A8: Curves & Surfaces

一、概述

1.目标：

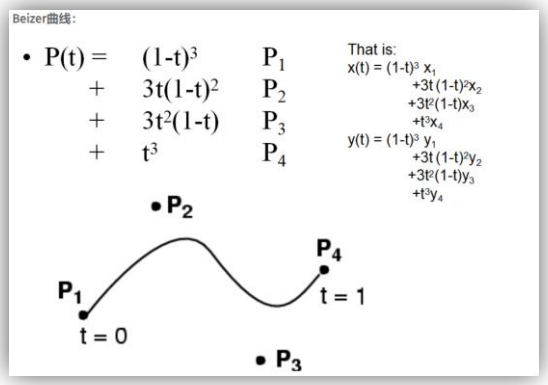
（1）实现绘制 Bezier、BSpline 曲线，并且可以通过鼠标添加、删除、移动点进行交互。
（2）实现 Bezier 和 BSpline 的互相转换，并存成文件。
（3）实现根据曲线生成三角面模型：旋转曲面和参数化曲面。

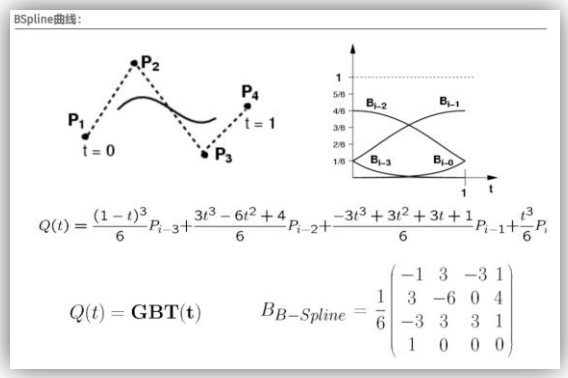
2.知识点介绍：

（1）曲线绘制方法：

Bezier：每四个点产生一条 Bezier 曲线，每个点的位置影响整条曲线。曲线经过 P1 和 P4，且和 P1P2 的连线、P3P4 的连线相切。这也意味着每条曲线的起始点不能是其他 Bezier 曲线的 P2 和 P3 位置的点。所以，7 个顶点会产生 2 条 Bezier 曲线，n 条 Bezier 曲线需要 3n+1 个点。

BSpline：相邻的四个点产生一条 BSpline 曲线，每个点的位置只影响曲线的局部，曲线不经过任何一个点，只是逼近。





设曲线的一般参数方程是：Q(t)=GBT（t）。T 是幂基，即 t 的直到曲线幂次的各次幂组成的向量（对本作业 [t3　t2　t　1]T). Bbezier 和 Bbspline 是样条基矩阵函数，对两种样条曲线都是常数。最后 Gbezier 和 Gbspline 表示控制点的几何位置。每个控制点是 G 矩阵的一列。如果根据曲线的参数方程利用 OpenGL 来画出曲线呢？因为 OpenGL 中只有画线段的函数，所以可以用一系列曲线上的离散点来近似画出曲线。

程序输入 curve_tessellation 值，也就是将该段曲线划分为多少段。假如输入 30，那么把 t 的范围映射到[0,1）范围内，将曲线划分为 30 份，每次迭代，将 t 值加 1/30，就可以得到 30 个曲线上的点。

（2）曲线交互方式：

- 移动点：根据鼠标移动坐标设置该顶点新的位置。

```
vertices[selectedPoint].Set (x, y, vertices[selectedPoint].z ());
```

- 删除点：一条 Bezier 曲线至少由 4 个点组成，小于 5 个点时不处理，否则每次删除三个点。对于 BSpline 曲线，也至少由 4 个点组成，但是大于 4 个点时每次只删除一个点就可以。
- 添加点：对于 Bezier 曲线，每次应该添加三个点。对于 BSpline 曲线，在选中的位置上添加一个点即可。

（3）曲线相互转换：已知参数方程等式中的三个值，可以通过矩阵运算求出另外一个值。

$$Q(t) = G_{bezier}B_{bezier}T = G_{bspline}B_{bspline}T$$

（5）曲线生成三角面模型：

A.旋转曲面：由曲线绕着 Y 轴旋转一周得到。旋转一周细分的次数由新的参数-revolution_tessellation 指定。而曲线的细分参数 curve_tessellation 则可以用来帮助计算模型的面数。具体原理是：若曲线细分参数为 4，意味着 1 个旋转曲线有 4 条边。根据顶点数和曲线的特性可以得到具体的曲线数目，即旋转面的边数。而整个模型的三角面的个数就是旋转面总边数*旋转次数*2。

B.根据给出的细分参数-patch_tessellation（u,v 方向相同）和曲面方程求出每个点的坐标。

$$\vec{p}(u,v)=\sum_{i=0}^{3}\sum_{j=0}^{3}\vec{b}_{i,j}B_{i,3}(u)B_{j,3}(v)$$

$$=[B_{0,3}(u)\quad B_{1,3}(u)\quad B_{2,3}(u)\quad B_{3,3}(u)]\begin{bmatrix}\vec{b}_{00}&\vec{b}_{01}&\vec{b}_{02}&\vec{b}_{03}\\ \vec{b}_{10}&\vec{b}_{11}&\vec{b}_{12}&\vec{b}_{13}\\ \vec{b}_{20}&\vec{b}_{21}&\vec{b}_{22}&\vec{b}_{23}\\ \vec{b}_{30}&\vec{b}_{31}&\vec{b}_{32}&\vec{b}_{33}\end{bmatrix}\begin{bmatrix}B_{0,3}(v)\\ B_{1,3}(v)\\ B_{2,3}(v)\\ B_{3,3}(v)\end{bmatrix}$$

$$=[1\quad u\quad u^2\quad u^3]\begin{bmatrix}1&0&0&0\\ -3&3&0&0\\ 3&-6&3&0\\ -1&3&-3&1\end{bmatrix}\begin{bmatrix}\vec{b}_{00}&\vec{b}_{01}&\vec{b}_{02}&\vec{b}_{03}\\ \vec{b}_{10}&\vec{b}_{11}&\vec{b}_{12}&\vec{b}_{13}\\ \vec{b}_{20}&\vec{b}_{21}&\vec{b}_{22}&\vec{b}_{23}\\ \vec{b}_{30}&\vec{b}_{31}&\vec{b}_{32}&\vec{b}_{33}\end{bmatrix}\begin{bmatrix}1&-3&3&-1\\ 0&3&-6&3\\ 0&0&3&-3\\ 0&0&0&1\end{bmatrix}\begin{bmatrix}1\\ v\\ v^2\\ v^3\end{bmatrix}$$

二、实现细节

1. 曲线绘制方法：

curve_tessellation 是给出的细分参数，也就是将这条曲线分成多少段（或者说插入多少个点）。下面是 Bezier 曲线的绘制代码，BSpline 也是类似，只是代入的公式不同。

```
    curve_tessellation = arg->curve_tessellation;
    GLfloat range = 1.0f / curve_tessellation;
    glColor3f (0, 1.0f, 0);    glPointSize (1.0f);
    glBegin (GL_LINES);
    //  Q(t)= p1*(1-t)^3 + 3*p2*t(1-t)^2 +  3*p3*t^2(1-t) +  p4*t^3
        for (int i = 0; i < num_vertices - 3; i += 3)//最后四个点是一条曲线{
            for (GLfloat t = 0; t < 1.0f; t += range){
                GLfloat x1 = vertices[i].x ()*pow (1.0f - t, 3) + 3 * vertices[1 + i].x ()*t*pow
(1.0f - t, 2) + 3 * vertices[2 + i].x ()*t*t*(1.0f - t) + vertices[3 + i].x ()*pow (t, 3);
                GLfloat y1 = vertices[i].y ()*pow (1.0f - t, 3) + 3 * vertices[1 + i].y ()*t*pow
(1.0f - t, 2) + 3 * vertices[2 + i].y ()*t*t*(1.0f - t) + vertices[3 + i].y ()*pow (t, 3);
                GLfloat z1 = vertices[i].z ()*pow (1.0f - t, 3) + 3 * vertices[1 + i].z ()*t*pow
(1.0f - t, 2) + 3 * vertices[2 + i].z ()*t*t*(1.0f - t) + vertices[3 + i].z ()*pow (t, 3);
                GLfloat t2 = t + range;
            //计算坐标方式同上…
                glVertex3f (x1, y1, z1);
                glVertex3f (x2, y2, z2);          }       }   }
    glEnd ();
```

2. 曲线相互转换：

```
void BSplineCurve::OutputBezier (FILE *file){
    //从BSpline转换成Bezier的方法：（相反同理）
    //已知bspline_G, bs_B, beizer_B, 求beizer_G
    //beizer_G = bspline_G * bs_B * beizer_B的逆矩阵;
    Matrix bspline_G;
    for (int i = 0; i < num_vertices; i++)  {
        bspline_G.Set (i,0,vertices[i].x());bspline_G.Set (i,1,vertices[i].y());
        bspline_G.Set (i, 2, vertices[i].z ()); }
    float beizer_b[16] = { -1, 3, -3, 1, 3, -6, 3, 0, -3, 3, 0, 0, 1, 0, 0, 0 };
    Matrix beizer_B (beizer_b);
    float bs_b[16] = { -1, 3, -3, 1, 3, -6, 0, 4, -3, 3, 3, 1, 1, 0, 0, 0 };
    Matrix bs_B (bs_b);    bs_B = bs_B * (1.0f / 6.0f);    beizer_B.Inverse ();
    Matrix beizer_G = bspline_G * bs_B*beizer_B;
    //save file...
    }
```

3. 曲线生成三角面模型：

（1）旋转曲面：因为曲线在 yox 平面上，z 的值为 0。所以计算曲线的坐标时不用算 z 值。当曲线绕着 Y 轴旋转后，可以根据旋转角度和 x 的坐标值，确定新的点的 x 和 z 坐标值。

```
curve_tessellation = args->curve_tessellation;
    int _v_tess = ((num_vertices / 4) + 1) * curve_tessellation;//旋转面分割的个数
    TriangleNet *tri = new TriangleNet (args->revolution_tessellation, _v_tess);
    double degree = 2 * M_PI / args->revolution_tessellation;//旋转次数
    if (type_id == Bezier) {
        int count = 0;
        for (int i = 0; i < num_vertices - 3; i += 3){
            for (int t2 = 0; t2 <= curve_tessellation; t2++){
                //先画出曲线上的每个点
                GLfloat t = t2 * 1.0f / curve_tessellation;
                GLfloat x = vertices[i].x ()*pow (1.0f - t, 3) + 3 * vertices[1 + i].x ()*t*pow
(1.0f - t, 2) + 3 * vertices[2 + i].x ()*t*t*(1.0f - t) + vertices[3 + i].x ()*pow (t, 3);
                GLfloat y = vertices[i].y ()*pow (1.0f - t, 3) + 3 * vertices[1 + i].y ()*t*pow
(1.0f - t, 2) + 3 * vertices[2 + i].y ()*t*t*(1.0f - t) + vertices[3 + i].y ()*pow (t, 3);
                //指定网格点的顶点位置: 0-10   1-11
                for (int j = 0; j <= args->revolution_tessellation; j++){
                    GLfloat x2 =  x * cos (j*degree);  GLfloat y2 = y;
                    GLfloat z2 = - x * sin (j*degree); //u_tess   v_tess
                    tri->SetVertex (j, t2 + count * curve_tessellation, Vec3f (x2, y2, z2));
            }            }
            count++;     }   }
```

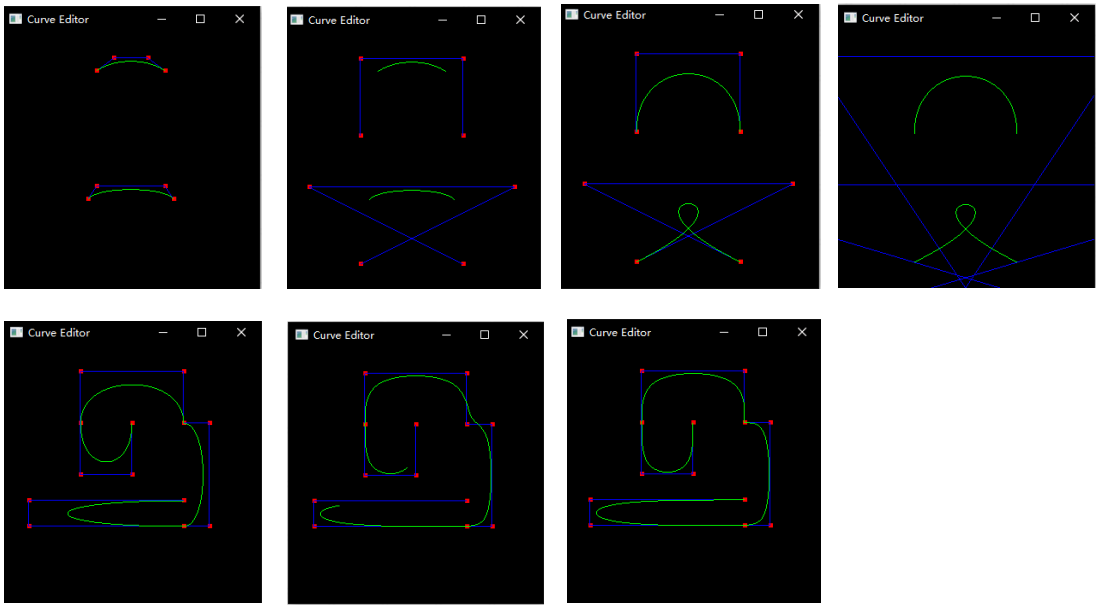
（2）参数化曲面：Bezier Patch

```
riangleMesh* BezierPatch::OutputTriangles (ArgParser *args){
    TriangleNet *tri = new TriangleNet (args->patch_tessellation, args->patch_tessellation);
    GLfloat range = 1.0f / args->patch_tessellation;
```

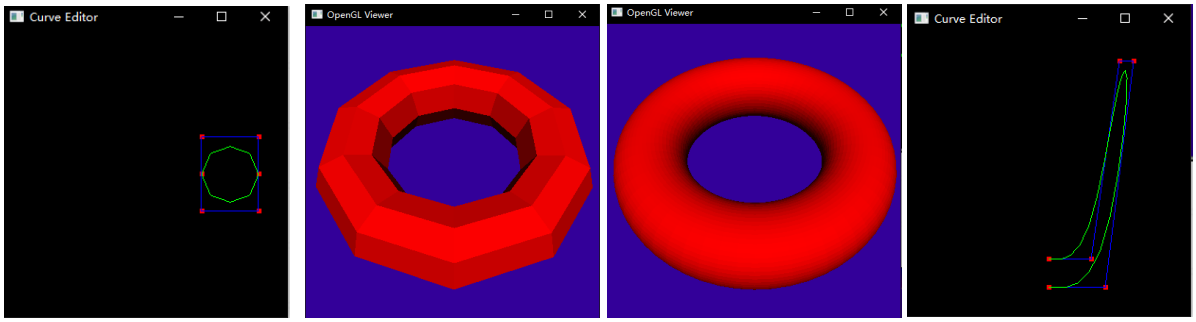
```
    GLfloat bm[16] = { -1, 3, -3, 1, 3, -6, 3, 0, -3, 3, 0, 0, 1, 0, 0, 0 };
    Matrix BM (bm);   Matrix BM_T = BM; BM_T.Transpose ();
    for (int u = 0; u <= args->patch_tessellation; u++){
        for (int v = 0; v <= args->patch_tessellation; v++){
            float u_tess = u * 1.0f / args->patch_tessellation;
            float v_tess = v * 1.0f / args->patch_tessellation;
            Vec3f vertex;
            GLfloat m_u[4] = { pow (u_tess, 3), pow (u_tess, 2), u_tess, 1 };
            Matrix mu (m_u); Matrix temp1 = mu * BM;    Vec3f vs[4];
            for (int i = 0; i < 4; i++){
                vs[i] = temp1.Get (0, 0)*vertices[0 + i] + temp1.Get (1, 0)*vertices[4 + i]
+ temp1.Get (2, 0)*vertices[8 + i] + temp1.Get (3, 0)*vertices[12 + i];}
            Vec3f vs2[4];
            for (int i = 0; i < 4; i++){
                vs2[i] = vs[0] * BM_T.Get (0, i) + vs[1] * BM_T.Get (1, i) + vs[2] *
BM_T.Get (2, i) + vs[3] * BM_T.Get (3, i);}
            vertex = vs2[0] * pow (v_tess, 3) + vs2[1] * pow (v_tess, 2) + vs2[2] * v_tess +
vs2[3];
            tri->SetVertex (v, u, vertex); }}
    return tri;}
```

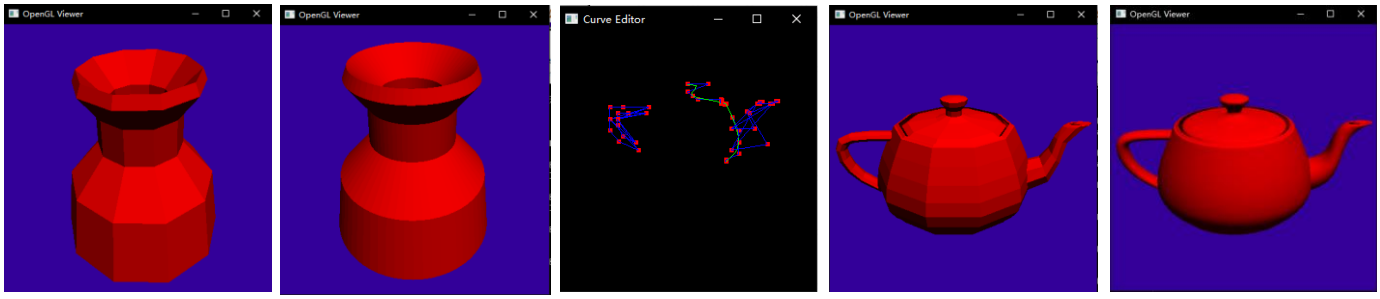
三、结果展示

1.曲线测试:

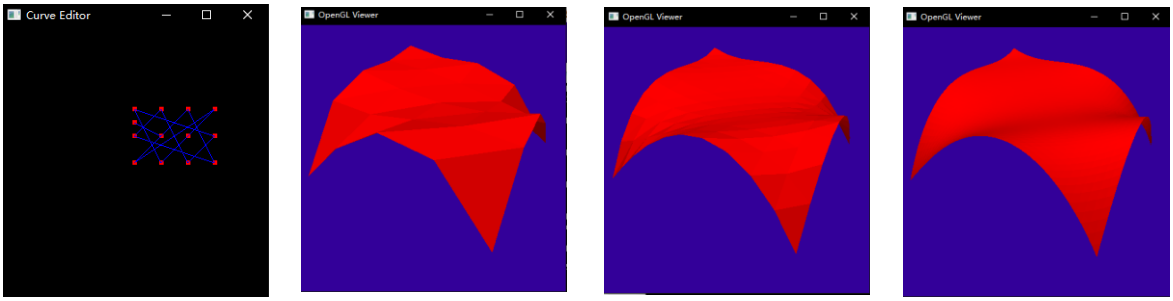


2.旋转曲面测试:





3. Bezier Patch 测试:



四、心得体会

通过这次作业感受到，在图形绘制中各种线性代数、高数中关于立体几何的知识很常用的。还有就是离散化是非常重要的一个思想，在有了公式之后还要进行适当的细分才能绘制出正确的图形。