计算机学院实验报告

实验题目：实验五：Bézier曲线与B样条		学号：202000120101
日期：2022.11.14	班级：20.2	姓名：尹国泰
Email：1018693208@qq.com		

实验目的：

掌握Bézier曲线与B样条的原理与基本生成过程

实现de Casteljau 算法来绘制使用不同数量的控制点表示Bézier 曲线

基于de boor 割角算法来绘制使用不同数量的控制点表示B样条曲线

支持insert/delete/move控制点，同时画出控制顶点/控制多边形/样条曲线。

实验环境介绍：

操作系统：Window10

编译器环境：MinGW，VSCode

OpenGL 环境：freeglut

解决问题的主要思路：

1. 根据下面的递推公式实现deCasteljau 算法来绘制Bézier 曲线

$$P_i^k = \begin{cases} P_i & k = 0 \\ (1-t)P_i^{k-1} + tP_{i+1}^{k-1} & k = 1,2,\ldots,n, i = 0,1,\ldots,n-k \end{cases}$$

得到的P[n][0]为最终的要绘制的Bézier 曲线

2. 实现如下deboor 割角算法来绘制B样条曲线，我实现的是t的区间均匀分布的均匀B样条曲线，同时实现了将两端点各重复K次的准均匀B样条曲线

先将t固定在区间$[t_j, t_{j+1})(k-1 \le j \le n)$上，

$$\mathbf{P}_i^{(r)}(t) = \begin{cases} \left(1-\tau_i^j\right)\mathbf{P}_{i-1}^{(r-1)}(t) + \tau_i^j\mathbf{P}_i^{(r-1)}(t) & \text{if } r = 1,2,\ldots,k-1 \\ \mathbf{P}_i & \text{if } r = 0 \end{cases}$$

$$\tau_i^r = \frac{t-t_i}{t_{i+k-r}-t_i} \qquad P(t) = P_j^{[k-1]}(t)$$

其中 ，最终得到

3. 要求支持插入、删除、移动点的操作

对于Bézier 曲线，每次插入、删除、移动点之后都要重新用de Casteljau 算法来绘制曲线

对于B样条曲线，由于我让t的区间是均匀划分的，每次插入、删除所有t区间都要重新划分，要重新运行一次deBoor算法来绘制曲线。对于移动操作，t区间并不需要重新划分，移动控制点后只需要改变后面K个区间的曲线即可，具体来说移动第i个控制点，只需要重新画[t(i),t(i+k))这一段曲线。

实验步骤：

1. 实现 deCasteljau 算法来绘制 Bézier 曲线，point_to_draw 用来临时存储曲线上要绘制的点

```
void deCasteljau(){
    point_to_draw.clear();
    int n = vertex.size() - 1;
    vector<point> p(vertex.size());

    for(float t = 0; t <= 1;t += 0.001){
        //k=0
        auto iver = vertex.begin();
        for(int i = 0; i <= n ; i++){
            p[i] = *iver;
            iver++;
        }
        //k>0
        for(int k = 1; k <= n; k++){
            for(int i = 0; i <= n-k; i++){
                p[i].x = (1-t)*p[i].x + t*p[i+1].x;
                p[i].y = (1-t)*p[i].y + t*p[i+1].y;

            }
        }
        point_to_draw.push_back(p[0]);
    }
    drawAll();
}
```

2. 实现 deBoor 算法，下面用来画[t_j ， t_{j+1})这一段曲线上的点，其中 use_option 为 2 画均匀 B 样条曲线，为 3 画准均匀 B 样条曲线

```
void deBoor_draw(int k,int j){ //绘制[ t_j , t_{j+1} )这一段曲线上的点
    vector<point> p(vertex.size() + 2*(k-1));
    auto stiver = vertex.begin();
    if(use_option=='3') for(int i = k-1 + (k-1); i < j; i++) stiver++;//前 k-1 个点是额外的 0 号点
    else if(use_option=='2') for(int i = k-1; i < j; i++) stiver++;
    for(float pt = t[j]; pt < t[j+1];pt += 0.001){
        //r==0
        auto iver = stiver;
        for(int i=j-k+1;i<=j;i++){
            p[i] = *iver;
```

```
            if(use_option=='3'){
                if(i >= k-1) iver++; //前 k-1 个点都用 vertex 中的 0 号点
                if(iver==vertex.end()) iver--; //后 k-1 个点都用 vertex
中的最后一个点
            }else if(use_option=='2'){
                iver++;
            }
        }
        //r>0
        for(int r=1;r<=k-1;r++){
            for(int i=j;i>=j-k+r+1;i--){
                p[i].x =(pt - t[i])/(t[i+k-r] - t[i]) * p[i].x
                        + (t[i+k-r] - pt)/(t[i+k-r] - t[i]) *
p[i-1].x;
                p[i].y =(pt - t[i])/(t[i+k-r] - t[i]) * p[i].y
                        + (t[i+k-r] - pt)/(t[i+k-r] - t[i]) *
p[i-1].y;
            }
        }
        point_to_draw[t_index[pt]] = p[j];
    }
}
```

第一次画 B 样条曲线时将[t_{k-1} , t_{n+1})这一段区间上的曲线全部画出

```
for(int j=k-1;j<=n;j++){
    deBoor_draw(k,j);
}
```

3. 实现插入、删除、移动控制点操作，在移动 B 样条曲线的控制点时，只需要移动一部分（t 区间的其中 K 段），这一部分的具体代码如下。其他诸如对于 Bézier 曲线控制点的插入删除移动、对 B 样条曲线控制点的插入删除，都需要重新绘制整条曲线。

```
void chg_deBoor(int chg_point){
    int st,ed;
    if(use_option=='2'){
        st=max(chg_point,K_deboor-1);
        ed=min(chg_point+K_deboor-1,(int)vertex.size()-1);
    }else if(use_option=='3'){
        st=chg_point + (K_deboor-1);//前面加了 k-1 个点
        ed=chg_point+K_deboor-1 + (K_deboor-1); //后面也加了 k-1 个
点,ed 最多可以到 n+2(k-1)
    }
    for(int j=st; j<=ed; j++){
```

```
        deBoor_draw(K_deboor,j);
    }
    drawAll();
}
```

4. 其他的一些键鼠操作实现

我实现了下面这样一个函数来获得鼠标左键选取的曲线控制点，返回该控制点的迭代器

```
list<point>::iterator getpoint(float x,float y){
    float accuracy = 5;
    for(auto iver = vertex.begin();iver!=vertex.end();iver++){
        if((abs(iver->x - x ) < accuracy) && (abs(iver->y - y ) < accuracy)){
            return iver;
        }
    }
    return vertex.end();
}
```

我用 act_option 来标识当前要进行 insert、delete、move 操作中的哪一种，分别按下键盘上的 i，d，m 来改变模式，默认为 insert。

用 use_option 来标识当前要绘制哪一种曲线，其中 1 为 Bézier 曲线，2 为均匀 B 样条曲线、3 为准均匀 B 样条曲线

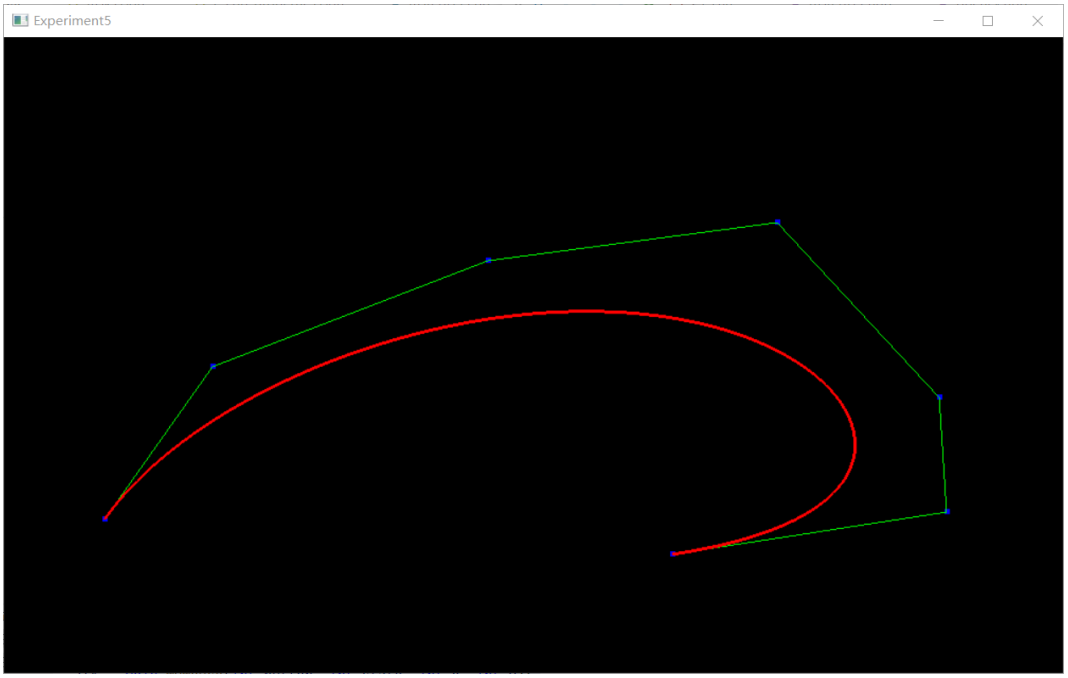
```
void myKeyboard(unsigned char key, int x, int y){
    if (key == '1'||key == '2'|| key=='3'){
        use_option = key;
        if(use_option=='1') deCasteljau();
        else deBoor_init(K_deboor);
    }
    if(key == 'i'){
        act_option = 'i';
    }
    if(key == 'd'){
        act_option = 'd';
    }
    if(key == 'm'){
        act_option = 'm';
    }
}
```

当按下鼠标左键，会调用前面的 getpoint 函数，获取选取的曲线控制点
当松开鼠标左键时,会根据 act_option 和 use_option 来修改曲线,act_option 标识 insert、delete、move 三种操作，use_option 标识哪一种曲线

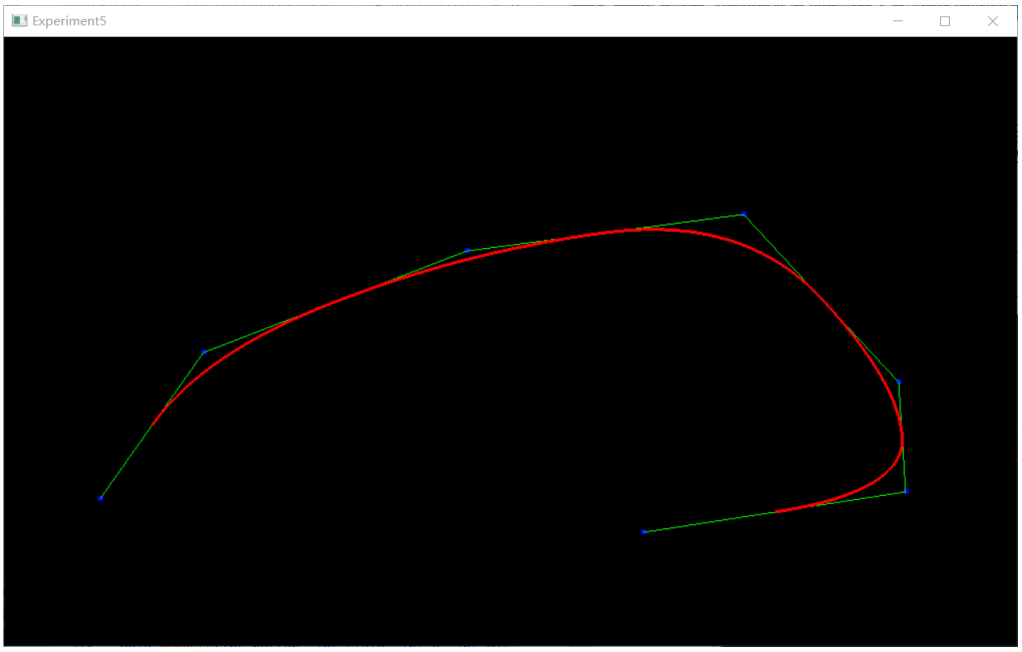
```
void mymouse(int button, int state, int x, int y){
    y = h - y;
    if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN){
        chs_ver = getpoint(x,y);
    }
    if (button == GLUT_LEFT_BUTTON && state == GLUT_UP){
        if(use_option == '1') act_deCasteljau(x,y);
        else act_deBoor(x,y);
    }
}
```

对于 delete 操作，会直接删除按下鼠标左键时选取的控制点
对于 move 操作，会将按下鼠标左键时选取的控制点，移动到松开左键时的位置
对于 insert 操作，会在按下鼠标左键时选取的控制点后面，插入新的控制点，新控制点位置为松开左键时的位置，倘若按下左键没有选到控制点，则会在最后面插入新的控制点

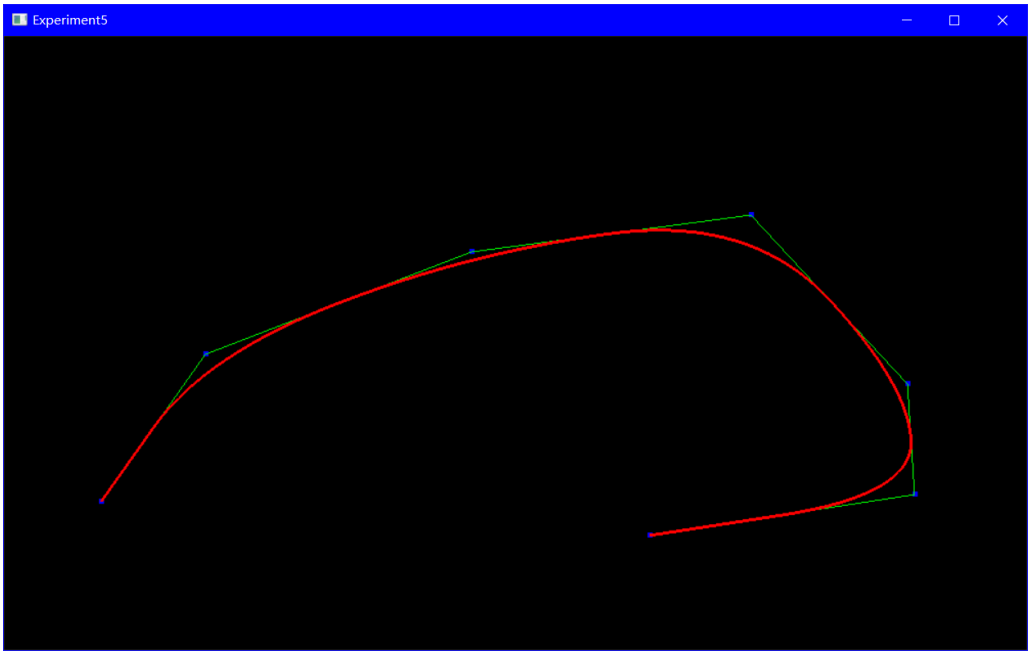
实验结果展示及分析：
1. 程序运行，默认是在执行 insert 操作，下面是同样的控制点下，绘制的 Bézier 曲线，均匀 B 样条曲线，准均匀 B 样条曲线
（1）Bézier 曲线



（2）均匀 B 样条曲线

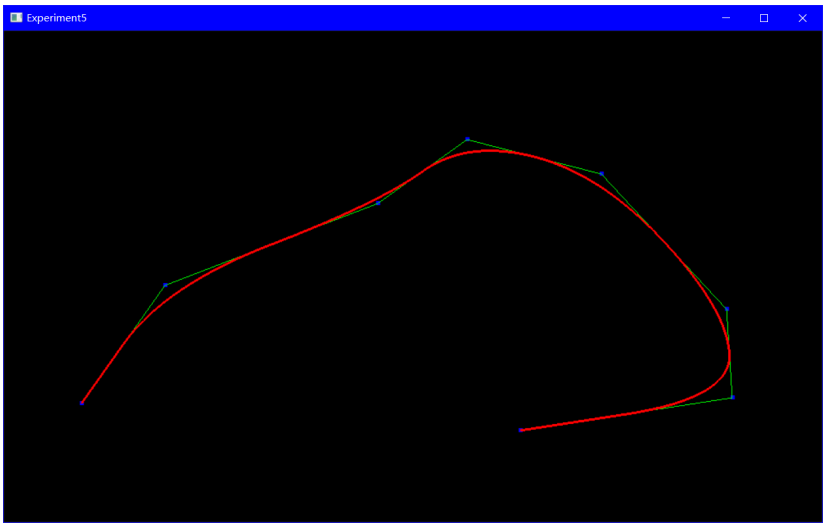


（3）准均匀 B 样条曲线

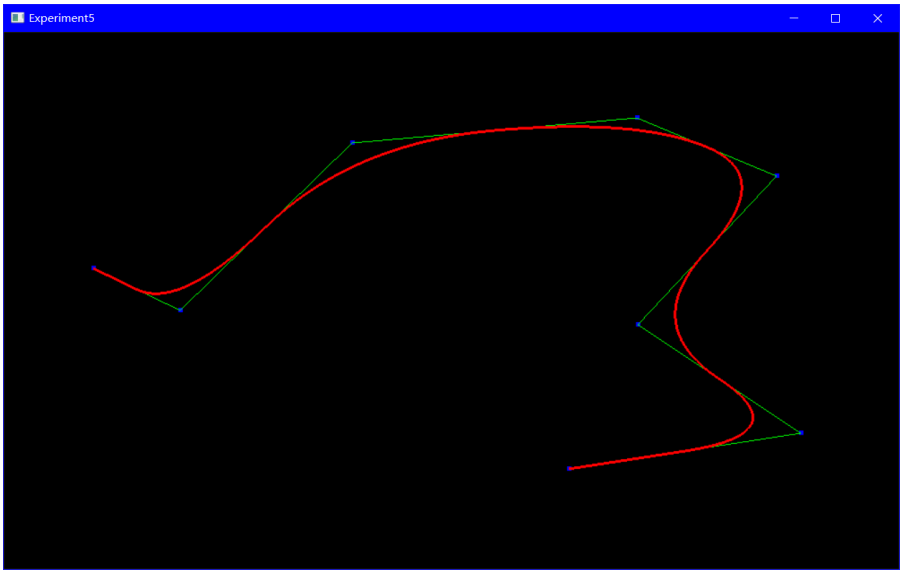


2. 下面以准均匀 B 样条曲线为例，测试 insert、delete、move 操作

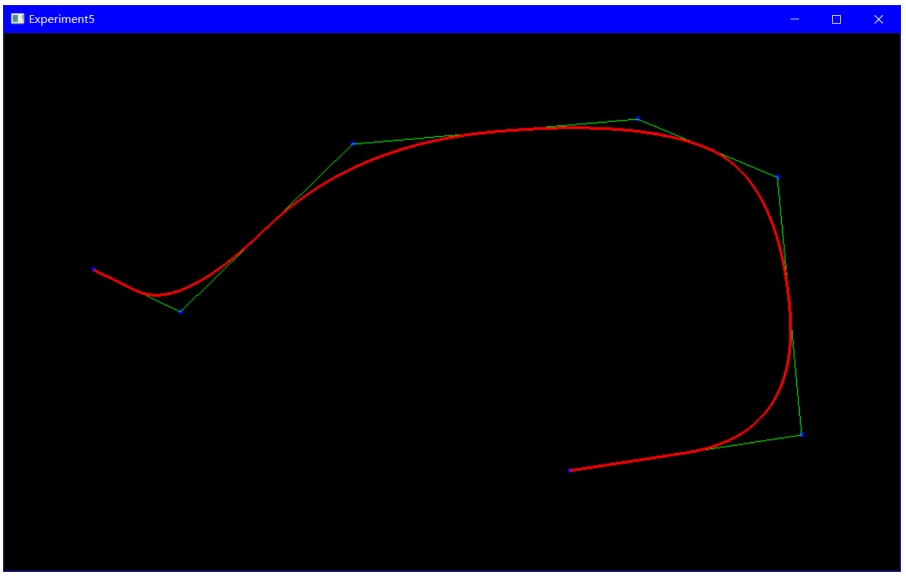
（1）在第 3 个控制点后面 insert 一个新的控制点



（2）move 控制点



（3）删除倒数第 3 个控制点



实验中存在的问题及解决：

1. 在实现 deCasteljau 和 deboor 算法时，递推公式中每一项都是二维的，倘若要创建一个二维的数组来实现，空间开销过大，能不能用滚动数组的方法优化为一维的数组？

这是可以的，就以 deboor 算法为例，注释的部分为我最开始写的二维数组的形式，后来用滚动数组的思想优化为了一维数组的形式，要注意内层循环必须从后向前遍历。

这是因为递推公式整体上是这样子的形式 p[r][i]=p[r-1][i]+p[r-1][i-1]，计算第二维的第 i 个数时，只会用到 i 之前的那些数，就可以去掉第一位，然后内层循环必须从后向前遍历，在第 r 次循环时算 p[i]时用到的 p[i]和 p[i-1]仍然是 r-1 次循环时求得的数值

```
for(int r=1;r<=k-1;r++){
    //for(int i=j-k+r+1;i<=j;i++){
        // p[r][i].x =(pt - t[i])/(t[i+k-r] - t[i]) * p[r-1][i].x
        //         + (t[i+k-r] - pt)/(t[i+k-r] - t[i]) * p[r-1][i-1].x;
        // p[r][i].y =(pt - t[i])/(t[i+k-r] - t[i]) * p[r-1][i].y
        //         + (t[i+k-r] - pt)/(t[i+k-r] - t[i]) * p[r-1][i-1].y;
    //}
    for(int i=j;i>=j-k+r+1;i--){
        p[i].x =(pt - t[i])/(t[i+k-r] - t[i]) * p[i].x
                + (t[i+k-r] - pt)/(t[i+k-r] - t[i]) * p[i-1].x;
        p[i].y =(pt - t[i])/(t[i+k-r] - t[i]) * p[i].y
                + (t[i+k-r] - pt)/(t[i+k-r] - t[i]) * p[i-1].y;
    }
}
//point_to_draw.push_back(p[k-1][j])
point_to_draw.push_back(p[j]);
```

2. 我在画曲线时以 0.001 为步长，也就是画了曲线上的 1001 个点，当曲线很长时，仍会出现断断续续的情况。

不再使用 `glBegin(GL_POINTS);` 而是用 `glBegin(GL_LINE_STRIP);` 去画这些连续的点，就会在这些连续的点两两之间画一条直线，也就是用 1000 段直线段来画这一条曲线，就不会出现断断续续的情况了

附件：代码

```
#include<iostream>
#include <GL/glut.h>
#include<algorithm>
#include<vector>
#include<stack>
#include<queue>
#include<list>
#include<map>
#include<Windows.h>
#include<cmath>
#include<string.h>
using namespace std;

const int w=1000,h=600;
const int K_deboor = 3;
const int MAXPOINTS =1000;
struct point{
    float x, y;
    point(){}
    point(float _x, float _y)
        :x(_x), y(_y) {}
    point(int _x, int _y)
        :x(_x), y(_y) {}
};
list<point> vertex;
vector<point> point_to_draw;
map<float,int> t_index;
void drawAll(){
    glClear(GL_COLOR_BUFFER_BIT);

    //drawControlPoint
    glColor3f(0, 0, 1);
    glPointSize(5);
    glBegin(GL_POINTS);
    for(auto i : vertex){
        glVertex2f(i.x, i.y);
    }
    glEnd();

    //drawControlPolygon
    glColor3f(0, 1, 0);
    glLineWidth(1);
    glBegin(GL_LINE_STRIP);
```

```
    for(auto i : vertex){
        glVertex2f(i.x, i.y);
    }
    glEnd();

    //drawPointToDraw
    glColor3f(1, 0, 0);
    glLineWidth(3);
    glBegin(GL_LINE_STRIP);
    for(auto i : point_to_draw){
        glVertex2f(i.x, i.y);
    }
    glEnd();
    glFlush();
}
void deCasteljau(){
    point_to_draw.clear();
    int n = vertex.size() - 1;
    vector<point> p(vertex.size());

    for(float t = 0; t <= 1;t += 0.001){
        //k=0
        auto iver = vertex.begin();
        for(int i = 0; i <= n ; i++){
            p[i] = *iver;
            iver++;
        }
        //k>0
        for(int k = 1; k <= n; k++){
            for(int i = 0; i <= n-k; i++){
                p[i].x = (1-t)*p[i].x + t*p[i+1].x;
                p[i].y = (1-t)*p[i].y + t*p[i+1].y;

            }
        }
        point_to_draw.push_back(p[0]);
    }
    drawAll();
}

char use_option;
//use_option='1' Bezier 曲线
//use_option='2' 均匀 B 样条曲线
//use_option='3' 准均匀 B 样条曲线
```

```
vector<float> t;
void deBoor_draw(int k,int j); //绘制[ t_j , t_{j+1} )这一段曲线上的点
void set_t(int k,int n){
    t.clear();
    float step=1.0/(n+k),nowt=0;
    t.push_back(0);
    for(int i=1;i<=n+k;i++){
        nowt+=step;
        t.push_back(nowt);
    }

    point_to_draw.clear();
    t_index.clear();
    for(int j=k-1;j<=n;j++){
        for(float pt = t[j]; pt < t[j+1];pt += 0.001){
            point_to_draw.push_back(point(0,0));
            t_index[pt] = point_to_draw.size()-1;
        }
    }
}
void deBoor_init(int k){

    int n = vertex.size() - 1;
    if(use_option == '3') n = n + 2*(k-1);//开头末尾各多了 k-1 个元素
    set_t(k,n);

    for(int j=k-1;j<=n;j++){
        deBoor_draw(k,j);
    }
    drawAll();
}

void deBoor_draw(int k,int j){ //绘制[ t_j , t_{j+1} )这一段曲线上的点
    vector<point> p(vertex.size() + 2*(k-1));
    auto stiver = vertex.begin();
    if(use_option=='3') for(int i = k-1 + (k-1); i < j; i++)
stiver++;//前 k-1 个点是额外的 0 号点
    else if(use_option=='2') for(int i = k-1; i < j; i++) stiver++;
    for(float pt = t[j]; pt < t[j+1];pt += 0.001){
        //r==0
        auto iver = stiver;
        for(int i=j-k+1;i<=j;i++){
            p[i] = *iver;
            if(use_option=='3'){
```

```
                if(i >= k-1) iver++; //前 k-1 个点都用 vertex 中的 0 号点
                if(iver==vertex.end()) iver--; //后 k-1 个点都用 vertex
中的最后一个点
            }else if(use_option=='2'){
                iver++;
            }
        }
        //r>0
        for(int r=1;r<=k-1;r++){
            for(int i=j;i>=j-k+r+1;i--){
                p[i].x =(pt - t[i])/(t[i+k-r] - t[i]) * p[i].x
                           + (t[i+k-r] - pt)/(t[i+k-r] - t[i]) * p[i-
1].x;
                p[i].y =(pt - t[i])/(t[i+k-r] - t[i]) * p[i].y
                           + (t[i+k-r] - pt)/(t[i+k-r] - t[i]) * p[i-
1].y;
            }
        }
        point_to_draw[t_index[pt]] = p[j];
    }
}

char act_option;
list<point>::iterator chs_ver;
list<point>::iterator getpoint(float x,float y){
    float accuracy = 5;
    for(auto iver = vertex.begin();iver!=vertex.end();iver++){
        if((abs(iver->x - x ) < accuracy) && (abs(iver->y - y ) <
accuracy)){
            return iver;
        }
    }
    return vertex.end();
}
void act_deCasteljau(int x,int y){
    if(act_option == 'i'){
        if(chs_ver != vertex.end()) chs_ver++;
        vertex.insert(chs_ver , point(x, y));
    }
    if(act_option == 'd'){
        if(chs_ver == vertex.end()) return;
        vertex.erase(chs_ver);
    }
    if(act_option == 'm'){
```

```
        if(chs_ver == vertex.end()) return;
        chs_ver = vertex.erase(chs_ver);
        vertex.insert(chs_ver , point(x, y));
    }
    deCasteljau();
}
void chg_deBoor(int chg_point){
    int st,ed;
    if(use_option=='2'){
        st=max(chg_point,K_deboor-1);
        ed=min(chg_point+K_deboor-1,(int)vertex.size()-1);
    }else if(use_option=='3'){
        st=chg_point + (K_deboor-1);//前面加了 k-1 个点
        ed=chg_point+K_deboor-1 + (K_deboor-1); //后面也加了 k-1 个点,ed
最多可以到 n+2(k-1)
    }
    for(int j=st; j<=ed; j++){
        deBoor_draw(K_deboor,j);
    }
    drawAll();
}
void act_deBoor(int x,int y){
    int chg_point = 0;
    for(auto i =vertex.begin();i!=chs_ver;i++) chg_point++;
    if(act_option == 'i'){
        if(chs_ver != vertex.end()) chs_ver++,chg_point++;
        chs_ver = vertex.insert(chs_ver,point(x,y));

        deBoor_init(K_deboor);
    }
    if(act_option == 'd'){
        if(chs_ver == vertex.end()) return;
        chs_ver = vertex.erase(chs_ver);

        deBoor_init(K_deboor);
    }
    if(act_option == 'm'){
        if(chs_ver == vertex.end()) return;
        chs_ver = vertex.erase(chs_ver);
        chs_ver = vertex.insert(chs_ver , point(x, y));

        chg_deBoor(chg_point);
    }
}
```

```
void mymouse(int button, int state, int x, int y){
    y = h - y;
    if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN){
        chs_ver = getpoint(x,y);
    }
    if (button == GLUT_LEFT_BUTTON && state == GLUT_UP){
        if(use_option == '1') act_deCasteljau(x,y);
        else act_deBoor(x,y);
    }
}
void myKeyboard(unsigned char key, int x, int y){
    if (key == '1'||key == '2'|| key=='3'){
        use_option = key;
        if(use_option=='1') deCasteljau();
        else deBoor_init(K_deboor);
    }
    if(key == 'i'){
        act_option = 'i';
    }
    if(key == 'd'){
        act_option = 'd';
    }
    if(key == 'm'){
        act_option = 'm';
    }
}

void display(){}

void Init(){
    //设置颜色
    glClearColor(0.0, 0.0, 0.0, 0.0);
    //颜色过渡形式
    glShadeModel(GL_SMOOTH);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0.0, (GLdouble)w, 0.0, (GLdouble)h);

}
int main(int argc, char** argv) {
    act_option = 'i';
    use_option = '1';

    glutInit(&argc, argv);
```

```
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);

    //设置初始窗口的位置
    glutInitWindowPosition(200, 50);
    //设置初始窗口的大小
    glutInitWindowSize(w, h);
    //根据前面设置建立窗口，参数设置为变体
    glutCreateWindow("Experiment5");
    Init();
    //绘图时被调用的函数
    glutDisplayFunc(display);
    glutMouseFunc(mymouse);
    glutKeyboardFunc(myKeyboard);

    //进行消息循环，用于显示窗体，窗体关闭后自动退出循环
    glutMainLoop();
    return 0;
}
```