

基于FPGA的直线反走样算法研究

贾银亮,张焕春,经亚枝

(南京航空航天大学自动化学院,江苏南京 210016)

摘要:反走样是计算机图形学的基本问题。为了提高直线反走样的效率,针对FPGA硬件实现的特点,结合经典的Wu反走样算法,提出一种新的直线反走样算法。该算法根据像素点中心到理想直线的距离来计算灰度值,在生成直线时预测直线相邻像素点之间的灰度值变化,并建立递推公式,使用整数移位、加法和比较来完成直线反走样,计算简单,便于硬件实现。经计算机和FPGA分别实现验证,新算法的反走样效果较好,运算速度快于Wu反走样算法并能通过FPGA进一步提高速度。

关键词:反走样;Wu算法;FPGA

中图分类号:TP391.72

文献标识码:A

文章编号:1673-629X(2011)02-0026-04

Research of Line Antialiasing Based on FPGA

JIA Yin-liang, ZHANG Huan-chun, JING Ya-zhi

(College of Automation Engineering, Nanjing University of Aeronautics and Astronautics,
Nanjing 210016, China)

Abstract: Antialiasing was one of the basic matters in computer graphics. Based on Wu algorithm, a new algorithm was presented for line antialiasing according to the characters of FPGA to improve efficiency. The new algorithm find the grayscale of each pixel according to the distance between the center of a pixel and the line. A recurrence method was built to forecast the grayscale change between neighbouring pixels using integer shift, addition and compare, so it was convenient to realize on hardware. The result shown that the antialiasing speed was improved and the efficiency can be further improved using FPGA.

Key words: antialiasing; Wu algorithm; FPGA

0 引言

理想的直线是连续的,而光栅显示器用一系列离散的像素点来表示一条直线,这将导致所显示的直线呈现锯齿状,影响显示效果。锯齿是走样现象的一种,减轻或者除去走样现象的技术称为反走样^[1]。

Wu直线反走样算法是反走样技术较早使用的方法,该方法适用于二像素宽直线的反走样。与经典的Bresenham算法生成的未进行反走样处理的直线相比,Wu直线反走样算法生成的直线视觉效果好的多。但Wu直线反走样算法比Bresenham算法复杂,Bresenham算法只需要整数加法和移位运算,而Wu直线反走样算法需要实数乘除法和取整等运算,所以Wu直线反走样算法显示效果的提高是以计算量增大为代价的^[2]。

许多文献对Wu直线反走样算法进行了改进,有

的消除了乘除运算,但却需要三角函数运算^[3,4];有的虽然进行了一些算法优化,但运算仍显复杂^[5-8]。有的虽然提高了反走样效果,但计算量却大幅增加^[9-11]。这些改进算法提升幅度有限,而且难以由硬件实现来进一步提高效率,如FPGA难以进行小数和除法运算^[12]。

文中针对硬件实现的特点,在Wu直线反走样算法基础上进行改进,提出一种新的直线反走样算法,完全消除实数的乘除运算,提高计算速度。

1 反走样算法概述

提高分辨率、区域采样和加权区域采样是常用的直线反走样技术。

1.1 提高分辨率

提高分辨率可以减少走样。假设显示器的分辨率在水平和垂直方向上各提高一倍。一条直线将穿过原来两倍的列数,虽然灰度跳变的次数增加了一倍,但跳变的幅度,无论是水平还是垂直方向都只有原来的一半。在视觉上,显示出的直线段比较平直光滑。

收稿日期:2010-06-17;修回日期:2010-09-26

基金项目:国家“十五”计划空装预研基金(102010303)

作者简介:贾银亮(1979-),男,江苏南京人,讲师,博士生,从事计算机图形学和嵌入式方面的研究。

提高分辨率后直线的显示效果要好一些,但是存储器容量和扫描转换时间都将大大增加。因此,增加分辨率的代价非常大,而且它只是减轻了锯齿现象,并不能消除这种现象^[1]。

1.2 区域采样

理想的直线是没有宽度的,但显示器上的直线至少有一个像素的宽度。在扫描转换直线时,对某个像素点,要么填充直线的灰度,要么不做任何改变。实际上,可以将直线看成宽度为一个像素的矩形,这样在其经过的每一列上将盖住若干个像素点。所以,直线经过的每一列不应该只选择一个点填充直线的灰度,而应该对其经过的每一个像素点填充相应的灰度。

根据一个像素点被直线覆盖的区域按照比例决定该点的灰度,这种计算覆盖面积的反走样技术称为区域采样。覆盖面积的计算有多种方法,一种简化的方法如下:将一个像素点划分成若干更细的矩形子像素栅格,然后计算代表直线的矩形所覆盖的子像素栅格数量^[1]。

1.3 加权区域取样

区域取样技术可以显著提高直线的显示效果。但是这种非加权区域采样方法有明显的缺点:像素点的灰度只和覆盖区域的面积相关,而与其它因素无关,即不管像素点中心到一条理想直线的距离远近,相同的覆盖面积就有相同的灰度。这仍然会导致锯齿效应。

要克服区域取样的缺点,可以让靠近像素点中心的区域比远离的区域对灰度的影响更大。通常定义一个权值函数来表示这种影响关系,使相交区域对像素点灰度的贡献依赖于该区域与像素点中心的距离。这种同样的覆盖面积发挥不同作用的反走样技术称为加权区域取样。

当直线经过一个像素点时,该像素点的灰度是在两者相交区域上对权值函数进行积分的结果。为了简化计算,可以采用与区域采样类似的方法,把一个像素点均匀分割成若干面积相同的子像素栅格。根据权值函数决定每个子像素栅格对原像素点灰度贡献的权值。在反走样时,找出中心被直线覆盖的子像素栅格,计算所有这些栅格对原像素点灰度贡献之和^[5]。

作为最基本的图形,直线在许多画面中大量存在。如飞机座舱标准图形画面中用来指示飞机航向的水平状态指示仪画面,就包括由 72 根直线组成的全罗盘。直线的反走样效率对整个画面的生成速度有直接的影响。所以在实际应用中,必须在反走样效果和算法复杂度中进行权衡。

基于区域采样原理的 Wu 直线反走样算法效果能满足通常的需要且计算也不复杂,在许多场合得到了广泛的应用。

2 Wu 直线反走样算法的基本思想

设直线的起点为 (x_0,y_0) ,终点为 (x_1,y_1) 。为简化起见,文中只对斜率在 $(0,1)$ 之间的直线进行讨论。直线方程为 $F(x,y)=kx-y$ (k 为斜率)。设 $x_0 < x_1,y_0 < y_1,dx=x_1-x_0,dy=y_1-y_0$,一般情况由变换不难求得。

按照上面的假设, $dx \geq dy$,可以设 x 轴为长轴, y 轴为短轴。Breshenham 算法的思想是沿着长轴方向前进一个像素单位,在短轴方向与理想直线距离最近的一个像素点以最大灰度点亮,所生成的直线有明显的走样。Wu 直线反走样算法思路非常简单,沿着长轴方向前进一个像素单位,在短轴方向与理想直线距离最近的有两个像素点,这两个像素点都点亮,如图 1 中的点 M 和 N ,但是这两个像素点对应的颜色灰度值是不同的,距离远的灰度小,距离近的灰度大,但两者灰度之和等于像素颜色的最大灰度。也就是根据点 M 和 N 的中心到理想直线的距离 l_1 和 l_2 的比值来分配灰度。显然 $l_1:l_2$ 和 M 和 N 的中心沿短轴方向到达直线的长度 $l_3:l_4$ 值相同,由于 l_3 和 l_4 更容易计算,可以按其分配灰度。若用一个像素大小为单位来表示距离,则 $l_3+l_4=1$ 。

一个像素点的灰度相对最大灰度的比率设为 $g(0\% \leq g \leq 100\%)$,该像素点到理想直线的距离设为 l 个像素,则:

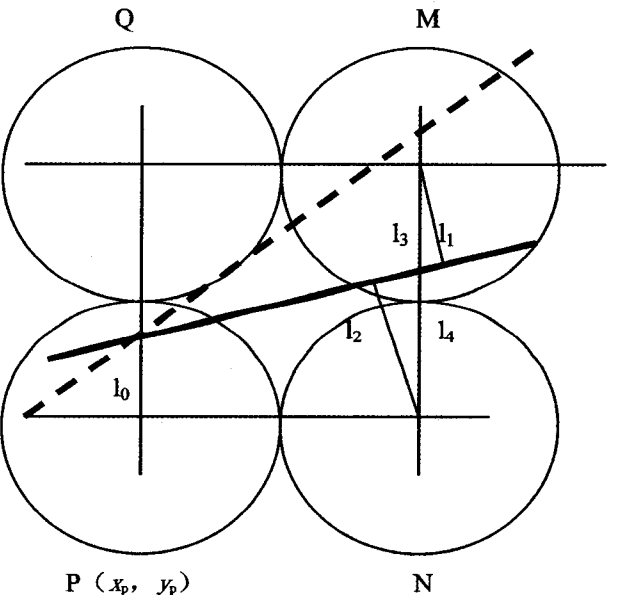


图 1 反走样直线的像素点

$$g = (1 - l) \times 100\% \tag{1}$$

利用(1)式即可计算各像素点的灰度,实现反走样。其效果如图 2 所示的部分直线。为表示清楚,图 2 中拉大了像素点之间的距离。这里画出了一条直线的相邻 6 个像素点。设像素点 A 中心距离理想线 0.2 个像素,直线斜率 $k = 0.2$ 。按照 Wu 直线反走样算法, A

点的灰度为最大灰度的 80%。像素点 B 中心距离理想线 0.4 个像素,灰度为最大灰度的 60%。其它各点灰度如图所示。由于每一列上下两个像素点的距离为 1 像素,所以其光强相加为 100% 最大灰度。

由图 2 可知,直线宽度为两个像素宽,Wu 直线反走样算法通过像素点与理想直线距离不同而调节不同的灰度,使所绘制直线灰度变化平滑,达到视觉上消除锯齿的目的。当然这种算法需要图形显示器具有不同的灰度等级支持。

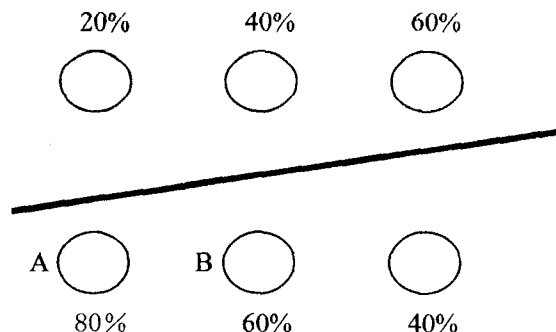


图 2 Wu 直线反走样算法生成的直线

Wu 直线反走样算法思路清晰,由软件实现起来也不困难,因而在计算机图形学中得到了广泛应用。由于需要计算斜率 k ,所以 Wu 算法需要除法和小数运算,且实际显示时需要用 g 乘以显示器的最大灰度值才能得到像素点的实际灰度值,运算较复杂,效率不高,且难以由硬件实现。文中即对此提出一种改进算法。

3 Wu 直线反走样算法的改进

Wu 直线反走样算法按照像素点中心和理想直线的距离来分配灰度,本文仍使用这个思路。以图 1 中的部分直线为例。假设像素点 $P(x_p, y_p)$ 和像素点 $Q(x_p, y_p + 1)$ 已被选中。若直线从像素点 $M(x_p + 1, y_p)$ 和 $N(x_p + 1, y_p + 1)$ 之间穿过(如图 1 中的实线所示),显然 M 和 N 都会被点亮。 M 和 N 的灰度相加为 100% 最大灰度,而各自的灰度正比于 l_3 和 l_4 。按此即可以计算出 M 和 N 的灰度,但直接计算需要实数运算,比较复杂。

考察像素点 P ,该点到理想直线的距离为 l_0 ,则 $l_4 = l_0 + k$,代入式(1),得到

$$g_N = (1 - l_0 - k) \times 100\% \quad (2)$$

所以像素点 N 相对像素点 P 的灰度变化量

$$\Delta g_1 = -k \times 100\% \quad (3)$$

同理,像素点 M 相对像素点 Q 的灰度变化量

$$\Delta g_2 = k \times 100\% \quad (4)$$

若直线从像素点 M 上方穿过(如图 1 中的虚线所示),由于 $l_4 > 1$,所以 $g_N < 0$,即 N 点不应该点亮,点亮

的应该是像素点 M 和像素点 $I(x_p + 1, y_p + 2)$ (该点图 1 未画出)。而此时 $l_3 = l_0 + k - 1$,代入式(1),得到

$$g_M = (2 - l_0 - k) \times 100\% \quad (5)$$

所以像素点 M 相对像素点 P 的灰度变化:

$$\Delta g_3 = (1 - k) \times 100\% \quad (6)$$

同理,像素点 I 相对像素点 Q 的灰度变化:

$$\Delta g_4 = (k - 1) \times 100\% \quad (7)$$

由此可以建立递推公式,若像素点 $P(x_p, y_p)$ 和像素点 $Q(x_p, y_p + 1)$ 已被选中,则生成下一对像素点时,先计算像素点 $N(x_p + 1, y_p)$ 的灰度 $g_N = (g_p - k) \times 100\%$ 。若 $g_N \geq 0$,则点亮 N 点和像素点 $M(x_p + 1, y_p + 1)$,且 $g_M = (g_Q + k) \times 100\%$ 。若 $g_N < 0$,则点亮像素点 $M(x_p + 1, y_p + 1)$ 和 $I(x_p + 1, y_p + 2)$,其灰度分别为 $g_M = (g_p + 1 - k) \times 100\%$; $g_I = (g_Q - 1 + k) \times 100\%$ 。这里认为起点 (x_0, y_0) 到理想直线的距离为 0,则直线的第一对像素点 (x_0, y_0) 和 $(x_0, y_0 + 1)$ 的灰度分别为 100% 和 0。

虽然按照以上的推导,可以生成一条反走样直线,但 k 是一个实数,而且这样计算得到的灰度,只是相对最大灰度的一个百分比,显示时还需要与最大灰度相乘才得到某个像素点实际的灰度值,所以运算仍然比较复杂。为消除实数和乘法运算,可以联系显示器的灰度级来考虑。

假设显示器为 64 级灰度,设像素点的灰度值为 f , f 是 $[0, 63]$ 中的整数,即一个像素点的 $f = \lceil g \times 64 \rceil$ ($\lceil \cdot \rceil$ 表示下取整)。

考虑直线上两个相邻的像素点,若其在同一像素行,如图 1 中的 M, Q 间的灰度变化, $\Delta g = k \times 100\%$, $f_M = \lceil (g_Q + k) \times 64 \rceil$,设 $\text{tem} = g_Q \times 64 - f_Q$,所以 $f_M = f_Q + \lceil \text{tem} + k \times 64 \rceil$ 。显然,若 tem 加上 $k \times 64$ 的小数部分 ≥ 1 ,即式(8)成立。

$$k \times 64 - \Delta f + \text{tem} \geq 1 \quad (8)$$

式(8)中 $\Delta f = \lceil k \times 64 \rceil$ 。若式(8)成立, M, Q 间灰度值变化量为 $1 + \Delta f$; 否则变化量为 Δf 。

若在不同像素行,如 M, I , $\Delta g = (k - 1) \times 100\%$, $f_I = \lceil (g_M + (k - 1)) \times 64 \rceil$,所以 $f_I = f_M + \lceil \text{tem} + k \times 64 \rceil - 64$ 。显然,若式(8)成立, E, F 间灰度值变化量为 $\Delta f - 63$; 否则变化量为 $\Delta f - 64$ 。

同理, N, P 或 M, P 间灰度值变化量也可以求出。

根据以上推导,通过判断式(8)是否成立,即可得到相邻像素点之间的灰度值变化。为了简化式(8)的计算,避免除法,可以在不等式两边乘上 dx ,即判断式(9)是否成立:

$$dy \times 64 - dx \times \Delta f + \text{tem}1 \geq dx \quad (9)$$

式(9)中 $\text{tem}1 = dx \times \text{tem}$ 。如图 1,若点 P, Q 已选中,在生成新的一对像素点时,先计算 $\text{tem}1 = dy \times 64 -$

$dx \times \Delta f + tem1$, 若此时 $tem1 \geq dx$, 则 $f_N = f_P - \Delta f - 1$, $f_M = f_Q + \Delta f + 1$, 同时需要修正 $tem1 = tem1 - dx$; 否则 $f_N = f_P - \Delta f$, $f_M = f_Q + \Delta f$ 。若 $f_N \geq 0$, 则应选中点 N ; 否则选中点 M , 此时 $f_I = f_M - 64$, $f_M = f_N + 64$ 。如此反复计算, 直到生成整条直线。由于起点 (x_0, y_0) 到理想直线的距离为 0, 所以 $tem1$ 的初值应为 0。

在求得 Δf 后, 新算法只使用整数移位、加法和比较来反走样, 计算简单。反走样效果如图 3 所示, 图 3 上面是用 Bresenham 算法生成的未反走样的直线, 下面是用新算法生成的反走样直线, 该直线的起终坐标分别是 $(0, 0)$ 和 $(193, 52)$ 。从图 3 可见, 反走样后的直线视觉效果较好。

在 Intel 酷睿 1.86G 双核, 内存 1GB 的计算机上, 用 C 语言编程完成该直线所有像素点的坐标和灰度值计算。Wu 算法需要 $8\mu s$ 左右, 新算法需要 $5\mu s$ 左右。

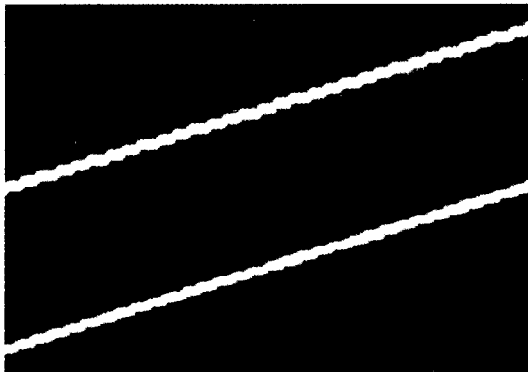


图 3 反走样效果比较

Wu 算法需要做除法、取整等实数运算, 用 FPGA 难以实现, 而新算法只在求 Δf 需要除法和取整。为了用硬件实现反走样以进一步提高效率, 可以借用逐次逼近型 A/D 转换器的工作原理来求 Δf 以避免除法。由于可以用 6 位 2 进制数来表示 64 级灰度, 即 Δf 为 6 位 2 进制数, 从高位到低位依次设为 $b_5 - b_0$, 逐位判断其取值。先判断最高位: 若 $dy \times 64 - dx \times 32 \geq 0$, 则 $b_5 = 1$, 否则 $b_5 = 0$ 。再判断 $dy \times 64 - b_5 \times dx \times 32 - dx \times 16 \geq 0$ 是否成立, 成立则 $b_4 = 1$, 否则 $b_4 = 0$; 如此反复进行, 直到每一位都完成计算。这样只需要整数加法、移位和比较就可以算出 Δf , 在计算 Δf 的同时, 不难算出 $tem1$ 的初值: $dy \times 64 - dx \times \Delta f$ 。

按照以上方法, 通过移位、减法和比较就可以算出 Δf 和 $tem1$ 。FPGA 在计算时, 可以每个时钟周期判断一位。由于一条直线只计算一次 Δf , 这些运算由 FPGA 实现的开销很小。

用 VHDL 语言编程, 在 Xilinx 公司的 Spartan3 系

列的 FPGA 芯片 xc3s400 上实现新算法。运行时每个时钟周期计算一对像素点的坐标和灰度值, 加上求 Δf 需要的时间, 当时钟频率为 100MHz 时, FPGA 可以在 $2\mu s$ 以内完成 $(0, 0)$ 到 $(193, 52)$ 直线的反走样计算, 且提高时钟频率可以进一步提高速度。

4 结束语

文中提出了一种新的直线反走样算法。该算法建立了相邻像素点之间坐标和灰度值的递推公式, 只使用整数移位、加法和比较来生成一条反走样直线, 计算简单。

通过运行验证, 新算法可以比 Wu 算法更快的生成反走样直线而视觉效果不变, 且新算法可以在诸如 FPGA 之类的硬件设备上运行, 进一步提高了效率。

参考文献:

- [1] Foley J D. 计算机图形学导论[M]. 北京: 机械工业出版社, 2004: 48-56.
- [2] 杭后俊, 付 勇. 一种基于加权区域采样的直线反走样生成算法[J]. 计算机技术与发展, 2009, 19(6): 138-141.
- [3] 王宇于. 光栅显示图形中直线反走样技术算法的改进[J]. 哈尔滨理工大学学报, 2008, 13(3): 51-53.
- [4] 杜晨辉, 经亚枝. 全罗盘画面反走样算法的研究和实现[J]. 南京航空航天大学学报, 2002, 34(4): 391-393.
- [5] 江 修, 张焕春, 经亚枝. 三像素宽反走样直线的绘制算法研究[J]. 南京航空航天大学学报, 2003, 35(2): 148-151.
- [6] Spitz J C. Nonlinear antialiasing of discrete lines[C]//4th Discrete Geometry for Computer Imagery 1994. Grenoble: [s. n.], 1994: 53-61.
- [7] 李震霄, 何援军. 任意宽度直线的绘制与反走样[J]. 武汉大学学报(工学版), 2006, 39(4): 130-133.
- [8] Jones T R, Ferry R N. Antialiasing with line samples[C]//Rendering Techniques 2000 Proceedings of the Eurographics Workshop. Brno: [s. n.], 2000: 197-205.
- [9] Wong K H, Ouyang X, Lim C W. Rendering anti-aliased line segments[C]//Computer Graphics International 2005. Stony Brook: [s. n.], 2005: 198-205.
- [10] 张 波, 张焕春, 经亚枝. 罗盘刻度线反走样快速绘制算法的改进研究[J]. 计算机辅助设计与图形学学报, 2003, 15(1): 71-75.
- [11] 谢 莹, 许荣斌, 赵宏坤. 基于嵌入式图形系统的改进 Bresenham 反走样算法[J]. 计算机技术与发展, 2006, 16(11): 100-102.
- [12] 孟宪元. FPGA 嵌入式系统设计[M]. 北京: 电子工业出版社, 2007.