曲线与曲面

● 简述

本次作业的主要内容是普通的（四个控制点）贝塞尔曲线、B 样条曲线的绘制；上述两种曲线的相互转换；贝塞尔曲线和 By 样条曲线的一般化；将样条曲线饶轴旋转生成曲面（建模）；16 个控制点的 Bezier 块。下面将分别介绍如何实现上述内容。

● 样条曲线的绘制

1. 贝塞尔曲线

我们知道贝塞尔曲线满足下列参数方程：

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

其中 t 是参数，一般从 0 取至 1。方程右侧的 P 向量代表曲线的控制点，n 代表曲线的阶数。程序在启动是需要有一个 curve_tessellation 参数来指定曲线的细分度。举个例子，细分度如果是 5 的话，在绘制曲线时就会取 t 为 0、1/5、2/5、3/5、4/5、1 这 6 个点，然后将这 6 个点按顺序连接，这样一条贝塞尔曲线就被绘制出来了。

可以看出来细分度越高曲线的将越顺滑，后面将不在赘述这个问题。

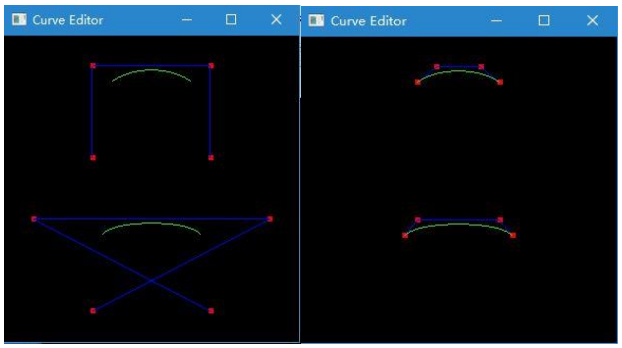
2. B 样条曲线

通过上面的例子，我们现在只要知道了 B 样条曲线的参数方程即可绘制出它，通过查阅资料知道该曲线满足如下方程：

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

$$F_{i,n}(t) = \sum_{j=0}^{n-i} (-1)^j C_{n+1}^j (t+n-i-j)^n \ , t \in [0,1]$$

在实现程序时我们需要在 BezierCurve 和 BsplineCurve 两个类中覆写虚函数 Paint，通过传入的 ArgParser 类取得细分度然后调用 OpenGL 函数即可绘出相应的曲线。



● 贝塞尔曲线和 B 样条曲线的相互转换

我们在绘制曲线的时候使用的是方程的一般形式，现在我们将上述方程转换成矩阵的形式，这样会更利于我们思考。

$$P(t) = \mathrm{GBT(t)}$$

$$\mathrm{G} = (\overrightarrow{P_1} \quad \overrightarrow{P_2} \quad \overrightarrow{P_3} \quad \overrightarrow{P_4})$$

$$\mathrm{B}_{bezier} = \begin{pmatrix} 1 & -3 & 3 & -1 \\ 0 & 3 & -6 & 3 \\ 0 & 0 & 3 & -3 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad \mathrm{B}_{bspline} = \frac{1}{6}\begin{pmatrix} 1 & -3 & 3 & -1 \\ 4 & 0 & -6 & 3 \\ 1 & 3 & 3 & -3 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\mathrm{T} = \begin{pmatrix} 1 \\ t \\ t^2 \\ t^3 \end{pmatrix}$$

通过上面的方程我们可以知道，线条不变那么 P(t)的值不应该发生变化，同时 T 矩阵又是个常量，我们可以得到如下矩阵方程方程：

$$G_1 B_1 = G_2 B_2$$

将方程两边同时右乘B_2的逆矩阵，得到：

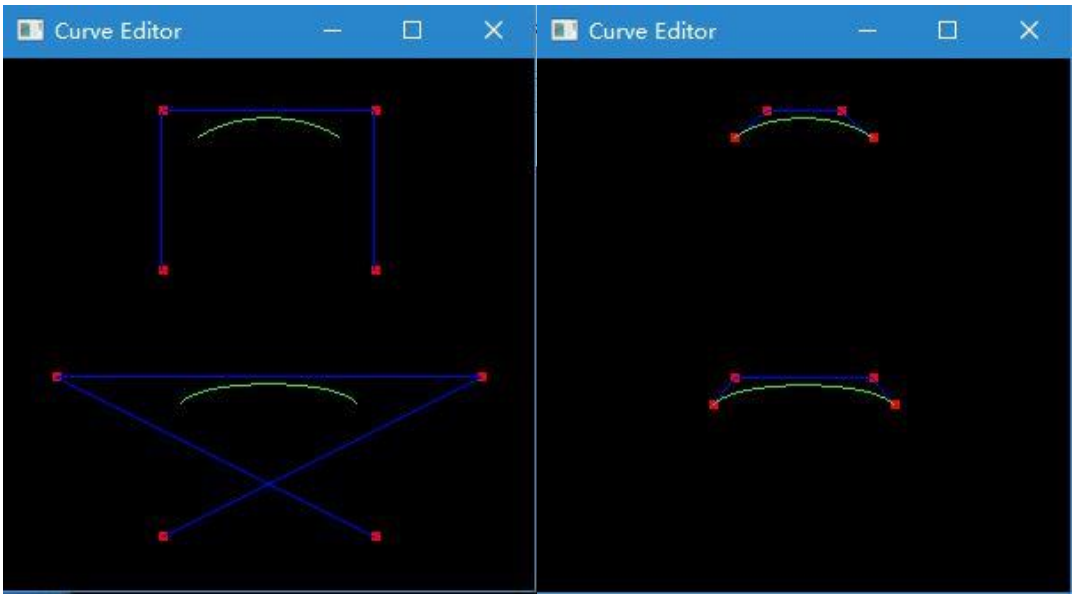
$$G_1 B_1 {B_2}^{-1} = G_2 B_2 {B_2}^{-1}$$

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

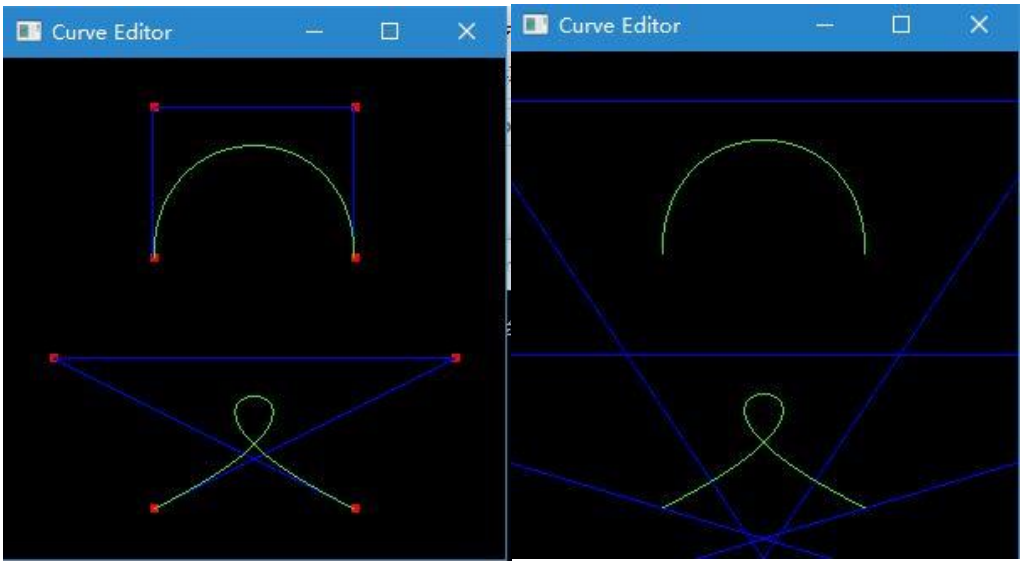
这样我们就知道只需要将原来控制点组成的矩阵 G 做上述运算，就能得到转换后曲线的控制点，同时曲线的形状保持不变。

在实现的时候，我们可以使用框架提供的 Matrix 类进行矩阵的乘法和逆运算。你只需要在 BezierCurve 和 BsplineCurve 两个类中覆写 OutputBezier 和 OutputBspline 这两个虚函数，进行完运算后对照输入参数文件的格式将转换后的文件通过传入的 File 类写到文件里即可。

（B 样条曲线转换为贝塞尔曲线）



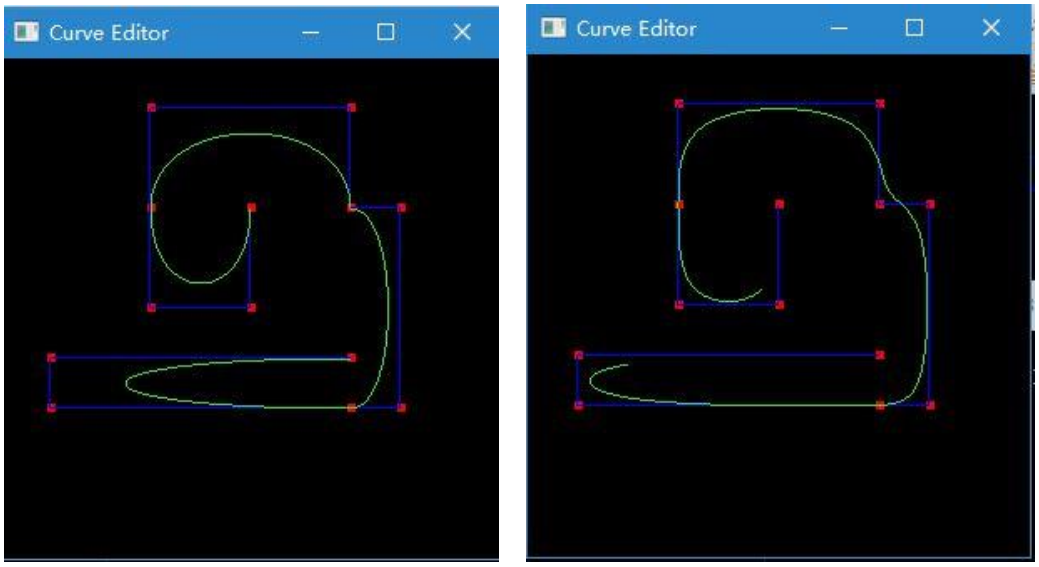
（贝塞尔曲线转换为 B 样条曲线）



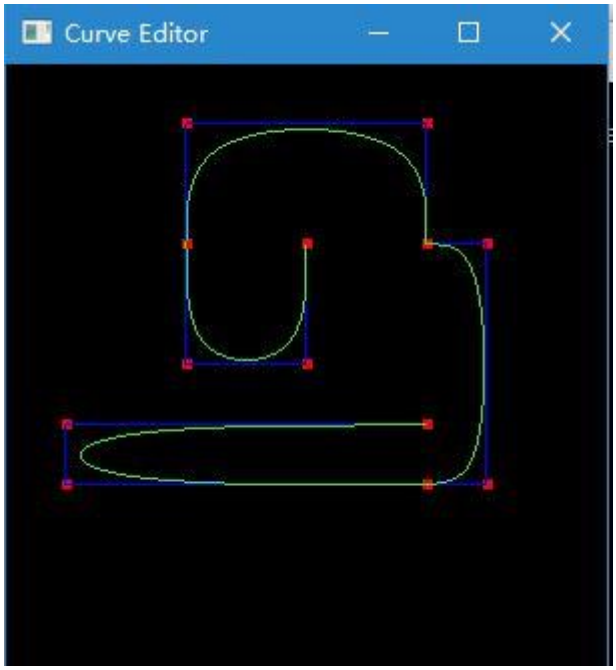
● 点的移动、添加和删除以及曲线的一般化

要实现点的移动、添加和删除只需要在基类 Spline 中实现 moveControlPoint, addControlPoint 和 deleteControlPoint 这三个函数即可。为了方便我在存储控制点的时候选择的方法是开大数组（100、1000….），在添加和移动的时候只要在指定下标的位置进行赋值或者修改操作即可，在删除控制点的时候需要进行数组的移位操作，保持控制点在数组中连续存储。

上面的试验中我们绘制的曲线都是四个控制点的，那么对于多于 4 个的怎么办呢？解决的方案是：对于贝塞尔曲线我们将 3n+1 个控制点一次绘制成 n 条首尾连接的贝塞尔曲线；对于 B 样条曲线，根据它的数学性质，我们依次绘制控制点中所有连续的四个控制点构成的 B 样条曲线，根据数学证明它们一定会连成一条曲线。要实现一般化，我们只需要对先前的 Paint 做一些修改即可。

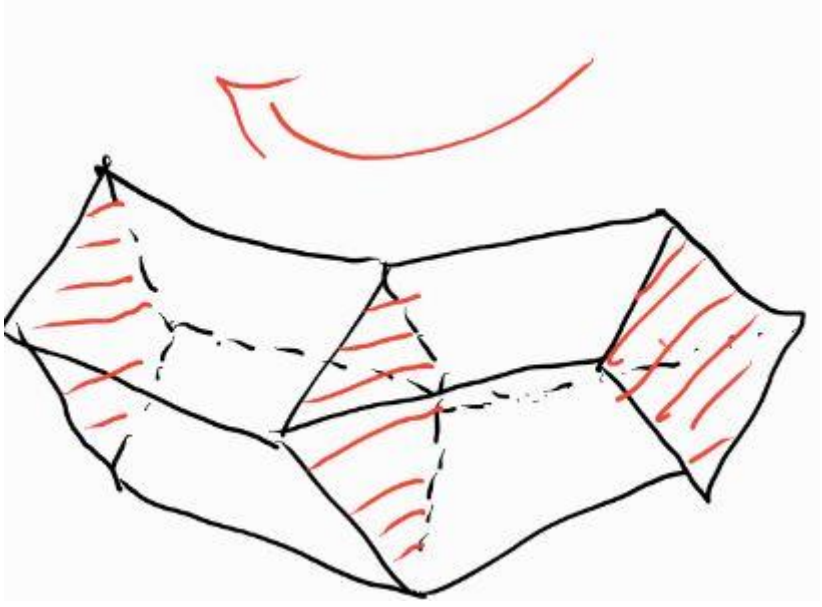


我们发现 B 样条曲线首尾是不与控制点相连接的，想要产生和贝塞尔曲线一样的效果我们可以在其首尾各添加三个控制点，分别与首尾控制点重合。



● 样条曲线旋转建模

旋转建模，很直观的想法就是把上面画好的样条曲线绕某个坐标轴旋转，把扫过的位置记录下来就成了模型了。在前面我们绘制曲线的时候使用到了曲线的细分度，这里在旋转的时候我们同样也需要一个这样一个细分度，在程序启动时的参数中我们可以看到 revolution_tessellation 这个参数，它就是上面所说的旋转细分度。假设它是 10，那么曲线没旋转 360/10 = 36 度我们就记录一次点的信息，由于曲线在绘制的过程中是由多个点连接而成，同样地，立体的模型模型也是由上述多个旋转面连接而成的，可以参考下图：



这样得到的模型是由无数的小矩形拟合而成，然后我们按照一定的规律将每一个矩形分成两个三角形就能得到模型的三角形网格。

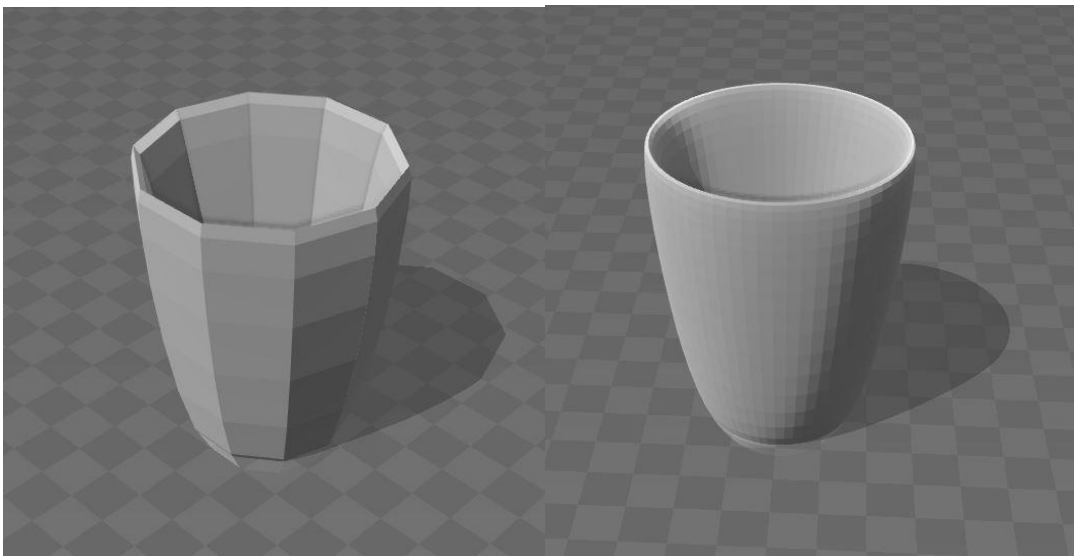
在程序实现的过程中我们需要使用 Matrix 类提供的 MakeAxisRotation 方法来替我们生成一个绕指定轴旋转的变换矩阵，接着我们对样条曲线的每一个点（我们通过细分度计

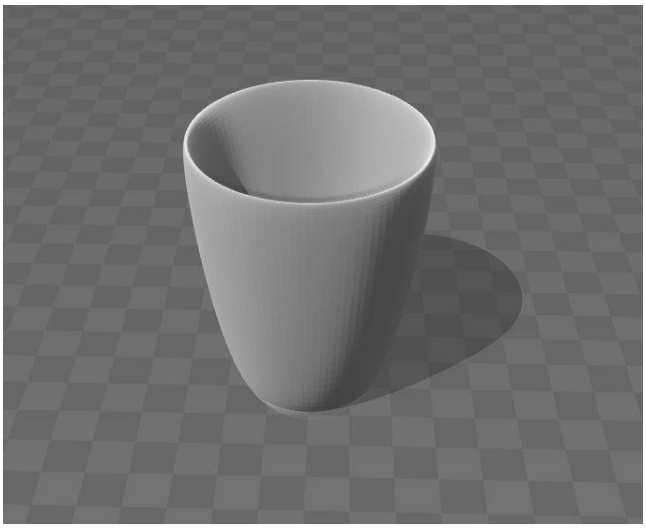
算出来的每一个点）进行矩阵变换得到新的点，这些点将会被加入到 TrangleMesh 类中，并且赋予编号。在声明 TrangleMesh 类的时候你需要根据给定的曲线细分度和曲面细分度计算出为了拟合模型需要多少个点和多少个三角形。在将所有的点加入 TrangleMesh 中后，我们需要使用它的 SetTriangle 方法设定三角形网格中每一个顶点的编号。然后我们在 OutputTriangles 这个函数中返回刚刚建立的 TrangleMesh 即可。

至于为什么这么做，可以参考下面这篇文章：

http://www.cnblogs.com/slysky/p/4081307.html

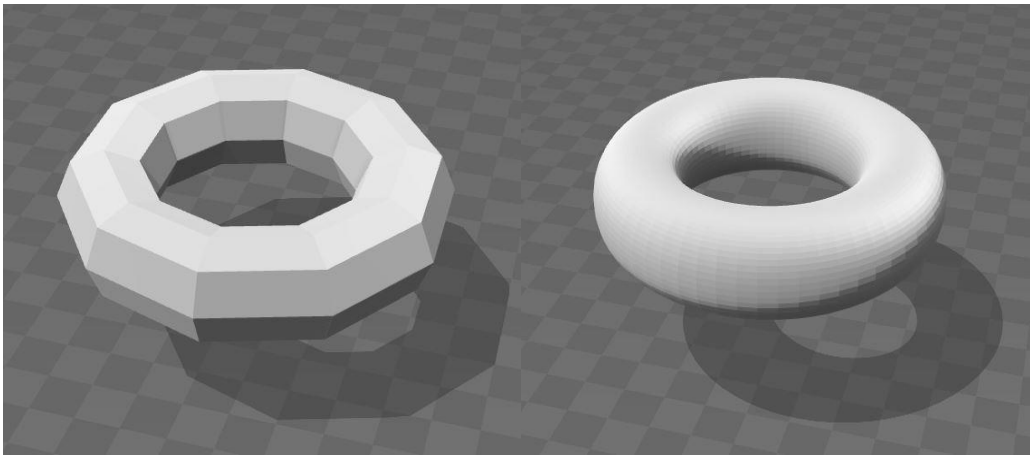
这篇文章讲解了 3D 模型 obj 文件的文件结构，看过之后你就会明白为什么之前要像上面那样做了。





上面的模型就是由同一个样条曲线生成，但是曲线和曲面的细分度不同，我们可以直观的感受到细分度越大，模型就越逼真。

下面的一组图能够更加明显的看出细分度对模型平滑度的影响。



● 贝塞尔曲面

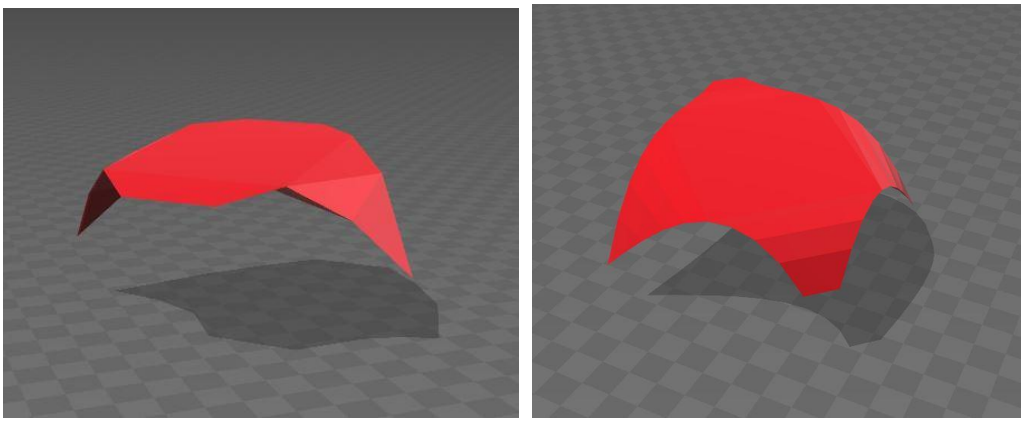
经过上面的阐述，绘制贝塞尔曲面也就没有上面别的问题了，下面给出 4*4 个控制下贝塞尔曲面的参数方程：

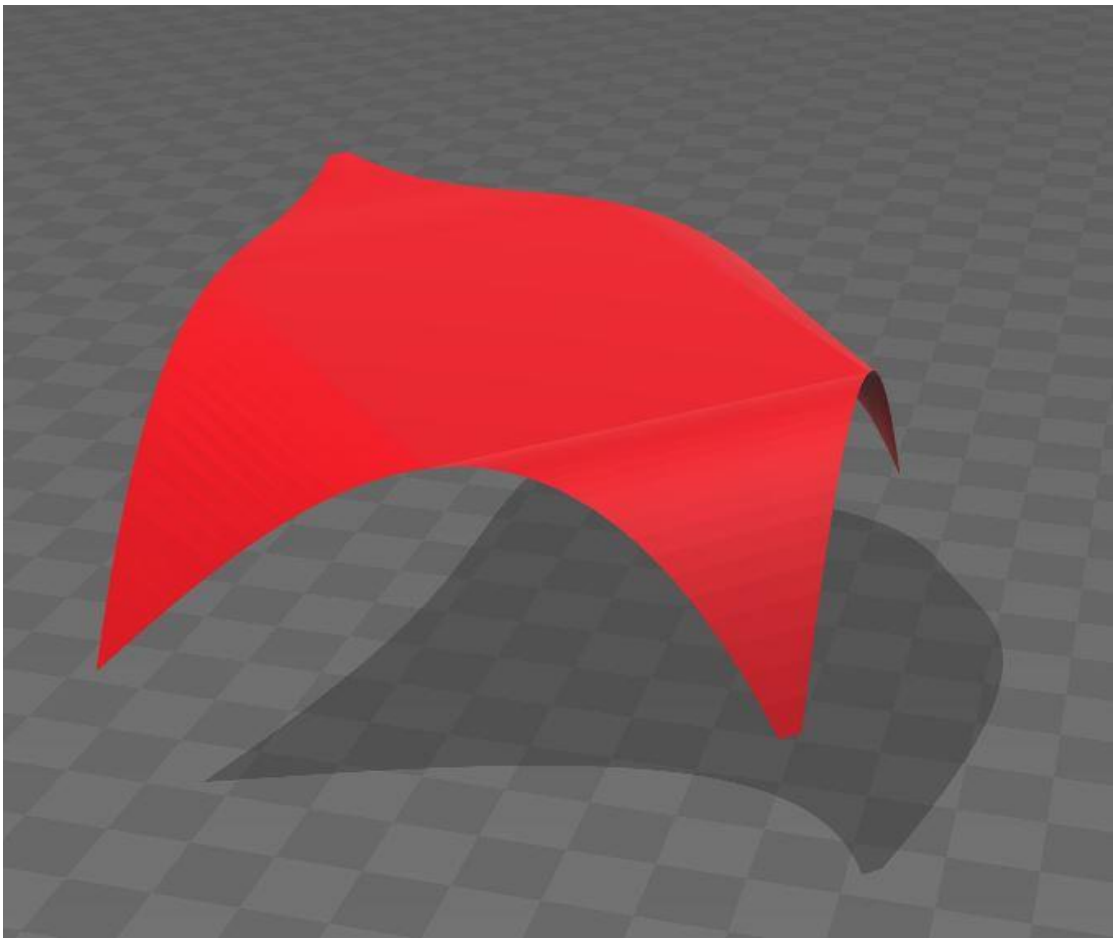
$$P(t) = B_1(u)P_1(v) + B_2(u)P_2(v) + B_3(u)P_3(v) + B_1(u)P_4(v)$$

$$P_i(v) = B_1(v)\overrightarrow{P_{i,1}} + B_2(v)\overrightarrow{P_{i,2}} + B_3(v)\overrightarrow{P_{i,3}} + B_1(v)\overrightarrow{P_{i,4}}$$

框架为我们提供了一个 TriangleNet 类，它继承自 TriangleMesh 类，通过阅读注释和源码，我们发现这个类负责专门处理 4*4 个控制点下的贝塞尔曲面的三角形网格的生成。我们要做的只是将细分度（patch_tessellation）作为参数构建 TriangleNet 类，然后依次将点添加进去就可以了。

结果如下图所示（细分度逐渐提高）：





最后，综合旋转曲面和贝塞尔曲面，我们使用试验提供的数据构建了下面这个茶壶的模型：