MOUNT KENYA UNIVERSITY

Bachelor of Business Information Technology

BIT 4101
Computer Graphics

04-06-2012

Distance Learning Module

Table of Contents

1 A Survey of Computer Graphics

1.1 Introduction

Computers have become a powerful tool for the rapid and economical production of pictures. There is virtually no area in which graphical displays cannot be used to some advantage, and so it is not surprising to find the use of computer graphics so widespread. Although early applications in engineering and science had to rely on expensive and cumbersome equipment, advances in computer technology have made interactive computer graphics a practical tool. Today, we find computer graphics used routinely in such diverse areas as science, engineering, medicine, business, industry, government, art, entertainment, advertising, education, and training

Before we get into the details of how to do computer graphics, we first take a short tour through a gallery of Graphics Applications

1.2 Computer Aided Design

A major use of computer graphics is in design processes, particularly for engineering and architectural systems, but almost all products are now computer designed. Generally referred to as CAD, computer-aided design methods are now routinely used in the design of buildings, automobiles, aircraft, watercraft, spacecraft, computers, textiles, and many, many other products. For some design applications; object are first displayed in a wireframe outline form that shows the overall sham and internal features of objects. Wireframe displays also allow designers to quickly see the effects of interactive adjustments to design shapes. Figures 1 give examples of wireframe displays in design

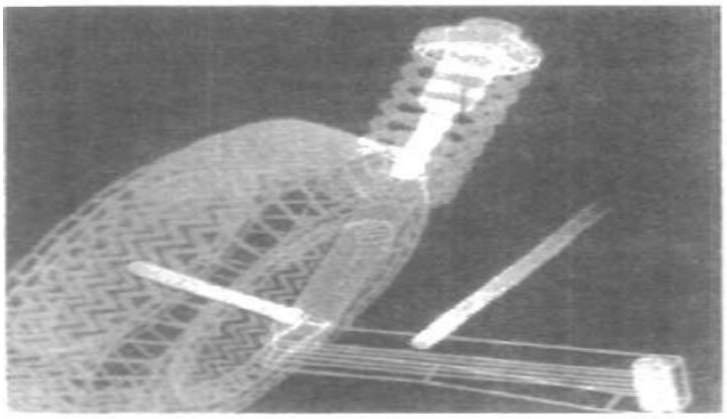

Figure 1
Color-coded wireframe display for an automobile wheel assembly. *(Courtesy of Evans and Sutherland.)*

Animations are often used in CAD applications. Real-time animations using wireframe displays on a video monitor are useful for testing performance of a vehicle or system, as demonstrated in Fig 1-2. When we do not display objects with rendered surfaces, the calculations for each segment of the animation can be performed quickly to produce a smooth real-time motion on the screen. Also, wireframe displays allow the designer to see into the interior of the vehicle and to watch the behavior of

inner components during motion. Animations in virtual reality environments are used to determine how vehicle operators are affected by certain motions.

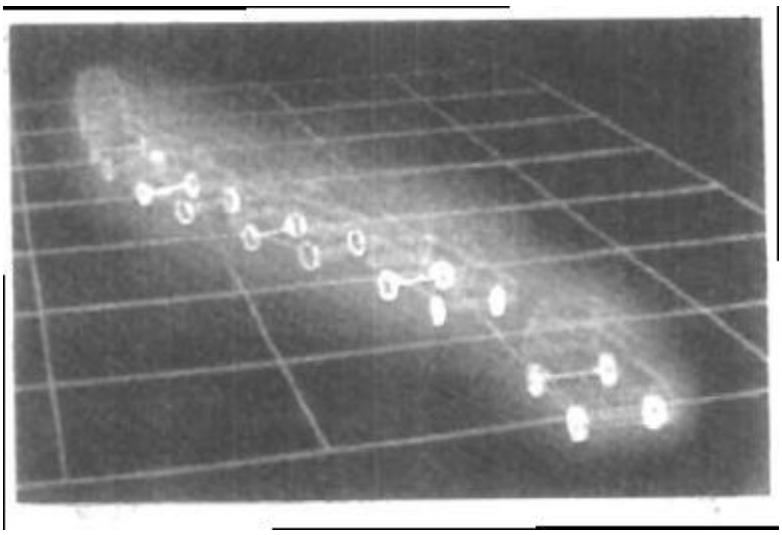

Figure 1-2 Simulation of vehicle performance during lane changes. *(Courtesy of Evans and Sutherland and Mechanical Dynamics, lnc.)*

1.3 Presentation Graphics

Another major application area is presentation graphics, used to produce illustrations for reports or to generate 35-mm slides or transparencies for use with projectors. Presentation graphics is commonly used to summarize financial, statistical, mathematical, scientific, and economic data for research reports, managerial reports, consumer information bulletins, and other types of reports. Workstation

Devices and service bureaus exist for converting screen displays into 35-mm slides or overhead transparencies for use in presentations. Typical examples of presentation graphics are bar charts, line graphs, surface graphs, pie charts, other displays showing relationships between multiple parameters.

Figure 1-3 gives examples of two-dimensional graphics combined with g geographical information. This illustration shows three color-coded bar charts combined onto one graph and a pie chart with three sections. Similar graphs and charts can be displayed in three dimensions to provide additional information.

Three-dimensional graphs are sometime used simply for effect; they can provide a more dramatic or more attractive presentation of data relationships. The charts in Fig. 1-4 include a three-dimensional bar graph and an exploded pie chart.

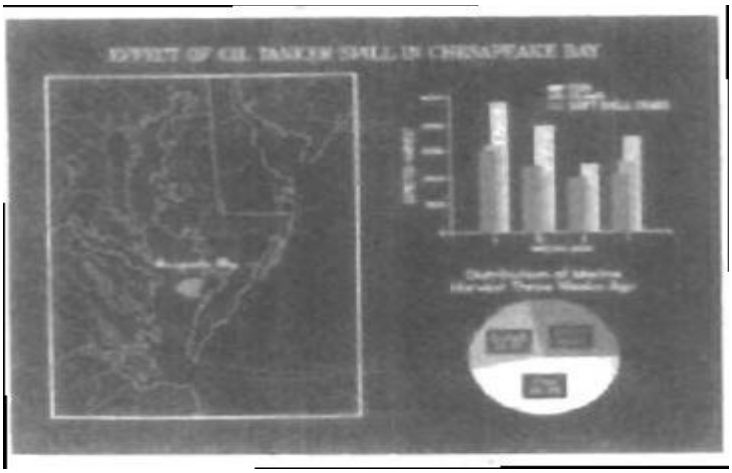

Figure 1-3

Two-dimensional bar chart and pie chart linked to a geographical chart. *(Courtesy of Computer Associates, copyright @1992: All rights reserved)*

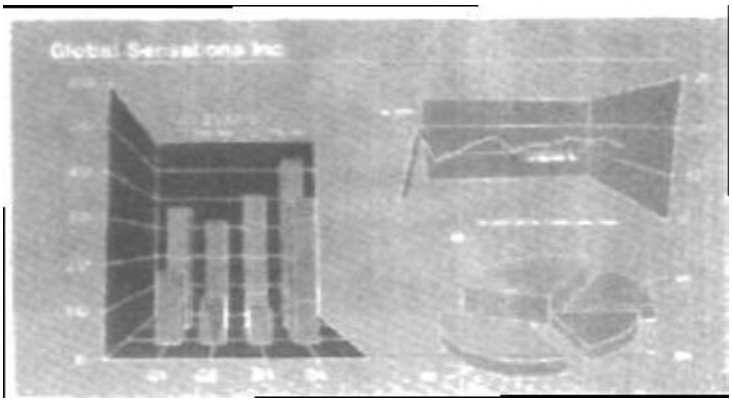

Figure 1-4

Three-dimensional bar chart, exploded pie chart, and line graph.(*Courtesy of Computer Associates, Copyright @ 1992: All rights reserved)*

1.4 Computer Art

Computer graphics methods are widely used in both fine art and commercial art applications. Artists use a variety of computer methods, including special-purpose e hardware, artist's paintbrush (such as Lumens), and other paint packages (such as Pixel paint and Super paint), specially developed software, symbolic mathematics packages (such as Mathematica), CAD packages, desktop publishing software, and animation packages that provide facilities for designing object shapes and specifying object motions. Figure 1-5 illustrates the basic idea behind a paintbrush program that allows artists to "paint" pictures on the screen of a video monitor. Actually, the picture is usually painted electronically on a graphics tablet (digitizer) using a stylus, which can simulate different brush strokes, brush widths, and colors. A paint brush program was used to m t e the characters in , who seem to be busy on a creation of their own.

A paintbrush system, with a Wacom cordless, pressure-sensitive stylus, was used to produce the electronic painting .The stylus translates changing hand pressure into variable line widths, brush sizes, and color gradations. A watercolor painting produced with this stylus and with software that allows the artist to create watercolor, pastel, or oil brush effects that simulate different drying out times, wetness, and footprint. Fine artists use a variety of other computer technologies to produce images. To create pictures the artist uses a combination of three-dimensional modeling packages, texture mapping, drawing programs, and CAD software.

Figure 1-5

Cartoon drawing produced with a paintbrush program, symbolically illustrating an artist at work on a video monitor. (*Courtesy of Gould Inc., Imaging 6 Graphics Division and Aurora Imaging*.)

1.5 Entertainment

Computer graphics methods are now commonly used in making motion pictures, music videos, and television shows. Sometimes the graphics scenes are displayed by themselves, and sometimes graphics objects are combined with the actors and live scenes. For example, the planet and spaceship are drawn in wireframe form and will be shared with rendering methods to produce solid surfaces. Figure 1-36 shows scenes generated with advanced modeling and surface rendering methods for two awards winning short film .Many TV series regularly employ computer graphics methods.

Music videos use graphic in several ways. Graphics objects can be combined with the live action, or graphics and image processing techniques can be used to produce a transformation of one person or object into another (morphing).

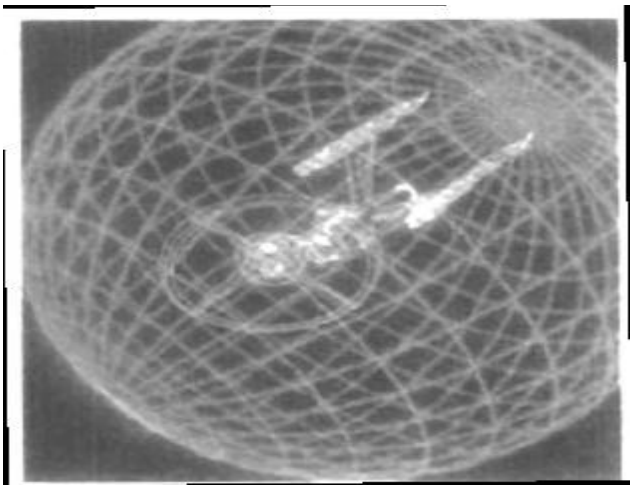

Figure 1-6

Graphics developed for the Paramount Pictures movie Starr Trek-The Wrath of Khan. (Courtesy of Evans & Sutherland.)

1.6 Education and Training

Computer-generated models of physical, financial, and economic systems are often used as educational aids. Models of physical systems, physiological systems, population trends, or equipment, such as the color coded diagram can help trainees to understand the operation of the system. For some training applications, special systems are designed. Examples of such specialized systems are the simulators for practice sessions or training of ship captains, aircraft pilots, heavy-equipment operators, and air traffic control personnel. Some simulators have no video screens; for example, a flight simulator with only a control panel for instrument flying. But most simulators provide graphics screens for visual operation.

Figure 1-7

An instructor's area in a flight simulator. The equipment allows the instructor to monitor flight conditions and to set airplane and environment parameters. (*Courtesy of Frasca International*.)

1.7 Visualization

Scientists, engineers, medical personnel, business analysts, and others often need to analyze large amounts of information or to study the behavior of certain processes. Numerical simulations carried out on supercomputers frequently produce data files containing thousands and even millions of data values. Similarly, satellite cameras and other sources are amassing large data files faster than they can be interpreted. Scanning these large sets of numbers to determine trends and relationships is a tedious and ineffective process. But if the data are converted to a visual form, the trends and patterns are often immediately apparent. Figure 1-8 shows an example of a large data set that has been converted to a color-coded display of relative heights above a ground plane. Once we have plotted the density values in this way, we can see easily the overall pattern of the data. Producing graphical representations for scientific, engineering, and medical data sets and processes 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 non-scientific areas.

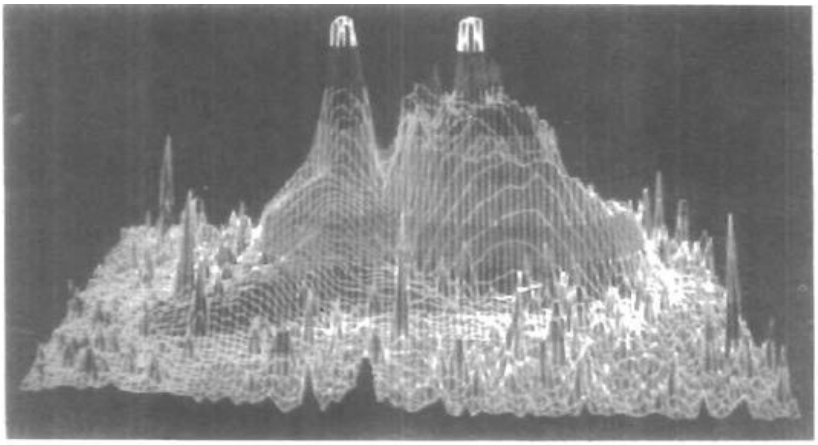

Figure 1-8

A color-coded plot with 16 million density points of relative observed from the Whirlpool Nebula reveals two distinct galaxies.(Courtesy of Los AIamos National Laboratory.)

1.8 Image Processing

Although methods used in computer graphics and Image processing overlap, the two areas are concerned with fundamentally different operations. In computer graphics, a computer is used to create a picture. Image processing, on the other hand, applies techniques to modify or interpret existing pictures, such as photographs' and TV scans.

Two principal applications of image processing are (1) improving picture quality and (2) machine perception of visual information, as used in robotics. To apply image processing methods, we first digitize a photograph or other picture into an image file. Then digital methods can be applied to rearrange picture parts, to enhance color separations, or to improve the quality of shading. An example of the application of image processing methods is to enhance the quality of a picture. These techniques are used extensively in commercial art applications that involve the retouching and rearranging of

sections of photographs and other artwork. Similar methods are used to analyze satellite photos of the earth and photos of galaxies. Medical applications also make extensive use of image processing techniques for picture enhancements, in tomography and in simulations of operations. Tomography is a technique of X-ray photography that allows cross-sectional views of physiological systems to be displayed. Both computed X-ray tomography (CT) and position emission tomography (PET) use tomography methods to reconstruct cross sections from digital data.

1.9 Graphical User Interfaces

It is common now for software packages to provide a graphical interface. A major component of a graphical interface is a window manager that allows a user to display multiple-window areas. Each window can contain a different process that can contain graphical or non-graphical displays. To make a particular window active, we simply click in that window using an interactive pointing device. Interfaces also display menus and icons for fast selection of processing options or parameter values. An icon is a graphical symbol that is designed to look like the processing option it represents. The advantages of icons are that they take up less screen space than corresponding textual descriptions and they can be understood more quickly if well designed. Menus contain lists of textual descriptions and icons.

2 Overview of Graphics Systems

2.1 Introduction

Due to the widespread recognition of the power and utility of computer graphics in virtually all fields, a broad range of graphics hardware and software systems is now available. Graphics capabilities for both two-dimensional and three-dimensional applications a x now common on general-purpose computers, including many hand-held calculators. With personal computers, we can use a wide variety of interactive input devices and graphics software packages. For higher quality applications, we can choose from a number of sophisticated special-purpose graphics hardware systems and technologies. In this chapter, we explore the basic features of graphics hardware components and graphics software packages.

2.2 Video Display Devices

Typically, the primary output device in a graphics system is a video monitor (Fig.2-1). The operation of most video monitors is based on the standard cathode-ray tube (CRT) design, but several other technologies exist and solid-state monitors may eventually predominate.

2.2.1 Raster-Scan Systems

Interactive raster graphics systems typically employ several processing units. In addition to the central processing unit, or CPU, a special-purpose processor, called the video controller or display controller, is used to control the operation of the display device. Organization of a simple raster system is shown in Fig. 2-1. Here, the frame buffer can be anywhere in the system memory, and the video controller accesses the frame buffer to refresh the screen. In addition to the video controller, more sophisticated raster systems employ other processors as coprocessors and accelerators to implement various graphics operations. Video Controller Figure 2-2 shows a commonly used organization for raster systems. A fixed area of the system memory is reserved for the frame buffer, and the video controller is given direct access to the frame-buffer memory. Frame-buffer locations, and the corresponding screen positions, are referenced in Cartesian coordinates. For many graphics monitors, the coordinate originate at the lower left screen comer.

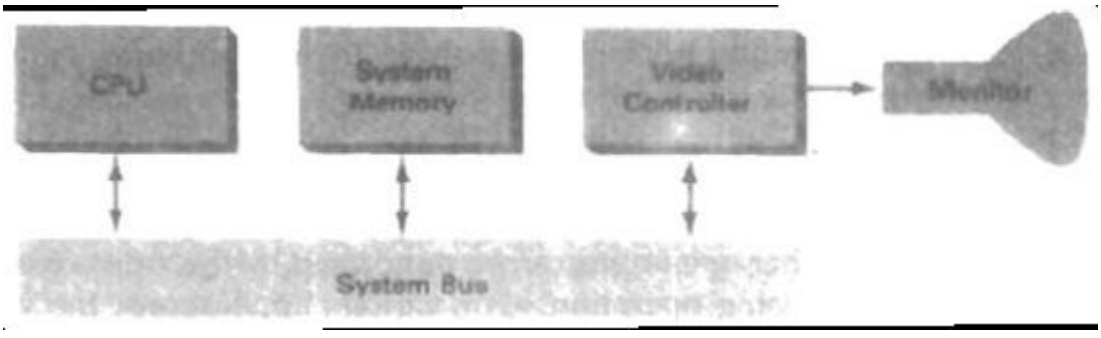

Figure 2-1

Architecture of a simple raster graphics system.

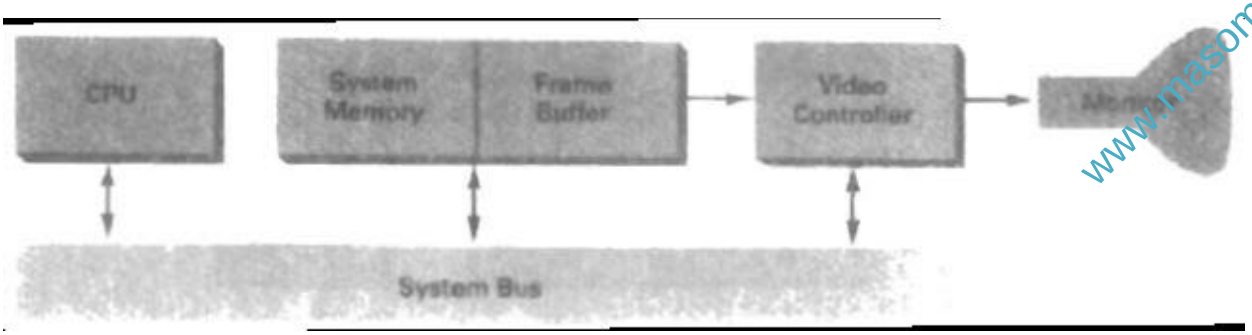

Figure 2-2

Architecture of a raster system with a fixed portion of the system memory reserved for the frame buffer.

2.3 Graphics Monitors and Workstations

Most graphics monitors today operate as raster scan displays, and here we survey a few of the many graphics hardware configurations available. Graphics systems range from small general-purpose computer systems with graphics capabilities to sophisticated full color systems that are designed specifically for graphics applications. A typical screen resolution for personal computer systems, such as the Apple Quadra. Overview of Graphics Systems though screens resolution and other system capabilities vary depending on the size and cost of the system. Diagonal screen dimensions for general-purpose personal computer systems can range from 12 to 21 inches, and allowable color selections range from 16 to over 32,000. For workstations specifically designed for graphics applications, typical screen resolution is 1280 by 1024, with a screen diagonal of 16 inches or more. Graphics workstations can be configured with from 8 to 24 bits per pixel (full-color systems), with higher screen resolutions, faster processors, and other options available in high-end systems

2.4 Input Devices

Various devices are available for data input on graphics workstations. Most systems have a keyboard and one or more additional devices specially designed for interactive input. These include a mouse, trackball, space ball, joystick, digitizers, dials, and button boxes. Some other input devices used in particular applications are data gloves, touch panels, image scanners, and voice systems.

2.4.1 Keyboards

An alphanumeric keyboard on a graphics system is used primarily as a device for entering text strings. The keyboard is an efficient device for inputting such nongraphic data as picture labels associated with a graphics display. Keyboards can also be provided with features to facilitate entry of screen coordinates, menu selections, or graphics functions

2.4.2 Mouse

A mouse is small hand-held box used to position the screen cursor. Wheels or rollers on the bottom of the mouse can be used to record the amount and direction of movement. Another method for detecting mouse motion is with an optical sensor. For these systems, the mouse is moved over a special mouse

pad that has a grid of horizontal and vertical lines. The optical sensor detects movement across the lines in the grid.

2.4.3 Joysticks

A joystick consists of a small, vertical lever (called the stick) mounted on a base that is used to steer the screen cursor around. Most by sticks select screen positions with actual stick movement; others respond to inksure on the stick. F I ~2-44 shows a movable joystick. Some joysticks are mounted on a keyboard; others function as stand-alone units. The distance that the stick is moved in any direction from its center position corresponds to screen-cursor movement in that direction. Potentiometers mounted at the base of the joystick measure the amount of movement, and springs return the stick to the center position when it is released. One or more buttons can be programmed to act as input switches to signal certain actions once a screen position has been selected

2.4.4 Data Glove

Data glove that can be used to grasp a "virtual" object. The glove is constructed with a series of sensors that detect hand and finger motions. Electromagnetic coupling between transmitting antennas and receiving antennas is used to provide information about the position and orientation of the hand. The transmitting and receiving antennas can each be structured as a set of three mutually perpendicular coils, forming a three-dimensional Cartesian coordinate system. Input from the glove can be used to position or manipulate objects in a virtual scene. A two-dimensional projection of the scene can be viewed on a video monitor, or a three-dimensional projection can be viewed with a headset.

2.4.5 Digitizers

A common device for drawing, painting, or interactively selecting coordinate positions on an object is a digitizer. These devices can be used to input coordinate values in either a two-dimensional or a three-dimensional space. Typically, a digitizer is used to scan over a drawing or object and to input a set of discrete coordinate positions, which can be joined with straight-line segments to approximate the curve or surface shapes. One type of digitizer is the graphics tablet (also referred to as a data tablet), which is used to input two-dimensional coordinates by activating a hand cursor or stylus at selected positions on a flat surface.

2.5 Hard-Copy Devices

We can obtain hard-copy output for our images in several formats. For presentations or archiving, we can send image files to devices or service bureaus that will produce 35-mm slides or overhead transparencies. To put images on film, we can simply photograph a scene displayed on a video monitor. And we can put our pictures on paper by directing graphics output to a printer or plotter. The quality of the pictures obtained from a device depends on dot size and the number of dots per inch, or Lines per inch, that can be displayed. To produce smooth characters in printed text strings, higher-quality printers shift dot positions so that adjacent dots overlap.

Printers produce output by either impact or nonimpact methods. Impact printer’s press formed character faces against an inked ribbon onto the paper. A line printer is an example of an impact device, with the typefaces mounted on bands, chains, drums, or wheels. Nonimpact printers and plotters use

laser techniques, ink-jet sprays, xerographic presses' (as used in photocopying machines),electrostatic methods, and electro thermal methods to get images onto Paper.

Character impact printers often have a dot-matrix print head containing a rectangular array of protruding wire pins, with the number of pins depending on the quality of the printer. Individual characters or graphics patterns are obtained by retracting certain pins so that the remaining pins form the pattern to be printed. Figure 2-3 shows a picture printed on a dot-matrix printer.

In a laser device, a laser beam mates a charge distribution on a rotating drum coated with a photoelectric material, such as selenium. Toner is applied to the drum and then transferred to paper. Figure 2-59 shows examples of desktop

Laser printers with a resolution of 360 dots per inch. Ink-jet methods produce output by squirting ink in horizontal rows across a roll of paper wrapped on a drum. The electrically charged ink stream is deflected by an electric field to produce dot-matrix patterns. A desktop ink-jet plotter with

Figure 2-3

A picture generated on a dot-matrix printer showing how the density of the dot patterns can be varied to produce light and dark areas. (*Courtesy of Apple Computer, Inc.)*

Drafting layouts and other drawings are typically generated with ink-jet or pen plotters. A pen plotter has one or more pens mounted on a carriage, or crossbar, that spans a sheet of paper. Pens with varying colors and widths are used to produce a variety of shadings and line styles. Wet-ink, ball-point, and felt-tip pens are all possible choices for use with a pen plotter. Plotter paper can lie flat or be rolled onto a drum or belt. Crossbars can be either moveable or stationary, while the pen moves back and forth along the bar. Clamps, a vacuum, or an electrostatic charge hold the paper in position. An example of a table-top flatbed pen plotter is given in Fig 2-4, and a larger, rollfeed pen plotter is shown in Fig. 2-5.

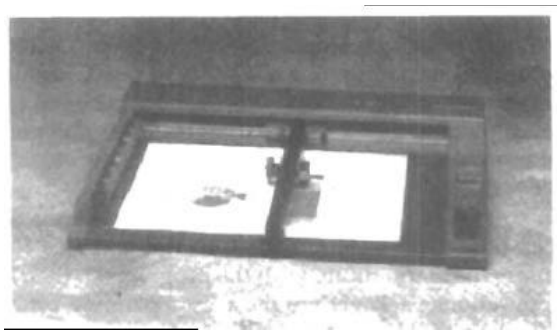

Figure 2-4

A desktop pen plotter with a resolution of 0.025 mm. (Courtesy of Summagraphics Corporation)

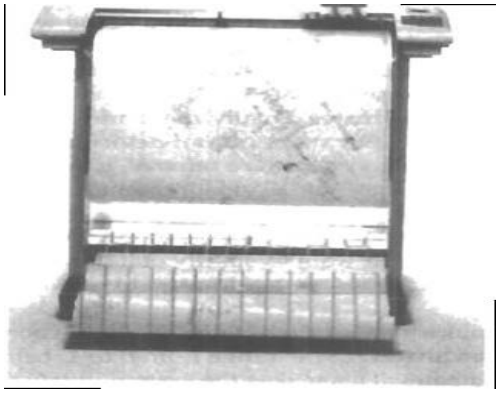

Figure 2-5

A large, rollfeed pen plotter with automatic multicolor-pen changer and a resolution of 0.0127 mm. (Courtesy of Summagraphics Corporation)

2.6 Graphics Software

There are two general classifications for graphics software: general programming packages and special-purpose applications packages. A general graphics programming package provides an extensive set of graphics functions that can be used in a high-level programming language, such as C or FORTRAN. An example of a general graphics programming package is the GL (Graphics Library) system on Silicon Graphics equipment. Basic functions in a general package include those for generating picture components (straight lines, polygons, circles, and other figures), setting color and intensity values, selecting views, and applying transformations. By contrast, application graphics packages are designed for nonprogrammers, so that users can generate displays without worrying about how graphics operations work. The interface to the graphics routines in such packages allows users to communicate with the programs in their own terms. Examples of such applications packages are the artist's painting programs and various business, medical, and CAD systems.

2.6.1 Coordinate Representations

With few exceptions, general graphics packages are designed to be used with Cartesian coordinate specifications. If coordinate values for a picture are specified in some other reference frame (spherical, hyperbolic, etc.), they must be converted to Cartesian coordinates before they can be input to the graphics package. Special-purpose packages may allow use of other coordinate frames that are appropriate to the application. In general; several different Cartesian reference frames are used to construct and display a scene. We can construct the shape of individual objects, such as trees or furniture, in a scene within separate coordinate reference frames called modeling coordinates, or sometimes local coordinates or master coordinates. Once individual object shapes have been specified, we can place the objects into appropriate positions within the scene using a reference frame called world coordinates. Finally, the world-coordinate description of the scene is transferred to one or more output-device reference frames for display. These display coordinate systems are referred to as device coordinates, or screen coordinates in the case of a video monitor. Modeling and world coordinate definitions allow us to set any convenient floating-point or integer dimensions without being hampered by the constraints of a particular output device. For some scenes, we might want to specify object dimensions in fractions of a foot, while for other applications we might want to use millimeters, kilometers, or light-years.

Generally, a graphics system first converts world-coordinate positions to normalized device coordinates, in the range from 0 to 1, before final conversion to specific device coordinates. This makes the system independent of the various devices that might be used at a particular workstation. Figure 2-65 illustrates the sequence of coordinate transformations from modeling coordinates to device coordinates for a two-dimensional application. An initial modeling-coordinate position (X_{mcr}, Y_{mc}) in this illustration is transferred to a device coordinate position (X_{dcr}, Y_{dc}) with the sequence:

$$(X_{mcr}, Y_{mc}) \rightarrow (X_{wcr}, Y_{wc}) \rightarrow (X_{ncr}, Y_{nc}) \rightarrow (X_{dcr}, Y_{dc})$$

The modeling and world-coordinate positions in this transformation can be any floating-point values; normalized coordinates satisfy the inequalities: $0 \leq x_{nc} \leq 1$, $0 \leq y_{nc} \leq 1$, and the device coordinates x_{dc} and y_{dc} are integers within the range (0, 0) to (x_{maxr},y_{max}) for a particular output device. To accommodate differences in scales and aspect ratios, normalized coordinates are mapped into a square area of the output device so that proper proportions are maintained.

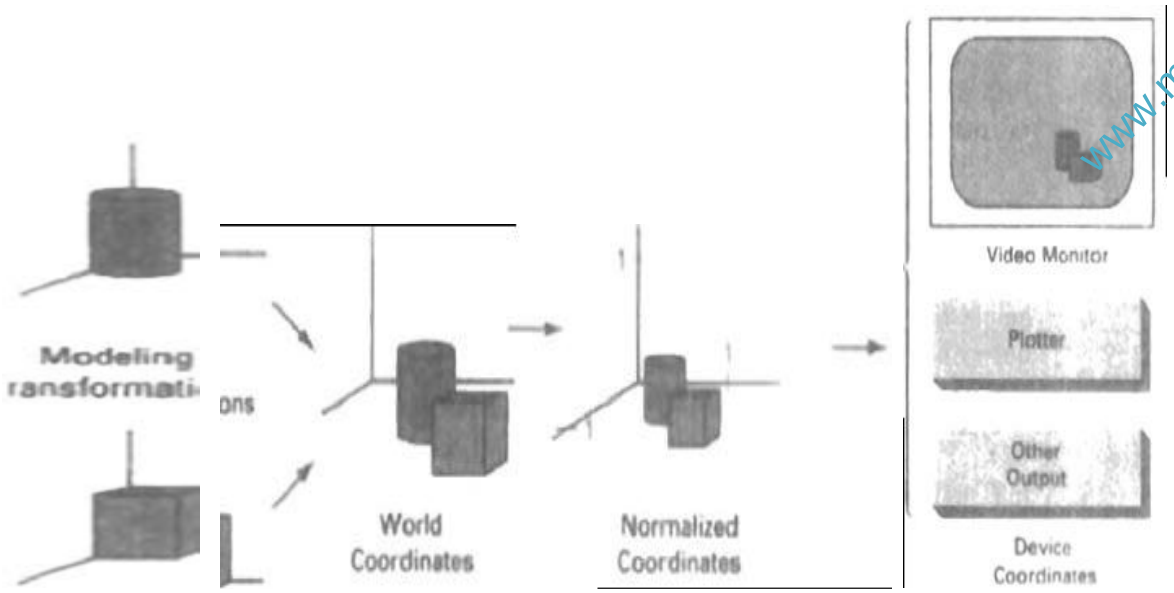

Figure 2-65

The transformation sequence from modeling coordinates to device coordinates for a two dimensional scene. Object shapes are defined in local modeling-coordinate systems, and then positioned within the overall world-coordinate scene. World-coordinate specifications are then transformed into normalized coordinates. At the final step, individual device drivers transfer the normalized coordinate representation of the scene to the output devices for display

2.7 Graphics Functions

A general-purpose graphics package provides users with a variety of functions for creating and manipulating pictures. These routines can be categorized according to whether they deal with output, input, attributes, transformations, viewing, or general control.

The basic building blocks for pictures are referred to as output primitives. They include character strings and geometric entities, such as points, straight lines, curved Lines, filled areas (polygons, circles, etc.), and shapes defined with arrays of color points. Routines for generating output primitives provide the basic tools for constructing pictures.

Attributes are the properties of the output primitives; that is, an attribute describes how a particular primitive is to be displayed. They include intensity and color specifications, line styles, text styles, and area-filling patterns. Functions within this category can be used to set attributes for an individual primitive class or for groups of output primitives.

We can change the size, position, or orientation of an object within a scene using geometric transformations. Similar modeling transformations are used to construct a scene using object descriptions given in modeling coordinates.

Given the primitive and attribute definition of a picture in world coordinates, a graphics package projects a selected view of the picture on an output device. Viewing transformations are used to specify the view that is to be presented and the portion of the output display area that is to be used.

Pictures can be subdivided into component parts, called structures or segments or objects, depending on the software package in use. Each structure defines one logical unit of the picture. A scene with several objects could reference each individual object in a-separate named structure. Routines for processing structures carry out operations such as the creation, modification, and transformation of structures.

Interactive graphics, applications use various kinds of input devices, such as a mouse, a tablet, or a joystick. Input functions are used to control and process the data flow from the interactive devices.

Finally, a graphics package contains a number of housekeeping tasks, such as clearing a display screen and initializing parameters, we can lump the functions for carrying out the chores under the heading control operations.

2.8 Software Standards

The primary goal of standardized graphics software is portability. When packages are designed with standard graphics functions, software can he moved easily from one hardware system to another and used in different implementations and applications. Without standards, programs designed for one hardware system often cannot be transferred to another system without extensive rewriting of the programs.

International and national standards planning organizations in many countries have cooperated in an effort to develop a generally accepted standard for computer graphics. After considerable effort, this work on standards led to the development of the Graphical Kernel System (GKS). This system was adopted as the first graphics software standard by the International standards Organization (ISO) and by various national standards organizations, including the American National Standards Institute (ANSI). Although GKS was originally designed as a two-dimensional graphics package, a three-dimensional GKS extension was subsequently developed. The second software standard to be developed and approved by the standards organizations was PHIGS (Programmer's Hierarchical Interactive Graphics standard), which is an extension of GKS. Increased capabilities for object modeling, color specifications, surface rendering, and picture manipulations are provided In PHIGS. Subsequently, an extension of PHIGS, called PHIGS+, was developed to provide three-dimensional surface-shading capabilities not available in PHIGS. Standard graphics functions are defined as a set of specifications that is Independent of any programming language. A language binding is then defined for a particular higher-level programming language. This binding gives the syntax for accessing the various standard graphics functions from this language. For example, the general form of the PHIGS (and GKS) function for specifying a sequence of n-1 connected two-dimensional straight Iine segments is

Polyline(n,x,y)

In FORTRAN, this procedure is implemented as a subroutine with the name GPL.A graphics programmer, using FORTRAN, would invoke this procedure with the subroutine call statement CALL GPL (N , X, Y), where X and Y are one dimensional arrays of coordinate values for the line endpoints. In C, the procedure would be invoked with ppclyline (n, pts), where pts is the list of coordinate endpoint positions. Each language binding is defined to make best use of the corresponding language capabilities and to handle various syntax issues, such as data types, parameter passing, and errors.

Although PHIGS presents a specification for basic graphics functions, it does not provide a standard methodology for a graphics interface to output devices. Nor does it specify methods for storing and transmitting pictures. Separate standards have been developed for these areas. Standardization for device interface methods is given in the Computer Graphics Interface (CGI) system. And the Computer Graphics Metafile (CGM) system specifies standards for archiving and transporting pictures.

3 Output Primitives

A picture can be described in several ways. Assuming we have a raster display, a picture is completely specified by the set of intensities for the pixel positions in the display. At the other extreme, we can describe a picture as a set of complex objects, such as trees and terrain or furniture and walls, positioned at specified coordinate locations within the scene. Shapes and colors of the objects can be described internally with pixel arrays or with sets of basic geometric structures, such as straight line segments and polygon color areas. The scene is then displayed either by loading the pixel arrays into the frame buffer or by scan converting the basic geometric-structure specifications into pixel patterns. Typically, graphics programming packages provide functions to describe a scene in terms of these basic geometric structures, referred to as output primitives, and to group sets of output primitives into more complex structures.

Each output primitive is specified with input coordinate data and other information about the way that object is to be displayed. Points and straight line segments are the simplest geometric components of pictures. Additional output primitives that can be used to construct a picture include circles and other conic sections, quadric surfaces, spline curves and surfaces, polygon color areas, and character strings. We begin our discussion of picture-generation procedures by examining device-level algorithms for displaying two-dimensional output primitives, with particular emphasis on scan-conversion methods for raster graphics systems. In this chapter, we also consider how output functions can be provided in graphics packages, and we take a look at the output functions available in the PHIGS language.

3.1 Points and Lines

Point plotting is accomplished by converting a single coordinate position furnished by an application program into appropriate operations for the output device in use. With a CRT monitor, for example, the electron beam is turned on to illuminate the screen phosphor at the selected location. How the electron beam is positioned depends on the display technology. A random-scan (vector) system stores point-plotting instructions in the display list, and coordinate values in these instructions are converted to deflection voltages that position the electron beam at the screen locations to be plotted during each refresh cycle. For a black and-white raster system, on the other hand, a point is plotted by setting the bit value corresponding to A specified screen position within the frame buffer to 1. Then, as the electron beam sweeps across each horizontal scan line, it emits a burst of electrons (plots a point) whenever a value of I is encountered in the frame buffer. With an RGB system, the frame buffer is loaded with the color Points and lines codes for the intensities that are to be displayed at the screen pixel positions.

Line drawing is accomplished by calculating intermediate positions along the line path between two specified endpoint positions. An output device is then directed to fill in these positions between the endpoints. For analog devices, such as a vector pen plotter or a random-scan display, a straight line can be drawn smoothly from one endpoint to the other. Linearly varying horizontal and vertical deflection voltages are generated that are proportional to the required changes in the x and y directions to produce the smooth line.

Digital devices display a straight line segment by plotting discrete points between the two endpoints. Discrete coordinate positions along the line path are calculated from the equation of the line. For a

raster video display, the line color (intensity) is then loaded into the frame buffer at the corresponding pixel coordinates. Reading from the frame buffer, the video controller then "plots" the screen pixels. Screen locations are referenced with integer values, so plotted positions may only approximate actual Line positions between two specified endpoints. A computed line position of (10.48, 20.51), for example, would be converted to pixel position (10, 21). This rounding of coordinate values to integers causes lines to be displayed with a stairstep appearance ("the jaggies"), as represented in Fig 3-1.The characteristic stairstep shape of raster lines is particularly noticeable on system with low resolution, and we can improve their appearance somewhat by displaying them on high-resolution systems. More effective techniques for smoothing raster lines are based on adjusting pixel intensities along the line paths.

For the raster-graphics device-level algorithms discussed in this chapter, object positions are specified directly in integer device coordinates. For the time being, we will assume that pixel positions are referenced according to scan-line number and column number (pixel position across a scan line). This addressing scheme is illustrated in Fig. 3-2. Scan lines are numbered consecutively from 0, starting at the bottom of the screen; and pixel columns are numbered from 0, left to right across each scan line. In Section 3-10, we consider alternative pixel addressing schemes.

To load a specified color into the frame buffer at a position corresponding to column x along scan line y, we will assume we have available a low-level procedure of the form

SetPixel(x,y)

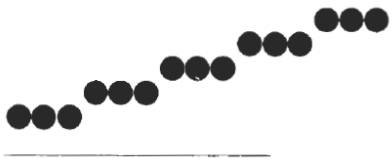

Figure 3-1

Stairstep effect (jaggies) produced when a line is generated as a series of pixel positions

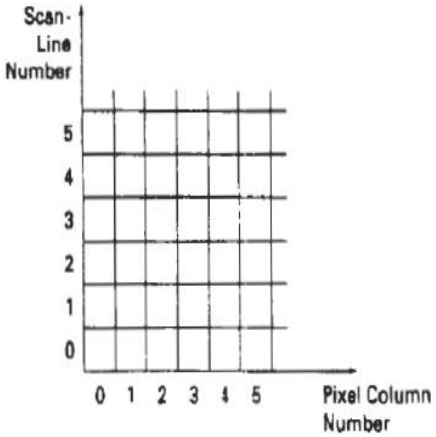

Figure 3-2
Pixel positions referenced by scan-line number and column number.

We sometimes will also want to be able to retrieve the current frame buffer intensity setting for a specified location. We accomplish this with the low-level function

GetPixel (x, y)

3.2 Line-Drawing Algorithms

The Cartesian slope-intercept equation for a straight line is

$$y=m.x+b \tag{3-1}$$

With m representing the slope of the line and b as they intercept. Given that the two endpoints of a h e segment are specified at positions (x_1, y_1) and (x_2, y_2), as shown in Fig. 3-3, we can determine values for the slope m and y intercept b with the following calculations.

$$m = \frac{y_2 - y_1}{x_2 - x_1} \tag{3-2}$$

$$b = y_1 - m \cdot x_1 \tag{3-3}$$

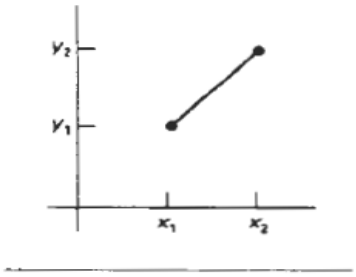

Figure 3-3 Lie path between endpoint positions $(x_1, y_1,)$ and (x_2, y_2).

Algorithms for displaying straight h e s are based on the line equation 3-1 and the calculations given in Eqs. 3-2 and 3-3.

For any given x interval Ax along a line, we can compute the corresponding y interval Δy from Eqs. 3-2 as

$$\Delta y=m\,\Delta x \tag{3-4}$$

Similarly, we can obtain the x interval Ax corresponding to a specified Ay as

$$\Delta x = \frac{\Delta y}{m} \tag{3-5}$$

These equations form the basis for determining deflection voltages in analog devices. For lines with slope magnitudes I m I < 1, Ax can be set proportional to a small horizontal deflection voltage and the corresponding vertical deflection is Line-Drawing Algorithms then set proportional to Ay as calculated from Eq. 3-4.

For lines whose slopes have magnitudes 1 m I > 1, Ay can be set proportional to a small vertical deflection voltage with the corresponding horizontal deflection voltage set proportional to Δx, calculated from Eq. 3-5. For lines with m = 1, Δx = Δy and the horizontal and vertical deflections voltages are equal. In each case, a smooth line with slope m is generated between the specified endpoints. On raster systems, lines are plotted with pixels, and step sizes in the horizontal and vertical directions are constrained by pixel separations. That is, we must "sample" a line at discrete positions and determine the nearest pixel to the line at each sampled position. This scan conversion process for straight lines is illustrated in Fig. 3-4, for a near horizontal line with discrete sample positions along the x axis.

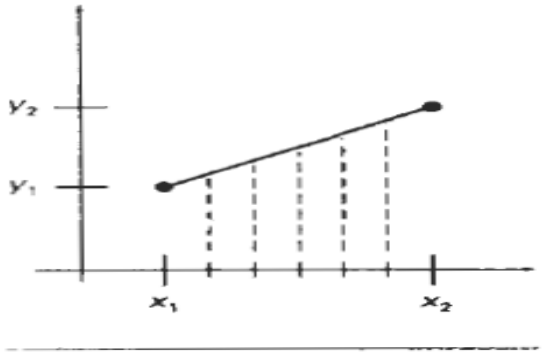

Fig 3-4 Straight line segment with five sampling positions along line path for the other coordinate along the x axis between x_1, and x_2.

3.2.1 DDA Algorithm

The digital differential analyzer (DDA) is a scan-conversion line algorithm based on calculating either Ay or Δx, using Eq. 3-4 or Eq. 3-5. We sample the line at unit in with intervals in one coordinate and determine corresponding integer values nearest the line path for the other coordinate. Consider first a line with positive slope, as shown in Fig. 3-3. If the slope is less than or equal to 1, we sample at unit x intervals (Δx = 1) and compute each successive y value as

$$y_{k+1} = y_k + m \qquad \textbf{(3-6)}$$

Subscript k takes integer values starting from 1, for the first point, and increases by 1 until the final endpoint is reached. Since n1 can be any real number between 0 and 1, the calculated y values must be rounded to the n e m t integer. For lines with a positive slope greater than 1, we reverse the roles of x and y. That is, we sample at unit y intervals (Δy = 1) and calculate each succeeding x value as

$$x_{k+1} = x_k + \frac{1}{m} \qquad (3\text{-}7)$$

Equations 3-6 and 3-7 are based on the assumption that lines are to be processed from the left endpoint to the right endpoint (Fig. 3-3). If this processing is reversed, so that the starting endpoint is at the right, then either we have Δx = -1 and

$$y_{k+1} = y_k - m \quad (3\text{-}8)$$

or (when the slope is greater than I) we have Ay = -1 with

$$x_{k+1} = x_k - \frac{1}{m} \quad (3\text{-}9)$$

Equations 3-6 through 3-9 can also be used to calculate pixel positions along a line with negative slope. If the absolute value of the slope is less than I and the start endpoint is at the left, we set Δx = 1 and calculate y values with Eq. 3-6.

When the start endpoint is at the right (for the same slope), we set Δx = -1 and obtain y positions from Eq. 3-8. Similarly, when the absolute value of a negative slope is water than 1, we use Δy = -1 and Eq. 3-9 or we use Δy = 1 and Eq. 3-7.

This algorithm is summarized in the following procedure, which accepts as input the two endpoints pixel positions. 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. Starting with pixel position (x_{a1} y_{a1}), we determine the offset needed at each step to generate the next pixel position along the line path. We loop through this process steps times. If the magnitude of dx is greater than the magnitude of dy and xa is less than xb, the values of the increments in the x and y directions are 1 and m, respectively. If the greater change is in the x direction, but xa is greater than xb, 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 l / m.

```
#include "device.h"

#define ROUND(a) ((int)(a+0.5))

void lineDDA (int xa, int ya, int xb, int yb)
{
  int dx = xb - xa, dy = yb - ya, steps, k;
  float xIncrement, yIncrement, x = xa, y = ya;

  if (abs (dx) > abs (dy)) steps = abs (dx);
  else steps = abs  dy);
  xIncrement = dx / (float) steps;
  yIncrement = dy / (float) steps;

  setPixel (ROUND(x), ROUND(y));
  for (k=0; k<steps; k++) {
    x += xIncrement;
    y += yIncrement;
    setPixel (ROUND(x), ROUND(y));
  }
}
```

The DDA algorithm is a faster method for calculating pixel positions than the direct use of Eq. 3-1. It eliminates the multiplication in Eq. 3-1 by making use of raster characteristics, so that appropriate increments are applied in the x or y direction to step to pixel positions along the line path. The accumulation of roundoff 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 procedure lineDDA are still time-consuming. We can improve the performance of the DDA algorithm by separating the increments m and l / m into integer and fractional parts so that all calculation & are reduced to integer operations. A method for calculating l / m increments in integer steps is discussed in Section 3-11. In the following sections, we consider more general scan-line procedures that can be applied to both lines and curves.

3.2.2 Bresenham's Line Algorithm

An accurate and efficient raster line-generating algorithm, developed by Bresenham scan converts lines using only incrementa1 integer calculations that can be adapted to display circles and other curves. Figures 3-5 and 3-6 illustrate sections of a display screen where straight line segments are to be drawn. The vertical axes show-scan-line positions, and the horizontal axes identify pixel columns. Sampling at unit x intervals in these examples, we need to decide which of two possible pixel positions is closer to the line path at each sample step. Starting from the left endpoint shown in Fig. 3-5, we need to determine at the next sample position whether to plot the pixel at position (11, 11) or the one at (11, 12). Similarly,

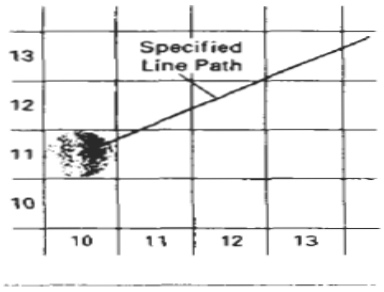

Figure 3-5 Section of a display screen where a straight line segment is to be plotted, starting from the pixel at column 10 on scan Line 11

Fig. 3-6 shows-a negative slope-line path starting from the left endpoint at pixel position (50, 50). In this one, do we select the next pixel position as (51,50) or as (51,49)? These questions are answered with Bresenham's line algorithm by testing the sign of an integer parameter, whose value is proportional to the difference between the separations of the two pixel positions from the actual line path.

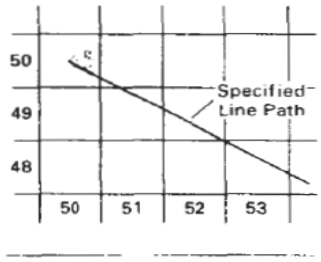

Figure 3-6 Section of a display screen where a negative slope line segment is to be plotted, starting from the pixel at column 50 on scan line 50.

To illustrate Bresenham approach, we- first consider the scan-conversion process for lines with positive slope less than 1. Pixel positions along a line path are then determined by sampling at unit x intervals. Starting from the left endpoint(x_0, y_0) 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. Figure 3-7 demonstrates the k^{th} step in this process. Assuming we have determined that the pixel at (x_k, y_k) is to be displayed, we next need to decide which pixel to plot in column x_{k+1}. Our choices are the pixels at positions (x_k+1, y_k) and (x_k+1, y_k+1).

At sampling position X_k+1, we label vertical pixel separations from the mathematical line path as d1, and d2 (Fig. 3-8). They coordinate on the mathematical line at pixel column position X_k+1 is calculated as

$$y = m(x_k + 1) + b \tag{3-10}$$

Then

$$d_1 = y - y_k$$

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

And

$$d_2 = y_k + 1) - y$$

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

The difference between these two separations is

$$d_1 - d_2 = 2m(x_k + 1) - 2y_k + 2b - 1 \tag{3-11}$$

A decision parameter p_k for the kth step in the line algorithm can be obtained by rearranging Eq. 3-11 so that it involves only integer calculations. We accomplish this by substituting m = ΔyIΔx, where Δy and Δx are the vertical and horizontal separations of the endpoint positions, and defining:

$$p_k = \Delta x(d_1 - d_2\ )$$

$$= 2\Delta y.\, x_k - 2\Delta x.\, y_k + c$$

The sign of p_k, is the same as the sign of $d_1 - d_2$ since $\Delta x > 0$ for our example. Parameter c is constant and has the value $2\Delta y - \Delta x$ (2b - l), which is independent of pixel position and will be eliminated in the recursive calculations for p_k. If the pixel at y_k is closer to the line path than the pixel at y_k+l (that is, $d_1 < d_2$), then decision parameter p_k is negative. In that case, we plot the lower pixel; otherwise, we plot the upper pixel.

Coordinate changes along the line occur in unit steps in either the x or y directions. Therefore, we can obtain the values of successive decision parameters using incremental integer calculations. At step k + 1, the decision parameter is evaluated from Eq. 3-12 as

$$p_{k+1} = 2\Delta y.\, x_k - 2\Delta x.\, y_k + c$$

Subtracting Eq. 3-12 from the preceding equation, we have

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

But $x_{k+1} = x_k + 1$, so that

$$p_{k+1} = p_k + 2\Delta y - 2\Delta x(y_{k+1} - y_k) \qquad (3\text{-}13)$$

Where the term y_{k+1}- y_k is either 0 or 1, depending on the sign of parameter p_k This recursive calculation of decision parameters is performed at each integer x position, starting at the left coordinate endpoint of the line. The first parameter, p,, is evaluated from Eq. 3-12 at the starting pixel position (x_0, y_0) and with m evaluated as $\Delta y/\Delta x$:

$$p_0 = 2\Delta y - \Delta x \qquad (3\text{-}14)$$

We can summarize Bresenham line drawing for a line with a positive slope less than 1 in the following listed steps. The constants $2\Delta y$ and $2\Delta y - 2\Delta x$ are calculated once for each line to be scan converted, so the arithmetic involves only integer addition and subtraction of these two constants

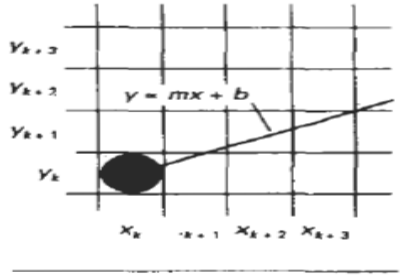

Figure 3-7 Section of the screen grid showing a pixel in column x_k on scan line y_k that is to be plotted along the path of a line segment with slope 0<m<1.

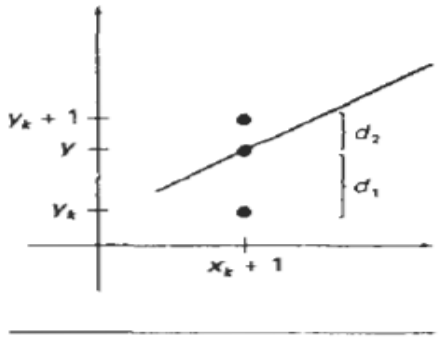

Figure 3-8 Distances between pixel positions and the line y coordinate at sampling position x_k+ I.

Bresenham's Line-Drawing Algorithm for $|m| < 1$

1. Input the two line endpoints and store the left endpoint in (x_0, y_0).
2. Load (x_0, y_0) into the frame buffer; that is, plot the first point.
3. Calculate constants Δx, Δy, $2\Delta y$, 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 times.

Example 3-1 Bresenham Line Drawing

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

$$\Delta x = 10$$

$$\Delta y = 8$$

The initial decision parameter has the value

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

$$=6$$

and the increments for calculating successive decision parameters are

$$2\Delta y = 16$$

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

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

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

A plot of the pixels generated along this line path is shown in Fig. 3-9.

An implementation of Bresenham line drawing for slopes in the range 0 <m < 1 is given in the following procedure. Endpoint pixel positions for the line are passed to this procedure, and pixels are plotted from the left endpoint to the right endpoint. The call to SetPixel loads a preset color value into the frame buffer at the specified (x, y) pixel position.

```
#include "device.h"

void lineBres (int xa, int ya, int xb, int yb)
{
   int dx = abs (xa - xb), dy = abs (ya - yb);
   int p = 2 * dy - dx;
   int twoDy = 2 * dy, twoDyDx = 2 * (dy - dx);
   int x, y, xEnd;

   /* Determine which point to use as start, which as end */
   if (xa > xb) {
      x = xb;
      y = yb;
      xEnd = xa;
   }
   else {
```

```
    x = xa;
    y = ya;
    xEnd = xb;
  }
  setPixel (x, y);

  while (x < xEnd) {
    x++;
    if (p < 0)
      p += twoDy;
    else {
      y++;
      p += twoDyDx;
    }
    setPixel (x, y);
  }
}
```

Bresenham's algorithm is generalized to lines with arbitrary slope by considering the symmetry between the various octants and quadrants of the xy plane. For a line with positive slope greater than 1, we interchange the roles of the x and y directions. That is, we step along they direction in unit steps and calculate successive x values nearest the line path. Also, we could revise the program to plot pixels starting from either endpoint. If the initial position for a line with positive slope is the right endpoint, both x and y decrease as we step from right to left. To ensure that the same pixels are plotted regardless of the starting endpoint, we always choose the upper (or the lower) of the two candidate pixels whenever the two vertical separations from the line path are equal (d_1, = d_2. For negative slopes, the procedures are similar, except that now one coordinates decreases as the other increases. Finally, specla1 cases can be handled separately: Horizontal lines (Δy = 0, vertical lines (Δr = 0), and diagonal lines with I Δy 1 =I Δy 1 each can be loaded directly into the frame buffer without processing them through the line-plotting algorithm.

4 Two dimensional Transformation