计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2017,26(8):180−183 [doi: 10.15888/j.cnki.csa.005958]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

直线的 Bresenham 并行绘制算法①

卫洪春

(四川文理学院 计算机学院, 达州 635000)

摘　要: 本文对直线的 Bresenham 并行绘制进行了研究, 并从概率上计算了当斜率 k 属于(0,1/2)时, 每条扫描线上的平均像素个数, 发现采用并行绘制方法在该区间可节约 3/4 的绘制时间. 根据理论分析, 结合经典 Bresenham 画直线算法, 实现了并行 Bresenham 画直线算法, 并将绘制结果与 windows 绘图程序和经典的 Bresenham 画直线算法结果进行了比较, 其绘图结果完全相同. 对于扫描线多点并行绘制而言, 具有很好的效果, 便于硬件实现, 以增强对实时绘图的响应.

关键词: 直线生成; Bresenham; 并行; 概率; 算法

引用格式: 卫洪春.直线的 Bresenham 并行绘制算法.计算机系统应用,2017,26(8):180–183. http://www.c-s-a.org.cn/1003-3254/5958.html

Bresenham Parallel Drawing Algorithm for Straight Line

WEI Hong-Chun

(School of Computer, Sichuan University of Arts and Science, Dazhou 635000, China)

Abstract: Studying the linear parallel drawing based on Bresenham, this paper calculates the average number of pixels in each scan line when the slope k belongs to (0,1/2) based on Probability, and finds that three-quarters of the drawing time can be saved with this method. According to the theoretical analysis, combined with classical Bresenham algorithm for generating line, it realizes a parallel Bresenham algorithm for generating line, and the results with this method are the same as the windows drawing program and the classic Bresenham drawing linear algorithm. It is very important to multi-point parallel rendering for scan line and is easy to design hardware which can enhance the response to real-time drawing.

Key words: generating straight line; Bresenham; parallel; probability; algorithm

1　引言

基本图元绘制是计算机图形学的重要内容, 直线生成在图形绘制、图像处理中用途十分广泛. 人们通过深入研究, 提出了很多有效的直线绘制方法, 如数值微分算法(DDA)、中点画线算法、Bresenham 画直线算法[1-4]等等. 其中 Bresenham 直线生成算法巧妙地利用误差项的符号来判断下一个像素点的位置, 其优点是不需进行小数和取整运算, 只需利用整数的加法和乘法即可完成待生成像素的计算. 该算法不仅效率高, 且能直接在硬件上实现, 因此被广泛应用. Bresenham 画直线算法虽然精炼高效, 但也存在计算相当浪费的现象[5]. 图 1 中, 存在若干个待绘制像素点均位于同一条扫描线上. 为了确定这些像素的位置, 需计算多次(例如 n 次)误差项、逐点比较判断后完成像素绘制. 但这 n 次计算及判断中, 有 n-1 次计算与判断可以合并为一次计算与判断, 并可同时并行绘制若干个连续点, 然后对边界点进行计算与判断, 这样就可以极大地提高绘图效率.

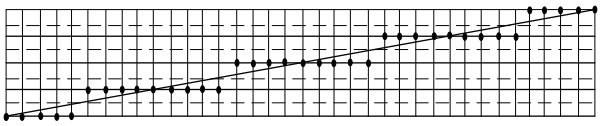

图 1　经典 Bresenham 算法生成直线

① 基金项目: 四川省教育厅项目(15ZB0326); 四川文理学院项目(2015TP003Y)
收稿时间: 2016-12-06; 采用时间: 2017-01-20

2 Bresenham 画线算法改进分析

如前所述, 经典 Bresenham 画线算法是逐点比较后完成像素绘制, 其中存在过多的判断和计算浪费, 执行效率不太理想, 因此需要对该算法进一步改进[6-8].

在图 2 中, 从 1 号点(1, 1)到 31 号点(31, 3)的直线段绘制过程中, 根据 Bresenham 画线算法, 需要计算 30 次误差项, 然后根据每次计算得到的误差项的正负符号决定下一像素点绘制在何处. 在图 2 中, dx=31-1=30, dy=3-1=2. 误差初值 p=2·dy-dx=4-30=-26; 由于 p<0, 且当从 1 号点连续绘制到 7 号点后, 待绘制的 8 号点的误差项为: p=p+6·(2dy)=-26+12dy=-26+24=-2, 此时 p<0, 故在如图位置绘 8 号点; 然后计算 9 号点的误差项 p=p+2·dy=-2+4=2>0, 则在如图位置绘制 9 号点, 此时的扫描线 y 已经从 1 变化到 2. 计算 10 号点的误差项, p=p+2dy-2dx=2+2·2-2·30=-54, 在图 2 中画 10 号点. 由于 p=p+13·2dy=-54+52=-2<0, 因为 10 号点到 23 号点这些点的误差项均满足 p 小于 0, 故这些点都在 y=2 这条扫描线上. 当绘制完 23 号点后重新计算 p 时, p=2, 在图上画 24 号点. 但在绘制扫描线 y=2 上的所有像素点时, 经典的 Bresenham 算法需要计算并判断 14 次, 但其中 10 点到 22 号点的计算与判断均是重复的, 这显然增加了计算与判断开销, 不能很好地满足实时绘制要求, 因此需要改进.

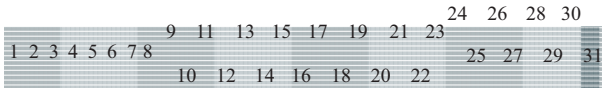

图 2 Bresenham 算法绘制点(1, 1)到点(31, 3)的直线段

设直线段的斜率 0<k≤1, 且线段的起点坐标(x_1, y_1), 终点坐标(x_2, y_2), dx=x_2-x_1, dy=y_2-y_1. 当 k=dy/dx 在(0, 1)时, 设 n=[1/k]=[dx/dy](即 n 是不超过 1/k 的最大正整数), 则对于每条扫描线 y 而言, 其上最少有 n 个点, 最多有 n+1 点. 因为若每条扫描线上的点小于 n(设为 n_{min}), 则 n_{min}·dy<dx; 若每条扫描线上的点大于 n+1(设为 n_{max}), 则 n_{max}·dy>dx. 显然这两种情况均是不可能的. 因此, 除扫描线 y1 与扫描线 y2 上的像素点外, 其余各条扫描线上的像素个数或为 n, 或为 n+1, 且相邻两条扫描线上的像素点数最多差 1.

计算扫描线 y=y_1 上需绘制的像素个数. 此时假设有 m 个像素点. 由误差初值 p=2dy-dx<0 可知, 当待绘制的像素点从扫描线 y=y_1 变化到扫描线 y=y_1+1 时, 误差项 p 必然大于 0(否则待绘制像素点仍位于扫描线 y=y1 上). 由经典算法可知, 当绘制 y=y_1 扫描线上的最后一个点前(如图 2 中的 8 号点), 误差项 p=p+(m-2)*(2dy)=2dy-dx +2(m-2)dy=2(m-1)dy-dx≤0, 绘制完 y=y_1 扫描线上的最后一个点(如图 2 中的 8 号点)后 p=p+2dy>0, 然后转到 y=y_1+1 扫描线上绘制下一个像素点(如图 2 中的 9 号点). 由 2(m-1)·dy-dx≤0 可知, m≤1+dx/(2dy), 即 y=y_1 扫描线上最多有不超过[1+dx/(2dy)]+1 个点. 这就确定了扫描线 y=y_1 上应绘制的像素数 m.

当绘制完扫描线 y_1 上的所有像素点后, 除扫描线 y_1 和扫描线 y_2 外, 其余各条扫描线上将连续绘制若干像素点(或为 n, 或为 n+1), 因此可以在每条扫描线上先连续绘制 n-1 个点, 然后计算误差项 p 并判断其正负符号, 以确定下一个待绘制像素点的位置. 这就大大减少计算与判断工作量, 提高绘图效率.

如上分析, 当斜率 k∈(1/2, 1)时, 每条扫描线上的像素点个数或为 1, 或为 2, 即最多不超过 2 个点, 此时可不进行改进; 当 k ∈(0, 1/2)时, k 是连续型随机变量, 且均匀分布在区间(0, 1/2)上, 随机变量 k 在该区间的分布函数为:

$$f(k)=\begin{cases}1/(1/2-0)=2 & (0<k<1/2)\\ 0 & \text{其他}\end{cases} \quad (1)$$

随机变量 k 在(0, 1/2)上的数学期望为:

$$E(k)=\int_{-\infty}^{+\infty} kf(k)dk=\int_0^{1/2} 2kdk=k^2\Big|_0^{1/2}=1/4 \quad (2)$$

故在(0, 1/2)上, 平均每条扫描线上有[1/k]=4 个扫描点. 若利用上述算法, 在(0, 1/2)区间上平均可以减少 3/4 的计算、判断、绘制工作量, 特别是对小斜率直线的绘制, 改善效果相当好.

3 改进算法的实现与实验结果分析

3.1 改进的 Bresenham 画线算法的实现

根据算法的改进分析, 当直线斜率 k∈(1/2, 1)时, 由于每条扫描线上的像素点个数最多不超过 2 个点, 改进效果不明显, 可利用经典算法; 当 k∈(0, 1/2)时, 利用改进算法; 当 dx 或 dy 为零时, 直接处理[9-15]. 在 VC6.0 平台上设计算法如下:

```
void DealEdge(int x1, int y1, int x2, int y2, CDC *pDC )
//线段起点(x1, y1), 终点(x2, y2)
{ int c=RGB(255, 0, 0); //待绘制点的颜色
```

```
int x, y; //待绘制的像素点坐标
int dx=x2-x1, dy=y2-y1; //坐标差
int n=dx / dy, i;// n 是每条扫描线上的平均像素数
int p, twoDy, twoDyDx; //误差项计算用变量
p=2*dy-dx; //误差初值
twoDy=2*dy;
twoDyDx=2*(dy-dx); x=x1, y=y1;
if(p>0) //斜率 1/2≤ K ≤ 1 时, 每行或有 1 个点,
//或有 2 个点, 此时采用经典算法.即逐点绘制.
    {pDC->SetPixel(x, y, c);
    while(x<x2){ x++;
      if(p<0) p+=twoDy;
      else{y++;p+=twoDyDx; }
      pDC->SetPixel(x, y, c); }}
else { //p <=0, 斜率 k 在 (0, 1/2)上, 每行有
//[dx/dy]个或[dx/dy]+1 个点, 此时采用改进算法
int n_1_2=n / 2;
for(i=0;i<=n_1_2-2;i++)//画扫描线 y1 上的所像素点
pDC->SetPixel(x++, y, c );//此结构可设计为并行绘制
p+=twoDy*(n_1_2-2); //twoDy=2*dy
if(p <=0){
while(p <=0){//虽然是循环结构, 但最多执行一次
pDC->SetPixel(x++, y, c);//绘制扫描线 y1 上的右边界点
p +=twoDy; //twoDy=2*dy
if(p>0) break; } }
for( y ++; y < y2; y++){
//扫描线 y1 与 y2 间每条扫描线上的所有像素点
if(p>0)
for(i=0;i<=n-1;i++)//连续绘制扫描线 y 上的 n 个点
pDC->SetPixel(x++, y, c);//此结构可设计为并行绘制
p +=twoDyDx; //此时 p<0, twoDyDx=2*(dy-dx)
p +=twoDy*(n-1); //twoDy=2*dy
if(p <=0){
while(p <=0){
pDC->SetPixel(x++, y, c);//绘制扫描线 y 上的右边界点
p +=twoDy; //twoDy=2*dy
if(p>0) break; } }}
for(; x <=x2; i++)//扫描线 y2 上的所有像素点
pDC->SetPixel(x++, y2, c);//此结构可设计为并行绘制
} }
```

本文讨论了斜率 k∈(0, 1)时的情况, 但在此基础上能较容易实现其它情况.

3.2 实验结果比较与分析

为测试本文所提算法, 当 k 在(0, 1/2)上时, 其时间效率已做了理论分析, 在起点(0, 0)、终点(50, y)(y 从 1 变化到 25)上的绘制结果与 windows 自带的绘图程序的绘制结果比较如图 3(此处列举了部分结果, 浅色是改进算法的结果, 深色为 windows 画图程序的绘制结果).

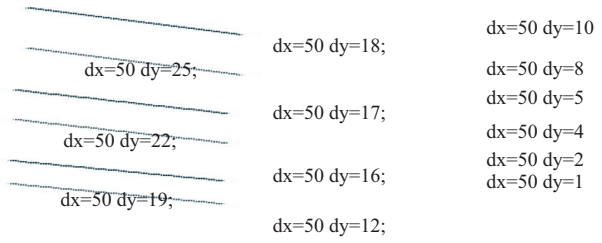

图 3 与 windows 自带绘图程序的绘制结果的比较

与经典 Bresenham 画线算法的比较如图 4(浅色是改进算法结果, 深色是经典 Bresenham 画线算法的结果).

图 4 经典 Bresenham 画线算法的绘制结果的比较

在时间效率方面, 将上述算法中注释为“并行绘制”的循环结构改写为 VC 平台下的 MoveTo()、LineTo()画直线函数后, 与经典 Bresenham 画直线算法对同一直线段重复绘制 1000 次的比较结果如表 1 所示. 从表 1 中的时间(毫秒)比较可得出, 在小斜率条件下, 改进算法明显优于经典算法.

4 结论

本文对直线的 Bresenham 并行算法从理论上进行了研究, 并从概率上计算了当斜率 k 在(0, 1/2)上时, 每

条扫描线上的平均像素个数, 得出了采用并行绘制方法可节约时间大约 3/4. 本文结合经典的 Bresenham 画直线算法, 实现了并行 Bresenham 画直线算法. 实验结果显示, 改进算法的绘制结果与 windows 绘图程序及经典的 Bresenham 画直线算法的绘图结果相同, 并且改进算法在小斜率(0 到 0.5)条件下的绘制效率明显高于经典算法. 这表明扫描线多点并行绘制算法在线段绘制过程中的效果相当好, 且便于硬件实现, 以增强对实时绘图的响应.

表 1 经典算法与改进算法的效率比较

起点	终点	经典算法	改进算法
10, 10	600, 12	2418	16
10, 10	600, 15	2418	32
10, 10	600, 20	2418	63
10, 10	600, 50	2434	312
10, 10	600, 110	2440	827

参考文献

1 王汝传, 黄海平, 林巧民, 等. 计算机图形学教程. 3 版. 北京: 人民邮电出版社, 2014.

2 苏小红, 李东, 唐好选, 等. 计算机图形学实用教程. 3 版. 北京: 人民邮电出版社, 2014.

3 Hearn D, Baker MP, Carithers WR. 计算机图形学. 4 版. 蔡士杰, 杨若瑜, 译. 北京: 电子工业出版社, 2014.

4 徐长青, 许志闻, 郭晓新, 等. 计算机图形学. 2 版. 北京: 机械工业出版社, 2010.

5 郑宏珍, 赵辉. 改进的 Bresenham 直线生成算法. 中国图象图形学报, 1999, 4(7): 606–609.

6 翟文正, 程耀东, 石广田. 空间直线生成的分段画线算法. 兰州交通大学学报(自然科学版), 2006, 25(4): 68–71.

7 李竹林, 邓石冬. 一种改进的等分迭代 Bresenham 直线生成算法. 电子设计工程, 2015, 23(7): 61–63.

8 丁宇辰. 圆弧的生成算法研究. 南京工程学院学报(自然科学版), 2010, 8(2): 59–62.

9 许金超, 曾国荪. 基于栈状态关系的动态软件水印算法. 计算机应用, 2013, 33(4): 1065–1069.

10 徐文鹏, 强晓焕, 侯守明. 面向问题解决的图形学教学改革探讨. 高等理科教育, 2013, (5): 107–111.

11 王志俊, 姜咏梅, 田记. 矩阵在图形学几何变换中的应用. 高等数学研究, 2014, 17(1): 87–88,99.

12 鄢涛, 余悦, 于曦. 基于 C++的大整数类型的设计与实现. 成都大学学报(自然科学版), 2016, 35(3): 252–255,270.

13 董鑫正, 单缅, 傅晓阳. C++教学中的知识点逻辑关系探讨. 计算机教育, 2016, (9): 163–166.

14 张麟华, 孔令德, 杨慧炯. 面向图形图像处理的 C++课程案例设计. 计算机教育, 2013, (4): 88–91.

15 袁小翠, 吴禄慎, 陈华伟. Delaunay 三角剖分算法改进与对比分析. 计算机应用与软件, 2016, 33(9): 163–166.