

基于嵌入式图形系统的改进 Bresenham 反走样算法

谢莹¹, 许荣斌¹, 赵宏坤²

(1. 安徽大学 计算智能与信号处理国家重点实验室, 安徽 合肥 230039;

2. 安庆师范学院, 安徽 安庆 240001)

摘要: 基于 Bresenham 算法, 依据去浮点数计算原理, 结合矩形滤波反走样技术, 提出了一种快速的反走样直线的优化算法, 并在配备 ARM7 微控制器 LPC2290 的 MagicARM2200 仪器上得以实现。该算法明显加快了反走样直线的生成速度并在低分辨率的显示环境中获得非常好的效果。

关键词: Bresenham 算法; 反走样; 嵌入式图形系统; ARM

中图分类号: TP301.6

文献标识码: A

文章编号: 1673-629X(2006)11-0100-03

Improved Bresenham Line Drawing Anti-Aliasing Algorithm
Based on Embedded Graphics SystemXIE Ying¹, XU Rong-bin¹, ZHAO Hong-kun²

(1. Key Lab. of Intelligent Computing & Signal Processing, Anhui University, Hefei 230039, China;

2. Anqing Normal College, Anqing 240001, China)

Abstract: In this article, gives a new anti-aliasing technique according to the symmetry of the straight line and the theory of non-operation of floating point, based on Bresenham algorithm and the rectangle filter anti-aliasing technique. In the end, the algorithm is put into practice on MagicARM2200, which equipped with LPC2290. This algorithm approves the speed and obtains the good show effect in the low resolution.

Key words: Bresenham algorithm; anti-aliasing; embedded graphics system; ARM

0 引言

嵌入式图形系统的图形显示, 通常用光栅图形显示器来实现, 光栅图形显示可以看作一个像素的矩阵。在光栅显示器上显示任何一种图形, 实际上都是一些具有一种或多种颜色的像素集合。确定最佳逼近图形的像素集合, 并用指定属性写像素的过程称为图形的光栅化^[1]。其显示的图元(如一条斜直线、圆(弧)或多边形边界等)会出现锯齿形或台阶状的现象。这种用离散量表示连续量所造成的失真效果, 称为走样。消除走样的方法称为反走样。在具有多灰度级的显示器上, 可以通过控制灰度级的缓慢变化来解决此类问题。

在光栅图形显示器上生成直线的算法中, Bresenham 算法构思简单, 并且由于只是进行整数运算, 速度较快, 具有比较明显的优点。文中在对 Bresenham 画线算法进行反走样设计中, 通过对其方法和其中参数的分析, 得出了一种显示效果好、执行速度快的反走样算法。该算法在

MagicARM2200 上得以实现。仪器配有 ARM7 微控制器 LPC2290, 系统时钟达 60MHz, 配有 16 位总线输出(经过驱动器), 8M 字节 RAM。

1 改进的 Bresenham 算法

Bresenham 算法是计算机图形学领域使用最广泛的直线扫描转换算法。传统算法原理如下: 过各行各列像素中心构造一组虚拟网格线。按直线从起点到终点的顺序计算直线与各垂直网格线的交点, 然后确定该列像素中与此交点最近的像素。该算法的巧妙之处在于采用增量计算, 使得对于每一列, 只要检查一个误差项的符号, 就可以确定该列的所求像素^[2]。

图 1 中, 设直线方程为 $y_{i+1} = y_i + k(x_{i+1} - x_i) + k$ 。假设行坐标像素已经确定为 x_i , 其列坐标为 y_i 。那么下一个像素的行坐标为 $x_i + 1$, 而列坐标要么为 y_i , 要么递增 1 为 $y_i + 1$ 。是否增 1 取决于误差项 d 的值, 图 2 为 d 的几何含义。误差项 d 的初值 $d_0 = 0$, x 坐标每增加 1, d 的值相应递增直线的斜率值 k , 即 $d = d + k$ 。一旦 $d \geq 1$, 就把它减去 1, 这样保证 d 在 0 和 1 之间。当 $d \geq 0.5$ 时, 直线与垂线 $x = x_{i+1}$ 交点最接近于当前像素 (x_i, y_i) 的右上方像素 $(x_i + 1, y_i + 1)$; 而当 $d < 0.5$ 时, 更接近于右方像素 $(x_i$

收稿日期: 2006-03-05

作者简介: 谢莹(1981-), 女, 安徽黄山人, 硕士研究生, 研究方向为嵌入式系统; 导师: 吴建国, 教授, 博导, 研究方向为嵌入式系统与 EDA。

$+1, y_i)$ 。为方便计算,令 $e = d - 0.5$, e 的初值为 -0.5 , 增量为 k 。当 $e \geq 0$ 时,取当前像素 (x_i, y_i) 的右上方像素 $(x_i + 1, y_i + 1)$; 而当 $e < 0$ 时,取 (x_i, y_i) 右方像素 $(x_i + 1, y_i)$ 。源程序见清单 1。

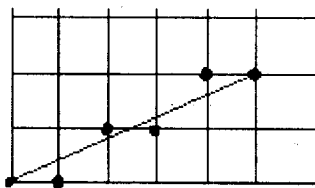


图 1 Bresenham 算法

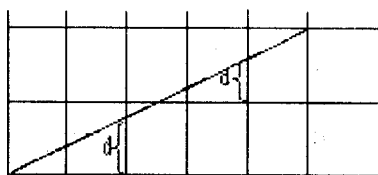


图 2 d 的几何含义

清单 1 Bresenham 画线算法程序:

/* 为方便计算,令 $e_0 = -0.5$, $e_{i+1} = d_{i+1} - 0.5$, 增量为 k 。
当 $e_{i+1} \geq 0$ 时,取当前像素 (x_i, y_i) 的右上方像素 $(x_i + 1, y_i + 1)$; 而
当 $e_{i+1} < 0$ 时,更接近于右方像素 $(x_i + 1, y_i)$ 。*/

```
void Bresenhamline (int x0, int y0, int x1, int y1, int color)
```

```
{ int x, y, dx, dy;
```

```
float k, e;
```

```
dx = x1 - x0; dy = y1 - y0; k = dy/dx; //dx, dy 是直线总的增量。
```

k 是实数的斜率

$e = -0.5$; $x = x_0$; $y = y_0$; //x, y 为起点。e 是调整数,以使问题成为 ≥ 0 , 还是 < 0 的问题

//这里利用 $k = dy/dx$ 这个斜率值,而不是利用逐个比较方法

//实数 y_i 的值 ≥ 0.5 , 则用 $y(\text{整数}) + 1$ 的像素,若 $y_i < 0.5$ 则 y 轴仍用上一个像素的 y 坐标

//一旦用 $y + 1$, 则 y_i 这个 y 轴的实数增量就会多出 1 这个增量。

所以,一旦 $y++$, 就要 y_i--

//这里,为了使程序不是判断是否 ≥ 0.5 , 而是改为判断是否 ≥ 0 , 需要对所有的 y_i 进行 -0.5 操作

//这里用 float e 表示 y_i , 并且赋 $e = -0.5$, 以后 $e = e + k$, 这样实际上就实现了 $e - 0.5$ 的目的

```
for (i=0; i<dx; i++) //循环 dx 次,即绘制 dx 个像素
```

```
{ drawpixel (x, y, color);
```

```
x = x + 1; e = e + k;
```

```
if (e >= 0)
```

```
{ y++; e = e - 1;
```

```
}
```

上述 Bresenham 算法在计算直线斜率与误差项时用到小数与除法。可以改用整数以避免除法。

改进算法分析如下:前文已经设定交点与网格线之间距离为误差项 d , 根据 d 判断网格中哪个像素点距离交点最近。当 $d > 0.5$, 直线更接近 $(x_i + 1, y_i + 1)$, 当 $d < 0.5$, 更接近 $(x_i + 1, y_i)$ 。可以发现改进此算法的关键在于 d 值, d 初值为 0, 每走一步都有 $d_{i+1} = d_i + k * 1$ 。一旦 y 方

向走了一步, 则 d 减去 1。为了计算方便, 令 $e = d - 0.5$ 。这样判断就变成 $e > 0$, 下一像素点 y 坐标加 1, $e < 0$ 则不加 1。此时 e 初值 -0.5 , 每走一步 $e_{i+1} = e_i + k * 1$ 。为了免去小数和除法运算, 因为只用判断误差项 e 的符号, 于是用 $2e * (dx)$ 来代替 e 。源程序见清单 2。

清单 2 改进的 Bresenham 画线算法程序:

```
void InterBresenhamline (int x0, int y0, int x1, int y1, int color)
```

```
{
```

```
int x, y, dx, dy;
```

```
int k, e;
```

```
dx = x1 - x0; dy = y1 - y0; e = -dx;
```

```
x = x0;
```

```
y = y0;
```

```
for (i=0; i<dx; i++)
```

```
{ drawpixel (x, y, color);
```

```
x++; e = e + 2 * dy;
```

```
if (e >= 0) { y++; e = e - 2 * dx; }
```

```
}
```

/* 主函数 */

```
Main()
```

```
{
```

```
int x1, y1, x2, y2;
```

```
int * gdriver, * gmode;
```

```
gdriver = DETECT;
```

```
gmode = 0;
```

```
initgraph (&gdriver, &gmode, "");
```

```
printf("Input the first coordinate:");
```

```
scanf("%d, %d", &x1, &y1);
```

```
printf("Input the second coordinate:");
```

```
scanf("%d, %d", &x2, &y2);
```

```
bres_line(x1, y1, x2, y2);
```

```
}
```

2 直线的矩形滤波反走样

直线生成算法的优劣对嵌入式系统图形界面至关重要。Bresenham 算法是最有效的直线生成算法, 利用去浮点数计算原理可以进一步加快生成速度。但由于受到光栅扫描显示器的限制, 用离散像素绘制的直线存在走样问题, 实际应用中经常需要进行反走样处理。

矩形滤波反走样是一种常用的反走样技术, 在本算法中, 每向主轴方向前进一个像素单位, 则沿从轴在该线段两侧各绘制一个像素点, 而每个像素点的亮度由该像素在从轴方向上与理想线段之间的距离决定。比如设灰度级别为 G , 如果两个像素与实际线段的距离分别为 $1/4$ 和 $3/4$ 像素间距, 则其灰度分别为 $3/4G$ 和 $1/4G$ 。其原理如图 3 所示^[3]。

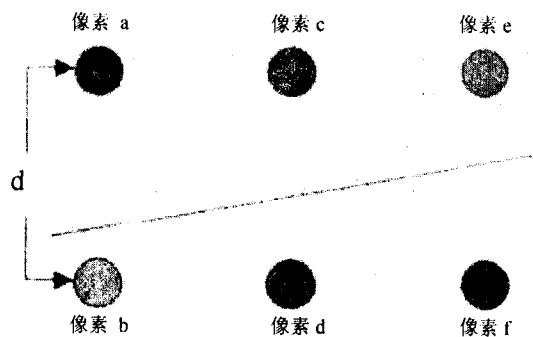


图 3 直线的反走样原理图

设相邻像素距离为 d , 且图形共有 16 个灰度级别。像素 a 与理想线距离为 $15/16 d$, 故 a 的亮度为 1, 像素 b 的亮度则为 15。同理, 像素 c 与理想线距离为 $9/16 d$, 其亮度为 7, 而像素 d 的亮度为 9。

可见, 这种以理想线段为中心的像素对, 按照亮度和等价于单个像素的方法绘制出来的线段, 实际显示效果是一条在正确位置的光滑线段, 从而实现了直线的反走样。

3 改进的 Bresenham 反走样算法

把上述两种算法结合起来, 便可得到一种新的快速反走样算法。其最大的优点是在低分辨率的显示环境中获得非常好的效果以及更快的速度。欲绘制从 a 点到 b 点的线段, 首先根据直线斜率 e , 确定是以 x 轴还是 y 轴为基本轴进行画线。本图直线以 x 轴为对称轴, 利用反走样的矩形滤波算法, x 轴在每次循环中, a 点和 b 点横坐标沿 x 轴一增一减, 并且利用沿 y 轴的增量 e 来确定实际直线的两点, 即一次主循环点亮 4 个像素点。直到把直线实际中心用两个像素点亮后, 反走样直线绘制完毕。

如上所示, 利用此方法, 优化计算并进行反走样处理, 加快了生成速度, 尤其在绘制比较长的直线过程中, 此算法的效率明显。

对于显示效果而言, 由于反走样直线的灰度级越多, 直线的平滑效果越好, 但直线的整体亮度会减弱。实际在光栅扫描显示器上, 像素的面积很小, 并且人的眼睛对不同的灰度也很难分辨。因此没有必要设很多灰度级^[4]。

如图 4 所示, 设总共有 8 个灰度级别, 即 $G = 8$ 。对于

直线 a , 在实际点 k_1 , 用像素点 A 和 B 来表示。在 k_1 处, 沿 y 轴增量为 e_1 , $1/8 < e_1 < 2/8$, 判断 $e_1 < 3/16$, 即 k_1 更靠近 $1/8$ 处, 于是 B 的灰度为 $7/8 G$, A 的灰度级为 $1/8 G$ 。同理, k_2 处可知 $5/8 < e < 6/8$, 但更接近 $6/8$ 处, 于是 C 的灰度设为 $6/8 G$, D 的灰度为 $2/8 G$ 。

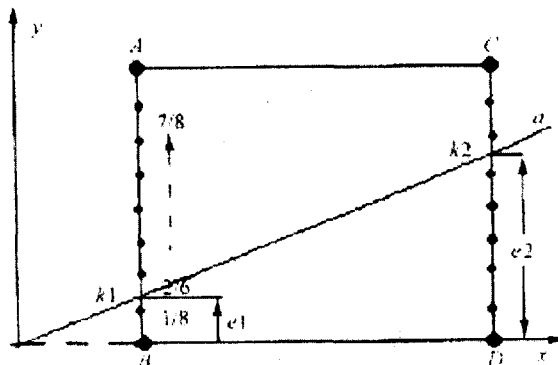


图 4 亮度计算方法

4 结论

综上所述, 通过对 Bresenham 算法的方法和其中参数的分析, 改进生成一种新的反走样算法。本改进算法在配有 ARM7 微控制器 LPC2290 的 MagicARM2200 中得以实现^[5], 实验证明可以明显加快反走样直线的生成速度, 并且在低分辨率的显示环境中获得了非常好的效果^[6]。

参考文献:

- [1] Abrash M. 图形程序开发人员指南[M]. 北京: 机械工业出版社, 1998.
- [2] 刘勇奎. 计算机图形学的基础算法[M]. 北京: 科学出版社, 2001.
- [3] 钟奉金. VGA16 上个灰度级别的设置[J]. 电脑开发与应用, 1995, 8(1): 60-63.
- [4] 孙德敏. 工程最优化方法及应用[M]. 合肥: 中国科学技术大学出版社, 2000.
- [5] 刘勇奎, 李彬. 多灰度显示器上的曲线绘制[J]. 计算机工程与设计, 1997, 18(4): 60-64.
- [6] 倪明田, 吴良芝. 计算机图形学[M]. 北京: 北京大学出版社, 1999.

(上接第 99 页)

决定了实时系统能否采用某种调度算法进行调度。文中采用了最坏响应时间的分析方法, 即计算出系统中每个任务的最坏响应时间, 然后判断每个任务的时限是否能满足条件, 从而决定实时任务的可行性。当然, 这种方法也在不断地发展和完善, 人们为提高分析方法的灵活性, 以及取消模型的更多限制而努力工作着。

参考文献:

- [1] 翟鸿鸣. 单处理器系统的实时调度算法研究[J]. 微机发展, 2003, 13(10): 99-101.

- [2] Joseph M, Pandya P. Finding Response Times in a Real-Time System[J]. The Computer Journal, 1986, 29(5): 390-395.
- [3] Liu C, Layland J. Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment[J]. Journal of the ACM, 1973, 20(1): 40-61.
- [4] Sprunt B, Sha L, Lehoczky J. Aperiodic Task Scheduling for Hard-Real-Time Systems[J]. Real Time Systems, 1989, 1(1): 27-60.
- [5] Spuri M, Buttazzo G. Scheduling aperiodic tasks in dynamic priority systems[J]. Real-time Systems J, 1996(1): 179-210.