厦門大學

计算机图形学实验

实验 1、OpenGL 初步

姓　　名：黄子安

学　　号：22920212204396

学　　院：信息学院

专　　业：软件工程

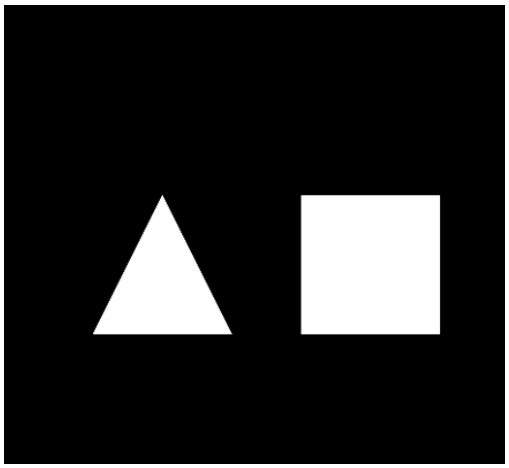
年　　级：2021 级

2023 年 3 月 24 日

目录

Task1: 示例 OpenGL 程序

1. 运行结果



2. 学习红宝书并思考

（1）回调函数是什么意思？GLUT 中有哪些回调函数？

回调函数是在程序执行过程中由系统调用的函数。在 OpenGL 和 GLUT 中，回调函数是指在特定事件发生时自动调用的函数。例如，当窗口需要重绘或一个鼠标事件发生时，系统会自动调用相应的回调函数。

GLUT 中的一些常用回调函数包括：

- glutDisplayFunc：当需要重新绘制窗口内容时调用。程序需要在这个回调函数中定义绘制图像的操作
- glutIdleFunc：当 OpenGL 没有任务时（例如窗口已经显示并等待用户事件）调用。在这个回调函数中，可以设置 OpenGL 在闲置时间运行的操作
- glutReshapeFunc：当窗口大小或形状改变时调用。程序需要在这个回调函数中重新定义视图参数，以适应新的窗口大小

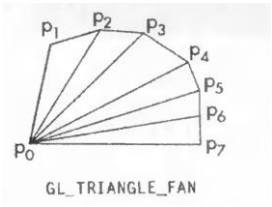
（2）用鼠标改变窗口大小会发生什么？哪个函数在影响整个过程？

答：当使用鼠标改变窗口大小时，窗口的大小和位置会发生变化，而 OpenGL 的坐标系也会相应地发生变化。glutReshapeFunc 函数被调用来重新定义 OpenGL 窗口视图的参数。在这个回调函数里，需要重新计算视图矩阵和投影矩阵。这些矩阵的改变会影响整个 OpenGL 渲染过程

Task2: 画一个圆

1. 绘制一个圆

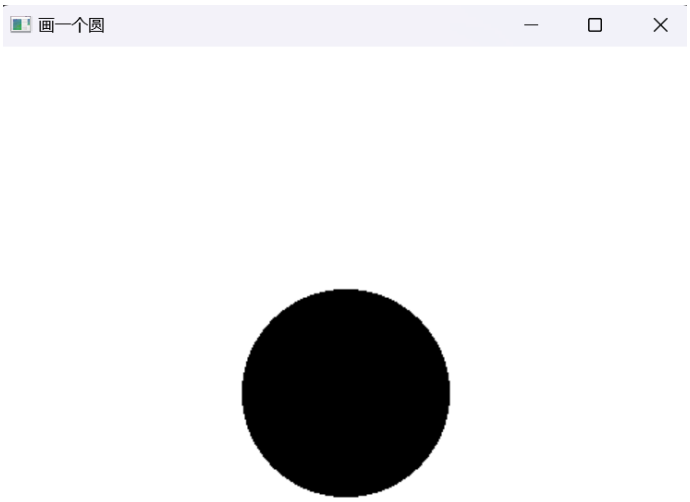
操作流程：采用极坐标的形式利用多个三角形进行以直代曲，因为该任务还需要对扇形进行着色，所以直接不采用正多边形进行拟合的效果，而是采用三角形拟合扇形再拼接成圆形，使用 glBegin 函数的 GL_TRIANGLE_FAN 填充方式绘制一系列首尾相接的四边形来近似代替圆环



关键代码截图：

```
glBegin(GL_TRIANGLE_FAN);
glVertex2d(0, 0);
for (int i = 0;i < n;++i)//n表示分成几个大的扇形
{
    int nn = 1000;//n表示一个扇形由多少个三角形进行近似
    double T = 2 * pi / n;//T表示每个扇形对应的圆周角大小
    glColor3f(0, 0, 0);
    for (int j = 0;j <= nn;++j)
    {
        double theta =i*T+ j*T / nn;
        double x = xx+r*cos(theta);
        double y = yy+r*sin(theta);
        glVertex2d(x, y);
    }
}
glEnd();
```

运行结果截图：



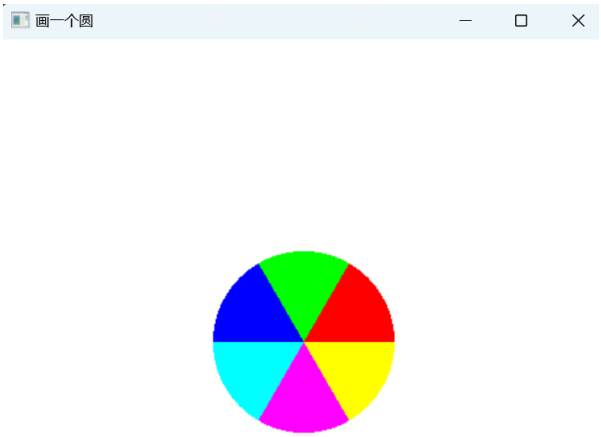
2.对扇形进行着色

操作流程：利用二维数组存储对应颜色的 RBG 值，之后调用 glColor3f 函数实现对颜色切换，采用循环实现对颜色数组遍历，在绘制不同扇形时采用不同的颜色

关键代码截图：color 数组存储对应的颜色值，在绘制扇形时根据 i 的数值更改着色来实现扇形颜色的改变

```
void display()
{
    glClear(GL_COLOR_BUFFER_BIT);
    glShadeModel(GL_FLAT);
    glRotatef(spin, 0, 0, 1);
    glPushMatrix();
    int n = 6;
    double r = 0.3;//r表示半径
    double xx = 0, yy = 0;//x,y表示圆心坐标
    double color[][3] = { {1.0,0,0},
                          {0,1.0,0},
                          {0,0,1.0},
                          {0,1.0,1.0},
                          {1.0,0,1.0},
                          {1.0,1.0,0}
    };
    glBegin(GL_TRIANGLE_FAN);
    glVertex2d(0, 0);
    for (int i = 0;i < n;++i)//n表示分成几个大的扇形
    {
        int nn = 1000;//n表示一个扇形由多少个三角形进行近似
        double T = 2 * pi / n;//T表示每个扇形对应的圆周角大小
        glColor3f(color[i][0], color[i][1], color[i][2]);
        for (int j = 0;j <= nn;++j)
        {
            double theta =i*T+ j*T / nn;
            double x = xx+r*cos(theta);
            double y = yy+r*sin(theta);
            glVertex2d(x, y);
        }
    }
    glEnd();
    glFlush();
    glPopMatrix();
    glutSwapBuffers();
}
```

运行结果截图：



3.实现圆的旋转

操作流程：利用 openGL 中的 glrotationf 改变物体的角度实现旋转效果，同时需要注册 glutIdleFunc 函数，实现当窗口空闲时自动调用重新绘制函数

关键代码截图：

1. 在 main 函数中注册 glutIdleFunc 函数，此时当窗口处于空闲时便会自动调用 spinDisplay 函数

```
int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
    glutInitWindowSize(500, 500);
    glutInitWindowPosition(500, 100);
    glutCreateWindow("画一个圆");
    glutDisplayFunc(display);
    glutIdleFunc(spinDisplay);
    init();
    glutMainLoop();
}
```

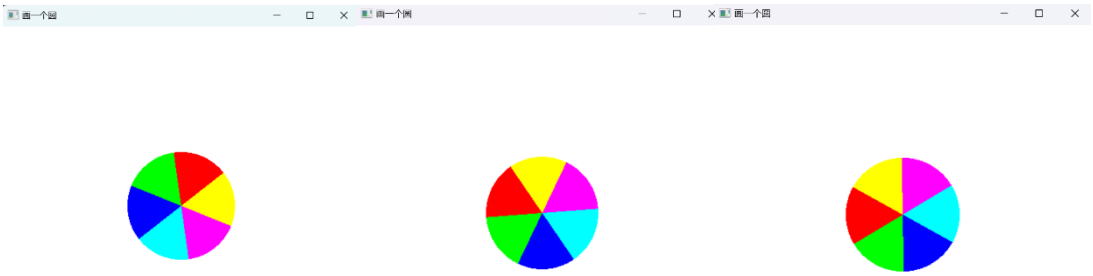
2. 在 spinDispaly 函数中调用 glutPostRedisplay 函数，对 openGL 发出信号此时需要对图像进行重新绘制(后续可以在该函数中实现调整角速度)

```
void spinDisplay()
{
    glutPostRedisplay();
}
```

3. 在 display 函数中加入函数 glRotatef，其中 spin 表示旋转的角度，第四个参数为 1 表示围绕 z 轴转动，在 openGL 窗口中 z 轴垂直于电脑屏幕，满足我们需要

```
void display()
{
    glClear(GL_COLOR_BUFFER_BIT);
    glShadeModel(GL_FLAT);
    glRotatef(spin, 0, 0, 1);
    glPushMatrix();
    int n = 6;
    double r = 0.3;//r表示半径
    double xx = 0, yy = 0;//x,y表示圆心坐标
    double color[][3] = { {1.0,0,0},
                          {0,1.0,0},
                          {0,0,1.0},
                          {0,1.0,1.0},
                          {1.0,0,1.0},
                          {1.0,1.0,0}
    };
```

运行结果截图： 圆实现围绕圆心匀速转动



Task3: 画奥运五环

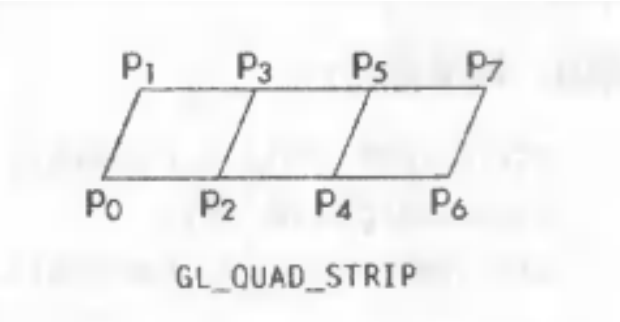
1. 绘制五环（不考虑交叉）

操作流程：

1. 颜色处理：通过查询获取五个环颜色的 RGB 精确值，之后除以 255 作为 glColor3f 的参数

颜色	R	G	B
蓝色	0	107	176
黄色	239	169	13
黑色	29	24	21
绿色	5	147	65
红色	220	47	31

2. 坐标处理：根据奥运五环“上面三环下面两环，圆环两两相交”的特点设置圆心的坐标
3. 圆环绘制 ：使用 glBegin 函数的 GL_QUAD_STRIP 填充方式绘制一系列首尾相接的四边形来近似代替圆环，绘制每个四边形四个点的时候采用极坐标的形式，同时通过设置数组和循环更改每一个环的颜色和位置



关键代码截图：

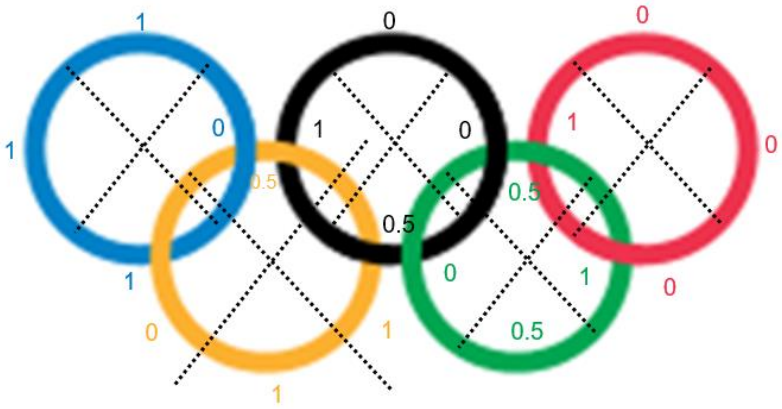
```
//颜色集合
GLfloat color[][3] = { {0,107.0/255,176.0/255},
                       {239.0/255,169.0/255,13.0/255},
                       {29.0/255,24.0/255,21.0/255},
                       {5.0/255,147.0/255,65.0/255},
                       {220.0/255,47.0/255,31.0/255},//蓝黄黑绿红
                     };
//圆心集合
GLfloat p[][2] = { {-0.45,0.2},{-0.25,-0.02},{0,0.2},{0.25,-0.02},{0.45,0.2} };

void display()
{
    glClear(GL_COLOR_BUFFER_BIT);
    for (int i = 0;i < 5;++i)
    {
        glColor3f(color[i][0], color[i][1], color[i][2]);
        const int n = 1000;
        float r[] = { 0.175,0.2 };
        float delta_theta = 2 * pi / n;
        glBegin(GL_QUAD_STRIP);
        for (int k = 0;k <= n;++k)
        {
            for (int j = 0;j < 2;++j)
            {
                float x = p[i][0] + r[j] * cos(delta_theta * k);
                float y = p[i][1] + r[j] * sin(delta_theta * k);
                glVertex2f(x, y);
            }
        }
        glEnd();
    }
    glFlush();
}
```

运行结果截图：

2.实现圆环交叉

操作流程： 利用深度测试功能，为了利用之前的颜色数组和循环一次性画出五个圆环，考虑到奥运五环的实际交叉效果，在这里建立一个深度数组，将每个圆按照 45° 初相分割成四个部分，之后对每一段匹配按照下图的数值匹配上不同的，匹配方式满足被覆盖的部分深度值要大于覆盖部分，如果一段扇环即覆盖别的部分又被别的部分覆盖则将深度值取为 0.5，之后将这些深度值存储到数组即可



注意点：

1. **深度测试的 glEnable 需要放在 display 中**：深度测试需要在每一帧渲染前启用，并在每个绘制调用之前进行设置。因此，通常将 glEnable 命令放在 display 回调函数中，因为该函数在每帧绘制时都会被调用。这样可以确保深度测试在每个绘制调用中都是启用的。如果将 glEnable 命令放在 main 函数中，则只会启用一次深度测试。这意味着在后续的绘制调用中，深度测试将保持禁用状态，从而导致深度排序和遮挡问题
2. **需要建立圆心角 theta 和四个深度值的对应关系**，根据坐标变换和周期知识，可以将利用下述公式

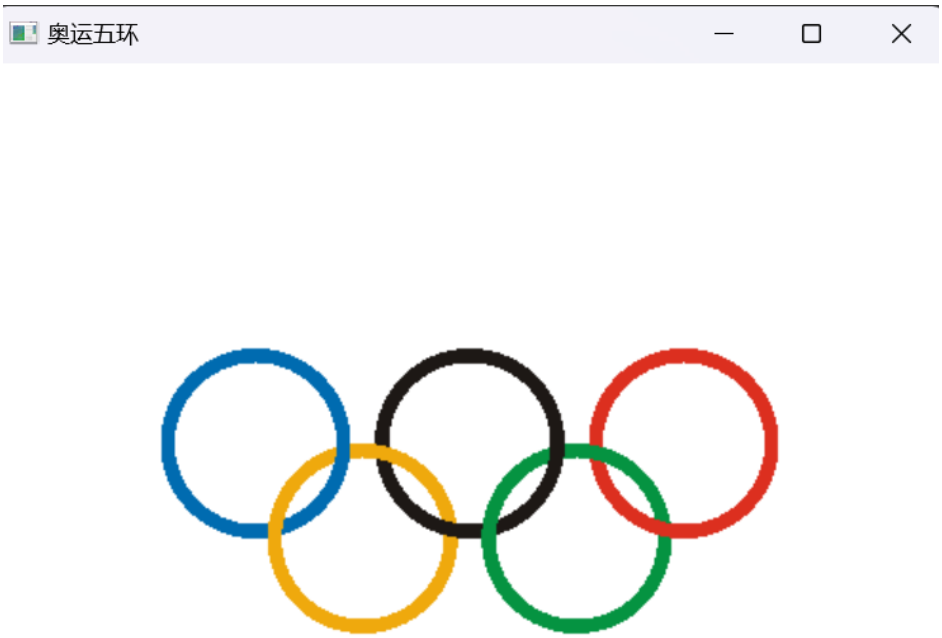
$$part = \frac{\left(k + \frac{n}{8}\right) \ mod \ n}{\frac{n}{4}}，part = 0,1,2,3$$

建立对应关系，其中 n 是对圆环的分割份数，$\frac{n}{8}$ 是给分割部分添加一个初相，使得可以一一对应

关键代码截图：

```
GLfloat p[][2] = { {-0.45,0.2},{-0.225,-0},{0,0.2},{0.225,-0},{0.45,0.2} };
GLfloat depth[][4] = { {0,1,1,1},{1,0.5,0,1},{0,0,1,0.5},
                       {1,0.5,0,0.5},{0,0,1,0} };
void display()
{
    glEnable(GL_DEPTH_TEST);
    glDepthFunc(GL_LEQUAL);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    for (int i = 0;i < 5;++i)
    {
        glColor3f(color[i][0], color[i][1], color[i][2]);
        const int n = 1000;
        float r[] = { 0.17,0.2 };
        float delta_theta = 2 * pi / n;
        glBegin(GL_QUAD_STRIP);
        for (int k = 0;k <= n;++k)
        {
            int part = ((k + n / 8) % n) / (n / 4);
            for (int j = 0;j < 2;++j)
            {
                float x = p[i][0] + r[j] * cos(delta_theta * k);
                float y = p[i][1] + r[j] * sin(delta_theta * k);
                glVertex3f(x, y,depth[i][part]);
            }
        }
        glEnd();
    }
    glFlush();
    glutSwapBuffers();
}
```

运行结果截图：



3.让五环在窗口大小改变后不变形

操作流程：需要注册一个窗口大小改变函数，在其中需要设置一个正交投影矩阵来实现预期的效果，正交投影矩阵会将场景中的物体按照三维空间中的位置和大小以固定的比例投射到二维视口中，因此窗口大小的改变不会影响物体的形状

关键代码截图：

```
void reshape(int w, int h)
{
    glViewport(0, 0, w, h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    if (w <= h)
    {
        // 窗口的宽度小于或等于高度
        glOrtho(-1.0f, +1.0f, -1.0f * (GLfloat)h / (GLfloat)w, 1.0f * (GLfloat)h / (GLfloat)w,1.0f,-1.0f);
    }
    else
    {
        // 窗口的宽度大于高度
        glOrtho(-1.0f * (GLfloat)w / (GLfloat)h, 1.0f * (GLfloat)w / (GLfloat)h, -1.0f, 1.0f,1.0f,-1.0f);
    }

    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}
```

```
glutReshapeFunc(reshape);
```

代码详细解读:

1. glViewport 函数设置了 OpenGL 视口的大小，视口就是窗口中实际绘制部分，在窗口大小发生变化时，需要重新设置视口大小，以确保渲染结果能够适应新的窗口大小
2. glMatrixMode(GL_PROJECTION) 设置矩阵模式为投影矩阵，将 3 维的模型投影到二维
3. glLoadIdentity()函数将当前矩阵重置为单位矩阵（只有主对角线为 1，其余为 0 的矩阵），清除之前的变换效果，使得接下来的变换不会受到之前变换的影响
4. glOrtho()函数窗口根据长度和宽度的大小，将边界按照窗口的 比例进行计算，保证在窗口大小改变时图形的比例可以保持不变

运行结果截图： 当窗口的形状发生改变时五环的形状不会发生变化