计算机图形学第三次上机实验

2018 年 11 月 19 日

张俊华

16030199025

西安电子科技大学 计算机科学与技术学院 图形学1班

一、实验内容

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。

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

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

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P1(0,0,0)、P2(1,1,1)、

   P3(2,-1,-1)、P4(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
   ```

二、实验环境

Microsoft Visual Studio Community 2017

VisualStudio.15.Release/15.8.6+28010.2041

Microsoft Visual C++ 2017

Windows10 SDK 10.0.17134.0

三、实验步骤

三次曲线的绘制

- 证明以下两条三次曲线的连续性：

$$P_1 = [t^2 - 2t + 1, \quad t^3 - 2t^2 + t]$$
$$P_2 = [t^2, t^3]$$

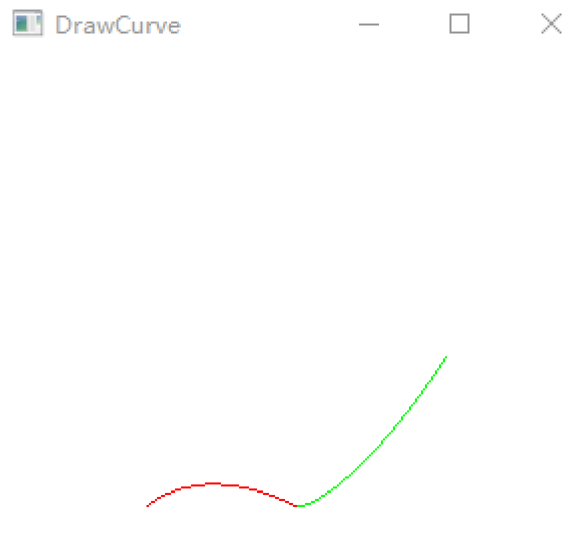
> 证明：
>
> $P_1(1) = P_2(0) = [0, 0]$
>
> $P_1' = [2t - 2, 3t^2 - 4t + 1], P_2' = [2t, 3t^2]$
>
> $P_1'(1) = [0, 0]$, $P_2'(0) = [0, 0]$, $P_1' = P_2'$, 故具有 C1 连续性。
>
> 由于曲线相接处切向量为 0，故不具有 G1 连续性

- 曲线的绘制

  采用直线拟合的方法进行曲线绘制，定义函数 `drawCurvel(int n)` , 以 $1/(n-1)$ 为精度，对于 t 在 0 到 1 区间内的点，进行采样绘制，以直线拟合曲线，完成曲线绘制。

```
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();
}
```

绘制的两条曲线如下图所示，完整的程序代码见附录



三次Hermite曲线的绘制

Hermite曲线是给定曲线段的两个端点坐标以及两端点处的切线矢量来描述的曲线。

三次 Hermit 曲线方程如下：

$$\mathrm{p(t)}=\left(2t^3-3t^2+1\right)p_0+\left(t^3-2t^2+t\right)2m_0+\left(t^3-t^2\right)3m_1+\left(-2t^3+3t^2\right)p_3$$

其中 m_0, m_1 为控制矢量，t 的值介于 0，1之间。

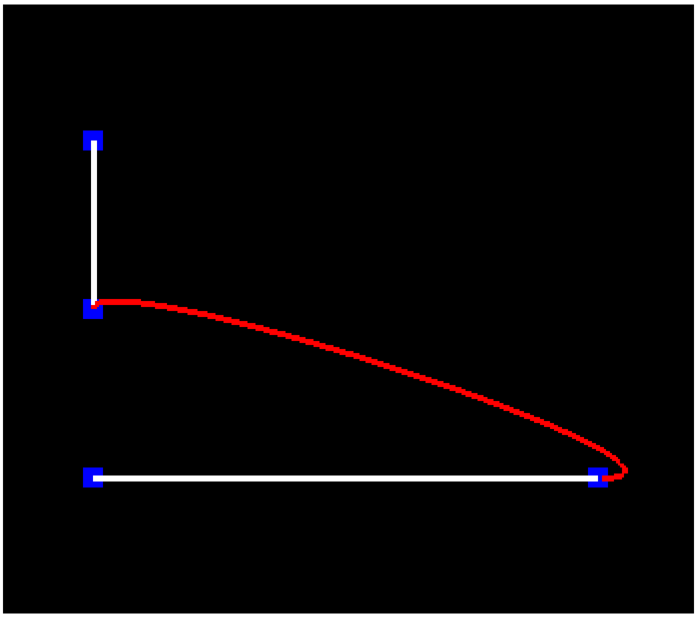
仍然用直线对 Hermite 曲线进行拟合，得到 Hermite 曲线的绘制函数

```
void Hermite(int n)     //精度
{
    //求出相对于控制点的向量（切线）

    Vertex tempC0((c0.x - p0.x) , (c0.y - p0.y) );
    Vertex tempC1((c1.x - p1.x) , (c1.y - p1.y) );

    double delTa = 1.0 / n;
    glBegin(GL_LINE_STRIP);
    for (int i = 0; i < n; i++)
    {
        double t = i * delTa;

        double t1 = 2 * pow(t, 3) - 3 * pow(t, 2) + 1;
        double t2 = -2 * pow(t, 3) + 3 * pow(t, 2);
        double t3 = pow(t, 3) - 2 * pow(t, 2) + t;
        double t4 = pow(t, 3) - pow(t, 2);

        glVertex2d(p0.x*t1 + p1.x*t2 + tempC0.x*t3 + tempC1.x*t4, p0.y*t1 + p1.y*t2 + tempC0.y*t3 + tempC1.y*t4);
    }
    glEnd();
}
```

可以将型值点和控制矢量在场景中进行绘制，便于观察

```
//画出型值点和控制点（蓝色）
glBegin(GL_POINTS);
glVertex2d(p0.x, p0.y);
glVertex2d(p1.x, p1.y);
glVertex2d(c0.x, c0.y);
glVertex2d(c1.x, c1.y);
glEnd();

//画出控制矢量  （白色）
glColor3f(1, 1, 1);
glLineWidth(3);
glBegin(GL_LINES);
glVertex2d(p0.x, p0.y);    glVertex2d(c0.x, c0.y);
glVertex2d(p1.x, p1.y);    glVertex2d(c1.x, c1.y);
glEnd();
```

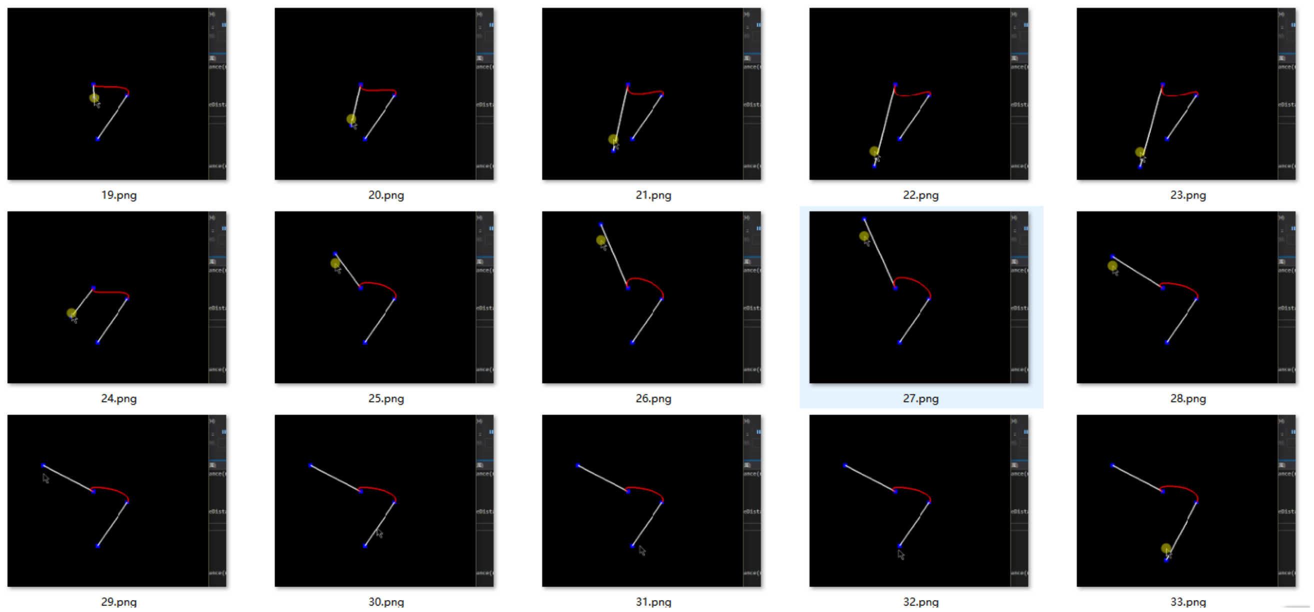
最终的绘制结果如下，红色线条为所要绘制的 Hermite 曲线

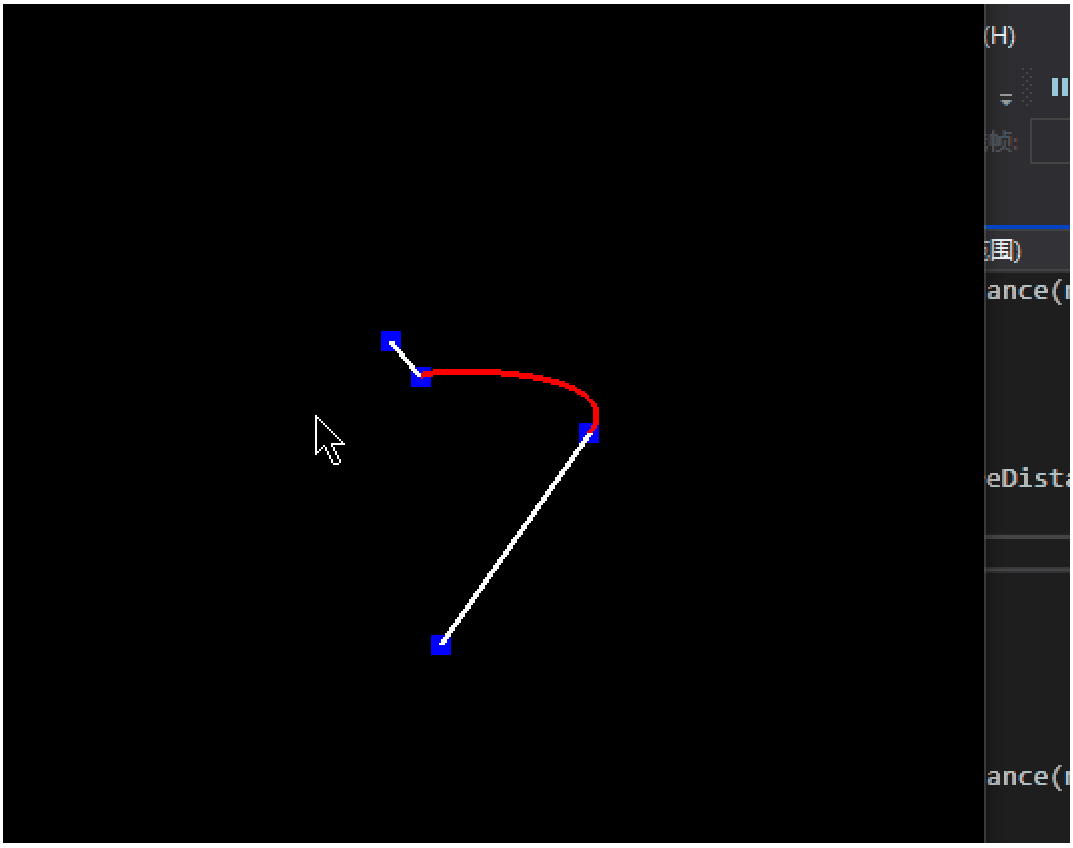


改变控制点观察曲线变化

要直观的观察到曲线的变化效果，可以通过注册鼠标监听函数，通过鼠标对控制点进行拾取和操作，移动鼠标，观察变化效果。

```
void mouse(int button, int state, int x, int y)       //监听鼠标动作
{
    if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN)
    {
        mouseLeftIsDown = true;
    }
    if (button == GLUT_LEFT_BUTTON && state == GLUT_UP)
    {
        mouseLeftIsDown = false;
    }
    if (button == GLUT_RIGHT_BUTTON && state == GLUT_DOWN)
    {
        mouseRightIsDown = true;
    }
    if (button == GLUT_RIGHT_BUTTON && state == GLUT_UP)
    {
        mouseRightIsDown = false;
    }
}
void motion(int x, int y)       //移动点
{
    GLfloat x0 = x;
    GLfloat y0 = y;
    Vertex mouse{ x0,y0 };
    ChangeMouse(mouse);
    if (mouseLeftIsDown)        //左键移动控制点
    {
        if (caculateSquareDistance(mouse, c0) < 0.6)     //防止鼠标移动过快点位无法及时读取
        {
            c0.x = mouse.x;
            c0.y = mouse.y;
        }
        else if (caculateSquareDistance(mouse, c1) < 0.6)
        {
            c1.x = mouse.x;
            c1.y = mouse.y;
        }
    }
    else if (mouseRightIsDown)      //右键移动型值点
    {
        if (caculateSquareDistance(mouse, p0) < 0.6)
        {
            p0.x = mouse.x;
            p0.y = mouse.y;
        }
        else if (caculateSquareDistance(mouse, p1) < 0.6)
        {
            p1.x = mouse.x;
            p1.y = mouse.y;
        }
    }
    glutPostRedisplay();        //重新构图
}
```





效果动画可以点击：http://media.sumblog.cn/blog/20181120/bJpFa5jKslJN.gif 查看

完整代码见附录。

Bezier 曲线、B样条曲线绘制

由题可知，给定的四点为空间中的点，因此，绘制的三维曲线应在三维空间中进行观察，

因此，配置三维投影矩阵 `gluPerspective(80, aspectRatio, 5, 70);`

- 绘制 Bezier 曲线：

  曲线方程：

$$B(t)=P_0(1-t)^3+3P_1t(1-t)^2+3P_2t^2(1-t)+P_3t^3, t\in[0,1]$$

```
void Bezier(Point p1, Point p2, Point p3, Point p4, int n)//精度
{

    double delTa = 1.0 / n;
    glBegin(GL_LINE_STRIP);
    for (int i = 0; i < n; i++)
    {
        double t = i * delTa;

        double t1 = pow(1 - t, 3);
        double t2 = 3 * t*pow(1 - t, 2);
        double t3 = 3 * pow(t, 2)*(1 - t);
        double t4 = pow(t, 3);

        GLfloat X = p1.x*t1 + p2.x*t2 + p3.x*t3 + p4.x*t4;
        GLfloat Y = p1.y*t1 + p2.y*t2 + p3.y*t3 + p4.y*t4;
        GLfloat Z = p1.z*t1 + p2.z*t2 + p3.z*t3 + p4.z*t4;

        glVertex3f(X, Y, Z);
    }
    glEnd();
}

```

- 绘制 B 样条曲线

  三次 B 样条曲线公式为：

  $$\begin{aligned}B(t)= & \frac{1}{6}\left(-t^3+3t^2-3t+1\right)P_0+\frac{1}{6}\left(3i^3-6t^2+4\right)P_1 \\ & +\frac{1}{6}\left(-3t^3+3t^2+3t+1\right)P_2+\frac{1}{6}t^3P_3\end{aligned}$$

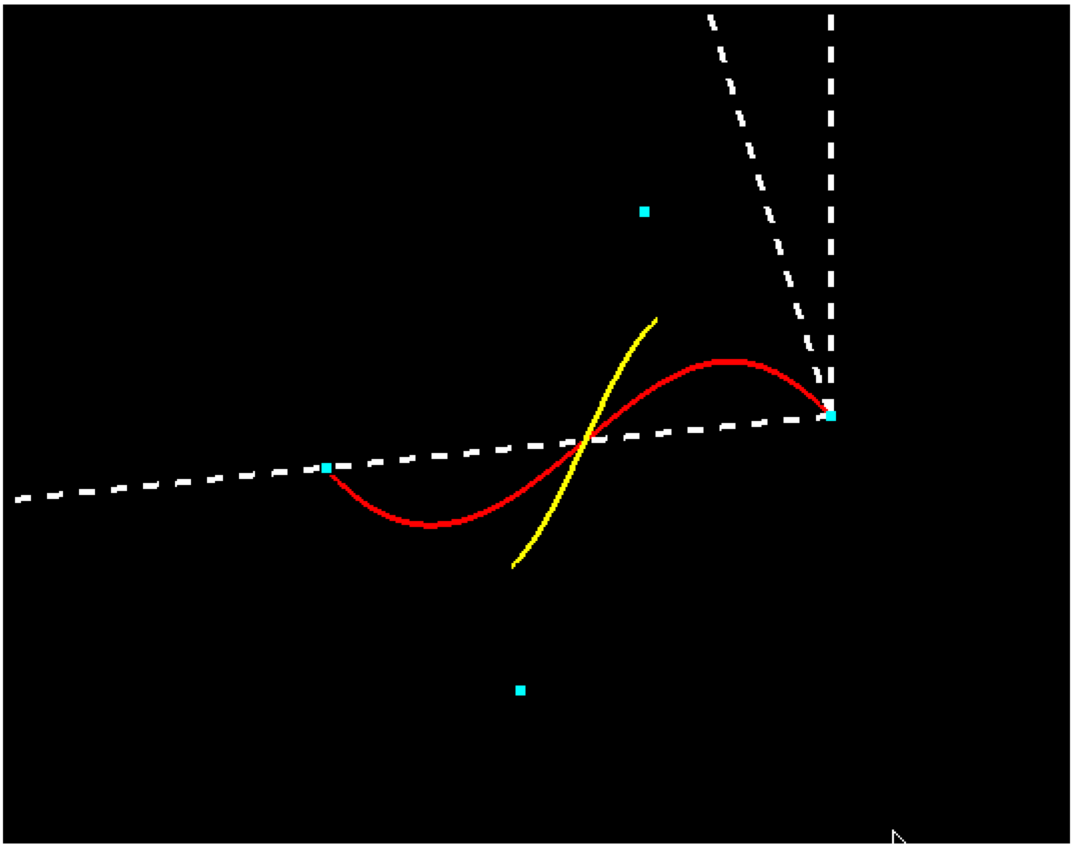
  按照上述公式，对三维空间内的曲线进行拟合绘制，代码如下：

```
void Bspline(Point p1, Point p2, Point p3, Point p4, int n)     //精度
{

    double delTa = 1.0 / n;
    glBegin(GL_LINE_STRIP);
    for (int i = 0; i < n; i++)
    {
        double t = i * delTa;

        double t1 = (- pow(t, 3)+3*pow(t,2)-3*t+1)/6;
        double t2 = (3 * pow(t, 3) - 6 * pow(t, 2) + 4) / 6;
        double t3 = (-3 * pow(t, 3) + 3 * pow(t, 2) + 3 * t + 1) / 6;
        double t4 = pow(t, 3)/6;

        GLfloat X = p1.x*t1 + p2.x*t2 + p3.x*t3 + p4.x*t4;
        GLfloat Y = p1.y*t1 + p2.y*t2 + p3.y*t3 + p4.y*t4;
        GLfloat Z = p1.z*t1 + p2.z*t2 + p3.z*t3 + p4.z*t4;

        glVertex3f(X, Y, Z);
    }
    glEnd();
}
```

在 `RenderScene` 函数中调用以上两个函数，即可完成三维空间中 Bzier 曲线和 B 样条曲线的绘制，为了增强观看效果，编写注册键鼠回调函数，可以调整观察视角。

最终绘制的曲线如图所示：

点击此链接可以查看调整视角的动态效果：http://media.sumblog.cn/blog/20181120/Uiyx1TLCDrF6.gif



图中白色虚线为坐标轴，蓝色点为控制点。可以看到，B 样条曲线（黄色）为不过差值点的非差值曲线。

物体动画绘制

该实验要求实现物体匀速直线运动，和旋转运动。可以在上次三角形动画的基础上完成。

- 修改二维视图到三维视图

  为了实现三维物体动画，由于本实验绘制了三维场景，故需要 `gluPerspective()` 这个函数设置三维透视投影矩阵，在执行命令 `glMatrixMode(GL_PROJECTION)` 和 `glLoadidentity()` 之后使用；它指定了观察的视景体在世界坐标系中的具体大小，其中的参数 `aspect` 应该与窗口的宽高比大小相同。这样显示出的物体才不会被扭曲。

  由 `gluPerspective()` 产生的矩阵是当前矩阵与指定的矩阵相乘得到的，就好像是调用 `glMatrix()` 产生的矩阵一样。为了使透视矩阵替代当前矩阵，在调用 `gluPerspective()` 之前要先调用 `glLoadidentity()` 这个函数，把当前矩阵重置为单位矩阵。

  最终代码如下：

```
void ChangeSize(GLsizei w, GLsizei h)
{
    float ratio;
    if (h == 0)
        h = 1;
    glViewport(0, 0, w, h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    ratio = (float)w / (float)h;
    gluPerspective(60, ratio, 10, 60);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}
```

- 编写 RenderScene() 函数进行场景渲染

绘制三维空间中的茶壶，可以通过调用 glut 库的 `glutWireTeapot` 函数实现：

```
glutWireTeapot(10.0);
```

执行这条语句，以原点为中心，绘制一个边长为 10 的茶壶。

要实现动画效果绘制，其实现方式之一是每次绘制时进行小幅度变换，并通过高速调用重绘函数，实现动画效果。

这里，我采用注册时钟回调函数，实现高速重绘

```
void timer(int value)
{
    time += 1;
    time = time % 365;
    glutTimerFunc(16, timer, 0);
    glutPostRedisplay();
}
```

每调用一次时钟回调函数，就对计时器 `time` 进行增一操作，调用 `glutPostRedisplay()` 函数重绘。 并在16毫秒之后重启时钟回调函数，进行下一帧图形的绘制。

在动画模式下，绘制函数通过对 `time` 中保存的数值，叠加微小变换，确定图形变换状态，修改模型视图矩阵，达到图形变换效果。

```
switch (currentMode)
{
case 0:
glTranslatef(-0.06*time, -0.06*time, 0);
break;
case 1:
glRotatef(1 * time, 0, 1, 0);
}
```

由于模型和视图的变换都通过矩阵运算来实现，在进行变换前，应先设置当前操作的矩阵为“模型视图矩阵”。设置的方法是以 `GL_MODELVIEW` 为参数调用 `glMatrixMode` 函数：

```
glMatrixMode(GL_MODELVIEW);
```

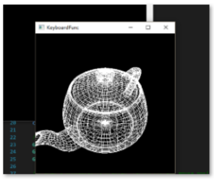
- 注册键鼠操作回调函数，实现运行时动态视角变换

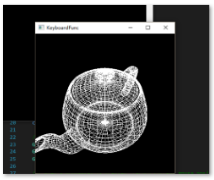
  通过鼠标键盘在执行时动态调整视角，获得更好的结果预览体验，视角变换通过 gluLookAt() 函数，完成从世界坐标系到眼坐标系的转换，通过注册

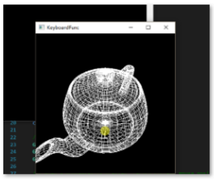
  ```
    glutMouseFunc(MouseFunc);
    glutMotionFunc(MouseMotion);
  ```

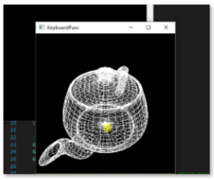
  按住鼠标右键可以上下左右旋转视角。按住鼠标左键并前后移动可以放大或缩小视景。具体函数实现见附录完整代码。

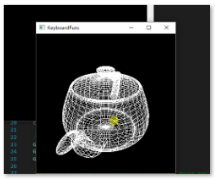
  实验结果如图所示：

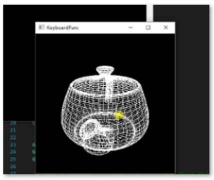

29.png

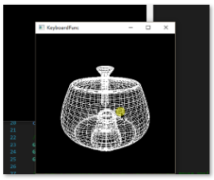

30.png

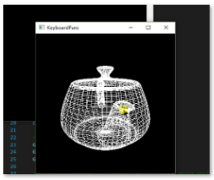

31.png

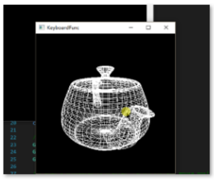

32.png

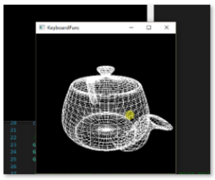

33.png

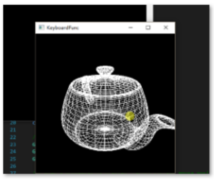

34.png

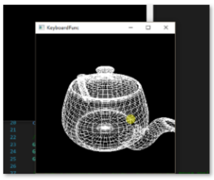

35.png

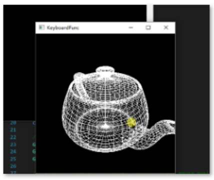

36.png

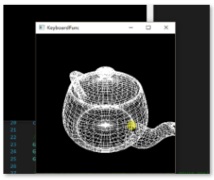

37.png

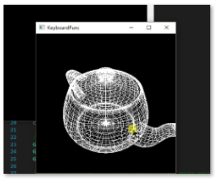

38.png

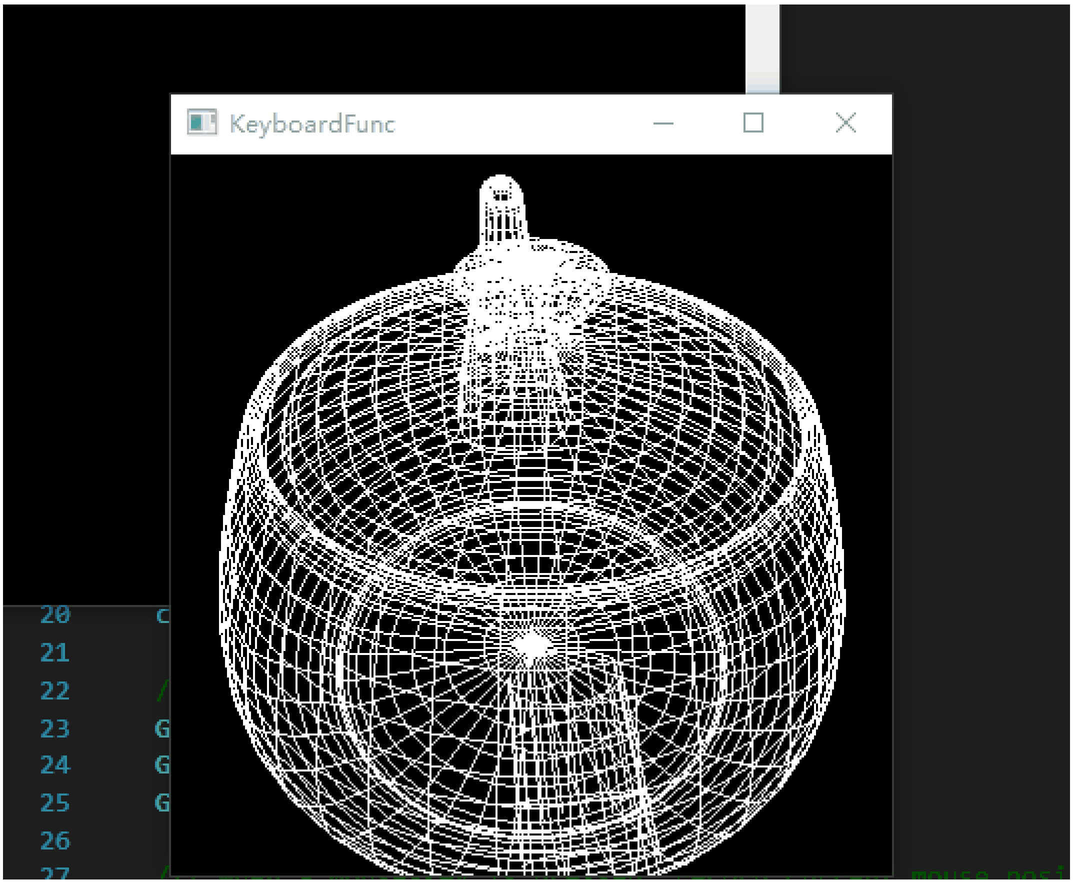

39.png


40.png


41.png


42.png


43.png



点击链接可以查看动画效果：http://media.sumblog.cn/blog/20181120/oWWU5ViBXTTS.gif

四、实验心得体会

通过本次实验，我掌握了 OpenGL 实现二维三维曲线绘制基本方法，了解了 Beizer 曲线、B-样条曲线、Hermite 曲线的参数方程和绘制方法。在此基础上使用键盘、鼠标回调函数对曲线控制点进行动态调节，实现了与绘制场景的基本交互。使我对课程所学知识有了更深刻的了解和印象，进一步增强了我的编程能力。

附录：

实验完整代码：

简单曲线绘制

```
#include <gl/glut.h>
typedef struct {
    float x, y;
}point;
void init()
{
    glClearColor(1.0, 1.0, 1.0, 1.0);
}
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 + 1;
        pixels[i].y = t3;
        glVertex2f(pixels[i].x, pixels[i].y);
    }
    glEnd();
}
void RenderScene()
{
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1.0, 0.0, 0.0);
    drawCurve1(100);
    glColor3f(0.0, 1.0, 0.0);
    drawCurve2(100);
    glFlush();
}
void ChangeSize(GLsizei w, GLsizei h)
{
    GLfloat aspectRatio;
    if (h == 0)
        h = 1;
    glViewport(0, 0, w, h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();

    aspectRatio = (GLfloat)w / (GLfloat)h;
    if (w <= h)
        glOrtho(-1.0, 3.0, -1.0 / aspectRatio, 3.0 / aspectRatio, 1.0, -1.0);
    else
        glOrtho(-1.0*aspectRatio, 3.0*aspectRatio, -1.0, 3.0, 1.0, -1.0);

    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}
int main()
{
    glutInitDisplayMode(GLUT_RGB | GLUT_SINGLE);
    glutCreateWindow("DrawCurve");
    init();
    glutDisplayFunc(RenderScene);
    glutReshapeFunc(ChangeSize);
    glutMainLoop();
    return 0;
}
```

Hermit 曲线绘制

```
#include<gl/glut.h>
#include<math.h>
#include<windows.h>
#include<algorithm>
using namespace std;
struct Vertex
{
    GLfloat x, y;
    Vertex(GLfloat tx, GLfloat ty)
    {
        x = tx;
        y = ty;
    }
};

Vertex p0(0, 1);        //型值点
Vertex p1(3, 0);
Vertex c0(0, 2);        //控制点
Vertex c1(0, 0);

GLsizei W;
GLsizei H;

bool mouseLeftIsDown = false;
bool mouseRightIsDown = false;

void ChangeMouse(Vertex &a)
{
    a.x = a.x / W * 20 - 10;
    a.y = - (a.y / H * 20 * (W / H) - 10 * (W / H));
}

```

```
GLfloat caculateSquareDistance(Vertex &a, Vertex b)
{


    //b.x = (b.x + 10) / 20 * W;
    //b.y = (b.y + 10 / (W / H)) / (20 / (W / H)) *H;
    return (a.x - b.x)*(a.x - b.x) + (a.y - b.y);
}
void Hermite(int n)     //精度
{
    //求出相对于控制点的向量（切线）

    Vertex tempC0((c0.x - p0.x) , (c0.y - p0.y) );
    Vertex tempC1((c1.x - p1.x) , (c1.y - p1.y) );

    double delTa = 1.0 / n;
    glBegin(GL_LINE_STRIP);
    for (int i = 0; i < n; i++)
    {
        double t = i * delTa;

        double t1 = 2 * pow(t, 3) - 3 * pow(t, 2) + 1;
        double t2 = -2 * pow(t, 3) + 3 * pow(t, 2);
        double t3 = pow(t, 3) - 2 * pow(t, 2) + t;
        double t4 = pow(t, 3) - pow(t, 2);

        glVertex2d(p0.x*t1 + p1.x*t2 + tempC0.x*t3 + tempC1.x*t4, p0.y*t1 + p1.y*t2 + tempC0.y*t3 + tempC1.y*t4);
    }
    glEnd();
}
void myDisplay()
{
    glClear(GL_COLOR_BUFFER_BIT);       //清除。GL_COLOR_BUFFER_BIT表示清除颜色

    glPointSize(10.0f);
    glColor3f(0, 0, 1);
    //画出型值点和控制点（蓝色）
    glBegin(GL_POINTS);
    glVertex2d(p0.x, p0.y);
    glVertex2d(p1.x, p1.y);
    glVertex2d(c0.x, c0.y);
    glVertex2d(c1.x, c1.y);
    glEnd();

    //画出控制矢量  （白色）
    glColor3f(1, 1, 1);
    glLineWidth(3);
    glBegin(GL_LINES);
    glVertex2d(p0.x, p0.y);    glVertex2d(c0.x, c0.y);
    glVertex2d(p1.x, p1.y);    glVertex2d(c1.x, c1.y);
    glEnd();

    glColor3f(1, 0, 0);
    Hermite(200);

    glFlush();
    glutSwapBuffers();
}
void mouse(int button, int state, int x, int y)       //监听鼠标动作
{
    if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN)
    {
        mouseLeftIsDown = true;
    }
    if (button == GLUT_LEFT_BUTTON && state == GLUT_UP)
    {
        mouseLeftIsDown = false;
    }
    if (button == GLUT_RIGHT_BUTTON && state == GLUT_DOWN)
    {
        mouseRightIsDown = true;
    }
    if (button == GLUT_RIGHT_BUTTON && state == GLUT_UP)
    {
        mouseRightIsDown = false;
    }
}
void motion(int x, int y)       //移动点
{
    GLfloat x0 = x;
    GLfloat y0 = y;
    Vertex mouse{ x0,y0 };
    ChangeMouse(mouse);
    if (mouseLeftIsDown)        //左键移动控制点
    {
        if (caculateSquareDistance(mouse, c0) < 0.6)      //防止鼠标移动过快点位无法及时读取
        {
            c0.x = mouse.x;
            c0.y = mouse.y;
        }
        else if (caculateSquareDistance(mouse, c1) < 0.6)
        {
            c1.x = mouse.x;
            c1.y = mouse.y;
        }
    }
    else if (mouseRightIsDown)      //右键移动型值点
    {
        if (caculateSquareDistance(mouse, p0) < 0.6)
        {
            p0.x = mouse.x;
            p0.y = mouse.y;
        }
        else if (caculateSquareDistance(mouse, p1) < 0.6)
        {
            p1.x = mouse.x;
            p1.y = mouse.y;
        }
    }
    glutPostRedisplay();        //重新构图
}
void ChangeSize(GLsizei w, GLsizei h)
{
    W = w;
    H = h;
    glViewport(0, 0, w, h);
    double aspectRatio = (GLfloat)w / (GLfloat)h;
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(-10, 10, -10/aspectRatio, 10/aspectRatio);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();

}


int main(int argc, char *argv[])
{
    glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE);
    glutCreateWindow("Hermit");

    puts("\n\t使用Hermite算法，用两顶点两控制点绘制三次曲线。");
    puts("\t左键移动控制点，右键移动型值点");

    glutDisplayFunc(myDisplay);
    glutReshapeFunc(ChangeSize);
    glutMouseFunc(mouse);
    glutMotionFunc(motion);

    glutMainLoop();

    return 0;
}
```

Beizer 和 B 样条

```
#include"GL/glut.h"
#include"GL/glu.h"
#include<memory>
#include <cmath>
using namespace std;

const GLfloat PI = 3.14;

/// record the state of mouse
GLboolean mouserdown = GL_FALSE;
GLboolean mouseldown = GL_FALSE;
GLboolean mousemdown = GL_FALSE;

/// when a mouse-key is pressed, record current mouse position
static GLint mousex = 0, mousey = 0;

static GLfloat center[3] = { 0.0f, 0.0f, 0.0f }; /// center position
static GLfloat eye[3]; /// eye's position

static GLfloat yrotate = PI / 4; /// angle between y-axis and look direction
static GLfloat xrotate = PI / 4; /// angle between x-axis and look direction
static GLfloat celength = 20.0f;/// lenght between center and eye

static GLfloat mSpeed = 0.4f; /// center move speed
static GLfloat rSpeed = 0.02f; /// rotate speed
static GLfloat lSpeed = 0.4f; /// reserved

typedef struct Point {
    GLfloat x, y, z;
};
                              /// calculate the eye position according to center position and angle,length
void CalEyePostion()
{
    if (yrotate > PI / 2.2) yrotate = PI / 2.2;   /// 限制看得方向
    if (yrotate < 0.01)  yrotate = 0.01;
    if (xrotate > 2 * PI)   xrotate = 0.01;
    if (xrotate < 0)   xrotate = 2 * PI;
    if (celength > 50)  celength = 50;     ///  缩放距离限制
    if (celength < 5)   celength = 5;
    /// 下面利用球坐标系计算 eye 的位置,
    eye[0] = center[0] + celength * sin(yrotate) * cos(xrotate);
    eye[2] = center[2] + celength * sin(yrotate) * sin(xrotate);
    eye[1] = center[1] + celength * cos(yrotate);
}

/// center moves
void MoveBackward()               /// center 点沿视线方向水平向后移动
{
    center[0] += mSpeed * cos(xrotate);
    center[2] += mSpeed * sin(xrotate);
    CalEyePostion();
}

void MoveForward()
{
    center[0] -= mSpeed * cos(xrotate);
    center[2] -= mSpeed * sin(xrotate);
    CalEyePostion();
}

/// visual angle rotates
void RotateLeft()
{
    xrotate -= rSpeed;
    CalEyePostion();
}

void RotateRight()
{
    xrotate += rSpeed;
    CalEyePostion();
}

void RotateUp()
{
    yrotate += rSpeed;
    CalEyePostion();
}

void RotateDown()
{
    yrotate -= rSpeed;
    CalEyePostion();
}

/// CALLBACK func for keyboard presses
void KeyFunc(unsigned char key, int x, int y)
{
    switch (key)
    {
    case 'a': RotateLeft(); break;
    case 'd': RotateRight(); break;
    case 'w': MoveForward(); break;
    case 's': MoveBackward(); break;
    case 'q': RotateUp(); break;
    case 'e': RotateDown(); break;
    }
    glutPostRedisplay();
}

/// CALLBACK func for mouse kicks
void MouseFunc(int button, int state, int x, int y)
{
    if (state == GLUT_DOWN)
    {
        if (button == GLUT_RIGHT_BUTTON) mouserdown = GL_TRUE;
        if (button == GLUT_LEFT_BUTTON) mouseldown = GL_TRUE;
        if (button == GLUT_MIDDLE_BUTTON)mousemdown = GL_TRUE;
    }
    else
    {
        if (button == GLUT_RIGHT_BUTTON) mouserdown = GL_FALSE;
        if (button == GLUT_LEFT_BUTTON) mouseldown = GL_FALSE;
        if (button == GLUT_MIDDLE_BUTTON)mousemdown = GL_FALSE;
    }
    mousex = x, mousey = y;
```

```
}

/// CALLBACK func for mouse motions
void MouseMotion(int x, int y)
{
    if (mouserdown == GL_TRUE)
    {       /// 所除以的数字是调整旋转速度的，
        xrotate += (x - mousex) / 180.0f;
        yrotate -= (y - mousey) / 120.0f;
    }

    if (mouseldown == GL_TRUE)
    {
        celength += (y - mousey) / 25.0f;
    }
    mousex = x, mousey = y;
    CalEyePostion();
    glutPostRedisplay();
}

void LookAt()            /// 调用 gluLookAt()，主要嫌直接调用要每次都写好几个参数。。
{
    CalEyePostion();
    gluLookAt(eye[0], eye[1], eye[2], center[0], center[1], center[2], 0, 1, 0);
}

void Bezier(Point p1, Point p2, Point p3, Point p4, int n)     //精度
{

    double delTa = 1.0 / n;
    glBegin(GL_LINE_STRIP);
    for (int i = 0; i < n; i++)
    {
        double t = i * delTa;

        double t1 = pow(1 - t, 3);
        double t2 = 3 * t*pow(1 - t, 2);
        double t3 = 3 * pow(t, 2)*(1 - t);
        double t4 = pow(t, 3);

        GLfloat X = p1.x*t1 + p2.x*t2 + p3.x*t3 + p4.x*t4;
        GLfloat Y = p1.y*t1 + p2.y*t2 + p3.y*t3 + p4.y*t4;
        GLfloat Z = p1.z*t1 + p2.z*t2 + p3.z*t3 + p4.z*t4;

        glVertex3f(X, Y, Z);
    }
    glEnd();
}

void Bspline(Point p1, Point p2, Point p3, Point p4, int n)     //精度
{

    double delTa = 1.0 / n;
    glBegin(GL_LINE_STRIP);
    for (int i = 0; i < n; i++)
    {
        double t = i * delTa;

        double t1 = (- pow(t, 3)+3*pow(t,2)-3*t+1)/6;
        double t2 = (3 * pow(t, 3) - 6 * pow(t, 2) + 4) / 6;
        double t3 = (-3 * pow(t, 3) + 3 * pow(t, 2) + 3 * t + 1) / 6;
        double t4 = pow(t, 3)/6;

        GLfloat X = p1.x*t1 + p2.x*t2 + p3.x*t3 + p4.x*t4;
        GLfloat Y = p1.y*t1 + p2.y*t2 + p3.y*t3 + p4.y*t4;
        GLfloat Z = p1.z*t1 + p2.z*t2 + p3.z*t3 + p4.z*t4;

        glVertex3f(X, Y, Z);
    }
    glEnd();
}
void RenderScene()
{

    glClearColor(0.0, 0.0, 0.0, 0.0);
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1.0, 1.0, 1.0);
    glLoadIdentity();
    glMatrixMode(GL_MODELVIEW);
    LookAt();
    //glRotatef(45.0, 1.0, 1.0, 1.0);//绕着向量（1,1,1）所指定的轴旋转45°
    glEnable(GL_LINE_STIPPLE);
    glLineStipple(1, 0x00FF);
    glBegin(GL_LINES);
    glVertex3f(50, 0, 0.0f);
    glVertex3f(0, 0.0f, 0.0f);
    glEnd();

    glBegin(GL_LINES);
    glVertex3f(0, 50, 0.0f);
    glVertex3f(0, 0.0f, 0.0f);
    glEnd();

    glBegin(GL_LINES);
    glVertex3f(0, 0, 50);
    glVertex3f(0, 0.0f, 0.0f);
    glEnd();
    glDisable(GL_LINE_STIPPLE);
    glLineWidth(3.0);
    glColor3f(1.0, 0.0, 0.0);

    Point p1{ 0,0,0 };
    Point p2{ 1,1,1 };
    Point p3{ 2,-1,-1 };
    Point p4{ 3,0,0 };
    Bezier(p1, p2, p3, p4, 100);
    glColor3f(1.0, 1.0, 0.0);
    Bspline(p1, p2, p3, p4, 100);

    /*
    GLUnurbsObj *theNurb;
    theNurb = gluNewNurbsRenderer();//创建NURBS对象theNurb
    gluNurbsProperty(theNurb, GLU_SAMPLING_TOLERANCE, 10);


    GLfloat ctrlpoints[4][3] = { {0,0,0} ,{1,1,1}, {2,-1,-1},{3,0,0} };

    GLfloat knots[13] = { 0,0,0,0,   0.174058,   0.386051, 0.551328 ,  0.693068, 0.834781,   1,1,1,1 };
    gluBeginCurve(theNurb);
    gluNurbsCurve(theNurb, 13, knots, 3, &ctrlpoints[0][0], 4, GL_MAP1_VERTEX_3);
    gluEndCurve(theNurb);

    */

    glPointSize(5.0);
    glColor3f(0.0, 1.0, 1.0);
    glBegin(GL_POINTS);
    glVertex3f(p1.x,p1.y,p1.z);
    glVertex3f(p2.x, p2.y, p2.z);
    glVertex3f(p3.x, p3.y, p3.z);
    glVertex3f(p4.x, p4.y, p4.z);
    glEnd();


    glFlush();
}

void ChangeSize(GLsizei w, GLsizei h)
{
    GLfloat aspectRatio;
    if (h == 0)
        h = 1;
    glViewport(0, 0, w, h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();

    aspectRatio = (GLfloat)w / (GLfloat)h;
    /*if (w <= h)
        glOrtho(-30.0, 30.0, -30.0 / aspectRatio, 30.0 / aspectRatio, -30.0, 30.0);
    else
        glOrtho(-30.0*aspectRatio, 30.0*aspectRatio, -30.0, 30.0, -30.0, 30.0);*/
    gluPerspective(80, aspectRatio, 5, 70);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}
int main()
{
    glutInitDisplayMode(GLUT_RGB | GLUT_SINGLE);
    glutCreateWindow("Cube");

    glutDisplayFunc(RenderScene);
    glutReshapeFunc(ChangeSize);
    glutKeyboardFunc(KeyFunc);
    glutMouseFunc(MouseFunc);
    glutMotionFunc(MouseMotion);

    glutMainLoop();
}
```

三维物体动画

```
#include <stdlib.h>
#include <stdio.h>
#include <GL/glut.h>
#include<memory>
#include <cmath>

using namespace std;

int currentMode = 0;
int time = 0;
const int ModeNums = 2;

GLfloat X1 = 3;
GLfloat Y1 = 20;
GLfloat X2 = 19;
GLfloat Y2 = 6;
GLfloat X3 = 46;
GLfloat Y3 = 40;

const GLfloat PI = 3.14;

/// record the state of mouse
GLboolean mouserdown = GL_FALSE;
GLboolean mouseldown = GL_FALSE;
GLboolean mousemdown = GL_FALSE;

/// when a mouse-key is pressed, record current mouse position
static GLint mousex = 0, mousey = 0;

static GLfloat center[3] = { 0.0f, 0.0f, 0.0f }; /// center position
static GLfloat eye[3]; /// eye's position

static GLfloat yrotate = PI / 4; /// angle between y-axis and look direction
static GLfloat xrotate = PI / 4; /// angle between x-axis and look direction
static GLfloat celength = 20.0f;/// lenght between center and eye

static GLfloat mSpeed = 0.4f; /// center move speed
static GLfloat rSpeed = 0.02f; /// rotate speed
static GLfloat lSpeed = 0.4f; /// reserved
```

```

void RotateLeft();
void RotateRight();
void MoveForward();
void MoveBackward();
void RotateUp();
void RotateDown();
void LookAt();
void init()
{
    glClearColor(0, 0, 0, 0);
}
void myKey(unsigned char key, int x, int y) //响应ASCII对应键，鼠标的当前x和y位置也被返回。
{
    switch (key)
    {
    case 'a': RotateLeft(); break;
    case 'd': RotateRight(); break;
    case 'w': MoveForward(); break;
    case 's': MoveBackward(); break;
    case 'q': RotateUp(); break;
    case 'e': RotateDown(); break;
    }
    switch (key)
    {
    case ' ': currentMode = (currentMode + 1) % ModeNums;
        time = 0;
        glutPostRedisplay();
        break;
    case 27:  exit(-1);
    }
}

GLfloat centx(GLfloat x1, GLfloat y1, GLfloat x2, GLfloat y2, GLfloat x3, GLfloat y3) {
    return (x1 + x2 + x3) / 3;
}
GLfloat centy(GLfloat x1, GLfloat y1, GLfloat x2, GLfloat y2, GLfloat x3, GLfloat y3) {
    return (y1 + y2 + y3) / 3;
}
void RenderScene()
{
    glClear(GL_COLOR_BUFFER_BIT);
    glLoadIdentity();
    glMatrixMode(GL_MODELVIEW);
    LookAt();
    switch (currentMode)
    {
    case 0:
        glTranslatef(-0.06*time, -0.06*time, 0);
        break;
    case 1:
        glRotatef(1 * time, 0, 1, 0);
    }
    glutWireTeapot(10);
    glEnd();
    glutSwapBuffers();

}

void timer(int value)
{

    time += 1;
    time = time % 365;
    glutTimerFunc(16, timer, 0);
    glutPostRedisplay();
}

void ChangeSize(GLsizei w, GLsizei h)
{
    float ratio;
    if (h == 0)
        h = 1;
    glViewport(0, 0, w, h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    ratio = (float)w / (float)h;
    gluPerspective(60, ratio, 10, 60);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}





                              /// calculate the eye position according to center position and angle,length
void CalEyePostion()
{
    if (yrotate > PI / 2.2) yrotate = PI / 2.2;   ///
    if (yrotate < 0.01)  yrotate = 0.01;
    if (xrotate > 2 * PI)    xrotate = 0.01;
    if (xrotate < 0)   xrotate = 2 * PI;
    if (celength > 50)  celength = 50;     ///
    if (celength < 5)   celength = 5;
    eye[0] = center[0] + celength * sin(yrotate) * cos(xrotate);
    eye[2] = center[2] + celength * sin(yrotate) * sin(xrotate);
    eye[1] = center[1] + celength * cos(yrotate);
}

/// center moves
void MoveBackward()              /// center
```

```
{
    center[0] += mSpeed * cos(xrotate);
    center[2] += mSpeed * sin(xrotate);
    CalEyePostion();
}

void MoveForward()
{
    center[0] -= mSpeed * cos(xrotate);
    center[2] -= mSpeed * sin(xrotate);
    CalEyePostion();
}

/// visual angle rotates
void RotateLeft()
{
    xrotate -= rSpeed;
    CalEyePostion();
}

void RotateRight()
{
    xrotate += rSpeed;
    CalEyePostion();
}

void RotateUp()
{
    yrotate += rSpeed;
    CalEyePostion();
}

void RotateDown()
{
    yrotate -= rSpeed;
    CalEyePostion();
}



/// CALLBACK func for mouse kicks
void MouseFunc(int button, int state, int x, int y)
{
    if (state == GLUT_DOWN)
    {
        if (button == GLUT_RIGHT_BUTTON) mouserdown = GL_TRUE;
        if (button == GLUT_LEFT_BUTTON) mouseldown = GL_TRUE;
        if (button == GLUT_MIDDLE_BUTTON)mousemdown = GL_TRUE;
    }
    else
    {
        if (button == GLUT_RIGHT_BUTTON) mouserdown = GL_FALSE;
        if (button == GLUT_LEFT_BUTTON) mouseldown = GL_FALSE;
        if (button == GLUT_MIDDLE_BUTTON)mousemdown = GL_FALSE;
    }
    mousex = x, mousey = y;
}

/// CALLBACK func for mouse motions
void MouseMotion(int x, int y)
{
    if (mouserdown == GL_TRUE)
    {
        xrotate += (x - mousex) / 180.0f;
        yrotate -= (y - mousey) / 120.0f;
    }

    if (mouseldown == GL_TRUE)
    {
        celength += (y - mousey) / 25.0f;
    }
    mousex = x, mousey = y;
    CalEyePostion();
    glutPostRedisplay();
}

void LookAt()
{
    CalEyePostion();
    gluLookAt(eye[0], eye[1], eye[2], center[0], center[1], center[2], 0, 1, 0);
}

int main()
{
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
    glutInitWindowPosition(50, 50);
    glutInitWindowSize(360, 360);
    glutCreateWindow("KeyboardFunc");

    init();
    glutDisplayFunc(RenderScene);
    glutReshapeFunc(ChangeSize);
    glutKeyboardFunc(myKey);  //为当前窗口设置键盘回调函数。
    glutMouseFunc(MouseFunc);
    glutMotionFunc(MouseMotion);
    glutTimerFunc(0, timer, 0);

    printf("Press space to continue,press escape to exit!\n");
    glutMainLoop();
}
```