## 题目名：

边缘填充算法

## 算法原理

边缘填充算法的基本思想是，逐边向右求补。可以按任意顺序处理多边形的每条边，在处理每条边时，首先求出该边与扫描线的交点，然后将每一条扫描线上交点右方的所有像素取补，多边形的所有边处理完毕之后，填充即完成

## 实现

### 1.相关数据结构:

储存所有的输入点即可

### 2.函数模块说明

EdgeFilling 边缘填充

DDA 直线实现的DDA算法

### 3.核心代码

// 边缘填充

void ImageRenderer::EdgeFilling(){

points[m\_cPointCount + 1] = points[0];

int x0, y0, x1, y1;

// 查找最小值

// 这不是必须的, 优化的话在PushPoint那里可以优化(记录/更新这些点)

int right = static\_cast<int>(std::max\_element(points, points + m\_cPointCount + 1,

[](D2D1\_POINT\_2F& p1, D2D1\_POINT\_2F& p2)->bool{return p1.x < p2.x; }

)->x);

//

int dx, dy, eps1, k, p, q, flag, temp;

float x, y, xIncre, yIncre;

for (size\_t i = 0; i <= m\_cPointCount; i++) {

x0 = to\_int(points[i].x);

y0 = to\_int(points[i].y);

x1 = to\_int(points[i + 1].x);

y1 = to\_int(points[i + 1].y);

if (y0 > y1) {//保证(x0,y0)在(x1,y1)的下面

temp = y0;

y0 = y1;

y1 = temp;

temp = x0;

x0 = x1;

x1 = temp;

}

dx = x1 - x0;

dy = y1 - y0;

x = to\_float(x0);

y = to\_float(y0);

eps1 = abs(dy);

xIncre = (float)dx / (float)eps1;

yIncre = (float)dy / (float)eps1;

flag = 1;

for (k = 0; k <= eps1; k++) {

p = (int)(x + 0.5);

q = (int)(y + 0.5);

//每条直线的上端点不处理，避免交点处处理两次，没达到填充的效果

if (k != 0){

//putpixel(p, q);

DDA(p, q, right, q);

}

x += xIncre;

y += yIncre;

}

}

//

}

// ImageRenderer

void ImageRenderer::DDA(int x0, int y0, int x1, int y1){

int dx, dy, eps1, k;

float x, y, xIncre, yIncre;

int p, q;

dx = x1 - x0;

dy = y1 - y0;

x = to\_float(x0);

y = to\_float(y0);

if (abs(dx) > abs(dy))

eps1 = abs(dx);

else

eps1 = abs(dy);

xIncre = (float)dx / (float)eps1;

yIncre = (float)dy / (float)eps1;

for (k = 0; k <= eps1; k++)

{

p = (int)(x + 0.5);

q = (int)(y + 0.5);

if (getpixel(p, q) == 0x00)

putpixel(p, q, 0xFF000000);

else

putpixel(p, q, 0x00);

x += xIncre;

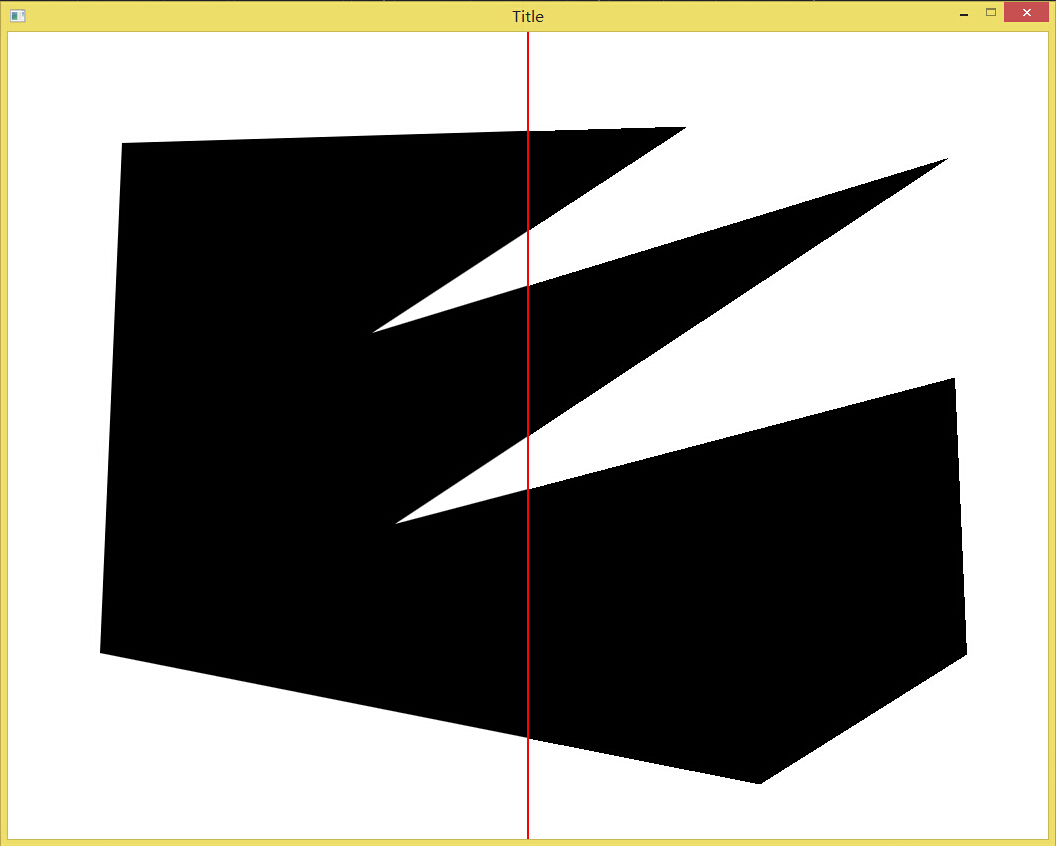
y += yIncre;

}

}

程序基于Direct2D, 这次改为内存位图载入

## 结果



左边的是Direct2D自带的方法，右边的是本算法

## 讨论

这次不再是使用DrawRectangle模拟设置像素点了，主要当然是效率问题。改用位图，在系统内存位图一次性修改再传送给显卡内存以提高效率