(18CS62)

Module-1

Computer graphics and OpenGL

Basics of Computer Graphics

What is Computer Graphics?

* Computer graphics is an art of drawing pictures, lines, charts etc. using computers with the help of programming.
* Computer graphics image is made up of number of pixels. pixels is the smallest addressable graphical unit represented on Computer screen

Applications of Computer Graphics:

a) Graphs and Charts:-

Data plotting with dramatic effect, commonly used to summarize financial, statistical, mathematical, scientific, engineering, and economic data for research reports, managerial summaries, consumer information bulletins and other types of publications

b) Computer-aided design:-

→ A major use of Computer graphics is in design processes - particularly for engineering and architectural systems

→ CAD, Computer-aided design or CADD, Computer drafting and design methods are now routinely used in automobiles, aircrafts, Computers:

c) Virtual - Reality Environments

Animations in virtual-reality environments are often used to train heavy-equipment operators or to analyze the effectiveness of various cabin configurations and control placements

d) Data visualizations :-

→ There are many different kinds of data sets and effective visualization schemes depend on the characteristics of the data.

→ A collection of data can contain scalar values, vectors or higher-order tensors

e) Education and Training :-

→ Computer generated models of physical, financial, political, social, economic, and other systems are often used as educational aids.

Eg :- Color-coded diagram in operation of a nuclear reactor, aircraft, naval

f) Computer art :-

→ Graphics are used in Fine Art, Commercial Art applications and Mathematical Art.

→ The artist uses a combination of 3D modeling packages, texture mapping, drawing programs and CAD software

→ pen plotters create Automatic Art.

G) Entertainment

→ Graphics are commonly used in making motion pictures, music videos & television shows

→ Graphics objects can be combined with live actions

Eg :- Avtar Movie

h) Image Processing:

→ Image Processing applies techniques to modify or interpret existing pictures. (photographs, TV Scans)

→ To apply image processing methods, the image must be digitized first.

→ used in CT, PET, CAT (Computer aixal tomography)

(Computed X-ray tomography) (position emission tomography)

I) Graphical User Interfaces

→ It is common now for applications software to provide graphical user interface (GUI)

→ A major Component of GUI is a window manager. allows a user to display multiple, rectangular screen areas called display windows.

Video Display Devices: Random Scan and Raster Scan displays, graphics software:

Video Display Devices:

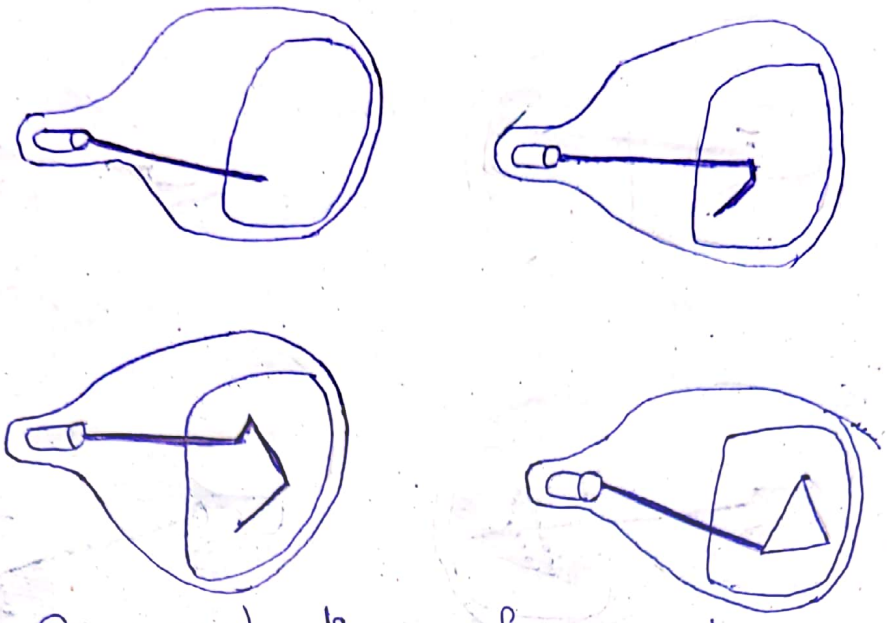
→ The primary output device in a graphics system is a video monitor.
→ Historically, the operation of most video monitors was based on the standard cathode ray tube (CRT) design, but several other technologies exist.

i) Random-Scan Displays: (Vector System Architecture)

Line drawing and stroke drawing in a random order

* Vector system consists of: display processor (controller), display buffer memory, CRT.

→ When operated as a random-scan display unit, a CRT has the electron beam directed only to the those parts of the screen where a picture is to be displayed.
→ pictures are generated as line drawings with the electron beam tracing out the component lines one after the other.
→ For this reason, random-scan monitors are also referred to as vector displays

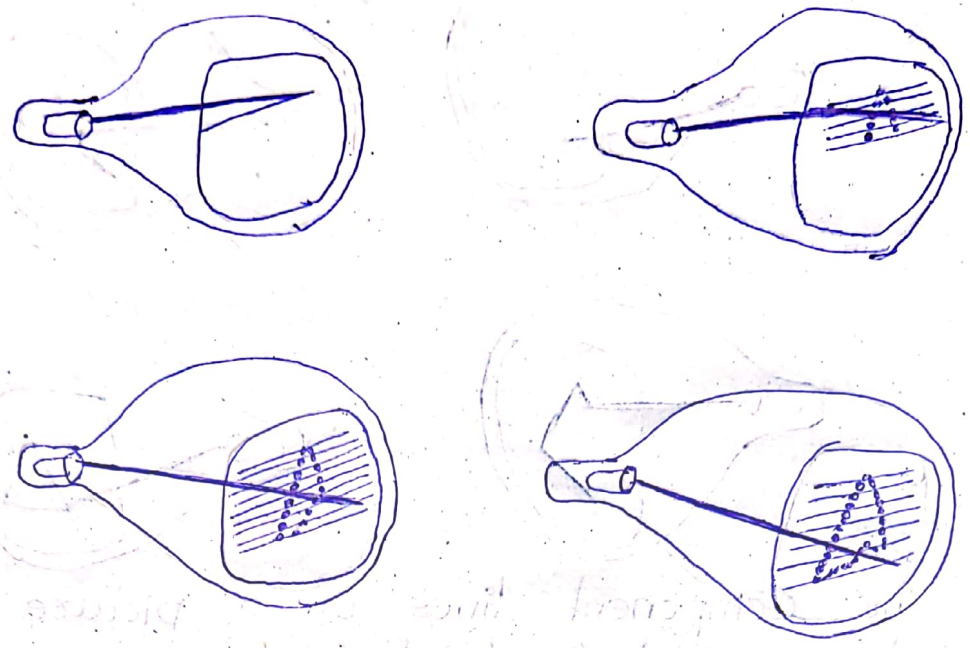


→ The component lines of a picture can be drawn and refreshed by a random-scan system in any specified order.
→ A pen plotter operates in a similar way and is an example of a random-scan hard copy device.
→ Refresh rate on a random-scan system depends on the numbers of lines to be displayed on that system.

---

ii) Raster-Scan Displays:

→ The electron beam is swept across the screen one row at a time from top to bottom.
→ As it moves across each row, the beam intensity is turned on and off to create a pattern of illuminated spots.
→ This scanning process is called refreshing. each complete scanning of a screen is normally called a frame.
→ Picture definition is stored in a memory area called the frame buffer.



Case-1:- In Case of black and white systems.
→ On black & white systems, the frame buffer storing the values is called a bitmap
→ each entry in the bitmap is a 1-bit data which determine the one (1) and off (0) of the intensity of the pixel.

- Case-2:- In Case of color Systems:
→ On color system, the frame buffer storing the values of the pixels is called a pixmap.
→ each entry in the pixmap occupies a number of bit to represent the color of the pixel.

Basic definitions:-
1) Raster:- A rectangular array of points or dots.
2) pixel:- One dot element of the raster, defined as smallest addressable area on screen.

3) Scan line: A row of pixels.

4) Resolution: # of pixel positions that can be plotted.

5) Aspect Ratio: # of horizontal points to vertical points.

6) Depth: # of bits per pixel in a frame buffer.

7) Bitmap: a frame buffer with multiple bit per pixel.

8) Pixmap: a frame buffer with multiple bits per pixel.

9) Horizontal retrace: return to the left of the screen, after refreshing each scan line.

10) Vertical retrace: electron beam returns to the top left corner of the screen to begin the next frame.

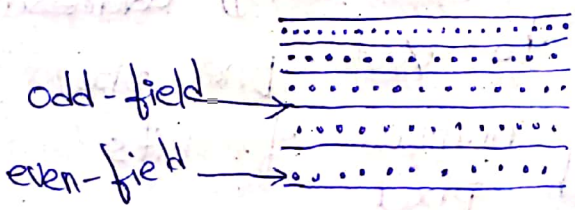
Refresh rates are described in units of cycles per seconds, or Hertz (Hz).

Types of refresh:

1) Interlaced (mostly for TV for reduced flickering effect - National Television System Committee NTSC) each frame is displayed in two passes

i) First pass: odd-field: odd-numbered scan lines

ii) Second pass: even-field: even-numbered scan lines.



2) Non-interlaced
(mostly for monitor)
→ refresh rate: eg: 60Hz or more

Difference between Raster scan System and Random scan System:

S.No	Random Scan	Raster Scan
1.	The resolution of random scan is higher than raster scan.	While the resolution of raster scan is lower than random scan.
2.	It is costlier than raster scan	while the cost of raster scan is lesser than random scan.
3.	Any alteration is easy in comparison of raster scan.	Any alteration is not so easy.
4.	Interweaving is not used.	Interweaving is used.
5.	It is suitable for applications requiring polygon drawings	It is suitable for creating realistic scenes.

Frame Buffer:
A frame buffer may be thought of as Computer memory organized as a two-dimensional array with each (x, y) addressable location corresponding to each one pixel
→ Bit planes is the number of bits corresponding to each pixel

→ A typical frame buffer resolution might be

640 X 480 X 8
1280 X 1024 X 8
1280 X 1024 X 24

Graphics Software:-

* There are two broad classifications for computer-graphics software

• Special-purpose packages: Special-purpose packages are designed for nonprogrammers.
eg:- generate pictures, graphs, charts etc.

• General programming packages: It provides a library of graphics functions that can be used in a programming language such as C, C++, Java
eg:- GL(Graphics library), Java 2D and Java 3D.

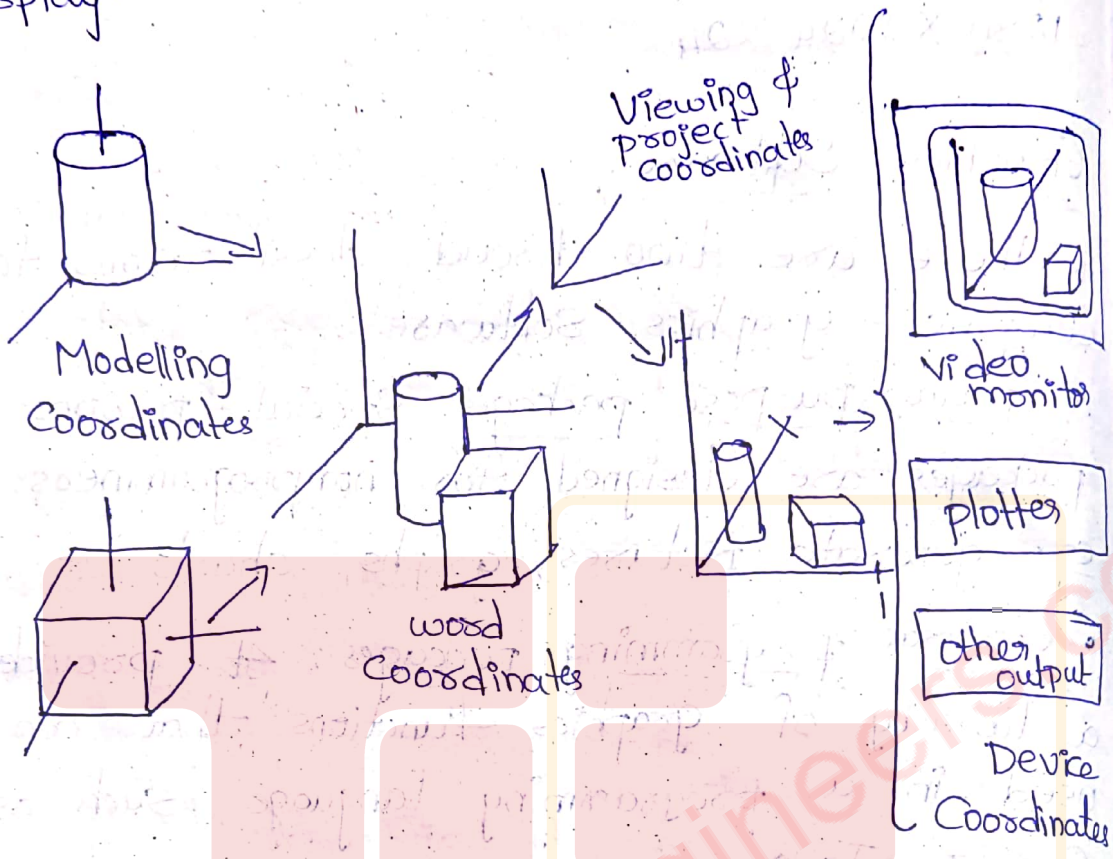
Coordinate Representations:-

→ To generate a picture using a programming package we first need to give the geometric descriptions of the objects that are to be displayed known as co-ordinates.

→ If co-ordinate values for a picture are given in some other reference frame, they must be converted to Cartesian Coordinates.

→ Several different Cartesian references frames are used in the process of constructing and displaying.

eg:- Fig briefly illustrates the sequence of coordinate transformations from modeling coordinates to device coordinates for a display.



$(x_{mc}, y_{mc}, z_{mc}) \rightarrow (x_{wc}, y_{wc}, z_{wc}) \rightarrow (x_{vc}, y_{vc}, z_{vc}) \rightarrow (x_{pc}, y_{pc}, z_{pc}) \rightarrow (x_{nc}, y_{nc}, z_{nc}) \rightarrow (x_{dc}, y_{dc})$

Graphics functions:

* Output primitives: plot character strings, points, straight lines, curved lines, polygons, etc.
* Attributes: set properties of output primitives such as color specifications, line styles, fill patterns, etc.
* Geometric transformations: change size, position, orientation of an object.

* Viewing transformations: Select a view of the scene, type of projection to be used, location on video monitor where the view is to be displayed.
* Input Functions: Control and process the data flow from interactive devices
* Control Operations: house-keeping tasks such as clearing a screen display area

Graphics Software

* Development of the OpenGL API
* OpenGL Architecture
  → OpenGL as a state machine
* Functions
  → Types
  → formats

→ In 1984, Graphical kernel System (GKS) was adopted as the first graphics software standard by the International Standards Organization (ISO)
→ The graphics workstations from silicon Graphics, Inc. (SGI), came with a set of routines called as GL (Graphics library)

Introduction to OpenGL:

What is OpenGL?

→ Is a Cross-language, Cross-platform API for rendering 2D and 3D vector graphics.

→ Development started by SGI (Silicon Graphics INC) at 1991 (developers groups name was ARB (IBM, Microsoft, Nvidia,...)
→ Constants & data type names begin with "GL".

OpenGL Libraries:

* OpenGL core library
  → OpenGL32 on Windows
  → GL on most unix/Linux systems
* OpenGL utility library (GLU)
→ provides functionality in OpenGL core but avoids having to rewrite code.
* Links with window system
→ GLX for X window system
→ WGL for windows
→ AGL for Macintosh
* GLUT (OpenGL utility library)
→ Provides functionality common to all window systems.
  → ~~Open~~

Header Files:

→ In all graphics programs, we will need to include the header file for the OpenGL core library.
→ in windows,

```
#include <windows.h>
```

```
#include <GL/gl.h>
#include <GL/glu.h>
```

Can be replaced by GLUT header file

```
#include <GL/glut.h>
```

→ Apple OS X systems,

```
#include <GLUT/glut.h>
```

OpenGL Function Format

```
glVertex3f(x, y, z)
```

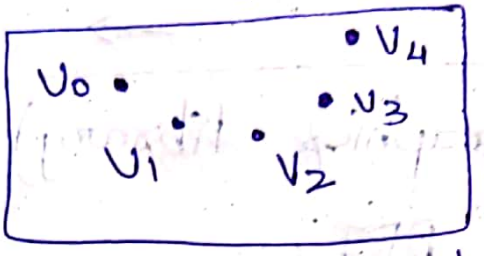
- `gl` belongs to GL library.
- `Vertex` — function name
- `3` is dimension can be 2 or 3
- `f` — x, y, z are floats

Program Structure :

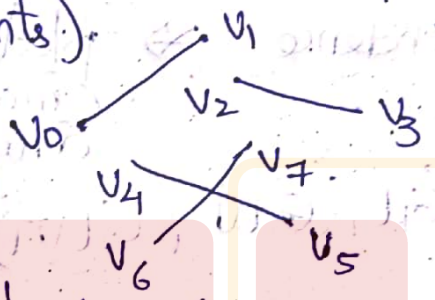
* **main():**
  - → defines the callback functions
  - → opens one or more windows with the required properties
  - → enters event loop
* **init():** sets the state variables
  - → Viewing
  - → Attributes
* **Callbacks**
  - → Display function
  - → Input & window function.

primitives :

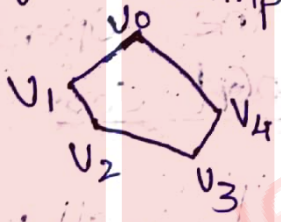
GL_POINTS → used to draw points (individual points)



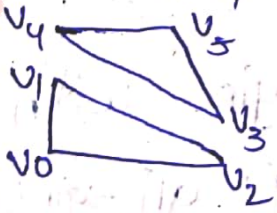
GL_LINES allows u to draw a line (pairs of vertices interpreted as individual line segments).



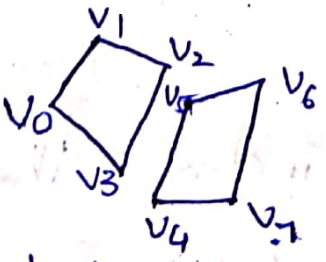
GL_POLYGON used to draw a polygon (boundary of a simple, convex polygon)



GL_TRIANGLES used to draw a triangles (triple of vertices interpreted as triangle)



GL_QUADS used to draw a quads (quadruple of vertices interpreted as four-sided polygon).



GL_LINE_STRIP - used to draw a line strip (series of connected line segments)

Various forms of glVertex function calls

glVertex 3 f v ( )

- 2 → two components
- 3 → three "
- 4 → four "
- f → float
- d → double
- i → int
- s → short
- v → vector

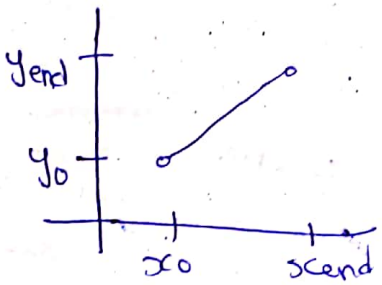
Attributes : These are the part of the OpenGL state & determine the appearance of objects

* color (points, lines, polygons)
* size & width (points, lines)
* stipple pattern (lines, polygon)
* polygon mode
  → display as filled
  → display edges
  → display vertices

Line drawing Algorithm :

→ A straight-line segment in a scene is defined by co-ordinate positions for the end points of the segment.
→ programmer specifies (x, y) values of end pixels
→ need algorithm to figure out which intermediate pixels are on line path
→ Actual computed intermediate line values may be floats
→ Rounding may be required.
→ sloped lines end up having jaggies
→ Vertical, horizontal lines, no jaggies

slope $y = m * x + c$



Digital differential Analyzer (DDA): (Line drawing algorithm)

→ The DDA is a scan-conversion line algorithm based on calculating δy or δx

→ A line is sampled at unit intervals in one coordinate and the corresponding integer values nearest the line path are determined.

→ DDA algorithm has three cases so from equation i.e., $m = (y_{k+1} - y_k)/(x_{k+1} - x_k)$

Case-1: If $m < 1$, x increment in unit intervals

i.e., $x_{k+1} = x_k + 1$

then, $m = (y_{k+1} - y_k)/(x_{k+1} - x_k)$

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

$$y_{k+1} = y_k + m \rightarrow (1)$$

Case-2: If $m > 1$, y increment in unit intervals

i.e., $y_{k+1} = y_k + 1$

then, $m = (y_k + 1 - y_k)/(x_{k+1} - x_k)$

$m(x_{k+1} - x_k) = 1$

$$x_{k+1} = (1/m) + x_k \rightarrow (2)$$

Case-3: If $m = 1$, both x and y increment in unit intervals

ie., $x_{k+1} = x_k + 1$ and $y_{k+1} = y_k + 1$

→ The DDA algorithm is faster method for calculating pixel position than one

that directly implements.

OpenGL code for DDA algorithm

```
#include <stdio.h>
inline int round (const float a) {return
                                  int(0 +0.5); }
void DDA Line(int x0, int y0, int xend,
                                  int yend)
{
    int dx = xend - x0, dy = yend - y0, steps,
                                        k ;
    float xinc, yinc, x = x0, y = y0 ;
    if (fabs(dx) > fabs(dy))
        steps = fabs(dx);
    else
        steps = fabs(dy);
    xinc = float (dx) / float (steps);
    yinc = float (dy) / float (steps);
    set pixel (round (x), round(y));
    for (k=0; k< steps; k+1)
    {
        x+ = xinc ;
        y+ = yinc;
        setpixel (round(x), round(y));
    }
}
```

Drawbacks:

* DDA is the simplest line drawing algorithm
  → Not very efficient
  → Round operation is expensive

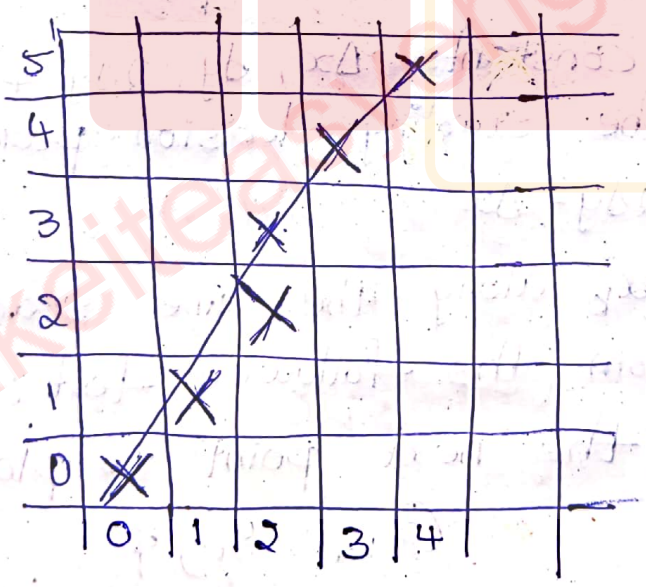
* Optimized algorithms typically used
→ integer DDA

problem: Example.

1) Apply DDA algorithm to draw line with end points (0,0) to (4,5)

↳ $m = \frac{y_2 - y_1}{x_2 - x_1} = \frac{5}{4} = 1.25$

x	y	xplot	yplot	(x,y)
0	0	0	0	(0,0)
0.8	1	1	1	(1,1)
1.6	2	2	2	(2,2)
2.4	3	3	3	(2,3)
3.2	4	4	4	(3,4)
4.0	5	5	5	(4,5)



Bresenham's line-drawing algorithm

* Accurate & efficient
* Uses only incremental integer calculations

→ The method is described for a line segment with a positive slope less than one.

→ The method generalizes to line segments of other slopes by considering the symmetry b/w various octants & quadrants of the xy plane

Bresenham's Line-drawing algorithm for $|m| < 1$

1. Input the two line endpoints and store the left endpoint in (x_0, y_0)
2. Load (x_0, y_0) into the frame buffer, that is, plot the first point.
3. Calculate constants Δx, Δy, $2\Delta y$ & $2\Delta y - 2\Delta x$ & obtain the starting decision parameter as

$$P_0 = 2\Delta y - \Delta x$$

4. At each x_k along the line, starting at $k=0$, perform the following test:

If $P_k < 0$, the next point to plot is (x_{k+1}, y_k) &

$$P_{k+1} = P_k + 2\Delta y$$

Otherwise, the next point to plot is (x_{k+1}, y_{k+1}) and

$$P_{k+1} = P_k + 2\Delta y - 2\Delta x$$

5. Repeat step 4 $\Delta x - 1$ times

Pseudo code:

```
#include< stdlib.h>
#include< math.h>
/* Bresenham line-drawing procedure for
   |m| < 1.0
Void lineBres(int x0, int y0, int xEnd,
                              int yEnd)
{
   int dx = fabs(xEnd - x0), dy = fabs(yEnd - y0);
   int x, y, P = 2 * dy - dx;
   int twoDy = 2 * dy, twoDyMinusDx =
                        2 * (dy - dx);
   if (x0 > xEnd) {
      x = xEnd; y = yEnd; xEnd = x0;
   }
   else {
      x = x0; y = y0;
   }
   setPixel(x, y);
   while (x < xEnd) {
      x++;
      if (P < 0)
         P+ = twoDy;
      else {
         y++;
         P+ = twoDyMinusDx;
      }
      setPixel(x, y);
   }
}
```

Eg:- Draw the line with end points (20, 10) & (30, 18)

↳ fix (end points) $x_0 = 20,\ y_0 = 10$

$\Delta x = x_2 - x_1 = 30 - 20 = 10 = 2\Delta x = 2(10) = 20$

$\Delta y = y_2 - y_1 = 18 - 10 = 8$

$2\Delta y = 2(8) = 16$

$2\Delta y - 2\Delta x = 16 - 20 = -4$

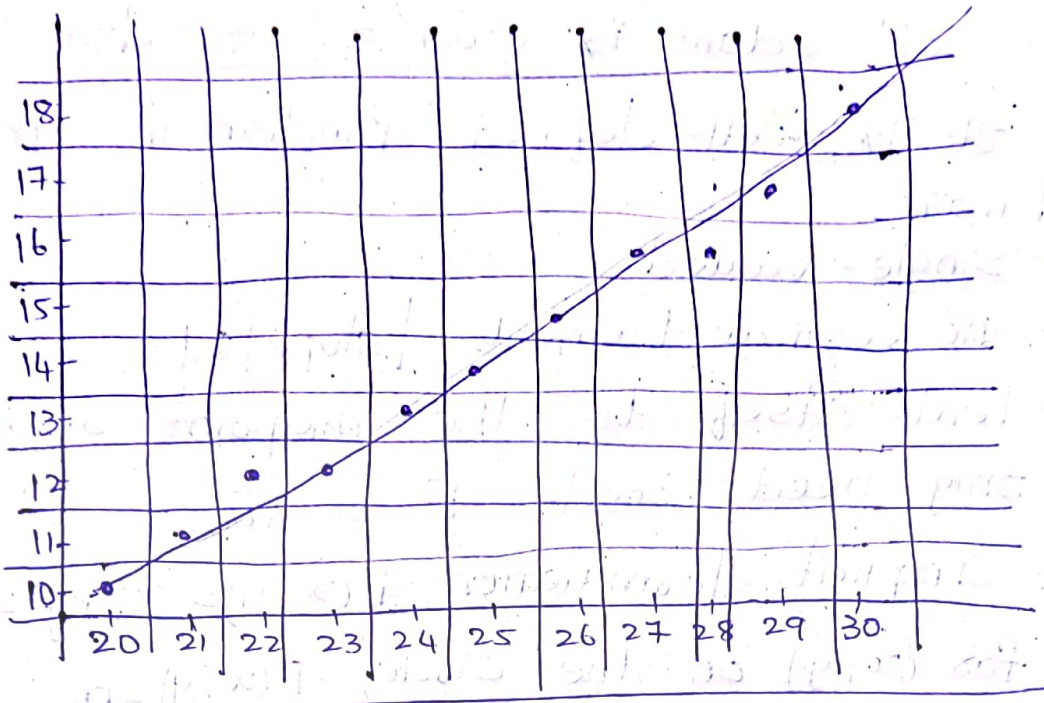
$P_0 = 2\Delta y - \Delta x$

$\Rightarrow 16 - 10 = 6$

Repeat $\Delta x - 1$
$10 - 1$
$= 9$

→ Initial decision parameter $P_0 = 6 > 0$

k	P_k	(x_{k+1}, y_{k+1})
0	6	(21, 11)
1	2	(22, 12)
2	-2	(23, 12)
3	14	(24, 13)
4	10	(25, 14)
5	6	(26, 15)
6	2	(27, 16)
7	-2	(28, 16)
8	14	(29, 17)
9	16	(30, 18).



Circle generation with mid-point:

→ Midpoint circle algorithm generates all points on a circle centered at the origin by incrementing all the way around circle.

→ To apply the midpoint method, we define a circle function as

Implicit: $f_{circ}(x, y) = x^2 + y^2 - r^2$

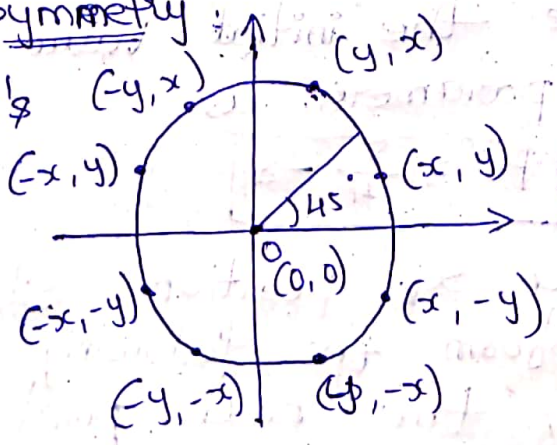
if $f(x, y) = 0$, then it is on the circle.

if $f(x, y) > 0$, then it is outside the circle

if $f(x, y) < 0$, then it is inside the circle.

Eight-way Symmetry:

Only one octant is calculation needed



The 2nd Octant is good arc to draw

* It is well-defined function in this domain
  - Single-valued
  - no vertical tangents : |slope| ≤ 1
* Lends itself to the midpoint approach
  - only need Consider E or SE
* Implicit formulation $F(x,y) = x^2+y^2-r^2$
  - for (x,y) on the circle, $F(x,y)=0$
  - $F(x,y) > 0 \Rightarrow (x,y)$ Outside
  - $F(x,y) < 0 \Rightarrow (x,y)$ Inside
* decision variable d is $x^2+y^2-r^2$
  → Then $d = F(M) \geq 0 \Rightarrow E$
  → or $d = F(M) < 0 \Rightarrow E$.

Midpoint circle Algorithm:

1. Input radius r and circle center (x_0, y_0) & obtain the first point on the circumference of a circle centered on the origin as

$$(x_0, y_0) = (0, r)$$

2. Calculate the initial value of the decision parameter as

$$p_0 = \frac{5}{4} - r$$

3. At each x_k position, starting at $k=0$, perform the following test:
   if $p_k < 0$, the next point along the

circle centered on (0,0) is (x_{k+1}, y_k) &

$$P_{k+1} = P_k + 2k+1+1$$

otherwise, the next point along the circle is $(x_{k+1}, y_k - 1)$ &

$$P_{k+1} = P_k + 2k+1 - 2y_{k+1}$$

where,

$$2k+1 = 2x_k + 2 \text{ and}$$
$$2y_{k+1} = 2y_k - 2$$

4. Determine symmetry points in the other seven octants.
5. Move each calculated pixel position (x, y) onto the circular path centered on (x_c, y_c) and plot the coordinate values:

$$x = x + x_c, \quad y = y + y_c$$

6. Repeat steps 3 through 5 until $x \geq y$.

Code :-

```
Void draw_pixel(GLint cx, GLint cy)
{
    glColor3f(0.5, 0.5, 0.0);
    glBegin(GL_POINTS);
        glVertex2i(cx, cy);
    glEnd();
}
Void plotpixels(GLint h, GLint k, GLint y)
{
    draw_pixel(x+h, y+k);
    draw_pixel(-x+h, y+k);
    draw_pixel(x+h, -y+k);
    draw_pixel(-x+h, -y+k);
```

```
    draw_pixel(y+h, x+k);
    draw_pixel(-y+h, x+k);
    draw_pixel(y+h, -x+k);
    draw_pixel(-y+h, -x+k);
}
void circle_draw(GLint xc, GLint yc,
                 GLint r)
{
    GLint d=1-r, x=0, y=r;
    while(y>r)
    {
        plotpixels(xc, yc, x, y);
        if(d<0) d+=2*x+3;
        else
        {
            d+=2*(x-y)+5;
            --y;
        }
        ++x;
    }
    plotpixels(xc, yc, x, y);
}
```