实验四　自由曲线交互式绘制

姓名 _刘子言_　学号_20002462_　专业班级_计 203_　成绩_____

实验日期_2022.5.12-2022.5.26_　实验地点_线上_　指导教师(签名)_李建华_

一．实验目的

1、　学习 OpenGL Bezier 曲线函数。
2、　利用鼠标、键盘的橡皮筋技术，交互式绘制 Bezier 曲线。

二．实验工具与设备

计算机，开发软件为 CodeBlocks （配置 OpenGL）

三、实验内容

在本科学习平台（s.ecust.edu.cn）资料栏下，下载以下 3 个文件

① graphicType.h：给出二维point、三维坐标点point3D、矩形rect、颜色color的结构定义。

② drawBezier.h、drawBezier.cpp：给出绘制Bezier曲线、绘制折线的方法。

1、 在实验二的基础上如图 1，增加“Bezier 曲线”菜单项，菜单项值为 4。利用键盘橡皮筋技术，交互确定控制点序列，绘制 Bezier 曲线和绘制控制点折线。

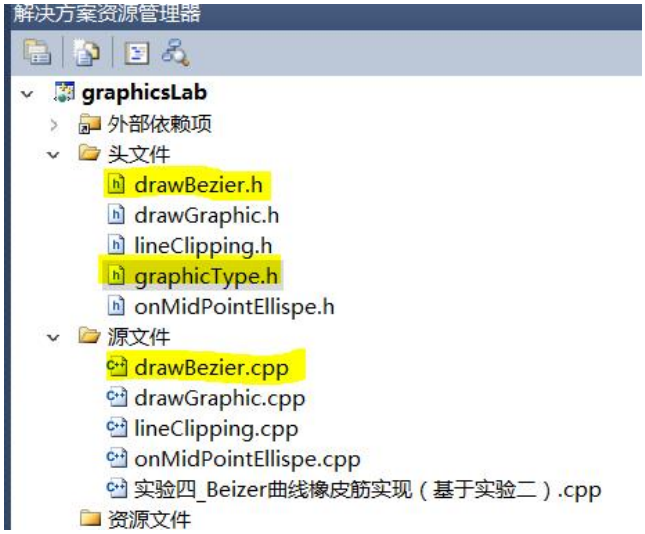


图1 在实验二的基础上的本实验

```
vector <point> pointVertex;    //控制多边形顶点序列
bool isEnd=false;              //标识控制多边形是否确定
const GLint ControlN=4;        //由4个控制点定义三次bezier曲线段
color redColor(1,0,0),blackColor(0,0,0);//红色、黑色
```

① 当isEnd为flase时，利用给定的drawPolygonalLine()方法、以虚线形式（见附件1）显示由控制点序列绘制的折线（控制多边形）。当鼠标移动时出现下一段折线的橡皮筋效果，

键盘'p'确定新的控制点坐标，并保存在pointVertex向量数组。键盘'e'结束橡皮筋交互确定控制点，并设置isEnd为true。(或者采用全鼠标控制的交互方式，利用鼠标右键单击获得一个新的控制点，鼠标左键单击获得最后一个控制点。)

② 根据pointVertex向量数组（见附件2），确定有效的控制点，如果控制点数n<4，利用给定的drawBezierCurve()方法（如图2）绘制一段n-1次Bezier曲线段，否则绘制一段三次Bezier曲线段。

```
//如果控制点数n<4，绘制一段n-1次Bezier曲线段，否则绘制一段三次Bezier曲线
drawBezierCurve(pointVertex,redColor);
```

图2 drawBezierCurve()方法1

实验代码：

1、头文件：graphicType.h

```
//三维点类型 point
typedef struct Point3D {
    double x, y,z;
    Point3D(double a = 0.0,double b = 0.0,double c = 0.0)
    {
        x = a, y = b;z=c;
    }
} point3D;
//二维点类型 point
typedef struct Point {
    int x, y;
    Point(int a = 0, int b = 0) {
        x = a;
        y = b;
    }
} point;
//矩形类型 rect
typedef struct Rectangle{
    float w_xmin,w_ymin;
    float w_xmax,w_yman;
    Rectangle(float xmin = 0.0, float ymin = 0.0,float xmax=0.0,float yman=0.0){
        w_xmin = xmin;    w_ymin = ymin;
        w_xmax = xmax;    w_yman = yman;
    }
}rect;
//颜色类型 color
typedef struct Color{
    int r,g,b;
    Color(int red= 0,int green = 0,int blue=0){
        r=red;
        g=green;
```

```
        b=blue;
    }
}color;
```

2、源文件：main.cpp

```
#include <iostream>
#include <vector>
#include <windows.h>
#include <GL/glut.h>
#include <cstdio>
#include "graphicType.h"
using namespace std;
int winWidth = 600, winHeight = 400;
int iMode = 0;                          //菜单选择

vector <point> pointVertex;             //控制折线顶点序列
bool isEnd=false;                       //标识折线是否绘制完成
const GLint ControlN = 4;               //由 4 个控制点定义三次 bezier 曲线段
color redColor(1,0,0),blackColor(0,0,0); //红色、黑色
int iMousePointNum = 0;                 //鼠标单击确定折线顶点个数

//根据点序列向量数组 points，绘制折线
void drawPolygonalLine(vector<point> &points,color &c )
{
    glColor3f(c.r, c.g, c.b);
    glBegin (GL_LINE_STRIP);
    for(int i=0;i<iMousePointNum;i++)
    {
        glVertex2i(points[i].x, points[i].y);
    }
    glEnd();
}

//如果控制点数 n<4，绘制一段 n-1 次 Bezier 曲线段，否则绘制一段三次 Bezier 曲线段
void drawBezierCurve(vector<Point> &points,color &c)
{
    GLfloat ControlP[4][3];
    int iPointNum = iMousePointNum;
    int i;
    if(iPointNum<4)   //控制点数<4，绘制一段 n-1 次 Bezier 曲线段
    {
        for(i=0;i<iPointNum;i++)
        {
```

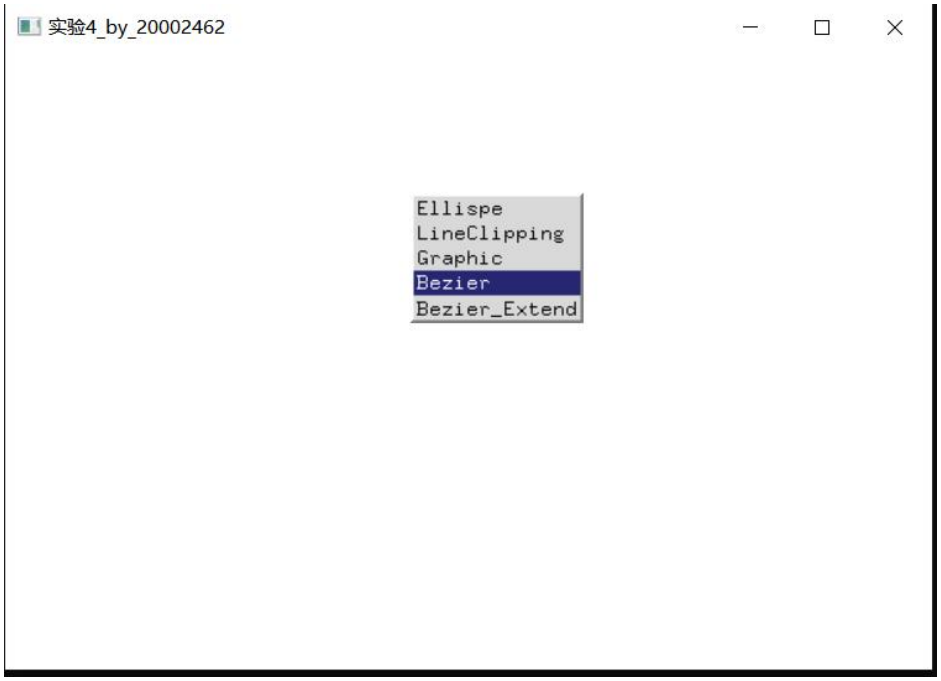
```
            ControlP[i][0]=points[i].x;
            ControlP[i][1]=points[i].y;
            ControlP[i][2]=0.0;
        }
    }
    else                    //控制点数>=4，绘制一段三次 Bezier 曲线段
    {
        for(i=0;i<4;i++)
        {
            ControlP[i][0]=points[i].x;
            ControlP[i][1]=points[i].y;
            ControlP[i][2]=0.0;
        }
    }
    glColor3f(c.r, c.g, c.b);
    glPointSize(2);
    if(iPointNum > 4)
        glMap1f(GL_MAP1_VERTEX_3, 0.0, 1.0, 3, 4, *ControlP);   //定义一维取值器
    else
        glMap1f(GL_MAP1_VERTEX_3, 0.0, 1.0, 3, iPointNum, *ControlP);   //定义一维取值器
    glEnable(GL_MAP1_VERTEX_3);
    glMapGrid1f(100, 0.0, 1.0);          //生成均匀分布的一维网格参数值
    glEvalMesh1(GL_LINE, 0, 100);   //绘制 Bezier 曲线
}

void Initial(void)                          //初始化窗口
{
    glClearColor(1.0f, 1.0f, 1.0f, 1.0f);   //设置窗口背景颜色为白色
}
void ChangeSize(int w, int h)
{
    winWidth = w;
    winHeight = h;
    glViewport(0, 0, w, h);                     //指定窗口显示区域
    glMatrixMode(GL_PROJECTION);        //设置投影参数
    glLoadIdentity();
    gluOrtho2D(0.0,winWidth,0.0,winHeight);
}
void ProcessMenu(int value)                     //处理菜单响应
{
    iMode = value;
    glutPostRedisplay();
}
```

```
void MousePlot(GLint button, GLint action, GLint xMouse, GLint yMouse)   //鼠标响应
{
    Point p0;
    if(button == GLUT_LEFT_BUTTON && action == GLUT_DOWN && iMode==4)
    {   //鼠标左击，绘制折线上的点
        p0.x = xMouse;
        p0.y = winHeight - yMouse;
        pointVertex.push_back(p0);
        iMousePointNum++;
    }
    else if(button == GLUT_RIGHT_BUTTON && action == GLUT_DOWN && iMode==4)
    {   //鼠标右击，结束折线绘制
        isEnd = TRUE;   //结束
    }
}

void PassiveMouseMove (GLint xMouse, GLint yMouse)   //鼠标移动过程中
{
    if(iMousePointNum>=1 && !isEnd)   //正在绘制折线
    {
        Point p0;
        p0.x = xMouse;
        p0.y = winHeight - yMouse;
        pointVertex.push_back(p0);
        glutPostRedisplay();                //窗口执行重新绘制操作
        pointVertex.pop_back();
    }
}

void Display(void)
{
    glClear(GL_COLOR_BUFFER_BIT);    //用当前背景色填充窗口
    switch(iMode)   //若选择菜单 4，则进行 Bezier 曲线的绘制
    {
        case 4:
            glColor3f(0.0f, 0.0f, 0.0f);          //设置折线颜色为黑色
            glLineStipple(1, 0x00FF);          //设置折线类型为虚线
            glEnable(GL_LINE_STIPPLE);
            if(iMousePointNum >= 1)            //绘制折线
            {
                drawPolygonalLine(pointVertex, blackColor);
                if(!isEnd)
                {
                    glBegin(GL_LINES);
```

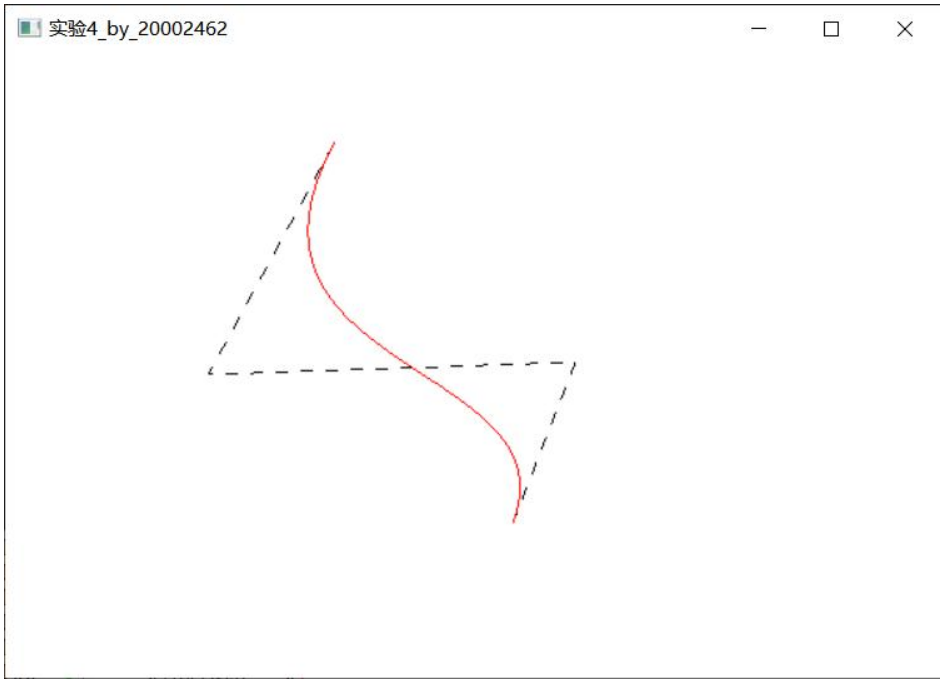
```
                    glVertex2i(pointVertex[iMousePointNum-1].x,pointVertex[iMousePointNum-1].y);
                    glVertex2i(pointVertex[iMousePointNum].x,pointVertex[iMousePointNum].y);
                    glEnd();
                }
            }
            glDisable(GL_LINE_STIPPLE);
            if(isEnd)               //如果折线停止绘制了，便绘制 Bezier 曲线
            {
                drawBezierCurve(pointVertex, redColor);
            }
            break;
        default: break;
    }
    glutSwapBuffers();              //交换缓冲区
}

int main(int argc, char* argv[])
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);  //使用“双缓存”以及 RGB 模型
    glutInitWindowSize(600,400);
    glutInitWindowPosition(200,200);
    glutCreateWindow("实验 4_by_20002462");

    //创建菜单并定义菜单回调函数
    glutCreateMenu(ProcessMenu);
    glutAddMenuEntry("Ellispe", 1);
    glutAddMenuEntry("LineClipping", 2);
    glutAddMenuEntry("Graphic", 3);
    glutAddMenuEntry("Bezier", 4);    //选择菜单 4
    glutAddMenuEntry("Bezier_Extend", 5);
    glutAttachMenu(GLUT_MIDDLE_BUTTON);  //将主菜单与鼠标中心键关联

    glutDisplayFunc(Display);
    glutReshapeFunc(ChangeSize);                    //指定窗口在整形回调函数
    glutMouseFunc(MousePlot);                       //指定鼠标响应函数
    glutPassiveMotionFunc(PassiveMouseMove);        //指定鼠标移动响应函数
    Initial();
    glutMainLoop();
    return 0;
}
```

运行结果：

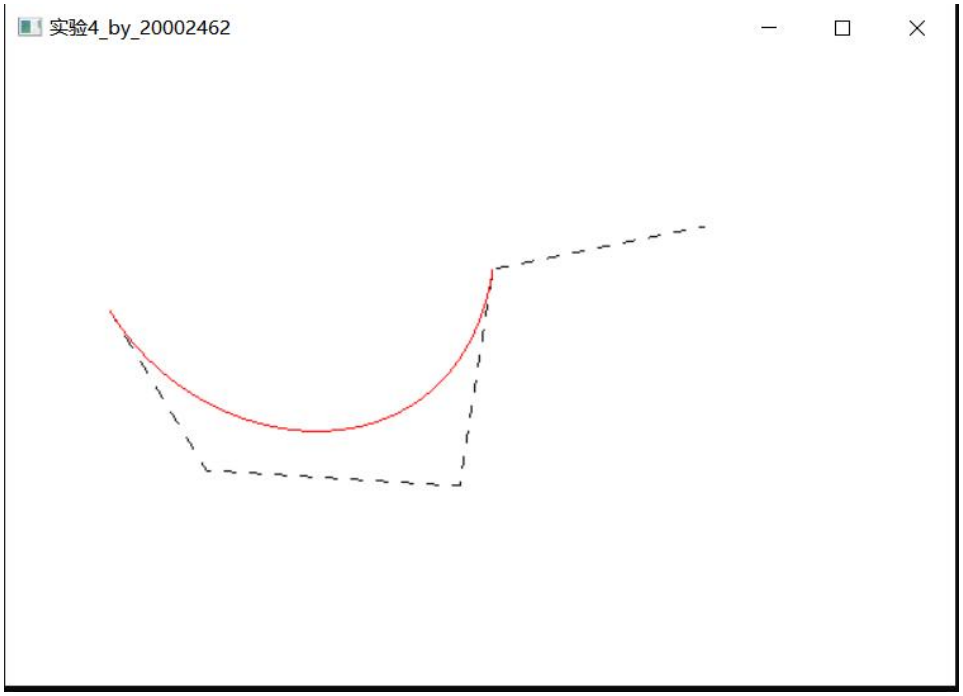
1、点击鼠标中心键，选择菜单“Bezier”：



2、点击鼠标左键，开始绘制折线段，如果确定的控制点数 n<=4，则在点击鼠标右键结束折线绘制后，依据确定的控制点绘制一段红色的 n-1 次的 Bezier 曲线：



3、如果绘制折线时控制点数 n>4，则在点击右键结束折线绘制之后，绘制一段三次的 Bezier 曲线：



四、拓展实验

1、 举一反三，实现 drawBezierCurve(vector<Point> &points,int n,color &c)方法，通过（m×3+1）控制点，实现 m 段三次 Bezier 曲线拼接，体现 Bezier 曲线的 G1 连续性。给出调用演示。

实验代码：

1、头文件：graphicType.h

```
//三维点类型 point
typedef struct Point3D {
    double x, y,z;
    Point3D(double a = 0.0,double b = 0.0,double c = 0.0)
    {
        x = a, y = b;z=c;
    }
} point3D;
//二维点类型 point
typedef struct Point {
    int x, y;
    Point(int a = 0, int b = 0) {
        x = a;
        y = b;
    }
} point;
//矩形类型 rect
typedef struct Rectangle{
    float w_xmin,w_ymin;
    float w_xmax,w_yman;
    Rectangle(float xmin = 0.0, float ymin = 0.0,float xmax=0.0,float yman=0.0){
        w_xmin = xmin;    w_ymin = ymin;
        w_xmax = xmax;    w_yman = yman;
    }
}rect;
//颜色类型 color
typedef struct Color{
    int r,g,b;
    Color(int red= 0,int green = 0,int blue=0){
        r=red;
        g=green;
        b=blue;
    }
}color;
```

2、源文件：main.cpp

```
#include <iostream>
#include <vector>
#include <windows.h>
#include <GL/glut.h>
#include <cstdio>
#include "graphicType.h"
using namespace std;
int winWidth = 600, winHeight = 400;
int iMode = 0;                          //菜单选择

vector <point> pointVertex;             //控制折线顶点序列
bool isEnd=false;                       //标识折线是否绘制完成
const GLint ControlN = 4;               //由 4 个控制点定义三次 bezier 曲线段
color redColor(1,0,0),blackColor(0,0,0); //红色、黑色
int iMousePointNum = 0;                 //鼠标单击确定折线顶点个数

//根据点序列向量数组 points，绘制折线
void drawPolygonalLine(vector<point> &points,color &c )
{
    glColor3f(c.r, c.g, c.b);
    glBegin(GL_LINE_STRIP);
    for(int i=0;i<iMousePointNum;i++)
    {
        glVertex2i(points[i].x, points[i].y);
    }
    glEnd();
}

//通过（m×3+1）控制点，绘制 m 段 3 次 Bezier 曲线段
void drawBezierCurve(vector<Point> &points,int n,color &c)
{
    for(int i=0;i<n-1;i++)
    {
        int j = 3*i;
        GLfloat ControlP[4][3];
        for(int k=0;k<4;k++)
        {
            ControlP[k][0] = points[j].x;
            ControlP[k][1] = points[j].y;
            ControlP[k][2] = 0.0;
            j++;
        }
        glColor3f(c.r, c.g, c.b);
```

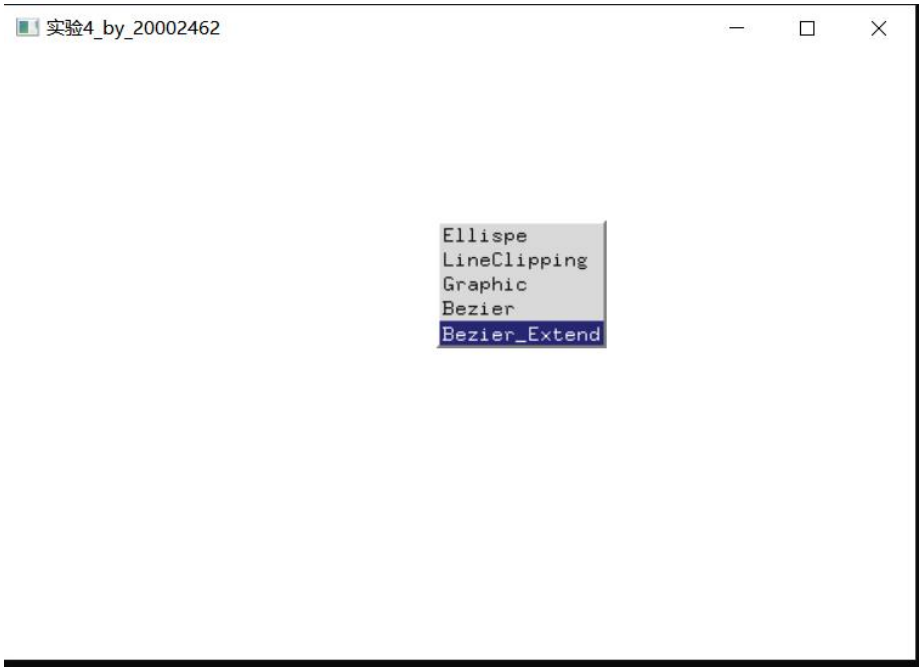
```
            glPointSize(2);
            glMap1f(GL_MAP1_VERTEX_3, 0.0, 1.0, 3, 4, *ControlP);
            glEnable(GL_MAP1_VERTEX_3);
            glMapGrid1f(100, 0.0, 1.0);              //生成均匀分布的一维网络参数值
            glEvalMesh1(GL_LINE, 0, 100);       //绘制 Bezier 曲线
            if(j>=iMousePointNum)break;
        }
}

void Initial(void)                                  //初始化窗口
{
    glClearColor(1.0f, 1.0f, 1.0f, 1.0f);        //设置窗口背景颜色为白色
}
void ChangeSize(int w, int h)
{
    winWidth = w;
    winHeight = h;
    glViewport(0, 0, w, h);                         //指定窗口显示区域
    glMatrixMode(GL_PROJECTION);          //设置投影参数
    glLoadIdentity();
    gluOrtho2D(0.0,winWidth,0.0,winHeight);
}
void ProcessMenu(int value)                     //处理菜单响应
{
    iMode = value;
    glutPostRedisplay();
}

void MousePlot(GLint button, GLint action, GLint xMouse, GLint yMouse)
{
    Point p0;
    if(button == GLUT_LEFT_BUTTON && action == GLUT_DOWN && iMode==5)
    {    //鼠标左击，绘制折线上的点
        p0.x = xMouse;
        p0.y = winHeight - yMouse;
        pointVertex.push_back(p0);
        iMousePointNum++;
    }
    else if(button == GLUT_RIGHT_BUTTON && action == GLUT_DOWN && iMode==5)
    {    //鼠标右击，结束绘制折线
        isEnd = TRUE;//结束
    }
}
```

```
void PassiveMouseMove (GLint xMouse, GLint yMouse)   //鼠标移动过程中
{
    if(iMousePointNum>=1 && !isEnd)   //正在绘制折线
    {
        Point p0;
        p0.x = xMouse;
        p0.y = winHeight - yMouse;
        pointVertex.push_back(p0);
        glutPostRedisplay();                //窗口执行重新绘制操作
        pointVertex.pop_back();
    }
}

void Display(void)
{
    glClear(GL_COLOR_BUFFER_BIT);    //用当前背景色填充窗口
    switch(iMode)    //若选择菜单 5，则进行 m 段 3 次 Bezier 曲线的绘制
    {
        case 5:
            glColor3f(0.0f, 0.0f, 0.0f);        //设置折线颜色为黑色
            glLineStipple(1, 0x00FF);        //设置折线类型为虚线
            glEnable(GL_LINE_STIPPLE);
            if(iMousePointNum >= 1)        //绘制折线
            {
                drawPolygonalLine(pointVertex, blackColor);
                if(!isEnd)
                {
                    glBegin(GL_LINES);
                   glVertex2i(pointVertex[iMousePointNum-1].x, pointVertex[iMousePointNum-1].y);
                    glVertex2i(pointVertex[iMousePointNum].x, pointVertex[iMousePointNum].y);
                    glEnd();
                }
            }
            glDisable(GL_LINE_STIPPLE);
            if(isEnd)          //如果结束折线段的绘制，则绘制 Bezier 曲线，m*3+1 个点对应 m 段
            {
                drawBezierCurve(pointVertex, 4, redColor);
            }
            break;
        default: break;
    }
    glutSwapBuffers();    //交换缓冲区
}
```

```
int main(int argc, char* argv[])
{
	glutInit(&argc, argv);
	glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);	//使用“双缓存”以及 RGB 模型
	glutInitWindowSize(600,400);
	glutInitWindowPosition(200,200);
	glutCreateWindow("实验 4_by_20002462");

	//创建菜单并定义菜单回调函数
	glutCreateMenu(ProcessMenu);
	glutAddMenuEntry("Ellispe", 1);
	glutAddMenuEntry("LineClipping", 2);
	glutAddMenuEntry("Graphic", 3);
	glutAddMenuEntry("Bezier", 4);
	glutAddMenuEntry("Bezier_Extend", 5);	//选择菜单 5
	glutAttachMenu(GLUT_MIDDLE_BUTTON);	//将主菜单与鼠标中心键关联

	glutDisplayFunc(Display);
	glutReshapeFunc(ChangeSize);			//指定窗口在整形回调函数
	glutMouseFunc(MousePlot);			//指定鼠标响应函数
	glutPassiveMotionFunc(PassiveMouseMove);	//指定鼠标移动响应函数
	Initial();
	glutMainLoop();
	return 0;
}
```

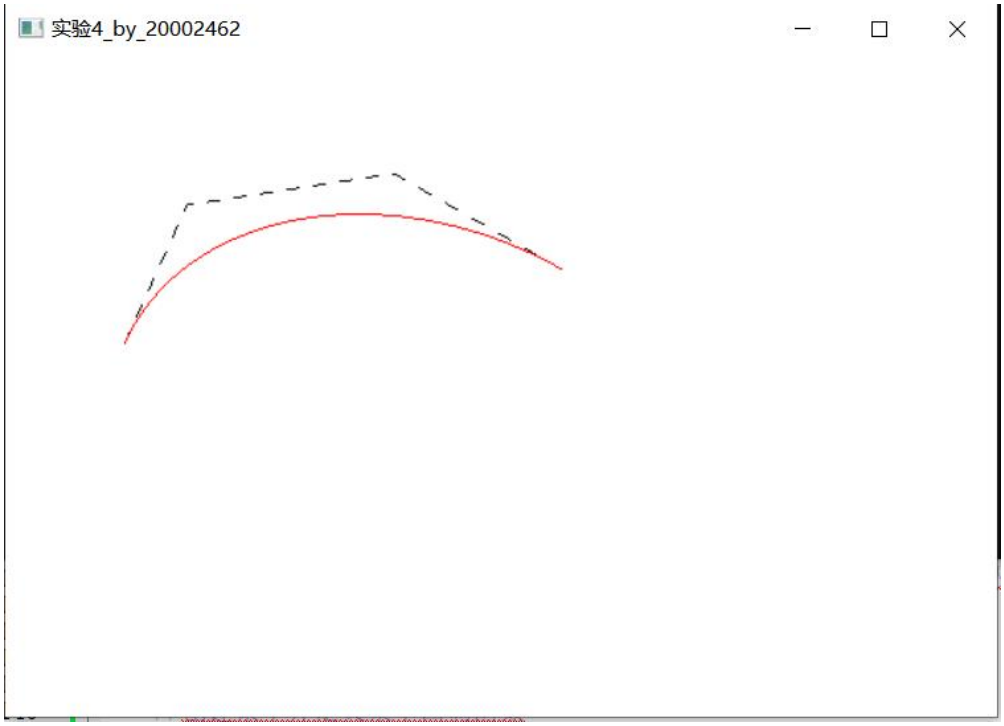
运行结果：

1、点击鼠标中心键，选择菜单“Bezier_Extend”：

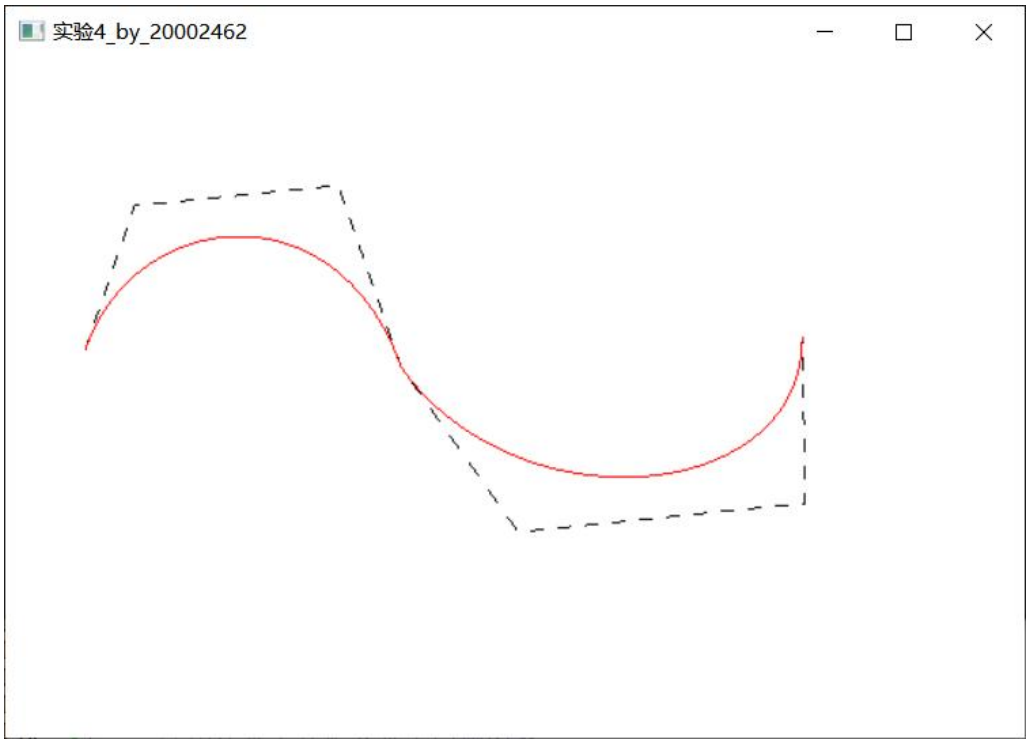


3、点击鼠标左键，开始绘制折线段，点击右键结束折线绘制，通过确定的（m×3+1）个控制点，绘制出 m 段 3 次 Bezier 曲线段：

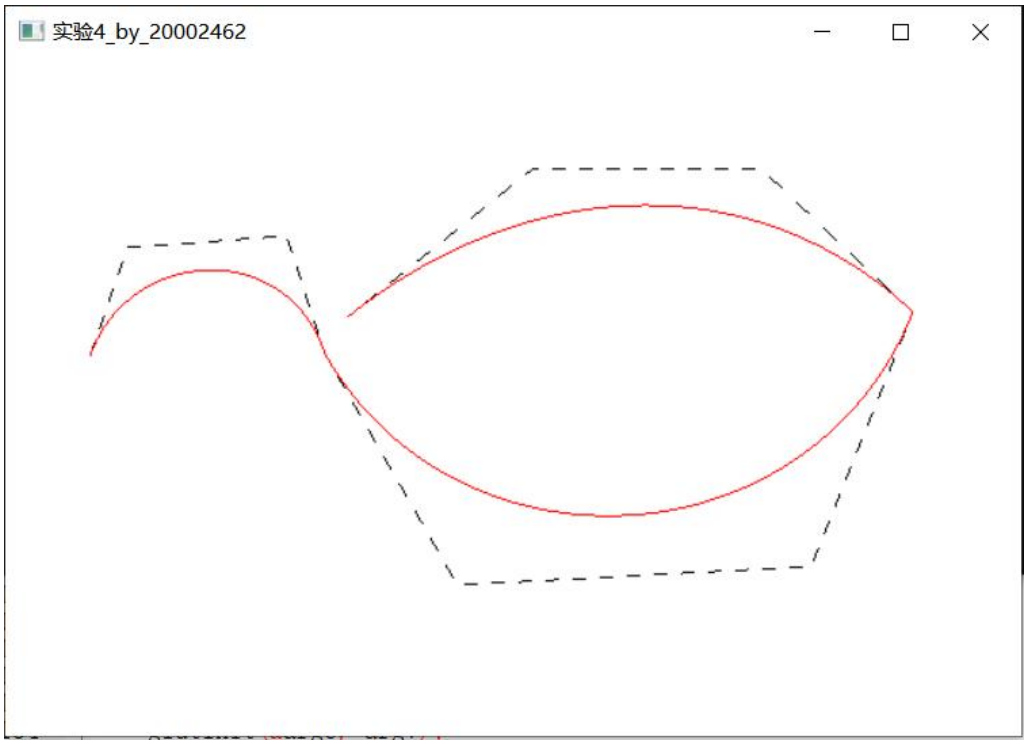
（1）n=4，1 段 3 次 Bezier 曲线：



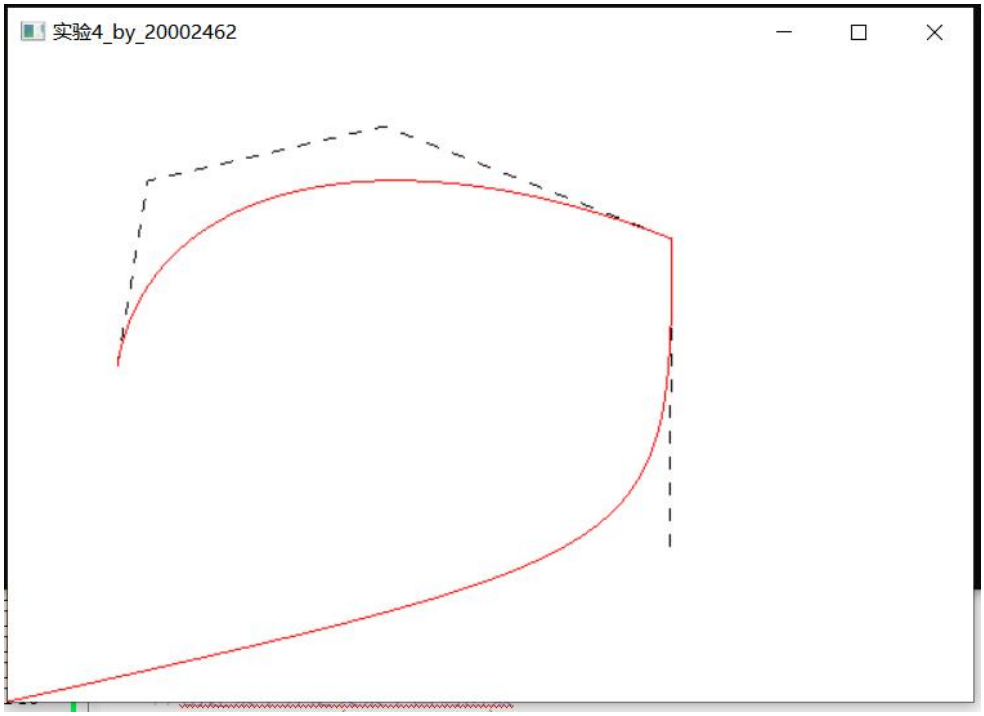
（2）n=7，2 段 3 次 Bezier 曲线：



（3）n=10，3 段 3 次 Bezier 曲线：



（4）如果 $n \neq 3m+1$，则绘制的 Bezier 曲线最后会与（0,0）点相连：



五．思考题

1．Bezier 曲线好不好用（优缺点）？

答：**（1）Bezier 曲线的优点：**

Bezier 曲线是面向几何而不是面向代数的，它以直观地交互式方式使人对设计对象的控制达到直接的几何化程度，这就使用户可以像美术家、式样设计师或设计工程师一样，能够利用易于控制的输入参数，来改变曲线的阶次和形状，直到输出与所预期的形状相符为止。

Bezier 曲线的形状趋向于控制多边形的形状，所以改变多边形的顶点就会改变曲线的形状，这就使观察者对输入、输出关系有直观的感觉。若要增加某一曲线段的阶次，仅需指定另一个中间顶点，大大地提高了灵活性。

Bezier 曲线各点均落在控制多边形各顶点构成的凸包中，这样的凸包性保证了曲线随控制点平稳前进而不会振荡。

Bezier 曲线具有几何不变形，它的形状仅与控制多边形各顶点的相对位置有关，与坐标系的选择无关。

Bezier 曲线具有变差减少性，如果控制多边形是一个平面图形，则该平面内的任意直线与该 Bezier 曲线的交点个数不多于该直线与控制多边形的交点个数；如果控制多边形不是平面图形，则任意平面与 Bezier 曲线的交点个数不会超过它与控制多边形的交点个数。它的变差减少性反映了 Bezier 曲线比控制多边形波动得少，即比控制多边形更加的光滑。

（2）Bezier 曲线的缺点：

一是控制多边形的顶点个数决定了 Bezier 曲线的阶数，即 n+1 个顶点的控制多边形必然产生 n 次 Bezier 曲线，而且当 n 较大的时候，控制多边形对曲线的控制将会减弱。通常这个缺点可以通过多段低阶 Bezier 曲线逼近的方法解决。

二是 Bezier 曲线不能进行局部修改，任何一个控制点的位置的变化对整条曲线都有影响。

附件 1

① glLineStipple(1,0x00FF);　　//定义线型模式
② glDisable(GL_LINE_STIPPLE);　//仅当橡皮筋时激活虚线
③ glDisable(GL_LINE_STIPPLE);　//仅当橡皮筋时激活虚线

附 2：向量（Vector）

https://www.runoob.com/w3cnote/cpp-vector-container-analysis.html

Vector 是一个封装了动态大小数组的顺序容器（Sequence Container），能够存放各种类型的对象。可以简单的认为，向量是一个能够存放任意类型的动态数组。

```
#include <vector>
using namespace std;
```

1.构造函数

①vector():创建一个空 vector
②vector(int nSize):创建一个 vector,元素个数为 nSize
③vector(int nSize,const t& t):创建一个 vector，元素个数为 nSize,且值均为 t
④vector(const vector&):复制构造函数
⑤vector(begin,end):复制[begin,end)区间内另一个数组的元素到 vector 中

2.增加函数

①void push_back(const T& x):向量尾部增加一个元素 X
②iterator insert(iterator it,const T& x):向量中迭代器指向元素前增加一个元素 x
③iterator insert(iterator it,int n,const T& x):向量中迭代器指向元素前增加 n 个相同的元素 x
④iterator insert(iterator it,const_iterator first,const_iterator last):向量中迭代器指向元素前插入另一个相同类型向量的[first,last)间的数据

3.删除函数

①iterator erase(iterator it):删除向量中迭代器指向元素
②iterator erase(iterator first,iterator last):删除向量中[first,last)中元素
③void pop_back():删除向量中最后一个元素
④void clear():清空向量中所有元素

4.遍历函数

①reference at(int pos):返回 pos 位置元素的引用
②reference front():返回首元素的引用
③reference back():返回尾元素的引用
④iterator begin():返回向量头指针，指向第一个元素
⑤iterator end():返回向量尾指针，指向向量最后一个元素的下一个位置
⑥reverse_iterator rbegin():反向迭代器，指向最后一个元素
⑦reverse_iterator rend():反向迭代器，指向第一个元素之前的位置

5.判断函数

①bool empty() const:判断向量是否为空，若为空，则向量中无元素

6.大小函数

①int size() const:返回向量中元素的个数
②int capacity() const:返回当前向量所能容纳的最大元素值
③int max_size() const:返回最大可允许的 vector 元素数量值

7.其他函数

①void swap(vector&):交换两个同类型向量的数据

②void assign(int n,const T& x):设置向量中第 n 个元素的值为 x
③void assign(const_iterator first,const_iterator last):向量中[first,last)中元素设置成当前向量元素