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。

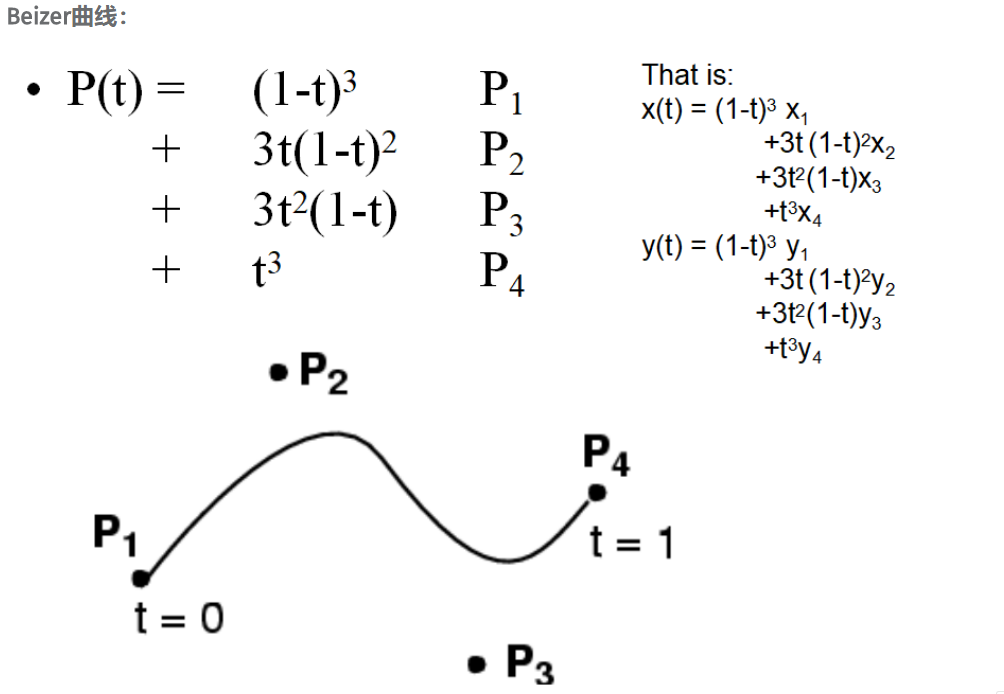
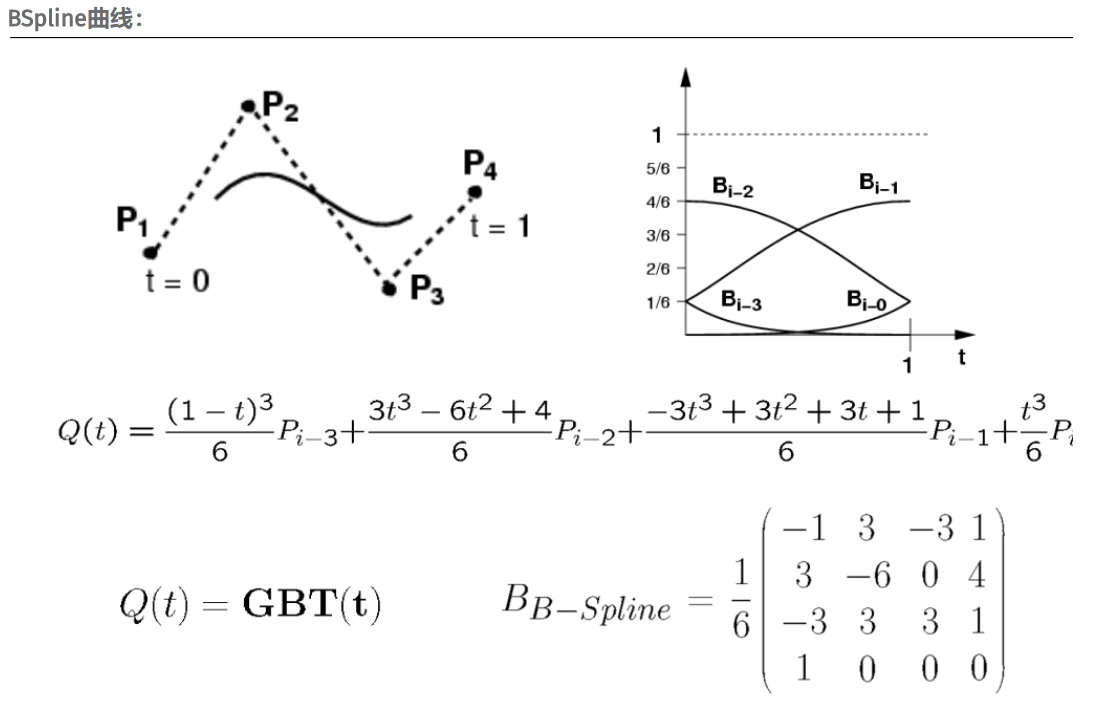
（2）实现Bezier和BSpline的互相转换，并存成文件。

（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。

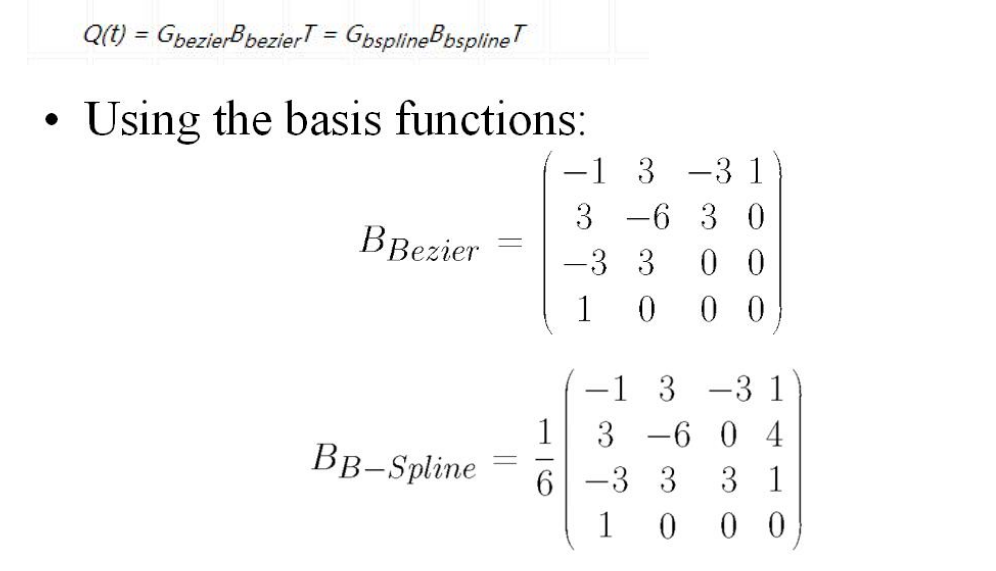
程序输入curve\_tessellation值，也就是将该段曲线划分为多少段。假如输入30，那么把t的范围映射到[0,1）范围内，将曲线划分为30份，每次迭代，将t值加1/30，就可以得到30个曲线上的点。

（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。

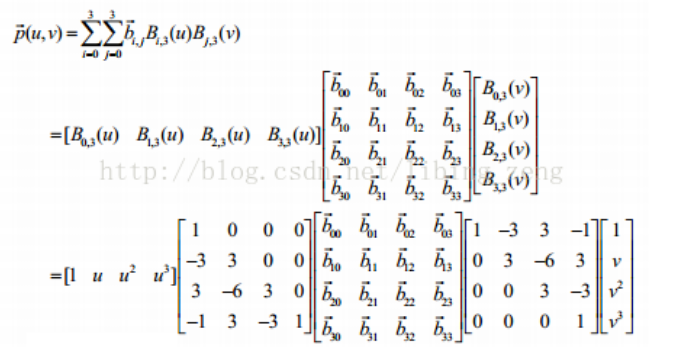
vertices[selectedPoint].Set (x, y, vertices[selectedPoint].z ());

* 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。

（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。

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：**

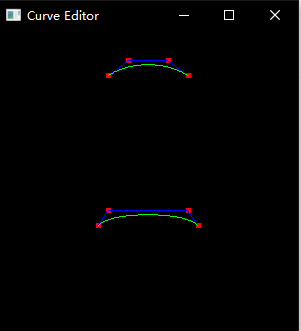
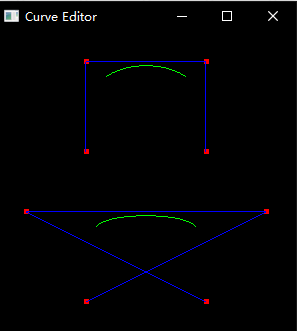
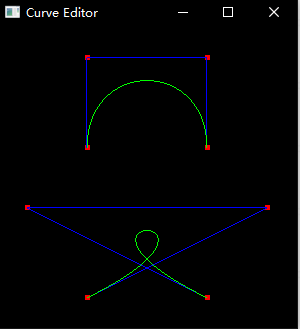
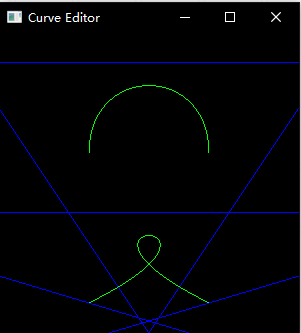
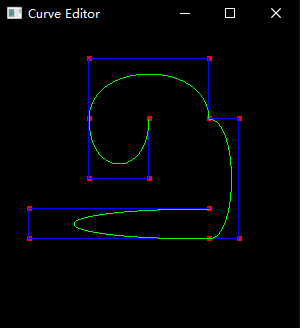
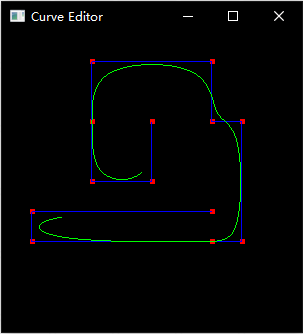
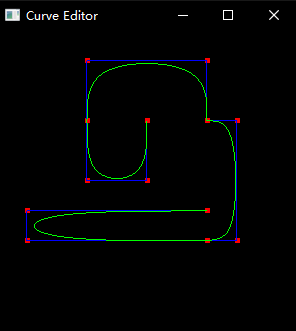
**（1）旋转曲面：因为曲线在yox平面上，z的值为0。所以计算曲线的坐标时不用算z值。当曲线绕着Y轴旋转后，可以根据旋转角度和x的坐标值，确定新的点的x和z坐标值。**

|  |
| --- |
| 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++; } } |

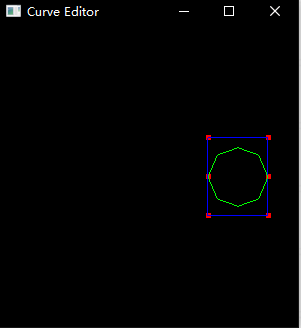
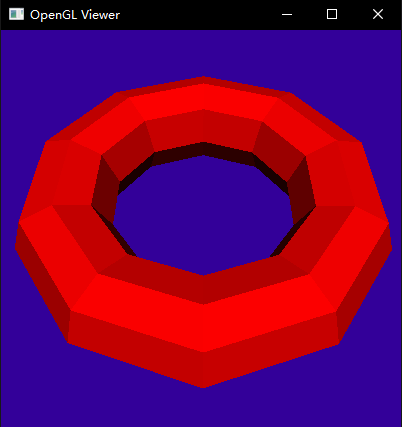
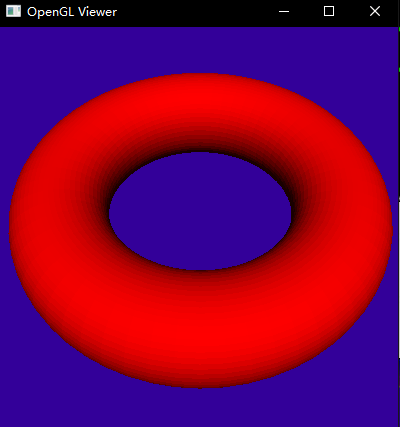
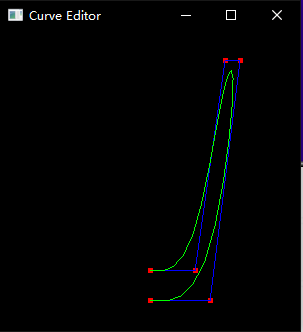
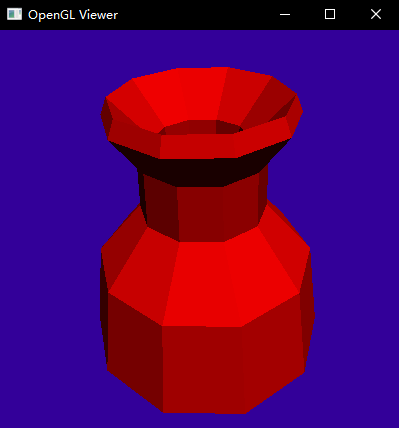
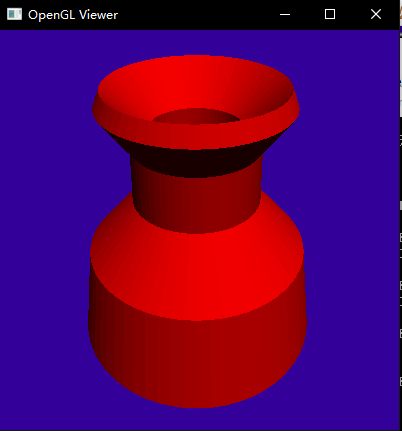
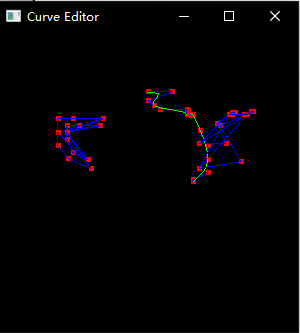
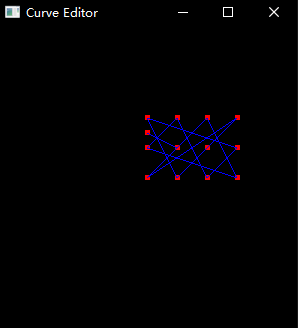
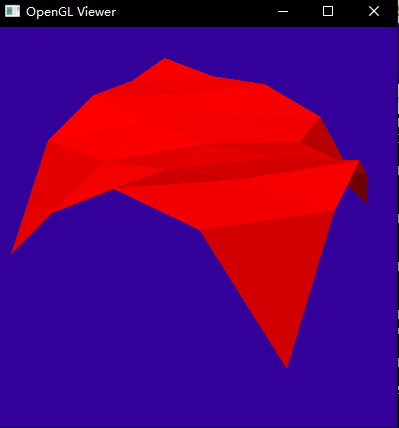
**（2）参数化曲面：Bezier Patch**

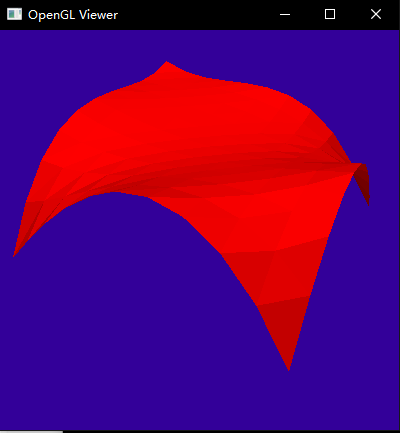
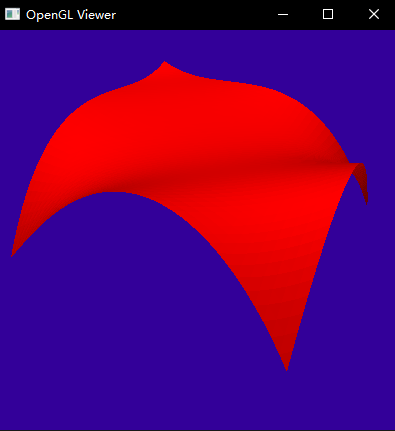
|  |
| --- |
| riangleMesh\* BezierPatch::OutputTriangles (ArgParser \*args){  TriangleNet \*tri = new TriangleNet (args->patch\_tessellation, args->patch\_tessellation);  GLfloat range = 1.0f / args->patch\_tessellation;  GLfloat bm[16] = { -1, 3, -3, 1, 3, -6, 3, 0, -3, 3, 0, 0, 1, 0, 0, 0 };  Matrix 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。