Digital Differential Analyzer for Line

A fast integer-only algorithm for drawing lines

This report develops a method that uses only integer calculations for drawing lines. Digital differential analysis is used to derive the algorithm. The techniques used here can be extended to circles, ellipses, parabolas, and hyperbolas.

Jon Kirwan
Copyright November 1999

Overview

I've always been interested in how things work, not just how they are applied. When I first heard about some special techniques used to draw rasterized lines and other mathematical shapes, I was curious. But it wasn't until some time later that I learned what one of these methods is called and still later before I was finally able to actually read about some of the details.

If you don't already know what a rasterized line is, it's just what happens when you try to draw a line on graph paper by only using completely filled squares of the grid. Instead of drawing the line with a ruler, you are forced to select and fill in entire squares in a way that gives the better appearance of the line that you intended to illustrate. This is the same problem faced when drawing lines by setting pixels on an IBM PC.

My first contact with digital differential analysis, applied to lines, was frustrating. The information was sparse and didn't make sense to me. The writer seemed to skip around and jump to conclusions I just couldn't reach and, finally, left me without an algorithm I could use to test the ideas. When I was finally able to understand how to do these things on my own, I discovered that the information in that article was also wrong. I guess I shouldn't be surprised.

A goal here is to present the reasoning behind the use of digital diff
algorithm you can use to test the ideas. Another goal is to do this clearly and in a fashion that can appeal to folks with only two years of high school algebra and an interest in computer programming. Finally, I'd like to shine some light in the direction of applying this kind of analysis in drawing circles, ellipses.

Let's start by looking at the mathematical line.

**The Line Equation**

If you've had to do any graphing in algebra, you've probably encountered the following general equation for a line:

1. \( y = m \cdot x + b \)

This equation shows an exact relationship between \( x \) and \( y \) values. The value of \( m \) because larger values generate steeper lines. And the value of \( b \) is called the \( y \)-axis where the line intersects it (when \( x \) is zero.) This style of expressing the line, when \( y \)-values, helps to quickly picture what the line might look like.

But this form of a line is rarely used in computer graphics. When given a starting point and an ending point, instead of a slope and \( y \)-intercept. Let's call the starting point, \( (x_1,y_1) \), ending point, \( (x_2,y_2) \).

We can calculate the slope and \( y \)-intercept from these points and mod the slope of a line is rate of change-in-\( y \) versus change-in-\( x \). Expressed

2. \( m = \frac{\Delta y}{\Delta x} = \frac{y_2 - y_1}{x_2 - x_1} \)

Now that we know how to compute \( m \), we can replace \( m \) in equation [1]

3. \( b = y - m \cdot x = y - \frac{y_2 - y_1}{x_2 - x_1} \cdot x \)

The only problem with equation [3] is that we still have the general variable with the values from the first or second point, our choice. (I'll use the first point):

4. \( b = y_1 - \frac{y_2 - y_1}{x_2 - x_1} \cdot x_1 \)

Now, let's take equations [3] and [4] and apply them to equation [1], to

5. \( y = \frac{y_2 - y_1}{x_2 - x_1} \cdot x + y_1 - \frac{y_2 - y_1}{x_2 - x_1} \cdot x \cdot x_1 = y_1 + (x - x_1) \cdot \frac{y_2 - y_1}{x_2 - x_1} \)

That last part isn't too bad. It simply says that \( y \) can be computed from adding a portion of the span between the \( y \)-values of the two points. Th
x-value of the starting point and the span between the x-values of the two points. Makes sense?

Our version of the general line equation, specified by two points, is:

6. \[ y = y_1 + (x - x_1) \cdot \frac{y_2 - y_1}{x_2 - x_1} \]

That's the basics. Now I plan to tinker with this equation a bit to gain some more familiarity with lines. If this is already well understood to you, you might skip to the end of the next section.

**More Fun with the Line Equation**

There's a minor "difficulty" with equation [6]. What if the two points are arranged vertically, so that their x-values equal? The zero in the divisor will cause some nasty problems. It turns out that there is another way to express a line, which handles this problem. Just multiply both sides by \((x_2 - x_1)\):

7. \[ y \cdot (x_2 - x_1) - \left( y_1 + (x - x_1) \cdot \frac{y_2 - y_1}{x_2 - x_1} \right) \cdot (x_2 - x_1) \]

\[ \quad - \left( y_1 \cdot (x_2 - x_1) + (x - x_1) \cdot (y_2 - y) \right) \]

Looks worse? Well, let me move a piece of it to the other side and combine:

8. \[ y \cdot (x_2 - x_1) - y_1 \cdot (x_2 - x_1) - (x - x_1) \cdot (y_2 - y_1) \]

\[ \quad (y - y_1) \cdot (x_2 - x_1) - (x - x_1) \cdot (y_2 - y_1) \]

Do you see the symmetry in this? Let's stop for a moment and imagine of area. Before we go on, it's time for a bit of geometry.

The above chart shows an example to illustrate the areas suggested by the area covered by the horizontal lines. The right side of the equation shows two areas overlap in the cross-hatched area.

The line shown divides the large rectangle into two triangles with equal areas. The right side of the equation shows how the remaining smaller rectangles are still divided. If we remove both, we get back the two areas described in equation [8]. That's it.
moment and see how that works out.

Equation [8] removes the possibility of dividing by zero. But it does: can now perform all of the calculations with integers. This means we a

Equation [8] is still a bit quirky, in spite of it's balance. If you know w versa. But before we make any more adjustments to equation [8], w calculate \( x \) from \( y \) or to calculate \( y \) from \( x \).

\section*{Independent and Dependent Variables}

When plotting rasterized lines between two points, you are faced independently between the \( x \)-values of your two points and compute th \( y \) around independently between the \( y \)-values of your two points an works, mathematically. As a practical matter though, you need to make

When the equation is set up so that you are free to adjust the value of \( x \) such as in equation [1], then \( x \) is considered the independent variable. In other course, you can switch this around in order to reverse the roles. (Notice neither variable as independent or dependent.)

It turns out that when you are plotting rasterized lines (and most anything) choice. The goal is to select the axis that forms the long side of independent variable. In other words, you'd pick \( x \) as your independe you'd pick \( y \) as your independent variable and when the opposite hold this true?

Well, let's look at this simple equation and imagine plotting it using \( x \) (2,20) to a point at (5,50). Let's recall equation [6] and use it for this pu

\begin{equation}
9. \quad y = \left[ 20 + (x - 2) \cdot \frac{50 - 20}{5 - 2} \right]_{x=2}^{5}
\end{equation}

The points would be (2,20), (3,30), (4,40), and (5,50). But this is onl line would look very sparse. What we'd rather plot is (2,20), (2,21), (: on. This way, the line will look properly solid. In this case, we should independent variable and \( x \) was the dependent one, like this:

\begin{equation}
10. \quad x = \left[ 2 + (y - 20) \cdot \frac{50 - 2}{50 - 20} \right]_{y=20}^{50}
\end{equation}

That will give us enough points to make a reasonable line. Of course, t one axis is more or less than it is on the other axis.
Keep this in mind as we continue. The algorithm to follow will have to

**The Keystone Equation**

We've covered the general equation for lines, given two points, in eq avoid dividing by zero in equation [8], although it may still be a m discussed independent and dependent variables and how to choose wh and [10].

I think you are about ready. It's time to develop something useful.

Let's rearrange equation [8] like this:

11. \[ f(x,y) = (x-x_1)(y_2-y_1)-(y-y_1)(x_2-x_1) = 0 \]

I'm also going to create two new terms that are pretty easy to understand and will help us when it comes time to make an algorithm. These two terms are simply the height and width of the rectangle we are supposed to draw a line. But I'll use a mathematically inclined r:

12. \[ \Delta x = x_2 - x_1 \]
12. \[ \Delta y = y_2 - y_1 \]

Using these two, we can rewrite equation [11] into our golden rule of lines:

13. \[ f(x,y) = (x-x_1)\Delta y - (y-y_1)\Delta x = 0 \]

This equation isn't just another way of looking at the same old line drawing lines.

**Some Definitions**

This is the point where we are going to convert the general equation for drawing lines. I've glossed over some of the details, but now that I'm a few definitions I've neglected.

I'll be using the convention of \( x \) and \( y \) axes, where the \( x \) axis is horizontal, the positive \( x \) axis is vertical, and the positive \( y \) direction is up. This isn't always the case, in practice, since both computer screens and printers often have the positive \( y \) direction going down. But for the mathematical convention and leaving it to you to apply it to particular sit
There are eight possible categories of lines, for drawing purposes. If \( y \) is located somewhere in the surrounding vicinity, then this destination will fall into one of the octants shown.

The table shown below the diagram illustrates the eight different octants. The independent variables are always stepped by +1 or -1, as the line is plotted. The dependent variable will then vary by a fractional value, for each of these independent variable steps.

Take a careful look at this table and make sure that you understand it. If you need to, refer back to the discussion on independent and dependent variables and how to choose between them.
Next, we're going to focus on the details of plotting lines where the destination is in octant 1. Once we've determined how to handle one of these, we can see what's different for the other cases and figure out how to adapt the algorithm to them.

**Octant 1: Asking the Right Question**

Let's return to our infamous equation, the seed corn we'll use for the algorithm:

\[ f(x, y) = (x - x_i) \cdot \Delta y - (y - y_i) \cdot \Delta x = 0 \]

where \( \Delta x = x_2 - x_1 \) and \( \Delta y = y_2 - y_1 \).

Equation [14] is zero for any point that is exactly on the line. If we take any given to see where the point is. If the value is zero, the point is on the line; if the value is positive, the point is below the line; and if the value is negative, the point is above the line. (You should test yourself to see if you can spot the exact reason why this is true.) We’re going to use these facts to help us plot our points.

Plotting the first point of our line is rather easy. We just plot the first point we were given, makes us think.

For endpoints in octant 1, \( x \) is the independent variable with a +1 increment we'll plot. Since \( y \) is the dependent variable, it will have a positive fractional change between adjacent points along the line, so the next point we need to plot will be either at \((x_1 + 1, y_1)\) or at amount is less or more than 0.5. And, no matter which one we choose, it will probably not fall exactly on the line we'd like to be drawing. So we'll need to keep track of our errors as we proceed.

So which of the next two points do we choose? What we'd really like is:

**Which next point does the line come closer to, \((x+1, y)\) or \((x+1, y+1)\)?**

If we knew the answer to that, we'd know which one to plot. Before we could answer that, we'd need to measure how closely the line went by each of the two possible points and then we'd need to compare them to see which was the smaller distance.

But another way of asking the same question more directly would be so:

**Does the line cross above or below the halfway point at \((x+1, y+1/2)\)?**

Notice that the point used in the above question is halfway between the two possible next points. If the line crosses over this point, then it must be closer to \((x+1, y+1)\) and that's the point we should use. If the line crosses below this point, then it must be closer to \((x+1, y)\) and that's the point to use, instead. If you are having any difficulty seeing why this is a good choice, take out a piece of graph paper and check it out. I think you'll see how it works.

**Octant 1: Digital Differential Analysis (DDA) Line Algorithm**

http://users.easystreet.com/jkirwan/dda.html

2007-5-7
Let's define a new equation. We'll use it to help us decide between the two possible choices, as each step in plotting a line in octant 1:

15. $g(x, y) = f(x+1, y+1/2)$

For lines plotted in octant 1, this equation answers the question, "Wha just to the right of the current point we've plotted and halfway between the current point of $y$?" If the value is positive, then the line must be passing above $(x+1, y+1)$ as the next point. If the value is negative, then the line must be passing under our halfway point and this means we should plot $(x+1, y)$ as the next point. An exact zero would suggest we could plot $(x+1, y)$ as the next point.

Our remaining problem now is to find a way to efficiently calculate the next point. We can do this by ignoring $g(x, y)$ for a moment and thinking about how it changes as we plot points in octant 1, instead. The first value will be $G_1 = g(x_1, y_1)$. The next value will depend on that value we will plot next. This means we have two cases to consider:

16. 
   \begin{align*}
   G_2 &= G_1 + g(x_1 + 1, y_1) - g(x_1, y_1), \text{ when } G_1 < 0 \\
   G_2 &= G_1 + g(x_1 + 1, y_1 + 1) - g(x_1, y_1), \text{ when } G_1 \geq 0
   \end{align*}

You will have this same pair of cases to choose between, after each point is plotted, will determine the following point to plot. This will repeat over and over.

Repeating sequences like this, with an initial boundary condition, can be generally expressed better as the following:

17. 
   \begin{align*}
   G_i &= G_{i-1} + g(x_{i-1}, y_{i-1}) - g(x_i, y_i) \\
   \{X_i = x_{i-1} + 1, \quad Y_i = \begin{cases} Y_{i-1}, & \text{when } G_{i-1} < 0 \\ Y_{i-1} + 1, & \text{when } G_{i-1} \geq 0 \end{cases} \} \\
   \text{for } i > 1
   \end{align*}

We've just used something called mathematical deduction to describe the local effects of the function, but tells us a lot about what is going on near the point of interest. Recurrences are ripe for mathematical induction, which reverses this process. But that's for another day.

To compute successive values of $G_i$, let's expand on its recurrence in [18]

18. 
   \begin{align*}
   \Delta g_i &= g(x_{i-1} + 1, y_{i-1}) - g(x_{i-1}, y_{i-1}), \text{ when } G_{i-1} < 0 \\
   \Delta g_i &= g(x_{i-1} + 1, y_{i-1} + 1) - g(x_{i-1}, y_{i-1}), \text{ when } G_{i-1} \geq 0 \\
   \text{for } i > 1
   \end{align*}

   \begin{align*}
   G_i &= G_{i-1} + \Delta g_i
   \end{align*}
Now we need to do some algebra to see if this can be simplified. This is detailed, but not hard to follow. So, let's start with the first case listed in [18]. Going back to equations [14] and [15]

\[
\begin{align*}
\Delta g_i &= g(x_{i-1} + 1, y_{i-1}) - g(x_{i-1}, y_{i-1}) \\
&- f(x_{i-1} + 2, y_{i-1} + 1/2) - f(x_{i-1} + 1, y_{i-1} + 1/2) \\
&- [x_{i-1} + 2 - x_1] \cdot \Delta y - (y_{i-1} + 1/2 - y_1) \cdot \Delta x \\
&- [x_{i-1} + 1 - x_1] \cdot \Delta y - (y_{i-1} + 1/2 - y_1) \cdot \Delta x \\
&- [y_{i-1} + 2 - y_1] - (y_{i-1} + 1/2 - y_1) \cdot \Delta x \\
&- \Delta y - \Delta x
\end{align*}
\]

for \( i > 1 \)

Now, let's do the same thing for the second case:

\[
\begin{align*}
\Delta g_i &= g(x_{i-1} + 1, y_{i-1} + 1) - g(x_{i-1}, y_{i-1}) \\
&- f(x_{i-1} + 2, y_{i-1} + 3/2) - f(x_{i-1} + 1, y_{i-1} + 1/2) \\
&- [x_{i-1} + 2 - x_1] \cdot \Delta y - (y_{i-1} + 3/2 - y_1) \cdot \Delta x \\
&- [x_{i-1} + 1 - x_1] \cdot \Delta y - (y_{i-1} + 1/2 - y_1) \cdot \Delta x \\
&- [y_{i-1} + 2 - y_1] - (y_{i-1} + 3/2 - y_1) \cdot \Delta x \\
&- \Delta y - \Delta x
\end{align*}
\]

for \( i > 1 \)

Now we can restate [18] in a more concrete form:

\[
\begin{align*}
\Delta g_i &= \Delta y, \text{ when } G_{i-1} < 0 \\
\Delta g_i &= \Delta y - \Delta x, \text{ when } G_{i-1} \geq 0
\end{align*}
\]

for \( i > 1 \)

\[
G_i = G_{i-1} + \Delta g_i
\]

Take a quick breath for a moment, because this problem just got a lot easier! Notice that the change in our condition function, used to decide which point to plot in case you've already forgotten, only depends on the original some very simple values that we can pre-compute before starting out. This is good

We've one more detail to take care of, the initial value:

\[
G_1 = g(x_1, y_1) \\
= g(x_1 + 1, y_1 + 1/2) \\
= [x_1 + 1 - x_1] \cdot \Delta y - (y_1 + 1/2 - y_1) \cdot \Delta x \\
= \Delta y - \Delta x / 2
\]

That fraction will cause us some slight trouble if we plan to use integers throughout. For that reason alone, we will just multiply everything by 2. That will clear up the problem completely.
Let's fix up recursion [17] with this new information:

\[ X_i = x_i \quad Y_i = y_i \quad G_i = 2 \cdot \Delta y - \Delta x \]

20.

\[
\begin{cases}
X_i = X_{i-1} + 1 \\
Y_i = Y_{i-1} + 1 \\
G_i = \begin{cases} 
G_{i-1} + 2 \cdot \Delta y & G_{i-1} < 0 \\
G_{i-1} - 2 \cdot \Delta y & G_{i-1} \geq 0
\end{cases}
\end{cases}
\]

for \( i > 1 \)

Heck. The code in any language will practically write itself from that!

I'll bet you were thinking that there can't be any more, right? Hehe. Well, that just covered cases where we are plotting a line in octant 1. There are seven more of them, you know. I'm not the least bit tired, so shall we continue?

The Rest of the DDA Story

Let's take a look at all the octants for a moment. The following table shows each of the decision functions to use when selecting between two alternate points along the line. There are four with

Of each of these four, two are positive directed and two are negative. Look it over and verify in your own mind that these make sense.

Now it also turns out that the eight equations can be cut down to four by recognizing that all octant 3 lines can be converted to octant 7 lines by just swapping the starting and ending points. Similarly, octant 4 becomes octant 8, octant 5 becomes octant 1, and octant 6 becomes octant 2. You can take any octant drawing direction and convert it to the diametrically opposite octant this way, in fact. So you can convert eight situations into just four. That's a bit better.

Actually, it's lots better. Intuitively, you might notice that all these patterns amount to simple reflections of each other and that you probably need just one basic piece of logic. It turns out the magnitude, variable and the dependent variable and their respective directions that variables, two directions for the independent variable, and two possible

I won't tabulate the eight recurrences for you. You can do those as recurrence for octant 8:

<table>
<thead>
<tr>
<th>Octant</th>
<th>Decision Equation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( g(x,y)=f(x+1,y+1/2) )</td>
</tr>
<tr>
<td>2</td>
<td>( g(x,y)=f(x+1/2,y+1) )</td>
</tr>
<tr>
<td>3</td>
<td>( g(x,y)=f(x-1/2,y+1) )</td>
</tr>
<tr>
<td>4</td>
<td>( g(x,y)=f(x-1,y+1/2) )</td>
</tr>
<tr>
<td>5</td>
<td>( g(x,y)=f(x-1,y-1/2) )</td>
</tr>
<tr>
<td>6</td>
<td>( g(x,y)=f(x-1/2,y-1) )</td>
</tr>
<tr>
<td>7</td>
<td>( g(x,y)=f(x+1/2,y-1) )</td>
</tr>
<tr>
<td>8</td>
<td>( g(x,y)=f(x+1,y-1/2) )</td>
</tr>
</tbody>
</table>

http://users.easystreet.com/jkirwan/dda.html
21. The details start getting pretty tedious at this point, as if they haven't already been that way. It turns out that you can use a single routine to handle drawing all eight octants, if you think of the independent and dependent, rather than as $x$ and $y$.

There are two basic orientations for drawing the lines. One when the dependent variable increases as the independent variable increases and one when the opposite occurs, when the dependent variable decreases as the independent variable increases. So the DDA routine will need the starting and ending points for the independent variable, the starting point for the dependent variable and its orientation to the independent variable, and the magnitude of the spans between the points. With that in place, you can use a single piece of DDA logic to get the whole job done.

Even though the DDA algorithm can handle vertical and horizontal lines quite well, it's often better to handle a few situations as special cases in your code for better efficiency. Horizontal and vertical lines are simple and don't need decision variables. In fact, you can cover both vertical and horizontal lines in a single routine, if you like. Or to take advantage of special capabilities on the hardware, you may prefer to keep these separate. Similarly, lines exactly along 45-degree lines can use special code and can be all handled in a single routine, too.

Now let's see some code.

**Sample Line-Drawing Code**

I'm using the C language to illustrate the routines that follow. The code does not deal with pixel colors or clipping. Although most practical routines have to deal with such things, they aren't always needed and they are beyond the scope here. Also, the SetPixel routine isn't shown. You'll need to provide it, if you plan to test these routines.

Finally, I don't show the special case code I mentioned earlier, to handle vertical, horizontal, and isolated points. The code below copes with those special cases just fine without any special handling and I figure you will have little problem adding such code, if you want it. The point here is to showcase the DDA algorithm.

```c
extern void SetPixel (int x, int y);

typedef void SetPixelFunc (int a, int b);

void xySetPixel (int x, int y) { SetPixel (x, y); }

void yxSetPixel (int y, int x) { SetPixel (x, y); }

void PlotLineDDA (SetPixelFunc* pfPlot, int a1, int a2, int b, int db, int da) {
    int g1 = db + db;
    int g2 = g1 - da - da;
    int g = g1 - da;
    for (i = 0; i < 8; i++) {
        X[i] = x[i]; Y[i] = y[i];
        if (g < 0) {
            g += 2 * da + 2 * db;
            if (g < 0) {
                g += 2 * da;
                g1 += db + db;
                g2 += da + da;
            }
            else {
                g1 -= db + db;
                g2 -= da + da;
            }
        }
        else {
            g += 2 * db;
            if (g < 0) {
                g += 2 * db;
                g1 -= da + da;
                g2 -= db + db;
            }
            else {
                g1 += da + da;
                g2 += db + db;
            }
        }
    }
}
```

http://users.easystreet.com/jkirwan/dda.html
for (int a = a1; a < a2; ++a) {
    pfPlot (a, b);
    if (g < 0)
        g = g + g1;
    else {
        g = g + g2;
        b = b + bc;
    }
}
pfPlot (a, b);

void PlotLine (int x1, int y1, int x2, int y2) {
    int dxabs = abs (x2 - x1);
    int dyabs = abs (y2 - y1);
    if (dxabs >= dyabs) {
        if (x1 < x2)
            PlotLineDDA (xySetPixel, x1, x2, y1, dxabs, dyabs, y1 < y2? 1:
        else
            PlotLineDDA (xySetPixel, x2, x1, y2, dxabs, dyabs, y2 < y1? 1:
    } else {
        if (y1 < y2)
            PlotLineDDA (yxSetPixel, y1, y2, x1, dyabs, dxabs, x1 < x2? 1:
        else
            PlotLineDDA (yxSetPixel, y2, y1, x2, dyabs, dxabs, x2 < x1? 1:
    }
}

Summary

Ok. Ok. I know. Ten pages of nasty math and all you get for it is th
dragging you through that mine field when there was a short cut throug

Well, it was good for you. Isn't a brisk bit of algebra just what it t
anyway? And imagine – now you can tackle my article on that circle-
to the end, skipping all that thrilling mathematics.)

Yours truly,
Jon

Quick Links:

- [mail me]
- [home page]
- [parent page]
Creation Date:  Mon 15-Nov-1999 01:59:16
Last Modified:  Mon 15-Nov-1999 01:59:19
Copyright (C) 1999 Jonathan Dale Kirwan