- 最简单的画线算法

for (float i = 0; i < 1.; i += .01) {  
 int x = x0 + (x1 - x0) \* i;  
 int y = y0 + (y1 - y0) \* i;  
 img.set(x, y, clr);  
}

**这段代码的问题在于, 除了效率以外, 这里的画线间隔i是个常数, 如果i足够小可以画出满足要求的线，但是每个点都会被重复的画很多次， 而如果i太大， 则会画出不连续的散点虚线.**

**Note: 由于是画线，所以不能出现方格(即四个像素组成的方块), 该算法满足要求.**

- 根据相似三角形斜率画线

for (int x = x0; x <= x1; ++x) {  
 float k = (x - x0) / (float )(x1 - x0);  
 int y = y0 + k \* (y1 - y0);  
 image.set(x, y, color);  
}

**当线段的斜率不是特别大时, 这个方法还行的通, 否则的话, 还是会出现不连续的散点虚线.**

- 通过关于对角线折叠解决画斜率大于一的斜线

bool fold = false;  
if (std::abs(x1-x0) < std::abs(y1-y0)){  
 fold = true;  
 std::swap(x0, y0);  
 std::swap(x1, y1);  
}

for (int x = x0; x <= x1; ++x) {  
 float k = (x - x0) / (float )(x1 - x0);  
 int y = y0 + k \* (y1 - y0);  
 int tx = x;  
 int ty = y;  
 if (fold) std::swap(tx, ty);  
 image.set(tx, ty, color);  
}

# - Bresenhams算法: 没有除法的高效算法

### [布雷森汉姆直线算法](http://www.baidu.com/link?url=tKBILe2a6FIp6Kc6ABEkDuBJLRxWD7fWTZmtk_2G-_gqLY4uwmb9n50rH4UzZ06qmxIi9yIGhm-ZD49uubo4zb5N0qURq7MqOFCsKfpUWKD3ldkE7kId4GRLPAgE7GJXDWxFO_i34kllvTjTRjMytqtjSDG643WXt2fgOBzNOqOYbaGNaynax-l89-9xaEb2" \t "/home/tsin/文档\\x/_blank)

核心部分[画线的最大斜率为1]:

int dx = x1 - x0;  
int dy = y1 - y0;  
  
float e = 0, t = std::abs(dy/(float)dx);  
  
for(int y = y0, x = x0; x < x1; x++){  
 image.set(x, y, color);  
 if((e += t) > .5){   
 y += (y1 > y0) ? 1 : -1;  
 e -= 1;*// the 1 is most important.* }  
}

**对该算法的理解:**

**画线的本质是将现实世界中的连续的直线在离散的像素屏幕上绘制出来.**

我们可以将这个直线看成一个函数f，对于每个给定的离散像素的横坐标x,得到一个连续函数的值,根据这个值判断这个像素是水平绘制还是上/下偏移一个位置(相对于前一个像素)

相对于这个连续函数，将每一个像素边长看为1， 则这个像素的中心点到各边的距离为0.5， 我们认为像素的中心，代表了其在连续世界中的映射, 所以两个像素的距离为1， 所以x对应的函数值可以通过累加得到 e += (t \* 1)，即为e+=t;

算法的过程只是循环以下过程:

先画x0,y0, 画下一个点x1,y1, 其中x1 = x0 + 1; 求y1.

核心在于:

e += t 不断累加表示连续函数上的点相对于起始点的纵坐标y0的真实高度位置(未超过0.5时)，

e -= 1 表示某个点高度超过0.5时，近似选择上方的像素点，起始纵坐标变化后(升高1)，更新函数的相对真实高度

------------------------------------------------------------------------------------------------------------------------------

note:有两点需要注意

1.直接使用 这个算法的核心无法画出斜率大于1 的斜线， 如果你非要试试他会把你的斜线clip斜率为1. 这是很显然的

for (xi = x0; xi < x1; ++xi) {

//绝了， 你的编程注意事项里的不在循环体里面，改变循环变量你记哪去了！！！

//if (steep)std::swap(xi, yi);

image.set(xi, yi, color);  
 e += t;  
 if (e > 0.5){  
 ++yi;  
 e -= 1;  
 }  
}

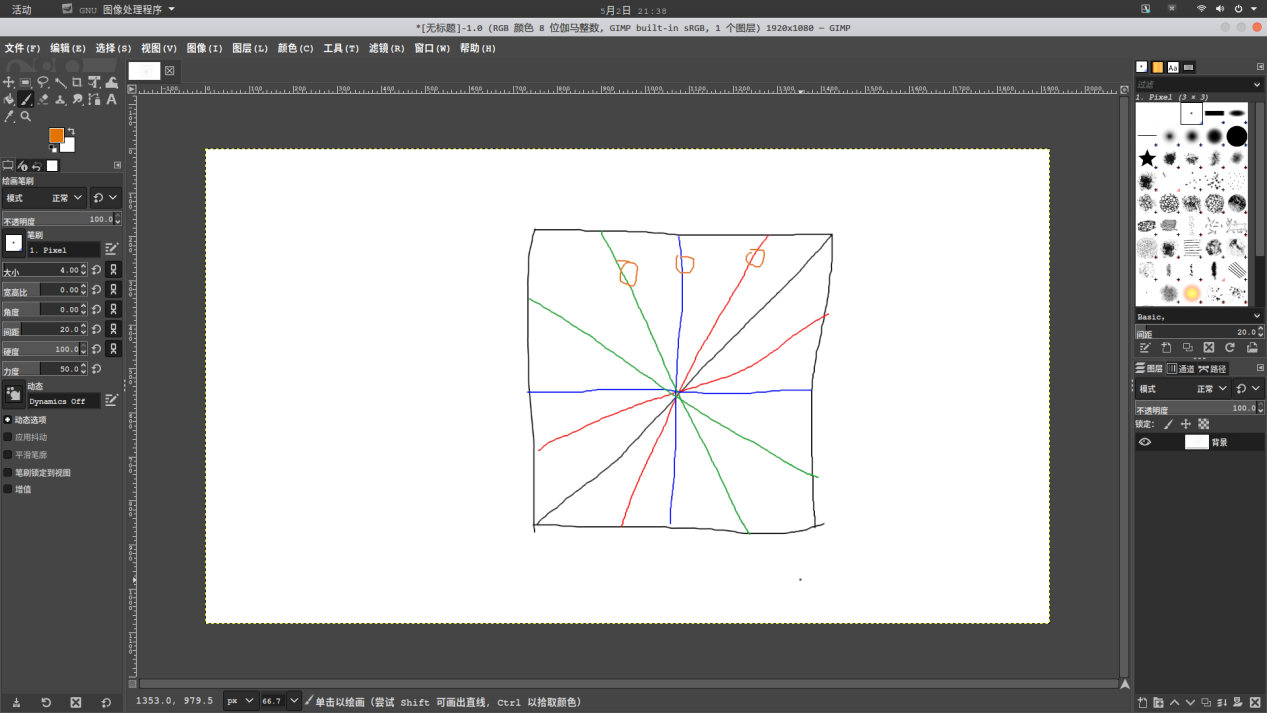
2.关于向下绘制的处理

总结： 任何斜率的绝对值大于1 的斜线， 关于y=x对称后，绝对值都会小于1, 小于零的仍小于零， 大于零的仍大于零， 然后我们这么检查是否向下呢， 记住 翻转过后向上的任然向上向下的任然向下, 但是向下的直线翻转过后，起始点会变成终止点， 我们将其相互交换即可(note:这和沿y=x翻转不同)， 向下的线是yi -= 1;

NOTE

float e = 0, t = std::abs((y1-y0)\*1.0/(x1-x0));

这个一定在交换之后写！！！！把所有的情况都处理好后写。



------------------------------------------------------------------------

请你务必徒手写出完整的代码， 进行反复严苛的逻辑训练

完整的代码是[]:

void line(vec2i a, vec2i b, TGAImage &image, TGAColor color){  
  
 int x0 = a.x, y0 = a.y;  
 int x1 = b.x, y1 = b.y;  
 int xi = -01, yi = -01;  
  
 bool steep = (std::abs(y1-y0) > std::abs(x1-x0)) ? true : false;  
  
 if (steep) {  
 std::swap(x0, y0);  
 std::swap(x1, y1);  
 }  
  
 bool reverse = (x0 > x1) ? true : false;  
 if (reverse) {  
 std::swap(x0, x1);  
 std::swap(y0, y1);  
 }  
  
 bool down = (y0 > y1)? true :false;  
  
 float e = 0, t = std::abs((y1-y0)\*1.0/(x1-x0));  
  
 for (xi = x0, yi = y0; xi < x1; ++xi) {  
 if (steep)  
 image.set(yi, xi, color);  
 else  
 image.set(xi, yi, color);  
  
 e += t;  
 if (e > 0.5){  
 yi += ((down) ? -1 : 1);  
 e -= 1;  
 }  
 }  
}