GAMES 102 - 作业 5

彭博

November 21, 2020

1. 本次作业实现逼近型和插值型两种曲线细分方法如 Fig.1所示。

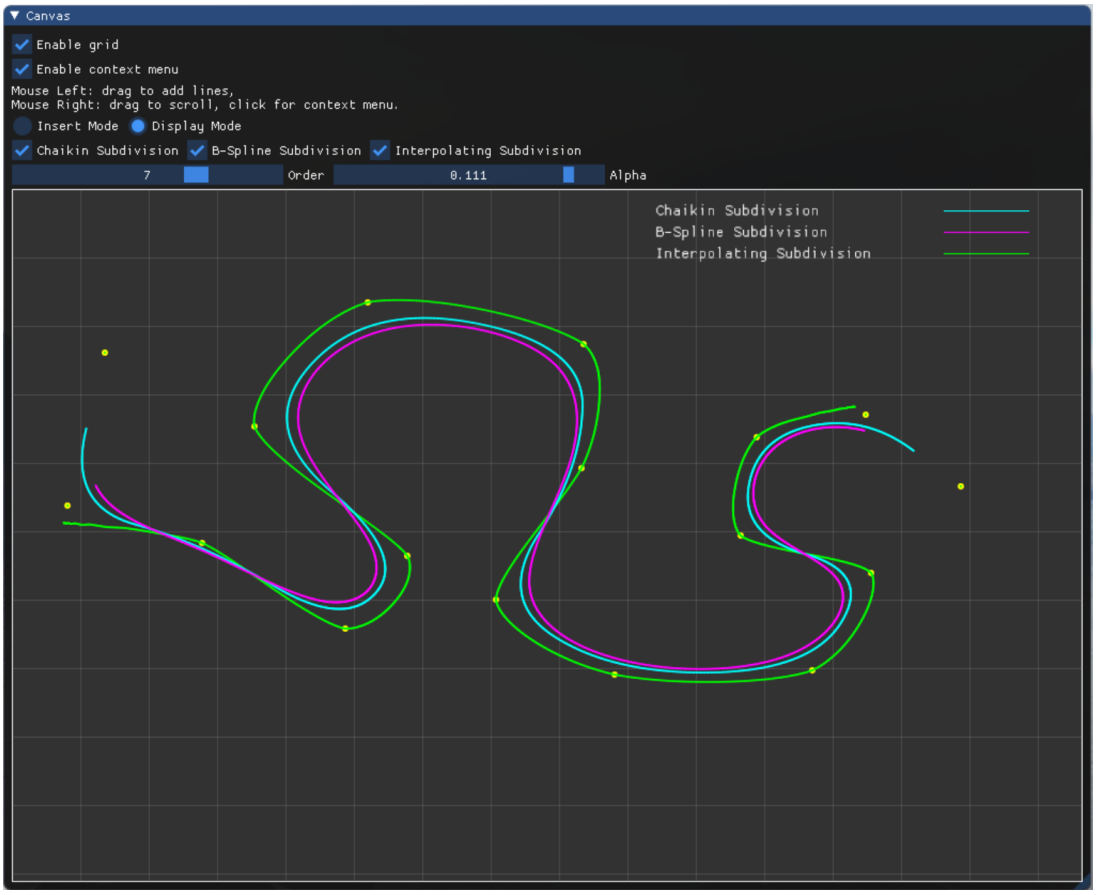

Figure 1: 曲线细分

逼近型细分方法包括 Chaikin 方法（二次 B 样条）以及三次 B 样条细分方法，两种算法实现流程如下：

(a) Chaikin 方法:

i. 每条边取中点，生成新点;
ii. 每个点与其相邻点平均;
iii. 迭代生成曲线。

使用 Chaikin 方法生成细分曲线代码可参考 Listing 1，曲线如 Fig.2所示。

(b) 三次 B 样条细分:

i. 插入新点点

$$v'_{2i} = \frac{1}{8}v_{i-1} + \frac{3}{4}v_i + \frac{1}{8}v_{i+1} \tag{1}$$

ii. 插入新边点

$$v'_{2i+1} = \frac{1}{2}v_i + \frac{1}{2}v_{i+1} \tag{2}$$

Listing 1 Chaikin 曲线细分

```
std::vector<pointf2> chaikin(std::vector<pointf2> points, int order) {
        if (points.size() < 2 || order == 0) return points;
        std::vector<pointf2> subdiv, _points;

        // split
        for (size_t i = 0; i+1 < points.size(); i++)
        {
                float px = (points[i][0] + points[i + 1][0]) / 2;
                float py = (points[i][1] + points[i + 1][1]) / 2;

                subdiv.push_back(pointf2(px, py));
        }

        for (size_t i = 0; i < subdiv.size(); i++)
        {
                points.insert(points.begin() + (2 * i + 1), subdiv[i]);
        }

        // average
        for (size_t i = 0; i+1 < points.size(); i++)
        {
                float px = (points[i][0] + points[i+1][0]) / 2;
                float py = (points[i][1] + points[i+1][1]) / 2;

                _points.push_back(pointf2(px, py));
        }

        return chaikin(_points, order - 1);
}
```

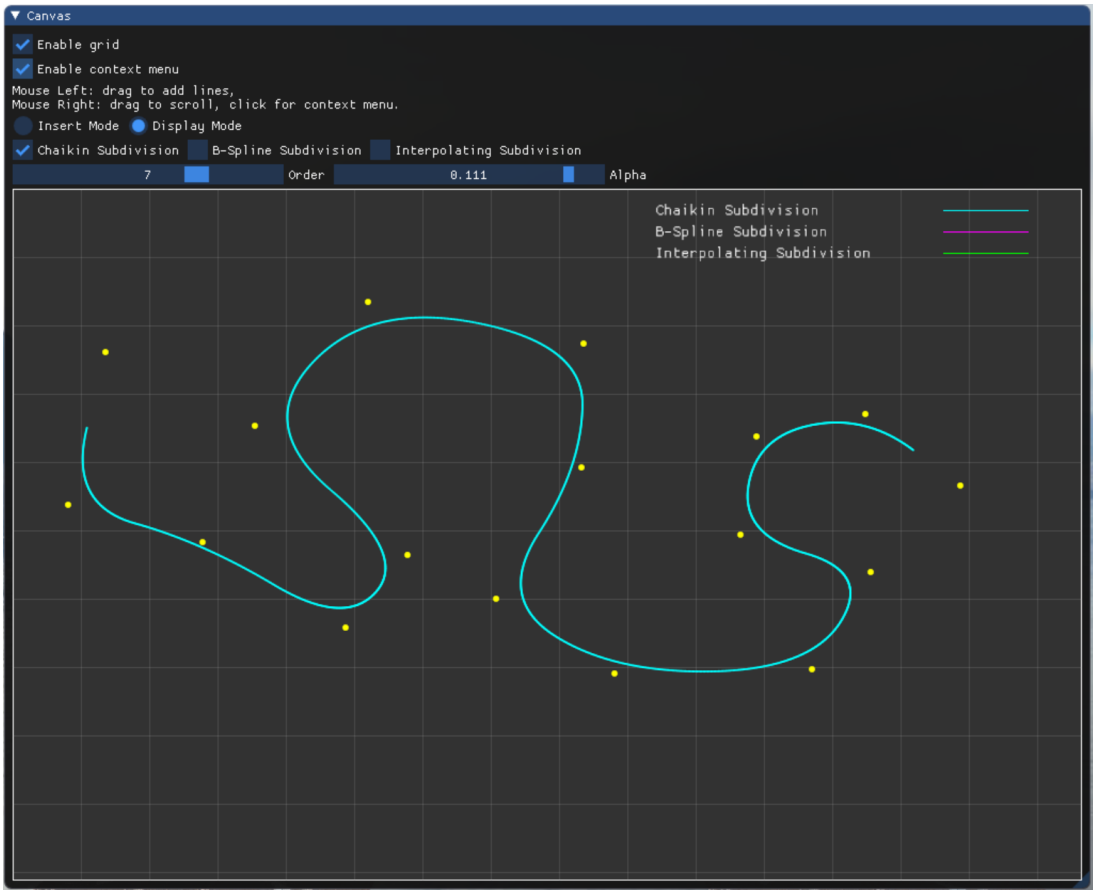

Figure 2: Chaikin 曲线细分

Listing 2 三次 B 样条曲线细分

```
std::vector<pointf2> bsplinesubdiv(std::vector<pointf2> points, int order) {
        if (points.size() < 3 || order == 0) return points;
        std::vector<pointf2> subdiv, _points;

        // split
        for (size_t i = 1; i + 1 < points.size(); i++)
        {
                float px = (points[i - 1][0] + 6 * points[i][0] + points[i + 1][0]) / 8;
                float py = (points[i - 1][1] + 6 * points[i][1] + points[i + 1][1]) / 8;

                subdiv.push_back(pointf2(px, py));
        }

        // average
        for (size_t i = 0; i + 1 < points.size(); i++)
        {
                float px = (points[i][0] + points[i + 1][0]) / 2;
                float py = (points[i][1] + points[i + 1][1]) / 2;

                _points.push_back(pointf2(px, py));
        }

        for (size_t i = 0; i < subdiv.size(); i++)
        {
                _points.insert(_points.begin() + (2 * i + 1), subdiv[i]);
        }

        return bsplinesubdiv(_points, order - 1);
}
```

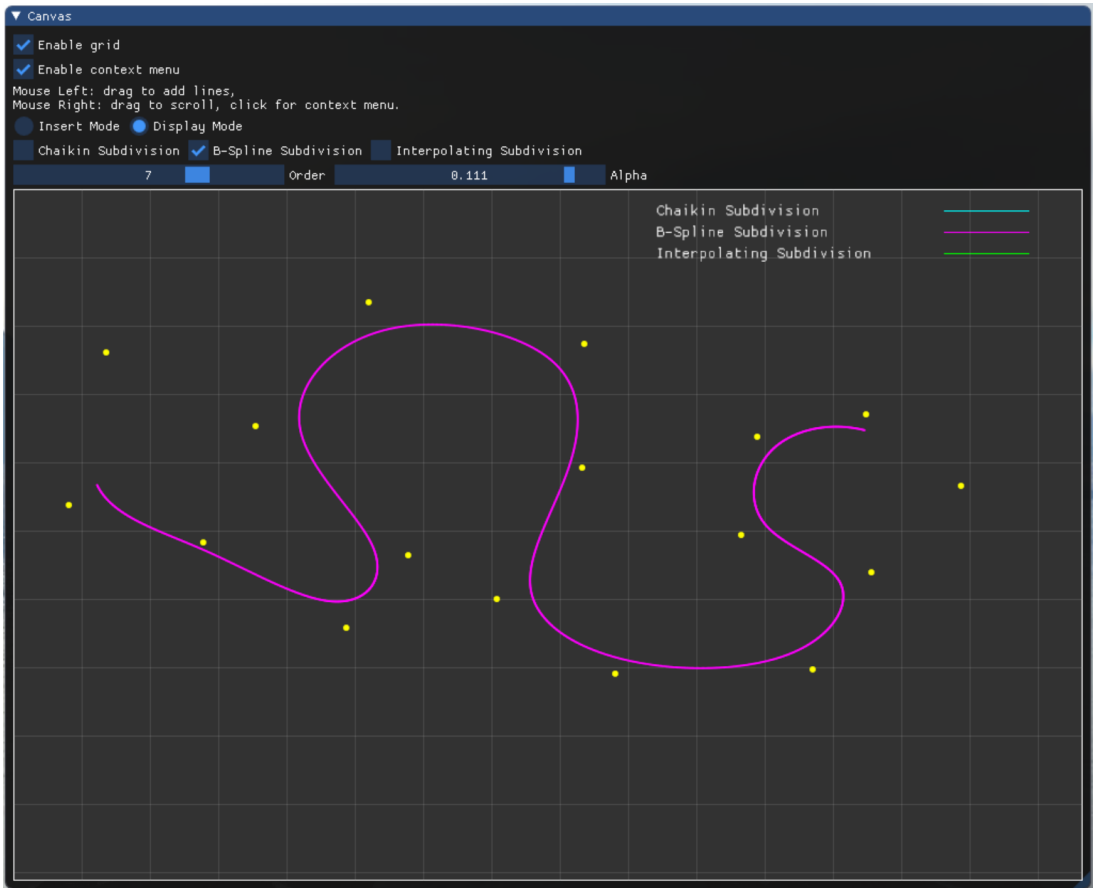

Figure 3: 三次 B 样条曲线细分

Listing 3 插值曲线细分

```
std::vector<pointf2> interpolatesubdiv(std::vector<pointf2> points, int order, float alpha) {
        if (points.size() < 4 || order == 0) return points;

        std::vector<pointf2> subdiv, _points;
        _points = points;
        for (size_t i = 1; i+2 < points.size(); i++)
        {
                float px = (points[i][0] + points[i + 1][0]) / 2 + alpha * ((points[i][0] + points[i + 1][0]) / 2 - (points[i - 1][0]
                float py = (points[i][1] + points[i + 1][1]) / 2 + alpha * ((points[i][1] + points[i + 1][1]) / 2 - (points[i - 1][1]

                subdiv.push_back(pointf2(px, py));
        }

        for (size_t i = 0; i < subdiv.size(); i++)
        {
                _points.insert(_points.begin()+(2*i+2), subdiv[i]);
        }

        return interpolatesubdiv(_points, order - 1, alpha);
}
```

iii. 迭代生成曲线。

使用三次 B 样条方法生成细分曲线代码可参考 Listing 2，曲线如 Fig.3所示。

插值型细分方法实现流程如下：

(a) 利用相邻 4 个点插入一个新节点：

$$p'_{2i+1} = \frac{p_i + p_{i+1}}{2} + \alpha(\frac{p_i + p_{i+1}}{2} + \frac{p_{i-1} + p_{i+2}}{2}) \tag{3}$$

(b) 迭代生成曲线。

使用插值方法生成细分曲线代码可参考 Listing 3，曲线如 Fig.4所示。同时需要说明的是系数 α 可以控制细分曲线的光滑程度，当 α 趋于 0 时细分曲线会趋于线性插值如 Fig.5所示，而当 α 逐渐增大时细分曲线会更加光滑。

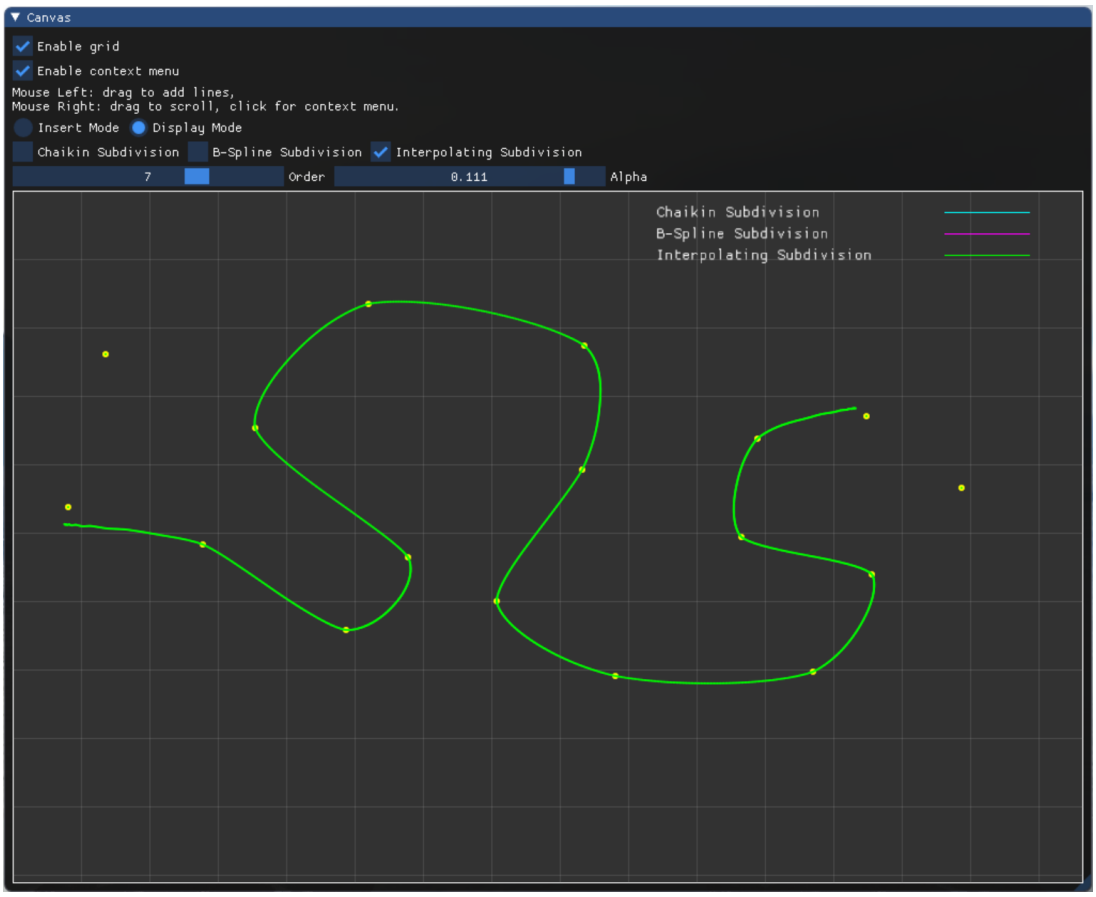

Figure 4: 插值曲线细分

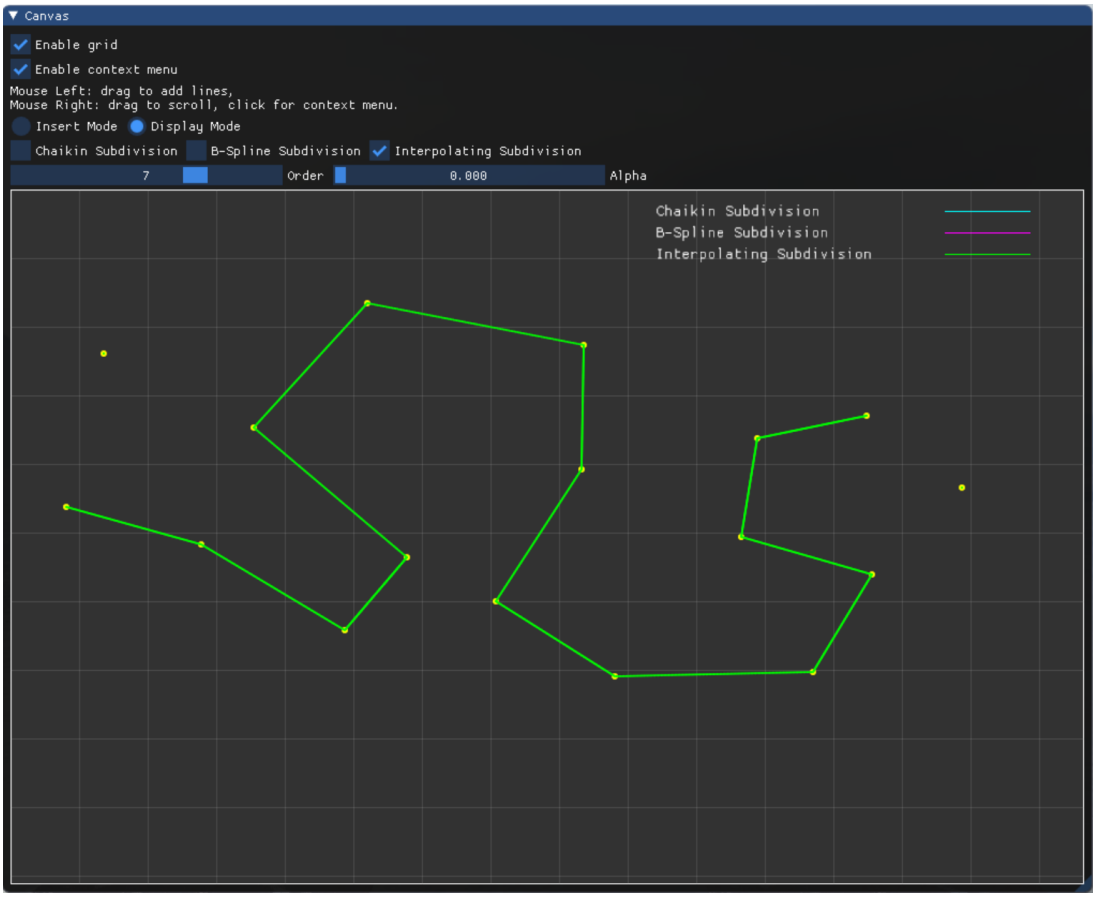

Figure 5: 插值曲线细分 ($\alpha = 0$)