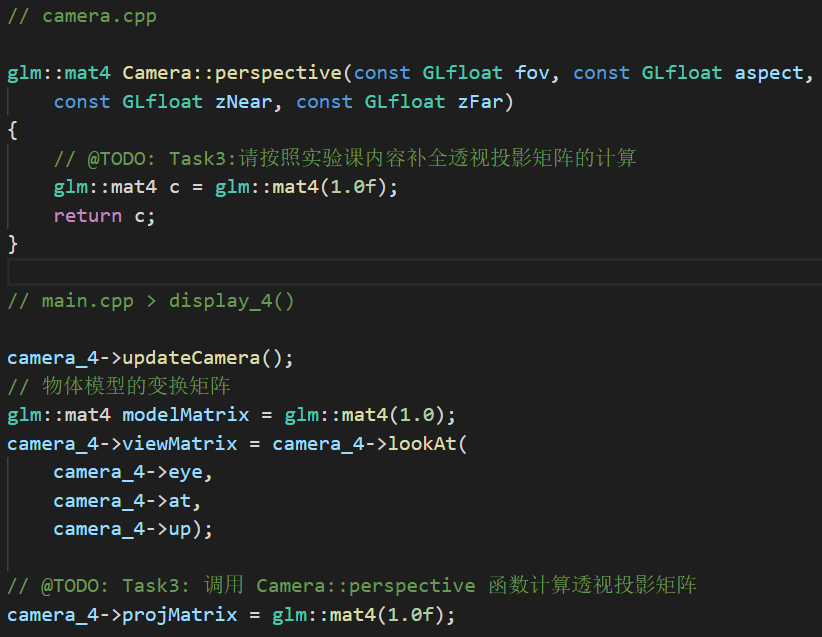
在camera.cpp中实现计算正交投影的ortho函数。

然后在main.cpp的display\_3调用ortho()生成正交投影矩阵。使用 Camera 中定义的投影参数 zNear 和 zFar 以及正交投影参数 scale 传入 Camera::ortho() 函数计算正交投影矩阵。



1. 实现透视投影

先实现计算透视投影的perspective函数，然后在main.cpp的display\_4调用perspective()生成透视投影矩阵。

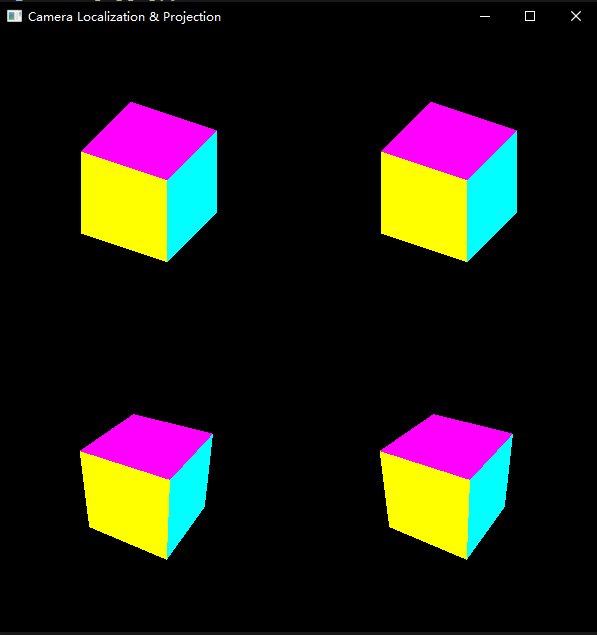


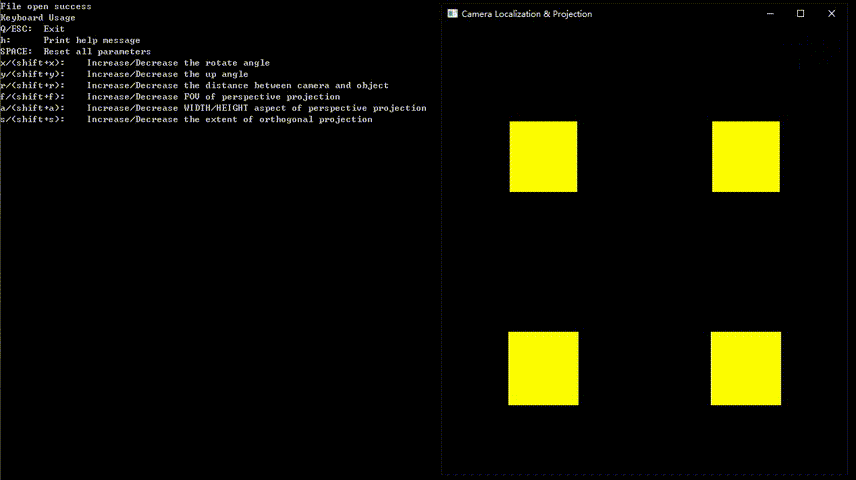
1. 实验内容
2. 实现模视变换，通过键盘控制，改变相机视点以观察不同参数对结果的影响
3. 具体修改内容如下：
   1. Task-1：在Camera.cpp中完善lookAt函数和updateCamera函数
   2. Task-2：在Camera.cpp中完善ortho函数，在main.cpp中完善display\_3函数
   3. Task-3：在Camera.cpp中完善perspective函数，在main.cpp中完善display\_4函数
4. 实现正交投影和视域指定透视投影的矩阵计算，观察两种投影方式绘制的结果

实验设计了四个窗口分别展示四种观察方式：

* 左上角窗口：使用模型变换矩阵display\_1()，已完成
* 左下角窗口：棱台指定视见体的透视投影display\_2()，待完成
* 右上角窗口：使用正交投影display\_3()，待完成
* 右下角窗口：视域指定视见体的透视投影display\_4()，待完成

完成实验后，上面为正交投影，下面为透视投影，最终效果应如下





1. 提交内容**【请各位务必严格按照以下要求执行】**
2. **程序代码**：本次实验提供参考代码，**程序运行窗口标题设为：“学号\_姓名\_实验3.1”**（用你自己的学号和姓名。如果出现乱码，可能是因为代码文件的字符格式问题，实验提供的文件字符编码为utf-8，改成GBK格式就不会出现乱码了）。最终提交代码中与实验内容相关部分必须写注释。
3. **上传格式**：按上述要求完成实验，一并提交源代码压缩包，压缩包名称为“学号\_姓名\_实验3.1”。
   1. 提交文件包括：源代码压缩包，命名格式均为“学号\_姓名\_实验3.1”。
   2. 源代码压缩包内要求有两个文件夹，一个为代码文件夹，命名为“code”，一个为可执行文件夹，命名为“exe”。
   3. 代码文件夹中只能包含代码和代码需要用到的资源文件（比如纹理图片、模型），其他由编辑器或者编译器创建项目时候生成的文件全部都不要加上，不清楚的同学可以询问助教。（参考实验一的要求）
   4. 可执行文件夹中，只包含可执行文件以及执行所需的动态库文件和资源文件等，要求可以直接点击该程序就可正常执行。
4. **截止时间：2024年10月29日 23:59分**