实验2：变换与投影

1 总览

到目前为止，我们已经学习了如何使用矩阵变换来排列二维或三维空间中的对象。所以现在是时候通过实现一些简单的变换矩阵来获得一些实际经验了。

本次作业的任务是填写一个旋转矩阵和一个透视投影矩阵。给定三维下三个点 $v_0(2.0, 0.0, -2.0), v_1(0.0, 2.0, -2.0), v_2(-2.0, 0.0, -2.0)$，你需要将这三个点的坐标变换为屏幕坐标并在屏幕上绘制出对应的线框三角形 (在代码框架中，我们已经提供了 *draw_triangle* 函数，所以你只需要去构建变换矩阵即可)。简而言之，我们需要进行模型、视图、投影、视口等变换来将三角形显示在屏幕上。在提供的代码框架中，我们留下了模型变换和投影变换的部分给你去完成。

如果你对上述概念有任何不清楚或疑问，请复习课堂笔记或询问助教。

以下是你需要在 **main.cpp** 中修改的函数（请不要修改任何的函数名和其他已经填写好的函数，并保证提交的代码是已经完成且能运行的）：

- **`get_model_matrix(float rotation_angle)`**：逐个元素地构建模型变换矩阵并返回该矩阵。在此函数中，你只需要实现三维中绕 z 轴旋转的变换矩阵，而不用处理平移与缩放。
- **`get_projection_matrix(float eye_fov, float aspect_ratio, float zNear, float zFar)`**：使用给定的参数逐个元素地构建透视投影矩阵并返回该矩阵。
- **`[Optional] main()`**：自行补充你所需的其他操作。

当你在上述函数中正确地构建了模型与投影矩阵，光栅化器会创建一个窗口显示出线框三角形。由于光栅化器是逐帧渲染与绘制的，所以你可以使用 **A** 和 **D** 键去将该三角形绕 z 轴旋转 (此处有一项提高作业，将三角形绕任意过原点的轴旋转)。当你按下 **Esc** 键时，窗口会关闭且程序终止。

另外，你也可以从命令行中运行该程序。你可以使用以下命令来运行和传递旋转角给程序，在这样的运行方式下，是不会生成任何的窗口，输出的结果图像会被存储在给定的文件中 (若未指定文件名，则默认存储在 **output.png** 中)。图像的存储位置在可执行文件旁，所以如果你的可执行文件是在 **build** 文件夹中，那么图像也会在该文件夹内。

命令行的使用命令如下:

```
./Rasterizer   //循环运行程序，创建一个窗口显示，且你可以
               //使用A键和D键旋转三角形。

./Rasterizer -r 20   //运行程序并将三角形旋转20度,然后将
                     //结果存在output.png中

./Rasterizer -r 20 image.png  //运行程序并将三角形旋转20度,
                              //然后将结果存在image.png中。
```

2 代码框架

在本次作业中，因为你并不需要去使用三角形类，所以你需要理解与修改的文件为:rasterizer.hpp 和 main.cpp。其中 **rasterizer.hpp** 文件作用是生成渲染器界面与绘制。

光栅化器类在该程序系统中起着重要的作用，其成员变量与函数如下。

成员变量:

- **`Matrix4f model, view, projection:`** 三个变换矩阵。
- **`vector<Vector3f> frame_buf:`** 帧缓冲对象，用于存储需要在屏幕上绘制的颜色数据。

成员函数：

- **`set_model(const Eigen::Matrix4f& m):`** 将内部的**模型矩阵**作为参数传递给光栅化器。
- **`set_view(const Eigen::Matrix4f& v):`** 将视图变换矩阵设为内部**视图矩阵**。
- **`set_projection(const Eigen::Matrix4f& p):`** 将内部的**投影矩阵**设为给定矩阵 p，并传递给光栅化器
- **`set_pixel(Vector2f point, Vector3f color):`** 将屏幕像素点 (x, y) 设为 (r, g, b) 的颜色，并写入相应的帧缓冲区位置。

在 main.cpp 中，我们模拟了图形管线。我们首先定义了光栅化器类的实例，然后设置了其必要的变量。然后我们得到一个带有三个顶点的硬编码三角形 (请不要修改它)。在主函数上，我们定义了三个分别计算模型、视图和投影矩阵的函数，每一个函数都会返回相应的矩阵。接着，这三个函数的返回值会被 **`set_model()`**，**`set_view()`** 和 **`set_projection()`** 三个函数传入光栅化器中。最后，光栅化器在屏幕上显示出变换的结果。

在用模型、视图、投影矩阵对给定几何体进行变换后，我们得到三个顶点的正则化空间坐标 (canonical space coordinate)。正则化空间坐标是由三个取值范围

在 [-1,1] 之间的 x, y, z 坐标构成。我们下一步需要做的就是视口变换，将坐标映射到我们的屏幕中 (`window_width` * `window_height`)，这些在光栅化器中都已完成，所以不需要担心。但是，你需要去理解这步操作是如何运作的，这一点十分重要。

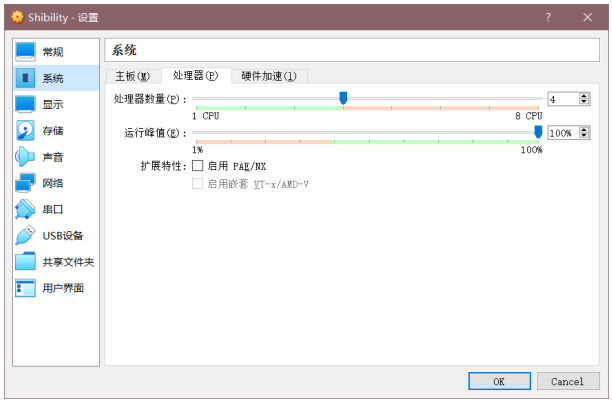
3 编译

如果使用自己的系统，本次程序作业中使用到的库为 Eigen 与 OpenCV，请确保这两个库的配置正确。

如果使用虚拟机并用 CMake 进行编译，请在终端 (命令行) 下输入以下内容：

```
mkdir build     // 创建build文件夹以保留的工程文件。
cd build        // 进入build文件夹。
cmake ..        // 通过提供CMakeLists.txt文件的路径
                // 作为参数来运行CMake。
make -j4        // 通过make编译代码， -j4 表示通过
                // 4个内核进行并行化编译。
./Rasterizer    // 运行代码。
```

如果没有对虚拟机的核心数进行过设置，可以在 **Virtual Box** 里点击“设置”，在弹出的窗口中选择“系统—处理器”，然后就可以设置虚拟机的核心数了。具体操作可见下图：



4 评分与提交

评分:

每次作业的评分，分为基础与提高两部分，即在作业批改时会给大家反馈两个成绩。

- [5 分] 正确构建模型矩阵。
- [5 分] 正确构建透视投影矩阵。
- [10 分] 你的代码可以在现有框架下正确运行，并能看到变换后的三角形。
- [10 分] 当按 A 键与 D 键时，三角形能正确旋转。或者正确使用命令行得到旋转结果图像。
- [提高项 5 分] 在 main.cpp 中构造一个函数，该函数的作用是得到绕任意过原点的轴的旋转变换矩阵。

  `Eigen::Matrix4f get_rotation(Vector3f axis, float angle)`

注意:

当你完成作业后，请清理你的项目，记得在你的文件夹中包含CMakeLists.txt和所有的程序文件 (无论是否修改)。同时，请添加一个README.md 文件写下是否完成提高题。