DDA Algorithm

CS 130 Spring

April 12, 2007

1 Introduction

This week you will be learn about and implement the Digital Difference Analyzer algorithm (DDA). First lets recall some basics about the equation of a line. Given two points in a plane (x_0, y_0) and (x_1, y_1) the slope-intercept form of a line can be defined as

$$y = m * x + b$$

Where m is the slope of the line

$$m = \frac{y_1 - y_0}{x_1 - x_0}$$

The y intercept b can be computed by isolating it, giving you

$$b = y_0 - m * x_0$$

One approach to rasterizing the line is through direct scan conversion, whereby each x-value in the interval $[x_0, x_1]$ is plugged into the slope-intercept equation to get the corresponding y-value. Then we can plot the pixel $(x, Round(y))$ onto the display. Although this works we are performing one floating point addition and multiplication to get each y-value. In turns out that there is a more efficient way of accomplishing the same goal, which brings us to DDA algorithm (Digital Difference Analyzer).

2 DDA

DDA works by using the pixel coordinates computed last and the slope. From the previous section you can easily compute how far to move over (δx) or how far to move up (δy) on the screen to draw your next point on the line.

$$\delta y = m * \delta x$$

$$\delta x = \frac{\delta y}{m}$$

Given that you already know you want to move over (δx) or up (δy) a certain number of units (for computers these units equate to pixels on the screen) you then plug these values into one of the two equations along with the slope and solve.

The DDA algorithm can be then broken down into the following cases.

1. A line with positive slope and $0 \leq m \leq 1$ then you can take $\delta x = 1$ and compute successive y values using.

   $$y_{k+1} = y_k + m$$

   Note: remember that because m is not necessarily an integer you must round your result.

2. A line with positive slope and $m > 1$ then you calculate x instead of y and take $\delta y = 1$ and compute succesive x values using

   $$x_{k+1} = x_k + \frac{1}{m}$$

   The same applies here you need to round!

Both of these calculations can generalize to lines with negative slope so you can use case 1 when $|m| \leq 1$ and case 2 when $|m| > 1$.

How is this algorithm faster than directly implementing the equation of the line?
What is a drawback to this algorithm?