BRESENHAM'S LINES ALGORITHM

Lokesh Madan[1], Kislay Anand[2]and Bharat Bhushan[3]

[1]*Department of Computer Science, Dronacharya College of Engineering, Gurgaon, India*

[2]*Department of Computer Science, Dronacharya College of Engineering, Gurgaon, India*

[3]*Department of Computer Science, Dronacharya College of Engineering, Gurgaon, India*

ABSTRACT

Line drawing on discrete graphics devices such as raster video displays, plotters, and image printers is one of the fundamental operations in computer graphics. Real-time interactive applications or high speed image output (such as on a Postscript laser printer) may require line drawing speeds in the millions of pixels per second. Such demands, which are ever increasing, push the efficiency of line generation.
For nearly thirty years Bresenham's algorithm has been the standard which subsequent efforts inline drawing have sought to surpass.
The basic "line drawing" algorithm used in computer graphics is Bresenham's Algorithm.
This paper describes a hybrid method which uses structural properties of raster lines, such as runs, to improve the efficiency of multi-point line generation. A quadruple-step algorithm is developed which requires fewer decision tests than other multi-point algorithms, while retaining the multi-point's advantage in pixel
output efficiency, particularly when implemented in hardware.
Abstract. Bresenham's algorithm minimizes error in drawing lines on integer grid points; leap year calculations, surprisingly, are a generalization.
KeyWords: Computer graphics, Multimedia, Algorithm,Flash movie, ActionScript , Rasterization, ScanConversion

INTRODUCTION

Rasterization: Process of determining which pixels
provide the best approximation to a desired line on the screen.

Scan Conversion: Combination of rasterization and generating the picture in scan line.

Rasterization (Scan Conversion)

- Convert high-level geometry description to pixel colors in the frame buffer
- Example: given vertex x,y coordinates determine pixel colors to draw line

- Two ways to create an image:
- Scan existing photograph
- Procedurally compute values (rendering)

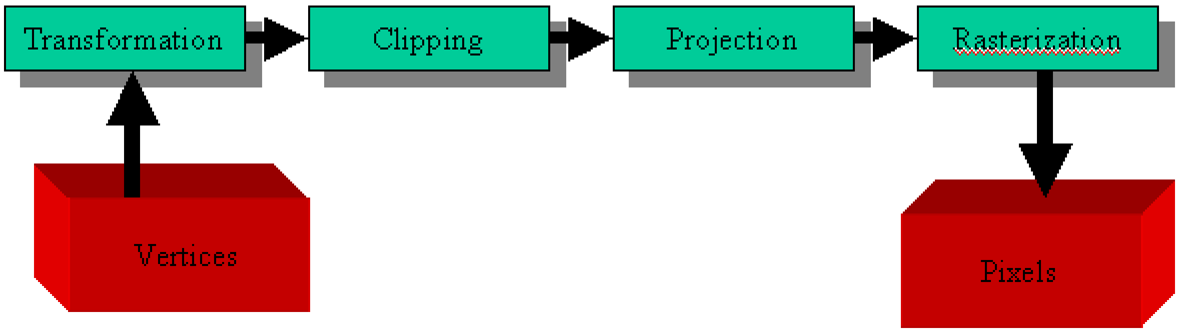
Rasterization

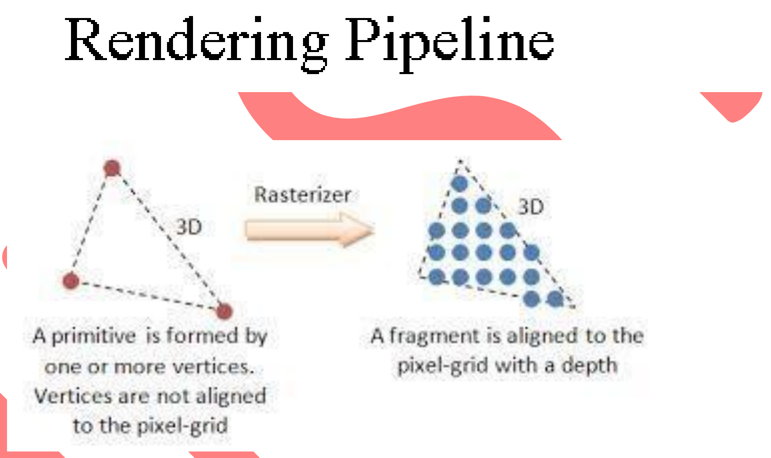
- A fundamental computer graphics function
- Determine the pixels' colors, illuminations, textures, etc.
- Implemented by graphics hardware
- Rasterization algorithms
- Lines
- Circles
- Triangles
- Polygons

Rasterization Operations

- Drawing lines on the screen
- Manipulating pixel maps (pixmaps): copying, Compositing images, defining and modifying regions
- Drawing and filling polygons
- Previously glBegin(GL_POLYGON), etc
- Aliasing and antialiasing methods

BLOCK DIAGRAM OF RASTERIZATION




Rendering Pipeline
3D
Rasterizer
3D
A primitive is formed by one or more vertices. Vertices are not aligned to the pixel-grid
A fragment is aligned to the pixel-grid with a depth

No groups rasterized Group A rasterized

Rasterizing causes layers to be rendered in the stacking order shown in the Layers List.

In 1965, Bresenham introduced what has become the standard measure for line drawing algorithms . This is an algorithm that determines which points in an *n*- dimensional raster should be plotted in order to form a close approximation to a straight line between two given points. It is commonly used to draw lines on a computer screen, as it uses only integer addition, subtraction and bit shifting, all of which are very cheap operations in standard computer architectures. It is one of the earliest algorithms developed in the field of computer graphics. A minor extension to the original algorithm also deals with drawing circles.

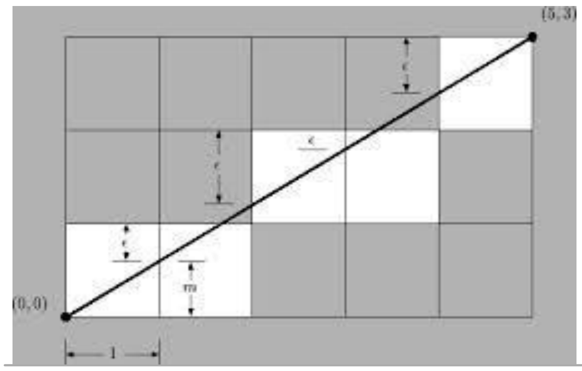
Bresenham’s algorithm [Bresenham 1965]is the classic technique for plotting lines onbitmaps: it approximates linear segments defined by rational coefficients using only integral points. The algorithm calculates a finite set of points on the integer lattice with minimum total vertical distance from the original line segment. The essential design features of the algorithm are minimization of error (it converts a line segment from a continuous domain to one that is discrete with as little error as possible), speed (it uses only integer arithmetic), and parsimony (as few pixels as possible are used to ensure adjacency). Beginning at one of the end points, Bresenham’s algorithm generates the line by deciding from its current position which of the neighboring pixels is closest to the true line and then "moving" to that new position. By dividing line space into eight octants, only two of the eight adjacent pixels need to be examined for a given line. The algorithm decides between these two pixels by sign-testing a "decision variable" (also called a "discriminator") that indicates which of the pixels is closer to the line. After each test, the decision variable is updated according to a simple recurrence relation. This process of testing and updating a decision variable to find the points of a line is commonly called "incremental" line drawing. In addition, the algorithm is said to generate Òbest-fitÓ lines, meaning the discrete representations that are closest to the actual lines. Using only simple integer arithmetic (addition, subtraction, shifting) and sign testing, Bresenham’s algorithm is both simple and efficient.

While algorithms such as Wu's algorithm are also frequently used in modern computer graphics because they can support antialiasing, the speed and simplicity of Bresenham's line algorithm means that it is still important. The algorithm is used in hardware such as plotters and in the graphics chips of modern graphics cards. It can also be found in many software graphics libraries. Because the algorithm is very simple, it is often implemented in either the firmware or the graphics hardware of modern graphics cards.

BRESENHAM'S ALGORITHM

A picture can be described in several ways. Assuming wehave a raster display, a picture is completely specified by the set of intensities for the pixel positions in the display. The scene is then displayed either by loading the pixel arrays into the frame buffer or by scan converting the basic geometric-structure specifications into pixel patterns.
Generally, graphic programming creates the basic of geometric structure, referred to as output primitives. Output primitive is displayed. Output primitive is specific coordinate data and other information that is input how the object is to be displayed. Simple output primitives are straight line, and it is the simplest geometric components of pictures. Additional output primitives that can be used to construct a picture include lines, circles and other conic sections, quadric surfaces, line, curves and surfaces, polygon color areas, and character strings.
Line drawing is accomplished by calculating intermediate positions along the line path between two specified endpoint positions. Some pixels are created in the position between the endpoints .
Bresenham's line algorithm is a line algorithm which calculates either X or Y coordinate, and using only incremental integer calculations to produce lines, circles and other curves.

Detailed Explanation:
Consider a line with initial point (x1, y1) and terminal point (x2, y2) in device space. If dx = x2 −x1 and dy = y2 − y1, we define the *driving axis* (*DA*) to be the x-axis if |dx| >=|dy|, and the y-axis if |dy| > |dx|. The *DA* is used as the "axis of control" for the algorithm and is the axis of maximum movement. Within the main loop of the algorithm, the coordinate corresponding to the *DA* is incremented by one unit. The coordinate corresponding to the other axis (usually denoted the *passive axis* or *PA*) is only incremented as needed.
The best way to describe Bresenham's algorithm is to work through an example.
Consider the following example, in which we wish to draw a line from (0, 0) to (5, 3) in device space.



Bresenham’s algorithm begins with the point (0, 0) and “illuminates” that pixel. Since x is the *DA* in this example, it then increments the x coordinate by one. Rather than keeping track of the y coordinate (which increases by m = dy/dx, each time the x increases by one), the algorithm keeps an error bound e at each stage, which represents the negative of the distance from the point where the line exits the pixel to the top edge of the pixel (see the figure). This value I first set to m − 1, and is incremented by m each time the x coordinate is incremented by one. If e becomes greater than zero, we know that the line has moved upwards one pixel, and that we must increment our y coordinate and readjust the error to represent the distance from the top of the new pixel – which is done by subtracting one from e.
The reader can examine the above illustration and the following table to see the complete operation of the algorithm on this example.

(x,y)	E	Description
(0, 0) (1, 0)	-0.4 0.2	illuminate pixel (0, 0) increment _ by 0.6 increment x by 1
(1,0) (1,1) (2,1)	0.2 -0.8 -0.2	illuminate pixel (1, 0) since e > 0 increment y by 1 decrement e by 1 increment e by 0.6

		increment x by 1
(2,1) (3,1)	-0.2 0.4	illuminate pixel (2, 1) increment e by 0.6 increment x by 1
(3,1) (3,2) (4,2)	0.4 -0.6 0.0	illuminate pixel (3, 1) since e > 0 increment y by 1 decrement e by 1 increment e by 0.6 increment x by 1
(4,2)	0.0	Illuminate pixel(4,2)

Assuming that the *DA* is the x-axis, an algorithmic description of Bresenham's algorithm is as follows:

Bresenham's Algorithm

The points (x1, y1) and (x2, y2) are assumed
not equal and integer valued.
e is assumed to be real.
Let dx = x2 − x1
Let dy = y2 − y1
Let m =dy/dx

```
Let j = y1
Let e = m − 1

for i = x1 to x2 − 1
illuminate (i, j)
if (e>= 0)
j + = 1
e− = 1.0


end if
i + = 1
e + = m
next i
finish
```

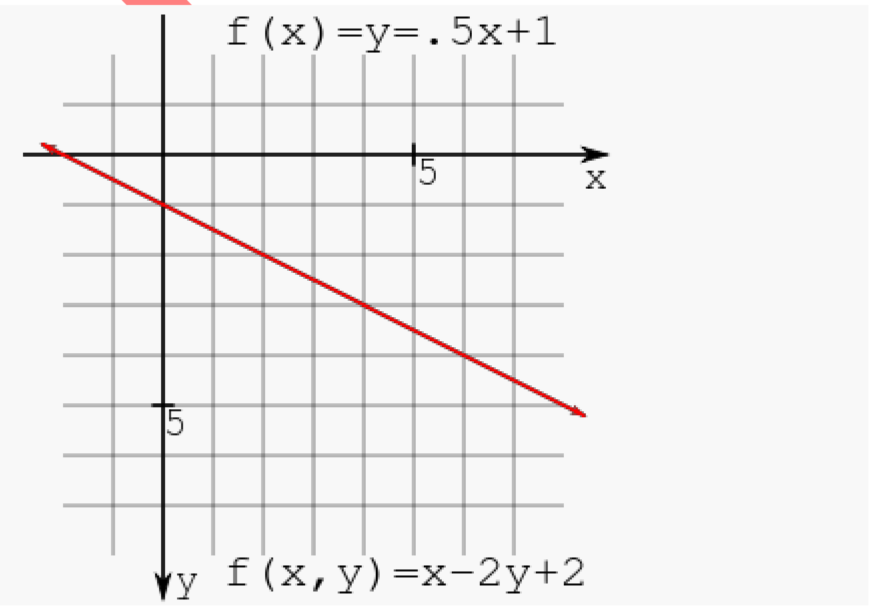
Note:

All algorithms presented in these notes assume that _x and _y are positive. If this is not the case, the algorithm is essentially the same except for the following:

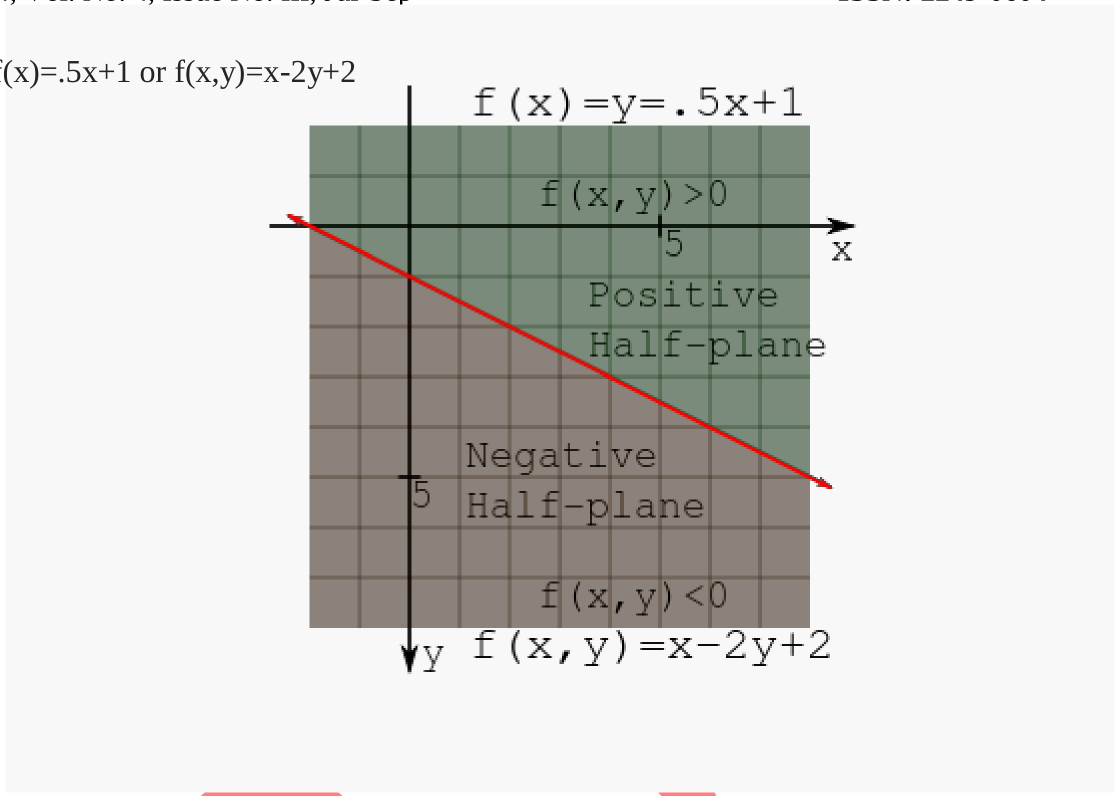
• e is calculated using |dy/dx |.

• x and y are decremented (instead of incremented) by one if the sign of dx or dy is less than zero, respectively.

Line Equations:



y=f(x)=.5x+1 or f(x,y)=x-2y+2



Positive and negative half-planes

The slope-intercept form of a line is written as

$$y = f(x) = mx + b$$

where m is the slope and b is the y-intercept. This is a function of only x and it would be useful to make this equation written as a function of both x and y. Using algebraic manipulation and recognition that the slope is the "rise over run" or $\Delta y/\Delta x$ then

$$\begin{aligned} y &= mx + b \\ y &= \frac{(\Delta y)}{(\Delta x)}x + b \\ (\Delta x)y &= (\Delta y)x + (\Delta x)b \\ 0 &= (\Delta y)x - (\Delta x)y + (\Delta x)b \end{aligned}$$

Letting this last equation be a function of x and y then it can be written as

$$f(x,y) = 0 = Ax + By + C$$

where the constants are

- $A = \Delta y$
- $B = -\Delta x$
- $C = (\Delta x)b$
-

The line is then defined for some constants A, B, and C and anywhere $f(x,y) = 0$. For any (x,y) not on the line then $f(x,y) \neq 0$. It should be noted that everything about this form involves only integers if x and y are integers since the constants are necessarily integers doesn't seem like it?.

As an example, the line $y = \frac{1}{2}x + 1$ then this could be written as $f(x,y) = x - 2y + 2$. The point (2,2) is on the line

$$f(2,2) = x - 2y + 2 = (2) - 2(2) + 2 = 2 - 4 + 2 = 0$$

and the point (2,3) is not on the line and

$$f(2,3) = (2) - 2(3) + 2 = 2 - 6 + 2 = -2$$

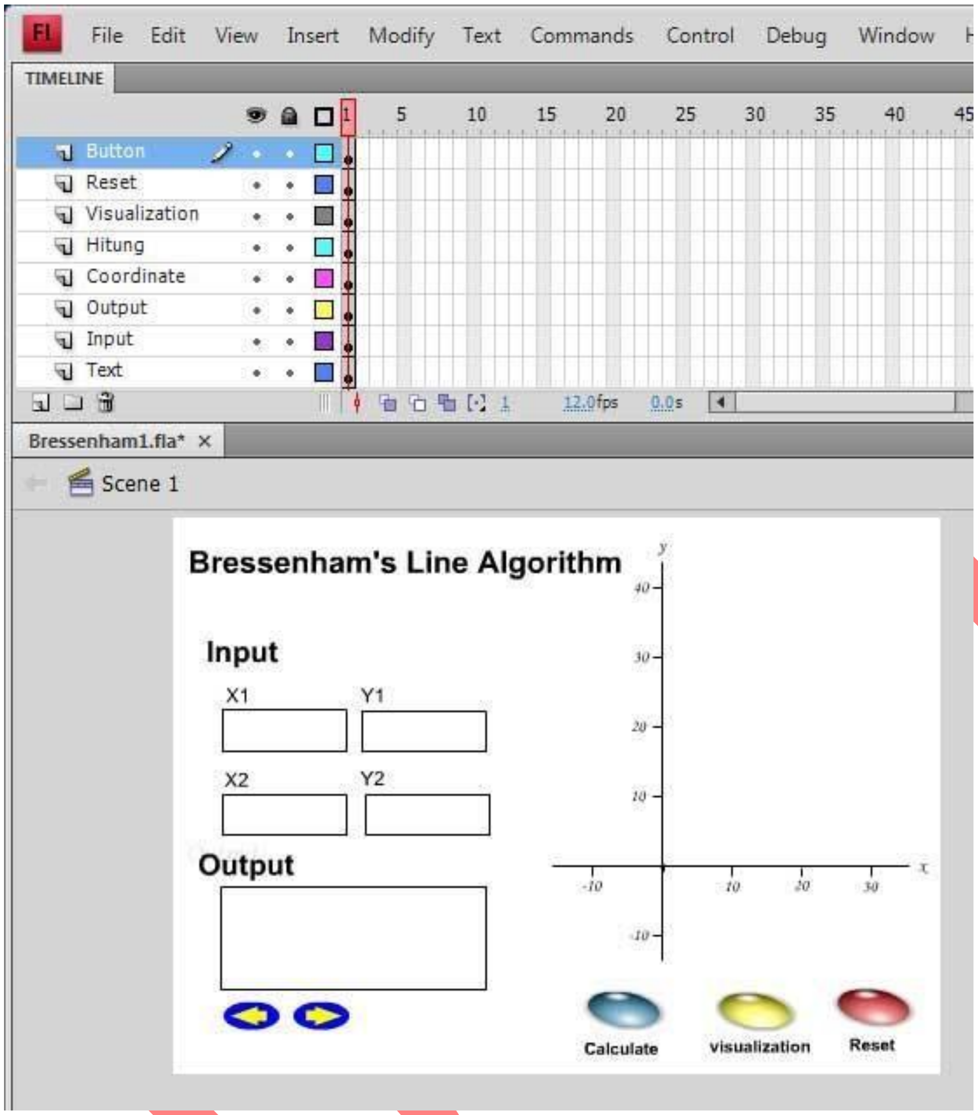
neither is the point (2,1)

$$f(2,1) = (2) - 2(1) + 2 = 2 - 2 + 2 = 2$$

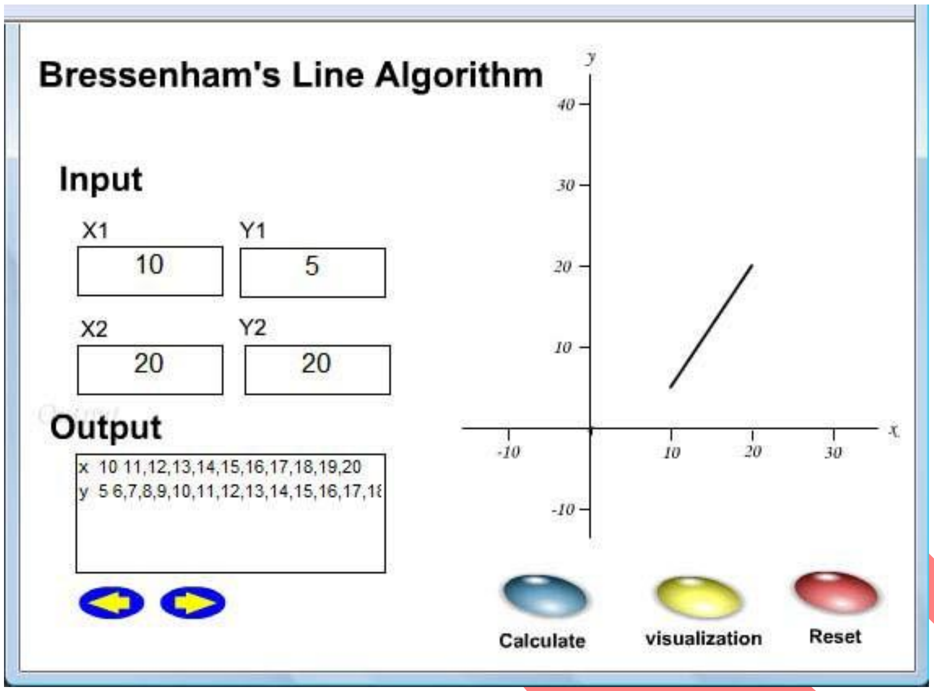
Notice that the points (2,1) and (2,3) are on opposite sides of the line and f(x,y) evaluates to positive or negative. A line splits a plane into halves and the half- plane that has a negative f(x,y) can be called the negative half-plane, and the other half can called the positive half-plane. This observation is very important in the remainder of the derivation.

- Flash movie is provided three action button to calculate, to visualize, and to reset as we can see in Figure 5 presents the stage layout in

Flash.



Stage layout in flash document



Bressenham's Line Algorithm

Input

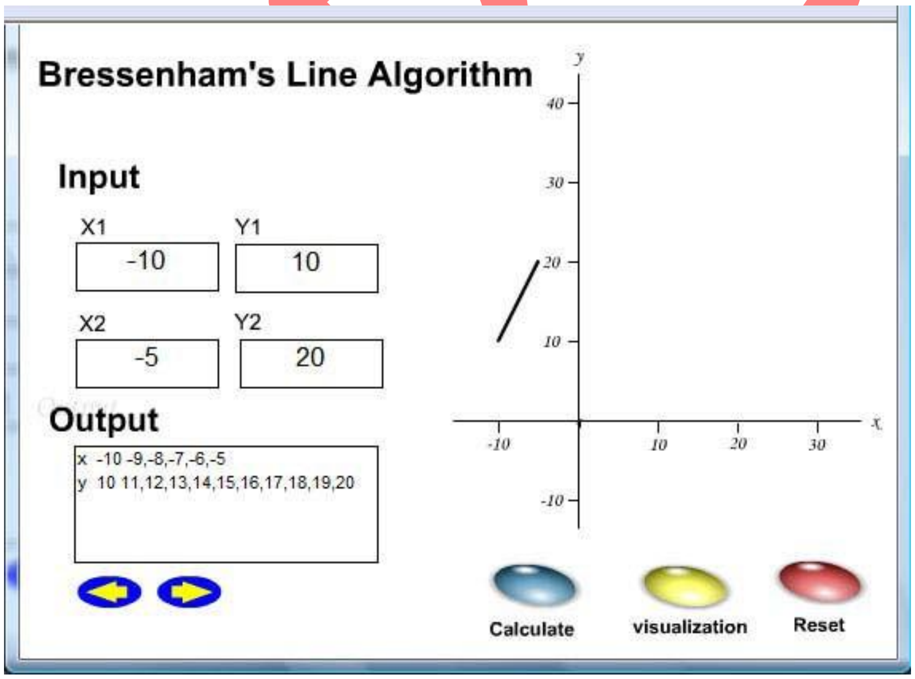
X1: 10 Y1: 5

X2: 20 Y2: 20

Output

```
x 10 11,12,13,14,15,16,17,18,19,20
y 5 6,7,8,9,10,11,12,13,14,15,16,17,18
```

Flash movie visualizes Bresenham's Line Algorithm producing line in the first quadrant



Bressenham's Line Algorithm

Input

X1: -10 Y1: 10

X2: -5 Y2: 20

Output

```
x -10 -9,-8,-7,-6,-5
y 10 11,12,13,14,15,16,17,18,19,20
```

Flash movie visualizes Bresenham's Line Algorithm producing line in the fourth quadrant.

FUTURE SCOPE

The Bresenham's line drawing algorithm is an efficient way of drawing straight lines. The lines can be drawn even faster than above examples by using techniques such as direct screen memory access instead of BIOS ROM function calls and by optimized assembly language programs that keep variables in registers instead of memory.

CONCLUSIONS

Bresenham's Algorithm is a fundamental algorithm in computer graphics. Its basic use it to draw lines on raster graphics devices, however it is useful as a driving engine for many other graphics routines. This algorithm was developed to draw lines on digital plotters, but has found wide-spread usage in computer graphics . The algorithm is fast – it can be implemented with integer calculations only – and very simple to describe.

Through this paper the Bresenham's Line Algorithm visualization has been presented. Adobe Flash is a timeline-based, authoring and object-oriented programming tools can be used to develop a scientific visualization.

REFERENCES

1. Sfenrianto, "A Model of Adaptive E-Learning System Based on Student's Motivation". ICCIT-09 proceedings, pp. 133-138 [*International Conference on Creative Communication and Innovative Technology*, Indonesia, 2009].

2 G. Bhatnager, S. Metha, and S. Mitra, *Introduction to Multimedia Systems*. London: Academic Press, 2001.

3 P. K. Anleigh and K. Thakar, *Multimedia Systems Design*. Upper Saddle River: Prentice Hall, 1997.

4. Jr. Hill, *Computer Graphics Using Open GL*. Upper Saddle River: Prentice Hall, 1990.

5. J. Voley et al, *Computer Graphics Principles and Practices*. New York: Addison Wesley, 1996.

6. https://en.wikipedia.org/wiki/Bresenham%27s_line_algorithm