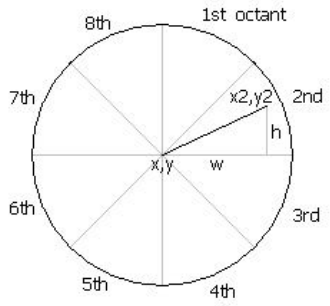
Drawing Line Using Bresenham Algorithm

11 October 2007, 12:01 AM

The Bresenham algorithm is probably the most efficient of all line drawing algorithm. It greatly simplifies line drawing by using only integer variables, and importantly removing that costly division operation for slope. Before we begin impementing the algorithm, it is advisable to revise the method for drawing line in an inefficient way. This can help lay out the fundamentals of line algorithm, and is very useful since bresenham algorithm itself is an extension of the inefficient one anyway. The figure below, is circle showing eight octants of region and in the second region lies a line (x,y,x2,y2). The following java code is the logic for drawing line just in that region.




```
public void line(int x,int y,int x2, int y2, int color) {
    int w = x2 - x ;
    int h = y2 - y ;
    double m = h/(double)w ;
    double j = y ;
    for (int i=x;i<=x2;i++) {
        putpixel(i,(int)j,color) ;
        j += m ;
    }
}
```

As you can see that is one inefficient code, but it works just fine. Take note however, above is not a full line function, it only draws properly in the second octant region. A full line function is available at the end of this article if you can't be bothered with understanding the algorithm.

If we look at the fraction m = h/w and the addition j += m, the operation is actually increasing the numerator h, and keeping the denominator w constant whenever m is added to j. When the numerator h becomes equal or greater than the denominator w, the numerator is subtracted with the denominator and j is increased by 1. This reasoning is what bresenham algorithm is about, an with this the algorithm is able to replace the division and real number operation with just additions and a simple rule. See code below for the implementation of bresenham algorithm in the second octant.

```
public void line(int x,int y,int x2, int y2, int color) {
    int w = x2 - x ;
    int h = y2 - y ;
    int dy1 = -1 ;
    int longest = Math.abs(w) ;
    int shortest = Math.abs(h) ;
    int numerator = longest >> 1;
    for (int i=0;i<=longest;i++) {
        putpixel(x,y,color) ;
        numerator += shortest ;
        if (!(numerator<longest)) {
            numerator -= longest ;
            y += dy1 ;
        }
        x++ ;
    }
}
```

Notice the line *numerator = longest >> 1*? Technically it means numerator is equal to half of *longest*, and is important to avoid y from being rounded at every whole number instead of halfway point.

Writing a complete bresenham line function will require consideration of all eight octants and it is easy to imagine to have eight copies of above code. Fortunately the copies are all very similar since they are just mirrors of each other, and it is very easy to write a generic code that takes care of all eight octants. But first let us look on the similarities in different octants.

Category

» Neural Network
» Path Finding
» Image Processing
» Graphics Programing
» Sorting

Recent Posts

» Quicksort
» Shell Sort
» Insertion Sort
» Selection Sort
» Bubble Sort
» Trilinear Interpolation Image Scaling

Popular Posts

» Nearest Neighbor Image Scaling
» Bilinear Image Scaling
» Drawing Line Using Bresenham Algorithm
» Dijkstra's algorithm java applet interactive demo
» Box Filtering
» Dijkstra's algorithm

Drawing a line in the third octant is simply drawing a line in the second octant with mirrored y. This can be reflected by setting dy1 = 1, instead of -1. See below at line number four for the difference.

```
public void line(int x,int y,int x2, int y2, int color) {
    int w = x2 - x ;
    int h = y2 - y ;
    int dy1 = 1 ;
    int longest = Math.abs(w) ;
    int shortest = Math.abs(h) ;
    int numerator = longest >> 1;
    for (int i=0;i<=longest;i++) {
        putpixel(x,y,color) ;
        numerator += shortest ;
        if (!(numerator<longest)) {
            numerator -= longest ;
            y += dy1 ;
        }
        x++ ;
    }
}
```

The sixth octant is the same with the third one, except with mirrored x. This can be done by changing the line x++ to x--.

```
public void line(int x,int y,int x2, int y2, int color) {
    int w = x2 - x ;
    int h = y2 - y ;
    int dy1 = 1 ;
    int longest = Math.abs(w) ;
    int shortest = Math.abs(h) ;
    int numerator = longest >> 1;
    for (int i=0;i<=longest;i++) {
        putpixel(x,y,color) ;
        numerator += shortest ;
        if (!(numerator<longest)) {
            numerator -= longest ;
            y += dy1 ;
        }
        x-- ;
    }
}
```

The same goes for the seventh octant, it is the same with the second octant except with mirrored x. Again this can be done by changing the line x++ to x--.

```
public void line(int x,int y,int x2, int y2, int color) {
    int w = x2 - x ;
    int h = y2 - y ;
    int dy1 = -1 ;
    int longest = Math.abs(w) ;
    int shortest = Math.abs(h) ;
    int numerator = longest >> 1;
    for (int i=0;i<=longest;i++) {
        putpixel(x,y,color) ;
        numerator += shortest ;
        if (!(numerator<longest)) {
            numerator -= longest ;
            y += dy1 ;
        }
        x-- ;
    }
}
```

Looking at all this similarities, we can now devise a more generic function that draws line in the second, third, sixth, and seventh octants. Below is the code just to do that,

```
public void line(int x,int y,int x2, int y2, int color) {
    int w = x2 - x ;
    int h = y2 - y ;
    int dx1 = 0, dy1 = 0, dx2 = 0, dy2 = 0 ;
    if (w<0) dx1 = -1 ; else if (w>0) dx1 = 1 ;
    if (h<0) dy1 = -1 ; else if (h>0) dy1 = 1 ;
    if (w<0) dx2 = -1 ; else if (w>0) dx2 = 1 ;
    int longest = Math.abs(w) ;
    int shortest = Math.abs(h) ;
    int numerator = longest >> 1 ;
    for (int i=0;i<=longest;i++) {
        putpixel(x,y,color) ;
        numerator += shortest ;
        if (!(numerator<longest)) {
```

```
            numerator -= longest ;
            x += dx1 ;
            y += dy1 ;
        } else {
            x += dx2 ;
            y += dy2 ;
        }
    }
}
```

At this point we have a function that can draw lines on the second, third, sixth, and seventh octants. However this is still incomplete, we need to consider the others before we can call it a full line function. To do that, we observe that lines that fall in the first, fourth, fifth, and eight octants will have its height longer than its width, unlike in the second, third, sixth, and seventh octant where the lines have widths longer than heights. By including this in the logic, the result will be a full implementation of bresenham algorithm for drawing line on all octants.

Here is the full implementation,

```
public void line(int x,int y,int x2, int y2, int color) {
    int w = x2 - x ;
    int h = y2 - y ;
    int dx1 = 0, dy1 = 0, dx2 = 0, dy2 = 0 ;
    if (w<0) dx1 = -1 ; else if (w>0) dx1 = 1 ;
    if (h<0) dy1 = -1 ; else if (h>0) dy1 = 1 ;
    if (w<0) dx2 = -1 ; else if (w>0) dx2 = 1 ;
    int longest = Math.abs(w) ;
    int shortest = Math.abs(h) ;
    if (!(longest>shortest)) {
        longest = Math.abs(h) ;
        shortest = Math.abs(w) ;
        if (h<0) dy2 = -1 ; else if (h>0) dy2 = 1 ;
        dx2 = 0 ;
    }
    int numerator = longest >> 1 ;
    for (int i=0;i<=longest;i++) {
        putpixel(x,y,color) ;
        numerator += shortest ;
        if (!(numerator<longest)) {
            numerator -= longest ;
            x += dx1 ;
            y += dy1 ;
        } else {
            x += dx2 ;
            y += dy2 ;
        }
    }
}
```

Last edited on 15 October 2007

Comments

~**nminhtai**, 7 September 2009, 08:00 AM
Great tutorial! Thanks a lot.

~**Marc**, 2 December 2009, 02:37 PM
thanks!

greetings form austria :)

~**loreen**, 26 March 2010, 11:56 AM
thanks a lot it is very helpful

~**Tomas**, 21 October 2010, 04:46 PM
Nice explanation. Thank you :)

~**Ron**, 7 April 2011, 09:02 AM
This helps me a lot, thank you :)

~**salamath**, 27 April 2011, 06:02 AM
Great tutorial! Thanks a lot.

~**ishwor**, 19 June 2011, 06:57 AM
it\'s so helpful

~**bhatta**, 19 June 2011, 06:59 AM
it,s great

~**gopi**, 1 August 2011, 11:49 AM
it is good

~**sameer gupta**, 22 September 2011, 06:25 AM