计算机图形学上机报告

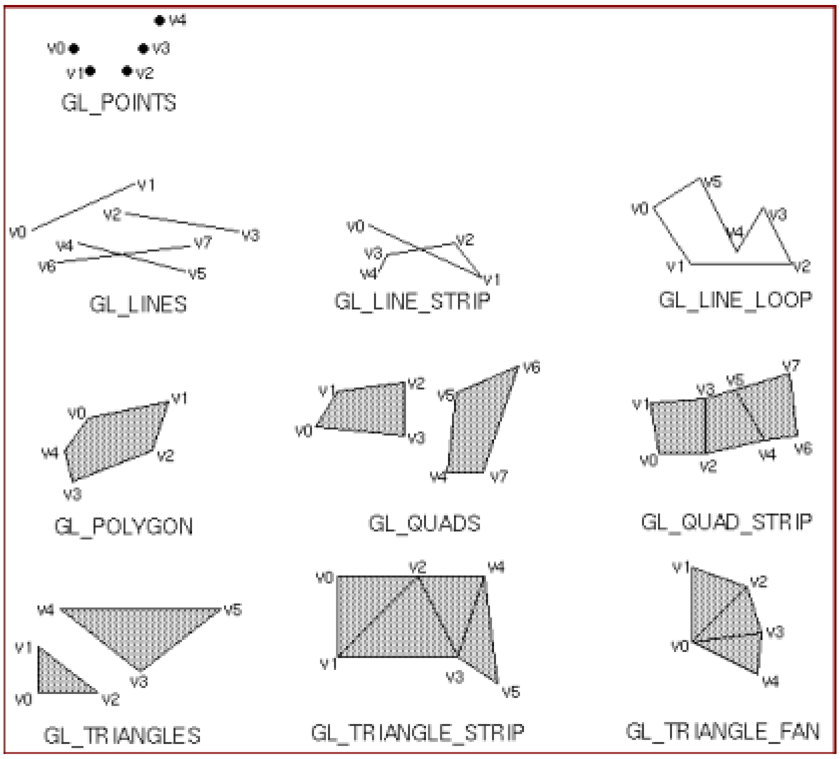
学号：**19049100002**　　　　姓名：张泽群　　　　任课老师：罗楠

1. 第一次上机作业

第一次上机作业

一、简单图元的绘制（借鉴 lec3 课件）

1. 在屏幕上绘制几何图元（如下图所示），自定义坐标和颜色。



二、算法模拟题（借鉴 lec3 课件，可二选一实现）

1. **采用中点线算法在屏幕上画一条直线。**
2. **采用中点圆算法在屏幕上画一个圆。**

1.1 简单图元的绘制

1.1.1 实现思路与函数使用

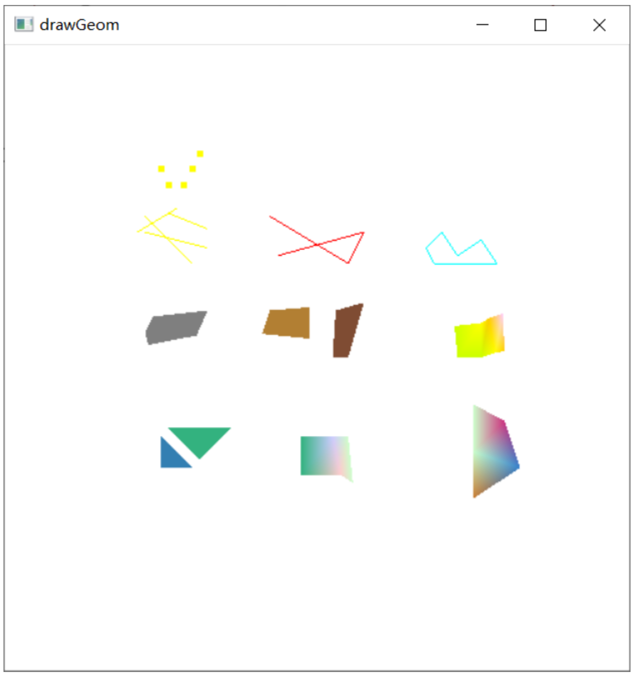
初始化状态以后，在渲染场景RenderScene函数中多次调用glBegin(*GLenum mode*)与glEnd()一对函数，根据以上显示的图形类型参数、点的个数以及几何图元个点的位置坐标进行图元的绘制，其中也可设置颜色等状态。

部分函数如下：

```
void DrawMyObjects()
{
    //画点
    glPointSize(5.0);
    glBegin(GL_POINTS);
    glColor3f(1.0, 1.0, 0.0);
    glVertex2f(-100.0, 120.0);
    glVertex2f(-95.0, 110.0);
    glVertex2f(-80.0, 120.0);
    glVertex2f(-85.0, 110.0);
    glVertex2f(-75.0, 130.0);
    glEnd();

    ...

    //画扇形三角形
    glBegin(GL_TRIANGLE_FAN);
    glVertex2f(100.0, -60.0);
    glVertex2f(100.0, -30.0);
    glColor3f(0.8, 0.2, 0.5);
    glVertex2f(120.0, -40.5);
    glColor3f(0.2, 0.5, 0.8);
    glVertex2f(130.0, -70.5);
    glColor3f(0.8, 0.5, 0.2);
    glVertex2f(100.0, -90.0);
    glEnd();
}
```

1.1.2 结果呈现

实现效果如下所示：



1.2 中点线算法

1.2.1 实现思路与函数使用

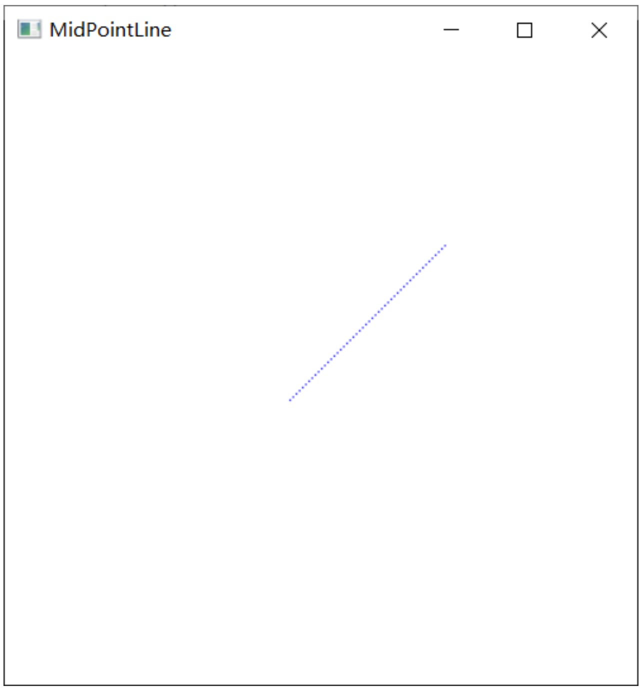
即根据中点线算法，我们将x取为步长方向，则初始值$d = dx - 2dy$，中点M在直线下方时，增量$d1 = 2dy - 2dy$；中点M在直线上方时，增量$d2 = -2dy$。根据算法过程写出相应代码，并将线上每一个得出的点存储在一个point类型的数组中，并利用glBegin()与glEnd()绘图函数(参数为 *GL_POINTS*)将每一个点画在场景中即可得到一条对应两点的直线。

主要函数部分如下所示：

```
int MidPointLine(int x1, int y1, int xn, int yn, point pixels[])
{
    int num, x, y, dx, dy,d,d1,d2;
    x = x1; y = y1;
    dx = xn - x1;
    dy = yn - y1;
    num = 0;
    d = dx - 2*dy;
    d1 = 2*dx - 2*dy;
    d2 = -2*dy;

    while (x < xn) {
        if (d < 0) {
            pixels[num].x = x;
            pixels[num].y = y;
            x++;
            y++;
            d += d1;
            num++;
        }
        else {
            pixels[num].x = x;
            pixels[num].y = y;
            x++;
            d += d2;
            num++;
        }
    }
    pixels[num].x = x;
    pixels[num].y = y;

    return num;
}

void drawLine(int x1, int y1, int x2, int y2)
{
    point pixels[100];
    int num;
    int i;
    num = MidPointLine(x1, y1, x2, y2, pixels);
    glBegin(GL_POINTS);
    for (i = 0; i < num; i++)
        glVertex2f(pixels[i].x, pixels[i].y);    //表示一个空间顶点
    glEnd();
}
```

1.2.2 结果呈现

实现效果如下所示：



1.3 中点圆算法

1.3.1 实现思路与函数使用

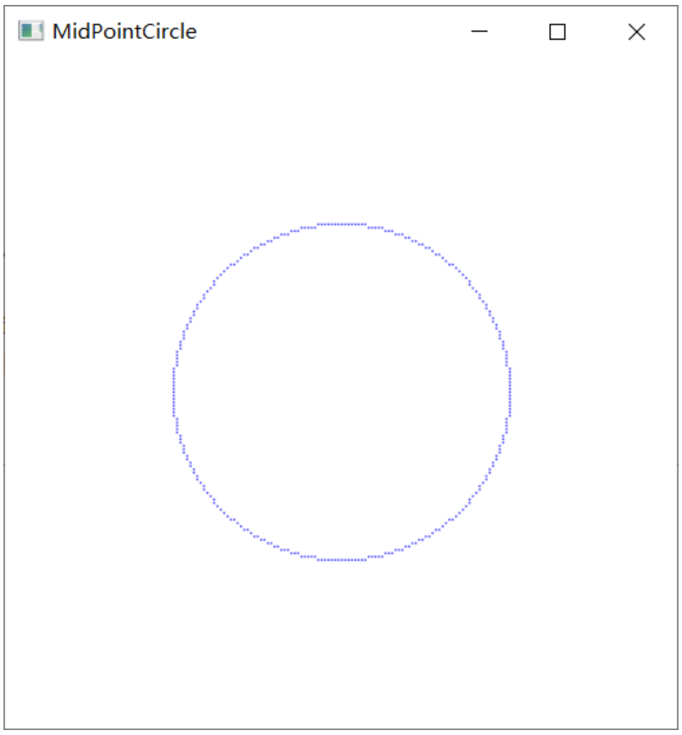
首先利用圆的八路对称方法可以写出函数CirclePoint(),可以只需考虑八分之一圆弧。根据中点圆算法，初始值d0=1.25-r,中点M在圆内时，增量$d1 = 2x+3$；中点M在圆外时，增量$d2 = 2x-2y+5$。根据算法过程写出相应代码，并将线上每一个得出的点存储在一个point类型的数组中，并利用glBegin()与glEnd()绘图函数(参数为 *GL_POINTS*)将每一个点画在场景中即可得到一条对应原点与半径r的中点圆。

主要函数部分如下所示：

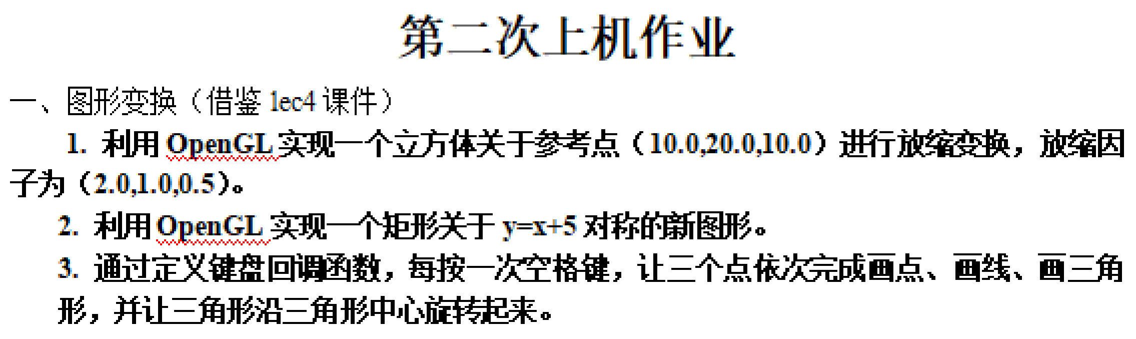
```
int CirclePoint(int x1, int y1, int x, int y,int num, point pixels[]) {
    pixels[num].x = x1+x;
    pixels[num].y = y1+y;
    num++;
    pixels[num].x = x1 + y;
    pixels[num].y = y1 + x;
    num++;
    pixels[num].x = x1  - x;
    pixels[num].y = y1 + y;
    num++;
    pixels[num].x = x1  - y;
    pixels[num].y = y1 + x;
    num++;;
    pixels[num].x = x1 - x;
    pixels[num].y = y1 - y;
    num++;
    pixels[num].x = x1 - y;
    pixels[num].y = y1 - x;
    num++;
    pixels[num].x = x1 + x;
    pixels[num].y = y1 - y;
    num++;
    pixels[num].x = x1 + y;
    pixels[num].y = y1 - x;
    num++;

    return num;
}

int MidPointCircle(int x1, int y1, int r, point pixels[])
{
    int num=0;
    int x = 0;
    int y = r;
    int d = 1 - r;
    num = CirclePoint(x1, y1, x, y, num, pixels);
    while (x < y) {
        if (d < 0)
            d += 2 * x + 3;
        else {
            d += 2 * (x - y) + 5;
            y--;
        }
        x++;
        num = CirclePoint(x1, y1, x, y, num, pixels);
    }

    return num;
}

void drawLine(int x1, int y1, int r)
{
    point pixels[10000];
    int num=0;
    int i;
    num = MidPointCircle(x1, y1, r, pixels);
    glBegin(GL_POINTS);
    for (i = 0; i < num; i++)
        glVertex2f(pixels[i].x, pixels[i].y);    //表示一个空间顶点
    glEnd();
}
```

1.3.2 结果呈现

实现效果如下所示:



2. 第二次上机作业



第二次上机作业

一、图形变换（借鉴 lec4 课件）

1. 利用 OpenGL 实现一个立方体关于参考点（10.0,20.0,10.0）进行放缩变换，放缩因子为（2.0,1.0,0.5）。

2. 利用 OpenGL 实现一个矩形关于 y=x+5 对称的新图形。

3. 通过定义键盘回调函数，每按一次空格键，让三个点依次完成画点、画线、画三角形，并让三角形沿三角形中心旋转起来。

2.1 立方体放缩变换

2.1.1 实现思路与函数使用

初始化背景，视图等状态以后，在渲染场景RenderScene函数中：① 利用glTranslatef(-10.0, -20.0, -10.0)将参考点平移,使参考点与原点重合，② 再利用glScalef(2.0, 1.0, 0.5); 将立方体缩放，③ 最后利用glTranslatef(10.0, 20.0, 10.0); 使参考点回到原来的位置，从而实现了立方体关于参考点的放缩变换。以上三个步骤都存储在变换矩阵中，因而最后利用glutWireCube(10.0)画出的立方体即完成放缩变换的立方体。

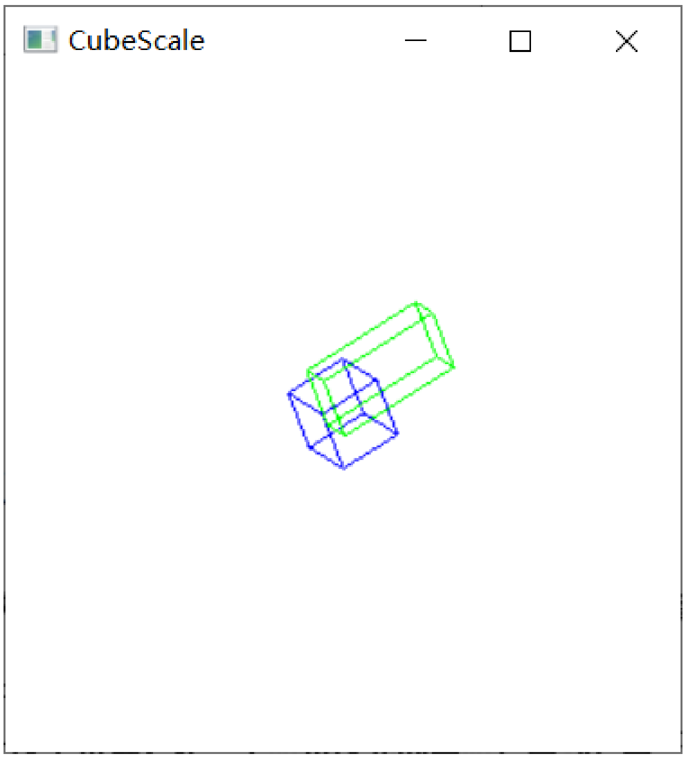
同时，为了加强结果呈现的清晰度与对比度，我们设置了一个未变化的立方体并把两个立方体都旋转了45°。

主要功能函数如下：

```
void RenderScene()
{
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1.0, 0.0, 0.0);

    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();

    glColor3f(0.0, 0.0, 1.0);
    glPushMatrix();
    glRotatef(45.0, 1.0, 1.0, 1.0);
    glutWireCube(10.0);    // 初始对比
    glPopMatrix();



    glColor3f(0.0, 1.0, 0.0);
    glRotatef(45.0, 1.0, 1.0, 1.0);    //旋转，为使显示更为清楚
    glPushMatrix();
    glTranslatef(-10.0, -20.0, -10.0); //平移,使参考点与原点重合
    glPushMatrix();
    glScalef(2.0, 1.0, 0.5);  //缩放
    glPushMatrix();
    glTranslatef(10.0, 20.0, 10.0); //使参考点回到原来的位置
    glPushMatrix();
    glutWireCube(10.0);

    glFlush();
}
```

2.1.2 结果呈现

实现效果如下所示：



2.2 矩形对称

2.2.1 实现思路与函数使用

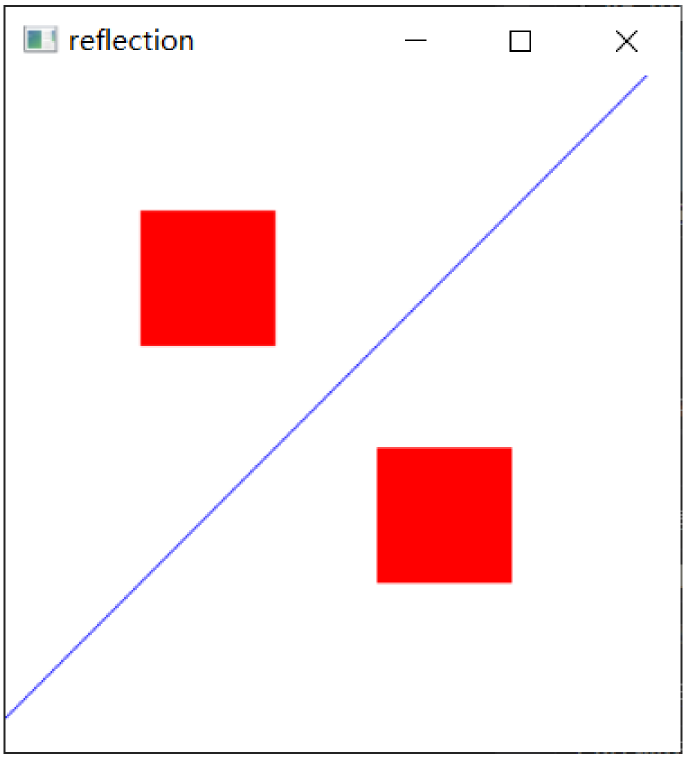
初始化背景，视图等状态以后，在渲染场景RenderScene函数中：① 首先画出了初始的矩形(-10.0, 10.0 ,-30.0, 30.0)与对称线 y = x + 5 ，② 然后我们通过对变换矩阵进行三部操作，glTranslatef(0, 5, 0);，glRotated(45, 0, 0, 1);，glScaled(1, -1, 1);即可得到对应图形关于对称线 y = x + 5的对称图形 ③ 最后，再次利用glRectf(-10.0, 10.0, -30.0, 30.0)画出初始的矩形，得到的乘以变换矩阵的对称图形。

主要功能函数如下：

```
void RenderScene()
{
    glClear(GL_COLOR_BUFFER_BIT);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    // 初始矩形
    glColor3f(1.0, 0.0, 0.0);
    glRectf(-10.0, 10.0 ,-30.0,  30.0);
    // 对称线 y = x + 5
    glBegin(GL_LINE_STRIP);
        glColor3f(0.0, 0.0, 1.0);
        glVertex2f(-50, -45);
        glVertex2f(45,50);
    glEnd();

    glTranslatef(0, 5, 0);
    glPushMatrix();
    glRotated(45, 0, 0, 1);
    glPushMatrix();
    glScaled(1, -1, 1);
    glPushMatrix();
    glRotated(-45, 0, 0, 1);
    glPushMatrix();
    glTranslated(0, -5, 0);
    glPushMatrix();

    glColor3f(1.0, 0.0, 0.0);
    glRectf(-10.0, 10.0, -30.0, 30.0);

    glFlush();
}
```

2.2.2 结果呈现

实现效果如下所示:



2.3 综合

2.3.1 实现思路与函数使用

首先设置三个全局变量，currentMode，angle，ModeNums，分别表示当前运行模式，当前旋转角度，模式总数（空白，画点，画线，画三角，平移，缩放，旋转 共7类）。

利用glutKeyboardFunc(myKey); 为当前窗口设置键盘回调函数，其中自定义的mykey函数通过空格键切换模式，切换公式为currentMode = (currentMode+1)%ModeNums 以此实现逐一变换。此外 ESC 设置为退出键。

在场景渲染函数RenderScene()中创建switch分支选择，currentMode的值作为分指选择的key。

在实现三角形旋转中，我注册了一个定时器，glutTimerFunc(33, timerFunction, 1); 功能是每运行一次计时器使得angle = (angle+3)%360，并且结合glRotatef(angle, 0.0, 0.0, 1.0)函数从而实现旋转。

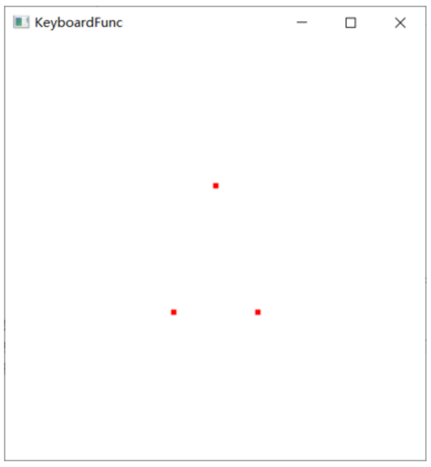
部分函数如下：

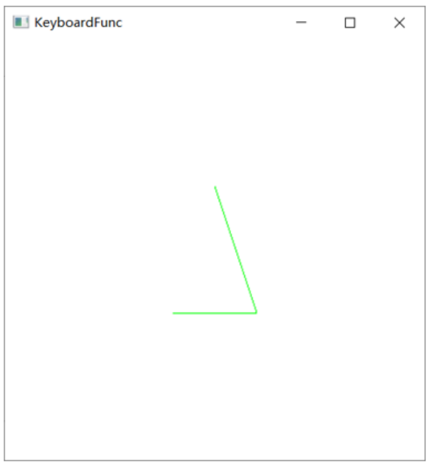
```
int currentMode = 0;
int angle = 0;
const int ModeNums = 7;

void myKey( unsigned char key, int x, int y)
{
    switch(key)
    {
        case ' ': currentMode = (currentMode+1)%ModeNums;
```

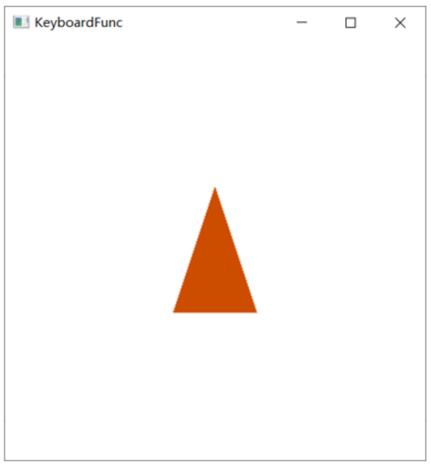
```
                glutPostRedisplay();
                break;
        case 27:  exit(-1);
    }
}

void timerFunction(int id)
{
    angle  = (angle+3)%360;

    glutPostRedisplay();
    glutTimerFunc(33, timerFunction, 1);
}

void RenderScene()
{
    glClear(GL_COLOR_BUFFER_BIT);
    switch(currentMode)
    {
        case 0: break;

        case 1: glPointSize(5);
                glBegin(GL_POINTS);
                glColor3f(1.0,0.0,0.0);
                glVertex2f(-2.0, -3.0);
                glVertex2f(2.0, -3.0);
                glVertex2f(0.0, 3.0);
                glEnd();

                break;
        case 2: glBegin(GL_LINE_STRIP);
                glColor3f(0.0,1.0,0.0);
                glVertex2f(-2.0, -3.0);
                glVertex2f(2.0, -3.0);
                glVertex2f(0.0, 3.0);
                glEnd();
                break;

        case 3: glBegin(GL_TRIANGLES);
                glColor3f(0.8, 0.3, 0.0);
                glVertex2f(-2.0, -3.0);
                glVertex2f(2.0, -3.0);
                glVertex2f(0.0, 3.0);
                glEnd();
                break;

        case 4: glPushMatrix();
            glTranslatef(0.5, 1.0, 0.5);
            glBegin(GL_TRIANGLES);
            glColor3f(0.8, 0.3, 0.0);
            glVertex2f(-2.0, -3.0);
            glVertex2f(2.0, -3.0);
            glVertex2f(0.0, 3.0);
            glEnd();
            glPopMatrix();
                break;

        case 5:glPushMatrix();
            glScalef(0.5, 0.5, 0.5);
            glBegin(GL_TRIANGLES);
            glColor3f(0.8, 0.3, 0.0);
            glVertex2f(-2.0, -3.0);
            glVertex2f(2.0, -3.0);
            glVertex2f(0.0, 3.0);
            glEnd();
            glPopMatrix();
            break;

        case 6:glPushMatrix();
            glRotatef(angle, 0.0, 0.0, 1.0);
            glBegin(GL_TRIANGLES);
            glColor3f(0.8, 0.3, 0.0);
            glVertex2f(-2.0, -3.0);
            glVertex2f(2.0, -3.0);
            glVertex2f(0.0, 3.0);
            glEnd();
            glPopMatrix();
            break;
    }
    glutSwapBuffers();
```

2.3.2 结果呈现

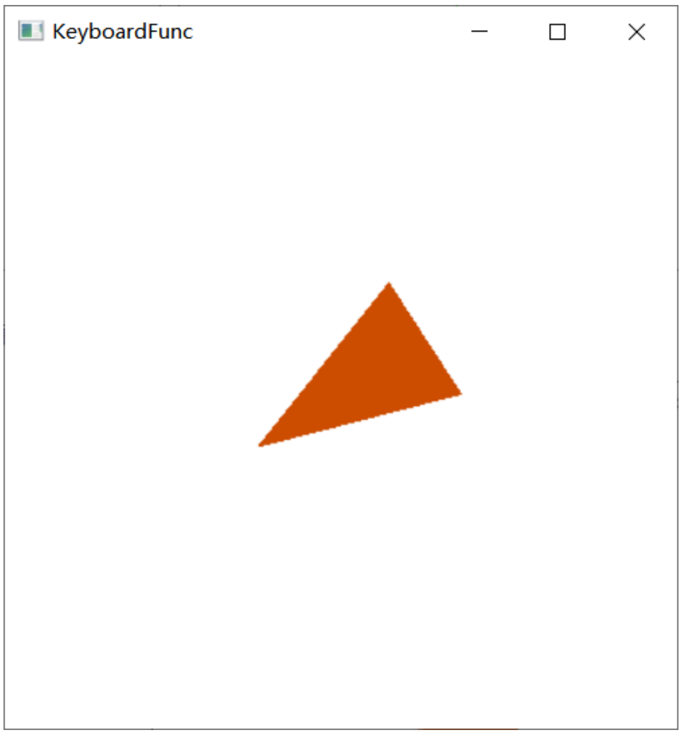
实现效果如下所示：



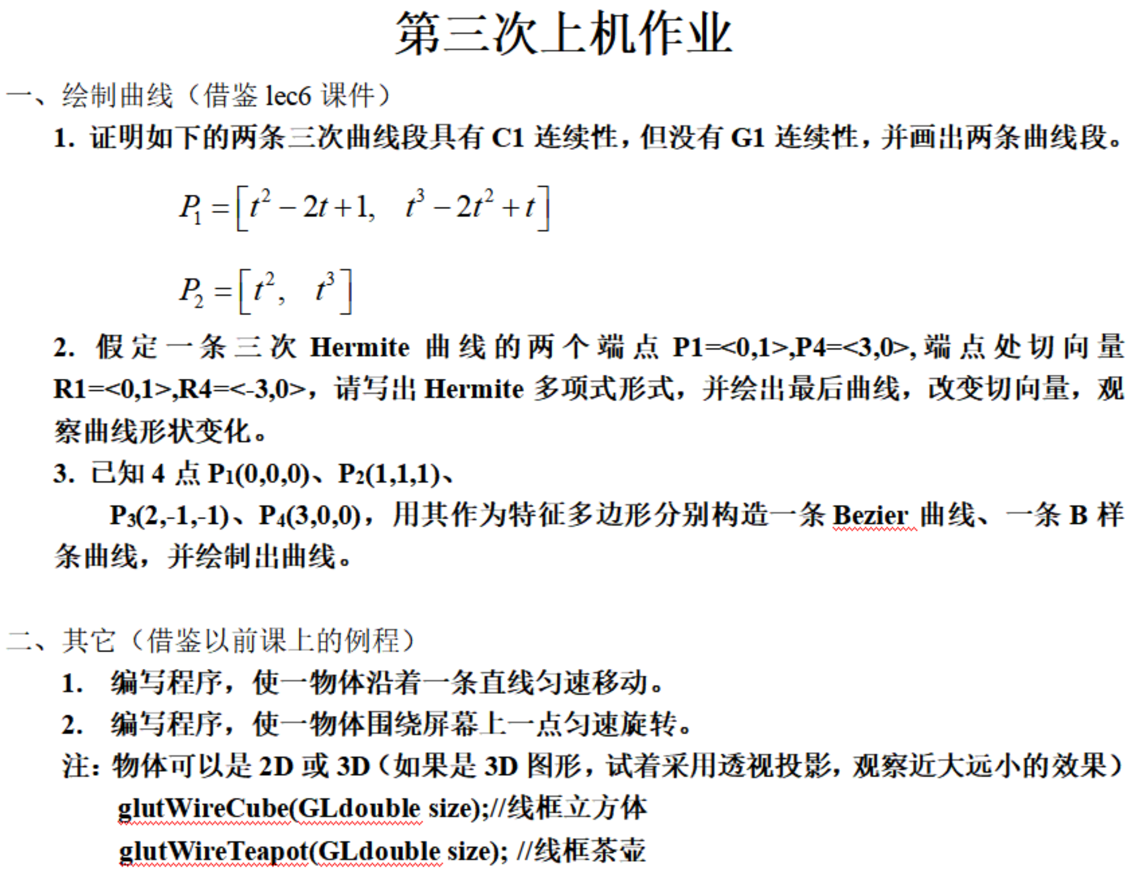




三角形旋转效果详见 三角形旋转.gif：



3. 第三次上机作业



第三次上机作业

一、绘制曲线（借鉴 lec6 课件）

1. 证明如下的两条三次曲线段具有 C1 连续性，但没有 G1 连续性，并画出两条曲线段。

$$P_1 = \left[t^2 - 2t + 1,\quad t^3 - 2t^2 + t\right]$$

$$P_2 = \left[t^2,\quad t^3\right]$$

2. 假定一条三次 Hermite 曲线的两个端点 P1=<0,1>,P4=<3,0>, 端点处切向量 R1=<0,1>,R4=<-3,0>，请写出 Hermite 多项式形式，并绘出最后曲线，改变切向量，观察曲线形状变化。

3. 已知 4 点 P_1(0,0,0)、P_2(1,1,1)、P_3(2,-1,-1)、P_4(3,0,0)，用其作为特征多边形分别构造一条 Bezier 曲线、一条 B 样条曲线，并绘制出曲线。

二、其它（借鉴以前课上的例程）

1. 编写程序，使一物体沿着一条直线匀速移动。

2. 编写程序，使一物体围绕屏幕上一点匀速旋转。

注：物体可以是 2D 或 3D（如果是 3D 图形，试着采用透视投影，观察近大远小的效果）

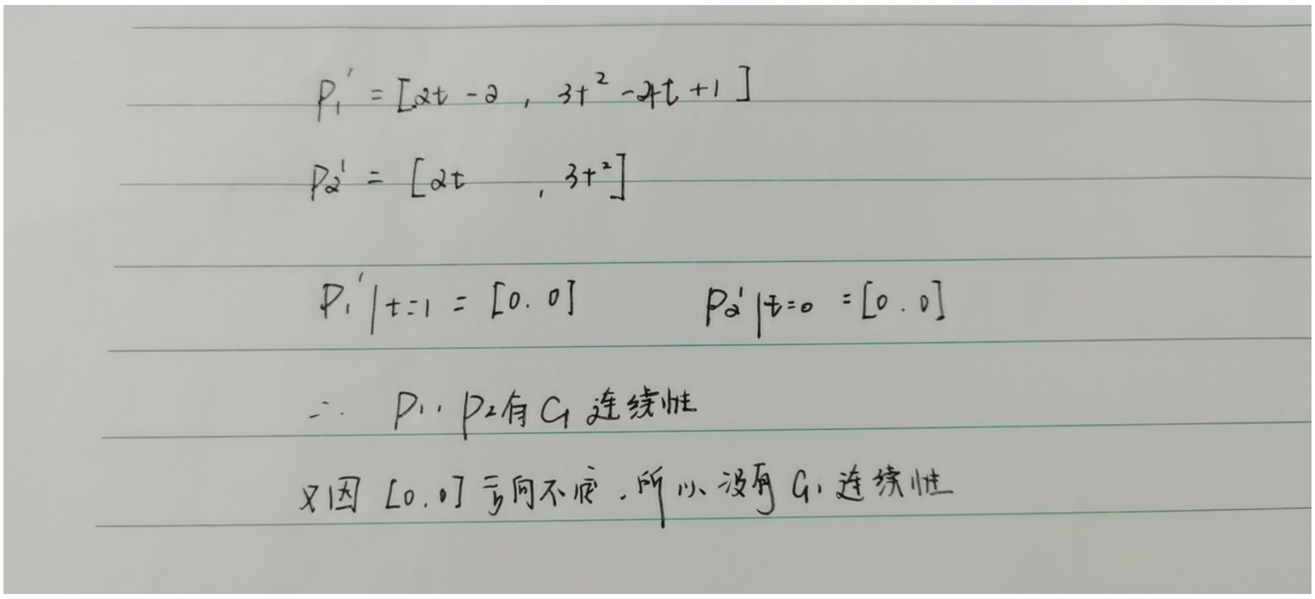
glutWireCube(GLdouble size);//线框立方体

glutWireTeapot(GLdouble size); //线框茶壶

3.1 绘制曲线 1

3.1.1 实现思路与函数使用

证明如图所示：



$P_1' = [2t-2, 3t^2-4t+1]$

$P_2' = [2t, 3t^2]$

$P_1'|_{t=1} = [0.0]$ $P_2'|_{t=0} = [0.0]$

$\therefore$ P_1, P_2有 C_1 连续性

又因 [0.0] 方向不定，所以没有 G_1 连续性

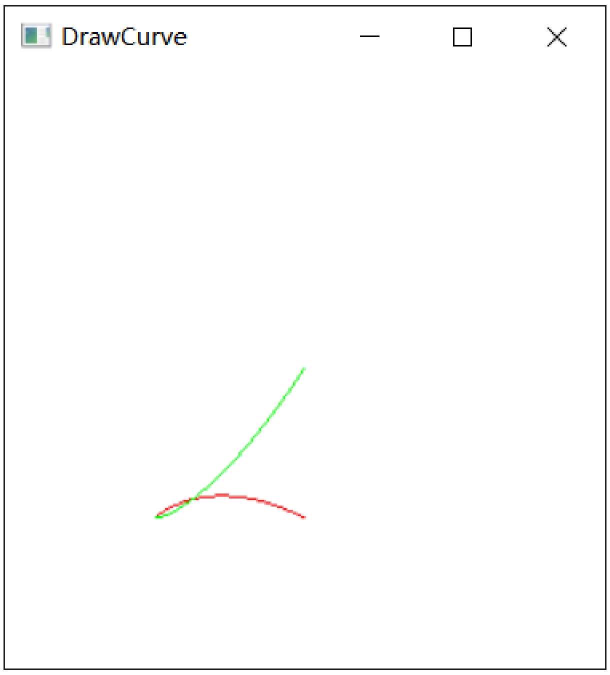
程序的实现即利用在渲染场景RenderScene函数中调用glBegin(*GLenum mode*)与glEnd()一对函数，根据坐标公式即可对直线上的每一个点进行绘制。

主要功能函数如下：

```
void drawCurve1(int n)
{
    point pixels[100];
    float delta,t,t2,t3;
    int i;

    delta = 1.0/(n-1);
    glBegin(GL_LINE_STRIP);
        for(i=0;i<n;i++)
        {
            t = i*delta;
            t2 = t*t;
            t3 = t2*t;
            pixels[i].x = t2-2*t+1;
            pixels[i].y = t3-2*t2+t;
            glVertex2f(pixels[i].x,pixels[i].y);
        }
    glEnd();
}

void drawCurve2(int n)
{
    point pixels[100];
    float delta,t,t2,t3;
    int i;

    delta = 1.0/(n-1);
    glBegin(GL_LINE_STRIP);
        for(i=0;i<n;i++)
        {
            t = i*delta;
            t2 = t*t;
            t3 = t2*t;
            pixels[i].x = t2;
            pixels[i].y = t3;
            glVertex2f(pixels[i].x,pixels[i].y);
        }
    glEnd();
}
```

3.1.2 结果呈现

实现效果如下所示:



3.2 绘制曲线 2

3.2.1 实现思路与函数使用

Hermite多项式形式为：

$$Q(t) = B_H * G_H$$
$$= (2t^3 - 3t^2 + 1)P_1 + (-2t^3 + 3t^2)P_4 + (t^3 - 2t^2 + t)R_1 + (t^3 - t^2)R_4$$

程序的实现即利用在渲染场景RenderScene函数中调用glBegin(*GLenum mode*)与glEnd()一对函数，根据相应曲线的坐标公式获得坐标即可对直线上的每一个点进行绘制，并且做相应参数的位置矢量进行计算即可。

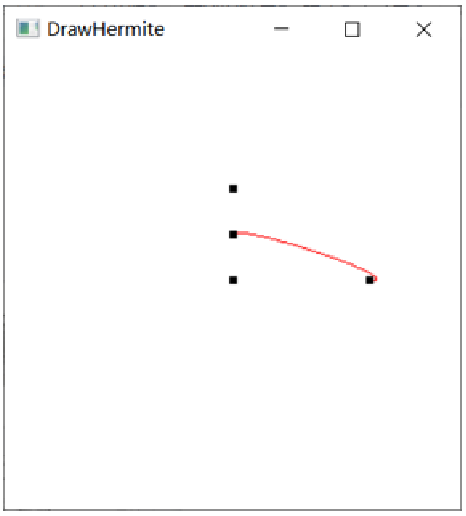
改变切向量为R1<0,1>,R4<-5,0>进行对比观察。

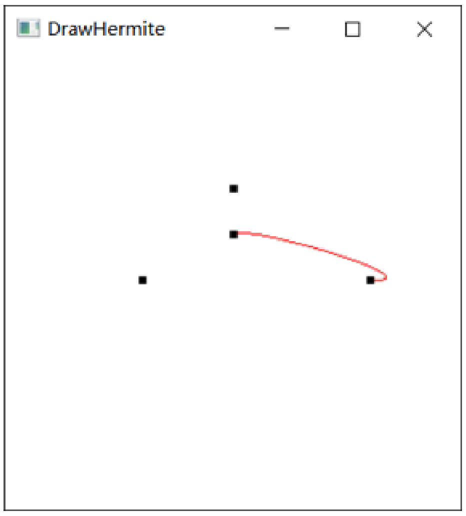
主要功能函数如下：

```
Point ctrlpoints[4] = { {0.0,1.0,0.0},{3.0,0.0,0.0},{0.0,1.0,0.0},
{-3.0,0.0,0.0} };

void DrawCurve(int n)
{
    Point p;
    double t, deltat, t2, t3;
    int i;
    deltat = 1.0 / (n - 1);
    glBegin(GL_LINE_STRIP);
    for (i = 0; i < 100; i++)
    {
        t = i * deltat;
        t2 = t * t;
        t3 = t * t2;

        p.x = (2 * t3 - 3 * t2 + 1) * ctrlpoints[0].x + (-2 * t3 + 3 *
t2) * ctrlpoints[1].x + (t3 - 2 * t2 + t) * ctrlpoints[2].x + (t3 - t2)
* ctrlpoints[3].x;
        p.y = (2 * t3 - 3 * t2 + 1) * ctrlpoints[0].y + (-2 * t3 + 3 *
t2) * ctrlpoints[1].y + (t3 - 2 * t2 + t) * ctrlpoints[2].y + (t3 - t2)
* ctrlpoints[3].y;
        p.z = (2 * t3 - 3 * t2 + 1) * ctrlpoints[0].z + (-2 * t3 + 3 *
t2) * ctrlpoints[1].z + (t3 - 2 * t2 + t) * ctrlpoints[2].z + (t3 - t2)
* ctrlpoints[3].z;
        glVertex3f(p.x, p.y, p.z);
    }
    glEnd();

    glPointSize(5.0);
    glColor3f(0.0, 0.0, 0.0);
    glBegin(GL_POINTS);
    for (i = 0; i < 2; i++) {
        glVertex3f(ctrlpoints[i].x, ctrlpoints[i].y, ctrlpoints[i].z);
        glVertex3f(ctrlpoints[i].x + ctrlpoints[i + 2].x,
ctrlpoints[i].y + ctrlpoints[i + 2].y, ctrlpoints[i].z + ctrlpoints[i +
2].z);
    }
    glEnd();

}
```

3.2.2 结果呈现

对比效果如下所示:





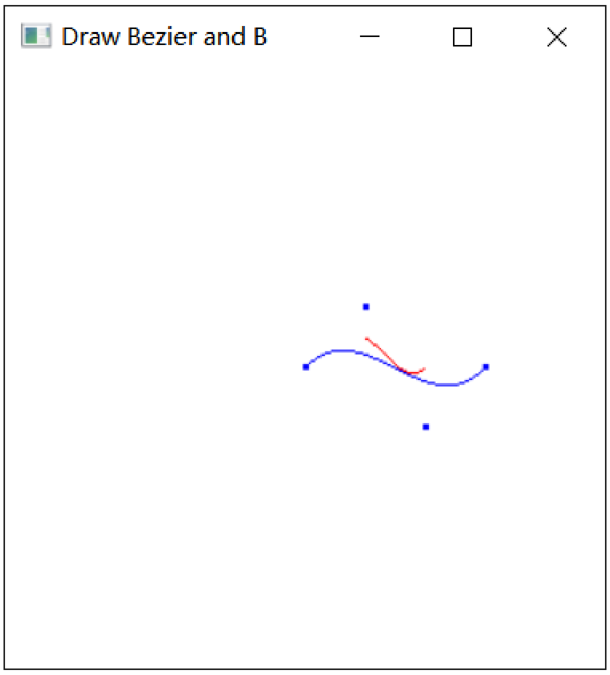
3.3 绘制曲线 3

3.3.1 实现思路与函数使用

此题与作业4中的第一题基本重复，实现思路详见作业4。

3.3.2 结果呈现

实现效果如下所示:



3.4 其它 1

3.4.1 实现思路与函数使用

利用glutPostRedisplay()函数进行场景重绘，并且每次重绘时都使得转移坐标z = z + speed，然后glTranslated(0.0, 0.0, -z)并且利用glutWireCube(10)画出立方体，即可实现立方体沿着Z轴进行移动。

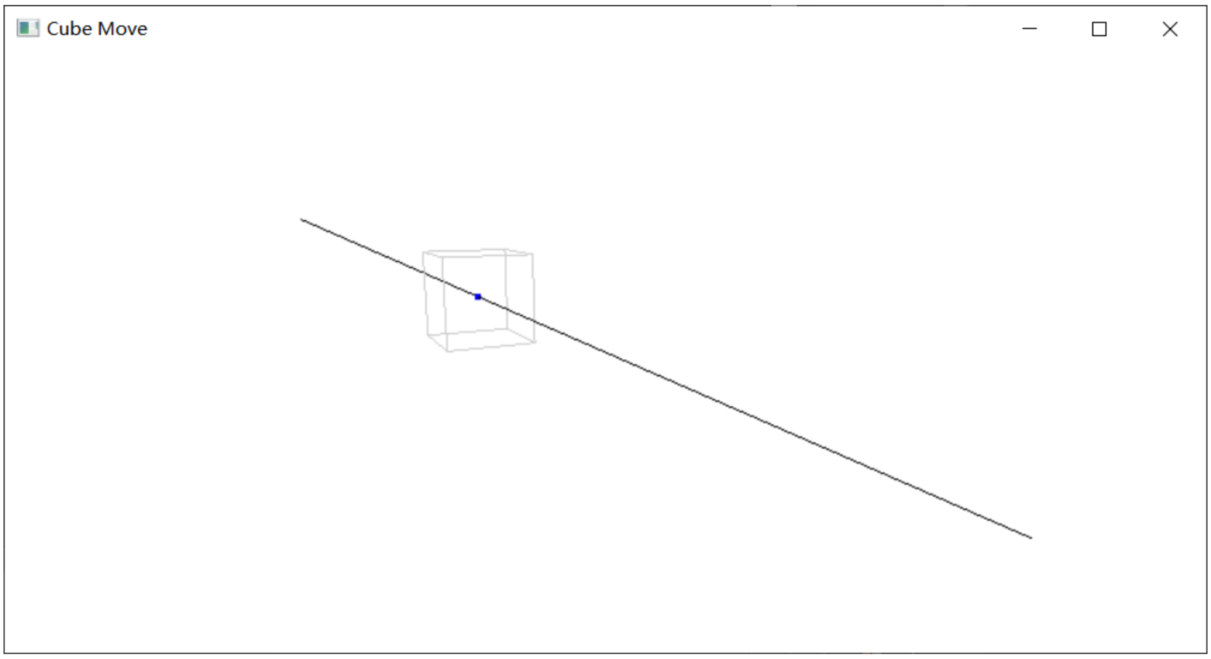
注册键盘事件，其中r键为启动运行，s键为停止，u键为设置正向速度，d键为设置反向速度。

主要功能函数如下：

```
void RenderScene()
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    //移动图形
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();

    if (z > depth)
        z = 20;
    else
        z = z + speed;

    glTranslated(0.0, 0.0, -z);
    glColor3f(0.8, 0.8, 0.8);
    glutWireCube(10);
    glPointSize(4);
    glBegin(GL_POINTS);
    glColor3f(0, 0, 1);
    glVertex3d(0, 0, 0);
    glEnd();

        //画两条平行线当做路
    glBegin(GL_LINES);
    glColor3f(0, 0, 0);
    glVertex3d(0, 0, -depth);
    glVertex3d(0, 0, depth);
    glEnd();

    //强制执行绘图命令，交换缓冲区进行显示
    glFlush();
    glutSwapBuffers();
    glLoadIdentity();

    if (RunMode == 1) {
        glutPostRedisplay();    // 触发窗口重绘
    }
}
```

3.4.2 结果呈现

实现效果详见 立方体沿直线旋转.gif:



3.5 其它 2

3.5.1 实现思路与函数使用

利用glutPostRedisplay()函数进行场景重绘，并且每次重绘时都使得旋转角度angle = angle + speed，然后进行旋转glRotated(angle, 0, 1, 0)并且利用glutSolidTeapot(60)画出茶壶，即可实现茶壶沿着原点进行旋转。

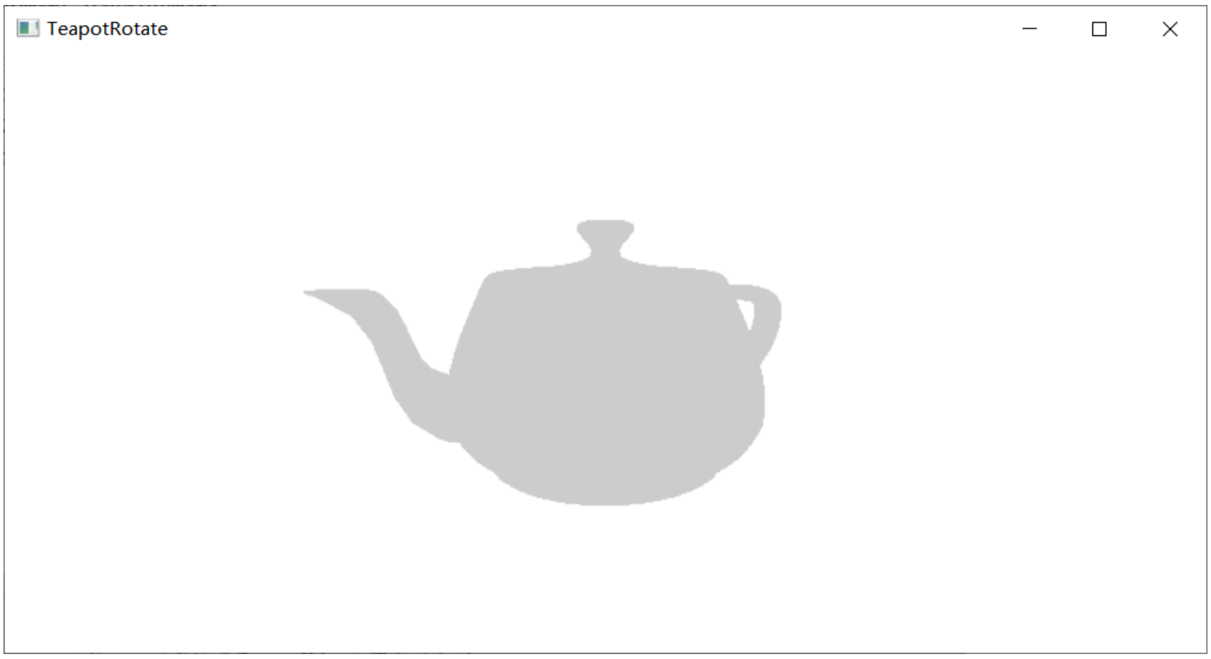
注册键盘事件，其中r键为启动运行，s键为停止，u键为设置正向速度，d键为设置反向速度。

主要功能函数如下：

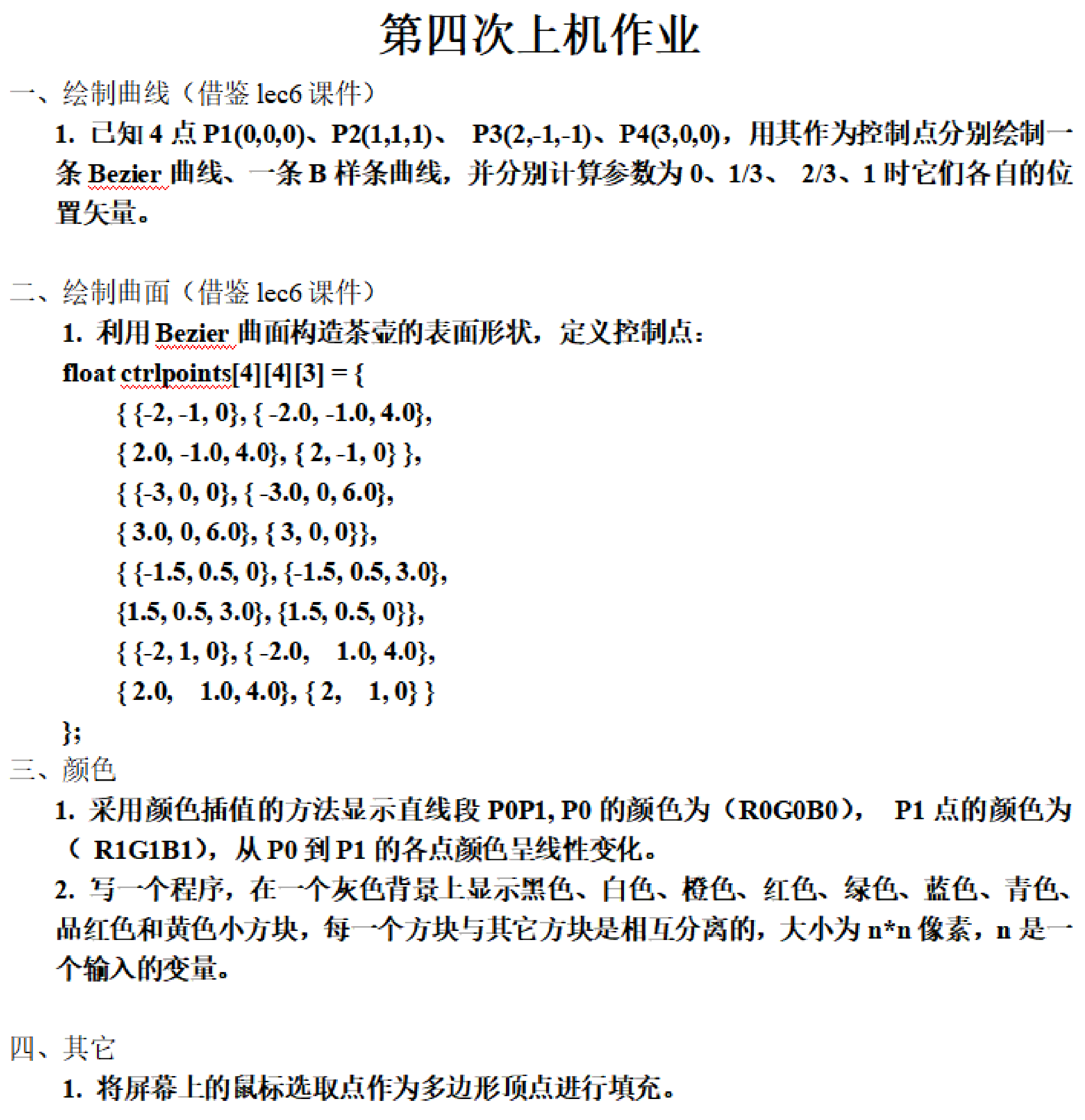
```
void RenderScene()
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    //移动图形
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();

    if (angle > 360)
        angle = 0;
    else
        angle = angle + speed;

    //glTranslated(0.0, -40.0, -200);//把整个场景移到到视图中
    glRotated(angle, 0, 1, 0);

    glColor3f(0.8, 0.8, 0.8);
    glutSolidTeapot(60);

    //强制执行绘图命令，交换缓冲区进行显示
    glFlush();
    glutSwapBuffers();
    glLoadIdentity();

    if (RunMode == 1) {
        glutPostRedisplay();    // 触发窗口重绘
    }
}
```

3.5.2 结果呈现

实现效果详见 茶壶旋转.gif：



4. 第四次上机作业



第四次上机作业

一、绘制曲线（借鉴 lec6 课件）

1. 已知 4 点 P1(0,0,0)、P2(1,1,1)、 P3(2,-1,-1)、P4(3,0,0)，用其作为控制点分别绘制一条 Bezier 曲线、一条 B 样条曲线，并分别计算参数为 0、1/3、 2/3、1 时它们各自的位置矢量。

二、绘制曲面（借鉴 lec6 课件）

1. 利用 Bezier 曲面构造茶壶的表面形状，定义控制点：

```
float ctrlpoints[4][4][3] = {
        { {-2, -1, 0}, { -2.0, -1.0, 4.0},
        { 2.0, -1.0, 4.0}, { 2, -1, 0} },
        { {-3, 0, 0}, { -3.0, 0, 6.0},
        { 3.0, 0, 6.0}, { 3, 0, 0}},
        { {-1.5, 0.5, 0}, {-1.5, 0.5, 3.0},
        {1.5, 0.5, 3.0}, {1.5, 0.5, 0}},
        { {-2, 1, 0}, { -2.0,   1.0, 4.0},
        { 2.0,   1.0, 4.0}, { 2,   1, 0} }
};
```

三、颜色

1. 采用颜色插值的方法显示直线段 P0P1, P0 的颜色为（R0G0B0）， P1 点的颜色为（ R1G1B1），从 P0 到 P1 的各点颜色呈线性变化。

2. 写一个程序，在一个灰色背景上显示黑色、白色、橙色、红色、绿色、蓝色、青色、品红色和黄色小方块，每一个方块与其它方块是相互分离的，大小为 n*n 像素，n 是一个输入的变量。

四、其它

1. 将屏幕上的鼠标选取点作为多边形顶点进行填充。

4.1 绘制曲线

4.1.1 实现思路与函数使用

程序的实现即利用在渲染场景RenderScene函数中调用glBegin(*GLenum mode*)与glEnd()一对函数，根据相应曲线的坐标公式获得坐标即可对直线上的每一个点进行绘制，并且做相应参数的位置矢量进行计算即可。

相应曲线的多项式如下:

Bezier曲线多项式:

$$Q(t)=T\bullet M_B\bullet G_B=(1-t)^3P_1+3t(1-t)^2P_2+3t^2(1-t)P_3+t^3P_4$$

B样条曲线多项式:

$$Q_i(t)=G_{Bsi}*B_{Bs}=B_{Bs-3}*P_{i-3}+B_{Bs-2}*P_{i-2}+B_{Bs-1}*P_{i-1}+B_{Bs}*P_i$$
$$=\frac{1}{6}(1-t)^3P_{i-3}+\frac{3t^3-6t^2+4}{6}P_{i-2}+\frac{-3t^3+3t^2+3t+1}{6}P_{i-1}$$
$$+\frac{t^3}{6}P_i\quad 0\le t<1$$

主要功能函数如下:

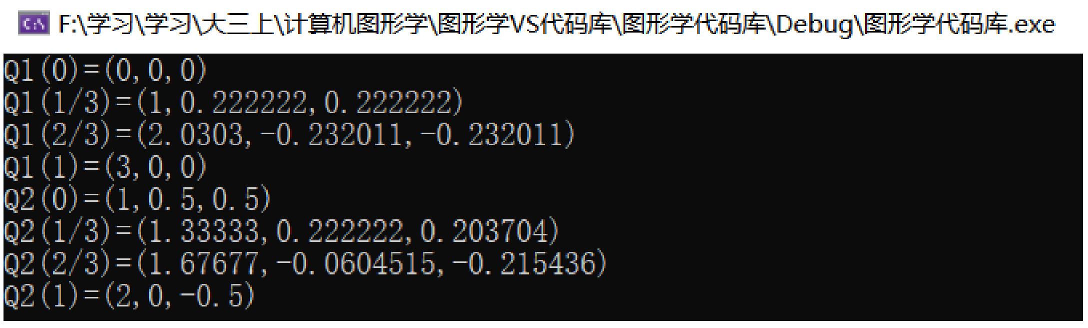
```
void DrawBezier3D(int n, Point P[])
{
    Point p;
    double t, deltat, t2, t3, et, et2, et3;
    int i;
    deltat = 1.0 / (n - 1);

    glBegin(GL_LINE_STRIP);
    for (i = 0; i < 100; i++)
    {
        t = i * deltat;
        et = 1 - t;
        t2 = t * t;
        et2 = et * et;
        t3 = t * t2;
        et3 = et * et2;
        p.x = et3 * P[0].x + 3 * t * et2 * P[1].x + 3 * t2 * et * P[2].x
+ t3 * P[3].x;
        p.y = et3 * P[0].y + 3 * t * et2 * P[1].y + 3 * t2 * et * P[2].y
+ t3 * P[3].y;
```

```
        p.z = et3 * P[0].z + 3 * t * et2 * P[1].z + 3 * t2 * et * P[2].z
+ t3 * P[3].z;

        if (i == 0)
            cout << "Q1(0)=(" << p.x << "," << p.y << "," << p.z << ")"
<< endl;
        else if (i == 33)
            cout << "Q1(1/3)=(" << p.x << "," << p.y << "," << p.z <<
")" << endl;
        else if (i == 67)
            cout << "Q1(2/3)=(" << p.x << "," << p.y << "," << p.z <<
")" << endl;
        else if (i == 99)
            cout << "Q1(1)=(" << p.x << "," << p.y << "," << p.z << ")"
<< endl;

        glVertex3f(p.x, p.y, p.z);
    }
    glEnd();
    //glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1, 0, 0);

    glPointSize(3);
    glBegin(GL_POINTS);
    glColor3f(1, 0, 0);
    for (i = 0; i < 4; i++)
    {
        glVertex2d(P[i].x, P[i].y);
    }
    glEnd();
}

void DrawB3D(int n, Point P[])
{
    Point p;
    double t, deltat, t2, t3, et, et2, et3;
    int i;
    deltat = 1.0 / (n - 1);
    glBegin(GL_LINE_STRIP);
    for (i = 0; i < 100; i++)
    {
        t = i * deltat;
        et = 1 - t;
        t2 = t * t;
        et2 = et * et;
        t3 = t * t2;
        et3 = et * et2;
        p.x = et3 * P[0].x / 6 + (3 * t3 - 6 * t2 + 4) * P[1].x / 6 +
(-3 * t3 + 3 * t2 + 3 * t + 1) * P[2].x / 6 + t3 * P[3].x / 6;
```

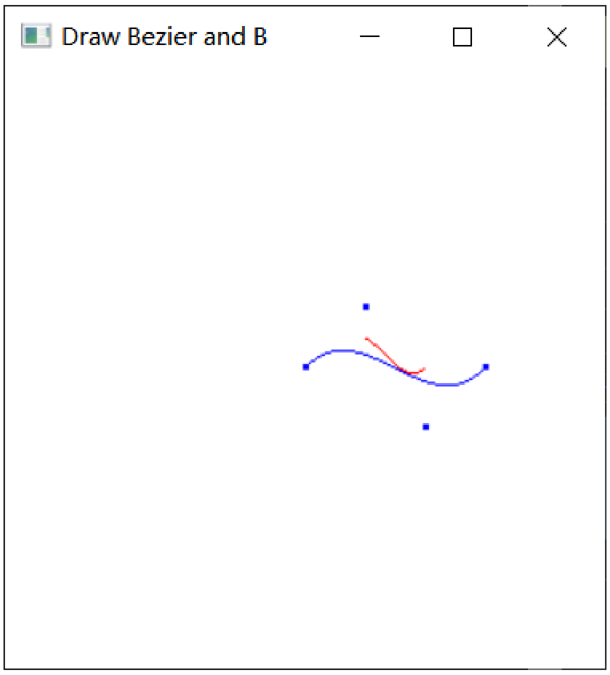
```
        p.y = et3 * P[0].y / 6 + (3 * t3 - 6 * t2 + 4) * P[1].y / 6 + 
(-3 * t3 + 3 * t2 + 3 * t + 1) * P[2].y / 6 + t3 * P[3].x / 6;
        p.z = et3 * P[0].z / 6 + (3 * t3 - 6 * t2 + 4) * P[1].z / 6 + 
(-3 * t3 + 3 * t2 + 3 * t + 1) * P[2].z / 6 + t3 * P[3].z / 6;
        if (i == 0)
            cout << "Q2(0)=(" << p.x << "," << p.y << "," << p.z << ")" 
<< endl;
        else if (i == 33)
            cout << "Q2(1/3)=(" << p.x << "," << p.y << "," << p.z << 
")" << endl;
        else if (i == 67)
            cout << "Q2(2/3)=(" << p.x << "," << p.y << "," << p.z << 
")" << endl;
        else if (i == 99)
            cout << "Q2(1)=(" << p.x << "," << p.y << "," << p.z << ")" 
<< endl;
        glVertex3f(p.x, p.y, p.z);
    }
    cout << "" << endl;
    glEnd();
    //glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(0, 0, 1);

    glPointSize(3);
    glBegin(GL_POINTS);
    glColor3f(0, 0, 1);
    for (i = 0; i < 4; i++)
    {
        glVertex2d(P[i].x, P[i].y);
    }

    glEnd();
}
```

4.1.2 结果呈现

实现效果如下所示：


F:\学习\学习\大三上\计算机图形学\图形学VS代码库\图形学代码库\Debug\图形学代码库.exe

```
Q1(0)=(0,0,0)
Q1(1/3)=(1,0.222222,0.222222)
Q1(2/3)=(2.0303,-0.232011,-0.232011)
Q1(1)=(3,0,0)
Q2(0)=(1,0.5,0.5)
Q2(1/3)=(1.33333,0.222222,0.203704)
Q2(2/3)=(1.67677,-0.0604515,-0.215436)
Q2(1)=(2,0,-0.5)
```



4.2 绘制曲面

4.2.1 实现思路与函数使用

首先在初始化时，利用glMap2f(GL_MAP2_VERTEX_3,0.0,1.0,3,4,0.0,1.0,12,4,&ctrlpoints[0][0][0])定义Beizer曲面并且利用glEnable(GL_MAP2_VERTEX_3)进行启用。

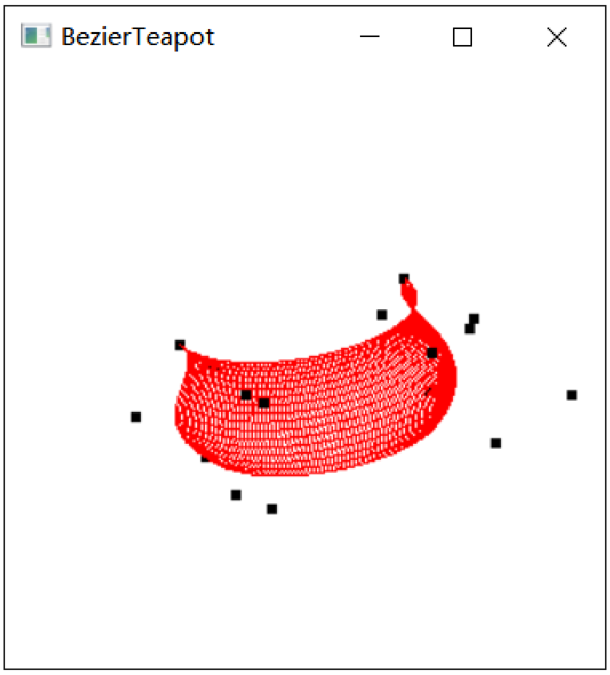
在渲染场景RenderScene函数中，使用高层函数glMapGrid2f(n,0.0,1.0,m,0.0,1.0)来定义网格建立一个2D映射网格，n和m指定了n和m方向网格划分的数量。在调用glEvalMesh2(GL_LINE,0,n,0,m)计算直线网格。最后将控制点画在场景中。

主要功能函数如下：

```
GLfloat ctrlpoints[4][4][3] = {
    { {-2, -1, 0}, { -2.0, -1.0, 4.0},
    { 2.0, -1.0, 4.0}, { 2, -1, 0} },
    { {-3, 0, 0}, { -3.0, 0, 6.0},
    { 3.0, 0, 6.0}, { 3, 0, 0}},
    { {-1.5, 0.5, 0}, {-1.5, 0.5, 3.0},
    {1.5, 0.5, 3.0}, {1.5, 0.5, 0}},
    { {-2, 1, 0}, { -2.0,  1.0, 4.0},
    { 2.0,  1.0, 4.0}, { 2,  1, 0} }
};

void init()
{
    glClearColor(1.0,1.0,1.0,1.0);
    glMap2f(GL_MAP2_VERTEX_3,0.0,1.0,3,4,0.0,1.0,12,4,&ctrlpoints[0][0]
[0]);
    glEnable(GL_MAP2_VERTEX_3);
    glEnable(GL_DEPTH_TEST);
}

void DrawCurvedSurface(int n,int m)
{
    int i,j;

    glMapGrid2f(n,0.0,1.0,m,0.0,1.0);
    glEvalMesh2(GL_LINE,0,n,0,m);
    glPointSize(5.0);
    glColor3f(0.0, 0.0, 0.0);
    glBegin(GL_POINTS);
    for (i = 0; i < 4; i++)
        for(j=0;j<4;j++)
            glVertex3f(ctrlpoints[i][j][0],ctrlpoints[i][j]
[1],ctrlpoints[i][j][2]);
    glEnd();
}
```

4.2.2 结果呈现

实现效果如下所示：



4.3 颜色 1

4.3.1 实现思路与函数使用

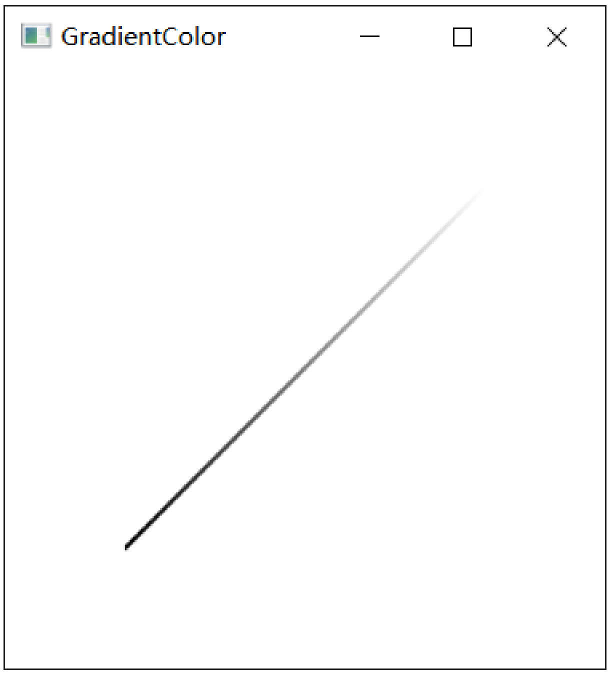
初始化状态以后，在渲染场景RenderScene函数中多次调用glBegin(*GLenum mode*)与glEnd()一对函数，通过对画线的每一个顶点后设置一个不同的颜色参数即可实现直线颜色的渐变。

主要功能函数如下：

```
void RenderScence()
{
    glClear(GL_COLOR_BUFFER_BIT);

    glLineWidth(3);
    glBegin(GL_LINES);
    glColor3f(0.0, 0.0, 0.0);
    glVertex2f(-3.0, -3.0);
    glColor3f(1.0, 1.0, 1.0);
    glVertex2f(3.0, 3.0);
    glEnd();

    glFlush();
}
```

4.3.2 结果呈现

实现效果如下所示：



4.4 颜色 2

4.4.1 实现思路与函数使用

声明一个Color数组存放题目所给的九种颜色，编写DrawSquare(int n,float x1,float x2)用于画出9种颜色的正方形（其中利用函数对 glBegin(GL_QUADS)；glEnd();)

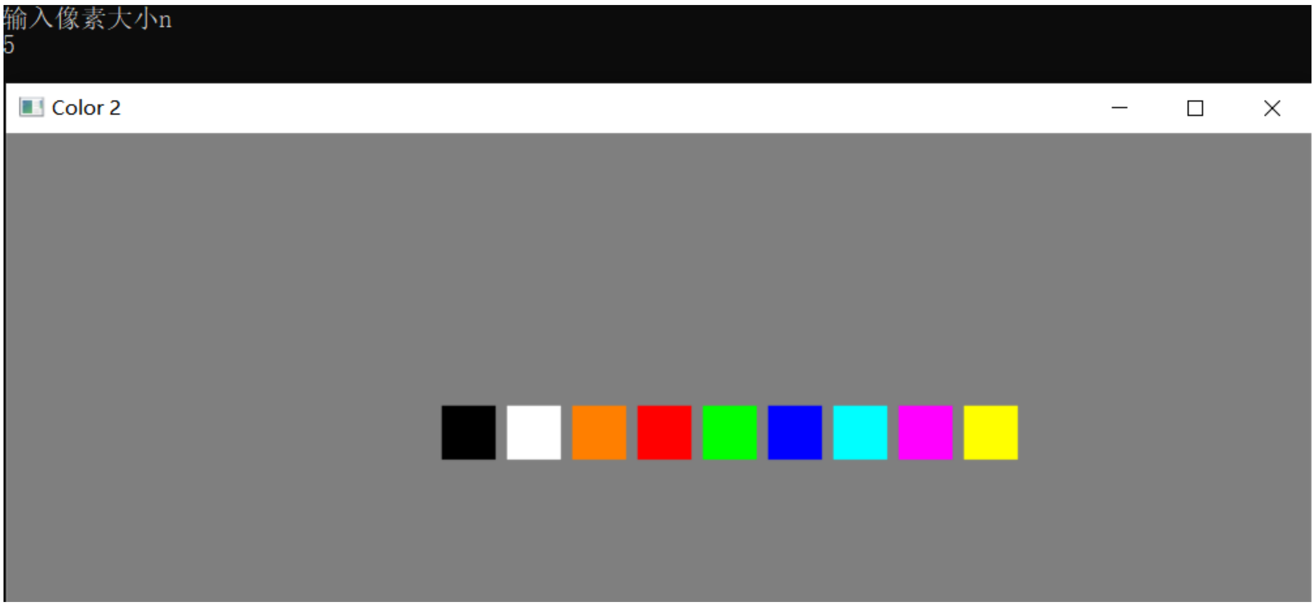
以上n为输入的像素大小，x1,x2为初始正方形的x轴坐标。

主要功能函数如下：

```
typedef struct {
    float r, g, b;
}point;

point Color[9] = { {0.0,0.0,0.0},{1.0,1.0,1.0},{1.0,0.5,0.0},
{1.0,0.0,0.0},{0.0,1.0,0.0},{0.0,0.0,1.0},{0.0,1.0,1.0},{1.0,0.0,1.0},
{1.0,1.0,0.0} };

void init()
{
    glClearColor(0.5, 0.5, 0.5, 1.0);
}

void DrawSquare(int n,float x1,float x2)
{
    for (int i = 0; i < 9; i++)
    {
        glColor3f(Color[i].r, Color[i].g, Color[i].b);
        glBegin(GL_QUADS);
        x1 += n ;
        x2 += n;
        glVertex2f(x1, 0.0);
        glVertex2f(x1, n);
        glVertex2f(x2, n);
        glVertex2f(x2, 0);
        glEnd();
        x1++;
        x2++;
    }
}

void RenderScence()
{
    glClear(GL_COLOR_BUFFER_BIT);
    DrawSquare(n,-5*n,-4*n);
    glFlush();
}
```

4.4.2 结果呈现

实现效果如下所示:



```
输入像素大小n
5
```

4.5 其它

4.5.1 实现思路与函数使用

首先声明一个全局变量数组PointArray用于存储鼠标事件所标记的点，并且用NumPts记录当前点的个数，mode用于控制当前页面显示（1，2，3），sum用于表示用户输入的多边形边数（3-9）。

注册鼠标事件，左键用于获取当前点的坐标，并利用addNewPoint将其存入PointArray，右键对应removeLastPoint即移除上一个点。

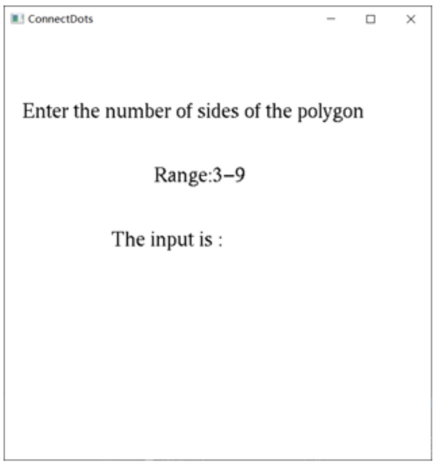
注册键盘事件，数字3-9表明进入绘图模式，s,f,r分别对应stop暂停功能，removeFirstPoint，removeAllPoint。

通过获取的点与模式即可画出对应多边形。

主要实现详见源代码：ConnectDots.cpp

4.5.2 结果呈现

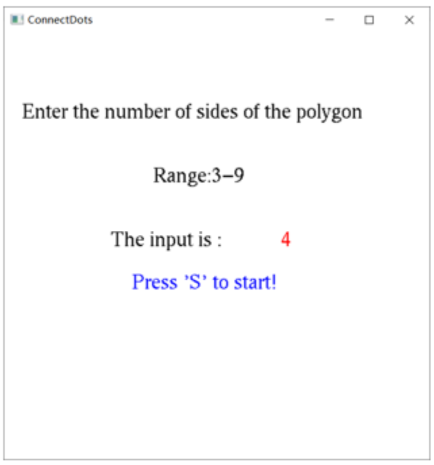
实现效果如下所示,详见 多边形.mp4:



Enter the number of sides of the polygon

Range:3–9

The input is :

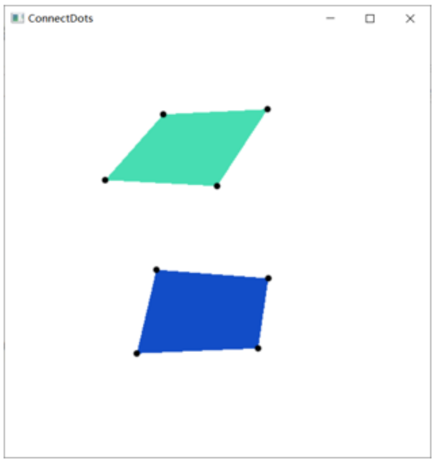


Enter the number of sides of the polygon

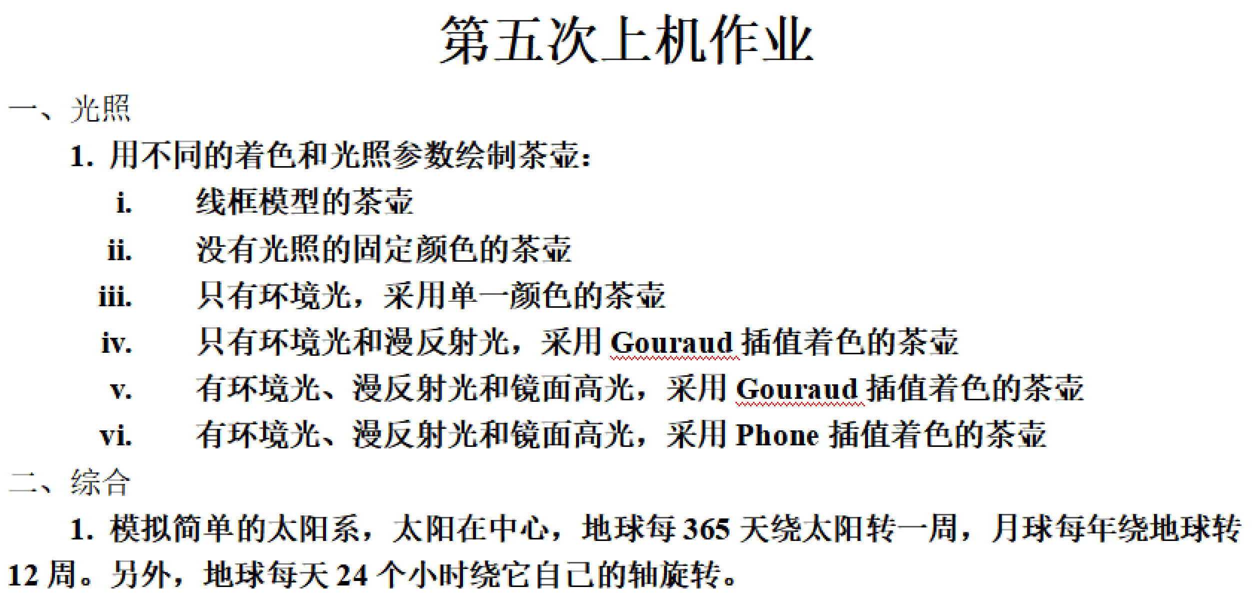
Range:3–9

The input is : 4

Press 'S' to start!



5. 第五次上机作业



第五次上机作业

一、光照

1. 用不同的着色和光照参数绘制茶壶：

- **i. 线框模型的茶壶**
- **ii. 没有光照的固定颜色的茶壶**
- **iii. 只有环境光，采用单一颜色的茶壶**
- **iv. 只有环境光和漫反射光，采用 Gouraud 插值着色的茶壶**
- **v. 有环境光、漫反射光和镜面高光，采用 Gouraud 插值着色的茶壶**
- **vi. 有环境光、漫反射光和镜面高光，采用 Phone 插值着色的茶壶**

二、综合

1. 模拟简单的太阳系，太阳在中心，地球每 365 天绕太阳转一周，月球每年绕地球转 12 周。另外，地球每天 24 个小时绕它自己的轴旋转。

5.1 光照

5.1.1 实现思路与函数使用

初始化的过程中，首先利用glLightfv()函数设置光源特性，并点亮光源。

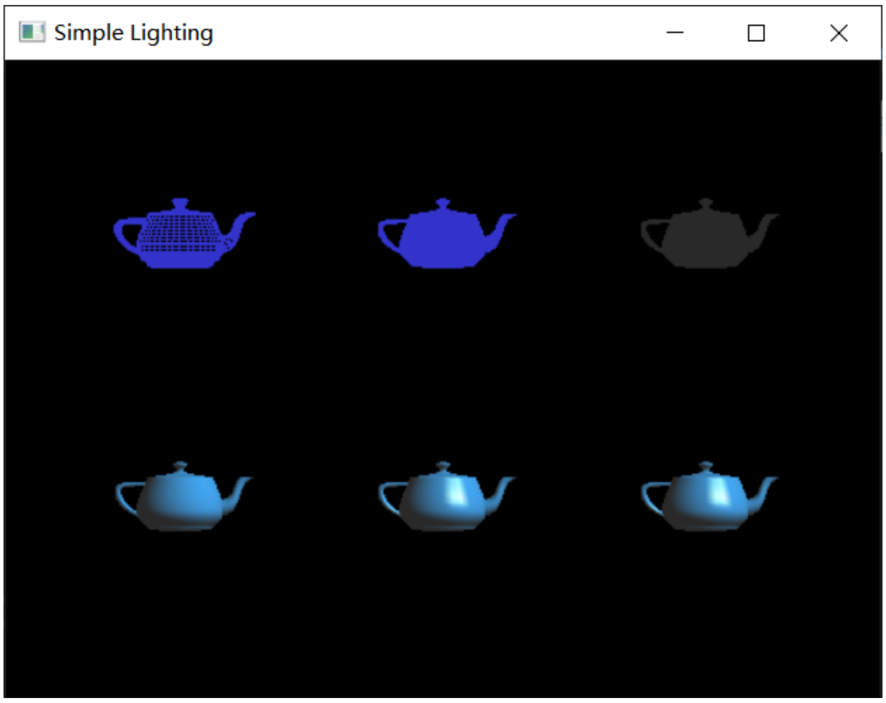
在渲染场景RenderScene函数中，首先设置材质的各种光的颜色成分反射率，然后根据题目要求的光源信息，利用glMaterialfv()函数来指定相应的物体表面反射系数，然后利用glutWire/SolidTeapot()和glTranslated()在不同位置画上拥有光照反射的茶壶。

主要功能函数如下：

```
void init()
{
    GLfloat light_position[]={1.0,1.0,1.0,0.0};
    glShadeModel(GL_SMOOTH);
    glLightfv(GL_LIGHT0,GL_POSITION,light_position);

    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT0);
    glEnable( GL_DEPTH_TEST );
}

void RenderScene()
{
        //设置材质的各种光的颜色成分反射率
    GLfloat no_mat[] ={0.0,0.0,0.0,1.0};
    GLfloat mat_ambient[] = {0.8,0.8,0.8,1.0};
    GLfloat mat_diffuse[] = {0.1,0.5,0.8,1.0};
    GLfloat mat_specular[] = {1.0,1.0,1.0,1.0};
    GLfloat no_shininess[] = {0.0};
    GLfloat low_shininess[] = {5.0};
    GLfloat high_shininess[] = {100.0};

    glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
    glTranslated(0.0,0.0,-20.0);

    glDisable(GL_LIGHTING);

    glColor3f(0.2, 0.2, 0.8);

    //线框
    glPushMatrix();
    glTranslated(-60.0, 60.0, 0.0);
    glutWireTeapot(10.0);
    glPopMatrix();

    //无光照
    glPushMatrix();
    glTranslated(0.0, 60.0, 0.0);
    glutSolidTeapot(10.0);
    glPopMatrix();

    glEnable(GL_LIGHTING);

    //仅有环境光
    glPushMatrix();
    glTranslated(60.0, 60.0, 0.0);
    glMaterialfv(GL_FRONT_AND_BACK,GL_AMBIENT,mat_ambient);
    glMaterialfv(GL_FRONT_AND_BACK,GL_DIFFUSE,no_mat);
    glMaterialfv(GL_FRONT_AND_BACK,GL_SPECULAR,no_mat);
    glMaterialfv(GL_FRONT_AND_BACK,GL_SHININESS,no_shininess);
    glutSolidTeapot(10.0);
    glPopMatrix();
    //有环境光和漫反射光，采用Gouraud插值着色的茶壶
    glPushMatrix();
    glTranslated(-60.0, 0.0, 0.0);
    glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT, mat_ambient);
    glMaterialfv(GL_FRONT_AND_BACK, GL_DIFFUSE, mat_diffuse);
    glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR, no_mat);
    glMaterialfv(GL_FRONT_AND_BACK, GL_SHININESS, no_shininess);
    glutSolidTeapot(10.0);
    glPopMatrix();

    //有环境光、漫反射光和镜面高光，采用Gouraud插值着色的茶壶
    glPushMatrix();
    glTranslated(0.0, 0.0, 0.0);
    glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT, mat_ambient);
    glMaterialfv(GL_FRONT_AND_BACK, GL_DIFFUSE, mat_diffuse);
    glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR, mat_specular);
    glMaterialfv(GL_FRONT_AND_BACK, GL_SHININESS, high_shininess);
    glutSolidTeapot(10.0);
    glPopMatrix();

    //有环境光、漫反射光和镜面高光，采用Phone插值着色的茶壶
    glPushMatrix();
    glTranslated(60.0, 0.0, 0.0);
    glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT, mat_ambient);
    glMaterialfv(GL_FRONT_AND_BACK, GL_DIFFUSE, mat_diffuse);
    glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR, mat_specular);
    glMaterialfv(GL_FRONT_AND_BACK, GL_SHININESS, high_shininess);
    glutSolidTeapot(10.0);
    glPopMatrix();

    glFlush();
}
```

5.1.2 结果呈现

实现效果如下所示:



5.2 综合-模拟太阳系

5.2.1 实现思路与函数使用

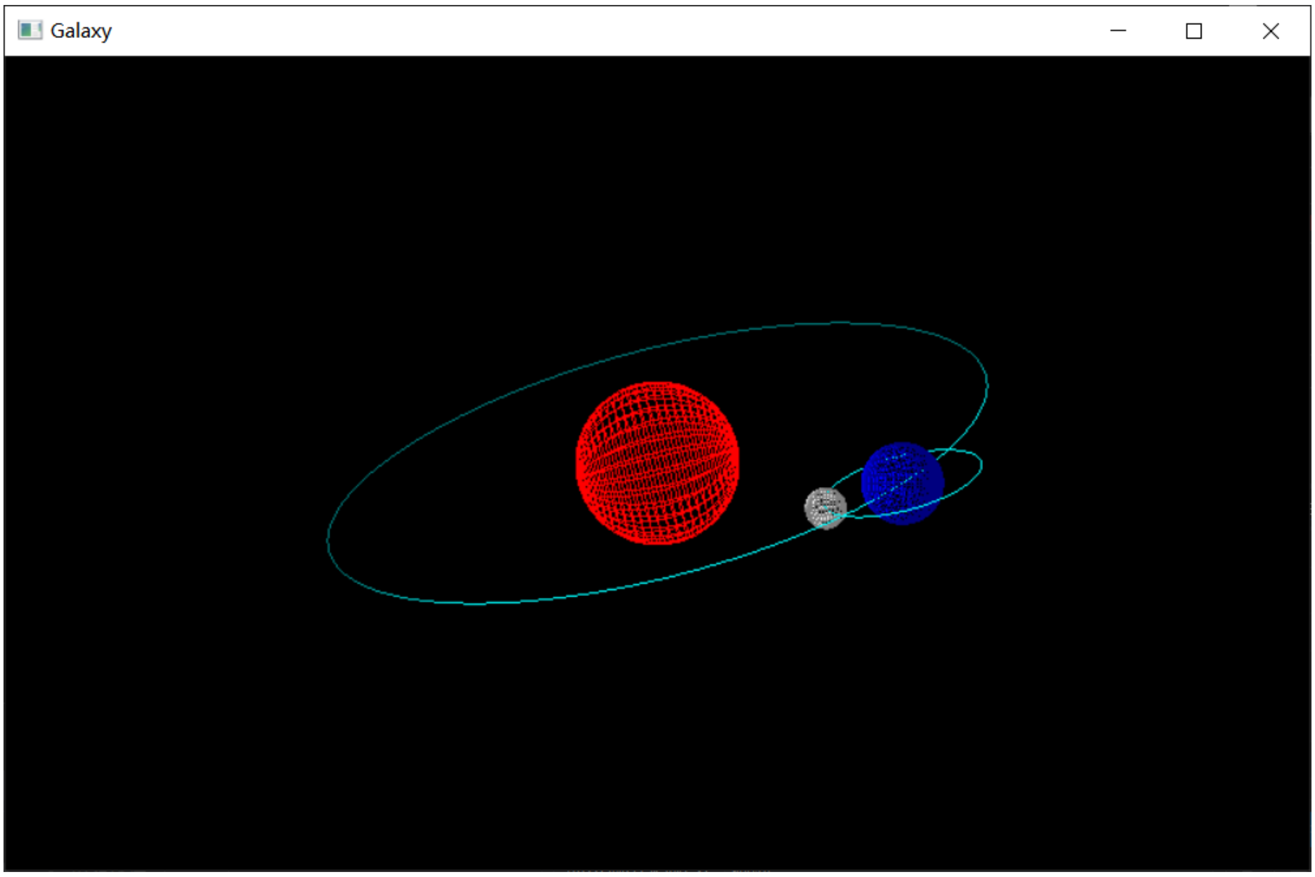
首先声明地-日，地-月，以及地月日的各类自转速度以及地，月的位置坐标，利用math.h中的cos，sin等三角函数来模拟地-月，地-日之间的旋转。然后，利用glRotated()函数，将声明的自转速度作为参数即可完成三个星体的自转。

其中还实现了光影效果以及构造了轨道线；将整个太阳系倾斜30°以便于观察。

主要实现详见源代码：galaxy.cpp

5.2.2 结果呈现

实现效果如下所示,详见 太阳系.mp4:



6. 源代码附录

6.1 第一次上机作业：

1.1 drawGeom.cpp

1.2 MidPointLine.cpp

1.3 MidPointCircle.cpp

6.2 第二次上机作业：

2.1 CubeScale.cpp

2.2 reflection.cpp

2.3 keyboundFunc.cpp

6.3 第三次上机作业：

3.1 drawCurve1.cpp

3.2 Hermite2.cpp

3.3 drawCurve3.cpp

3.4 CubeMove.cpp

3.5 TeapotRotate.cpp

6.4 第四次上机作业：

4.1 BaB.cpp

4.2 BezierTeapot.cpp

4.3 GradientColor.cpp

4.4 Color.cpp

4.5 ConnectDots.cpp

6.5 第五次上机作业：

5.1 light.cpp

5.2 galaxy.cpp