中山大學
SUN YAT-SEN UNIVERSITY

《计算机图形学》

实验报告

（作业一）

学 院 名 称	：	数据科学与计算机学院
专业（班级）	：	17 计科 7 班
学 生 姓 名	：	张富瑞
学　　号	：	17341197
时　　间	：	2019 年 10 月 31 日

一.实验题目

在作业模板中的scene_0()函数使用了OpenGL函数（如，glBegin(GL_LINE_STRIP)，glTranslatef()及glRotatef()等）绘制一个三角形及一个四边形的场景。同学们需要在不使用以上函数的情况下重现同样场景（在模板中的scene_1()函数汇总实现）。

二.算法原理

1.在实现自己编写的scene_1()之前，为了复刻scene_0()，首先需要搞懂的地方是原本的函数是如何运行的。scene_0()中有几个重要的函数，glTranslatef()以及glRotatef()。经过查询资料，我知道了glTranslatef(x,y,z)是将坐标轴的原点沿(x,y,z)向量进行平移，而glRotatef(angle,x,y,z)是以(x,y,z)向量作为旋转轴旋转angle角度。由于这两个函数都是对于坐标系进行操作，并且在绘制一个点的过程中调用了多个函数。由于我们不能够实现对于整个坐标系的操作，所以我们选择对画出的点进行操作。因为给出了最后操作完成后的坐标系中的要画的点的坐标，所以我们需要对这个点进行操作以达到操作坐标系后的结果。由于对于坐标系的操作是从后到前的顺序，所以在对点的操作时需要从前到后来进行操作。
2.平移操作只需要对坐标进行x,y,z向量的加减即可达到效果。
3.旋转操作，根据老师上课的pdf中给出的矩阵操作，我们进行矩阵的相乘也能够完成。
4.光栅化操作，计算经过平移旋转后的点后，我们只需要运用Bresenham’s algorithm即可完成。

三.关键代码展示

1.平移

```
void trans(float &x,float &y,float &z,float dx,float dy,float dz)
{
    x+=dx;
    y+=dy;
    z+=dz;
}
```

对坐标进行向量的加减。

2.旋转

```
void rotate(float &x,float &y,float &z,float angle,float a,float b,float c)
{
    float len = sqrt(a*a+b*b+c*c);
    a/=len;b/=len;c/=len;
    float sin_val = float(sin(-angle * (PI/180.0)));
    float cos_val = float(cos(angle * (PI/180.0)));
```

```
    float matrix[3][4] = {{a*a*(1-cos_val)+cos_val, a*b*(1-
cos_val)+c*sin_val, a*c*(1-cos_val)-b*sin_val, 0},
                          {a*b*(1-cos_val)-c*sin_val, b*b*(1-
cos_val)+cos_val, b*c*(1-cos_val)+a*sin_val, 0},
                          {a*c*(1-cos_val)+b*sin_val, b*c*(1-
cos_val)-a*sin_val, c*c*(1-cos_val)+cos_val, 0}};
    float t[3];
    for (int i=0; i<3; ++i)
        t[i] = x * matrix[i][0] + y * matrix[i][1] + z * matri
x[i][2] + matrix[i][3];
    x = t[0];
    y = t[1];
    z = t[2];
}
```

思想是根据pdf上给出的矩阵进行相乘，来得到旋转后的坐标。实现的过程需要完成旋转轴的缩放，要将轴的向量模长缩放为1，这是十分容易让人遗忘的地方。之后就是建好矩阵，进行矩阵的乘法。

3.直线光栅化

```
void Drawline(int x1, int y1, int x2, int y2)
{
     int dx = x2 - x1;
     int dy = y2 - y1;
     int ux = (dx > 0) ? 1:-1;
     int uy = (dy > 0) ? 1:-1;
     int x = x1, y = y1, eps = 0;//eps 为累加误差
     dx = abs(dx); dy = abs(dy);
     if (dx > dy)
     {
         for (x = x1; x != x2; x += ux)
         {
              glVertex2i(x,y);
//            printf("%d %d\n",x,y);
              eps += dy;
              if ((eps << 1) >= dx)
              {
                   y += uy; eps -= dx;
              }
         }
     }
     else
     {
```

```
        for (y = y1; y != y2; y += uy)
        {
            glVertex2i(x,y);
            eps += dx;
            if ((eps << 1) >= dy)
            {
                x += ux; eps -= dy;
            }
        }
    }
}
```

这里使用的是Bresenham算法，这也是老师有讲过的重要的算法，伪代码如下：

```
ξ ← 0, y ← y1
For x ← x1 to x2 do
    Plot Point at (x, y)
    If (2(ξ + dy) < dx)
        ξ ←ξ + dy
    Else
        y ← y + 1,ξ ←ξ + dy – dx
    End If
End For
```

算法本质上是对(x+1,y)还是(x+1,y+1)的选择。由于这里不只是有|dx|>|dy|的情况，所以需要特判。同时还要考虑到x,y轴相加的方向上的增量(即x+1还是x-1的增量)。

4.scene_1()

```
void MyGLWidget::scene_1()
{
    glClear(GL_COLOR_BUFFER_BIT);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(0.0f, width(), 0.0f, height(), -1000.0f, 1000.0f);

    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();

    //glBegin(GL_POINTS);
 //   glPointSize(2);
   // glBegin(GL_LINE_LOOP);
    //your implementation
    //glVertex2i()
    glColor3f(0.839f, 0.153f, 0.157f); //rgb
    float node[4][4] = {0.0};
```

```
    for (int i = 0;i < 4;i++){
        for (int j = 0;j < 4;j++){
            printf("%f",node[i][j]);
        }
    }
    for(int i = 1;i <= 3;i++)node[i][3] = 0.0;
    node[1][1] = 10.0;node[1][2] = 10.0;
    node[2][1] = 50.0;node[2][2] = 50.0;
    node[3][1] = 80.0;node[3][2] = 10.0;
    for(int i = 1;i <= 3;i++){
        trans(node[i][1],node[i][2],node[i][3],-50.0f, -
30.0f, 0.0f);
    }
    for(int i = 1;i <= 3;i++){
        rotate(node[i][1],node[i][2],node[i][3],45.0f, 1.0f, 0
.0f, 1.0f);
    }
    for(int i = 1;i <= 3;i++){
        trans(node[i][1],node[i][2],node[i][3],-20.0f, -
10.0f, 0.0f);
    }
    for(int i = 1;i <= 3;i++){
        trans(node[i][1],node[i][2],node[i][3],50.0f, 50.0f, 0
.0f);
    }
    for(int i = 1;i <= 3;i++){
        printf("node%d: %f %f %f\n",i,node[i][1],node[i][2],no
de[i][3]);
    }
    glBegin(GL_POINTS);
    Drawline(int(node[1][1]*width()/100),int(node[1][2]*height
()/100),int(node[2][1]*width()/100),int(node[2][2]*height()/10
0));
    Drawline(int(node[1][1]*width()/100),int(node[1][2]*height
()/100),int(node[3][1]*width()/100),int(node[3][2]*height()/10
0));
    Drawline(int(node[3][1]*width()/100),int(node[3][2]*height
()/100),int(node[2][1]*width()/100),int(node[2][2]*height()/10
0));
    glEnd();

    //
```

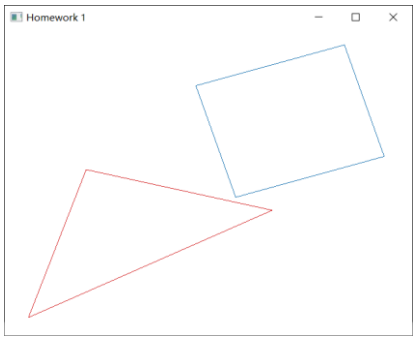
```
    glColor3f(0.122f, 0.467f, 0.706f);
    float node1[5][5] = {0.0};
    for(int i = 1;i <= 4;i++)node1[i][3] = 0.0;
    node1[1][1] = -20.0f;node1[1][2] = -20.0f;
    node1[2][1] = 20.0f;node1[2][2] = -20.0f;
    node1[3][1] = 20.0f;node1[3][2] = 20.0f;
    node1[4][1] = -20.0f;node1[4][2] = 20.0f;
    for(int i = 1;i <= 4;i++)rotate(node1[i][1],node1[i][2],no
de1[i][3],30.0f, 1.0f, 1.0f, 1.0f);
    for(int i = 1;i <= 4;i++)trans(node1[i][1],node1[i][2],nod
e1[i][3],20.0f, 20.0f, 0.0f);
    for(int i = 1;i <= 4;i++)trans(node1[i][1],node1[i][2],nod
e1[i][3],50.0f, 50.0f, 0.0f);
    glBegin(GL_POINTS);
    Drawline(int(node1[1][1]*width()/100),int(node1[1][2]*heig
ht()/100),int(node1[2][1]*width()/100),int(node1[2][2]*height(
)/100));
    Drawline(int(node1[2][1]*width()/100),int(node1[2][2]*heig
ht()/100),int(node1[3][1]*width()/100),int(node1[3][2]*height(
)/100));
    Drawline(int(node1[3][1]*width()/100),int(node1[3][2]*heig
ht()/100),int(node1[4][1]*width()/100),int(node1[4][2]*height(
)/100));
    Drawline(int(node1[4][1]*width()/100),int(node1[4][2]*heig
ht()/100),int(node1[1][1]*width()/100),int(node1[1][2]*height(
)/100));
    glEnd();
}
```

这是scene_1()的实现代码，根据scene_0()中模型变换的相反顺序进行实现。同时为了达到老师所要求的图像，我将视景体的规格由100x100改为了视图的高度和宽度，使绘图点更加密集，效果更可观。相应的，我将端点的值也进行了放大。

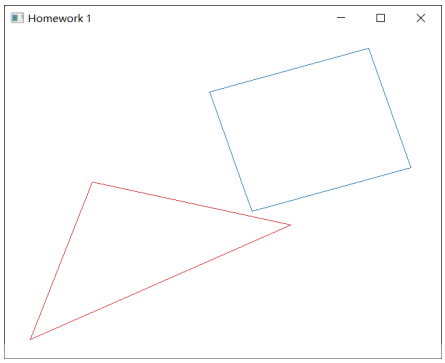
四.效果展示

1.

scene_0()原图



scene_1()



可以看出基本复刻了scene_0()中图形。

2.模型变换顺序的变换

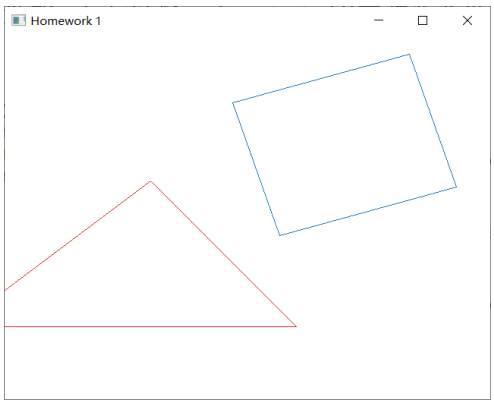
①如果模型变换中没有旋转的变换，单纯只有平移的变换，改变变换的顺序没有影响。

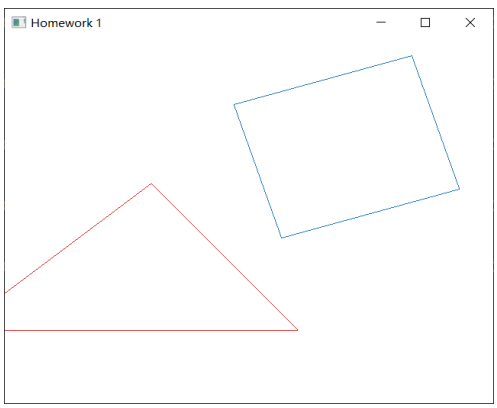
Test：我将scene_0()中画三角形的rotate注释，在scene_1()中改变变换的顺序。

```
//draw a triangle
glPushMatrix();
glColor3f(0.839f, 0.153f, 0.157f);
glTranslatef(-20.0f, -10.0f, 0.0f);
glRotatef(45.0f, 1.0f, 0.0f, 1.0f);
glTranslatef(-50.0f, -30.0f, 0.0f);
glBegin(GL_LINE_LOOP);
glVertex2f(10.0f, 10.0f);
glVertex2f(50.0f, 50.0f);
glVertex2f(80.0f, 10.0f);
glEnd();
glPopMatrix();
```

```
for(int i = 1;i <= 3;i++){
    trans(node[i][1],node[i][2],node[i][3],50.0f, 50.0f, 0.0f);
}
for(int i = 1;i <= 3;i++){
    trans(node[i][1],node[i][2],node[i][3],-50.0f, -30.0f, 0.0f);
}
/*
for(int i = 1;i <= 3;i++){
    rotate(node[i][1],node[i][2],node[i][3],45.0f, 1.0f, 0.0f, 1.0f);
}
*/
for(int i = 1;i <= 3;i++){
    trans(node[i][1],node[i][2],node[i][3],-20.0f, -10.0f, 0.0f);
}
```

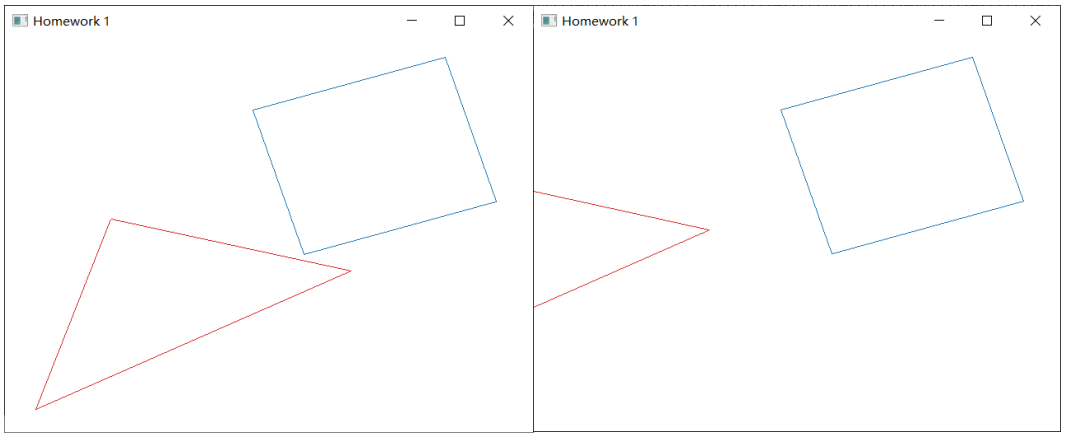
效果图：





可以看出两个图一模一样，证明没有影响。

②中间有旋转变换的加入，改变scene_1()中的顺序



可以看出不再相同，所以变换的顺序有了旋转的加入变得不一样了，不能够随意更改变换的顺序。