实验 5

实验名称 Transformation 变换 2

姓　　名：雷昱

学　　号：22920202204666

学　　院：信息学院

专　　业：软件工程

年　　级：2020 级

二〇二二年 **5** 月 **7** 日

建议阅读资料：

（1） 课件

（2） OpenGL 编程基础 （第三版）

学习要求：

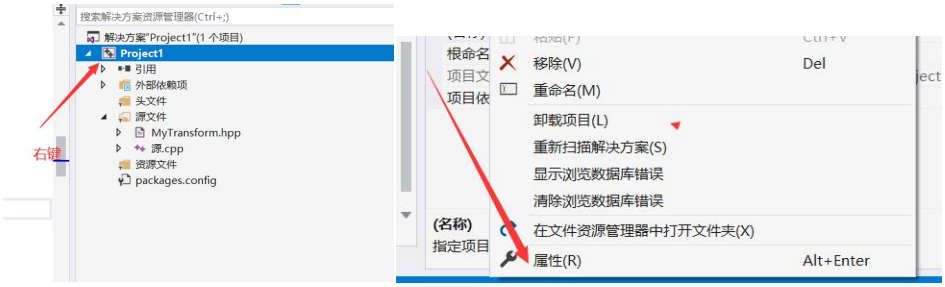
（1） 掌握 gluLookAt 对应的矩阵，能自己实现 OpenGL 中 gluLookAt 功能

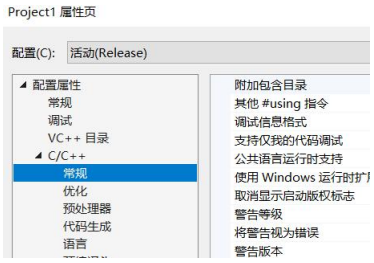
（2） 掌握 glOrtho 对应的矩阵，能自己实现 OpenGL 的 glOrtho 功能

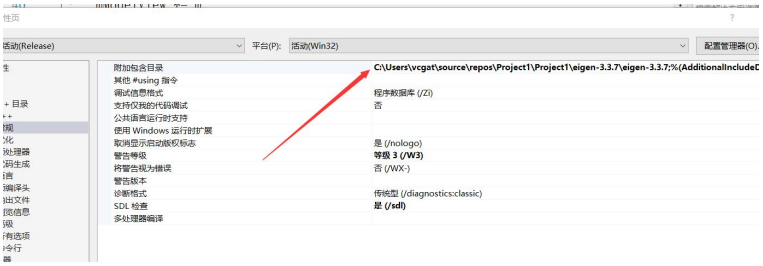
（3） 掌握 glViewport 对应的矩阵，能自己实现 OpenGL 的 glViewport 功能

Task1. 自行实现视点、投影、视口函数，请严格按照以下要求实现：

1. **创建空工程，配置 freeglut 库（之前已经教会大家）**
2. **配置 eigen 库（eigen 用于矩阵和向量的计算）。做法是：将 eigen 库解压，同时在刚才创建的工程中设置 eigen 库路径。右键点击工程->属性->C/C++ 常规 -> 将 eigen 所在目录填到“附加包含目录”这一栏。**

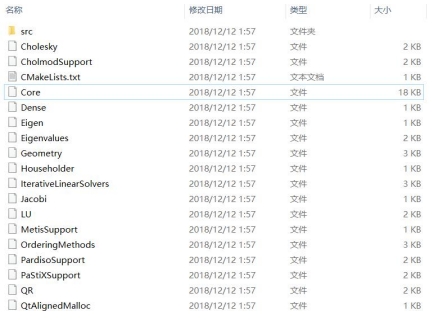






注意：上述路径是我的路径，请根据自己的解压位置填写路径。

应该定位到这个目录的上一级：



名称	修改日期	类型	大小
src	2018/12/12 1:57	文件夹	
Cholesky	2018/12/12 1:57	文件	2 KB
CholmodSupport	2018/12/12 1:57	文件	2 KB
CMakeLists.txt	2018/12/12 1:57	文本文档	1 KB
Core	2018/12/12 1:57	文件	18 KB
Dense	2018/12/12 1:57	文件	1 KB
Eigen	2018/12/12 1:57	文件	1 KB
Eigenvalues	2018/12/12 1:57	文件	2 KB
Geometry	2018/12/12 1:57	文件	3 KB
Householder	2018/12/12 1:57	文件	1 KB
IterativeLinearSolvers	2018/12/12 1:57	文件	3 KB
Jacobi	2018/12/12 1:57	文件	1 KB
LU	2018/12/12 1:57	文件	2 KB
MetisSupport	2018/12/12 1:57	文件	1 KB
OrderingMethods	2018/12/12 1:57	文件	3 KB
PardisoSupport	2018/12/12 1:57	文件	2 KB
PaStiXSupport	2018/12/12 1:57	文件	2 KB
QR	2018/12/12 1:57	文件	2 KB
QtAlignedMalloc	2018/12/12 1:57	文件	1 KB

3. 将本次实验给大家的两个代码文件加入工程中，它们是 **exp5.cpp** 和 **MyTransform.hpp**。在本次实验过程中，除非特别说明，不允许改动 **exp5.cpp** 的内容（也不必关心具体代码）。

4. 不做任何更改，如果配置正确，你应该能看到一个旋转的彩色三角形。

5. 在 **MyTransform.hpp** 中，请自行阅读并自己修改 **void Test()**函数。此函数仅用于让你熟悉 **eigen** 中矩阵和向量的用法，你可以自己随意尝试。我们提供了矩阵、向量设置和乘法的基本代码，其他更多函数可参考 **eigen** 官方文档。

   http://eigen.tuxfamily.org/dox/group__QuickRefPage.html

6. 完成 **MyTransform.hpp** 中如下函数：

   myFrustum, myOrtho, myLookAt, myViewPort

要求仅在代码中提示的地方填入/修改代码，其余地方均不许改动！

```
Eigen::Matrix4f myLookAt(float ex, float ey, float ez, float atx, float aty,
{
    Eigen::Matrix4f m;
    m.setZero();

    //请在下面空白处完成此函数


    m.setIdentity();//这句应该去掉


    //请在上面空白处完成此函数

    mModelView *= m;
    return m;
}
```

不许引入任何头文件，不得改动函数原型。程序正确结果应该是一个旋转的三角形动画。

如果配置成功，但还没开始填入代码，您应该看不到任何图形。

说明：请按照 myFrustum，myOrtho，myLookAt，myViewPort 这个顺序完成代码填写。

Task1.1 将 myFrustum，myLookAt，myViewPort 填写正确，运行 exp5.cpp 查看 myFrustum 的结果。

Task1.2 将 myOrtho，myLookAt，myViewPort 填写正确，在 exp5.cpp 中注释掉第 56 行，取消第 57 行的注释，观察 myOrtho 的效果；

所有代码都填写正确的话，您将看到两个三角形在窗口中做旋转。

标准程序结果已放在压缩包中。frustum.exe、ortho.exe 分别展示了 frustum.exe 和 ortho.exe 的标准结果。（注：若无法复现标准结果，可将自己的结果和代码私发给助教，助教将评估已有结果和发送的结果哪个正确。若能够成功辅助助教老师更正结果，可以给予优秀作业。）

本次 Task 需提交 2 个可执行文件，Task1.1 为使用 myFrustum 的结果，Task1.2 为使用 MyOrtho 的效果。

作业提交说明：

本次实验共有 2 个任务，分别是 Task1.1，Task1.2。

提交方式为：*将代码、可执行文件、实验报告*放到一个文件夹中，命名为“您的学号_姓名”，打包上传到 ftp 服务器中相应目录下。请确保提交的可执行文件可以运行（打分的重要依据）。每次实验作业的提交截止日期为下一次实验课前一天晚上。

特别说明：本次实验 2 个任务（Task1.1，Task1.2）都需要提交。您需要提供一个完整的文档（不限格式），逐条说明您完成每一条任务的具体情况。

实验步骤如下：

（1）myFrustum 函数实现

①透视变换转成正交变换

②矩阵相乘计算后 myFrustum 矩阵所示

$$\mathbf{P}=\mathbf{NSH}=\begin{bmatrix} \frac{2near}{right-left} & 0 & \frac{right+left}{right-left} & 0 \\ 0 & \frac{2near}{top-bottom} & \frac{top+bottom}{top-bottom} & 0 \\ 0 & 0 & -\frac{far+near}{far-near} & -\frac{2far*near}{far-near} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

③代码实现

```
m(0, 0) = 2 * n / (r - l); m(0, 2) = (r + l) / (r - l);
m(1, 1) = 2 * n / (t - b); m(1, 2) = -(t + b) / (t - b);
m(2, 2) = -(f + n) / (f - n); m(2, 3) = -2 * f * n / (f - n);
m(3, 2) = -1;
```

（2）myOrtho 函数实现

①先平移，把中心移到原点

$$\mathbf{T}(-(left+right)/2, -(bottom+top)/2, (near+far)/2))$$

②再缩放，进行缩放使视景体边长为 2

$$\mathbf{S}(2/(right-left), 2/(top-bottom), -2/(far-near))$$

③矩阵相乘计算后 myOrtho 矩阵所示

$$\mathbf{P}=\mathbf{ST}=\begin{bmatrix} \frac{2}{right-left} & 0 & 0 & -\frac{right+left}{right-left} \\ 0 & \frac{2}{top-bottom} & 0 & -\frac{top+bottom}{top-bottom} \\ 0 & 0 & \frac{-2}{far-near} & -\frac{far+near}{far-near} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

④代码实现

```
m(0, 0) = 2 * n / (r - l); m(0, 2) = (r + l) / (r - l);
m(1, 1) = 2 * n / (t - b); m(1, 2) = -(t + b) / (t - b);
m(2, 2) = -(f + n) / (f - n); m(2, 3) = -2 * f * n / (f - n);
m(3, 2) = -1;
```

（3）myLookAt 函数实现

①相机变换

②myLookAt 矩阵所示

- the eye position $\mathbf{e}$,
- the gaze direction $\mathbf{g}$,
- the view-up vector $\mathbf{t}$.

$$\mathbf{w} = -\frac{\mathbf{g}}{\|\mathbf{g}\|},$$

$$\mathbf{u} = \frac{\mathbf{t}\times\mathbf{w}}{\|\mathbf{t}\times\mathbf{w}\|},$$

$$\mathbf{v} = \mathbf{w}\times\mathbf{u}.$$

$$\mathbf{M}_{\text{cam}} = \begin{bmatrix} \mathbf{u} & \mathbf{v} & \mathbf{w} & \mathbf{e} \\ 0 & 0 & 0 & 1 \end{bmatrix}^{-1} = \begin{bmatrix} x_u & y_u & z_u & 0 \\ x_v & y_v & z_v & 0 \\ x_w & y_w & z_w & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -x_e \\ 0 & 1 & 0 & -y_e \\ 0 & 0 & 1 & -z_e \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

③代码实现

```
Eigen::Vector3f e(ex, ey, ez);
Eigen::Vector3f g(atx-ex, aty-ey, atz-ez);
Eigen::Vector3f t(upx, upy, upz);

Eigen::Vector3f w = -g / g.norm();
Eigen::Vector3f u = t.cross(w) / t.cross(w).norm();
Eigen::Vector3f v = w.cross(u);//差积
m << u, v, w, e,
    0, 0, 0, 1;//设置矩阵元素
m = m.inverse().eval();//矩阵转置
```

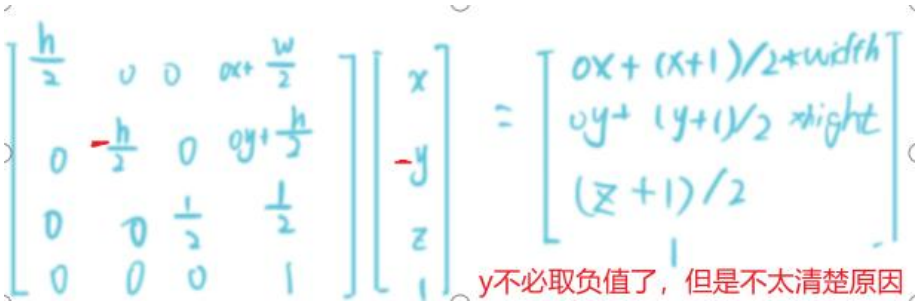
（4）myViewPort 函数实现

①根据规范化视景体计算

- 规范化视景体坐标(x,y,z)
- WinX = Ox + (x+1)/2*width
- WinY = Oy + (y+1)/2*height
- WinZ = (z+1)/2 (规范化到[0,1]之间)

注意此处 WinY 坐标和窗口系统的坐标朝向是反的

②myViewPort 矩阵所示

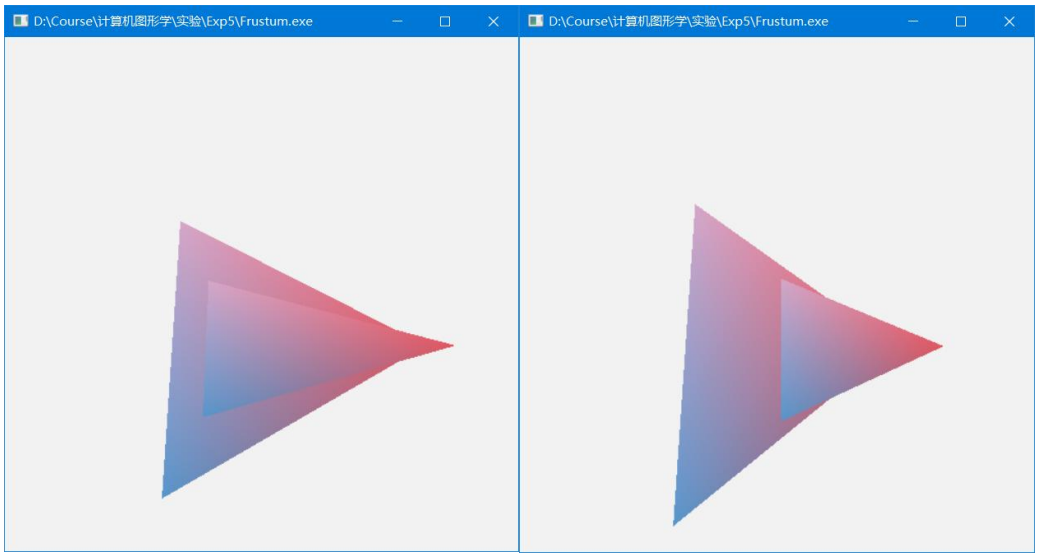


③代码实现

```
m(0, 0) = h / 2.0f;
m(0, 3) = x + w / 2.0f;
m(1, 1) = h / 2.0f;//注意此处y是负值,不知道为什么又不用了
m(1, 3) = y + h / 2.0f;
m(2, 2) = 1 / 2.0f;
m(2, 3) = 1 / 2.0f;
m(3, 3) = 1;
```

（5）结果展示

myFrustum



myOrtho