1 DDA Algorithm

DDA Algorithm is used to efficiently compute raycasting (not to be confused with raytracing) on a 2D grid. Personally I find it very elegant when it comes to separation of algorithm's logic and displaying/rendering that logic.

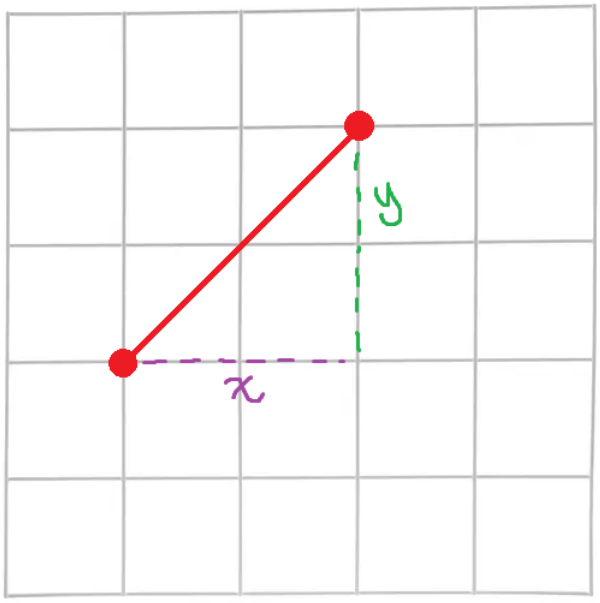

Figure 1: DDA on grid aligned points

DDA at its core is just pythagoras' theorem and some simple logic, one assumption we need to make first is that all our squares are unit size (i.e. 1 by 1). Working from that assumption we can easily calculate lengths of 'rays' we're interested in.

What DDA is actually doing is: it's walking down a ray and detecting collisions along that ray, naively, something like this could be achieved by incrementing some kind of 'raystep' by a small amount that however may yield incorrect results. DDA avoids that problem by travelling the 'perfect' (for some definitions of perfect) distance and detecting collisions at an exact point.

Previously I stated that the core of DDA is pythagoras' theorem plus an important assumption that all our squares are unit size. This has important implications on how we can calculate the length of hypotenuse moving either one unit to the left/right or up/down.

Normally length of a hypotenuse looks like this:

$$l = \pm\sqrt{x^2 + y^2}$$

After we divide by x or y (to move exactly one unit in either direction) we get this:

$$l_x = \pm\sqrt{1 + \frac{y^2}{x^2}} \qquad l_y = \pm\sqrt{1 + \frac{x^2}{y^2}}$$

Of course in both cases we're only interested in the + case, since distance cannot be negative (disregarding SDFs and such).

Believe it or not this is half of the entire algorithm already, knowing this we know how much we need to move in order to reach the next horizontal or vertical wall. Choosing the smaller between the two (horizontal or vertical length) we can check if we're hitting the wall.

It is also worth noting that because this gives us unit length, meaning, when we move 1 unit in either direction we know how much we moved along hypotenuse, it is easy to calculate the length of that hypotenuse for any number of units we move, it's just a matter of mulitplication.

$$S \cdot l_x = S\sqrt{1 + \frac{y^2}{x^2}}$$

Where S is how many units we moved along x axis (in this case).

Last thing worth addressing is, what about the case when our points are not grid aligned? Well, because we're dealing with computers and integers etc. When we have a starting point like $(3.25; 4.77)$ we can just truncate it to be $(3; 4)$, after all we're checking what cell we're in and which cells have walls that our ray(s) intersect.

To finish off the algorithm we just need to take the shorter distance between the two x and y directions after that it's a matter of incrementing the step in that smaller direction.

One thing worth adding is that because we're axis aligned when we move to the right or up we need to check distance by adding 1 before subtraction, what I mean by that is hard for me to put into words so I leave some helpful code implementation on my github (github.com/zermil/graphics-demos) to play around with.