图形学作业 1:

曲线和曲面 Curves & Surfaces

学　　号：20171002157

班级序号：111172

姓　　名：杨杰

指导教师：罗忠文

中国地质大学信息工程学院软件工程系

2019 年 9 月

目录

作业简介

本次作业要求你完整实现一个简单的 2D 样条曲线编辑器。该编辑器应支持 Bezier 曲线和 B 样条曲线，可以实现两种曲线的转换(仅对四个控制点的双三次样条曲线)。在你的曲线程序能工作后，将转到由这些曲线来产生曲面：旋转曲面和双三次 Bezier 片。

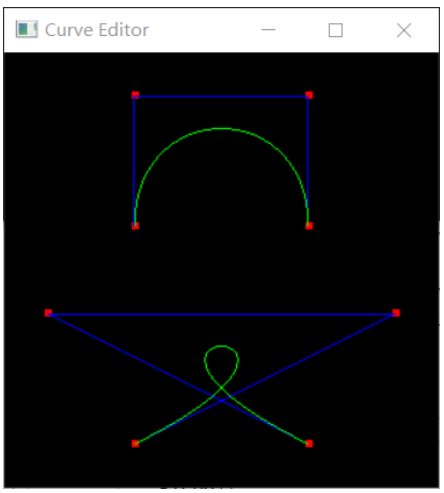
具体实现

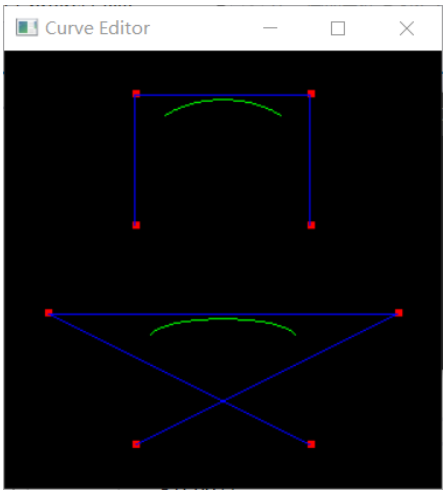
为 Curve 类实现 Paint(ArgParser*)函数

该函数是绘制控制点和控制点连线函数。已知点的数量为 vertices_num，点的坐标在 vertices_array 中。

首先绘制出点，遍历 vertices_num 个点，每次使用 Get()得到需要绘制的点的坐标，然后使用函数 glBegin(GL_POINTS)、glBegin(GL_LINES)、glPointSize(int)、glLineWidth(int)、glColor3f(float,float,float)和 glVertex3f(float,float,float)将点画出。

然后绘制线，遍历 vertices_num-1 个线段，Get()函数分别得到线段起点和终点的点坐标，使用 glBegin(GL_LINES)等函数将线段画出。





绘制 Bezier 曲线和 BSplines 曲线

绘制 Bezier 曲线，首先调用 Curve::Paint(args)，将控制点绘制出。然后通过 args->curve_tessellation 得到输入的曲线的细分程度。遍历 vertices_num – 1 个线段，线段中遍历线段的每一细分程度的线段，画每一段细分后的线段。

$$P(t) = \sum_{i=0}^{n} C_n^i t^i (1-t)^{n-i} \vec{P_i}\ , t \in [0,1]$$

完成函数 Get_Bezier_vertice(GLfloat t, int index)得到 Bezier 曲线上的点坐标，其中其中 t = （vertices_num - 1 个线段的角标）/ 细分程度，index 为所选四个点的首给点的角标。

具体函数如下：

```
virtual void Paint(ArgParser* args){
    Curve::Paint(args);//控制点绘制
    int subdivision = args->curve_tessellation;
    for (int i = 0; i < vertices_num - 1;){//每由四个点通过函数绘制出曲线上的一个点
        glColor3f(0, 1, 0);
        glBegin(GL_LINE_STRIP);
        for (int j = 0; j <= subdivision; j++) {//画每一段线段
            GLfloat t = (GLfloat)j / ((GLfloat)subdivision);
            Vec3f v = Get_Bezier_vertice(t, i);//获得线上点坐标
            glVertex3f(v.x(), v.y(), v.z());
        }
        glEnd();
        i += 3;
    }
}
```

```
Vec3f Get_Bspline_vertice(GLfloat t, int index) {
    float p1 = (1.0 - t) * (1.0 - t) * (1.0 - t);   //(1-t)^3
    float p2 = 3.0 * t * (1.0 - t) * (1.0 - t);     //3t(1-t)^2
    float p3 = 3.0 * t * t * (1.0 - t);             //3t^2(1-t)
    float p4 = t * t * t;                           //t^3

    float x = vertices_array[index].x() * p1 + vertices_array[index + 1].x() * p2
        +vertices_array[index + 2].x() * p3 + vertices_array[index + 3].x() * p4;
    float y = vertices_array[index].y() * p1 + vertices_array[index + 1].y() * p2
        +vertices_array[index + 2].y() * p3 + vertices_array[index + 3].y() * p4;
    float z = vertices_array[index].z() * p1 + vertices_array[index + 1].z() * p2
        +vertices_array[index + 2].z() * p3 + vertices_array[index + 3].z() * p4;

    return Vec3f(x, y, z);
}
```

绘制 BSplines 曲线，基本思路与上述同同理。差别在 Vec3f Get_BSpline_vertice(GLfloat t, int index)函数中计算曲线上点的公式的不同。

$$P(t) = \sum_{i=0}^{n} \vec{P_i} F_{i,n}(t)\ , t \in [0,1]$$

根据公式

得到获取坐标的函数：

```
Vec3f Get_BSpline_vertice(GLfloat t, int index){
    float p1 = (-1.0 * (t * t * t) + 3.0 * (t * t) - 3.0 * t + 1.0) / 6.0;
    float p2 = (3.0 * (t * t * t) - 6.0 * (t * t) + 4.0) / 6.0;
    float p3 = (-3.0 * (t * t * t) + 3.0 * (t * t) + 3.0 * t + 1.0) / 6.0;
    float p4 = (t * t * t) / 6.0;

    float x = vertices_array[index].x() * p1 + vertices_array[index + 1].x() * p2
        + vertices_array[index + 2].x() * p3 + vertices_array[index + 3].x() * p4;
    float y = vertices_array[index].y() * p1 + vertices_array[index + 1].y() * p2
        +vertices_array[index + 2].y() * p3 + vertices_array[index + 3].y() * p4;
    float z = vertices_array[index].z() * p1 + vertices_array[index + 1].z() * p2
        +vertices_array[index + 2].z() * p3 + vertices_array[index + 3].z() * p4;

    return Vec3f(x, y, z);
}
```

实现 Bezier 曲线和 BSplines 曲线之间的转换

Bezier 的 OutputBezier()和 BSplines 的 OutputBSplines ()，直接遍历 vertices_num 个点，然后将 x、y、z 坐标输出即可。

Bezier 的 OutputBSplines ()和 BSplines 的 OutputBezier()通过对描述样条曲线的矩阵方程进行运算来实现：

$$Q(t) = \mathbf{GBT(t)} = \text{Geometry } \mathbf{G} \cdot \text{Spline Basis } \mathbf{B} \cdot \text{Power Basis } \mathbf{T(t)}$$

$$B_{Bezier} = \begin{pmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} \qquad B_{B-Spline} = \frac{1}{6}\begin{pmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 0 & 4 \\ -3 & 3 & 3 & 1 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

$$G_1 = G_2 B_2 B_1^{-1} \qquad G_2 = G_1 B_1 B_2^{-1}$$

在 Bezier 的 OutputBSpline(FILE* file)首先用 float 数组描述 Bbezier 和 Bbspline 矩阵的值，然后用 Matrix 构造对应的矩阵类，将 bezier 矩阵 * bspline 的逆矩阵，得到公式中的后两参数，暂时叫 B1andB2Inverse，然后通过一个中介 float 数组，将需要转换的矩阵点转换为矩阵类型，该结果乘上述 B1andB2Inverse 即得到最终结果，存入 Vec3f* 数组中，再写入文件制定文件中，释放上述用到的内存即可。

在 BSplines 的 OutputBezier(FILE* file)中基本同理上述过程，只需将后面 B1andB2Inverse 所需的矩阵互换即可。

输出文件截图及测试截图如下图所示：

output01_bezier.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
num_splines 2
bezier num_vertices 4
-2.0 1.0 0.0
-2.0 4.0 0.0
2.0 4.0 0.0
2.0 1.0 0.0
bezier num_vertices 4
-2.0 -4.0 0.0
4.0 -1.0 0.0
-4.0 -1.0 0.0
2.0 -4.0 0.0
```

output01_bspline.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
num_splines 2
bspline num_vertices 4
 6.0 -14.0 0.0
-6.0 4.0 0.0
6.0 4.0 0.0
-6.0 -14.0 0.0
bspline num_vertices 4
 -48.0 -19.0 0.0
12.0 -1.0 0.0
-12.0 -1.0 0.0
48.0 -19.0 0.0
```

output02_bezier.txt - 记事本

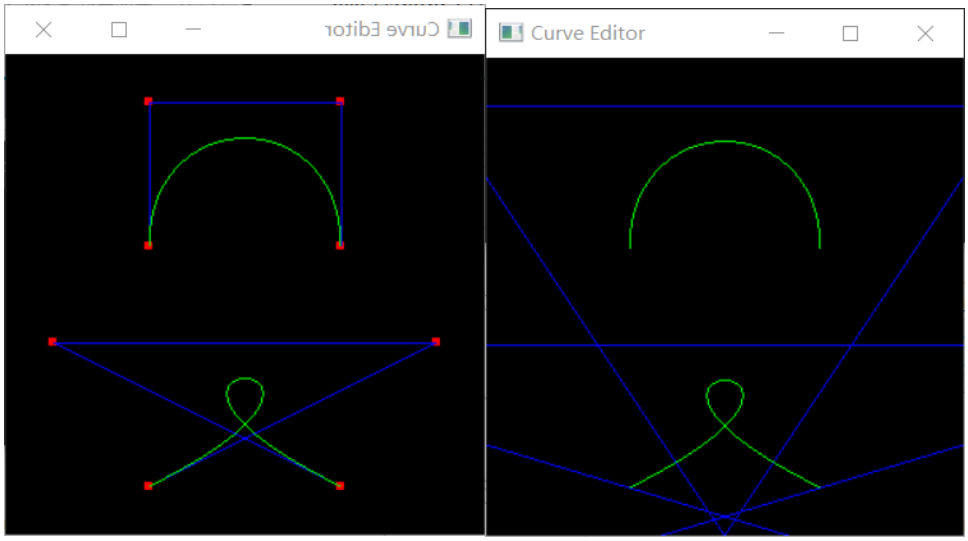
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

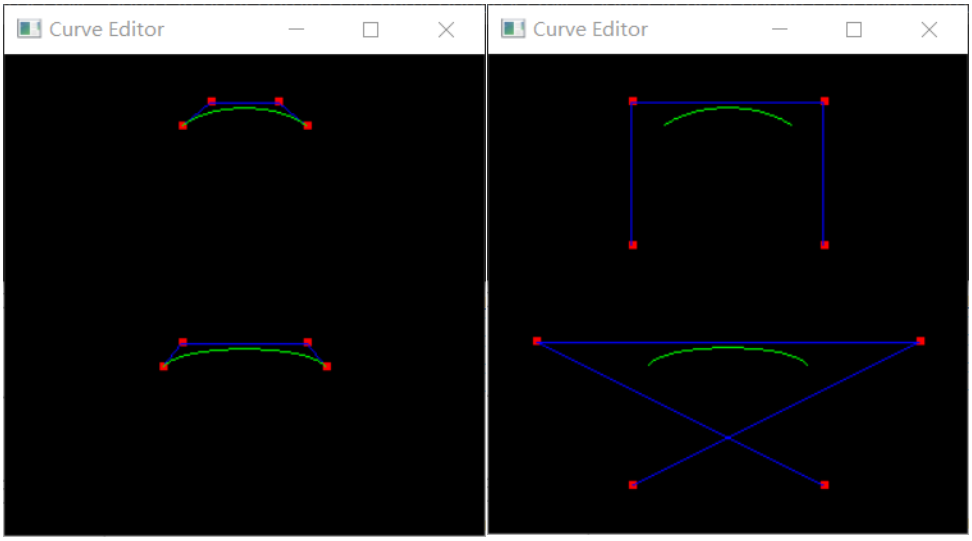
```
num_splines 2
bezier num_vertices 4
 -1.3 3.5 0.0
-0.7 4.0 0.0
0.7 4.0 0.0
1.3 3.5 0.0

bezier num_vertices 4
 1.7 -1.5 0.0
1.3 -1.0 0.0
-1.3 -1.0 0.0
-1.7 -1.5 0.0
```

output02_bspline.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
num_splines 2
bspline num_vertices 4
-2.0 1.0 0.0
-2.0 4.0 0.0
2.0 4.0 0.0
2.0 1.0 0.0
bspline num_vertices 4
-2.0 -4.0 0.0
4.0 -1.0 0.0
-4.0 -1.0 0.0
2.0 -4.0 0.0
```





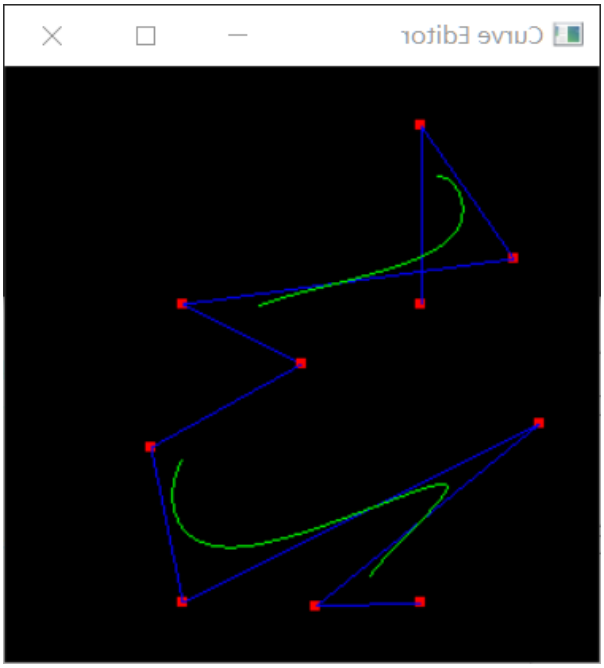
实现控制点编辑函数

所需要实现的函数为 spline 类中的函数，在添加、移动点的时候，对指定下标的数组进行修改操作即可，删除控制点时进行数组的移位操作即可，对应函数如下：

```
virtual int getNumVertices(){
    return vertices_num;
}
virtual Vec3f getVertex(int i){
    return vertices_array[i];
}
```

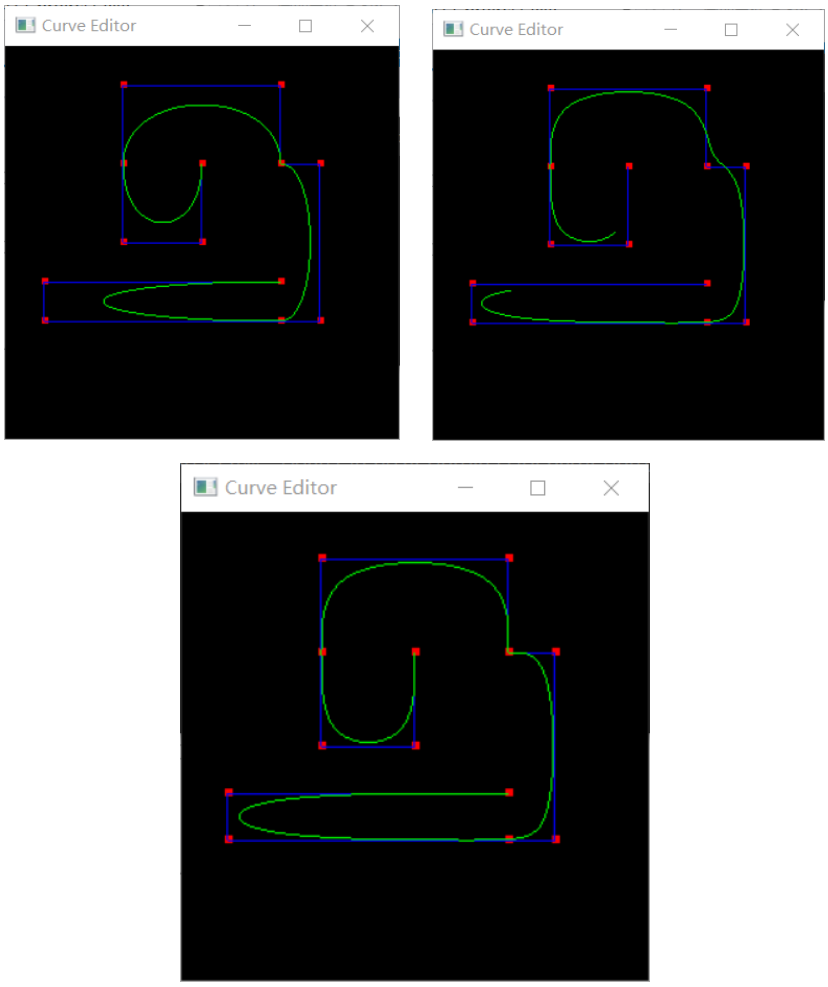
```
// 用于编辑操作的FOR EDITING OPERATIONS
virtual void moveControlPoint(int selectedPoint, float x, float y){
    Vec3f v = Vec3f(x, y, 0);
    vertices_array[selectedPoint] = v;
}
virtual void addControlPoint(int selectedPoint, float x, float y){
    Vec3f v = Vec3f(x, y, 0);
    vertices_array[selectedPoint] = v;
    vertices_num++;
}
virtual void deleteControlPoint(int selectedPoint){
    for (int i = selectedPoint;  i < vertices_num - 1; i++){
        vertices_array[selectedPoint] = vertices_array[selectedPoint + 1];
    }
    vertices_num--;
}
```

添加控制点后的测试截图：



实现 >4 个控制点的曲线转换

初始化 vertices_array 数组的时候让它尽量的大即可，这里给了 vertices_array=new Vec3f[1000]。这里在之前的 paint()函数中已经实现。



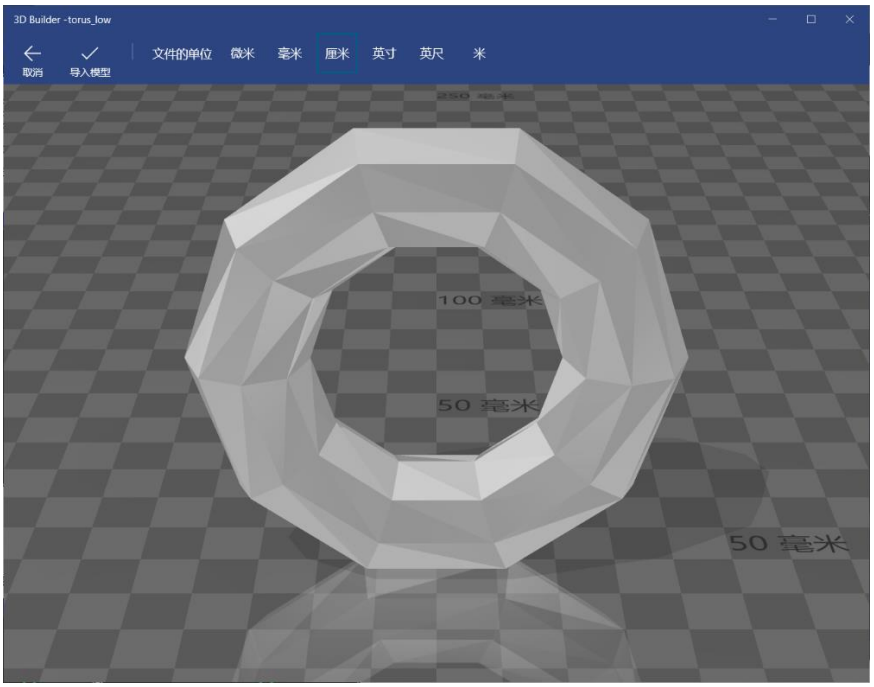
实现 SurfaceOfRevolution 类

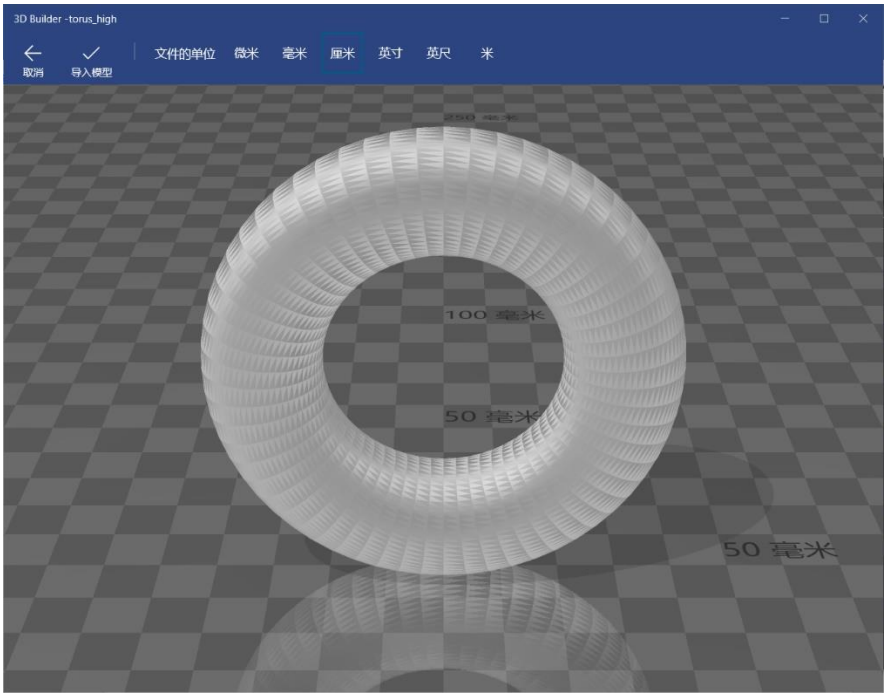
这里是实现样条曲线旋转建模，创建 obj 文件供查看 3D 图像。旋转体由旋转面连接而成的，由无数给矩形构成，矩形又由三角形构成，而三角形可以得到模型的三角形网格。使用 Matrix 类中 提供的 MakeAxisRotation()方法，生成一个绕指定轴旋转的变换矩阵，这里用到了曲线的细分度的和的旋转的细分度，由细分度可以得到每次旋转的角度和矩阵变换后的新点等，把他们加入到 TriangleMesh* mesh = new TriangleMesh(num_verts, num_tris)其中

int num_verts = ((((float)vertices_num - 1.0) / 3.0) * args->curve_tessellation + 1) * args->revolution_tessellation;

int num_tris = (((float)vertices_num - 1.0) / 3.0) * args->curve_tessellation * args->revolution_tessellation * 2;

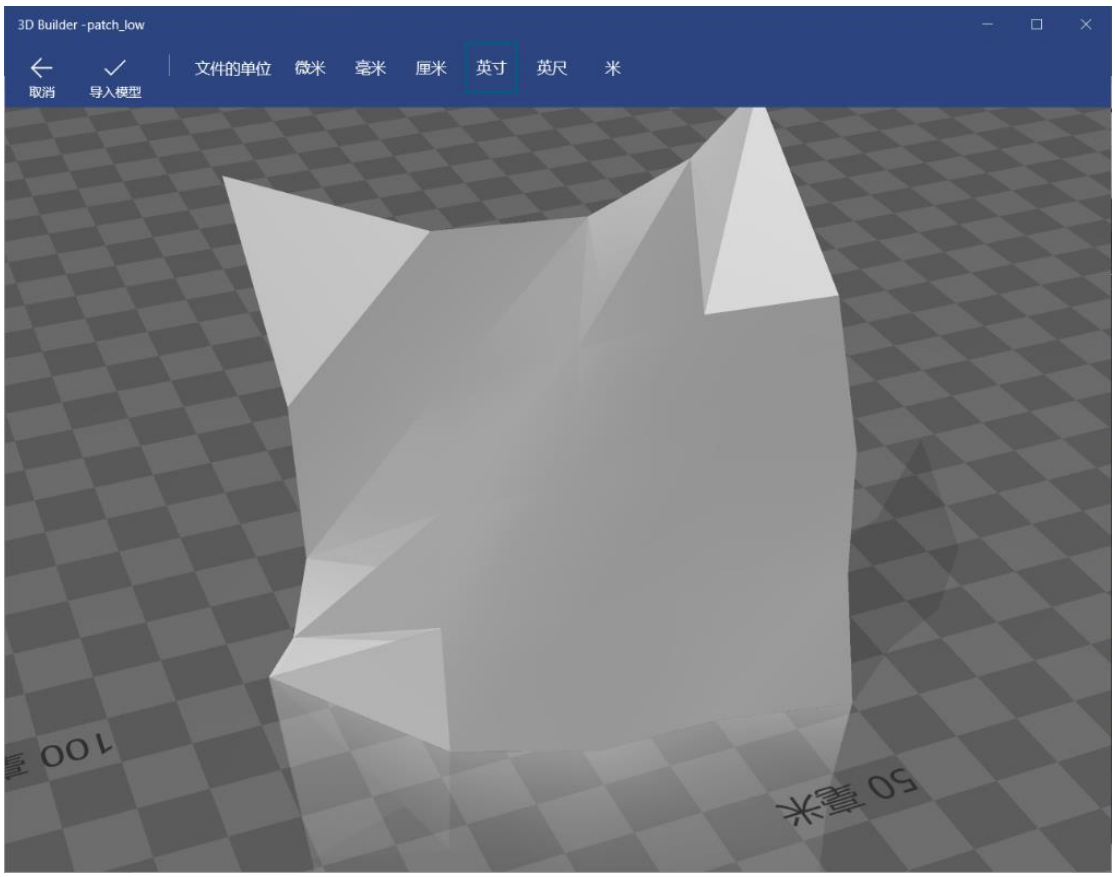
然后 SetTriangle()给三角形网格中每一个顶点编号，最后在 OutputTriangles() 这个函数中调用 TrangleMesh 生成 obj 文件。

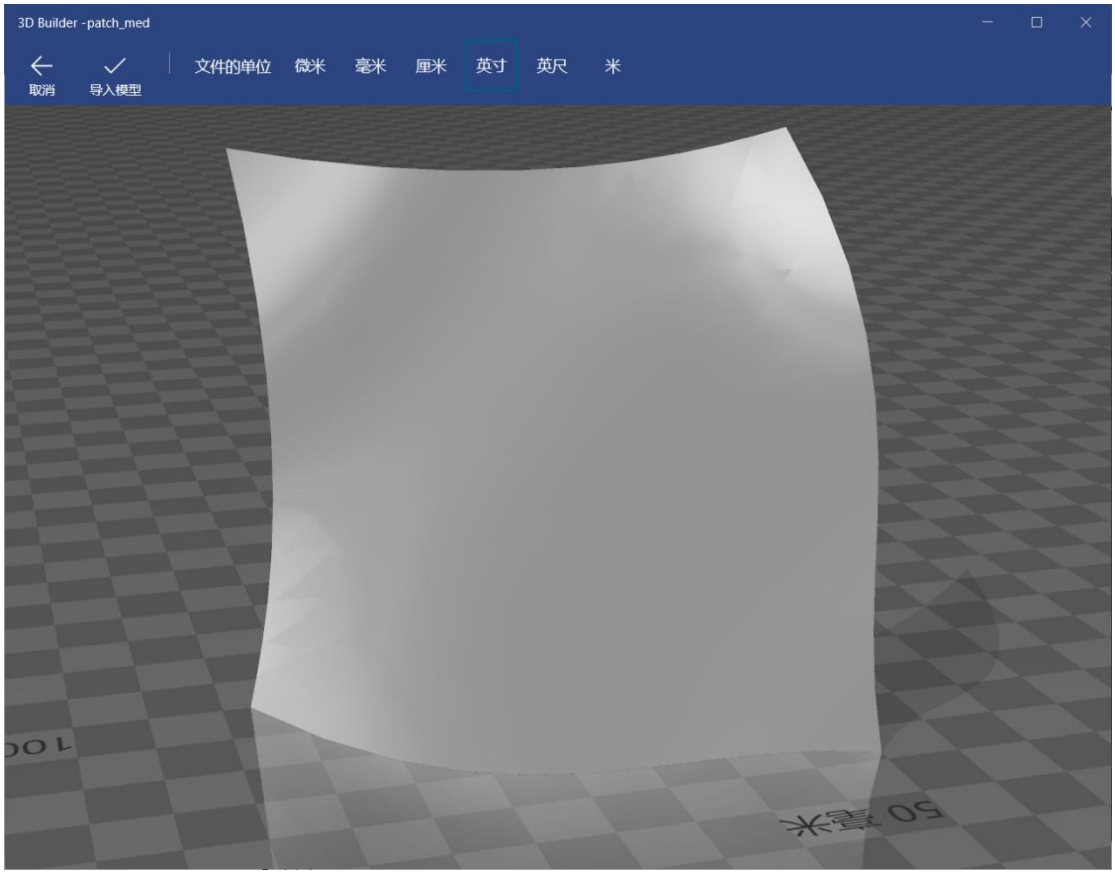


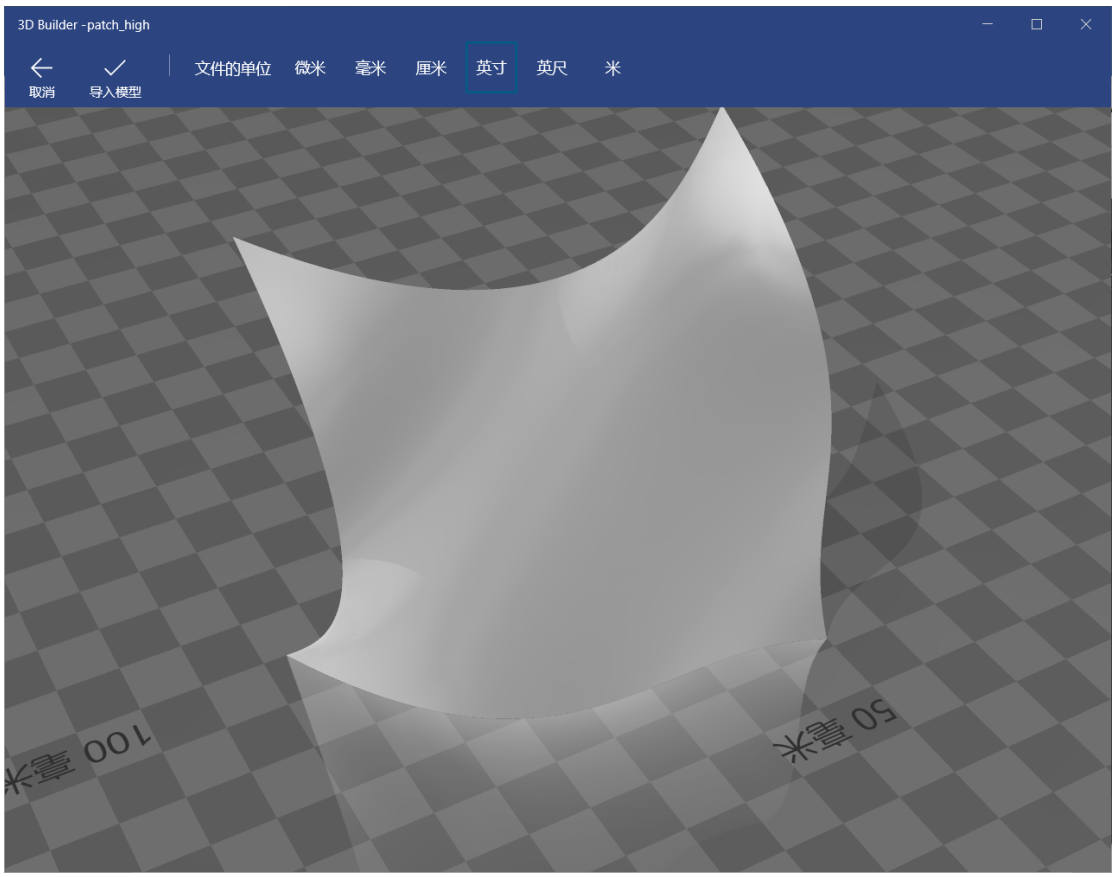


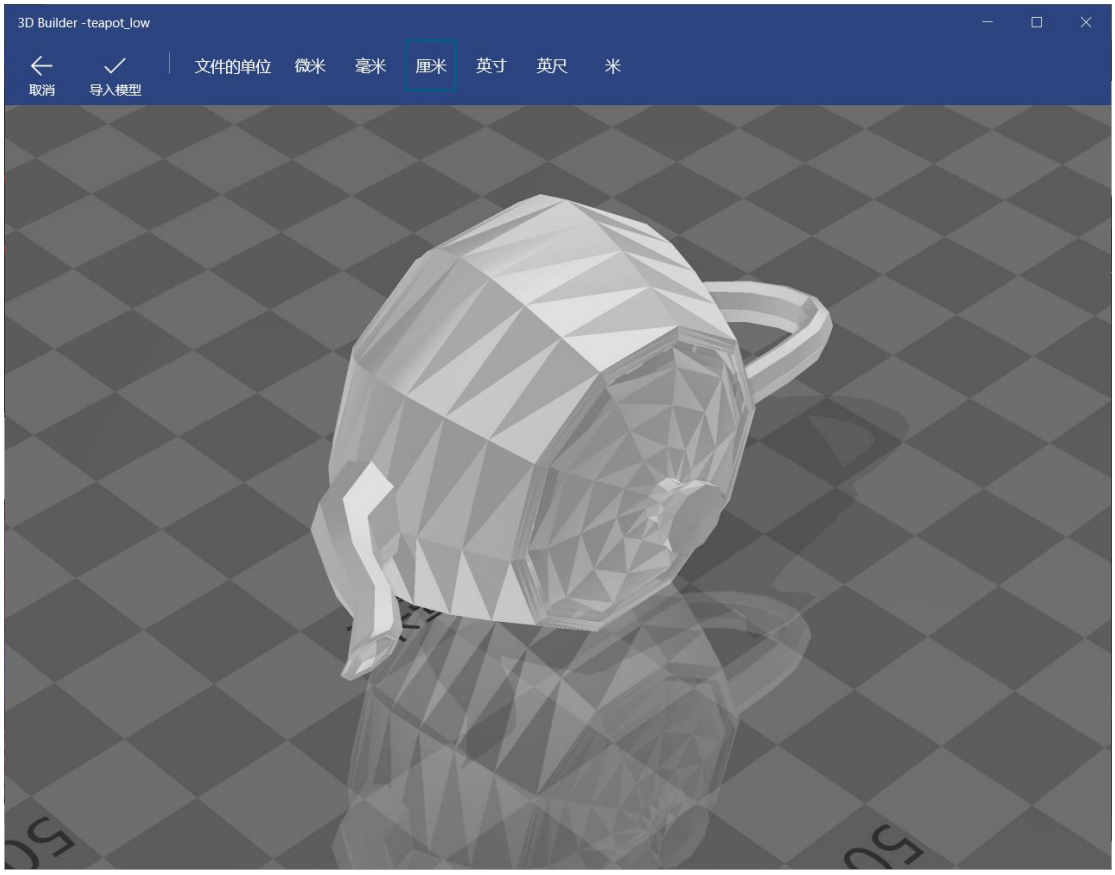
实现 16 个控制点的 4x4 Bezier 块

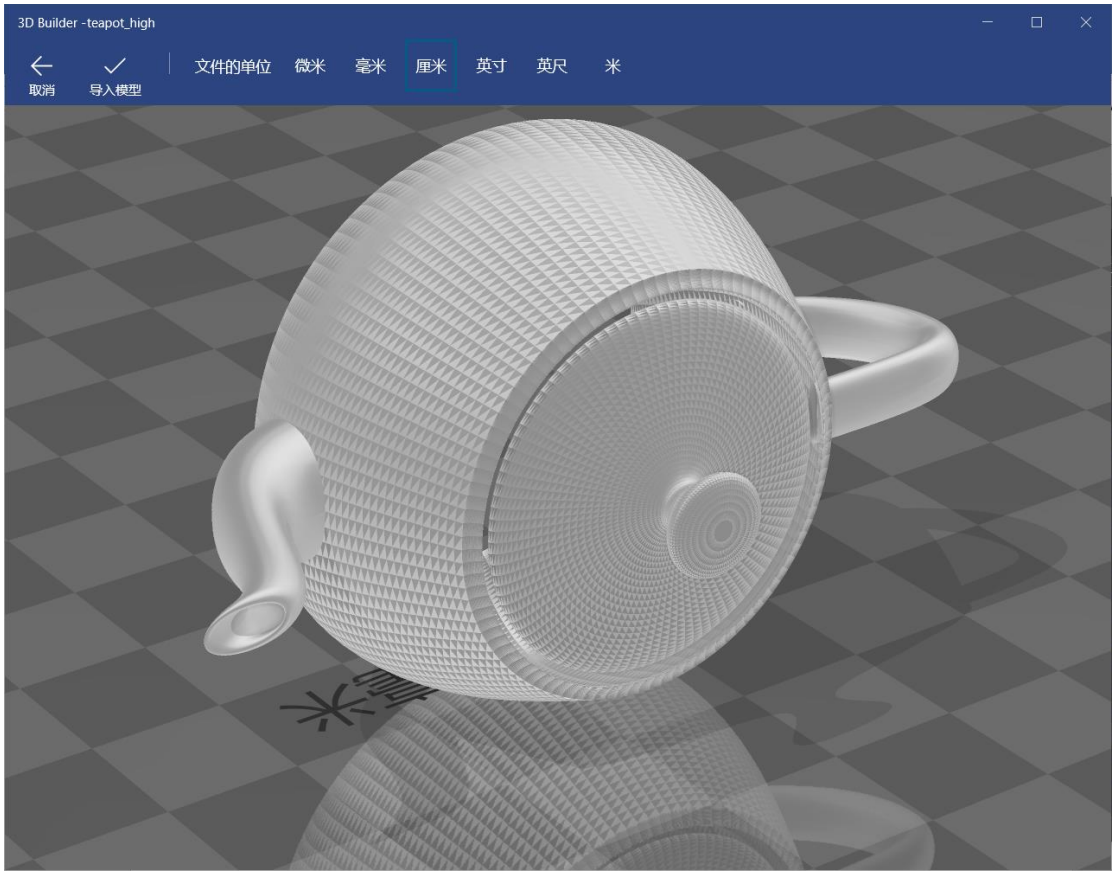
框架供了 TriangleNet 类，这个类负责处理 4*4 个控制点下的贝塞尔曲面三角形网格的生成。构建以 patch_tessellation 为参数的 TriangleNet 类，再添加点即可。


3D Builder - patch_low
取消
导入模型
文件的单位
微米
毫米
厘米
英寸
英尺
米


3D Builder - patch_med
取消
导入模型
文件的单位
微米
毫米
厘米
英寸
英尺
米


3D Builder -patch_high
取消
导入模型
文件的单位
微米
毫米
厘米
英寸
英尺
米


3D Builder -teapot_low
取消
导入模型
文件的单位
微米
毫米
厘米
英寸
英尺
米


3D Builder -teapot_high
取消
导入模型
文件的单位
微米
毫米
厘米
英寸
英尺
米