A
Close Look at
Breshenham Algorithm
(Line Circle and Ellipse)

Prakash Gautam

First Published Jan 19, 2014
Last Compiled April 16, 2015

Contents

Preface

I really don't know what should a good preface look like, but since before reading any book one should at least go through its preface which in turn will decide whether the reader will go any further, I shall try to encourage to at least go through all the pages of this book(?).

This book is my attempt to try to explain what I pretend to understand. At first I thought Breshenham algorithm was just some kind of rounding off algorithm to approximate the floating point output of equation to integers to plot into the always-integer-valued pixels. But this algorithm turns out to be way more than that. It is very very simple and yet very powerful that cuts short the complex calculations needed otherwise to find the points of certain geometric shapes.

I don't assume that the reader has knowledge of Breshenham algorithm prior to this yet prior knowledge does no harm but help. Optional knowledge of C++ programming is good if the reader wants to use the C++ program codes given.
This book is completely my original writing. Except for the Q/A I did on the class with my teacher, I've not referred any books or internet or anything. But the success of programs written and tested times and again in C++ suggest that the discussion given on the book are correct. As far as possible I've tried to explain the matter in plain language with common words. No figures used in this book are taken from any existing source.

I would like to thank my friends form encouraging me to write this and their appreciation has always pushed me further. Also I shall thank my friends in the class for coping me and my rigorous questions. I would sincerely thank my teacher who suggested me to use LaTeX and answered my questions in the class without any irritation. I shall thank him for his lecture in the class which introduced me to Breshenham algorithm,

Comments and suggestions are always welcome and are sincerely expected and accepted.

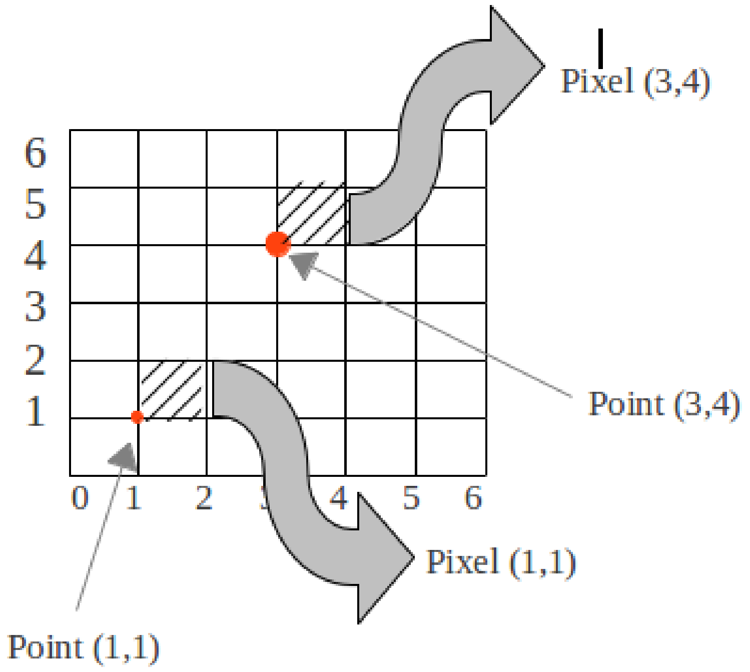
Prakash Gautam
pranphy@gmail.com
http://pranphy.wordpress.com/

http://facebook.com/pranphy
http://facebook.com/respectthedignity

Before We Begin

The earliest I can remember of pixel is of a advertisement of a famous television brand which said *more pixel more detail* in it. I had no idea then what it meant but *pixel* was a nice word, easy on the ears. Now I pretend to realize what that meant. *more pixel more detail.* Pixel is derived from two English words, *Picture* and *element. Pic+el=Pixel.* It differs from what is taken as the display unit, as what exactly is a pixel, but commonly referred pixel is a light bulb(possibly their collection).
So how do we justify, *more pixel more detail.* We would rather not buy the television which broadcast this advertisement, rather as students, we will build up our own Visual Display Unit.
Lets assemble a few LED's. If I give you just 3 LED's and ask you to make a shape of circle, the best you could do is to arrange them at the corners of a equilateral triangle and light them all. If I see your misery and give two extra, still the best you could is to arrange them in the corners of a pentagon. But the latter would certainly be a better circle than the former one. If I go on reducing your misery of making a circle and keep giving you more and more LED's you would make better and better circle.
In the first case with three LED's, if you leave the circle to my mom- who doesn't know much about your limitation of pixel- she would exclaim you arranged the led's into a triangle not a circle. You certainly gave less detail of the shape to my mom so she exclaimed that . Because you assumed the space between the two LED's were a curved path but for my mom its more natural to assume a straight line, so to everybody else.
With increase of LED's the circle becomes finer and finer. You see more details where does the arc go between two leds. The more LED's I give you, the better detail of a circle you can give me back. Hence *more pixel more detail.*

Figure 1: One Is pixel and other is point



Since we are going to deal with determining various geometrical shapes in this document we want to be clear how do we represent pixel throughout the book. Though there could be alternative ways of doing so, we would stick to the above method in the book.

Chapter 1

Breshenham Algorithm : Line

1.1 Background

It took me quite a while to appreciate the beauty of Brehsenham Line Drawing algorithm. Initially I thought It was just rounding off the decimal point to the nearest integer only in a zig-zag Barcelona way but to my fascination it turned out to be tremendously interesting yet very very finely simple. If two non-coincident points $(x_0, y_0)$and (x_1, y_1) be given. The equation of line passing through these two points will be.

$$y = y_0 + \frac{y_1 - y_0}{x_1 - x_0}(x - x_0) \tag{1.1}$$

$$y = y_0 + \tfrac{\Delta y}{\Delta x}(x - x_0)$$

$$y = y_0 + m(x - x_0)$$

$$y = mx + (y_0 - mx_0) \tag{1.2}$$

Comparing equation 1.2 with $y = mx + c$ we see

$$c = y_0 - mx_0 \tag{1.3}$$

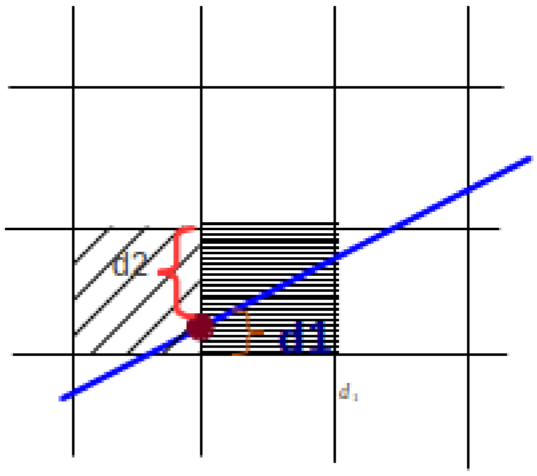
It is not difficult to see if two points (x_0, y_0) and (x_1, y_1) are given then we can easily calculate the remaining points at any x with the use of equation 1.1 simply substuting value of x. Let us think how do we calculate the points with a computer program. We would calculate the slope m and c then start a loop from x_0 with step 1 and put value of x in equation 1.2 and simply find the values of y corresponding to the value of x. The value of m is generally a floating point number. Computer needs bit more labour to calculate the result of a floating point. Modern day VDU's are millions pixels in dimension. Just to draw a line across the VDU, poor CPU needs to do millions of floating point calculations which sucks more out of CPU. Lets fiddle around if we could somehow help the needy CPU here to avoid floating point multiplications.

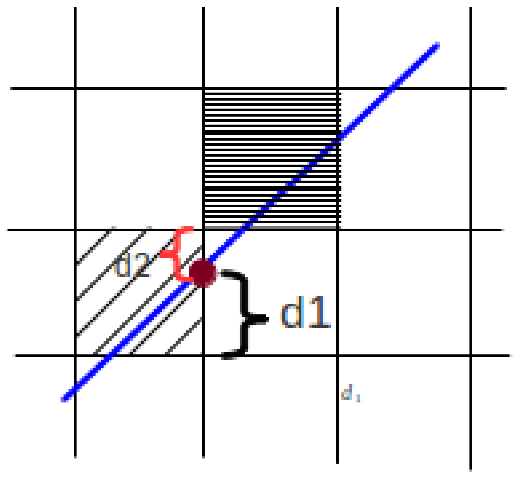
1.2 The Decision Parameter

Lets first concentrate on special case where the slope of line $0 \leq m \leq 1$. If the first point is (x_0, y_0) and then with unit increase in the value of x should increase y by the value of slope m which is less than 1. But m is a floating point number, and the pixel coordinates are always integers. Let us assume that a intermediate point (x_k, y_k) is calculated already, the next point (x_{k+1}, y_{k+1}) is either $(x_k + 1, y_k)$ or $(x_k + 1, y_k + 1)$, (Note in both cases x increases by just 1). If we could somehow decide which one between these two to choose then our line would be known by induction. Let us find out which one of the integer is closer to the actual line. The actual coordinate of the line at $x_k + 1$ is $y = m(x_k + 1) + c$ The distance between the actual position from the two points to be decided are.

$$d_1 = [m(x_k + 1) + c] - y_k \tag{1.4}$$

$$d_2 = y_k + 1 - [m(x_k + 1) + c] \tag{1.5}$$





The line crosses the ordinate at point below the mid point, which implies the lower pixel is nearer to line so the ordinate is not increased. But in the second case the actual line crosses the ordinate above the mid point which implies higher pixel is closer to line, so the ordinate is increased.

We will chose the next y coordinate y_k if $d_1 \leq d_2$ or else we will choose $y_k +$ 1.Subtracting equation 1.5 from equation 1.4 we get.

$$d_1 - d_2 = [m(x_k + 1) + c] - y_k - (y_k + 1 - [m(x_k + 1) + c])$$

$$\Delta d = 2mx_k - 2y_k + 2m + 2c - 1 \tag{1.6}$$

Here we are concentrating on line with slope $0 \leq m \leq 1$ so the value $(\Delta x = x_1 - x_0) \geq 0$. so multiplying by Δx on both sides of equation 1.6 preserves sign of $d_1 - d_2$. Lets call this quantity P_k.

$$P_k = \Delta x(2mx_k - 2y_k + 2m + 2c - 1)$$

$$\Rightarrow P_k = 2\Delta y x_k - 2\Delta x y_k + 2\Delta y + \Delta x(2c - 1) \tag{1.7}$$

So, Δd and $P_k < 0$ have the same sign. If $P_k < 0$ we'll chose $(x_k + 1, y_k)$ else $(x_k + 1, y_k + 1)$ as the next point.

1.3 Recursive Relation Of Decision Parameter

This expression above is the discriminant which decides which one between the two points in contention do we chose. For this reason P_k is called the decision parameter. If P_k determines the k^{th} point then the next point is decided by P_{k+1}. Simply putting $k+1$ for k in equation 1.7 and keeping in mind that $x_{k+1} = x_k + 1$ we get:

$$P_{k+1} = 2\Delta y(x_k + 1) - 2\Delta x y_{k+1} + 2\Delta y + \Delta x(2c - 1) \tag{1.8}$$

Simplifying equation 1.8 and equation 1.7 we get:

$$P_{k+1} = P_k + 2\Delta y - 2\Delta x(y_{k+1} - y_k) \tag{1.9}$$

We are down to just knowing the sign of P_k to decide whether the next point after (x_k, y_k) is $(x_k + 1, y_k)$ or $(x_k + 1, y_k + 1)$.

$$y_{k+1} = \begin{cases} y_k & \text{if } P_k \leq 0 \\ y_k + 1 & \text{if } P_k > 0 \end{cases} \tag{1.10}$$

1.4 Initial Decision Parameter

The first point undoubtedly is (x_0, y_0) but what is the first decision parameter? After first point (x_0, y_0) is known we can know P_0. Simply put $(x_k, y_k) = (x_0, y_0)$ in equation 1.7.

$$P_0 = 2\Delta y x_0 - 2\Delta x y_0 + 2\Delta y + \Delta x(2c - 1)$$

Putting the value of c from equation 1.3

$$P_0 = 2\Delta y x_0 - 2\Delta x y_0 + 2\Delta y + \Delta x[2(y_0 - m x_0) - 1]$$

$$\Rightarrow P_0 = 2\Delta y - \Delta x \tag{1.11}$$

1.5 Finding The Next Point

So after (x_0, y_0) is found, we can calculate P_0 by knowing value of Δx, Δy,x_0 and y_0. If $P_0 \leq 0$ the next point is $(x_0 + 1, y_0)$ if $P_0 > 0$ the next point is $(x_1, y_1) = (x_0 + 1, y_0 + 1)$. So P_0 is calculated from (x_0, y_0) and determines (x_1, y_1) and so on. P_k is calculated from (x_k, y_k) and determines (x_{k+1}, y_{k+1})
Summarizing this:

$$y_1 = \begin{cases} y_0 & \text{if } P_0 \leq 0 \\ y_0 + 1 & \text{if } P_0 > 0 \end{cases} \tag{1.12}$$

Putting $k = 0$ in equation 1.9 we get:

$$P_1 = P_0 + 2\Delta y - 2\Delta x(y_1 - y_0)$$

Which determines (x_2, y_2) and so on.. and hence all the points of the line.

1.6 The Breshenham Algorithm

Writing out algorithm for this:

```
Data: Get end points $(x_0, y_0)$ $(x_1, y_1)$
$P = 2\Delta y - \Delta x$;
$(x, y) = (x_0, y_0)$;
while $x \leq y$ do
    Plot$(x, y)$;
    $t \leftarrow y$;
    if $P \geq 0$ then
        $y \leftarrow y + 1$;
    end
    $P \leftarrow P + 2\Delta y - 2\Delta x(y - t)$;
    $x \leftarrow x + 1$;
end
```

Algorithm 1: Breshenham Line Algorithm for $0 \leq m \leq 1$

1.7 Lines With Any Slope

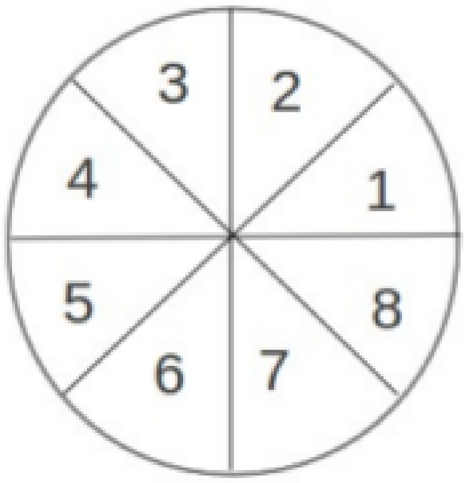
But line in real life are never always restricted to have the slope $0 \leq m \leq 1$. How can we extend this method to find the points of line in all the situation where the value of slope may be any number from the real axis?

Ok, lets begin the ride.

Lets us divide the Cartesian plane into 8 equal octant. The lines with slope with multiples of $\frac{\pi}{4}$ will suffice our division of the planes into octant. From our journey so far we have completed the calculation of points of line anywhere in the first octant. Now the job is to conquer the rest of the octant. There is a nice symmetry between the rest of the octant with the first. There exist a unique transformation matrix that transforms the line back and forth between first and the rest octant. Now the job is almost over. Identify the octant and transform it to first octant, calculate the points with the artifice we have done so far, and transform back to the original octant to plot. Bravo!! we conquered Everest. Lets do the mathematics of the idea then.

1.7.1 Finding The Octant

Let us, for our ease, transform the co-ordinate system with axes parallel but the origin transferred to the initial point of line. We do this because we would not then have to worry about two points to determine a line, only the final point would describe the line completely as the first point will be the origin of our new coordinate system. It is easy to determine the octant, as we have divided above, which the line lies in, simply by finding out which octant the final point lies in.



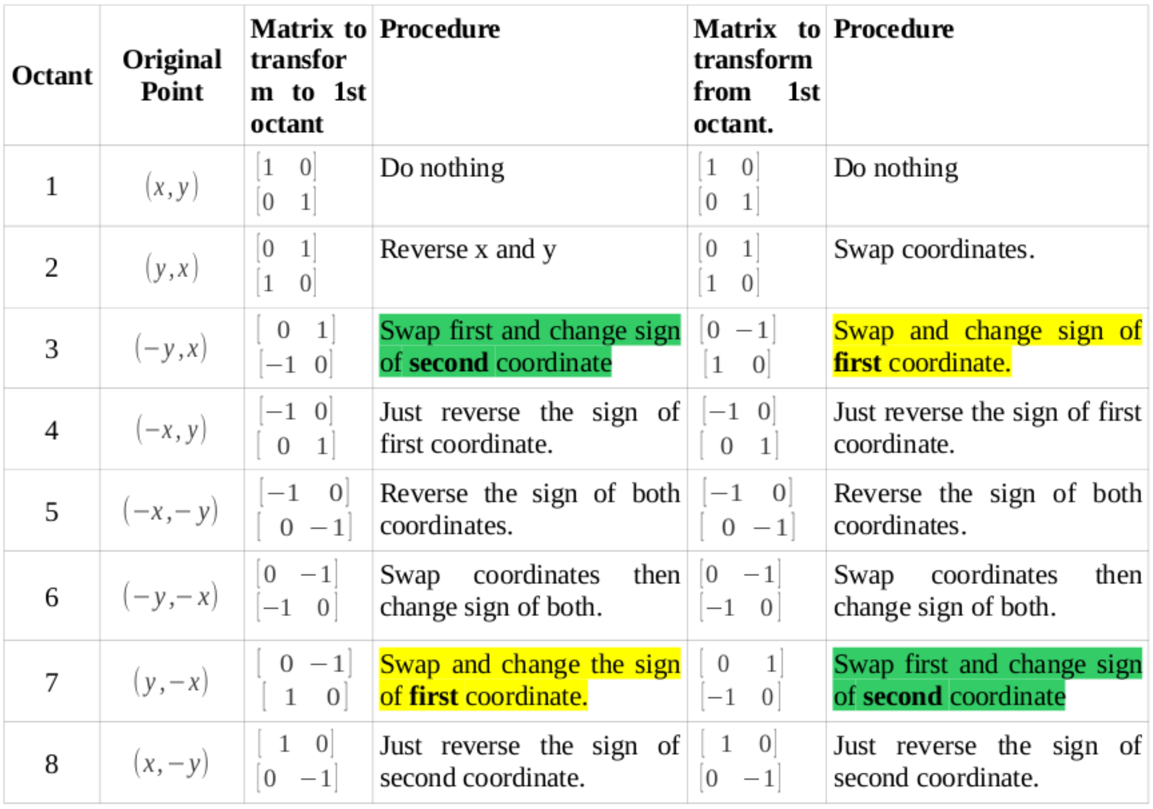
If the x coordinate is positive there are four candidate for the octant namely 1,2,8,7. Among these four the candidates are down to 1 and 2 if the y coordinate is +ve as well. And whether the y coordinate is greater than x determines whether the line lies on 2nd quadrant or first. Similarly the rest of the octants are determined.

The table below shows how to figure how the octant of the line with the coordinate of the end point.

$x \geq 0$	$y \geq 0$	$x \geq y$	1
		$x < y$	2
	$y < 0$	$\|x\| \geq \|y\|$	8
		$\|x\| < \|y\|$	7
$x < 0$	$y \geq 0$	$\|x\| \geq \|y\|$	4
		$\|x\| < \|y\|$	3
	$y < 0$	$\|x\| \geq \|y\|$	5
		$\|x\| < \|y\|$	6

1.7.2 The Transformation Matrix

After we have found out the octant of the line with this. Our task is transform every line into first octant then find the corresponding points on first octant and transform back to original octant to plot. What transforms the points back and forth between the given octant and the octant number 1. All the points are transformed to first octant where the coordinate of the point is (x, y)



Octant	Original Point	Matrix to transfor m to 1st octant	Procedure	Matrix to transform from 1st octant.	Procedure
1	(x,y)	$\begin{bmatrix}1 & 0\\0 & 1\end{bmatrix}$	Do nothing	$\begin{bmatrix}1 & 0\\0 & 1\end{bmatrix}$	Do nothing
2	(y,x)	$\begin{bmatrix}0 & 1\\1 & 0\end{bmatrix}$	Reverse x and y	$\begin{bmatrix}0 & 1\\1 & 0\end{bmatrix}$	Swap coordinates.
3	$(-y,x)$	$\begin{bmatrix}0 & 1\\-1 & 0\end{bmatrix}$	Swap first and change sign of **second** coordinate	$\begin{bmatrix}0 & -1\\1 & 0\end{bmatrix}$	Swap and change sign of **first** coordinate.
4	$(-x,y)$	$\begin{bmatrix}-1 & 0\\0 & 1\end{bmatrix}$	Just reverse the sign of first coordinate.	$\begin{bmatrix}-1 & 0\\0 & 1\end{bmatrix}$	Just reverse the sign of first coordinate.
5	$(-x,-y)$	$\begin{bmatrix}-1 & 0\\0 & -1\end{bmatrix}$	Reverse the sign of both coordinates.	$\begin{bmatrix}-1 & 0\\0 & -1\end{bmatrix}$	Reverse the sign of both coordinates.
6	$(-y,-x)$	$\begin{bmatrix}0 & -1\\-1 & 0\end{bmatrix}$	Swap coordinates then change sign of both.	$\begin{bmatrix}0 & -1\\-1 & 0\end{bmatrix}$	Swap coordinates then change sign of both.
7	$(y,-x)$	$\begin{bmatrix}0 & -1\\1 & 0\end{bmatrix}$	Swap and change the sign of **first** coordinate.	$\begin{bmatrix}0 & 1\\-1 & 0\end{bmatrix}$	Swap first and change sign of **second** coordinate
8	$(x,-y)$	$\begin{bmatrix}1 & 0\\0 & -1\end{bmatrix}$	Just reverse the sign of second coordinate.	$\begin{bmatrix}1 & 0\\0 & -1\end{bmatrix}$	Just reverse the sign of second coordinate.

From above table we see that same matrix transforms back and forth between all the octants except for octant number 3 and 7. It seems that we have to keep track of whether we are transferring from octant 3 to 1 or 1 to 3. And similarly for octant 7.

We can do this by using one transformation to transform from given octant to the first. And the other to transform from 1st to the given octant. For all but 3 and 7 both the transformation would be same. Just two of these odd octants are trying to force us to make two transformations for the well behaved 6 of the rest. Can we get over the odds?? Luckily yes. We get extremely lucky here, there is a symmetry between the odd ones. Once the point is transferred from 3rd to the 1st octant the point transforms back to 3rd octant exactly as the point is transferred from 7th to the first. And also once the point is transferred from 7th to 1st octant the point transfers back to the 7th octant exactly as the point is transferred from 3rd to 1st. Lets go fooling around. First transfrom the point normally from the given octant. If the octant is one of 7 or 3 then fool the system by changing the octant index to 3 or 7 respectively. Then do normally.

1.8 Implementation On C++

Below is a computer program written in C++ implementing the algorithm.

```
#include<iostream>
using namespace std;
struct Point {int x,y;};

ostream& operator<<(ostream&,Point);
istream& operator>>(istream&,Point&);

Point operator+(Point,Point);
Point operator-(Point,Point);
int FindOctant(Point);
Point ProperPoint(Point,int);
void BreshenhamLine(Point,Point);

template <typename DataType>
void Swap(DataType&,DataType&);
```

```
int main()
{
    Point InitialPoint,FinalPoint;
    cin>>InitialPoint>>FinalPoint;
    BreshenhamLine(InitialPoint,FinalPoint);
    return 0;
}

void BreshenhamLine(Point InitialPoint,Point FinalPoint)
{
    Point P=FinalPoint-InitialPoint; //transfer origin
    int OctantIndex=FindOctant(P);
    P=ProperPoint(P,OctantIndex); //transform point
    OctantIndex=(OctantIndex==7?3:(OctantIndex==3)?7:OctantIndex); //fooling around you
        see?
    Point CurrentPoint={0,0};
    int p=2*P.y-P.x;
    do
    {
        cout<<InitialPoint+ProperPoint(CurrentPoint,OctantIndex)<<endl; //transfer origin
             back
        int t=CurrentPoint.y;
        if(p>0) CurrentPoint.y++;
        p=p+2*P.y-2*P.x*(CurrentPoint.y-t);
        CurrentPoint.x++;
    }
    while(CurrentPoint.x<=P.x);
}

int FindOctant(Point P)
{
    if(P.x>=0)
        if(P.y>=0)
            return (P.y>P.x?2:1);
        else
            return (-P.y<P.x?8:7);
    else
        if(P.y>0)
            return (P.y>-P.x?3:4);
        else
            return(-P.y>-P.x?6:5);

}

Point ProperPoint(Point P,int d)
{
    if(d==2||d==3||d==6||d==7)
         Swap(P.x,P.y);
    if(d==7||d==4||d==5||d==6)
         P.x*=-1;
    if(d==3||d==8||d==5||d==6)
         P.y*=-1;
    return P;
}

Point operator+(Point P1,Point P2)
{
    Point P;
    P.x=P1.x+P2.x;
    P.y=P1.y+P2.y;
    return P;
}

Point operator-(Point P1,Point P2)
{
    Point P;
    P.x=P1.x-P2.x;
    P.y=P1.y-P2.y;
    return P;
}

ostream& operator<<(ostream& op,Point p)
{
    cout<<"("<<p.x<<","<<p.y<<")";
    return op;
}

istream& operator>>(istream&ip,Point&p)
{
    cout<<" Enter point"; cin>>p.x>>p.y;
    return ip;
}

template <typename DataType>
void Swap(DataType& p1,DataType&p2)
{
    DataType p;
    p=p1;
    p1=p2;
    p2=p;
}
```

Chapter 2

Breshenham Algorithm : Circle

2.1 Definition

A *circle* is locus of a moving point which moves such that it's distance from a fixed point- called center- is always constant- called the radius. This is the basic definition of circle which completely describes a circle.

2.2 Equation of Circle

The most well known equation of a circle which uses the above definition is:

$$(x - x_0)^2 + (y - y_0)^2 = r^2 \tag{2.1}$$

where (x_0, y_0) is the coordinate of the center and r is the radius of the circle. if $(0, 0)$ is the center of the circle the equation reduces to:

$$x^2 + y^2 = r^2 \tag{2.2}$$

Circle can be mathematically known with other equations as well. The parametric equation of a circle are:

$$x = r\cos\theta \tag{2.3}$$

$$y = r\sin\theta \tag{2.4}$$

If our task is to find the points of circle with given center and radius, we would be more inclined to use the relation 2.2. Simplifying it.

$$y = \sqrt{r^2 - x^2} \tag{2.5}$$

Using this equation we can find the value of y for any given value of x. For reasons mentioned earlier we would rather not ask CPU to work this equation out for long. There has to be some way. Let's find the way.

2.3 Two Possible Points

With the process mentioned earlier, lets again divide the plane into 8 octants. Assuming the circle is centered at origin and given radius r lets find points of the lying in the second octant,(octant divided as in the figure in the previous chapter). We begin with the known point on the second octant $(0, r)$. Then for the 2^{nd} octant the slope of tangent to the circle is between $\leq |m| \leq 1$ or $-1 \leq m \leq 0$. So with unit increase in x coordinate the y coordinate just decreases approximately by slope at that point. But we have are in advantage of knowing that the slope is not less than 1. So the y coordinate doesn't fall by more than one. So we could safely say if (x_k, y_k) is a intermediate point in the second octant, the next point is either $(x_k + 1, y_k)$ or $(x_k + 1, y_k - 1)$. Notice since the slope is negative the y coordinate falls, contrary to line on the first octant. We are finally down just between two points to figure out the entire circle by induction. If somehow we could figure out which one between these points to chose then our circle would be known completely.

2.4 The Decision Parameter

Let's restate the equation of circle once again this time rather differently. Lets call it a circle function.

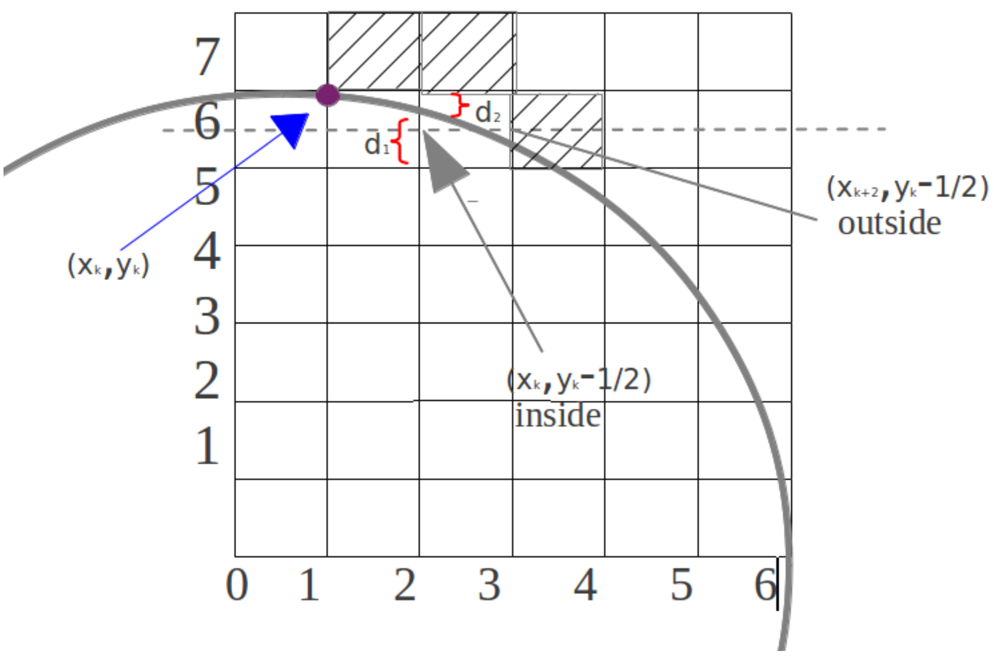
$$f(x, y) = x^2 + y^2 - r^2 \tag{2.6}$$

If any point satisfies the circle function then the point lies exactly on the locus. But what happens if it doesn't satisfy? The resulting quantity will be either positive or negative if it doesn't satisfy. If the result is positive then the point lies on the outside of the circle or else inside (area enclosed by circle) the circle.
Let us now suppose that an intermediate point (x_k, y_k) has been determined. The next point (x_{k+1}, y_{k+1}) from above discussion is either$(x_k + 1, y_k)$ or $(x_k + 1, y_k - 1)$. Lets decide which one between them do we chose. Having determined current point.
If we put the current point on the circle function, it only tells whether the current point is inside or outside the circle, and tells nothing more about next point. One way is to test whether the point $(x_k + 1, y_k)$ lies inside the circle. But the better point to test would be $(x_k + 1, y_k - \frac{1}{2})$, the point midway between the points of contention, because if this point lies outside the circle, then the circle would be closer to the point $(x_k + 1, y_k - 1)$ and if it lies inside, the circle would be closer the point $(x_k + 1, y_k)$ for our idea is to select the closest integer approximation of the actual circle ordinate.
Putting $(x_k + 1, y_k - \frac{1}{2})$ in equation 2.6 we get:

$$P_k = f(x_k + 1, y_k - \frac{1}{2}) = (x_k + 1)^2 + (y_k - \frac{1}{2})^2 - r^2 \tag{2.7}$$

If this comes out to be positive the point midway between the points of contention $(x_k + 1, y_k - \frac{1}{2})$ lies outside the circle suggesting $(x_k + 1, y_k - 1)$ lies closer to the circle or else if it comes negative the point $(x_k + 1, y_k)$ lies closer to the actual circle.

$$y_{k+1} = \begin{cases} y_k & \text{if } P_k \leq 0 \\ y_k - 1 & \text{if } P_k > 0 \end{cases} \tag{2.8}$$



2.5 Recursive Relation Of Decision Parameter

For quite obvious reasons from our discussion of previous chapter, we would like to have a recursive relation of decision parameter P_k. Putting $k+1$ for k in equation 2.7.

$$P_{k+1} = (x_k + 2)^2 + \left(y_{k+1} - \frac{1}{2}\right)^2 - r^2 \tag{2.9}$$

Subtracting equation 2.7 from equation 2.9 we get.

$$P_{k+1} = P_k + 2x_k + (y_{k+1}^2 - y_k^2) - (y_{k+1} - y_k) + 3 \tag{2.10}$$

Which is the decision parameter for a circle.

2.6 Initial Decision Parameter

Now obviously, since we are starting from point on second octant $(0, r)$ is the first point we chose. Therefore $(x_0, y_0) = (0, r)$. Putting $k = 0$ in equation 2.7 we get.

$$\begin{aligned} P_0 &= (x_0 + 1)^2 + (y_0 - \tfrac{1}{2})^2 - r^2 \\ &= (0 + 1)^2 + (r - \tfrac{1}{2})^2 - r^2 \end{aligned}$$

$$P_0 = \frac{5}{4} - r \tag{2.11}$$

This is a tidy initial decision parameter.

2.7 The Breshenham Algorithm

Writing out algorithm for this:

```
Data: Get radius $r$ and center $(h, k)$
$P \leftarrow 5/4 - r$;
$(x, y) \leftarrow (x_0, y_0)$;
while $x \leq y$ do
    Plot$(x + h, y + k)$;
    $t \leftarrow y$;
    if $P \geq 0$ then
        $y \leftarrow y - 1$;
    end
    $P \leftarrow P + 2x + (y^2 - t^2) - (y - t) + 3$;
    $x \leftarrow x + 1$;
end
```

Algorithm 2: Mid Point Circle Algorithm

.

Compare it with the breshenham algorithm for the straight line and get amazed. The algorithm is exactly the same. Only the formula for the decision parameter are different.

For a circle, there is radial symmetry. This symmetry allows us determine the circle completely just by knowing the points of just the first octant. We would implement this process until $x_k \leq y_k$ which indicates the point lies on the second octant. For a point (x, y), its transformation to the rest of the octant also are the closest points of the circle. So, for (x, y) lying on second octant (y, x) lies on first octant. And also for the rest of octant find the transformed point and plot it.

2.8 Implementation in C++

Breshenham Circle Drawing Algorithm adopted in a program in C++.

```cpp
#include<iostream>
#include<cstdio>

using namespace std;

struct CircleParameter { int x,y,r;};

void DrawMidPointCircle(CircleParameter);
void ShowPoint(int,int);

int main()
{

    CircleParameter CP;
    cout<<"Enter the radius"<<endl;
    cin>>CP.r>>CP.x>>CP.y;
    DrawMidPointCircle(CP);
    return 0;
}


void DrawMidPointCircle(CircleParameter CP)
{
    float p=5./4-CP.r;
    int x=0,y=CP.r;
    ShowPoint(x,y); cout<<"  "<<p<<endl;
    do
    {
        int t=y;
        if(p>0) y--;
        p=p+2*x+1+(y*y-t*t)-(y-t); //decision parameter
        x++;
        ShowPoint(x,y); cout<<"  "<<p<<endl;
    }while(y>=x);
}


void ShowPoint(int x,int y)
{
    cout<<"("<<x<<","<<y<<")";
}
```

Chapter 3

The Breshenham Algorithm : Ellipse

3.1 Defintion

An ellipse is the locus of a point which moves such that the ratio of its distance from a fixed point(called the focus) and a straight line (called the directrix) is always constant, the constant is called the eccentricity of the ellipse and is less than one. If the eccentricity is greater than one the locus would be a hyperbola. Ellipse is a closed curved.
Another equivalent definition which also gives us ellipse is that, ellipse is the locus of moving point whose sum of distances from two points called the foci is always constant. Both these definition are just two different demonstration of the same geometrical shape and are interchangeable.

3.2 Equation of Ellipse

We would not go into the mathematical details of the ellipse but rather we will ask a few equations from the mathematician and state the equation of ellipse as they provide us.

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1 \tag{3.1}$$

Above equation can be modified into a different form as:

$$b^2x^2 + a^2y^2 - a^2b^2 = 0$$

If we had to determine the points to plot in a graph or to disply in a vdu we would rather like the equation that gives the value of y coordinate for corresponding value of x coordinates. Making y as a function of x from above relation we get:

$$y = \frac{b}{a}\sqrt{a^2 - x^2} \tag{3.2}$$

To work out this formula we need to do lots of calculation as square of a floating point number, fractions, their product and tedious square root.
There is another equation as well called the polar equation of the ellipse.

$$r = \frac{l}{1 + \epsilon cos(\theta)} \tag{3.3}$$

For reasons mentioned earlier we would like to have a simpler process to find out the points of an ellipse.

3.3 Two Parts on Quardrant

Unlike a circle, the ellipse doesn't posses one symmetry of reflection on the line $y = x$. So as in circle we cant just calculate point on a single octant and reflect on other octants to plot. We certainly would exploit the symmetry along the axes but we have first to determine all point on one of the quardrants so they can be transformed to other quardants. But for our misery the slope of tangent to ellipse on the first quardrant takes all values from zero to infinity. But so far we have used Breshenham algorithm with one commong thing(in line and circle) in mind, the solpe had the property.$0 \leq |m| \leq 1$ or $-1 \leq m \leq 0$. The ellipse doesn't posses symmetry in the octant and doesn't have the required property, so we are in deep deep trouble. What do we do now??
Don't panic just yet. Lets go for yet another trickery again.

Keeping in mind that we will exploit the property $0 \leq |m| \leq 1$ or $-1 \leq m \leq 0$. Lets divide the ellipse on the first quardrant into two parts, one part of which has the slope of tangent exaclty between $0 \leq |m| \leq 1$, Starting from $(0, b)$ where the slope is 0, the slope decreases and reaches -1 at ceratin point. Lets find that point.
To find the point lets find the slope of tangent to the ellipse. Differentiating equation 3.1 we get

$$\frac{dy}{dx} = -\frac{b^2x}{a^2y}$$

The for the slope to be greater than -1 we must have.

$$\frac{dy}{dx} \geq -1$$
$$\Rightarrow -\frac{b^2x}{a^2y} \geq -1$$
$$\Rightarrow b^2x \leq a^2y \tag{3.4}$$

equation 3.4 must be satisfied for the slope at that point to be $0 \leq |m| \leq 1$

3.4 The Two Points

Assuming the ellipse is centered at origin and given semi major axis a and semi minor axis b lets find points of the lying in the first part of the first quardrant,.

We begin with the known point on the second octant $(0, b)$. Then for the first part in the first quardrant the slope of tangent to the ellipse is between $0 \leq |m| \leq 1$ or $-1 \leq m \leq 0$. So with unit increase in x coordinate the y coordinate just decreases approximately by slope at that point. But we have are in advantage of knowing that absolute value of the slope is not less than 1. So the y coordinate doesn't fall by more than one. So we could safely say if (x_k, y_k) is a intermediate point in the first part of first quardrant, the next point is either $(x_k + 1, y_k)$ or $(x_k + 1, y_k - 1)$. Notice since the slope is negative the y coordinate falls, contrary to line on the first octant but similar to the circle. We are finally down just between two points to figure out the entire ellipse by induction.

If somehow we could figure out which one between these points to chose then our ellipse would be known completely.

3.5 The Decision Parameter

Lets again mention the equation of ellipse but this time rather differently. Lets call it an ellipse function.

$$f(x,y) = bx^2 + ay^2 - a^2b^2 \tag{3.5}$$

Any point satisfying this equation lies on the ellipse and any other point not satisfying lies either inside the ellipse or outside teh ellipse as an ellipse is a geometrically closed curve.
When a point substituted in this function gives a negative value the point lies inside the ellipse and when the point gives positive value then the point lies outside the ellipse.
Let us now suppose that an intermediate point (x_k, y_k) has been determined. The next point (x_{k+1}, y_{k+1}) from above discussion is either(x_k+1, y_k) or (x_k+1, y_k-1). Lets decide which one between them do we choose having determined current point.

Similar to the discussion made for circle, If we put the current point on the ellipse function, it only tells whether the current point is inside or outside the ellipse, and tells nothing more about next point. One way is to test whether the point (x_k+1, y_k) lies inside the ellipse. But the better point to test would be $(x_k+1, y_k-\frac{1}{2})$, the point midway between the points of contention, because if this point lies outside the ellipse, then the ellipse would be closer to the point (x_k+1, y_k-1) and if it lies inside, the ellipse would be closer the point (x_k+1, y_k) for our idea is to select the closest integer approximation of the actual ellipse ordinate.
Putting $(x_k+1, y_k-\frac{1}{2})$ in equation 3.5 we get:

$$P_k = f(x_k+1, y_k-\frac{1}{2}) = b^2(x_k+1)^2 + a^2(y_k-\frac{1}{2})^2 - a^2b^2 \tag{3.6}$$

If this comes out to be positive the point midway between the points of contention $(x_k+1, y_k-\frac{1}{2})$ lies outside the ellipse suggesting (x_k+1, y_k-1) lies closer to the ellipse or else if it comes negative the point (x_k+1, y_k) lies closer to the actual ellipse

$$y_{k+1} = \begin{cases} y_k & \text{if } P_k \le 0 \\ y_k - 1 & \text{if } P_k > 0 \end{cases} \tag{3.7}$$

3.6 Recursive Relation Of Decision Parameter

For quite obvious reasons from our discussion of previous chapter, we would like to have a recursive relation of decision parameter P_k. Putting $k+1$ for k in equation 3.6.

$$P_{k+1} = a^2(x_k+2)^2 + b^2(y_{k+1}-\frac{1}{2})^2 - a^2b^2 \tag{3.8}$$

Subtracting equation 3.6 from equation 3.8 we get.

$$P_{k+1} = P_k + 2b^2x_k + a^2(y_{k+1}+y_k-1)(y_{k+1}-y_k) + b^2 \tag{3.9}$$

Which is the decision parameter for an ellipse.

3.7 Initial Decision Parameter

Now obviously, since we are starting from point $(0, b)$ is the first point we chose. Therefore $(x_0, y_0) = (0, b)$. Putting $k = 0$ in equation 3.6 we get.

$$\begin{aligned} P_0 &= b^2(x_0+1)^2 + a^2(y_0 - \tfrac{1}{2})^2 - a^2b^2 \\ &= b^2(0+1)^2 + a^2(b - \tfrac{1}{2})^2 - a^2b^2 \end{aligned}$$

$$P_0 = b^2 - a^2b + \frac{a^2}{4} \tag{3.10}$$

This is a again neat and tidy initial decision parameter.

3.8 The Breshenham Algorithm

Writing out algorithm for this:

```
Data: Get semi major axis $a$ and semi minor axis and $b$ center $(h, k)$
$P \leftarrow b^2 - a^2b + a^2/4$;
$(x, y) \leftarrow (0, b)$;
while $b^2x \leq a^2y$ do
    Plot$(x + h, y + k)$;
    $t \leftarrow y$;
    if $P \geq 0$ then
        $y \leftarrow y - 1$;
    end
    $P \leftarrow P + 2b^2x + a^2(y + t - 1)(y - t) + b^2$;
    $x \leftarrow x + 1$;
end
```

Algorithm 3: Mid Point Ellipse Algorithm

.

Compare it with the breshenham algorithm for the straight line and circle and get even more amazed. The algorithm is exactly the same. Only the formula for the decision parameter are different.

3.9 All the points

Equipped with above arms we can calculate the points of ellipse on first part of first quardrant. We would like then to find the points on the second quardrant. The idea is, get the points of first part first. Then swap the semi major and semi minor axis. Find the points for the swapped ellipse. Then swap back the calculated points. With this simple artifice we can find points on the second art as well.
Now we have completed the ellipse in the first quardrant. To find points on other quardrant explioit the axial symmetry and transform the points to rest of the three quardrants as well. Our ellipse is then ready. Nice and tidy.

3.10 Implementaion On C++

```
#include<iostream>
#include<cstdio>
using namespace std;

struct Ellipse {int h,k,a,b;};

istream& operator>>(istream&,Ellipse&);

void ShowPoint(Ellipse,int,int);
void BreshenhamEllipse(Ellipse);
void PartEllipse(Ellipse,int=0);

template <typename DataType>
void Swap(DataType&,DataType&);

int main()
{
    Ellipse E;
    cin>>E;
    BreshenhamEllipse(E);
    return 0;
}

void BreshenhamEllipse(Ellipse E)
{
    PartEllipse(E);
    Swap(E.a,E.b);
    PartEllipse(E,1);
}
void PartEllipse(Ellipse E,int SwapIndex)
{
    int B=E.b*E.b,A=E.a*E.a; // A=a^2 and B=b^2 squares of
       semi major and minor axes resp.
    float p=B-(A*E.b)+A/4.;  //initial decision parameter
    int x=0,y=E.b;  //starting point to plot.
    do
    {
        if(SwapIndex==0) ShowPoint(E,x,y); else ShowPoint(E,y
           ,x);cout<<endl;
        int t=y; //preserve y_k to calculate p_{k+1}
        if(p>0) y--; //decide between updating y coordinate
        p=p+2*B*x+B+A*(y+t-1)*(y-t);//recursive formula for
           decision parameter
        x++; //always increase x coordinate by just 1;
    }
    while(B*x<=A*y); //loop while the points are within the
       limit of slope.
}
```

```
void ShowPoint(Ellipse E,int x,int y)
{
  cout<<"("<<E.h+x<<","<<E.k+y<<")";
}


istream& operator>>(istream&ip,Ellipse&E)
{
    cout<<"Enter the centre coordinates     - : "; cin>>E.h>>
       E.k;
    cout<<"Enter semi-major and-minor axes  - : "; cin>>E.a>>
       E.b;
    return ip;
}

template <typename DataType>
void Swap(DataType& p1,DataType&p2)
{
    DataType p;
    p=p1;
    p1=p2;
    p2=p;
}
```