ELSEVIER

Journal of Computational and Applied Mathematics 102 (1999) 3–19

JOURNAL OF COMPUTATIONAL AND APPLIED MATHEMATICS

An integral linear interpolation approach to the design of incremental line algorithms

Chengfu Yao, Jon G. Rokne *

Department of Computer Science, The University of Calgary, Calgary, Alberta, Canada, T2N 1N4

Received 28 June 1997; received in revised form 12 May 1998

Abstract

A unified treatment of incremental line-drawing algorithms understood from the viewpoint of rounded interpolation, covering Bresenham's algorithm, run-length algorithms, and multistep versions of both.

AMS classification: 65Y25; 68R99; 68V05

Keywords: Scan conversion of lines; Integral linear interpolation; Incremental algorithms; Run-length algorithms

1. Introduction

The generation of line segment raster images (called lines hereafter) is an important 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. Considerable interest has therefore been shown in designing efficient line scan-conversion algorithms. These algorithms select the pixels nearest to the line based on the geometry of a line relative to a coordinate grid, an abstraction of the raster display where grid points represent the centers of pixels. Most of the algorithms are incremental algorithms [1–4, 8, 10, 13, 16]. Incremental algorithms are distinguished by the fact that they generate the rastered image of a line from one endpoint of a line to the other by selecting one or multiple pixels in each incremental step along a certain axis (x-axis for the lines with slope between 0 and 1). The choice of which pixels to set is made by testing the sign of a function called a discriminator. The discriminator obeys a simple recurrence formula which may be evaluated using only integer arithmetic and binary shifts. The first such algorithm was due to Bresenham [2]. His algorithm was easy to implement and it effectively set a standard for subsequent line scan conversion algorithms.

* Corresponding author. E-mail: rokne@cpsc.ucalgary.ca.

0377-0427/99/$ – see front matter
PII: S0377-0427(98)00205-2

The number of pixels generated in each incremental step may be either fixed or variable. Algorithms which generate a fixed number of pixels in each incremental step are usually named according to the length of the incremental step along a chosen axis. We thus have single-step line algorithms which are represented by the first incremental line algorithm due to Bresenham; the double-step line algorithm of Wu and Rokne [16] and the quadruple-step line algorithm of Bao and Rokne [1]. The double-step algorithm [16] is similar to Bresenham's algorithm but takes advantage of the special double-step pixel patterns and therefore reduces the number of incremental steps by one half. The quadruple-step algorithm [1] generates four pixels in each incremental step at the cost of one to three decision tests, with the average being slightly less than two. More recently, Graham and Iyengar [11] presented a double- and triple-step incremental line algorithm with which has a double-step line generator potentially being able to set a third pixel in some of the loop iterations. Gill [9] suggested N-step incremental line algorithms based on Bresenham's line algorithm. In Bresenham's algorithm, the sign of a discriminator E predicts the pixel to be chosen at any step. For each possible N-step move, the changes to E for each step in the scan conversion process gives a set of equations that must be satisfied such that the N-step pattern is next in the sequence making up a line. These equations form a test set that predicts the N-step move in advance.

The incremental line algorithms which generate a variable number of pixels in each incremental step are based on the observation that the rastered image of a line with slope between 0 and 1 can be divided into slices of horizontal runs (pixels with the same ordinate) or diagonal runs (pixels forming a 45° segment). A run of pixels is generated in each incremental step. Line algorithms of this type are usually referred to as run-length slice algorithms [4, 8]. A two-state discriminator can be used to determine the length of the next run due to the property that the lengths of the runs are confined to two successive integers except for the first and the last run. This results in a scheme similar to the one used in algorithms generating fixed number of pixels in each step.

The run-length properties were first proven by Reggiori [12]. In [15] Rosenfeld derived them from the chord property of a straight digital arc, the digitization of a straight line segment. In the horizontal run-length slice line algorithm, the incremental direction is along the y-axis for calculating the horizontal runs of the line with slope between 0 and 1 since two consecutive horizontal runs have an increment of one in their ordinates. In [4] Bresenham used the property of the so-called complementary line of a line to calculate the diagonal runs of that line whose slope is between 0.5 and 1. The run-length slice algorithms in [4] can be viewed as a single-step algorithm in the sense that the incremental step is one in the direction of y-axis. Fung, Nicholl and Dewdney [8] presented a double-step version of the run-length slice algorithm to further increase the efficiency of line generation. A major drawback of the run-length slice algorithm is the division operation required in the initialization part of the algorithm. A method to avoid division was suggested in [8].

This paper focuses on the methodology in the design of line algorithms of incremental type. As has been mentioned above, the core of the existing incremental line algorithms is to choose a pixel or a group of pixels in each iteration step. The analyses are based on the geometry. Pixel patterns are studied individually for the derivation of single-step, double-step, and quadruple-step algorithms. From the view point of numerical computation, selecting pixels to approximate a continuous line amounts to a problem of linear interpolation. Because the pixels have integer coordinates, the interpolation should be treated in a discrete setting, i.e., the interpolation values should be rounded to integers and we call it *integral linear interpolation*. Integral linear interpolation will be used to derive incremental line algorithms. In its simplest case, we first discuss the relationship between

Bresenham's line algorithm and integral linear interpolation. We then illustrate how variations of the original Bresenham's line algorithm such as double-step and quadruple-step incremental line algorithms can be derived by using double-step and quadruple-step integral linear interpolation. This method also applies to the derivation of the incremental run-length slice line algorithms, however, the discussion of its relation to the linear interpolation is less straightforward.

Careful attention is given to the equal-error problem for each of the algorithms because of its importance for code portability.

Algorithms to perform fast integral linear interpolation have been discussed by Field [5], Rokne and Yao [14], and Graham and Iyengar [11], the latter being a generalization of their double- and triple-step line algorithm [10].

The topics in this paper are well known from other investigations, however, the approach is different from other relevant work. This results in a treatment that unifies a considerable body of literature on incremental line drawing.

The paper is organized as follows. Section 2 gives some notations and conventions used throughout this paper. Section 3 introduces the technique of integral linear interpolation. Sections 4 and 5 derive the single-step and double-step incremental line algorithms by means of integral linear interpolation. Section 6 uses the integral linear interpolation to derive run-length slice line algorithms. A conclusion is given in Section 7.

2. Notations and conventions

The discussion of the line scan-conversion problem is constrained to lines with slopes between 0 and 1 whose endpoint coordinates are integers. More specifically, we denote the two endpoints of the line by (x_s, y_s) and (x_e, y_e) and we assume that $x_e \geqslant x_s$ and $y_e \geqslant y_s$. Other lines with integer endpoints can be transformed to meet this condition via sign changes and/or coordinate swaps.

The following notations are used throughout this paper:

1. $\Delta x = x_e - x_s$ and $\Delta y = y_e - y_s$.
2. In the context of linear interpolation k is used to denote the distance between two consecutive interpolation points. For instance, performing linear interpolation over interval $[a, b]$ with $n+1$ equidistant points including a and b, we have $k = (b-a)/n$. In the context of line generation, however, $k = \Delta y/\Delta x$ denotes the slope of the line. In this case k can still be considered as the distance of two consecutive interpolation points in a linear interpolation over interval $[y_s, y_e]$ with $n = \Delta x$.
3. $\lfloor x \rfloor$ denotes the largest integer which is not larger than x; $\lceil x \rceil$ denotes the smallest integer which is not smaller than x. $\lfloor \cdot \rfloor$ and $\lceil \cdot \rceil$ are called the *floor* and *ceiling* functions.
4. A letter with a dot accent denotes the integer approximation to a linear interpolation point. The integer which is nearest to an linear interpolation point x_i is $\dot{x}_i = \lfloor x_i + 0.5 \rfloor$ except when x_i has a fractional part of 0.5. In this case $\dot{x}_i = \lfloor x_i \rfloor$ or $\dot{x}_i = \lfloor x_i + 0.5 \rfloor$ is acceptable with the latter being the default choice. We also use $\dot{x}_i = \lfloor x_i \rfloor$ or $\dot{x}_i = \lceil x_i \rceil$ as the integer approximation of x_i in situations where the use of these definitions helps to derive incremental run-length slice line algorithms.
5. $c = \lfloor k \rfloor$.
6. $C = \lfloor 2k \rfloor$.

3. Integral linear interpolation

3.1. Least error integral linear interpolation

The problem of linear interpolation is to find a set of $n+1$ equidistant points on an interval $[a,b]$, where the lower and upper bounds, a and b, are assumed to be integers for the time being. This is not necessary as the problem itself suggests, however, when this problem is related to line generation, a and b are usually integers or real numbers with some special relation which allows integer arithmetic as we will see in Section 6.

Let us denote the original set of interpolation points by

$$a = a_0, a_1, \ldots, a_n = b,$$

where $a_i = a + i(b-a)/n = a + ik$ for $i = 0, 1, \ldots, n$. Then the integral approximation to the ith point can be obtained by rounding it to the nearest integer, i.e.,

$$\dot{a}_i = \left\lfloor a + \frac{b-a}{n} i + 0.5 \right\rfloor \tag{1}$$

for $i = 0, 1, \ldots, n$. Since $\dot{a}_i$ is obtained by rounding a_i to the nearest integer, we call this type of linear interpolation least error integral linear interpolation.

Since

$$\dot{a}_i = \lfloor a_i + 0.5 \rfloor, \quad i = 0, 1, \ldots, n,$$

we have

$$a_i - 0.5 < \dot{a}_i \leqslant a_i + 0.5,$$

$$a_{i+1} - 0.5 < \dot{a}_{i+1} \leqslant a_{i+1} + 0.5,$$

i.e.,

$$a + ik - 0.5 < \dot{a}_i \leqslant a + ik + 0.5, \tag{2}$$

$$a + (i+1)k - 0.5 < \dot{a}_{i+1} \leqslant a + (i+1)k + 0.5. \tag{3}$$

Subtracting Eq. (2) from Eq. (3), we get

$$k - 1 < \dot{a}_{i+1} - \dot{a}_i < k + 1,$$

or

$$k - 1 < \Delta\dot{a}_i < k + 1.$$

If k is nonintegral, then $\Delta\dot{a}_i$ can either assume the value $\lceil k-1 \rceil = \lfloor k \rfloor = c$ or the value $\lfloor k+1 \rfloor = \lfloor k \rfloor + 1 = c + 1$. If k is integral, $\Delta\dot{a}_i = k = \lfloor k \rfloor = c$. Letting $\varepsilon_i = a_{i+1} - \dot{a}_i - (c + 0.5)$ we obtain

$$\dot{a}_{i+1} = \begin{cases} \dot{a}_i + c, & \varepsilon_i < 0, \\ \dot{a}_i + c + 1, & \varepsilon_i \geqslant 0. \end{cases}$$

It follows from $n > 0$ that $D_i = 2n\varepsilon_i$ retains the sign of ε_i. Since D_i turns out to be a conveniently calculated quantity, it is chosen to be the discriminator and we have

$$\dot{a}_{i+1} = \begin{cases} \dot{a}_i + c, & D_i < 0, \\ \dot{a}_i + c + 1, & D_i \geqslant 0. \end{cases} \tag{4}$$

Noting that

$$\begin{aligned} \varepsilon_i &= a_{i+1} - \dot{a}_i - (c + 0.5) \\ &= a + (i+1) \cdot \frac{b-a}{n} - \dot{a}_i - (c + 0.5), \end{aligned}$$

it follows that

$$D_i = 2na + 2(i+1)(b-a) - 2n\dot{a}_i - n(2c+1). \tag{5}$$

Subtracting D_i from D_{i+1} yields

$$D_{i+1} - D_i = 2(b-a) - 2n(\dot{a}_{i+1} - \dot{a}_i).$$

Hence,

$$D_{i+1} = \begin{cases} D_i + 2(b-a) - 2nc, & D_i < 0, \\ D_i + 2(b-a) - 2n(c+1), & D_i \geqslant 0. \end{cases} \tag{6}$$

The initial value for $\dot{a}_i$ is

$$\dot{a}_0 = a. \tag{7}$$

Evaluating Eq. (5) for $i = 0$ determines the initial value of the discriminator

$$D_0 = 2(b-a) - n(2c+1). \tag{8}$$

An integerized algorithm for least error integral linear interpolation is thus obtained according to the initial values given by Eqs. (7) and (8) and the recurrence formulas of Eqs. (4) and (6).

3.2. Rounding-up and rounding-down integral linear interpolation

If, instead of defining the integral interpolation points using Eq. (1), we round up each interpolation point a_i to the smallest integer which is greater than or equal to a_i, i.e., we define

$$\dot{a}_i = \lceil a_i \rceil = \lceil a + ki \rceil \tag{9}$$

then we obtain the rounding-up integral linear interpolation.

Analogous to the derivation of the recurrence formulas for the least error integral linear interpolation, we can generate similar recurrence formulas for the rounding-up integral linear interpolation with only integer arithmetic. It follows from Eq. (9) that

$$a_i \leqslant \dot{a}_i < a_i + 1, \tag{10}$$

$$a_{i+1} \leqslant \dot{a}_{i+1} < a_{i+1} + 1. \tag{11}$$

Proceeding as before we obtain the recurrences

$$\dot{a}_{i+1} = \begin{cases} \dot{a}_i + c, & D_i \leqslant 0, \\ \dot{a}_i + c + 1, & D_i > 0, \end{cases} \tag{12}$$

$$D_{i+1} = \begin{cases} D_i + b - a - nc, & D_i \leqslant 0, \\ D_i + b - a - n(c+1), & D_i > 0. \end{cases} \tag{13}$$

The initial value of the interpolation point is

$$\dot{a}_0 = a \tag{14}$$

and of the discriminator

$$D_0 = b - a - nc. \tag{15}$$

Similarly, letting $\dot{a}_i = \lfloor a_i \rfloor = \lfloor a + ki \rfloor$, we define rounding-down integral linear interpolation. The recurrence formulas for rounding-down integral linear interpolation are almost the same as the formulas for the rounding-up integral linear interpolation and we get

$$\dot{a}_{i+1} = \begin{cases} \dot{a}_i + c, & D_i < 0, \\ \dot{a}_i + c + 1, & D_i \geqslant 0, \end{cases} \tag{16}$$

$$D_{i+1} = \begin{cases} D_i + b - a - nc, & D_i < 0, \\ D_i + b - a - n(c+1), & D_i \geqslant 0. \end{cases} \tag{17}$$

with

$$\dot{a}_0 = a, \tag{18}$$

$$D_0 = b - a - n(c+1). \tag{19}$$

The difference between (12), (13) and (16), (17) is just in the equality case for the update formulas and in the initial value for D_0.

4. Bresenham line algorithm and integral linear interpolation

Consider a line from (x_s, y_s) to (x_e, y_e) which lies in the raster plane with an overlaid rectangular coordinate grid mesh. The line is assumed to meet the conditions imposed in Section 2. Starting from $x = x_s$, $y = y_s$, Bresenham's algorithm generates the rastered image of a line by increasing the integer abscissa value by one in each iteration, then deciding which of the two neighboring pixels, $(x + 1, y)$ and $(x + 1, y + 1)$ is closer to the true line and then moving to that position. The algorithm chooses one of the two pixels by testing the sign of a discriminator. The decision made effectively rounds the real ordinate value of the point on the real line with integer abscissa $x + 1$ to the nearest integer. If an equal error instance occurs, Bresenham's algorithm will round the real ordinate value up, i.e., choose the integer ordinate to be $y + 1$. The discriminator is updated according to a simple recurrence formula. The following code constitutes the core of Bresenham's line algorithm for generating lines with slopes between 0 and 1.

$D = 2\Delta y - \Delta x;$

```
for (i = 0; i ≤ Δx; ++ i) {
    plot(x, y);
    if (D ≥ 0) {
        y = y + 1;
        D = D − 2Δx;
    }
    x = x + 1;
    D = D + 2Δy;
}
```

Now let us view the line generation problem from a different perspective. Denote the ordered sequence of abscissa of pixels on the line from x_s to x_e; by $\mathscr{X}$ and the ordered sequence of ordinate of pixels from y_s to y_e by $\mathscr{Y}$. There is a one to one correspondence between the elements of these two sequences. Since $\Delta x \geqslant \Delta y$, x is increased by one each step a pixel is determined. Hence

$$\mathscr{X} = \{x_s, x_s + 1, \ldots, x_e\}.$$

The elements in $\mathscr{Y}$ are the results of the least error integral linear interpolation performed on the interval $[y_s, y_e]$:

$$\mathscr{Y} = \{y_s = \dot{y}_0, \dot{y}_1, \ldots, \dot{y}_n = y_e\},$$

where $n = \Delta x$. In the case of $\Delta y < n$ there exist $\dot{y}_i$ and $\dot{y}_{i+1}$ for $i = 0, 1, \ldots, n-1$ such that $\dot{y}_i = \dot{y}_{i+1}$. Replacing a, b by y_s, y_e and n by Δx in the least error integral linear interpolation formulas obtained in Section 2.1, we get the following recurrence relations:

$$\dot{y}_{i+1} = \begin{cases} \dot{y}_i + c, & D_i < 0, \\ \dot{y}_i + c + 1, & D_i \geqslant 0. \end{cases} \tag{20}$$

$$D_{i+1} = \begin{cases} D_i + 2\Delta y - 2c\Delta x, & D_i < 0, \\ D_i + 2\Delta y - 2(c+1)\Delta x, & D_i \geqslant 0. \end{cases} \tag{21}$$

The initial value of the discriminator becomes

$$D_0 = 2\Delta y - (2c + 1)\Delta x. \tag{22}$$

The relation between Eqs. (20)–(22) and Bresenham line algorithm is revealed by examining the value of c which is trivially determined:

$$c = \begin{cases} 0, & 0 \leqslant k < 1, \quad \Delta y < \Delta x, \\ 1, & k = 1, \qquad \Delta x = \Delta y, \end{cases} \tag{23}$$

where k is the slope of the line. If $c = 0$ Eqs. (20) and (21) can be simplified to

$$\dot{y}_{i+1} = \begin{cases} \dot{y}_i, & D_i < 0, \\ \dot{y}_i + 1, & D_i \geqslant 0. \end{cases} \tag{24}$$

$$D_{i+1} = \begin{cases} D_i + 2\Delta y, & D_i < 0, \\ D_i + 2\Delta y - 2\Delta x, & D_i \geqslant 0, \end{cases} \tag{25}$$

and the initial value of the discriminator becomes

$$D_0 = 2\Delta y - \Delta x. \tag{26}$$

These recurrence relations are identical to those employed in Bresenham's algorithm. If $c = 1$, i.e., $\Delta x = \Delta y$, the line forms a 45° angle with respect to the positive x-direction. The recurrence formulas become

$$\dot{y}_{i+1} = \begin{cases} \dot{y}_i + 1, & D_i < 0, \\ \dot{y}_i + 2, & D_i \geqslant 0, \end{cases} \tag{27}$$

$$D_{i+1} = \begin{cases} D_i, & D_i < 0, \\ D_i - 2\Delta x, & D_i \geqslant 0, \end{cases} \tag{28}$$

and the initial value of the discriminator becomes

$$D_0 = -\Delta x. \tag{29}$$

These formulas differ from those used in Bresenham's algorithm. But since the initial value of discriminator is negative, and according to Eq. (28) it will never change in the process of iteration, therefore, according to Eq. (27), the value of $\dot{y}_i$ will increase by one in each iteration to form a 45° move. We therefore conclude that least error integral linear interpolation over the interval $[y_s, y_e]$ results in a line algorithm which is equivalent to Bresenham line algorithm.

5. Double-step line algorithm and least error integral linear interpolation

5.1. Double-step line algorithm

The double-step line algorithm [16] which improves Bresenham's algorithm is based on the observation that if a point (x_i, y_i) at the lower left corner of a 2×2 mesh representing an already plotted pixel in the line with slope between 0 and 1 is given, then only the four pixel patterns shown in Fig. 1 can be formed in a double-step increment in the x-direction under the conditions on the line given in Section 2. Starting from (x_s, y_s), 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

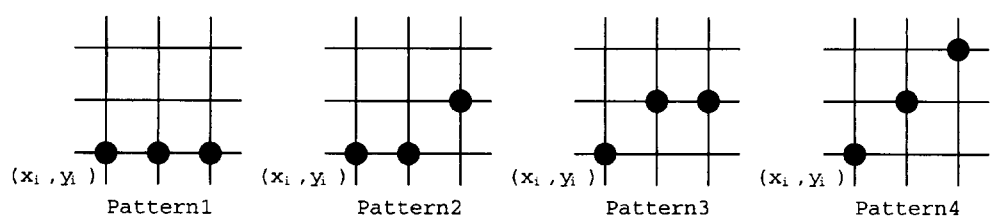

Fig. 1. The four double-step patterns.

conjectured by Freeman [6, 7] and proved by Reggiori [12] (see also [13, 16]) that only two pattern types may occur simultaneously: either 1 and 2 (3) or 2 (3) and 4. This can be proven in a more straightforward manner using the run-length properties of a rastered line which will be discussed in Section 6. From these results the double-step strategy is given by

1. If $0 \leqslant k < 0.5$, then

$$D_0 = 4\Delta y - \Delta x, \tag{30}$$

$$D_{i+1} = \begin{cases} D_i + 4\Delta y, & D_i < 0 \text{ (pattern 1)}, \\ D_i + 4\Delta y - 2\Delta x, & D_i \geqslant 0 \text{ (pattern 2 or 3)}. \end{cases} \tag{31}$$

2. If $0.5 \leqslant k \leqslant 1$, then

$$D_0 = 4\ \Delta y - 3\Delta x, \tag{32}$$

$$D_{i+1} = \begin{cases} D_i + 4\Delta y - 2\Delta x, & D_i < 0 \text{ (pattern 2 or 3)}, \\ D_i + 4(\Delta y - \Delta x), & D_i \geqslant 0 \text{ (pattern 4)}. \end{cases} \tag{33}$$

To distinguish between pattern 2 and 3 requires the test

$$D_i < \begin{cases} 2\Delta y, & \text{if } 0 \leqslant k < 0.5, \\ 2(\Delta y - \Delta x) & \text{if } 0.5 \leqslant k \leqslant 1 \end{cases} \tag{34}$$

resulting in pattern 2 if the test is passed, pattern 3 if not.

5.2. *Designing double-step line algorithm using double-step integral linear interpolation*

In this subsection we show that the integral linear interpolation methodology lends itself to the design of double-step line algorithm by extending it to a double-step incremental version. Again the integral linear interpolation used is of the least error type.

Double-step integral linear interpolation was discussed in [14] and we therefore only outline the algorithm for double-step linear interpolation here. We then show its relation to the double-step line algorithm.

We first introduce following notation:

$$\left.\begin{aligned} A_i &= a_{2i} = a + 2ik, \\ \dot{A}_i &= \lfloor A_i + 0.5 \rfloor = \dot{a}_{2i} \end{aligned}\right\}, \quad i = 0, 1, \ldots, \left\lfloor \frac{n}{2} \right\rfloor.$$

We can easily prove that

$$2k - 1 < \Delta\dot{A}_i = \dot{A}_{i+1} - \dot{A}_i < 2k + 1.$$

The values that can be assumed by $\Delta\dot{A}_i$ are therefore $\lfloor 2k \rfloor = C$ or $\lfloor 2k + 1 \rfloor = C + 1$. Following the same development as in Section 3 we obtain that the formulas to calculate $\dot{A}_i$ and $\dot{D}_i$ are

$$\dot{A}_{i+1} = \begin{cases} \dot{A}_i + C, & D_i < 0, \\ \dot{A}_i + C + 1, & D_i \geqslant 0, \end{cases} \tag{35}$$

and

$$D_{i+1}=\begin{cases} D_i+4(b-a)-2nC, & D_i<0,\\ D_i+4(b-a)-2n(C+1), & D_i\geqslant 0 \end{cases} \tag{36}$$

with the initial values being

$$\dot{A}_0=a,\quad D_0=4(b-a)-n(2C+1). \tag{37}$$

The midpoint between $\dot{A}_i$ and $\dot{A}_{i+1}$, i.e., $\dot{a}_{2i+1}$, can be determined by an extra test:

$$\dot{a}_{2i+1}=\begin{cases} \dot{A}_i+c, & D_i<2(b-a)-2n(C-c),\\ \dot{A}_i+c+1, & D_i\geqslant 2(b-a)-2n(C-c). \end{cases} \tag{38}$$

The test can be saved if $\Delta\dot{A}_i$ is even, i.e., $\Delta\dot{A}_i=2m$ where m is a nonnegative integer. In this case, we simply have

$$\dot{a}_{2i+1}=\dot{A}_i+m.$$

The above recurrence formulas result directly in a double-step line algorithm since what we really need to calculate is the set of y-coordinates of the pixels on the rastered line, and it is readily understood that this can be achieved by performing double-step linear interpolation over interval $[y_s, y_e]$. Replacing a, b by y_s, y_e, $\dot{A}, \dot{a}$ by $\dot{Y}, \dot{y}$, and n by Δx in Eqs. (35) and (36) yields the following recurrence formulas for calculating y coordinates of the pixels:

$$\dot{Y}_{i+1}=\begin{cases} \dot{Y}_i, +C, & D_i<0,\\ \dot{Y}_i+C+1, & D_i\geqslant 0, \end{cases} \tag{39}$$

$$D_{i+1}=\begin{cases} D_i+4\Delta y-2\Delta xC, & D_i<0,\\ D_i+4\Delta y-2\Delta x(C+1), & D_i\geqslant 0, \end{cases} \tag{40}$$

where $\dot{Y}_i=\dot{y}_{2i}$ and C is interpretered as $\lfloor 2\Delta y/\Delta x\rfloor=\lfloor 2k\rfloor$. If $\Delta\dot{Y}_i$ is odd, i.e., C is odd if $D_i<0$ or $C+1$ is odd if $D_i\geqslant 0$, according to Eq. (39) we have

$$\dot{y}_{2i+1}=\begin{cases} \dot{Y}_i+c, & D_i<2\Delta y-2\Delta x(C-c),\\ \dot{Y}_i+c+1, & D_i\geqslant 2\Delta y-2\Delta x(C-c). \end{cases} \tag{41}$$

In the case of $\Delta\dot{Y}_i$ being even, the midpoint $\dot{y}_{2i+1}$ is readily obtained. The initial value of the discriminator becomes

$$D_0=4\Delta x-(2C+1)\Delta y. \tag{42}$$

Noting that the value of c is determined according to Eq. (23) then the value of C is, from the following equation,

$$C=\begin{cases} 0, & 0\leqslant k<0.5,\\ 1, & 0.5\leqslant k<1,\\ 2, & k=1, \end{cases} \tag{43}$$

and we can now easily see that the above recurrence formulas for double-step integral linear interpolation over interval $[y_s, y_e]$ are identical to the recurrence formulas used for double-step line

algorithm in [16] for the case of $0 \leqslant k < 1$, and they are correct in the case of $k = 1$ though the resulting formulas are different from those of double-step line algorithm. We omit the derivation here since it is quite easy.

The advantage of using linear interpolation technique in designing double-step line algorithm is obvious. The discussion on double-step pixel patterns and the differentiation of the two cases depending on the slope of a line being greater or less than 0.5 is no longer needed. Only one set of recurrence formulas is derived which corresponds to different sets of formulas derived in [16] depending on the value of C. In an analogous manner we can design a quadruple-step least error integral linear interpolation algorithm, and it is readily understood that quadruple-step line generation [1] is just a special case of quadruple-step least error integral linear interpolation.

6. Run-length line algorithm based on integral linear interpolation

6.1. Horizontal run-length line algorithm

In Sections 2 and 3, we reduced the problem of line generation to the problem of least error integral linear interpolation over the interval $[y_s, y_e]$ with $n = \Delta x$. An alternative method one would naturally consider is to reduce the problem of line generation to the problem of performing linear interpolation over the interval $[x_s, x_e]$ with $n = \Delta y$. Since $\Delta y \leqslant \Delta x$, this will in general reduce the number of iteration steps. Unfortunately, this is not helpful as is illustrated by Fig. 2.

The fact that there exist identical elements in the sequence $\mathscr{Y}$ when $0 \leqslant k < 1$ is visualized by the horizontal runs of the pixels in the rasterized line. This provides us with a clue for finding the start of each horizontal runs by means of integral linear interpolation since the start of each run has an integral x-coordinate. We impose a set of horizontal mid-lines $y = y_s + i - 0.5$, $i = 1, \ldots, \Delta y$ (see Fig. 3). The x-coordinate of the intersection of the line from (x_s, y_s) to (x_e, y_e) with line $y = y_s + i - 0.5$ is denoted by x_i. It is then readily understood that rasterized image of the half open line segment whose projection on the x axis is the half open interval $[x_i, x_{i+1})$ is the horizontal run from $\lceil x_i \rceil$ to $\lceil x_{i+1} \rceil - 1$. Thus, except for the first horizontal run which starts from x_s, each horizontal run starts from $\lceil x_i \rceil$. To compute $\lceil x_i \rceil$ we convert this problem to a problem of rounding-up integral linear interpolation as introduced in Section 2.2.

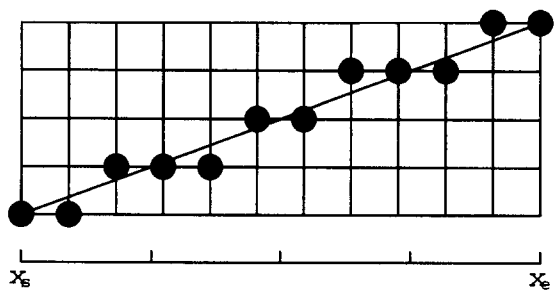

Fig. 2. Linear interpolation over the interval $[x_s, x_e]$.

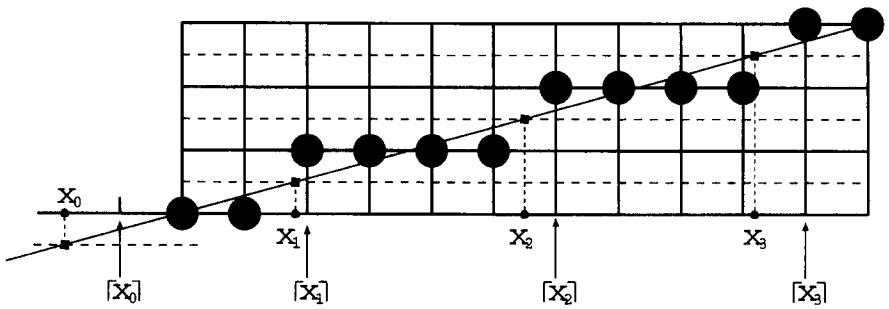

Fig. 3. Horizontal run of the pixels in a rasterized line and finding the starting position of each run by rounding-up integral linear interpolation.

Introducing auxiliary point $x_0 = x_s - \Delta x/2\Delta y$, we get $n+1$ $(n = \Delta y)$ equally spaced points since we use n mid-lines to intersect the line from (x_s, y_s) to (x_e, y_e) (see Fig. 3). The distance between x_i and x_{i+1} is $\Delta x/\Delta y$, and the last point is $x_n = x_e - \Delta x/2\Delta y$. Remember that what we really need are the integer points $\dot{x}_1, \ldots, \dot{x}_n$ where $\dot{x}_i = \lceil x_i \rceil$ is the beginning of $(i+1)$th horizontal run. The beginning of the first horizontal run is x_s. We now perform rounding-up integral linear interpolation over interval $[x_0, x_n]$ to obtain these integer points. Referring to the derivation of the recurrence formulas for rounding-up integral linear interpolation in Section 2.2, we substitute x_0, x_n for a and b, respectively. Here n is changed to Δy; a_i is changed to x_i; and $c = \lfloor k \rfloor = \lfloor (x_n - x_0)/n \rfloor$. The lower and upper bounds of the interval are no longer integers, which seems to be a barrier in performing integral linear interpolation using only integral arithmetic. This difficulty can be overcome by examining the equations $x_n - x_0 = x_e - x_s = \Delta x$. The discriminator D_i used is different from that defined in Section 2.2. Since

$$\begin{aligned} x_{i+1} - \dot{x}_i - c &= x_0 + (i+1)\frac{x_n - x_0}{\Delta y} - \dot{x}_i - c \\ &= x_s - \frac{\Delta x}{2\Delta y} + (i+1)\frac{\Delta x}{\Delta y} - \dot{x}_i - c, \end{aligned}$$

we define

$$\begin{aligned} D_i &= 2\Delta y(x_{i+1} - \dot{x}_i - c) \\ &= 2\Delta y x_s - \Delta x + 2(i+1)\Delta x - 2\Delta y \dot{x}_i - 2\Delta y c. \end{aligned}$$

Hence,

$$\begin{aligned} D_{i+1} - D_i &= 2\Delta x - 2\Delta y(\dot{x}_{i+1} - \dot{x}_i) \\ &= \begin{cases} 2\Delta x - 2\Delta y c, & D_i \leqslant 0, \\ 2\Delta x - 2\Delta y(c+1), & D_i > 0. \end{cases} \end{aligned}$$

Let

$$r = \Delta x \bmod \Delta y \quad \text{and} \quad \tilde{r} = \Delta x \bmod 2\Delta y,$$

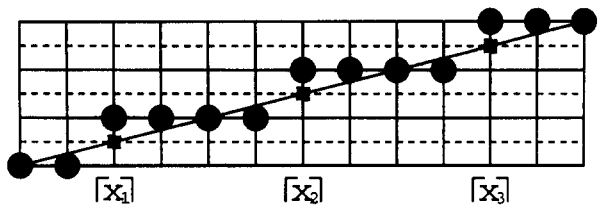

Fig. 4. A line with equal error cases.

then

$$\Delta yc = \Delta x - r, \quad 2\Delta y\lfloor \Delta x/2\Delta y \rfloor = \Delta x - \tilde{r}$$

and we have the recurrence formulas

$$\dot{x}_{i+1} = \begin{cases} \dot{x}_i + c, & D_i \leqslant 0, \\ \dot{x}_i + c + 1, & D_i > 0, \end{cases} \tag{44}$$

$$D_{i+1} = \begin{cases} D_i + 2r, & D_i \leqslant 0, \\ D_i + 2r - 2\Delta y, & D_i > 0. \end{cases} \tag{45}$$

The initial values of $\dot{x}_i$ and D_i are

$$\dot{x}_0 = \left\lceil x_s - \frac{\Delta x}{2\Delta y} \right\rceil = x_s - \left\lfloor \frac{\Delta x}{2\Delta y} \right\rfloor = x_s - (c \gg 1), \tag{46}$$

$$\begin{aligned} D_0 &= 2\Delta y(x_s - \dot{x}_0 - c) + \Delta x = 2\Delta y \left\lfloor \frac{\Delta x}{2\Delta y} \right\rfloor - 2\Delta yc + \Delta x \\ &= \Delta x - \tilde{r} - 2(\Delta x - r) + \Delta x = 2r - \tilde{r}, \end{aligned} \tag{47}$$

where $\gg$ denotes the right binary shift operation.

Equal error instances are treated in a manner consistent with the Bresenham line algorithm as is illustrated in Fig. 4. This is because we use rounding-up to obtain the initial point of the horizontal run. Some properties of a rastered line can be derived directly from Eq. (45):

1. Except for the first and the last run, the lengths of horizontal runs are confined to two consecutive integers c and $c + 1$.
2. The sum of the lengths of the first and the last horizontal run is either $c + 1$ or $c + 2$.
3. Pixel patterns 1 and 4 in Fig. 1 cannot occur in one line since the occurrence of pattern 1 implies a horizontal run of length at least 3 and the occurrence of pattern 4 implies a horizontal run of length 1 which is neither the first nor the last horizontal run of the line. This contradictions property 1.

Here we see that properties of the pixel patterns of rastered lines are derived by integral linear interpolation without referring to geometry.

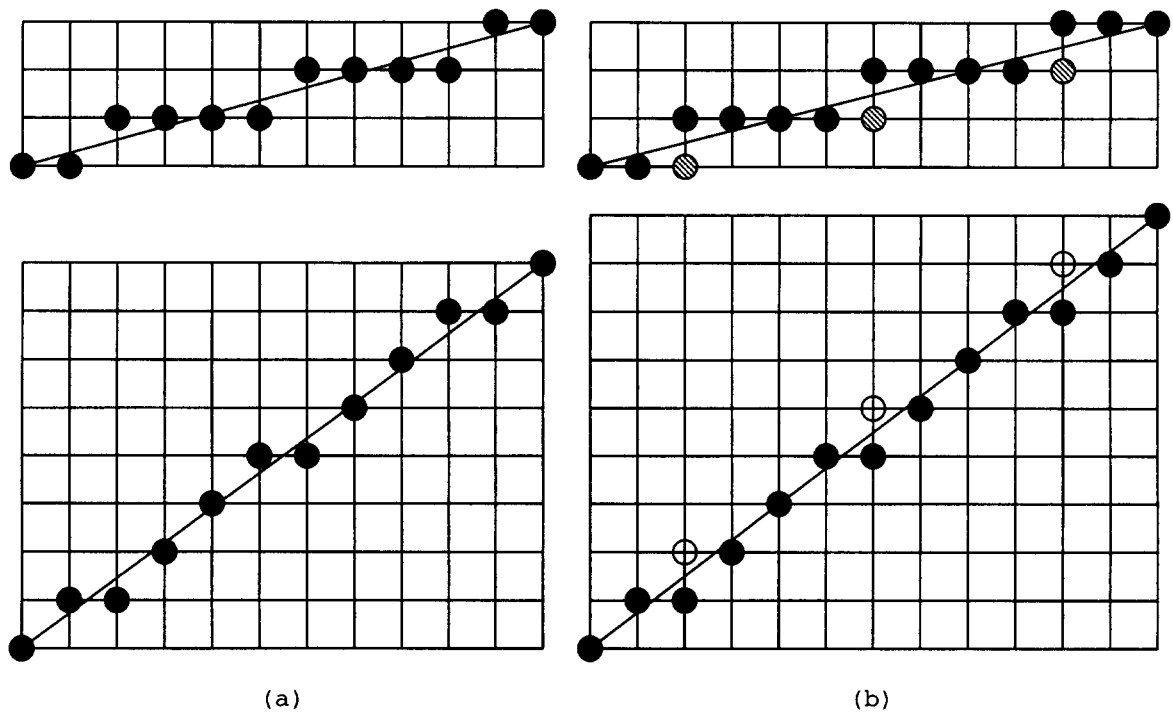

Fig. 5. Lines (bottom) and their complementary lines (top). Diagonal runs in lines are obtained by calculating horizontal runs of the complementary lines. Empty circles in (b) denote the pixels which should be generated by Bresenham line algorithm. This suggest that we use rounding-down integral linear interpolation to calculate the end positions of horizontal runs of the complementary line. The hatched circles denote this adjustment.

6.2. Diagonal run-length algorithm

A rasterized line can be divided into slices of horizontal runs so that multiple pixels can be rendered in each iteration, as shown above. However, when the slope of the line is greater than 0.5, the length of each horizontal run reduces to 1 or 2 since $c = \lfloor \Delta x/\Delta y \rfloor = 1$. In this case, the advantage of the horizontal run-length algorithm is greatly reduced. In the extreme case of slope being 1, the length of each run reduces to 1 and we actually obtain a single-step algorithm. We can, however, divide the line into diagonal runs since the starting of a new horizontal run is equivalent to a diagonal move of the pixel and each horizontal move of a pixel can be viewed as the starting of a new diagonal run.

The diagonal runs of a line can be obtained by calculating the horizontal runs of its complementary line [4]. For a line with $\Delta y > 0.5\Delta x$, i.e., slope of line > 0.5, we compute the horizontal runs of its complementary line starting from (x_s, y_s) and ending at (x_e, y'_e) where $y'_e = \Delta x - y_e$, and reverse the role of horizontal steps to diagonal steps to obtain the complementary incremental step sequence of the line to be drawn (see Fig. 5(a)). Interchanging the role of horizontal steps and diagonal steps also interchanges the step choice for the equal error instance. So if we use the horizontal run-length algorithm of Section 5.1 to compute the diagonal runs of a line with slope > 0.5, the equal error

default is a horizontal move (see Fig. 5(b)), resulting in a discrepancy between lines generated by Bresenham line algorithm and lines generated in this manner. This discrepancy can be eliminated by obtaining the horizontal runs through calculating the end positions of the horizontal runs of its complementary line using rounding-down integral linear interpolation and then changing horizontal runs to diagonal runs. Thus $\dot{x}_i = \lfloor x_i \rfloor$ is the end of the ith run, and we simply obtain a diagonal run which ends at abscissa $\dot{x}_i$. In the case of drawing a line with slope less than or equal to 0.5, we still use rounding-up integral linear interpolation to calculate the horizontal runs of the line since a rastered line generated in this manner is identical to the rastered line generated by Bresenham's algorithm no matter if there exist equal error instances.

If we define

$$\tilde{c} = \left\lceil \frac{\Delta x}{2\Delta y} \right\rceil$$

then the derivation of recurrence formulas for the rounding-down integral linear interpolation is similar to the derivation of recurrence formulas for rounding-up integral linear interpolation and we get

$$\dot{x}_{i+1} = \begin{cases} \dot{x}_i + c, & D_i < 0, \\ \dot{x}_i + c + 1, & D_i \geqslant 0, \end{cases} \tag{48}$$

$$D_{i+1} = \begin{cases} D_i + 2r, & D_i < 0, \\ D_i + 2r - 2\Delta y, & D_i \geqslant 0, \end{cases} \tag{49}$$

with

$$\dot{x}_0 = \left\lfloor x_s - \frac{\Delta x}{2\Delta y} \right\rfloor = x_s - \tilde{c}, \tag{50}$$

$$D_0 = \Delta x + 2\Delta y(\tilde{c} - c - 1). \tag{51}$$

6.3. Double-step run-length algorithm

Combining the double-step technique and the run-length technique we can easily derive the double-step run-length line algorithm by using integral linear interpolation. Here we give a brief derivation since the method is almost the same as the method we have used to derive the double-step incremental line algorithm except that we use rounding-up linear interpolation here rather than least error integral interpolation.

Notations used in Section 6.1 will still be used in this section. Define

$$X_i = x_{2i} = x_s - \frac{\Delta x}{2\Delta y} + 2i\frac{\Delta x}{\Delta y}, \quad i = 0, 1, \ldots, \left\lfloor \frac{n}{2} \right\rfloor,$$

$$\dot{X}_i = \lceil X_i \rceil = \dot{x}_{2i}.$$

Using arguments similar to what was used to derive Eqs. (40) and (41) in Section 5.1 we obtain the following recurrence formulas:

$$\dot{X}_{i+1} = \begin{cases} \dot{X}_i + C, & D_i \leqslant 0, \\ \dot{X}_i + C + 1, & D_i > 0, \end{cases} \tag{52}$$

$$D_{i+1} = \begin{cases} D_i + 2R, & D_i \leqslant 0, \\ D_i + 2R - 2\Delta y, & D_i > 0, \end{cases} \tag{53}$$

where

$$C = \lfloor 2k \rfloor = \left\lfloor 2\frac{\Delta x}{\Delta y} \right\rfloor, \quad R = 2\Delta x \bmod \Delta y,$$

$$\begin{aligned} D_i &= 2\Delta y(X_{i+1} - \dot{X}_i - C) \\ &= 2\Delta y x_s - \Delta x + 4(i+1)\Delta x - 2\Delta y \dot{X}_i - 2\Delta y C. \end{aligned}$$

The initial values of X_i and D_i are:

$$\dot{X}_0 = \dot{x}_0 = x_s - c \gg 1, \tag{54}$$

$$D_0 = 2R - \tilde{r}, \tag{55}$$

where $\tilde{r} = \Delta x \bmod 2\Delta y$.

To calculate the midpoint $\dot{x}_{2i+1}$ between $\dot{X}_i = \dot{x}_{2i}$ and $\dot{X}_{i+1} = \dot{x}_{2i+2}$, we note the fact that $\dot{x}_{2i+1} = \lceil x_{2i+1} \rceil$ and the difference $\dot{x}_{2i+1} - \dot{X}_i$ can be either c or $c+1$. We thus have

$$\dot{x}_{2i+1} = \begin{cases} \dot{X}_i + c, & x_{2i+1} - \dot{X}_i - c \leqslant 0, \\ \dot{X}_i + c + 1, & x_{2i+1} - \dot{X}_i - c > 0. \end{cases}$$

Let

$$d_i = 2\Delta y(x_{2i+1} - \dot{X}_i - c).$$

Deriving a bit further we get

$$d_i = D_i - 2\Delta x + 2\Delta y(C - c).$$

The formula to calculate $\dot{x}_{2i+1}$ is therefore

$$\dot{x}_{2i+1} = \begin{cases} \dot{X}_i + c, & D_i \leqslant 2\Delta x - 2\Delta y(C - c), \\ \dot{X}_i + c + 1, & D_i > 2\Delta x - 2\Delta y(C - c). \end{cases} \tag{56}$$

In the case of $\dot{X}_{i+1} - \dot{X}_i$ being even, no extra test is need, and we simply have

$$\dot{x}_{2i+1} = \dot{X}_i + m,$$

where m is an integer and $\dot{X}_{i+1} - \dot{X}_i = 2m$.

We thus have all the formulas to implement a double-step horizontal run-length line algorithm. A double-step run-length line algorithm was also discussed by Fung et al. [8] using a different method.

7. Conclusion

We have presented an approach to the design of incremental line algorithms in this paper which reduces the problem of designing incremental line algorithms to the problem of integral linear interpolation over the y extent or the x extent of a line. This new treatment unifies a considerable body of literature on incremental line drawing and it has the advantage of simplifying the derivation. The variations of the original Bresenham line algorithm become natural extensions of the algorithm under this framework, and the properties of rastered lines upon which the double-step and the run-length slice line algorithms are based are the direct results of the integral linear interpolation. Further enhancement of existing algorithms, e.g., to incorporate the double-step technique into a run-length slice algorithm, becomes straightforward using this new technique, as is shown in the last subsection.

References

[1] P. Bao, J. Rokne, Quadruple-step line generation, Comput. Graph. 13 (4) (1989) 461–469.
[2] J.E. Bresenham, Algorithm for computer control of digiter plotter, IBM Syst. J. 4 (1965) 25–30.
[3] J.E. Bresenham, Incremental line compaction, Comput. J. 25 (1982) 116–120.
[4] J.E. Bresenham, Run length slice algorithms for incremental lines, in: R.A. Earnshaw (Ed.), Fundamental Algorithms for Computer Graphics, NATO Computer and Systems Series, vol. 17, Springer, New York, 1985, pp. 59–104.
[5] D. Field, Incremental linear interpolation, ACM Trans. Graph. (1985) 1–11.
[6] H. Freeman, On the encoding of arbitrary geometric configurations, IRE Trans. EC-102 (1961) 260–268.
[7] H. Freeman, Boundary encoding and processing, in: B.S. Lipkin, A. Rosenfeld (Eds.), Picture Processing and Psychopictories, Academic Press, New York, 1970, pp. 241–266.
[8] K.Y. Fung, T.M. Nicholl, A.K. Dewdney, A run-length slice line drawing algorithm without division operations, Comput. Graph. Forum 3 (1992) 267–277.
[9] G.W. Gill, N-step and incremental straight-line algorithms, IEEE Comput. Graph. Appl. May (1994) 66–72.
[10] P. Graham, S. Iyengar, Double- and triple-step incremental generation of lines, in: Proc. 1993 ACM Computer Science Conf., New York, 1993.
[11] P. Graham, S. Iyengar, Double- and triple-step incremental linear interpolation, IEEE Comput. Graph. Appl. May (1994) 49–53.
[12] G.B. Regiori, Digital computer transformations for irregular line drawings, Tech. Rep. 403-22, Department of Electrical Engineering and Computer Science, New York Univ., April 1972.
[13] J. Rokne, B. Wyvill, X. Wu, Fast line scan-conversion, ACM Trans. Graph. 9 (4) (1990) 376–388.
[14] J. Rokne, C. Yao, Double-step incremental linear interpolation, ACM Trans. Graph. 11 (2) (1992) 183–192.
[15] A. Rosenfeld, Digital straight line segments, IEEE Trans. Comput. C-23 (1974) 1264–1269.
[16] X. Wu, J. Rokne, Double-step generation of lines and circles, Comput. Vision Graph. Image Process. 37 (1987) 331–344.