3+32222222

计算机学院实验报告

实验题目： Bézier 曲线与 B 样条		学号：202000130143
日期：11.28	班级：计科 20.1	姓名：郑凯饶
Email：1076802156@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：

Dell Latitude 5411

Intel(R) Core(TM) i5-10400H CPU @ 2.60GHz(8GPUs),~2.6GHz

Windows 10 家庭中文版 64 位（10.0，版本 18363）

Visual Studio 2022

解决问题的主要思路：

① Bezier 曲线：

根据 Bezier 曲线的定义，给定空间中$(\mathrm{n}+1)$个点的位置矢量$\mathrm{P_i}(i=0,1,2,\ldots,n)$，则 Bezier 曲线段的参数方程为：

$$\mathrm{p(t)}=\sum_{\mathrm{i=0}}^{\mathrm{n}} P_i B_{i,n}(t),\mathrm{t}\in[0,1]$$

其中，$B_{i,n}(t)=C_n^i t^i(1-t)^{n-i}=\frac{n!}{i!(n-i)!}t^i(1-t)^{n-i}\quad(i=0,1,\ldots,n)$ 为伯恩斯坦基函数，被证明与 Bezier 曲线的基函数等价。

而前者具有递推性质：$B_{i,n}(t)=(1-t)B_{i,n-1}(t)+tB_{i-1,n-1}(t)\quad(i=0,1,\cdots n)$，可由组合数关系 $C_n^i=(C_{n-1}^i+C_{n-1}^{i-1})$ 推出。

基函数的递推性质使得不同阶数的 Bezier 曲线上的点之间也存在递推关系，进而整条曲线可通过递推计算：

由(n+1)个控制点P_i(i=0,1,...,n)定义的n次Bezier曲线P_0^n可被定义为分别由前、后n个控制点定义的两条(n-1)次Bezier曲线P_0^{n-1}与P_1^{n-1}的线性组合：

$$P_0^n = (1-t)P_0^{n-1} + tP_1^{n-1} \qquad t \in [0,1]$$

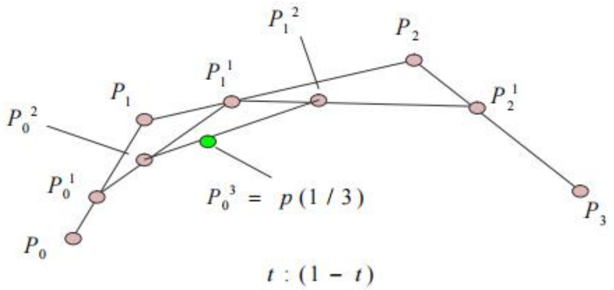
由此得到Bezier曲线的递推计算公式：

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

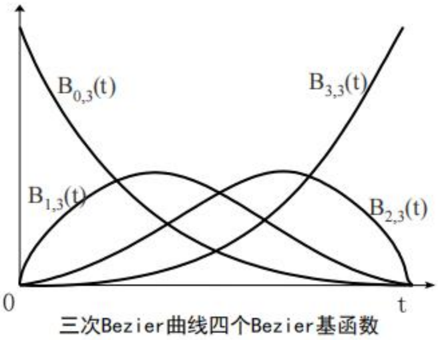
这便是著名的de Casteljau算法。用这一递推公式，在给定参数下，求Bezier曲线上一点P(t)非常有效

这里的符号定义P_i^k意义为编号位于区间$[i, i+k-1]$的 k 个控制点定义 Bezier 曲线，特别地 $k=0$ 则指代编号为 i 控制点。

编程实现 de Casteljau 算法时，以一定步长$\Delta t = 5e-5$遍历区间$[0,1]$，从**<u>所有</u>**控制点出发计算 Bezier 曲线上对应的一点，多个点连成曲线。



② B 样条曲线：
Bezier 曲线的不足：
（1）一旦确定了控制顶点数（n+1），决定了曲线的阶次（n）；
（2）拼接比较复杂；
（3）无法作局部修改。
第（3）点是因为 Bernstein 多项式在整个区间[0,1]上都有支撑，即函数值不为0，所有每个控制项点对整条曲线都有影响。



三次Bezier曲线四个Bezier基函数

因此，Gordon 等人提出了 B 样条方法，样条（spline）是分段连续多项式。
B 样条曲线的数学表达式为：

$$P(u) = \sum_{i=0}^{n} P_i B_{i,k}(u) \qquad u \in [u_{k-1}, u_{n+1}]$$

$P_i(i = 0,1,\cdots,n)$ 是控制多边形的顶点

只是缩小了各个控制点的支撑区间，改变了基函数的定义域，并不改变它的递推性质，因此有 de Boor-Cox 递推定义：

$$B_{i,1}(u) = \begin{cases} 1 & u_i < u < u_{i+1} \\ 0 & Otherwise \end{cases}$$

这里应取等

并约定：$\frac{0}{0} = 0$

$$B_{i,k}(u) = \frac{u - u_i}{u_{i+k-1} - u_i} B_{i,k-1}(u) + \frac{u_{i+k} - u}{u_{i+k} - u_{i+1}} B_{i+1,k-1}(u)$$

当 $k = 2$ 时，有

$$B_{i,2}(u) = \begin{cases} \dfrac{u - u_i}{u_{i+1} - u_i} & u_i \le u \le u_{i+1} \\ \dfrac{u_{i+2} - u}{u_{i+2} - u_{i+1}} & u_{i+1} \le u \le u_{i+2} \\ 0 & \text{其它} \end{cases}$$

可以观察到 B 样条基函数的$B_{i,k}(u)$支撑区间为$[u_i, u_{i+k}]$，即在该区间上函数值不为 0，对应系数分母部分的含义。

同样 B 样条曲线也具有递推性质。

编程实现 de Boor 算法，先将 t 固定在区间$[u_j, u_{j+1}]$，以一定步长$\Delta t = 5e-5$ 遍历该区间，从<u>**对该区间曲线有影响的（支撑区间和该区间有交集）**</u>控制点出发，事实上这些点的编号区间为$[i-k+1, i]$共 k 个点，计算 B 样条曲线上对应的一点，多个点连成曲线。

实验步骤：

1. 实现基础交互逻辑，支持 insert/delete/move 控制点，通过编写 glutKeyboardFunc 回调函数实现功能之间的切换，编写 glutMouseFunc 回调函数实现选中点，并执行当前模式下的对应功能，glutMotionFunc 在按下鼠标并移动时生效，对应实现 move 功能。

2. 实现 de Casteljau 算法绘制 Bezier 曲线：

```
void de_Casteljau() {
    bezier_point.clear();
    std::vector<point> tmp;

    tmp = ctrl_point;
    int n = tmp.size();

    for (double t = 0; t <= 1; t += (delta_t / n)) {
        tmp = ctrl_point;
        for (int i = 1; i < n; i++) {  // 轮数 i
            for (int j = 0; j < n - i; j++) {  // 第 i 轮控制点数为(n-i+1)
                tmp[j].x = (1 - t) * tmp[j].x + t * tmp[j + 1].x;
                tmp[j].y = (1 - t) * tmp[j].y + t * tmp[j + 1].y;
            }
        }
        bezier_point.push_back(tmp[0]);
    }

    draw_line(bezier_point, b_color);
}
```

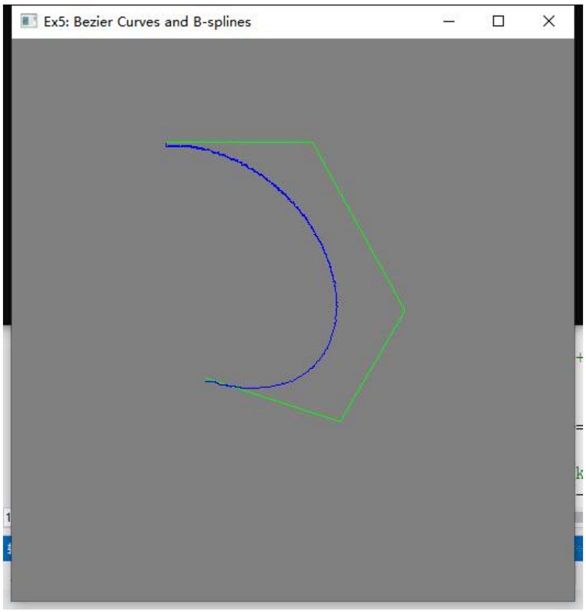
3. 实现 de Boor 算法绘制 B 样条曲线：

```
void de_Boor() {
    bspline_point.clear();
    std::vector<point> tmp;

    tmp = ctrl_point;
    int n = tmp.size() - 1;     // 这里定义 n 为控制点最大编号

    std::vector<double> c(1e5);     // 节点矢量
    // 非递减参数序列：t0,t1,...,t(n+k)

    // 准均匀 B 样条
    for (int i = 0; i < k; i++) c[i] = 0;
    for (int i = k; i <= n; i++) c[i] = c[i - 1] + 1.0 / (n - k + 2);
```

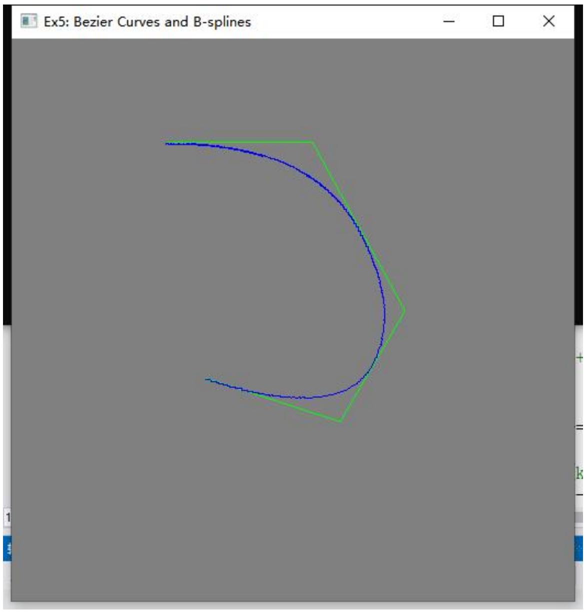
```
    for (int i = n + 1; i <= n + k; i++) c[i] = 1;

    // 均匀 B 样条
    //for (int i = 0; i <= n + k; i++) c[i] = i + 1;

    for (int j = k - 1; j <= n; j++) {
        for (double t = c[j]; t <= c[j + 1]; t += (delta_t / n)) {    // 将 t 固定在
区间[c[j],c[j+1]]
            tmp = ctrl_point;
            for (int r = 1; r < k; r++) {  // (k-1)层割角
                for (int i = j; i > j - k + r; i--) {   // 控制点区间:[(j-k+1)+r,j]
                    double x1, x2, y;
                    x1 = t - c[i];
                    x2 = c[i + k - r] - t;
                    y = c[i + k - r] - c[i];   // x1/y×Pi + x2/y×P(i+1)

                    double c1, c2;
                    if (abs(x1) < 1e-9 && abs(y) < 1e-9) { c1 = 0; }
                    else { c1 = x1 / y; }
                    if (abs(x2) < 1e-9 && abs(y) < 1e-9) { c2 = 0; }
                    else { c2 = x2 / y; } // 0/0=0

                    tmp[i].x = c1 * tmp[i].x + c2 * tmp[i - 1].x;
                    tmp[i].y = c1 * tmp[i].y + c2 * tmp[i - 1].y;
                }
            }
            bspline_point.push_back(tmp[j]);
        }
    }

    draw_line(bspline_point, b_color);
}
```

实验结果展示及分析：

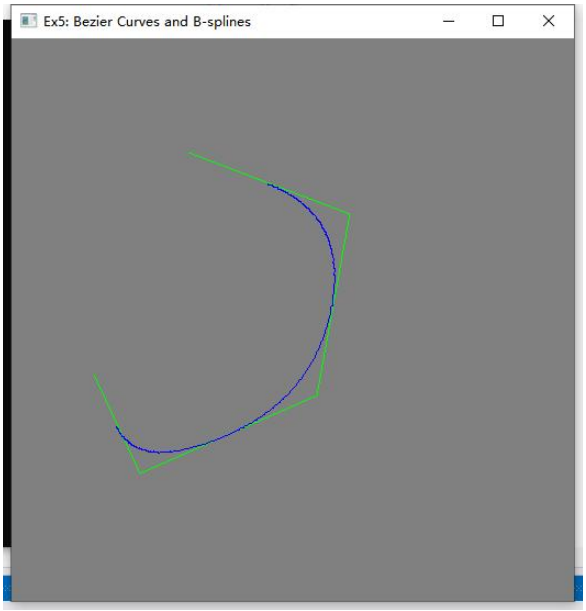
1. Bezier 曲线：



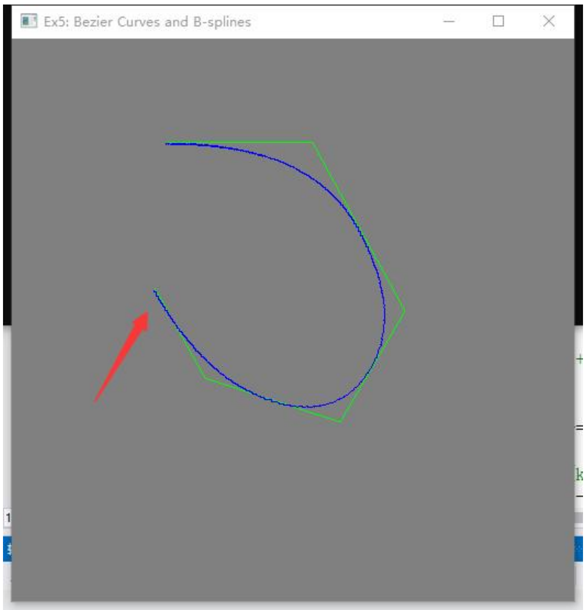
2. 相同控制点集合下绘制出的**准均匀**B 样条：



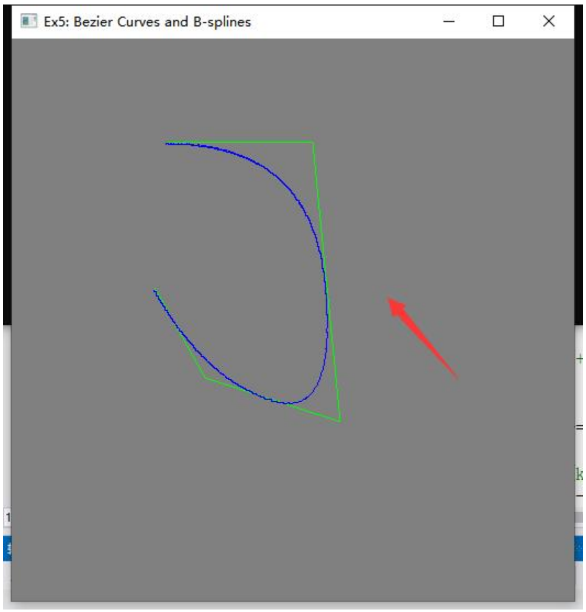
3. **均匀**B 样条：



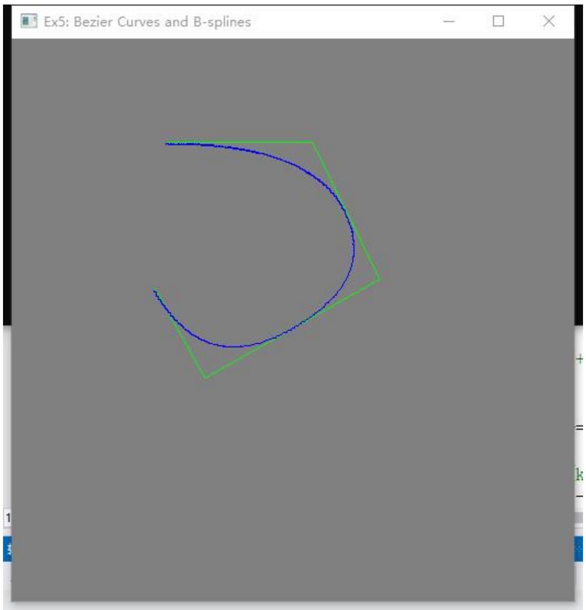
4. Insert 功能，增加点图中箭头指示位置：



5. Delete 功能，删除点图中箭头指示位置：



6. Move 功能，移动点（和上图比较）：



实验中存在的问题及解决：
一开始不能实时渲染，动作存在延迟，后发现所有改变后应调用 glutPostRedisplay()，触发回调函数 glutDisplayFunc，重新渲染，达到实时更新的效果。