Straight Lines: a step by step method

CHALOPIN François-Pierre
BOURDIN Jean-Jacques
Département Informatique
Université Paris 8
2, rue de la Liberté
93526 Saint-Denis Cedex 2
[jj-chalopin]@ai.univ-paris8.fr

Abstract
Drawing straight lines is a major field in computer graphics. Most methods are improvements of the Discrete Differential Analysis method first presented by Bresenham [1]. Combinatory analysis method, as presented by Castle [2] or Dulucq [3] or Berstel [4] are not commonly used: they imply multiple string copies and are therefore slow. A new approach, combining combinatorial analysis and DDA is presented. The DDA does not apply to each point but to a step computed by combinatorial method. This algorithm is tested and proves to be more than four times faster than Bresenham’s algorithm.

Key Words: Algorithms, Computer Graphics, incremental curve generation, line generators.

1. Introduction
To draw a line is one of the most used function of any graphics displays. It is done by drawing a path between two extremities with maximum linearity. The points of the path have to be the discrete points closest to the ones of the real line. The Bresenham’s algorithm [1] is often used. Two reasons concern this: this algorithm is fast and easy to program. Some other approaches of producing the linear path exist [4, 3, 5, 2] but they imply string copies and, therefore, are not efficiently implemented. When improved algorithms are presented, they are based on the DDA method [6, 7, 8]. Our work has been to combine these two major approaches to produce a new algorithm. The benchmarks prove its improvements significant.

2. Properties of the Straight line
Let p (x_p, y_p) and q (x_q, y_q) be the extremities of the line. We want to produce the linear path [p, q]. This path will be mentioned as the line from p to q. The continuous line will refer to the line segment of the real plane. The line is an exact translation of the line from (0, 0) to (u, v) where:

\[ u = x_q - x_p \]
\[ v = y_q - y_p. \]

The values of u and v give the slope of the line. Hence we will refer to the line of slope v/u rather than to the extreme points of the segment. We will limit our studies on the case where u>v>0. It has been noted [9] that other cases are simple symmetries of this one. Here there is only one y_x for each x of the segment.
Let the trace word of the line v/u be the series of letters c_x where

\[ c_x = y_{x+1} - y_x \]

The trace words have been studied [5, 3, 10, 4, 11, 12]. Some important properties have been demonstrated there and will be used hereafter. First of all, for each line, there are three different way of computing the approximation of the continuous segment:
• Considering the best approximation as in [1]
• Considering the integer part of the real ordinate (the greater minor integer).
• Considering the upper integer (the littler major integer).
Let \( w, w' \) and \( w'' \) be the three respective trace words. These words are combined. For example, the three words of the line 3/11 are

\[
\begin{align*}
w &= 01000100010 \\
w' &= 00010001001 \\
w'' &= 10010001000
\end{align*}
\]

Let \( n=01 \), and \( \bar{n}=10 \) the reverse of \( n \), one can see that:

\[
\begin{align*}
w.n &= n.w' \\
\bar{n}.w &= w''.\bar{n}
\end{align*}
\]

A left factor \( n \) of the trace word \( w \) exists in every case. Let \( l \) be the number of letters of the left factor \( n \), Berstel [4] proves that \( l \) respects

\[
v \cdot 1 + \lfloor u/2 \rfloor \equiv_u 0
\]

Here \( l=2 \). This property permits the use of any of the three trace words. A simple translation will produce the best approximation from each of the two other words.

Therefore, in our works we focused on the word \( w' \). It is the word our algorithm produces in the first place. The word \( w \) can be deduced from it.

3. Lines drawing algorithms

We present now two main algorithms to perform the drawing of a line. The first one [1] is the base of a whole field in computer graphics, the incremental, DDA, algorithms. The other one has not been presented in this form before, but is a rewriting of [3] or [2].

3.1. Bresenham's algorithm

The error done when one choose \( y_x \) as approximation of the value \( \frac{v \cdot x}{u} \) can not be greater than \( \frac{1}{2} \). Moreover, the measure of the gap between the real value and the approximation is given by:

\[
gap (x, y_x) = u \cdot y_x - v \cdot x
\]

Let delta be the sum

\[
delta (x, y_x) = \gap (x, y_x) + \gap (x, y_x+1)
\]

The knowledge of the sign of delta is enough to know which one of \( y_x + 1 \) or \( y_x \) is the best approximation of the real value. In C language algorithm 1 presents Bresenham's.
3.2. Recursive algorithm

Trace words are recurrences of 0 and 1 that form steps. For example if \( u > 2v \) the number of 0 (\( u - v \)) is more important than the number of 1 (\( v \)). Therefore the trace word is a long string of 0 separated by some 1. Let a step be a set of subsequent 0. As the word describes a straight line, the steps are almost of the same length. As each 1 is a separator of two different steps, there are \( v \) different steps. The average length of a step is then \( \frac{u - v}{v} \). That is, if this value is not an integer, some steps are \( \lceil \frac{u}{v} \rceil \) wide, and others are \( \lfloor \frac{u}{v} \rfloor - 1 \) wide. The second kind of these steps are more oblique and the long kind are more horizontal. The distribution of the two kinds of steps is the most equitable. Then there are \( (u - v) - v \frac{u - v}{v} \) long (horizontal) steps. The same method may be used to divide up the two kinds of steps. The method is therefore recursive. Int the algorithm 2 the trace word of slope \( v/u \) is named \( s(u, v) \).

```c
void Bresenham (int xp, yp, xq, yq)
{
    int x, y, delta, incob1, inchor;
    delta = 2* (yq - yp) - (xq - xp);
    incob1 = 2* (yq - yp) - 2* (xq - xp);
    inchor = 2* (yp - yq);
    y = yp;
    for (x=xp; x<=xq; x++)
    {
        plot (x, y);
        if (delta > 0)
        {
            y++; 
            delta = delta + incob1;
        }
        else
        {
            delta = delta + inchor;
        }
    }
}
```

**Algorithm 1: Bresenham’s**

<table>
<thead>
<tr>
<th>Condition</th>
<th>Equation</th>
</tr>
</thead>
<tbody>
<tr>
<td>( v = 0 )</td>
<td>( s(u, v) = 0^u )</td>
</tr>
<tr>
<td>( u = v )</td>
<td>( s(u, v) = 1^u )</td>
</tr>
<tr>
<td>( v = 1 )</td>
<td>( s(u, v) = 0^{u-1} 1 )</td>
</tr>
<tr>
<td>( \lfloor \frac{u}{v} \rfloor \leq 1 )</td>
<td>( s(u, v) = F_m (s(p, q)) ), where ( p = u - v ) (i.e. ( u \mod v )) ( q = v \mod (u-v) ) ( m = v \div (u-v) ) (Rq ( m \geq 1 ))</td>
</tr>
<tr>
<td>( \lfloor \frac{u}{v} \rfloor &gt; 1 )</td>
<td>( s(u, v) = G_m (s(p, q)) ), where ( p = v ) ( q = v - (u \mod v) ) ( m = (u - v) \div v ) (Rq ( m \geq 1 ))</td>
</tr>
</tbody>
</table>

Where

\[
F_m : \begin{cases} 
0 \rightarrow 0^1 m \\
1 \rightarrow 0^{1m+1}
\end{cases}
\]

and

\[
G_m : \begin{cases} 
0 \rightarrow 0^{m+1} 1 \\
1 \rightarrow 0^m 1
\end{cases}
\]

**Algorithm 2: Recursive method**
3.3. More remarks

If Bresenham’s algorithm is the most used of these algorithms, it is first because it computes the best approximation of the line and second because it is the quickest. Even if there are a lot of improvements [6, 7, 8] the gain in speed is not realised in practice because of the inevitable pixel write operations. Otherwise algorithms as algorithm 2 are slowed down by the many string copies. We develop now a new method that combines the improvements of both methods by using their main characteristics.

4. Step DDA method

We are now to combine these two different approaches and implement a new method. The principle of our work is to consider the steps to be distributed as the letters of a straight line. The recursive function is therefore called only once. The algorithm is achieve with the use of the DDA algorithm. The DDA algorithm is rebuild to accept the steps as arguments.

The first part of our algorithm is done by the function line that is mainly a switch. For example, the line 3/11 is obtained by the used of the function line (11, 3). This function computes the calling to the function drawing, here the call is drawing (3, 1, 0001, 001). The word 00010001001 is drawn by the putrow functions.

For example the trace word of line 31/97 is:

```
H H H H H H O H H H H H H O H H H H H H H O
```

This is obtained by the call of the function drawing (31, 4, 0001, 001)

<table>
<thead>
<tr>
<th>delta</th>
<th>-4</th>
<th>23</th>
<th>19</th>
<th>15</th>
<th>11</th>
<th>7</th>
<th>3</th>
<th>-1</th>
<th>26</th>
<th>22</th>
<th>18</th>
<th>14</th>
<th>10</th>
<th>6</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Step</td>
<td>h</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>h</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td></td>
</tr>
<tr>
<td>delta</td>
<td>-2</td>
<td>25</td>
<td>21</td>
<td>17</td>
<td>13</td>
<td>9</td>
<td>5</td>
<td>1</td>
<td>-3</td>
<td>24</td>
<td>20</td>
<td>16</td>
<td>12</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>Step</td>
<td>h</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>h</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td></td>
</tr>
</tbody>
</table>

As soon as the putrow primitive is hardware implemented, for example on SGI Elan, the algorithm proves a good improvement in speed. When Bresenham’s algorithm computes and draw the line pixel per pixel with a putpixel primitive, when Rokne’s algorithm [7] computes the pixels two by two, our algorithm uses the putrow primitive and draw the line step by step.

![Figure 2: first part of the 31/97 line](image-url)
The complexity of the algorithm is not \( o(u) \) but \( o(v) \). The minor width of a step is 2 (when \( u \approx 2v \)) and then the speed up is only of a factor 2. In other cases, the speed up increases.

5. **Benchmarks**

We performed two series of benchmarks. The first one is done with drawing of the lines, without a putrow primitive. The drawing is achieved for 10000 lines of random lines in the range given. Even in that case, the new algorithm is proved to be an improvement of previous algorithms.

<table>
<thead>
<tr>
<th></th>
<th>Bresenham’s</th>
<th>Castle &amp; Pitterway’s</th>
<th>recursive algorithm</th>
<th>New Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>(10-50)</td>
<td>1314</td>
<td>1651</td>
<td>1991</td>
<td>905</td>
</tr>
<tr>
<td>(50-100)</td>
<td>2417</td>
<td>2945</td>
<td>3125</td>
<td>1539</td>
</tr>
<tr>
<td>(100-150)</td>
<td>3457</td>
<td>4469</td>
<td>4633</td>
<td>2212</td>
</tr>
<tr>
<td>(150-200)</td>
<td>4957</td>
<td>5665</td>
<td>6157</td>
<td>2821</td>
</tr>
<tr>
<td>(200-250)</td>
<td>6051</td>
<td>7210</td>
<td>7422</td>
<td>3481</td>
</tr>
<tr>
<td>(250-300)</td>
<td>7159</td>
<td>8776</td>
<td>8965</td>
<td>4299</td>
</tr>
<tr>
<td>(300-350)</td>
<td>8485</td>
<td>9938</td>
<td>10002</td>
<td>5207</td>
</tr>
<tr>
<td>(350-400)</td>
<td>10125</td>
<td>11479</td>
<td>11785</td>
<td>5608</td>
</tr>
<tr>
<td>(400-450)</td>
<td>11077</td>
<td>12855</td>
<td>13294</td>
<td>6415</td>
</tr>
<tr>
<td>(450-500)</td>
<td>12461</td>
<td>14355</td>
<td>14543</td>
<td>7002</td>
</tr>
</tbody>
</table>

The second benchmark was done by testing only the two fastest algorithms above: Bresenham’s and ours. For this benchmark the putrow primitive is considered and the results are even better. For each value \( u \) in the range, every value \( v \) is used. For the longest lines, the drawing was not possible as soon as the length of the line is greater than the screen definition. This does not change the ratio of time.
<table>
<thead>
<tr>
<th>Sizes</th>
<th>Bresenham’s</th>
<th>New Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-200</td>
<td>221</td>
<td>55</td>
</tr>
<tr>
<td>200-400</td>
<td>1552</td>
<td>366</td>
</tr>
<tr>
<td>400-600</td>
<td>4218</td>
<td>972</td>
</tr>
<tr>
<td>600-800</td>
<td>8218</td>
<td>1875</td>
</tr>
<tr>
<td>800-1000</td>
<td>13555</td>
<td>3074</td>
</tr>
<tr>
<td>1000-1200</td>
<td>20223</td>
<td>4574</td>
</tr>
<tr>
<td>1200-1400</td>
<td>28228</td>
<td>6367</td>
</tr>
<tr>
<td>1400-1600</td>
<td>37569</td>
<td>8456</td>
</tr>
<tr>
<td>1600-1800</td>
<td>47244</td>
<td>10844</td>
</tr>
<tr>
<td>1800-2000</td>
<td>60255</td>
<td>13531</td>
</tr>
<tr>
<td>2000-2200</td>
<td>73590</td>
<td>16511</td>
</tr>
<tr>
<td>2200-2400</td>
<td>88269</td>
<td>19786</td>
</tr>
<tr>
<td>2400-2600</td>
<td>104270</td>
<td>23355</td>
</tr>
<tr>
<td>2600-2800</td>
<td>121638</td>
<td>27231</td>
</tr>
<tr>
<td>2800-3000</td>
<td>140319</td>
<td>31398</td>
</tr>
</tbody>
</table>

These results have been obtained on a Silicon Graphics International Indigo Elan. Other tests are in progress on other computer graphics hardware. But the ratio of speed up seems to be a little greater than 4 and remains constant.

6. Conclusion
A new algorithm for the drawing of straight lines have been presented. It is proved to be 4 times faster than previous algorithms. It seems to be very important to implement such an algorithm on hardware for any computer graphics device. Our further works consist in the implementation of this method to perform directly antialiased lines. We shall also try to mix the algorithm of Rokne [6] to this work. It could be another improvement.

References