# UNIT- I

## Computer Graphics Basic, OpenGL and Line, Circle Drawing

Introduction: Introduction to computer graphics, basics of graphics systems, raster and random scan, basic display processor. OpenGL – Introduction, Graphics function, OpenGL Interface, primitives and attributes, Control functions, programming events Line Drawing: DDA Line drawing algorithm, Bresenham Line drawing algorithm Circle Drawing: Bresenham circle drawing algorithm Character Generation: Stroke principle, starburst principle, bitmap method, Introduction to aliasing and anti-aliasing.

a) Introduction CG :Introduction to **Computer Graphics** , basics of **Graphics Systems**, **raster** and **random** scan, **Basic Display Processor.**

**b) OpenGL** – Introduction – Graphics function, OpenGL Interface, primitives and attributes, Control functions, programming events.

c) Line Drawing : **DDA Line** drawing algorithm, **Bresenham Line** drawing algorithm, Circle Drawing: **Bresenham circle** drawing algorithm

d) **Character Generation**:

- Stroke principle,
- starburst principle,
- bitmap method.

e) Introduction to **Aliasing and Anti-aliasing.**

# Introduction CG :Introduction to **Computer Graphics**

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

1. Computer is information processing machine.
2. User needs to communicate with computer and the computer graphics is one of the most effective and commonly used ways of communication with the user.
3. It displays the information in the form of graphical objects such as pictures, charts, diagram and graphs.
4. In computer graphics picture or graphics objects are presented as a collection of discrete pixels.
5. We can control intensity and color of pixel which decide how picture look like.

## Rasterization

The special procedure determines which pixel will provide the best approximation to the desired picture or graphics object this process is known as "**Rasterization**."

## Scan Conversion

The process of representing continuous picture or graphics object as a collection of discrete pixels is called "**Scan Conversion.**"

## Application of Computer Graphics

1. **Education and Training:** Computer-generated model of the physical, financial and economic system is often used as educational aids. Model of physical systems, physiological system, population trends or equipment can help trainees to understand the operation of the system.
For some training applications, particular systems are designed. For example Flight Simulator.

2. **Flight Simulator:** It helps in giving training to the pilots of airplanes. These pilots spend much of their training not in a real aircraft but on the ground at the controls of a Flight Simulator.

3.**Use in Biology:** Molecular biologist can display a picture of molecules and gain insight into their structure with the help of computer graphics.

4. **Computer-Generated Maps:** Town planners and transportation engineers can use computer-generated maps which display data useful to them in their planning work.

5. **Architect:** Architect can explore an alternative solution to design problems at an interactive graphics terminal. In this way, they can test many more solutions that would not be possible without the computer.

**6. Presentation Graphics:** Example of presentation Graphics are bar charts, line graphs, pie charts and other displays showing relationships between multiple parameters. Presentation Graphics is commonly used to summarize.

**6. Computer Art:** Computer Graphics are also used in the field of commercial arts. It is used to generate television and advertising commercial.

**7. Entertainment:** Computer Graphics are now commonly used in making motion pictures, music videos and television shows.

**8. Visualization:** It is used for visualization of scientists, engineers, medical personnel, business analysts for the study of a large amount of information.

**9. Educational Software:** Computer Graphics is used in the development of educational software for making computer-aided instruction.

**10. Printing Technology:** Computer Graphics is used for printing technology and textile design.

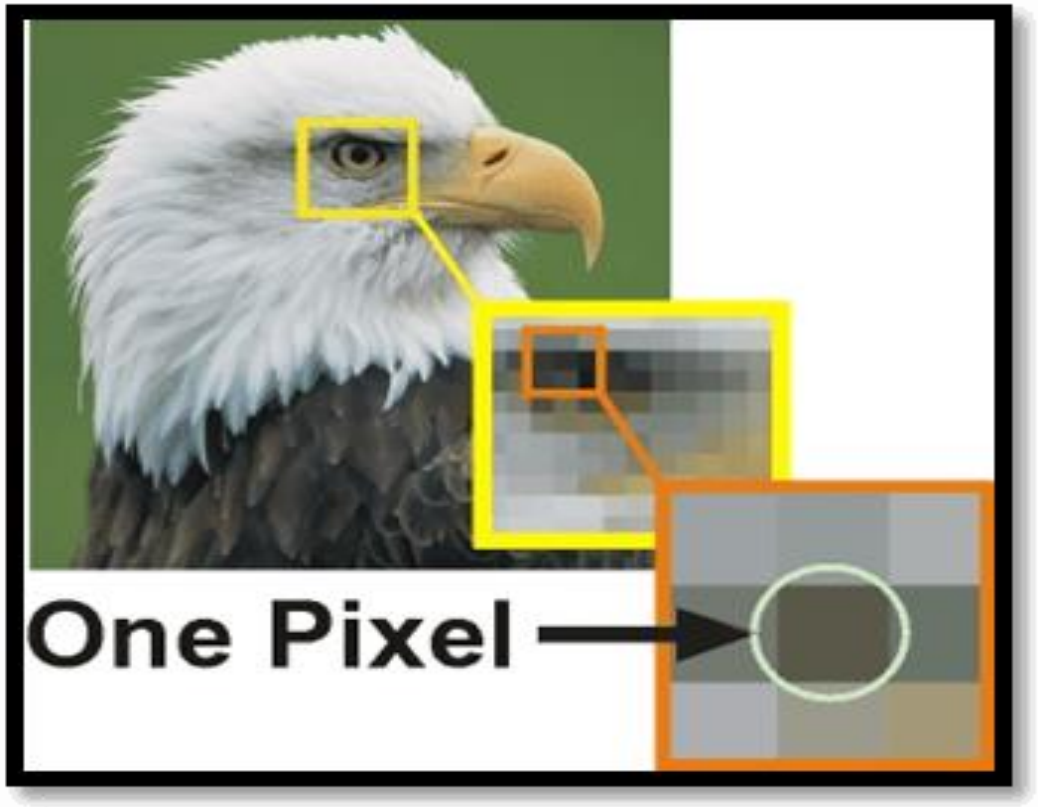
# Basics of **Graphics Systems**

The display adapter determines the maximum resolution, refresh rate and number of colors that can be displayed, which the monitor must also be able to support. On most PCs, these graphics circuits are built into the motherboard's chipset

A resolution of a computer screen depends upon graphics card and display monitor, the quantity, size and color combination of pixels.

## Pixel

1. The full form of the pixel is "Picture Element." It is also known as "PEL."
2. This is a unit for measuring image resolution
3. Pixel is the smallest element of an image on a computer display, whether they are LCD or CRT monitors.
4. A screen is made up of a matrix of thousands or millions of pixels.
5. A pixel is represented with a dot or a square on a computer screen.
6. The good thing is that a pixel cannot be seen as they are very small which result in a smooth and clear image rather than "pixelated." Each pixel has a value, or we can say a unique logical address.
7. It can have only one color at a time. Colour of a pixel is determined by the number of bits which is used to represent it.




As we know that an image is build up of thousands and millions of pixels. In the above images, if we zoom in the image, we will be able to see some of the pixels.

## Calculation of the total number of pixels

Below is the formula to calculate the total number of pixel in an image.

| Total number of pixels | = | Number of rows | X | Number of columns |
|---|---|---|---|---|

**For example:** let rows=300 & columns=200

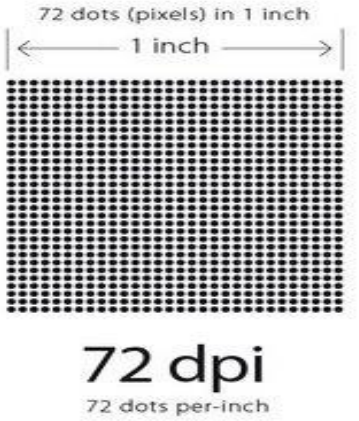
Total number of pixels= 300 X 200

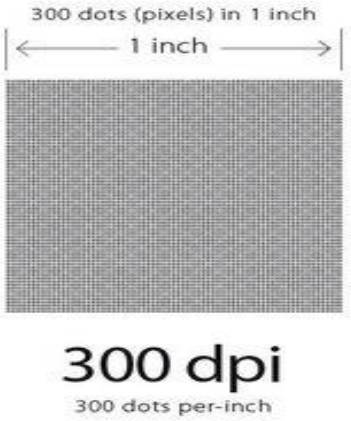
# Resolution :

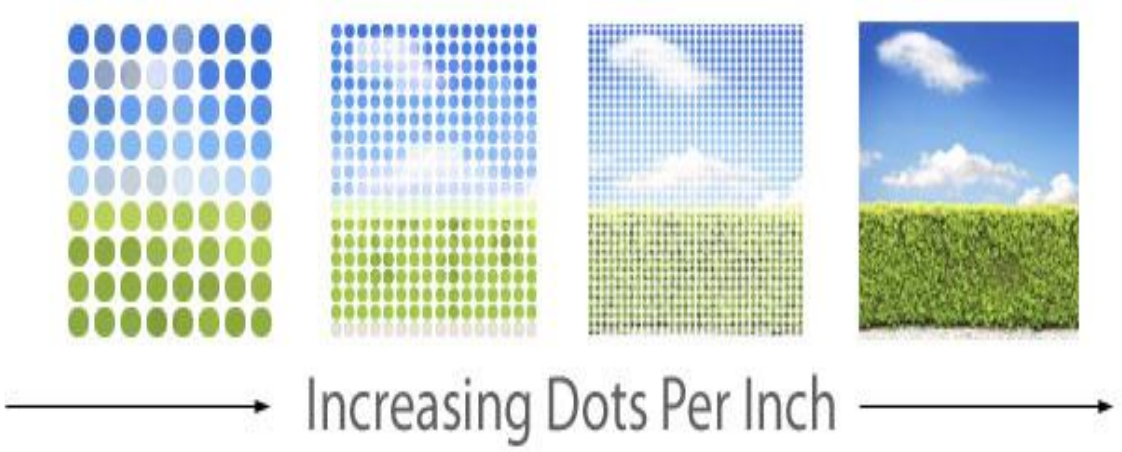
The resolution of a display is determined by counting the horizontal and vertical pixels.

A number of standards currently exist when it comes to display resolutions:

| Name(s) | Resolution in pixels |
|---|---|
| High Definition (HD) | 1280 x 720 |
| Full HD, FHD | 1920 x 1080 |
| 2K, Quad HD, QHD | 2560 x 1440 |
| 4K, Ultra HD | 3840 x 2160 |



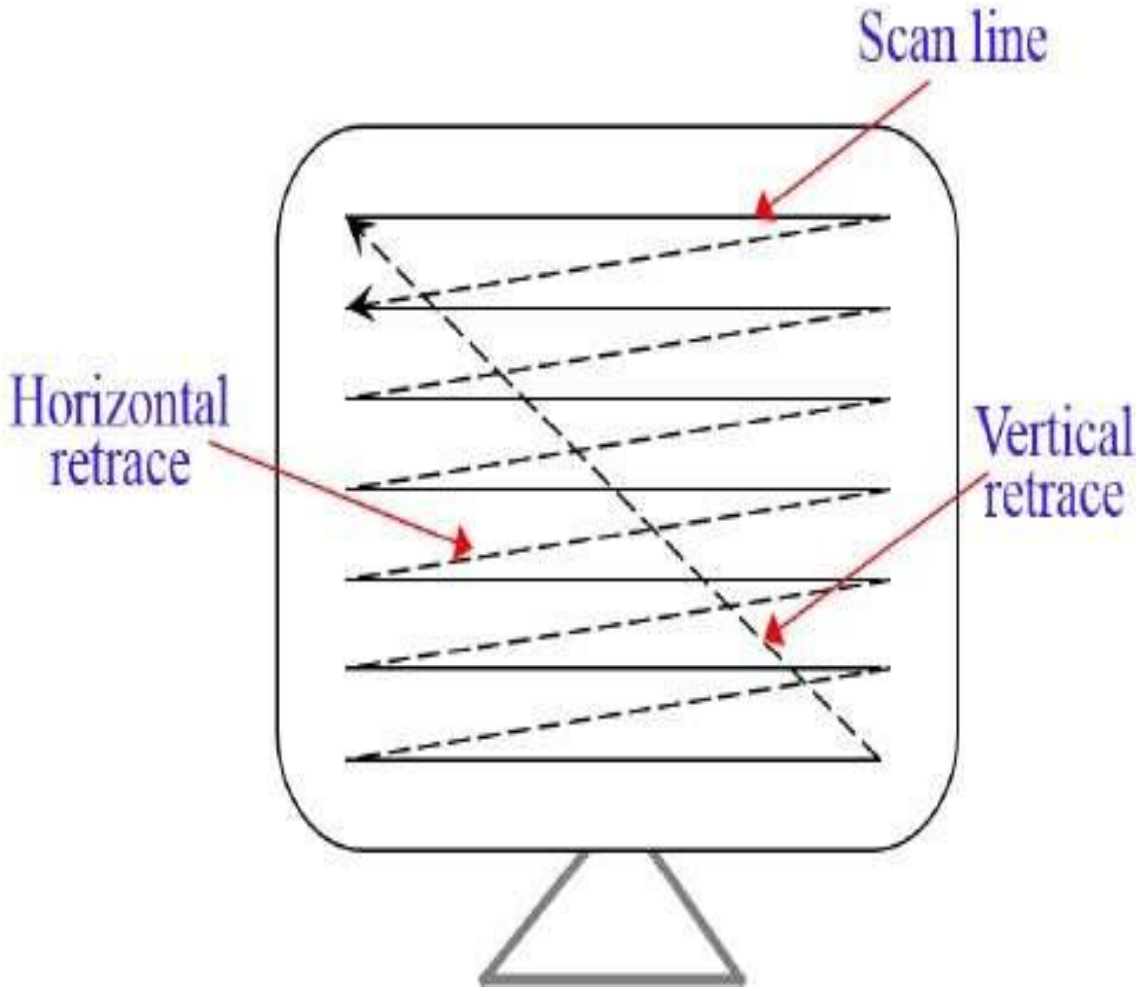



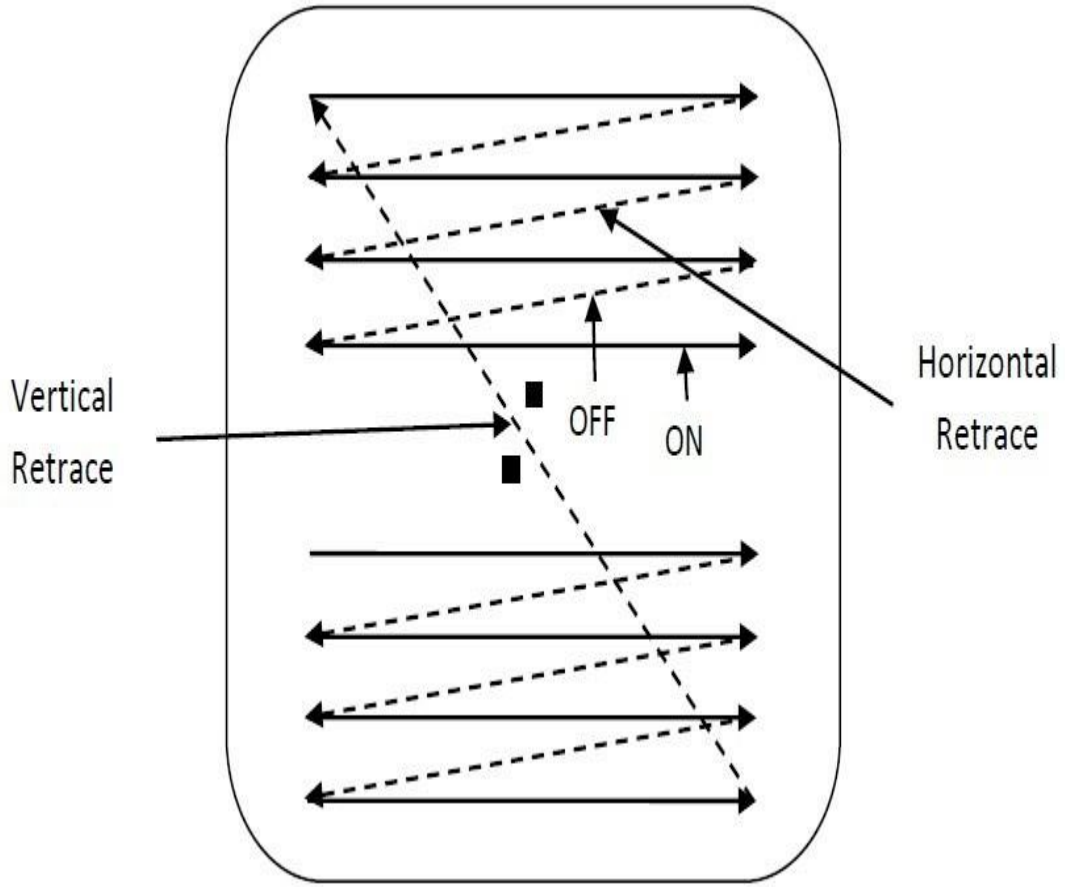
# Refresh Rate

The refresh rate of your display refers to how many times per second the display is able to draw a new image. This is measured in Hertz (Hz)

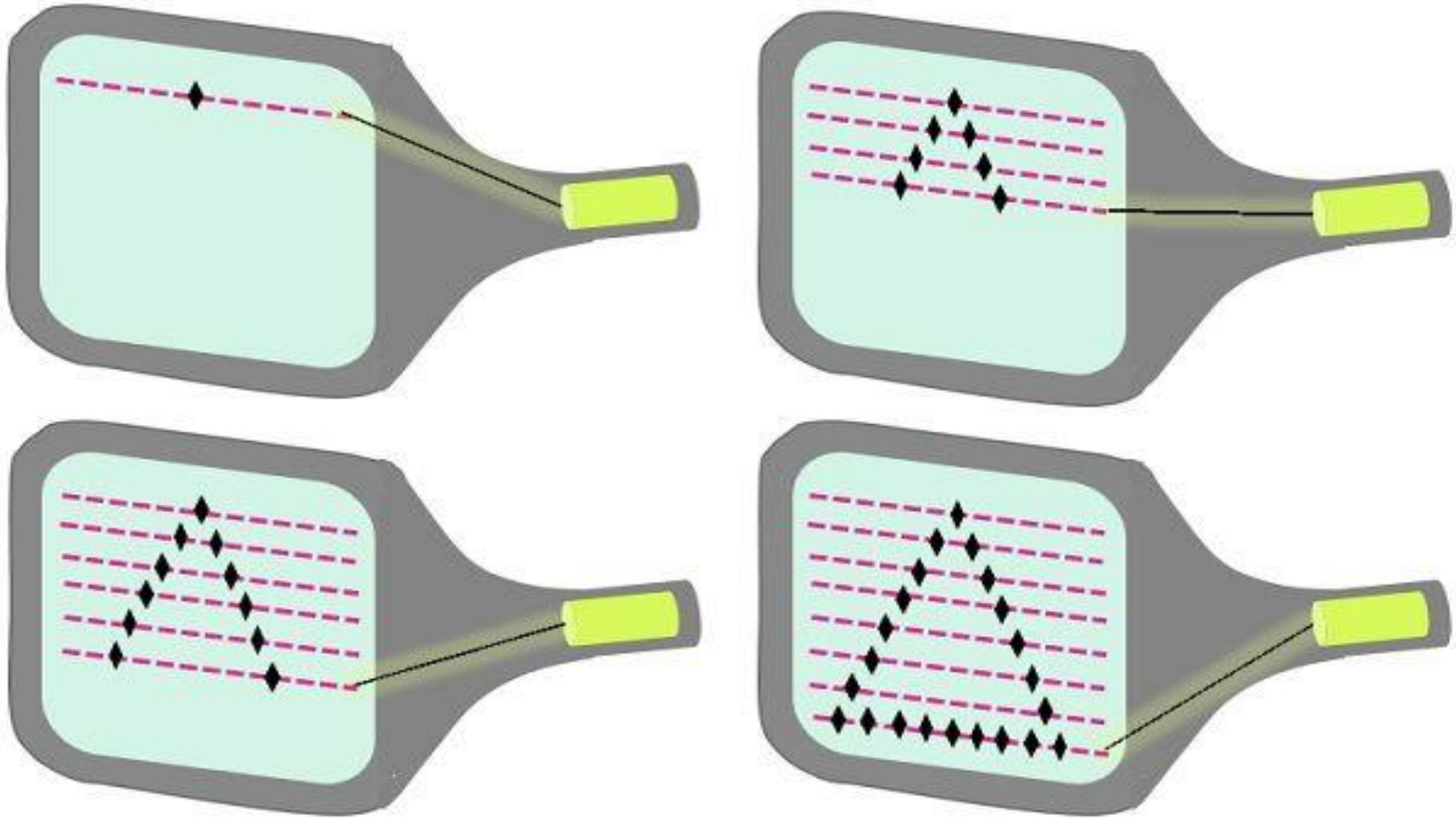
# Raster Scan

1. Raster Scan Displays are most common type of graphics monitor which employs CRT.
2. It is based on television technology.
3. In raster scan system electron beam sweeps across the screen, from top to bottom covering one row at a time
4. A pattern of illuminated pattern of spots is created by turning beam intensity on and off as it moves across each row.
5. A memory area called refresh buffer or frame buffer stores picture definition.
6. This memory area holds intensity values for all screen points.
7. Stored intensity values are restored from frame buffer and painted on screen taking one row at a time. Each screen point is referred to as pixels.
8. In raster scan systems refreshing is done at a rate of 60-80 frames per second

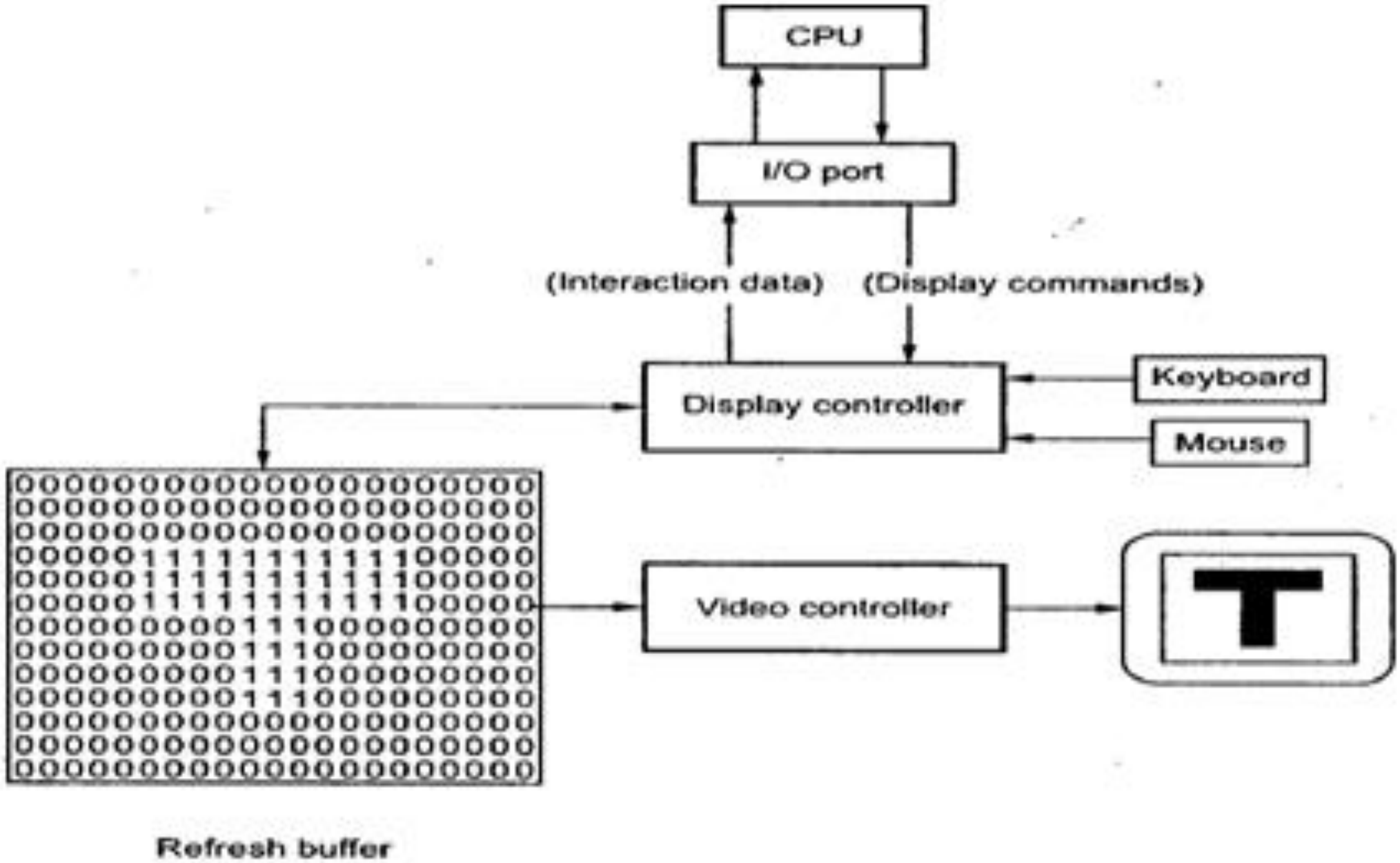

Scan line
Horizontal retrace
Vertical retrace


Vertical
Retrace
OFF
ON
Horizontal
Retrace

# Raster Scan

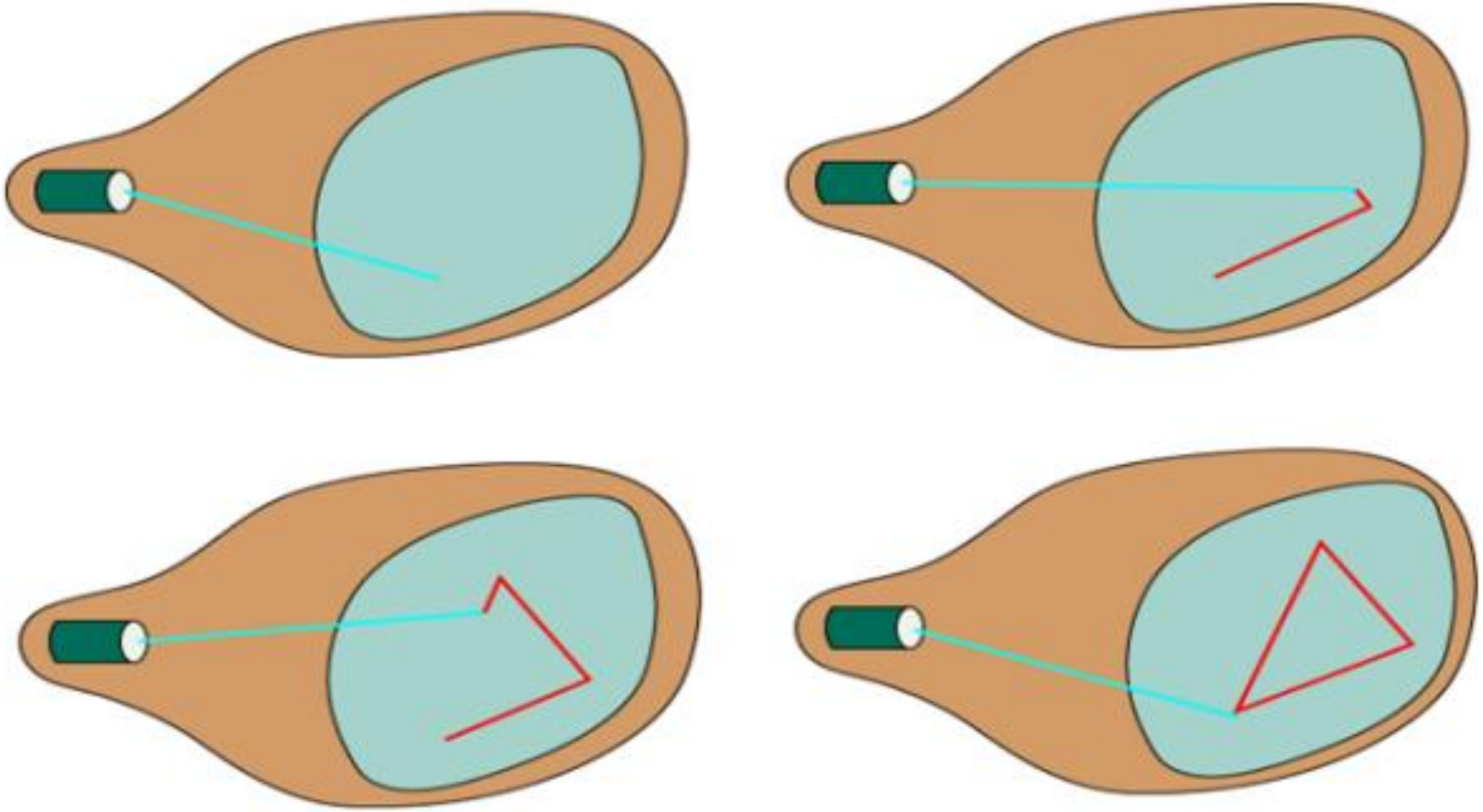


**Raster Scan**: A raster scan system displays an item as a group of separate points along each screen line


CPU
I/O port
(Interaction data)
(Display commands)
Display controller
Keyboard
Mouse
Video controller
Refresh buffer

# Random-Scan Display

1. In Random-Scan Display electron beam is directed only to the areas of screen where a picture has to be drawn. It is also called vector display, as it draws picture one line at time.
2. Random Scan System uses an electron beam which operates like a pencil to create a line image on the CRT screen.
3. The picture is constructed out of a sequence of straight-line segments.
4. Each line segment is drawn on the screen by directing the beam to move from one point on the screen to the next, where its x & y coordinates define each point
5. After drawing the picture. The system cycles back to the first line and design all the lines of the image 30 to 60 time each second.

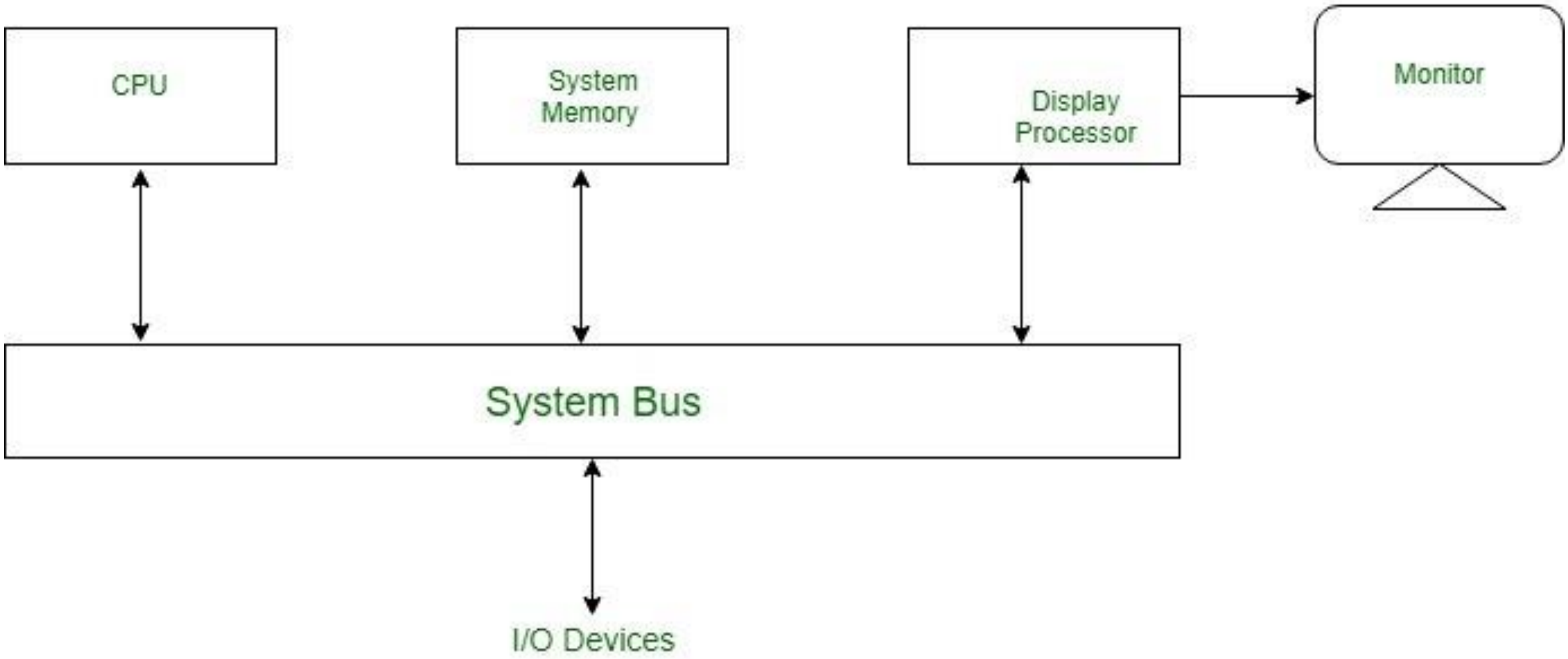


**Random-scan monitors are also known as vector displays or stroke-writing displays or calligraphic displays**

# Random-Scan Display

1. Input in the form of an application program is stored in the system memory along with graphics package
2. Graphics package translates the graphic commands in application program into a display file stored in system memory.
3. This display file is then accessed by the display processor to refresh the screen
4. The display processor cycles through each command in the display file program
5. Sometimes the display processor in a random-scan is referred as Display Processing Unit / Graphics Controller.

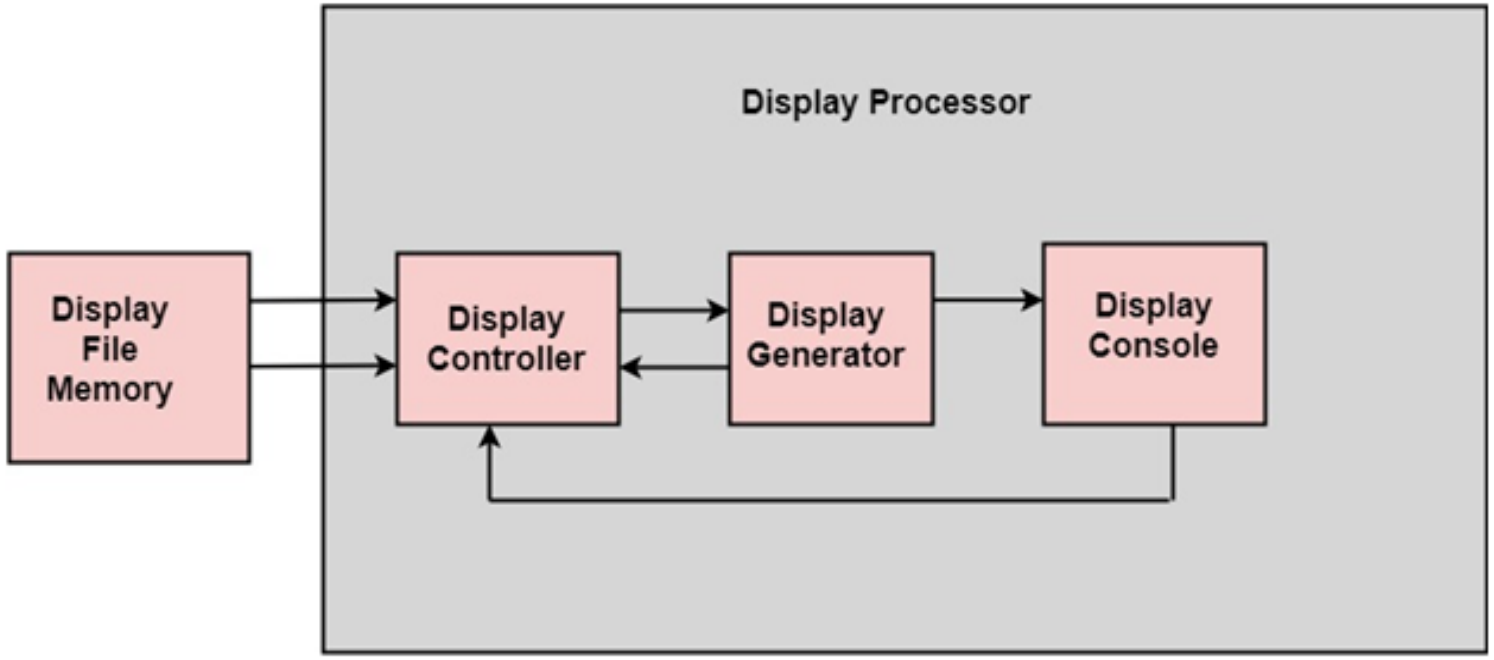
# Random-Scan Display



| Raster Scan | Random Scan |
| --- | --- |
| The electron beam is swept across the screen, one row at a time, from top to bottom. | The electron beam is directed only to the parts of screen where a picture is to be drawn. |
| Its resolution is poor because raster system in contrast produces zigzag lines that are plotted as discrete point sets. | Its resolution is good because this system produces smooth lines drawings because CRT beam directly follows the line path. |
| Picture definition is stored as a set of intensity values for all screen points, called pixels in a refresh buffer area. | Picture definition is stored as a set of line drawing instructions in a display file. |
| The capability of this system to store intensity values for pixel makes it well suited for the realistic display of scenes contain shadow and color pattern. | These systems are designed for line- drawing and can't display realistic shaded scenes. |
| Screen points/pixels are used to draw an image. | Mathematical functions are used to draw an image. |

| Raster Scan | Random Scan |
| --- | --- |
| Cost is less. | Cost is more. |
| Raster display has ability to display areas, filled with solid colors or patterns. | Vector display only draws lines and characters. |
| Refreshing on Raster scan display is carryout at the Rate of 60 to 80 frames per seconds. | Refreshing in Random scan display to draw all the components of lines of pictures 30 to 60 frames per second. |
| It is used for photos.That is Why photoshop is Raster editing program. | It is used for text ,logs,letterheads. |
| Raster Scan Mainly in photos like JPG,PNG,GIF File Format | Random Scan system High Quality Images In SVG,the Size is Rescaleable like Logo in sites |

# Display Processor




Block diagram of Display System

**Display Console**: It consists of a CRT, Light pen, keyboard, and Deflection system.

**Display File Memory:** It is used to create an image. It is also used for visual object recognition.

**Display Generator:** It is used to produce the character. It is also used to create curves.

# Display Processor

- Display Processor  Is the interpreter or a hardware that converts display processor code into picture.
- The Display Processor converts the digital information from CPU to analog values.
- The main purpose of the Digital Processor is to free the CPU from most of the graphic chores.
- It is also called the Video controller. It is used to control the operations of the display device. Its functions are as follows:
    - It is used to handle the interrupt.
    - It is used to interpret the instructions.
    - It is also used to manage time.

# **OpenGL** – Introduction

- OpenGL is mainly considered an API (an Application Programming Interface) that provides us with a large set of functions that we can use to manipulate graphics and images.
- Open Graphics Library (OpenGL) is a cross-language (language independent), cross-platform (platform-independent) API for rendering 2D and 3D Vector Graphics(use of polygons to represent image). OpenGL API is designed mostly in hardware.

A low-level graphics library specification.

A small set of geometric primitives:

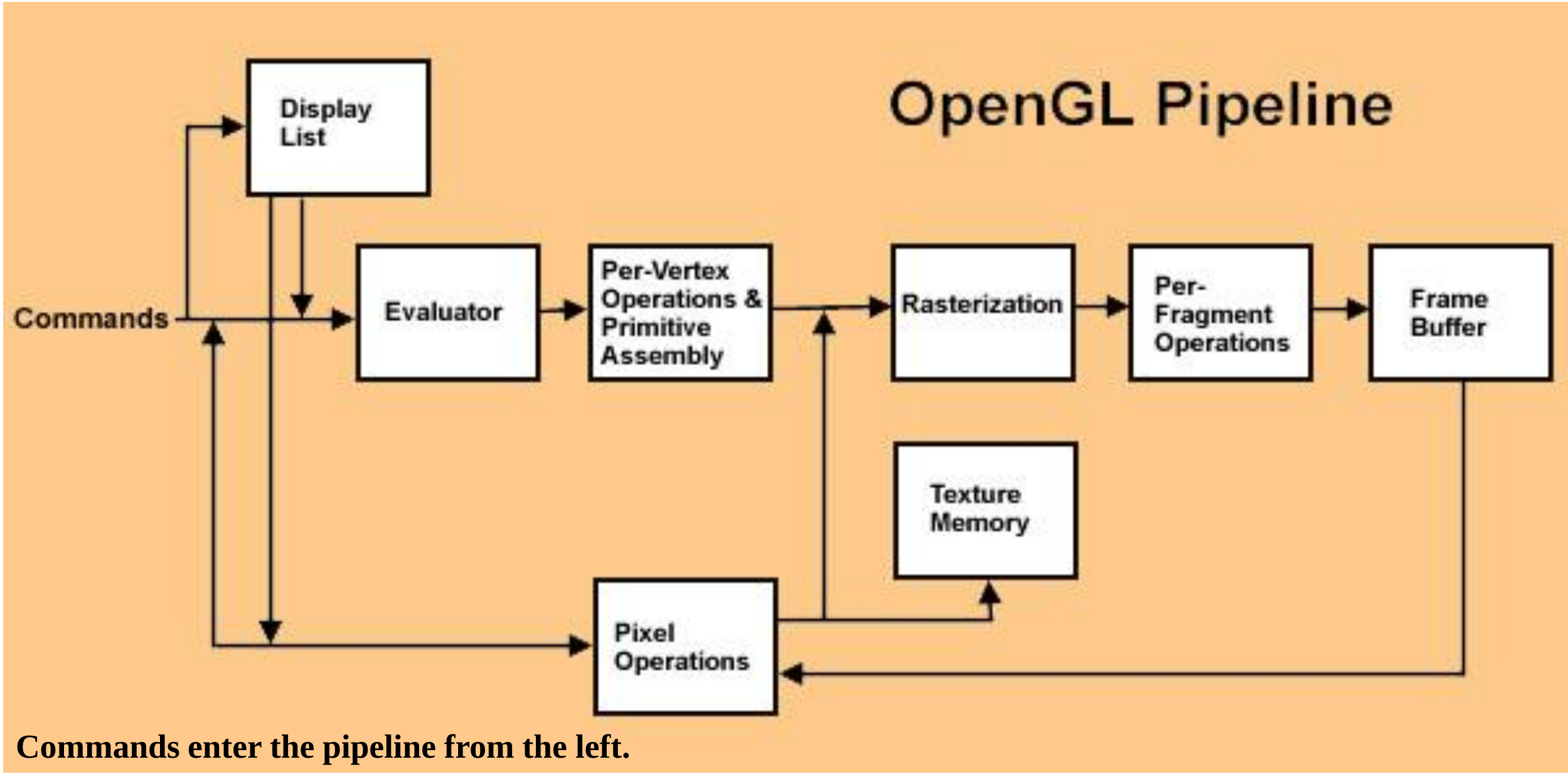
1. Points
2. Lines
3. Polygons
4. Images

- OpenGL (Open Graphics Library) is a cross-language, cross-platform application programming interface (API) for rendering 2D and 3D vector graphics.
- OpenGL is a hardware-independent, operating system independent
- The API is typically used to interact with a graphics processing unit (GPU), to achieve hardware-accelerated rendering.
- The OpenGL specification describes an abstract API for drawing 2D and 3D graphics.
- Although it is possible for the API to be implemented entirely in software, it is designed to be implemented mostly or entirely in hardware.
- OpenGL is window system independent, and therefore contains no windowing operations or mechanisms for user input. Also, OpenGL does not provide direct support for complex geometrical shapes, such as cubes or spheres.

# OpenGL interface

- OpenGL is the software interface to graphics hardware
- Modern computers have dedicated GPU (Graphics Processing Unit) with its own memory to speed up graphics rendering.
- OpenGL graphic rendering commands issued by your applications could be directed to the graphic hardware and accelerated.
- The API is typically used to interact with a graphics processing unit (GPU), to achieve hardware-accelerated rendering.
- The OpenGL specification describes an abstract API for drawing 2D and 3D graphics.
- Although it is possible for the API to be implemented entirely in software, it is designed to be implemented mostly or entirely in hardware.

**The OpenGL architecture** is structured as a state-based pipeline




**Commands enter the pipeline from the left.**

- Commands enter the pipeline from the left
- The first stage evaluator : evaluates them into their corresponding vertex and attribute commands.
- The second stage per-vertex operations, including transformations, lighting, primitive assembly, clipping, projection, and viewport mapping.
- The third stage is rasterization- produces fragments, which are series of frame buffer addresses and values, from the viewport-mapped primitives as well as bitmaps and pixel rectangles.
- The fourth stage is the per-fragment - Before fragments go to the frame buffer, they may be subjected to a series of conditional tests and modifications, such as blending or z-buffering.
- Parts of the frame buffer may be fed back into the pipeline as pixel rectangles.
- Texture memory may be used in the rasterization process when texture mapping is enabled.

# libraries in our OpenGL programs:

## 1. Core OpenGL (GL):

- Consists of hundreds of commands, which begin with a prefix "gl" (e.g., glColor , glVertex, glTranslate, glRotate).
- The Core OpenGL models an object via a set of geometric primitives such as point, line and polygon.

## 2. OpenGL Utility Library (GLU):

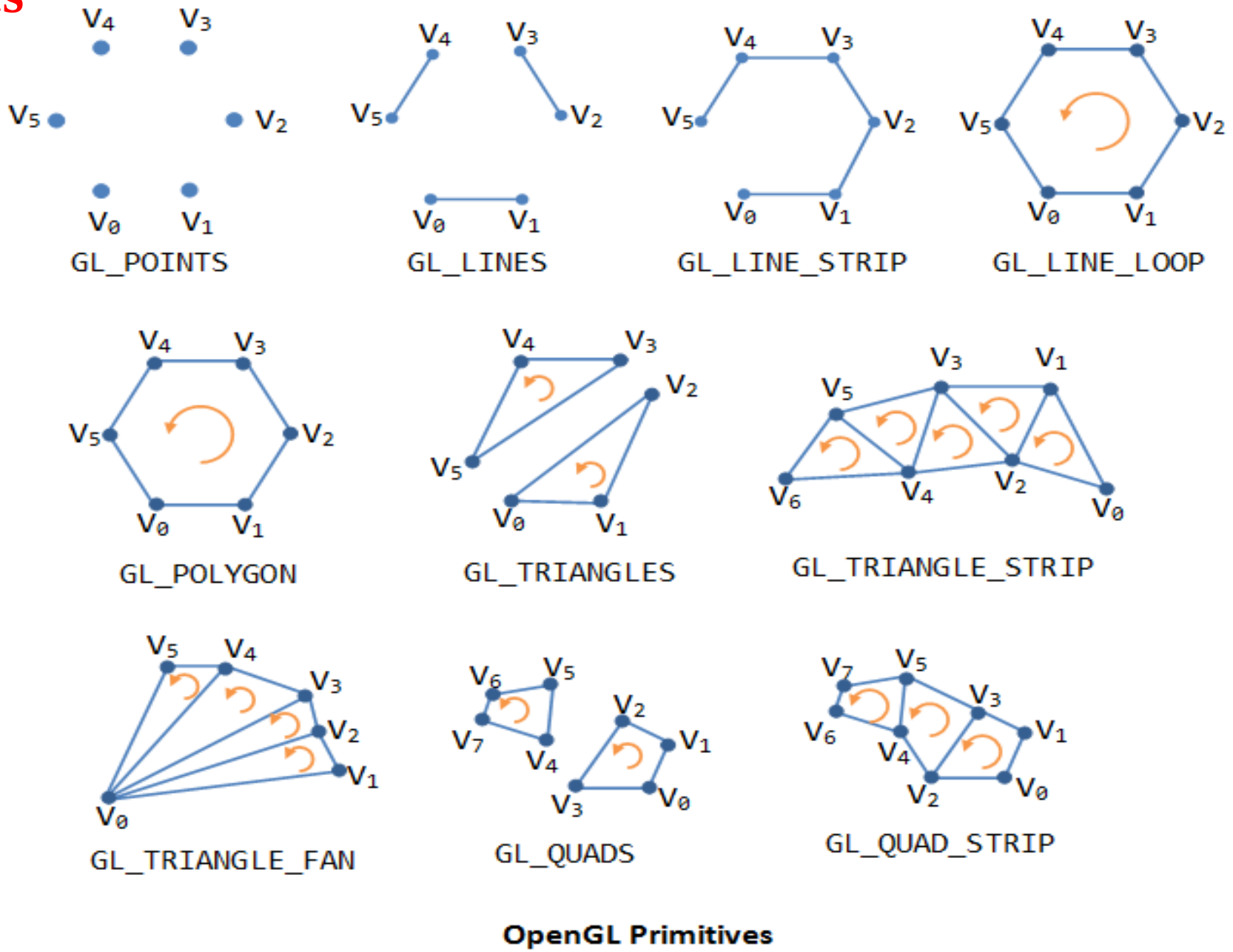
- Built on-top of the core OpenGL to provide important utilities (such as setting camera view and projection) and more building models (such as qradric surfaces and polygon tessellation).
- GLU commands start with a prefix glu (e.g., gluLookAt, gluPerspective).

# OpenGL Utilities Toolkit (GLUT):

1. OpenGL is designed to be independent of the windowing system or operating system.
2. GLUT is needed to interact with the Operating System (such as creating a window, handling key and mouse inputs);
3. it also provides more building models (such as sphere and torus).
4. GLUT commands start with a prefix of "glut" (e.g., glutCreatewindow, glutMouseFunc).
5. GLUT is platform independent, which is built on top of platform-specific
6. "GLUT is designed for constructing small to medium sized OpenGL programs.
7. While GLUT is well-suited to learning OpenGL and developing simple OpenGL applications,
8. GLUT is not a full-featured toolkit so large applications requiring sophisticated user interfaces are better off using native window system toolkits.
9. GLUT is simple, easy, and small.

# Primitives and Attributes

In OpenGL, an object is made up of geometric primitives such as triangle, quad, line segment and point. A primitive is made up of one or more vertices. All of these primitives are specified using a sequence of vertices.




**OpenGL Primitives**

```
void display()
{
        glClearColor(0.f, 1.0f, 0.0f, 1.0f);            // Set background colour to black and opaque
        glClear(GL_COLOR_BUFFER_BIT);       // Clear the colour buffer
        glPointSize(10.0f);
        glBegin(GL_POINTS);                  //starts drawing of points
        glColor3f(1,0,0);
        glVertex2f(-1.0, -1.0 );                        // red for vertexes
        glVertex2f (0.0, 0.0);                          // Points
        glEnd();                                        //end drawing of points
        glFlush();                                       // Render now

}
```

A geometric primitive is defined by specifying its vertices **via glVertex** function, enclosed within a pair **glBegin** and **glEnd.**

```
glBegin(GL_LINES);
   glVertex2f(-1.0f,-1.0f);
   glVertex2f(1.0f,1.0f);
glEnd();
```

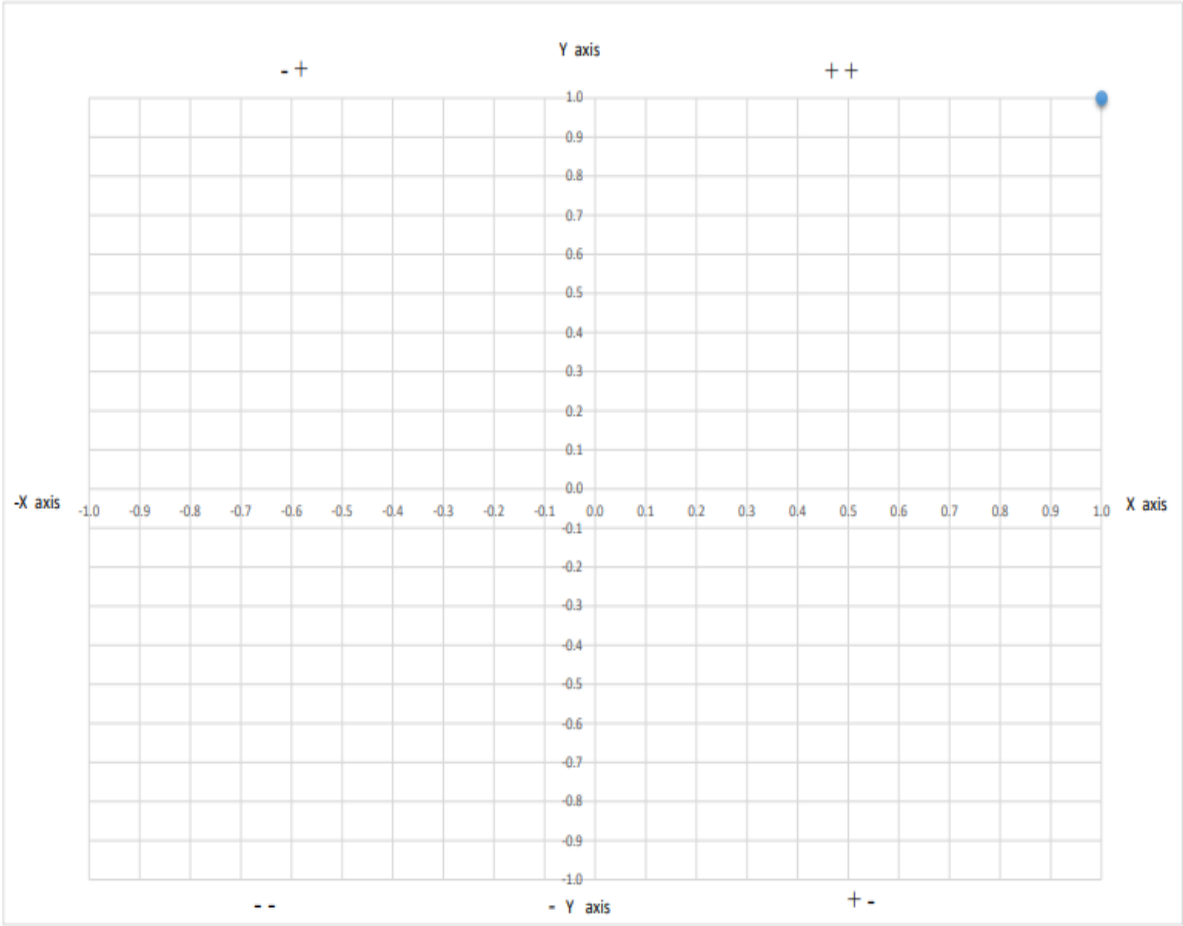
The various forms of glVertex function calls are defined as follows:

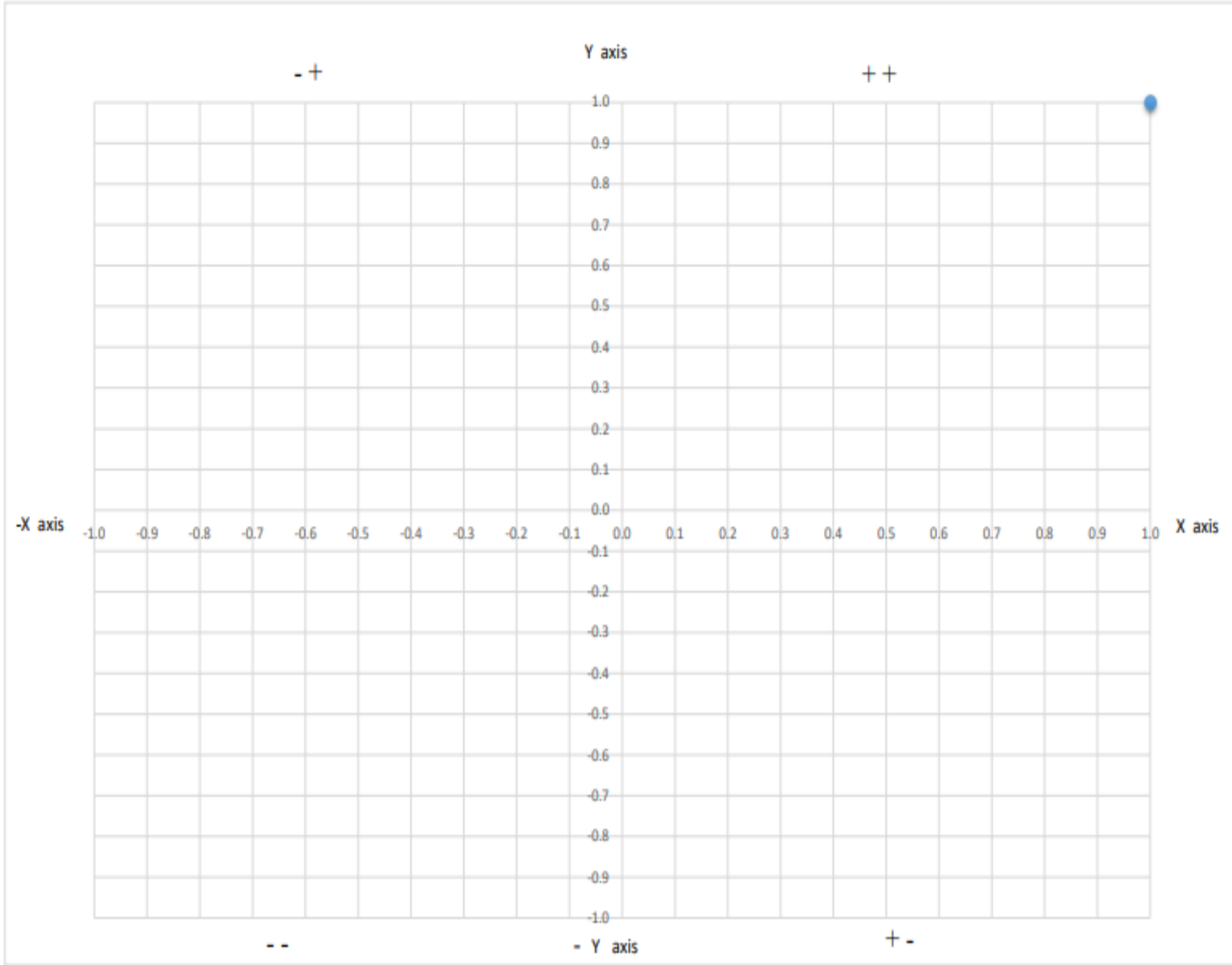
```
glVertex3fv(    )
```

indicates coordinates are stored in a vector if the 'v' is present.

- f float
- d double
- i int
- s short

- 2 two components
- 3 three components
- 4 four components



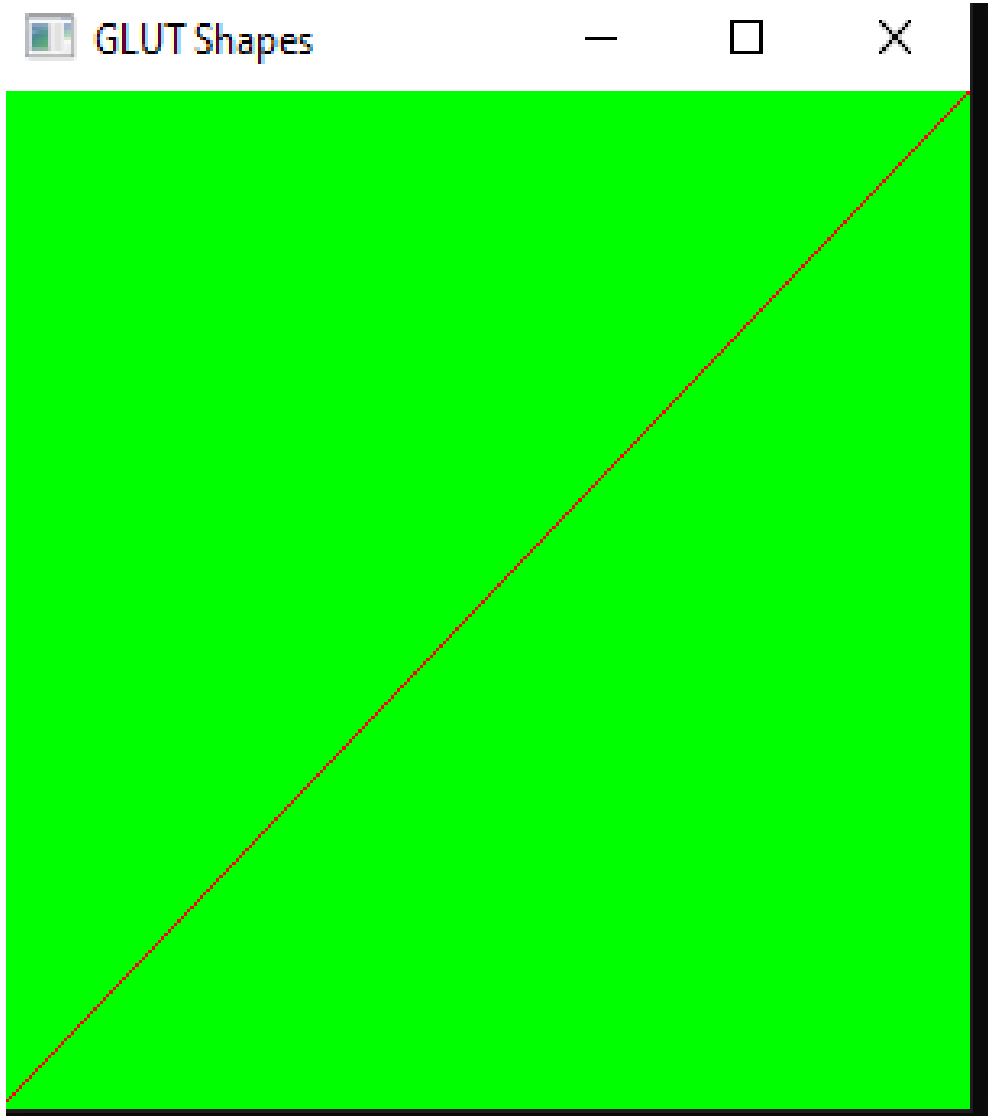

Y axis
- +
+ +
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
-0.1
-0.2
-0.3
-0.4
-0.5
-0.6
-0.7
-0.8
-0.9
-1.0
-1.0 -0.9 -0.8 -0.7 -0.6 -0.5 -0.4 -0.3 -0.2 -0.1 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
-X axis
X axis
- -
- Y axis
+ -

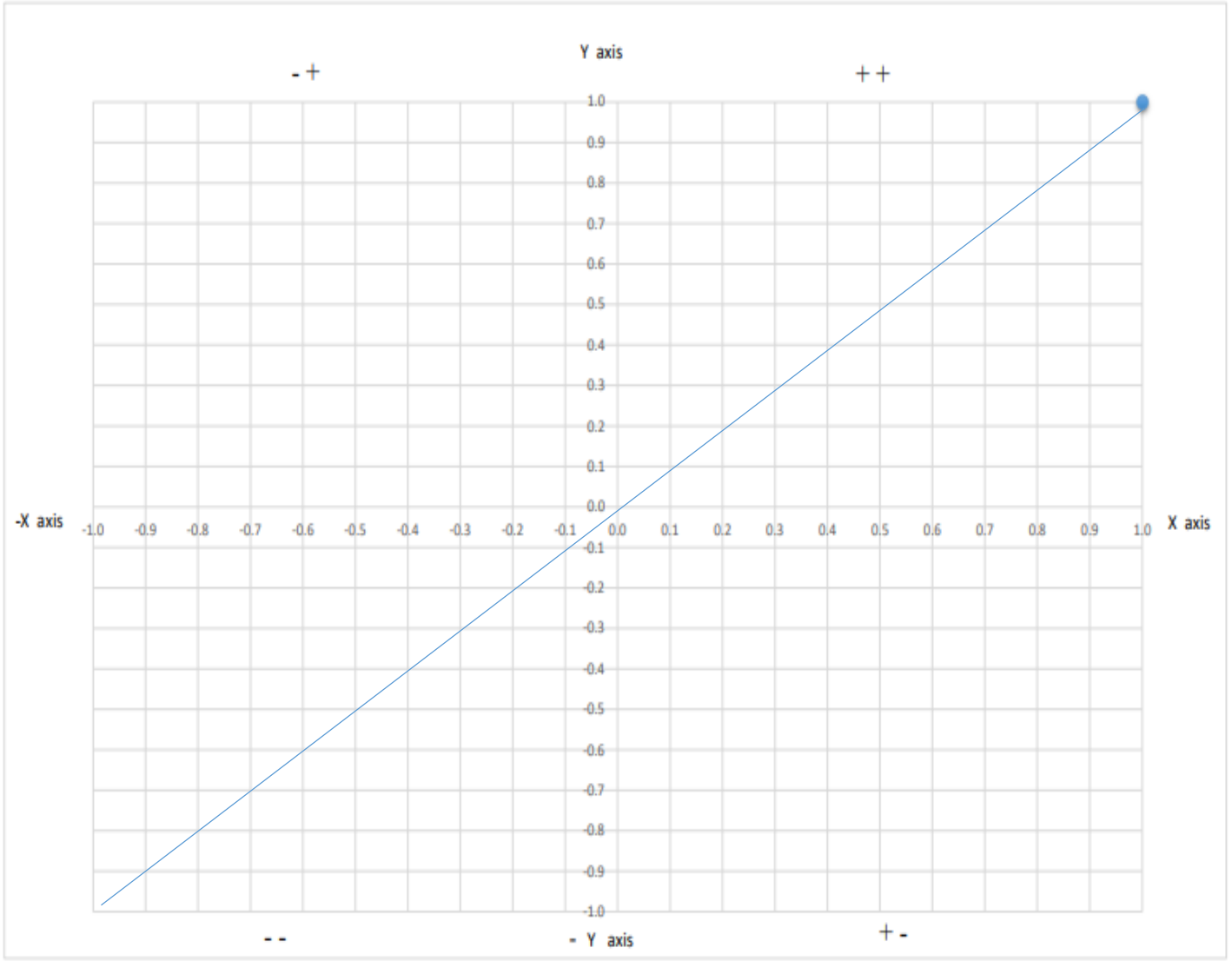
```
// *argv[] is string array hold the string which is enter on the command line
//  argc  is count the argument
//  argc stands for argument count and argv stands for argument values
/* Main function: GLUT runs as a console application starting at main()  */
/* Program entry point */


int main (int argc, char *argv[])
{
    glutInit(&argc, argv);                  // use to initialize glut library
    glutCreateWindow("GLUT Shapes");        // Create a window with the given title
    glutInitDisplayMode(GLUT_RGB);          // glutInitDisplayMode sets the initial display mode
    glutInitWindowPosition(0,0);            // Position the window's initial top-left corner
    glutInitWindowSize(500,500);            // Set the window's initial width & height
    glutDisplayFunc(display);               // Sets the display callback for the current window
    glutMainLoop();                         // Tells the program to enter the GLUT event processing loop
    // glutMainLoop never exits, so it should always be the last line of the main program
    return 0;
}
```

```
#include<Gl/gl.h>
#include<Gl/glut.h>
#include<Gl/glu.h>
#include <windows.h>  // For MS Windows
#include <iostream>
using namespace std;
```

**/* Handler for window-repaint event. Call back when the window first appears and whenever the window needs to be re-painted. */**

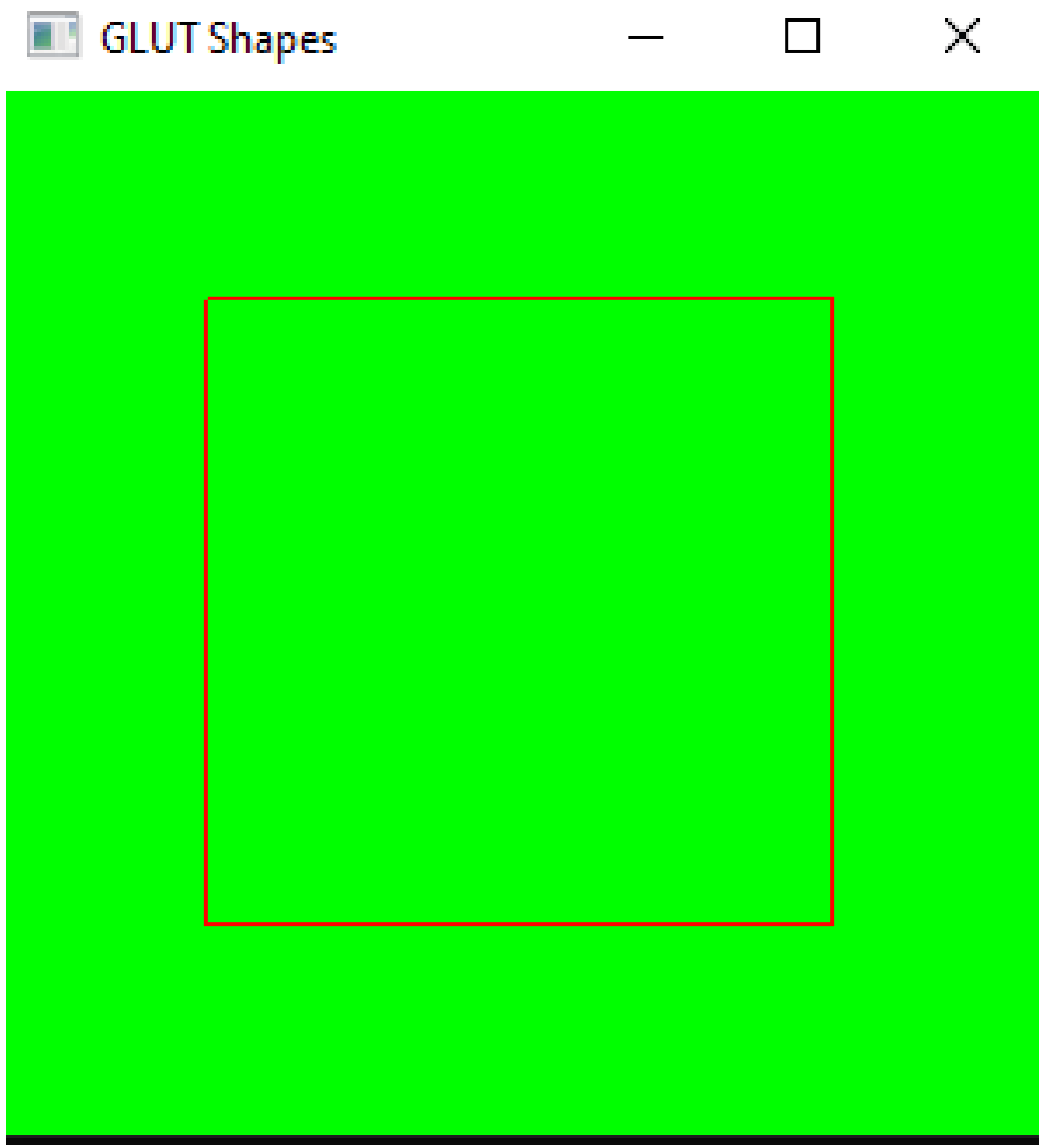
```
void display()
{
    glClearColor(0.f, 1.0f, 0.0f, 1.0f);        // Set background color to black and opaque
    glClear(GL_COLOR_BUFFER_BIT);               // Clear the color buffer
    glBegin(GL_LINES);                          //starts drawing lines
    glColor3f(1.0,0.0,0.0);                     //it can be used to give each vertex its own color.
    glVertex2f(-1.0, -1.0 );                    // red for vertixes
    glVertex2f (1.0, 1.0);                      // Points
    glEnd();                                    //end drawing of points
    glFlush();                                  // Render now
}
```


GLUT Shapes


Y axis
- +
+ +
-X axis
X axis
- -
- Y axis
+ -
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
-0.1
-0.2
-0.3
-0.4
-0.5
-0.6
-0.7
-0.8
-0.9
-1.0

# GL_LINE_LOOP

```
void display()
{
        glClearColor(0.f, 1.0f, 0.0f, 1.0f);
        glClear(GL_COLOR_BUFFER_BIT);
        glBegin(GL_LINE_LOOP);
        glColor3f(1,0,0);
        glVertex2f(-0.6, 0.6 );
        glVertex2f(0.6, 0.6 );
        glVertex2f(0.6, -0.6 );
        glVertex2f(-0.6, -0.6 );
        glEnd();          //end drawing of points
        glFlush();        // Render now
}
```



# Bresenham’s Line Drawing Algorithm

1. This algorithm was introduced by **“Jack Elton Bresenham”** in **1962.**

2. This algorithm helps us to perform scan conversion of a line.

3. It is a powerful, useful, and accurate method. We use incremental integer calculations to draw a line. The integer calculations include addition, subtraction, and multiplication.

4. In **Bresenham’s Line Drawing algorithm**, we have to calculate the slope (**m**) between the starting point and the ending point.

# Bresenham's Line Drawing Algorithm

## Procedure-

Given-
Starting coordinates = (X0, Y0)
Ending coordinates = (Xn, Yn)

The points generation using Bresenham Line Drawing Algorithm involves the following steps-

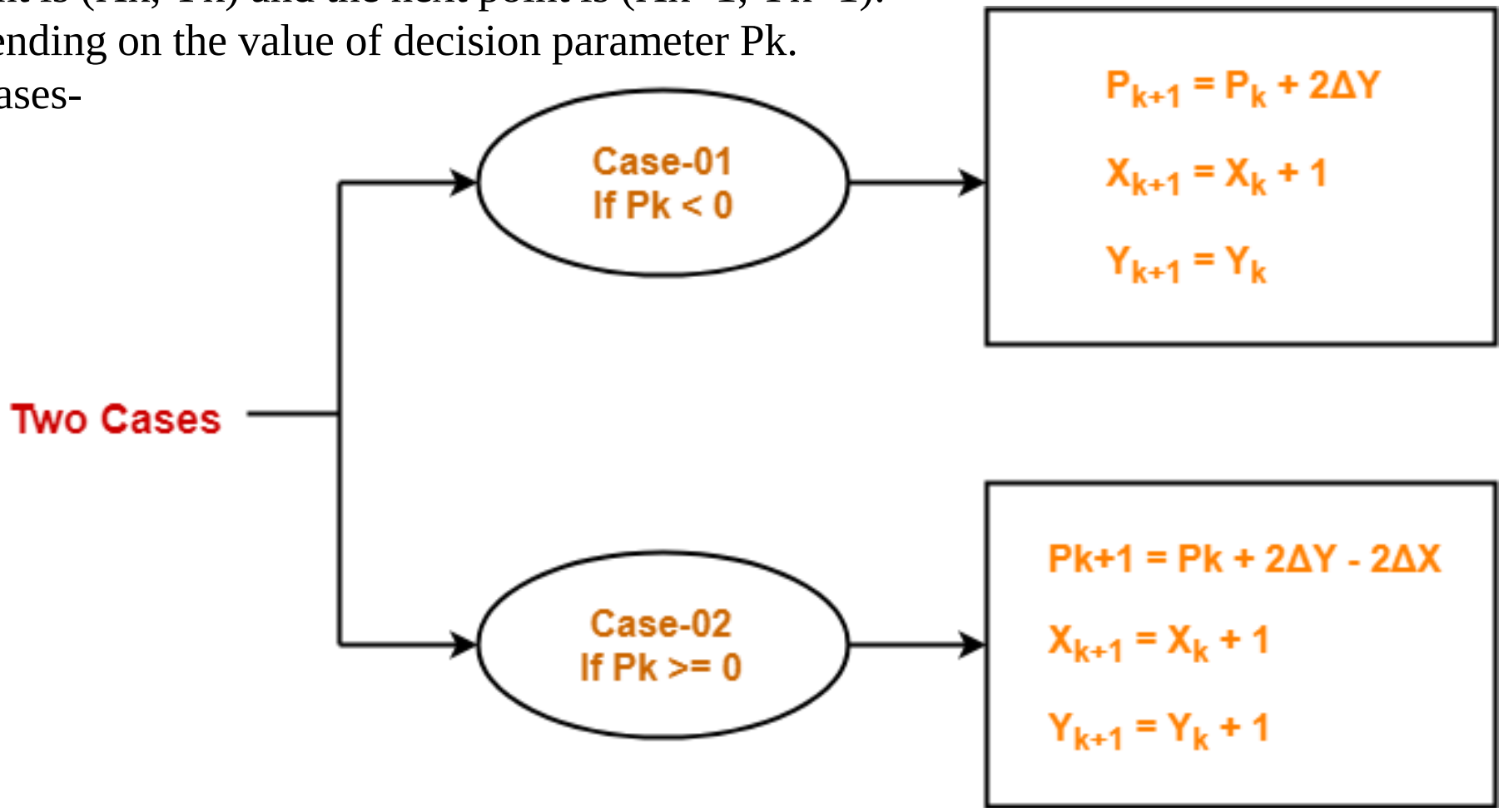
### Step-01:
Calculate ΔX and ΔY from the given input.
These parameters are calculated as-
**ΔX = Xn – X0**
**ΔY =Yn – Y0**

### Step-02:
Calculate the decision parameter Pk.
It is calculated as- **Pk = 2ΔY – ΔX**

# Bresenham's Line Drawing Algorithm

**Step-03:**

Suppose the current point is (Xk, Yk) and the next point is (Xk+1, Yk+1).
Find the next point depending on the value of decision parameter Pk.
Follow the below two cases-




**Step-04:**

Keep repeating Step-03 until the end point is reached or number of iterations equals to (ΔX-1) times.

# Bresenham's Line Drawing Algorithm

## Problem 1:

Calculate the points between the starting coordinates (9, 18) and ending coordinates (14, 22).

## Solution-

## Given-

Starting coordinates = **(X0, Y0) = (9, 18)**

Ending coordinates = **(Xn, Yn) = (14, 22)**

## Step-01:

Calculate ΔX and ΔY from the given input.

**ΔX = Xn – X0 = 14 – 9 = 5**

**ΔY =Yn – Y0 = 22 – 18 = 4**

# Bresenham's Line Drawing Algorithm

**Step-02:**

Calculate the decision parameter. Pk

**= 2ΔY – ΔX**

**= 2 x 4 – 5**

**= 3**

So, decision parameter Pk = 3

**Step-03:**

As Pk >= 0, so case-02 is satisfied. Thus,

**Pk+1 = Pk + 2ΔY – 2ΔX = 3 + (2 x 4) – (2 x 5) = 1**

**Xk+1 = Xk + 1 = 9 + 1 = 10**

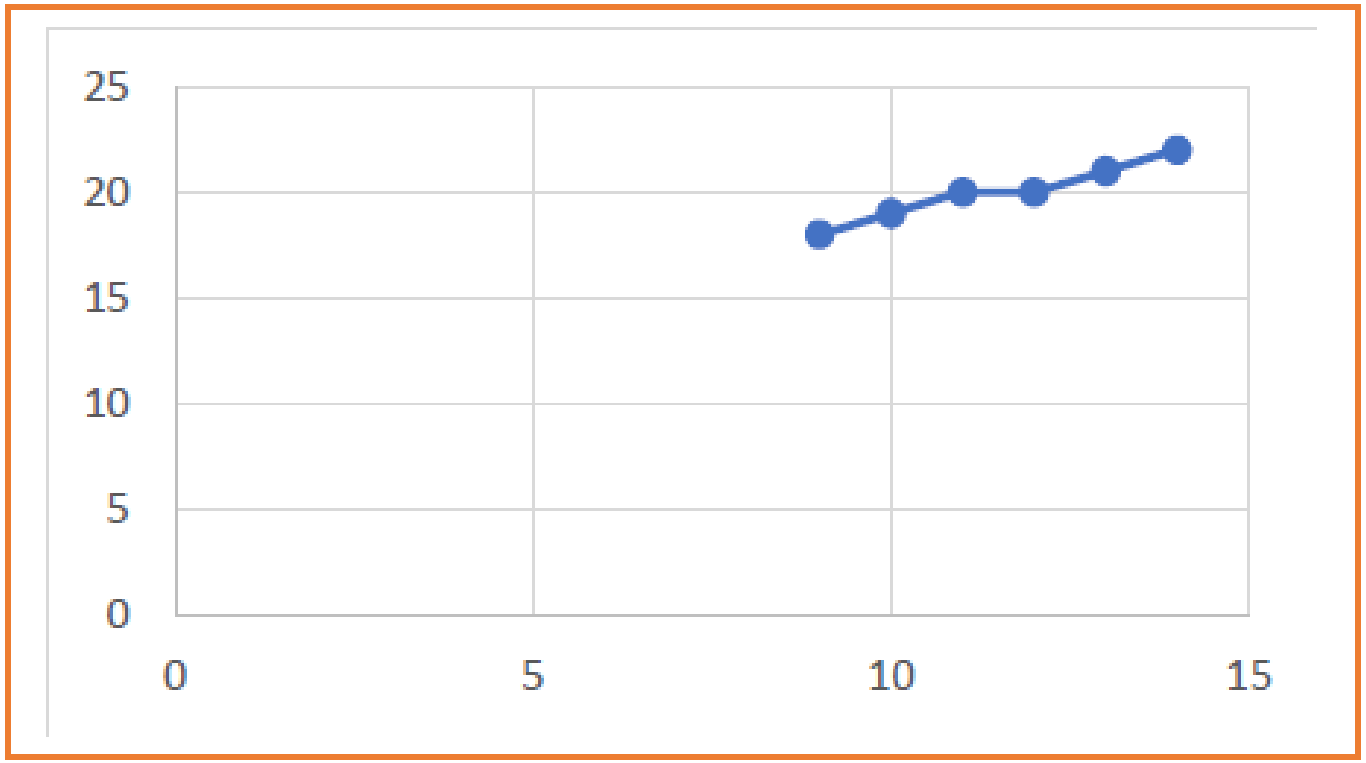
**Yk+1 = Yk + 1 = 18 + 1 = 19**

# Bresenham's Line Drawing Algorithm

Similarly, Step-03 is executed until the end point is reached or number of iterations equals to 4 times.

**(Number of iterations = ΔX – 1 = 5 – 1 = 4)**

| $P_k$ | $P_{k+1}$ | $X_{k+1}$ | $Y_{k+1}$ |
|---|---|---|---|
| | | 9 | 18 |
| 3 | 1 | 10 | 19 |
| 1 | -1 | 11 | 20 |
| -1 | 7 | 12 | 20 |
| 7 | 5 | 13 | 21 |
| 5 | 3 | 14 | 22 |



# Bresenham's Line Drawing Algorithm

## Problem-02:

Calculate the points between the starting coordinates (20, 10) and ending coordinates (30, 18).

## Solution-

## Given-

Starting coordinates = **(X0, Y0) = (20, 10)**

Ending coordinates = **(Xn, Yn) = (30, 18)**

## Step-01:

Calculate ΔX and ΔY from the given input.

**ΔX = Xn – X0 = 30 – 20 = 10**

**ΔY =Yn – Y0 = 18 – 10 = 8**

# Bresenham's Line Drawing Algorithm

**Step-02:**

Calculate the decision parameter. Pk

**= 2ΔY – ΔX**

**= 2 x 8 – 10 = 6**

So, decision parameter Pk = 6

**Step-03:**

As Pk >= 0, so case-02 is satisfied.

Thus,

**Pk+1 = Pk + 2ΔY – 2ΔX = 6 + (2 x 8) – (2 x 10) = 2**

**Xk+1 = Xk + 1 = 20 + 1 = 21**

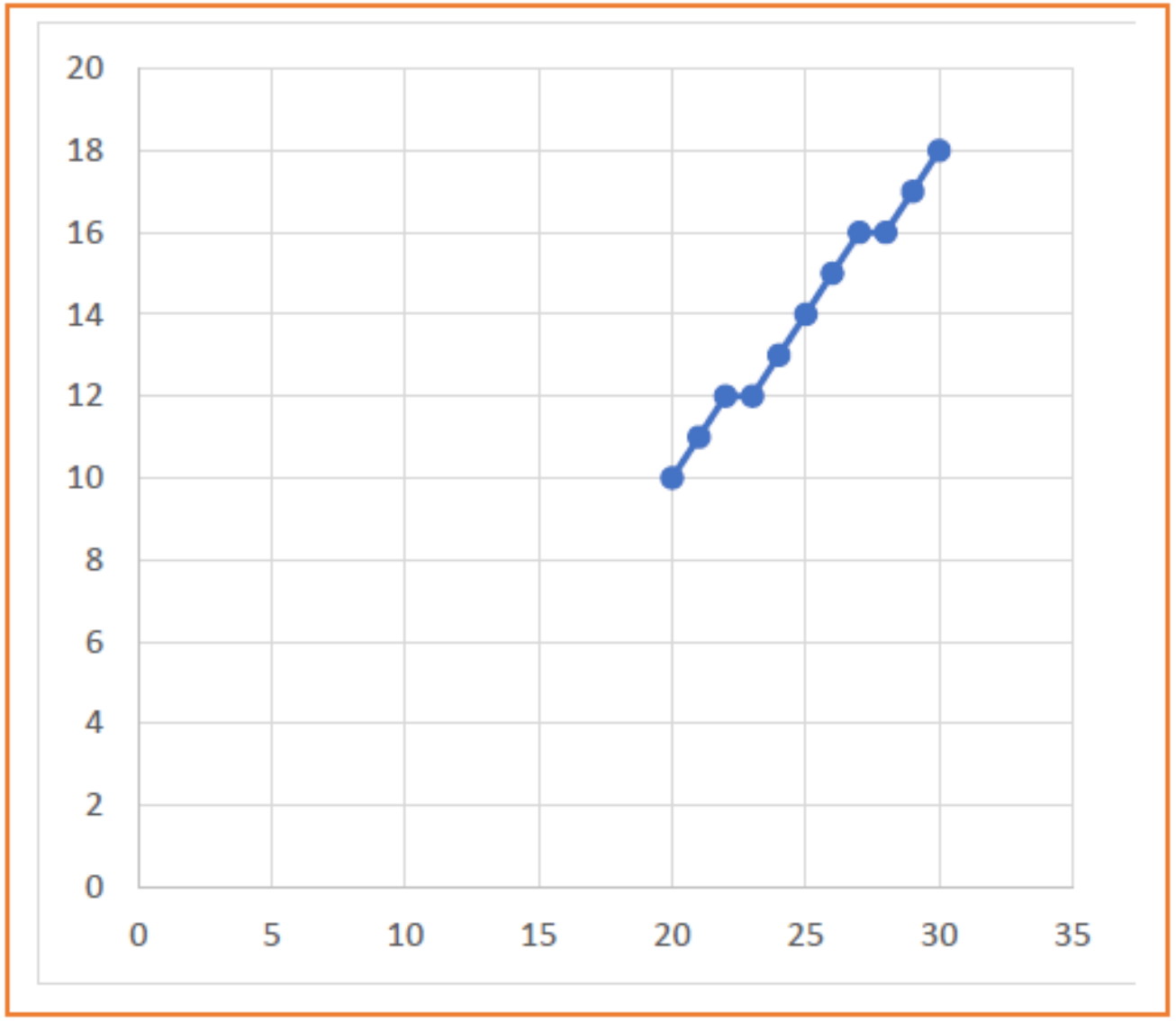
**Yk+1 = Yk + 1 = 10 + 1 = 11**

# Bresenham's Line Drawing Algorithm

Similarly, Step-03 is executed until the end point is reached or number of iterations equals to 9 times.

**(Number of iterations = ΔX – 1 = 10 – 1 = 9)**

| $P_k$ | $P_{k+1}$ | $X_{k+1}$ | $Y_{k+1}$ |
|---|---|---|---|
| | | 20 | 10 |
| 6 | 2 | 21 | 11 |
| 2 | -2 | 22 | 12 |
| -2 | 14 | 23 | 12 |
| 14 | 10 | 24 | 13 |
| 10 | 6 | 25 | 14 |
| 6 | 2 | 26 | 15 |
| 2 | -2 | 27 | 16 |
| -2 | 14 | 28 | 16 |
| 14 | 10 | 29 | 17 |
| 10 | 6 | 30 | 18 |



# DDA Line generation Algorithm

DDA (Digital Differential Analyzer) is a line drawing algorithm used in computer graphics to generate a line segment between two specified endpoints. It is a simple and efficient algorithm that works by using the incremental difference between the x-coordinates and y-coordinates of the two endpoints to plot the line.

**Remember the steps:**

- If slope (m) is less than 1 (m<1) then increment x as x1+1 and calculate y as y1=y1+m
- If slope (m) is greater than 1 (m>1) then increment y as y1+1 and calculate x1=x1+1/m
- If slope is equal to 1 (m=1) then increment both x and y. x1=x1+1 , y1=y1+1.

# DDA Line generation Algorithm

The steps involved in DDA line generation algorithm are:

S-1: Lets take the starting points of line as (x1,y1) and ending points as (x2,y2)

S-2: m=(y2-y1)/(x2-x1)

S-3: If slope (m) is less than 1 (m<1) then increment x as x1+1 and calculate y as y1=y1+m

S-4: If slope (m) is greater than 1 (m>1) then increment y as y1+1 and calculate x1=x1+1/m

S-5 : If slope is equal to 1 (m=1) then increment both x and y. x1=x1+1 , y1=y1+1

S-6 : Repeat the steps till end of line (x2,y2) is reached.

# DDA Line generation Algorithm

**Draw a line from (1,1) to (8,7) using DDA and BLA algorithms.**

---

**DDA- Digital Differential Analyser**

This case is for slope (m) less than 1. Slope (m) =(7-1)/(8-1) = 6/7 .

S-1: x1=1; y1=1; x2=8; y2=7.

S-2: m=(7-1)/(8-1) = 6/7 which is less than 1.

S-3: As m (6/7) is less than 1 therefore x is increased and y is calculated.

S-4 : The step will be x1=x1+1 and y1= y1+6/7

S-5 : The points genrated would be x1=1+1 and Y1=1+(5/7) => 1+0.9=>1.9=> approx 2. So X 1=2 and Y1= ~2

| S.No. | X1 | Y1 | Pixel Plotted |
|---|---|---|---|
| 1 | 2 | 2 | **2,2** |
| 2 | **3** | **2+6/7 = 2.9** | **3,3** |
| 3 | **4** | **2.9 + 6/7 = 3.8** | **4,4** |
| 4 | **5** | **3.8 + 6/7 = 4.7** | **5,5** |
| 5 | **6** | **4.7 + 6/7 = 5.6** | **6,6** |
| 6 | **7** | **5.6 + 6/7 = 7.0** | **7,7** |

The algorithm will stop here as the x value has reached 7.

# DDA Line generation Algorithm

**Draw a line using DDA Algorithm from (0,0) to (4,6)**

This case is for slope (m) greater than 1. Slope (m) =(6-0)/(4-0) = 6/4 .

S-1: x1=0; y1=0; x2=4; y2=6

S-2: m=(6-0)/(4-0) = 6/4 which is more than 1.

S-3: As m (6/4) is greater than 1 therefore y is increased and x is calculated.

S-4 : Now increase the value of y and calculate value of x.

- To calculate x, take line equation and find x , x2=x1+1/m
- The step will be y1=y1+1 and x1 =x1+1/( 6/4) , After Simplification, Every time y1=y1+1 and x1=x1+4/6

| Step | Y1 | X1 | Pixel Plotted |
|---|---|---|---|
| 1 | 0 | 0 | (0,0) |
| 2 | 1 | x1= (0)+4/6=0.67 =1 | (1,1) |
| 3 | 2 | 0.67+4/6 = 1.34 | (1,2) |
| 4 | 3 | 1.34+4/6=2.01 | (2,3) |
| 5 | 4 | 2.01+4/6= 2.68 | (3,4) |
| 6 | 5 | 2.68+4/6=3.35 | (3,5) |
| 7 | 6 | 3.35+4/6=4.02 | (4,6) |

The algorithm will stop here because the Y and X values have reached the End point (4,6).

# Bresenham's Circle Algorithm

**Step1:** Start Algorithm

**Step2:** Declare p, q, x, y, r, d variables p, q are coordinates of the centre of the circle r is the radius of the circle.

**Step3:** Enter the value of r

**Step4:** Calculate d = 3 - 2r

**Step5:** Initialize x=0 & y= r

**Step6:** Check if the whole circle is scan converted If x > = y Stop

# Bresenham's Circle Algorithm

**Step7:** Plot eight points by using concepts of eight-way symmetry. The center is at (p, q). Current active pixel is (x, y).

putpixel (x+p, y+q)
putpixel (y+p, x+q)
putpixel (-y+p, x+q)
putpixel (-x+p, y+q)
putpixel (-x+p, -y+q)
putpixel (-y+p, -x+q)
putpixel (y+p, -x+q)
putpixel (x+p, -y-q)

# Bresenham's Circle Algorithm

**Step8:** Find location of next pixels to be scanned

If $d < 0$

then $d = d + 4x + 6$

increment $x = x + 1$

If $d \geq 0$

then $d = d + 4 (x - y) + 10$

increment $x = x + 1$

decrement $y = y – 1$

**Step9:** Go to step 6

**Step10:** Stop Algorithm

# Bresenham's Circle Algorithm

**Example:** Plot 6 points of circle using Bresenham Algorithm. When radius of circle is 10 units. The circle has centre (50, 50).

**Solution:** Let r = 10 (Given)

**Step 1:** Take initial point (0, 10)

$d = 3 - 2r$

$d = 3 - 2 * 10 = -17$

$d < 0 \therefore d = d + 4x + 6$

$= -17 + 4 (0) + 6$

$= -11$

# Bresenham's Circle Algorithm

**Step 2:** Plot (1, 10)

$d = d + 4x + 6$ ($\because d < 0$)

$= -11 + 4(1) + 6$

$= -1$

**Step 3:** Plot (2, 10)

$d = d + 4x + 6$ ($\because d < 0$)

$= -1 + 4 \times 2 + 6$

$= 13$

# Bresenham's Circle Algorithm

**Step 4:** Plot (3, 9) d is > 0 so x = x + 1, y = y - 1

$$d = d + 4(x-y) + 10 \quad (\because d > 0)$$

$$= 13 + 4(3-9) + 10$$

$$= 13 + 4(-6) + 10$$

$$= 23-24=-1$$

**Step 5:** Plot (4, 9)

$$d = -1 + 4x + 6$$

$$= -1 + 4(4) + 6$$

$$= 21$$

# Bresenham's Circle Algorithm

**Step 6:** Plot (5, 8)

$d = d + 4 (x-y) + 10 (\because d > 0)$

$= 21 + 4 (5-8) + 10$

$= 21-12 + 10 = 19$

So P1 $(0,0) \Longrightarrow (50,50)$

P2 $(1,10) \Longrightarrow (51,60)$

P3 $(2,10) \Longrightarrow (52,60)$

P4 $(3,9) \Longrightarrow (53,59)$

P5 $(4,9) \Longrightarrow (54,59)$

P6 $(5,8) \Longrightarrow (55,58)$

## Character Generation:

a) Stroke principle,

b) starburst principle,

c) bitmap method.

## Character generation methods: stroke, starburst, bitmap.

- We can display letters and numbers in variety of size and style. The overall design style for the set of character is called typeface.
- Today large numbers of typefaces are available for computer application for example Helvetica, New York platina etc.
- Originally, the term font referred to a set of cast metal character forms in a particular size and format, such as 10-point Courier Italic or 12- point Palatino Bold.
- Now, the terms font and typeface are often used interchangeably, since printing is no longer done with cast metal forms. Two different representations are used for storing computer fonts.

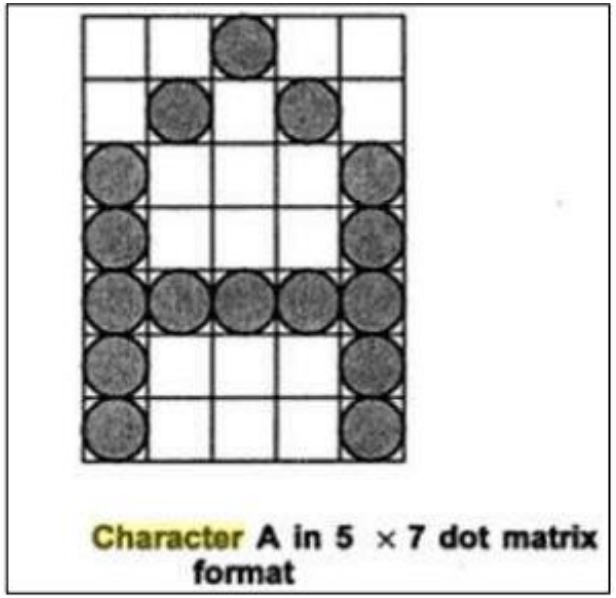
## Bitmap Font/ Bitmapped Font

- A simple method for representing the character shapes in a particular typeface is to use rectangular grid patterns Figure below shows pattern for particular letter.
- When the pattern in figure is copied to the area of frame buffer, the 1 bits designate which pixel positions are to be displayed on the monitor. Bitmap fonts are the simplest to define and display as character grid only need to be mapped to a frame- buffer position, Bitmap fonts require more space because each variation (size and format) must be stored in a font cache. It is possible to generate different size and other variation from: set but this usually does not. produce good result.

# Bitmap Font/ Bitmapped Font

| | | | | | | |
|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 0 | 1 | 1 | 1 | 1 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 | 0 |

Fig. : - Grid pattern for letter B.

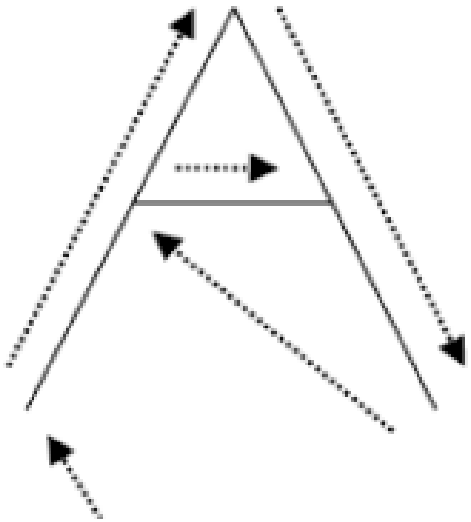


Character A in 5 × 7 dot matrix format

## Stroke Method:

- It uses small line segments to generate a character. The small series of line segments are drawn like a stroke of a pen to form a character as shown in figure.
- We can generate our own stroke method by calling line drawing algorithm. Here it is necessary to decide which line segments are needs for each character and then draw that line to display character.
- It support scaling by changing length of line segment.

# Stroke Method:




∴ - Stroke Method for Letter A.

## Starbust Method:

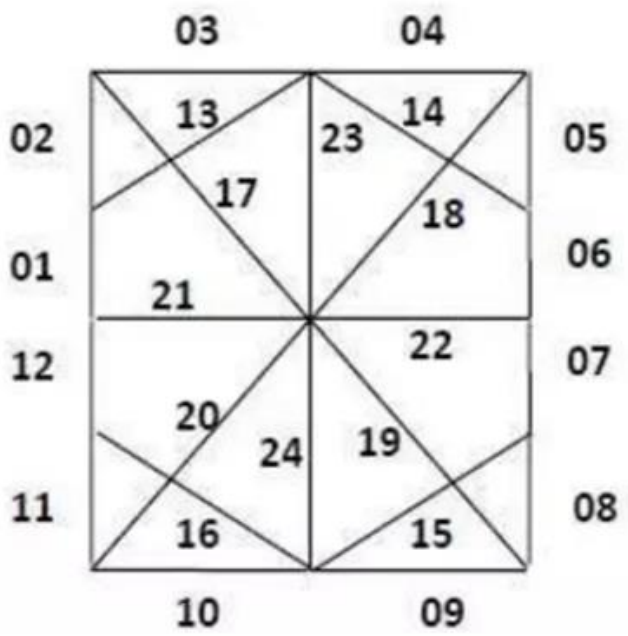
- In this method a fix pattern of line segments are used to generate characters.
- As shown in figure there are 24 line segments.
- We highlight those lines which are necessary to draw a particular character.
- Pattern for particular character is stored in the form of 24 bit code.
- In which each bit represents corresponding line having that number.
- That code contains 0 or 1 based on line segment need to highlight.
- We put bit value 1 for highlighted line and 0 for other line.

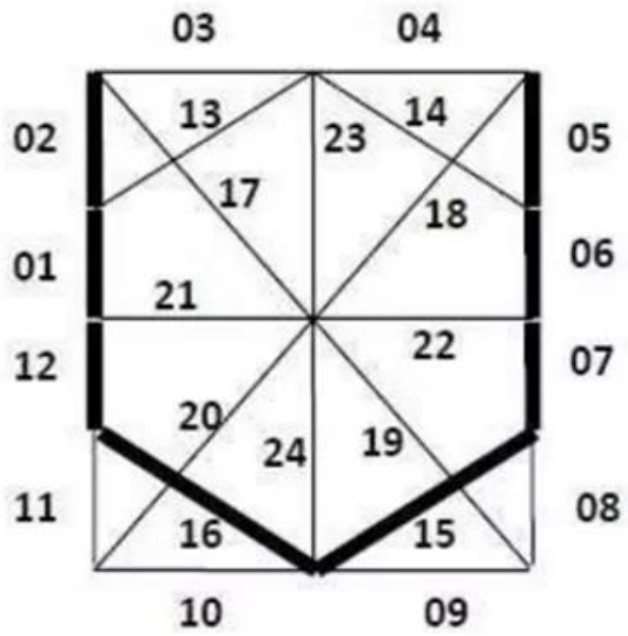
## Starbust Method:

- This technique is not used now a days because:
  i. It requires more memory to store 24 bit code for single character.
  ii. It requires conversion from code to character.
  iii. It does not provide the curve shape.

# Starbust Method:

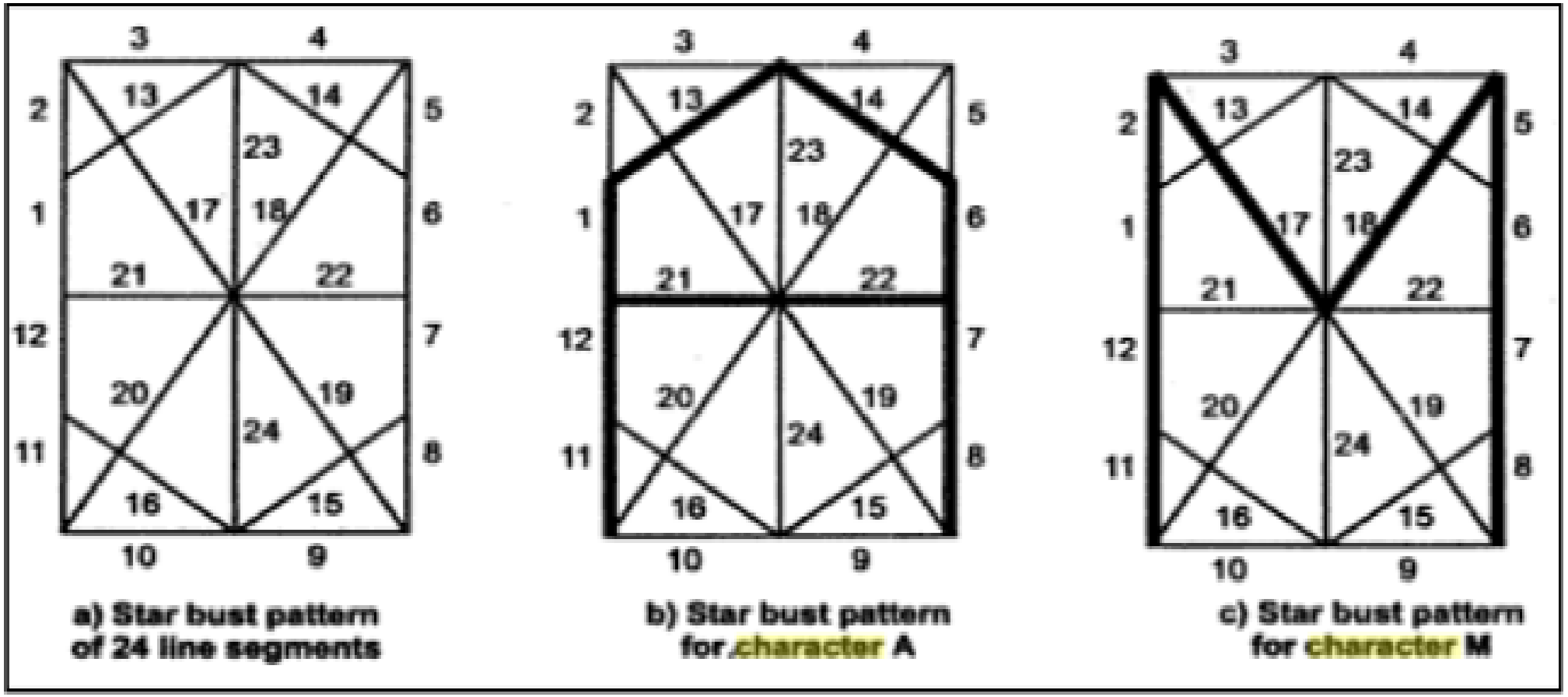
Code for letter V is 110011100001001100000000




| 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |

24 bit line segment code




Character A : 0011 0000 0011 1100 1110 0001

Character M:0000 0011 0000 1100 1111 0011

# Introduction to aliasing and anti-aliasing:

- Antialiasing is a computer graphics method that removes the aliasing effect.
- The aliasing effect occurs when rasterised images have jagged edges, sometimes called "jaggies" (an image rendered using pixels).
- Technically, jagged edges are a problem that arises when scan conversion is done with low-frequency sampling, also known as under-sampling, this under-sampling causes distortion of the image.
- Moreover, when real-world objects made of continuous, smooth curves are rasterised using pixels, aliasing occurs.
- Under-sampling is an important factor in anti-aliasing. The information in the image is lost when the sample size is too small.

- When sampling is done at a frequency lower than the Nyquist sampling frequency, under-sampling takes place. We must have a sampling frequency that is at least two times higher than the highest frequency appearing in the image in order to prevent this loss.

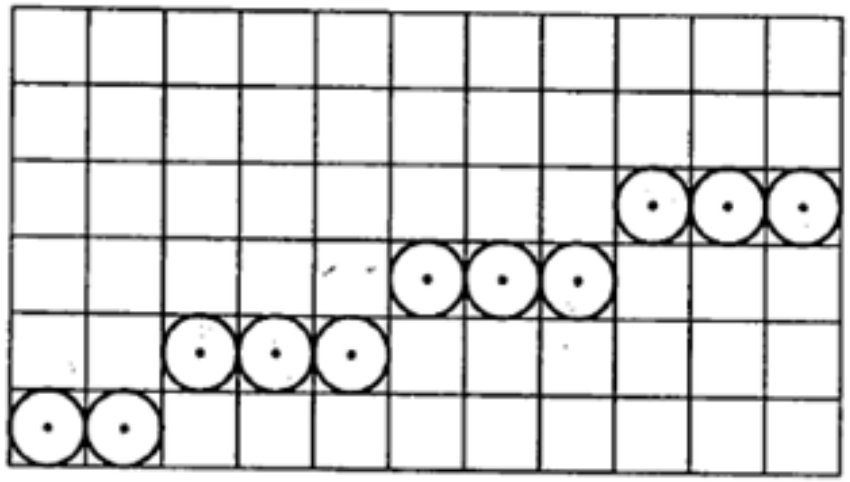
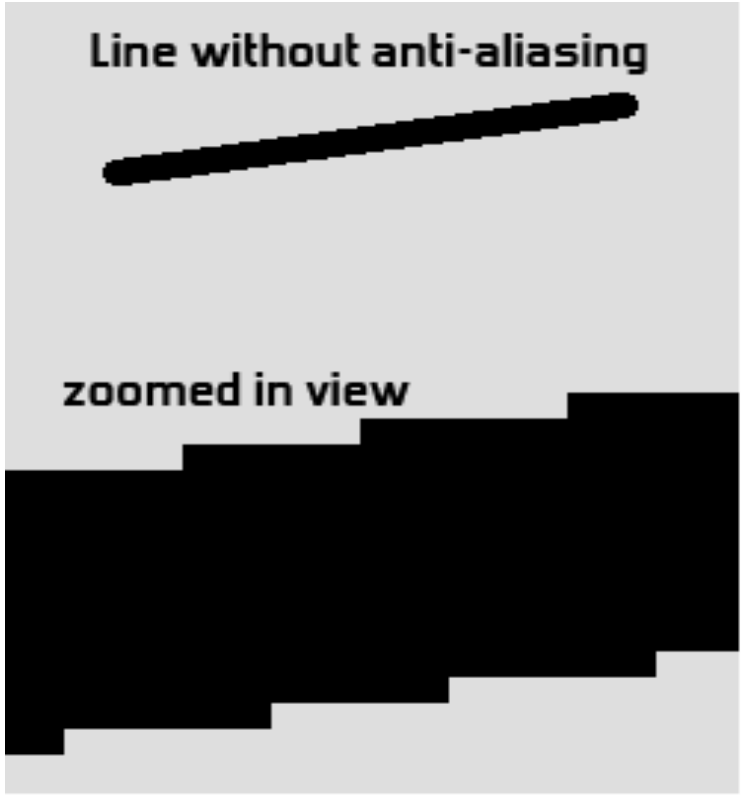
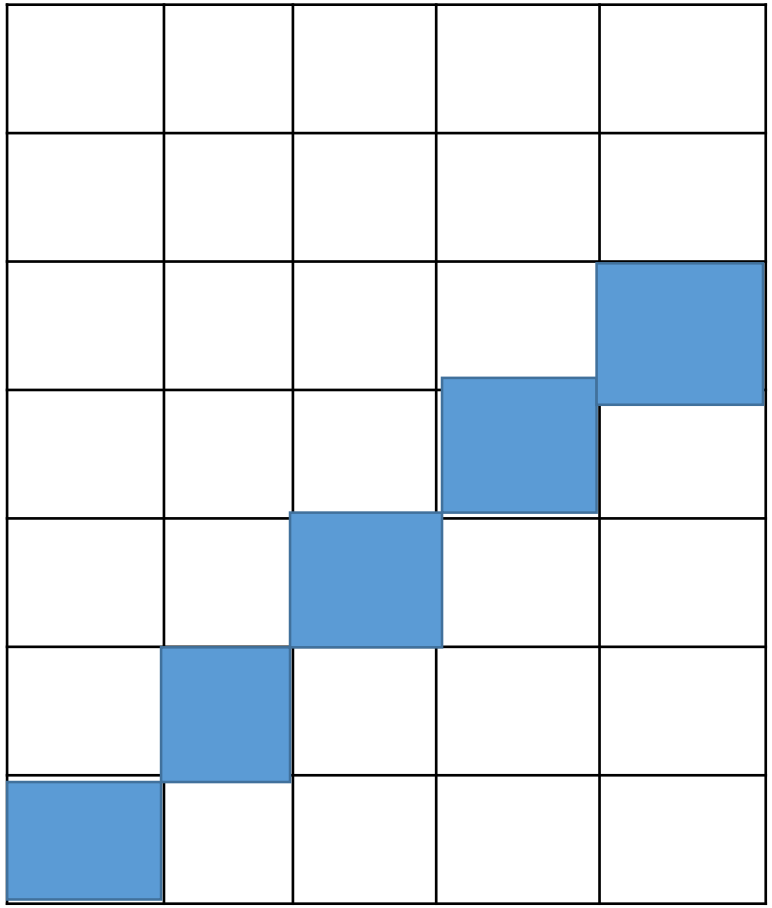


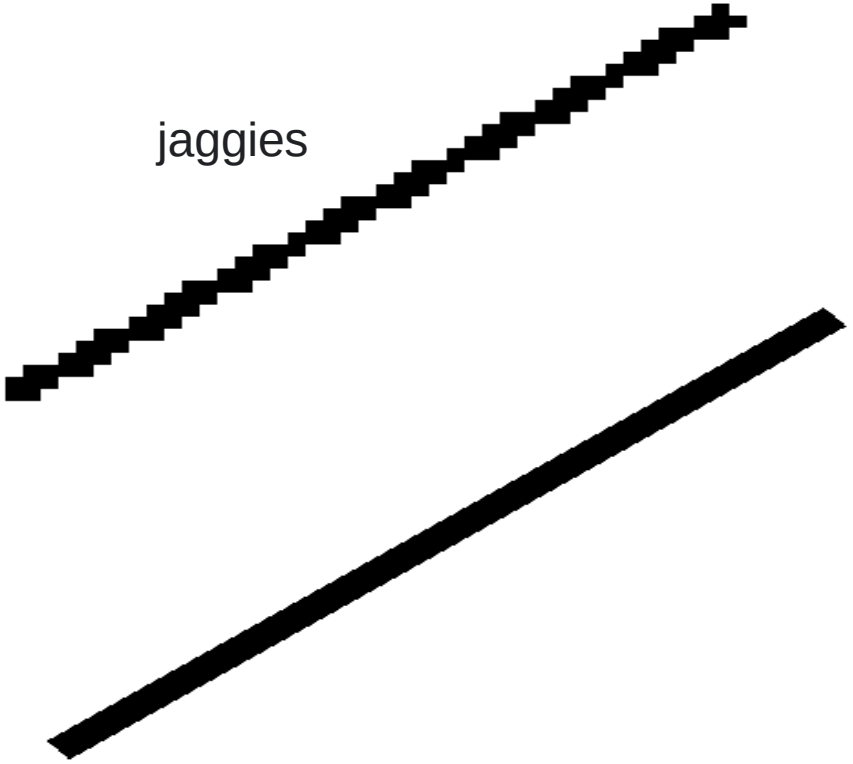
Fig. Aliasing effect

# Stair- case

- stair-stepped appearance of diagonal lines when there are not enough pixels in the image or on screen to represent them realistically. Also called "stair-stepping" and "jaggies





# Anti-Aliasing Methods:

- **Using High-Resolution Display -** Displaying objects at a greater resolution is one technique to decrease aliasing impact and boost the sampling rate. When using high resolution, the jaggies are reduced to a size that renders them invisible to the human eye. As a result, sharp edges get blurred and appear smooth.
- **Real-Life Applications:** For example, OLED displays and retina displays in Apple products both have high pixel densities, which results in jaggies that are so microscopic that they are blurry and invisible to the human eye.

# Using High-Resolution Display

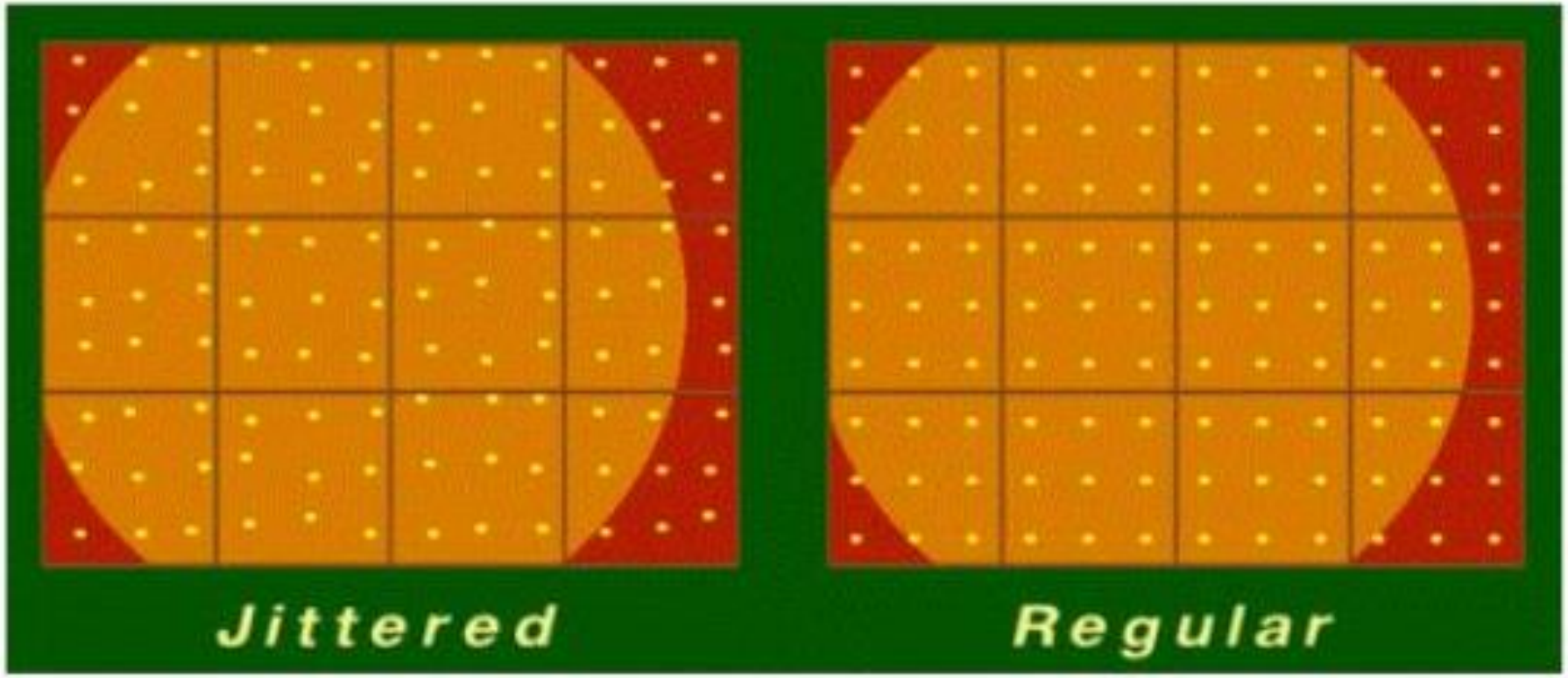
## Anti-Aliasing Methods:

- **Post-Filtering or Super-Sampling -** With this technique, we reduce the adequate pixel size while improving the sampling resolution by treating the screen as though it were formed of a much finer grid. The screen resolution, however, does not change. Now, the average pixel intensity is determined from the average of the intensities of the subpixels after each subpixel's intensity has been calculated. In order to display the image at a lesser resolution or screen resolution, we do sampling at a higher resolution, a process known as supersampling. Due to the fact that this process is carried out after creating the rasterised image, this technique is also known as post filtration.

# Anti-Aliasing Methods:

- **Real-Life Applications:** The finest image quality in gaming is produced with SSAA (Super-sample Antialiasing) or FSAA (full-scene Antialiasing). It is frequently referred to as the "pure AA," which is extremely slow and expensive to compute. When no better AA techniques were available, this technique was frequently utilised in the beginning. Other SSAA modes are available, including 2X, 4X, 8X, and others that indicate sampling that is done x times (greater than) the present resolution. MSAA (multisampling Antialiasing), a quicker and more accurate version of super-sampling AA, is a better AA type. Its computational cost is lower. Companies that produce graphics cards, such as CSAA by NVIDIA and CFAA by AMD, are working to improve and advance super-sampling techniques.

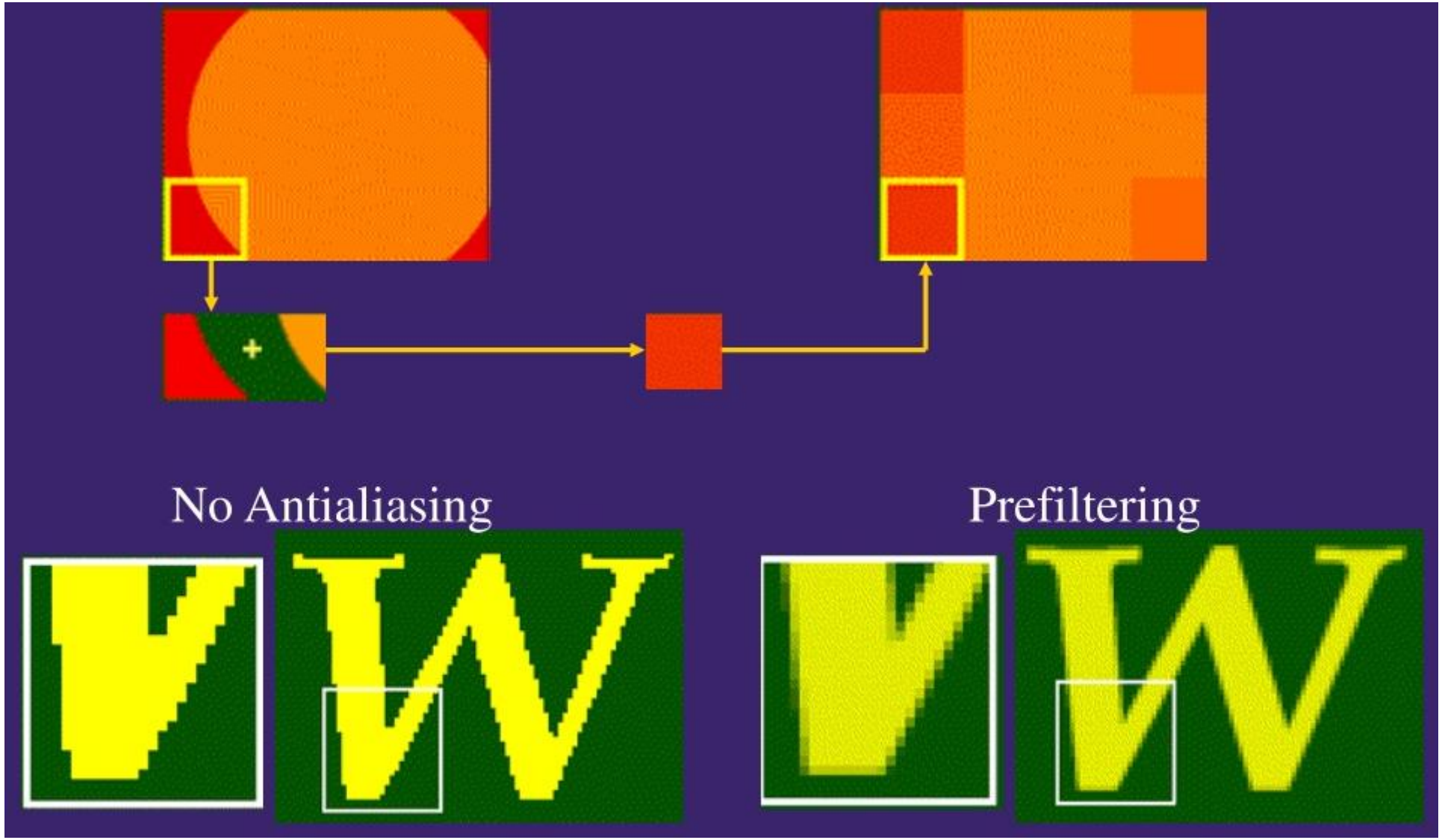
# Post-Filtering or Super-Sampling



## Anti-Aliasing Methods:

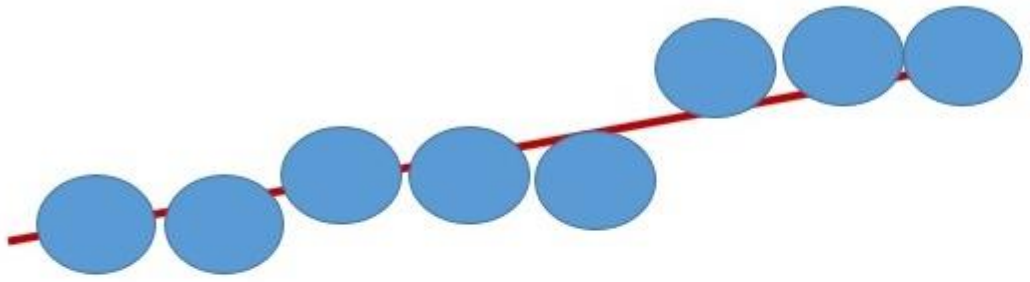
- **Pre-Filtering or Area-Sampling -** The areas of each pixel's overlap with the objects displayed are taken into account while calculating pixel intensities in area sampling. In this case, the computation of pixel colour is centred on the overlap of scene objects with a pixel region.
- **Example:** Let's say a line crosses two pixels. A pixel that covers a larger amount of a line (90%) displays 90% intensity, whereas a pixel that covers a smaller piece (10%) displays 10-15% intensity. If a pixel region overlaps with multiple colour areas, the final pixel colour is calculated as the average of those colours. Pre-filtering is another name for this technique because it is used before rasterising the image. Some basic graphics algorithms are used to complete it.

# Pre-Filtering or Area-Sampling

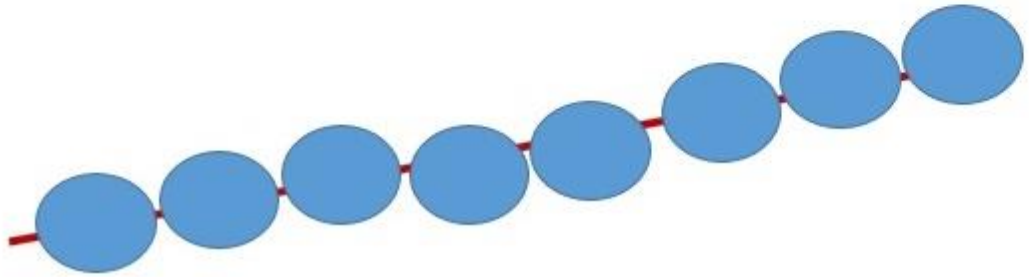


# Anti-Aliasing Methods:

- **Pixel Phasing -** It is a method to eliminate aliasing. In this case, pixel coordinates are altered to virtually exact positions close to object geometry. For dispersing intensities and aiding with pixel phasing, some systems let you change the size of individual pixels.



Before Pixel Phasing



After Pixel Phasing