Fast Line Scan-Conversion

J. G. ROKNE, BRIAN WYVILL
University of Calgary
and
XIAOLIN WU
University of Western Ontario

A major bottleneck in many graphics displays is the time required to scan-convert straight line segments. Most manufacturers use hardware based on Bresenham's [5] line algorithm. In this paper an algorithm is developed based on the original Bresenham scan-conversion together with the symmetry first noted by Gardner [18] and a recent double-step technique [31]. This results in a speed-up of scan-conversion by a factor of approximately 4 as compared to the original Bresenham algorithm. Hardware implementations are simple and efficient since the property of using only shift and increment operations is preserved.

Categories and Subject Descriptors: I.3.3 [**Computer Graphics**]: Picture/Image Generation—*display algorithms*

General Terms: Algorithms

Additional Key Words and Phrases: Computer graphics, incremental curve generation, line generators

1. INTRODUCTION

In computer graphics the majority of algorithms for drawing mathematical curves are of the incremental type [2, 5, 6, 11, 14, 22, 23, 26, 28, 29, 31]. These algorithms generate discrete loci of curves in the raster plane by selecting one of two possible pixels once a one-step increment along a certain axis has been made. The choice between the two possible pixels is made by testing the sign of a discriminator. This discriminator obeys a simple recurrence formula which may be evaluated using only integer arithmetic and binary shift. The incremental algorithms are therefore computationally inexpensive, hence popular and widely used. The first such algorithm was due to Bresenham [5]. His algorithm was simple, robust, and efficient, as evidenced by its recent incorporation as the algorithm for scan-converting lines in the Texas Instruments 34010 graphics processor (see [1]).

This work was supported by grants from the Natural Science and Engineering Research Council of Canada.

The generation of line segments (called lines in the sequel) is the basic graphics primitive. This is evidenced by the fact that graphics hardware tends to be benchmarked by the speed by which it can generate lines. A considerable interest has therefore been shown in attempting to improve the basic Bresenham algorithm [2, 6, 7, 11, 14, 20, 21, 29, 31]. These studies aim at increasing the scan-conversion speed while maintaining the same choice of scan-converted pixels as generated by the original Bresenham algorithm. Properties of discrete line segments have also been investigated, in particular in the thesis by Dorst [12] and also by Brons [8].

In this paper an algorithm is developed that combines the symmetry principle noted by Gardner [18] with the double-step technique developed by Wu and Rokne [31]. This results in an algorithm that roughly gains a factor of 2 in scan-conversion speed from the symmetry and another factor of roughly 2 from the double-step technique, for an overall gain in speed of about 4 over the original Bresenham algorithm. In [18], the awkward case of the line crossing exactly midway between two adjacent integer coordinates caused a slight deviation from the original Bresenham pixel sequence. This was rectified in [4] at a cost of one extra test for each pixel, scan-converted in order to generate exactly reversible pixel sequences. The symmetric nature of the pixel sequences was not used to speed up the scan-conversion, however. Instead the testing for the awkward cases is incorporated into the double-step algorithm from [31] in such a manner that the testing only has to be performed very infrequently.

Multiple pixel generation has also been discussed by Sproull [28]. In [28], selected pixels in the scan-conversion sequence, a fixed increment apart, are first computed. Then the in-between pixels are filled in. Although this has similarities with the double-step technique, it does not take advantage of the special properties available when the step-size is exactly 2.

Other approaches to speeding up scan-conversion include run-length algorithms (see [7, 26]) and approaches based on Euclid's algorithm (see [9, 10]).

The paper is organized as follows. First, some properties of discrete loci of nonparametric curves $f(x, y) = 0$ in the raster plane are summarized from [31]. These properties make it possible to interpolate half of the pixels on the curve at low cost. This means that the speed of scan-conversion of curves may be roughly doubled. The application of the double-step idea to scan-conversion of lines was discussed in [31], and a summary is given here. The symmetry principle for lines having integer end coordinates first noted in [18] is then discussed. A combined algorithm is given that speeds up the scan-conversion of lines by a factor of approximately 4 over the original Bresenham implementation. This gain in speed is not realized in actual practice because of the inevitable pixel write operations. If they dominate, then the algorithm may not be viably implemented in current hardware. In future hardware the bottleneck may well be in the scan-conversion such that the proposed algorithm may be more appropriate.

2. DOUBLE-STEP SCAN-CONVERSION OF LINES

The discussions of the double-step principle and its application to the scan-conversion of lines given in [31] is now summarized. The basis for this discussion is the following property of curves on the raster plane.

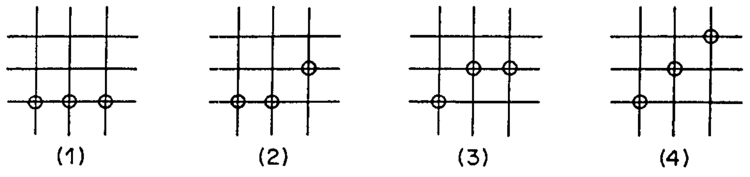


Fig. 1. The four double-step patterns.

Let $f(x, y) = 0$ be a two-dimensional curve having a continuous first derivative, and assume it is divided into segments that satisfy one of the following cases:

$$\begin{array}{llll} \text{a)} & 0 \le \dfrac{dy}{dx} \le 1 & \text{b)} & 1 < \dfrac{dy}{dx} < \infty \\ \text{c)} & -\infty < \dfrac{dy}{dx} < -1 & \text{d)} & -1 \le \dfrac{dy}{dx} \le 0. \end{array} \tag{1}$$

The discussion is restricted to case a) since the other cases can be reduced to case a) by swapping x and y and/or changing incremental direction. Consider therefore a 2×2 mesh with starting point (x_0, y_0) at the lower left corner representing a portion of the discrete range space of $f(x, y) = 0$. On this mesh the curve $f(x, y) = 0$ can only form the four patterns shown in Figure 1 (except for a very rare case of a right angle stair which cannot occur for lines; see also discussion in [33]).

Starting from (x_0, y_0), the x coordinate is now incremented by two raster units. Then if the pixel at the right-lower (the right-upper) corner of the 2×2 mesh is selected, it is clear that pattern 1 (4) occurs. This means that in each case the middle pixel can be plotted with no extra work. If pattern 2 or 3 occurs (abbreviated pattern 2 (3) in the sequel), then some extra work has to be done in order to distinguish which of the two patterns have to be plotted. It was conjectured by Freeman [16, 17] and proven by Regiori [26] (see also [31, 32]) that only two pattern types may occur simultaneously: either 1 and 2 (3) or 2 (3) and 4. From these results an effective double-step strategy was developed. This strategy is now summarized.

Assume that the line is defined as $y = (dy/dx)x$, $0 \le dy/dx \le 1$. If $(dy/dx) \le \frac{1}{2}$, then the double-step algorithm is given by

Algorithm 1. *Let* $D_1 = 4dy - dx$, $\alpha_1 = 4dy$ *and* $\alpha_2 = 4dy - 2dx$. *For* $i = 1, 2, \ldots$

$$D_{i+1} = \begin{cases} D_i + \alpha_1 & \textit{if } D_i < 0 \ (\textit{pattern } 1) \\ D_i + \alpha_2 & \textit{otherwise } (\textit{pattern } 2\ (3)), \end{cases}$$

and if $(dy/dx) \ge \frac{1}{2}$ *then the algorithm is*

Algorithm 2. *Let* $D_1 = 4dy - 3dx$, $\beta_1 = 4dy - 2dx$ *and* $\beta_2 = 4(dy - dx)$. *For* $i = 1, 2, \ldots$

$$D_{i+1} = \begin{cases} D_i + \beta_1 & \textit{if } D_i < 0 \ (\textit{pattern } 2\ (3)) \\ D_i + \beta_2 & \textit{otherwise } (\textit{pattern } 4). \end{cases}$$

To distinguish between patterns 2 and 3 requires the test

$$D_i < \begin{cases} 2dy & \text{if} \quad 0 \le dy/dx < \frac{1}{2} \\ 2(dy - dx) & \text{if} \quad \frac{1}{2} \le dy/dx \le 1 \end{cases}$$

giving pattern 2 if the test is passed, pattern 3 if not.

The above algorithm was implemented and Pascal code was given as Figure 3 in [31]. It was shown to be almost twice as fast as Bresenham's original algorithm if pixel I/O was ignored in both cases.

3. SYMMETRY WITH DOUBLE-STEP

We again consider line segments between (0, 0) and (dx, dy) where dx and dy are integers. The line segment is therefore

$$y = \left(\frac{dy}{dx}\right)x, \qquad 0 \le x \le dx.$$

If the scan-conversion is started at (0, 0) then the sequence of coordinates (or set of pixels) is denoted by

$$(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n) \tag{2}$$

where $x_i = i$, $i = 0, 1, \ldots, n$ with $(x_0, y_0) = (0, 0)$ and $(x_n, y_n) = (dx, dy)$. From the properties of the scan-conversion in the Bresenham sense (see [31]) it follows that

$$|y_i - y(x_i)| \le \tfrac{1}{2}, \qquad i = 0, 1, \ldots, n.$$

If the scan-conversion is started at (dx, dy) then similarly the sequence

$$(x_0', y_0'), (x_1', y_1'), \ldots, (x_n', y_n') \tag{3}$$

is generated where again $x_i' = a' - i$, $i = 0, 1, \ldots, n$ with $(x_0', y_0') = (dx, dy)$ and $(x_n', y_n') = (0, 0)$. This sequence satisfies

$$|y_i' - y(x_i')| \le \tfrac{1}{2}, \qquad i = 0, 1, \ldots, n.$$

It was noted in Gardner [18] that the scan-converted lines were symmetric around the midpoint of the line if slight deviations from the line scan-converted by Bresenham's algorithm were accepted. These deviations occur when the line crosses the vertical lines going through the integer x coordinates exactly midway between two integer y values. It was furthermore noted in [9] that the lines were palindromic, i.e., symmetric, if dx and dy were relative prime. Here we formalize this symmetry principle.

THEOREM 1. *Let $y = (dy/dx)x$, $0 \le x \le dx$ define a line segment with integer endpoints* (0, 0) *and* (dx, dy). *Then the pixel sequences defined by* (2) *and* (3) *satisfy*

$$(x_i, y_i) = (x_{n-i}', y_{n-i}'),$$

unless $|y_i - y(x_i)| = \frac{1}{2}$ *in which case*

$$(x_i, y_i - 1) = (x_{n-i}', y_{n-i}').$$

PROOF. Since the x (x') directions are incremented (decremented) one unit at a time it is clear that $x_i = x'_{n-i}$, $i = 0, 1, \ldots, n$. Assume first that $|y_i - y(x_i)| \neq \frac{1}{2}$, $i = 0, 1, \ldots, n$. Then the closest integer point to $y(x_i)$ is y_i. Similarly y'_{n-1}, is the closest point to $y(x'_{n-i})$. Since $x_n = x'_{n-i}$ it follows that $y_i = y'_{n-i}$, $i = 0, 1, \ldots, n$.

If $|y_i - y(x_i)| = \frac{1}{2}$ for some i, then, since the relation "closest point" is defined to be the $\leq$ relation, it follows that if y_i is closest to $y(x_i)$ in the positive y direction, then y'_{n-i} is closest in the negative y direction and $y'_{n-i} - y_i = 1$. □

Based on this theorem it is clear that only the first half of the line segment needs to be scan-converted. The second half may be copied as the first half is generated, but in reverse order except when the relation $|y_i - y(x_i)| = \frac{1}{2}$ is true.

The awkward case of $|y_i - y(x_i)| = \frac{1}{2}$ for some i, called a $\frac{1}{2}$ crossing in the sequel, is now considered in detail.

First of all we note that if this case occurs, then dx is even. In fact if $|y_i - y(x_i)| = \frac{1}{2}$, then the original line segment from (0, 0) to (dx, dy) passes through either $(x_i, y_i + \frac{1}{2})$ or $(x_i, y_i - \frac{1}{2})$ where x_i and y_i are both integers. We only consider the first case where the intercept is $(x_i, y_i + \frac{1}{2})$ since the second case gives the identical result. Clearly the triangles formed by the points (0, 0), $(x_i, y_i + \frac{1}{2})$ and by the points (0, 0), $(dx, 0)$, (dx, dy) are congruent. It is immediately clear that $(dy/dx) = (y_i + \frac{1}{2})/x_i$. From this it follows that $2(dy)x_i = dx(2y_i + 1)$, which implies that dx must be even since dx, dy, x_i, and y_i must all be integers.

The condition that dx is even is only necessary. A further sufficient (but not necessary) condition is that dy is odd, which can be verified by a simple calculation of $y(dx/2)$.

The decision procedure to be employed is therefore:

(i) If dx is odd then scan-convert the line from (0, 0) to (dx, dy) using the double-step and symmetry ideas as discussed above, ignoring tests for $\frac{1}{2}$ crossings.

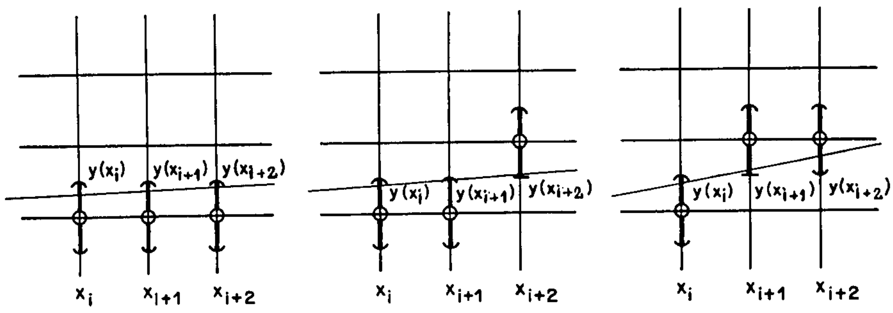
(ii) If dx is even and dy is odd then scan-convert the line from (0, 0) to (dx, dy) using the double-step technique and symmetry, while testing for $\frac{1}{2}$ crossings.

(iii) If both dx and dy are even then find largest common factor 2^k, scan-convert the line-segment (0, 0) to $(dx/2^k, dy/2^k)$ and repeat 2^k times, thus reducing this case to one of the cases (i) or (ii) above.

A technique similar to the above case (iii) was employed in [15] to develop a recursive scan-conversion algorithm. The algorithm was only valid for lines whose x extent was a power of 2.

We now consider the effect of $\frac{1}{2}$ crossings. When a $\frac{1}{2}$ crossing has been found then this might imply a further symmetry of the line. Using the same arguments as in Theorem 1 it is easy to see that the initial scan-converted pixel pattern may be reversed and copied starting at the $\frac{1}{2}$ crossing. Although this may be used to speed up the algorithm further it is not used here.

The checking for $\frac{1}{2}$ crossings may be streamlined when the double-step algorithm is used in conjunction with the symmetry idea. Consider therefore Figure 2 and assume $0 \leq (dy/dx) \leq \frac{1}{2}$ (the other cases may be treated in a similar



(a) Pattern 1 (b) Pattern 2 (c) Pattern 3

Fig. 2. Testing for $\frac{1}{2}$ crossings ($\frown$, $\smile$ no $\frac{1}{2}$ crossing;—possible $\frac{1}{2}$ crossing).

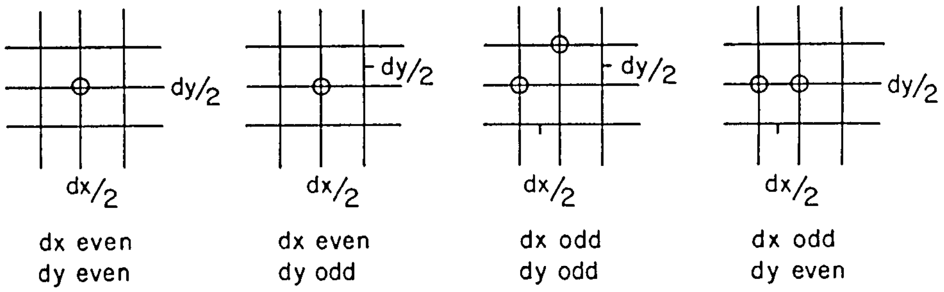


Fig. 3. Midpoint termination.

manner). In this figure the possible range of values of $y(x_{i+\tau})$ is indicated under the various constraints and cases given a $y_{i+\tau}$ ($\tau = 0, 1, 2$). Each of the three possible patterns are now treated separately.

Pattern 1. From the previous double-step it follows that $|y(x_i) - y_i| < \frac{1}{2}$ (if this were not true, then a $\frac{1}{2}$ crossing would have been found and the symmetry pixels would have been plotted). In the current step, the decision procedure has already decided that $y(x_{i+2}) - y_{i+2} < \frac{1}{2}$. Hence also $y(x_{i+1}) - y_{i+1} < \frac{1}{2}$. Thus no additional test is necessary for Pattern 1 (see Figure 2a).

Pattern 2. In Pattern 2, similarly, no test is needed at x_i. In deciding that this was Pattern 2 we have $y(x_{i+1}) - y_{i+1} < \frac{1}{2}$, hence a $\frac{1}{2}$ crossing cannot occur at x_{i+1}. It is, however, necessary to test if $y(x_{i+2}) - y_{i+2} = \frac{1}{2}$ (see Figure 2b). If true, then the patterns are symmetric about x_{i+2}.

Pattern 3. Again no test is necessary at x_i. However, it is necessary to test if $y(x_{i+1}) - y_{i+1} = \frac{1}{2}$. If this is true, the already scan-converted line is symmetric about x_{i+1}. Whether it is symmetric or not, it is clear that $y(x_{i+2}) - y_{i+2} \neq \frac{1}{2}$ since otherwise a Pattern 4 would occur, which cannot happen since $y(x_i) - y_i < \frac{1}{2}$ and $y(x_{i+1}) - y_{i+1} > \frac{1}{2}$. This means that no further tests are needed (see Figure 2c).

The overhead for the half-way crossing test is therefore less than one test per double-step when the checking is necessary.

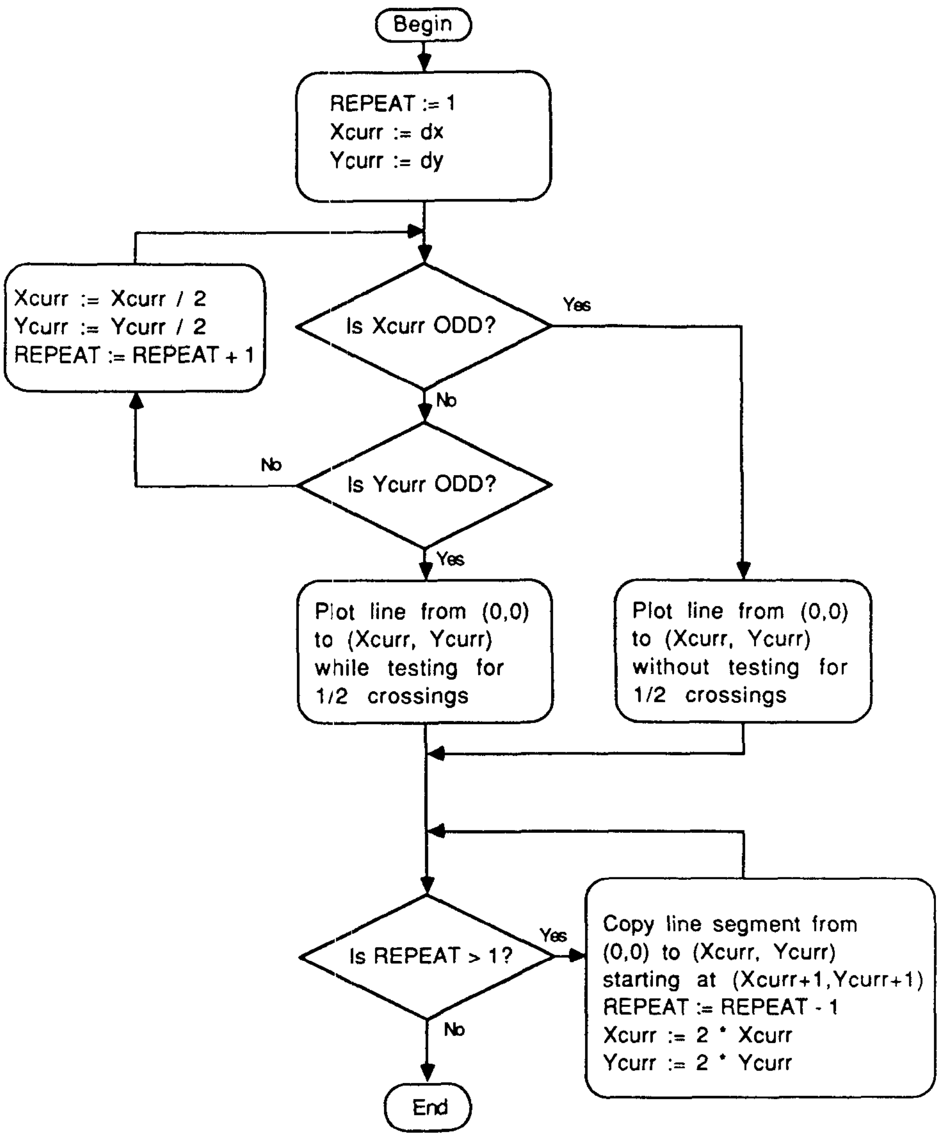

Begin
REPEAT := 1
Xcurr := dx
Ycurr := dy
Xcurr := Xcurr / 2
Ycurr := Ycurr / 2
REPEAT := REPEAT + 1
Is Xcurr ODD?
Yes
No
Is Ycurr ODD?
No
Yes
Plot line from (0,0) to (Xcurr, Ycurr) while testing for 1/2 crossings
Plot line from (0,0) to (Xcurr, Ycurr) without testing for 1/2 crossings
Is REPEAT > 1?
Yes
No
Copy line segment from (0,0) to (Xcurr, Ycurr) starting at (Xcurr+1,Ycurr+1)
REPEAT := REPEAT - 1
Xcurr := 2 * Xcurr
Ycurr := 2 * Ycurr
End

Figure 4a

The termination check occurs at the midpoint of the currently considered line segment. The situation here is shown in Figure 3. The four cases in Figure 3 depend on the parity of *dx* and *dy*.

4. ALGORITHM DESCRIPTION

An algorithm incorporating double-step and symmetry ideas was developed. This algorithm is now described using the flowcharts given in Figures 4(a) and 4(b).

Figure 4a describes the parity checking on *dx* and *dy* discussed in the last section. This parity checking is performed in order to avoid some of the $\frac{1}{2}$ crossing

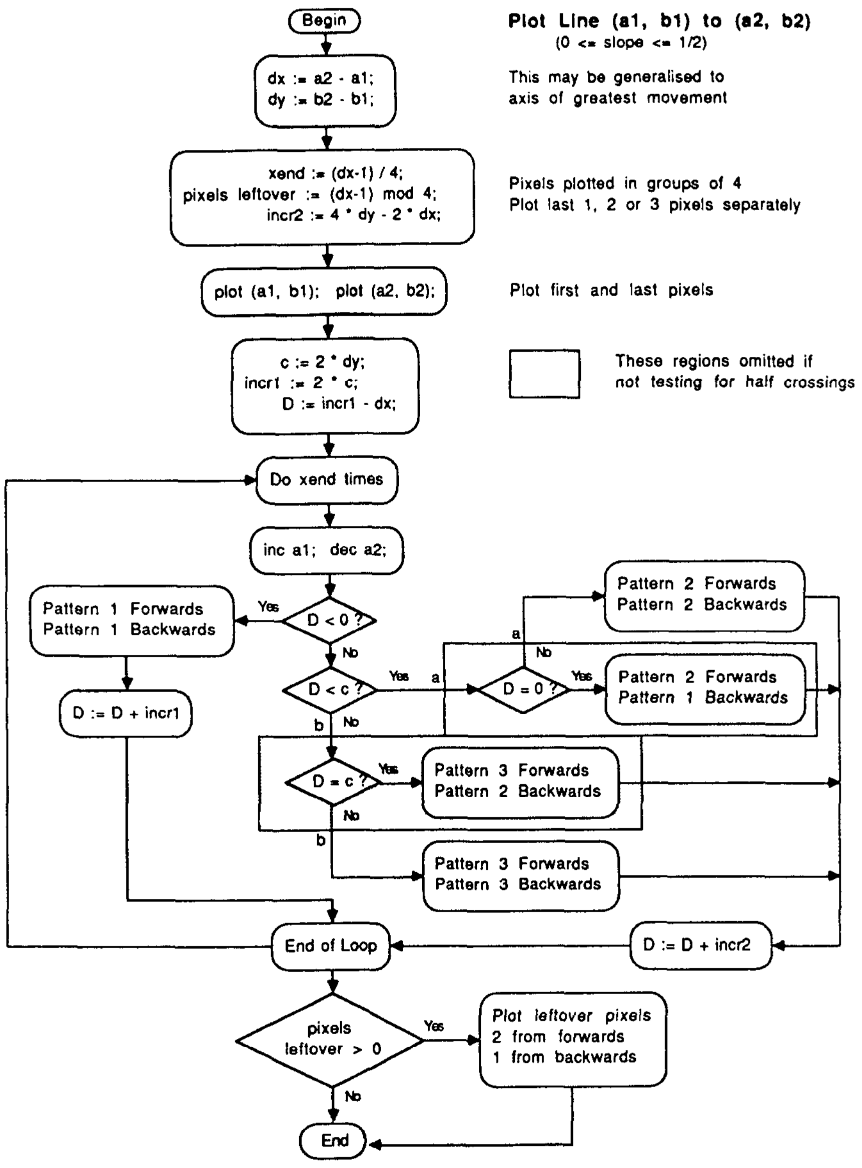


Figure 4b

tests. The flowchart includes the repeat loop for the case that both *dx* and *dy* have a common factor 2^k for some k.

If the cost of copying the pixels that are repeated in the patterns exceeds the cost of regenerating the pixel sequence, then the check for even *dx* and *dy* may be omitted, together with the pattern repetition part. This would result in a different flowchart, which is not shown here since it is easily constructed.

The parity checking on dx and dy may be omitted altogether for an easy implementation of the algorithm by always checking for $\frac{1}{2}$ crossings in the flowchart in Figure 4b.

Figure 4b contains a condensation of the algorithm to scan-convert lines without checking for $\frac{1}{2}$ crossings (omitting boxes within the dashed lines) and with checking for $\frac{1}{2}$ crossings (including boxes within dashed lines).

In the first case the dashed boxes are omitted and the $\frac{1}{2}$ crossings are not tested for. The flow of the algorithm is maintained by joining points $a - a$ and $b - b$ as marked on the flowchart and omitting the flowchart elements contained in the grey areas. The algorithm is essentially the algorithm described in Figure 3 of [31] with the addition of backward plotting patterns. The backward patterns are described via a coordinate system rotated 180° with respect to the original coordinate system in order to reverse plotting direction. Patterns in the original coordinate system are called *forward patterns* and in the rotated coordinate system *backward patterns*. If there are no $\frac{1}{2}$ crossings, then the forward pattern sequence in the original coordinate system and the backward pattern sequence in the rotated coordinate system are the same as shown in Theorem 1.

In the second case the $\frac{1}{2}$ crossings are tested for by including all the items of the flowchart as given in Figure 4b. The added features incorporate the discussion of the $\frac{1}{2}$ crossings with respect to the forward Patterns 1, 2, and 3 in the previous section.

Since forward Pattern 1 cannot have any $\frac{1}{2}$ crossings, it follows that the reverse pattern is always backward Pattern 1 and no checking is necessary.

In forward Pattern 2, we had to check to see if the last pixel of the pattern was a $\frac{1}{2}$ crossing. If this were the case, then the last pixel of the backward double-step would be set opposite to the expected pattern, that is, backward Pattern 1 instead of backward Pattern 2.

Similarly we have to check if the middle pixel is a $\frac{1}{2}$ crossing in forward Pattern 3. If so, then the middle pixel has to be set to the opposite pixel in the reverse pattern, thus generating backward Pattern 2 instead of backward Pattern 3.

It is also possible to use the $\frac{1}{2}$ crossings as symmetry centers and simply repeat pixel sequences when they have been found. This has not been incorporated here.

5. COMPLEXITY

For the complexity discussion we assume that the algorithm has been implemented as in Figure 4a, without the repeating loop if dx and dy are both even. This means that in the average case the $\frac{1}{2}$ crossings only have to be checked for in 50 percent of the cases. It is furthermore assumed that the $\frac{1}{2}$ crossings are checked and corrected for exactly as in Figure 4b. This means that the further possible symmetries around $\frac{1}{2}$ crossings are not used. The pixel sequences after $\frac{1}{2}$ crossings are therefore recalculated, and no use is made of symmetry beyond the basic symmetry.

The complexity is also measured with respect to lines between (0, 0) and integer endpoints (dx, dy). Furthermore, it is assumed that $0 \leq (dy/dx) \leq \frac{1}{2}$ since the results are the same for other slopes.

The scan-conversion of the above line requires (dx) additions and $2(dx)$ tests using the original Bresenham algorithm.

In [31] it was shown that scan-conversion using the double-step algorithm required $\lfloor(dx)/2\rfloor$ additions and $5\lfloor(dx)/4\rfloor$ tests on the average.

Except for the increment size and the equality tests in the dashed boxes of Figure 4b, the repetitive loop is exactly as the loop in [31]. This means that the symmetric double-step algorithm requires $\lfloor(dx)/4\rfloor$ additions and $5\lfloor(dx)/8\rfloor$ tests if no checking for $\frac{1}{2}$ crossings is performed. When checking for $\frac{1}{2}$ crossings is done, this requires on the average $dx/4 * \frac{1}{2} * \frac{1}{4} * \frac{2}{3} \approx 0.02 * dx$ additional tests, a negligible extra cost.

6. NUMERICAL RESULTS

Bresenham's algorithm, the double-step algorithm, and the symmetric double-step algorithm were implemented in C on the Apple Mac+ computer and some tests were run. The results given in Tables I, II, and III were obtained. The lines were drawn from the origin (0, 0), in each case to the coordinate of the endpoint as given in the first column of the tables. The lines form the first quadrant spokes of wheels centered at the origin having radii 10, 100, and 1000 respectively. The overhead was measured for each algorithm by counting one for each control statement executed (1 for each if, 1 for each assignment, etc.). Where the algorithms did equivalent work, as in calling the set pixel subroutine, no overhead was recorded. The lines in Table I are 10 pixels long, 100 pixels long in Table II, and 1000 pixels long in Table III. For comparison purposes the ratio of counts between Bresenham and double-step and between Bresenham and symmetric double-step were also computed.

It can be seen that for the longer lines the symmetrical double-step algorithm has between 3 and 3.9 times less overhead than Bresenham's original algorithm depending on the line angle. The double-step has between 1.3 and 2 times less overhead as would be expected from the foregoing analysis. The overhead varies with both line angle and line length. Indeed, short lines produce noticeably less speed-up than long lines, as would be expected.

It is acknowledged by the authors that in practice, in a hardware line-drawing algorithm, the bottleneck is currently in the time taken to set a given pixel, and hence the reduction in overhead may not produce as significant a reduction in the total time required to raster a given line.

In Tables I, II, and III the following abbreviations are used:

Bres	Bresenham's algorithm
Doub	Double-Step
Symm	Symmetrical Double-Step

In the previous section our analysis for the additions and tests showed that the Bresenham algorithm would cost roughly $3.0dx$, the Double-Step $0.875dx$, and the Symmetrical Double-Step 0.875. This results in theoretical ratios of

Bres/Doub	1.71
Bres/Symm	3.43

Table I. 10 Pixel Lines

	Times			Rel. Times	
Endpoint	Bres	Doub	Symm	Bres/Doub	Bres/Symm
(10, 0)	31	29	24	1.1	1.3
(9, 0)	29	27	21	1.1	1.4
(9, 1)	29	28	21	1.0	1.4
(9, 2)	29	29	23	1.0	1.3
(9, 3)	29	30	23	1.0	1.3
(9, 4)	29	31	23	0.9	1.3
(8, 4)	27	31	26	0.9	1.0
(8, 5)	27	30	26	0.9	1.0
(7, 6)	25	26	23	1.0	1.1
(7, 7)	25	25	23	1.9	1.1
(6, 7)	29	29	26	1.0	1.1
(5, 8)	31	33	29	0.9	1.1
(4, 9)	33	34	26	1.0	1.3
(3, 9)	33	33	26	1.0	1.3
(2, 9)	33	32	26	1.0	1.3
(1, 9)	33	31	24	1.1	1.4
(0, 9)	33	30	24	1.1	1.4

Table II. 100 Pixel Lines

	Times			Rel. Times	
Endpoint	Bres	Doub	Symm	Bres/Doub	Bres/Symm
(100, 0)	211	119	68	1.8	3.1
(99, 8)	209	125	7[illegible]	1.7	2.8
(98, 17)	207	134	84	1.5	2.5
(96, 25)	203	140	90	1.5	2.3
(93, 34)	197	145	93	1.4	2.1
(90, 42)	191	151	94	1.3	2.0
(86, 49)	183	142	90	1.3	2.0
(81, 57)	173	130	76	1.4	2.3
(76, 64)	163	107	67	1.5	2.4
(70, 70)	151	89	54	1.7	2.8
(64, 76)	167	110	70	1.5	2.4
(57, 81)	177	126	79	1.4	2.2
(50, 86)	187	144	95	1.3	2.0
(42, 90)	195	154	97	1.3	2.0
(34, 93)	201	148	96	1.4	2.1
(25, 96)	207	143	93	1.4	2.2
(17, 98)	211	137	87	1.5	2.4
(8, 99)	213	128	79	1.7	2.7
(0, 99)	213	120	71	1.8	3.0

These ratios are not achieved for short lines. However, they are approximated reasonably well in Table III. The reason for this is that both the Double-Step and the Symmetrical Double-Step algorithms incur overhead over and above that of the Bresenham algorithm. Hence the theoretical speed-up is only attained when this overhead is amortized over longer lines.

Table III. 1000 Pixel Lines

	Times			Rel. Times	
Endpoint	Bres	Doub	Symm	Bres/Doub	Bres/Symm
(1000, 0)	2011	1019	518	2.0	3.9
(996, 87)	2003	1102	599	1.8	3.3
(984, 173)	1979	1176	674	1.7	2.9
(965, 258)	1941	1241	741	1.6	2.6
(939, 342)	1889	1299	801	1.5	2.4
(906, 422)	1823	1347	774	1.4	2.4
(866, 499)	1743	1252	728	1.4	2.4
(819, 573)	1649	1083	607	1.5	2.7
(766, 642)	1543	909	498	1.7	3.1
(707, 707)	1425	725	373	2.0	3.8
(642, 766)	1547	912	501	1.7	3.1
(573, 819)	1653	1086	610	1.5	2.7
(500, 866)	1747	1254	734	1.4	2.4
(422, 906)	1827	1350	777	1.4	2.4
(342, 939)	1893	1302	804	1.5	2.4
(258, 965)	1945	1244	744	1.6	2.6
(173, 984)	1983	1179	677	1.7	2.9
(87, 996)	2007	1105	602	1.8	3.3
(0, 999)	2013	1020	521	2.0	3.9

7. CONCLUSION

Three improvements to the scan-conversion of lines have been discussed. These are the double-step technique previously reported in [31], the use of line symmetry of lines with respect to scan-conversion direction first discussed by [18] and the combination of the above two techniques into a new efficient algorithm. The latter algorithm speeds up the scan-conversion (in the sense of the computational effort) of a line by a factor of roughly three over the original Bresenham version when the lines are sufficiently long.

REFERENCES

1. Asal, M., Short, G., Preston, T., Simpson, R. Roskell, D., and Guttag, K. The Texas Instruments 34010 Graphics System Processor. *IEEE Comput. Gr. Appl. 6* (Oct. 1986), 24–39.
2. Barros, J., and Fuchs, H. Generating smooth line drawings on video displays. In *SIGGRAPH '79 Proceedings. Comput. Gr. 13* (Aug. 1979), 260–269.
3. Belzer, K. Comment on 'An improved algorithm for the generation of non-parametric curves'. *IEEE Trans. Comput. C-25* (Jan. 1976), 103.
4. Bototheroyd, J., and Hamilton, P. A. Exactly reversible plotter paths. *Australian Comput. J. 2* (1970), 20–21.
5. Bresenham, J. E. Algorithm for computer control of digital plotter. *IBM Syst. J. 4* (1965), 25–30.
6. Bresenham, J. E. Incremental line compaction. *Comput. J. 25* (Jan. 1982), 116–120.
7. Bresenham, J. E. Run length slice algorithms for incremental lines. In *Fundamental Algorithms for Computer Graphics*, R. A. Earnshaw, Ed. NATO ASI Series, Springer Verlag, New York, 1985, 59–104.
8. Brons, R. Linguistic methods for the description of a straight line on a grid. *Comput. Gr. Image Process. 9* (1979), 183–195.

9. CASTLE, C. M. A., AND PITTEWAY, M. L. V. An application of Euclid's algorithm to drawing straight lines. In *Fundamental Algorithms for Computer Graphics*, R. A. Earnshaw, Ed. NATO ASI Series, Springer Verlag, New York, 1985, 135–139.
10. CASTLE, C. M. A., AND PITTEWAY, M. L. V. An efficient structural technique for encoding 'best-fit' straight lines. *Comput. J. 30* (1987), 168–175.
11. DANIELSSON, PER E. Incremental curve generation. *IEEE Trans. Comput. C-19* (Sept. 1970), 783–793.
12. DORST, L. Discrete straight line segments: Parameters, primitives and properties. Ph.D. thesis, T. H. Delft, June 1986. Offsetdrukkerij, Kanters BV, Ablasserdam, Holland.
13. EARNSHAW, R. A. Line tracking for incremental plotters. *Comput. J. 23* (1980), 46–52.
14. FIUME, E. L. A mathematical semantic and theory of raster graphics. Tech. Rep. CRSI-185, Computer Systems Research Institute, Univ. of Toronto, Toronto, Canada, 1986.
15. FOLEY, J. D., AND VAN DAM, A. In *Fundamentals of Interactive Computer Graphics*. Addison-Wesley, Reading, Mass., 1982.
16. FREEMAN, H. Boundary encoding and processing. In *Picture Processing and Psychopictorics*. B. S. Lipkin and A. Rosenfeld, Eds., Academic Press, New York 1970, 241–266.
17. FREEMAN, H. On the encoding of arbitrary geometric configurations. *IRE Trans. EC-102* (1961), 260–268.
18. GARDNER, P. L. Modifications of Bresenham's algorithm for displays. *IBM Tech. Disclosure Bull. 18* (1975), 1595–1596.
19. JORDAN, B. W., LENNON, W. J., AND HOLM, B. C. An improved algorithm for the generation of non-parametric curves. *IEEE Trans. Comput. C-22* (Dec. 1973), 1052–1060.
20. MCILROY, M. D. A note on discrete representation of lines. *AT&T Tech. J. 64* (Feb. 1985), 481–490.
21. PILLER, E. AND WIDNER, H. Real-time raster scan unit with improved picture quality. *Comput. Gr. 14* (1980), 15–38.
22. PITTEWAY, M. Algorithm for drawing ellipses or hyperbolae with digital plotter. *Comput. J. 10* (1967), 282–289.
23. PITTEWAY, M. Algorithms of conic generation. In *Fundamental Algorithms for Computer Graphics*, R. A. Earnshaw, Ed. NATO ASI Series, Springer Verlag, New York, 1985, 219–237.
24. PITTEWAY, M. L. V., AND GREEN, A. J. R. Bresenham's algorithm with run line coding shortcut. *Comput. J. 25* (1982), 114–115.
25. RAMOT, J. Non-parametric curves. *IEEE Trans. Comput. C-25* (Jan. 1976), 103–104.
26. REGIORI, G. B. Digital computer transformations for irregular line drawings. Tech. Rep. 403-22, Department of Electrical Engineering and Computer Science. New York Univ., Apr. 1972.
27. RUBIN, F. Generation of non-parametric curves. *IEEE Trans. Comput. C-25* (Jan. 1976), 103.
28. SPROULL, R. F. Using program transformations to derive line-drawing algorithms. *ACM Trans. Gr. 1* (Oct. 1982), 259–273.
29. SUENAGA, Y., KAMAE, T., AND KOBAYASHI, T. A high-speed algorithm for the generation of straight lines and circular arcs. *IEEE Trans. Comput. C-28* (1979), 728–736.
30. THOMPSON, J. R. Straight lines and graph plotters. *Comput. J. 7* (1964), 227.
31. WU, X., AND ROKNE, J. G. Double-step incremental generation of lines and circles. *Comput. Vision, Gr. Image Process. 37* (Mar. 1987), 331–344.
32. WU, X., AND ROKNE, J. G. Double-step incremental generation of canonical ellipses. *IEEE Comput. Gr. Appl. 9* (May 1989), 56–69.
33. WU, X., AND ROKNE, J. G. On properties of discetized convex curves. *IEEE Trans. Pattern Anal. Mach. Intell. 11* (Feb. 1989), 217–223.

Received February 1987; revised March 1988; accepted July 1989

Editor: L. Guibas