Computer Graphics involves display, manipulation and storage of pictures and experimental data for proper visualization using a computer.

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. Figure 1-1 summarizes the many applications of graphics in simulations, education, and graph presentations.

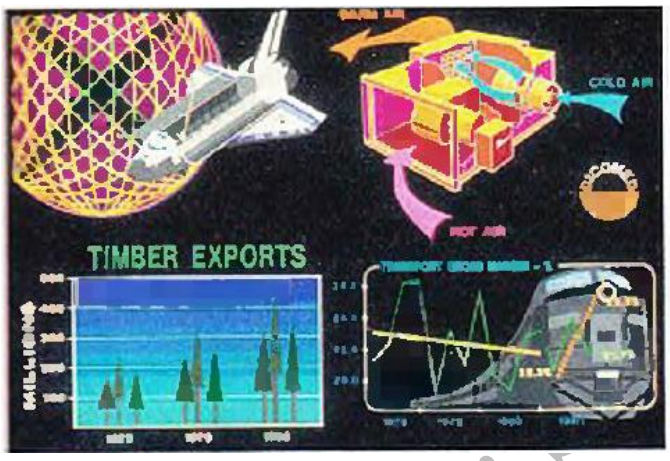


Figure 1.1
Examples of computer graphics applications.

Applications of Computer Graphics:

- Graphs and Arts
- Computer-Aided Design
- Virtual Reality Environments
- Data Visualizations
- Education and Training
- Computer Art
- Entertainment
- Image Processing
- Graphical User Interfaces

Graphs and Charts:

An early application for computer graphics is the display of simple data graphs, usually plotted on a character printer.

Data Plotting is still one of the most common graphics applications, but today we can easily generate graphs showing highly complex data relationships for printed reports or for presentations using 35mm slides, transparencies, or animated videos.

Graphs and charts are commonly used to summarize financial, statistical, mathematical, scientific, engineering, and economic data for research reports, managerial summaries, consumer information bulletins, and other types of publications.

A variety of commercial graphing packages are available, and workstation devices and service bureaus exist for converting screen displays into film, slides or overhead transparencies for use in presentations or archiving.

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

FIGURE 1-2 Two-dimensional line graphs, bar charts, and a pie chart. (*Courtesy of UNIRAS, Inc.*)

FIGURE 1-3 Two color-coded data sets displayed as a three-dimensional bar chart on the surface of a geographical region. (*Reprinted with permission from ISSCO Graphics, San Diego, California.*)

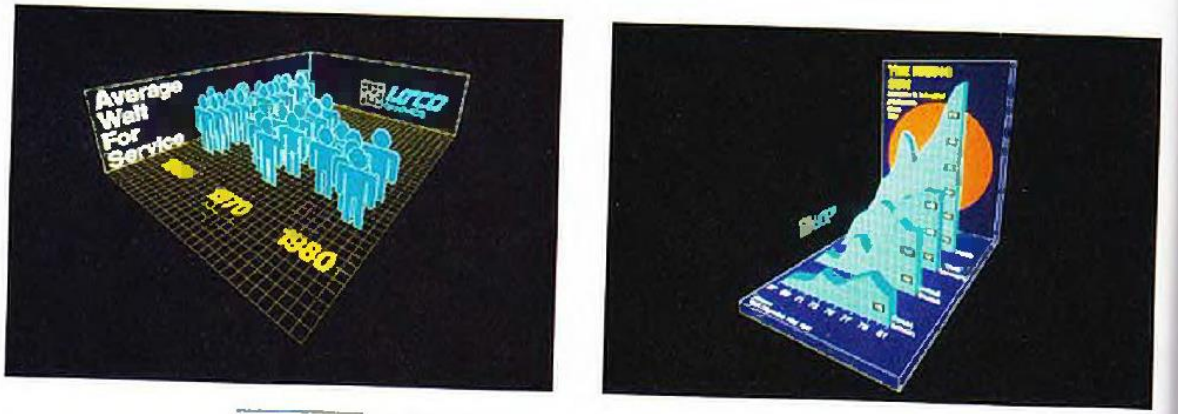


FIGURE 1-4 Two three-dimensional graphs designed for dramatic effect. (*Reprinted with permission from ISSCO Graphics, San Diego, California.*)

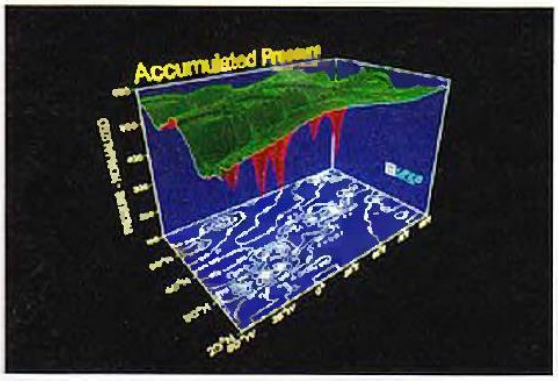


FIGURE 1–5 Plotting two-dimensional contours in a ground plane, with a height field plotted as a surface above the ground plane. (*Reprinted with permission from ISSCO Graphics, San Diego, California.*)

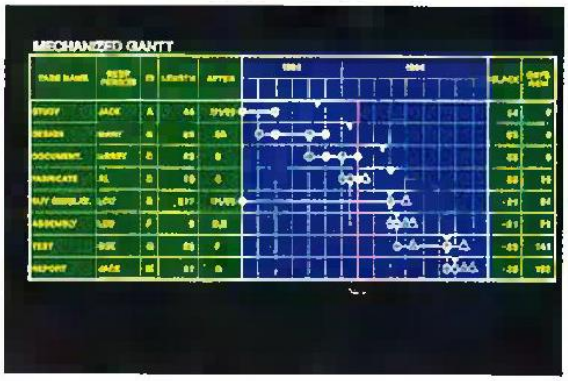


FIGURE 1–6 A time chart displaying scheduling and other relevant information about project tasks. (*Reprinted with permission from ISSCO Graphics, San Diego, California.*)

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**, or **CADD, computer-aided drafting and 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; objects 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 1-7 and 1-8 give examples of wireframe displays in design applications.

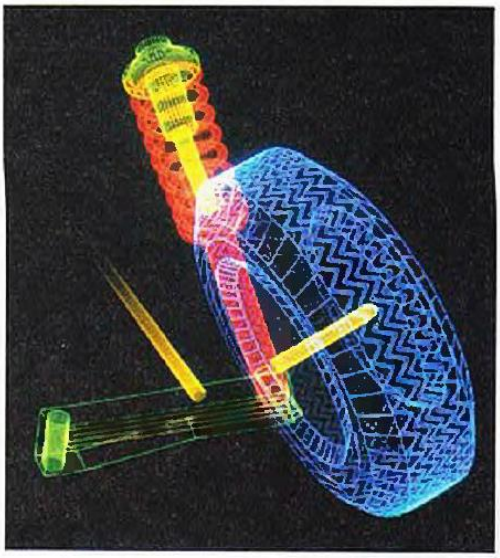
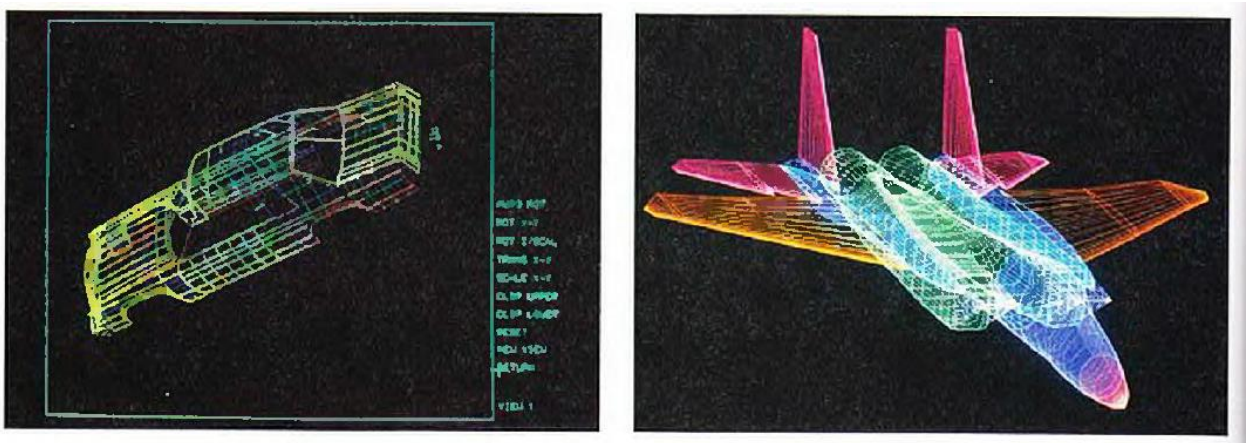


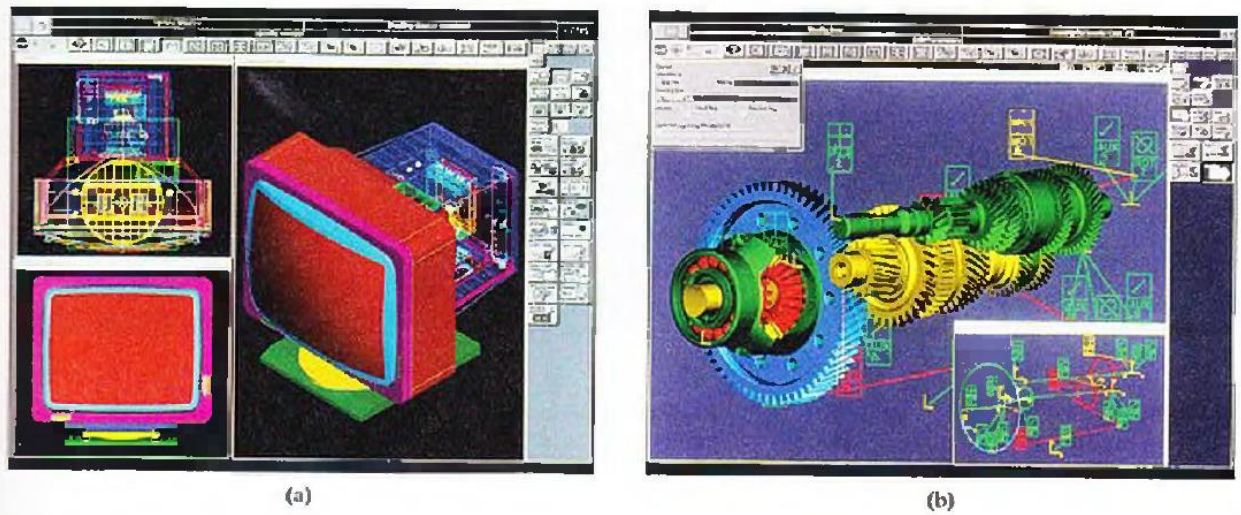
FIGURE 1–7 Color-coded, wire-frame display for an automobile wheel assembly. (*Courtesy of Evans & Sutherland.*)



(a) (b)

FIGURE 1–8 Color-coded, wire-frame outlines of body designs for an automobile and an aircraft. (*Courtesy of* (a) *Peritek Corporation and* (b) *Evans & Sutherland.*)

Software packages for CAD applications typically provide the designer with a multi-window environment, as in Figs. 1-9 and 1-10. The various displayed windows can show enlarged sections or different views of objects.



(a) (b)

FIGURE 1–9 Multiple-window, color-coded CAD workstation displays. (*Courtesy of Intergraph Corporation.*)

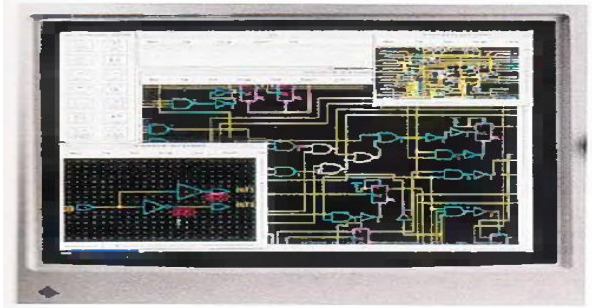


FIGURE 1–10 A circuit design application, using multiple windows and color-coded logic components. (*Courtesy of Sun Microsystems.*)

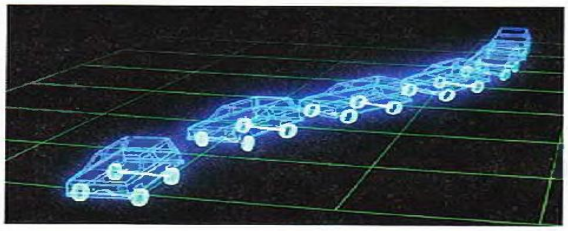


FIGURE 1–11 Simulation of vehicle performance during lane changes. (*Courtesy of Evans & Sutherland and Mechanical Dynamics, Inc.*)

Circuits such as the one shown in Fig. 1-10 and networks for communications, water supply, or other utilities are constructed with repeated placement of a few graphical shapes. The shapes used in a design represent the different network or circuit components. Standard shapes for electrical, electronic, and logic circuits are often supplied by the design package. For other applications, a designer can create personalized symbols that are to be used to construct the network or circuit. The system is then designed by successively placing components into the layout, with the graphics package automatically providing the connections between components. This allows the designer to quickly try out alternate circuit schematics for minimizing the number of components or the space required for the system.

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, as demonstrated in Figure 1-12. Because a wire-frame image is not displayed with rendered surfaces, the calculations for each segment of the animation can be performed quickly to produce a smooth motion on the screen. Also, a wire-frame displays allow the designer to see into the interior of the vehicle and to watch the behavior of inner components during motion.

When object designs are complete, or nearly complete, realistic lighting conditions and surface rendering are applied to produce displays that will show the appearance of the final product.

FIGURE 1-12 Realistic renderings of engineering designs. (*Courtesy of* (a) *Intergraph Corporation and* (b) *Evans & Sutherland.*)

The manufacturing process is also tied in to the computer description of designed objects so that the fabrication of product can be automated, using methods that are referred to as **CAM, Computer Aided manufacturing.**

A circuit board layout, for example, can be transformed into a description of the individual processes needed to construct the layout. Some mechanical parts are manufactured by describing how the surfaces are to be formed with machine tools. Figure 1-14 shows the path to be taken by

machine tools over the surfaces of an object during its construction. Numerically controlled machine tools are then set up to manufacture the part according to these construction layouts.

FIGURE 1–13 Studio lighting effects and realistic surface-rendering techniques are applied by computer-graphics programs to produce advertising pieces for finished products. This computer-generated image of a Chrysler Laser automobile was produced from data supplied by the Chrysler Corporation. (*Courtesy of Eric Haines, Autodesk, Inc.*)

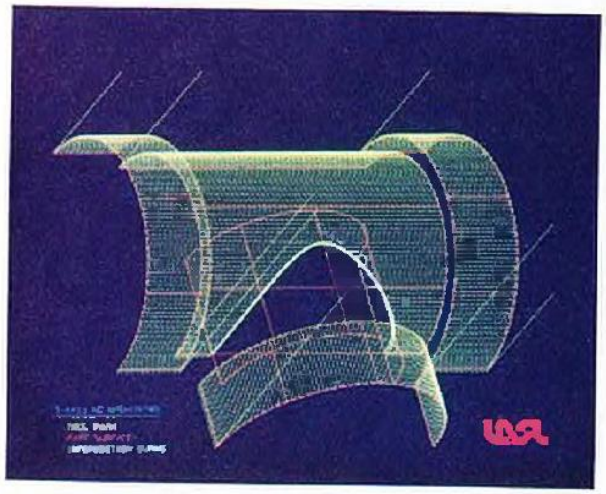


FIGURE 1–14 A CAD layout for describing the numerically controlled machining of a part. The part surface is displayed in one color and the tool path in another color. (*Courtesy of Los Alamos National Laboratory.*)

Architects use interactive graphics methods to lay out floor plans, such as Fig. 1-15, that show the positioning of rooms, doors, windows, stairs, shelves, counters, and other building features. Working from the display of a building layout on a video monitor, an electrical designer can try out arrangements for wiring, electrical outlets, and fire warning systems. Also, facility-layout packages can be applied to the layout to determine space utilization in an office or on a manufacturing floor.

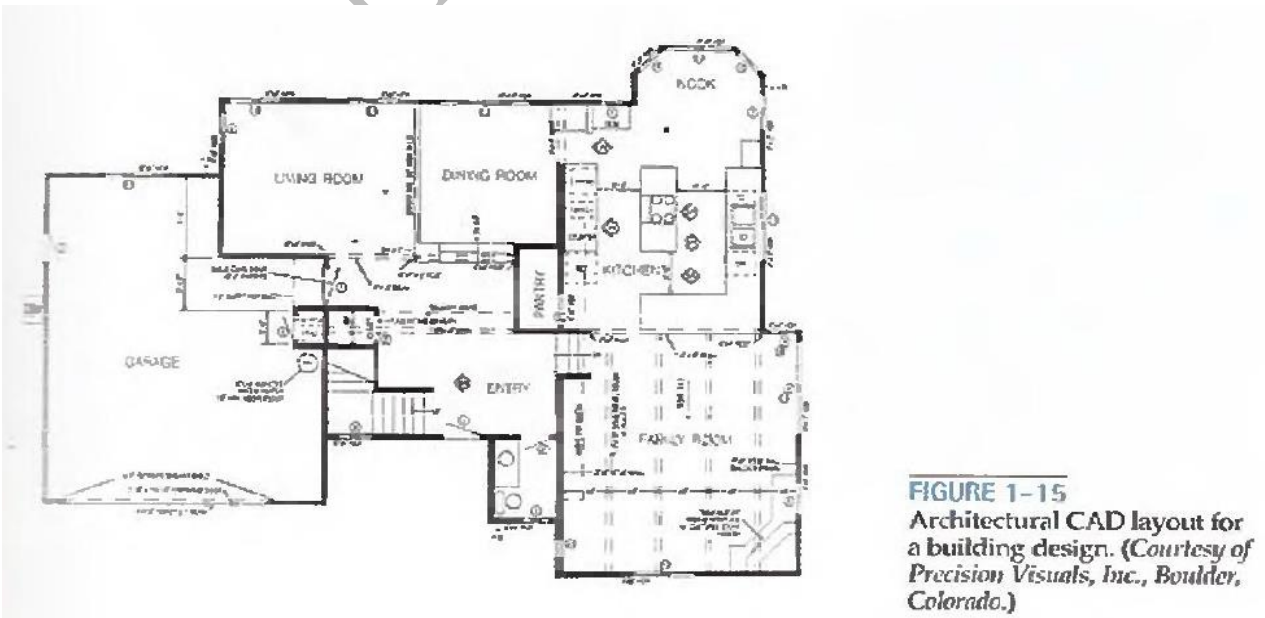


FIGURE 1–15
Architectural CAD layout for a building design. (*Courtesy of Precision Visuals, Inc., Boulder, Colorado.*)

Realistic displays of architectural designs, as in Fig. 1-16, permit both architects and their clients to study the appearance of a single building or a group of buildings, such as a campus or industrial complex. With virtual-reality systems, designers can even go for a simulated "walk" through the rooms or around the outsides of buildings to better appreciate the overall effect of a particular design. In addition to realistic exterior building displays, architectural CAD packages also provide facilities for experimenting with three-dimensional interior layouts and lighting (Fig. 1-17).

FIGURE 1-16 Realistic, three-dimensional renderings of building designs. (a) A street-level perspective for the World Trade Center project. (*Courtesy of Skidmore, Owings, & Merrill.*) (b) Architectural visualization of an atrium, created for a computer animation by Marialine Prieur, Lyon, France. (*Courtesy of Thomson Digital Image, Inc.*)

Many other kinds of systems and products are designed using either general CAD packages or specially developed CAD software. Figure 1-18, for example, shows a rug pattern designed with a CAD system.

FIGURE 1-17 A hotel corridor that provides a sense of movement by positioning light fixtures along an undulating path and creates a sense of entry by placing a light tower at the entrance to each room. (*Courtesy of Skidmore, Owings, & Merrill.*)

FIGURE 1-18 Oriental rug pattern created with computer-graphics design methods. (*Courtesy of Lexidata Corporation.*)

Virtual Reality Environments:

Virtual Reality Environments allows a user to 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.

As the tractor operator in Fig. 1-19 manipulates the controls, the headset presents a stereoscopic view (Fig. 1-20) of the front-loader bucket or the backhoe, just as if the operator were in the tractor seat. This allows the designer to explore various positions of the bucket or backhoe that might obstruct the operator's view, which can then be taken into account in the overall tractor design. Figure 1-21 shows a composite, wide-angle view from the tractor seat, displayed on a standard video monitor instead of in a virtual three dimensional scene. And Fig. 1-22 shows a view of the tractor that can be displayed in a separate window or on another monitor.

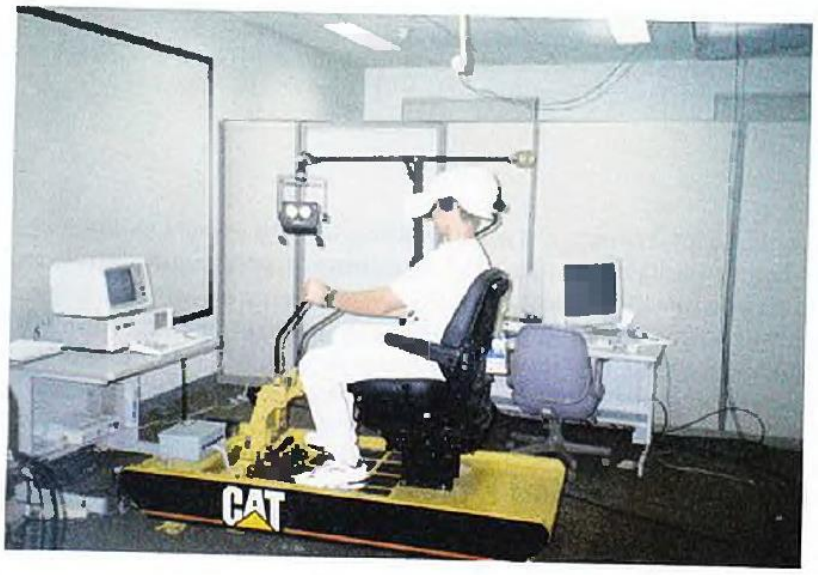


FIGURE 1-19 Operating a tractor in a virtual-reality environment. As the controls are moved, the operator views the front loader, backhoe, and surroundings through the headset. (*Courtesy of the National Center for Supercomputing Applications, University of Illinois at Urbana-Champaign, and Caterpillar, Inc.*)

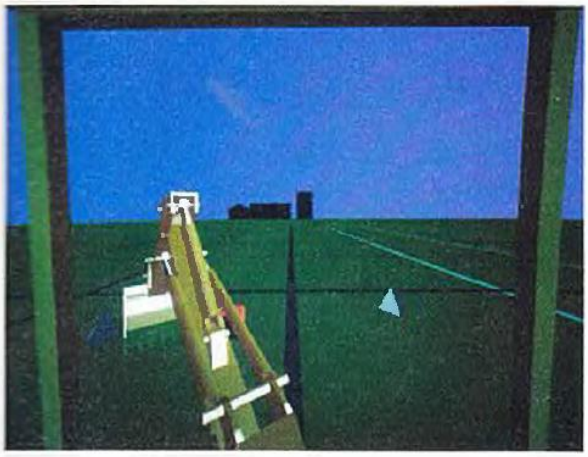


FIGURE 1-20 A headset view of the backhoe presented to a tractor operator in a virtual-reality environment. (*Courtesy of the National Center for Supercomputing Applications, University of Illinois at Urbana-Champaign, and Caterpillar, Inc.*)

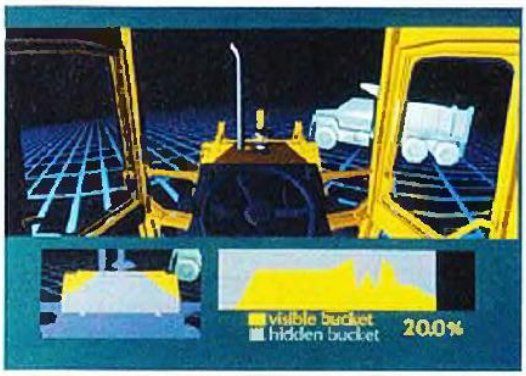


FIGURE 1-21 Operator's view of the tractor bucket, composited in several sections to form a wide-angle view on a standard monitor. (*Courtesy of the National Center for Supercomputing Applications, University of Illinois at Urbana-Champaign, and Caterpillar, Inc.*)

FIGURE 1-22 View of the tractor displayed on a standard monitor. (*Courtesy of the National Center for Supercomputing Applications, University of Illinois at Urbana-Champaign, and Caterpillar, Inc.*)

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

Data Visualizations:

Producing graphical representations for scientific, engineering , and medical sets and processes is another fairly new application of computer graphics, which is generally referred to as **scientific visualization.**

Business Visualization is used In conjunction with data sets related to commerce, industry and other nonscientific areas.

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- 23 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.

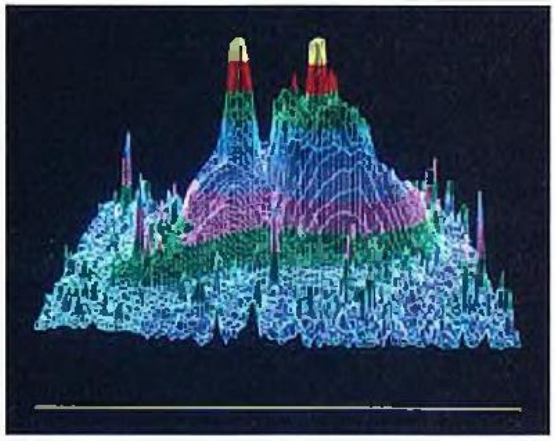


FIGURE 1–23 A color-coded plot with sixteen million density points of relative brightness observed for the Whirlpool Nebula reveals two distinct galaxies. (*Courtesy of Los Alamos National Laboratory.*)

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, higher-order tensors, or any combination of these data types. And data sets can be distributed over a two-dimensional of space or three-dimensional region, or a higher-dimensional space. Color coding is just one way to visualize a data set. Additional techniques include contour plots, graphs and charts, surface renderings, and visualizations of volume interiors.

Mathematicians, physical scientists, and others use visual techniques to analyze mathematical functions and processes or simply to produce interesting graphical representations. A color plot

of mathematical curve functions is shown in Fig. 1-24, and a surface plot of a function is shown in Fig. 1-25. Fractal procedures using quaternions generated the object shown in Fig. 1-27, and a topological structure is displayed in Fig. 1-27. Scientists are also developing methods for visualizing general classes of data. Figure 1-28 shows a general technique for graphing and modeling data distributed over a spherical surface.

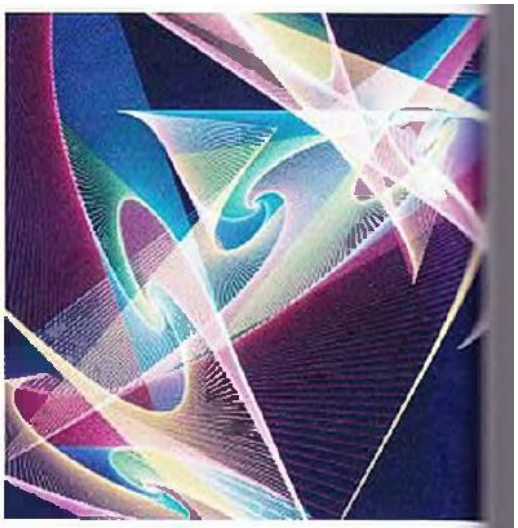


FIGURE 1-24 Mathematical curve functions plotted in various color combinations. (*Courtesy of Melvin L. Prueitt, Los Alamos National Laboratory.*)

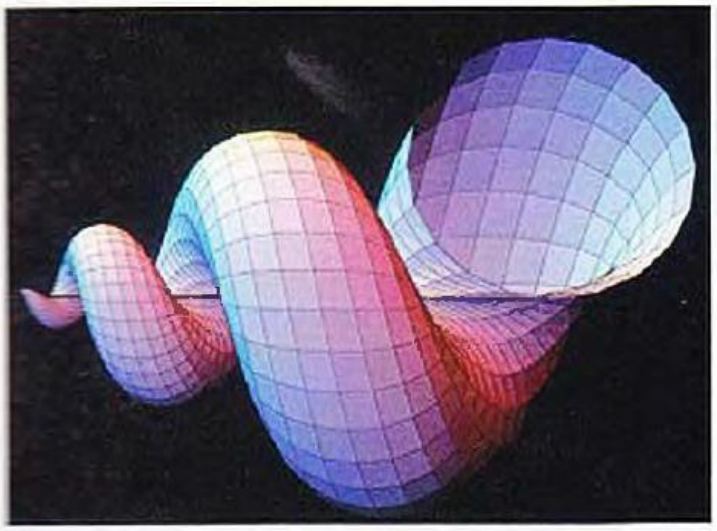


FIGURE 1-25 Lighting effects and surface-rendering techniques were applied to produce this surface representation for a three-dimensional function. (*Courtesy of Wolfram Research, Inc., The Maker of Mathematica.*)

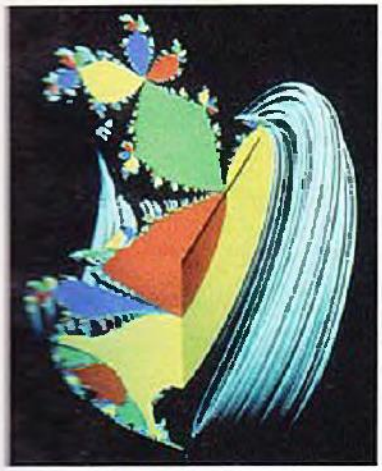


FIGURE 1-26 A four-dimensional object projected into three-dimensional space, then projected to the two-dimensional screen of a video monitor and color coded. The object was generated using quaternions and fractal squaring procedures, with an octant subtracted to show the complex Julia set. (*Courtesy of John C. Hart, Department of Computer Science, University of Illinois at Urbana-Champaign.*)

FIGURE 1-27 Four views from a real-time, interactive computer-animation study of minimal surfaces ("snails") in the 3-sphere projected to three-dimensional Euclidean space. (*Courtesy of George Francis, Department of Mathematics and the National Center for Supercomputing Applications, University of Illinois at Urbana-Champaign.* © 1993.)

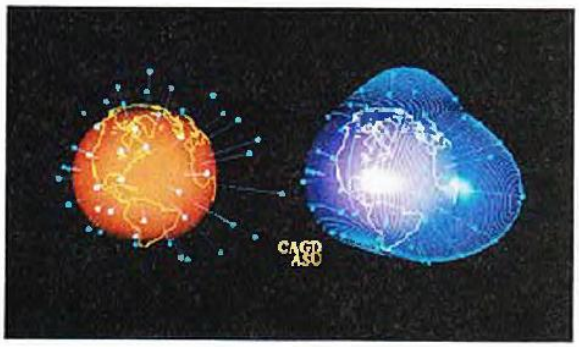


FIGURE 1-28 A method for graphing and modeling data distributed over a spherical surface. (*Courtesy of Greg Nielson, Computer Science Department, Arizona State University.*)

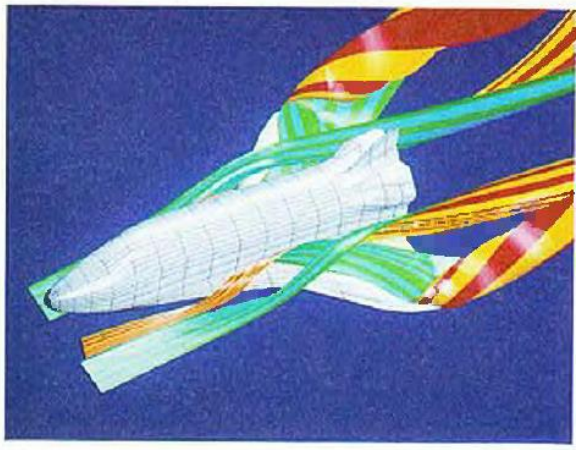


FIGURE 1-29 A visualization of stream surfaces flowing past a space shuttle, devised by Jeff Hultquist and Eric Raible, NASA Ames. (*Courtesy of Sam Uselton, NASA Ames Research Center.*)

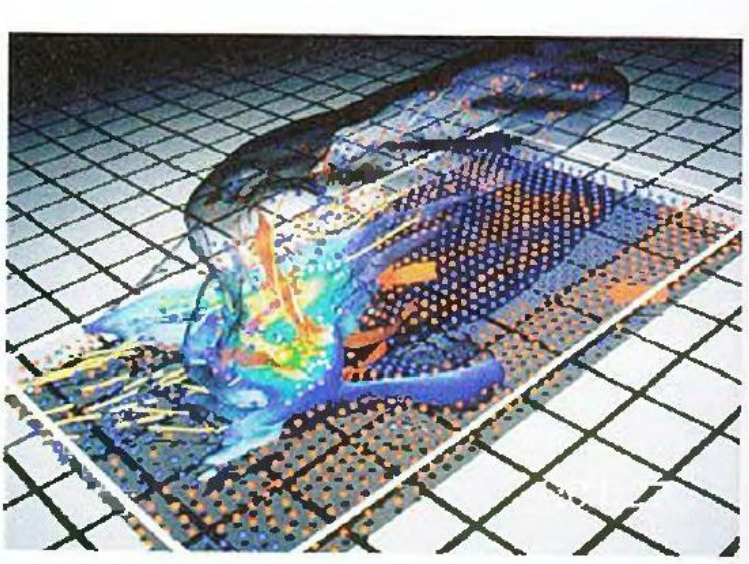


FIGURE 1-30 Numerical model of airflow inside a thunderstorm. (*Courtesy of Bob Wilhelmson, Department of Atmospheric Sciences and the National Center for Supercomputing Applications, University of Illinois at Urbana-Champaign.*)

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 in Fig. 1- 43, can help trainees to understand the operation of the system.

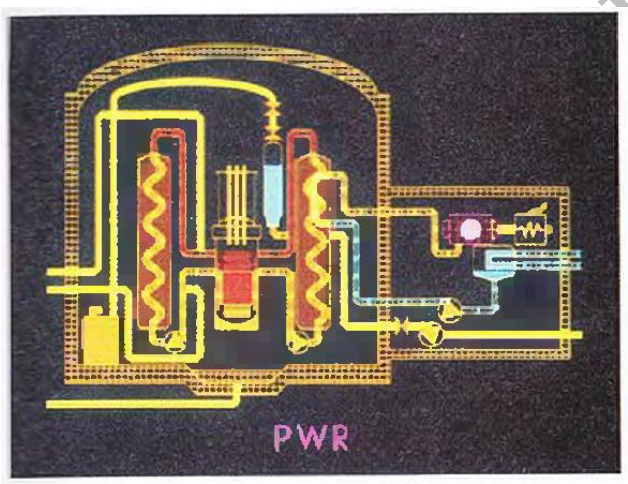


FIGURE 1-43 Color-coded diagram used to explain the operation of a nuclear reactor. (*Courtesy of Los Alamos National Laboratory.*)

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. Two examples of large simulators with internal viewing systems are shown in Figs. 1-44 and 1-45. Another type of viewing system is

shown in Fig. 1-46 . Here a viewing screen with multiple panels is mounted in front of the simulator and color projectors display the flight scene on the screen panels. Similar viewing systems are used in simulators for training aircraft control-tower personnel. Figure 1-45 gives an example of the instructors area in a flight simulator. The keyboard is used to input parameters affecting the airplane performance or the environment, and the pen plotter is used to chart the path of the aircraft during a training session.

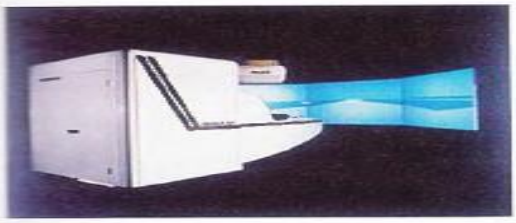
Scenes generated for various simulators are shown in Figs. 1-48 through 1- 50. An output from an automobile-driving simulator is given in Fig. 1-50. This simulator is used to investigate the behavior of drivers in critical situations. The drivers' reactions are then used as a basis for optimizing vehicle design to maximize traffic safety.

FIGURE 1-44 A large, enclosed flight simulator with a full-color visual system and six degrees of freedom in its motion. (*Courtesy of Frasca International.*)

FIGURE 1-45 A military tank simulator with a visual imagery system. (*Courtesy of Meditech and GE Aerospace.*)

(a)

(b)



(c)

FIGURE 1-46 The cabin interior (a) of a dual-control flight simulator, and an external full-color viewing system (b) and (c) for a small flight simulator. (*Courtesy of Frasca International.*)

FIGURE 1-47 An instructor's area behind the cabin of a small flight simulator. The equipment allows the instructor to monitor flight conditions and to set airplane and environment parameters. (*Courtesy of Frasca International.*)

FIGURE 1-48 Flight-simulator imagery. (*Courtesy of Evans & Sutherland.*)

FIGURE 1-49 Imagery generated for a naval simulator. (*Courtesy of Evans & Sutherland.*)

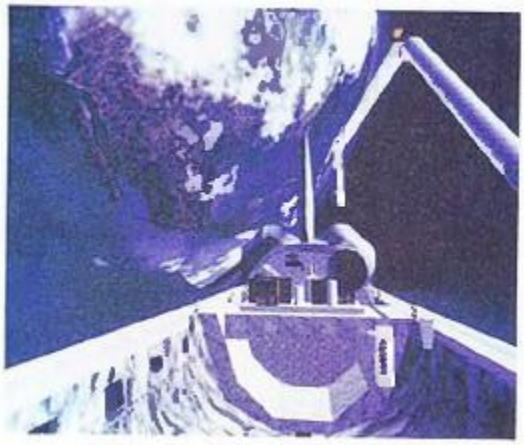


FIGURE 1-50 Space-shuttle imagery. (*Courtesy of Mediatech and GE Aerospace.*)

FIGURE 1-51 The interior of an automobile-simulator (a) and a street-scene view (b) that can be presented to a driver. (*Courtesy of Evans & Sutherland.*)

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 hardware, artist's paintbrush (such as Lumens), other paint packages (such as Pixelpaint and Superpaint), specially developed software, symbolic mathematics packages (such as Mathematics), CAD packages, desktop publishing software, and animation packages that provide facilities for designing object shapes and specifying object motions.

Figure 1-52 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 paintbrush program was used to the characters in Fig. 1-53, who seem to be busy on a creation of their own.

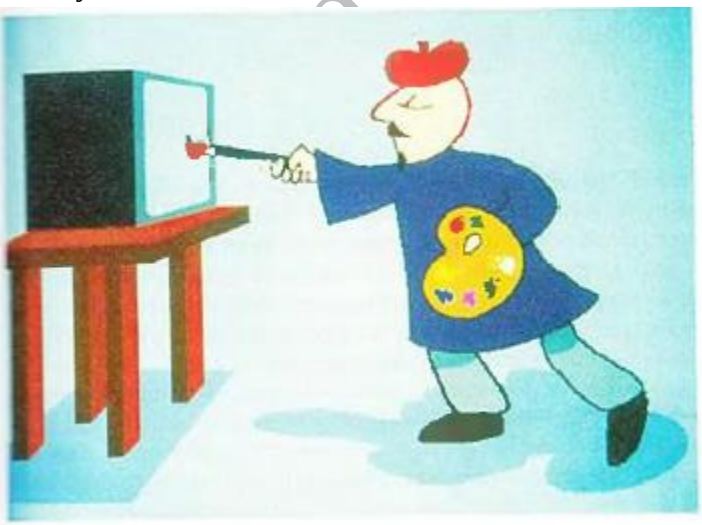


FIGURE 1-52 Cartoon drawing produced with a paintbrush program, symbolically illustrating an artist at work on a video monitor. (*Courtesy of Gould Inc., Imaging & Graphics Division, and Aurora Imaging.*)

FIGURE 1-53 Cartoon demonstrations of an "artist" creating a picture with a paintbrush system. In (a), the picture is drawn on a graphics tablet as elves watch the development of the image on the video screen. In (b), the artist and elves are superimposed on the famous Thomas Nast drawing of Saint Nicholas, which was input to the system with a video camera, then scaled and positioned. (*Courtesy of Gould Inc., Imaging & Graphics Division, and Aurora Imaging.*)

A paintbrush system, with a Wacom cordless, pressure-sensitive stylus, was used to produce the electronic painting in Fig. 1-54 that simulates the brush strokes of Van Gogh. The stylus translates changing hand pressure into variable line widths, brush sizes, and color gradations. Figure 1-55 shows 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, wet-ness, and footprint. Figure 1-56 gives an example of paintbrush methods combined with scanned images.

FIGURE 1-54 A Van Gogh look-alike created by graphics artist Elizabeth O'Rourke with a cordless, pressure-sensitive stylus. (*Courtesy of Wacom Technology Corporation.*)

Fine artists use a variety of other computer technologies to produce images such as a combination of three-dimensional modeling packages, texture mapping, drawing programs, and CAD software.

FIGURE 1-55 An electronic watercolor, painted by John Derry of Time Arts, Inc. using a cordless, pressure-sensitive stylus and Lumena gouache-brush software. (*Courtesy of Wacom Technology Corporation.*)

FIGURE 1-56 The artist of this picture, entitled *Electronic Avalanche*, makes a statement about our entanglement with technology, using a personal computer with a graphics tablet and Lumena software to combine renderings of leaves, flower petals, and electronics components with scanned images. (*Courtesy of the Williams Gallery.* © *1991 Joan Truckenbrod, The School of the Art Institute of Chicago.*)

In "mathematical" art, an artist uses a combination of mathematical functions, fractal procedures, Mathematics software, ink-jet printers, and other systems to create a variety of three-dimensional and two-dimensional shapes and stereoscopic image pairs.

These methods are also applied in commercial art for logos and other designs, page layouts combining text and graphics, TV advertising spots, and other areas. For many applications of commercial art (and in motion pictures and other applications), photorealistic techniques are used to render images of a product for product advertising.

Animations are also used frequently in advertising, and television commercials are produced frame by frame, where each frame of the motion is rendered and saved as an image file. In each successive frame, the motion is simulated by moving object positions are slightly displaced from their positions in the previous frame. When all frames in the animation sequence have been rendered, the frames are transferred to film or stored in a video buffer for playback. Film animations require 24 frames for each second in the animation sequence. If the animation is to be played back on a video monitor, 30 frames per second are required.

A common graphics method employed in many commercials is morphing, where one object is transformed (metamorphosed) into another.

Entertainment:

Television productions, motion pictures and music videos routinely used computer graphic methods. Sometimes the graphics scenes are displayed by themselves, and sometimes graphics objects are combined with the live actors and scenes, and sometimes the films are completely generated using computer rendering and animation techniques.

Many TV series, motion pictures and music videos regularly employ computer graphics methods to produce special effects.

Computer-graphics methods can also be employed to simulate a human actor. Using digital files of an actor's facial features, an animation program can generate film sequences that contain a computer generated replica of that person. In the event of an illness or accident during the filming of a motion picture, these simulation methods can be sed to replace the actor in subsequent film scenes.

Image Processing:

The modification or interpretation of existing pictures, such as photographs and TV scans, is called 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 techniques, on the other hand are used to improve the picture quality, analyze images, or recognize visual patterns for robotics applications.

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 to enhance the quality of a picture is shown in Fig. 1-70. 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.

FIGURE 1–70 A blurred photograph of a license plate becomes legible after the application of image-processing techniques. (*Courtesy of Los Alamos National Laboratory.*)

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.

Computed X-ray tomography (CT) , position emission tomography (PET) and Computed axial Tomography (CAT) use projection methods to reconstruct cross sections from digital data. These techniques are also used to monitor internal functions and show cross sections during surgery. Other medical imaging techniques include ultrasonics and nuclear medicine scanners. With ultrasonics, high-frequency sound waves, instead of X-rays, are used to generate digital data. Nuclear medicine scanners collect digital data from radiation emitted from ingested radionuclides and plot color coded images.

lmage processing and computer graphics are typically combined in many applications. Medicine, for example, uses these techniques to model and study physical functions, to design artificial limbs, and to plan and practice surgery. The last application is generally referred to as computer-aided surgery. Two-dimensional cross sections of the body are obtained using imaging techniques. Then the slices are viewed and manipulated using graphics methods to simulate actual surgical procedures and to try out different surgical cuts.

Graphical User Interfaces:

It is common now for software packages to provide a **graphical user interface**. A major component of a graphical interface is a window manager that allows a user to display multiple, rectangular window areas called display windows. Each window can contain a different process that can contain graphical or non-graphical information. To make a particular window active, we simply click in that window using an interactive pointing device, such as a mouse.

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 suggest the 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.

FIGURE 1-73 A graphical user interface, showing multiple display windows, menus, and icons. (*Courtesy of Image-En Corporation.*)

Video Display Devices:

Typically, the primary output device in a graphics system is a video monitor. 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.

Refresh Cathode-Ray Tubes

Fig 2-2 illustrates the basic operation of a CRT. A beam of electrons (cathode rays), 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. Because the light emitted by the phosphor fades very rapidly, some method is needed for maintaining the screen picture. One way to keep the phosphor glowing is to redraw the picture repeatedly by quickly directing the electron beam back over the same points. This type of display is called a **refresh CRT,** and the frequency at which the picture is redrawn on the screen is referred to as the **refresh rate.**

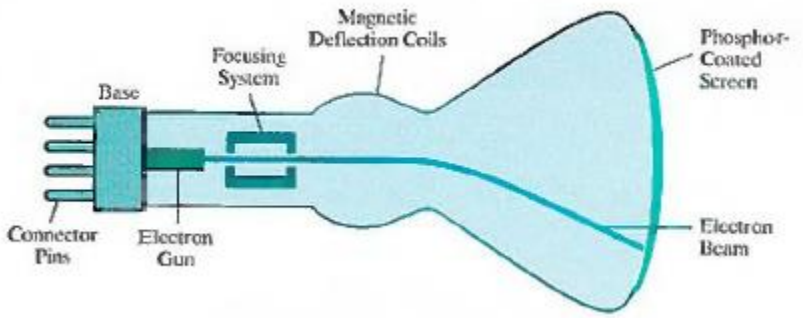


FIGURE 2–2 Basic design of a magnetic-deflection CRT.

The primary components of an electron gun in a CRT are the heated metal cathode and a control grid Fig. 2-3. 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. In the vacuum 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 a positively charged metal coating on the side of the CRT envelope near the phosphor screen, or an accelerating anode can be used, as in Fig. 2-3. Sometimes the electron gun is built to contain the accelerating anode and focusing system within the same unit.

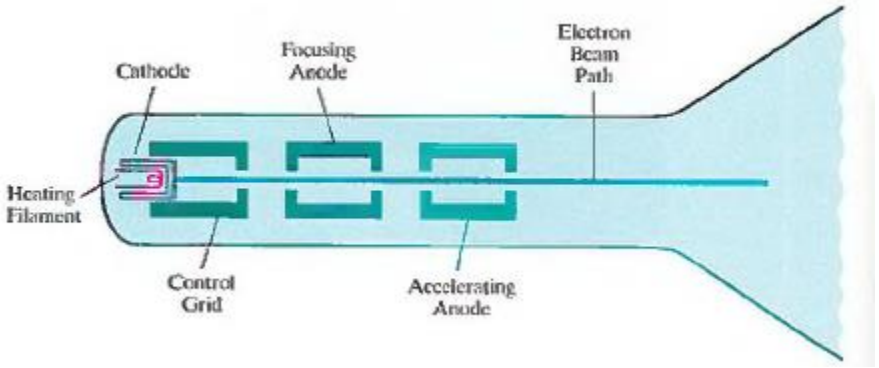


FIGURE 2–3 Operation of an electron gun with an accelerating anode.

Intensity of the electron beam is controlled by setting voltage levels on the control grid, which is a metal cylinder that fits over the cathode. A high negative voltage applied to the control grid will shut off the beam by repelling electrons and stopping them from passing through the small hole at the end of the control grid structure. A smaller negative voltage on the control grid simply decreases the number of electrons passing through. Since the amount of light emitted by the phosphor coating depends on the number of electrons striking the screen, we control the brightness of a display by varying the voltage on the control grid.

The focusing system in a CRT is needed to force the electron beam to converge into a small spot as it strikes the phosphor. Otherwise, the electrons would repel each other, and the beam would spread out as it approaches the screen. Focusing is accomplished with either electric or magnetic fields. Electrostatic focusing is commonly used in television and computer graphics monitors. With electrostatic focusing, the electron beam passes through a positively charged metal cylinder that forms an electrostatic lens, as shown in Fig. 2-3. The action of the electrostatic lens focuses the electron beam at the center of the screen, in exactly the same way that an optical lens focuses a beam of light at a particular focal distance. Similar lens focusing effects can be accomplished with a magnetic field set up by a coil mounted around the outside of the CRT envelope. Magnetic lens focusing produces the smallest spot size on the screen.

Additional focusing hardware is used in high-precision systems to keep the beam in focus at all screen positions. The distance that the electron beam must travel to different points on the screen varies because the radius of curvature for most CRTs is greater than the distance from the focusing system to the screen center. Therefore, the electron beam will be focused properly only at the center of the screen. As the beam moves to the outer edges of the screen, displayed images become blurred. To compensate for this, the system can adjust the focusing according to the screen position of the beam.

As with focusing, deflection of the electron beam can be controlled either with electric fields or with magnetic fields. Cathode-ray tubes are now commonly constructed with magnetic deflection coils mounted on the outside of the CRT envelope, as illustrated in Fig. 2-2. Two pairs of coils are used, with the coils in each pair mounted on opposite sides of the neck of the CRT envelope. One pair is mounted on the top and bottom of the neck, and the other pair is mounted on opposite sides of the neck. The magnetic, field produced by each pair of coils results in a transverse deflection force that is perpendicular both to the direction of the magnetic field and to the direction of travel of the electron beam. Horizontal deflection is accomplished with one pair of coils, and vertical deflection by the other pair. The proper deflection amounts are attained by adjusting the current through the coils.

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 the vertical deflection, and the other pair is mounted vertically to control horizontal deflection (Fig. 2-4).

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 the kinetic energy is absorbed by the phosphor. Part of the beam energy is converted by

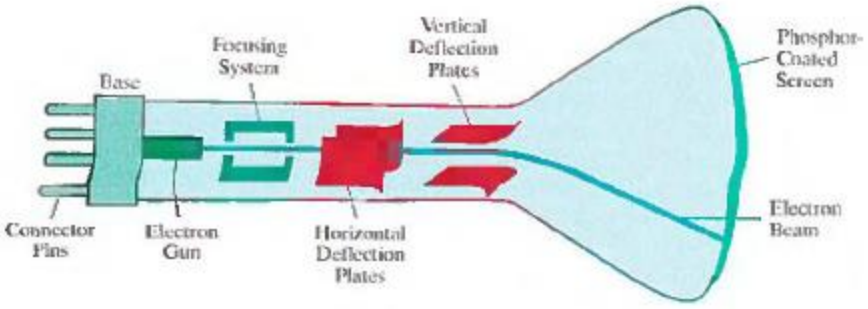


FIGURE 2–4 Electrostatic deflection of the electron beam in a CRT.

friction into 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 quantums of Light energy. What we see on the screen is the combined effect of all the electron light emissions: a glowing spot that quickly fades after all the excited phosphor electrons have returned to their ground energy level. 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.

Different kinds of phosphors are available for use in a CRT. Besides color, a major difference between phosphors is their **persistence**: how long they continue to emit light (that is, have excited electrons returning to the ground state) after the CRT beam is removed. *Persistence is defined as the time 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 on the screen without flicker. A phosphor with low persistence is useful for animation; a high-persistence phosphor is useful for displaying highly complex, static pictures. Although some phosphors have a persistence greater than 1 second, graphics monitors are usually constructed with a persistence in the range from 10 to 60 microseconds.

Figure 2-5 shows the intensity distribution of a spot on the screen. The intensity is greatest at the center of the spot, and decreases with a Gaussian distribution out to the edges of the spot. This distribution corresponds to the cross-sectional electron density distribution of the CRT beam.

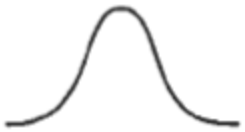


Figure 2-5
Intensity distribution of an illuminated phosphor spot on a CRT screen.

The maximum number of points that can be displayed without overlap on a CRT is referred to as the resolution. A more precise definition of resolution is the number of points per centimeter that can be plotted horizontally and vertically, although it is often simply stated as the total number of points in each

direction. Spot intensity has a Gaussian distribution as shown in Fig. 2-5.

Figure 2-6
Two illuminated phosphor spots are distinguishable when their separation is greater than the diameter at which a spot intensity has fallen to 60 percent of maximum.

Spot size also depends on intensity. As more electrons are accelerated toward the phosphor per second, the CRT beam diameter and the illuminated spot increase. In addition, the increased excitation energy tends to spread to neighboring phosphor atoms not directly in the path of the beam, which further increases the spot diameter. Thus, resolution of a CRT is dependent on the type of phosphor, the intensity to be displayed, and the focusing and deflection systems.

Typical resolution on high-quality systems is 1280 by 1024, with higher resolutions available on many systems. High resolution systems are often referred to as high-definition systems. The physical size of a graphics monitor is given as the length of the screen diagonal, with sizes varying from about 12 inches to 27 inches or more. A CRT monitor can be attached to a variety of computer systems, so the number of screen points that can actually be plotted depends on the capabilities of the system to which it is attached.

Raster-Scan Displays:

In a raster-scan system, the electron beam is swept across the screen, one row at a time from top to bottom. Each row is referred to as **scan line**. As the electron beam moves across each row, the beam intensity is turned on and off to create a pattern of illuminated spots. Picture definition is stored in a memory area called the **refresh buffer** or **frame buffer**. This memory area holds the set of color values for the screen points. These stored color values are then retrieved from the refresh buffer and used to control the intensity of the electron beam as it moves from spot to spot across the screen,.In this way the picture is "painted" on the screen one row (scan line) at a time (Fig. 2-7). Each screen point is referred to as a **pixel** or **pel** (shortened forms of **picture element**). The capability of a raster-scan system to store intensity information for each screen point makes it well suited for the realistic display of scenes containing subtle shading and color patterns. Home television sets and printers are examples of other systems using raster-scan methods.

Raster systems are commonly characterized by their resolution, which is the number of pixel positions that can be plotted.

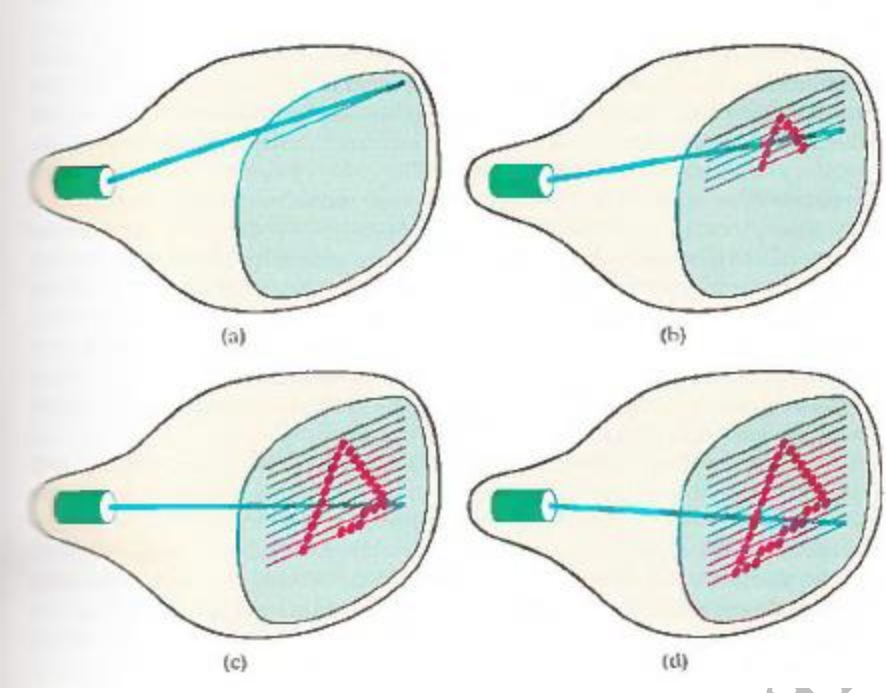


FIGURE 2-7 A raster-scan system displays an object as a set of discrete points across each scan line.

Another property of video monitors is **aspect ratio**., which is defined as the number of pixel columns divided by the number of scan lines(or vice versa) 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. An aspect ratio of 3/4 means that a vertical line plotted with three points has the same length as a horizontal line plotted with four 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 the frame buffer.

In a simple black-and-white system, each screen point is either on or off, so only one bit per pixel is needed to control the intensity of screen positions. A bit value of 1 indicates that the electron beam is to be turned on at that position, and a value of 0 indicates that the beam intensity is to be off. Additional bits are needed when color and intensity variations can be displayed. Up to 24 bits per pixel are included in high-quality systems, which can require several megabytes of storage for the frame buffer, depending on the resolution of the system. For example, a system with 24 bits per pixel and a screen resolution of 1024 by 1024 requires 3 megabytes of storage for the refresh buffer.

The number of bits per pixel in a frame buffer is sometimes referred to as either the **depth** of the buffer area or the number of **bit planes**. Also, a frame buffer with one bit per pixel is commonly called a bitmap and a frame buffer with multiple bits per pixel is a **pixmap.**

As each screen refresh takes place, we tend to see each frame as a smooth continuation of the patterns in the previous frame, as long as the refresh rate is not too low.

Refreshing on raster-scan displays is carried out at the rate of 60 to 80 frames per second, although some systems are designed for higher refresh rates. Sometimes, refresh rates are described in units of cycles per second, or Hertz (Hz), where a cycle corresponds to one frame. Using these units, we would describe a refresh rate of 60 frames per second as simply 60 Hz. At the end of each scan line, the electron beam returns to the left side of the screen to begin displaying the next scan line. The return to the left of the screen, after refreshing each scan line, is called the **horizontal retrace** of the electron beam. And at the end of each frame (displayed in 1/80th to 1/60th of a second), the electron beam returns (**vertical retrace**) to the top left comer of the screen to begin the next frame.

On some raster-scan systems (and in TV sets), each frame is displayed in two passes using an interlaced refresh procedure. In the first pass, the beam sweeps across every other scan line from top to bottom. Then after the vertical retrace, the beam sweeps out the remaining scan lines (Fig. 2-8). Interlacing of the scan lines in this way allows us 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. Interlacing is primarily used with slower refreshing rates. On an older, 30 frame per- second, non-interlaced display, for instance, some flicker is noticeable. But with interlacing, each of the two passes can be accomplished in 1/60th of a second, which brings the refresh rate nearer to 60 frames per second. This is an effective technique for avoiding flicker, providing that adjacent scan lines contain similar display information.

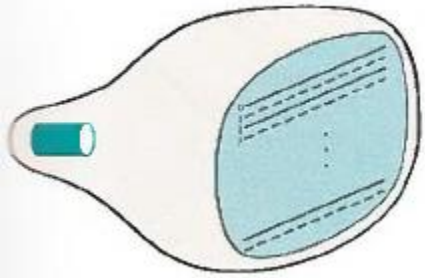


FIGURE 2-8 Interlacing scan lines on a raster-scan display. First, all points on the even-numbered (solid) scan lines are displayed; then all points along the odd-numbered (dashed) lines are displayed.

Random-Scan Displays:

When operated as a random-scan display unit, a CRT has the electron beam directed only to the parts of the screen where a picture is to be drawn. Random scan monitors draw a picture one line at a time and for this reason are also referred to as **vector displays (or stroke-writing or calligraphic displays**). The component lines of a picture can be drawn and refreshed by a random-scan system in any specified order (Fig. 2-9). A pen plotter operates in a similar way and is an example of a random-scan, hard-copy device.

Refresh rate on a random-scan system depends on the number of lines to be displayed. Picture definition is now stored as a set of line drawing commands in an area of memory r e f e d to as the **refresh display file**. Sometimes the refresh display file is called the **display list, display program,** or simply the refresh buffer. To display a specified picture, the system cycles

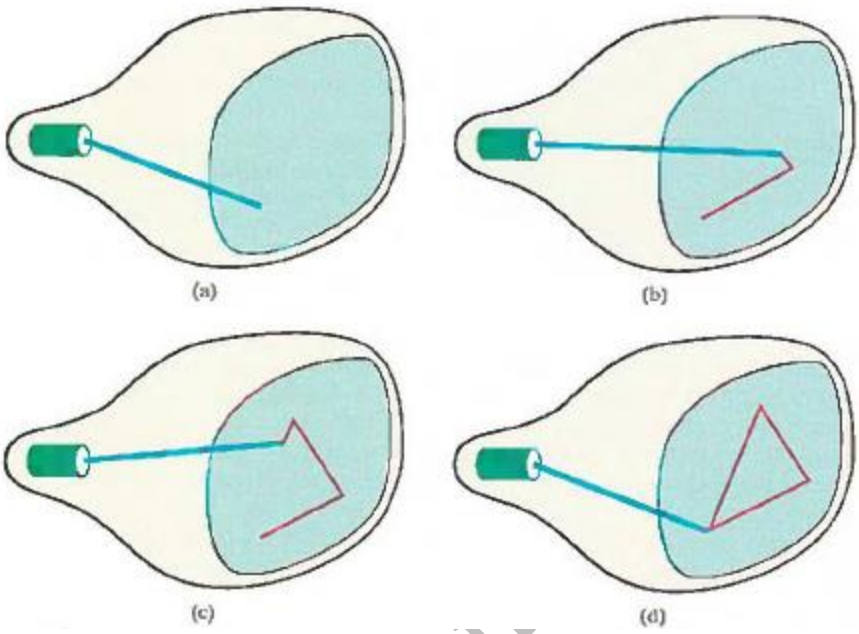


FIGURE 2-9 A random-scan system draws the component lines of an object in any specified order.

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. High quality vector systems are capable of handling approximately 100,000 "short" lines at this refresh rate. When a small set of lines is to be displayed, each rrfresh cycle is delayed to avoid refresh rates greater than 60 frames per second. Otherwise, faster refreshing of the set of lines could burn out the phosphor.

Random-scan systems are designed for line drawing applications and cannot display realistic shaded scenes. Since picture definition is stored as a set of line drawing instructions and not as a set of intensity values for all screen points, vector displays generally have higher resolution than raster systems. Also, vector displays produce smooth line drawings because the CRT beam directly follows the line path. A raster system, in contrast, produces jagged lines that are plotted as discrete point sets. However the greater flexibility and improved line-drawing capabilities of raster systems have resulted in the abandonment of vector technology.

Color CRT Monitors:

A CRT monitor displays color pictures by using a combination of phosphors that emit different-colored light. By combining the emitted light from the different phosphors, a range of colors can be generated. The two basic techniques for producing color displays with a CRT are the **beam-penetration method** and the **shadow-mask method.**

The **beam-penetration** method for displaying color pictures has been used with random-scan monitors. Two layers of phosphor, usually red and green, are coated onto the inside of the CRT screen, and the displayed color depends on how far the electron beam penetrates into the phosphor layers. A beam of slow electrons excites only the outer red layer. A beam of very fast electrons penetrates through the red layer and excites the inner green layer. At intermediate beam

speeds, combinations of red and green light are emitted to show two additional colors, orange and yellow. The speed of the electrons, and hence the screen color at any point, is controlled by the beam-acceleration voltage. Beam penetration has been an inexpensive way to produce color in random-scan monitors, but only four colors are possible, and the quality of pictures is not as good as with other methods.

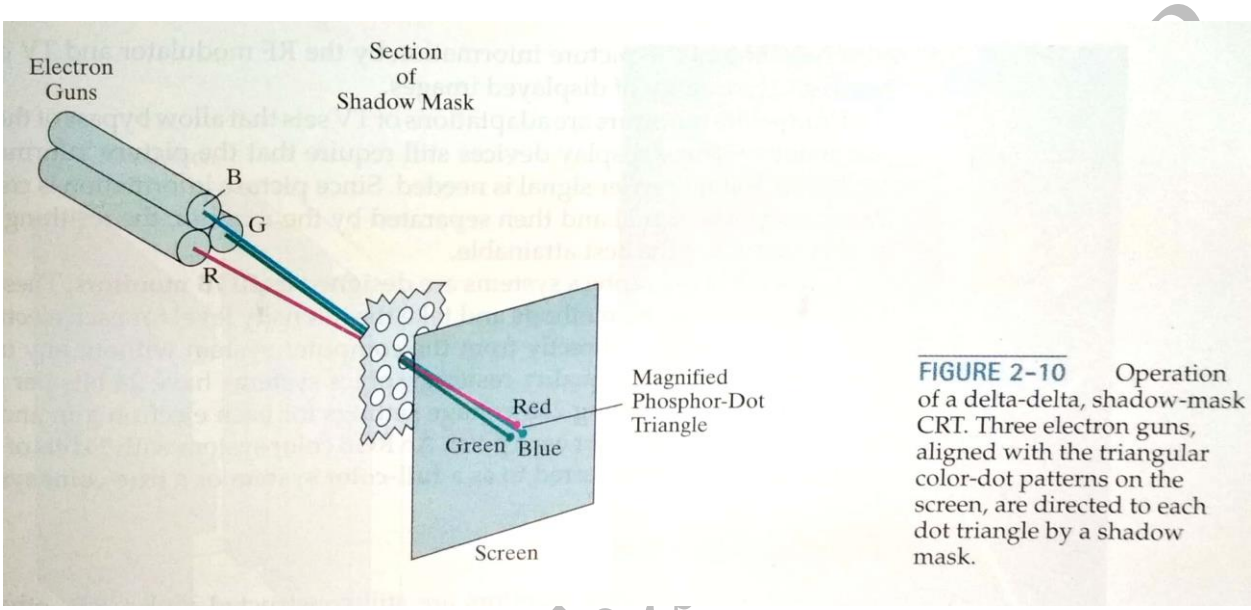


FIGURE 2-10 Operation of a delta-delta, shadow-mask CRT. Three electron guns, aligned with the triangular color-dot patterns on the screen, are directed to each dot triangle by a shadow mask.

Shadow-mask methods are commonly used in raster scan systems (including color TV) because they produce a much wider range of colors than the beam penetration method. A shadow-mask CRT has three phosphor color dots at each pixel position. One phosphor dot emits a red light, another emits a green light, and the third emits a blue light. This type of CRT has three electron guns, one for each color dot, and a shadow-mask grid just behind the phosphor-coated screen. Figure 2-10 illustrates the *delta-delta* shadow-mask method, commonly used in color CRT systems. The three electron beams are deflected and focused as a group onto the shadow mask, which contains a series of holes aligned with the phosphor-dot patterns. When the three beams pass through a hole in the shadow mask, they activate a dot triangle, which appears as a small color spot on the screen. The phosphor dots in the triangles are arranged so that each electron beam can activate only its corresponding color dot when it passes through the shadow mask. Another configuration for the three electron guns is an *in-line* arrangement in which the three electron guns, and the corresponding red-green-blue color dots on the screen, are aligned along one scan line instead of in a triangular pattern. This in-line arrangement of electron guns is easier to keep in alignment and is commonly used in high-resolution color CRTs.

We obtain color variations in a shadow-mask CRT by varying the intensity levels of the three electron beams. By turning off the red and green guns, we get only the color coming h m the blue phosphor. Other combinations of beam intensities produce a small light spot for each pixel position, since our eyes tend to merge the three colors into one composite. The color we see

depends on the amount of excitation of the red, green, and blue phosphors. A white (or gray) area is the result of activating all three dots with equal intensity. Yellow is produced with the green and red dots only, magenta is produced with the blue and red dots, and cyan shows up when blue and green are activated equally. In some low-cost systems, the electron beam can only be set to on or off, limiting displays to eight colors. More sophisticated systems can set intermediate intensity levels for the electron beams, allowing several million different colors to be generated.

Composite monitors are adaptations of TV sets that allow bypass of the broadcast circuitry. These display devices still require that the picture information be combined, but no carrier signal is needed. Picture information is combined into a composite signal and then separated by the monitor, so the resulting picture quality is still not the best attainable.

Color CRTs in graphics systems are designed as **RGB monitors**. These monitors use shadow-mask methods and take the intensity level for each electron gun (red, green, and blue) directly from the computer system without any intermediate processing. High-quality raster-graphics systems have 24 bits per pixel in the frame buffer, allowing 256 voltage settings for each electron gun and nearly 17 million color choices for each pixel. An RGB color system with 24 bits of storage per pixel is generally referred to as a **full-color system** or a **true-color system.**

Flat Panel Displays:

Although most graphics monitors are still constructed with CRTs, other technologies are emerging that may soon replace CRT monitors. The term **flat-panel** display refers to a class of video devices that have reduced volume, weight, and power requirements compared to a CRT. A significant feature of flat-panel displays is that they are thinner than CRTs, and we can hang them on walls or wear them on our wrists. Since we can even write on some flat-panel displays, they will soon be available as pocket notepads. Current uses for flat-panel displays include small TV monitors, calculators, pocket video games, laptop computers, armrest viewing of movies on airlines, as advertisement boards in elevators, and as graphics displays in applications requiring rugged, portable monitors.

We can separate flat-panel displays into two categories: **emissive displays** and **nonemissive displays**. The emissive displays (or **emitters**) are devices that convert electrical energy into light. Plasma panels, thin-film electroluminescent displays, and Light-emitting diodes are examples of emissive displays. Flat CRTs have also been devised, in which electron beams arts accelerated parallel to the screen, then deflected 90' to the screen. But flat CRTs have not proved to be as successful as other emissive devices. Nonemissive displays (or **nonemitters**) use optical effects to convert sunlight or light from some other source into graphics patterns. The most important example of a nonemissive flat-panel display is a liquid-crystal device.

Plasma panels, also called gas-discharge displays, are constructed by filling the region between two glass plates with a mixture of gases that usually includes neon. A series of vertical conducting ribbons is placed on one glass panel, and a set of horizontal ribbons is built into the other glass panel (Fig. 2-11). Firing voltages applied to a pair of horizontal and vertical

conductors cause the gas at the intersection of the two conductors to break down into a glowing plasma of electrons and ions. Picture definition is stored in a refresh buffer, and the firing voltages are applied to refresh the pixel positions (at the intersections of the conductors) 60 times per second. Alternating current methods are used to provide faster application of the firing voltages, and thus brighter displays. Separation between pixels is provided by the electric field of the conductors. Figure 2-12 shows a high definition plasma panel. One disadvantage of plasma panels has been that they were strictly monochromatic devices, but systems have been developed that are now available of with multicolor capabilities

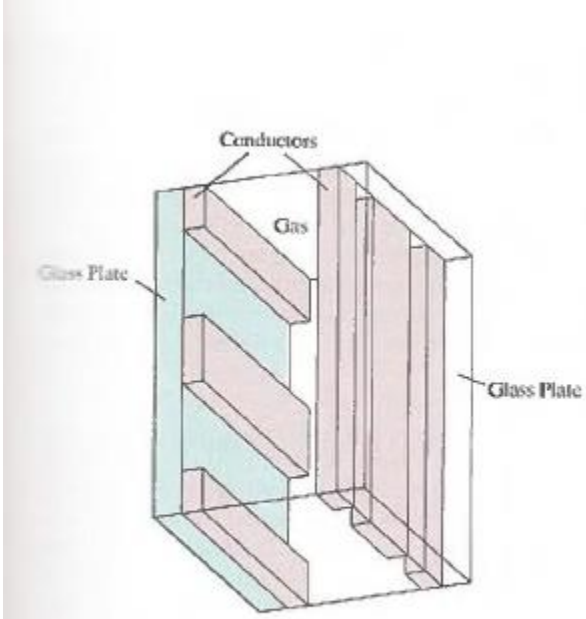


FIGURE 2-11 Basic design of a plasma-panel display device.

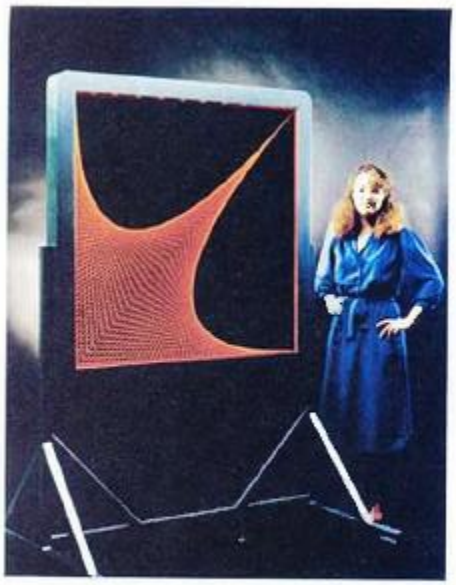


FIGURE 2-12 A plasma-panel display with a resolution of 2048 by 2048 and a screen diagonal of 1.5 meters. (*Courtesy of Photonics Systems.*)

.

Thin-film electroluminescent displays are similar in construction to a plasma panel. The difference is that the region between the glass plates is filled with a phosphor, such as zinc sulfide doped with manganese, instead of a gas (Fig. 2-13). When a sufficiently high voltage is applied to a pair of crossing electrodes, the phosphor becomes a conductor in the area of the intersection of the two electrodes. Electrical energy is then absorbed by the manganese atoms, which then release the energy as a spot of light similar to the glowing plasma effect in a plasma panel. Electroluminescent displays require more power than plasma panels, and good color and gray scale displays are hard to achieve.

A third type of emissive device is the **light-emitting diode (LED)**. A matrix of diodes is arranged to form the pixel positions in the display, and picture definition is stored in a refresh buffer. As in scan-line refreshing of a CRT, information is read from the refresh buffer and converted to voltage levels that are applied to the diodes to produce the light patterns in the display.

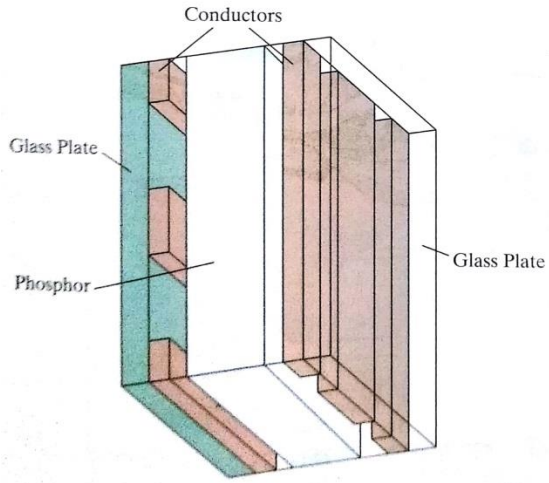


FIGURE 2-13 Basic design of a thin-film electroluminescent display device.

Liquid Crystal Displays (LCDs) are commonly used in small systems, such - as calculators and portable, laptop computers. These nonemissive devices produce a picture by passing polarized light from the surroundings or from an internal light s o w through a liquid-crystal material that can be aligned to either block or transmit the light.

The term *liquid crystal* refers to the fact that these compounds have a crystalline arrangement of molecules, yet they flow like a liquid. Flat-panel displays commonly use nematic (threadlike) liquid-crystal compounds that tend to keep the long axes of the rod-shaped molecules aligned. A flat-panel display can then be constructed with a nematic liquid crystal, as demonstrated in Fig. 2-15. Two glass plates, each containing a light polarizer at right angles to the-other plate, sandwich the liquid-crystal material. Rows of horizontal transparent conductors are built into one glass plate, and columns of vertical conductors are put into the other plate. The intersection of two conductors defines a pixel position. Normally, the molecules are aligned as shown in the "on state" of Fig. 2-165. Polarized light passing through the material is twisted so that it will pass through the opposite polarizer. The light is then reflected back to the viewer. To turn off the pixel, we apply a voltage to the two intersecting conductors to align the molecules so that the light is not twisted. This type of flat-panel device is referred to as a **passive-matrix** LCD. Picture definitions are stored in a refresh buffer, and the screen is refreshed at the rate of 60 frames per second, as in the emissive devices. Back lighting is also commonly applied using solid-state electronic devices, so that the system is not completely dependent on outside light. Colors can be displayed by using different materials or dyes and by placing a triad of color pixels at each screen location. Another method for constructing LCDs is to place a transistor at each pixel location, using thin-film transistor technology. The transistors are used to control the voltage at pixel locations and to prevent charge from gradually leaking out of the liquid-crystal cells. These devices are called **active-matrix** displays.

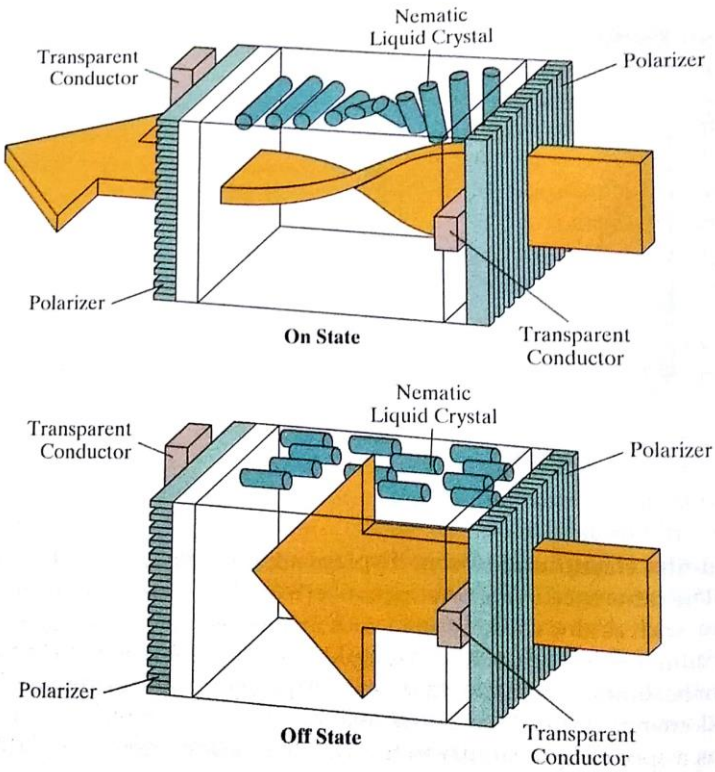


FIGURE 2-15 The light-twisting, shutter effect used in the design of most liquid-crystal display devices.

Three-Dimensional Viewing Devices:

Graphics monitors for the display of three-dimensional scenes have been devised using a technique that reflects a CRT image from a vibrating, flexible mirror. The operation of such a system is demonstrated in Fig. 2-16. As the varifocal mirror vibrates, it changes focal length. These vibrations are synchronized with the display of an object on a CRT so that each point on the object is reflected from the mirror into a spatial position corresponding to the distance of that point from a specified viewing position. This allows us to walk around an object or scene and view it from different sides.

Figure 2-17 shows the Genisco SpaceCraph system, which uses a vibrating mirror to project three-dimensional objects into a 25cm by 25cm by 25-cm volume. This system is also capable of displaying two-dimensional cross-sectional "slices" of objects selected at different depths. Such systems have been used in medical applications to analyze data from ultrasonography and CAT scan devices, in geological applications to analyze topological and seismic data, in design applications involving solid objects, and in three-dimensional simulations of systems, such as molecules and terrain.

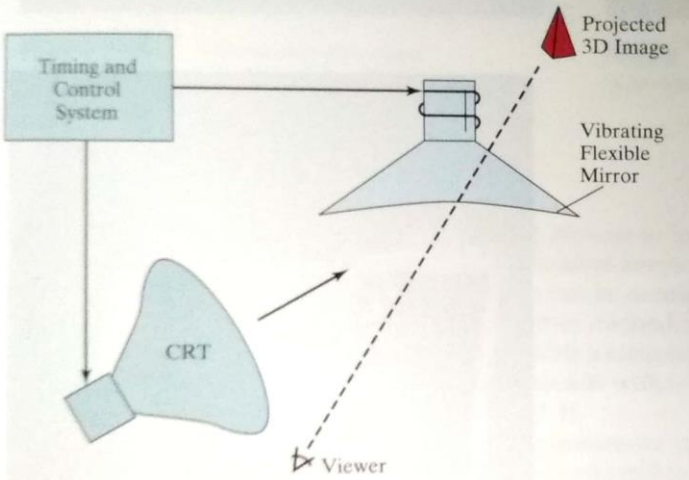


FIGURE 2-16 Operation of a three-dimensional display system using a vibrating mirror that changes focal length to match the depths of points in a scene.

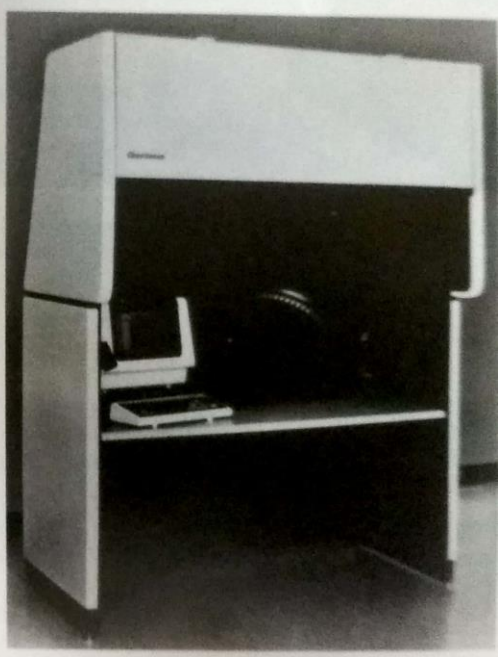


FIGURE 2-17 The SpaceGraph interactive graphics system displays objects in three dimensions using a vibrating, flexible mirror. (*Courtesy of Genisco Computers Corporation.*)

Stereoscopic and Virtual-Reality Systems:

Overview of Graphics Systems Another technique for representing three-dimensional objects is displaying stereoscopic views. This method does not produce true three-dimensional images, but it does provide a three-dimensional effect by presenting a different view to each eye of an observer so that scenes do appear to have depth (Fig. 2-18).

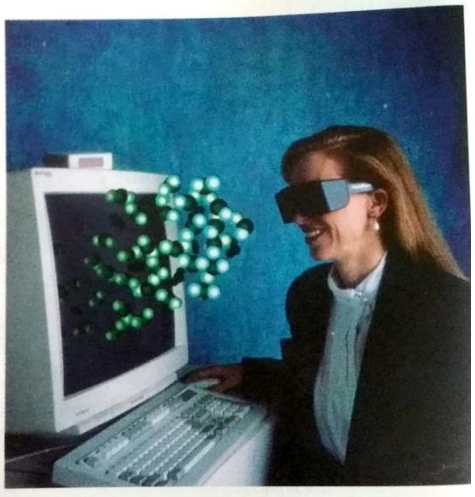


FIGURE 2-18 Simulated viewing of a stereoscopic projection. (*Courtesy of StereoGraphics Corporation.*)

To obtain a stereoscopic projection, we first need to obtain two views of a scene generated from viewing direction corresponding to each eye (left and right). We can construct the two views as computer-generated scenes with different viewing positions, or we can use a

stereo camera pair to photograph an object or scene. When we simultaneously look at the left view with the left eye and the right view with the right eye, the two views merge into a single image and we perceive a scene with depth. Figure 2-19 shows two views of a computer generated scene for Stereoscopic projection. To increase viewing comfort, the areas at the left and right edges of this scene that are visible to only one eye have been eliminated.

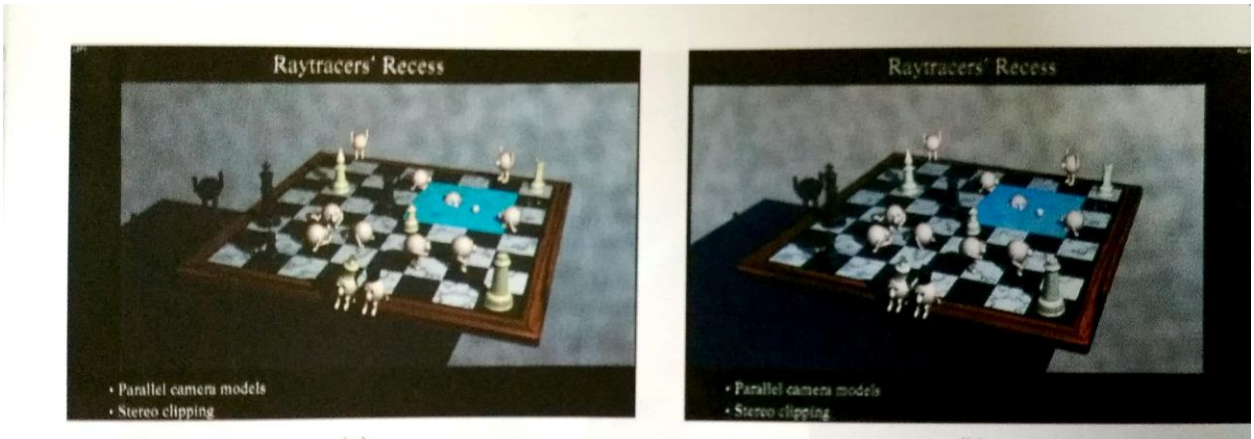


FIGURE 2-19 A stereoscopic viewing pair. (*Courtesy of Jerry Farm.*)

One way to produce a stereoscopic effect on a raster system is to display each of the two views with on alternate refresh cycles. The screen is viewed through glasses, with each lens designed to act as a rapidly alternating shutter that is synchronized to block out one of the views. Figure 2-20 shows a pair of stereoscopic glasses constructed with liquid crystal shutters and an infrared emitter that synchronizes the glasses with the views on the screen.

Stereoscopic viewing is also a component in **virtual-reality** systems, where users can step into a scene and interact with the environment. A headset (Fig. 2-21) containing an optical system to generate the stereoscopic views is commonly used in conjunction with interactive input devices to locate and manipulate objects in the scene. A sensing system in the headset keeps track of the viewer's position, so that the front and back of objects can be seen as the viewer "walks through" and interacts with the display.

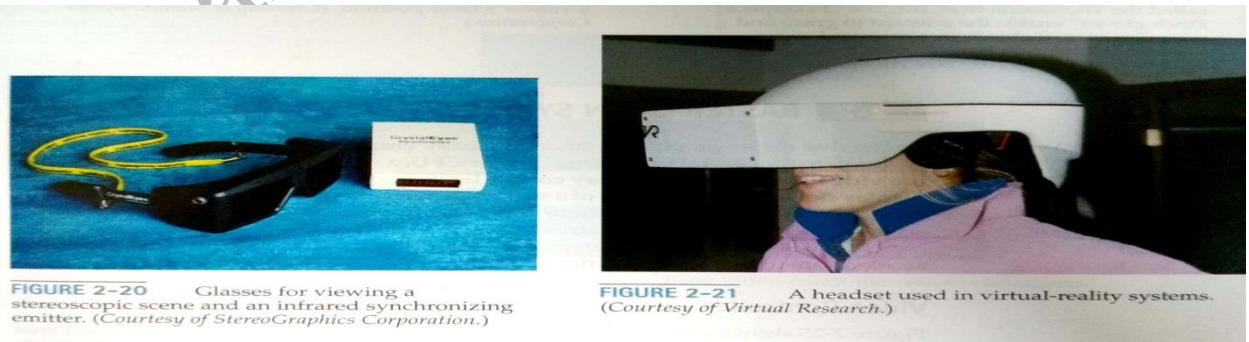


FIGURE 2-20 Glasses for viewing a stereoscopic scene and an infrared synchronizing emitter. (*Courtesy of StereoGraphics Corporation.*)

FIGURE 2-21 A headset used in virtual-reality systems. (*Courtesy of Virtual Research.*)

Another method for creating virtual reality environment is to use projectors to generate a scene within an arrangement of walls, as shown in Figure 2-22, where a viewer interacts with a virtual display using stereoscopic glasses and data gloves.

An interactive virtual-reality environment can also be viewed with stereoscopic glasses and a video monitor, instead of a headset. This provides a means for obtaining a lowercost virtual-reality system. As an example, Fig. 2-23 shows an ultrasound tracking device with six degrees of freedom. The tracking device is placed on top of the video display and is used to monitor head movements so that the viewing position for a scene can be changed as head position changes.

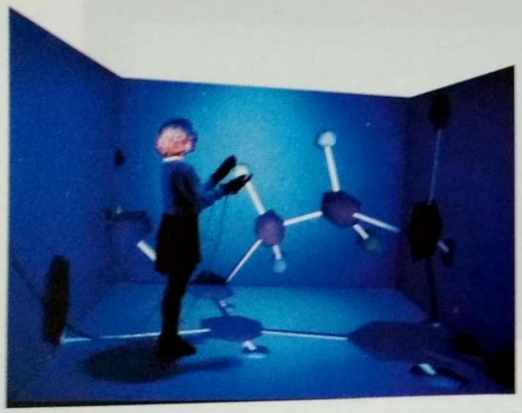


FIGURE 2-22 A molecular biologist analyzing molecular structures inside a virtual-reality system called the Trimension ReaCTor. The "Fakespace Pinch gloves" enable the scientist to grasp and rearrange virtual objects in a projected scene. (*Courtesy Silicon Graphics, Inc. and Trimension Systems ReaCTor. © 2003 SGI. All rights reserved.*)

FIGURE 2-23 An ultrasound tracking device used with stereoscopic glasses to record changes in a viewer's head position. (*Courtesy of StereoGraphics Corporation.*)

RASTER-SCAN SYSTEMS:

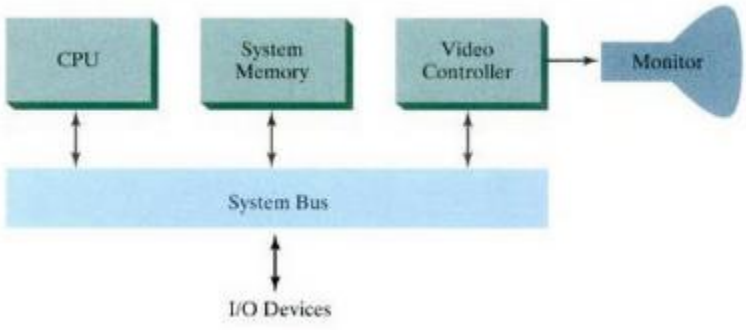


FIGURE 2-24
Architecture of a simple raster-graphics system.

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. 2-24. 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 co- processors and accelerators to implement various graphics operations.

Video Controller:

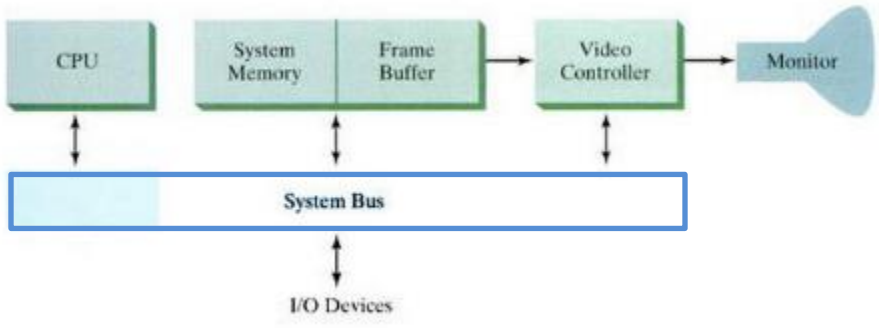


FIGURE 2-25
Architecture of a raster system with a fixed portion of the system memory reserved for the frame buffer.

Figure 2-25 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.

FIGURE 2-26 A Cartesian reference frame with origin at the lower-left corner of a video monitor.

Frame-buffer locations, and the corresponding screen positions, are referenced in Cartesian coordinates. In an application program, we use the command within a graphics software package to set coordinate positions for displayed objects relative to the origin of the Cartesian reference frame. The coordinate origin is referenced at the lower left comer of a screen display area by the software commands, although we can typically set the origin at any convenient location for a particular application. Fig. 2-26 shows a two dimensional Cartesian reference frame with origin at the lower left screen corner. The screen surface is then represented as the first quadrant of a two-dimensional system, with positive x values increasing to the right and positive y values increasing from bottom to top. Pixel positions are then assigned integer x values that range from 0 to x_{max} across the screen, left to right, and integer y values that vary from 0 to y_{max}, bottom to top. (On some software systems, the coordinate origin is referenced at the upper left comer of the screen, so the y values are inverted.)

In Fig. 2-27, the basic refresh operations of the video controller are diagrammed. Two registers are used to store the coordinates of the screen pixels. Initially, the x register is set to 0 and the y register is set to y. The value stored in the frame buffer for this pixel position is then retrieved and used to set the intensity of the CRT beam. Then the x register is incremented by 1,

and the process repeated for the next pixel on the top scan line. This procedure is repeated for each pixel along the scan line. After the last pixel on the top scan line has been processed, the x register is reset to 0 and the y register is decremented by 1. Pixels along this scan line are then processed in turn, and the procedure is repeated for each successive scan line. After cycling through all pixels along the bottom scan line (y = O), the video controller resets the registers to the first pixel position on the top scan line and the refresh process starts over.

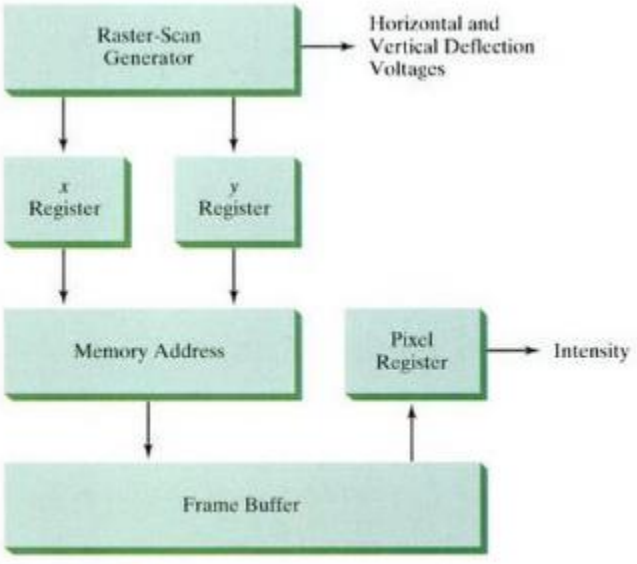


FIGURE 2-27 Basic video-controller refresh operations.

Since the screen must be refreshed at the rate of 60 frames per second, the simple procedure illustrated in Fig. 2-27 cannot be accommodated by typical RAM chips. The cycle time is too slow. To speed up pixel processing, video controllers can retrieve multiple pixel values from the refresh buffer on each pass. The multiple pixel intensities are then stored in a separate register and used to control the CRT beam intensity for a group of adjacent pixels. When that group of pixels has been processed, the next block of pixel values is retrieved from the frame buffer.

A number of other operations can be performed by the video controller, besides the basic refreshing operations. For various applications, the video controller can retrieve pixel intensities from different memory areas on different refresh cycles. In high quality systems, for example, two have buffers are often provide so that one buffer can be used for refreshing while the other is being filled with intensity values. Then the two buffers can switch roles. This provides a fast mechanism-for generating real-time animations, since different views of moving objects can be successively loaded into the refresh buffers. Also, some transformations can be accomplished by the video controller. Areas of the screen can be enlarged, reduced, or moved from one location to another during the refresh cycles. In addition, the video controller often contains a lookup table, so that pixel values in the frame buffer are used to access the lookup table instead of controlling the CRT beam intensity directly. This provides a fast method for changing screen intensity

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

Raster-Scan Display Processor

Figure 2-28 shows one way to set up the organization of a raster system containing a separate display processor, sometimes referred to as a graphics controller or a display coprocessor. The purpose of the display processor is to free the CPU from the graphics chores. In addition to the system memory, a separate display- processor memory area can also be provided.

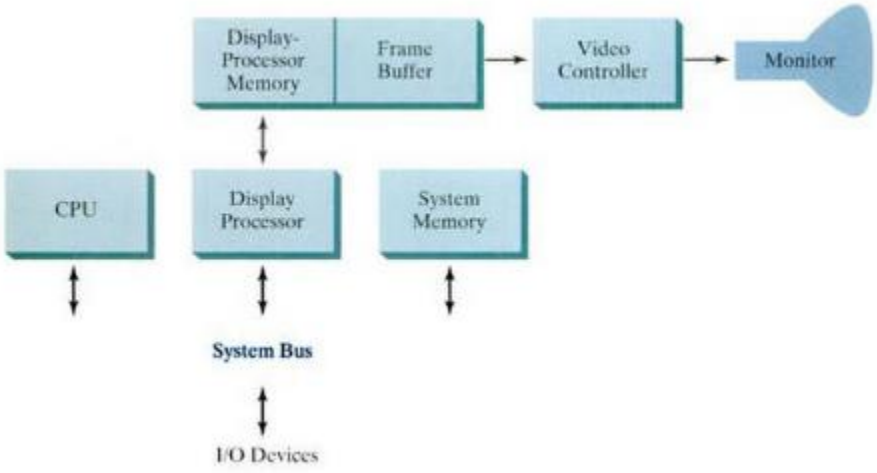


FIGURE 2-28
Architecture of a raster-graphics system with a display processor.

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

Graphics commands specifying straight lines and other geometric objects are scan converted into a set of discrete intensity points. Scan converting a straight-line segment, for example, means that we have to locate the pixel positions closest to the line path and store the intensity for each position in the frame buffer. Similar methods are used for scan converting curved lines and polygon outlines. Characters can be defined with rectangular grids, as in Fig. 2-29, or they can be defined with curved outlines, as in Fig. 2-30. The array size for character grids can vary from about 5 by 7 to 9 by 12 or more for higher-quality displays. A character grid is displayed by superimposing the rectangular grid pattern into the frame buffer at a specified coordinate position. With characters that are defined as curve outlines, character shapes are scan converted into the frame buffer.

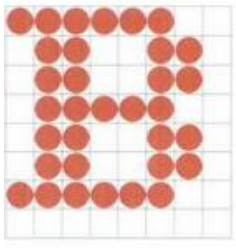


FIGURE 2-29 A character defined as a rectangular grid of pixel positions.

FIGURE 2-30 A character defined as an outline shape.

Display processors are also designed to perform a number of additional operations. These functions include generating various line styles (dashed, dotted, or solid), displaying color areas, and performing certain transformations and manipulations on displayed objects. Also, display processors are typically designed to interface with interactive input devices, such as a mouse.

In an effort to reduce memory requirements in raster systems, methods have been devised for organizing the frame buffer as a linked list and encoding the color information. One way to do this is to store each scan line as a set of number pairs. The first number in each pair indicates a color value, and the second number specifies the number of adjacent pixels on the scan line that are to have that color. This technique, called **run-length encoding** can result in a considerable saving in storage space if a picture is to be constructed mostly with long runs of a single color each. A similar approach can be taken when pixel colors change linearly. Another approach is to encode the raster as a set of rectangular areas (**cell encoding**). The disadvantages of encoding runs are that color changes are difficult to make and storage requirements actually increase as the length of the runs decreases. In addition, it is difficult for the display controller to process the raster when many short runs are involved.

2-3 GRAPHICS WORKSTATIONS AND VIEWING SYSTEMS

Most graphics monitors today operate as raster-scan displays, and both CRT and flat-panel systems are in common use. Graphics workstations range from small general-purpose computer systems to multi-monitor facilities, often with ultra-large viewing screens. For a personal computer, screen resolutions vary from about 640 by 480 to 1280 by 1024, and diagonal screen lengths measure from 12 inches to over 21 inches. Most general-purpose systems now have considerable color capabilities, and many are full-color systems. For a desktop workstation specifically designed for graphics applications, the screen resolution can vary from 1280 by 1024 to about 1600 by 1200, with a typical screen diagonal of 18 inches or more. Commercial workstations can also be obtained with a variety of devices for specific applications. Figure 2-31 shows the features in one type of artist's workstation.

High-definition graphics systems, with resolutions up to 2560 by 2048, are commonly used in medical imaging, air-traffic control, simulation, and CAD. A 2048 by 2048 flat-panel display is shown in Fig. 2-32.

Many high-end graphics workstations also include large viewing screens, often with specialized features. Figure 2-33 shows a large-screen system for stereoscopic viewing, and Fig. 2-34 is a multi-channel wide-screen system.

Multi-panel display screens are used in a variety of applications that require "wall-sized" viewing areas. These systems are designed for presenting graphics displays at meetings, conferences, conventions, trade shows, retail stores, museums, and passenger terminals. A multi-panel display can be used to show a large view of a single scene or several individual images. Each panel in the system displays one section of the overall picture, as illustrated in Fig. 2-35. Large graphics displays can also be presented on curved viewing screens, such as the system in Fig. 2-36. A large, curved-screen system can be useful for viewing by a group of people studying a particular graphics application, such as the examples in Figs. 2-37 and 2-38. A control center, featuring a battery of

FIGURE 2-31 An artist's workstation, featuring a monitor, a keyboard, a graphics tablet with a hand cursor, and a light table, in addition to data storage and telecommunications devices. (*Courtesy of DICOMED Corporation.*)

FIGURE 2-32 A high-resolution (2048 by 2048) graphics monitor. (*Courtesy of BarcoView.*)

FIGURE 2-33 The SGI Reality Center 2000D, featuring an ImmersaDesk R2 and displaying a large-screen stereoscopic view of pressure contours in a vascular blood-flow simulation superimposed on a volume-rendered anatomic data set. (*Courtesy*

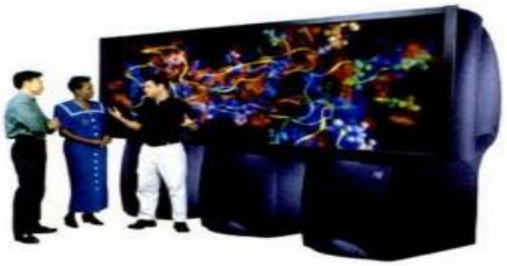


FIGURE 2-34 A wide-screen view of a molecular system displayed on the three-channel SGI Reality Center 3300W. (*Courtesy of Silicon Graphics, Inc. and Molecular Simulations. © 2003 SGI. All rights reserved.*)

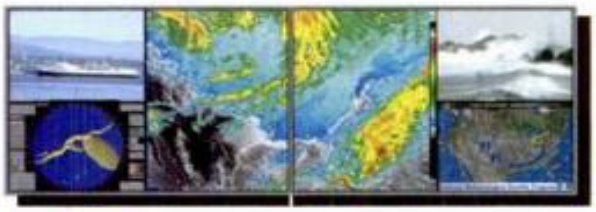


FIGURE 2-35 A multi-panel display system called the "Super Wall". (*Courtesy of RGB Spectrum.*)

FIGURE 2-36 A homeland security study displayed using a system with a large curved viewing screen. (*Courtesy of Silicon Graphics, Inc. © 2003. All rights reserved.*)

standard monitors, allows an operator to view sections of the large display and to control the audio, video, lighting, and projection systems using a touch-screen menu. The system projectors provide a seamless, multichannel display that includes edge blending, distortion correction, and color balancing. And a surround-sound system is used to provide the audio environment. Fig 2-39 shows a 360° paneled viewing system in the NASA control-tower simulator, which is used for training and for testing ways to solve air-traffic and runway problems at airports.

FIGURE 2-37 A curved-screen graphics system displaying an interactive walk-through of a natural gas plant. (*Courtesy of Silicon Graphics, Inc., Trimension Systems, and the Cadcentre, Cortaillod, Switzerland. © 2003 SGI. All rights reserved.*)

FIGURE 2-38 A geophysical visualization presented on a 25-foot semicircular screen, which provides a 160° horizontal and 40° vertical field of view. (*Courtesy of Silicon Graphics, Inc., the Landmark Graphics Corporation, and Trimension Systems. © 2003 SGI. All rights reserved.*)

FIGURE 2-39 The 360° viewing screen in the NASA airport control-tower simulator, called the FutureFlight Central Facility. (*Courtesy of Silicon Graphics, Inc. and NASA. © 2003 SGI. All rights reserved.*)

2-4 INPUT DEVICES

Graphics workstations can make use of various devices for data input. Most systems have a keyboard and one or more additional devices specifically designed for interactive input. These include a mouse, trackball, spaceball, and joystick. Some other input devices used in particular applications are digitizers, dials, button boxes, data gloves, touch panels, image scanners, and voice systems.

Keyboards, Button Boxes, and Dials

An alphanumeric **keyboard** on a graphics system is used primarily as a device for entering text strings, issuing certain commands, and selecting menu options. 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.

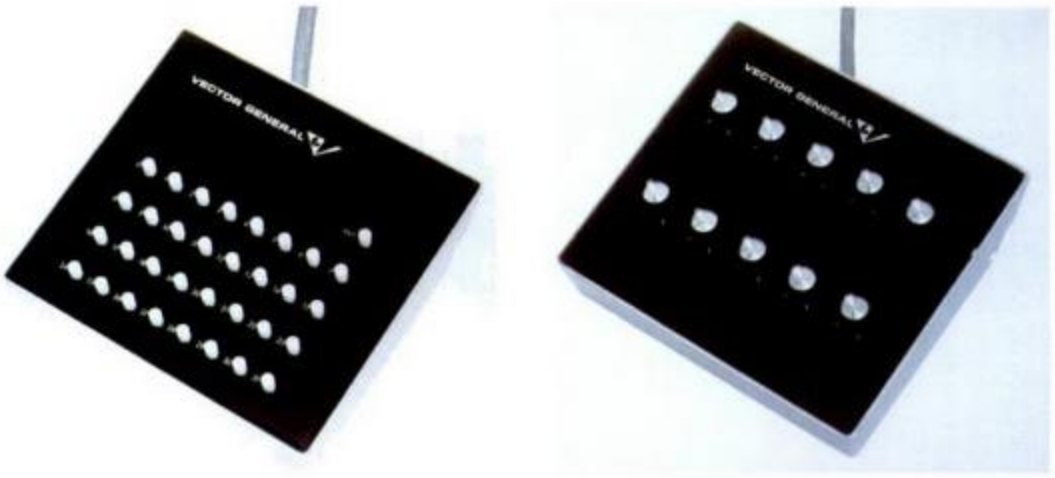
Cursor-control keys and function keys are common features on general-purpose keyboards. Function keys allow users to select frequently accessed operations with a single keystroke, and cursor-control keys are convenient for selecting a displayed object or a location by positioning the screen cursor. A keyboard can also contain other types of cursor-positioning devices, such as a trackball or joystick, along with a numeric keypad for fast entry of numeric data. In addition to these features, some keyboards have an ergonomic design (Fig. 2-40) that provides adjustments for relieving operator fatigue.

For specialized tasks, input to a graphics application may come from a set of buttons, dials, or switches that select data values or customized graphics operations. Figure 2-41 gives an example of a **button box** and a set of **input dials.** Buttons and switches are often used to input predefined functions, and dials are common devices for entering scalar values. Numerical values within some defined range are selected for input with dial rotations. A potentiometer is used to measure dial rotation, which is then converted to the corresponding numerical value.

Mouse Devices

Figure 2-40 illustrates a typical design for one-button **mouse,** which is a smali hand-held unit that is usually moved around on a flat surface 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

FIGURE 2-40
Ergonomically designed keyboard with removable palm rests. The slope of each half of the keyboard can be adjusted separately. A one-button mouse, shown in front of the keyboard, has a cable attachment for connection to the main computer unit. (*Courtesy of Apple Computer, Inc.*)



(a) (b)

FIGURE 2-41 A button box (a) and a set of input dials (b). (*Courtesy of Vector General.*)

FIGURE 2-42 The Z mouse features three buttons, a mouse ball underneath, a thumbwheel on the side, and a trackball on top. (*Courtesy of the Multipoint Technology Corporation.*)

motion is with an optical sensor. For some optical 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. Other optical mouse systems can operate on any surface. And some are cordless, communicating with computer processors using digital radio technology.

Since a mouse can be picked up and put down at another position without change in cursor movement, it is used for making relative changes in the position of the screen cursor. One, two, three, or four buttons are included on the top of the mouse for signaling the execution of operations, such as recording cursor position or invoking a function. Most general-purpose graphics systems now include a mouse and a keyboard as the primary input devices.

Additional features can be included in the basic mouse design to increase the number of allowable input parameters. The Z mouse in Fig. 2-42 has three buttons, a thumbwheel on the side, a trackball on the top, and a standard mouse

ball underneath. This design provides six degrees of freedom to select spatial positions, rotations, and other parameters. With the Z mouse, we can select an object displayed on a video monitor, rotate it, and move it in any direction. We could also use the Z mouse to navigate our viewing position and orientation through a three-dimensional scene. Applications of the Z mouse include virtual reality, CAD, and animation.

Trackballs and Spaceballs

A trackball is a ball device that can be rotated with the fingers or palm of the hand to produce screen-cursor movement. Potentiometers, connected to the ball, measure the amount and direction of rotation. Laptop keyboards are often equipped with a trackball to eliminate the extra space required by a mouse. A trackball can be mounted also on other devices, such as the Z mouse shown in Fig. 2-42, or it can be obtained as a separate add-on unit that contains two or three control buttons.

An extension of the two-dimensional trackball concept is the spaceball (Fig. 2-44), which provides six degrees of freedom. Unlike the trackball, a spaceball does not actually move. Strain gauges measure the amount of pressure applied to the spaceball to provide input for spatial positioning and orientation as the ball is pushed or pulled in various directions. Spaceballs are used for three-dimensional positioning and selection operations in virtual-reality systems, modeling, animation, CAD, and other applications.

Joysticks

Another positioning device is the joystick, which consists of a small, vertical lever (called the stick) mounted on a base. We use the joystick to steer the screen cursor around. Most joysticks, such as the unit in Fig. 2-43, select screen positions with actual stick movement; others respond to pressure on the stick. Some joysticks are mounted on a keyboard, and some are designed as stand-alone units.

The distance that the stick is moved in any direction from its center position corresponds to the relative screen-cursor movement in that direction.

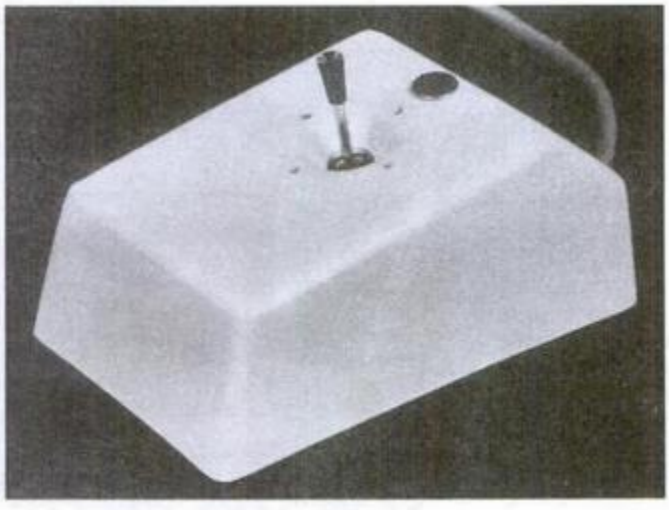


FIGURE 2-43 A movable joystick. (*Courtesy of the CalComp Group, Sanders Associates, Inc.*)

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 actions that are to be executed once a screen position has been selected.

In another type of movable joystick, the stick is used to activate switches that cause the screen cursor to move at a constant rate in the direction selected. Eight switches, arranged in a circle, are sometimes provided so that the stick can select any one of eight directions for cursor movement. Pressure-sensitive joysticks, also called *isometric joysticks*, have a non-movable stick. A push or pull on the stick is measured with strain gauges and converted to movement of the screen cursor in the direction of the applied pressure.

Data Gloves

Figure 2-44 shows a **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 are 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 reference system. Input from the glove is 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.

Digitizers

A common device for drawing, painting, or interactively selecting positions is a **digitizer.** These devices can be designed to input coordinate values in either a two-dimensional or a three-dimensional space. In engineering or architectural applications, a digitizer is often used to scan a drawing or object and to input a set of discrete coordinate positions. The input positions are then joined with straight-line segments to generate an approximation of a curve or surface shape.

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. A hand cursor contains cross hairs for sighting positions, while a stylus is a pencil-shaped device that is pointed at positions on the tablet. Figures 2-45 and 2-46 show examples of desktop and

FIGURE 2-44 A virtual-reality scene, displayed on a two-dimensional video monitor, with input from a data glove and a spaceball. (*Courtesy of The Computer Graphics Center, Darmstadt, Germany.*)

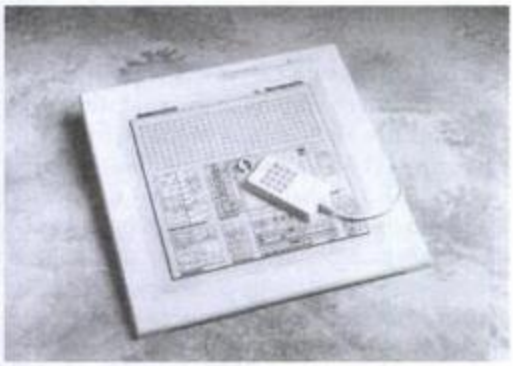


FIGURE 2-45 The SummaSketch III desktop tablet with a sixteen-button hand cursor. (*Courtesy of Summagraphics Corporation.*)

FIGURE 2-46 The Microgrid III tablet with a sixteen-button hand cursor, designed for digitizing larger drawings. (*Courtesy of Summagraphics Corporation.*)

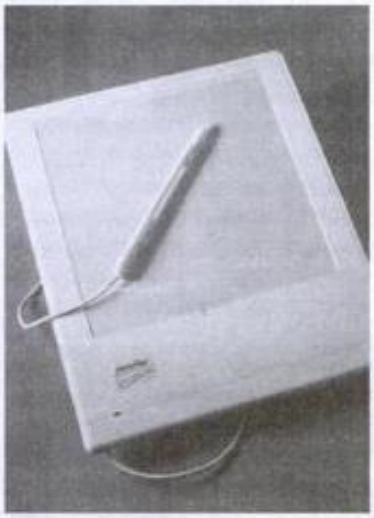


FIGURE 2-47 The NotePad desktop tablet with stylus. (*Courtesy of CalComp Digitizer Division, a part of CalComp, Inc.*)

FIGURE 2-48 An artist's digitizer system, with a pressure-sensitive, cordless stylus. (*Courtesy of Wacom Technology Corporation.*)

floor-model tablets, using hand cursors that are available with two, four, or sixteen buttons. Examples of stylus input with a tablet are shown in Figs. 2-47 and 2-48. The artist's digitizing system in Fig. 2-48 uses electromagnetic resonance to detect the three-dimensional position of the stylus. This allows an artist to produce different brush strokes by applying different pressures to the tablet surface. Tablet size varies from 12 by 12 inches for desktop models to 44 by 60 inches or

larger for floor models. Graphics tablets provide a highly accurate method for selecting coordinate positions, with an accuracy that varies from about 0.2 mm on desktop models to about 0.05 mm or less on larger models.

Many graphics tablets are constructed with a rectangular grid of wires embedded in the tablet surface. Electromagnetic pulses are generated in sequence along the wires, and an electric signal is induced in a wire coil in an activated stylus or hand-cursor to record a tablet position. Depending on the technology, signal strength, coded pulses, or phase shifts can be used to determine the position on the tablet.

An *acoustic* (or *sonic*) *tablet* uses sound waves to detect a stylus position. Either strip microphones or point microphones can be employed to detect the sound emitted by an electrical spark from a stylus tip. The position of the stylus is calculated by timing the arrival of the generated sound at the different microphone positions. An advantage of two-dimensional acoustic tablets is that the microphones can be placed on any surface to form the "tablet" work area. For example, the microphones could be placed on a book page while a figure on that page is digitized.

Three-dimensional digitizers use sonic or electromagnetic transmissions to record positions. One electromagnetic transmission method is similar to that employed in the data glove: a coupling between the transmitter and receiver is used to compute the location of a stylus as it moves over an object surface. Figure 2-49 shows a digitizer recording the locations of positions on the surface of a three-dimensional object. As the points are selected on a nonmetallic object, a wire-frame outline of the surface is displayed on the computer screen. Once the surface outline is constructed, it can be rendered using lighting effects to produce a realistic display of the object. Resolution for this system is from 0.8 mm to 0.08 mm, depending on the model.

Image Scanners

Drawings, graphs, photographs, or text can be stored for computer processing with an **image scanner** by passing an optical scanning mechanism over the

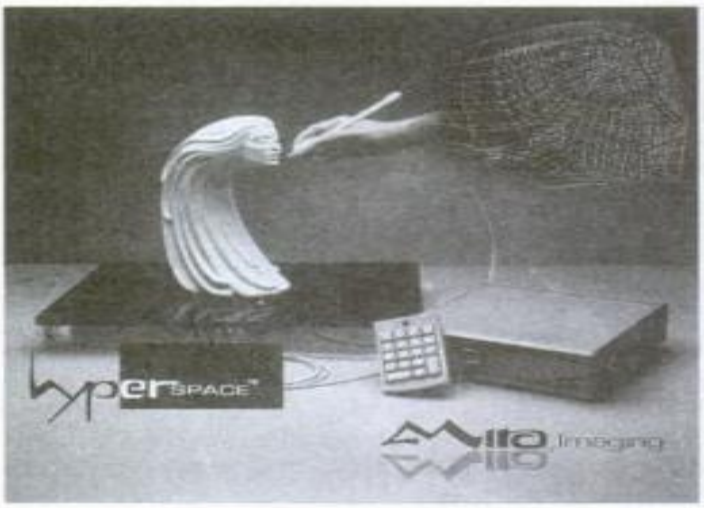
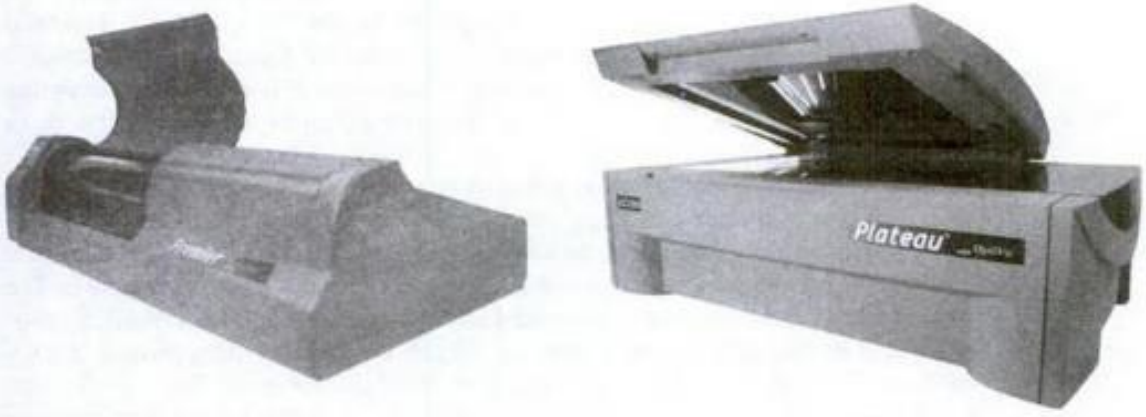


FIGURE 2-49 A three-dimensional digitizing system for use with Apple Macintosh computers. (*Courtesy of Mira Imaging.*)

information to be stored. The gradations of gray scale or color are then recorded and stored in an array. Once we have the internal representation of a picture, we can apply transformations to rotate, scale, or crop the picture to a particular screen area. We can also apply various image-processing methods to modify the array representation of the picture. For scanned text input, various editing operations can be performed on the stored documents. Scanners are available in a variety of sizes and capabilities. A small hand-model scanner is shown in Fig. 2-50, while Figs. 2-51 and 2-52 show larger models.

FIGURE 2-50 A hand-held scanner that can be used to input either text or graphics images. (*Courtesy of Thunderware, Inc.*)



(a) (b)

FIGURE 2-51 Desktop scanners: (a) drum scanner and (b) flatbed scanner. (*Courtesy of Aztek, Inc., Lake Forest, California.*)

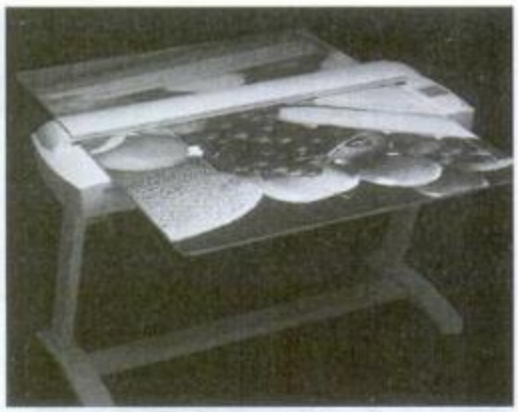


FIGURE 2-52
A wide-format scanner. (*Courtesy of Aztek, Inc., Lake Forest, California.*)

(a)

(b)

FIGURE 2-53 Plasma panels with touch screens. (*Courtesy of Photonics Systems.*)

Touch Panels

As the name implies, **touch panels** allow displayed objects or screen positions to be selected with the touch of a finger. A typical application of touch panels is for the selection of processing options that are represented as a menu of graphical icons. Some monitors, such as the plasma panels shown in Fig. 2-53, are designed with touch screens. Other systems can be adapted for touch input by fitting a transparent device (Fig. 2-54) containing a touch-sensing mechanism over the video monitor screen. Touch input can be recorded using optical, electrical, or acoustical methods.

Optical touch panels employ a line of infrared light-emitting diodes (LEDs) along one vertical edge and along one horizontal edge of the frame. Light detectors are placed along the opposite vertical and horizontal edges. These detectors are used to record which beams are interrupted when the panel is touched. The two crossing beams that are interrupted identify the horizontal and vertical coordinates of the screen position selected. Positions can be selected with an accuracy of about 1/4 inch. With closely spaced LEDs, it is possible to break two horizontal or two vertical beams simultaneously. In this case, an average position between the two interrupted beams is recorded. The LEDs operate at infrared frequencies so that the light is not visible to a user.

An electrical touch panel is constructed with two transparent plates separated by a small distance. One of the plates is coated with a conducting material, and

FIGURE 2-54 Resistive touch-screen overlays. (*Courtesy of Elo TouchSystems, Inc.*)

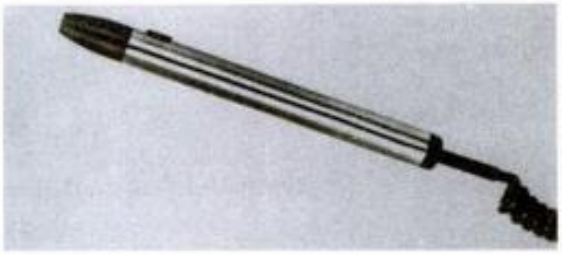


FIGURE 2-55 A light pen with a button activator. (*Courtesy of Interactive Computer Products.*)

the other plate is coated with a resistive material. When the outer plate is touched, it is forced into contact with the inner plate. This contact creates a voltage drop across the resistive plate that is converted to the coordinate values of the selected screen position.

In acoustical touch panels, high-frequency sound waves are generated in horizontal and vertical directions across a glass plate. Touching the screen causes part of each wave to be reflected from the finger to the emitters. The screen position at the point of contact is calculated from a measurement of the time interval between the transmission of each wave and its reflection to the emitter.

Light Pens

Figure 2-55 shows the design of one type of **light pen.** Such pencil-shaped devices are used to select screen positions by detecting the light coming from points on the CRT screen. They are sensitive to the short burst of light emitted from the phosphor coating at the instant the electron beam strikes a particular point. Other light sources, such as the background light in the room, are usually not detected by a light pen. An activated light pen, pointed at a spot on the screen as the electron beam lights up that spot, generates an electrical pulse that causes the coordinate position of the electron beam to be recorded. As with cursor-positioning devices, recorded light-pen coordinates can be used to position an object or to select a processing option.

Although light pens are still with us, they are not as popular as they once were since they have several disadvantages compared to other input devices that have been developed. For example, when a light pen is pointed at the screen, part of the screen image is obscured by the hand and pen. And prolonged use of the light pen can cause arm fatigue. Also, light pens require special implementations for some applications since they cannot detect positions within black areas. To be able to select positions in any screen area with a light pen, we must have some nonzero light intensity emitted from each pixel within that area. In addition, light pens sometimes give false readings due to background lighting in a room.

Voice Systems

Speech recognizers are used with some graphics workstations as input devices for voice commands. The **voice system** input can be used to initiate graphics

FIGURE 2-56 A speech-recognition system. (*Courtesy of Threshold Technology, Inc.*)

operations or to enter data. These systems operate by matching an input against a predefined dictionary of words and phrases.

A dictionary is set up by speaking the command words several times. The system then analyzes each word and establishes a dictionary of word frequency patterns, along with the corresponding functions that are to be performed. Later, when a voice command is given, the system searches the dictionary for a frequency-pattern match. A separate dictionary is needed for each operator using the system. Input for a voice system is typically spoken into a microphone mounted on a headset, as in Fig. 2-56, and the microphone is designed to minimize input of background sounds. Voice systems have some advantage over other input devices, since the attention of the operator need not switch from one device to another to enter a command.

Graphics Networks:

- Various resources, such as processors, printers, plotters and data files, can be distributed on a network and shared by multiple users.
- A graphics monitor on a network is generally referred to as a **graphics server**, or simply a server. The monitor includes standard input devices such as a keyboard and a mouse or trackball. In that case, the system can provide input, as well as being an output server.
- The computer on the network that is executing the graphics application program is called the **client**, and the output of the program is displayed on the server.
- A workstation that includes processors, as well as a monitor and input devices, can function as both a server and a client.

Graphics on the Internet:

- The **World Wide Web** provides the hypertext system that allows users to locate and view documents that contain text, graphics and audio.
- Resuorces such as graphics files are identified by the **uniform resource locator (URL).**
- The URL contains two parts:
  1.) the protocol for transferring the document, and
  2.) the server that contains document and optionally, the location (directoty) on the server.
- Documents on the internet can be constructed with the Hypertext Markup Language (HTML).

- The National Center for Supercomputing Applications(NCSA) developed a "browser" called Mosaic that made it easier for users to search for web resources. The Mosaic browser later evolved into the browser called Netscape Navigator.
- The HTML provides a simple method for developing graphics on the Internet, but has limited capabilities. Therefore other languages have been developed for grphics applications.

Graphics Software:

There are two general classifications for graphics software:

1. Special-purpose applications packages and
2. General programming packages and

Special-purpose application packages:

- Special-purpose applications packages are designed for nonprogrammers who want to generate pictures, graphs or charts in some application area without worrying about the graphics procedures that might be needed to produce such displays.
- The interface to a special purpose package is typically a set of menus that allows the users to communicate with the programs in their own terms.
- Ex: Artist's painting program ans various architectural, business, medical and engineering CAD systems.

General programming packages:

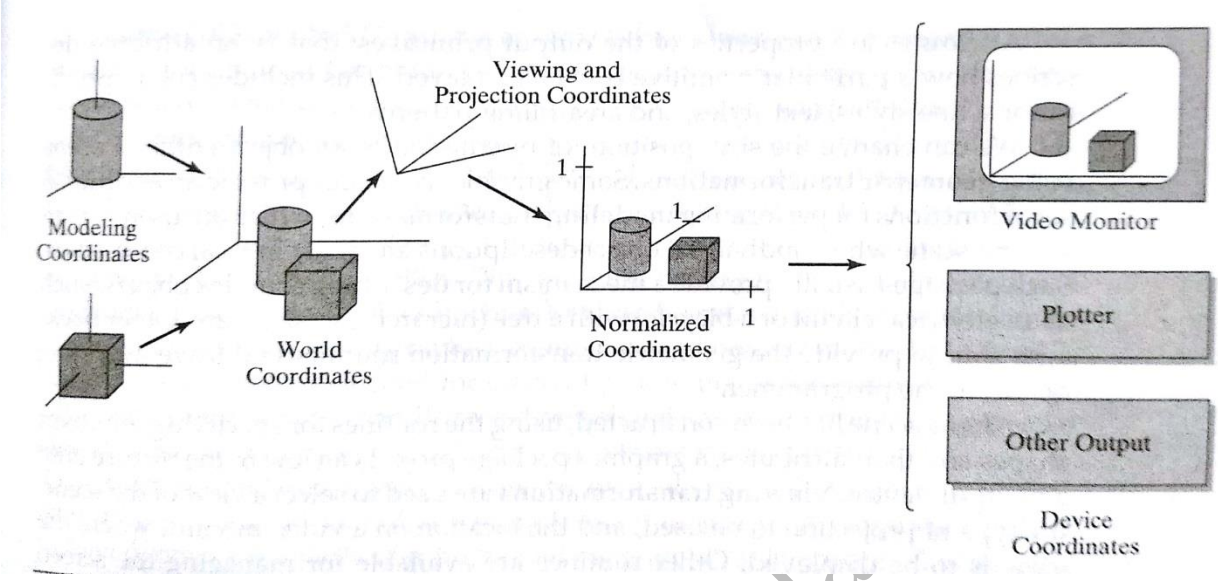
- General programming package provides a library of graphic functions that can be used in a programming language such as C, C++,Java or Fortran.
- Basic functions in a typical graphics library include those for specifying picture components (straight lines, polygons, etc) , setting color values, selecting views of a scene, applying rotations and other transformations.
- Example of General programming packages:
    - GL (graphics library), OpenGL, VRML (Virtual Reality Modeling Language), Java 2D and Java 3D.
- A set of graphic functions is called as **Computer Graphics Application Programming Interface (CG API)**, because the library provides a software interface between a programming language and the hardware.

Coordinate Representations:

- To generate a picture using a programming package, we first need to give geometric descriptions of the objects, that are to be displayed.
- These descriptions determine the locations and shapes of the objects.
- For example, a box is specified by the positions of its corners (vertices)

- General graphics packages require geometric descriptions to be specified in a standard, right handed, Cartesian-coordinate reference frame.
- 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 construct a scene by placing the objects into appropriate locations within the scene using a reference frame called **world coordinates**.
- 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.
- After all parts of a scene have been specified, the overall world-coordinate description is processed through various routines onto one or more output device reference frame for display. This process is called **viewing pipeline.**
- World coordinate positions are then converted to **viewing coordinates** corresponding to the view we want of a scene, based on the position and orientation of a hypothetical camera.
- Then the object coordinates are transformed into **projection coordinates** , which corresponds to what we see on the output device.
- Then the object coordinates are transformed into **normalized device coordinates** , which makes a graphic package independent of the coordinate range for any specific output device.
- 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.

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



- Deice coordinates (x_{dc},y_{dc}) are integers within the range (0,0) to (x_{max},y_{max}) for a particular device.

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.

Graphics output primitives:

- The basic building blocks for pictures
- 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:

- Attributes are the properties of the output primitives; that is, an attribute describes how a particular primitive is to be displayed.
- They include color specifications, line styles, text styles, and area-filling patterns.

Geometric transformations:

- We can change the size, position, or orientation of an object within a scene using geometric transformations.

Modelling Coordinates:

- They used to construct a scene using object descriptions given in local coordinates.

Viewing transformations:

- 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 select a view of the scene. The type of projection used and the location on a video monitor where the view is to be displayed.

Other routines are available for managing the screen display area by specifying its position, size, and structure. For three dimensional scenes, visible objects are identified and the lighting conditions are applied.

Input functions:

- 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 these interactive devices.

Control operations:

- Finally, a graphics package contains a number of housekeeping tasks, such as clearing a screen display area to a selected color and initializing parameters. We can lump the functions for carrying out these chores under the heading control operations.

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 hierarchical object

rnodeling, 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.
- The workstations from Silicon Graphics ,Inc. (SGI) came with a set of routines called **GL (Graphics Library)** which became a de facto graphics standard.
- The GL routines were designed for fast , real-time rendering , and soon this package was being extended to other systems. As a result, **OpenGL** was developed as a hardware independent version of GL in the 1990s. This graphics package is now maintained and updated by the **OpenGL Architecture Review Board**, which is a consortium of representatives from many graphics companies and organizations.

Graphics functions in any package are typically defined as a set of specifications that are independent of any programming language. A **language binding** is then defined for a particular high-level programming language. This binding gives the syntax for accessing the various graphics functions from that language. 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. Specifications for implementing a graphics package in a particular language are set by the International Standards Organization. The OpenGL bindings for C and C++ languages are the same. Other OpenGL bindings are also available, such as those for Ada and Fortran.

Other Graphics Packages:

Many other computer-graphics programming libraries have been developed. Some provide general graphics routines, and some are aimed at specific applications or particular aspects of computer graphics, such as animation, virtual reality, or graphics on the internet.

- A package called ***Open Inventor*** furnishes a set of object-oriented routines for describing a scene that is to be displayed with calls to OpenGL.
- The ***Virtual Reality Modeling Language (VRML)*** allows us to set up three-dimensional models of virtual worlds on the internet.
- We can also construct pictures on the web using graphics libraries developed for the Java language. Ex. Java 2D, Java 3D.
- The ***Renderman Interface*** from the Pixar Corporation , we can generate scenes using a variety of lighting models.
- Finally, graphics libraries are often provided in other types of systems, such as Mathematica, Matlab and Maple.

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. Since OpenGL is designed to be hardware independent, therefore many operations, such as input and output routines are not included in the basic library. However, input and output routines and many additional functions are available in auxiliary libraries that have developed for OpenGL programs.

Basic OpenGL Syntax:

Functions in **OpenGL basic library** are prefixed with gl, and each component word within a function name has its first letter capitalized.

Ex:

glBegin, glClear, glCopyPixels, glPolygonMode

Certain functions require that one (or more) of their arguments be assigned a symbolic constant specifying, for instance, a parameter name, a value for a parameter, or a particular mode. All such constants begin with uppercase letters GL and the underscore (_)is used as a separator between all component words in the name.

Ex:

GL_LINES, GL_POLYGON, GL_AMBIENT_AND_DIFFUSE

The OpenGL functions also expect specific data types. For example, an OenGL 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

Each data-type name begins with the capital letters GL and the remainder of the name is a standard data-type designation, written in lower-case letters.

Related Libraries:

In addition to the OpenGL basic (core) library, there are a number of associated libraries for handling special operations. The **OpenGL Utility (GLU)** 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 surface rendering operations and other complex tasks.

Every OpenGL implementation includes the GLU library and all GLU function names start with the prefix **glu**.

To create a graphics display using OpenGL, we first need to set up a **display window** on our video screen. This is the rectangular area of the screen in which our picture will be displayed. We cannot create the display window directly with the basic OpenGL functions, since this library contains only device –independent graphics functions, and window-management operations depend on the computer we are using.

There are several window-system libraries that support OpenGL functions for a variety for a variety of machine.

- The OpenGL extension to the **X Window System (GLX)** provides a set of routines that are prefixed with the letters **glx**.
- Apple systems can use the **Apple GL (AGL)** interface for window management operations. Function names for this library are prefixed **agl**.
- For Microsoft Windows systems, the **WGL** routines provide a **Windows-to-OpenGL** interface. These routines are prefixed with the letters **wgl**.
- The **Presentation Manager to OpenGL (PGL)** is an interface for the IBM OS/2., which uses the prefix **pgl** for the library routines.
- An **OpenGL Utility Toolkit (GLUT)** provides library of functions for interacting with any screen-windowing system. The GLUT library functions are prefixed with glut, and this library also contain methods for describing and rendering quadric curves and surfaces. We can use GLUT so that our programs will be device independent.

Header Files:

In all of our programs we will need to include the header file for the OpenGL core library. For most applications we also need GLU. And we need to include the header file for the window system. For instance, with Microsoft Windows, the header file that accesses the WGL routines is **windows.h**. This header file must be listed before the OpenGL and GLU header files because it contains macros needed by the Microsoft Windows version of the OpenGL libraries. So the source file in this case should begin with

```
#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>
```

In addition, we will often need to include header files that are required by the C++ code.

Ex:

```
#include<stdio.h>
#include<stdlib.h>
```

#include<math.h>

Display-Window Management Using GLUT:

Since we using the OpenGL Utility Toolkit, our first step is to initialize GLUT. This initialization function could also process any command line arguments. We perform the GLUT initialization with the statement:

glutInit(&argc,argv);

Next, we can state that a display window is to be created on the screen with a given option 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 we want to use for the display-window title.

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

We need one more GLUT function to complete the window-processing operations. After execution of the following statement, all display windows that we have created, including their graphic content, are now activated.

glutMainLoop();

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

Although the display window that we created will be in some default location and size, we can set these parameters using additional GLUT functions. We use the **glutInitWindowPosition** function to give an initial location for the top-left corner of the display window. The position is specified in integer screen coordinates, whose origin is at the upper-left corner of the screen. For instance the following statement specifies that the top-left corner on the display window should be placed 50 pixels to the right of the left edge on the screen and 100 pixels down from the top edge of the screen.

glutInitWindowPosition(50,100);

Similarly, the **glutInitWindowSize** function is used to set the initial pixel width and height of the display window. Thus, we specify a display window with an initial width of 400 pixels and a height of 300 pixels with the statement

glutInitWindowSize(400,300);

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

We can also set a number of other options for the display window, such as buffering a choice of color modes, with the **glutInitDisplayMode** function. Arguments for this routine are assigned symbolic GLUT constants. For example, the following command specifies that a single refresh buffer is to be used for the display window and that the RGB (red, green, blue) color mode is to be used for selecting color values.

glutInitDisplayMode(GLUT_SINGE | GLUT_RGB);

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

A complete OpenGL Program

```
#include <GL/gl.h>
#include <GL/glut.h>   //Or others , depending on the system we use

void init(void)
{
  glClearColor(1.0,1.0,1.0,0.0);     //Set display-window color to white.
  glMatrixMode(GL_PROJECTION);      //Set projection parameters
  gluOrtho2D(0.0,200.0,0.0,200.0);
}

void lineSegment(void)
{
  glClear(GL_COLOR_BUFFER_BIT);      //Clear display window
  glColor3f(1.0,0.0,0.0);           //Set line segment color to red
  glBegin(GL_LINES);
    glVertex2i(180,15);            //Specify line segment geometry
    glVertex2i(10,145);
  glEnd();
  glFlush();     //Process all OpenGL routines as quickly as possible
}

int 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
}
```

- To set the background color for the display window to white, we use the following OpenGL function

glClearColor(1.0,1.0,1.0,0.0);

  - The first three arguments in this function set each of the red, green and blue component colors to the value 1.0. Thus we get a white 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.
  - If each of the red, green and blue components were set to the same intermediate value between 0.0 and 1.0, we would get some shade of gray.
  - The fourth parameter in the **glClearColor** function is called the alpha value for the specified color, which is used 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 indicates a totally transparent object, and an alpha value of 1.0 indicates an opaque object.
- 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 that are tobe set to the values indicated in the glClearColor function.
- To choose a color scheme for the objets we want to display in a scene, we use the following function

glColor3f(1.0,1.0,1.0);

  - The suffix 3f on the glColor function indicates that we are specifying the three RGB color components using floating-point(f) values. These values must be in the range from 0.0 to 1.0, and we have set red = 1.0 and green and blue = 0.0
- To display a two-dimensional line segment, we need to tell OpenGL how we want to "project" our picture onto the display window., because two dimensional picture is

treated by OpenGL as a special case of three-dimensional viewing.We can set the projection type (mode) and other viewing parameters that we need with the following two functions

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

- This function specifies that an orthogonal projection is used to map the contents of a two-dimensional (2D) rectangular area of world coordinates to the screen, and the x-coordinate values within this rectangle range from 0.0 to 200.0, with y-coordinate values ranging from 1.0 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 specified above 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.

- The following code defines a two-dimensional, straight-line segment with integer, Cartesian endpoints (180,15) and (10,145)

```
glBegin(GL_LINES);
    glVertex2i(180,15);          //Specify line segment geometry
    glVertex2i(10,145);
glEnd();
```

- To formce the xecution of our OpenGL functions, which are stored by computer systemsin bufferes in different locations, we use the following function

glFlush();

- We generally place all initializations and related one-time parameter settings in procedure **init**.
- The geometric description of the "picture" we want to display is in the ***displaycallback function*** named **lineSegment,** which is the procedure that will be referenced by the GLUT function glutDisplayFunc.
- The main procedure contains the GLUT functions for setting up the display window and getting our line segment onto the screen.

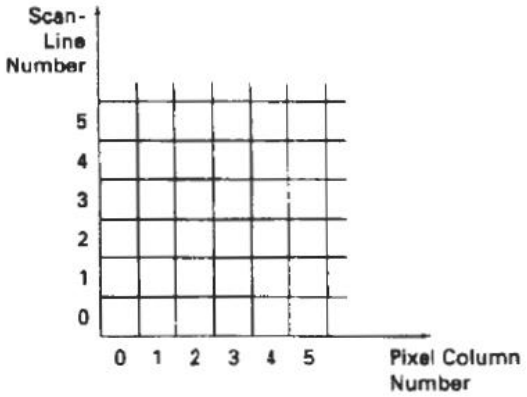
Coordinate Reference frames:

- To describe a picture , we decide upon a convineint Cartesian coordinate sytem, called the world coordinate reference frame, which could either be two dimensional or three dimensional.

- We then describe the objects in our picture by giving their geometric specifications in terms of positions in world coordinates. Ex: We define a straight-line segment with two end point 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, which are 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. For two dimensional figure, the coordinate extents are sometimes called an object's **bounding rectangle.**
- Objects are then displayed by passing the scene information to the viewing routines, which identify the visible surfaces and ultimately map the objects to positions on the video monitor.
- The scan-conversion process stores information about the scene, such as color values, at the appropriate location in the frame buffer, and the objects in the scene.

Screen Coordinates:

- Locations on a video monitor are referenced in integer screen coordinates, which correspond to the pixel positions in the frame buffer.
- Pixel coordinate value give the ***scan line number*** (the y value) and the ***column number*** (the x value along a scan line).
- Hardware processes, such as screen refreshing, typically address pixel positions with respect to the top-left corner of the screen.
- Scan lines are then referenced from 0, at the top of the screen to some integer value, y_{max}, at the bottom of the screen, and pixel positions along each scan line are numbered from 0 to $x_{max,}$ left to right.
- However, with software commands we can set up any convenient reference frame for screen positions. For example, we could specify an integer range for screen positions with the coordinate origin at the lower-left of a screen area



Scan-line algorithms for the graphics primitives use the defining coordinate descriptions to determine the locations of pixels that are to be displayed. Once the pixel positions have been identified for an object, the appropriate color values must be stored in the frame buffer. For this purpose, we will assume that we have a low level procedure of the form

setPixel(x,y);

This procedure stores the current color setting into the frame buffer at integer position (x,y), relative to the selected position of the screen –coordinate origin.

We sometimes also will want to be able to retrieve the current frame-buffer setting for a pixel location. So we will assume that we have the following low-level function for obtaining a frame-buffer color value.

getPixel(x,y,color);

In this function, parameter color receives an integer value corresponding to the combined RGB bit codes stored for the specified pixel at position (x,y). For three dimensional scenes we need to specify one additional screen coordinate vlue that references the depth of object positions relative to the viewing position.

Absolute and Relative Coordinate Specifications:

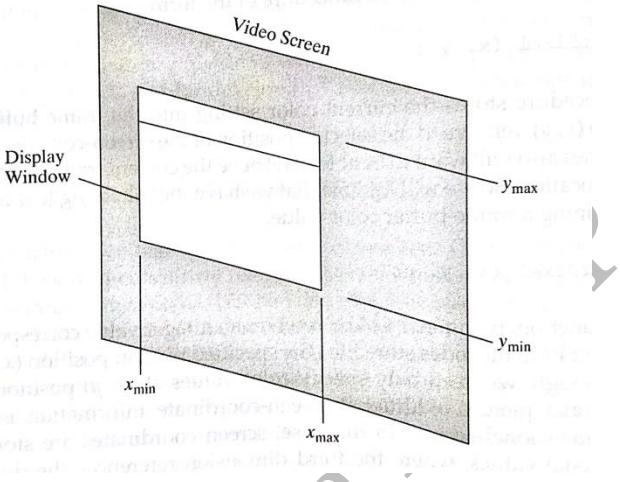
Absolute Coordinates values: The coordinate values specified are the actual positions within the coordinate system in use.

Relative Coordinates values: The coordinate position is specified as an offset from the last position that was referenced (called the **current position**).

For example, if location (3,8) is the last position that has been referenced in an application program, a relative coordinate specification of (2,-1) corresponds to an absolute position of (5,7).An additional function is then used to set a current position before any coordinates for primitive functions are specified .To describe an object, such as a series of connected line segments, we then need to give only a sequence of relative coordinates (offsets), once a starting position has been established.

Specifying a Two-Dimensional world-coordinate reference frame in OpenGL:

- The OpenGL function **gluOrtho2D** is used to set up any two-dimensional Cartesian reference frame.
- The arguments for this function are the four values defining x and y coordinate limits for the picture we want to display.
- Since gluOrtho2D function specifies an orthogonal projection, we need to make 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.
- For two dimensional programs 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 the above figure.

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 the 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.

OpenGL Point Functions:

We use the following OpenGL function to state the coordinate values for a single position:

glVertex* ();

where the asterisk(*) indicates that suffix codes are required for tis function.

A **glVertex** function must be placed between a **glBegin** function and a **glEnd** function. The argument of the **glBegin** function is used to identify the kind of output primitive that is to be displayed, and glEnd takes no arguments.

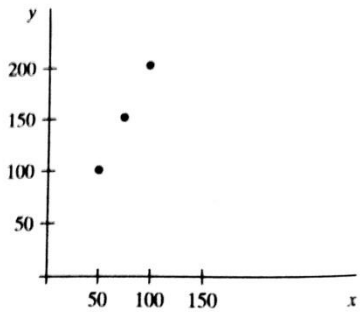
For point plotting. The argument of the glBegin function is the symbolic constant GL_POINTS. Thus, the form of an OpenGL specification of a point position is

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

The **glVertex** function is used to specify coordinates for any point position. In this way, a single function is used for point, line , and polygon specifications.
Coordinate positions in OpenGL can be given in two, three or four dimensions. We use the suffix value of 2,3, or 4 on the **glVertex** function to indicate the dimensionality of a coordinate position.
We need to state also which data type is to be used for the numerical value specifications of the coordinates. This is accomplished within a second suffix code on the **glVertex** function. Suffix codes for specifying a numerical data type can be listed explicitly in the **glVertex** function, or a single argument can be used that references a coordinate position as an array. If we use an array specification for a coordinate position, we need to append a third suffix code: v (for "vector")

The default color for primitives is white and the default point size is equal to the size of one screen pixel.
In the following example, three equally spaced points are plotted along a two-dimensional straight-line path with a slope of 2 as shown in the figure. The coordinates are given as a pair of integers.

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



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

```
int point1[ ] = {50,100};
int point2[ ] = {75,150};
int point3[ ] = {100,200};
```

and call the OpenGL functions for plotting the three points as

```
glBegin(GL_POINTS);
    glVertex2i(point1);
    glVertex2i(point2);
    glVertex2i(point3);
glEnd();
```

The following is an example of specifying two point positions in a three-dimensional world reference frame. In this case, we give the coordinates as explicit floating-point values.

```
glBegin(GL_POINTS);
    glVertex3f(-78.05, 909.72, 14.60);
    glVertex2i(261.91, -5200.67, 188.33);
glEnd();
```

We could also define a C++ class or structure(struct) for specifying point positions in various dimensions. For example,

```
class wcPt2D {
public:
    GLfloat x,y;
};
```

Using the above class definition, we could specify a two-dimensional, world-coordinate point position with the statements

```
wcPt2D pointPos;

pointPos.x = 120.75;
pointPos.y =  45.30;
glBegin (GL_POINTS);
    glVertex2f(pointPos.x, pointPos.y);
glEnd();
```

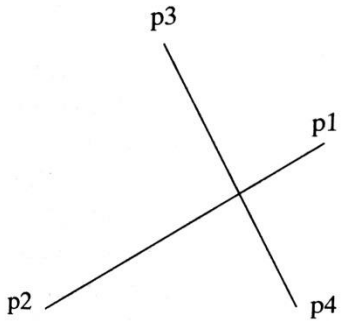
OpenGL Line Functions:

There are three symbolic constants in OpenGL that we can use to specify how a list of endpoint positions should be connected to form a set of straight-line segments. By default, each symbolic constant displays solid, white lines.

A set of straight-line segments between each successive pair of endpoints in a list is generated using the primitive line constant **GL_LINES**. In general, this will result in a set of unconnected lines unless some coordinate positions are repeated. Nothing is displayed if only

one endpoint is specified, and the last endpoint is not processed if the number of endpoints listed is odd. For example, if we five coordinate positions, labeled p1 through p5, and each is represented as a two-dimensional array, then the following code could generate the display as shown in the following figure.

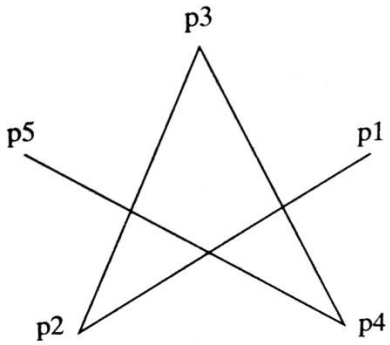
```
glBegin(GL_LINES);
    glVertex2iv (p1);
    glVertex2iv (p2);
    glVertex2iv (p3);
    glVertex2iv (p4);
    glVertex2iv (p5);
glEnd();
```



Thus, we obtain one line segment between the first and the second coordinate positions, and another line segment between the third and fourth positions. In the above case, the number of specified endpoints is odd, so the last coordinate position is ignored.

We can use the OpenGL primitive constant **GL_LINE_STRIP**, to obtain a **polyline** as shown in the following figure.

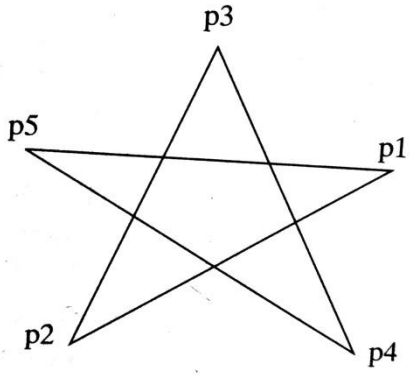
```
glBegin(GL_LINE_STRIP);
    glVertex2iv (p1);
    glVertex2iv (p2);
    glVertex2iv (p3);
    glVertex2iv (p4);
    glVertex2iv (p5);
glEnd();
```



In the above case, the display is a sequence of connected line segments between the first endpoint in the list and last endpoint. The first line segment in the polygon is displayed between the first endpoint and the second endpoint; the second line segment is between the second and third endpoints; and so forth, up to the last line endpoint. Nothing is displayed if we do not list at least two coordinate positions.

The OpenGL primitive constant **GL_LINE_LOOP** is used to obtain a **closed polyline** as shown in the following figure.

```
glBegin(GL_LINE_LOOP);
    glVertex2iv (p1);
```

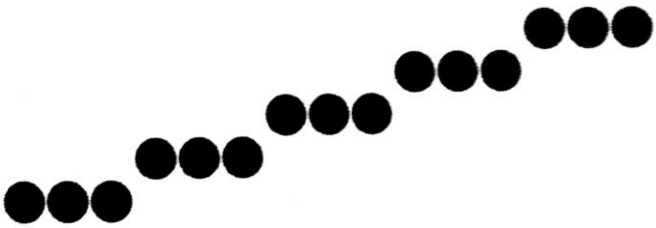


```
    glVertex2iv (p2);
    glVertex2iv (p3);
    glVertex2iv (p4);
    glVertex2iv (p5);
glEnd();
```

In the above case, an additional line is added to the line sequence from the previous example, so that the last coordinate endpoint in the sequence is connected to the first coordinate endpoint of the polyline.

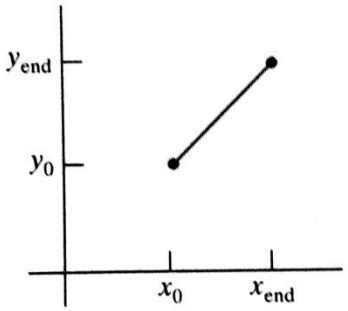
Line Drawing Algorithms:

- A straight line-segment in a scene is defined by the 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.
- Reading from the frame buffer, the video controller plots the screen pixels. This process digitizes the line into a set of discrete integer positions that, in general only approximates the actual line path.
- A computed line position of (10.48,20.51), for example, is converted to pixel, position (10,21). This rounding of coordinate values to integers causes all but horizontal and vertical lines to be displayed with a stair-step appearance ("the jaggies") as shown in the following figure.



Line Equations:

We determine pixel positions along a straight-line path from the geometric properties of the lien. The Cartesian slope-intercept equation for a straight line is



$$y = m \cdot x + b \qquad (1)$$

with m as the slope of the line and b as the y intercept.

Given that the two endpoints of a line segment are specified at positions (x_0,y_0) and (x_{end},y_{end}), as shown in the figure, we can determine values for the slope m and y intercept b with following calculations:

$$m = \frac{y_{end} - y_0}{x_{end} - x_0} \quad (2)$$

$$b = y_0 - m \cdot x_0 \quad (3)$$

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

$$\delta y = m \cdot \delta x \quad (4)$$

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

$$\delta x = \frac{\delta y}{m} \quad (5)$$

DDA Algorithm:

The *digital differential analyzer* (DDA) is a scan-conversion line algorithm based on calculating either **δy** or **δx,** using the above equations (4) and (5)

This equation is used to find all the intermediate pixels between the starting point and end point,

Suppose we want to find all the intermediate line between two endpoint (x_k,y_k) and (x_{k+1},y_{k+1})

So, Equation (2) can be rewritten as

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

They are three cases that needs to be considered.

Case 1: |m| < 1

If the slope of the line is positive and less than or equal to 1,then x will change in unit intervals (**δx = 1**)

$$x_{k+1} = x_k + 1 \quad (6)$$

and successive values of y can be computed as ,

$$y_{k+1} = y_k + m \quad (7)$$

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

value must be rounded to the nearest integer corresponding to a screen pixel position in the x column we are processing.

Case 2: |m| < 1

If the slope of the line is greater than 1,then y will change in unit intervals (**δy = 1**)

$$y_{k+1} = y_k + 1 \qquad (8)$$

and successive values of x can be computed as ,

$$x_{k+1} = x_k + (1/m) \qquad (9)$$

In this case, each computed x value is rounded to then nearest pixel position along the current y scan line.

The above equations are based on the assumptions that lines are to be processed from left endpoint to the right endpoint. If this processing is reversed, so that the staring endpoint is at the right, then either we have **δx = -1** and

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

or (when the slope is greater than 1) we have **δy = -1** with

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

Similar calculations are carried out using above equations to determine pixel positions along a line with negative slope.

The DDA algorithm is a faster method for calculating pixel positions than the direct use of Eq. (1). It eliminates the multiplication in Eq. (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 round-off error in successive additions of the floating-point increment, however, can cause the calculated pixel positions to drift away from the true line path for long line segments. Furthermore, **the rounding operations and floating-point arithmetic in procedure line DDA 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 calculations are reduced to integer operations.

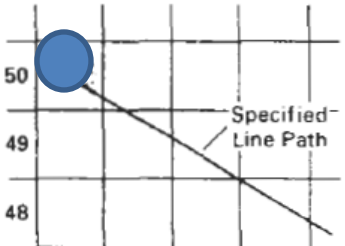
```
#include <stdlib.h>
#include <math.h>

inline int round (const float a)  { return int (a + 0.5); }

void lineDDA (int x0, int y0, int xEnd, int yEnd)
{
   int dx = xEnd - x0,  dy = yEnd - y0,  steps,  k;
   float xIncrement, yIncrement, x = x0, y = y0;

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

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

This algorithm is summarized in the above procedure, which accepts as input the two endpoint 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 (x0, y0), 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 x0 is less than xEnd, 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 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 l / m .

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, 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.

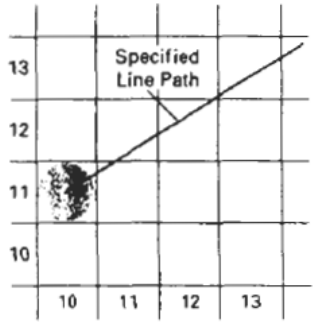


Figure 3-5
Section of a display screen where a straight line segment is to be plotted, starting from the pixel at column 10 on scan line 11.

To illustrate Bresenham's approach, we first consider the scan-conversion process for lines with positive slope less than 1.0. Pixel positions along a line path are then determined by sampling at unit x intervals. Starting from the left endpoint (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}= x_k + 1.Our choices are the pixels at positions (x_k+l, y_k).and (x_k+l, y_k+l).

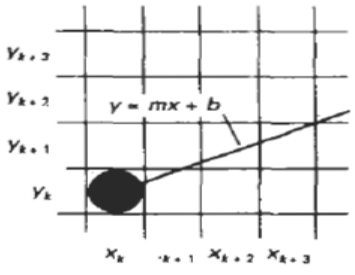


Figure 3-7
Section of the screen grid showing a pixel in column x_k on scan line y_k that is to be plotted along the path of a line segment with slope $0 < m < 1$.

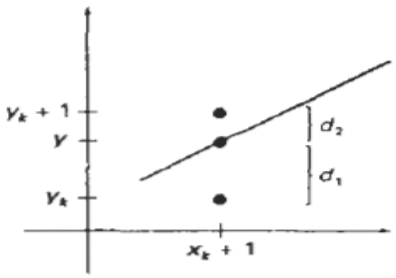


Figure 3-8
Distances between pixel positions and the line *y* coordinate at sampling position $x_k + 1$.

At sampling position x_k+l, 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+l is calculated as

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

Then

$$\begin{aligned} d_1 &= y - y_k \\ &= m(x_k + 1) + b - y_k \end{aligned}$$

and

$$\begin{aligned} d_2 &= (y_k + 1) - y \\ &= y_k + 1 - m(x_k + 1) - b \end{aligned}$$

The difference between these two separations is

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

A c

3-11 so that it involves only integer calculations. We accomplish this by substituting m = Δy/Δx, where Δy and Δx are the vertical and horizontal separations of the endpoint positions, and defining:

$$\begin{aligned} p_k &= \Delta x(d_1 - d_2) \\ &= 2\Delta y \cdot x_k - 2\Delta x \cdot y_k + c \end{aligned} \tag{3-12}$$

The sign of $\boldsymbol{p_k}$, is the same as the sign of d1 – d2, since Δx > 0 for our example. Parameter $\boldsymbol{c}$ is constant and has the value 2Δy + Δy(2b - l), which is independent of pixel position and will be eliminated in the recursive calculations for $\boldsymbol{p_k}$. If the pixel at $\boldsymbol{y_k}$ is closer to the line path than the pixel at $\boldsymbol{y_k}$ +l (that is, dl < d2), then decision parameter $\boldsymbol{p_k}$ 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

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

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

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

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_0, 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.

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.

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

$$\Delta x = 10, \qquad \Delta y = 8$$

The initial decision parameter has the value

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

and the increments for calculating successive decision parameters are

$$2\Delta y = 16, \qquad 2\Delta y - 2\Delta x = -4$$

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

k	p_k	(x_{k+1}, y_{k+1})	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.

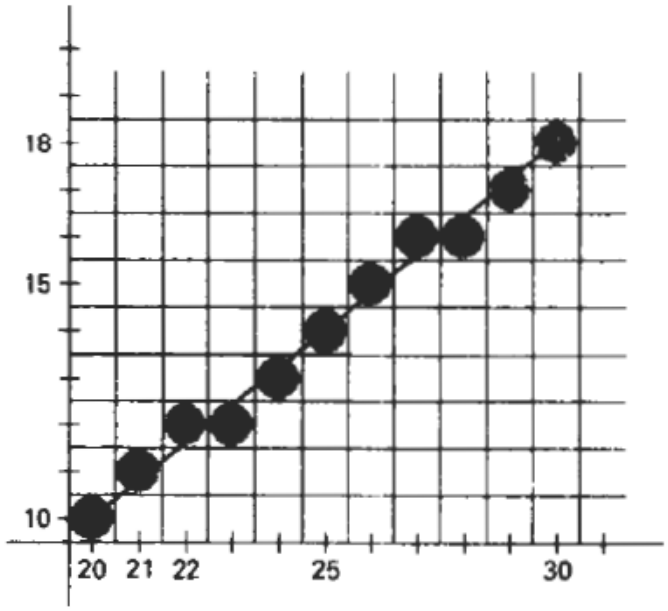


Figure 3-9
Pixel positions along the line path between endpoints (20, 10) and (30, 18), plotted with Bresenham's line algorithm.

An implementation of Bresenham line drawing for slopes in the range $0 < rn < 1$ 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.

```
/Bresenham line-drawing procedure for |m| < 1.0
void lineBres(GLint x0, GLint y0, GLint xEnd, GLint yEnd)
{
        GLint dx = fabs(xEnd - x0);
        GLint dy = fabs(yEnd - y0);
        GLint p = 2 * dy - dx;
        GLint twoDy = 2 * dy;
        GLint twoDyMinusDx = 2 * (dy-dx);
        GLint x,y;
        // determine which endpoint to use as start position
        if (x0 > xEnd){
                x = xEnd;
                y = yEnd;
                xEnd = x;
              }
              else{
                        x = x0;
                        y = y0;
                    }
        setPixel(x,y);
        while(x<xEnd){
```

```
        x++;
        if(p<0)
                p += twoDy;
        else {
           y++;
           p += twoDyMinusDx;
        }
        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 intelrhange 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 coordinate decreases as the other increases. Finally, special cases can be handled separately: Horizontal lines ($\Delta y = 0$), vertical lines ($\Delta x = 0$), and diagonal lines with ($|\Delta y| = |\Delta x|$) each can be loaded directly into the frame buffer without processing them through the line-plotting algorithm.

OpenGL Curve Functions:

- Routines for generating basic curves, such as circles and ellipses, are not included as primitive functions in the OpenGL, core library. But this library does contain functions for displaying Beizier splines, which are polynomials that are defined with a discrete point set.
- Another method we can use to generate a display of a simple curve is to approximate it using a polyline. We just need to locate a set of points along the curve path and connect the points with straight-line segments. The more line sections we include in a polyline, the smoother the appearance of the curve.
- A third alternative is to write our own curve-generation functions based on ceratin algorithms.

CIRCLE-GENERATING ALGORITHMS:

Properties of Circles

A circle is defined as the set of points that are all at a given distance r from a center position (x_c, y_c) (Fig. 3-12). This distance relationship is expressed by the Pythagorean theorem in Cartesian coordinates as

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

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

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

But this is not the best method for generating a circle. One problem with this approach is that it involves considerable computation at each step. Moreover, the spacing between plotted pixel positions is not uniform, as demonstrated in Fig. 3-13.

Another way to eliminate the unequal spacing shown in Fig. 3-13 is to calculate points along the circular boundary using polar coordinates r and θ (Fig. 3-12). Expressing the circle equation in parametric polar form yields the pair of equations

$$x = x_c + r\cos\theta$$

$$y = y_c + r\sin\theta$$

When a display is generated with these equations using a fixed angular step size, a circle is plotted with equally spaced points along the circumference. The step size chosen for θ depends on the application and the display device. Larger angular separations along the circumference can be connected with straight line segments to approximate the circular path. For a more continuous boundary on a raster display, we can set the step size at l/r. This plots pixel positions that are approximately one unit apart.

Computation can be reduced by considering the symmetry of circles. The shape of the circle is similar in each quadrant. We can generate the circle section in the second quadrant of the xy

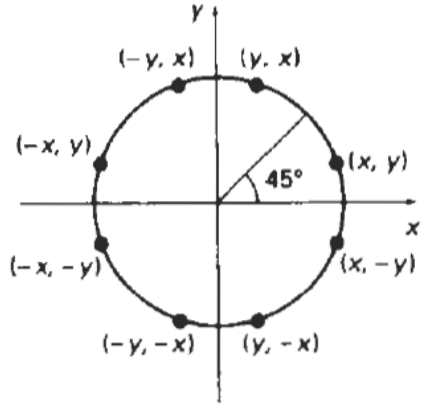


Figure 3-14
Symmetry of a circle. Calculation of a circle point (x, y) in one octant yields the circle points shown for the other seven octants.

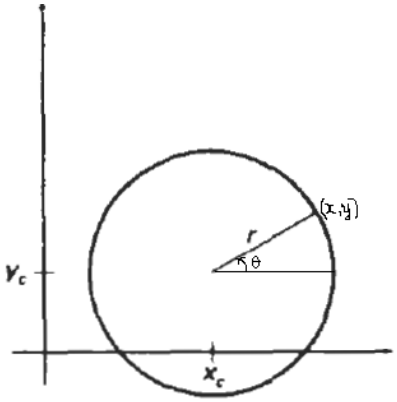


Figure 3-12
Circle with center coordinates (x_c, y_c) and radius r.

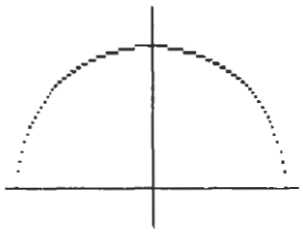


Figure 3-13
Positive half of a circle plotted with Eq. 3-25 and with $(x_c, y_c) = (0, 0)$.

plane by noting that the two circle sections are symmetric with respect to the y axis. And circle sections in the third and fourth quadrants can be obtained from sections in the first and second quadrants by considering symmetry about the x axis. We can take this one step further and note that there is also symmetry between octants. Circle sections in adjacent octants within one quadrant are symmetric with respect to the 45' line dividing the two octants. These symmehy conditions are illustrated in Fig.3-14, where a point at position (x, y) on a one-eighth circle sector is mapped into the seven circle points in the other octants of the xy plane. Taking advantage of the circle symmetry in this way we can generate all pixel positions around a circle by calculating only the points within the sector from x = 0 to x = y.

Bresenham's line algorithm for raster displays is adapted to circle generation by setting up decision parameters for finding the closest pixel to the circumference at each sampling step.

A method for direct distance comparison is to test the halfway position between two pixels to determine if this midpoint is inside or outside the circle boundary. This method is more easily applied to other conics; and for an integer circle radius, the midpoint approach generates the same pixel positions as the Bresenham circle algorithm.

Midpoint Circle Algorithm:

For a given radius r and screen center position (x , y,), we can first set up our algorithm to calculate pixel positions around a circle path centered at the coordinate origin (0,0). Then each calculated position (x, y) is moved to its proper screen position by adding x_c to x and y_c to y. Along the circle section from x = 0 to x = y in the first quadrant, the slope of the curve varies from 0 to -1. Therefore, we can take unit steps in the positive x direction over this octant and use a decision parameter to determine which of the two possible y positions is closer to the circle path at each step. Positions in the other seven octants are then obtained by symmetry.

To apply the midpoint method, we define a circle function:

$$f_{circle}(x, y) = x^2 + y^2 - r^2 \qquad (3\text{-}27)$$

Any point (x , y) on the boundary of the circle with radius r satisfies the equation $f_{circle}(x, y) = 0$. If the point is in the interior of the circle, the circle function is negative. And if the point is outside the circle, the circle function is positive. To summarize, the relative position of any point (x,y) can be determined by checking the sign of the circle function:

$$f_{circle}(x, y) \begin{cases} < 0, & \text{if } (x, y) \text{ is inside the circle boundary} \\ = 0, & \text{if } (x, y) \text{ is on the circle boundary} \\ > 0, & \text{if } (x, y) \text{ is outside the circle boundary} \end{cases} \qquad (3\text{-}28)$$

The circle-function tests in 3-28 are performed for the midpositions between pixels near the circle path at each sampling step. Thus, the circle function is the decision parameter in the midpoint algorithm, and we can set up incremental calculations for this function as we did in the line algorithm.

Figure 3-15 shows the midpoint between the two candidate pixels at Sampling position $x_k + 1$. Assuming we have just plotted the pixel at (x_k, y_k), we next need to determine whether the pixel at position $(x_k + 1, y_k)$ or the one at position $(x_k + 1, y_k - 1)$ is closer to the circle. Our decision parameter is the circle function 3-27 evaluated at the midpoint between these two pixels:

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

$$= (x_k + 1)^2 + \left(y_k - \frac{1}{2}\right)^2 - r^2 \qquad (3\text{-}29)$$

If $p_k < 0$, this midpoint is inside the circle and the pixel on scan line y_k is closer to the circle boundary. Otherwise, the midposition is outside or on the circle boundary, and we select the pixel on scanline $y_k - 1$.

Successive decision parameters are obtained using incremental calculations. We obtain a recursive expression for the next decision parameter by evaluating the circle function at sampling position $x_{k+1} + 1 = x_k + 2$:

$$p_{k+1} = f_{\text{circle}}\left(x_{k+1} + 1, y_{k+1} - \frac{1}{2}\right)$$

$$= [(x_k + 1) + 1]^2 + \left(y_{k+1} - \frac{1}{2}\right)^2 - r^2$$

or

$$p_{k+1} = p_k + 2(x_k + 1) + (y_{k+1}^2 - y_k^2) - (y_{k+1} - y_k) + 1 \qquad (3\text{-}30)$$

Increments for obtaining p_{k+1}, are either $2p_{k+1} + 1$ (if p_k is negative) or $2x_{k+1} + 1 - 2y_{k+1}$ Evaluation of the terms $2x_{k+1}$ and $2y_{k+1}$ can also be done incrementally as

$$2x_{k+1} = 2x_k + 2$$

$$2y_{k+1} = 2y_k - 2$$

At the start position (0, r), these two terms have the values 0 and 2r, respectively. Each successive value is obtained by adding 2 to the previous value of 2x and subtracting 2 from the previous value of 2y.

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

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

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

or

$$p_0 = \frac{5}{4} - r \qquad (3\text{-}31)$$

If the radius r is specified as an integer, we can simply round p_0 to

$$p_0 = 1 - r \qquad \text{(for } r \text{ an integer)}$$

since all increments are integers.

As in Bresenham's line algorithm, the midpoint method calculates pixel positions along the circumference of a cirde using integer additions and subtractions, assuming that the circle parameters are specified in integer screen coordinates. We can summarize the steps in the midpoint circle algorithm as follows.

Midpoint Circle Algorithm

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

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

2. Calculate the initial value of the decision parameter as

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

3. At each x_k position, starting at $k = 0$, perform the following test: If $p_k < 0$, the next point along the circle centered on (0, 0) is (x_{k+1}, y_k) and

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

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

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

where $2x_{k+1} = 2x_k + 2$ and $2y_{k+1} = 2y_k - 2$.

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

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

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

Example 3-2 Midpoint Circle-Drawing

Given a circle radius $r = 10$, we demonstrate the midpoint circle algorithm by determining positions along the circle octant in the first quadrant from $x = 0$ to $x = y$. The initial value of the decision parameter is

$$p_0 = 1 - r = -9$$

For the circle centered on the coordinate origin, the initial point is $(x_0, y_0) = (0, 10)$, and initial increment terms for calculating the decision parameters are

$$2x_0 = 0, \qquad 2y_0 = 20$$

Successive decision parameter values and positions along the circle path are calculated using the midpoint method as

k	p_k	(x_{k+1}, y_{k+1})	$2x_{k+1}$	$2y_{k+1}$
0	−9	(1, 10)	2	20
1	−6	(2, 10)	4	20
2	−1	(3, 10)	6	20
3	6	(4, 9)	8	18
4	−3	(5, 9)	10	18
5	8	(6, 8)	12	16
6	5	(7, 7)	14	14

A plot of the generated pixel positions in the first quadrant is shown in Fig. 3-16.

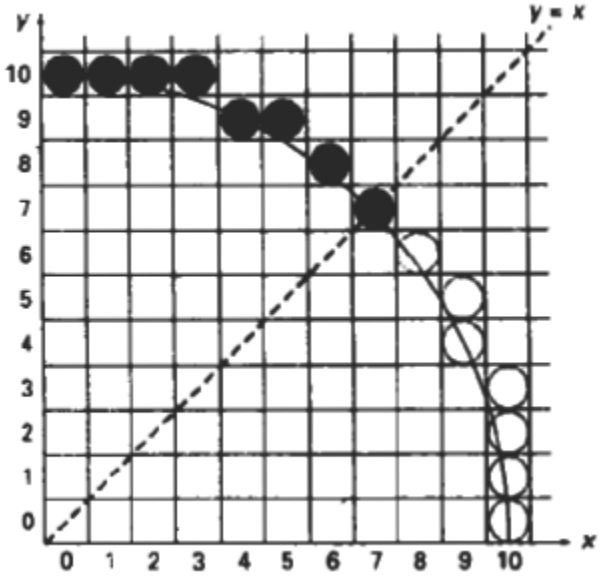


Figure 3-16
Selected pixel positions (solid circles) along a circle path with radius $r = 10$ centered on the origin, using the midpoint circle algorithm. Open circles show the symmetry positions in the first quadrant.

The following code segment illustrates procedures that could be used to implement the midpoint circle algorithm:

```
#include <stdio.h>
#include <math.h>
#include <GL/gl.h>
#include <GL/glut.h>

// Center of the cicle = (320, 240)
int xc = 320, yc = 240;

// Plot eight points using circle's symmetrical property
void plot_point(int x, int y)
{
	glBegin(GL_POINTS);
	glVertex2i(xc+x, yc+y);
	glVertex2i(xc+x, yc-y);
	glVertex2i(xc+y, yc+x);
	glVertex2i(xc+y, yc-x);
	glVertex2i(xc-x, yc-y);
	glVertex2i(xc-y, yc-x);
	glVertex2i(xc-x, yc+y);
	glVertex2i(xc-y, yc+x);
	glEnd();
}

// Function to draw a circle using bresenham's
// circle drawing algorithm
void bresenham_circle(int r)
{
	int x=0,y=r;
	float pk=(5.0/4.0)-r;

	/* Plot the points */
	/* Plot the first point */
	plot_point(x,y);
	int k;
	/* Find all vertices till x=y */
	while(x < y)
	{
		x = x + 1;
		if(pk < 0)
		  pk = pk + 2*x+1;
		else
```

```
            {
                y = y - 1;
                pk = pk + 2*(x - y) + 1;
            }
            plot_point(x,y);
        }
        glFlush();
}

// Function to draw two concentric circles
void concentric_circles(void)
{
        /* Clears buffers to preset values */
        glClear(GL_COLOR_BUFFER_BIT);

        int radius1 = 100, radius2 = 200;
        //bresenham_circle(radius1);
        bresenham_circle(radius2);
}

void Init()
{
        /* Set clear color to white */
        glClearColor(1.0,1.0,1.0,0);
        /* Set fill color to black */
        glColor3f(0.0,0.0,0.0);
        gluOrtho2D(0 , 640 , 0 , 480);
}

int main(int argc, char **argv)
{
        /* Initialise GLUT library */
        glutInit(&argc,argv);
        /* Set the initial display mode */
        glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
        /* Set the initial window position and size */
        glutInitWindowPosition(0,0);
        glutInitWindowSize(640,480);
        /* Create the window with title "DDA_Line" */
        glutCreateWindow("bresenham_circle");
        /* Initialize drawing colors */
        Init();
        /* Call the displaying function */
        glutDisplayFunc(concentric_circles);
```

```
    /* Keep displaying untill the program is closed */
    glutMainLoop();
}
```