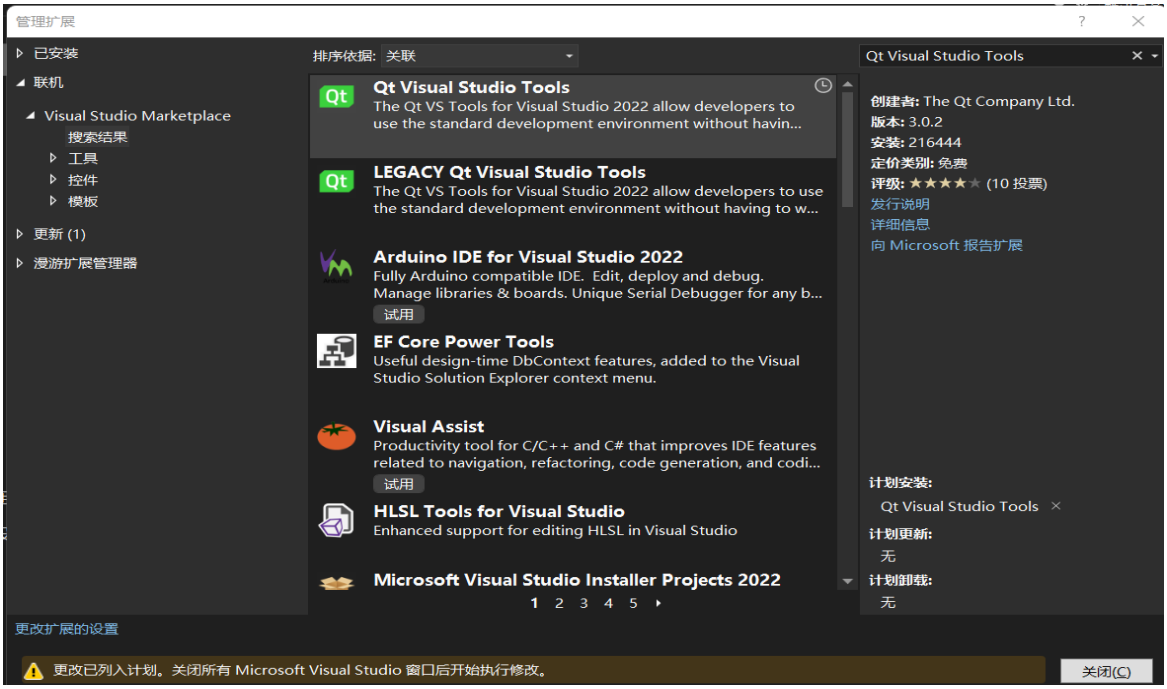
21307077 凌国明 CG-HW1

环境配置

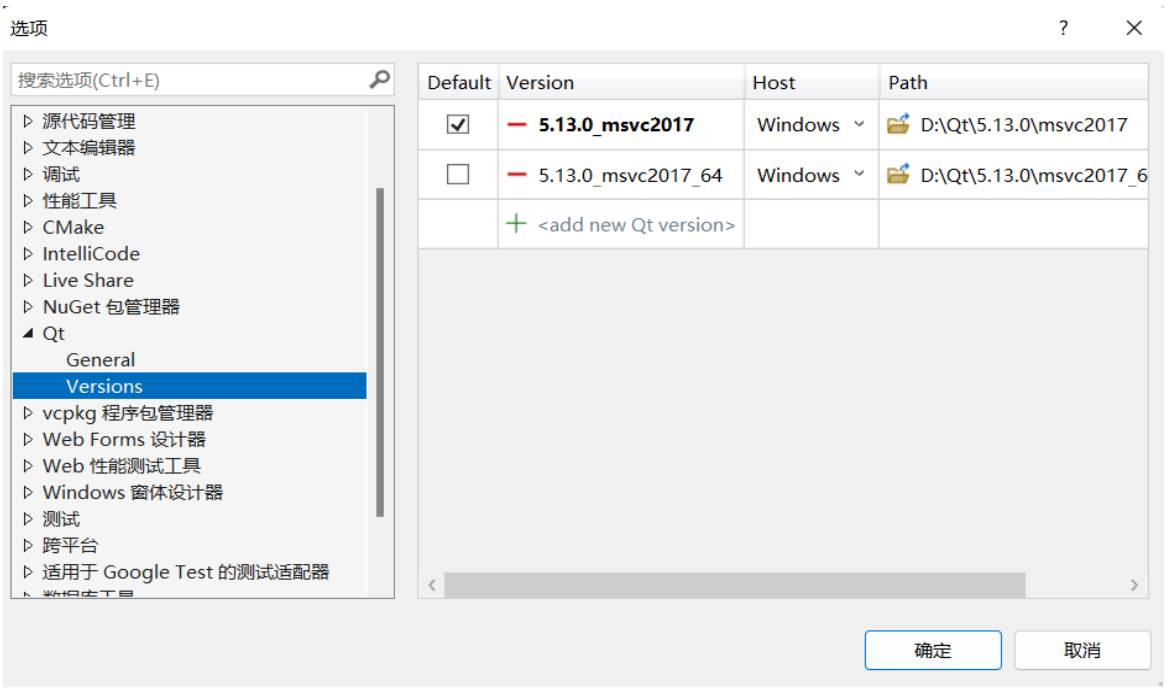
第一步：首先下载 Visual Studio 2022

第二步：下载 Qt

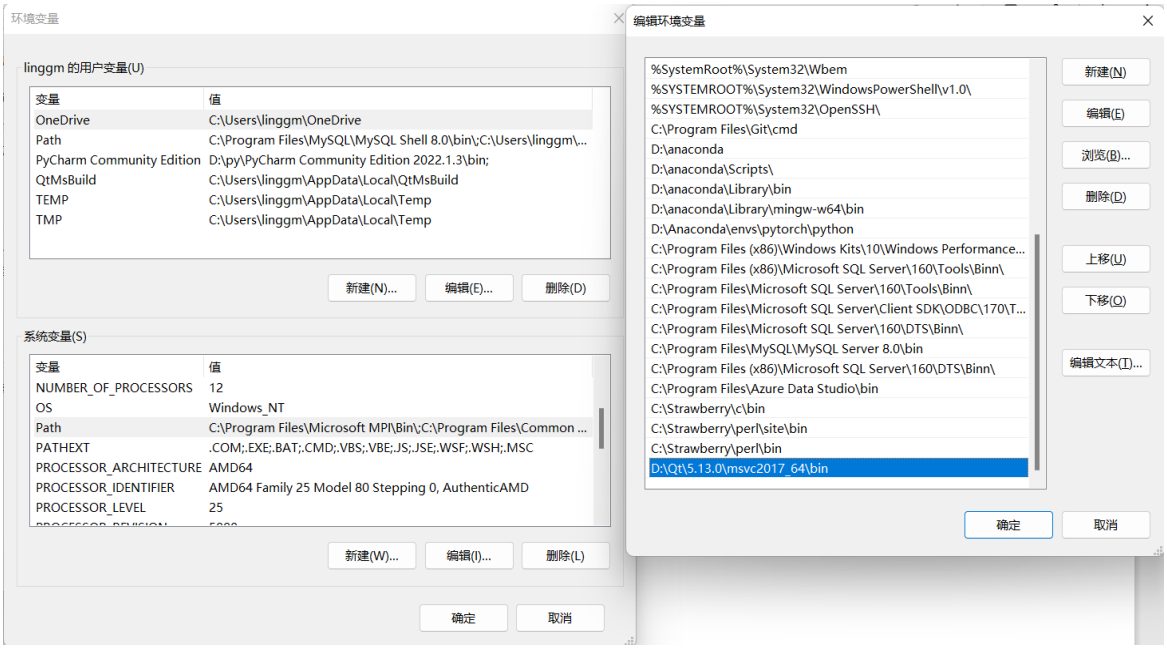
第三步：在 VS 中下载 Qt Visual Studio Tools



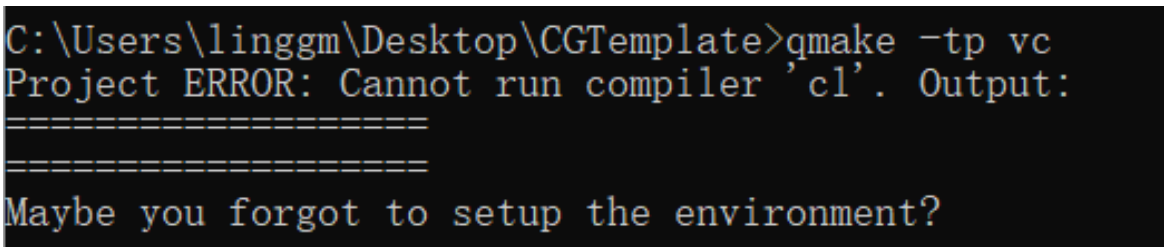
第四步：在 VS 的 Qt Visual Studio Tools 中设置 Qt 路径和版本



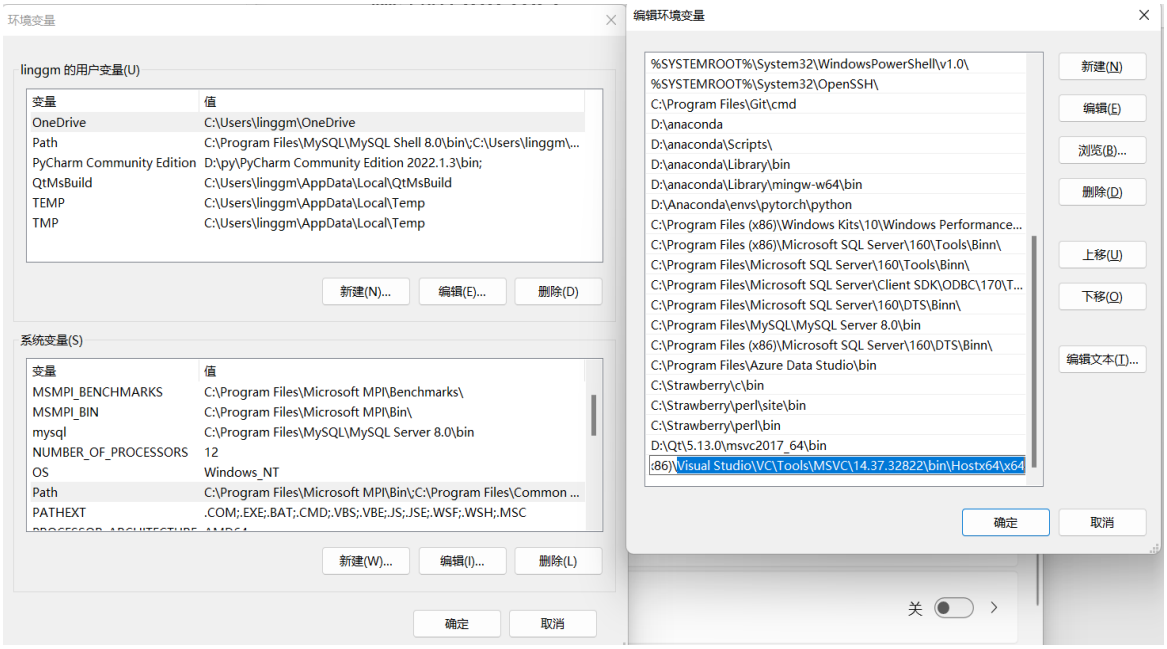
第五步：配置 qmake 环境变量



第六步：在 cmd 中，在 CGTemplate 文件夹下使用 qmake 编译，发现没有配置 cl.exe



第七步：将 VS 下的 cl.exe 的路径加入环境变量的 path

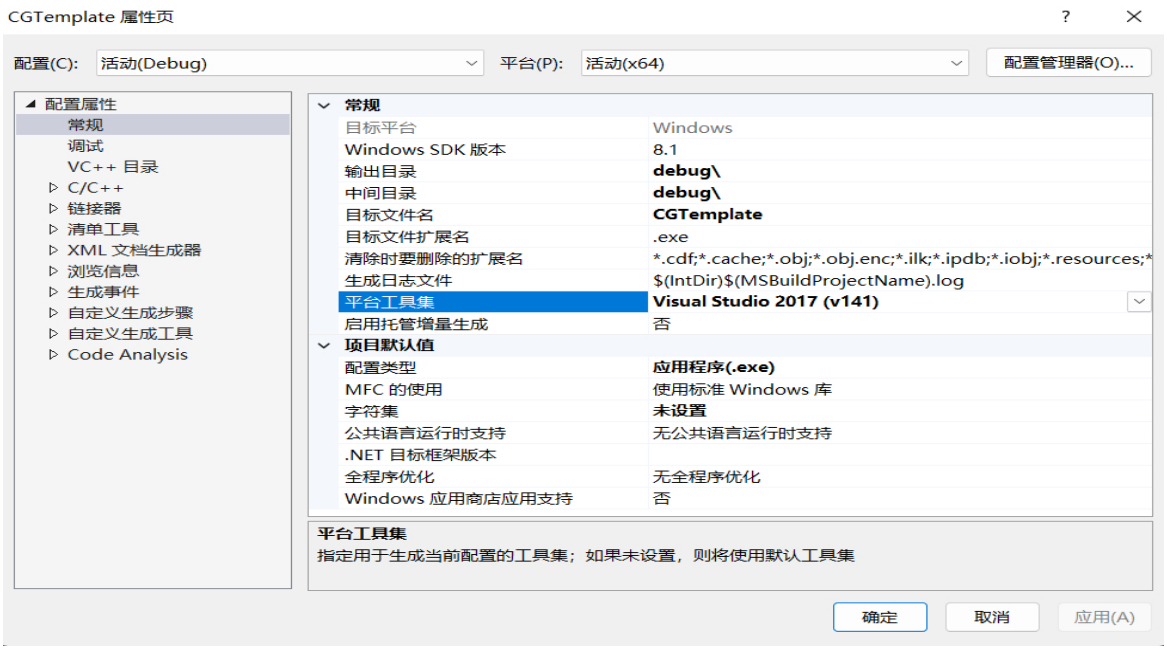


第八步：使用 qmake -tp vc 编译，成功

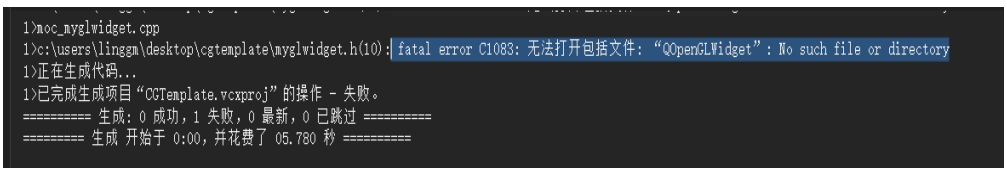
```
C:\Users\linggm\Desktop\CGTemplate>
C:\Users\linggm\Desktop\CGTemplate>
C:\Users\linggm\Desktop\CGTemplate>
C:\Users\linggm\Desktop\CGTemplate>qmake -tp vc
Info: creating stash file C:\Users\linggm\Desktop\CGTemplate\.qmake.stash

C:\Users\linggm\Desktop\CGTemplate>
```

第九步：在 VS 中运行 main.cpp，发现工具版本不对应要求的 v141（忘记截图了，网上找了张类似的）

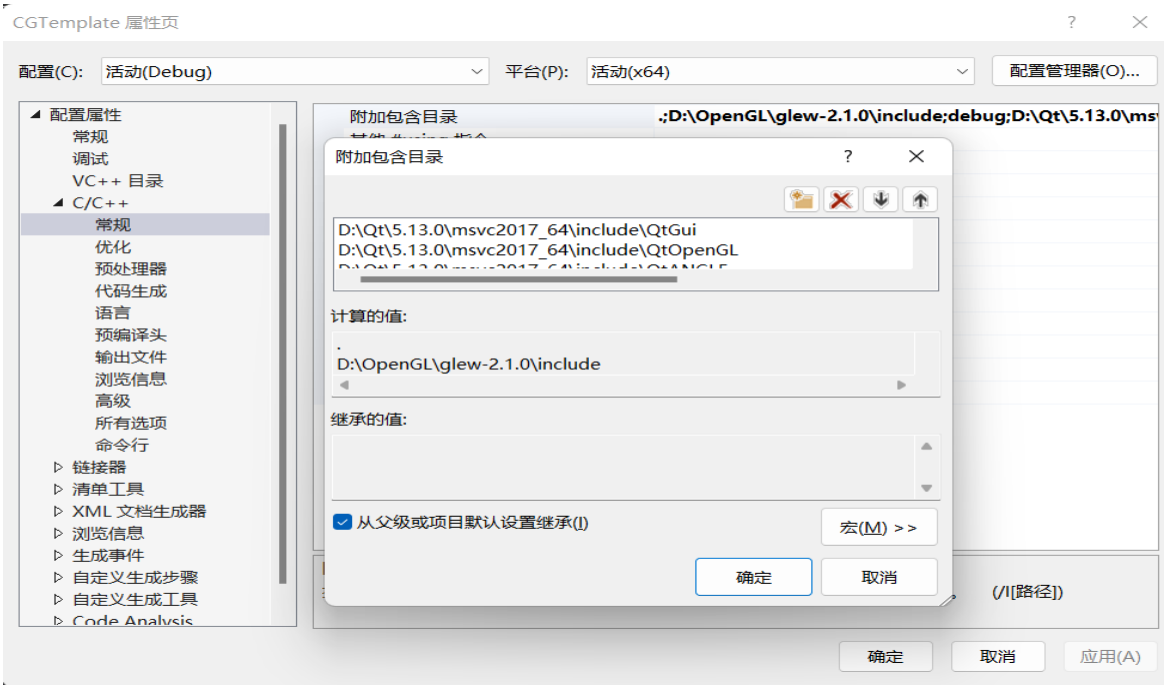
第十步：在 VS Installer 中下载平台工具集 v141，在项目属性中配置平台工具集



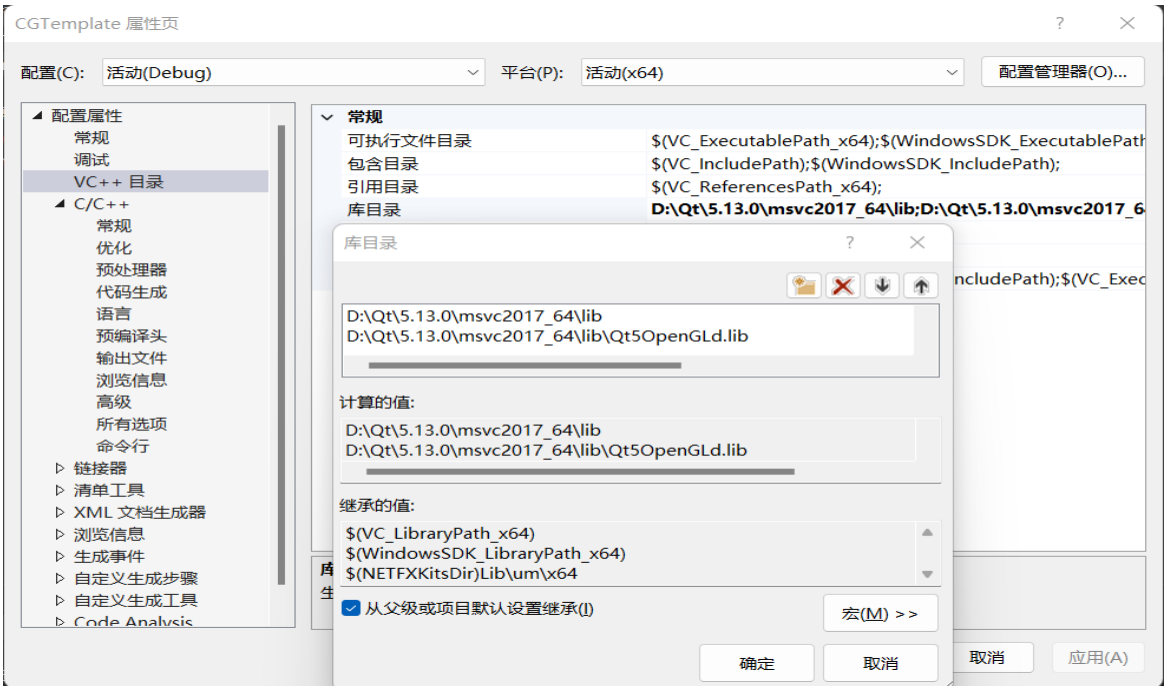
第十一步：运行 main.cpp，发现没有 include 相应的文件



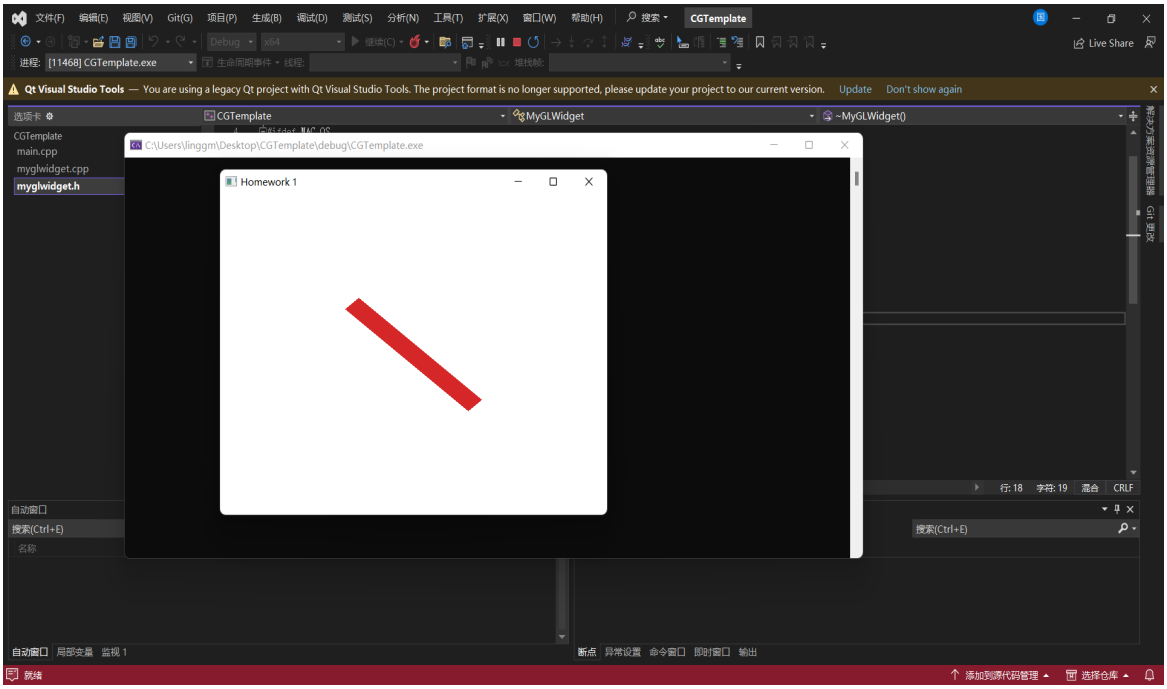
第十二步：在 CGTemplate 项目属性页中添加相应的 include 目录



第十三步：在 CGTemplate 项目属性页中添加相应的 lib 库



第十四步：运行 main.cpp，成功



平面绘制

首先，阅读模板中的代码，写上相应的注释

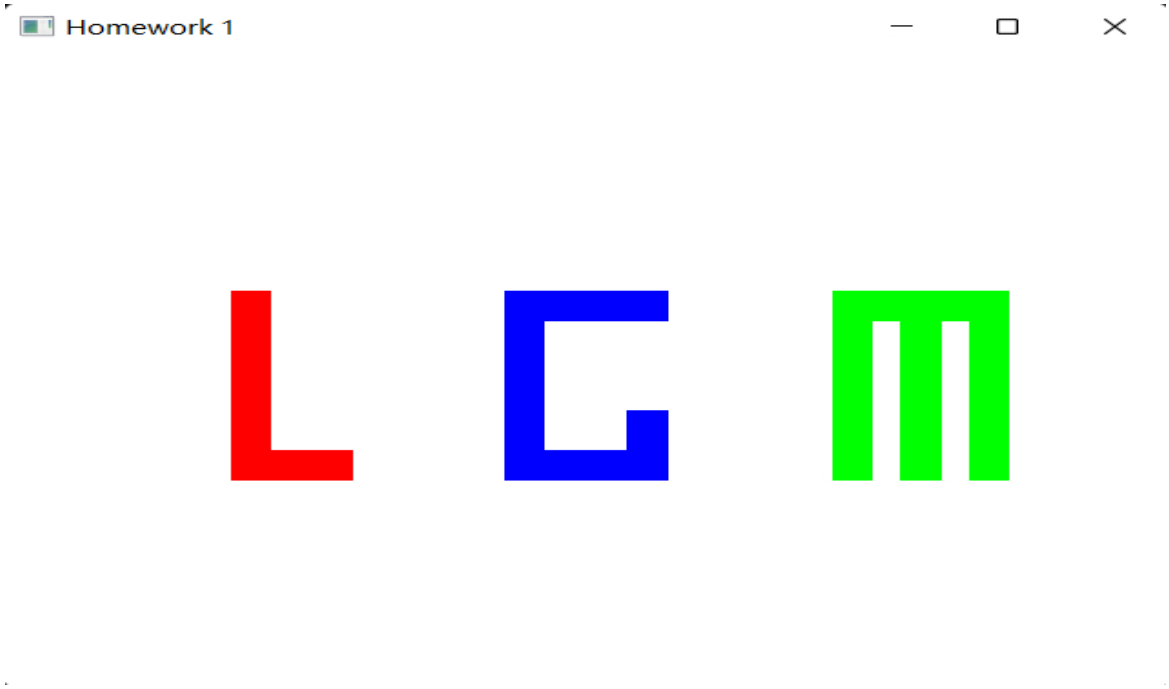
```
void MyGLWidget::initializeGL()
{
	/*
	功能：设置OpenGL的视口（Viewport）。
	参数：(0, 0)表示视口的左下角坐标，width()和height()从QOpenGLWidget继承，返回窗口的宽度和高度。
	作用：这告诉OpenGL渲染的区域大小和位置，这里设置为覆盖整个窗口。
	*/
	glViewport(0, 0, width(), height());
	/*
	功能：设置清除屏幕时使用的颜色。
	参数：四个浮点值分别代表红色、绿色、蓝色和透明度的值，范围从0.0到1.0。这里设置为白色，不透明。
	作用：当调用glClear(GL_COLOR_BUFFER_BIT)时，屏幕会被填充为此颜色。
	*/
	glClearColor(1.0f, 1.0f, 1.0f, 1.0f);
	// 禁用深度测试，在2D渲染或不需要考虑物体前后关系时，通常禁用深度测试，从而提高渲染效率。
	// glDisable(GL_DEPTH_TEST);
}
```

```
void MyGLWidget::scene_0()
{
        // 清除颜色缓冲区，使用在initializeGL函数中设置的清除颜色（白色）来清空屏幕。
        glClear(GL_COLOR_BUFFER_BIT);
        // 设置当前矩阵模式为投影矩阵，设置摄像机视角。
        glMatrixMode(GL_PROJECTION);
        // 重置当前指定的矩阵为单位矩阵，清除任何之前的矩阵变换，以便从头开始定义新的变换。
        glLoadIdentity();
        // 定义一个正交投影矩阵，定义视景体的左、右、下、上、近、远平面。
        // 这个函数描述了一个平行修剪空间。这种投影意味着离观察者较远的对象看上去不会变小（与透视投影相反
        // 这种投影的视景体是一个矩形的平行管道，也就是一个长方体。正射投影的最大一个特点是无论物体距离相
        glOrtho(0.0f, 100.0f, 0.0f, 100.0f, -1000.0f, 1000.0f);
        // 切换当前矩阵模式为模型视图矩阵，准备进行模型的变换操作。
        glMatrixMode(GL_MODELVIEW);
        // 重置模型视图矩阵为单位矩阵。
        glLoadIdentity();
        // 将模型移动到窗口中心位置。这是对I的第三个操作
        glTranslatef(50.0f, 50.0f, 0.0f);
        // 准备绘制一个对角线上的"I"形状。
        glPushMatrix();
        // 设置绘制颜色为深红色。
        glColor3f(0.839f, 0.153f, 0.157f);
        // 把当前矩阵和一个表示旋转物体的矩阵相乘。将"I"形状旋转45度。绕着（0，0，1）逆时针旋转45度。由
        glRotatef(45.0f, 0.0f, 0.0f, 1.0f);
        // 把当前矩阵和一个表示移动物体的矩阵相乘。将"I"形状平移到适当的位置。由于结合律，平移在前，这是
        glTranslatef(-2.5f, -22.5f, 0.0f);
        // 开始绘制三角形，用以构成"I"形状。
        glBegin(GL_TRIANGLES);
        // 定义第一个三角形的三个顶点。
        glVertex2f(0.0f, 0.0f);
        glVertex2f(5.0f, 0.0f);
        glVertex2f(0.0f, 45.0f);
        // 定义第二个三角形的三个顶点，与第一个三角形共同构成矩形的"I"形状。
        glVertex2f(5.0f, 0.0f);
        glVertex2f(0.0f, 45.0f);
        glVertex2f(5.0f, 45.0f);
        // 结束三角形的绘制。
        glEnd();
        // 恢复之前的矩阵状态。
        glPopMatrix();
}
```

阅读得知，MyGLWidget 类对 QOpenGLWidget 类进行了继承，重载了 QOpenGLWidget 的很多函数。其中 scene_0 是绘图函数，利用“三角形”这一基本图元构建图形。模仿这个函数的写法，可以在 xoy 平面上绘制出 LGM 三个字母

```
void MyGLWidget::scene_1()
{
        // 清除颜色缓冲区，使用在initializeGL函数中设置的清除颜色（白色）来清空屏幕。
        glClear(GL_COLOR_BUFFER_BIT);
        // 设置当前矩阵模式为投影矩阵，设置摄像机视角。
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        // 定义一个正交投影矩阵，定义视景体的左、右、下、上、近、远平面。
        glOrtho(0.0f, width(), 0.0f, height(), -1000.0f, 1000.0f);
        // 切换当前矩阵模式为模型视图矩阵，准备进行模型的变换操作。
        glMatrixMode(GL_MODELVIEW);
        glLoadIdentity();
        // 将模型移动到窗口中心位置。
        glTranslatef(0.5 * width(), 0.5 * height(), 0.0f);
        // 设置相机位置
        gluLookAt(0.0f, 0.0f, 100.0f, 0.0f, 0.0f, 0.0f, 0.0f, 1.0f, 0.0f);
        // 相机位置 观察点位置 上方向

        glPushMatrix(); // 保持当前矩阵状态
        // 绘制 "L"
        glBegin(GL_TRIANGLES);
        glColor3f(1.0f, 0.0f, 0.0f); // 红色
        // 竖直部分
        glVertex2f(-65.0f, 20.0f);
        glVertex2f(-57.5f, 20.0f);
        glVertex2f(-57.5f, -20.0f);
        // 省略部分 glVertex2f 语句

        // 绘制 "G"
        // 省略部分 glVertex2f 语句
        // 绘制 "M"
        // 省略部分 glVertex2f 语句

        glEnd();
        // 恢复矩阵状态
        glPopMatrix();
}
```

结果如图所示



比较 GL_TRIANGLES, GL_TRIANGLE_STRIP, GL_QUAD_STRIP 的绘制开销

glBegin 和 glEnd 之间有 glVertex，根据 glBegin 模式的不同，绘制规则和开销也不同。

1. GL_TRIANGLES 的绘制规则：共有 $3 \cdot n$ 个 glVertex（顶点数一定要是 3 的倍数），每 3 个 glVertex 构成一个三角形。
2. GL_TRIANGLE_STRIP 的绘制规则：共有 n 个 glVertex，可以绘制 $n-2$ 个三角形。若当前点索引是奇数 k，则按 $k, k+1, k+2$ 的顺序连接三点绘制三角形；若当前点索引是偶数 k，则按 $k+1, k, k+2$ 的顺序连接三点绘制三角形。
3. GL_QUAD_STRIP 的绘制规则：共有 n 个 glVertex，可以绘制 $\frac{n}{2}-1$ 个四边形。若当前点索引是整数 k，则按照 $2n-1, 2n, 2n+2, 2n+1$ 的顺序连接四点构成四边形。

按照以上规则，可以得知绘制一样的图形，三种模式有不同的 glVertex 个数，下面来进行比较

1. GL_TRIANGLES 的绘制开销：L 需要 4 个三角形，G 需要 8 个三角形，M 需要 8 个三角形。GL_TRIANGLES 绘制一个三角形需要 3 个 glVertex。因此一共需要 $(4+8+8) \cdot 3 = 60$ 个 glVertex
2. GL_TRIANGLE_STRIP 的绘制开销：L 需要 2 个四边形，G 需要 4 个四边形，M 需要 4 个四边形。GL_TRIANGLE_STRIP 绘制一个四边形需要 4 个 glVertex。因此一共需要 $(2+4+4) \cdot 4 = 40$ 个 glVertex
3. GL_QUAD_STRIP 的绘制开销：L 需要 2 个四边形，G 需要 4 个四边形，M 需要 4 个四边形。GL_QUAD_STRIP 绘制一个四边形需要 4 个 glVertex。因此一共需要 $(2+4+4) \cdot 4 = 40$ 个 glVertex

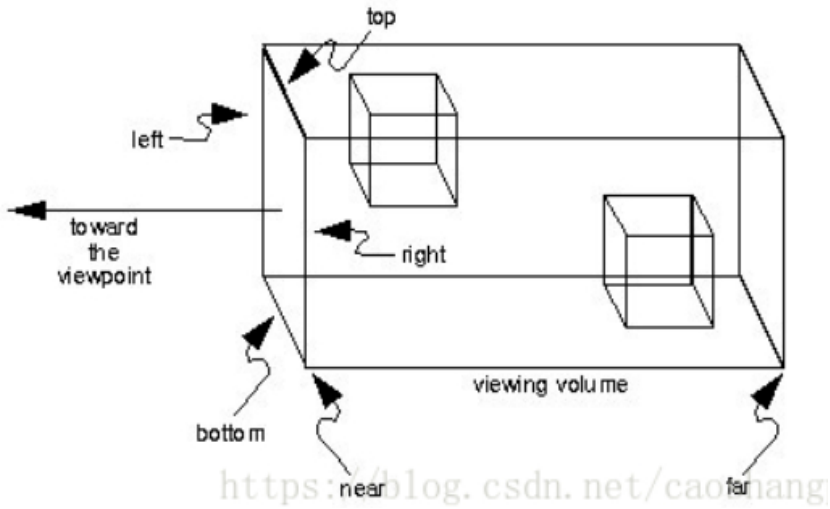
参考材料：

1. GL_TRIANGLE_STRIP https://blog.csdn.net/shulianghan/article/details/112799758
2. GL_QUAD_STRIP https://blog.csdn.net/shulianghan/article/details/112851868

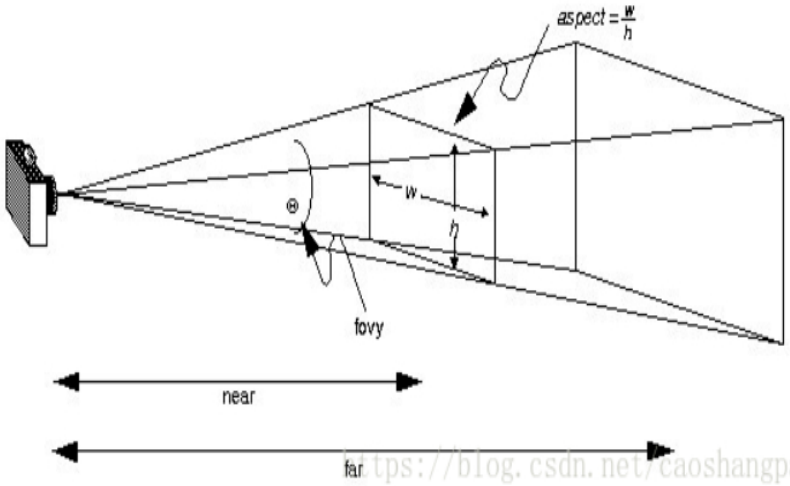
不同的相机模式和视角的结果

```
// 这个函数可以调整视角，调整 相机位置 观察点位置 上方向
gluLookAt(100.0f, 100.0f, 100.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 1.0f);
```

这种投影的视景体是一个矩形的平行管道，也就是一个长方体。正射投影的最大一个特点是无论物体距离相机多远，投影后的物体大小尺寸不变



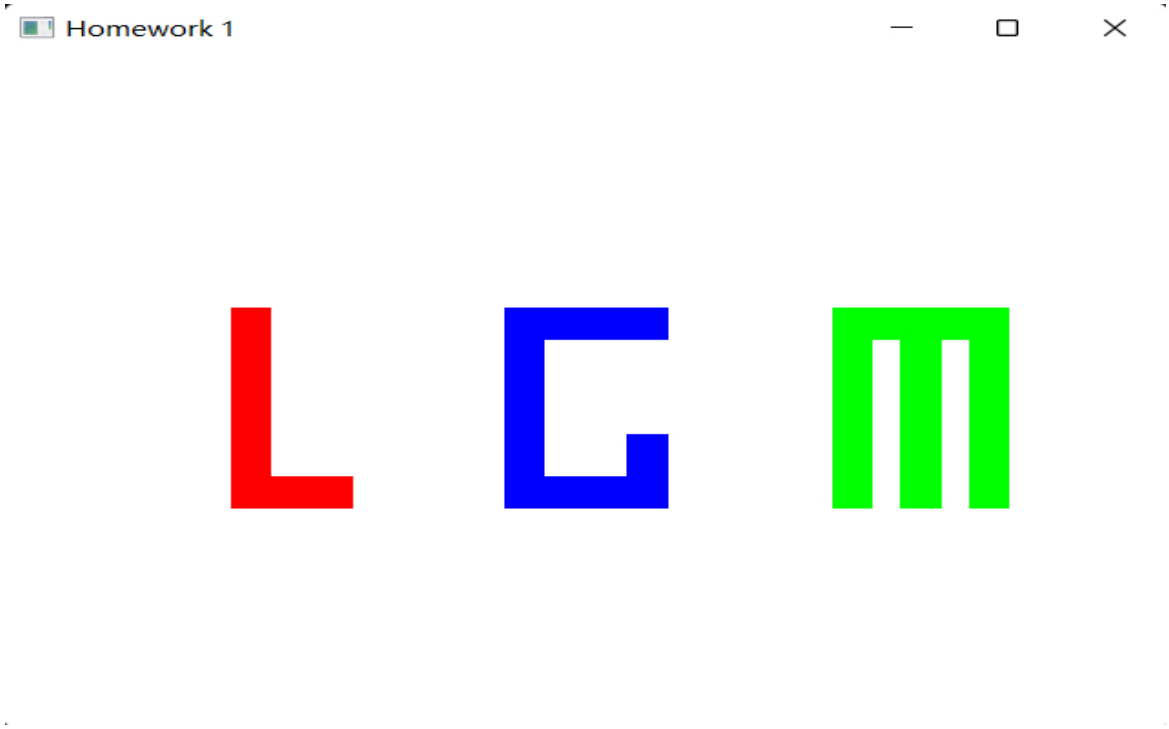
透视投影所产生的结果类似于照片，有近大远小的效果



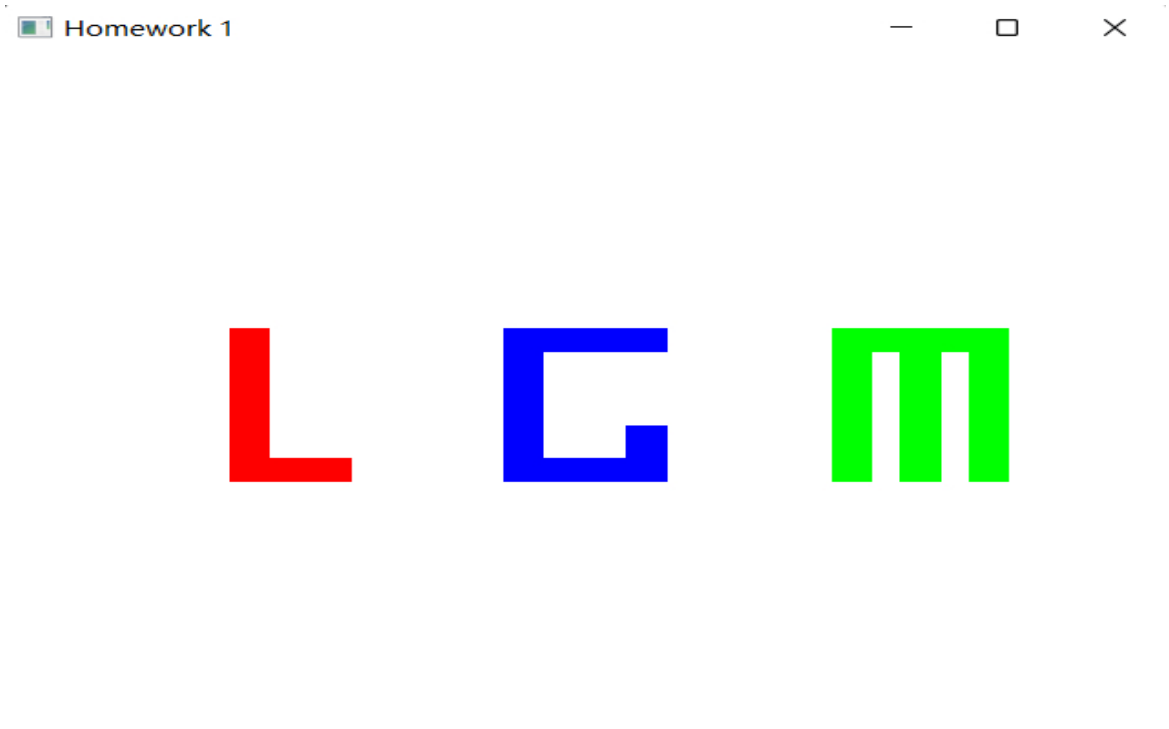
参考材料：

1. glMatrixMode https://blog.csdn.net/caoshangpa/article/details/80266028

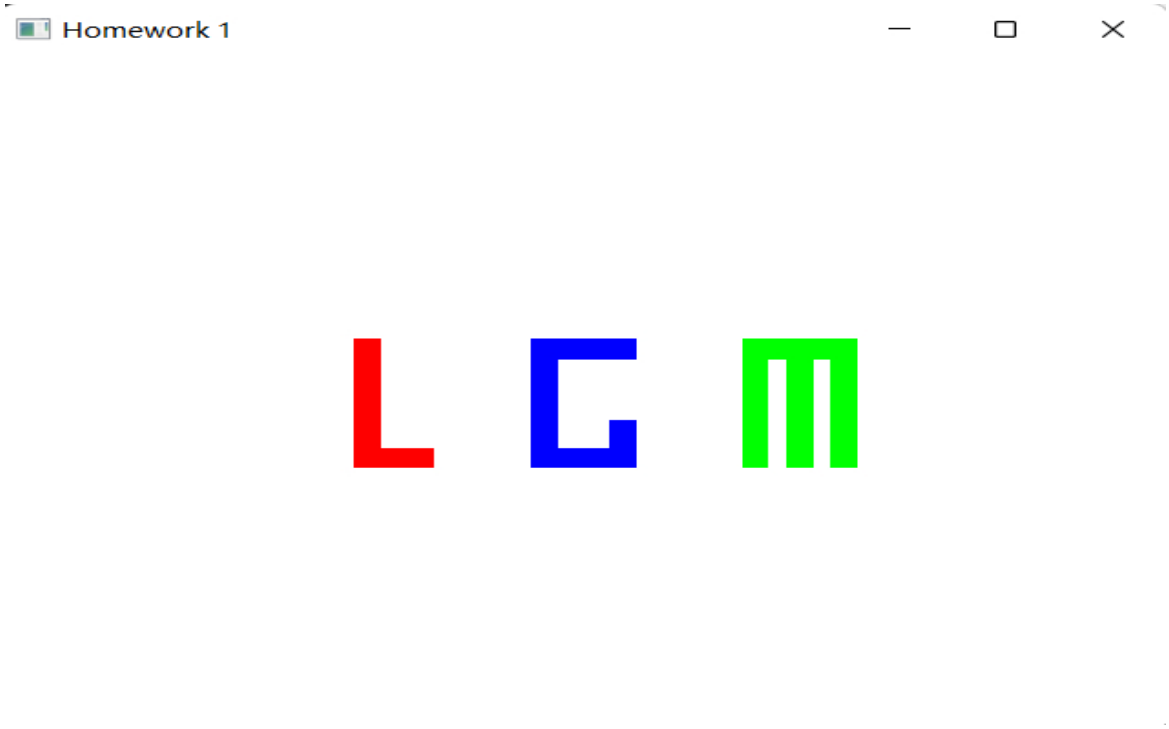
正交投影，从(0, 0, d)看向原点(0, 0, 0)



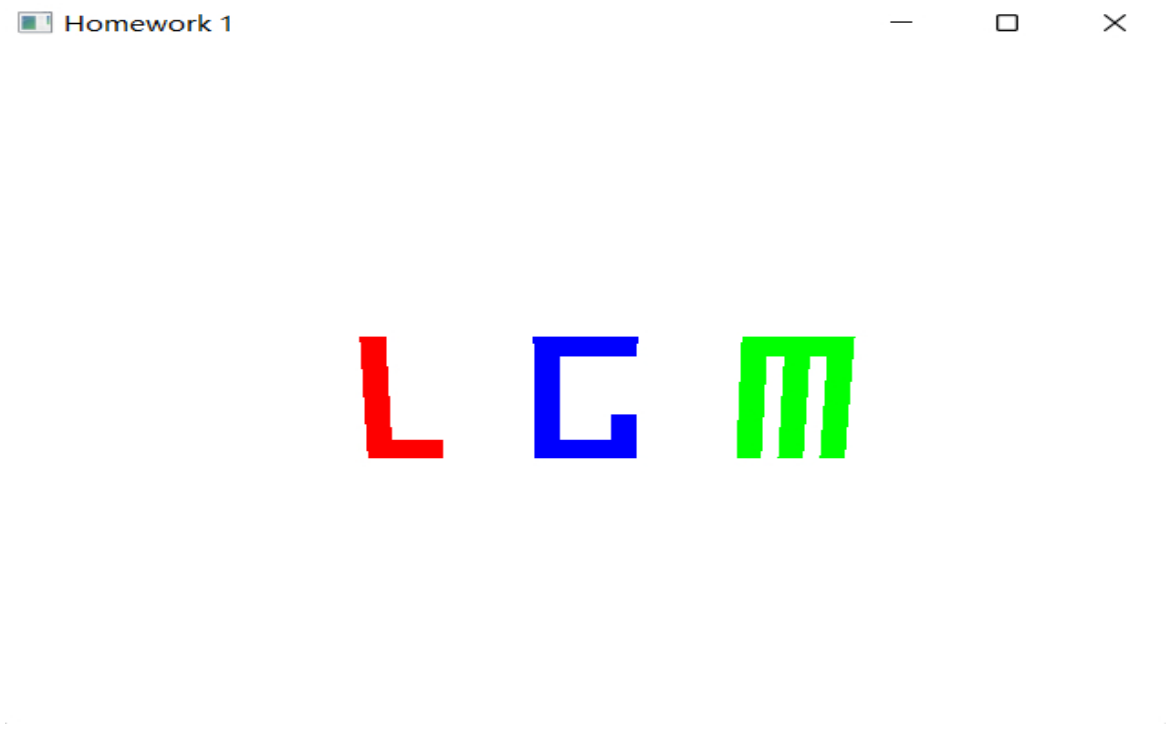
正交投影，从(0, d/2, d)看向原点(0, 0, 0)



透视投影，从(0, 0, d)看向原点(0, 0, 0)



透视投影，从(0, d/2, d)看向原点(0, 0, 0)



立体绘制

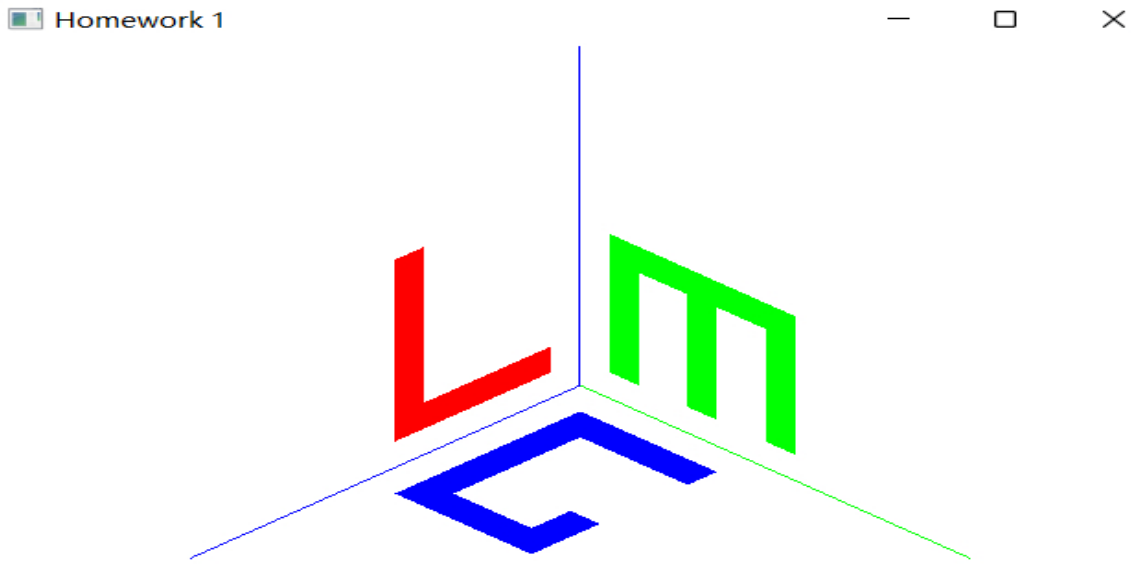
其他代码与平面绘制一致，只需要将 glVertex2f 改成 glVertex3f，并在三维空间中绘制三个字母。旋转的实现：先绘制 xyz 坐标轴，然后 glPushMatrix 使得接下来的旋转变换对坐标轴不生效，然后绘制 LGM 三个字母，并应用旋转矩阵，最后 glPopMatrix。

```
void MyGLWidget::scene_5(){
        // 前面设置与之前的代码一致
        // 绘制XYZ坐标轴
        glBegin(GL_LINES);
        // X轴（红色）
        glColor3f(0.0f, 0.0f, 1.0f); // 红色
        glVertex3f(0.0f, 0.0f, 0.0f);
        glVertex3f(100.0f, 0.0f, 0.0f);
        // 省略 Y轴（绿色） Z轴（蓝色）
        glPushMatrix(); // 保持当前矩阵状态，使得旋转对坐标轴不生效
        glRotatef(this->angle, 0.0, 0.0, 1.0); // 只旋转三个字母
        // 绘制 "L"
        glBegin(GL_QUADS);
        glColor3f(1.0f, 0.0f, 0.0f); // 红色
        // 竖直部分
        glVertex3f(40.0f, 0.0f, 55.0f);
        // 省略一堆glVertex3f代码
        // 恢复矩阵状态
        glPopMatrix();
}
```

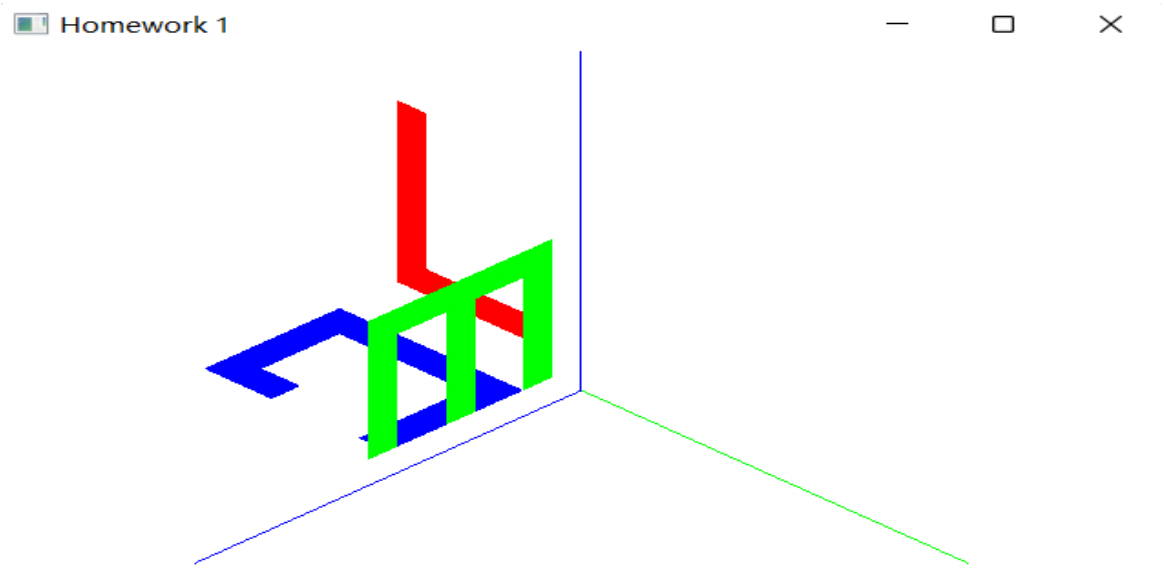
为了实现按键旋转，按下方向键时，将现有旋转角度进行变化，然后 update 绘制新的图

```
void MyGLWidget::keyPressEvent(QKeyEvent *e) {
        // 省略部分代码
        else if (e->key() == Qt::Key_Left) {
                angle -= 15;
                update();
        }
        else if (e->key() == Qt::Key_Right) {
                angle += 15;
                update();
        }
}
```

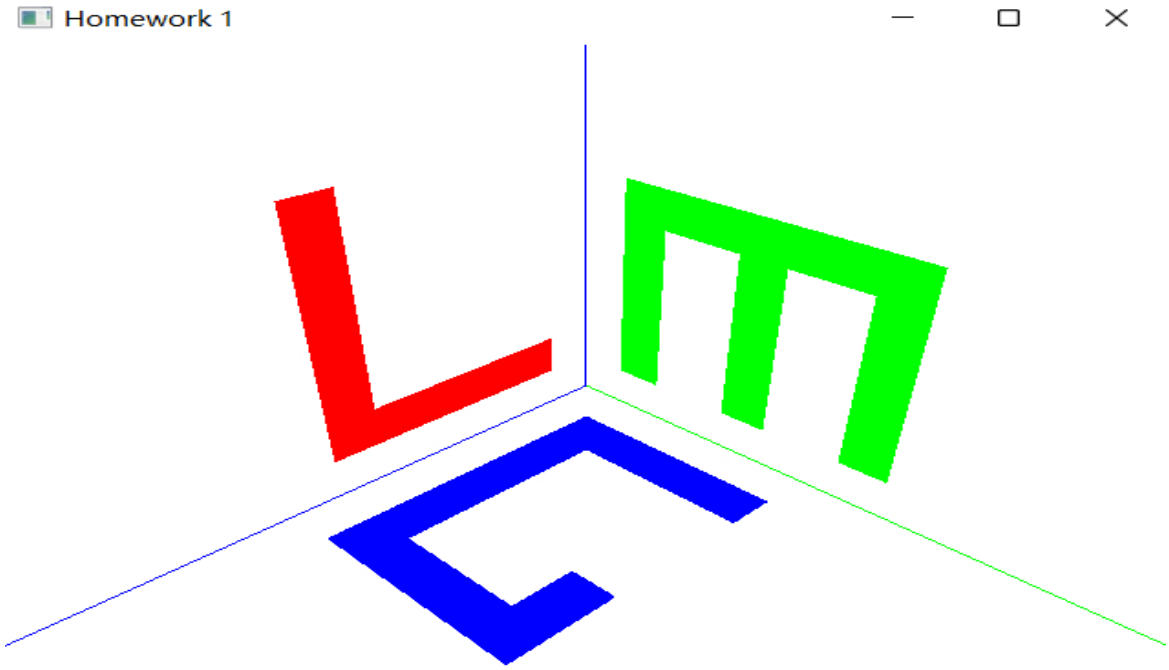
正交投影，从(d, d, d)看向原点(0, 0, 0)，旋转前



正交投影，从(d, d, d)看向原点(0, 0, 0)，旋转后



透视投影，从(d, d, d)看向原点(0, 0, 0)，旋转前



透视投影，从(d, d, d)看向原点(0, 0, 0)，旋转后