本科生实验报告

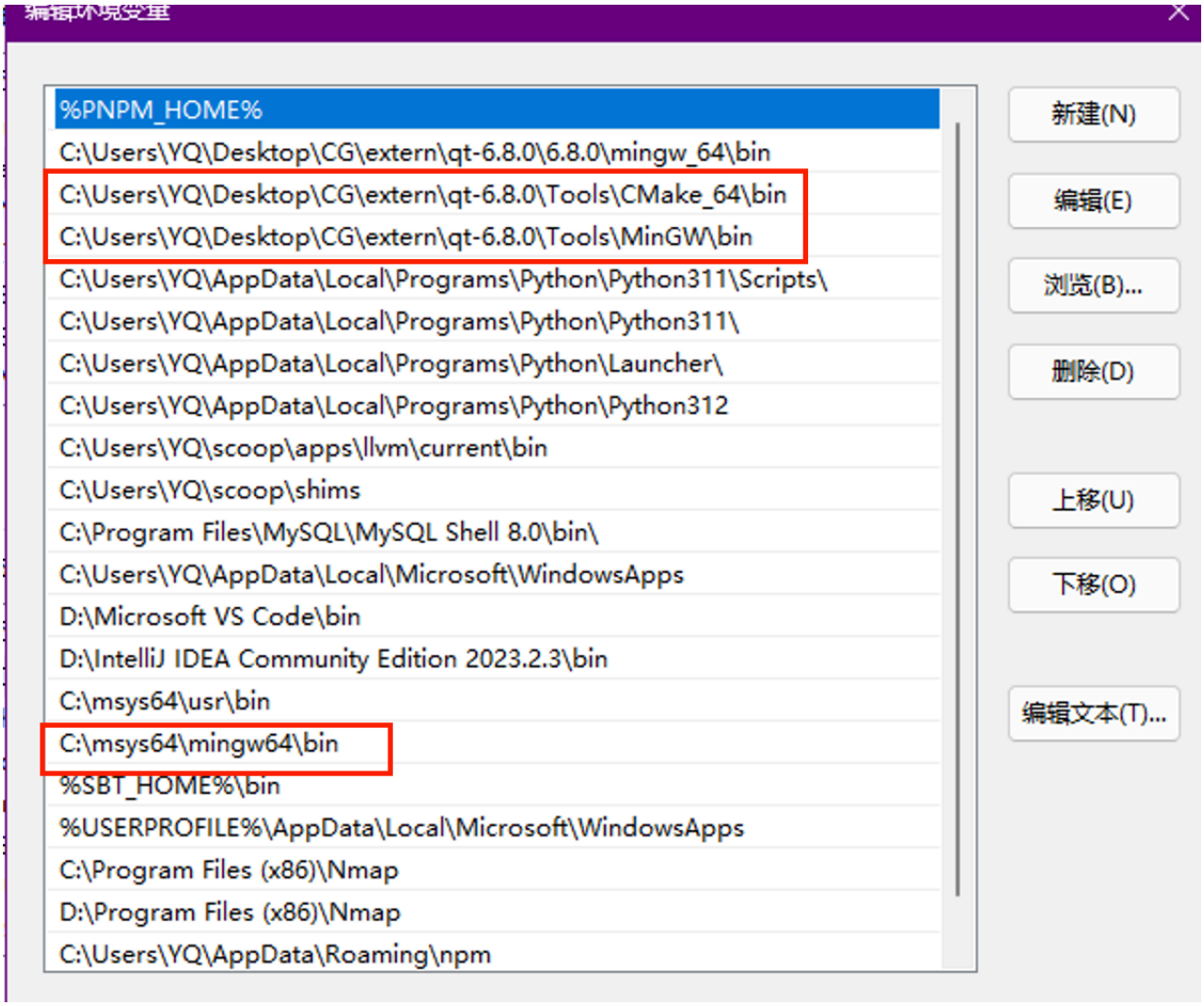
实验课程	计算机图形学第一次作业
专业名称	计算机科学与技术
学生姓名	李世源
学生学号	22342043
提交时间	2024年11月23日

环境搭建

我安装了 Qt 版本 6.8.0，然后安装 glew 来使用 OpenGL。

Visual Studio 过于庞大，而且项目本身很小，所以我没有使用 Visual Studio 进行开发，而是编写简单的 `Makefile` 和 `CMakeLists.txt`，使用 `cmake` 和 `make` 进行编译和链接。

环境搭建遇到了不少问题，主要都是编译器、make、cmake 版本的问题。Qt 的库编译和链接对这些版本的要求非常敏感。我的 Windows 原本就有 mingw 安装的 gcc、g++ 编译器和 scoop 安装的 make、cmake，但是这些版本除了 cmake 之外都比 Qt 6.8.0 要求的要高。于是 `make` 编译之后就遇到各种各样的链接错误。经过排查才发现是上述版本问题。为了解决这些问题，对于编译器，我修改了环境变量，在环境变量路径中添加 Qt 6.8.0 自带的编译器、cmake 所在路径，并放在最前面（我原来的编译器前面）如下图所示。



然后开发的过程中，我使用 VSCode，配合 clangd 插件作为 language server，定义 C++ 版本为 `-std=c++17`，在这个开发环境下为了实现语法检查，我稍微修改了模版中的 `#include`。对于一些头文件的引用使用更详细的引用方式。

项目结构

项目结构非常简单，为了使项目结构更清晰，我自己实现的 `MyGLWidget` 放在单独一个路径 `myglwidget` 下，并做了简单的模块拆分。

编译运行

不使用 Visual Studio，只需要类似我上述方法配置环境变量之后，在项目根目录下运行：

```
make
```

但是这一步注意，`CMakeLists.txt` 里我硬编码了自己电脑上的 Qt 和 glew 动态链接库的路径，您的电脑上运行也需要相应修改成对应的路径。

然后生成的 `build` 目录下就有可执行文件 `CG-HW1.exe`

使用 Visual Studio，我上传的 `build` 里面就是已经生成好的 Visual Studio 项目文件。生成方法是在项目根目录下运行：

```
make sln
```

但是这一步注意，`Makefile` 里是硬编码了我自己电脑上 Visual Studio 的路径，您的电脑上运行也需要相应修改成对应的路径。另外若上一次 `make` 不是 `sln` 还需要将上一次的 `build` 清理掉，否则会有报错。

直接运行我上传的程序 `CG-HW1.exe` 即可见到我上传视频中的最终效果。

最终我利用 `glRotatef` 实现的 3D 字母有绕 X, Y, Z 轴旋转的方式，分别使用键盘控制：

- `W` 和 `S`：绕 X 旋转
- `A` 和 `D`：绕 Y 旋转
- `Q` 和 `E`：绕 Z 旋转

讨论1

`GL_TRIANGLES`、`GL_TRIANGLE_STRIP` 和 `GL_QUAD_STRIP` 的绘制开销主要体现在所需的 `glVertex` 调用次数上。下面是对这三种绘制方式的比较：

绘制方式	所需顶点调用次数	解释
`GL_TRIANGLES`	`3 * n`	每个三角形独立，不共享顶点。
`GL_TRIANGLE_STRIP`	`n + 2`	共享顶点，减少顶点数量。
`GL_QUAD_STRIP`	`2 * n + 2`	共享顶点定义连续的四边形。

虽然 `GL_TRIANGLES` 的代码可能更简单、更直观，但在处理复杂模型时，使用条带方式能够显著提高效率，使用 `GL_TRIANGLE_STRIP` 和 `GL_QUAD_STRIP` 可以减少 `glVertex` 调用的数量，从而提高性能，特别是在绘制大量三角形或四边形的时候。

在我的实现中，由于图形比较简单，为了方便就只采用了 `GL_TRIANGLES` 和 `GL_QUADS` 。

讨论2

视角 1：从 `(0,0,d)` 看向原点 `(0,0,0)`

Orthogonal 投影

- 特性：
  - 正交投影没有透视效果，物体的大小不随距离变化。
  - 视图中的所有线段平行于投影面。
- 效果：
  - 由于观察点在 z 轴正方向 `(0,0,d)`，整个场景的投影是直接沿 z 轴方向平行投影到视平面上。
  - 任何远近关系都被忽略，图像中的物体尺寸完全按照实际尺寸投影。
  - 图像会呈现一个从正面看下去的效果（"俯视" z 轴平面），没有深度感。

Perspective 投影

- 特性：
  - 透视投影具有透视效果，物体的大小会随着距离远近变化。
  - 近大远小的效果由投影矩阵实现。
- 效果：
  - 观察点在 `(0,0,d)`，原点 `(0,0,0)` 是视点的焦点，因此投影会模拟人眼的自然视觉。
  - z 轴上的物体会看起来逐渐缩小，产生深度感。
  - 图像看起来像是从一个 "正面观察" 的视角，但具有透视效果。

视角 2：从 `(0,0.5*d,d)` 看向原点 `(0,0,0)`

Orthogonal 投影

- 特性：
  - 同样没有透视效果。
  - 观察点 `(0,0.5*d,d)` 和 up 方向 `(0,1,0)` 会使视图稍微倾斜，但投影仍然是平行的。
- 效果：
  - 相较于视角 1，图像会稍微偏向 z 轴与 y 轴之间的方向。
  - 由于正交投影中没有深度感，图像中的物体仍然保持实际大小，尺寸与位置不随距离变化。
  - 整体上，图像看起来像是从一个 "倾斜的角度" 俯视场景。

Perspective 投影

- 特性：
  - 仍然具有透视效果，物体的大小随距离变化。
  - 观察点的位置和 up 向量会影响视角方向，焦点仍然是原点。
- 效果：

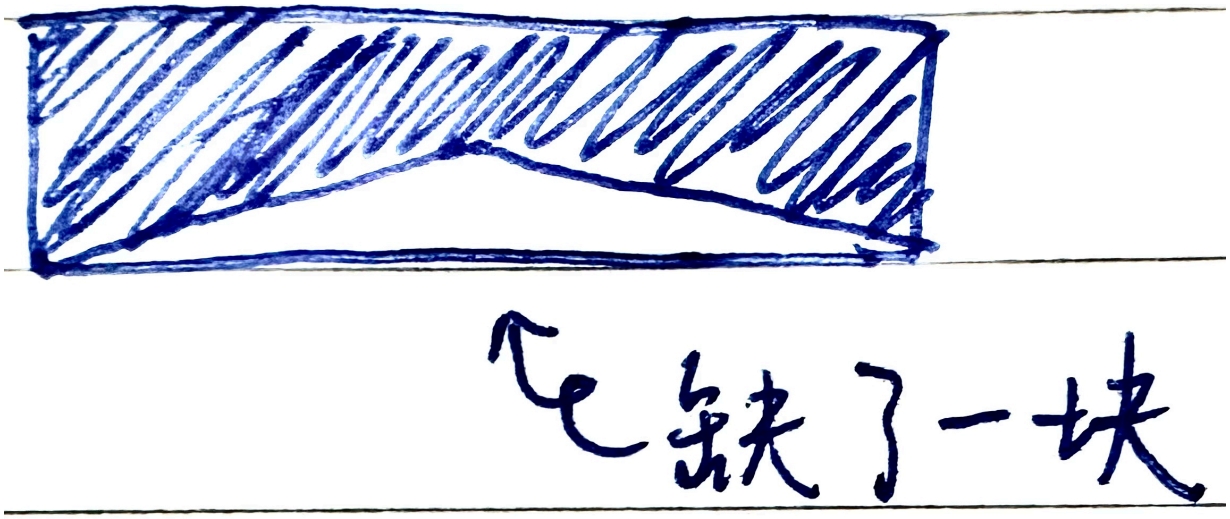
- 相较于视角 1，图像会呈现一个从上方稍微倾斜的视角。
- 物体的近大远小效果更加显著，尤其是沿 y 轴和 z 轴方向的物体。
- 透视效果增强了空间感，使得图像看起来更接近人眼在高位置俯视场景时的视角。

投影方式	视角 1: `(0,0,d)`	视角 2: `(0,0.5*d,d)`
Orthogonal 投影	没有透视效果，正面俯视场景，物体尺寸不变。	没有透视效果，稍微倾斜俯视场景，物体尺寸不变。
Perspective 投影	具有透视效果，从正面观察，近大远小，具有深度感。	具有透视效果，从倾斜视角俯视，近大远小，空间感更强。

- **Orthogonal 投影**更适用于工程绘图或技术制图，因为它消除了透视的干扰，让物体保持真实尺寸。
- **Perspective 投影**更贴近人眼感知，适合用来生成逼真的三维场景。

其他

实验中我遇到过一个有趣的问题，画完之后我的 3D 图形很多面居然出现了缺了一块的感觉，大致如下图所示：



这导致画出来的三维图有很残缺的感觉，非常不好看。

我一开始还很困惑，然后也是问了问有经验的学长才知道，应该是画矩形的时候顶点顺序错了！去看看 API 文档才得知 OpenGL 画的矩形是顶点按顺序连线围出来的，难怪会有上图这样的现象。修改了顺序之后，就解决了。