## 题目名：

种子填充算法

## 算法原理

种子填充算法又称为边界填充算法。其基本思想是：从多边形区域的一个内点开始，由内向外用给定的颜色画点直到边界为止。如果边界是以一种颜色指定的，则种子填充算法可逐个像素地处理直到遇到边界颜色为止。

种子填充算法常用四连通域和八连通域技术进行填充操作。

从区域内任意一点出发，通过上、下、左、右四个方向到达区域内的任意像素。用这种方法填充的区域就称为四连通域；这种填充方法称为四向连通算法。

从区域内任意一点出发，通过上、下、左、右、左上、左下、右上和右下八个方向到达区域内的任意像素。用这种方法填充的区域就称为八连通域；这种填充方法称为八向连通算法。

这里使用4联通.实现

### 1.相关数据结构:

使用栈作为基本数据存储结构，储存遍历过的点

### 2.函数模块说明

Push2Stack 4联通核心算法，将有效点压入栈

PushPoint 鼠标左键处理函数

DrawLine 中点直线算法

### 3.核心代码

// Push to Stack

void ImageRenderer::Push2Stack(int x, int y){

putpixel(x, y, 0xFF000000);

int temp = x - 1;

if (temp >= 0 && !getpixel(temp, y)){

putpixel(temp, y, 0xFF000000);

m\_stackSeed.push({ temp, y });

}

temp = x + 1;

if (temp < m\_cWidth && !getpixel(temp, y)){

putpixel(temp, y, 0xFF000000);

m\_stackSeed.push({ temp, y });

}

temp = y - 1;

if (temp >= 0 && !getpixel(x, temp)){

putpixel(x, temp, 0xFF000000);

m\_stackSeed.push({ x, temp });

}

temp = y + 1;

if (temp < m\_cHeight && !getpixel(x, temp)){

putpixel(x, temp, 0xFF000000);

m\_stackSeed.push({ x, temp });

}

}

// 推送点

void ImageRenderer::PushPoint(int x, int y){

// 刻画线段中

if (now\_point){

now\_point->x = static\_cast<float>(x);

now\_point->y = static\_cast<float>(y);

++now\_point;

++m\_cPointCount;

if (now\_point >= points + MAX\_POINT\_SIZE){

now\_point = nullptr;

}

}

// 设置种子起始点

else{

//

m\_stackSeed.push({ x, y });

while (m\_stackSeed.size())

{

auto pt = m\_stackSeed.top();

m\_stackSeed.pop();

Push2Stack(pt.x, pt.y);

}

// 载入

m\_pNowBitmap->CopyFromMemory(nullptr, m\_pBuffer, sizeof(RGBQUAD)\*m\_cWidth);

// 清理

ZeroMemory(m\_pBuffer, m\_cWidth\*m\_cHeight \* sizeof(RGBQUAD));

this->OnRender();

}

}

// 刻画线条

void ImageRenderer::DrawLine(int x1, int y1, int x2, int y2){

float k = float(y2 - y1) / float(x2 - x1);

int flag = 0;

if (k > 1 || k < -1) {

flag = 1;

x1 ^= y1 ^= x1 ^= y1;

x2 ^= y2 ^= x2 ^= y2;

k = float(y2 - y1) / float(x2 - x1);

}

float d = 0.5f - k;

if (x1 > x2) {

x1 ^= x2 ^= x1 ^= x2;

y1 ^= y2 ^= y1 ^= y2;

}

while (x1 != x2)

{

if (k > 0 && d < 0){

++y1, ++d;

}

else if (k < 0 && d > 0){

--y1, --d;

}

d -= k;

++x1;

if (flag) putpixel(y1, x1, 0xFF000000);

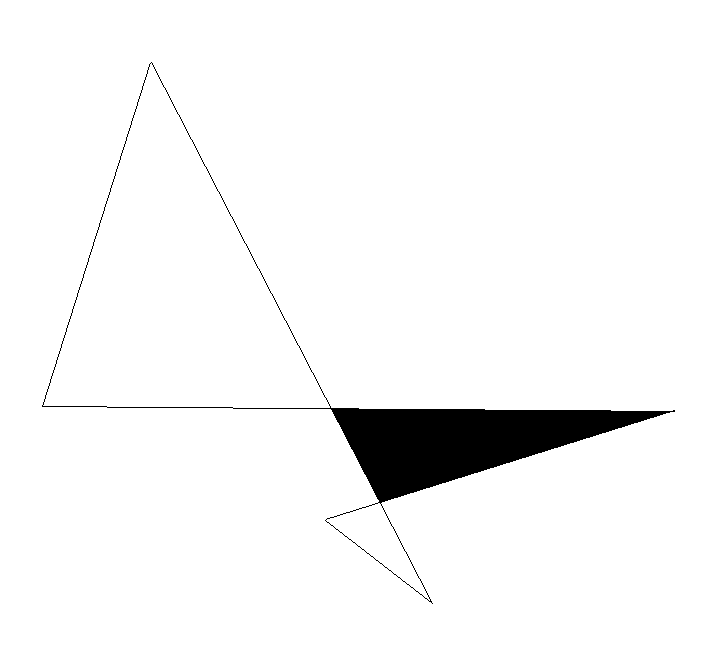
else putpixel(x1, y1, 0xFF000000);

}

}

程序基于Direct2D, 这次改为内存位图载入

## 结果



## 讨论

这次全靠CPU实现，再一次写入位图，可能一次性写入几十万点，这时耗时大致在1秒左右。而且使用默认std的栈时,扫尾工作过于耗时。

,可以考虑使用CPU加速实现，由于时间关系，并未在这里给出相关代码，请参考Direct2D的特效(Effect)