hearn-50265; ISBN: 0-13-015390-7 *book* *August 8, 2003* *14:44*

CHAPTER 2

Overview of Graphics Systems

A wide, curved-screen, computer-graphics presentation system and its control desk.
(Courtesy of Silicon Graphics, Inc. and Trimension Systems. © 2003 SGI. All rights reserved.)

The power and utility of computer graphics is widely recognized, and a broad range of graphics hardware and software systems are now available for applications in virtually all fields. Graphics capabilities for both two-dimensional and three-dimensional applications are now common, even on general-purpose computers and handheld calculators. With personal computers, we can use a variety of interactive input devices and graphics software packages. For higher-quality applications, we can choose from a number of sophisticated special-purpose graphics hardware systems and technologies. In this chapter, we explore the basic features of graphics hardware components and graphics software packages.

2-1 VIDEO DISPLAY DEVICES

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

FIGURE 2-1 A computer-graphics workstation. *(Courtesy of Silicon Graphics, Inc., Why Not Films, and 525 Post Production. © 2003 SGI. All rights reserved.)*

hearn-50265; ISBN: 0-13-015390-7 book July 29, 2003 14:54

Refresh Cathode-Ray Tubes

Figure 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 do this is to store the picture information as a charge distribution within the CRT. This charge distribution can then be used to keep the phosphors activated. However, the most common method now employed for maintaining phosphor glow is to redraw the picture repeatedly by quickly directing the electron beam back over the same screen points. This type of display is called a **refresh CRT,** and the frequency at which a picture is redrawn on the screen is referred to as the **refresh rate.**

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 inside of the CRT envelope near the phosphor screen, or an accelerating anode, as in Fig. 2-3, can be used to provide the positive voltage. Sometimes the electron gun is designed so that the accelerating anode and focusing system are within the same unit.

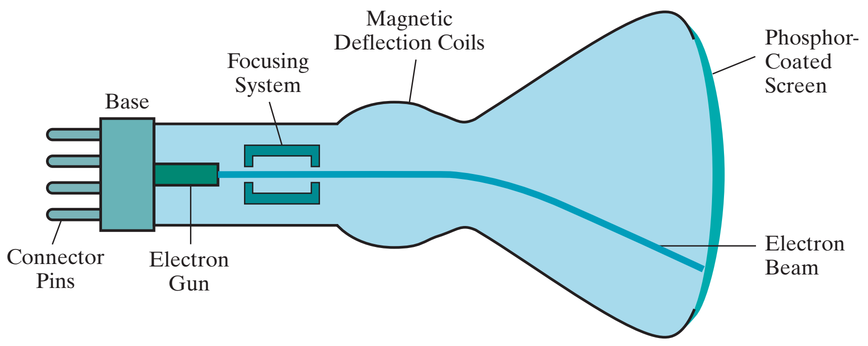

FIGURE 2-2 Basic design of a magnetic-deflection CRT.

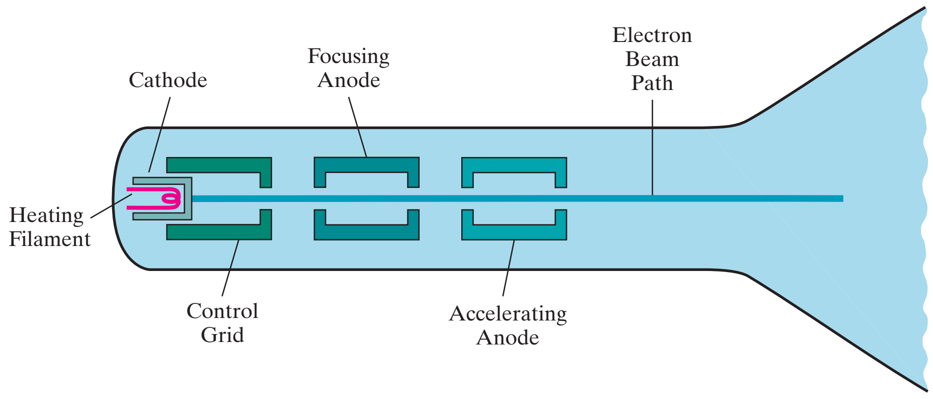

FIGURE 2-3 Operation of an electron gun with an accelerating anode.

hearn-50265; ISBN: 0-13-015390-7 *book* *July 29, 2003* *14:54*

Intensity of the electron beam is controlled by the voltage at 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, the brightness of a display point is controlled by varying the voltage on the control grid. This brightness, or intensity level, is specified for individual screen positions with graphics software commands, as discussed in Chapter 3.

The focusing system in a CRT forces the electron beam to converge to a small cross section 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. With electrostatic focusing, the electron beam is passed through a positively charged metal cylinder so that electrons along the centerline of the cylinder are in an equilibrium position. This arrangement forms an electrostatic lens, as shown in Fig. 2-3, and the electron beam is focused at the center of the screen in 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, and magnetic lens focusing usually 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 with either electric or 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 for this purpose. One pair is mounted on the top and bottom of the CRT neck, and the other pair is mounted on opposite sides of the neck. The magnetic field produced by each pair of coils results in a transverse deflection force that is perpendicular to both the direction of the magnetic field and the direction of travel of the electron beam. Horizontal deflection is accomplished with one pair of coils, and vertical deflection with 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 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 their kinetic energy is absorbed by the phosphor. Part of the beam energy is converted by 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 called photons. What we see on the screen is the combined effect of all

hearn-50265; ISBN: 0-13-015390-7 book July 29, 2003 14:54

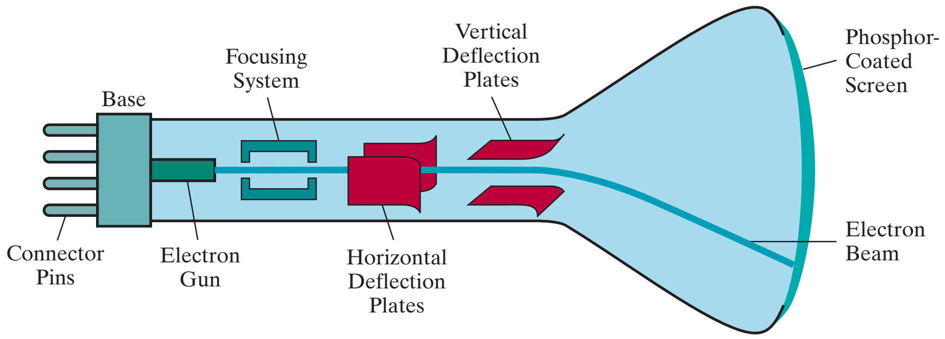

FIGURE 2-4 Electrostatic deflection of the electron beam in a CRT.

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 CRTs. Besides color, a major difference between phosphors is their **persistence:** how long they continue to emit light (that is, how long before all excited electrons have returned to the ground state) after the CRT beam is removed. Persistence is defined as the time that it takes the emitted light from the screen to decay to one-tenth of its original intensity. Lower-persistence phosphors require higher refresh rates to maintain a picture on the screen without flicker. A phosphor with low persistence can be useful for animation, while high-persistence phosphors are better suited for displaying highly complex, static pictures. Although some phosphors have persistence values greater than 1 second, general-purpose graphics monitors are usually constructed with persistence in the range from 10 to 60 microseconds.

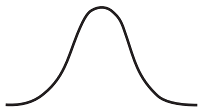

FIGURE 2-5 Intensity distribution of an illuminated phosphor spot on a CRT screen.

Figure 2-5 shows the intensity distribution of a spot on the screen. The intensity is greatest at the center of the spot, and it 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.

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 (Fig. 2-5), so two adjacent spots will appear distinct as long as their separation is greater than the diameter at which each spot has an intensity of about 60 percent of that at the center of the spot. This overlap position is illustrated in Fig. 2-6. Spot size also depends on intensity. As more electrons are accelerated toward the phosphor per second, the diameters of the CRT beam 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, on the other hand, 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 also depends on the capabilities of the system to which it is attached.

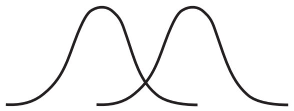

FIGURE 2-6 Two illuminated phosphor spots are distinguishable when their separation is greater than the diameter at which a spot intensity has fallen to 60 percent of maximum.

hearn-50265; ISBN: 0-13-015390-7 *book* *July 29, 2003* *14:54*

Raster-Scan Displays

The most common type of graphics monitor employing a CRT is the **raster-scan display,** based on television technology. 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 a **scan line.** As the electron beam moves across a scan line, the beam intensity is turned on and off (or set to some intermediate value) to create a pattern of illuminated spots. Picture definition is stored in a memory area called the **refresh buffer** or **frame buffer,** where the term **frame** refers to the total screen area. 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 scan line at a time, as demonstrated in Fig. 2-7. Each screen spot that can be illuminated by the electron beam is referred to as a **pixel** or **pel** (shortened forms of **picture element**). Since the refresh buffer is used to store the set of screen color values, it is also sometimes called a **color buffer.** Also, other kinds of pixel information, besides color, are stored in buffer locations, so all the different buffer areas are sometimes referred to collectively as the "frame buffer". The capability of a raster-scan system to store color 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. Another property of video monitors is **aspect ratio,** which is now often defined as the number of pixel columns divided by the number of scan lines that can be displayed by the system. (Sometimes the term aspect ratio is used to refer to the number of scan lines divided by the number of pixel columns.) Aspect ratio can also be described as the number of horizontal

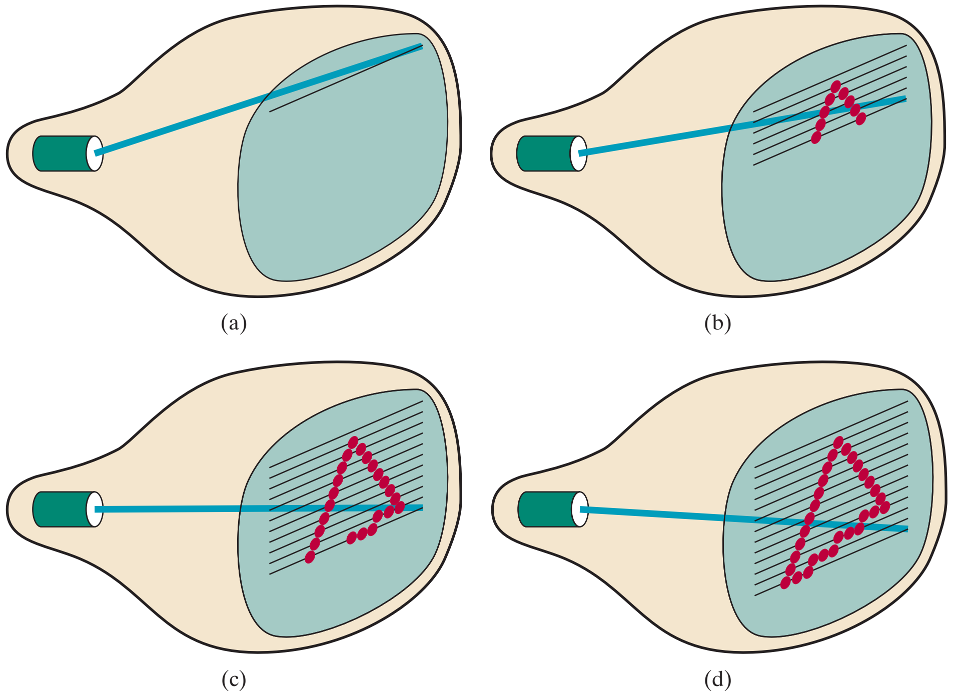

FIGURE 2-7 A raster-scan system displays an object as a set of discrete points across each scan line.

points to vertical points (or vice versa) necessary to produce equal-length lines in both directions on the screen. Thus, an aspect ratio of 4/3, for example, means that a horizontal line plotted with four points has the same length as a vertical line plotted with three points, where line length is measured in some physical units such as centimeters. Similarly, the aspect ratio of any rectangle (including the total screen area) can be defined to be the width of the rectangle divided by its height.

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. For 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, for example, indicates that the electron beam is to be turned on at that position, and a value of 0 turns the beam off. Additional bits allow the intensity of the electron beam to be varied over a range of values between "on" and "off". 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.** But the terms bitmap and pixmap are also used to describe other rectangular arrays, where a bitmap is any pattern of binary values and a pixmap is a multicolor pattern.

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. Below about 24 frames per second, we can usually perceive a gap between successive screen images, and the picture appears to flicker. Old silent films, for example, show this effect because they were photographed at a rate of 16 frames per second. When sound systems were developed in the 1920s, motion-picture film rates increased to 24 frames per second, which removed flickering and the accompanying jerky movements of the actors. Early raster-scan computer systems were designed with a refresh rate of about 30 frames per second. This produces reasonably good results, but picture quality is improved, up to a point, with higher refresh rates on a video monitor because the display technology on the monitor is basically different from that of film. A film projector can maintain the continuous display of a film frame until the next frame is brought into view. But on a video monitor, a phosphor spot begins to decay as soon as it is illuminated. Therefore, current raster-scan displays perform refreshing at the rate of 60 to 80 frames per second, although some systems now have refresh rates of up to 120 frames per second. And some graphics systems have been designed with a variable refresh rate. For example, a higher refresh rate could be selected for a stereoscopic application so that two views of a scene (one from each eye position) can be alternately displayed without flicker. But other methods, such as multiple frame buffers, are typically used for such applications.

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 $\frac{1}{80}$ to $\frac{1}{60}$ of a second), the electron beam returns to the top left corner of the screen (**vertical retrace**) to begin the next frame.

hearn-50265; ISBN: 0-13-015390-7 *book* *July 29, 2003* *14:54*

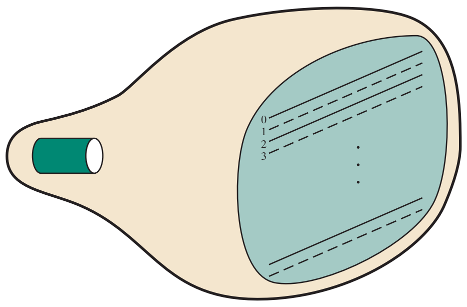

FIGURE 2-8 Interlacing scan lines on a raster-scan display. First, all points on the even-numbered (solid) scan lines are displayed; then all points along the odd-numbered (dashed) lines are displayed.

On some raster-scan systems and 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. After the vertical retrace, the beam then 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. This technique is primarily used with slower refresh 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 $\frac{1}{60}$ of a second, which brings the refresh rate nearer to 60 frames per second. This is an effective technique for avoiding flicker—provided that adjacent scan lines contain similar display information.

Random-Scan Displays

When operated as a **random-scan display** unit, a CRT has the electron beam directed only to those parts of the screen where a picture is to be displayed. Pictures are generated as line drawings, with the electron beam tracing out the component lines one after the other. For this reason, random-scan monitors are also referred to as **vector displays** (or **stroke-writing displays** 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 on that system. Picture definition is now stored as a set of line-drawing commands in an area of memory referred to as the **display list, refresh display file, vector file,** or **display program.** To display a specified picture, the system cycles through the set of commands in the display file, drawing each component line in turn. After all line-drawing commands have been processed, the system cycles back to the first line command in the list. Random-scan displays are designed to draw all the component lines of a picture 30 to 60 times each second, with up to 100,000 "short" lines in the display list. When a small set of lines is to be displayed, each refresh cycle is delayed to avoid very high refresh rates, which could burn out the phosphor.

Random-scan systems were designed for line-drawing applications, such as architectural and engineering layouts, and they cannot display realistic shaded scenes. Since picture definition is stored as a set of line-drawing instructions rather than as a set of intensity values for all screen points, vector displays generally have higher resolutions than raster systems. Also, vector displays produce smooth line drawings because the CRT beam directly follows the line path. A raster system, by

hearn-50265; ISBN: 0-13-015390-7 *book* *July 29, 2003* *14:54*

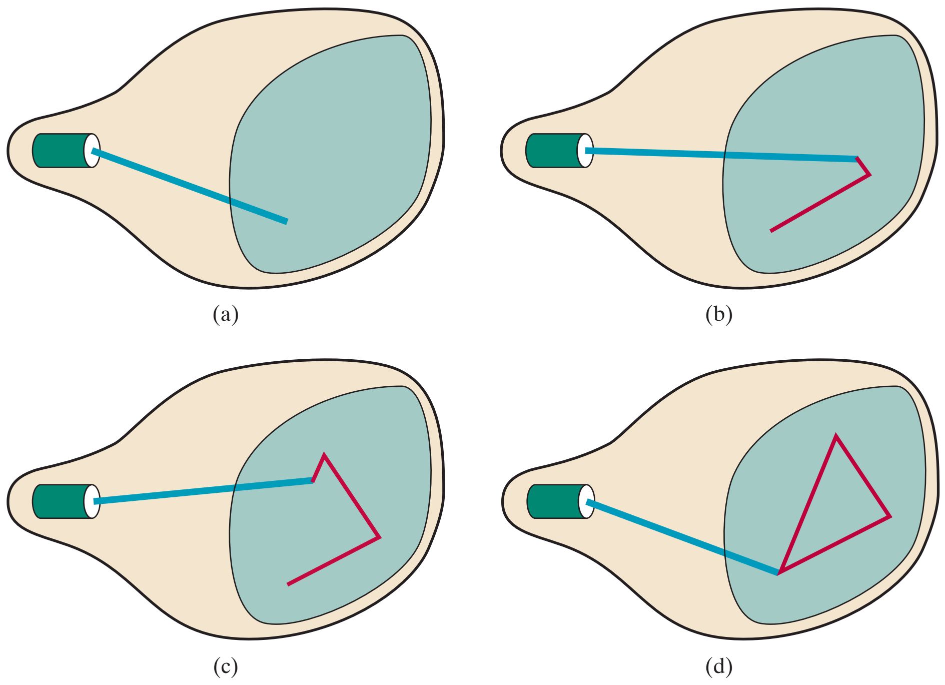

FIGURE 2–9 A random-scan system draws the component lines of an object in any specified order.

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. The emitted light from the different phosphors merges to form a single perceived color, which depends on the particular set of phosphors that have been excited.

One way to display color pictures is to coat the screen with layers of different-colored phosphors. The emitted color depends on how far the electron beam penetrates into the phosphor layers. This approach, called the **beam-penetration** method, typically used only two phosphor layers: red and green. A beam of slow electrons excites only the outer red layer, but 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, but only a limited number of colors are possible, and picture quality is not as good as with other methods.

Shadow-mask methods are commonly used in raster-scan systems (including color TV) since they produce a much wider range of colors than the beam-penetration method. This approach is based on the way that we seem to perceive colors as combinations of red, green, and blue components, called the **RGB color model.** Thus, a shadow-mask CRT uses 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

hearn-50265; ISBN: 0-13-015390-7 book July 30, 2003 15:15

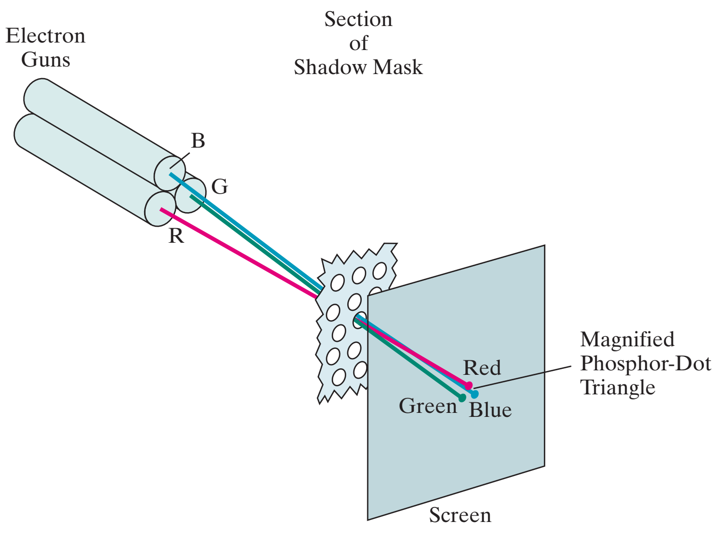

FIGURE 2–10 Operation of a delta-delta, shadow-mask CRT. Three electron guns, aligned with the triangular color-dot patterns on the screen, are directed to each dot triangle by a shadow mask.

color dot, and a shadow-mask grid just behind the phosphor-coated screen. The light emitted from the three phosphors results in a small spot of color at each pixel position, since our eyes tend to merge the light emitted from the three dots into one composite color. 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 two of the three guns, we get only the color coming from the single activated phosphor (red, green, or blue). When all three dots are activated with equal beam intensities, we see a white color. Yellow is produced with equal intensities from the green and red dots only, magenta is produced with equal blue and red intensities, and cyan shows up when blue and green are activated equally. In an inexpensive system, each of the three electron beams might be restricted to either on or off, limiting displays to eight colors. More sophisticated systems can allow intermediate intensity levels to be set for the electron beams, so that several million colors are possible.

Color graphics systems can be used with several types of CRT display devices. Some inexpensive home-computer systems and video games have been designed for use with a color TV set and an RF (radio-frequency) modulator. The purpose of the RF modulator is to simulate the signal from a broadcast TV station. This means that the color and intensity information of the picture must be combined and superimposed on the broadcast-frequency carrier signal that the TV requires as input. Then the circuitry in the TV takes this signal from the RF modulator, extracts the picture information, and paints it on the screen. As we might expect, this

hearn-50265; ISBN: 0-13-015390-7 *book* *July 29, 2003* *14:54*

extra handling of the picture information by the RF modulator and TV circuitry decreases the quality of displayed images.

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. Since picture information is combined into a composite signal and then separated by the monitor, 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 are also available as pocket notepads. Some additional uses for flat-panel displays are as small TV monitors, calculator screens, pocket video-game screens, laptop computer screens, armrest movie-viewing stations on airlines, advertisement boards in elevators, and 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 are accelerated parallel to the screen and then deflected 90° onto 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 conducting ribbons is built into the other glass panel (Fig. 2-11). Firing voltages applied to an intersecting 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 are now available with multicolor capabilities.

hearn-50265; ISBN: 0-13-015390-7 book July 29, 2003 14:54

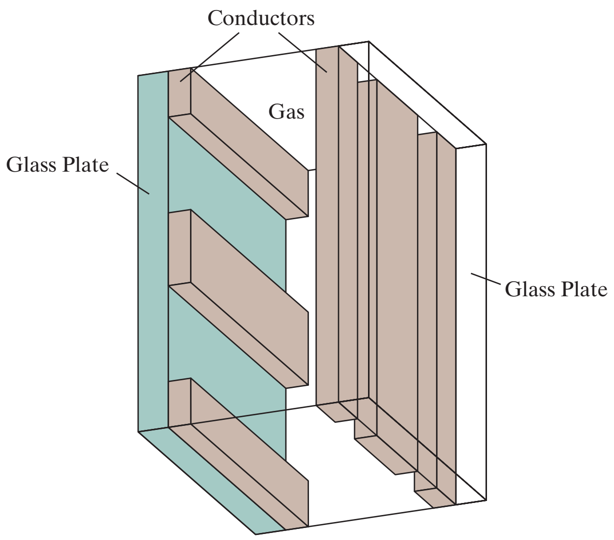

FIGURE 2-11 Basic design of a plasma-panel display device.

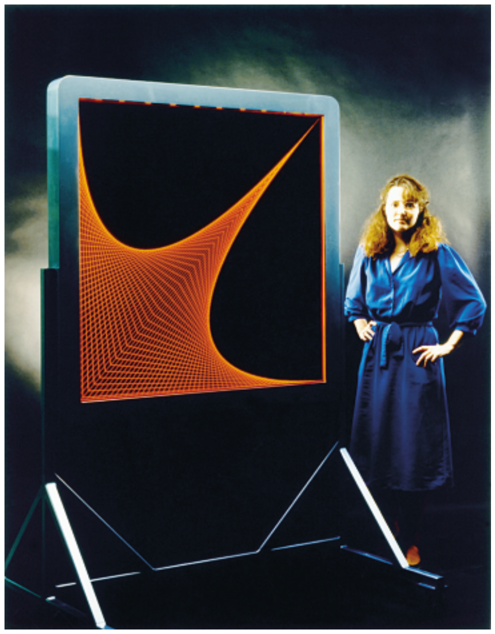

FIGURE 2-12 A plasma-panel display with a resolution of 2048 by 2048 and a screen diagonal of 1.5 meters. (*Courtesy of Photonics Systems.*)

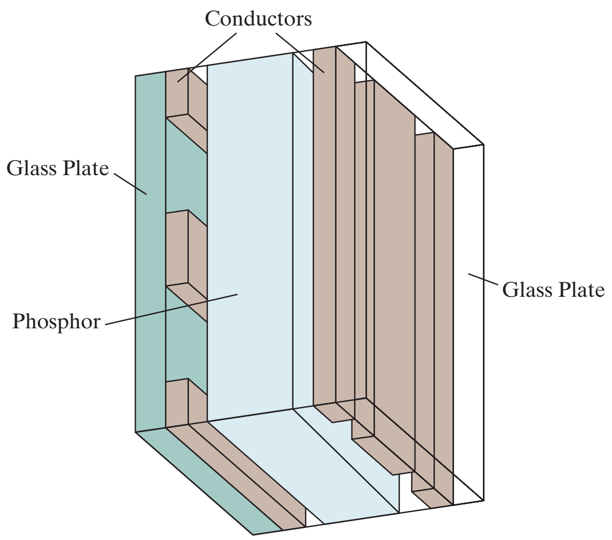

FIGURE 2-13 Basic design of a thin-film electroluminescent display device.

Thin-film electroluminescent displays are similar in construction to plasma panels. 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 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.

hearn-50265; ISBN: 0-13-015390-7 book July 29, 2003 14:54

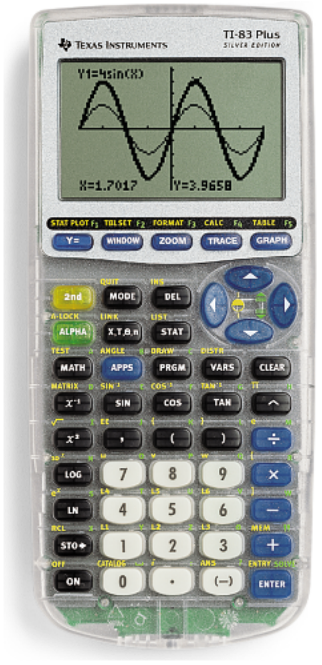

FIGURE 2–14 A handheld calculator with an LCD screen. (*Courtesy of Texas Instruments.*)

Electroluminescent displays require more power than plasma panels, and good color displays are harder 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.

Liquid-crystal displays (**LCDs**) are commonly used in small systems, such as laptop computers and calculators (Fig. 2-14). These nonemissive devices produce a picture by passing polarized light from the surroundings or from an internal light source 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 that is aligned at a right angle 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. 2-15. 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

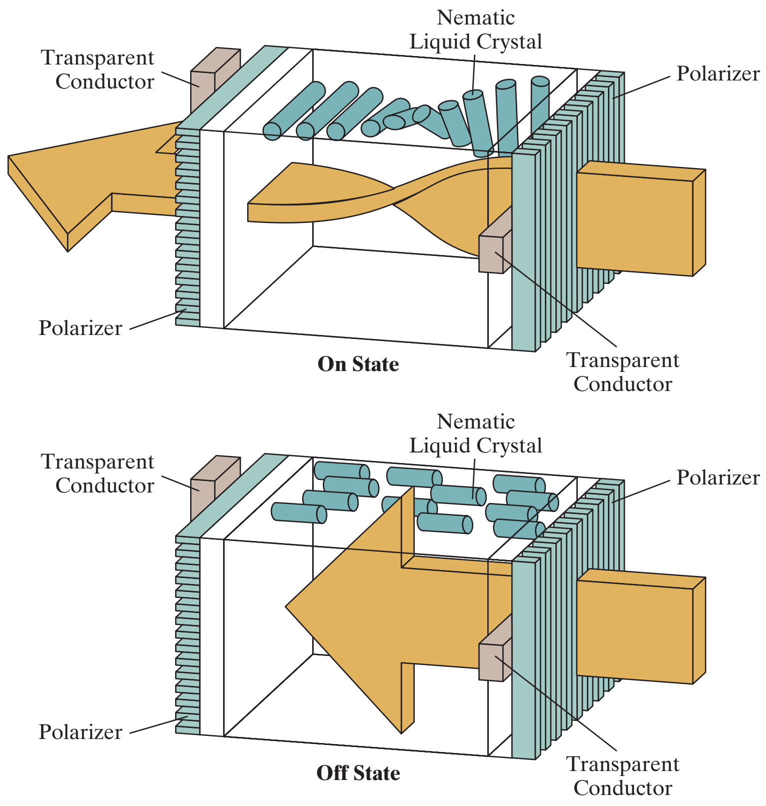

FIGURE 2–15 The light-twisting, shutter effect used in the design of most liquid-crystal display devices.

hearn-50265; ISBN: 0-13-015390-7 book July 29, 2003 14:54

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. Backlighting is also commonly applied using solid-state electronic devices, so that the system is not completely dependent on outside light sources. 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.

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 (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 location. This allows us to walk around an object or scene and view it from different sides.

Figure 2-17 shows the Genisco SpaceGraph system, which uses a vibrating mirror to project three-dimensional objects into a 25-cm by 25-cm by 25-cm volume. This system is also capable of displaying two-dimensional cross-sectional

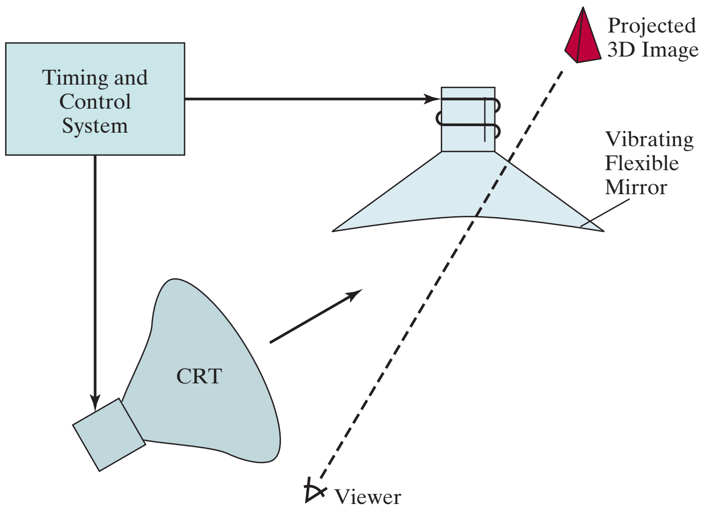

FIGURE 2-16 Operation of a three-dimensional display system using a vibrating mirror that changes focal length to match the depths of points in a scene.

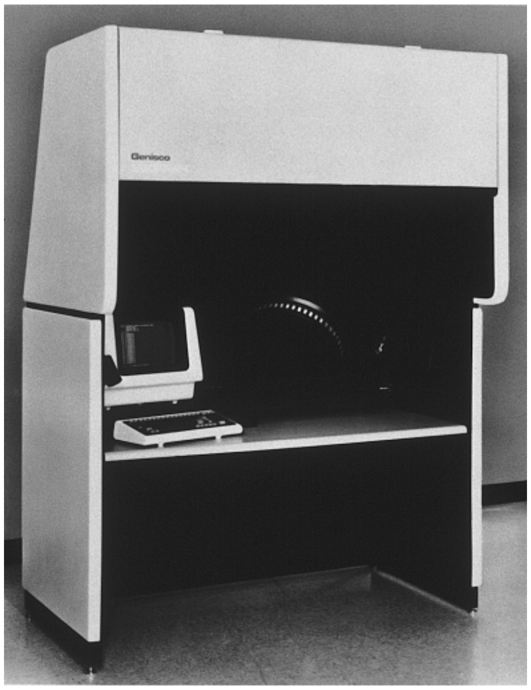

FIGURE 2-17 The SpaceGraph interactive graphics system displays objects in three dimensions using a vibrating, flexible mirror. (*Courtesy of Genisco Computers Corporation.*)

"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.

Stereoscopic and Virtual-Reality Systems

Another technique for representing a three-dimensional object is to display stereoscopic views of the object. 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).

To obtain a stereoscopic projection, we must obtain two views of a scene generated with viewing directions along the lines from the position of each eye (left and right) to the scene. 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.

One way to produce a stereoscopic effect on a raster system is to display each of the two views 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 can

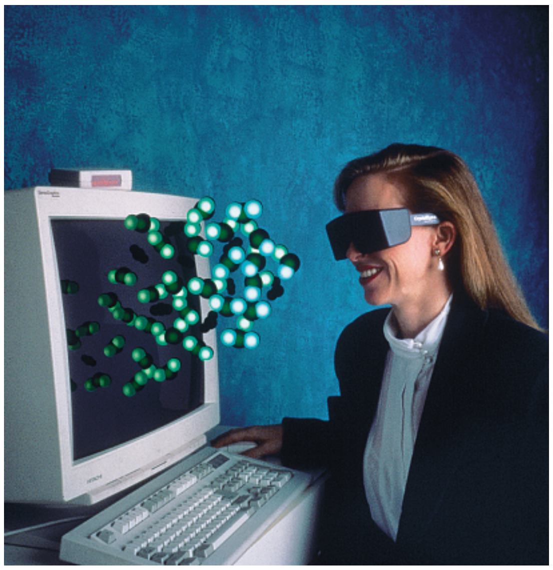

FIGURE 2-18 Simulated viewing of a stereoscopic projection. (*Courtesy of StereoGraphics Corporation.*)

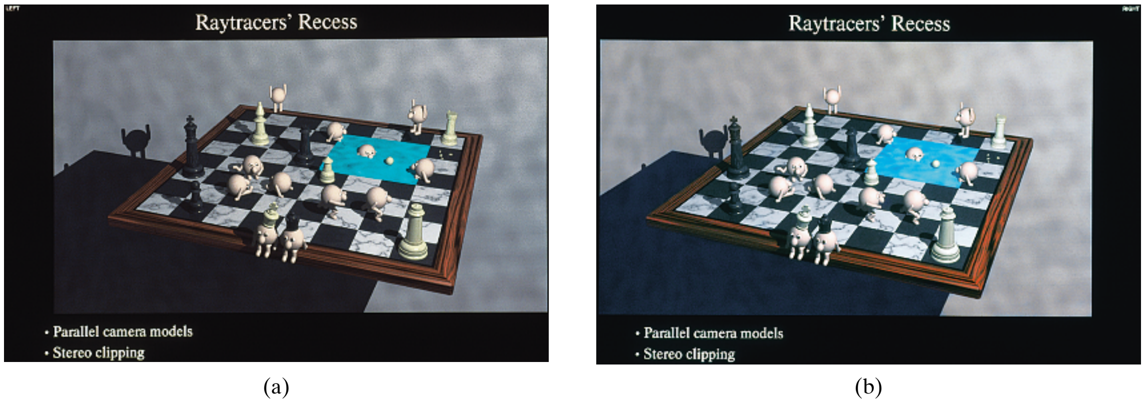

FIGURE 2-19 A stereoscopic viewing pair. (*Courtesy of Jerry Farm.*)

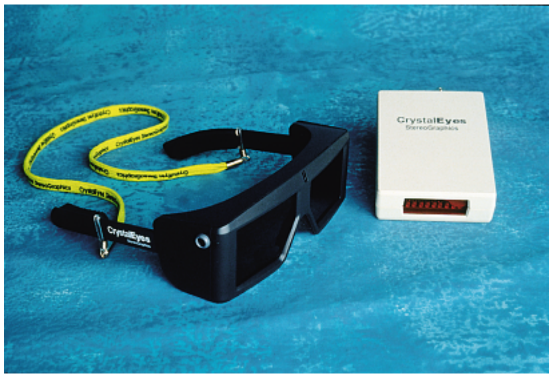

FIGURE 2-20 Glasses for viewing a stereoscopic scene and an infrared synchronizing emitter. (*Courtesy of StereoGraphics Corporation.*)

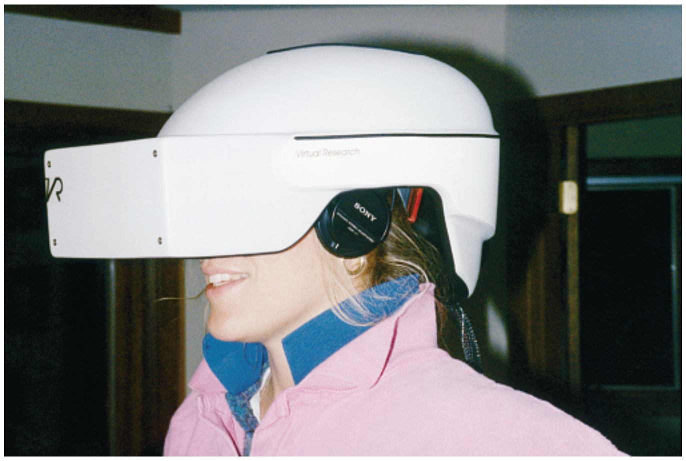

FIGURE 2-21 A headset used in virtual-reality systems. (*Courtesy of Virtual Research.*)

be 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. Another method for creating a virtual-reality environment is to use projectors to generate a scene within an arrangement of walls, as in Figure 2-22, where a viewer interacts with a virtual display using stereoscopic glasses and data gloves (Section 2-4).

Lower-cost, interactive virtual-reality environments can be set up using a graphics monitor, stereoscopic glasses, and a head-tracking device. Figure 2-23 shows an ultrasound tracking device with six degrees of freedom. The tracking device is placed above the video monitor and is used to record 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.*)

2-2 RASTER-SCAN SYSTEMS

Interactive raster-graphics systems typically employ several processing units. In addition to the central processing unit, or CPU, a special-purpose processor, called the **video controller** or **display controller,** is used to control the operation of the display device. Organization of a simple raster system is shown in Fig. 2-24. 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 coprocessors and accelerators to implement various graphics operations.

Video Controller

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.

Frame-buffer locations, and the corresponding screen positions, are referenced in Cartesian coordinates. In an application program, we use the commands

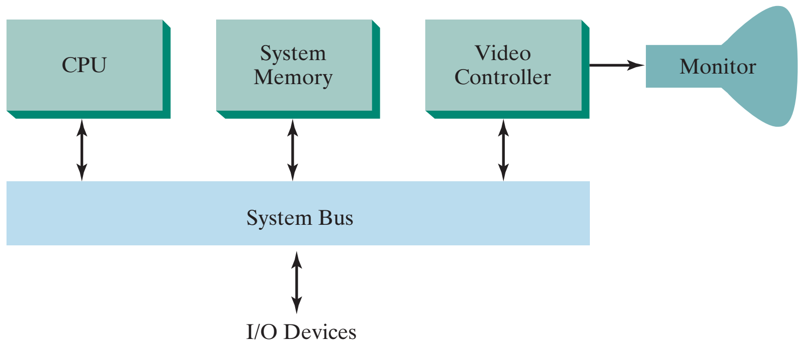

FIGURE 2-24 Architecture of a simple raster-graphics system.

hearn-50265; ISBN: 0-13-015390-7 book July 29, 2003 14:54

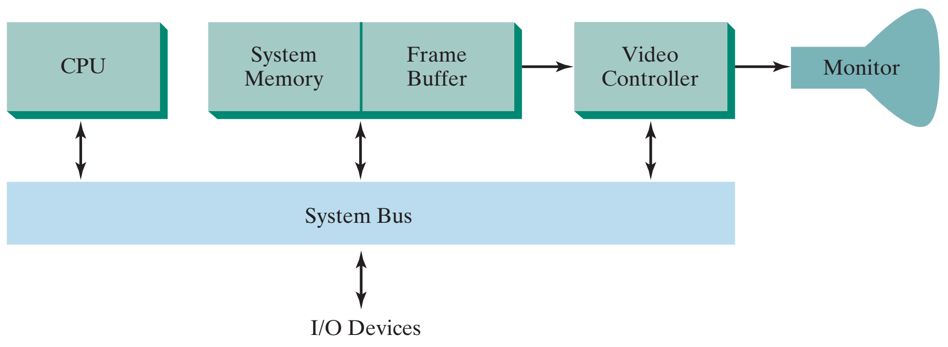

FIGURE 2-25 Architecture of a raster system with a fixed portion of the system memory reserved for the frame buffer.

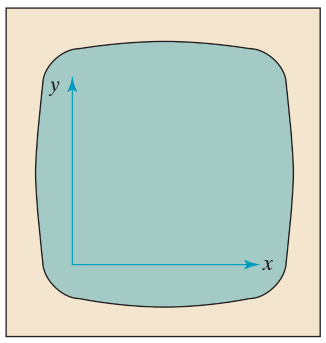

FIGURE 2-26 A Cartesian reference frame with origin at the lower-left corner of a video monitor.

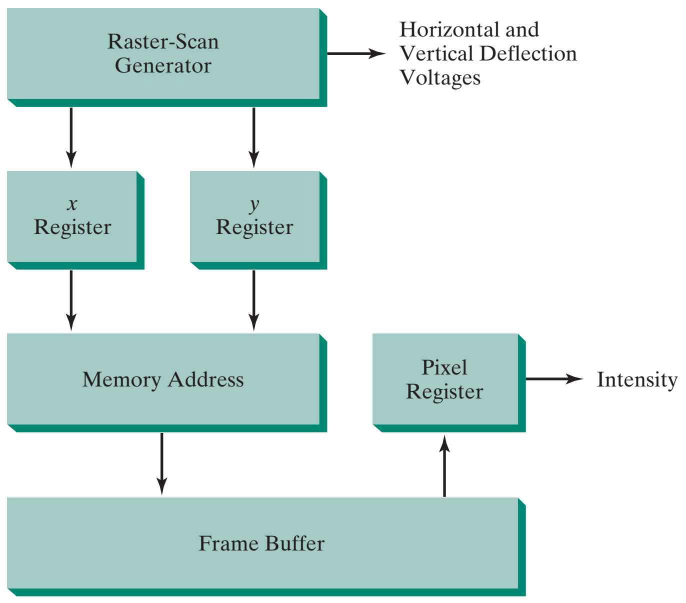

FIGURE 2-27 Basic video-controller refresh operations.

within a graphics software package to set coordinate positions for displayed objects relative to the origin of the Cartesian reference frame. Often, the coordinate origin is referenced at the lower-left corner of a screen display area by the software commands, although we can typically set the origin at any convenient location for a particular application. Figure 2-26 shows a two-dimensional Cartesian reference frame with the 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 from left to right and positive y values increasing from the bottom of the screen to the 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. However, hardware processes such as screen refreshing, as well as some software systems, reference the pixel positions from the top-left corner of the screen.

In Fig. 2-27, the basic refresh operations of the video controller are diagrammed. Two registers are used to store the coordinate values for the screen

hearn-50265; ISBN: 0-13-015390-7 book July 29, 2003 14:54

pixels. Initially, the x register is set to 0 and the y register is set to the value for the top scan line. The contents of the frame buffer at this pixel position are then retrieved and used to set the intensity of the CRT beam. Then the x register is incremented by 1, and the process is repeated for the next pixel on the top scan line. This procedure continues for each pixel along the top 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 set to the value for the next scan line down from the top of the screen. 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, the video controller resets the registers to the first pixel position on the top scan line and the refresh process starts over.

Since the screen must be refreshed at a rate of at least 60 frames per second, the simple procedure illustrated in Fig. 2-27 may not be accommodated by typical RAM chips if 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 video controller can be designed to perform a number of other operations. For various applications, the video controller can retrieve pixel values from different memory areas on different refresh cycles. In some systems, for example, multiple frame buffers are often provided so that one buffer can be used for refreshing while pixel values are being loaded into the other buffers. Then the current refresh buffer can switch roles with one of the other buffers. This provides a fast mechanism for generating real-time animations, for example, since different views of moving objects can be successively loaded into a buffer without interrupting a refresh cycle. Another video-controller task is the transformation of blocks of pixels, so that screen areas 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, and lookup tables are discussed in more detail in Chapter 4. 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 organize the components of a raster system that contains a separate **display processor,** sometimes referred to as a **graphics controller** or a **display coprocessor.** The purpose of the display processor is to free the CPU from the graphics chores. In addition to the system memory, a separate display-processor memory area can be provided.

A major task of the display processor is digitizing a picture definition given in an application program into a set of pixel values for storage in the frame buffer. This digitization process is called **scan conversion.** Graphics commands specifying straight lines and other geometric objects are scan converted into a set

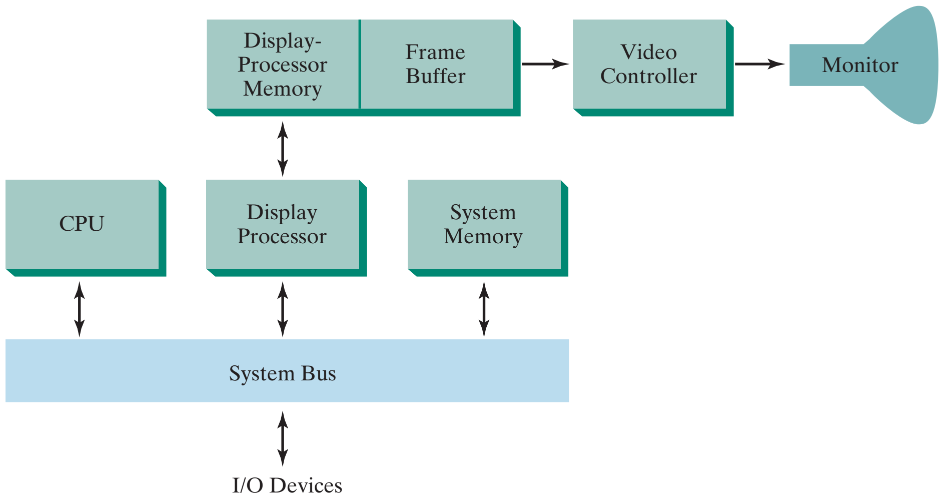

FIGURE 2-28 Architecture of a raster-graphics system with a display processor.

of discrete points, corresponding to screen pixel positions. 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 color for each position in the frame buffer. Similar methods are used for scan converting other objects in a picture definition. Characters can be defined with rectangular pixel grids, as in Fig. 2-29, or they can be defined with outline shapes, 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. For characters that are defined as outlines, the shapes are scan converted into the frame buffer by locating the pixel positions closest to the outline.

Display processors are also designed to perform a number of additional operations. These functions include generating various line styles (dashed, dotted, or solid), displaying color areas, and applying transformations to the objects in a scene. 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 organization scheme is to store each scan line as a set of number pairs. The first number in each pair can be a reference to a color value, and the second number can specify the number of adjacent pixels on the scan line that are to be displayed in that color. This technique, called **run-length encoding,** can result in a considerable saving in storage space if a picture is to be constructed mostly with long runs of a single color each. A similar approach can be taken when pixel colors change linearly. 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 record and storage requirements increase as the lengths of the runs decrease. In addition, it is difficult for the display controller to process the raster when many short runs are involved. Moreover, the size of the frame buffer is no longer a major concern, because of sharp declines in memory costs. Nevertheless, encoding methods can be useful in the digital storage and transmission of picture information.

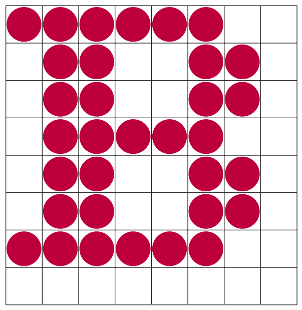

FIGURE 2-29 A character defined as a rectangular grid of pixel positions.

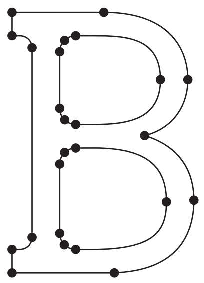

FIGURE 2-30 A character defined as an outline shape.

hearn-50265; ISBN: 0-13-015390-7 book July 29, 2003 14:54

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

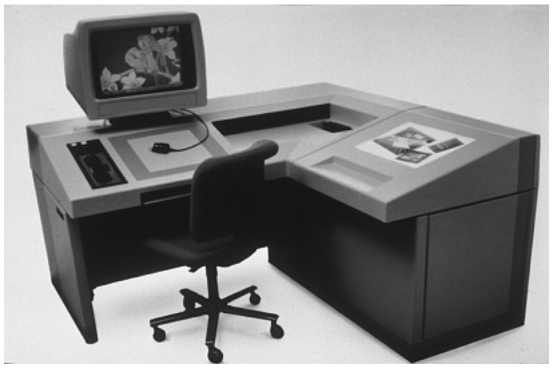

FIGURE 2-31 An artist's workstation, featuring a monitor, a keyboard, a graphics tablet with a hand cursor, and a light table, in addition to data storage and telecommunications devices. (*Courtesy of DICOMED Corporation.*)

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

hearn-50265; ISBN: 0-13-015390-7 book August 15, 2003 15:40

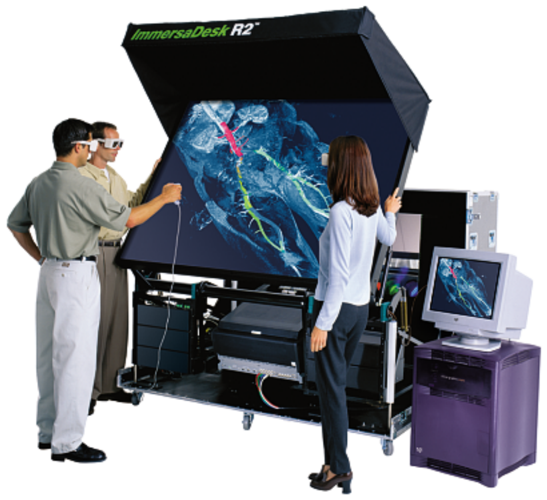

FIGURE 2-33 The SGI Reality Center 2000D, featuring an ImmersaDesk R2 and displaying a large-screen stereoscopic view of pressure contours in a vascular blood-flow simulation superimposed on a volume-rendered anatomic data set. (*Courtesy of Silicon Graphics, Inc. and Professor Charles Taylor, Stanford University. © 2003 SGI. All rights reserved.*)

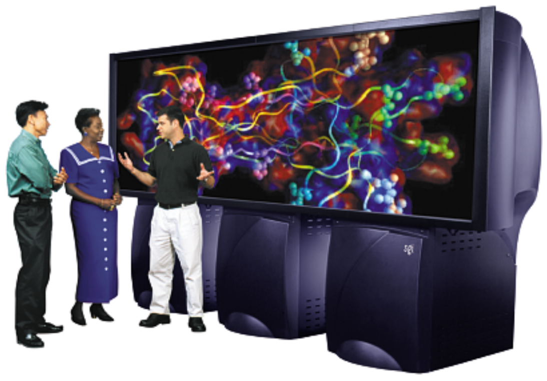

FIGURE 2-34 A wide-screen view of a molecular system displayed on the three-channel SGI Reality Center 3300W. (*Courtesy of Silicon Graphics, Inc. and Molecular Simulations. © 2003 SGI. All rights reserved.*)

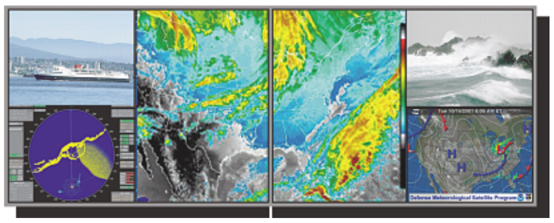

FIGURE 2-35 A multi-panel display system called the "Super Wall". (*Courtesy of RGB Spectrum.*)

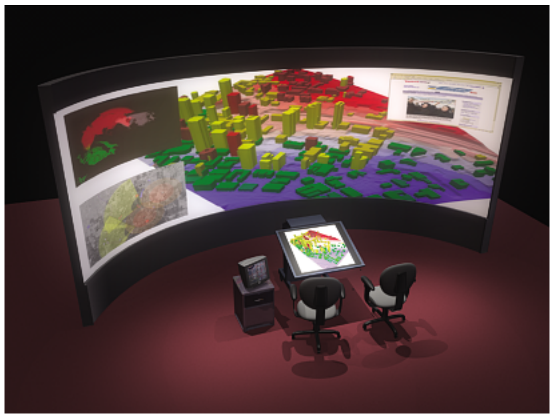

FIGURE 2-36 A homeland security study displayed using a system with a large curved viewing screen. (*Courtesy of Silicon Graphics, Inc. © 2003. All rights reserved.*)

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.

hearn-50265; ISBN: 0-13-015390-7 book July 29, 2003 14:54

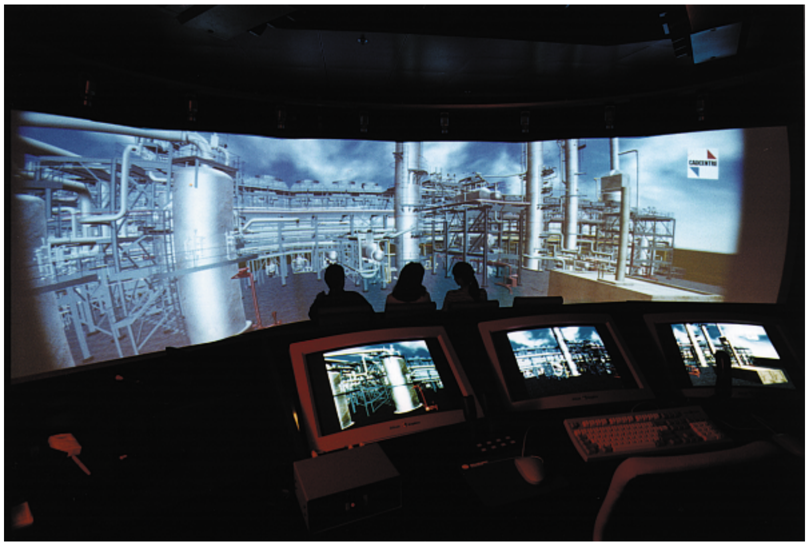

FIGURE 2-37 A curved-screen graphics system displaying an interactive walk-through of a natural gas plant. (*Courtesy of Silicon Graphics, Inc., Trimension Systems, and the Cadcentre, Cortaillod, Switzerland. © 2003 SGI. All rights reserved.*)

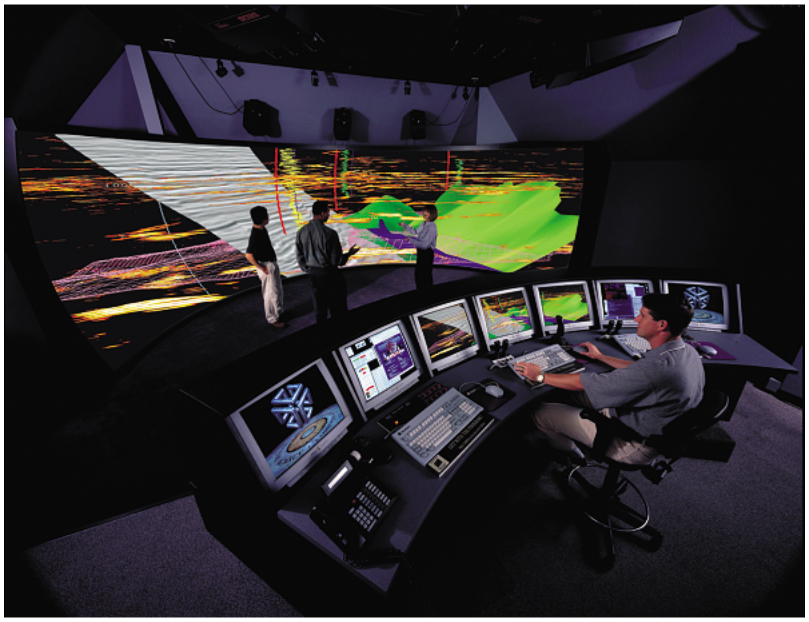

FIGURE 2-38 A geophysical visualization presented on a 25-foot semicircular screen, which provides a 160° horizontal and 40° vertical field of view. (*Courtesy of Silicon Graphics, Inc., the Landmark Graphics Corporation, and Trimension Systems. © 2003 SGI. All rights reserved.*)

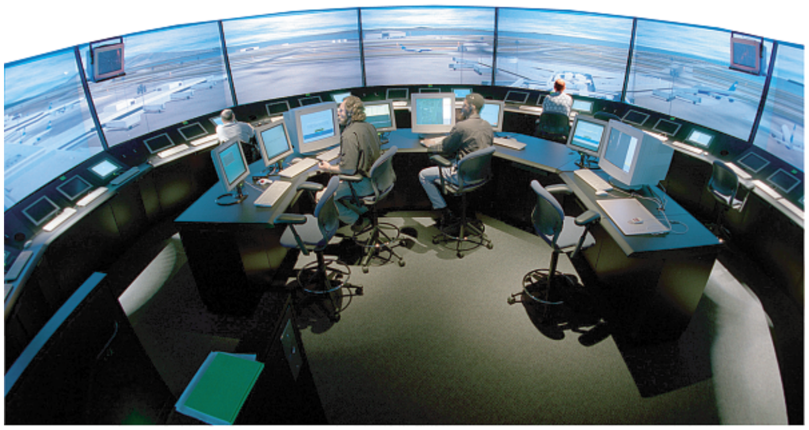

FIGURE 2-39 The 360° viewing screen in the NASA airport control-tower simulator, called the FutureFlight Central Facility. (*Courtesy of Silicon Graphics, Inc. and NASA. © 2003 SGI. All rights reserved.*)

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 small 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

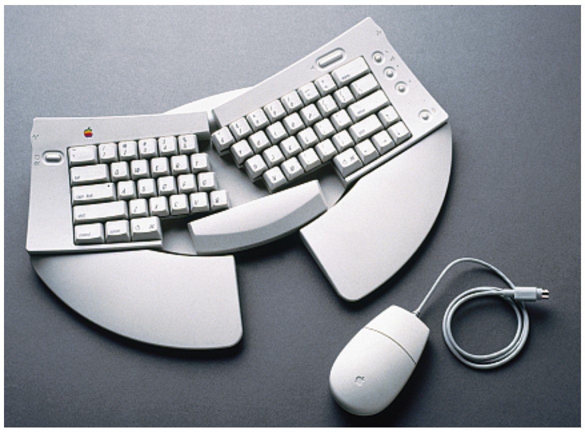

FIGURE 2-40
Ergonomically designed keyboard with removable palm rests. The slope of each half of the keyboard can be adjusted separately. A one-button mouse, shown in front of the keyboard, has a cable attachment for connection to the main computer unit. (*Courtesy of Apple Computer, Inc.*)

hearn-50265; ISBN: 0-13-015390-7 book July 29, 2003 14:54

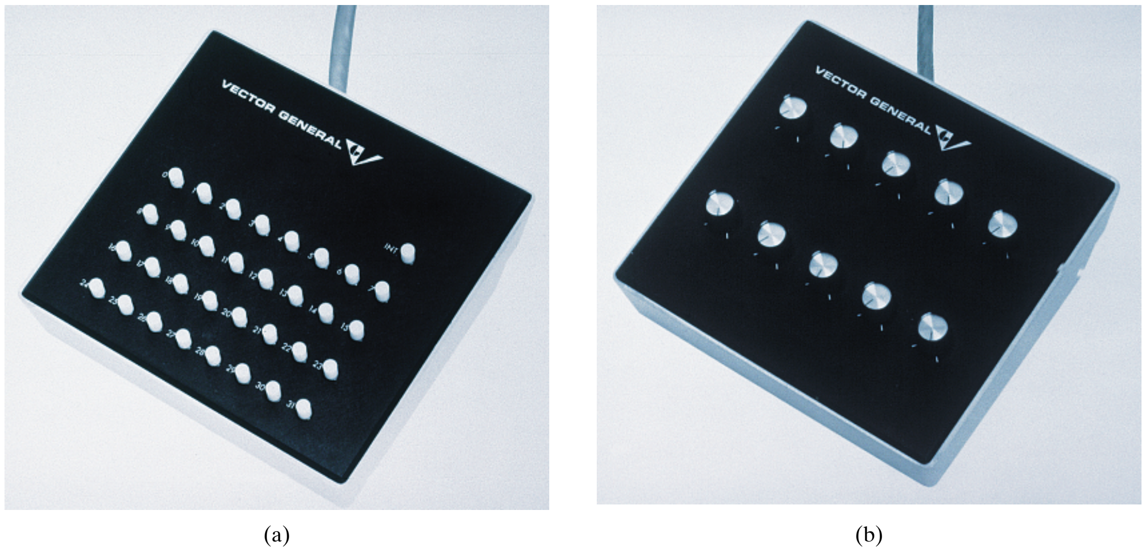

FIGURE 2-41 A button box (a) and a set of input dials (b). (*Courtesy of Vector General.*)

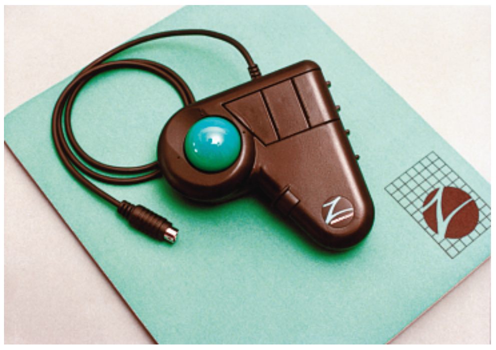

FIGURE 2-42 The Z mouse features three buttons, a mouse ball underneath, a thumbwheel on the side, and a trackball on top. (*Courtesy of the Multipoint Technology Corporation.*)

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.

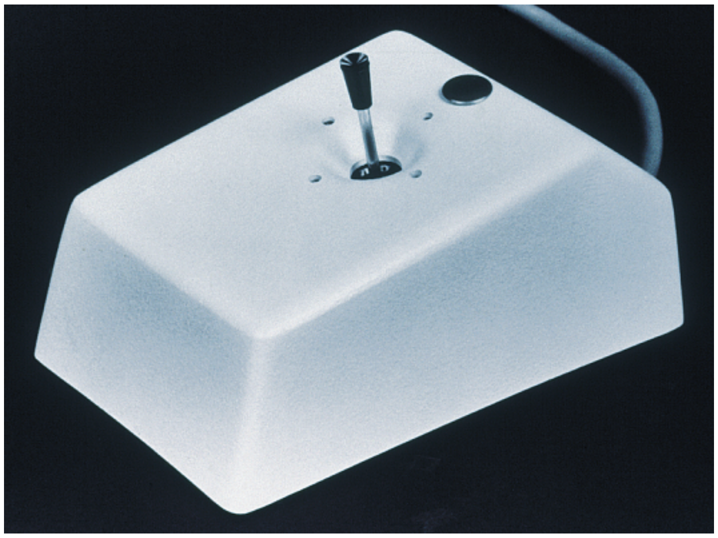

FIGURE 2-43 A movable joystick. (*Courtesy of the CalComp Group, Sanders Associates, Inc.*)

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

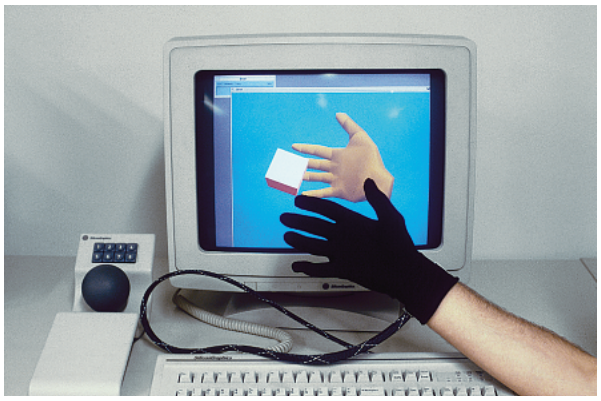

FIGURE 2-44 A virtual-reality scene, displayed on a two-dimensional video monitor, with input from a data glove and a spaceball. (*Courtesy of The Computer Graphics Center, Darmstadt, Germany.*)

hearn-50265; ISBN: 0-13-015390-7 book July 29, 2003 14:54

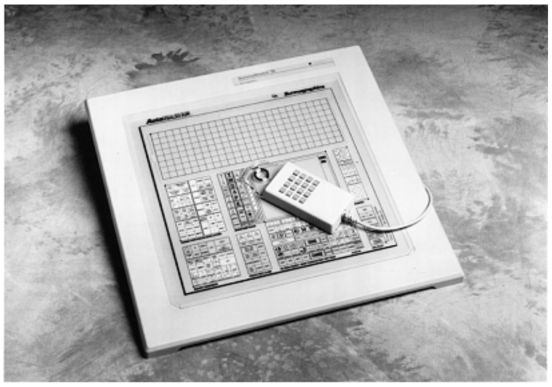

FIGURE 2-45 The SummaSketch III desktop tablet with a sixteen-button hand cursor. (*Courtesy of Summagraphics Corporation.*)

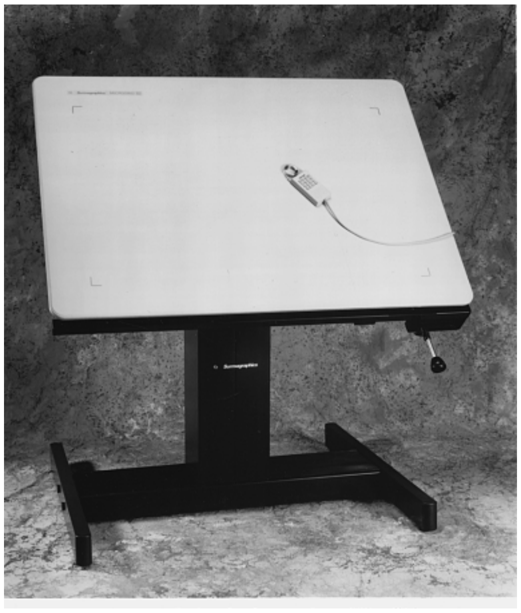

FIGURE 2-46 The Microgrid III tablet with a sixteen-button hand cursor, designed for digitizing larger drawings. (*Courtesy of Summagraphics Corporation.*)

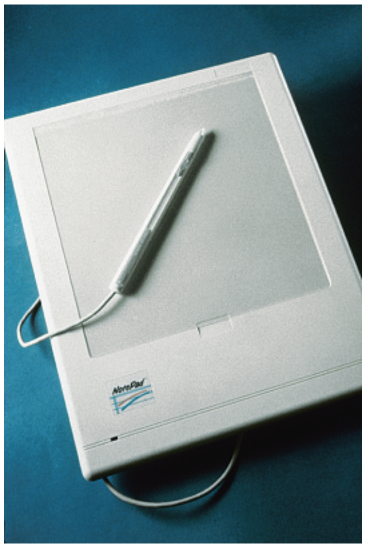

FIGURE 2-47 The NotePad desktop tablet with stylus. (*Courtesy of CalComp Digitizer Division, a part of CalComp, Inc.*)

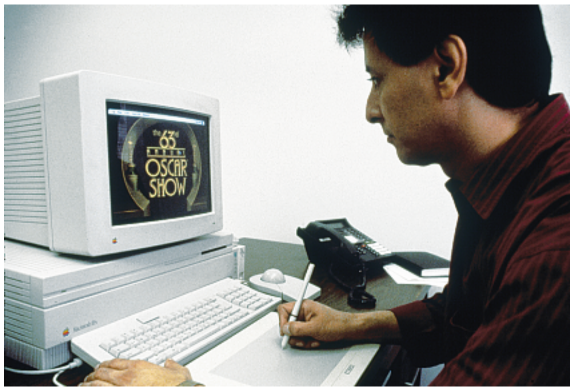

FIGURE 2-48 An artist's digitizer system, with a pressure-sensitive, cordless stylus. (*Courtesy of Wacom Technology Corporation.*)

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

hearn-50265; ISBN: 0-13-015390-7 book July 29, 2003 14:54

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

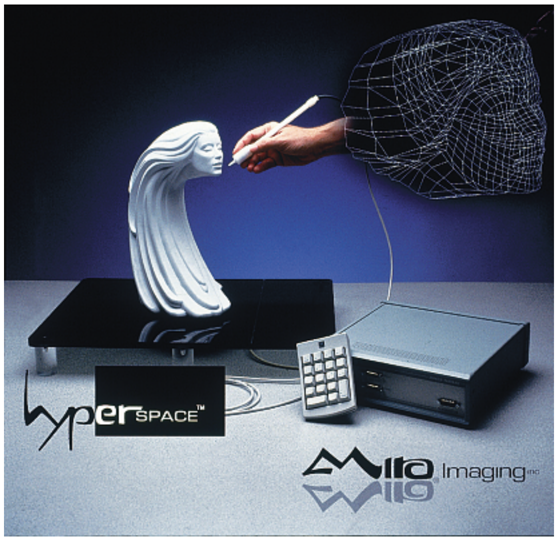

FIGURE 2-49 A three-dimensional digitizing system for use with Apple Macintosh computers. (*Courtesy of Mira Imaging.*)

hearn-50265; ISBN: 0-13-015390-7 book August 15, 2003 15:40

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.

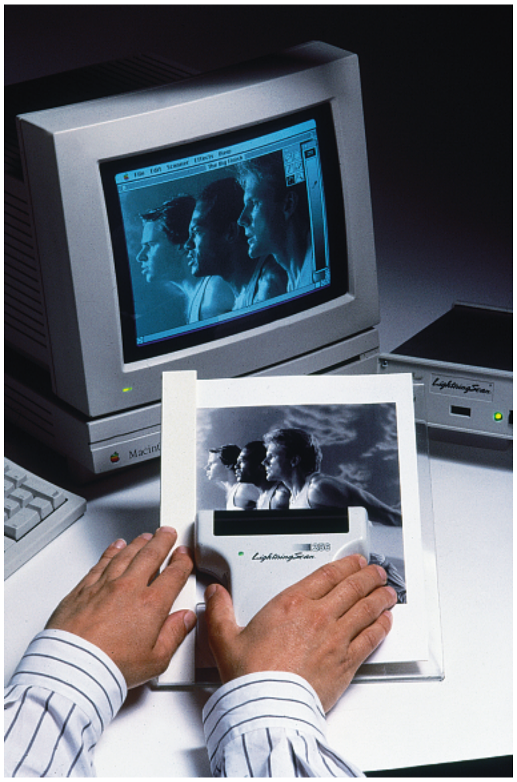

FIGURE 2-50 A hand-held scanner that can be used to input either text or graphics images. (*Courtesy of Thunderware, Inc.*)

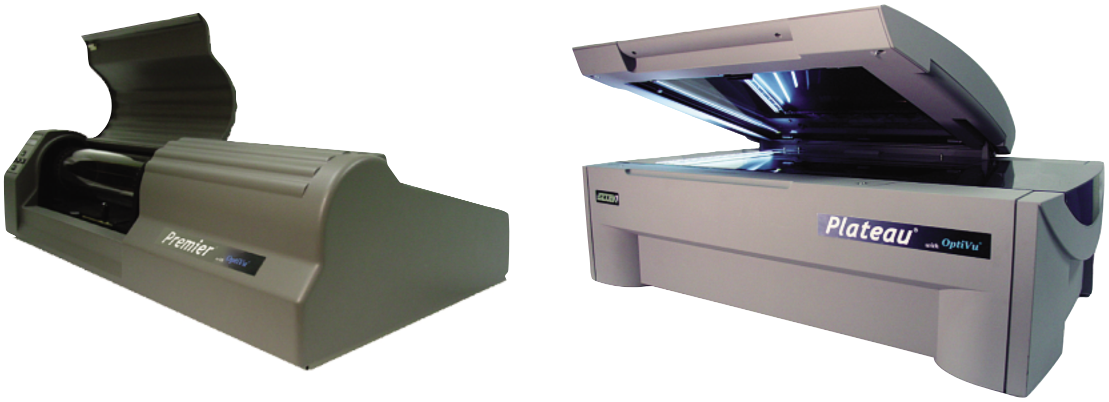

FIGURE 2-51 Desktop scanners: (a) drum scanner and (b) flatbed scanner. (*Courtesy of Aztek, Inc., Lake Forest, California.*)

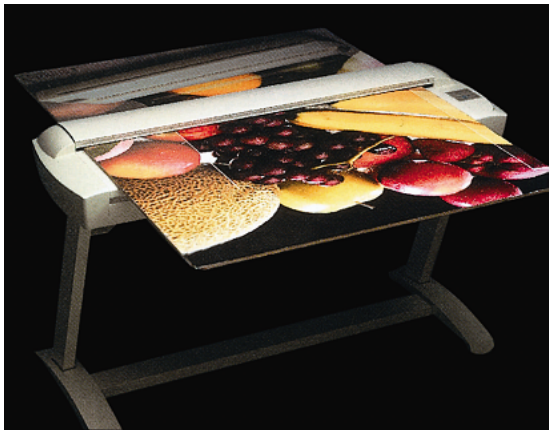

FIGURE 2-52
A wide-format scanner. (*Courtesy of Aztek, Inc., Lake Forest, California.*)

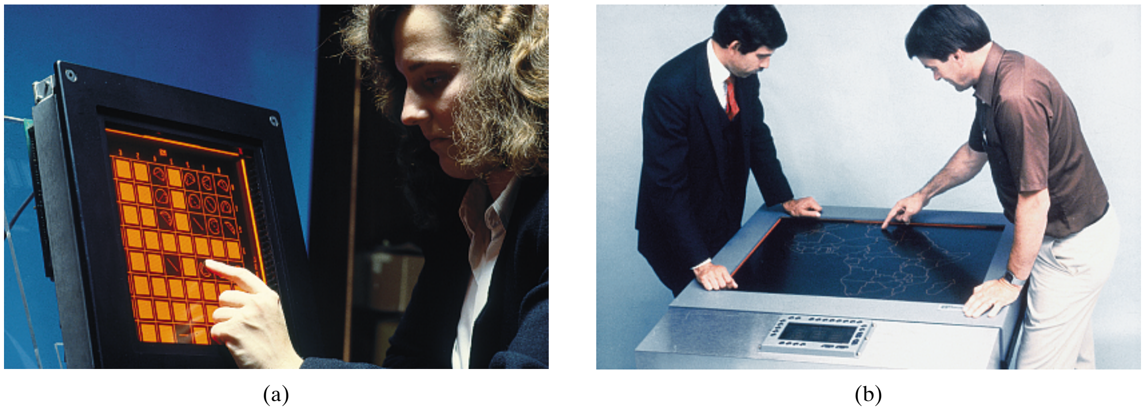

FIGURE 2-53 Plasma panels with touch screens. (*Courtesy of Photonics Systems.*)

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

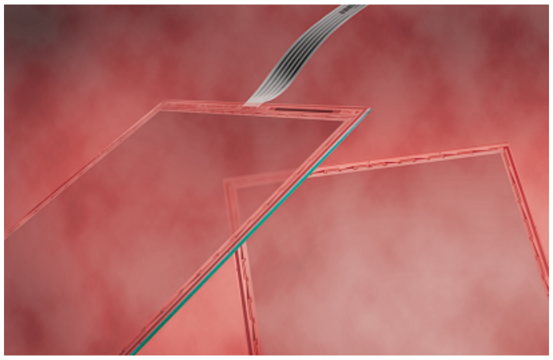
FIGURE 2-54 Resistive touch-screen overlays. (*Courtesy of Elo TouchSystems, Inc.*)

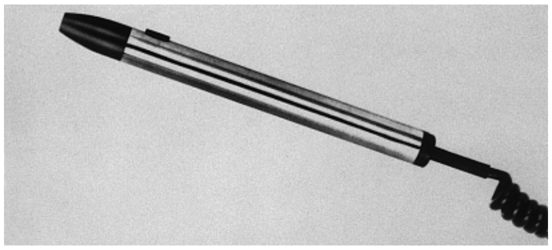
FIGURE 2-55 A light pen with a button activator. (*Courtesy of Interactive Computer Products.*)

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

hearn-50265; ISBN: 0-13-015390-7 book July 29, 2003 14:54

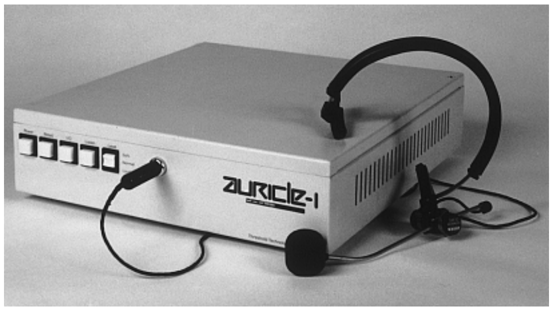

FIGURE 2-56 A speech-recognition system. (*Courtesy of Threshold Technology, Inc.*)

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.

2-5 HARD-COPY DEVICES

We can obtain hard-copy output for our images in several formats. For presentations or archiving, we can send image files to devices or service bureaus that will produce overhead transparencies, 35mm slides, or film. And we can put our pictures on paper by directing graphics output to a printer or plotter.

The quality of the pictures obtained from an output device depends on dot size and the number of dots per inch, or lines per inch, that can be displayed. To produce smooth patterns, higher-quality printers shift dot positions so that adjacent dots overlap.

Printers produce output by either impact or nonimpact methods. *Impact* printers press formed character faces against an inked ribbon onto the paper. A line printer is an example of an impact device, with the typefaces mounted on bands, chains, drums, or wheels. *Nonimpact* printers and plotters use laser techniques, ink-jet sprays, electrostatic methods, and electrothermal methods to get images onto paper.

Character impact printers often have a *dot-matrix* print head containing a rectangular array of protruding wire pins, with the number of pins dependent upon the quality of the printer. Individual characters or graphics patterns are obtained by retracting certain pins so that the remaining pins form the pattern to be printed. Figure 2-57 shows a picture printed on a dot-matrix printer.

In a *laser* device, a laser beam creates a charge distribution on a rotating drum coated with a photoelectric material, such as selenium. Toner is applied to the drum and then transferred to paper. *Ink-jet* methods produce output by squirting ink in horizontal rows across a roll of paper wrapped on a drum. The electrically charged ink stream is deflected by an electric field to produce dot-matrix patterns. And an *electrostatic* device places a negative charge on the paper, one complete

hearn-50265; ISBN: 0-13-015390-7 *book* *July 29, 2003* *14:54*

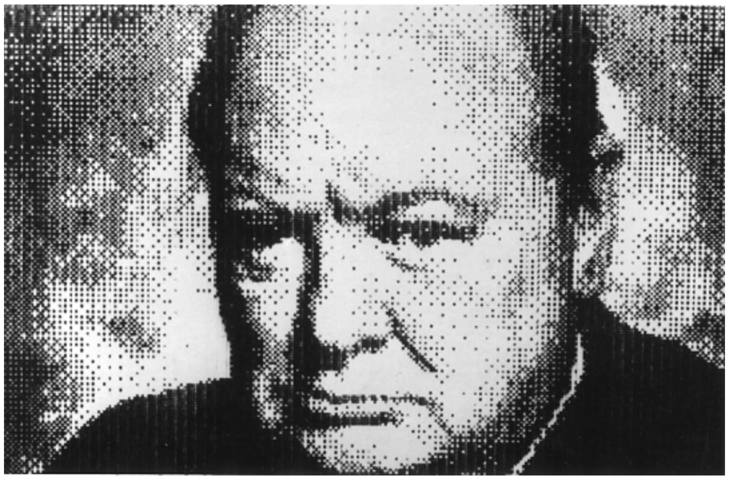

FIGURE 2-57 A picture generated on a dot-matrix printer, illustrating how the density of dot patterns can be varied to produce light and dark areas. (*Courtesy of Apple Computer, Inc.*)

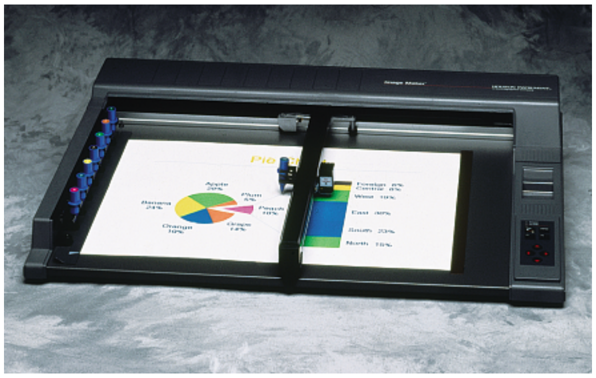

FIGURE 2-58 A desktop pen plotter with a resolution of 0.025 mm. (*Courtesy of Summagraphics Corporation.*)

row at a time across the sheet. Then the paper is exposed to a positively charged toner. This causes the toner to be attracted to the negatively charged areas, where it adheres to produce the specified output. Another output technology is the *electrothermal* printer. With these systems, heat is applied to a dot-matrix print head to output patterns on heat-sensitive paper.

We can get limited color output on some impact printers by using different-colored ribbons. Nonimpact devices use various techniques to combine three different color pigments (cyan, magenta, and yellow) to produce a range of color patterns. Laser and electrostatic devices deposit the three pigments on separate passes; ink-jet methods shoot the three colors simultaneously on a single pass along each print line.

Drafting layouts and other drawings are typically generated with ink-jet or pen plotters. A pen plotter has one or more pens mounted on a carriage, or crossbar, that spans a sheet of paper. Pens with varying colors and widths are used to produce a variety of shadings and line styles. Wet-ink, ball-point, and felt-tip pens are all possible choices for use with a pen plotter. Plotter paper can lie flat or it can be rolled onto a drum or belt. Crossbars can be either movable or stationary, while the pen moves back and forth along the bar. The paper is held in position using clamps, a vacuum, or an electrostatic charge. An example of a table-top, flatbed pen plotter is given in Figure 2-58, and a larger, roll-feed pen plotter is shown in Fig. 2-59.

hearn-50265; ISBN: 0-13-015390-7 book July 29, 2003 14:54

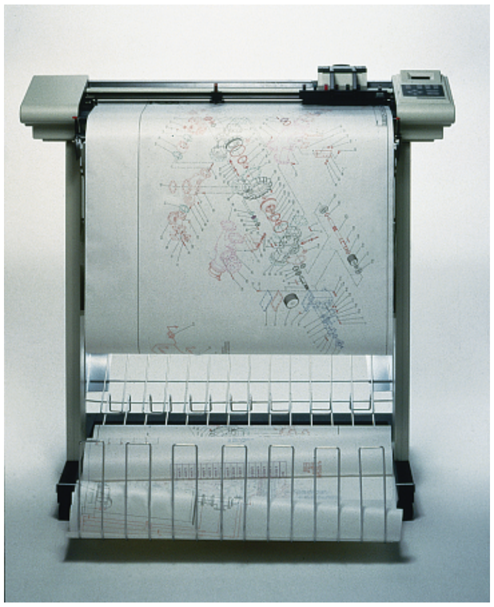

FIGURE 2-59 A large, roll-feed pen plotter with an automatic multicolor eight-pen changer and a resolution of 0.0127 mm. (*Courtesy of Summagraphics Corporation.*)

2-6 GRAPHICS NETWORKS

So far, we have mainly considered graphics applications on an isolated system with a single user. However, multiuser environments and computer networks are now common elements in many graphics applications. Various resources, such as processors, printers, plotters, and data files, can be distributed on a network and shared by multiple users.

A graphics monitor on a network is generally referred to as a **graphics server,** or simply a **server.** Often, 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 a graphics application program is called the **client,** and the output of the program is displayed on a server. A workstation that includes processors, as well as a monitor and input devices, can function as both a server and a client.

When operating on a network, a client computer transmits the instructions for displaying a picture to the monitor (server). Typically, this is accomplished by collecting the instructions into packets before transmission, instead of sending the individual graphics instructions one at a time over the network. Thus, graphics software packages often contain commands that affect packet transmission, as well as the commands for creating pictures.

2-7 GRAPHICS ON THE INTERNET

A great deal of graphics development is now done on the **Internet,** which is a global network of computer networks. Computers on the Internet communicate using TCP/IP (*transmission control protocol/internetworking protocol*). In addition, the **World Wide Web** provides a hypertext system that allows users to locate and view documents that can contain text, graphics, and audio. Resources, such as graphics files, are identified by a *uniform resource locator* (*URL*). Each URL, sometimes also referred to as a universal resource locator, contains two parts: (1) the protocol for transferring the document, and (2) the server that contains the document and, optionally, the location (directory) on the server. For example, the URL *http://www.siggraph.org* indicates a document that is to be transferred with the *hypertext transfer protocol* (*http*) and that the server is www.siggraph.org, which is the home page of the Special Interest Group in Graphics (SIGGRAPH) of the Association for Computing Machinery. Another common type of URL begins with *ftp://*. This identifies an "ftp site", where programs or other files can be downloaded using the *file-transfer protocol*.

Documents on the Internet can be constructed with the *Hypertext Markup Language* (*HTML*). The development of HTML provided a simple method for describing a document containing text, graphics, and references (hyperlinks) to other documents. Although resources could be made available using HTML and URL addressing, it was difficult originally to find information on the Internet. Subsequently, 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 Hypertext Markup Language provides a simple method for developing graphics on the Internet, but it has limited capabilities. Therefore, other languages have been developed for internet graphics applications, and we discuss these languages in Section 2-8.

2-8 GRAPHICS SOFTWARE

There are two broad classifications for computer-graphics software: special-purpose packages and general programming packages. Special-purpose 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 users to communicate with the programs in their own terms. Examples of such applications include artist's painting programs and various architectural, business, medical, and engineering CAD systems. By contrast, a general programming package provides a library of graphics 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, spheres, and other objects), setting color values, selecting views of a scene, and applying rotations or other transformations. Some examples of general graphics programming packages are GL (Graphics Library), OpenGL, VRML (Virtual-Reality Modeling Language), Java 2D, and Java 3D. A set of graphics functions is often called a **computer-graphics application programming interface** (**CG API**), because the library provides a software interface between a programming language (such as C++) and the hardware. So when we write an application program in C++, the graphics routines allow us to construct and display a picture on an output device.

Coordinate Representations

To generate a picture using a programming package, we first need to give the geometric descriptions of the objects that are to be displayed. These descriptions determine the locations and shapes of the objects. For example, a box is specified by the positions of its corners (vertices), and a sphere is defined by its center position and radius. With few exceptions, general graphics packages require geometric descriptions to be specified in a standard, right-handed, Cartesian-coordinate reference frame (Appendix A). If coordinate values for a picture are given in some other reference frame (spherical, hyperbolic, etc.), they must be converted to Cartesian coordinates before they can be input to the graphics package. Some packages that are designed for specialized applications may allow use of other coordinate frames that are appropriate for those applications.

In general, several different Cartesian reference frames are used in the process of constructing and displaying a scene. First, we can define the shapes of individual objects, such as trees or furniture, within a separate coordinate reference frame for each object. These reference frames are called **modeling coordinates,** or sometimes **local coordinates** or **master coordinates.** Once the individual object shapes have been specified, we can construct ("model") a scene by placing the objects into appropriate locations within a scene reference frame called **world coordinates.** This step involves the transformation of the individual modeling-coordinate frames to specified positions and orientations within the world-coordinate frame. As an example, we could construct a bicycle by defining each of its parts (wheels, frame, seat, handle bars, gears, chain, pedals) in a separate modeling-coordinate frame. Then, the component parts are fitted together in world coordinates. If both bicycle wheels are the same size, we only need to describe one wheel in a local-coordinate frame. Then the wheel description is fitted into the world-coordinate bicycle description in two places. For scenes that are not too complicated, object components can be set up directly within the overall world-coordinate object structure, bypassing the modeling-coordinate and modeling-transformation steps. Geometric descriptions in modeling coordinates and world coordinates can be given in any convenient floating-point or integer values, without regard for the constraints of a particular output device. For some scenes, we might want to specify object geometries in fractions of a foot, while for other applications we might want to use millimeters, or 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 frames for display. This process is called the **viewing pipeline.** World-coordinate positions are first converted to *viewing coordinates* corresponding to the view we want of a scene, based on the position and orientation of a hypothetical camera. Then object locations are transformed to a two-dimensional projection of the scene, which corresponds to what we will see on the output device. The scene is then stored in **normalized coordinates,** where each coordinate value is in the range from -1 to 1 or in the range from 0 to 1, depending on the system. Normalized coordinates are also referred to as *normalized device coordinates,* since using this representation makes a graphics package independent of the coordinate range for any specific output device. We also need to identify visible surfaces and eliminate picture parts outside of the bounds for the view we want to show on the display device. Finally, the picture is scan converted into the refresh buffer of a raster system for display. The coordinate systems for display devices are generally called **device coordinates,** or **screen coordinates** in the case of a video monitor. Often, both normalized coordinates and screen coordinates are specified in a left-handed coordinate reference frame so that increasing positive distances from the xy plane (the screen, or viewing plane) can be interpreted as being farther from the viewing position.

hearn-50265; ISBN: 0-13-015390-7 book July 30, 2003 15:15

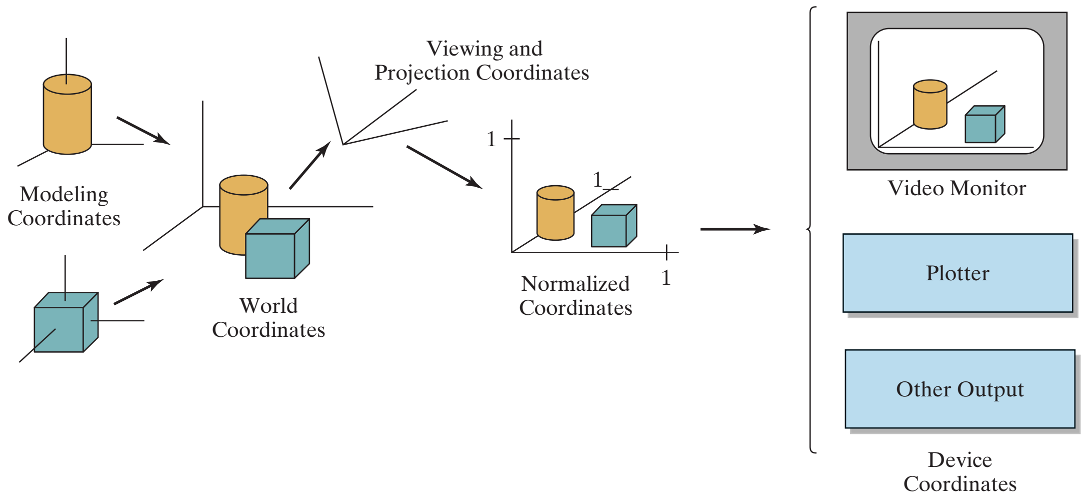

FIGURE 2-60 The transformation sequence from modeling coordinates to device coordinates for a three-dimensional scene. Object shapes can be individually defined in modeling-coordinate reference systems. Then the shapes are positioned within the world-coordinate scene. Next, world-coordinate specifications are transformed through the viewing pipeline to viewing and projection coordinates and then to normalized coordinates. At the final step, individual device drivers transfer the normalized-coordinate representation of the scene to the output devices for display.

Figure 2-60 briefly illustrates the sequence of coordinate transformations from modeling coordinates to device coordinates for a display that is to contain a view of two three-dimensional objects. An initial modeling-coordinate position (x_{mc}, y_{mc}, z_{mc}) in this illustration is transferred to world coordinates, then to viewing and projection coordinates, then to left-handed normalized coordinates, and finally to a device-coordinate position (x_{dc}, y_{dc}) with the sequence:

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

Device coordinates (x_{dc}, y_{dc}) are integers within the range (0, 0) to $(x_{\max}, y_{\max})$ for a particular output device. In addition to the two-dimensional positions (x_{dc}, y_{dc}) on the viewing surface, depth information for each device-coordinate position is stored for use in various visibility and surface-processing algorithms.

Graphics Functions

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

The basic building blocks for pictures are referred to as **graphics output primitives.** They include character strings and geometric entities, such as points, straight lines, curved lines, filled color areas (usually polygons), and shapes defined with arrays of color points. Additionally, some graphics packages provide functions for displaying more complex shapes such as spheres, cones, and cylinders. Routines for generating output primitives provide the basic tools for constructing pictures.

hearn-50265; ISBN: 0-13-015390-7 book July 29, 2003 14:54

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

We can change the size, position, or orientation of an object within a scene using **geometric transformations.** Some graphics packages provide an additional set of functions for performing **modeling transformations,** which are used to construct a scene where individual object descriptions are given in local coordinates. Such packages usually provide a mechanism for describing complex objects (such as an electrical circuit or a bicycle) with a tree (hierarchical) structure. Other packages simply provide the geometric-transformation routines and leave modeling details to the programmer.

After a scene has been constructed, using the routines for specifying the object shapes and their attributes, a graphics package projects a view of the picture onto an output device. **Viewing transformations** are used to select a view of the scene, the type of projection to be 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.

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

Some graphics packages also provide routines for subdividing a picture description into a named set of component parts. And other routines may be available for manipulating these picture components in various ways.

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 be 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**) in 1984. 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 soon 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 modeling, color specifications, surface rendering, and picture manipulations are provided in PHIGS. Subsequently, an extension of PHIGS, called PHIGS+, was developed to provide three-dimensional surface-rendering capabilities not available in PHIGS.

hearn-50265; ISBN: 0-13-015390-7 book July 29, 2003 14:54

As the GKS and PHIGS packages were being developed, the graphics workstations from Silicon Graphics, Inc. (SGI) became increasingly popular. These workstations came with a set of routines called **GL** (**Graphics Library**), which very soon became a widely used package in the graphics community. Thus GL 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 hardware systems. As a result, OpenGL was developed as a hardware-independent version of GL in the early 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. The OpenGL library is specifically designed for efficient processing of three-dimensional applications, but it can also handle two-dimensional scene descriptions as a special case of three dimensions where all the z coordinate values are 0.

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 the C and C++ languages are the same. Other OpenGL bindings are also available, such as those for Ada and Fortran.

In the following chapters, we use the C/C++ binding for OpenGL as a framework for discussing basic graphics concepts and the design and application of graphics packages. Example programs in C++ illustrate applications of OpenGL and the general algorithms for implementing graphics functions.

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*), which began as a subset of Open Inventor, 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. With *Java 2D,* we can create two-dimensional scenes within Java applets, for example. Or we can produce three-dimensional web displays with *Java 3D*. And with 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.

2-9 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. As we noted in the last section, OpenGL is designed to be hardware independent, therefore many operations, such as input and output routines,

hearn-50265; ISBN: 0-13-015390-7 book July 29, 2003 14:54

are not included in the basic library. However, input and output routines and many additional functions are available in auxiliary libraries that have been developed for OpenGL programs.

Basic OpenGL Syntax

Function names in the **OpenGL basic library** (also called the **OpenGL core library**) are prefixed with `gl`, and each component word within a function name has its first letter capitalized. The following examples illustrate this naming convention.

```
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 the uppercase letters GL. In addition, component words within a constant name are written in capital letters, and the underscore (_) is used as a separator between all component words in the name. Following are a few examples of the several hundred symbolic constants available for use with OpenGL functions.

```
GL_2D,  GL_RGB,  GL_CCW,  GL_POLYGON,  GL_AMBIENT_AND_DIFFUSE
```

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

```
GLbyte,  GLshort,  GLint,  GLfloat,  GLdouble,  GLboolean
```

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.

Some arguments of OpenGL functions can be assigned values using an array that lists a set of data values. This is an option for specifying a list of values as a pointer to an array, rather than specifying each element of the list explicitly as a parameter argument. A typical example of the use of this option is in specifying *xyz* coordinate values.

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 the surface-rendering operations, and other complex tasks. Every OpenGL implementation includes the GLU library, and all GLU function names start with the prefix `glu`. There is also an object-oriented toolkit based on OpenGL, called **Open Inventor,** which provides routines and predefined object shapes for interactive three-dimensional applications. This toolkit is written in C++.

To create a graphics display using OpenGL, we first need to set up a **display window** on our video screen. This is simply 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. However, there are several window-system libraries that support OpenGL functions for a variety of machines. 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 with `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. And the **OpenGL Utility Toolkit** (**GLUT**) provides a library of functions for interacting with any screen-windowing system. The GLUT library functions are prefixed with `glut`, and this library also contains methods for describing and rendering quadric curves and surfaces.

Since GLUT is an interface to other device-specific window systems, we can use GLUT so that our programs will be device independent. Information regarding the latest version of GLUT and download procedures for the source code are available at the Web site:

```
http://reality.sgi.com/opengl/glut3/glut3.html
```

Header Files

In all of our graphics programs, we will need to include the header file for the OpenGL core library. For most applications we will 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 would 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>
```

We could include `gl.h` and `glu.h` as well, but doing so would be redundant and could affect program portability.

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

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

With the new ISO/ANSI standard for C++, these header files are called `cstdio`, `cstdlib`, and `cmath`.

Display-Window Management Using GLUT

To get started, we can consider a simplified, minimal number of operations for displaying a picture. Since we are using the OpenGL Utility Toolkit, our first step is to initialize GLUT. This initialization function could also process any command-line arguments, but we will not need to use these parameters for our first example programs. We perform the GLUT initialization with the statement

```
glutInit (&argc, argv);
```

Next, we can state that a display window is to be created on the screen with a given caption for the title bar. This is accomplished with the function

```
glutCreateWindow ("An Example OpenGL Program");
```

where the single argument for this function can be any character string 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);
```

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

```
glutMainLoop ( );
```

This function must be the last one in our program. It displays the initial graphics and puts the program into an infinite loop that checks for input from devices such as a mouse or keyboard. Our first example will not be interactive, so the program will just continue to display our picture until we close the display window. In later chapters, we consider how we can modify our OpenGL programs to handle interactive input.

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. This 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 of the display window should be placed 50 pixels to the right of the left edge of the screen and 100 pixels down from the top edge of the screen.

```
glutInitWindowPosition (50, 100);
```

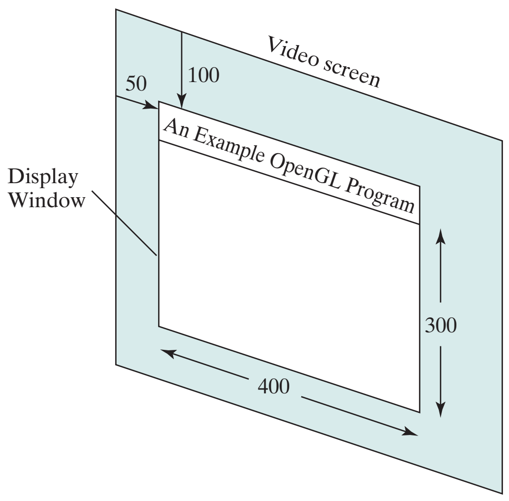

FIGURE 2-61 A 400 by 300 display window at position (50, 100) relative to the top-left corner of the video display.

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 (Fig. 2-61) 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 and 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_SINGLE | GLUT_RGB);
```

The values of the constants passed to this function are combined using a logical *or* operation. Actually, single buffering and RGB color mode are the default options. But we will use the function now as a reminder that these are the options that are set for our display. Later, we discuss color modes in more detail, as well as other display options such as double buffering for animation applications and selecting parameters for viewing three-dimensional scenes.

A Complete OpenGL Program

There are still a few more tasks to perform before we have all the parts we need for a complete program. For the display window, we can choose a background color. And we need to construct a procedure that contains the appropriate OpenGL functions for the picture that we want to display.

Using RGB color values, we set the background color for the display window to be white, as in Fig. 2-61, with the OpenGL function

```
glClearColor (1.0, 1.0, 1.0, 0.0);
```

hearn-50265; ISBN: 0-13-015390-7 book July 29, 2003 14:54

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. And 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. One use for the alpha value is as a "blending" parameter. When we activate the OpenGL blending operations, alpha values can be used to determine the resulting color for two overlapping objects. An alpha value of 0.0 indicates a totally transparent object, and an alpha value of 1.0 indicates an opaque object. Blending operations will not be used for a while, so the value of alpha is irrelevant to our early example programs. For now, we simply set alpha to 0.0.

Although the `glClearColor` command assigns a color to the display window, it does not put the display window on the screen. To get the assigned window color displayed, we need to invoke the following OpenGL function.

```
glClear (GL_COLOR_BUFFER_BIT);
```

The argument `GL_COLOR_BUFFER_BIT` is an OpenGL symbolic constant specifying that it is the bit values in the color buffer (refresh buffer) that are to be set to the values indicated in the `glClearColor` function. (We discuss other buffers in later chapters.)

In addition to setting the background color for the display window, we can choose a variety of color schemes for the objects we want to display in a scene. For our initial programming example, we will simply set object color to be red and defer further discussion of the various color options until Chapter 4:

```
glColor3f (1.0, 0.0, 0.0);
```

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 = blue = 0.0.

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

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

This specifies that an orthogonal projection is to be used to map the contents of a two-dimensional (2D) rectangular area of world coordinates to the screen, and that the x-coordinate values within this rectangle range from 0.0 to 200.0 with y-coordinate values ranging from 0.0 to 150.0. Whatever objects we define within this world-coordinate rectangle will be shown within the display window. Anything outside this coordinate range will not be displayed. Therefore, the GLU function `gluOrtho2D` defines the coordinate reference frame within the

hearn-50265; ISBN: 0-13-015390-7 book July 29, 2003 15:55

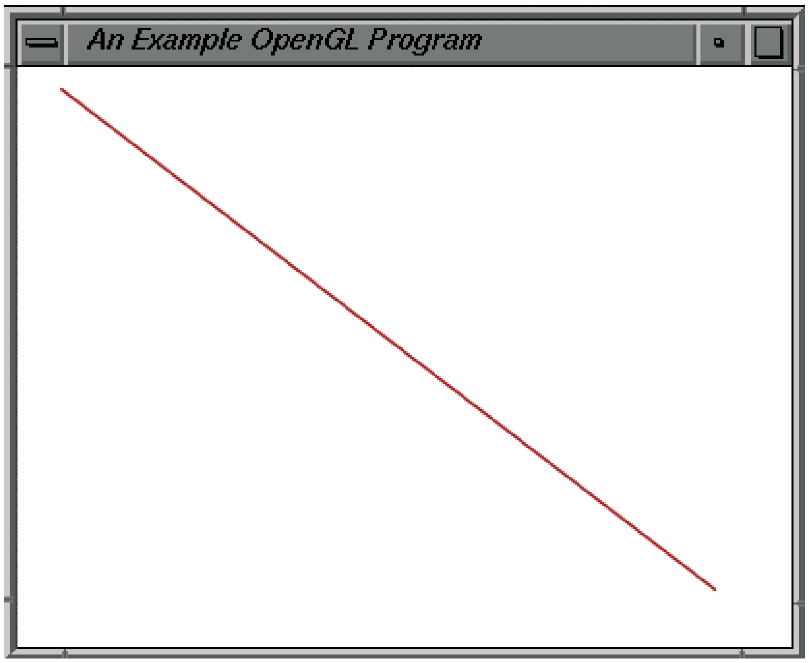

FIGURE 2-62 The display window and line segment produced by the example program.

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. Since we are only describing a two-dimensional object, the orthogonal projection has no other effect than to "paste" our picture into the display window that we defined earlier. For now, we will use a world-coordinate rectangle with the same aspect ratio as the display window, so that there is no distortion of our picture. Later, we will consider how we can maintain an aspect ratio that is not dependent upon the display-window specification.

Finally, we need to call the appropriate OpenGL routines to create our line segment. The following code defines a two-dimensional, straight-line segment with integer, Cartesian endpoint coordinates (180, 15) and (10, 145). In Chapter 3, we present a detailed explanation of these functions and the other OpenGL functions for generating graphics primitives.

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

Now we are ready to put all the pieces together. The following OpenGL program is organized into three procedures. We place all initializations and related one-time parameter settings in procedure `init`. Our geometric description of the "picture" we want to display is in procedure `lineSegment`, which is the procedure that will be referenced by the GLUT function `glutDisplayFunc`. And the `main` procedure contains the GLUT functions for setting up the display window and getting our line segment onto the screen. Figure 2-62 shows the display window and red line segment generated by this program.

hearn-50265; ISBN: 0-13-015390-7 book July 29, 2003 14:54

```
#include <GL/glut.h>      // (or others, depending on the system in use)

void init (void)
{
    glClearColor (1.0, 1.0, 1.0, 0.0);  // Set display-window color to white.

    glMatrixMode (GL_PROJECTION);       // Set projection parameters.
    gluOrtho2D (0.0, 200.0, 0.0, 150.0);
}

void lineSegment (void)
{
    glClear (GL_COLOR_BUFFER_BIT);  // Clear display window.

    glColor3f (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.
}

void main (int argc, char** argv)
{
    glutInit (&argc, argv);                         // Initialize GLUT.
    glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);   // Set display mode.
    glutInitWindowPosition (50, 100);   // Set top-left display-window position.
    glutInitWindowSize (400, 300);      // Set display-window width and height.
    glutCreateWindow ("An Example OpenGL Program"); // Create display window.

    init ( );                            // Execute initialization procedure.
    glutDisplayFunc (lineSegment);       // Send graphics to display window.
    glutMainLoop ( );                    // Display everything and wait.
}
```

At the end of procedure `lineSegment` is a function, `glFlush`, that we have not yet discussed. This is simply a routine to force execution of our OpenGL functions, which are stored by computer systems in buffers in different locations, depending on how OpenGL is implemented. On a busy network, for example, there could be delays in processing some buffers. But the call to `glFlush` forces all such buffers to be emptied and the OpenGL functions to be processed.

The procedure `lineSegment` that we set up to describe our picture is referred to as a *display callback function*. And this procedure is described as being "registered" by `glutDisplayFunc` as the routine to invoke whenever the display window might need to be redisplayed. This can occur, for example, if the display window is moved. In subsequent chapters we will take a look at other types of callback functions and the associated GLUT routines that we use to register them. In general, OpenGL programs are organized as a set of callback functions that are to be invoked when certain actions occur.

hearn-50265; ISBN: 0-13-015390-7 book July 29, 2003 14:54

2-10 SUMMARY

In this introductory chapter, we surveyed the major hardware and software features of computer-graphics systems. Hardware components include video monitors, hardcopy output devices, various kinds of input devices, and components for interacting with virtual environments. Some software systems, such as CAD packages and paint programs, are designed for particular applications. Other software systems provide a library of general graphics routines that can be used within a programming language such as C++ to generate pictures for any application.

The predominant graphics display device is the raster refresh monitor, based on television technology. A raster system uses a frame buffer to store the color value for each screen position (pixel). Pictures are then painted onto the screen by retrieving this information from the frame buffer (also called a refresh buffer) as the electron beam in the CRT sweeps across each scan line, from top to bottom. Older vector displays construct pictures by drawing straight-line segments between specified endpoint positions. Picture information is then stored as a set of line-drawing instructions.

Many other video display devices are available. In particular, flat-panel display technology is developing at a rapid rate, and these devices are now used in a variety of systems, including both desktop and laptop computers. Plasma panels and liquid-crystal devices are two examples of flat-panel displays. Other display technologies include three-dimensional and stereoscopic-viewing systems. Virtual-reality systems can include either a stereoscopic headset or a standard video monitor.

For graphical input, we have a range of devices to choose from. Keyboards, button boxes, and dials are used to input text, data values, or programming options. The most popular "pointing" device is the mouse, but trackballs, spaceballs, joysticks, cursor-control keys, and thumbwheels are also used to position the screen cursor. In virtual-reality environments, data gloves are commonly used. Other input devices are image scanners, digitizers, touch panels, light pens, and voice systems.

Hardcopy devices for graphics workstations include standard printers and plotters, in addition to devices for producing slides, transparencies, and film output. Printers produce hardcopy output using dot-matrix, laser, inkjet, electrostatic, or electrothermal methods. Graphs and charts can be produced with an ink-pen plotter or with a combination printer-plotter device.

Standard graphics-programming packages developed and approved through ISO and ANSI are GKS, 3D GKS, PHIGS, and PHIGS+. Other packages that have evolved into standards are GL and OpenGL. Many other graphics libraries are available for use in a programming language, including Open Inventor, VRML, RenderMan, Java 2D, and Java 3D. Other systems, such as Mathematica, MatLab, and Maple, often provide a set of graphics-programming functions.

Normally, graphics-programming packages require coordinate specifications to be given in Cartesian reference frames. Each object for a scene can be defined in a separate modeling Cartesian-coordinate system, which is then mapped to a world-coordinate location to construct the scene. From world coordinates, three-dimensional objects are projected to a two-dimensional plane, converted to normalized device coordinates, and then transformed to the final display-device coordinates. The transformations from modeling coordinates to normalized device coordinates are independent of particular output devices that might be used in an application. Device drivers are then used to convert normalized coordinates to integer device coordinates.

Functions that are available in graphics programming packages can be divided into the following categories: graphics output primitives, attributes, geometric and modeling transformations, viewing transformations, input functions, picture-structuring operations, and control operations.

The OpenGL system consists of a device-independent set of routines (called the core library), the utility library (GLU), and the utility toolkit (GLUT). In the auxiliary set of routines provided by GLU, functions are available for generating complex objects, for parameter specifications in two-dimensional viewing applications, for dealing with surface-rendering operations, and for performing some other supporting tasks. In GLUT, we have an extensive set of functions for managing display windows, interacting with screen-window systems, and for generating some three-dimensional shapes. We can use GLUT to interface with any computer system, or we can use GLX, Apple GL, WGL, or another system-specific software package.

REFERENCES

A general treatment of electronic displays is available in Tannas (1985) and in Sherr (1993). Flat-panel devices are discussed in Depp and Howard (1993). Additional information on raster-graphics architecture can be found in Foley, van Dam, Feiner, and Hughes (1990). Three-dimensional and stereoscopic displays are discussed in Johnson (1982) and in Grotch (1983). Head-mounted displays and virtual-reality environments are discussed in Chung, et al. (1989).

Standard sources for information on OpenGL are Woo, Neider, Davis, and Shreiner (1999) and Shreiner (2000). Open Inventor is explored in Wernecke (1994). McCarthy and Descartes (1998) can be consulted for discussions of VRML. A presentation on RenderMan can be found in Upstill (1989). Examples of graphics programming in Java 2D are given in Knudsen (1999), Hardy (2000), and Horstmann and Cornell (2001). Graphics programming using Java 3D is explored in Sowizral, Rushforth, and Deering (2000); Palmer (2001); Selman (2002); and Walsh and Gehringer (2002).

For information on PHIGS and PHIGS+, see Howard, Hewitt, Hubbold, and Wyrwas (1991); Hopgood and Duce (1991); Gaskins (1992); and Blake (1993). Information on the two-dimensional GKS standard and on the evolution of graphics standards is available in Hopgood, Duce, Gallop, and Sutcliffe (1983). An additional reference for GKS is Enderle, Kansy, and Pfaff (1984).

EXERCISES

2-1 List the operating characteristics for the following display technologies: raster refresh systems, vector refresh systems, plasma panels, and LCDs.

2-2 List some applications appropriate for each of the display technologies in Exercise 2-1.

2-3 Determine the resolution (pixels per centimeter) in the x and y directions for the video monitor in use on your system. Determine the aspect ratio, and explain how relative proportions of objects can be maintained on your system.

2-4 Consider three different raster systems with resolutions of 640 by 480, 1280 by 1024, and 2560 by 2048. What size frame buffer (in bytes) is needed for each of these systems to store 12 bits per pixel? How much storage is required for each system if 24 bits per pixel are to be stored?

2-5 Suppose an RGB raster system is to be designed using an 8 inch by 10 inch screen with a resolution of 100 pixels per inch in each direction. If we want to store 6 bits per pixel in the frame buffer, how much storage (in bytes) do we need for the frame buffer?

2-6 How long would it take to load a 640-by-480 frame buffer with 12 bits per pixel, if 10^5 bits can be transferred per second? How long would it take to load a 24-bit-per-pixel frame buffer with a resolution of 1280 by 1024 using this same transfer rate?

2-7 Suppose we have a computer with 32 bits per word and a transfer rate of 1 mip (one million instructions per second). How long would it take to fill the frame buffer of a 300 dpi (dot per inch) laser printer with a page size of $8\,^1/_2$ inches by 11 inches?

2-8 Consider two raster systems with resolutions of 640 by 480 and 1280 by 1024. How many pixels could be accessed per second in each of these systems by a display controller that refreshes the screen at a rate of 60 frames per second? What is the access time per pixel in each system?

2-9 Suppose we have a video monitor with a display area that measures 12 inches across and 9.6 inches high. If the resolution is 1280 by 1024 and the aspect ratio is 1, what is the diameter of each screen point?

2-10 How much time is spent scanning across each row of pixels during screen refresh on a raster system with a resolution of 1280 by 1024 and a refresh rate of 60 frames per second?

2-11 Consider a noninterlaced raster monitor with a resolution of n by m (m scan lines and n pixels per scan line), a refresh rate of r frames per second, a horizontal retrace time of t_{horiz}, and a vertical retrace time of t_{vert}. What is the fraction of the total refresh time per frame spent in retrace of the electron beam?

2-12 What is the fraction of the total refresh time per frame spent in retrace of the electron beam for a noninterlaced raster system with a resolution of 1280 by 1024, a refresh rate of 60 Hz, a horizontal retrace time of 5 microseconds, and a vertical retrace time of 500 microseconds?

2-13 Assuming that a certain full-color (24-bit-per-pixel) RGB raster system has a 512-by-512 frame buffer, how many distinct color choices (intensity levels) would we have available? How many different colors could we display at any one time?

2-14 Compare the advantages and disadvantages of a three-dimensional monitor using a varifocal mirror to those of a stereoscopic system.

2-15 List the different input and output components that are typically used with virtual-reality systems. Also, explain how users interact with a virtual scene displayed with different output devices, such as two-dimensional and stereoscopic monitors.

2-16 Explain how virtual-reality systems can be used in design applications. What are some other applications for virtual-reality systems?

2-17 List some applications for large-screen displays.

2-18 Explain the differences between a general graphics system designed for a programmer and one designed for a specific application, such as architectural design.

2-19 Explain the differences between the OpenGL core library, the OpenGL Utility, and the OpenGL Utility Toolkit.

2-20 What command could we use to set the color of an OpenGL display window to light gray? What command would we use to set the color of the display window to black?

2-21 List the statements needed to set up an OpenGL display window whose lower-right corner is at pixel position (200, 200), with a window width of 100 pixels and a height of 75 pixels.

2-22 Explain what is meant by the term "OpenGL display callback function".