Module 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.

Applications of Computer Graphics

1. Graphs and Charts
2. Computer-Aided Design
3. Virtual-Reality Environments
4. Data Visualizations
5. Education and Training
6. Computer Art
7. Entertainment
8. Image Processing
9. Graphical User Interfaces

1. Graphs and Charts

Computer graphics is used to display simple data graphs usually plotted on a character printer. Data plotting is 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, consumer information bulletins and other types of publications.

Typically examples of data plots are line graphs, bar charts, pie charts, surface graphs and contour plots and other displays showing relationships between multiple parameters in two dimensions, three dimensions, or higher-dimensional spaces.

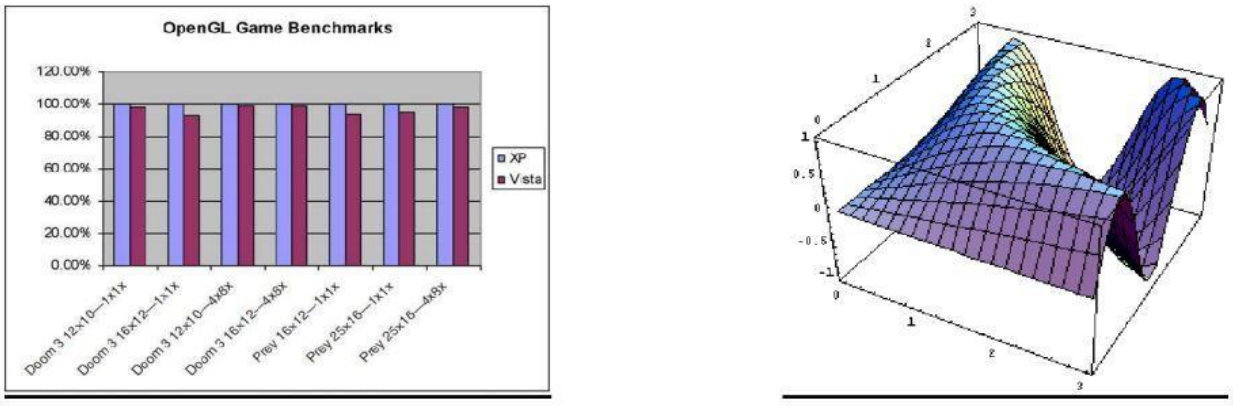

Figure 1.1: Graphs and Charts

2. 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, networks for communications, water supply are constructed with repeated placement of a few graphical 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.

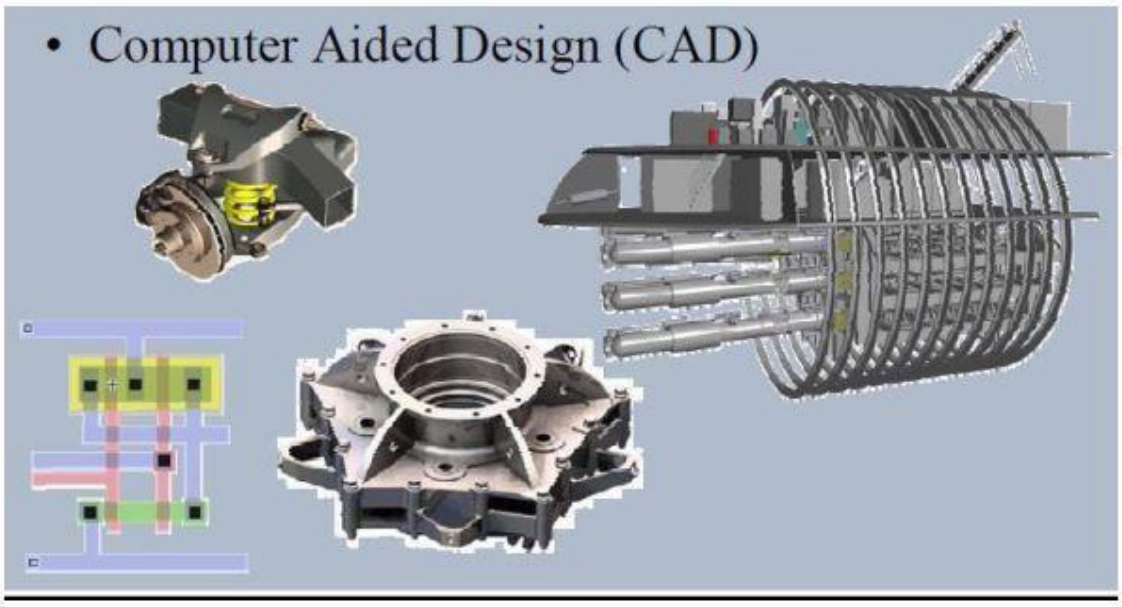

Figure 1.2: Computer Aided Design (CAD)

3. Virtual Reality Environments

In virtual reality environments, user can interact with the objects in a three dimensional scene. Specialized hardware devices provide three-dimensional viewing effects and allow the user to pick up objects in a scene. 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, user can "grasp" objects in a scene and turn them over or move them from one place to another.

Figure 1.3: Virtual Reality 4. Data Visualizations

Producing graphical representations for scientific, engineering and medical data sets and processes is generally referred to as scientific visualization. Business visualization is used in connection with data sets related to commerce, industry and other nonscientific areas. Researchers, analysts often need to deal with large amount of information's. When data are converted to a visual form, the trends and patterns are often immediately apparent. Visual techniques are also used to aid in the understanding and analysis of complex processes and mathematical functions. 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.

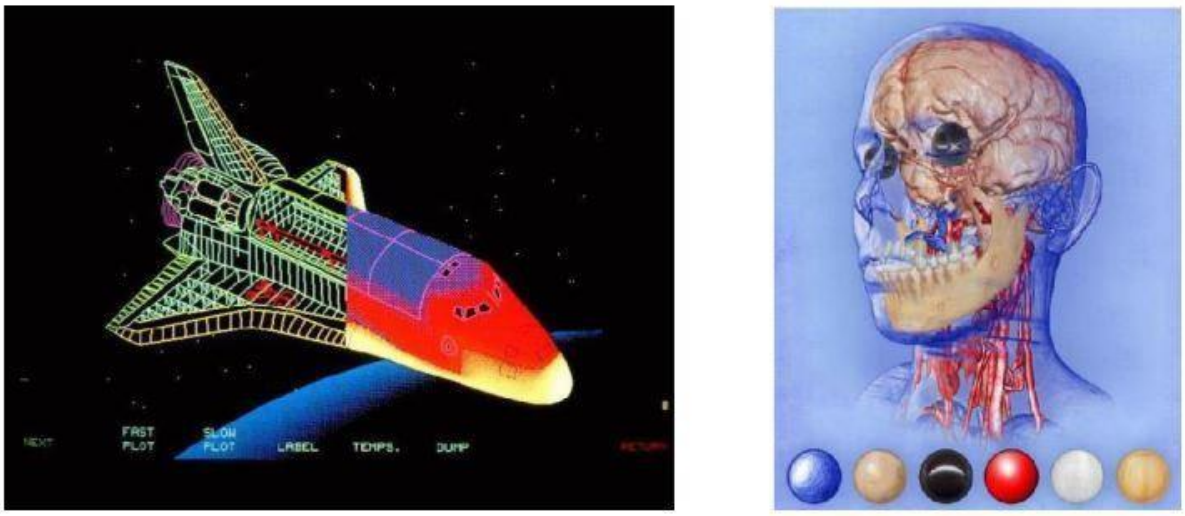

Figure 1.4: Data Visualization 5. Education and Training

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 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

Figure 1.5: Education and Training 6. Computer Art

Both fine art and commercial art make use of computer-graphics methods. Artists have variety of computer methods and tools that provides facilities for designing object shapes and specifying object motions. 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.

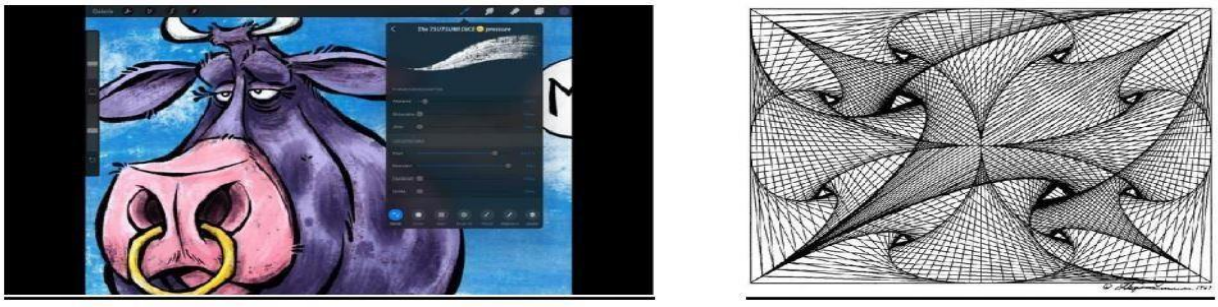

Figure 1.6: Computer Art

7. Entertainment

Computer graphics methods are commonly used in making motion pictures, music videos and television shows. Sometimes graphics images are combined with live actors and scenes, and sometimes the films are completely generated using computer rendering and animation techniques. Graphics images are combined with live actors and scenes . The films are completely generated using computer rendering and animation techniques. Computer graphics is used in making of cartoon animation films. 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.

Figure 1.7: Entertainment 8. Image Processing

The modification or interpretation of existing pictures, such as photographs and TV scans is called image processing. In computer graphics, a computer is used to create a picture. Image processing techniques, 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. Image processing and computer graphics are combined in medical applications to model and study physical functions, to design artificial limbs and to plan and practice surgery.

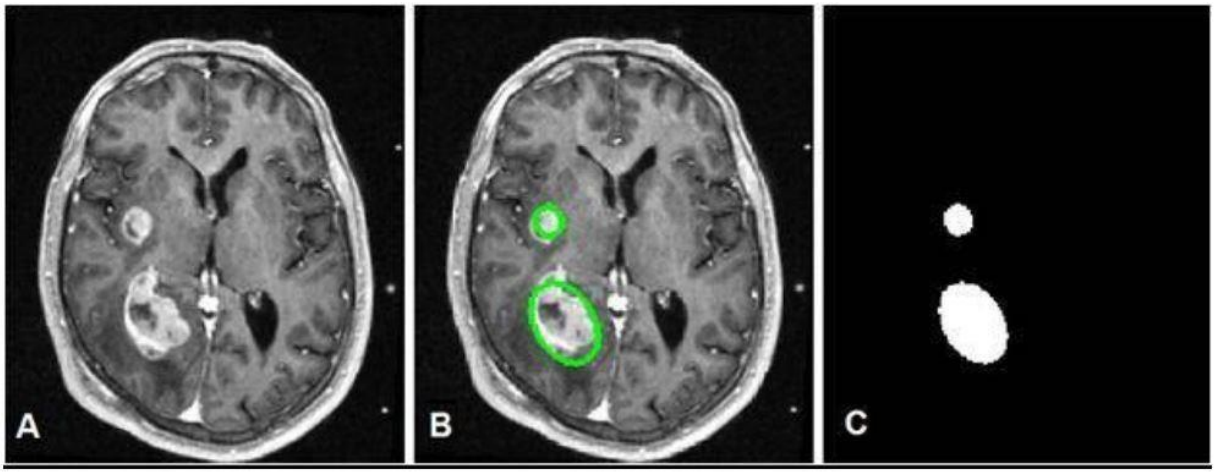

Figure 1.8: Image Processing 9. Graphical User Interfaces

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.

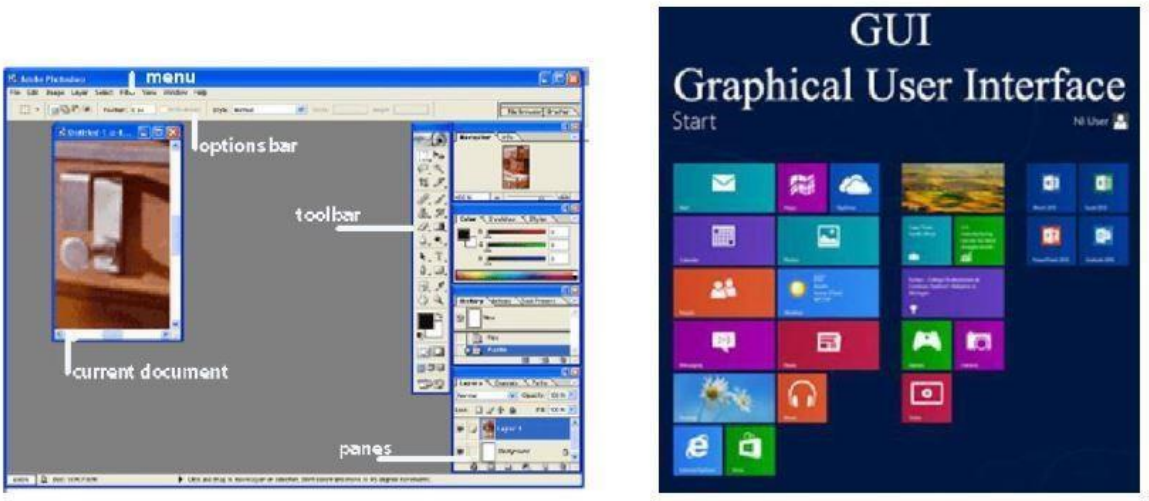

Figure 1.9: Graphical User Interfaces

Video Display Devices

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

Refresh Cathode-Ray Tubes

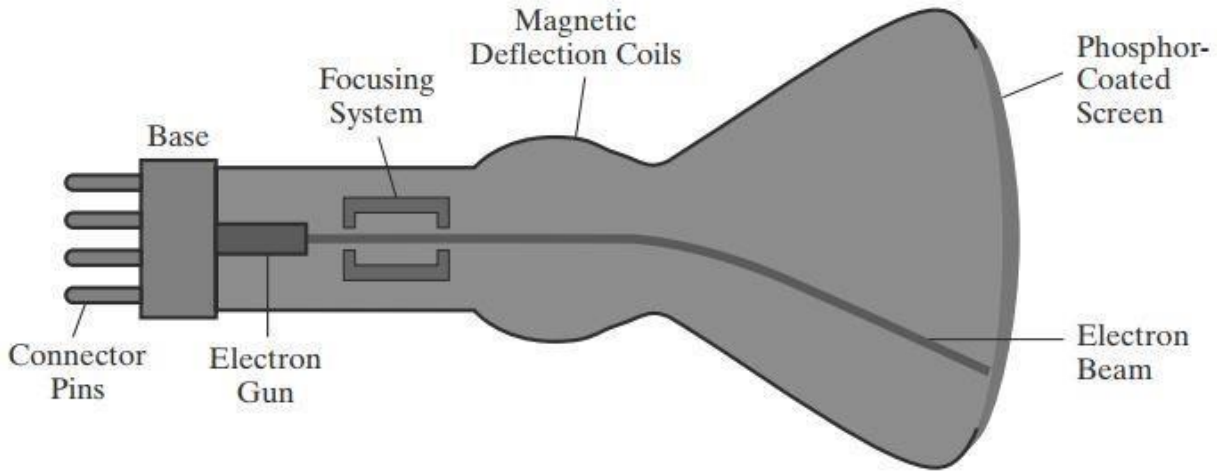

Figure 1.10: Basic Design of magnetic-deflection CRT

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. 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.

Operation of an electron gun with an accelerating anode

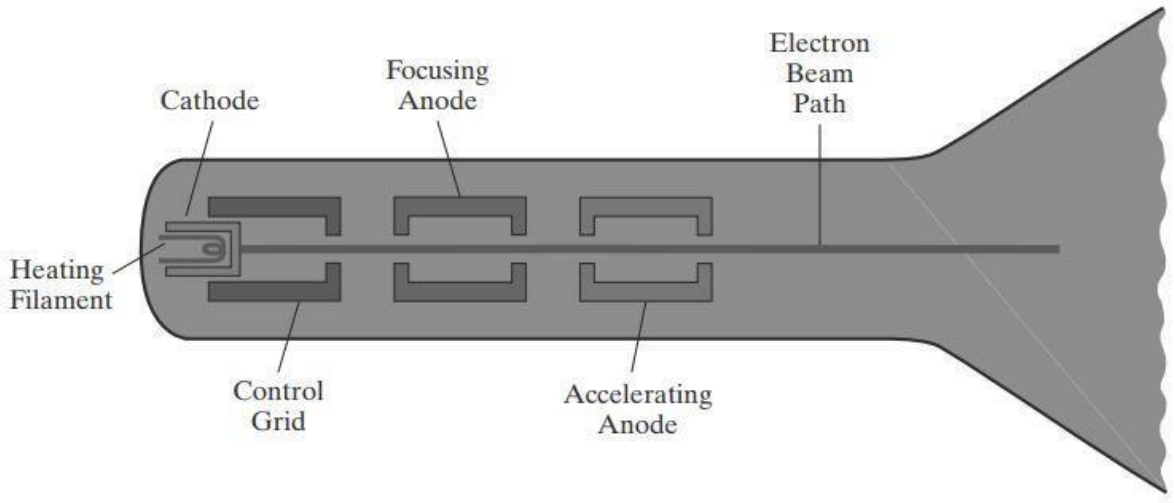

Figure 1.11: Operation of an electron gun with an accelerating anode

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.

The accelerating voltage can be generated with

1. Positive charged metal coating on the inside of the CRT envelope
2. Accelerating anode

Intensity of the electron beam is controlled by the voltage at the control grid. 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.

Focusing System

The focusing system in a CRT forces the electron beam to converge to a small cross section as it strikes the phosphor. Focusing is accomplished with either **electric fields** 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 fields 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 is produced by each pair of coils. The magnetic field produces a deflection force that is perpendicular to both the direction of the magnetic field and the direction of travel of the electron beam. Horizontal deflection is accomplished with 1 pair of coils. Vertical deflection is accomplished with other 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. One pair of plates is mounted horizontally to control vertical deflection. 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 to the heat energy, and the remainder causes electrons 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.

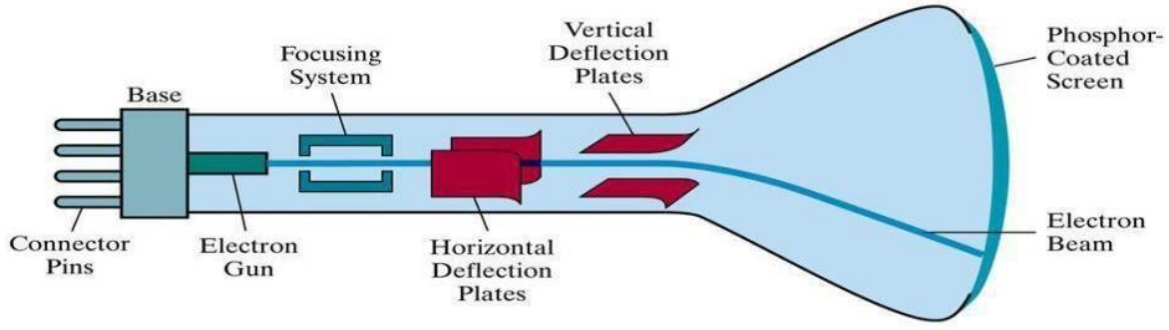

Figure 1.12: Electrostatic Deflection of the electron beam in a CRT

The frequency (or color) of the light emitted by the phosphor is proportional to the energy difference between the **excited quantum state and the ground state**.

Persistence

It is defined as the time that it takes the emitted light from the screen to decay to one-tenth of its original intensity. Lower persistence phosphors require higher refresh rates to maintain a picture without any flicker.

Resolution

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 type of phosphor, the intensity to be displayed, the focusing and deflection systems. High-resolution systems are often referred to as high-definition systems.

Raster-Scan Displays

The most common type of graphics monitor employing a CRT is the **raster-scan display,** based on television technology. The electron beam is swept across the screen one row at a time from top to bottom. Each row is called as scan line. As the electron beam move across a scan line, 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**, where the term **frame** refers to the total screen area. Frame buffer stores the intensity values for all the screen points. Refresh buffer holds the set of color values for the screen points. These stored color values are then retrieved from refresh buffer and used to control the intensity of electron beam as it moves from spot to spot across the screen. Each screen point that can be illuminated by the electron beam is called as a **pixel** (picture element). Since the refresh buffer is used to store the set of screen color values, it is also sometimes called a **color buffer**

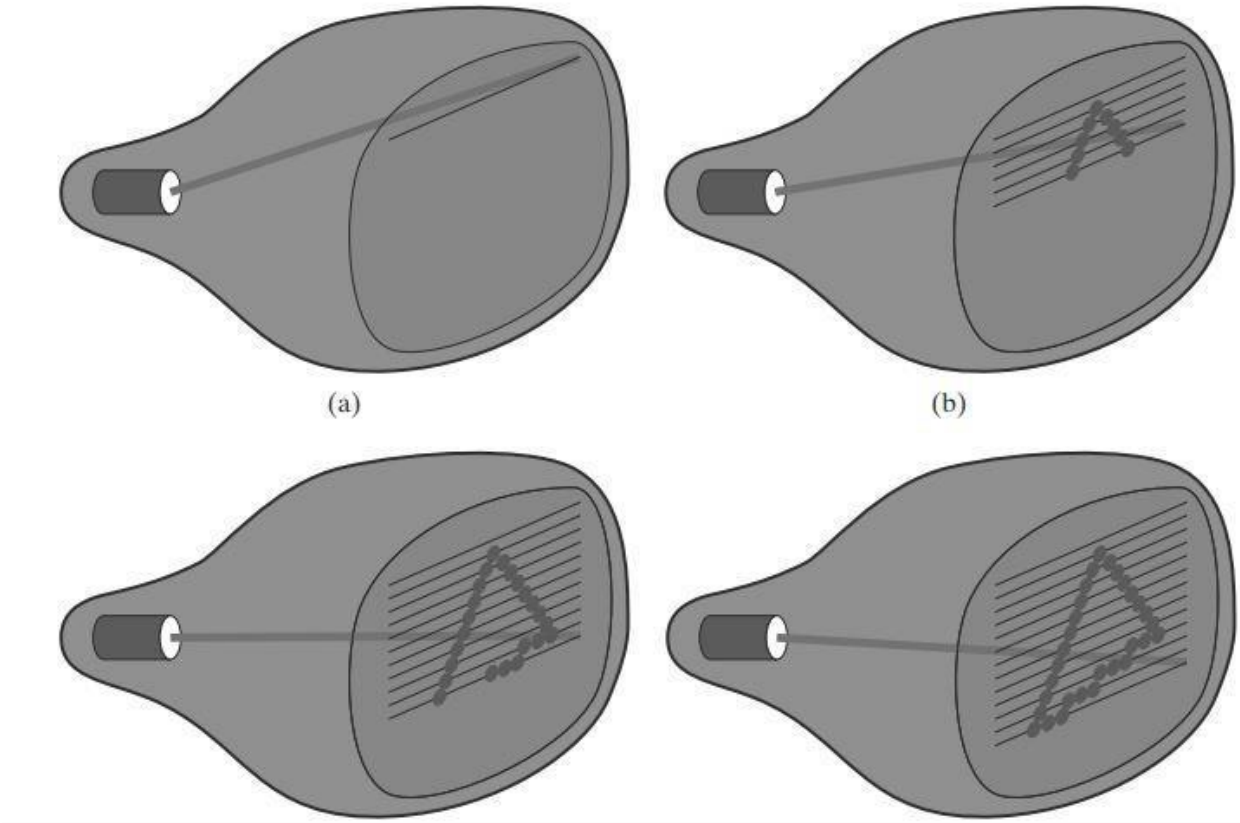

Figure 1.13: A raster scan system displays an object as a set of discrete points across scan line

Aspect ratio is defined as number of pixel columns divided by number of scan lines that can be displayed by the system. **Aspect ratio** can also be described as the number of horizontal points to vertical points (or vice versa) necessary to produce equal-length lines in both directions on the screen. Aspect ratio of 4/3 means horizontal points plotted with four points has same length as a vertical line plotted with 3 points.

The range of colors or shades of gray that can be displayed on a raster system depends on both the types of phosphor used in the CRT and the number of bits per pixel available in frame buffer.

1. Black and White Systems

For **black and white systems**, each screen point is either on or off. So only **one bit per pixel** is needed to control the intensity of screen positions. A frame buffer with one bit per pixel is called **bitmap**. A frame buffer with multiple bits per pixel is called **pixmap**.

2. Colored Systems

In case of **color Systems** many bits per pixel is needed to control the intensity of screen positions. (Example: 24 bits per pixel). Each entry in the pixmap occupies a number of bits to represent the color of the pixel.

Current raster scan displays perform refreshing at the rate of 60 to 80 frames per second. Refresh rates are described in units of cycles per second or Hertz (60 fps or 60Hz). The return to the left of the screen, after refreshing each scan line is called **horizontal retrace**. The electron beam returns to the top left corner of the screen to begin next frame is called **vertical retrace**.

Interlacing

In some raster scan systems and TV sets, each frame is displayed in two passes using an interlaced display procedure. In the first pass, beam sweeps across every other scan line from top to bottom. After the vertical retrace, then the beep sweeps out the remaining scan lines. Interlacing the screen allows user to see the entire screen displayed in one-half the time it would have taken to sweep across all the lines at once from top to bottom. This technique is used with slower refresh rates.

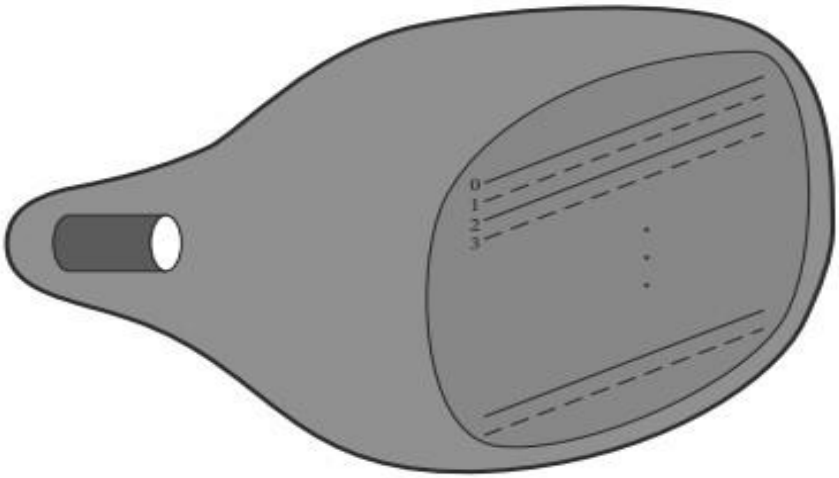

Figure 1.14: Interlacing scan line on a raster-scan display. First all points on the evennumbered(solid) scan lines are displayed. Then all points along the odd-numbered (dashed) lines are displayed.

Random-Scan Displays

Random-scan monitors are also referred to as **vector displays** (or stroke writing displays or calligraphic displays). 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. The component lines of a picture can be drawn and refreshed by a random-scan system in any specified order.

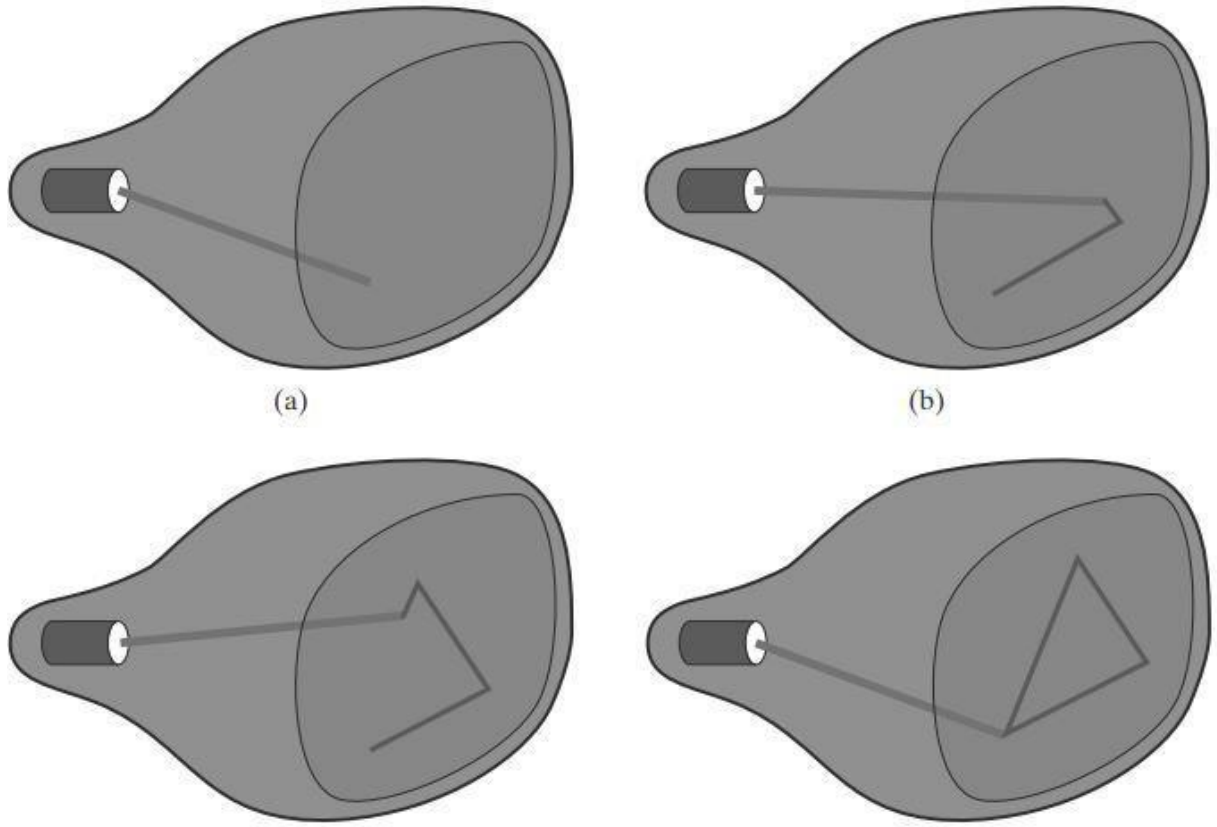

Figure 1.15: A random-scan system draws the component lines of an object in any specified order.

Refresh rate on a random-scan system depends on the number of lines to be displayed on that system. Picture definition is 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.

Difference between Raster scan system and Random scan system

Raster Scan System	Random Scan System
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 it 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 commands in a display file.
It is suitable for the realistic display of scenes containing shadow and color pattern.	These systems are designed for line-drawing and can't display realistic scenes.
Screen points/pixels are used to draw an image.	Mathematical functions are used to draw an image.

Introduction to OpenGL

A basic library of functions is provided in OpenGL for specifying graphics primitives, attributes, geometric transformations, viewing transformations, and many other operations.

Basic OpenGL Syntax

Function names in the OpenGL basic library (OpenGL core library) are prefixed with **gl**. Each component word within a function name has its first letter capitalized.

Example:

glBegin , glClear , glEnd

Symbolic Constant that are used with certain functions as parameters are all in capital letters, preceded by "GL", and component are separated by underscore.

Example: GL_POLYGON, GL_POINTS, GL_LINES

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)

All function names are 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.
All GLU function names start with the prefix glu.

2. Open Inventor

Open Inventor is toolkit based on OpenGL. It provides routines and predefined object shapes for interactive three dimensional applications. This toolkit is 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 windowsystem libraries that supports OpenGL functions for a variety of machines.

Example:

- Apple systems use **Apple GL(AGL) interface** for Window management operations.

- For Microsoft Windows, **WGL routines** provide a Windows-to-OpenGL interface.
- **Presentation Manager to OpenGL(PGL)** is an interface for IBM OS/2 **4. OpenGL Utility Toolkit(GLUT)**

OpenGL Utility Toolkit(GLUT) provides a library of functions for interacting with any screen windowing system. The GLUT library functions are prefixed with "**glut**". This library also contains methods for describing and rendering quadric curves and surfaces.

Header Files

For all graphics programs, header file for the OpenGL core library has to be included. In windows to include OpenGL core libraries and GLU we can use the following header files.

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

However, if we use GLUT to handle the window-managing operations, we do not need to include **gl.h** and **glu.h** because GLUT ensures that these will be included correctly. Thus, we can replace the header files for OpenGL and GLU with

#include<GL/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

OpenGL Utility Toolkit is used. First step is to initialize GLUT. This initialization function could also process any command line arguments. GLUT initialization can be performed using following 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

There is a need to specify what display window is to contain. 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: Complete the window processing operations **glutMainloop()** function is needed to complete the window processing operations. After execution of the following statement, all display windows created, including their graphic content will be activated.

glutMainloop();

This function must be the last one in 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.

Additional GLUT functions

1. glutInitWindowPosition

The above function is used 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.

The following statement specifies that the upper-left corner of the display window should be placed 50 pixels to the right of the left edge of the screen and 100 pixels down from the top edge of the screen:

glutInitWindowPosition(50,100);

2. glutInitWindowSize

The above function is used to set the initial pixel width and height of the display window.

Usage: glutInitWindowSize

(400, 300);

Display window is specified with initial width of 400 pixels and a height of 300 pixels

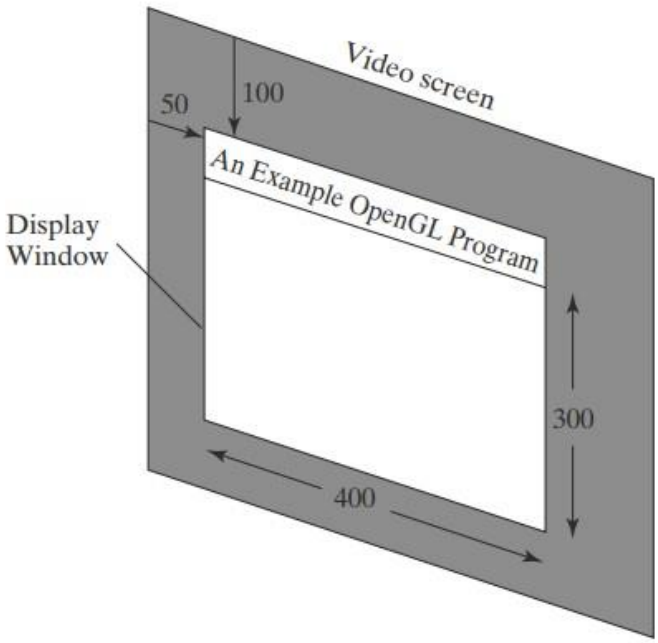

Figure 1.16: A 400 by 300 display window at position (50, 100) relative to the top-left corner of the video display.

3. glutInitDisplayMode

This function is used for specifying **buffering** and choice of **color modes**.

Arguments for this routine are assigned symbolic GLUT constants.

Usage:

glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);

Example: Above 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. The values of the constants passed to this function are combined using a logical or operation. Single buffering and RGB color mode are the default options.

A Complete OpenGL Program

1. glClearColor **glClearColor** function is used to

set background color.

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**. An alpha value of 1.0 indicates an **opaque object**. Although the glClearColor command assigns a color to the display window, it does not put the display window on the screen.

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.

3. To set Color to object glColor3f(1.0,0.0,0.0);

Above function is used to set color (red) to the object.

The suffix 3f on the glColor function indicates that we are specifying the three RGB color components using floating-point (f) values

Dark green color for object can be set using following function. **glColor3f(0.0,0.4,0.2);**

This function requires that the values be in the range from 0.0 to 1.0, and in above example we have set red = 0.0, green = 0.4, and blue = 0.2

Consider a program to display a two dimensional line-segment. 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.

Projection type (mode) and other viewing parameters can be set using 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 two dimensional 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.

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
( );
```

The following OpenGL program is organized into three functions.

1. **init:** All initializations and related one-time parameter settings are placed in function init.
2. **lineSegment:** The geometric description of the "picture" that to be displayed is in function lineSegment, which is the function that will be referenced by the GLUT function glutDisplayFunc.
3. **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 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. This procedure is described as being "registered" by glutDisplayFunc as the routine to invoke whenever the display window might need to be redisplayed. **OpenGl program to display a line segment**

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

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. }
```

Output

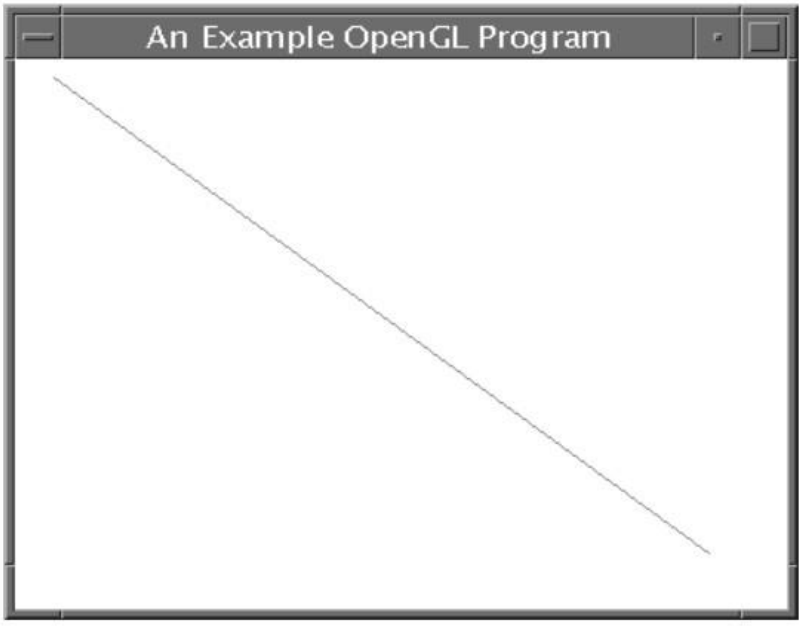

Figure 1.17: The display window and line segment produced by the example program.

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 information about the objects, such as their color and their coordinate 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. **Example:** 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

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

The values specified are the actual positions within the coordinate system in use.

Relative coordinates

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.

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.

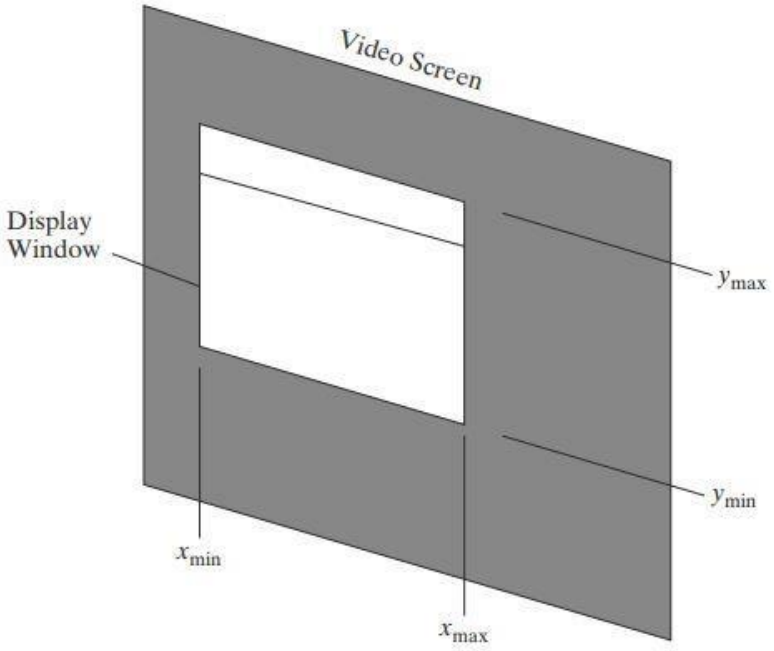

Figure 1.18: World-coordinate limits for a display window, as specified in the glOrtho2D function

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 Point Functions

To specify the geometry of a point, coordinate position has to be given in the world reference frame. Then this coordinate position, along with other geometric descriptions in the scene is passed to the viewing routines. 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**.

Following OpenGL function is used to state the coordinate values for a single position.

glVertex*();

Asterisk (*) indicates suffix codes are required for this function. Suffix codes are used to identify the spatial dimension. Numerical data type to be used for the coordinate values. A **glVertex**

function must be placed between a **glBegin** and a **glEnd** function. The argument of **glBegin** indicates the kind of output primitive to be displayed. For point plotting, the argument of the **glBegin** function is the symbolic constant **GL_POINTS**.

The form for an OpenGL specification of a point position is

glBegin(GL_POINTS) glVertex*() glEnd();

Coordinate positions in OpenGL can be given in two, three, or four dimensions. We use a suffix value of 2, 3, or 4 on the **glVertex** function to indicate the dimensionality of a coordinate position. Suffix code for specifying numerical data type are i(Integer), s(short), f(float), d(double)

Coordinate values can be listed explicitly in glVertex function. Coordinate position can be given using array specification. In such cases suffix code v has to be used. (v stands for vector).

In the following example, three equally spaced points are plotted along a two dimensional, straight-line path with a slope of 2

```
glBegin (GL_POINTS);
   glVertex2i (50, 100);
glVertex2i (75, 150);
glVertex2i (100, 200);
glEnd();
```

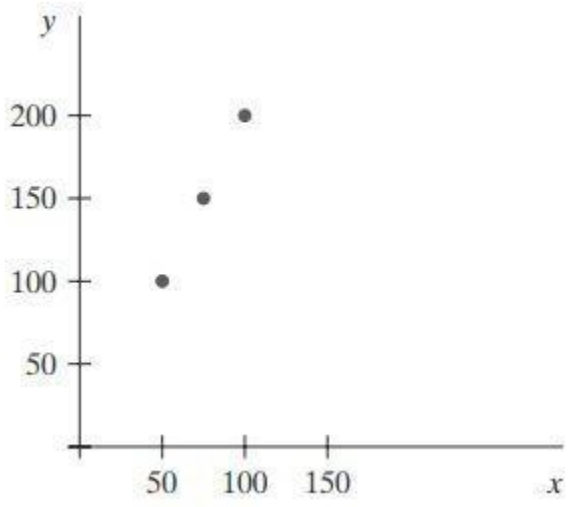

Figure 1.19: Display of three point positions generated with glBegin(GL_POINTS)

Specifying coordinates values for the points in arrays

int point1 [ ] = {50, 100}; int point2 [ ] = {75, 150}; int

point3 [ ] = {100, 200};

OpenGL functions for plotting three points are as follows:

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

Specifying two point positions in a three dimensional world reference frame Coordinates

are given as explicit floating values

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

OpenGL line functions

A set of straight-line segments between each successive pair of endpoints in a list is generated using the primitive line constant **GL_LINES**. Successive pairs of vertices are considered as endpoints and they are connected to form an individual line segments. Successive segments usually are disconnected because the vertices are processed on a pair-wise basis.

For example, if we have five coordinate positions, labeled **p1** through **p5**, and each is represented as a two- dimensional array, then the following code could generate the display shown in figure below.

```
glBegin(GL_LINES);
```

```
        glVertex2iv (p1);
        glVertex2iv (p2);
        glVertex2iv (p3);
        glVertex2iv (p4);
        glVertex2iv (p5);
glEnd( );
```

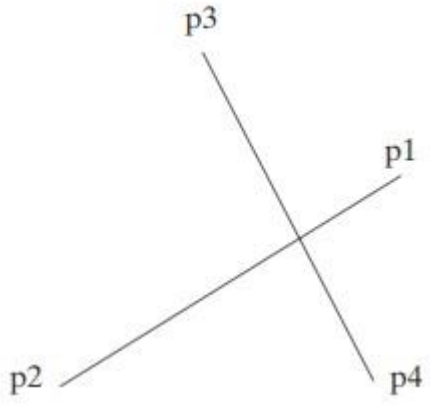

Figure 1.20: An unconnected set of lines generated with the primitive line constant GL_LINES

One line segment is between the first and second coordinate positions and another line segment is between the third and fourth coordinate positions. In this case, the number of specified endpoints is odd, so the last coordinate position is ignored.

With the OpenGL primitive constant **GL_LINE_STRIP** polyline is obtained. Successive vertices are connected using line segments. However, the final vertex is not connected to the initial vertex.

```
glBegin (GL_LINE_STRIP);
glVertex2iv (p1);
glVertex2iv (p2);
glVertex2iv (p3);
glVertex2iv (p4);
glVertex2iv (p5); glEnd ( );
```

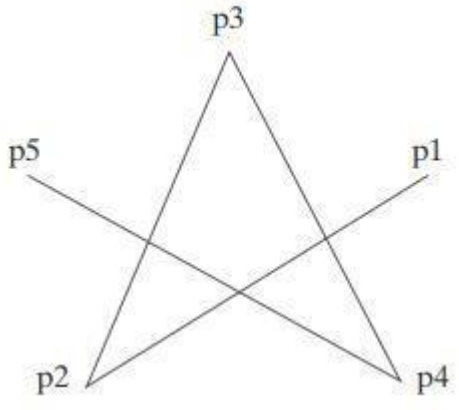

Figure 1.21: A polyline generated with GL_LINE_STRIP

The OpenGL line primitive **GL_LINE_LOOP** produces a closed polyline. 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_LINE_LOOP);
glVertex2iv (p1);
glVertex2iv (p2);
glVertex2iv (p3);
glVertex2iv (p4);
glVertex2iv (p5); glEnd( );
```

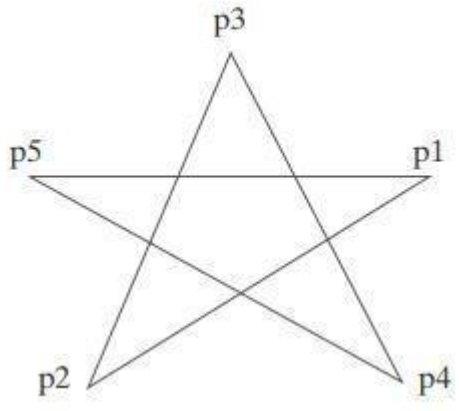

Figure 1.22: A closed polyline generated with GL_LINE_LOOP

Point Attributes

Two attributes can be set to points.

1. Color
2. 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.

OpenGL Point-Attribute Functions

The displayed color of a designated point position is controlled by the current color values in the state list. A color is specified with either the **glColor** function or the **glIndex** function.

Example:

glColor3f(1.0,0.0,0.0) // Assign red color to primitive.

Size

The size of OpenGL point is set using following function:

glPointSize(size);

Point is displayed as a square block of pixels. Parameter size is assigned a positive floating-point value, which is rounded to an integer. The number of horizontal and vertical pixels in the display of the point is determined by parameter **size**. A point size of 1.0 displays a single pixel, and a point size of 2.0 displays a 2×2 pixel array The default value for point size is 1.0

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

Code for plotting three points in varying colors and sizes

```
glColor3f (1.0, 0.0, 0.0); glBegin (GL_POINTS);
   glVertex2i (50, 100);      // Standard-size red point
glPointSize (2.0);     glColor3f (0.0, 1.0, 0.0);
```

```
glVertex2i (75, 150);        // Double-size green point
glPointSize (3.0);    glColor3f (0.0, 0.0, 1.0);
glVertex2i (100, 200);          // Triple-size blue point
glEnd ( );
```

The above code segment plots three points in varying colors and sizes. The first is a standardsize red point, the second is a double-size green point, and the third is a triple-size blue point.

Line Attributes

A straight line segment can be displayed with three basic attributes.

i) Color **ii)**

Width **iii)**

Style

Line color is set with the function **glColor3f**. Line width and line style are selected with separate line functions. Lines can be also generated using pen and brush strokes.

Line Width

A heavy line could be displayed on a video monitor as adjacent parallel lines. For raster implementations, a standard-width line is generated with single pixels at each sample position. Thicker lines are displayed as positive integer multiples of the standard line by plotting additional pixels along adjacent parallel line paths.

Line Style

Line style attributes include **solid lines**, **dashed lines** and **dotted lines**. Line drawing algorithm can be modified to generate such lines by setting the length and spacing of displayed solid sections along the line path. Using graphics package user can select the length of both the dashes and inter-dash spacing.

Pen and Brush Options

With some packages, particularly painting and drawing systems, we can select different pen and brush styles directly. Options in this category include shape, size and pattern for pen or brush.

OpenGL Line-Attribute Functions

OpenGL provides functions for

1. Setting Color for the line
2. Setting width of a line
3. Specifying a line style (Ex: Dashed,dotted)

Color can assigned to line using **glColor3f** function.

OpenGL Line Width Function

Line width is set in OpenGL with the function **glLineWidth(width)**.

Floating-point value is assigned 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.

OpenGL Line-Style Function

By default, a straight-line segment is displayed as a solid line. User can display dashed lines, dotted lines or a line with combination of dashes and dots. We can vary the length of the dashes and the spacing between dashes and dots.

Current display style for lines can be set using 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.

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. 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.

Activating line style

Before a line can be displayed in the current line-style pattern, we must activate the line style feature of OpenGL. To activate the line style feature of OpenGL following function is used.

glEnable(GL_LINE_STIPPLE);

If we forget to include this enable function, solid lines are displayed.

i.e the default pattern 0xFFFF is used to display line segments. The

line-pattern feature can be turned off with following function:

glDisable(GL_LINE_STIPPLE);

This replaces the current line-style pattern with the default pattern (solid lines).

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. 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. Curved lines can be drawn with a paint program using various shapes and patterns. Shape of curve can be drawn using **short line segments**. Curved segments can be drawn using **splines**. Curved segments can be drawn using **evaluator** functions or functions from the GLU library which draw splines.

Line Drawing Algorithms

1. DDA(Digital differential analyzer algorithm)
2. Bresenham's 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)** where m is the slope of the line and b is the y intercept.

Given that the two endpoints of a line segment are specified at positions (x_0,y_0) and (x_{end},y_{end})

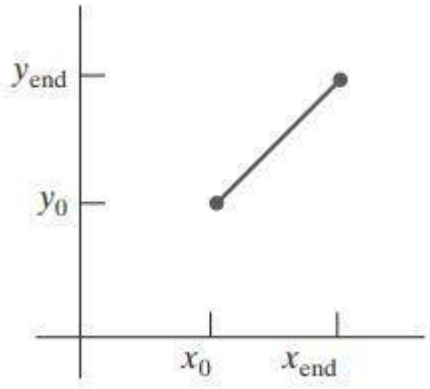

Figure 1.23: Line path between end positions (x_0,y_0) and (x_{end},y_{end})

Slope value is determined as follows

$$m = \frac{y_{end} - y_0}{x_{end} - x_0}$$

(2) y intercept is determined as follows

$$b=y_0-mx_0 \qquad (3)$$

Algorithms for displaying straight line are based on the line equation (1) and calculations given in equations 2 and 3.

For any given x interval δx along a line, we can compute the corresponding y interval δy from equation (2) as:

$$\delta y=m.\ \delta x \qquad (4)$$

The x interval δx corresponding to a specified δy is computed as

$$\delta x=\delta y/m \qquad (5)$$

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.

Case1: (Line is plotted from left to right)

if m<=1,x increment in unit intervals

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

Successive **y** values are calculated as:

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

where k takes integer values starting from 0 for the first point and increases by 1 until final endpoint is reached.

Case2: (Line is plotted from left to right) if

m>1, y increment in unit intervals

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

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

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

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

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

Successive **x** values are calculated as:

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

Case3: (Line is plotted from right to left)

If m<=1, x decrement in unit intervals.

i.e $\delta x= -1$

$x_{k+1}=x_k-1$

$y_{k+1} = y_k – m$

Case4: (Line is plotted from right to left)

If m>1,y decrement in unit intervals.

i.e $\delta y= -1$ **$x_{k+1}=$**

x_k - (1/m)

Advantages

- The DDA algorithm is faster method for calculating pixel position.
- 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.

Disadvantages

- 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.

- 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. **Procedure for DDA Algorithm**

```
#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));   }

}
```

Explanation of Algorithm

This algorithm accepts as input two integer screen positions for the endpoints of a line segment. Horizontal and vertical differences between the endpoint positions are assigned to parameters **dx** and **dy**. The difference with the greater magnitude determines the value of parameter **steps**. This value is the number of pixels that must be drawn beyond the starting pixel; from it. We calculate

the *x* and *y* increments needed to generate the next pixel position at each step along the line path. We draw the starting pixel at position (**x0**, **y0**), and then draw the remaining pixels iteratively, adjusting *x* and *y* at each step to obtain the next pixel's position before drawing it.

If the magnitude of **dx** is greater than the magnitude of **dy** and **x0** is less than **xEnd**, the values for the increments in the *x* and *y* directions are 1 and *m*, respectively. If the greater change is in the *x* direction, but **x0** is greater than **xEnd**, then the decrements -1 and -*m* are used to generate each new point on the line. Otherwise, we use a unit increment (or decrement) in the *y* direction and an *x* increment (or decrement) of 1/m.

Bresenham's Algorithm(Line Drawing)

It is an accurate and efficient raster line generating algorithm developed by Bresenham. This algorithm uses only incremental integral calculations.

Consider a scan line process for lines with positive slope less than 1. Pixel positions along a line path is determined by sampling at unit x intervals. Starting from the left endpoint (x_0,y_0) of a given line, step to each successive column(x position) and plot the pixel whose scan-line y value is closest to the path.

Assuming that we have determined the pixel at (x_k,y_k) is to be displayed, we next decide which pixel to plot in column x_{k+1}. The choices are pixels at position (x_k+1,y_k) and (x_k+1,y_k+1)

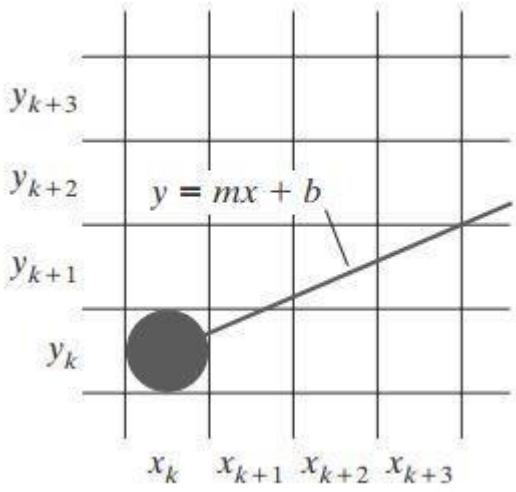

Figure 1.24: A section of the screen showing a pixel in column x_k on scan line y_k that is to be plotted along the path of a line segment with slope 0 < m < 1

In Bresenham's algorithm a **decision parameter** is used to select a pixel position from either of the two choices by testing the sign of an integer parameter.

The value of integer parameter is proportional to the difference between the separations of the two pixel positions from the actual line path.

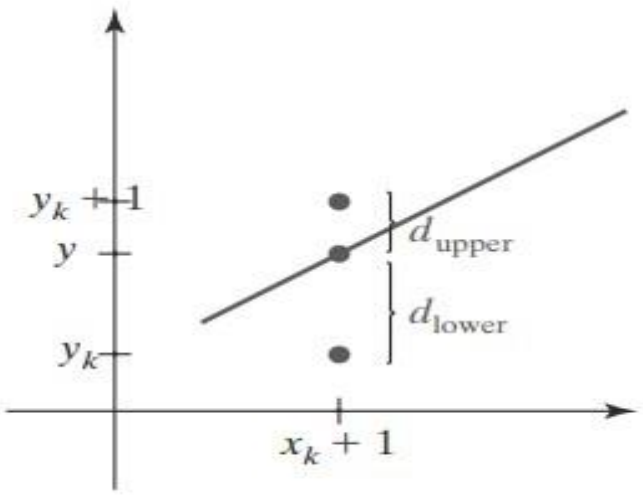

Figure 1.25: Vertical distances between pixel positions and the line y coordinate at sampling position $x_k + 1$

At sampling position $x_k + 1$, we label vertical pixel separations from the mathematical line path as d_{lower} and d_{upper}

The y coordinate on the mathematical line at pixel column position $x_k + 1$ is calculated as

$y=m(x_k+1)+b$

$d_{lower}=y-y_k$

$d_{lower}=m(x_k+1)+b-y_k$

$d_{upper}=(y_k+1)-y$ d_{upper}

$=(y_k+1)-m(x_k+1)-b$

The difference between two separations is d_{lower} -

$d_{upper} = m(x_k+1)+b-y_k - ((y_k+1)-m(x_k+1)-b))$ **d_{lower}-**

$d_{upper} = 2m(x_k+1)-2y_k +2b-1$ (1)

A decision parameter p_k for the k^{th} step in the line algorithm can be obtained by rearranging Equation 1 so that it involves only integer calculations.

By substituting $m=\Delta y/\Delta x$

The decision parameter is defined as

$p_k = \Delta x(d_{lower} - d_{upper})$ $p_k=\Delta x$

$(2(\Delta y/\Delta x) (x_k+1)-2y_k +2b-1)$

$p_k=2\Delta y(x_k+1) - 2 \Delta x y_k + \Delta x (2b-1)$

$p_k=2\Delta y x_k +2\Delta y - 2 \Delta x y_k + \Delta x (2b-$

1) $p_k=2\Delta y x_k - 2 \Delta x y_k +2\Delta y + \Delta x$

(2b-1) **$p_k=2\Delta y x_k - 2 \Delta x y_k +c$ (2)**

Parameter c is constant and has value $2\Delta y + \Delta x (2b-1)$

The parameter c is independent of pixel position (i.e x_k and y_k) and hence eliminated during calculation of p_k

The sign of p_k is same as sign of $d_{lower}- d_{upper}$

If pixel at y_k is closer to the line path than the pixel at y_k+1 (i.e $d_{lower} < d_{upper}$), then decision parameter p_k is negative. In such case lower pixel is plotted.

Otherwise upper pixel is plotted. At

step k + 1, the decision parameter is

$p_{k+1}=2\Delta y x_{k+1} - 2 \Delta x y_{k+1} +c$ (3)

(3)-(2) $p_{k+1}- p_k =2\Delta y(x_{k+1}-x_k)-2\Delta x(y_{k+1}-y_k)$ But

$x_{k+1}=x_k+1$ $p_{k+1}= p_k+2\Delta y(x_k+1-x_k)-2\Delta x(y_{k+1}-y_k)$ //

$(x_k+1-x_k)=1$ $p_{k+1}= p_k+2\Delta y-2\Delta x(y_{k+1}-y_k)$

$p_{k+1}= p_k+2\Delta y-2\Delta x(y_{k+1}-y_k)$

The term $y_{k+1}- y_k$ is either 0 or 1 depending on the sign of parameter p_k The

first parameter p_0 at the starting pixel position (x_0,y_0) is

$p_0=2\Delta y-\Delta x$

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

1. Input the two line endpoints and store the left endpoint in (x_0, y_0)
2. Set the color for frame-buffer position (x_0, y_0); i.e., plot the first point
3. Calculate the constants Δx, Δy, $2\Delta y$, and $2\Delta y - 2\Delta x$, and obtain the starting value for the decision parameter as **$p_0 = 2\Delta y-\Delta x$**
4. At each x_k along the line, starting at k = 0, perform the following test: If $p_k < 0$, the next point to plot is $(x_k + 1, y_k)$ and **$p_{k+1} = p_k + 2\Delta y$**

   Otherwise, the next point to plot is $(x_k + 1, y_k + 1)$ and **p_{k+1}**

   $= p_k + 2\Delta y - 2\Delta x$
5. Repeat step 4 Δx - 1 more times.

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);
}
}
```

Example:

Digitize the line with endpoints (20, 10) and (30, 18).

Slope=(18-10)/(30-20)=8/10=0.8 The

initial decision parameter has the value

$p_0=2\Delta y- \Delta x=6$

Increments for calculating successive decision parameters are

$2\Delta y=16$

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

We plot the initial point ($x0$, $y0$) = (20, 10), and determine successive pixel positions along the line path from the decision parameter as follows:

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)

A plot of the pixels generated along this line path is shown in following figure.

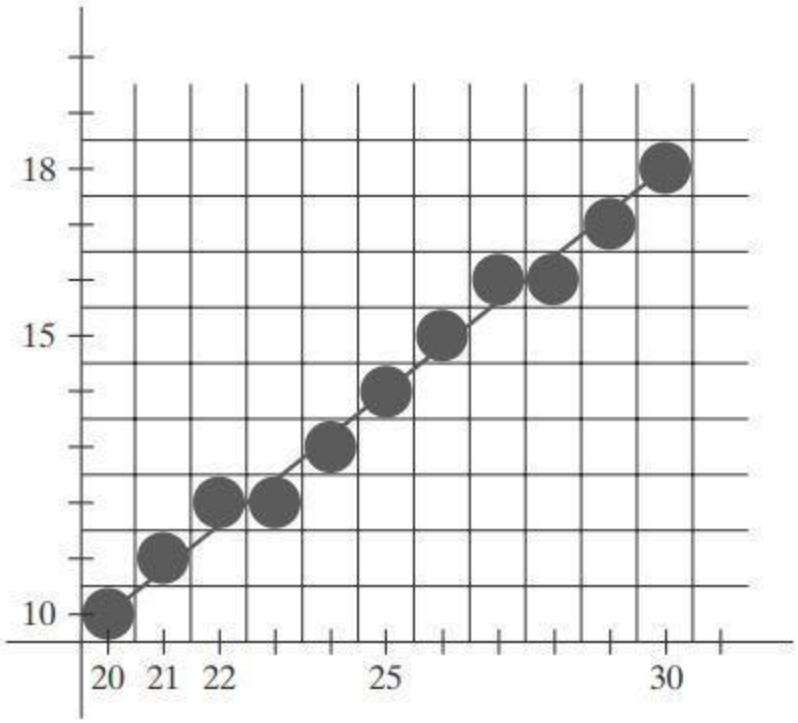

Figure 1.26: Pixel positions along the line path between endpoints (20, 10) and (30, 18), plotted with Bresenham's line algorithm.

VTU Question Paper Questions

1. With neat diagram, explain the basic design and operation of cathode ray tube.
2. Write Bresenham's line drawing algorithm for m < 1.0 . Digitize the line with endpoints (20,10) and (30,18)
3. Describe various applications of computer graphics with appropriate examples.
4. With necessary steps, explain Bresenham's line drawing algorithm. Consider the line from (5,5) to (13,9), use the bresenham's algorithm to rasterize the line.
5. Compare random scan display with raster scan display and list the applications of computer graphics.
6. What is OpenGL ? With the help of block diagram explain library organization of openGL program and give the general structure of OpenGL program.

7. What is DDA? With the help of a suitable example demonstrate the working principle of Bresenham's line drawing algorithm for different slopes of a line.

8. Define the following terms with respect to computer graphics.
   i)Bitmap ii) Pixmap iii) Aspect ratio iv) Frame buffer 9.
   List and explain any six applications of computer graphics.

10. Explain refresh cathode ray tubes with diagram.

11. What is Computer Graphics ? Explain the applications of computer graphics?

12. Explain Bresenham's line drawing algorithm with an example?

13. Develop the code of the bresenham's line drawing algorithm.Also illustrate the algorithm, the line end points are (20,10) and (30,18).

14. Explain coordinate reference frames? How is a 2D world coordinate reference frame specified using OpenGL?

15. Digitize the line by using Bresenham's line drawing algorithm with end points (-2,5) and (5,12).List the drawbacks of DDA line drawing algorithm?

16. What is Computer Graphics? Mention the list of applications. How they are classified?

17. Write Bresenham's line drawing algorithm. Using bresenham's line drawing algorithm calculate the pixel positions for the screen coordinates (1,1) and (6,7).