Asian Journal of Information Technology 4 (9): 835-839, 2005

An Improved Algorithm for Scan-converting a Line

[1]Md. Hasanul Kabir, [2]Md. Imrul Hassan and [1]Abdullah Azfar
[1]Department of Computer Science and Information Technology,
[2]Department of Electrical and Electronic Engineering,
Islamic University of Technology, Bangladesh

Abstract: For scan-converting a line, Bresenham's Line Algorithm is an efficient incremental method. In this study, presented an improved algorithm over the Bresenham's Algorithm for line scan-conversion. Depending on the slope of the line, we computed up to which point the line is going to have a unique increasing/decreasing pattern in one of the directions while there are unit increments in the other direction. Then we have used this increasing/decreasing pattern to draw the remaining pixels of the line without the need of checking the sign of the decision variable for pixel choice. The difference with the Bresenham's Line Algorithm is, in that algorithm, for pixel choosing, we had to decide till to the other end point of the line, but in present algorithm we only have to decide up to a portion of that line and then we can continue drawing pixels without decision making.

Key words: Bresenham's line algorithm, DLE, DDA, GCD

INTRODUCTION

In computer graphics, scan converting a straight-line segment is the most basic operation[1]. Many curves, wire frame objects and complex scenes are composed of line segments. The speed of graphics rendering depends heavily on the speed of scan converting a line. The ability to scan convert a line quickly and efficiently is an important factor in graphical library. Bresenham's line algorithm is a very important algorithm and has been widely used for scan conversion. Many new methods have been proposed in attempt to speed up the scan conversion of line. Here we restrict our discussion to scan converting a straight line, based on Bresenham's line algorithm.

A line in computer graphics typically refers to a line segment, which is a portion of a straight line that extends indefinitely that extends in opposite directions[2]. Scan converting (i.e. rasterizing) a straight-line segment (or simply line) is the most basic operation. For the past 40 years, there have been many improvements in the algorithms for scan conversion of a line. But out of those, only four algorithms are used most widely. Brief descriptions of these algorithms are as follows.

Direct line equation algorithm: The Direct Line Equation (DLE) algorithm uses the simplest technique of scan converting a line. At first the two end points are scan converted to pixel coordinates. Then it calculates the slope and the y intercept of the line. If the absolute value of the slope is less than or equal to 1, then for every integer value of the x coordinate between and excluding the two x coordinate endpoint values, calculates the corresponding y coordinate value[2]. Conversely if the absolute value of the slope is greater than 1, then for every integer value of the y coordinate between and excluding the two y coordinate endpoint values, calculates the corresponding x coordinate value and scan converts the calculated values[2]. A major drawback of the DLE algorithm is that it involves floating point computation in every step[2].

Digital differential analyzer algorithm: The Digital Differential Analyzer (DDA) algorithm is an incremental scan conversion method. This approach is characterized by performing calculations at each step using results from the preceding step[2]. It calculates points on the line without any floating point multiplication. But a floating point addition is still needed in determining each successive point. The DDA algorithm is faster than the DLE algorithm. But cumulative error due to limited precision in floating point representation may cause
calculated points to drift away from their true position when the line is relatively long[2].

Corresponding Author: Md. Hasanul Kabir, Department of Computer Science and Information Technology (CIT)
Islamic University of Technology (IUT), Bangladesh

Midpoint line algorithm: Midpoint line algorithm[3] uses only integer operations to scan convert a line. The choice of pixels is made by testing the sign of a Discriminator based on the Midpoint principle. The Discriminator obeys a simple recurrence formula that can be calculated using only integer arithmetic shift operation. When it begins to scan-convert the next pixel, it first modifies the Discriminator based on its original value by a few integer arithmetic shift operation. After that it tests the sign of this new Discriminator to decide which pixel should be selected. (The selected pixel is the closest to the actual line). The Discriminator sign testing approach is simple, robust and efficient. It can also be implemented in the hardware easily.

Bresenham's line algorithm: Bresenham's line algorithm is a highly efficient incremental method for scan-converting lines[2]. It produces mathematically accurate results using only integer addition, subtraction and multiplication by 2, which can be accomplished by a simple arithmetic shift operation. The method works as follows. The line we want to scan convert is shown in Fig. 1, where $0<m<1$. We start with pixel $P_1(x_1,y_1)$. Now we choose either the pixel on the right or the pixel right and up.

The coordinates of the last chosen pixel upon entering step i are (x_i,y_i). We choose the next between the bottom pixel S and the top pixel T.

If the chosen pixel is the top pixel T ($d_i \geq 0$) then $x_{i+1} = x_i+1$ and $y_{i+1} = y_i+1$ and so

$$d_{i+1} = d_i+2 (\triangle y - \triangle x)$$

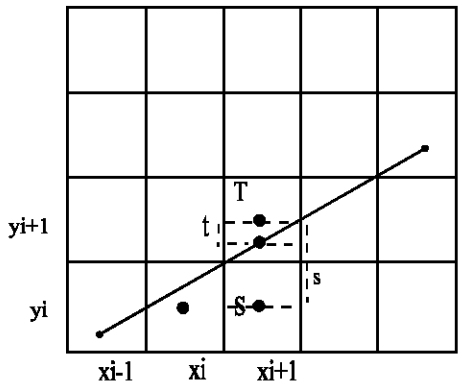

Fig. 1: Pixel selection according to Bresenham's Line Algorithm

If the chosen pixel is pixel S ($d_i<0$) then $x_{i+1}= x_i+1$ and $y_{i+1} = y_i$ and so

$$d_{i+1} = d_i+2\triangle y$$

where $d_i = 2\triangle y * x_i - 2\triangle x * y_i +C$
and $C = 2\triangle y + \triangle x (2b - 1)$

We use here a decision variable d_i. For the value of each d_i we calculate the corresponding value of d_{i+1}.

The code for implementing Bresenham's line algorithm is given below:

```
Void Bresenham( )
{
Line 1:          dx=x2-x1;
Line 2:          dy=y2-y1;
Line 3:          dT=2*(dy-dx);
Line 4:          dS=2*dy;
Line 5:          d=(2*dy)-dx;
Line 6:          putpixel(x1,y1);
Line 7:          while(x1<x2)
Line 8:          {
Line 9:                  x1++;
Line 10:         if(d<0)
Line 11:         {
Line 12:         d=d+dS;
Line 13:         putpixel(x1,y1);
Line 14:         }
Line 15:         else
Line 16:         {
Line 17:         y1++;
Line 18:         d=d+dT;
Line 19:         putpixel(x1,y1);
Line 20:         }
Line 21:    }
Line 22:    putpixel(x2,y2);
}
```

THE IDEA BEHIND

A scan converted line may contain many identical pixel segments in their relative positions[1]. If any two points of a line repeat their relative positions in the squares of the raster grid field, the line can be cut into segments and the corresponding pixel arrangement of the scan converted line segment will repeat also[1]. In other words if (x_0 , y_0) and $(x_0 +r, y_0+r)$ are on the line where r and s are two arbitrary integers then the corresponding scan converted pixel arrangement from $x = |x_0|$ to $x = |x_0|+r$ will be same as the pixel arrangement from $x = |x_0|+r$ to $x = |x_0|+2r$. Therefore multiple segments (or pixels) of a line can be replicated.

Let f(x,y) be a straight line and m is the slope of the line. We have four cases of m:

a) $0 \le m \le 1$
b) $1 \le m \le 8$
c) $-8 \le m \le -1$
d) $-1 \le m \le 0$

Our discussion will be restricted to case (a). The other three cases can be transformed into case (a) by swapping x and y and/or changing the incremental direction.

Let the two endpoints of the line be (x_0, y_0) and (x_n, y_n) respectively. The the slope of the line is

$$m = (y_n - y_0) / (x_n - x_0) = dy/dx$$

Since we only study case (a) mentioned above, we have

$$|y_n - y_0| \le |x_n - x_0|$$

Let's assume $(x_0, y_0) = (0,0)$ and $(x_n, y_n) = (dx,dy)$ which are integer end points. We have the line segment equation $y = mx$, where $m = dy/dx$ and $0 \le x \le x_n$. Let g be the Greatest Common Divisor (GCD) of x_n and y_n. Then m can be represented as $m = y_n / x_n = Pg / Qg = P / Q$. Here P and Q are positive integers $0 \le P \le Q$, $g \le 1$ and GCD(P,Q) = 1.

How segments can be replicated: Let $y = (y_n / x_n)*x = (Pg / Qg)*x = (P / Q)*x$ be a line with integer endpoints (x_0, y_0) and (x_n, y_n), where $0 \le x \le x_n$, $0 \le y \le y_n$, $y_n \le x_n$, $g \le 1$, and GCD(P,Q) = 1[1]. This line can be broken up into g = 1 segment, and each segment has Q pixels. The pixel arrangements of the g segments of the line take the same shape after scan conversion. If dy=0, we define g=dx , Q=1and p=0.

Proof: From the equation of the line, we know that after x increases Q pixels from $x_0 = 0$, then y will increase P pixels. So the point on this line (Q, P) is located on a pixel. (0,0) and (Q,P) are the two corresponding endpoints of the first and second segments. The second segment starts from (Q,P), and the third segment starts from (2Q,2P). Since the slopes of the line segments are same, The pixel arrangements of the g segments are also the same. All the scan converted segments of the line can be considered to be the parallel translations of the first segment. The segments' end points are as follows :

Segment 1 : (0,0) - (Q-1, P-r)
Segment 2 : (Q,P) - (2Q-1, 2P-r)
.....
Segment g:((g-1)Q,(g-1)P)-(gQ-1,gP-r)

Where r = rounded up value of (P / Q). If we extend one extra segment for every pixel, we have:

Segment 1 : (0,0) - (Q, P)
Segment 2 : (Q,P) - (2Q, 2P)
.....
Segment g : ((g-1)Q, (g-1)P) - (gQ, gP)

The above discussion tells us that multiple segments or pixels of a line can be replicated or scan converted in parallel.

PROPOSAL

As from the above discussion, a line segment maintains a pattern according to which it repeats identical shape. A line segment repeats its identical nature after the value of x coordinate has reached up to $x_0 + (dx / GCD(dx,dy))$. Our basic idea is to plot the pixel values up to x_k by Bresenham's line algorithm, where $x_k = x_0 + (dx/GCD(dx/dy))$. While plotting the pixel values by resenham's line algorithm, we keep track of for which values of x the value of y changes. Now, after getting the first segment of the line, we plot the remaining pixels according to the obtained pattern. For example, the next repeatative segment will be from (x_k+1) to x_q where $(x_k+1) - x_q = x_k$. This is continued until we reach x_n.

The code for present proposed algorithm is given below:

```
Void newline( )
{
Line 1:                 dx=x2-x1;
Line 2:                 dy=y2-y1;
Line 3:                 dT=2*(dy-dx);
Line 4:                 dS=2*dy;
Line 5:                 d=(2*dy)-dx;
Line 6:                 putpixel(x1,y1);
Line 7:                 inc = gcd(dx,dy);
Line 8:                 incr = dx/inc;
Line 9:                 incx = x1+incr;
Line 10:        while(x1<incx)
Line 11:        {
Line 12:                x1++;
Line 13:                if(d<0)
Line 14:                {
Line 15:                d=d+dS;
Line 16:                putpixel(x1,y1);
Line 17:                a[i]=0;
Line 18:                }
Line 19:                else
Line 20:                {
```

```
Line 21:                    y1++;
Line 22:                    d=d+dT;
Line 23:                    putpixel(x1,y1);
Line 24:                    a[i]=0;
Line 25:                    }
Line 26:          }
Line 27:          int s=0;
Line 28:          while(x1<x2)
Line 29:          {
Line 30:                    x1++;
Line 31:                    y1=y1+a[s];
Line 32:                    s++;
Line 33:                    if(s==incr)
Line 34:                              s=0;
Line 35:                    putpixel(x1,y1,2);
Line 36:          }
Line 37:          putpixel(x2,y2,5);
}
```

A DETAIL OVERVIEW

Let the two endpoints of the line be (x_0, y_0) and (x_n, y_n) respectively. Then $dx = x_n - x_0$ and $dy = y_n - y_0$. The GCD of dx and dy is computed by a function named *gcd*. After getting the value of the GCD, Bresenham's line algorithm is applied for the points x_0 to x_k, where $x_k = x_0 + (dx / GCD(dx,dy))$. As each value of x corresponds to a value of y, we keep the track for which values of x the value of y increases from its previous y value. We maintain this information in an array. Let a be the array. Then if the corresponding y value of x_i+1 is same as the corresponding y value of x_i then the value of (i+1)th element of array a is a[i+1]=0. If the corresponding y value of x_i+1 is not same as the corresponding y value of x_i then the value of (i+1)th element of array a is a[i+1]=1.

Now up to x_k the line segment is drawn. The remaining portion of the line segment is plotted from the stored information in the array. For each segment x_r to x_t where $x_t - x_r = x_k$ we only compare the relative x value with the corresponding element of the array. If the corresponding array element is 1 then the y value increases from the previous y value and if the corresponding array element is 0 then the y value remains same as the previous value.

PERFORMANCE ANALYSIS AND EVALUATION

After analyzing both the Bresenham's algorithm and present proposed algorithm, we find that both of them have the complexity level in order of O(n). Neither of the two algorithms have any nested looping conditions that increases the complexity in order of $O(n^2)$. Only simple looping conditions keep the complexity level in order of O(n).

The main advantage of present proposed algorithm is there is no need to iterate over Bresenham's line algorithm for all values of x, rather only up to the value where the first repetition starts is evaluated through Bresenham's line algorithm. After that only a simple comparison is made to plot the remaining values. It becomes a huge benefit that we get the pattern of the whole line just after plotting the first segment as we have a clear-cut view about the line.

Now, let us make a line-by-line comparison of present proposed algorithm with Bresenham's line algorithm. Let us consider the length of the line n where the length of the line up to *incr* as in line 8 of present proposed algorithm is r and there are M repetitive elements of length r. Let us consider the cost of each iteration for the while loop in line 7 of Bresenham's line algorithm as C. So the total cost for implementing Bresenham's line algorithm is C.r+C.Mr.

In present proposed algorithm the cost to draw for the line up to length r is C.r+r. This additional r is needed because we make an assignment in line 17 or in line 24. Now comes the part of implementing the m repetitive segments. After analyzing with Bresenham's line algorithm we find that present proposed algorithm has to make an additional increment each time in line 32 unless otherwise Bresenham's algorithm iterates through line 17 for each value of x. Bresenham's algorithm iterates through line 17 only if it matches a condition. But our algorithm iterates through line 32 for each value of x.

Replication in hardware level is much faster than replication in software level. Bresenham's line algorithm is not dependent on any kind of replication. But present proposed algorithm replicates multiple segments of line. This replication can be implemented in hardware level which will make the overall scan conversion method much faster than Bresenham's scan conversion method.

CONCLUSIONS

We have introduced a new method of scan converting straight lines. Instead of scan converting the whole line step by step, we can scan convert multiple segments of line through replicating. We believe present work is a significant contribution to implementing basic graphics primitives. We plan to further investigate this idea and extend the method to curved lines.

REFERENCES

1. Chen, J.X., 1998. Fast Floating Point Line Scan-Conversion and Antialiasing, Research, Department of Computer Science, George Mason University, http://graphics.gmu.edu/ raster/ floatLine.pdf
2. Xiang, Z. and R.A. Plastock, 2000. Schaum's Outline of Theory and Problems of Computer Graphics, Second Edition, McGraw-Hill, Reprint Edition.
3. Bresenham, J.E., 1965. Algorithm for Computer control of digital plotter. IBM Syst. J., 4: 25-30.
4. Foley, Vandam, Feiner and Hughes, 1994. Introduction to computer graphics principles and Practice, Addison Wesley.
5. Edward Angel, 1990. Computer Graphics, Addison Wesley.
6. Hearn, D. and M.P. Baker, 1986. Computer Graphics, Prentice-Hall.