CG Assignment1

17341046 郭梓煜

环境配置

- Qt 5.13.0
- Qt Creator
- opengl
- glew 2.1.0

主要算法

Bresenham’s algorithm (my_GLlines_for_scene1/my_GLlines_for_scene2)

- 考虑两点重合，以及斜率调整为[0,1]的情况
  - 两点重合直接对一个像素点着色即可
  - 斜率>1则swap两个坐标轴即可
  - 相关代码如下：

```
//考虑两点重合与否
int dx = abs(x_2 - x_1), dy = abs(y_2 - y_1);
if(dx == 0 && dy == 0)
{
    glBegin(GL_POINTS);
    glVertex2i(x_1, y_1);
    glEnd();
    glFlush();
    return;
}
//将斜率绝对值变换到[0,1]区间
//翻转xy轴
int flag = 0;
if(dx < dy)
{
     flag = 1;
      std::swap(x_1, y_1);
     std::swap(x_2, y_2);
      std::swap(dx, dy);
}
```

- 考虑适应窗口大小问题
  - 应对不同窗口的不同width(),height()对点坐标进行缩放
  - 相关代码如下：

```
//适应窗口大小
x_1 *= width() / 100.0f;
y_1 *= height() / 100.0f;
x_2 *= width() / 100.0f;
y_2 *= height() / 100.0f;
```

- 其余根据老师PPT中伪码即可
  - 相关代码如下：

```
//考虑两点位置关系
int tmp_x = (x_2 - x_1) > 0 ? 1 : -1;
int tmp_y = (y_2 - y_1) > 0 ? 1 : -1;
//记录当前x,y值
int cur_x = x_1;
int cur_y = y_1;
//计算2* deta(y),2(deta(y)-deta(x)),2*deta(y)-deta(x)
int ds = 2 * dy;
int dt = 2 * (dy - dx);
int p = 2 * dy - dx;
//画点成线
while(cur_x != x_2)
{
    if(p < 0)
        p += ds;
    else
    {
        cur_y += tmp_y;
        p += dt;
    }
    //抗锯齿
    //glEnable(GL_POINT_SMOOTH);
    //glHint(GL_POINT_SMOOTH, GL_NICEST);
    glBegin(GL_POINTS);
    //画点
    if(flag)
        glVertex2i(cur_y, cur_x);
    else
        glVertex2i(cur_x, cur_y);
    glEnd();
    glFlush();
    glDisable(GL_POINT_SMOOTH);
    cur_x += tmp_x;
}
```

抗锯齿

- 调用glEnable,glHint实现
- 可使用按键2查看效果
- 代码如下：

```
glEnable(GL_POINT_SMOOTH);
glHint(GL_POINT_SMOOTH, GL_NICEST);
```

平移(my_Translate)

- 构造平移矩阵，对变换矩阵进行update即可
- 代码如下：

```
//构造平移矩阵
GLfloat tran_Matrix[4][4] =
{
    {1.0f, 0.0f, 0.0f, x},
    {0.0f, 1.0f, 0.0f, y},
    {0.0f, 0.0f, 1.0f, z},
    {0.0f, 0.0f, 0.0f, 1.0f}
};
my_MultMatrix(my_Matrix, tran_Matrix);
```

旋转(my_Rotate)

- 构造旋转矩阵，对变换矩阵进行update即可
- 需要注意的是对旋转角度单位转换以及不同精度的数计算的类型转换
    - 如GLfloat(float)与double的变换
- 代码如下：

```
//首先单位化旋转轴
GLfloat denominator = sqrtf(x * x + y * y + z * z);
x /= denominator;
y /= denominator;
z /= denominator;
//转换角度单位
double k = -angle * M_PI / 180.0f;
//构造旋转矩阵
GLfloat rotate_Matrix[4][4] =
{
    {static_cast<GLfloat>(x*x * (1 - cos(k)) + cos(k)), static_cast<GLfloat>
(x*y * (1 - cos(k)) + z * sin(k)), static_cast<GLfloat>(x*z * (1 - cos(k)) -
y * sin(k)), 0.0f},
    {static_cast<GLfloat>(x*y * (1 - cos(k)) - z * sin(k)),
static_cast<GLfloat>(y*y * (1 - cos(k)) + cos(k)), static_cast<GLfloat>(y*z
* (1 - cos(k)) + y * sin(k)), 0.0f},
    {static_cast<GLfloat>(x*z * (1 - cos(k)) + y * sin(k)),
static_cast<GLfloat>(z*y * (1 - cos(k)) - x * sin(k)), static_cast<GLfloat>
(z*z * (1 - cos(k)) + cos(k)), 0.0f},
    {0.0f, 0.0f, 0.0f, 1.0f}
};
my_MultMatrix(my_Matrix, rotate_Matrix);
```

基于四元数实现旋转 (my_quaternion_Rotate)

- 根据旋转角度以及点坐标构造四元数，求出其x,y,z,w
- 注意角度应该除以2，以及对轴向量进行归一化
- 用矩阵进行表示
- 代码如下：

```
//首先单位化旋转轴
GLfloat denominator = sqrtf(a * a + b * b + c * c);
a /= denominator;
b /= denominator;
c /= denominator;
//将角度除以2并转换单位
double k = angle * M_PI / 360.0f;
//计算Quaternion的x,y,z,w
GLfloat x = static_cast<GLfloat>(a * sin(k));
GLfloat y = static_cast<GLfloat>(b * sin(k));
GLfloat z = static_cast<GLfloat>(c * sin(k));
GLfloat w = static_cast<GLfloat>(cos(k));
//利用四元数构造矩阵
GLfloat rotate_Matrix[4][4] =
{
    {1 - 2 * y * y - 2 * z * z, 2 * (x * y - w * z), 2 * (x * z + w * y),
0.0f},
    {2 * (x * y + w * z), 1 - 2 * x * x - 2 * z * z, 2 * (y * z - w * x),
0.0f},
    {2 * (x * z - w * y), 2 * (y * z + w * x), 1 - 2 * x *x - 2 * y * y,
0.0f},
    {0.0f, 0.0f, 0.0f, 1.0f}
};
my_MultMatrix(my_Matrix, rotate_Matrix);
```

矩阵乘法(my_MultiMatrix)

- 使用循环实现矩阵乘
- 较为简单，代码不多加展示

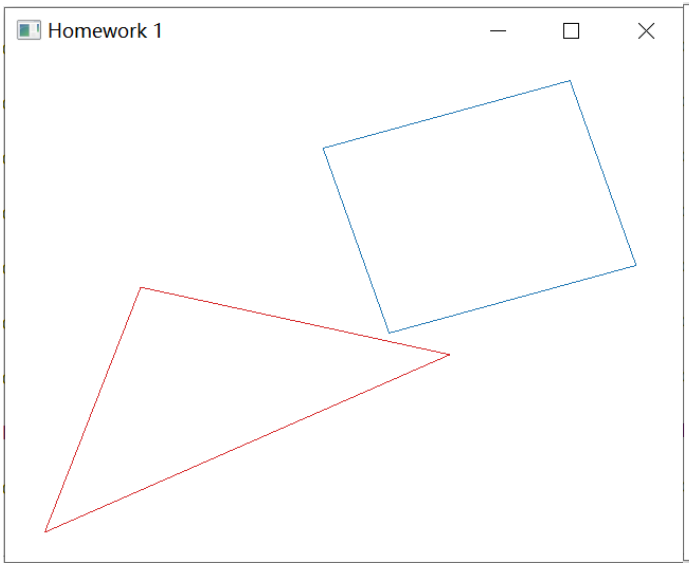
点坐标变换(my_change_point)

- 使用变换矩阵对点坐标进行变换
- 即矩阵-向量乘法
- 较为简单，代码不多加展示

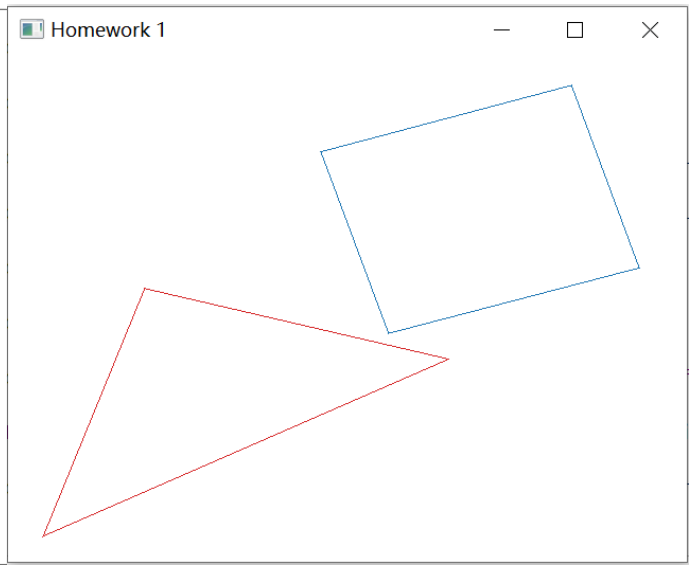
结果展示

---

scene 0

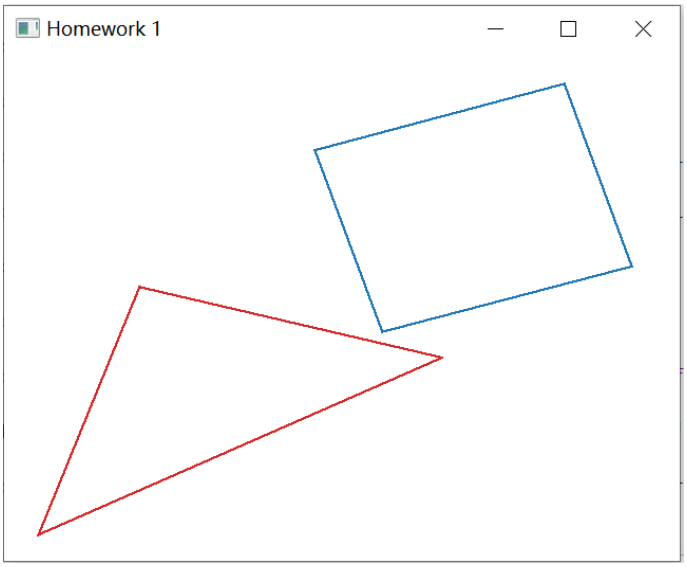


scene 1



- 由于在画线时采用整数运算，所以与scene 0在细微处会有一些差异。

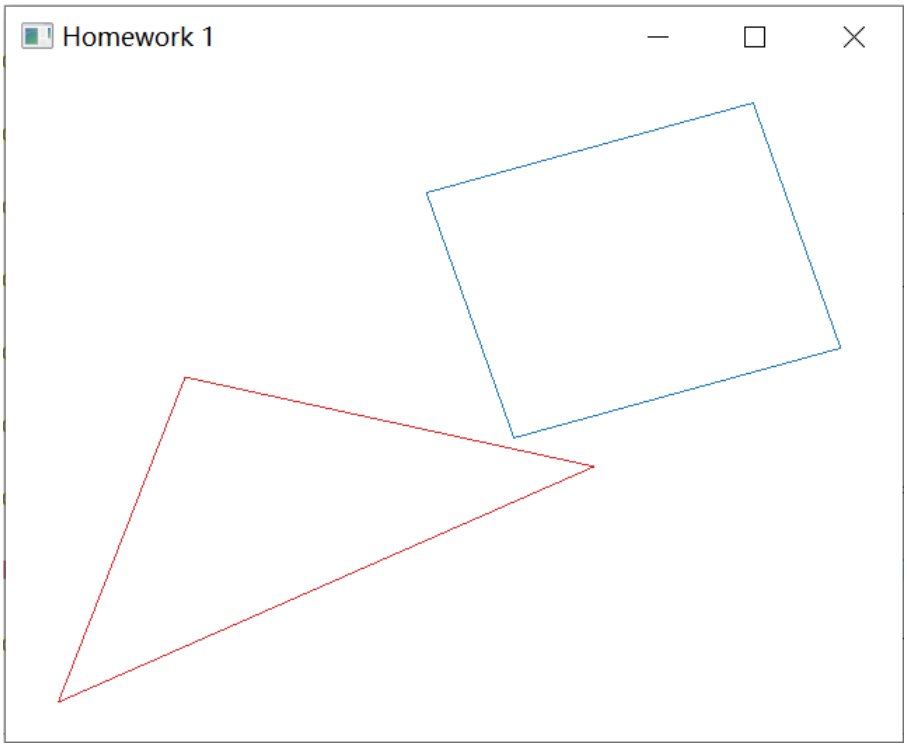
scene 2 (四元数旋转+抗锯齿)
按下按键2即可查看



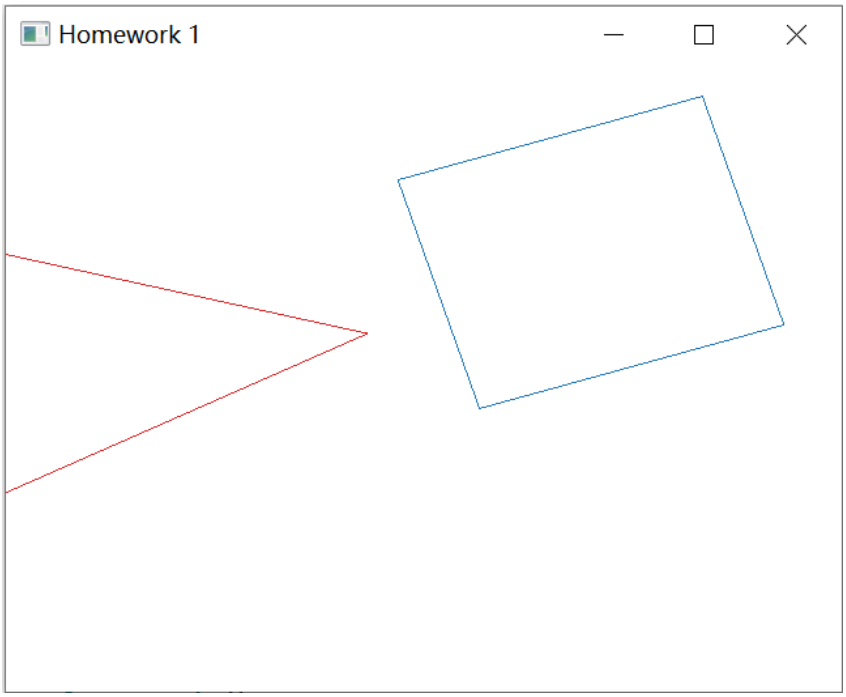
变换顺序对结果影响

- 改变绘制三角形中的平移与旋转顺序
    - 总共是两个平移和一个旋转

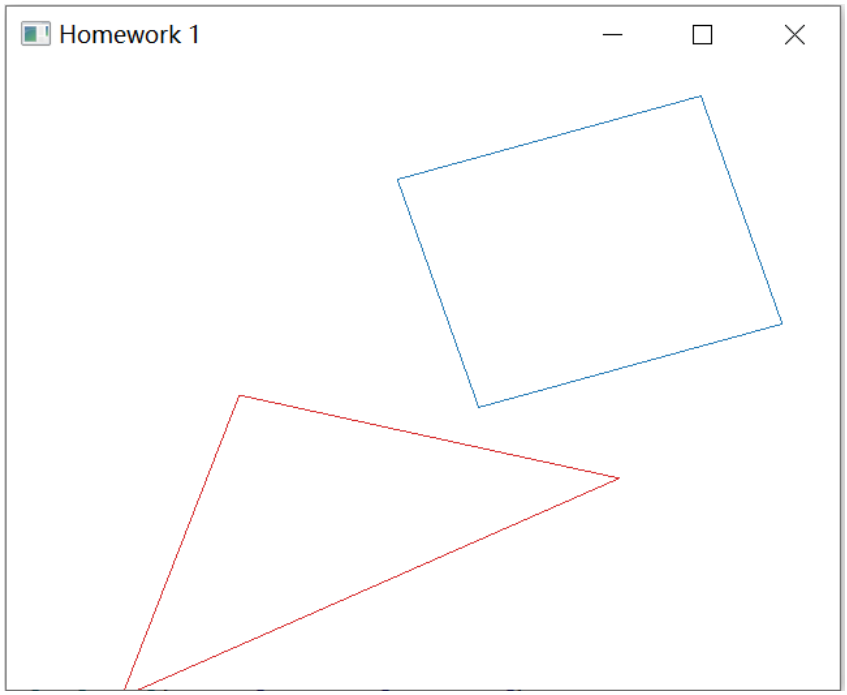
- 初始顺序：translate1--rotate--translate2



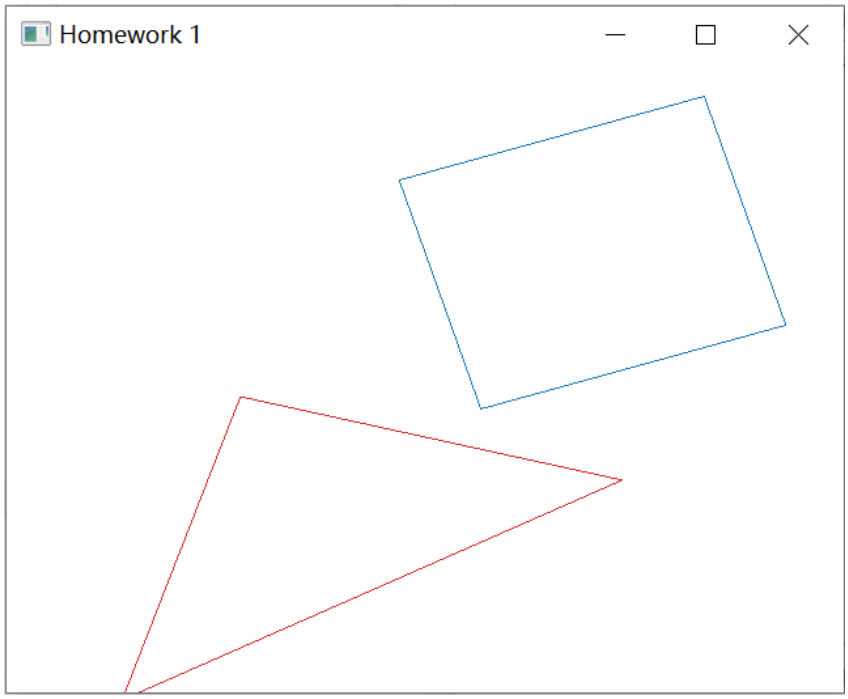
- 现改变顺序：
  - translate1--translate2--rotate



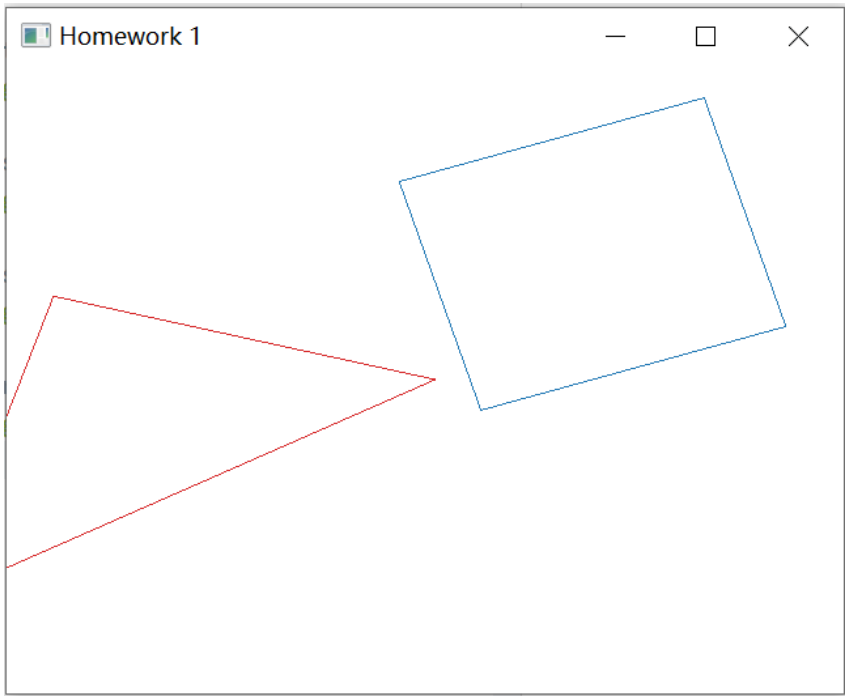
- rotate--translate1--translate2



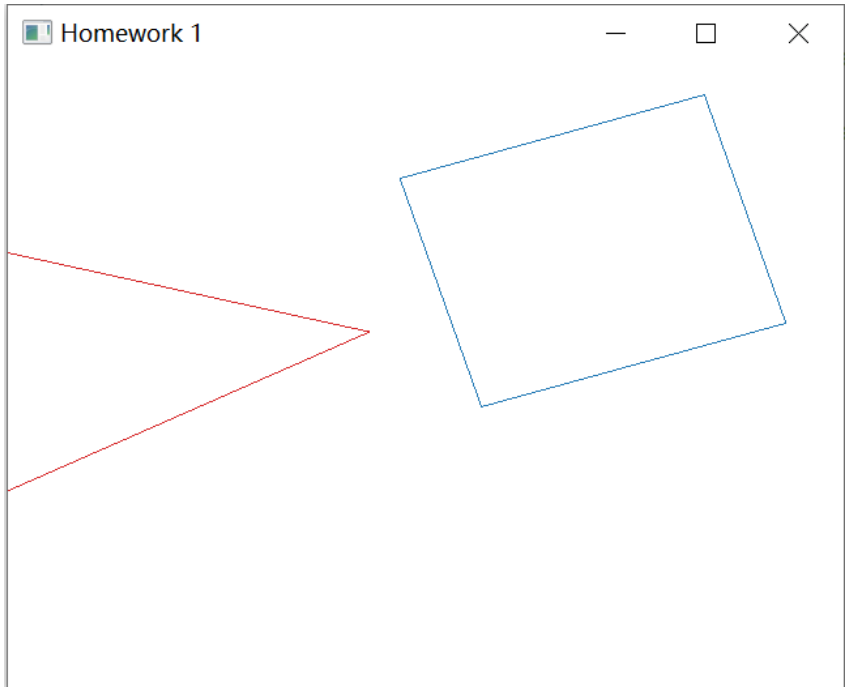
- rotate--translate2--translate1



- translate2--rotate--translate1

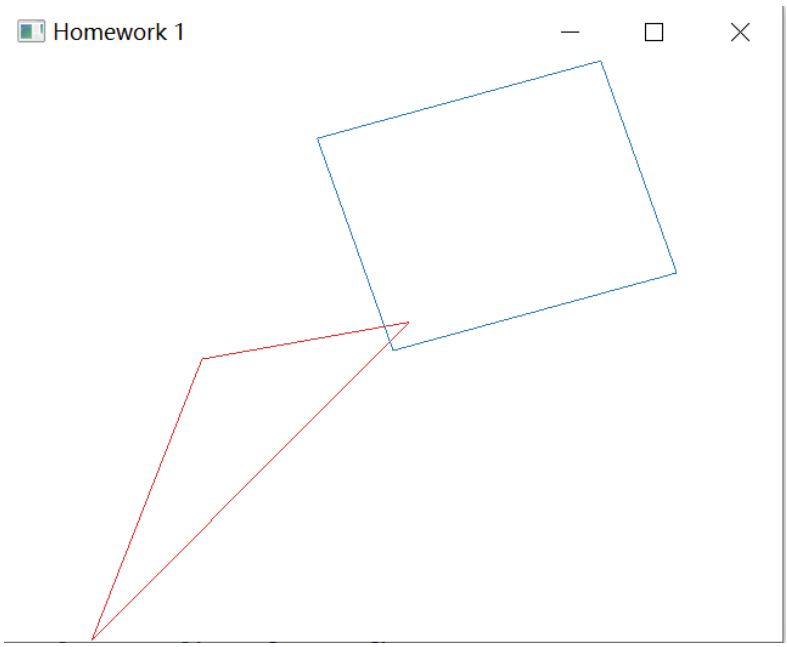


- translate2--translate1--rotate

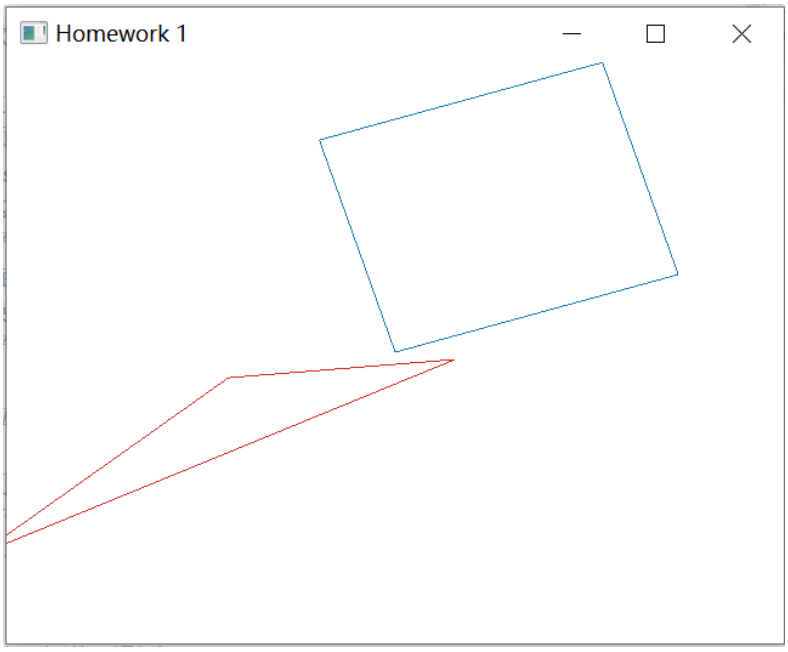


- 增加旋转操作
  - 旋转轴不同

- rotate1--rotate2

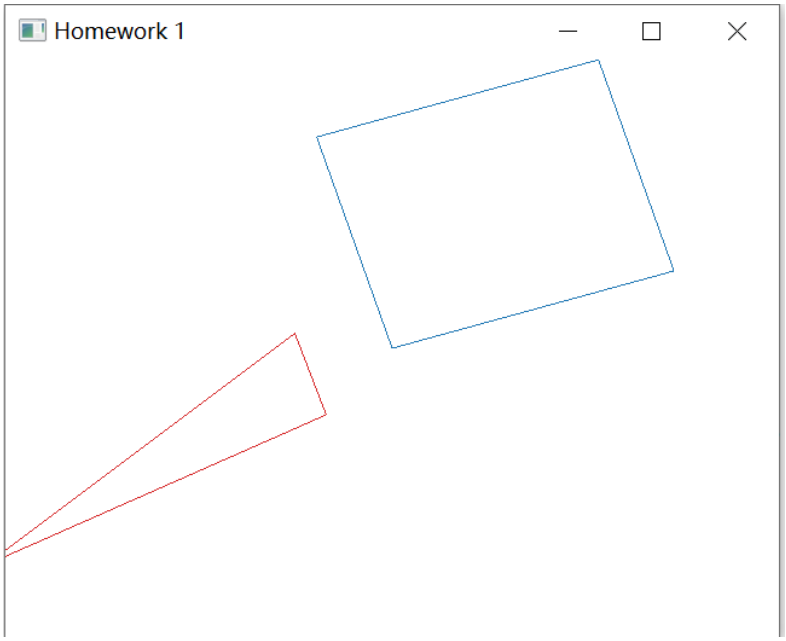


- rotate2--rotate1

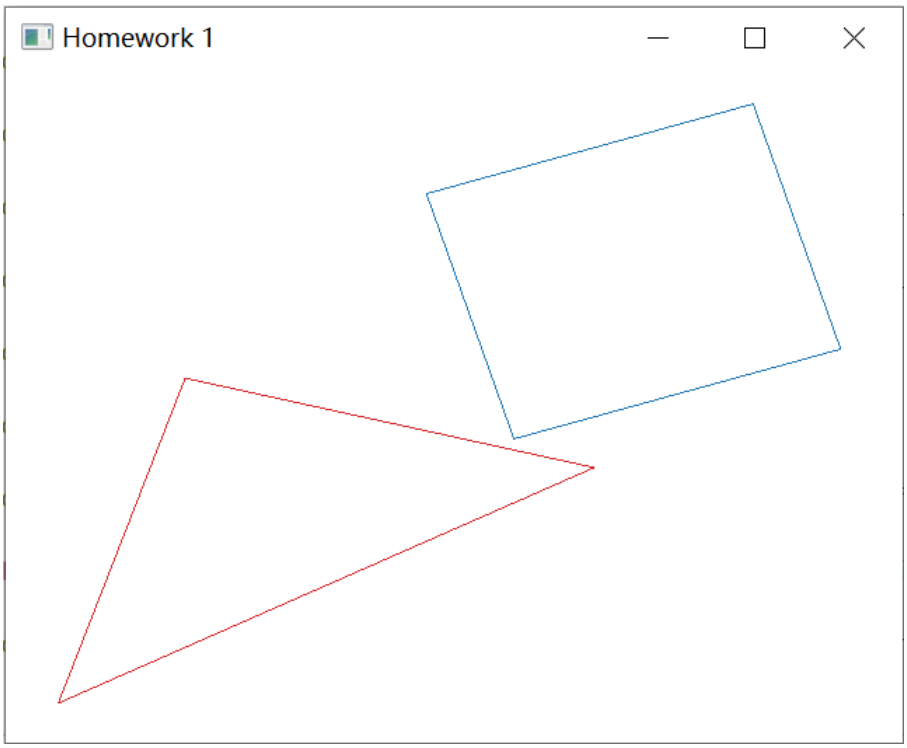


- 旋转轴相同

- rotate1--rotate2/rotate2--rotate1结果一致

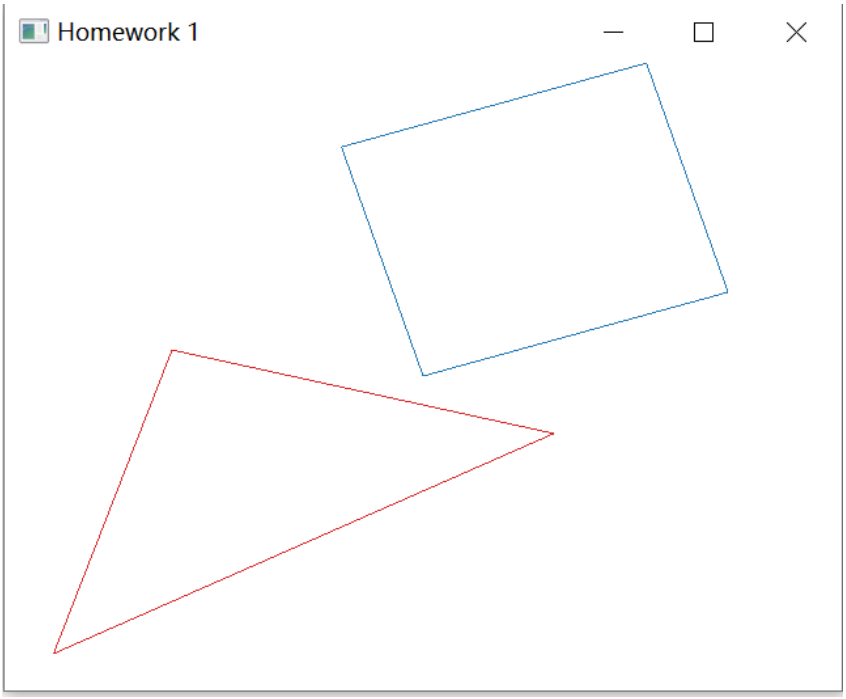


- 改变绘制矩形中的平移与旋转顺序
  - 分别是平移加旋转
  - 初始顺序：translate--rotate



  - 现改变顺序：

    - rotate--translate



- 结果分析
    - 由于矩形只有两次操作，且由于平移单位数较少以及旋转轴的关系，改变顺序造成影响的效果不明显。只能勉强从与三角形的位置关系看出影响，所以这里不对其做深入分析，主要分析三角的顺序影响。
    - 观察三角形不同顺序的结果截图可发现
        - rotate--translate* 2两张截图以及translate*2--rotate两张截图他们各自都是一样的。这说明平移操作在相邻时调换顺序不会影响结果。
        - 相反地，如果平移操作之间存在旋转，调换顺序会影响结果。
        - 而如果不同的旋转操作调换顺序则与平移操作不同，即使操作是相邻的，如果旋转轴不一样，调换顺序也会影响结果，如果旋转轴一样，则不会影响结果。
        - 另外，旋转和平移操作互相调换顺序一般会影响结果，除非平移操作与旋转轴平行。
        - 最后，通过对位置的观察发现，在OpenGL中最后调用的变换最先被应用。
    - Conclusion
        - translate之间
            - 相邻：调换不影响
            - 不相邻(中间存在旋转)：调换影响
        - rotate之间
            - 相邻：
                - 旋转轴一致：调换不影响
                - 旋转轴不一致：调换影响
            - 不相邻：调换影响
        - translate，rotate之间
            - 平移与旋转轴平行：不影响
            - 平移与旋转轴不平行：影响
        - 最后调用的变换最先被应用

心得体会

通过这一次的实验，对库函数实现的GL_lines,GLRotate,GLTranslate等函数的不同实现方法都有了一定程度的了解，对四元数以及抗锯齿也有了具体应用的实现。然后是最后对多个变换调换顺序对结果造成的影响，也对理论课上最后调用的变换最先被应用有了实践上的理解。总的来说，这次实验让我真正走进了OpenGL的世界，获益匪浅。