Geometry

一、 Bezier 曲线

1.1 圆的绘制与曲线原理

Bezier 曲线通过控制点来定义，控制点决定了曲线的形状，n 次贝塞尔曲线需要 n+1 个控制点。它使用线性插值计算控制点间的过度，对于某点 P(t) 在区间 [0,1] 上，它由若干控制点通过逐步线性插值得到。每个 Bezier 线可以通过多个子曲线表示。每个子曲线的控制点是原始控制点的线性插值。

下面是利用 Bezier 曲线绘制圆的简单叙述。在绘制圆的过程中为简便起见，使用三次 Bezier 曲线。三次 Bezier 曲线的参数化方程如下：

$$B(t) = (1-t)^3 P_0 + 3(1-t)^2 t P_1 + 3(1-t)t^2 P_2 + t^3 P_3$$

其中，P_0, P_1, P_2, P_3 是控制点，$t \in [0,1]$

代码的核心思想是通过多个 Bezier 曲线段来近似表示圆形，即使用多个三次 Bezier 曲线段逐渐逼近圆的形状。我们将圆均分分割成若干个点来生成控制点。这些控制点位于圆的周长上，形成一个多边形，圆的半径和分段数目决定了控制点的位置和数量。控制点的坐标计算如下：

$$P_i = (rcos(\frac{2\pi i}{n}), rsin(\frac{2\pi i}{n}))$$

```python
def generate_control_points(radius, num_segments=8):
  control_points = []
  for i in range(num_segments):
    angle = 2 * np.pi * i / num_segments
    control_points.append([radius * np.cos(angle), radius * np.sin(angle)])
  return np.array(control_points)
```

每 4 个相邻的控制点构成一段三次 Bezier 曲线。每个段落选取 4 个控制点，其中 P_0 是当前控制点，P_1, P_2 是当前控制点和下一个控制点间的中间点，取这两个点的平均位置来平滑过渡，P_3 是下下个控制点。

1.2 使用 de Casteljau 算法绘制与结果图像

de Casteljau 算法：给定 4 个控制点 P_0, P_1, P_2, P_3 和一个参数 t, 通过如下递归计算曲线上一点：

1. 线性插值：

$$Q_0 = (1-t)P_0 + tP_1, \quad Q_1 = (1-t)P_1 + tP_2, \quad Q_2 = (1-t) * p_2 + t * P_3$$

2. 再一次线性插值：

$$R_0 = (1-t)Q_0 + tQ_1, \quad R_1 = (1-t) * Q_1 + t * Q_2$$

3. 再一次插值，计算 R:

$$R = (1-t) * R_0 + t * R_1$$

最终，R 就是 Bezier 曲线上的点，随着 t 的变化，R(t) 的轨迹就是 Bezier 曲线。

```
def de_casteljau(P0, P1, P2, P3, t):
  Q0 = (1 - t) * P0 + t * P1
  Q1 = (1 - t) * P1 + t * P2
  Q2 = (1 - t) * P2 + t * P3
  R0 = (1 - t) * Q0 + t * Q1
  R1 = (1 - t) * Q1 + t * Q2
  R = (1 - t) * R0 + t * R1
  return R
```

```
def bezier_curve(P0, P1, P2, P3, num_points=100):
  curve_points = []
  for t in np.linspace(0, 1, num_points):
    point = de_casteljau(P0, P1, P2, P3, t)
    curve_points.append(point)
  return np.array(curve_points)
```

获得的最终效果图如下：

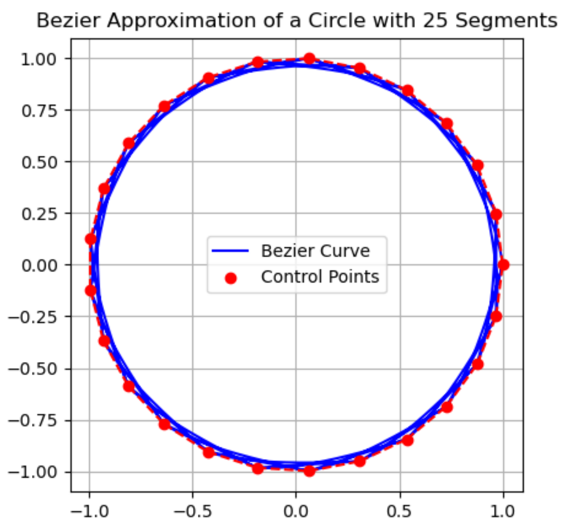

图 1 **Bezier Curve**

二、 Utah Teapot 曲面

2.1 数学原理

Bezier 曲面的构造是对 Bezier 曲线的扩展。在三维空间中，Bezier 曲面由一个控制点网格（4x4 矩阵）定义，类似于 Bezier 曲线由若干控制点定义。三次 Bezier 曲面上的一个点可以通过两个参数 u 和 v 来描述，这两个参数分别对应于 x 和 y 方向。

Bezier 曲面的数学公式如下：

$$S(u,v)=\sum_{i=0}^{3}\sum_{i=0}^{3}\binom{3}{i}\binom{3}{j}u^i(1-u)^{3-i}v^j(1-v)^{3-j}P_{i,j}$$

其中：

- S(u,v) 表示曲面上的点，u 和 v 是曲面在两个方向上的参数。
- $P_{i,j}$ 是控制点矩阵中的元素，表示曲面上的一个控制点。
- 组合数 $\binom{3}{i},\binom{3}{j}$ 是沿 u 和 v 方向的权重系数。
- 参数 u，v 的取值范围是 [0,1]。

2.2 代码实现与效果图

1. 组合数

```
from math import factorial
def comb(n, k):
  return factorial(n) // (factorial(k) * factorial(n - k))
```

2.Bezier 基函数

```
def bezier_basis(t, degree=3):
  return [comb(degree, i) * (t ** i) * ((1 - t) ** (degree - i)) for i in range(degree + 1)]
```

3.Bezier 曲面计算

```
def bezier_surface(control_points, u, v):
  Bu = bezier_basis(u)
  Bv = bezier_basis(v)
  point = np.zeros(3)
  for i in range(4):
    for j in range(4):
      point += control_points[i, j] * Bu[i] * Bv[j]
  return point
```

4. 绘制曲面

```
def plot_teapot(teapot_vertices, teapot_patches, u_steps=20, v_steps=20):
```

```
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

u_vals = np.linspace(0, 1, u_steps)
v_vals = np.linspace(0, 1, v_steps)

for patch in teapot_patches:
   control_points = np.array([teapot_vertices[i - 1] for i in patch]).reshape((4, 4, 3))

   X, Y, Z = [], [], []
   for u in u_vals:
      for v in v_vals:
         point = bezier_surface(control_points, u, v)
         X.append(point[0])
         Y.append(point[1])
         Z.append(point[2])

ax.plot_trisurf(X, Y, Z, color='cyan', linewidth=0.1, alpha=0.5)

plt.show()
```

5. 效果图

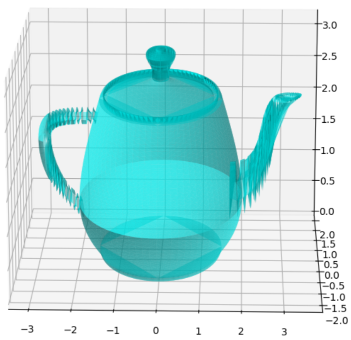

图 2　**Utah Teapot 1**

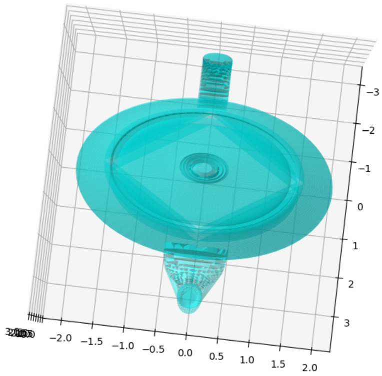

图 3　**Utah Teapot 2**

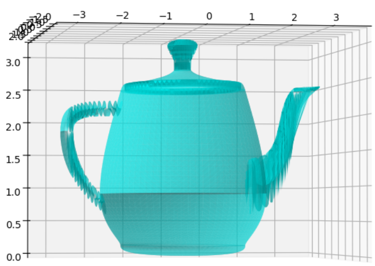

图 4　**Utah Teapot 3**