HW5 细分曲线的生成

一、 算法分析与实现

本次作业主要研究如何以一个简单多边形为输入，生成一条与之关联的光滑曲线的问题，即曲线的细分问题。作业中以输入一个由相邻型值点连接而成的封闭多边形为例，实现了 3 种不同的曲线细分方法，包括 2 种逼近型细分和 1 种插值型细分法：

1. Chaikin 割角法（逼近型）

   Chaikin 割角法通过对原多边形顶点的坐标进行线性组合，每次细分都将生成约 2 倍数目的新顶点并舍弃旧顶点（逼近），多次迭代后，即可得到一条光滑的曲线。可以证明，该方法的极限情况是一条二次均匀 B 样条曲线，其拓扑规则相当于“割角”，每次细分的算法如下：

$$\begin{cases} v'_{2i} = \dfrac{1}{4}v_{i-1} + \dfrac{3}{4}v_i \\ v'_{2i+1} = \dfrac{3}{4}v_i + \dfrac{1}{4}v_{i+1} \end{cases}$$

   具体代码实现截取如下：

```
for (int i = 0; i < input_points.size(); i++) {
    if (i > 0) {
        output_points[2 * i][0] = 0.25f * input_points[i - 1][0] + 0
.75f * input_points[i][0];
        output_points[2 * i][1] = 0.25f * input_points[i - 1][1] + 0
.75f * input_points[i][1];
    }
    else
    {
        output_points[2 * i][0] = 0.25f * input_points[input_points.
size() - 1][0] + 0.75f * input_points[i][0];
```

```
        output_points[2 * i][1] = 0.25f * input_points[input_points.size() - 1][1] + 0.75f * input_points[i][1];
    }
    if (i < input_points.size() - 1) {
        output_points[2 * i + 1][0] = 0.75f * input_points[i][0] + 0.25f * input_points[i + 1][0];
        output_points[2 * i + 1][1] = 0.75f * input_points[i][1] + 0.25f * input_points[i + 1][1];
    }
    else {
        output_points[2 * i + 1][0] = 0.75f * input_points[i][0] + 0.25f * input_points[0][0];
        output_points[2 * i + 1][1] = 0.75f * input_points[i][1] + 0.25f * input_points[0][1];
    }
}
```

2. 三次均匀 B 样条曲线细分方法（逼近型）

与 Chaikin 细分类似，作为一种逼近型细分方法，该方法的极限情况是一条三次均匀 B 样条曲线，其拓扑规则相当于将每条边“分裂”，每次细分的算法如下：

$$\begin{cases} v'_{2i} = \dfrac{1}{8}v_{i-1} + \dfrac{3}{4}v_i + \dfrac{1}{8}v_{i+1} \\ v'_{2i+1} = \dfrac{1}{2}v_i + \dfrac{1}{2}v_{i+1} \end{cases}$$

具体代码实现截取如下：

```
for (int i = 0; i < input_points.size(); i++) {
    if (i > 0 && i < input_points.size() - 1) {
        output_points[2 * i][0] = 0.125f * input_points[i - 1][0] + 0.75f * input_points[i][0] + 0.125f * input_points[i + 1][0];
        output_points[2 * i][1] = 0.125f * input_points[i - 1][1] + 0.75f * input_points[i][1] + 0.125f * input_points[i + 1][1];
    }
    else if (i == 0)
    {
        output_points[2 * i][0] = 0.125f * input_points[input_points.size() - 1][0] + 0.75f * input_points[i][0] + 0.125f * input_points[i + 1][0];
```

```
        output_points[2 * i][1] = 0.125f * input_points[input_points.size() - 1][1] + 0.75f * input_points[i][1] + 0.125f * input_points[i + 1][1];
    }
    else {
        output_points[2 * i][0] = 0.125f * input_points[i - 1][0] + 0.75f * input_points[i][0] + 0.125f * input_points[0][0];
        output_points[2 * i][1] = 0.125f * input_points[i - 1][1] + 0.75f * input_points[i][1] + 0.125f * input_points[0][1];
    }
    if (i < input_points.size() - 1) {
        output_points[2 * i + 1][0] = 0.5f * input_points[i][0] + 0.5f * input_points[i + 1][0];
        output_points[2 * i + 1][1] = 0.5f * input_points[i][1] + 0.5f * input_points[i + 1][1];
    }
    else {
        output_points[2 * i + 1][0] = 0.5f * input_points[i][0] + 0.5f * input_points[0][0];
        output_points[2 * i + 1][1] = 0.5f * input_points[i][1] + 0.5f * input_points[0][1];
    }
}
```

3. 4 点细分方法（插值型）

与上述逼近型细分不同，4 点细分方法在保留旧顶点的基础上，对每条边都增加了 1 个新顶点（插值），其拓扑规则相当于“补角”，每次细分的算法如下：

$$\begin{cases} v'_{2i} = v_i \\ v'_{2i+1} = \dfrac{v_i + v_{i+1}}{2} + \alpha\left(\dfrac{v_i + v_{i+1}}{2} - \dfrac{v_{i-1} + v_{i+2}}{2}\right) \end{cases}$$

其中，参数$\alpha \in (0, \frac{1}{8})$时，可以生成光滑的细分曲线；否则将生成不光滑的分形曲线。具体代码实现截取如下：

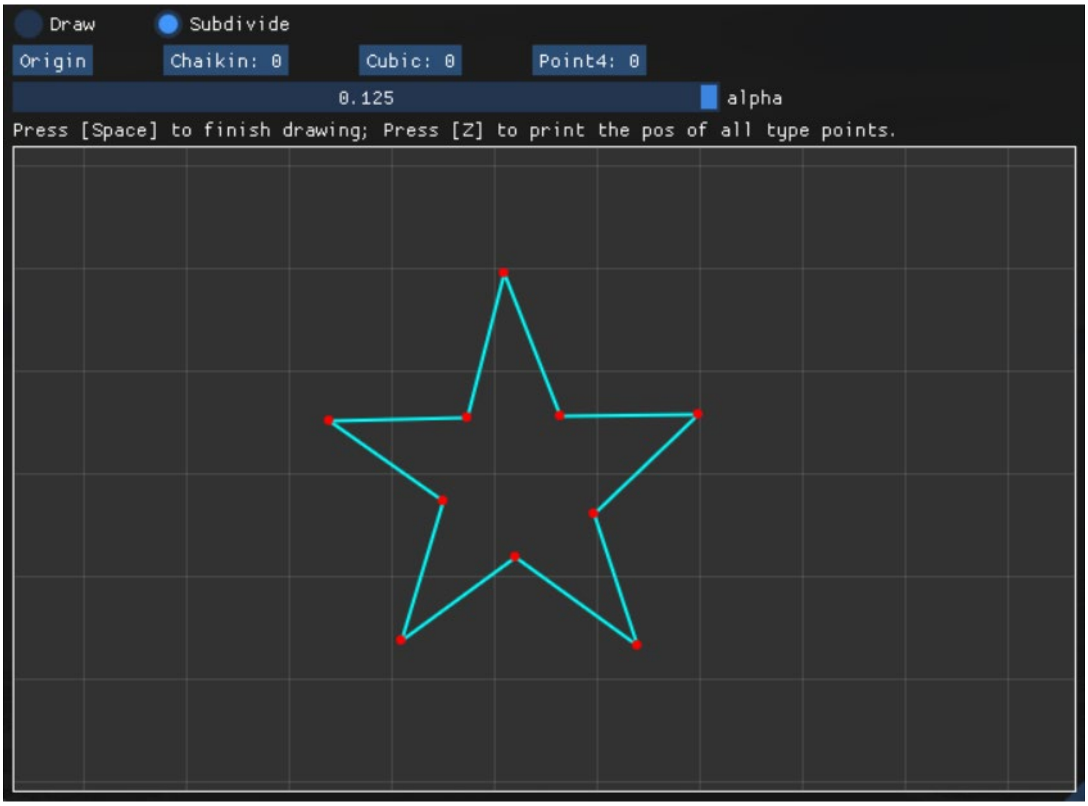
```
for (int i = 0; i < input_points.size(); i++) {
    output_points[2 * i] = input_points[i];
    if (i > 0 && i < input_points.size() - 2) {
```

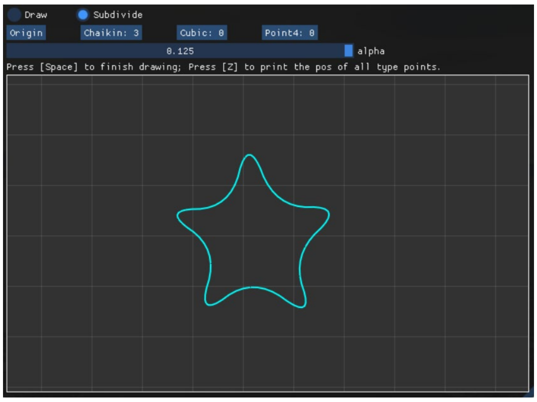
```
        output_points[2 * i + 1][0] = (1.0f + alpha) / 2.0f * (input_points[i][0] + input_points[i + 1][0]) - alpha / 2.0f * (input_points[i - 1][0] + input_points[i + 2][0]);
        output_points[2 * i + 1][1] = (1.0f + alpha) / 2.0f * (input_points[i][1] + input_points[i + 1][1]) - alpha / 2.0f * (input_points[i - 1][1] + input_points[i + 2][1]);
    }
    else if (i == 0)
    {
        output_points[2 * i + 1][0] = (1.0f + alpha) / 2.0f * (input_points[i][0] + input_points[i + 1][0]) - alpha / 2.0f * (input_points[input_points.size() - 1][0] + input_points[i + 2][0]);
        output_points[2 * i + 1][1] = (1.0f + alpha) / 2.0f * (input_points[i][1] + input_points[i + 1][1]) - alpha / 2.0f * (input_points[input_points.size() - 1][1] + input_points[i + 2][1]);
    }
    else if (i == input_points.size() - 2) {
        output_points[2 * i + 1][0] = (1.0f + alpha) / 2.0f * (input_points[i][0] + input_points[i + 1][0]) - alpha / 2.0f * (input_points[i - 1][0] + input_points[0][0]);
        output_points[2 * i + 1][1] = (1.0f + alpha) / 2.0f * (input_points[i][1] + input_points[i + 1][1]) - alpha / 2.0f * (input_points[i - 1][1] + input_points[0][1]);
    }
    else {
        output_points[2 * i + 1][0] = (1.0f + alpha) / 2.0f * (input_points[i][0] + input_points[0][0]) - alpha / 2.0f * (input_points[i - 1][0] + input_points[1][0]);
        output_points[2 * i + 1][1] = (1.0f + alpha) / 2.0f * (input_points[i][1] + input_points[0][1]) - alpha / 2.0f * (input_points[i - 1][1] + input_points[1][1]);
    }
}
```

二、 实验分析与结论

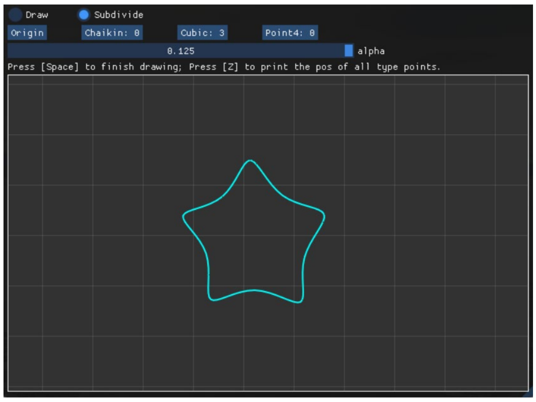
1. 实验结果

以下各图示出了对 3 种多边形分别采用以上 3 种方法进行细分处理的结果:

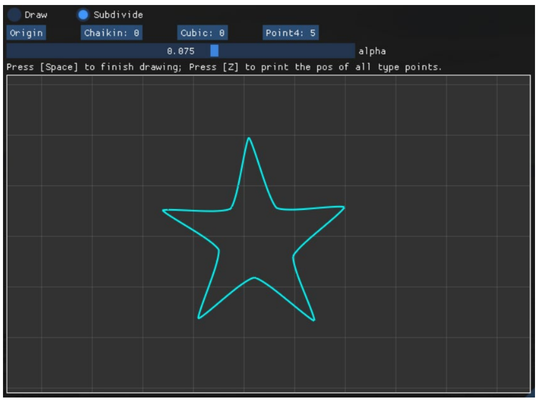




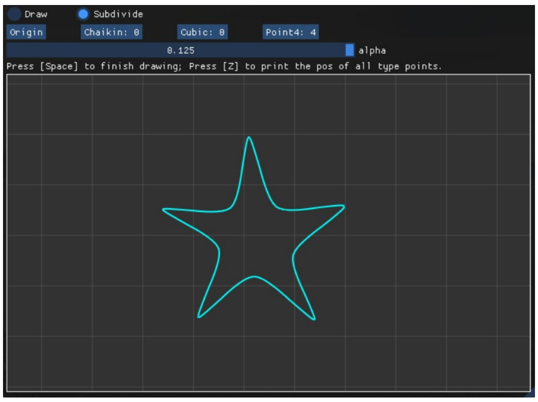
Chaikin 细分 3 次



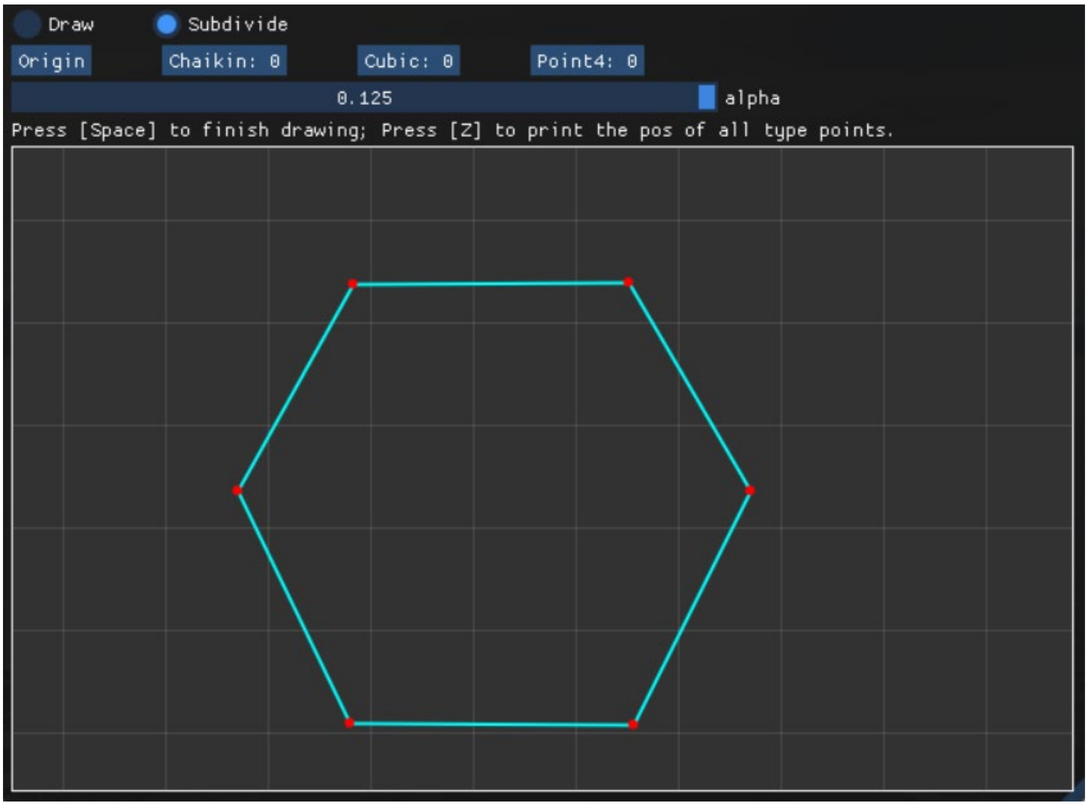
三次均匀 B 样条细分 3 次

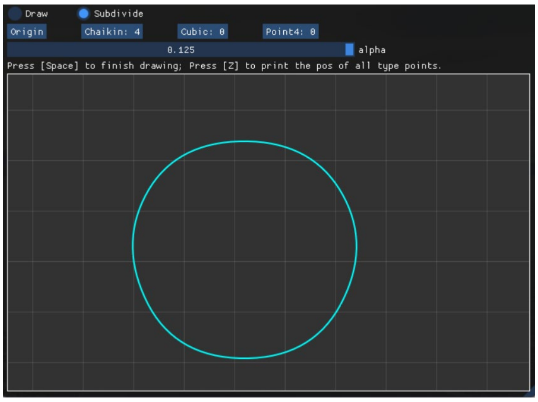


4 点细分 5 次， $\alpha = 0.075$

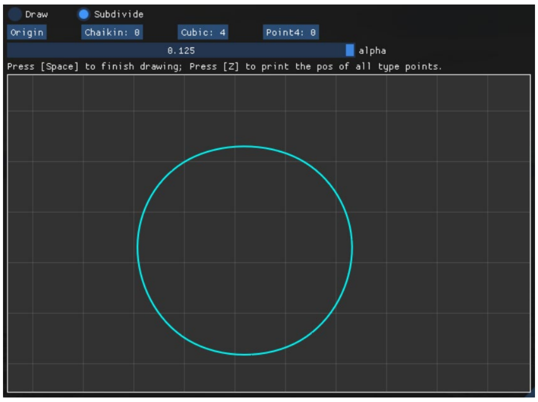


4 点细分 4 次， $\alpha = 0.125$

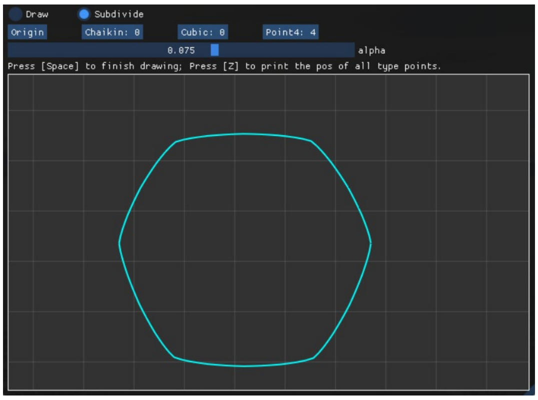




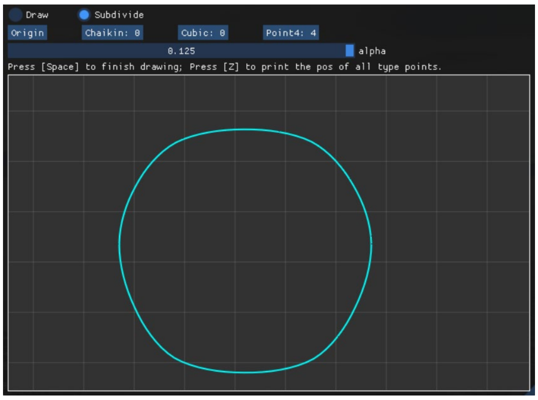
Chaikin 细分 4 次



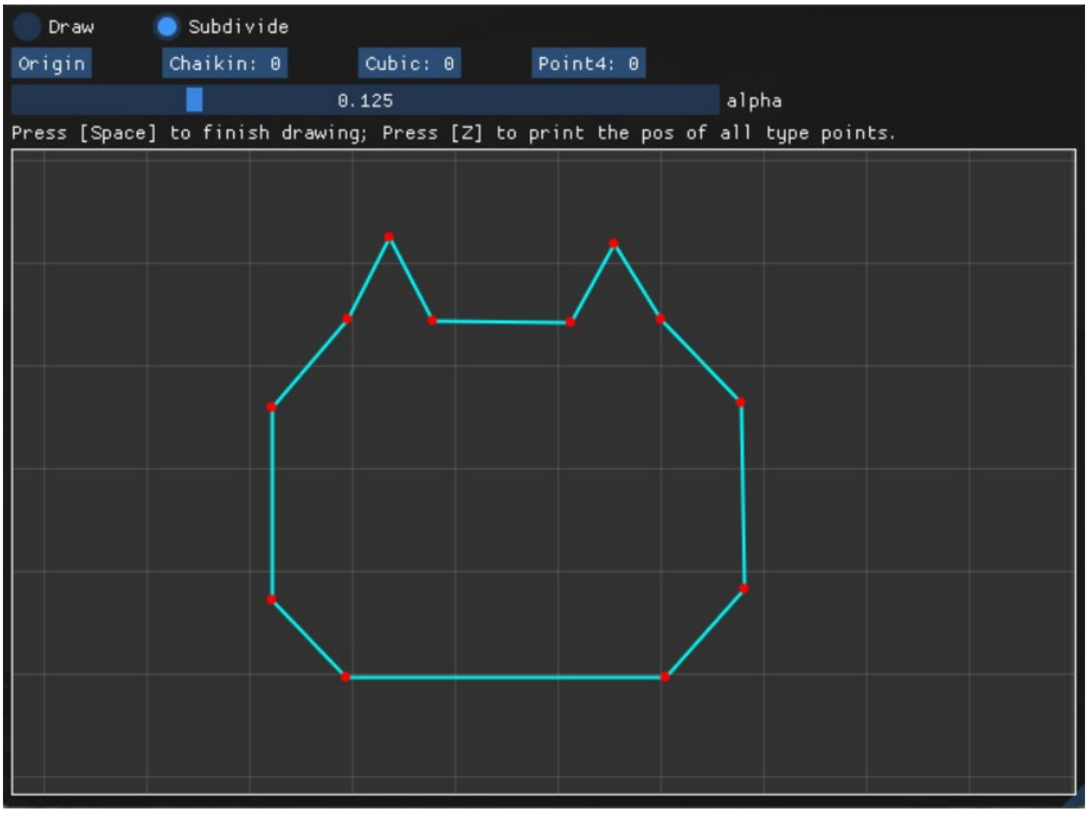
三次均匀 B 样条细分 4 次

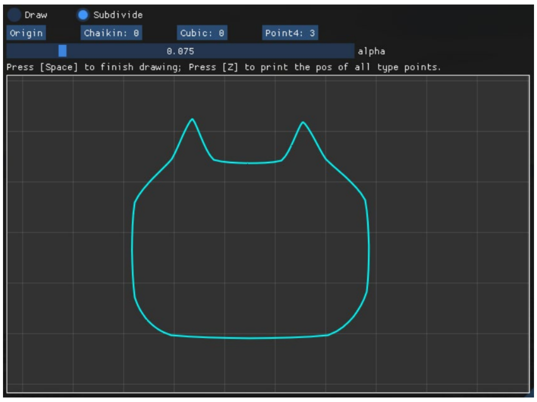


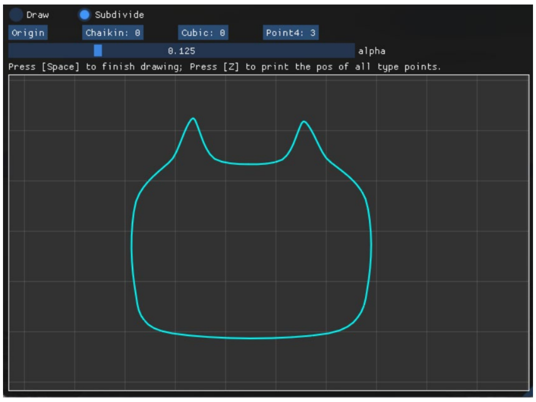
4 点细分 4 次，$\alpha = 0.075$



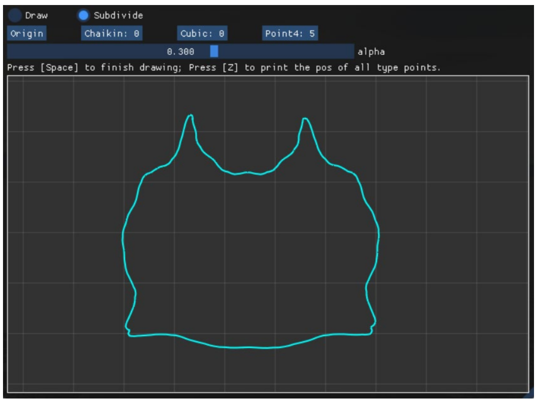
4 点细分 4 次，$\alpha = 0.125$

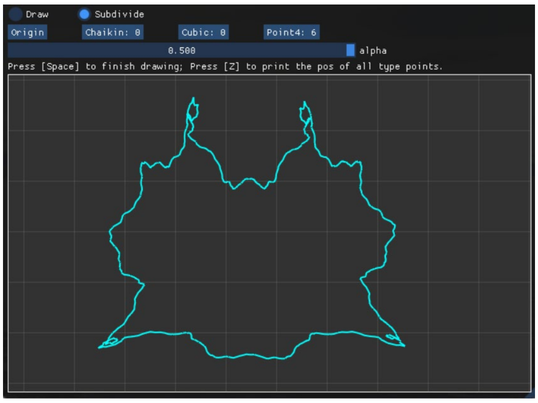






4 点细分 3 次，$\alpha = 0.075$　　4 点细分 3 次，$\alpha = 0.125$





4 点细分 5 次，$\alpha = 0.300$　　4 点细分 6 次，$\alpha = 0.500$

2. 结论

   a) 对于任意给定的多边形，以上三种方法无论是逼近型或插值型细分，在经过一定次数的迭代后，都能生成非常光滑的曲线，非顶点处的连续性甚至能达到C^{∞}；

   b) 可以看到，对于插值型 4 点细分法，参数α取值越大，多次迭代后生成的细分曲线越光滑，连续性越好；但当$\alpha > \frac{1}{8}$时，生成的曲线将不再光滑，甚至出现分形。