Computer graphics and visualization (18CS62)

1. Overview: Computer Graphics and OpenGL

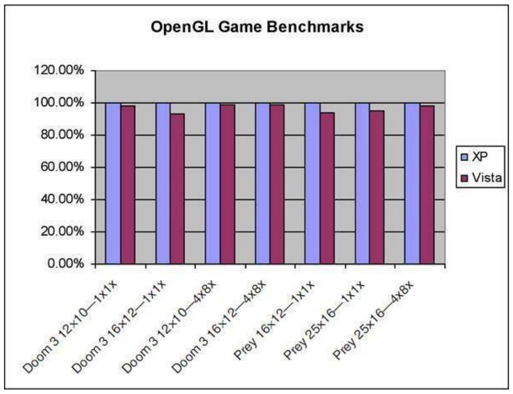
Basics of computer graphics
Application of Computer Graphics,
Video Display Devices
Random Scan and Raster Scan displays,
Graphics software.
OpenGL:
Introduction to OpenGL ,
- **Coordinate reference frames,**
- **Specifying two-dimensional world coordinate reference frames in OpenGL,OpenGL point functions,**
- **OpenGL line functions, point attributes,Line attributes,**
- **Curve attributes,**
- **OpenGL point attribute functions,OpenGL line attribute functions,**
- **Line drawing algorithms(DDA, Bresenham’s),Circle generation algorithms (Bresenham’s).**

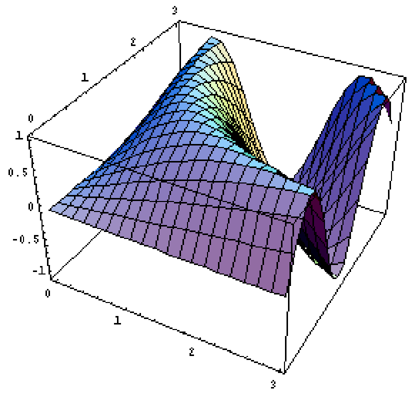
1.1 Basics of Computer Graphics

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. Pixel is the smallest addressable graphical unit represented on the computer screen.

1.2 Applications of Computer Graphics

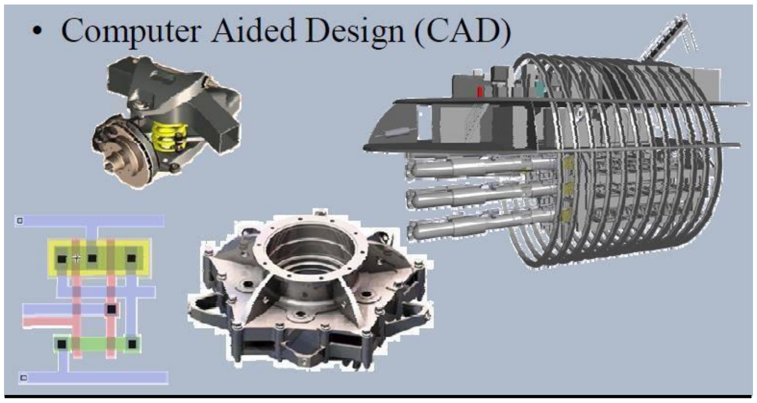
a. Graphs and Charts





- ✓ An early application for computer graphics is the display of simple data graphs usually plotted on a character printer. Data plotting is still one of the most common graphics application.
- ✓ Graphs & charts are commonly used to summarize functional, statistical, mathematical, engineering and economic data for research reports, managerial summaries and other types of publications.
- ✓ Typically examples of data plots are line graphs, bar charts, pie charts, surface graphs, contour plots and other displays showing relationships between multiple parameters in two dimensions, three dimensions, or higher-dimensional spaces

b. Computer-Aided Design



- ✓ A major use of computer graphics is in design processes-particularly for engineering and architectural systems.

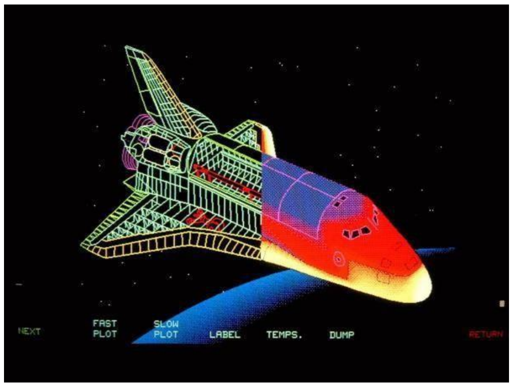
- ✓ CAD, computer-aided design or CADD, computer-aided drafting and design methods are now routinely used in the automobiles, aircraft, spacecraft, computers, home appliances.
- ✓ Circuits and networks for communications, water supply or other utilities are constructed with repeated placement of a few geographical shapes.
- ✓ Animations are often used in CAD applications. Real-time, computer animations using wire-frame shapes are useful for quickly testing the performance of a vehicle or system.

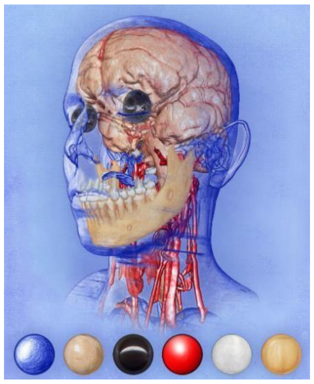
c. Virtual-Reality Environments

- ✓ 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.
- ✓ With virtual-reality systems, designers and others can move about and interact with objects in various ways. Architectural designs can be examined by taking simulated “walk” through the rooms or around the outsides of buildings to better appreciate the overall effect of a particular design.
- ✓ With a special glove, we can even “grasp” objects in a scene and turn them over or move them from one place to another.

d. Data Visualizations

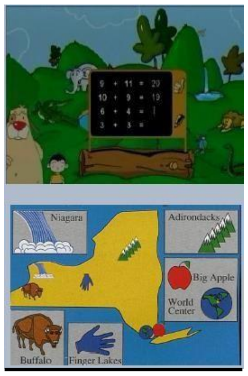
- ✓ Producing graphical representations for scientific, engineering and medical data sets and processes is another fairly new application of computer graphics, which is generally referred to as scientific visualization. And the term business visualization is used in connection with data sets related to commerce, industry and other nonscientific areas.





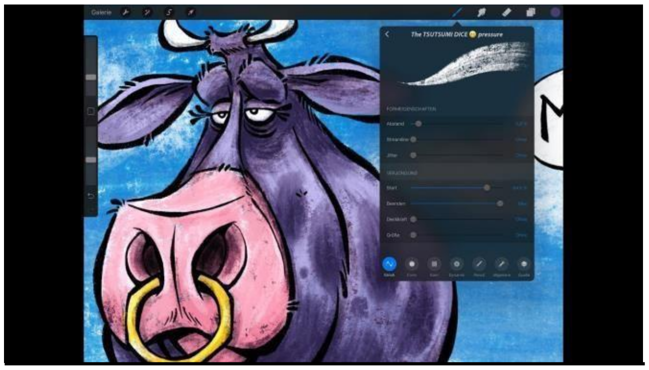
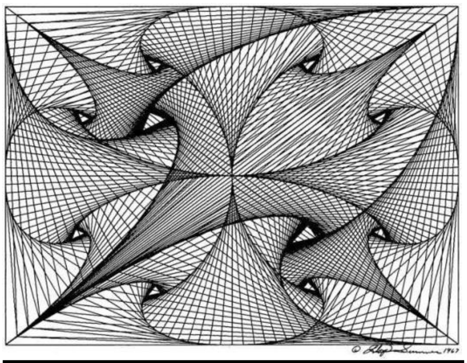
- ✓ There are many different kinds of data sets and effective visualization schemes depend on the characteristics of the data. A collection of data can contain scalar values, vectors or higher-order tensors.

e. Education and Training



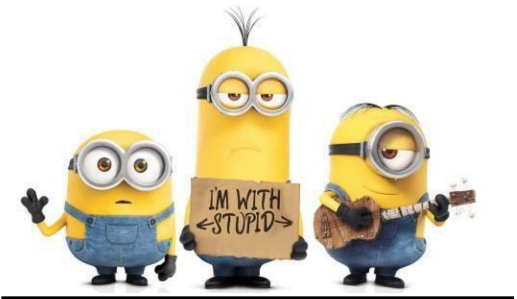
- ✓ Computer generated models of physical,financial,political,social,economic & other systems are often used as educational aids.
- ✓ Models of physical processes physiological functions,equipment, such as the color coded diagram as shown in the figure, can help trainees to understand the operation of a system.
- ✓ For some training applications,special hardware systems are designed.Examples of such specialized systems are the simulators for practice sessions ,aircraft pilots,air traffic-control personnel.
- ✓ Some simulators have no video screens,for eg: flight simulator with only a control panel for instrument flying

f. Computer Art

- ✓ The picture is usually painted electronically on a graphics tablet using a stylus, which can simulate different brush strokes, brush widths and colors.
- ✓ Fine artists use a variety of other computer technologies to produce images. To create pictures the artist uses a combination of 3D modeling packages, texture mapping, drawing programs and CAD software etc.
- ✓ Commercial art also uses theses "painting" techniques for generating logos & other designs, page layouts combining text & graphics, TV advertising spots & other applications.
- ✓ A common graphics method employed in many television commercials is morphing, where one object is transformed into another.

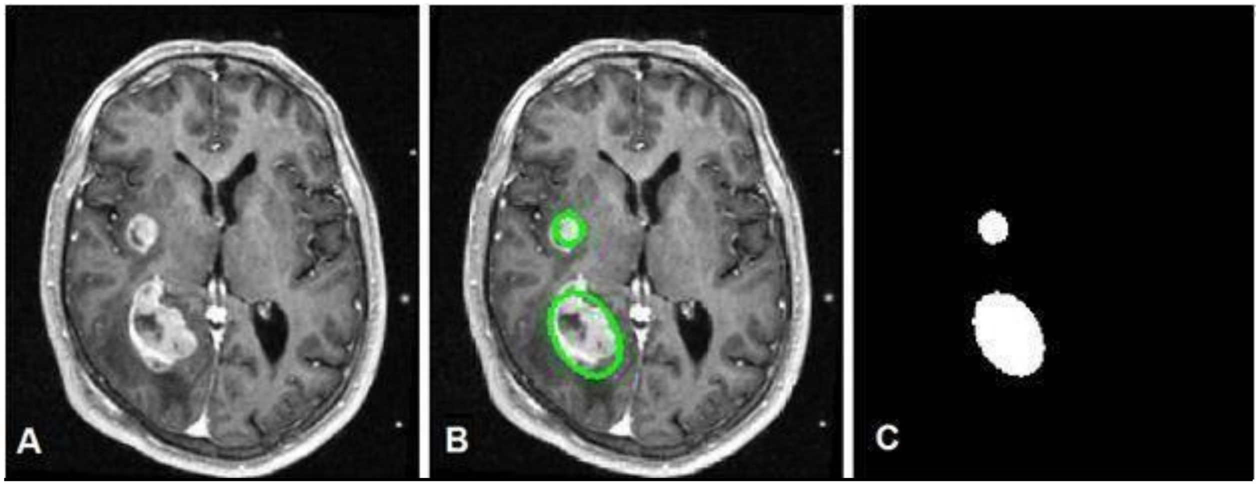
g. Entertainment



- ✓ Television production, motion pictures, and music videos routinely a computer graphics methods.
- ✓ Sometimes graphics images are combined a live actors and scenes and sometimes the films are completely generated a computer rendering and animation techniques.

- ✓ Some television programs also use animation techniques to combine computer generated figures of people, animals, or cartoon characters with the actor in a scene or to transform an actor's face into another shape.

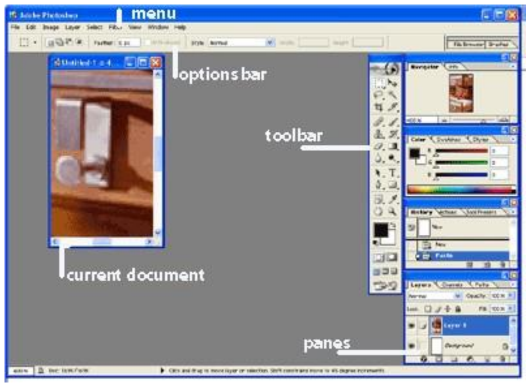
h. Image Processing

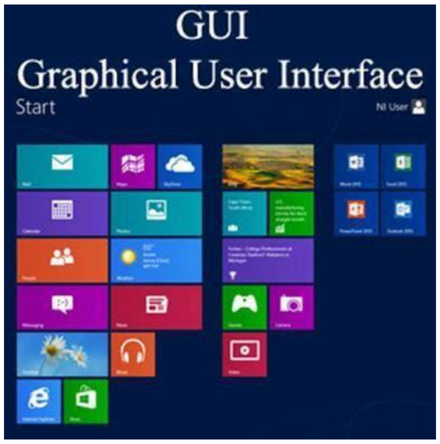


- ✓ The modification or interpretation of existing pictures, such as photographs and TV scans is called image processing.
- ✓ Methods used in computer graphics and image processing overlap, the two areas are concerned with fundamentally different operations.
- ✓ Image processing methods are used to improve picture quality, analyze images, or recognize visual patterns for robotics applications.
- ✓ Image processing methods are often used in computer graphics, and computer graphics methods are frequently applied in image processing.
- ✓ Medical applications also make extensive use of image processing techniques for picture enhancements in tomography and in simulations and surgical operations.
- ✓ It is also used in computed X-ray tomography(CT), position emission tomography(PET),and computed axial tomography(CAT).

i. Graphical User Interfaces

- ✓ It is common now for applications software to provide graphical user interface (GUI).
- ✓ A major component of graphical interface is a window manager that allows a user to display multiple, rectangular screen areas called display windows.



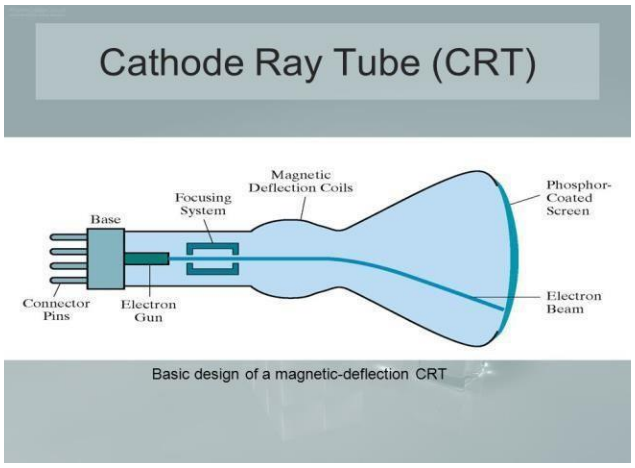


- ✓ Each screen display area can contain a different process, showing graphical or non-graphical information, and various methods can be used to activate a display window.
- ✓ Using an interactive pointing device, such as mouse, we can active a display window on some systems by positioning the screen cursor within the window display area and pressing the left mouse button.

Video Display Devices

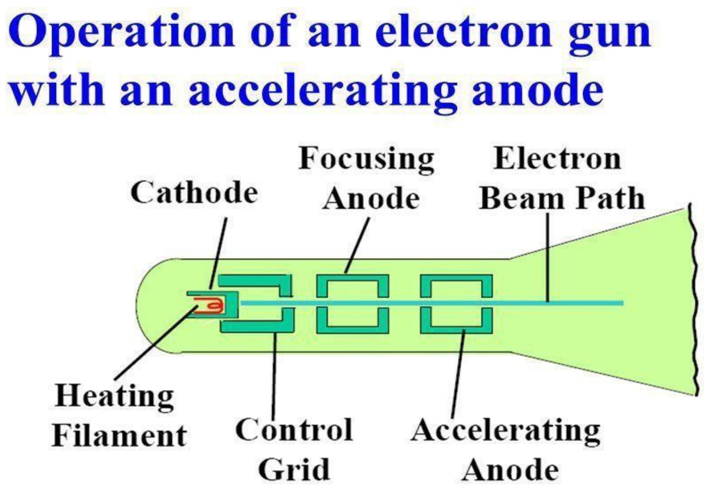
- ✓ The primary output device in a graphics system is a video monitor.
- ✓ Historically, the operation of most video monitors was based on the standard cathoderay tube (CRT) design, but several other technologies exist.
- ✓ In recent years, flat-panel displays have become significantly more popular due to their reduced power consumption and thinner designs.

Refresh Cathode-Ray Tubes



- ✓ A beam of electrons, emitted by an electron gun, passes through focusing and deflection systems that direct the beam toward specified positions on the phosphor-coated screen.
- ✓ The phosphor then emits a small spot of light at each position contacted by the electron beam and the light emitted by the phosphor fades very rapidly.
- ✓ One way to maintain the screen picture is to store the picture information as a charge distribution within the CRT in order to keep the phosphors activated.
- ✓ The most common method now employed for maintaining phosphor glow is to redraw the picture repeatedly by quickly directing the electron beam back over the same screen points. This type of display is called a refresh CRT.
- ✓ The frequency at which a picture is redrawn on the screen is referred to as the refresh rate.

<u>Operation of an electron gun with an accelarating anode</u>



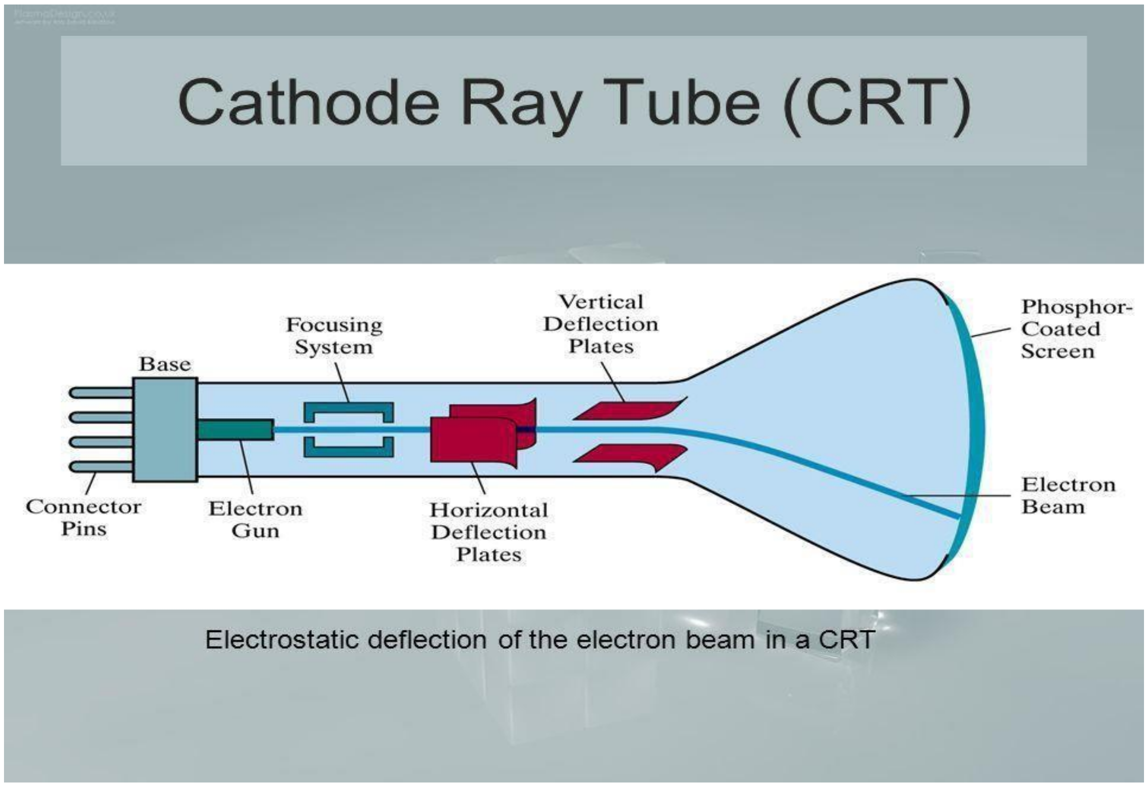
- ✓ The primary components of an electron gun in a CRT are the heated metal cathode and a control grid.
- ✓ The heat is supplied to the cathode by directing a current through a coil of wire, called the filament, inside the cylindrical cathode structure.
- ✓ This causes electrons to be “boiled off” the hot cathode surface.
- ✓ Inside the CRT envelope, the free, negatively charged electrons are then accelerated toward the phosphor coating by a high positive voltage.

- ✓ Intensity of the electron beam is controlled by the voltage at the control grid.
- ✓ Since the amount of light emitted by the phosphor coating depends on the number of electrons striking the screen, the brightness of a display point is controlled by varying the voltage on the control grid.
- ✓ The focusing system in a CRT forces the electron beam to converge to a small cross section as it strikes the phosphor and it is accomplished with either electric or magnetic fields.
- ✓ With electrostatic focusing, the electron beam is passed through a positively charged metal cylinder so that electrons along the center line of the cylinder are in equilibrium position.
- ✓ Deflection of the electron beam can be controlled with either electric or magnetic fields.
- ✓ Cathode-ray tubes are commonly constructed with two pairs of magnetic-deflection coils
- ✓ One pair is mounted on the top and bottom of the CRT neck, and the other pair is mounted on opposite sides of the neck.
- ✓ The magnetic field produced by each pair of coils results in a traverse deflection force that is perpendicular to both the direction of the magnetic field and the direction of travel of the electron beam.
- ✓ Horizontal and vertical deflections are accomplished with these pair of coils

Electrostatic deflection of the electron beam in a CRT

- ✓ When electrostatic deflection is used, two pairs of parallel plates are mounted inside the CRT envelope where, one pair of plates is mounted horizontally to control vertical deflection, and the other pair is mounted vertically to control horizontal deflection.
- ✓ Spots of light are produced on the screen by the transfer of the CRT beam energy to the phosphor.
- ✓ When the electrons in the beam collide with the phosphor coating, they are stopped and their kinetic energy is absorbed by the phosphor.
- ✓ Part of the beam energy is converted by the friction in to the heat energy, and the remainder causes electros in the phosphor atoms to move up to higher quantum-energy levels.

- ✓ After a short time, the "excited" phosphor electrons begin dropping back to their stable ground state, giving up their extra energy as small quantum of light energy called photons.



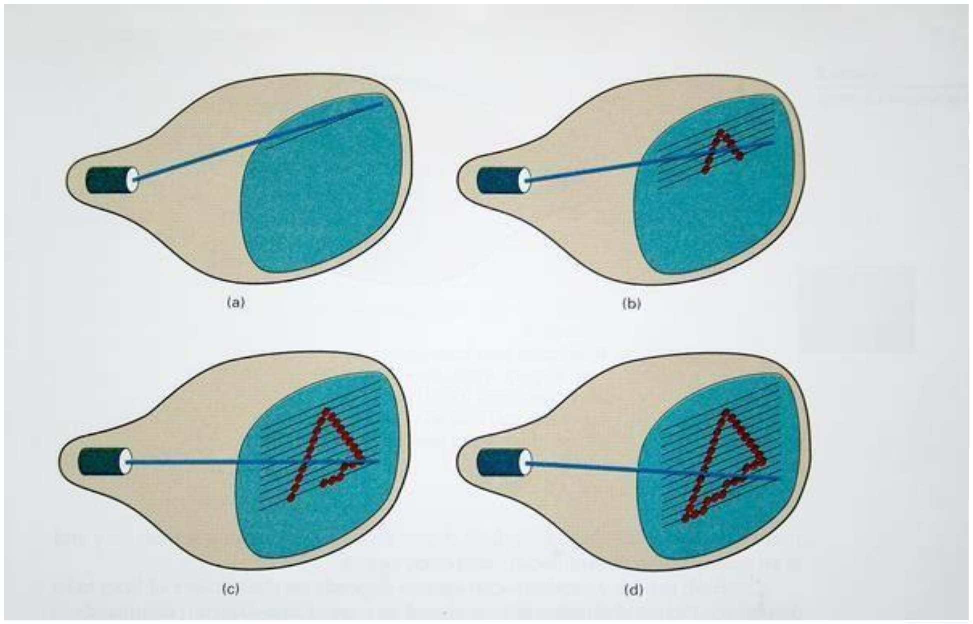
Electrostatic deflection of the electron beam in a CRT

- ✓ What we see on the screen is the combined effect of all the electrons light emissions: a glowing spot that quickly fades after all the excited phosphor electrons have returned to their ground energy level.
- ✓ The frequency of the light emitted by the phosphor is proportional to the energy difference between the excited quantum state and the ground state.
- ✓ Lower persistence phosphors required higher refresh rates to maintain a picture on the screen without flicker.
- ✓ The maximum number of points that can be displayed without overlap on a CRT is referred to as a resolution.
- ✓ Resolution of a CRT is dependent on the type of phosphor, the intensity to be displayed, and the focusing and deflection systems.
- ✓ High-resolution systems are often referred to as high-definition systems.

1.3.1 Raster-Scan Displays and Random Scan Displays

i) 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.
- ❖ The refreshing rate, called the frame rate, is normally 60 to 80 frames per second, or described as 60 Hz to 80 Hz.
- ❖ Picture definition is stored in a memory area called the frame buffer.
- ❖ This frame buffer stores the intensity values for all the screen points. Each screen point is called a pixel (picture element).
- ❖ Property of raster scan is Aspect ratio, which defined as number of pixel columns divided by number of scan lines that can be displayed by the system.



Case 1: In case of black and white systems

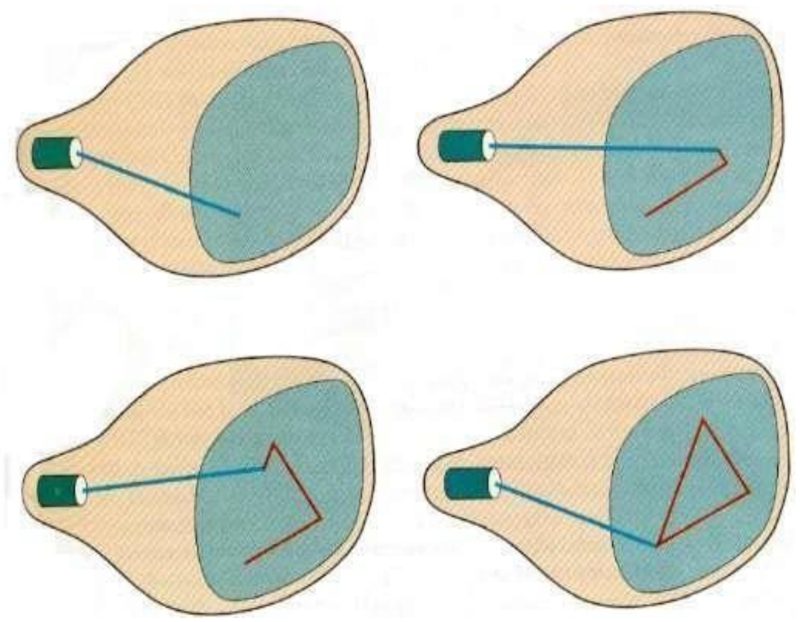
- ✓ On black and white systems, the frame buffer storing the values of the pixels is called a bitmap.
- ✓ Each entry in the bitmap is a 1-bit data which determine the on (1) and off (0) of the intensity of the pixel.

Case 2: In case of color systems

- ❖ On color systems, the frame buffer storing the values of the pixels is called a pixmap (Though now a days many graphics libraries name it as bitmap too).
- ❖ Each entry in the pixmap occupies a number of bits to represent the color of the pixel. For a true color display, the number of bits for each entry is 24 (8 bits per red/green/blue channel, each channel 28=256 levels of intensity value, ie. 256 voltage settings for each of the red/green/blue electron guns).

ii). Random-Scan Displays

- ✓ When operated as a random-scan display unit, a CRT has the electron beam directed only to 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 (or strokewriting displays or calligraphic 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 number of lines to be displayed on that system.
- ✓ Picture definition is now stored as a set of line-drawing commands in an area of memory referred to as the display list, refresh display file, vector file, or display program
- ✓ To display a specified picture, the system cycles through the set of commands in the display file, drawing each component line in turn.
- ✓ After all line-drawing commands have been processed, the system cycles back to the first line command in the list.
- ✓ Random-scan displays are designed to draw all the component lines of a picture 30 to 60 times each second, with up to 100,000 "short" lines in the display list.
- ✓ When a small set of lines is to be displayed, each refresh cycle is delayed to avoid very high refresh rates, which could burn out the phosphor.

<u>Difference between Raster scan system and Random scan system</u>

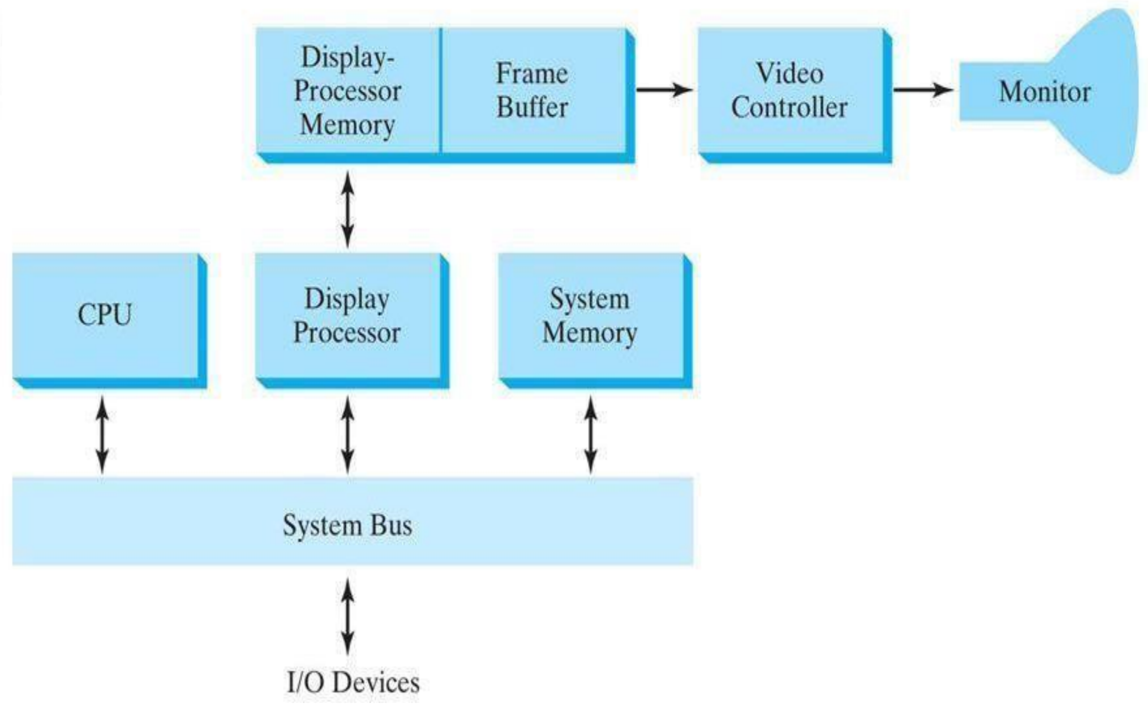
Base of Difference	Raster Scan System	Random Scan System
Electron Beam	The electron beam is swept across the screen, one row at a time, from top to bottom	The electron beam is directed only to theparts of screen where a picture is to be drawn
Resolution	Its resolution is poor because raster system in 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	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.
Realistic Display	The capability of this system to store intensity values for pixel makes it well suited for the realistic display of scenes	These systems are designed for line-drawing and can't display realistic shaded scenes.

	contain shadow and color pattern.	
Draw an Image	Screen points/pixels are used to draw an image	Mathematical functions re used to draw an image

- ✓ Finally, some systems are designed to allow the video controller to mix the framebuffer image with an input image from a television camera or other input device

<u>b) Raster-Scan Display Processor</u>

- ✓ Figure shows one way to organize the components of a raster system that contains a separate display processor, sometimes referred to as a graphics controller or a display coprocessor.



- ✓ The purpose of the display processor is to free the CPU from the graphics chores.
- ✓ In addition to the system memory, a separate display-processor memory area can be provided.

<u>Scan conversion:</u>

- ✓ A major task of the display processor is digitizing a picture definition given in an application program into a set of pixel values for storage in the frame buffer.
- ✓ This digitization process is called scan conversion.

<u>Example 1: displaying a line</u>

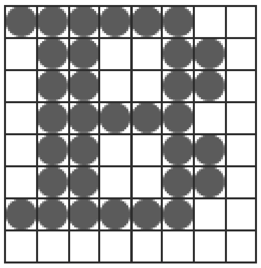
- ➔ Graphics commands specifying straight lines and other geometric objects are scan converted into a set of discrete points, corresponding to screen pixel positions.
- ➔ Scan converting a straight-line segment.

Example 2: displaying a character

➔ Characters can be defined with rectangular pixel grids

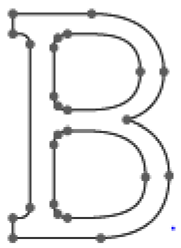
➔ The array size for character grids can vary from about 5 by 7 to 9 by 12 or more for higher-quality displays.

➔ A character grid is displayed by superimposing the rectangular grid pattern into the frame buffer at a specified coordinate position.



Using outline:

➔ For characters that are defined as outlines, the shapes are scan-converted into the frame buffer by locating the pixel positions closest to the outline.



Additional operations of Display processors:

➔ Display processors are also designed to perform a number of additional operations.

➔ These functions include generating various line styles (dashed, dotted, or solid), displaying color areas, and applying transformations to the objects in a scene.

➔ Display processors are typically designed to interface with interactive input devices, such as a mouse.

Methods to reduce memory requirements in display processor:

➔ In an effort to reduce memory requirements in raster systems, methods have been devised for organizing the frame buffer as a linked list and encoding the color information.

➔ One organization scheme is to store each scan line as a set of number pairs.

- ➔ Encoding methods can be useful in the digital storage and transmission of picture information

i) Run-length encoding:

- ❈ The first number in each pair can be a reference to a color value, and the second number can specify the number of adjacent pixels on the scan line that are to be displayed in that color.
- ❈ This technique, called run-length encoding, can result in a considerable saving in storage space if a picture is to be constructed mostly with long runs of a single color each.
- ❈ A similar approach can be taken when pixel colors change linearly.

ii) Cell encoding:

- ❈ Another approach is to encode the raster as a set of rectangular areas (cell encoding).

Disadvantages of encoding:

- ❖ The disadvantages of encoding runs are that color changes are difficult to record and storage requirements increase as the lengths of the runs decrease.
- ❖ In addition, it is difficult for the display controller to process the raster when many short runs are involved.
- ❖ Moreover, the size of the frame buffer is no longer a major concern, because of sharp declines in memory costs

Graphics workstations and viewing systems

- ✓ Most graphics monitors today operate as raster-scan displays, and both CRT and flat panel systems are in common use.
- ✓ Graphics workstation range from small general-purpose computer systems to multi monitor facilities, often with ultra –large viewing screens.
- ✓ High-definition graphics systems, with resolutions up to 2560 by 2048, are commonly used in medical imaging, air-traffic control, simulation, and CAD.
- ✓ Many high-end graphics workstations also include large viewing screens, often with specialized features.

Graphics Networks

- ➔ So far, we have mainly considered graphics applications on an isolated system with a single user.
- ➔ Multiuser environments & computer networks are now common elements in many graphics applications.
- ➔ Various resources, such as processors, printers, plotters and data files can be distributed on a network & shared by multiple users.
- ➔ A graphics monitor on a network is generally referred to as a graphics server.
- ➔ The computer on a network that is executing a graphics application is called the client.
- ➔ A workstation that includes processors, as well as a monitor and input devices can function as both a server and a client.

Graphics on Internet

- ✓ A great deal of graphics development is now done on the Internet.
- ✓ Computers on the Internet communicate using TCP/IP.
- ✓ Resources such as graphics files are identified by URL (Uniform resource locator).
- ✓ The World Wide Web provides a hypertext system that allows users to loacate and view documents, audio and graphics.
- ✓ Each URL sometimes also called as universal resource locator.
- ✓ The URL contains two parts Protocol- for transferring the document, and Server-contains the document.

Graphics Software

- ✓ There are two broad classifications for computer-graphics software

1. Special-purpose packages: Special-purpose packages are designed for nonprogrammers

   Example: generate pictures, graphs, charts, painting programs or CAD systems in some application area without worrying about the graphics procedure
2. General programming packages: general programming package provides a library of graphics functions that can be used in a programming language such as C, C++, Java, or FORTRAN.

   Example: GL (Graphics Library), OpenGL, VRML (Virtual-Reality Modeling Language), Java 2D And Java 3D

NOTE: A set of graphics functions is often called a computer-graphics application programming interface (CG API)

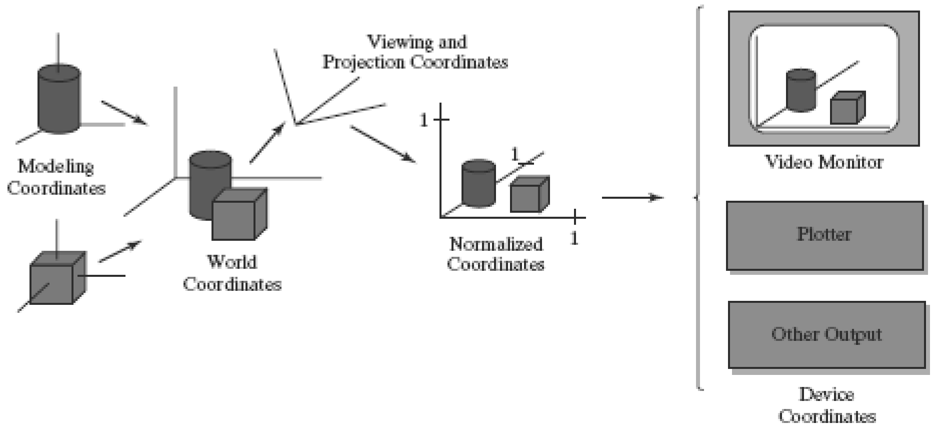
Coordinate Representations

- ✓ 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 coordinates.
- ✓ If coordinate values for a picture are given in some other reference frame (spherical, hyperbolic, etc.), they must be converted to Cartesian coordinates.
- ✓ Several different Cartesian reference frames are used in the process of constructing and displaying
- ✓ First we define the shapes of individual objects, such as trees or furniture, These reference frames are called modeling coordinates or local coordinates
- ✓ Then we place the objects into appropriate locations within a scene reference frame called world coordinates.
- ✓ After all parts of a scene have been specified, it is processed through various output-device reference frames for display. This process is called the viewing pipeline.
- ✓ The scene is then stored in normalized coordinates. Which range from −1 to 1 or from 0 to 1 Normalized coordinates are also referred to as normalized device coordinates.
- ✓ The coordinate systems for display devices are generally called device coordinates, or screen coordinates.

NOTE: Geometric descriptions in modeling coordinates and world coordinates can be given in

floating-point or integer values.

- ✓ Example: Figure 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

- ➔ It provides users with a variety of functions for creating and manipulating pictures
- ➔ The basic building blocks for pictures are referred to as graphics output primitives
- ➔ Attributes are properties of the output primitives
- ➔ We can change the size, position, or orientation of an object using geometric transformations
- ➔ Modeling transformations, which are used to construct a scene.
- ➔ Viewing transformations are used to select a view of the scene, the type of projection to be used and the location where the view is to be displayed.
- ➔ Input functions are used to control and process the data flow from these interactive devices(mouse, tablet and joystick)
- ➔ Graphics package contains a number of tasks .We can lump the functions for carrying out many tasks by under the heading control operations.

<u>Software Standards</u>

- ✓ The primary goal of standardized graphics software is portability.

- ✓ In 1984, Graphical Kernel System (GKS) was adopted as the first graphics software standard by the International Standards Organization (ISO)
- ✓ The second software standard to be developed and approved by the standards organizations was Programmer's Hierarchical Interactive Graphics System (PHIGS).
- ✓ Extension of PHIGS, called PHIGS+, was developed to provide 3-D surface rendering capabilities not available in PHIGS.
- ✓ The graphics workstations from Silicon Graphics, Inc. (SGI), came with a set of routines called GL (Graphics Library)

<u>Other Graphics Packages</u>

- ✓ Many other computer-graphics programming libraries have been developed for

1. general graphics routines
2. Some are aimed at specific applications (animation, virtual reality, etc.)
   Example: Open Inventor Virtual-Reality Modeling Language (VRML).

We can create 2-D scenes with in Java applets (java2D, Java 3D)

Introduction To OpenGL

- ✓ OpenGL basic(core) library :-A basic library of functions is provided in OpenGL for specifying graphics primitives, attributes, geometric transformations, viewing transformations, and many other operations.

<u>Basic OpenGL Syntax</u>

- ➔ Function names in the OpenGL basic library (also called the OpenGL core library) are prefixed with gl. The component word first letter is capitalized.
- ➔ For eg:- glBegin, glClear, glCopyPixels, glPolygonMode
- ➔ Symbolic constants that are used with certain functions as parameters are all in capital letters, preceded by "GL", and component are separated by underscore.
- ➔ For eg:- GL_2D, GL_RGB, GL_CCW, GL_POLYGON, GL_AMBIENT_AND_DIFFUSE.

- ➔ The OpenGL functions also expect specific data types. For example, an OpenGL function parameter might expect a value that is specified as a 32-bit integer. But the size of an integer specification can be different on different machines.
- ➔ To indicate a specific data type, OpenGL uses special built-in, data-type names, such as GLbyte, GLshort, GLint, GLfloat, GLdouble, Glboolean

Related Libraries

- ➔ In addition to OpenGL basic(core) library(prefixed with gl), there are a number of associated libraries for handling special operations:-

  1) **OpenGL Utility(GLU):-** Prefixed with "glu". It provides routines for setting up viewing and projection matrices, describing complex objects with line and polygon approximations, displaying quadrics and B-splines using linear approximations, processing the surface-rendering operations, and other complex tasks.

  -Every OpenGL implementation includes the GLU library

  2) **Open Inventor:-** provides routines and predefined object shapes for interactive three-dimensional applications which are written in C++.

  3) **Window-system libraries:-** To create graphics we need display window. We cannot create the display window directly with the basic OpenGL functions since it contains only device-independent graphics functions, and window-management operations are device-dependent. However, there are several window-system libraries that supports OpenGL functions for a variety of machines.

  Eg:- Apple GL(AGL), Windows-to-OpenGL(WGL), Presentation Manager to OpenGL(PGL), GLX.

  4) **OpenGL Utility Toolkit(GLUT):-** provides a library of functions which acts as interface for interacting with any device specific screen-windowing system, thus making our program device-independent. The GLUT library functions are prefixed with "glut".

Header Files

- ✓ In all graphics programs, we will need to include the header file for the OpenGL core library.

- ✓ In windows to include OpenGL core libraries and GLU we can use the following header files:-

  #include <windows.h> //precedes other header files for including Microsoft windows ver of OpenGL libraries

  #include<GL/gl.h>

  #include <GL/glu.h>
- ✓ The above lines can be replaced by using GLUT header file which ensures gl.h and glu.h are included correctly,
- ✓ #include <GL/glut.h> //GL in windows
- ✓ In Apple OS X systems, the header file inclusion statement will be,
- ✓ #include <GLUT/glut.h>

Display-Window Management Using GLUT

- ✓ We can consider a simplified example, minimal number of operations for displaying a picture.

Step 1: initialization of GLUT

- ❈ We are using the OpenGL Utility Toolkit, our first step is to initialize GLUT.
- ❈ This initialization function could also process any command line arguments, but we will not need to use these parameters for our first example programs.
- ❈ We perform the GLUT initialization with the statement

  glutInit (&argc, argv);

Step 2: title

- ❈ We can state that a display window is to be created on the screen with a given caption for the title bar. This is accomplished with the function

  glutCreateWindow ("An Example OpenGL Program");
- ❈ where the single argument for this function can be any character string that we want to use for the display-window title.

Step 3: Specification of the display window

- ❈ Then we need to specify what the display window is to contain.
- ❈ For this, we create a picture using OpenGL functions and pass the picture definition to the GLUT routine glutDisplayFunc, which assigns our picture to the display window.

- ❈ Example: suppose we have the OpenGL code for describing a line segment in a procedure called lineSegment.
- ❈ Then the following function call passes the line-segment description to the display window:

  glutDisplayFunc (lineSegment);

Step 4: one more GLUT function

- ❈ But the display window is not yet on the screen.
- ❈ We need one more GLUT function to complete the window-processing operations.
- ❈ After execution of the following statement, all display windows that we have created, including their graphic content, are now activated:

  glutMainLoop ( );

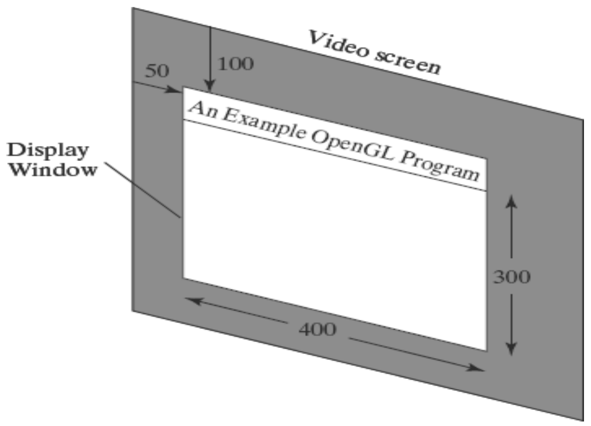
- ❈ This function must be the last one in our program. It displays the initial graphics and puts the program into an infinite loop that checks for input from devices such as a mouse or keyboard.

Step 5: these parameters using additional GLUT functions

- ❈ Although the display window that we created will be in some default location and size, we can set these parameters using additional GLUT functions.

GLUT Function 1:

- ➔ We use the glutInitWindowPosition function to give an initial location for the upper left corner of the display window.
- ➔ This position is specified in integer screen coordinates, whose origin is at the upper-left corner of the screen.



GLUT Function 2:

After the display window is on the screen, we can reposition and resize it.

GLUT Function 3:

➔ We can also set a number of other options for the display window, such as buffering and a choice of color modes, with the glutInitDisplayMode function.

➔ Arguments for this routine are assigned symbolic GLUT constants.

➔ Example: the following command specifies that a single refresh buffer is to be used for the display window and that we want to use the color mode which uses red, green, and blue (RGB) components to select color values:

glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);

➔ The values of the constants passed to this function are combined using a logical or operation.

➔ Actually, single buffering and RGB color mode are the default options.

➔ But we will use the function now as a reminder that these are the options that are set for our display.

➔ Later, we discuss color modes in more detail, as well as other display options, such as double buffering for animation applications and selecting parameters for viewing threedimensional scenes.

A Complete OpenGL Program

➔ There are still a few more tasks to perform before we have all the parts that we need for a complete program.

Step 1: to set background color

➔ For the display window, we can choose a background color.

➔ Using RGB color values, we set the background color for the display window to be white, with the OpenGL function:

glClearColor (1.0, 1.0, 1.0, 0.0);

➔ The first three arguments in this function set the red, green, and blue component colors to the value 1.0, giving us a white background color for the display window.

➔ If, instead of 1.0, we set each of the component colors to 0.0, we would get a black background.

- ➔ The fourth parameter in the glClearColor function is called the alpha value for the specified color. One use for the alpha value is as a "blending" parameter
- ➔ When we activate the OpenGL blending operations, alpha values can be used to determine the resulting color for two overlapping objects.
- ➔ An alpha value of 0.0 indicates a totally transparent object, and an alpha value of 1.0 indicates an opaque object.
- ➔ For now, we will simply set alpha to 0.0.
- ➔ Although the glClearColor command assigns a color to the display window, it does not put the display window on the screen.

Step 2: to set window color

- ➔ To get the assigned window color displayed, we need to invoke the following OpenGL function:

```
glClear (GL_COLOR_BUFFER_BIT);
```

- ➔ The argument GL COLOR BUFFER BIT is an OpenGL symbolic constant specifying that it is the bit values in the color buffer (refresh buffer) that are to be set to the values indicated in the glClearColor function. (OpenGL has several different kinds of buffers that can be manipulated.

Step 3: to set color to object

- ➔ In addition to setting the background color for the display window, we can choose a variety of color schemes for the objects we want to display in a scene.
- ➔ For our initial programming example, we will simply set the object color to be a dark green

```
glColor3f (0.0, 0.4, 0.2);
```

- ➔ The suffix 3f on the glColor function indicates that we are specifying the three RGB color components using floating-point (f) values.
- ➔ This function requires that the values be in the range from 0.0 to 1.0, and we have set red = 0.0, green = 0.4, and blue = 0.2.

Example program

- ➔ For our first program, we simply display a two-dimensional line segment.
- ➔ To do this, we need to tell OpenGL how we want to "project" our picture onto the display window because generating a two-dimensional picture is treated by OpenGL as a special case of three-dimensional viewing.
- ➔ So, although we only want to produce a very simple two-dimensional line, OpenGL processes our picture through the full three-dimensional viewing operations.
- ➔ We can set the projection type (mode) and other viewing parameters that we need with the following two functions:

```
glMatrixMode (GL_PROJECTION);
gluOrtho2D (0.0, 200.0, 0.0, 150.0);
```

- ➔ This specifies that an orthogonal projection is to be used to map the contents of a twodimensional rectangular area of world coordinates to the screen, and that the x-coordinate values within this rectangle range from 0.0 to 200.0 with y-coordinate values ranging from 0.0 to 150.0.
- ➔ Whatever objects we define within this world-coordinate rectangle will be shown within the display window.
- ➔ Anything outside this coordinate range will not be displayed.
- ➔ Therefore, the GLU function gluOrtho2D defines the coordinate reference frame within the display window to be (0.0, 0.0) at the lower-left corner of the display window and (200.0, 150.0) at the upper-right window corner.
- ➔ For now, we will use a world-coordinate rectangle with the same aspect ratio as the display window, so that there is no distortion of our picture.
- ➔ Finally, we need to call the appropriate OpenGL routines to create our line segment.
- ➔ The following code defines a two-dimensional, straight-line segment with integer,
- ➔ Cartesian endpoint coordinates (180, 15) and (10, 145).

```
glBegin (GL_LINES);
glVertex2i (180, 15);
glVertex2i (10, 145);
glEnd ( );
```

- ➔ Now we are ready to put all the pieces together:

The following OpenGL program is organized into three functions.

- ➔ init: We place all initializations and related one-time parameter settings in function init.
- ➔ lineSegment: Our geometric description of the “picture” that we want to display is in function lineSegment, which is the function that will be referenced by the GLUT function glutDisplayFunc.
- ➔ main function main function contains the GLUT functions for setting up the display window and getting our line segment onto the screen.
- ➔ glFlush: This is simply a routine to force execution of our OpenGL functions, which are stored by computer systems in buffers in different locations,depending on how OpenGL is implemented.
- ➔ The procedure lineSegment that we set up to describe our picture is referred to as a display callback function.
- ➔ And this procedure is described as being “registered” by glutDisplayFunc as the routine to invoke whenever the display window might need to be redisplayed.

Example: if the display window is moved.

Following program to display window and line segment generated by this program:

```
#include <GL/glut.h> // (or others, depending on the system in use)
void init (void)
{
        glClearColor (1.0, 1.0, 1.0, 0.0);          // Set display-window color to white.
        glMatrixMode (GL_PROJECTION);     // Set projection parameters.
        gluOrtho2D (0.0, 200.0, 0.0, 150.0);
}
void lineSegment (void)
{
        glClear (GL_COLOR_BUFFER_BIT);       // Clear display window.
        glColor3f (0.0, 0.4, 0.2);                      // Set line segment color to green.
        glBegin (GL_LINES);
        glVertex2i (180, 15);                          // Specify line-segment geometry.
        glVertex2i (10, 145);
        glEnd ( );
        glFlush ( ); // Process all OpenGL routines as quickly as possible.
}
void main (int argc, char** argv)
{
        glutInit (&argc, argv);                                   // Initialize GLUT.
        glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB); // Set display mode.
        glutInitWindowPosition (50, 100);     // Set top-left display-window position.
        glutInitWindowSize (400, 300);       // Set display-window width and height.
        glutCreateWindow ("An Example OpenGL Program"); // Create display window.
        init ( );                                  // Execute initialization procedure.
        glutDisplayFunc (lineSegment);        // Send graphics to display window.
        glutMainLoop ( );                        // Display everything and wait.
}
```

1.13 Coordinate Reference Frames

To describe a picture, we first decide upon

- A convenient Cartesian coordinate system, called the world-coordinate reference frame, which could be either 2D or 3D.
- We then describe the objects in our picture by giving their geometric specifications in terms of positions in world coordinates.
- Example: We define a straight-line segment with two endpoint positions, and a polygon is specified with a set of positions for its vertices.
- These coordinate positions are stored in the scene description along with other info about the objects, such as their color and their coordinate extents
- Co-ordinate extents :Co-ordinate extents are the minimum and maximum x, y, and z values for each object.
- A set of coordinate extents is also described as a bounding box for an object.
- Ex:For a 2D figure, the coordinate extents are sometimes called its bounding rectangle.
- Objects are then displayed by passing the scene description to the viewing routines which identify visible surfaces and map the objects to the frame buffer positions and then on the video monitor.

- ❋ The scan-conversion algorithm stores info about the scene, such as color values, at the appropriate locations in the frame buffer, and then the scene is displayed on the output device.

Screen co-ordinates:

- ✓ Locations on a video monitor are referenced in integer screen coordinates, which correspond to the integer pixel positions in the frame buffer.
- ✓ Scan-line algorithms for the graphics primitives use the coordinate descriptions to determine the locations of pixels
- ✓ Example: given the endpoint coordinates for a line segment, a display algorithm must calculate the positions for those pixels that lie along the line path between the endpoints.
- ✓ Since a pixel position occupies a finite area of the screen, the finite size of a pixel must be taken into account by the implementation algorithms.
- ✓ For the present, we assume that each integer screen position references the centre of a pixel area.
- ✓ Once pixel positions have been identified the color values must be stored in the frame buffer

Assume we have available a low-level procedure of the form

i) setPixel (x, y);

- stores the current color setting into the frame buffer at integer position(x, y), relative to the position of the screen-coordinate origin

ii) getPixel (x, y, color);

- Retrieves the current frame-buffer setting for a pixel location;
- Parameter color receives an integer value corresponding to the combined RGB bit codes stored for the specified pixel at position (x,y).
- Additional screen-coordinate information is needed for 3D scenes.
- For a two-dimensional scene, all depth values are 0.

Absolute and Relative Coordinate Specifications

Absolute coordinate:

- So far, the coordinate references that we have discussed are stated as absolute coordinate values.
- This means that the values specified are the actual positions within the coordinate system in use.

Relative coordinates:

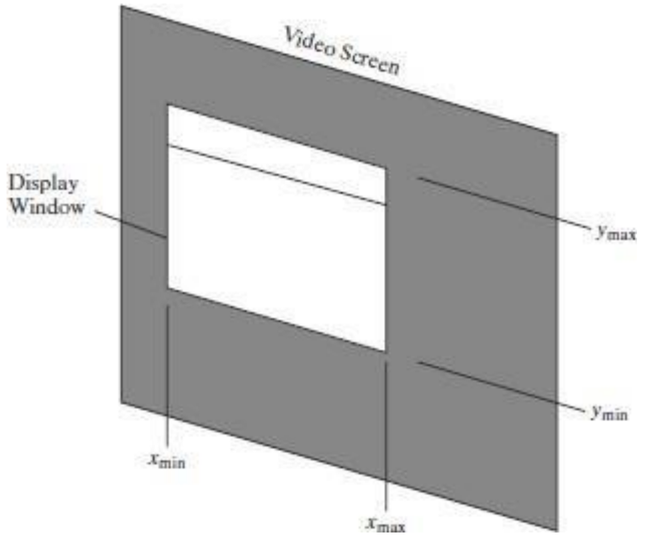
- However, some graphics packages also allow positions to be specified using relative coordinates.
- This method is useful for various graphics applications, such as producing drawings with pen plotters, artist's drawing and painting systems, and graphics packages for publishing and printing applications.
- Taking this approach, we can specify a coordinate position as an offset from the last position that was referenced (called the current position).

Specifying a Two-Dimensional World-Coordinate Reference Frame in OpenGL

- The gluOrtho2D command is a function we can use to set up any 2D Cartesian reference frames.
- The arguments for this function are the four values defining the x and y coordinate limits for the picture we want to display.
- Since the gluOrtho2D function specifies an orthogonal projection, we need also to be sure that the coordinate values are placed in the OpenGL projection matrix.
- In addition, we could assign the identity matrix as the projection matrix before defining the world-coordinate range.
- This would ensure that the coordinate values were not accumulated with any values we may have previously set for the projection matrix.
- Thus, for our initial two-dimensional examples, we can define the coordinate frame for the screen display window with the following statements

```
glMatrixMode (GL_PROJECTION);
glLoadIdentity ( );
gluOrtho2D (xmin, xmax, ymin, ymax);
```

- The display window will then be referenced by coordinates (xmin, ymin) at the lower-left corner and by coordinates (xmax, ymax) at the upper-right corner, as shown in Figure below

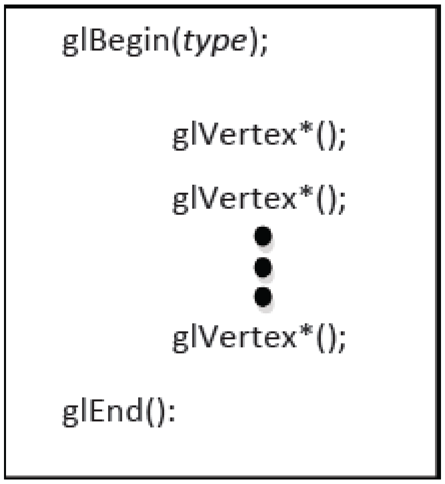


- We can then designate one or more graphics primitives for display using the coordinate reference specified in the gluOrtho2D statement.
- If the coordinate extents of a primitive are within the coordinate range of the display window, all of the primitive will be displayed.
- Otherwise, only those parts of the primitive within the display-window coordinate limits will be shown.
- Also, when we set up the geometry describing a picture, all positions for the OpenGL primitives must be given in absolute coordinates, with respect to the reference frame defined in the gluOrtho2D function.

OpenGL Functions

Geometric Primitives:

- It includes points, line segments, polygon etc.
- These primitives pass through geometric pipeline which decides whether the primitive is visible or not and also how the primitive should be visible on the screen etc.
- The geometric transformations such rotation, scaling etc can be applied on the primitives which are displayed on the screen.The programmer can create geometric primitives as shown below:



```
glBegin(type);

        glVertex*();
        glVertex*();
           ●
           ●
           ●
        glVertex*();

glEnd():
```

where:

glBegin indicates the beginning of the object that has to be displayed

glEnd indicates the end of primitive

OpenGL Point Functions

- The type within glBegin() specifies the type of the object and its value can be as follows:

  GL_POINTS
- Each vertex is displayed as a point.
- The size of the point would be of at least one pixel.
- Then this coordinate position, along with other geometric descriptions we may have in our scene, is passed to the viewing routines.
- Unless we specify other attribute values, OpenGL primitives are displayed with a default size and color.
- The default color for primitives is white, and the default point size is equal to the size of a single screen pixel

Syntax:

Case 1:

glBegin (GL_POINTS);

glVertex2i (50, 100);

glVertex2i (75, 150);

glVertex2i (100, 200);

glEnd ( );

Case 2:

- we could specify the coordinate values for the preceding points in arrays such as

  int point1 [ ] = {50, 100};

  int point2 [ ] = {75, 150};

  int point3 [ ] = {100, 200};

and call the OpenGL functions for plotting the three points as

glBegin (GL_POINTS);
glVertex2iv (point1);
glVertex2iv (point2);
glVertex2iv (point3);
glEnd ( );

Case 3:

- specifying two point positions in a three dimensional world reference frame. In this case, we give the coordinates as explicit floating-point values:

glBegin (GL_POINTS);
glVertex3f (-78.05, 909.72, 14.60);
glVertex3f (261.91, -5200.67, 188.33);
glEnd ( );

OpenGL LINE FUNCTIONS

- Primitive type is GL_LINES
- Successive pairs of vertices are considered as endpoints and they are connected to form an individual line segments.
- Note that successive segments usually are disconnected because the vertices are processed on a pair-wise basis.
- we obtain one line segment between the first and second coordinate positions and another line segment between the third and fourth positions.
- if the number of specified endpoints is odd, so the last coordinate position is ignored.

Case 1: Lines

glBegin (GL_LINES);

 glVertex2iv (p1);

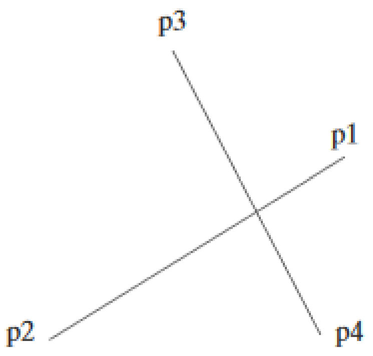
 glVertex2iv (p2);

 glVertex2iv (p3);

 glVertex2iv (p4);

 glVertex2iv (p5);

glEnd ( );



Case 2: GL_LINE_STRIP:

Successive vertices are connected using line segments. However, the final vertex is not connected to the initial vertex.

glBegin (GL_LINES_STRIP);

 glVertex2iv (p1);

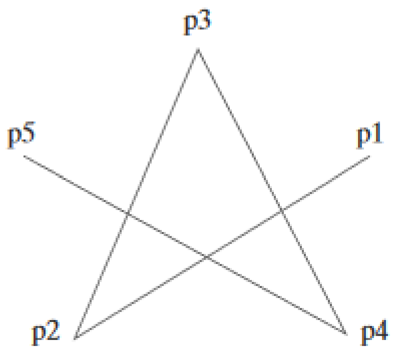
 glVertex2iv (p2);

 glVertex2iv (p3);

 glVertex2iv (p4);

 glVertex2iv (p5);

glEnd ( );



Case 3: GL_LINE_LOOP:

Successive vertices are connected using line segments to form a closed path or loop i.e., final vertex is connected to the initial vertex.

glBegin (GL_LINES_LOOP);

 glVertex2iv (p1);

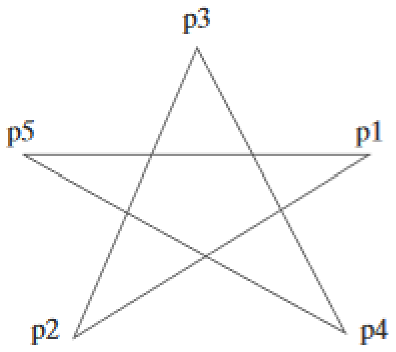
 glVertex2iv (p2);

 glVertex2iv (p3);

 glVertex2iv (p4);

 glVertex2iv (p5);

glEnd ( );



1.16 Point Attributes

➔ Basically, we can set two attributes for points: color and size.

➔ In a state system: The displayed color and size of a point is determined by the current values stored in the attribute list.

➔ Color components are set with RGB values or an index into a color table.

➔ For a raster system: Point size is an integer multiple of the pixel size, so that a large point is displayed as a square block of pixels

<u>Opengl Point-Attribute Functions</u>

<u>Color:</u>

➔ The displayed color of a designated point position is controlled by the current color values in the state list.

➔ Also, a color is specified with either the glColor function or the glIndex function.

<u>Size:</u>

➔ We set the size for an OpenGL point with

glPointSize (size);

and the point is then displayed as a square block of pixels.

➔ Parameter size is assigned a positive floating-point value, which is rounded to an integer (unless the point is to be antialiased).

➔ The number of horizontal and vertical pixels in the display of the point is determined by parameter size.

➔ Thus, a point size of 1.0 displays a single pixel, and a point size of 2.0 displays a 2×2 pixel array.

➔ If we activate the antialiasing features of OpenGL, the size of a displayed block of pixels will be modified to smooth the edges.

➔ The default value for point size is 1.0.

<u>Example program:</u>

➔ Attribute functions may be listed inside or outside of a glBegin/glEnd pair.

➔ Example: the following code segment plots three points in varying colors and sizes.

➔ The first is a standard-size red point, the second is a double-size green point, and the third is a triple-size blue point:

Ex:

```
glColor3f (1.0, 0.0, 0.0);
        glBegin (GL_POINTS);
        glVertex2i (50, 100);
        glPointSize (2.0);
        glColor3f (0.0, 1.0, 0.0);
        glVertex2i (75, 150);
        glPointSize (3.0);
        glColor3f (0.0, 0.0, 1.0);
        glVertex2i (100, 200);
glEnd ( );
```

1.17 Line-Attribute Functions OpenGL

➔ In OpenGL straight-line segment with three attribute settings: line color, line-width, and line style.

➔ OpenGL provides a function for setting the width of a line and another function for specifying a line style, such as a dashed or dotted line.

OpenGL Line-Width Function

➔ Line width is set in OpenGL with the function

Syntax: glLineWidth (width);

➔ We assign a floating-point value to parameter width, and this value is rounded to the nearest nonnegative integer.

➔ If the input value rounds to 0.0, the line is displayed with a standard width of 1.0, which is the default width.

➔ Some implementations of the line-width function might support only a limited number of widths, and some might not support widths other than 1.0.

- ➔ That is, the magnitude of the horizontal and vertical separations of the line endpoints, deltax and deltay, are compared to determine whether to generate a thick line using vertical pixel spans or horizontal pixel spans.

OpenGL Line-Style Function

- ➔ By default, a straight-line segment is displayed as a solid line.
- ➔ But we can also display dashed lines, dotted lines, or a line with a combination of dashes and dots.
- ➔ We can vary the length of the dashes and the spacing between dashes or dots.
- ➔ We set a current display style for lines with the OpenGL function:

Syntax: glLineStipple (repeatFactor, pattern);

Pattern:

- ➔ Parameter pattern is used to reference a 16-bit integer that describes how the line should be displayed.
- ➔ 1 bit in the pattern denotes an "on" pixel position, and a 0 bit indicates an "off" pixel position.
- ➔ The pattern is applied to the pixels along the line path starting with the low-order bits in the pattern.
- ➔ The default pattern is 0xFFFF (each bit position has a value of 1),which produces a solid line.

repeatFactor

- ➔ Integer parameter repeatFactor specifies how many times each bit in the pattern is to be repeated before the next bit in the pattern is applied.
- ➔ The default repeat value is 1.

Polyline:

- ➔ With a polyline, a specified line-style pattern is not restarted at the beginning of each segment.

- ➔ It is applied continuously across all the segments, starting at the first endpoint of the polyline and ending at the final endpoint for the last segment in the series.

Example:

- ➔ For line style, suppose parameter pattern is assigned the hexadecimal representation 0x00FF and the repeat factor is 1.
- ➔ This would display a dashed line with eight pixels in each dash and eight pixel positions that are "off" (an eight-pixel space) between two dashes.
- ➔ Also, since low order bits are applied first, a line begins with an eight-pixel dash starting at the first endpoint.
- ➔ This dash is followed by an eight-pixel space, then another eight-pixel dash, and so forth, until the second endpoint position is reached.

<u>Activating line style:</u>

- ➢ Before a line can be displayed in the current line-style pattern, we must activate the line-style feature of OpenGL.

  glEnable (GL_LINE_STIPPLE);
- ➢ If we forget to include this enable function, solid lines are displayed; that is, the default pattern 0xFFFF is used to display line segments.
- ➢ At any time, we can turn off the line-pattern feature with

  glDisable (GL_LINE_STIPPLE);
- ➢ This replaces the current line-style pattern with the default pattern (solid lines).

Example Code:

```
typedef struct { float x, y; } wcPt2D;
wcPt2D dataPts [5];
void linePlot (wcPt2D dataPts [5])
{
        int k;
        glBegin (GL_LINE_STRIP);
        for (k = 0; k < 5; k++)
                glVertex2f (dataPts [k].x, dataPts [k].y);
        glFlush ( );
        glEnd ( );
}
/* Invoke a procedure here to draw coordinate axes. */
glEnable (GL_LINE_STIPPLE); /* Input first set of (x, y) data values. */
glLineStipple (1, 0x1C47); // Plot a dash-dot, standard-width polyline.
linePlot (dataPts);
/* Input second set of (x, y) data values. */
glLineStipple (1, 0x00FF); / / Plot a dashed, double-width polyline.
glLineWidth (2.0);
linePlot (dataPts);
/* Input third set of (x, y) data values. */
glLineStipple (1, 0x0101); // Plot a dotted, triple-width polyline.
glLineWidth (3.0);
linePlot (dataPts);
glDisable (GL_LINE_STIPPLE);
```

1.18 Curve Attributes

- ➔ Parameters for curve attributes are the same as those for straight-line segments.
- ➔ We can display curves with varying colors, widths, dot-dash patterns, and available pen or brush options.
- ➔ Methods for adapting curve-drawing algorithms to accommodate attribute selections are similar to those for line drawing.
- ➔ Raster curves of various widths can be displayed using the method of horizontal or vertical pixel spans.

  Case 1: Where the magnitude of the curve slope |m| <= 1.0, we plot vertical spans;

  Case 2: when the slope magnitude |m| > 1.0, we plot horizontal spans.

<u>Different methods to draw a curve:</u>

<u>Method 1:</u> Using circle symmetry property, we generate the circle path with vertical spans in the octant from x = 0 to x = y, and then reflect pixel positions about the line y = x to y=0

Method 2: Another method for displaying thick curves is to fill in the area between two Parallel curve paths, whose separation distance is equal to the desired width. We could do this using the specified curve path as one boundary and setting up the second boundary either inside or outside the original curve path. This approach, however, shifts the original curve path either inward or outward, depending on which direction we choose for the second boundary.

Method 3: The pixel masks discussed for implementing line-style options could also be used in raster curve algorithms to generate dashed or dotted patterns

Method 4: Pen (or brush) displays of curves are generated using the same techniques discussed for straight-line segments.

Method 5: Painting and drawing programs allow pictures to be constructed interactively by using a pointing device, such as a stylus and a graphics tablet, to sketch various curve shapes.

Line Drawing Algorithm

- ✓ A straight-line segment in a scene is defined by coordinate positions for the endpoints of the segment.
- ✓ To display the line on a raster monitor, the graphics system must first project the endpoints to integer screen coordinates and determine the nearest pixel positions along the line path between the two endpoints then the line color is loaded into the frame buffer at the corresponding pixel coordinates
- ✓ The Cartesian slope-intercept equation for a straight line is

  $y=m * x + b$ ----------->(1)

  with m as the slope of the line and b as the y intercept.
- ✓ Given that the two endpoints of a line segment are specified at positions (x0,y0) and (xend, yend) ,as shown in fig.

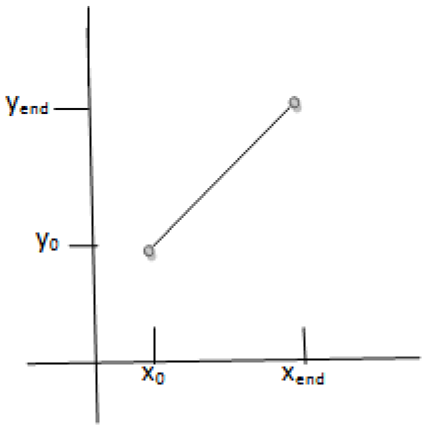


fig. Line path between endpoint positions (x_0,y_0) and (x_{end}, y_{end}).

- ✓ We determine values for the slope m and y intercept b with the following equations:

  m=(yend - y0)/(xend - x0) ---------------->(2)

  b=y0 - m.x0 ------------->(3)

- ✓ Algorithms for displaying straight line are based on the line equation (1) and calculations given in eq(2) and (3).
- ✓ For given x interval δx along a line, we can compute the corresponding y interval δy from eq.(2) as

  δy=m. δx --------------- >(4)

- ✓ Similarly, we can obtain the x interval δx corresponding to a specified δy as

  δx=δy/m ----------------->(5)

- ✓ These equations form the basis for determining deflection voltages in analog displays, such as vector-scan system, where arbitrarily small changes in deflection voltage are possible.
- ✓ For lines with slope magnitudes
  - ➔ |m|<1, δx can be set proportional to a small horizontal deflection voltage with the corresponding vertical deflection voltage set proportional to δy from eq.(4)
  - ➔ |m|>1, δy can be set proportional to a small vertical deflection voltage with the corresponding horizontal deflection voltage set proportional to δx from eq.(5)
  - ➔ |m|=1, δx=δy and the horizontal and vertical deflections voltages are equal

DDA Algorithm (DIGITAL DIFFERENTIAL ANALYZER)

➔ The DDA is a scan-conversion line algorithm based on calculating either δy or δx.

➔ A line is sampled at unit intervals in one coordinate and the corresponding integer values nearest the line path are determined for the other coordinate

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

Case1:

if m<1,x increment in unit intervals

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

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

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

$\mathbf{y_{k+1} = y_k + m}$ **----------- >(1)**

➔ where k takes integer values starting from 0,for the first point and increases by 1 until final endpoint is reached. Since m can be any real number between 0.0 and 1.0,

Case2:

if m>1, y increment in unit intervals

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

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

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

$\mathbf{x_{k+1} =(1/m)+ x_k}$ **------------------------(2)**

Case3:

if m=1,both x and y increment in unit intervals

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

Equations (1) and (2) are based on the assumption that lines are to be processed from the left endpoint to the right endpoint. If this processing is reversed, so that the starting endpoint is at the right, then either we have $\delta x=-1$ and

$y_{k+1} = y_k - m$ ----------------- (3)

or(when the slope is greater than 1)we have $\delta y=-1$ with

$x_{k+1} = x_k - (1/m)$--------------- (4)

- ➔ Similar calculations are carried out using equations (1) through (4) to determine the pixel positions along a line with negative slope. thus, if the absolute value of the slope is less than 1 and the starting endpoint is at left ,we set δx==1 and calculate y values with eq(1).
- ➔ when starting endpoint is at the right(for the same slope),we set δx=-1 and obtain y positions using eq(3).
- ➔ This algorithm is summarized in the following procedure, which accepts as input two integer screen positions for the endpoints of a line segment.
- ➔ if m<1,where x is incrementing by 1

  $y_{k+1} = y_k + m$
- ➔ So initially x=0,Assuming (x0,y0)as initial point assigning x= x0,y=y0 which is the starting point .
  - Illuminate pixel(x, round(y))
  - x1= x+ 1 , y1=y + 1
  - Illuminate pixel(x1,round(y1))
  - x2= x1+ 1 , y2=y1 + 1
  - Illuminate pixel(x2,round(y2))
  - Till it reaches final point.
- ➔ if m>1,where y is incrementing by 1

  $x_{k+1} =(1/m)+ x_k$
- ➔ So initially y=0,Assuming (x0,y0)as initial point assigning x= x0,y=y0 which is the starting point .
  - Illuminate pixel(round(x),y)
  - x1= x+(1/m) ,y1=y
  - Illuminate pixel(round(x1),y1)
  - x2= x1+ (1/m) , y2=y1
  - Illuminate pixel(round(x2),y2)
  - Till it reaches final point.

- ➔ The DDA algorithm is faster method for calculating pixel position than one that directly implements .

- ➔ It eliminates the multiplication by making use of raster characteristics, so that appropriate increments are applied in the x or y directions to step from one pixel position to another along the line path.
- ➔ The accumulation of round off error in successive additions of the floating point increment, however can cause the calculated pixel positions to drift away from the true line path for long line segments. Furthermore ,the rounding operations and floating point arithmetic in this procedure are still time consuming.
- ➔ we improve the performance of DDA algorithm by separating the increments m and 1/m into integer and fractional parts so that all calculations are reduced to integer operations.

```
#include <stdlib.h>
#include <math.h>
inline int round (const float a)
{
        return int (a + 0.5);
}
void lineDDA (int x0, int y0, int xEnd, int yEnd)
{
        int dx = xEnd - x0, dy = yEnd - y0, steps, k;
        float xIncrement, yIncrement, x = x0, y = y0;
        if (fabs (dx) > fabs (dy))
                steps = fabs (dx);
        else
                steps = fabs (dy);
        xIncrement = float (dx) / float (steps);
        yIncrement = float (dy) / float (steps);
        setPixel (round (x), round (y));
        for (k = 0; k < steps; k++) {
                x += xIncrement;
                y += yIncrement;
                setPixel (round (x), round (y));
        }
```

}

<u>Bresenham's Algorithm:</u>

- ➔ It is an efficient raster scan generating algorithm that uses incremental integral calculations
- ➔ To illustrate Bresenham's approach, we first consider the scan-conversion process for lines with positive slope less than 1.0.
- ➔ Pixel positions along a line path are then determined by sampling at unit x intervals. Starting from the left endpoint (x0, y0) of a given line, we step to each successive column (x position) and plot the pixel whose scan-line y value is closest to the line path.

- ➔ Consider the equation of a straight line y=mx+c where m=dy/dx

<u>Bresenham's Line-Drawing Algorithm for |m| < 1.0</u>

1. Input the two line endpoints and store the left endpoint in (x0, y0).
2. Set the color for frame-buffer position (x0, y0); i.e., plot the first point.
3. Calculate the constants Δx, Δy, 2Δy, and 2Δy − 2Δx, and obtain the starting value for the decision parameter as

$$\mathbf{p0 = 2\Delta y - \Delta x}$$

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

If pk < 0, the next point to plot is (xk + 1, yk) and

$$\mathbf{p_{k+1} = p_k + 2\Delta y}$$

Otherwise, the next point to plot is (xk + 1, yk + 1) and

$$\mathbf{p_{k+1} = p_k + 2\Delta y - 2\Delta x}$$

5. Repeat step 4 Δx − 1 more times.

Note:

If |m|>1.0

Then

$$\mathbf{p0 = 2\Delta x - \Delta y}$$

and

If pk < 0, the next point to plot is (xk , yk +1) and

$$\mathbf{p_{k+1} = p_k + 2\Delta x}$$

Otherwise, the next point to plot is (xk + 1, yk + 1) and

$$\mathbf{p_{k+1} = p_k + 2\Delta x - 2\Delta y}$$

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 p = 2 * dy - dx;
        int twoDy = 2 * dy, twoDyMinusDx = 2 * (dy - dx);
        int x, y;
        /* Determine which endpoint to use as start position. */
        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);
        }
}
```

Properties of Circles

- ➔ A circle is defined as the set of points that are all at a given distance r from a center position (x_c , y_c).
- ➔ For any circle point (x, y), this distance relationship is expressed by the Pythagorean theorem in Cartesian coordinates as

$$(x - x_c)^2 + (y - y_c)^2 = r^2$$

- ➔ We could use this equation to calculate the position of points on a circle circumference by stepping along the x axis in unit steps from $x_c - r$ to $x_c + r$ and calculating the corresponding y values at each position as

$$y = y_c \pm \sqrt{r^2 - (x_c - x)^2}$$

- ➔ One problem with this approach is that it involves considerable computation at each step. Moreover, the spacing between plotted pixel positions is not uniform.
- ➔ We could adjust the spacing by interchanging x and y (stepping through y values and calculating x values) whenever the absolute value of the slope of the circle is greater than 1; but this simply increases the computation and processing required by the algorithm.
- ➔ Another way to eliminate the unequal spacing is to calculate points along the circular boundary using polar coordinates r and θ
- ➔ Expressing the circle equation in parametric polar form yields the pair of equations

$$x = x_c + r\cos\theta$$
$$y = y_c + r\sin\theta$$

Midpoint Circle Algorithm

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

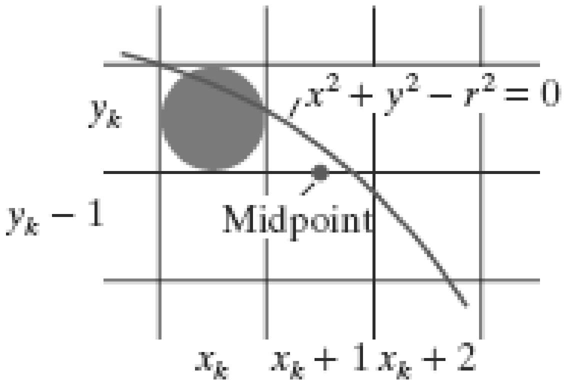
➔ The strategy is to select which of 2 pixels is closer to the circle by evaluating a function at the midpoint between the 2 pixels

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

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

➔ To summarize, the relative position of any point (x, y) can be determined by checking the sign of the circle function as follows:

$$f_{\text{circ}}(x, y) \begin{cases} < 0, & \text{if } (x, y) \text{ is inside the circle boundary} \\ = 0, & \text{if } (x, y) \text{ is on the circle boundary} \\ > 0, & \text{if } (x, y) \text{ is outside the circle boundary} \end{cases}$$



Eight way symmetry

➔ The shape of the circle is similar in each quadrant.

➔ Therefore ,if we determine the curve positions in the first quadrant ,we can generate the circle positions in the second quadrant of xy plane.

➔ The circle sections in the third and fourth quadrant can be obtained from sections in the first and second quadrant by considering the symmetry along X axis

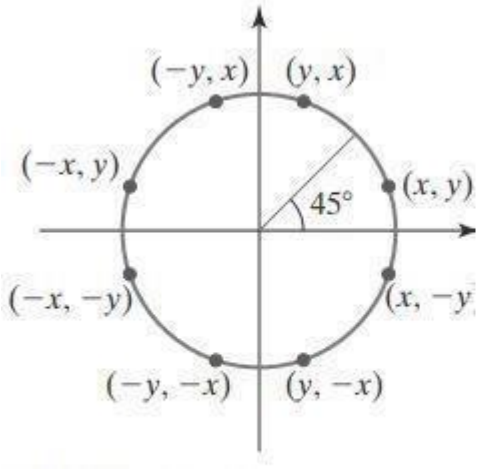


FIGURE 13
Symmetry of a circle. Calculation of a circle point (x, y) in one octant yields the circle points shown for the other seven octants.

➔ Conside the circle centered at the origin,if the point (x, y) is on the circle,then we can compute 7 other points on the circle as shown in the above figure.

➔ Our decision parameter is the circle function evaluated at the midpoint between these two pixels:

$$p_k = f_{\text{circ}}\left(x_k + 1, y_k - \frac{1}{2}\right)$$
$$= (x_k + 1)^2 + \left(y_k - \frac{1}{2}\right)^2 - r^2$$

➔ Successive decision parameters are obtained using incremental calculations.

➔ We obtain a recursive expression for the next decision parameter by evaluating the circle function at sampling position $x_{k+1} + 1 = x_k + 2$:

$$p_{k+1} = f_{\text{circ}}\left(x_{k+1} + 1, y_{k+1} - \frac{1}{2}\right)$$
$$= [(x_k + 1) + 1]^2 + \left(y_{k+1} - \frac{1}{2}\right)^2 - r^2$$

or

$$p_{k+1} = p_k + 2(x_k + 1) + \left(y_{k+1}^2 - y_k^2\right) - (y_{k+1} - y_k) + 1$$

where y_{k+1} is either y_k or $y_k - 1$, depending on the sign of p_k.

➔ The initial decision parameter is obtained by evaluating the circle function at the start position $(x_0, y_0) = (0, r)$:

$$p_0 = f_{\text{circ}}\left(1, r - \frac{1}{2}\right)$$

$$= 1 + \left(r - \frac{1}{2}\right)^2 - r^2$$

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

➔ If the radius r is specified as an integer, we can simply round p0 to

p0 = 1 − r (for r an integer)

because all increments are integers.

Midpoint Circle Algorithm

1. Input radius *r* and circle center (x_c, y_c), then set the coordinates for 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

$$p0 = 1\text{-}r$$

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, yk) and

$$p_{k+1} = p_k + 2x_{k+1} + 1$$

Otherwise, the next point along the circle is $(x_k + 1, y_k - 1)$ and

$$p_{k+1} = p_k + 2x_{k+1} + 1 - 2y_{k+1}$$

where $2x_{k+1} = 2xk + 2$ and $2y_{k+1} = 2yk - 2$.

4. Determine symmetry points in the other seven octants.

5. Move each calculated pixel position (x, y) onto the circular path centered at (x_c, y_c) and plot the coordinate values as follows:

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

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 x, 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>x)
        {
                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);
}
```

Problems based on DDA algorithm:

1) **Calculate the points between the starting point (5, 6) and ending point (8, 12).**

Solution :

Given-

– Starting coordinates = (X_0, Y_0) = (5, 6)

– Ending coordinates = (X_n, Y_n) = (8, 12)

Step-01:

- Calculate ΔX, ΔY and M from the given input.

  $\Delta X = X_n - X_0 = 8 - 5 = 3$

  $\Delta Y = Y_n - Y_0 = 12 - 6 = 6$

  $M = \Delta Y / \Delta X = 6 / 3 = 2$

Step-02:

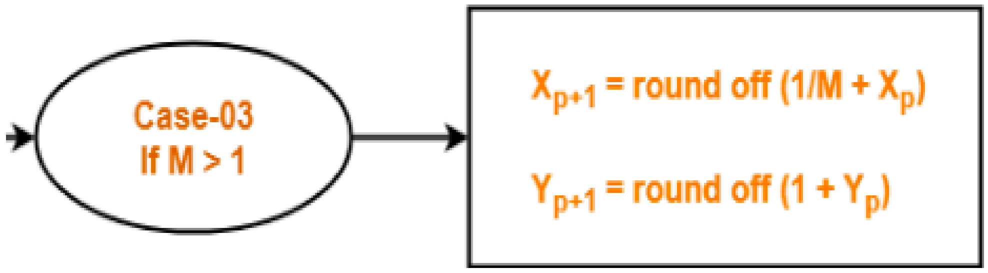
- Calculate the number of steps.

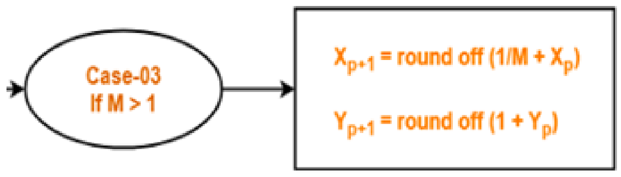
  as $|\Delta X| < |\Delta Y| = 3 < 6$,

  so number of steps = $\Delta Y = 6$

Step-03:

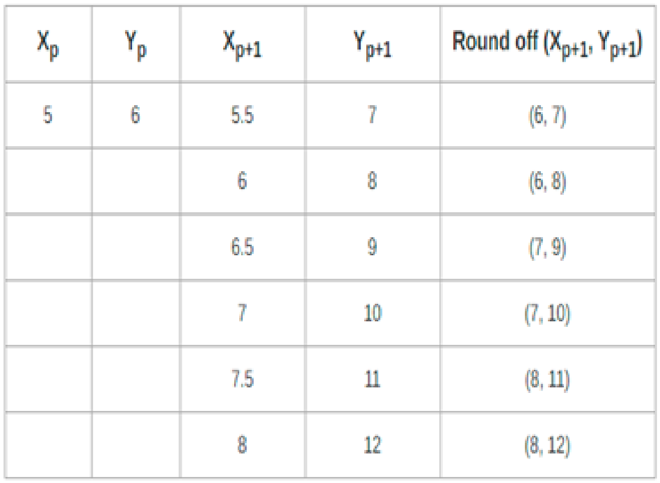
- As $M > 1$, so case-03 is satisfied.
- Now, Step-03 is executed until Step-04 is satisfied.


Case-03
If M > 1
X_{p+1} = round off (1/M + X_p)
Y_{p+1} = round off (1 + Y_p)

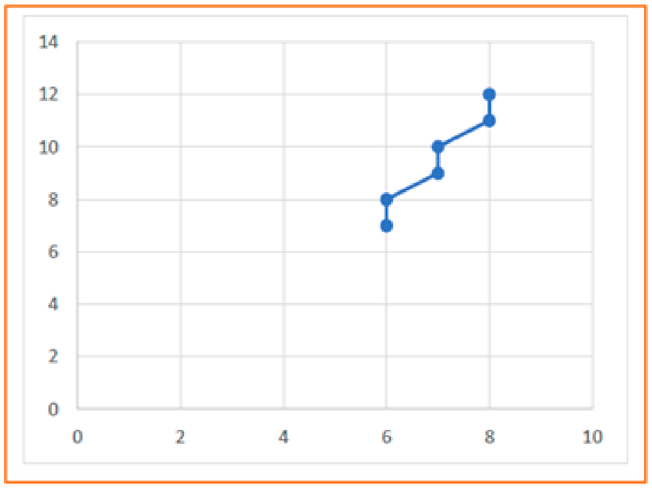


Case-03
If M > 1
X_{p+1} = round off (1/M + X_p)
Y_{p+1} = round off (1 + Y_p)

$M = \Delta Y / \Delta X = 6 / 3 = 2$



X_p	Y_p	X_{p+1}	Y_{p+1}	Round off (X_{p+1}, Y_{p+1})
5	6	5.5	7	(6, 7)
		6	8	(6, 8)
		6.5	9	(7, 9)
		7	10	(7, 10)
		7.5	11	(8, 11)
		8	12	(8, 12)



2) **Calculate the points between the starting point (1, 7) and ending point (11, 17).**

Solution :

Given:

– Starting coordinates = (X_0, Y_0) = (1, 7)

– Ending coordinates = (X_n, Y_n) = (11, 17)

<u>Step-01:</u>

- **Calculate ΔX, ΔY and M from the given input.**

$\Delta X = X_n – X_0 = 11 – 1 = 10$

$\Delta Y = Y_n – Y_0 = 17 – 7 = 10$

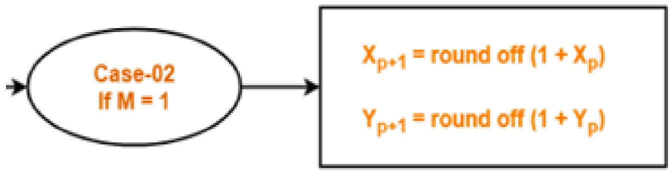
$M = \Delta Y / \Delta X = 10 / 10 = 1$

Step-02:

- **Calculate the number of steps.**
- **As |ΔX| = |ΔY| = 10 = 10, so number of steps = ΔX = ΔY = 10**

Step-03:

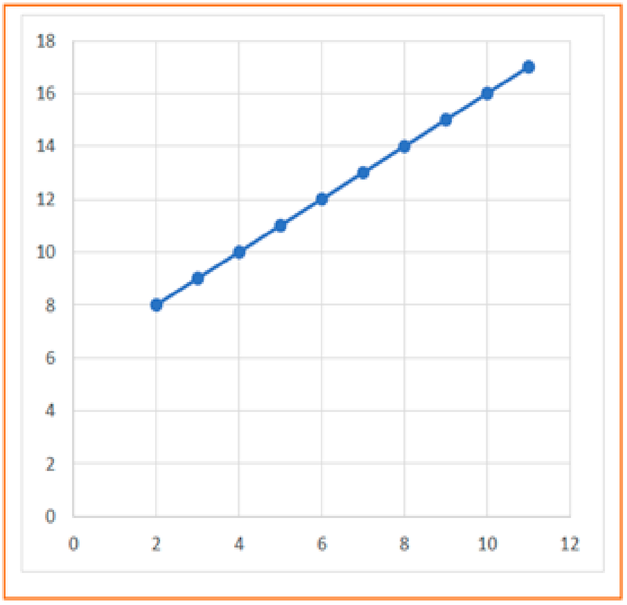
- **As M = 1, so case-02 is satisfied.**
- **Now, Step-03 is executed until Step-04 is satisfied.**



Case-02
If M = 1
X_{p+1} = round off (1 + X_p)
Y_{p+1} = round off (1 + Y_p)

M = ΔY / ΔX = 10 / 10 = 1

X_p	Y_p	X_{p+1}	Y_{p+1}	Round off (X_{p+1}, Y_{p+1})
1	7	2	8	(2, 8)
		3	9	(3, 9)
		4	10	(4, 10)
		5	11	(5, 11)
		6	12	(6, 12)
		7	13	(7, 13)
		8	14	(8, 14)
		9	15	(9, 15)
		10	16	(10, 16)
		11	17	(11, 17)



Problems on Bresenham's algorithm :

Example 1 : Calculate the points between the starting coordinates (9, 18) and ending coordinates (14, 22).

<u>Solution-</u>

Given-
Starting coordinates = (X_0, Y_0) = (9, 18)
Ending coordinates = (X_n, Y_n) = (14, 22)

<u>Step-01:</u>

Calculate ΔX and ΔY from the given input.
$\Delta X = X_n - X_0 = 14 - 9 = 5$
$\Delta Y = Y_n - Y_0 = 22 - 18 = 4$

<u>Step-02:</u>

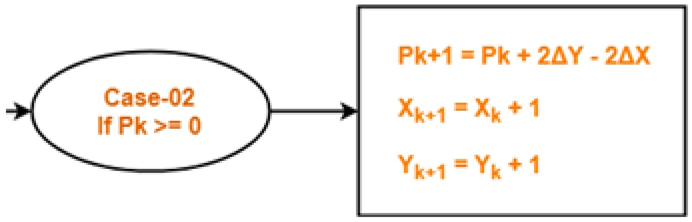
Calculate the decision parameter.
$P_k = 2\Delta Y - \Delta X$
$= 2 \times 4 - 5$
$= 3$
So, decision parameter $P_k = 3$

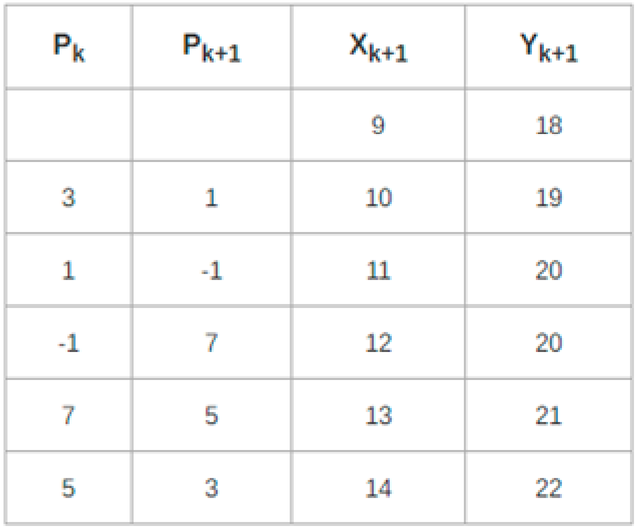
<u>Step-03:</u>

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

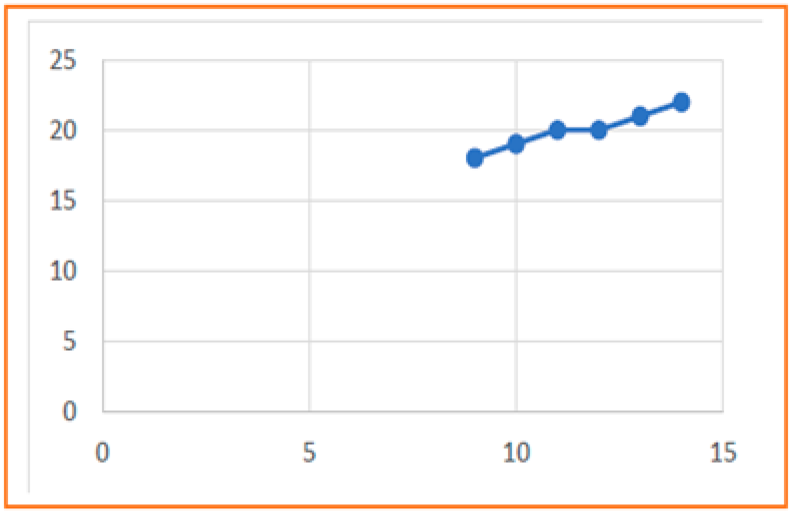
Thus,
$P_{k+1} = P_k + 2\Delta Y – 2\Delta X = 3 + (2 \times 4) – (2 \times 5) = 1$
$X_{k+1} = X_k + 1 = 9 + 1 = 10$
$Y_{k+1} = Y_k + 1 = 18 + 1 = 19$

- **Similarly, Step-03 is executed until the end point is reached or number of iterations equals to 4 times.**
- **(Number of iterations = $\Delta X – 1 = 5 – 1 = 4$)**





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 Algorithm

To illustrate the algorithm, we digitize the line with endpoints *(20, 10)* and **(30, 18)**
This line has a **slope** of **0.8**, with

dx =10, **dy =8**

The **initial decision parameter** has the value:

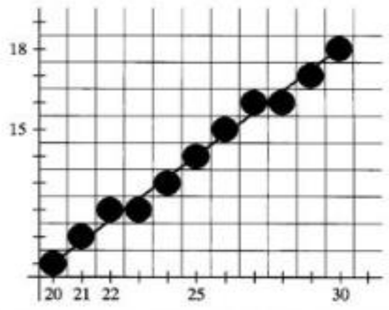
$$p = 20dy - dx = 6$$

and the **increments** for calculating successive decision parameters are

$$20dy = 16, \qquad 2dy - 2dx = -4$$

- We plot the **initial** point (x_0, y_0)= **(20, 10),** and determine successive pixel positions along the line path from the decision parameter as:

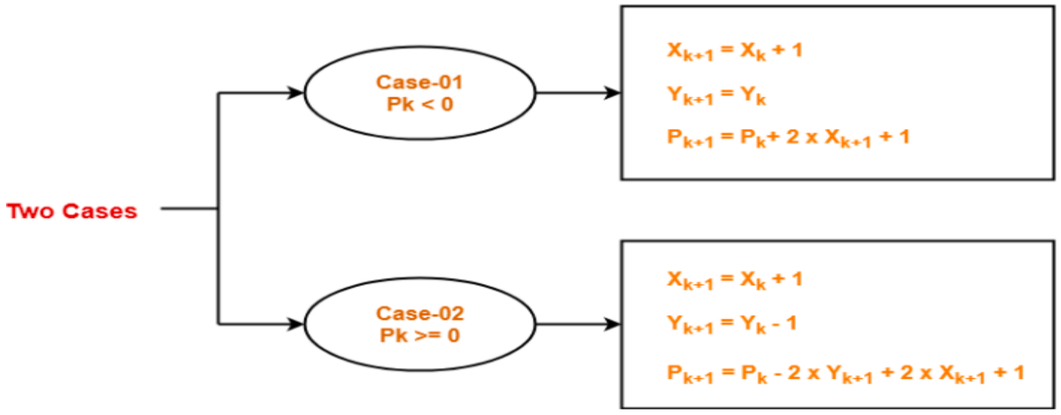
k	p_k	(x_{k+1}, y_{k+1})
0	6	(21, 11)
1	2	(22, 12)
2	-2	(23, 12)
3	14	(24, 13)
4	10	(25, 14)
5	6	(26, 15)
6	2	(27, 16)
7	-2	(28, 16)
8	14	(29, 17)
9	10	(30, 18)



•Pixel positions along the line path between endpoints (20, 10) and (30, 18), plotted with Bresenham's line algorithm.

Problems on Midpoint circle algorithm:

1) Given the centre point coordinates (0, 0) and radius as 10, generate all the points to form a circle.



Solution :

Given-

Centre Coordinates of Circle $(X_0, Y_0) = (0, 0)$

Radius of Circle = 10

Step-01:

Assign the starting point coordinates (X_0, Y_0) as-

$X_0 = 0$

$Y_0 = R = 10$

Step-02:

Calculate the value of initial decision parameter P_0 as-

$P_0 = 1 – R$

$P_0 = 1 – 10$

$P_0 = -9$

Step-03:

As $P_{initial} < 0$, so case - 01 is satisfied.

Thus,

$X_{k+1} = X_k + 1 = 0 + 1 = 1$

$Y_{k+1} = Y_k = 10$

$P_{k+1} = P_k + 2 \times X_{k+1} + 1 = -9 + (2 \times 1) + 1 = -6$

Step-04:

This step is not applicable here as the given centre

point coordinates is (0, 0).

$X_{plot} = X_c + X_0$

$Y_{plot} = Y_c + Y_0$

<u>Step-04:</u>

Step-03 is executed similarly until $X_{k+1} >= Y_{k+1}$

Similarly Step -03 will executed as follows:

<u>Step-03:</u>

As $P_{K+1} < 0$, so case - 01

$X_{k+1} = X_k + 1 = 1 + 1 = 2$

$Y_{k+1} = Y_k = 10$

$P_{k+1} = P_k + 2 . X_{k+1} + 1 = -6 + (2 x 2) + 1 = -1$

As $P_{K+1} < 0$, so case - 01

$X_{k+1} = X_k + 1 = 2 + 1 = 3$

$Y_{k+1} = Y_k = 10$

$P_{k+1} = P_k + 2 . X_{k+1} + 1 = - 1 + (2 x 3) + 1 = 6$

As $P_{K+1} > 0$, so case - 02

$X_{k+1} = X_k + 1 = 3 + 1 = 4$

$Y_{k+1} = Y_k - 1= 10-1 = 9$

$P_{k+1} = P_k - 2 . Y_{k+1} +2 . X_{k+1} +1= 6-(2.9)+(2.4)+ 1 = -3$

As $P_{K+1} < 0$, so case - 01
$X_{k+1} = X_k + 1 = 4 + 1 = 5$
$Y_{k+1} = Y_k = 9$
$P_{k+1} = P_k + 2 . X_{k+1} + 1 = -3 + (2 x 5) + 1 = 8$

As $P_{K+1} > 0$, so case - 02
$X_{k+1} = X_k + 1 = 5 + 1 = 6$
$Y_{k+1} = Y_k -1 = 9-1 = 8$
$P_{k+1} = P_k - 2 . Y_{k+1} + 2 . X_{k+1} = 8- (2 x 8) +(2.6)+1 = 5$

As $P_{K+1} > 0$, so case - 02
$X_{k+1} = X_k + 1 = 6 + 1 = 7$
$Y_{k+1} = Y_k - 1= 8-1 = 7$
$P_{k+1} = P_k - 2 . Y_{k+1} +2 . X_{k+1} +1= 9 - (2.7)+(2.7)+ 1 = 10$

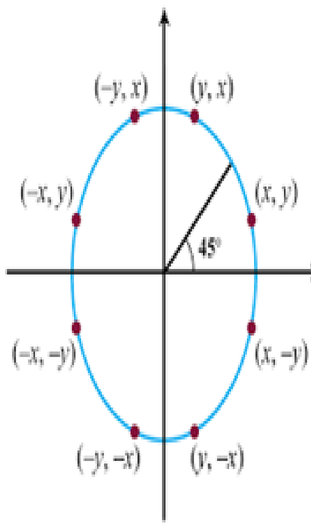
Algorithm Terminates when (X>=Y)
These are all points for Octant-1

The following table gives the values for first octant :

P_k	P_{k+1}	(X_{k+1}, Y_{k+1})	$2X_{k+1}$	$2Y_{k+1}$
		(0, 10)		
-9	-6	(1, 10)	2	20
-6	-1	(2, 10)	4	20
-1	6	(3, 10)	6	20
6	-3	(4, 9)	8	18
-3	8	(5, 9)	10	18
8	5	(6, 8)	12	16
5	10	(7,7)	14	14

8 octants with (x,y) points

Octant-1 Points (X,Y)	Octant-2 Points (X,Y)	Octant-3 Points (-X, Y)	Octant-4 Points (-X,Y)	Octant-5 Points (-X,-Y)	Octant-6 Points (-X,-Y)	Octant-7 Points (X,-Y)	Octant-8 Points (X,-Y)
(0, 10)	(7,7)	(0, 10)	(-7,7)	**(0, -10)**	(-7,-7)	(0, -10)	(7,-7)
(1, 10)	(8, 6)	(-1, 10)	(-8, 6)	**(-1, -10)**	(-8,-6)	(1, -10)	(8, -6)
(2, 10)	(9, 5)	(-2, 10)	(-9, 5)	**(-2, -10)**	(-9, -5)	(2, -10)	(9, -5)
(3, 10)	(9, 4)	(-3, 10)	(-9, 4)	**(-3,-10)**	(-9, -4)	(3, -10)	(9, -4)
(4, 9)	(10, 3)	(-4, 9)	(-10, 3)	**(-4, -9)**	(-10,-3)	(4, -9)	(10, -3)
(5, 9)	(10, 2)	(-5, 9)	(-10, 2)	**(-5, -9)**	(-10,-2)	(5, -9)	(10, -2)
(6, 8)	(10, 1)	(-6, 8)	(-10, 1)	**(-6, -8)**	(-10, -1)	(6, -8)	(10, -1)
(7,7)	(10, 0)	(-7,7)	(-10, 0)	**(-7,-7)**	(-10, 0)	**(7,-7)**	(10, 0)



Eight-way Symmetry