SPCA 202

MASTER OF COMPUTER APPLICATIONS

SECOND YEAR

THIRD SEMESTER

CORE PAPER - XII

COMPUTER GRAPHICS

INSTITUTE OF DISTANCE EDUCATION

UNIVERSITY OF MADRAS

MASTER OF COMPUTER APPLICATIONS
SECOND YEAR - THIRD SEMESTER

CORE PAPER - XII
COMPUTER GRAPHICS

WELCOME

Warm Greetings.

It is with a great pleasure to welcome you as a student of Institute of Distance Education, University of Madras. It is a proud moment for the Institute of Distance education as you are entering into a cafeteria system of learning process as envisaged by the University Grants Commission. Yes, we have framed and introduced Choice Based Credit System(CBCS) in Semester pattern from the academic year 2018-19. You are free to choose courses, as per the Regulations, to attain the target of total number of credits set for each course and also each degree programme. What is a credit? To earn one credit in a semester you have to spend 30 hours of learning process. Each course has a weightage in terms of credits. Credits are assigned by taking into account of its level of subject content. For instance, if one particular course or paper has 4 credits then you have to spend 120 hours of self-learning in a semester. You are advised to plan the strategy to devote hours of self-study in the learning process. You will be assessed periodically by means of tests, assignments and quizzes either in class room or laboratory or field work. In the case of PG (UG), Continuous Internal Assessment for 20(25) percentage and End Semester University Examination for 80 (75) percentage of the maximum score for a course / paper. The theory paper in the end semester examination will bring out your various skills: namely basic knowledge about subject, memory recall, application, analysis, comprehension and descriptive writing. We will always have in mind while training you in conducting experiments, analyzing the performance during laboratory work, and observing the outcomes to bring out the truth from the experiment, and we measure these skills in the end semester examination. You will be guided by well experienced faculty.

I invite you to join the CBCS in Semester System to gain rich knowledge leisurely at your will and wish. Choose the right courses at right times so as to erect your flag of success. We always encourage and enlighten to excel and empower. We are the cross bearers to make you a torch bearer to have a bright future.

With best wishes from mind and heart,

DIRECTOR

MASTER OF COMPUTER APPLICATIONS
SECOND YEAR - THIRD SEMESTER

CORE PAPER - XII
COMPUTER GRAPHICS

COURSE WRITER

Dr. Ramesh. K
Department of Computer Applications
Anna University Regional Campus
Tirunelveli - 627007

CO-ORDINATOR

Dr. S. Sasikala
Assistant Professor in Computer Science
Institute of Distance Education
University of Madras
Chepauk Chennnai - 600 005.

MASTER OF COMPUTER APPLICATIONS

SECOND YEAR

THIRD SEMESTER

Core Paper - XII

COMPUTER GRAPHICS

SYLLABUS

Unit 1:

Introduction to computer Graphics – Video display devices – Raster Scan Systems – Random Scan Systems - Interactive input devices – Hard Copy devices - Graphics software –Area fill attributes – Character attributes inquiry function - Output primitives – line drawing algorithms – initializing lines – line function – Circle Generating algorithms – Ellipse Generating algorithms - Attributes of output primitives – line attributes – Color and Grayscale style.

Unit 2:

Two dimensional transformation – Basic transformation – Matrix representation and Homogeneous co-ordinates - Composite transformation – Matrix representation – other transformations – two dimensional viewing – window – to- viewport co-ordinate transformation.

Unit 3

Clipping algorithms – Point clipping -line clipping - polygon clipping – Curve clipping - text clipping – Exterior clipping –– Three dimensional transformations – translation- rotation- scaling – composite-shears and reflections - Three dimensional viewing – Projection – Orthogonal and oblique parallel projections.

Unit 4:

Viewing - perspective projection – Three dimensional clipping algorithms- Visible surface detection methods — backface detection, depth buffer, A-buffer, scan-line, depth sorting, BSP-tree, area subdivision, octree and other methods.

Unit 5 :

Computer Animation - Three dimensional object representations – Spline representation - Bezier curves and surfaces – B-Spline curves and surfaces –– Color models and color applications.

Recommended Text:

1) D. Hearn, M.P. Baker, and W.R. Carithers, 2011 – Computer Graphics with openGL, 4th Edition, Pearson Education

Reference Books:

1) W.M. Neumann and R. F. Sproull, Principles of Interactive Computer Graphics, Tata McGraw-Hill, New Delhi.

2) S. Harrington, 1989, Fundamentals of Computer Graphics, Tata McGraw-Hill, New Delhi.

3) D. F. Rogers, J. A. Adams, 2002, Mathematical elements for Computer Graphics, 2nd Edition, Tata McGraw-Hill, New Delhi.

4) D. F. Rogers, 2001, Procedural elements for Computer Graphics, 2nd Edition, Tata McGraw-Hill, New Delhi.

5) Foley, Van Dan, Feiner, Hughes, 2000, Computer Graphics, Addison Wesley, Boston

Website and E-Learning Source:

1) http://forum.jntuworld.com/showthread.php?3846-Computer-Graphics- Notes-All-8-Units

2) http://www.cs.kent.edu/~farrell/cg05/lectures/index.html

MASTER OF COMPUTER APPLICATIONS

SECOND YEAR

THIRD SEMESTER

Core Paper - XII

COMPUTER GRAPHICS

SCHEME OF LESSONS

LESSON -1
GRAPHICS SYSTEMS

Structure of the lesson

1.1 Introduction

Computer Graphics is the creation of pictures with the help of specialized graphical hardware and software. Computer Graphics is used in diverse area as science, engineering, medicine, business, industry, government, art, entertainment, advertisement, education and training. Computer graphics system uses a variety of interactive input devices, output devices and graphics software packages. Interactive input devices connected to a computer graphics system allow the user to design graphics by entering data and to interact with the application. Output devices are used to visualize the graphics outcomes. To create graphics varies software packages are available.

This lesson will familiarize you with the primary output devices with detail on CRT, random scan display, raster scan display, and other popular display mechanisms.

1.2 Objective of this Lesson

Students will gain knowledge and understanding about

- Video display devices
- CRT

- Random scan System
- Raster scan System and other popular display mechanisms.

1.3 Video Display Devices

A *display device* is an output *device* for presentation of information in *visual.* The primary output device in a graphics system is a video monitor. The operations of most video monitors are based on the standard cathode-ray-tube (CRT) design.

The following are the various video display devices:

- Refresh Cathode-Ray tubes
- Raster Scan displays
- Random scan displays
- Color CRT monitors
- Direct View storage tubes
- Flat panel displays
- Three Dimensional viewing devices
- Stereoscopic and virtual reality systems

Most video monitors are based on the standard cathode-ray tube (CRT) design. In recent years, flat-panel displays have become significantly more popular.

Refresh Cathode-Ray Tubes

The following section describes 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). 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 Figure 2, 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.

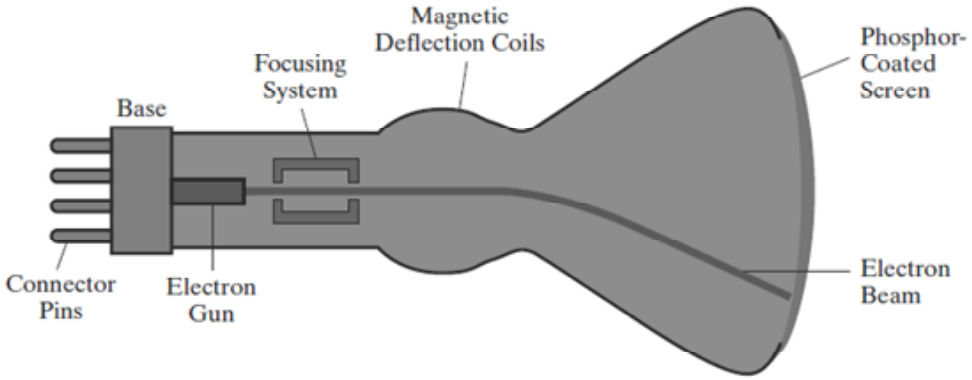


Figure 1: Basic design of a magnetic-deflection CRT.

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.

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 center line of the cylinder are in an equilibrium position. This arrangement forms an electrostatic lens, as shown in Figure 2, 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.

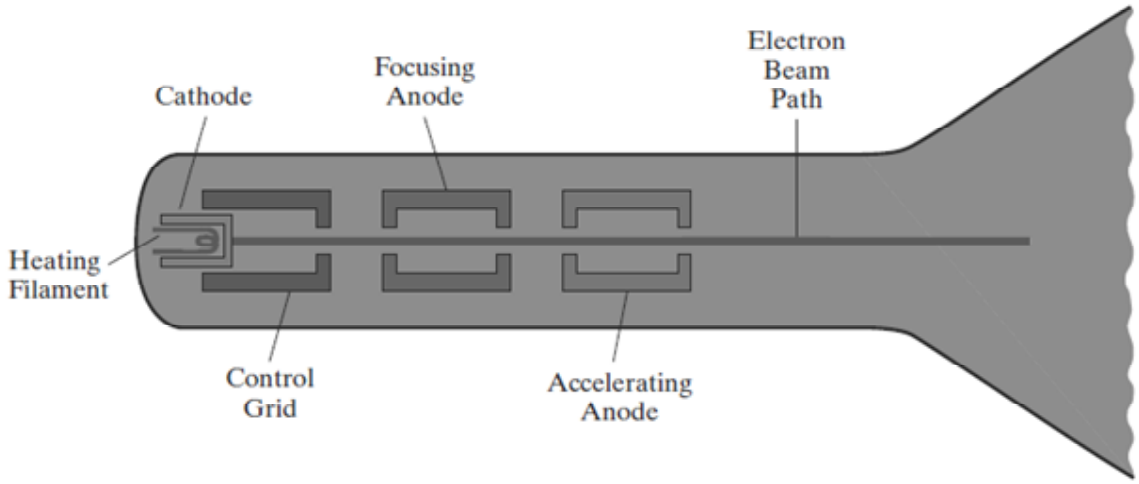


Figure 2: Operation of an electron gun with an accelerating anode.

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 Figure 1. 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. 3).

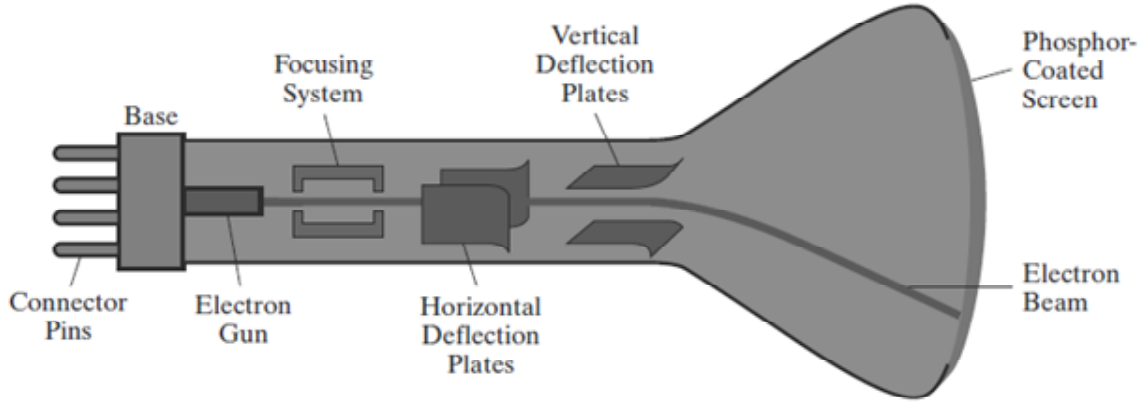


Figure 3: Electrostatic deflection of the electron beam in a CRT.

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 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 in proportion 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 it is 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.

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, 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. 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.

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.

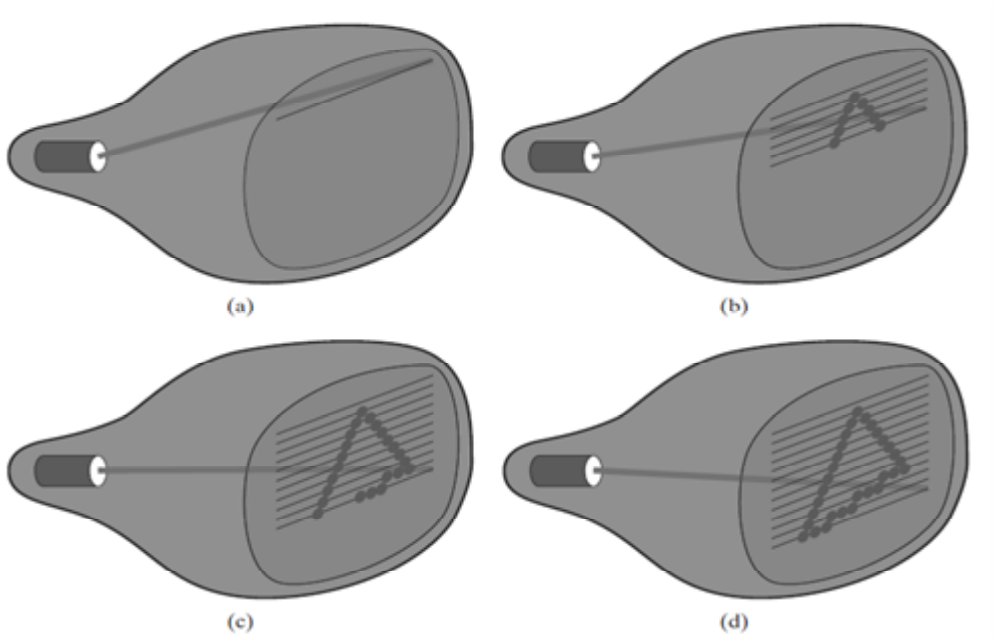


Figure 4: Raster-scan system displays

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 Figure 4. 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 this term 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 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.

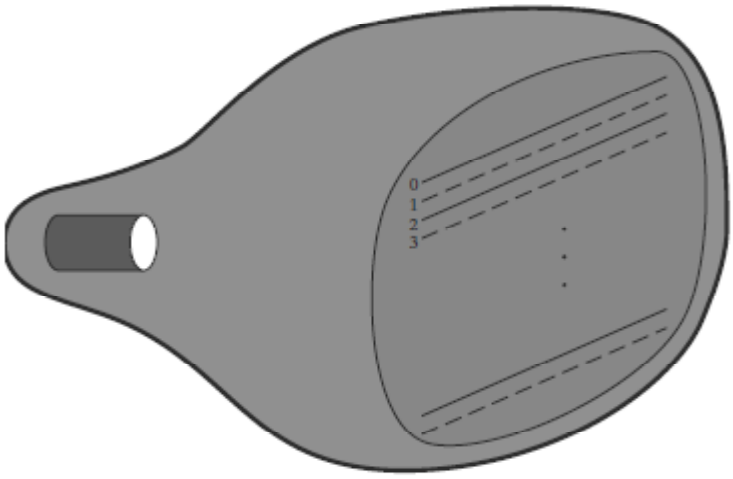


Figure 5: Interlacing scan lines on a raster-scan display.

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 MB 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. 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 these terms 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, so 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, the electron beam returns to the upper-left corner of the screen (vertical retrace) to begin the next frame.

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. 5). Interlacing of the scan lines in this way allows us to see the entire screen displayed in half the time that 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. 6). 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 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.

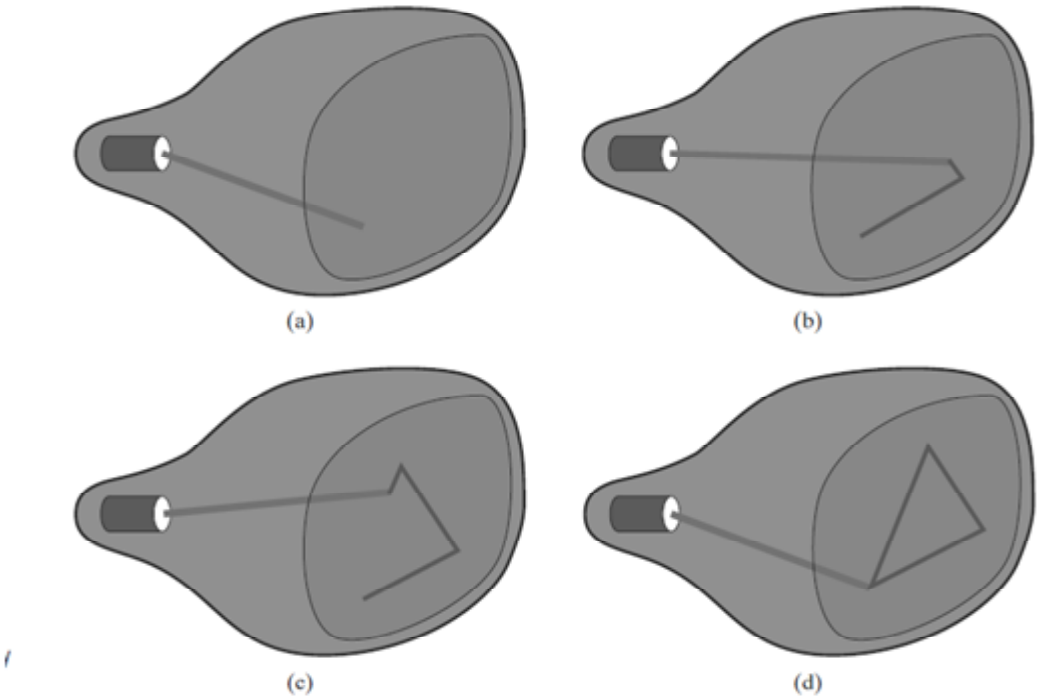


Figure 6: A random-scan system

Color CRT monitors

A CRT monitor displays color pictures by using a combination of phosphors that emit different-colored light.

Two basic techniques for producing color displays with CRT are

1) Beam penetration method

2) Shadow mask method

Beam penetration method

In beam penetration method the emitted color depends on how far the electron beam penetrates into the phosphor layers. Typical system uses only two phosphor layers: red and green. At intermediate beam speeds, combinations of red and green light are emitted to show two additional colors: orange and yellow. 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.

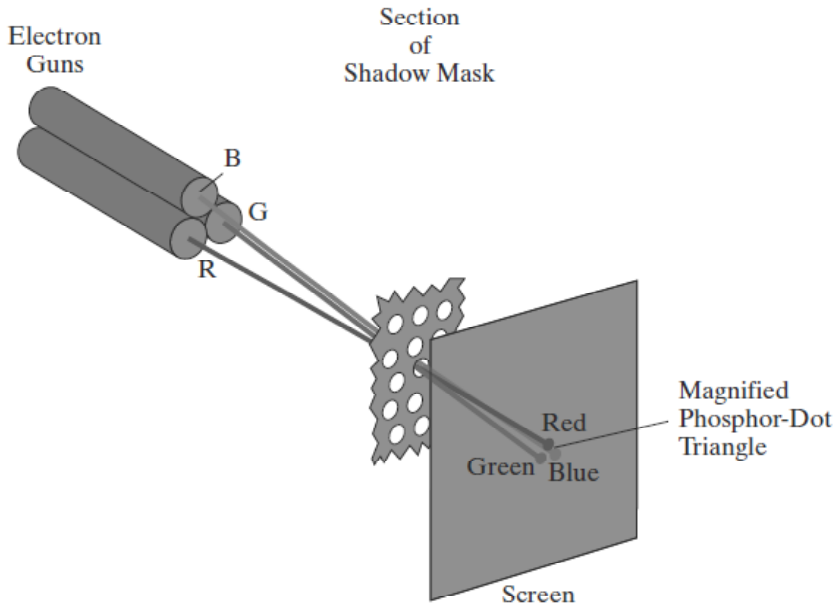


Figure 7: Operation of a delta-delta, shadow-mask CRT.

Shadow mask method

Shadow-mask methods are commonly used in raster-scan systems. This approach is based on the combinations of red, green, and blue components, called the RGB color model. 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 color dot, and a shadow-mask grid just behind the phosphor-coated screen. 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. Color variations in a shadow-mask CRT can be obtained by varying the intensity levels of the three electron beams. The color and intensity information of the picture must be combined and superimposed on the broadcast-frequency carrier signal. 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.

Direct View Storage Tubes (DVST)

In DVST the picture information is stored inside the CRT. It stores the picture information as a charge distribution just behind the phosphor coated screen. Two electron guns are used in DVST. The primary gun is used to store the picture pattern and the flood gun maintains the picture display detail.

Flat Panel Display:

Flat-panel display refers to a class of video devices that have reduced volume, weight, and power requirements compared to a CRT. Flat-panel displays are categories into two namely 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.

Nonemissive displays (or nonemitters) use optical effects to convert sunlight or light from some other source into graphics patterns. Liquid-crystal device is an non emissive display device. Based on the technique used flat panel display are further categories as

- Plasma panels
- Thin-film electroluminescent displays
- light-emitting diode (LED)
- Liquid-crystal displays (LCDs)

Plasma panels

Plasma panels, also called gas-discharge displays, are constructed by filling the region between two glass plates with a mixture of gases. 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. Firing voltages applied to an 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 60 times per second.

Thin-film electroluminescent displays

Thin-film electroluminescent displays are similar to plasma panels. The difference is that the region between the glass plates is filled with a phosphor instead of a gas. When a high voltage is applied to a pair of crossing electrodes, electrical energy is absorbed by the manganese atoms, which then release the energy as a spot of light.

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. Picture 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)

LCDs are nonemissive devices which produces 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. A flat-panel display can then be constructed with a nematic liquid crystal. Two glass plates, each containing a light polarizer is aligned at a right angle to each other. 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. Picture definitions are stored in a refresh buffer, and the screen is refreshed at the rate of 60 frames per second. Another method for constructing LCDs is to use thin-film transistor technology at each pixel position. 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. 8). 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. In addition to displaying three-dimensional images, these systems are often capable of displaying two-dimensional cross-sectional “slices” of objects selected at different depths, such as in medical applications to analyze data from ultrasonography and CAT scan devices, in geological applications to analyze topological and seismic data, in design applications involving solid objects, and in three-dimensional simulations of systems, such as molecules and terrain.

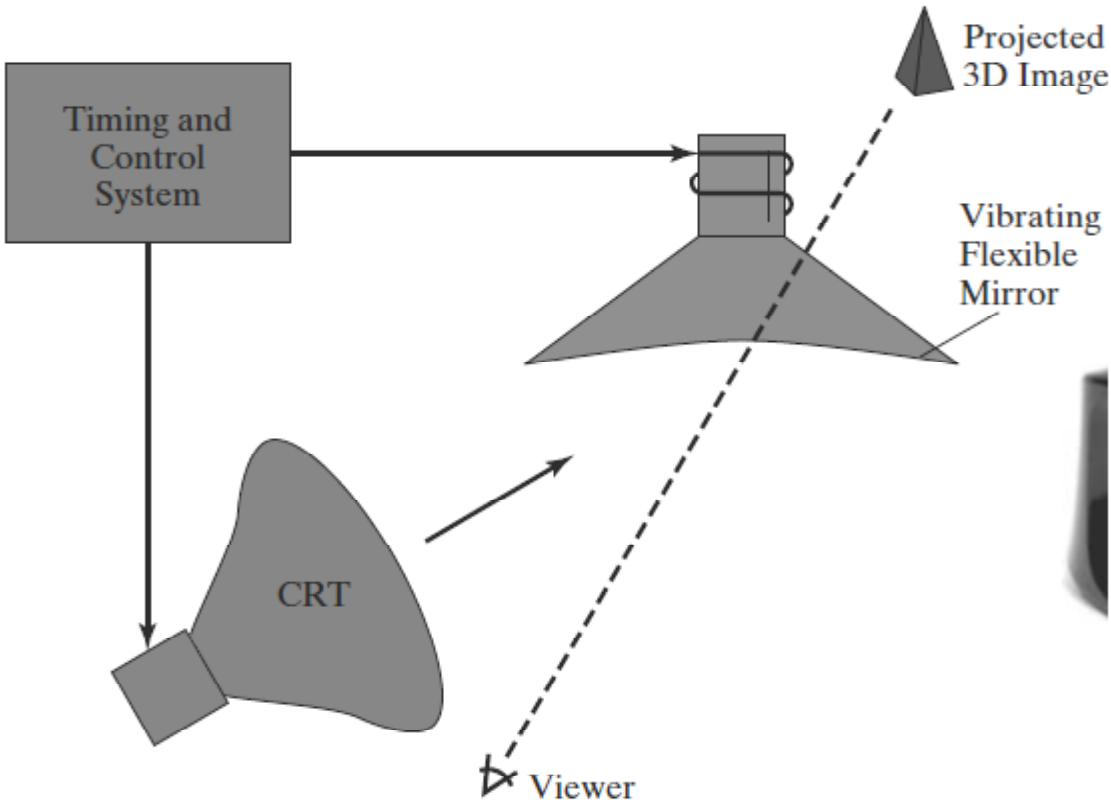


Figure 8: Operation of a three-dimensional display system using a vibrating mirror that changes focal length to match the depths of points in a scene.

Stereoscopic and virtual reality system

A stereoscopic view of the object is another technique for representing a three-dimensional object. Stereoscopic projection method does not produce true three dimensional images but it does provide a 3D effect by presenting a different view to each eye of the observer. To obtain a stereoscopic projection we need to obtain two views of a scene generated from a viewing direction corresponding to each eye. One way to produce a stereoscopic effect on a raster system is to display each of the two views on alternate refresh cycles.

Stereoscopic viewing is also a component in virtual-reality systems, Virtual Reality system is where users can step into a scene and interact with the environment. A headset containing

an optical system is used to locate and manipulate objects in the scene. The headset keeps track of the viewer's position and helps the viewer to "walks through" and interacts with the display.

1.4 Raster Scan Systems:

Raster scan system employs several processing units such as CPU, video controller, system memory, and monitor and I/O devices.

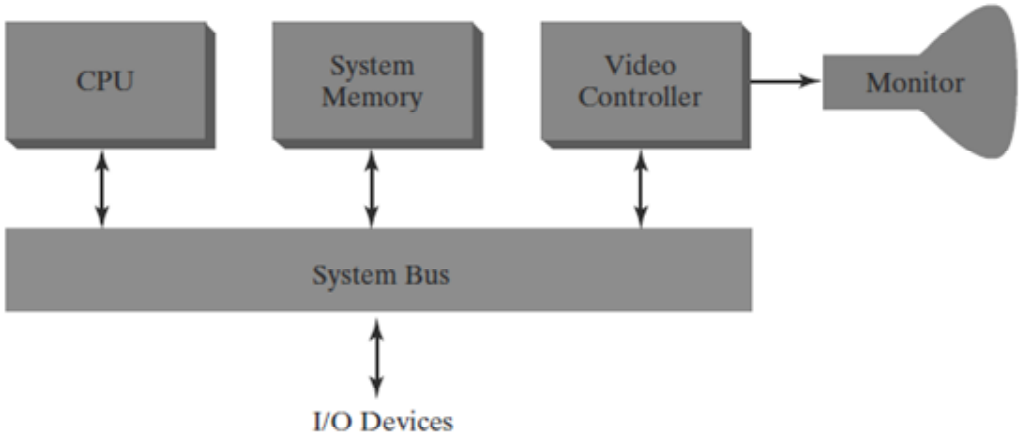


Figure 9: Architecture of a simple raster-graphics system.

Video controller:

A special-purpose processor, called the video controller or display controller**,** is used to control the operation of the display device. 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 location and the corresponding are referenced in Cartesian coordinates. Two registers are used to store the coordinate values for the screen pixels. The contents of the frame buffer at this pixel position are then retrieved and used to set the intensity of the CRT beam. Pixels along this scan line are then processed in turn, and the procedure is repeated for each successive scan line. The video controller resets the registers to the first pixel position on the top scan line and the refresh process starts over.

The basic refresh operations of the video controller are listed below. The video controller can retrieve pixel values from different memory areas on different refresh cycles. 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. The video controller 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.

Raster-Scan Display Processor

A raster system contains a separate display processor, also called as a graphics controller or a display coprocessor. The purpose of the display processor is to free the CPU from the graphics chores. A major task of the display processor is digitizing a picture definition into a set of pixel values for storage in the frame buffer. This digitization process is called scan conversion.

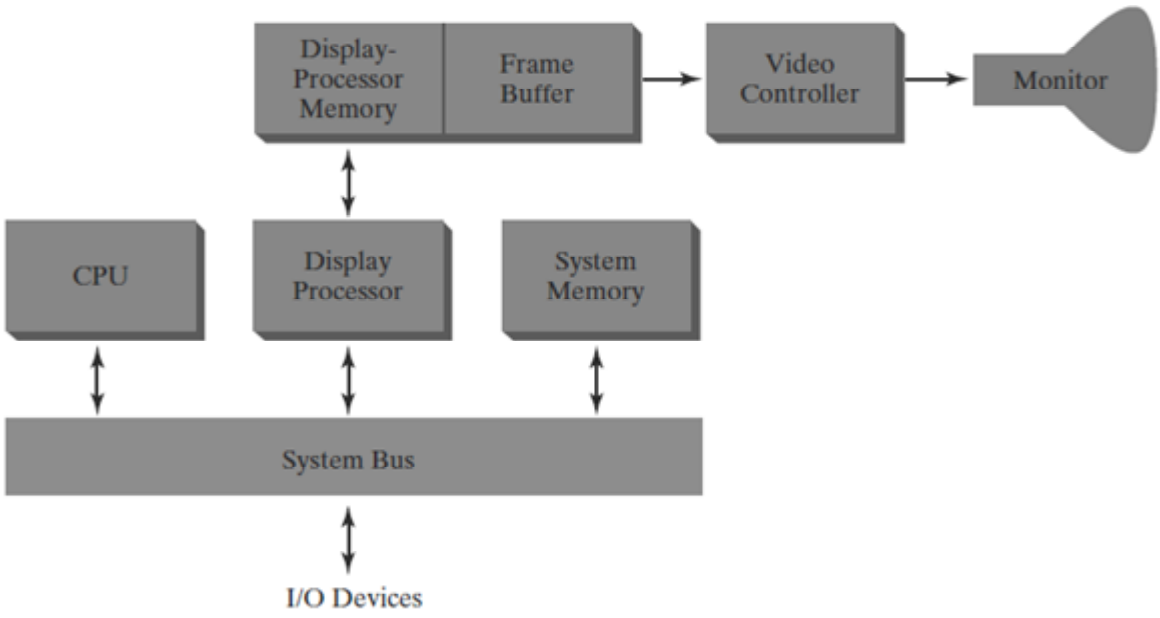


Figure 10: Architecture of a raster-graphics system with a display processor.

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.

1.5 Random Scan Systems

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

Refresh rate on a random-scan system depends on the number of lines to be displayed . Picture definition is now stored as a set of line-drawing commands in an area of memory referred to as the refresh display file. Sometimes the refresh display file is called the display list, display program, or simply the refresh buffer. To display a specified picture, the system cycles 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 al the component lines of a picture 30 to 60times each second.

1.6 Summary

In this lesson we have gain knowledge about video display devices and their working mechanism.

- A *display device* is an output *device.*
- Most video monitors are based on the standard cathode-ray tube (CRT) design.
- Raster scan system the electron beam is swept across the screen, one row at a time, from top to bottom.
- Two basic techniques for producing color displays with CRT are Beam penetration method and Shadow mask method.
- Raster scan system employs several processing units such as CPU, video controller, system memory, and monitor and I/O devices.
- A raster system contains a separate display processor, also called as a graphics controller or a display coprocessor. Raster scan system the electron beam is swept across the screen, one row at a time, from top to bottom.

1.7 Model Questions

1. Define Computer Graphics.
2. Define Raster scan displays.
3. Explain in detail about video display devices.
4. Explain in detail about graphics input devices.
5. Explain in detail about raster and random scan systems.
6. Explain refresh cathode ray tube.

LESSON – 2
INTERACTIVE INPUT AND HARD COPY DEVICES

Structure of the Lesson

2.1 Introduction

Graphics programs use several kinds of input data, such as coordinate positions, attribute values, character-string specifications, geometric-transformation values, viewing conditions, and illumination parameters. Many graphics packages, including the International Standards Organization (ISO) and American National Standards Institute (ANSI) standards, provide an extensive set of input functions for processing such data. But input procedures require interaction with display-window managers and specific hardware devices. Therefore, some graphics systems, particularly those that provide mainly device-independent functions, often include relatively few interactive procedures for dealing with input data.

To receive data inputs graphics workstations uses various interactive input devices. We can use a wide variety of interactive input devices and graphics software packages. Selection of input devices depends on the application. In this chapter we explore the basic features of interactive input devices and graphics software packages.

2.2 Objective of this Lesson

At the end of the reading the student will get idea about input and output devices. They also get basic knowledge on

- Interactive Input Devices used to feed input in graphics system.
- Hard Copy Devices
- Graphics Software used to design graphics.
- Attributes of Output Primitives

2.3 Logical Classification of Input Devices

When input functions are classified according to data type, any device that is used to provide the specified data is referred to as a logical input device for that data type. The standard logical input-data classifications are

- LOCATOR - A device for specifying one coordinate position.
- STROKE - A device for specifying a set of coordinate positions.
- STRING - A device for specifying text input.
- VALUATOR - A device for specifying a scalar value.
- CHOICE - A device for selecting a menu option.
- PICK - A device for selecting a component of a picture.

Locator Devices

Interactive selection of a coordinate point is usually accomplished by positioning the screen cursor at some location in a displayed scene, although other methods, such as menu options, could be used in certain applications. We can use a mouse, touchpad, joystick, trackball, spaceball, thumbwheel, dial, hand cursor, or digitizer stylus for screen-cursor positioning. In addition, various buttons, keys, or switches can be used to indicate processing options for the selected location.

Keyboards are used for locator input in several ways. A general-purpose keyboard usually has four cursor-control keys that move the screen cursor up, down, left, and right. With an

additional four keys, we can move the cursor diagonally as well. Rapid cursor movement is accomplished by holding down the selected cursor key. Sometimes a keyboard includes a touchpad, joystick, trackball, or other device for positioning the screen cursor. For some applications, it may also be convenient to use a keyboard to type in numerical values or other codes to indicate coordinate positions.

Other devices, such as a light pen, have also been used for interactive input of coordinate positions. But light pens record screen positions by detecting light from the screen phosphors, and this requires special implementation procedures.

Stroke Devices

This class of logical devices is used to input a sequence of coordinate positions, and the physical devices used for generating locator input are also used as stroke devices. Continuous movement of a mouse, trackball, joystick, or hand cursor is translated into a series of input coordinate values. The graphics tablet is one of the more common stroke devices. Button activation can be used to place the tablet into “continuous” mode. As the cursor is moved across the tablet surface, a stream of coordinate values is generated. This procedure is used in paintbrush systems to generate drawings using various brush strokes. Engineering systems also use this process to trace and digitize layouts.

String Devices

The primary physical device used for string input is the keyboard. Character strings in computer-graphics applications are typically used for picture or graph labeling. Other physical devices can be used for generating character patterns for special applications. Individual characters can be sketched on the screen using a stroke or locator-type device. A pattern recognition program then interprets the characters using a stored dictionary of predefined patterns.

Valuator Devices

We can employ valuator input in a graphics program to set scalar values for geometric transformations, viewing parameters, and illumination parameters. In some applications, scalar

input is also used for setting physical parameters such as temperature, voltage, or stress-strain factors. A typical physical device used to provide valuator input is a panel of control dials. Dial settings are calibrated to produce numerical values within some predefined range. Rotary potentiometers convert dial rotation into a corresponding voltage, which is then translated into a number within a defined scalar range, such as -10.5 to 25.5. Instead of dials, slide potentiometers are sometimes used to convert linear movements into scalar values.

Any keyboard with a set of numeric keys can be used as a valuator device. Although dials and slide potentiometers are more efficient for fast input.

Joysticks, trackballs, tablets, and other interactive devices can be adapted for valuator input by interpreting pressure or movement of the device relative to a scalar range. For one direction of movement, say left to right, increasing scalar values can be input. Movement in the opposite direction decreases the scalar input value. Selected values are usually echoed on the screen for verification.

Another technique for providing valuator input is to display graphical representations of sliders, buttons, rotating scales, and menus on the video monitor. Cursor positioning, using a mouse, joystick, spaceball, or other device, can be used to select a value on one of these valuators. As a feedback mechanism for the user, selected values are usually displayed in text or color fields elsewhere within the graphical display belonging to the application.

Choice Devices

Menus are typically used in graphics programs to select processing options, parameter values, and object shapes that are to be used in constructing a picture. Commonly used choice devices for selecting a menu option are cursor-positioning devices such as a mouse, trackball, keyboard, touch panel, or button box.

Keyboard function keys or separate button boxes are often used to enter menu selections. Each button or function key is programmed to select a particular operation or value, although preset buttons or keys are sometimes included on an input device.

For screen selection of listed menu options, we use a cursor-positioning device. When a screen-cursor position (x, y) is selected, it is compared to the coordinate extents of each listed menu item. A menu item with vertical and horizontal boundaries at the coordinate values x_{min}, x_{max}, y_{min}, and y_{max} is selected if the input coordinates satisfy the inequalities $x_{\min} \leq x \leq x_{\max}, \qquad y_{\min} \leq y \leq y_{\max}$

For larger menus with relatively few options displayed, a touch panel is commonly used. A selected screen position is compared to the coordinate extents of the individual menu options to determine what process is to be performed. Alternate methods for choice input include keyboard and voice entry. A standard keyboard can be used to type in commands or menu options. For this method of choice input, some abbreviated format is useful. Menu listings can be numbered or given short identifying names. A similar encoding scheme can be used with voice input systems. Voice input is particularly useful when the number of options is small (20 or fewer).

Pick Devices

We use a pick device to select a part of a scene that is to be transformed or edited in some way. Several different methods can be used to select a component of a displayed scene, and any input mechanism used for this purpose is classified as a pick device. Most often, pick operations are performed by positioning the screen cursor. Using a mouse, joystick, or keyboard, for example, we can perform picking by positioning the screen cursor and pressing a button or key to record the pixel coordinates. This screen position can then be used to select an entire object, a facet of a tessellated surface, a polygon edge, or a vertex. Other pick methods include highlighting schemes, selecting objects by name, or a combination of methods.

Using the cursor-positioning approach, a pick procedure could map a selected screen position to a world-coordinate location using the inverse viewing and geometric transformations that were specified for the scene. Then, the world coordinate position can be compared to the coordinate extents of objects. If the pick position is within the coordinate extents of a single object, the pick object has been identified. The object name, coordinates, or other information about the object can then be used to apply the desired transformation or editing operations.

But if the pick position is within the coordinate extents of two or more objects, further testing is necessary. Depending on the type of object to be selected and the complexity of a scene, several levels of search may be required to identify the pick object. For example, if we are attempting to pick a sphere whose coordinate extents overlap the coordinate extents of some other three-dimensional object, the pick position could be compared to the coordinate extents of the individual surface facets of the two objects. If this test fails, the coordinate extents of individual line segments can be tested.

When coordinate-extent tests do not uniquely identify a pick object, the distances from the pick position to individual line segments could be computed. For a two-dimensional line segment with pixel endpoint coordinates (x_1, y_1) and (x_2, y_2), the perpendicular distance squared from a pick position (x, y) to the line is calculated as

$$d^2 = \frac{[\Delta x(y - y_1) - \Delta y(x - x_1)]^2}{\Delta x^2 + \Delta y^2}$$

where $\Delta x = x_2 - x_1$ and $\Delta y = y_2 - y_1$. Other methods, such as comparing distances to endpoint positions, have been proposed to simplify the line-picking operations. Pick procedures can be simplified if coordinate-extent testing is not carried out for the surface facets and line segments of an object. When the pick position is within the coordinate extents of two or more objects, the pick procedures can simply return a list of all candidate pick objects.

Another picking technique is to associate a pick window with a selected cursor position. The pick window is centered on the cursor position and clipping procedures are used to determine which objects intersect the pick window. For line picking, we can set the pick-window dimensions w and h to very small values, so that only one line segment intersects the pick window.

Some graphics packages implement three-dimensional picking by reconstructing a scene using the viewing and projection transformations with the pick window as the clipping window. Nothing is displayed from this reconstruction, but clipping procedures are applied to determine which objects are within the pick view volume. A list of information for each object in the pick view volume can then be returned for processing. This list can contain information such as

object name and depth range, where the depth range could be used to select the nearest object in the pick view volume.

2.4 Interactive Input Devices

For graphical inputs, we use a wide range of interactive input devices. Often used input devices are keyboard, mouse, trackball, spaceball, joystick, digitizers, dials, button boxes, data gloves, touch panels, image scanners, light pens, and voice systems. In the following section we will have a detail view of the popular input devices.

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 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. 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

A mouse is a small handheld unit that is usually moved around on a flat surface to position the screen cursor. One or more buttons on the top of the mouse provide a mechanism for

communicating selection information to the computer; wheels or rollers on the bottom of the mouse can be used to record the amount and direction of movement. Another method for detecting mouse motion is with an optical sensor. For 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. Some mouse systems 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 and the functionality of the mouse.

Each input can be configured to perform a wide range of actions, from traditional single-click inputs to macro operations containing multiple keystrongs, mouse events, and pre-programmed delays between operations. The laser-based optical sensor can be configured to control the degree of sensitivity to motion, allowing the mouse to be used in situations requiring different levels of control over cursor movement. In addition, the mouse can hold up to five different configuration profiles to allow the configuration to be switched easily when changing applications.

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 also can be mounted on other devices, 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, 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 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. 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

A data glove is a device that fits over the user's hand and 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 crosshairs for sighting positions, while a stylus is a pencil-shaped device that is pointed at positions on the tablet. The tablet size varies from 12 by 12 inches for desktop models to 44 by 60 inches or larger for floor models. Graphics tablets provide a highly accurate method for selecting coordinate positions, with an accuracy that varies from about 0.2 mm on desktop models to about 0.05 mm or less on larger models.

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

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

Three-dimensional digitizers use sonic or electromagnetic transmissions to record positions. One electromagnetic transmission method is similar to that employed in the data glove: A coupling between the transmitter and receiver is used to compute the location of a

stylus as it moves over an object surface. 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.

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 information to be stored. The gradations of grayscale or color are then recorded and stored in an array. Once we have the internal representation of a picture. In the scanned image we can apply transformations to rotate, scale, or crop the picture to a particular screen area. Scanners are available in a variety of sizes and capabilities, including small handheld models, drum scanners, and flatbed scanners.

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 are designed with touch screens. Other systems can be adapted for touch input by fitting a transparent device 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 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

Light pens are 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 because 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. In addition, prolonged use of the light pen can causearmfatigue, and light pens require special implementations for some applications because 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 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; the microphone is designed to minimize input of background sounds. Voice systems have some advantage over other input devices because the attention of the operator need not switch from one device to another to enter a command.

2.5 Hard-Copy Devices

A *hard copy* is a *printed copy* of information from a computer. A *hard copy* is so-called because it exists as a *physical* object. The devices used to produce the physical object of the picture or image is called hard-copy devices. Hard-copy output for images can be obtained in several formats. The pictures can be put on paper or slides, or film 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 printer 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.

Dot-Matrix printer

It has print head containing a rectangular array of protruding wire pins, with the number of pins varying depending upon the quality of the printer.

Laser Device

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

It produces 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. Ink-jet methods shoot the three colors simultaneously on a single pass along each print line.

Electrostatic device

An electrostatic device places a negative charge on the paper, one complete 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.

Pen plotter

A pen plotter has one or more pens mounted on a carriage, or crossbar, that spans a sheet of paper. Plotter paper can lies 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.

2.6 Graphics Software

There are two general classifications of graphics software

1. General programming packages
2. Special purpose applications packages

General programming package provides an extensive set of graphics function that can be used in high-level programming language such as C, C++, Java, or Fortran. Some examples

of general graphics programming packages are GL (Graphics Library), OpenGL, VRML (Virtual-Reality Modeling Language), Java 2D, and Java 3D.

Special purpose programming packages are designed for non-programmers, so that users can generate displays without worrying about how graphics operations work. Examples of such applications include artists' painting programs and various architectural, business, medical, and engineering CAD systems.

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. 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 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, handlebars, 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 need to describe only 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.

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. In addition, 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. 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 use various kinds of input devices, including a mouse, a tablet, and 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.

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,wecan create two-dimensional scenes within Java applets, for example; or we can produce three-dimensional web displays with Java 3D.

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.

2.7 Attributes of Output Primitives

Any parameter that affects the way a primitive is to be displayed is referred as an attribute parameter.

Area fill attributes

A defined area can be filled by solid color or patterns fill. The fill options can be applied to polygon region or areas defined within a boundary. The area can be painted using various brush styles, colors and transparency parameters.

Fill Styles

Areas are displayed with three basic fill styles:

1. Hollow with a color border
2. Filled with a solid color
3. Filled with a specified pattern or design

Pattern Fill

Pattern Filling is the procedure of filling the bounded region with a particular pattern. The process of filling an area with a rectangular pattern is called tiling.

Soft fill

If the fill color is combined with the background color then it is referred as soft-fill or tint-fill algorithm.

Character Attributes

The appearances of the characters are controlled by attributes such as font, size, color, and orientation.

Text attributes

Font is the basic text attribute, which is a set of characters with a particular design style. The characters in a selected font can also be displayed with assorted underlining styles, in

boldface, in italics and in outline or shadow styles. Fonts can be divided into two broad groups: serif and sans serif. Text size can be adjusted by scaling the overall dimension.

Marker attributes

A marker symbol is a single character that can be displayed in different colors and in different sizes. Marker attributes are implemented by procedures that load the chosen character into the raster at the defined position with the specified color and size.

Inquiry Functions

Inquiry functions are used to retrieve values for current attribute settings and for various other parameters. Using inquiry function we can get the current color settings, current position, check whether current routine is enabled or disabled.

2.8 Summary

In this lesson, we surveyed the major input, hard copy devices and graphics software features of computer-graphics systems.

- 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, ink-jet, electrostatic, or electrothermal methods. Graphs and charts can be produced with an ink-pen plotter or with a combination printer-plotter device.

2.9 Model Questions

1. What is the difference between impact and non-impact printers?
2. Define output primitives.
3. What are inquiry functions?
4. Explain the different types of input devices used by graphics systems.
5. Write short notes on the categories of graphics software.

LESSON - 3
OUTPUT PRIMITIVES

3.1 Introduction

In this chapter, we discuss the device-level algorithms for implementing output primitives. Exploring the implementation algorithms for a graphics library will give us valuable insight into the capabilities of these packages. It will also provide us with an understanding of how the functions work, perhaps how they could be improved, and how we might implement graphics routines ourselves for some special application. Graphics programming packages provide functions to describe a scene in terms of basic geometric structures are referred as output

primitives. Exploring the implementation algorithms for a graphics library will give us valuable insight into the capabilities of these packages. Three methods that can be used to locate pixel positions along a straight-line path are the DDA algorithm, Bresenham's algorithm, and the midpoint method.

3.2 Objective of this Lesson

The objective of this lesson is to give insight in various algorithms for implementing output primitives. We will discuss line drawing algorithm, circle drawing algorithms, ellipse drawing algorithm. In addition we will also learn the line and color attributes of the output primitive.

3.3 Line drawing algorithm

A straight-line segment in a scene is defined by the coordinate positions for the endpoints of the segment.

The Cartesian slope-intercept equation for a straight line is

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

With *m* as the slope of the line and *b* as the *y* intercept. Given that the two endpoints of a line segment are specified at positions (x_1, y_1) and (x_2, y_2) the slope *m* and *y* intercept *b* can be determined with the following calculations:

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

$$b = y_0 - m.x_0 \qquad (3)$$

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

For any given *x* interval Δx along a line, we can compute the corresponding *y* interval, Δy, from equation (2) as

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

Similarly, we can obtain the Δx interval corresponding to a specified Δy as

$$\Delta x = \frac{\Delta y}{m} \qquad (5)$$

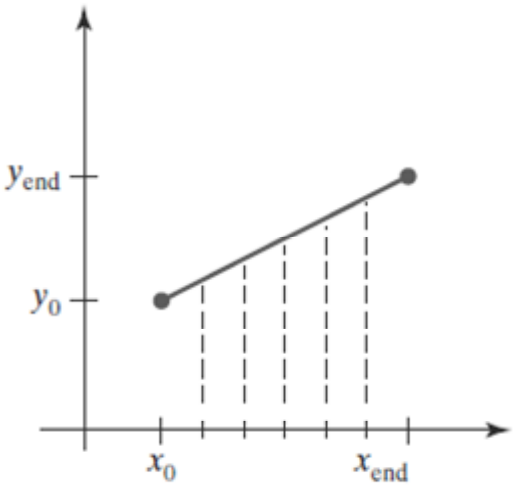


Figure 1:Straight-line segment with five sampling positions along the x axis between x_0 and x_{end}.

3.4 DDA Algorithm

Digital Differential Analyzer (DDA) algorithm is the simple line generation algorithm. DDA is a scan-conversion line algorithm based on calculating either Δ_x or Δ_y . A line is sampled at unit intervals in one coordinate and the corresponding integer values nearest the line path are determined for the other coordinate.

Positive slope:

Case(i) [slope ≤1]

If the slope is less than or equal to 1, sample at unit (Δx = 1) intervals and compute successive y values

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

Case(ii) [slope >1]

If the slope greater than 1, sample at unit y intervals ($\Delta y = 1$) and compute successive x values

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

Negative slope: If this processing is reversed, so that the starting endpoint is at the right

Case(iii) [slope ≤ 1]

If the slope is less than or equal to 1, sample at unit x intervals ($\Delta x=-1$) and compute successive y values

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

Case(iv) [slope >1]

If the slope greater than 1, sample at unit y intervals ($\Delta y=-1$) and compute successive x values

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

Pseudo Code for DDA line drawing algorithm

```
#include <stdlib.h>
#include <math.h>
inline int round (const float a) { return int (a + 0.5); }
void lineDDA (int x0, int y0, int xEnd, int yEnd)
{
        int dx = xEnd - x0, dy = yEnd - y0, steps, k;
```

```
        float xIncrement, yIncrement, x = x0, y = y0;
            if (fabs (dx) > fabs (dy))
                steps = fabs (dx);
            else
                steps = fabs (dy);
        xIncrement = float (dx) / float (steps);
        yIncrement = float (dy) / float (steps);
        setPixel (round (x), round (y));
        for (k = 0; k < steps; k++)
        {
            x += xIncrement;
            y += yIncrement;
        setPixel (round (x), round (y));
        }
}
```

This algorithm is summarized in the following procedure, which accepts as input two integer screen positions for the endpoints of a line segment. Horizontal and vertical differences between the endpoint positions are assigned to parameters dx and dy. The difference with the greater magnitude determines the value of parameter steps. This value is the number of pixels that must be drawn beyond the starting pixel; from it, calculate the *x* and *y* increments needed to generate the next pixel position at each step along the line path. Draw the starting pixel at position (x0, y0), and then draw the remaining pixels iteratively, adjusting *x* and *y* at each step to obtain the next pixel's position before drawing it.

If the magnitude of dx is greater than the magnitude of dy and x0 is less than x1, the values for the increments in the *x* and *y* directions are 1 and *m*, respectively. If the greater change is in the *x* direction, but x0 is greater than x1, then the decrements -1 and -*m* are used to generate each new point on the line. Otherwise, we use a unit increment (or decrement) in the *y* direction and an *x* increment (or decrement) of 1*/m.*

The main advantage of DDA algorithm is the speed in the calculation of the pixel position. The disadvantage is the round-off error in successive addition.

3.5 Bresenham's Line Algorithm

Bresenham's Line Algorithm is an accurate and efficient raster line-generating algorithm, developed by Bresenham. Pixel positions along a line path are then determined by sampling at unit *x* intervals. Starting from the left endpoint (*x*0, *y*0) of a given line, we step to each successive column (*x* position) and plot the pixel whose scan-line *y* value is closest to the line path.

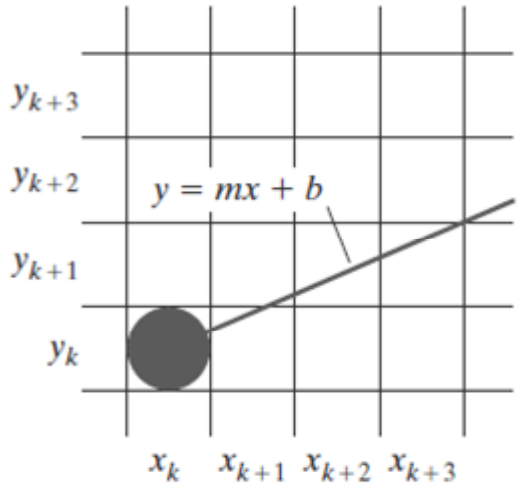


Figure 2: A section of the screen showing a pixel in column x_k on scan line y_k that is to be plotted along the path of a line segment with slope 0 < m < 1

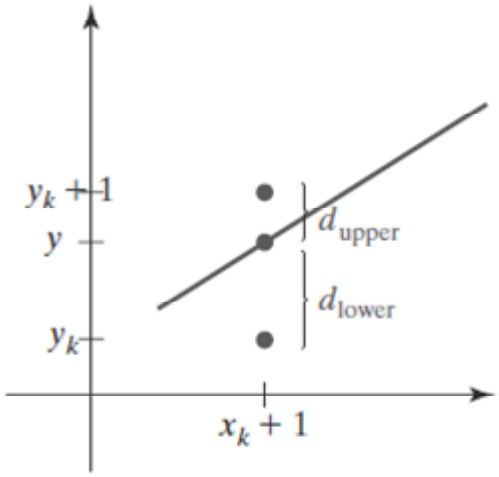


Figure 3: Vertical distances between pixel positions and the line y coordinate at sampling position $x_k + 1$.

Assuming that we have determined that the pixel at (x_k , y_k) is to be displayed, we next need to decide which pixel to plot in column x_k+1 = x_k + 1. Our choices are the pixels at positions (x_k + 1, y_k) and (x_k + 1, y_k + 1).

Starting from the left end point (x_0,y_0) of a given line, we step to each x-position and plot the pixel whose scan-line y-value is closest to the line path.

At sampling position x_k + 1, we label vertical pixel separations from the mathematical line path as d_{lower} and d_{upper} . The y coordinate on the mathematical line at pixel column position x_k + 1 is calculated as

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

$$d_{lower} = y - y_k$$

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

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

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

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

The decision parameter P_k

$$\begin{aligned} p_k &= \Delta x(d_{\text{lower}} - d_{\text{upper}}) \\ &= 2\Delta y \cdot x_k - 2\Delta x \cdot y_k + c \end{aligned}$$

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

At step $k + 1$, the decision parameter P_{k+1}

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

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

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

The first parameter P_0 at the starting pixel (x_0, y_0) is

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

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

1. Input the two line endpoints and store the left endpoint in (x_0, y_0).

2. Set the color for frame-buffer position (x_0, y_0); i.e., plot the first point.

3. Calculate the constants Δx, Δy, $2\Delta y$, and $2\Delta y - 2\Delta x$, and obtain the starting value for the decision parameter as

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

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

   If $p_k < 0$,

   the next point to plot is (x_{k+1}, y_k) and $p_{k+1} = p_k + 2\Delta y$

   Otherwise,

   the next point to plot is (x_{k+1}, y_{k+1}) and $p_{k+1} = p_k + 2\Delta y - 2\Delta x$

5. Repeat step 4 Δx times.

Implementation of Bresenham line-drawing procedure for |m| < 1.0 in C.

```
#include <stdlib.h>
#include <math.h>
void lineBres (int x0, int y0, int xEnd, int yEnd)
{
int dx = fabs (xEnd - x0), dy = fabs(yEnd - y0);
int p = 2 * dy - dx;
int twoDy = 2 * dy, twoDyMinusDx = 2 * (dy - dx);
int x, y;
if (x0 > xEnd)
{
x = xEnd;
y = yEnd;
xEnd = x0;
```

```
}
else
{
x = x0;
y = y0;
}
setPixel (x, y);
while (x < xEnd)
{
x++;
if (p < 0)
p += twoDy;
else
{
y++;
p += twoDyMinusDx;
}
setPixel (x, y);
}
}
```

3.6 Parallel Line Algorithms

Using parallel processing, we can calculate multiple pixel positions along a line path simultaneously by partitioning the computations among the various processors available.

Given n_p processors, we can set up a parallel Bresenham line algorithm by subdividing the line path into n_p partitions and simultaneously generating line segments in each of the subintervals. For a line with slope $0 < m < 1.0$ and left endpoint coordinate position (x_0, y_0), we partition the line along the positive x direction. The distance between beginning x positions of adjacent partitions can be calculated as

$$\Delta x_p = \frac{\Delta x + n_p - 1}{n_p}$$

Where Δx – width of the line

n_p - number of processors

The starting x coordinate of the k^{th} partition is

$$x_k = x_0 + k\Delta x_p$$

To apply Bresenham's algorithm over the partitions, initial value of the y coordinate and the initial value of the decision parameter in each partition is needed.

The distance between the beginning y positions of adjacent partitions can be calculated as

$$\Delta y_p = m\Delta x_p$$

At the kth partition, the starting y coordinate is

$$y_k = y_0 + \text{round}(k\Delta y_p)$$

The initial decision parameter p_k is

$$p_k = (k\Delta x_p)(2\Delta y) - \text{round}(k\Delta y_p)(2\Delta x) + 2\Delta y - \Delta x$$

Each processor then calculates pixel positions over its assigned subinterval using the preceding starting decision parameter value and the starting coordinates (*xk* , *yk*).

3.7 Circle-Generating Algorithms

Properties of Circles

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

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

This equation can be used to calculate the position of points on a circle circumference by stepping along the *x* axis in unit steps from x_c - *r* to x_c + *r* and calculating the corresponding *y* values at each position as

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

The circle equation in parametric polar form cab be expressed with the following pair of equations

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

3.8 Midpoint Circle Algorithm

As in the raster line algorithm, midpoint circle algorithm sample pixel point at unit intervals and determine the closest pixel position to the specified circle path at each step. For a given radius *r* and screen center position (x_c , y_c), we can first set up our algorithm to calculate pixel positions around a circle path centered at the coordinate origin (0, 0).

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

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

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

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

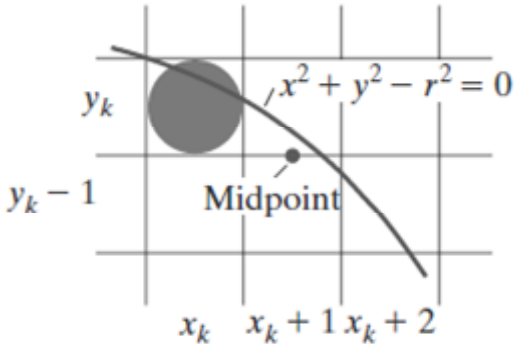


Figure 4: Midpoint between candidate pixels at sampling position $x_k + 1$ along a circular path.

Assuming that we have just plotted the pixel at (x_k, y_k), we next need to determine whether the pixel at position (x_{k+1}, y_k) or the one at position (x_{k+1}, y_{k-1}) is closer to the circle. Decision parameter evaluated at the midpoint between these two pixels:

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

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

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

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

At the start position $(0, r)$, these two terms have the values 0 and $2r$, respectively. Each successive value for the $2x_{k+1}$ term is obtained by adding 2 to the previous value, and each successive value for the $2y_{k+1}$ term is obtained by subtracting 2 from the previous value.

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

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

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

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

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

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

since all increments are integers.

Pseudo code of Midpoint Circle Algorithm

1. Input radius r and circle center (x_c, y_c), then set the coordinates for the first point on the circumference of a circle centered on the origin as $(x_0, y_0) = (0, r)$

2. Calculate the initial value of the decision parameter as $p_0 = \frac{5}{4} - r$

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

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

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

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

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

4. Determine symmetry points in the other seven octants.

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

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

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

C code for implementing midpoint circle algorithm:

```
#include<graphics.h>

#include<stdio.h>

void pixel(int xc,int yc,int x,int y);

int main()

{

  x=0;

  y=r;

  p=1-r;
```

```
pixel(xc,yc,x,y);
while(x<y)
{
  if(p<0)
  {
        x++;
        p=p+2*x+1;
  }
  else
  {
        x++;
        y - - ;
        p=p+2*(x-y)+1;
  }
  pixel(xc,yc,x,y);
}
}
void pixel(int xc,int yc,int x,int y)
{
  putpixel(xc+x,yc+y,WHITE);
  putpixel(xc+x,yc-y,WHITE);
  putpixel(xc-x,yc+y,WHITE);
  putpixel(xc-x,yc-y,WHITE);
```

```
    putpixel(xc+y,yc+x,WHITE);
    putpixel(xc+y,yc-x,WHITE);
    putpixel(xc-y,yc+x,WHITE);
    putpixel(xc-y,yc-x,WHITE);
}
```

3.9 Ellipse-Generating Algorithms

An ellipse is an elongated circle. An ellipse can also be described as a modified circle whose radius varies from a maximum value in one direction to a minimum value in the perpendicular direction. The straight-line segments through the interior of the ellipse in these two perpendicular directions are referred to as the major and minor axes of the ellipse.

Properties of Ellipses

Definition of an ellipse can be given in terms of the distances from any point on the ellipse to two fixed positions, called the foci of the ellipse.

If the distances to the two focus positions from any point P = *(x, y)* on the ellipse are labeled d_1 and d_2, then the general equation of an ellipse can be stated as

$$d_1 + d_2 = \text{constant}$$

Expressing distances d_1 and d_2 in terms of the focal coordinates $F_1 = (x_1, y_1)$ and

$F_2 = (x_2, y_2)$, we have

$$\sqrt{(x - x_1)^2 + (y - y_1)^2} + \sqrt{(x - x_2)^2 + (y - y_2)^2} = \text{constant}$$

The general ellipse equation is

$$A x^2 + B y^2 + C x y + D x + E y + F = 0$$

where the coefficients *A*, *B*, *C*, *D*, *E*, and *F* are evaluated in terms of the focal coordinates and the dimensions of the major and minor axes of the ellipse.

The equation for the ellipse can be written in terms of the ellipse center coordinates and parameters r_x and r_y as

$$\left(\frac{x - x_c}{r_x}\right)^2 + \left(\frac{y - y_c}{r_y}\right)^2 = 1$$

Using polar coordinates *r* and θ, we can also describe the ellipse in standard position with the parametric equations

$$x = x_c + r_x \cos\theta$$
$$y = y_c + r_y \sin\theta$$

3.10 Midpoint Ellipse Algorithm

Midpoint ellipse algorithm is similar to that used in displaying a raster circle. Given parameters r_x, r_y, and (x_c , y_c), determine curve positions (x, y) for an ellipse in standard position centered on the origin, then shift all the points using a fixed offset so that the ellipse is centered at (x_c, y_c). The midpoint ellipse method is applied throughout the first quadrant in two parts. The division of the first quadrant according to the slope of an ellipse with $r_x < r_y$.

We define an ellipse function

$$f_{\text{ellipse}}(x, y) = r_y^2 x^2 + r_x^2 y^2 - r_x^2 r_y^2$$

which has the following properties:

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

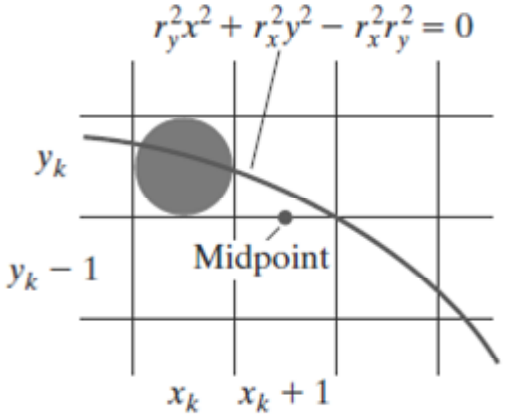


Figure 5: Midpoint between candidate pixels at sampling position x_{k+1} along an elliptical path.

At the next sampling position ($x_{k+1}+1 = x_{k+2}$), the decision parameter for region 1 is evaluated as

$$\begin{aligned} p1_{k+1} &= f_{\text{ellipse}}\left(x_{k+1} + 1, y_{k+1} - \frac{1}{2}\right) \\ &= r_y^2[(x_k + 1) + 1]^2 + r_x^2\left(y_{k+1} - \frac{1}{2}\right)^2 - r_x^2 r_y^2 \end{aligned}$$

or

$$p1_{k+1} = p1_k + 2r_y^2(x_k + 1) + r_y^2 + r_x^2\left[\left(y_{k+1} - \frac{1}{2}\right)^2 - \left(y_k - \frac{1}{2}\right)^2\right]$$

here y_{k+1} is either y_k or y_{k-1}, depending on the sign of $p1_k$.

Decision parameters are incremented by the following amounts:

$$\text{increment} = \begin{cases} 2r_y^2 x_{k+1} + r_y^2, & \text{if } p1_k < 0 \\ 2r_y^2 x_{k+1} + r_y^2 - 2r_x^2 y_{k+1}, & \text{if } p1_k \geq 0 \end{cases}$$

The steps for displaying an ellipse using the midpoint algorithm:

Midpoint Ellipse Algorithm

1. Input r_x, r_y, and ellipse center (x_c , y_c), and obtain the first point on an ellipse centered on the origin as

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

2. Calculate the initial value of the decision parameter in region 1 as

$$p1_0 = r_y^2 - r_x^2 r_y + \frac{1}{4} r_x^2$$

3. At each *xk* position in region 1, starting at $k = 0$, perform the following test: If $p1_k < 0$, the next point along the ellipse centered on (0, 0) is (x_{k+1}, y_k) and

$$p1_{k+1} = p1_k + 2r_y^2 x_{k+1} + r_y^2$$

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

$$p1_{k+1} = p1_k + 2r_y^2 x_{k+1} - 2r_x^2 y_{k+1} + r_y^2$$

4. Calculate the initial value of the decision parameter in region 2 as

$$p2_0 = r_y^2\left(x_0 + \frac{1}{2}\right)^2 + r_x^2(y_0 - 1)^2 - r_x^2 r_y^2$$

where (x_0, y_0) is the last position calculated in region 1.

5. At each y_k position in region 2, starting at $k = 0$, perform the following test: If $p2_k > 0$, the next point along the ellipse centered on (0, 0) is $(x_k , y_k - 1)$ and

$$p2_{k+1} = p2_k - 2r_x^2 y_{k+1} + r_x^2$$

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

$$p2_{k+1} = p2_k + 2r_y^2 x_{k+1} - 2r_x^2 y_{k+1} + r_x^2$$

6. For both regions, determine symmetry points in the other three quadrants.

7. Move each calculated pixel position (x, y) onto the elliptical path centered on (x_c , y_c) and plot these coordinate values:

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

3.11 Attributes of Output Primitives

Any parameter that affects the way a primitive is to be displayed is referred as attribute parameters. Attribute parameters color and size are the fundamental characteristics of a primitive.

3.12 Line Attributes

Basic attributes of a straight line segment are its type or style, its width and its color.

Line Width

Implementation of line-width options depends on the capabilities of the output device. For raster implementations, a standard-width line is generated with single pixels at each sample position, as in the Bresenham algorithm. Thicker lines are displayed as positive integer multiples of the standard line by plotting additional pixels along adjacent parallel line paths. The number of pixels to be displayed in each column is set equal to the integer value of the line width. At each x sampling position, we calculate the corresponding y coordinate and plot pixels at screen coordinates (x, y) and (x, y+1).

Other methods for producing thick lines include displaying the line as a filled rectangle or generating the line with a selected pen or brush pattern. Displaying thick polylines using horizontal and vertical pixel spans. A round join is produced by capping the connection between the two segments with a circular boundary whose diameter is equal to the line width. A bevel join is generated by displaying the line segments with butt caps and filling in the triangular gap where the segments meet.

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

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

Line Style

Raster line algorithms display line-style attributes by plotting pixel spans. For dashed, dotted, and dot-dashed patterns, the line-drawing procedure outputs sections of contiguous pixels along the line path, skipping over a number of intervening pixels between the solid spans. With a line slope greater than 1.0 in magnitude, we can display thick lines using horizontal spans. Similarly, a thick line with slope less than or equal to 1.0 can be displayed using vertical pixel spans.

We set a current display style for lines with the OpenGL function

glLineStipple (repeatFactor, pattern);

Parameter pattern is used to reference a 16-bit integer that describes how the line should be displayed. Integer parameter repeatFactor specifies how many times each bit in the pattern is to be repeated before the next bit in the pattern is applied. The default repeat value is 1.

Pen and Brush Options

Pen and brush shapes can be stored in a pixel mask that identifies the array of pixel positions that are to be set along the line path. Lines generated with pen (or brush) shapes can be displayed in various widths by changing the size of the mask.

Line Color

When a system provides color (intensity) options, a parameter giving the current color index is included in the list of system attribute values. The number of color choices depends on the number of bits available per pixel in the frame buffer.

3.13 Color and Grayscale style

A basic attribute for all primitives is color. Various color options can be made available to a user, depending on the capabilities and design objectives of a particular system. Color options can be specified numerically or selected from menus or displayed slider scales.

RGB Color Components

In a color raster system, the number of color choices available depends on the amount of storage provided per pixel in the frame buffer. We can store red, green, and blue (RGB) color codes directly in the frame buffer, or we can put the color codes into a separate table and use the pixel locations to store index values referencing the color-table entries. Color tables are an alternate means for providing extended color capabilities to a user without requiring large frame buffers.

Color Tables

A color table can be useful in a number of applications, and it can provide a “reasonable” number of simultaneous colors without requiring large frame buffers. When a color value is changed in the color table, all pixels with that color index immediately change to the new color.

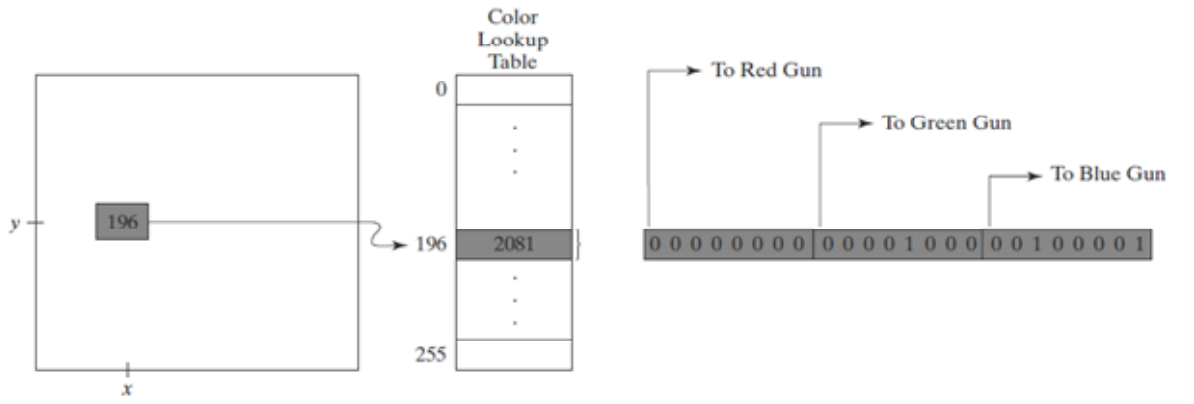


Figure 6: A color lookup table with 24 bits per entry that is accessed from a frame buffer with 8 bits per pixel.

Grayscale

Because color capabilities are now common in computer-graphics systems, we use RGB color functions to set shades of gray, or grayscale, in an application program. When an RGB color setting specifies an equal amount of red, green, and blue, the result is some shade of gray. Values close to 0 for the color components produce dark gray, and higher values near 1.0 produce light gray. Applications for grayscale display methods include enhancing black-and-white photographs and generating visualization effects.

3.14 Summary

The output primitives discussed in this lesson provides the basic tools for constructing straight line, circle and ellispse.

- Three methods that can be used to locate pixel positions along a straight-line path are the DDA algorithm, Bresenham’s algorithm, and the midpoint method.

- Bresenham's line algorithm and the midpoint line method are equivalent, and they are the most efficient.
- Circles and ellipses can be efficiently and accurately scan-converted using midpoint methods.
- Attributes control the display characteristics of graphics primitives.
- Line segment primitives can be displayed with three basic attributes: color, width, and style.
- A common attribute for all primitives is color, which is most often specified in terms of RGB components.
- Color selections can also be made using color-lookup tables.

3.15 Model Questions

1. Explain DDA line drawing algorithm with its drawbacks.
2. Explain bresanham's line drawing algorithms.
3. Explain midpoint Circle algorithm.
4. Explain midpoint ellipse algorithm.
5. What are the basic attributes of output primitives?
6. Name the attributes of a line.
7. Write the significance of color attribute.

LESSON - 4
TWO DIMENSIONAL TRANSFORMATIONS

Structure of the lesson

4.1 Introduction

In this lesson we look at transformation operations that we can apply to objects to reposition or resize them. These operations are also used in the viewing routines that convert a world-coordinate scene description to a display for an output device. In addition, they are used in a variety of other applications, such as computer-aided design (CAD) and computer animation. An architect, for example, creates a layout by arranging the orientation and size of the component parts of a design, and a computer animator develops a video sequence by moving the “camera” position or the objects in a scene along specified paths.

Operations that are applied to the geometric description of an object to change its position, orientation, or size are called geometric transformations. Modeling transformations are used to construct a scene or to give the hierarchical description of a complex object that is composed of several parts, which in turn could be composed of simpler parts, and so forth.

In general, modeling transformations are used to construct a scene or to give the hierarchical description of a complex object that is composed of several parts, which in turn could be composed of simpler parts, and so forth. Geometric transformations, on the other hand, can be used to describe how objects might move around in a scene during an animation sequence or simply to view them from another angle. Therefore, some graphics packages provide two sets of transformation routines, while other packages have a single set of functions that can be used for both geometric transformations and modeling transformations.

4.2 Objective of this Lesson

From this lesson the reader will gain knowledge and understanding about two-dimensional transformations namely translation, rotation and scaling. The student will also learn shearing and reflection. In addition to simple transformation, we will discuss the mechanism for performing composite transformations.

4.3 Basic Transformation

There are three basic transformations for two dimensional objects.

- Translation
- Rotation
- Scaling

The geometric-transformation functions that are available in all graphics packages are translation, rotation, and scaling. In addition to the basic we have another two transformations reflection and shearing. To introduce the general concepts associated with geometric transformations, we first consider operations in two dimensions. Once we understand the basic concepts, we can easily write routines to perform geometric transformations on objects in a two-dimensional scene.

4.4 Two-Dimensional Translation

Translation is the reposition of an object along a straight line. We perform a translation on a single coordinate point by adding offsets to its coordinates so as to generate a new coordinate position. A translation is applied to an object that is defined with multiple coordinate positions,

such as a quadrilateral, by relocating all the coordinate positions by the same displacement along parallel paths. Then the complete object is displayed at the new location.

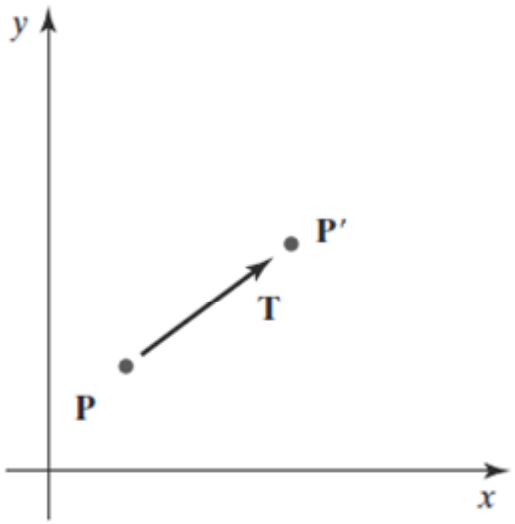
To translate a two-dimensional position, we add translation distances t_x and t_y to the original coordinates (x, y).

(x', y') where x' = x' + tx and

y' = y + fy to obtain the new coordinate position

The above equations can be expressed as a single matrix equation by using the following column vectors to represent coordinate positions and the translation vector: $P = \begin{bmatrix} x \\ y \end{bmatrix}, p' = \begin{bmatrix} x' \\ y' \end{bmatrix}, T = \begin{bmatrix} t_x \\ t_y \end{bmatrix}$

The translation distance pair (t_x, t_y) is called a translation vector or shift vector.



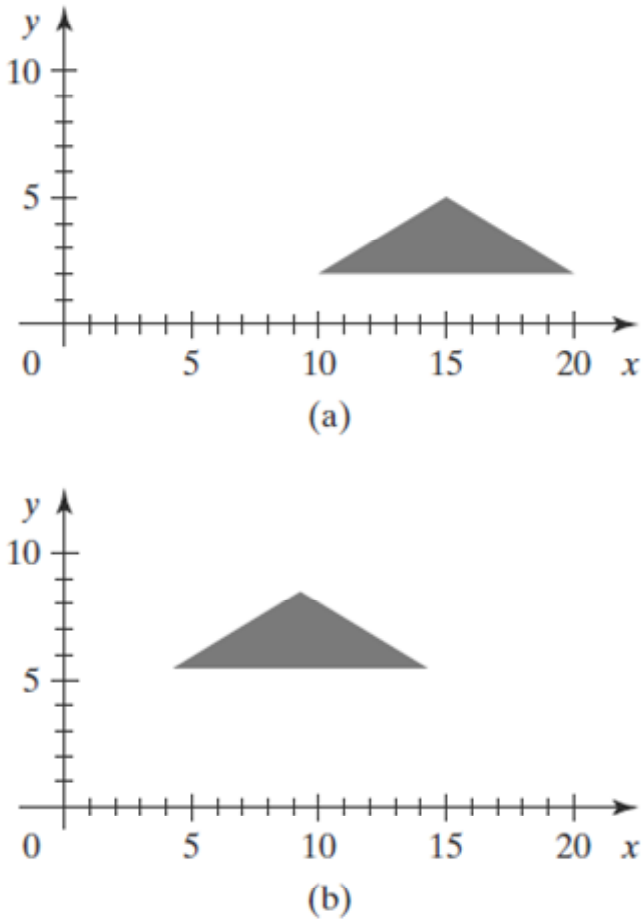
The two-dimensional translation equations in the matrix form is $P' = P + T$

Figure 1:Translating a point from position P to position P' using a translation vector T.

Translation is a rigid-body transformation that moves objects without deformation. That is, every point on the object is translated by the same amount. That is, every point on the object is translated by the same amount. A straight-line segment is translated by applying translation to each of the two line endpoints and redrawing the line between the new endpoint positions. A

polygon is translated similarly. We add a translation vector to the coordinate position of each vertex and then regenerate the polygon using the new set of vertex coordinates.

Similar methods are used to translate other objects. To change the position of a circle or

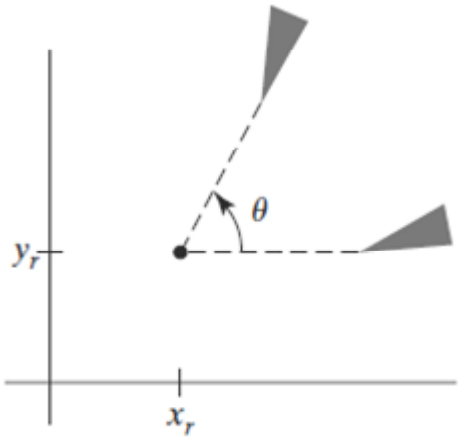


ellipse, we translate the center coordinates and redraw the figure in the new location. For a spline curve, we translate the points that define the curve path and then reconstruct the curve sections between the new coordinate positions.

Figure 2: Translation of a polygon.

4.5 Two-Dimensional Rotation

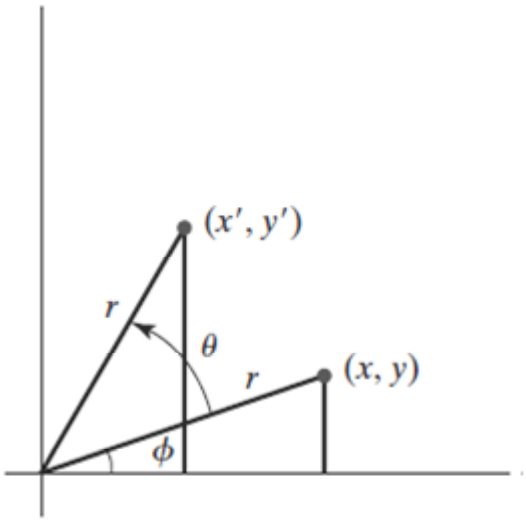
A two-dimensional rotation of an object is obtained by repositioning the object along a circular path in the *xy* plane.



We generate a rotation transformation of an object by specifying a rotation axis and a rotation angle. All points of the object are then transformed to new positions by rotating the points through the specified angle about the rotation axis.

Figure 3. Rotation of an object through angle θ about the pivot point (x_r , y_r).

Parameters for the two-dimensional rotation are the rotation angle θ and a position (x_r, y_r), called the rotation point (or pivot point), about which the object is to be rotated (Figure 3). The



pivot point is the intersection position of the rotation axis with the *xy* plane. A positive value for the angle θ defines a counterclockwise rotation about the pivot point, and a negative value rotates objects in the clockwise direction.

Figure 4. Rotation of a point from position (x, y) to position (x‘, y’) through an angle θ

relative to the coordinate origin.

We first determine the transformation equations for rotation of a point position **P** when the pivot point is at the coordinate origin. The angular and coordinate relationships of the original and transformed point positions are shown in Figure 3. In this figure, *r* is the constant distance of the point from the origin, angle is the original angular position of the point from the horizontal, and θ is the rotation angle. Using standard trigonometric identities, we can express the transformed coordinates in terms of angles θ and ϕ as

$$x' = r\cos(\phi + \theta) = r\cos\phi\cos\theta - r\sin\phi\sin\theta$$
$$y' = r\sin(\phi + \theta) = r\cos\phi\sin\theta + r\sin\phi\cos\theta$$

The original coordinates of the point in polar coordinates are

$$x = r\cos\phi, \qquad y = r\sin\phi$$

Substituting *x* and *y*, we obtain the transformation equations for rotating a point at position (*x*, *y*) through an angle θ about the origin:

$$x' = x\cos\theta - y\sin\theta$$
$$y' = x\sin\theta + y\cos\theta$$

we can write the rotation equations in the matrix form $P' = R.P$

where the rotation matrix is $R = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$

Rotations are rigid-body transformations that move objects without deformation. Every point on an object is rotated through the same angle.

4.6 Two-Dimensional Scaling

A scaling transformation alters the size of an object. Scaling operation is performed by multiplying object positions (*x*, *y*) by scaling factors s_x and s_y to produce the transformed coordinates (x', y'):

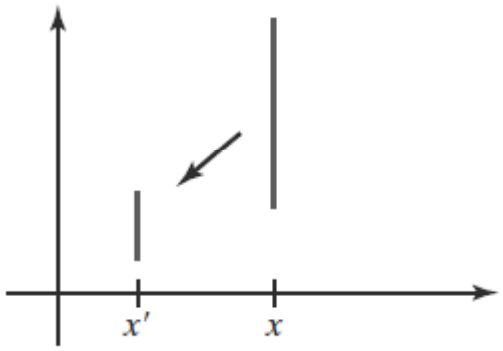
Scaling factor s_x scales an object in the x direction, while s_y scales in the y direction. The basic two-dimensional scaling equations can also be written in the following matrix form:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix}$$

P' = S.P

where S is the 2 × 2 scaling matrix

Any positive values can be assigned to the scaling factors s_x and s_y. Values less than 1 reduce the size of objects; values greater than 1 produce enlargements. Specifying a value of 1 for both s_x and s_y leaves the size of objects unchanged. When s_x and s_y are assigned the same value, a uniform scaling is produced. Unequal values for s_x and s_y result in a differential



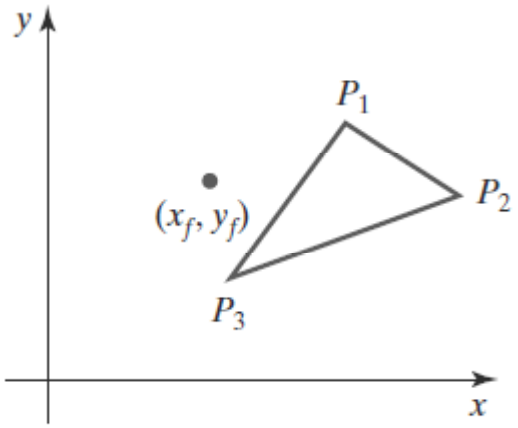
scaling, that is often used in design applications, where pictures are constructed from a few basic shapes that can be adjusted by scaling and positioning transformations.

Figure 5: A line scaled using $s_x = s_y = 0.5$

In some systems, negative values can also be specified for the scaling parameters. This not only resizes an object, it reflects it about one or more of the coordinate axes.

Scaling factors with absolute values less than 1 move objects closer to the coordinate origin, while absolute values greater than 1 move coordinate positions farther from the origin. Figure 5 illustrates scaling of a line by assigning the value 0.5 to both s_x and s_y. Both the line length and the distance from the origin are reduced by a factor of ½.

We can control the location of a scaled object by choosing a position, called the fixed point, that is to remain unchanged after the scaling transformation. Coordinates for the fixed point, (x f , yf), are often chosen at some object position, such as its centroid, but any other



spatial position can be selected.

Figure 6: Scaling relative to a chosen fixed point (x_f , y_f).

Objects are now resized by scaling the distances between object points and the point (Figure 6). For a coordinate position (x, y), the scaled coordinates (x', y') are then calculated from the following relationships:

$$x' - x_f = (x - x_f)s_x, \qquad y' - y_f = (y - y_f)s_y$$

We can rewrite the above equations to separate the multiplicative and additive terms as

$$x' = x \cdot s_x + x_f(1 - s_x)$$
$$y' = y \cdot s_y + y_f(1 - s_y)$$

where the additive terms $x_f (1 - s_x)$ and $y_f (1 - s_y)$ are constants for all points in the object.

Polygons are scaled by applying transformations to each vertex, then regenerating the polygon using the transformed vertices. For other objects, we apply the scaling transformation equations to the parameters defining the objects. To change the size of a circle, we can scale its radius and calculate the new coordinate positions around the circumference. And to change

the size of an ellipse, we apply scaling parameters to its two axes and then plot the new ellipse positions about its center coordinates.

Matrix Representations and Homogeneous Coordinates

Many graphics applications involve sequences of geometric transformations. The matrix representations can be reformulated so that such transformation sequences can be processed efficiently. In design and picture construction applications, we perform translations, rotations, and scalings to fit the picture components into their proper positions. The viewing transformations involve sequences of translations and rotations to take us from the original scene specification to the display on an output device. Here, we consider how the matrix representations discussed in the previous sections can be reformulated so that such transformation sequences can be processed efficiently.

The three basic two-dimensional transformations (translation, rotation, and scaling) can be expressed in the general matrix form

$$\mathbf{P}' = \mathbf{M}_1 \cdot \mathbf{P} + \mathbf{M}_2$$

with coordinate positions P and P' represented as column vectors. Matrix M_1 is a 2 × 2 array containing multiplicative factors, and M_2 is a two-element column matrix containing translational terms. For translation, M_1 is the identity matrix. For rotation or scaling, M_2 contains the translational terms associated with the pivot point or scaling fixed point. To produce a sequence of transformations with these equations, such as scaling followed by rotation and then translation, we could calculate the transformed coordinates one step at a time. First, coordinate positions are scaled, then these scaled coordinates are rotated, and finally, the rotated coordinates are translated. A more efficient approach, however, is to combine the transformations so that the final coordinate positions are obtained directly from the initial coordinates, without calculating intermediate coordinate values.

Homogeneous Coordinates

Multiplicative and translational terms for a two-dimensional geometric transformation can

be combined into a single matrix if we expand the representations to 3 × 3 matrices. Then we can use the third column of a transformation matrix for the translation terms, and all transformation equations can be expressed as matrix multiplications.

A standard technique for accomplishing this is to expand each two dimensional coordinate-position representation (x, y) to a three-element representation (x_h, y_h, h), called homogeneous coordinates, where the homogeneous parameter h is a nonzero value such that

$$x = \frac{x_h}{h}, \qquad y = \frac{y_h}{h}$$

A general two-dimensional homogeneous coordinate representation could also be written as ($h.x$, $h.y$, h). For geometric transformations, we can choose the homogeneous parameter h to be any nonzero value. Thus, each coordinate point (x, y) has an infinite number of equivalent homogeneous representations. A convenient choice is simply to set h = 1. Each two-dimensional position is then represented with homogeneous coordinates (x, y, 1). Other values for parameter h are needed, for example, in matrix formulations of three-dimensional viewing transformations.

Homogeneous coordinates is used in mathematics to refer to the effect of this representation on Cartesian equations. When a Cartesian point (x, y) is converted to a homogeneous representation (x_h, y_h, h), equations containing x and y, such as $f(x, y) = 0$, become homogeneous equations.

Expressing positions in homogeneous coordinates allows us to represent all geometric transformation equations as matrix multiplications, which is the standard method used in graphics systems. Two-dimensional coordinate positions are represented with three-element column vectors, and two-dimensional transformation operations are expressed as 3 × 3 matrices.

Two-Dimensional Translation Matrix

Using a homogeneous-coordinate approach, a two-dimensional translation of a coordinate position using the following matrix multiplication:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

This translation operation can be written in the abbreviated form

$$\mathbf{P}' = \mathbf{T}(t_x, t_y) \cdot \mathbf{P}$$

with $\mathbf{T}(t_x, t_y)$ as the 3 × 3 translation matrix.

Two-Dimensional Rotation Matrix

Two-dimensional rotation transformation equations about the coordinate origin can be expressed in the matrix form

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

or as $\mathbf{P}' = \mathbf{R}(\theta) \cdot \mathbf{P}$

The rotation transformation operator $\mathbf{R}(\theta)$ is the 3 × 3 matrix.

A rotation about any other pivot point must then be performed as a sequence of transformation operations.

Two-Dimensional Scaling Matrix

A scaling transformation relative to the coordinate origin can now be expressed as the matrix multiplication.

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Or $\mathbf{P}' = \mathbf{S}(s_x, s_y) \cdot \mathbf{P}$

The scaling operator **S** (s_x, s_y) is the 3 × 3 matrix with parameters s_x and s_y.

4.7 Composite Transformations

Using matrix representations, we can set up a sequence of transformations as a composite transformation matrix by calculating the product of the individual transformations. Forming products of transformation matrices is often referred to as a concatenation, or composition, of matrices. Because a coordinate position is represented with a homogeneous column matrix, we must premultiply the column matrix by the matrices representing any transformation sequence. Also, because many positions in a scene are typically transformed by the same sequence, it is more efficient to first multiply the transformation matrices to form a single composite matrix. Thus, if we want to apply two transformations to point position P, the transformed location would be calculated as

$$\begin{aligned} \mathbf{P}' &= \mathbf{M}_2 \cdot \mathbf{M}_1 \cdot \mathbf{P} \\ &= \mathbf{M} \cdot \mathbf{P} \end{aligned}$$

The coordinate position is transformed using the composite matrix **M**, rather than applying the individual transformations $\mathbf{M}_1$ and then$\mathbf{M}_2$.

Composite Two-Dimensional Translations

If two successive translation vectors (t_{1x}, t_{1y}) and (t_{2x}, t_{2y}) are applied to a two dimensional coordinate position **P**, the final transformed location $\boldsymbol{P'}$ is calculated as

$$\begin{aligned}\mathbf{P}' &= \mathbf{T}(t_{2x}, t_{2y}) \cdot \{\mathbf{T}(t_{1x}, t_{1y}) \cdot \mathbf{P}\} \\ &= \{\mathbf{T}(t_{2x}, t_{2y}) \cdot \mathbf{T}(t_{1x}, t_{1y})\} \cdot \mathbf{P}\end{aligned}$$

Also, the composite transformation matrix for this sequence of translations is

$$\begin{bmatrix} 1 & 0 & t_{2x} \\ 0 & 1 & t_{2y} \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & t_{1x} \\ 0 & 1 & t_{1y} \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_{1x} + t_{2x} \\ 0 & 1 & t_{1y} + t_{2y} \\ 0 & 0 & 1 \end{bmatrix}$$

Or $$\mathbf{T}(t_{2x}, t_{2y}) \cdot \mathbf{T}(t_{1x}, t_{1y}) = \mathbf{T}(t_{1x} + t_{2x}, t_{1y} + t_{2y})$$

which demonstrates that two successive translations are additive.

Composite Two-Dimensional Rotations

Two successive rotations applied to a point **P** produce the transformed position.

$$\begin{aligned}\mathbf{P}' &= \mathbf{R}(\theta_2) \cdot \{\mathbf{R}(\theta_1) \cdot \mathbf{P}\} \\ &= \{\mathbf{R}(\theta_2) \cdot \mathbf{R}(\theta_1)\} \cdot \mathbf{P}\end{aligned}$$

By multiplying the two rotation matrices, we can verify that two successive rotations are additive:

$$\mathbf{R}(\theta_2) \cdot \mathbf{R}(\theta_1) = \mathbf{R}(\theta_1 + \theta_2)$$

The final rotated coordinates of a point can be calculated with the composite rotation matrix as

$$\mathbf{P}' = \mathbf{R}(\theta_1 + \theta_2) \cdot \mathbf{P}$$

Composite Two-Dimensional Scaling

Concatenating transformation matrices for two successive scaling operations in two dimensions produces the following composite scaling matrix

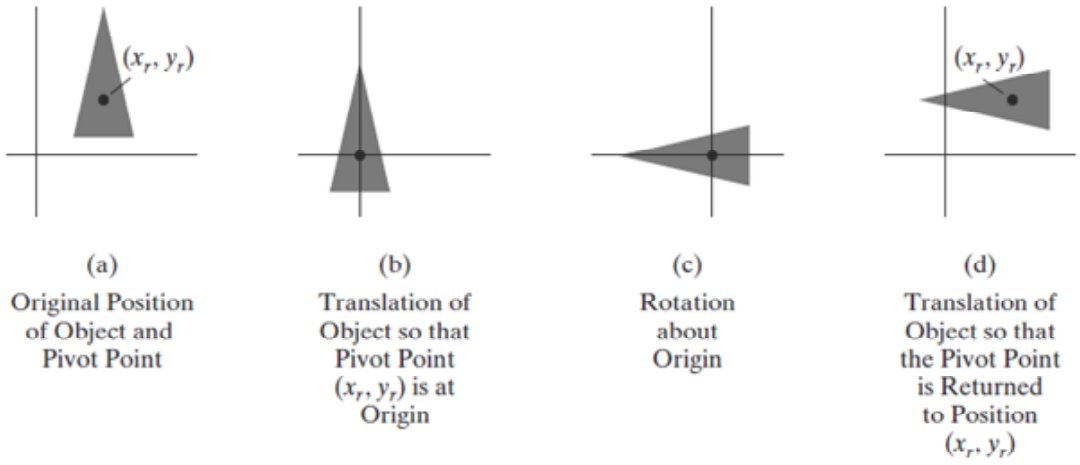
$$\begin{bmatrix} s_{2x} & 0 & 0 \\ 0 & s_{2y} & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} s_{1x} & 0 & 0 \\ 0 & s_{1y} & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} s_{1x} \cdot s_{2x} & 0 & 0 \\ 0 & s_{1y} \cdot s_{2y} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Or $\mathbf{S}(s_{2x}, s_{2y}) \cdot \mathbf{S}(s_{1x}, s_{1y}) = \mathbf{S}(s_{1x} \cdot s_{2x}, \quad s_{1y} \cdot s_{2y})$

Successive scaling operations are multiplicative.

General Two-Dimensional Pivot-Point Rotation

We can generate a two-dimensional rotation about any other pivot point (x_r , y_r) by performing the following sequence of translate-rotate-translate operations:



1. Translate the object so that the pivot-point position is moved to the coordinate origin.
2. Rotate the object about the coordinate origin.
3. Translate the object so that the pivot point is returned to its original position.

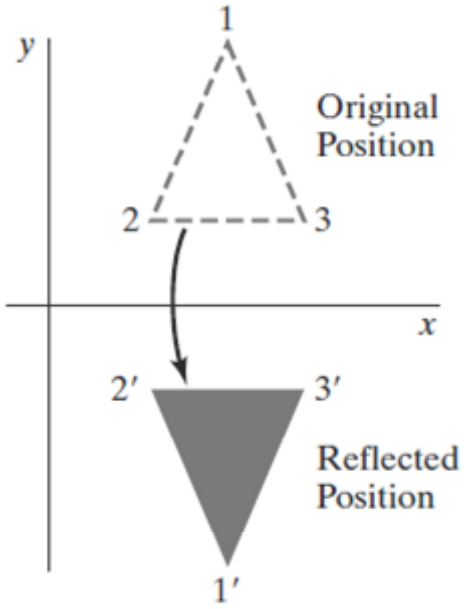
Figure 7: General pivot-point rotation

4.8 Other Two-Dimensional Transformations

In addition to the three basic transformations translation, rotation, and scaling, graphics packages offers another two transformations are reflection and shear.

Reflection

A transformation that produces a mirror image of an object is called a reflection. For a two-dimensional reflection, this image is generated relative to an axis of reflection by rotating the object 180° about the reflection axis. We can choose an axis of reflection in the xy plane or perpendicular to the xy plane. When the reflection axis is a line in the xy plane, the rotation path about this axis is in a plane perpendicular to the xy plane. For reflection axes that are perpendicular



to the xy plane, the rotation path is in the xy plane. Some examples of common reflections follow.

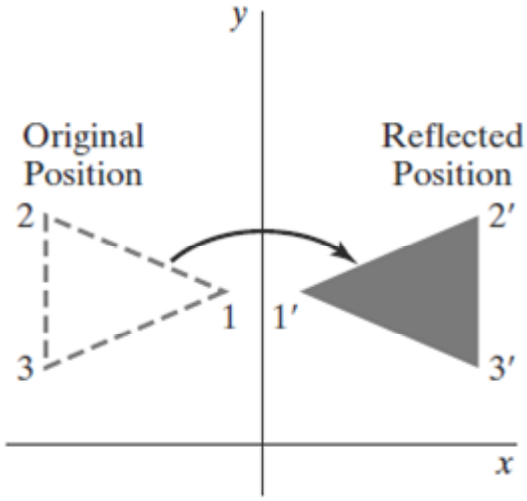
Reflection about the line y = 0 (the x axis) is accomplished with the transformation matrix

Figure 8: Reflection of an object about the x axis.

Reflection about the line $y = 0$ (the x axis) is accomplished with the transformation matrix.

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

A reflection about the line $x = 0$ (the y axis) flips x coordinates while keeping y coordinates the same. The matrix for this transformation is



$$\begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

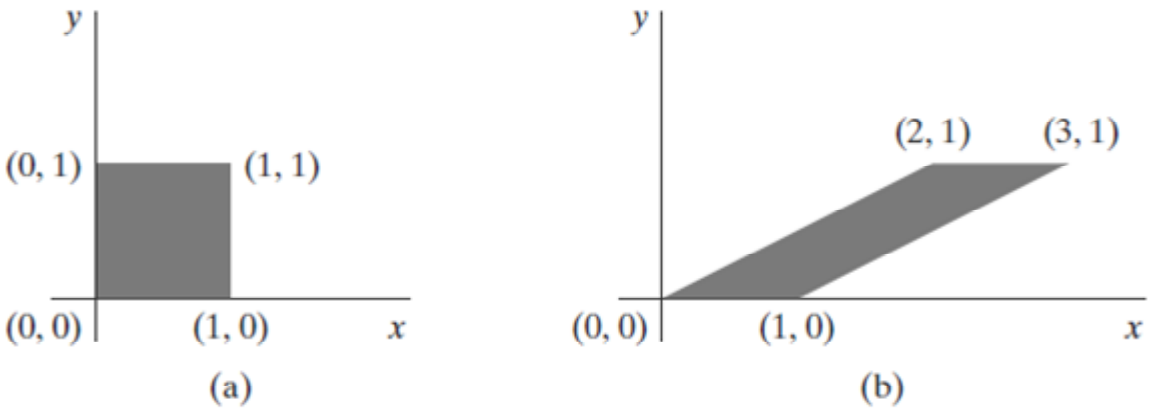
Figure 9: Reflection of an object about the y axis.

Shear

A transformation that distorts the shape of an object such that the transformed shape appears as if the object were composed of internal layers that had been caused to slide over each other is called a shear. Two common shearing transformations are those that shift coordinate x values and those that shift y values.

An x-direction shear relative to the x axis is produced with the transformation matrix.

$$\begin{bmatrix} 1 & sh_x & 0 \end{bmatrix}$$



which transforms coordinate positions as $x' = x + sh_x \cdot y, \qquad y' = y$

Figure 10: A unit square (a) is converted to a parallelogram (b) using the x -direction shear

Any real number can be assigned to the shear parameter sh_x. A coordinate position (x, y) is then shifted horizontally by an amount proportional to its perpendicular distance (y value) from the x axis. Negative values for sh_x shift coordinate positions to the left.

We can generate x-direction shears relative to other reference lines with

$$\begin{bmatrix} 1 & sh_x & -sh_x \cdot y_{\text{ref}} \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Now, coordinate positions are transformed as

$$x' = x + sh_x(y - y_{\text{ref}}), \qquad y' = y$$

A y-direction shear relative to the line $x = x_{ref}$ is generated with the transformation matrix

$$\begin{bmatrix} 1 & 0 & 0 \\ sh_y & 1 & -sh_y \cdot x_{\text{ref}} \\ 0 & 0 & 1 \end{bmatrix}$$

which generates the transformed coordinate values

$$x' = x, \qquad y' = y + sh_y(x - x_{\text{ref}})$$

This transformation shifts a coordinate position vertically by an amount proportional to its distance from the reference line $x = x_{ref}$. Shearing operations can be expressed as sequences of basic transformations. Shifts in the positions of objects relative to shearing reference lines are equivalent to translations.

4.9 Summary

The basic geometric transformations are translation, rotation, and scaling.

Translation moves an object in a straight-line path from one position to another.

Rotation moves an object from one position to another along a circular path around a specified rotation axis.

Scaling transformations change the dimensions of an object relative to a fixed position.

A three-element column matrix (vector) is referred to as a homogeneous-coordinate representation.

A transformation that produces a mirror image of an object is called a reflection.

A transformation that distorts the shape of an object such that the transformed shape appears as if the object were composed of internal layers that had been caused to slide over each other is called a shear.

4.10 Model Questions

1. What is Transformation?

2. What is shearing?

3. What is reflection?

4. Write a detailed note on the basic two dimensional transformations.

5. Explain the procedure for performing composite transformation.

LESSON – 5
TWO-DIMENSIONAL VIEWING

Structure of the lesson

5.1 Introduction

In this lesson we examine in more detail the procedures for displaying views of a two-dimensional picture on an output device. Typically, a graphics package allows a user to specify which part of a defined picture is to be displayed and where that part is to be placed on the display device. Any convenient Cartesian coordinate system, referred to as the world-coordinate reference frame, can be used to define the picture. For a two-dimensional picture, a view is selected by specifying a region of the xy plane that contains the total picture or any part of it. A user can select a single area for display, or several areas could be selected for simultaneous display or for an animated panning sequence across a scene. The picture parts within the selected areas are then mapped onto specified areas of the device coordinates. When multiple view areas are selected, these areas can be placed in separate display locations, or some areas could be inserted into other, larger display areas.

Two-dimensional viewing transformations from world to device coordinates involve translation, rotation, and scaling operations, as well as procedures for deleting those parts of the picture that are outside the limits of a selected scene area.

5.2 Objective of this Lesson

At the end of the reading the students will get knowledge in

- Two-Dimensional Viewing Pipeline
- Viewing coordinates and device coordinates
- Clipping Window
- Normalization and Viewport Transformations

5.3 Two-Dimensional Viewing Pipeline

A section of a two-dimensional scene that is selected for display is called a clipping window because all parts of the scene outside the selected section are “clipped” off. The only part of the scene that shows up on the screen is what is inside the clipping window. Sometimes the clipping window is alluded to as the world window or the viewing window. And, at one time, graphics systems referred to the clipping window simply as “the window,” but there are now so many windows in use on computers that we need to distinguish between them. For example, a window-management system can create and manipulate several areas on a video screen, each of which is called “a window,” for the display of graphics and text. So we will always use the term clipping window to refer to a selected section of a scene that is eventually converted to pixel patterns within a display window on the video monitor. Graphics packages allow us also to control the placement within the display window using another “window” called the viewport. Objects inside the clipping window are mapped to the viewport, and it is the viewport that is then positioned within the display window. The clipping window selects what we want to see; the viewport indicates where it is to be viewed on the output device.

By changing the position of a viewport, we can view objects at different positions on the display area of an output device. Multiple viewports can be used to display different sections of a scene at different screen positions. Also, by varying the size of viewports, we can change the size and proportions of displayed objects. We achieve zooming effects by successively mapping different-sized clipping windows onto a fixed-size viewport. As the clipping windows are made smaller, we zoom in on some part of a scene to view details that are not shown with the larger clipping windows. Similarly, more overview is obtained by zooming out from a section of a

scene with successively larger clipping windows. And panning effects are achieved by moving a fixed-size clipping window across the various objects in a scene.

Usually, clipping windows and viewports are rectangles in standard position, with the rectangle edges parallel to the coordinate axes. Other window or viewport geometries, such as general polygon shapes and circles, are used in some applications, but these shapes take longer to process. We first consider only rectangular viewports and clipping windows, as illustrated in Figure 1.

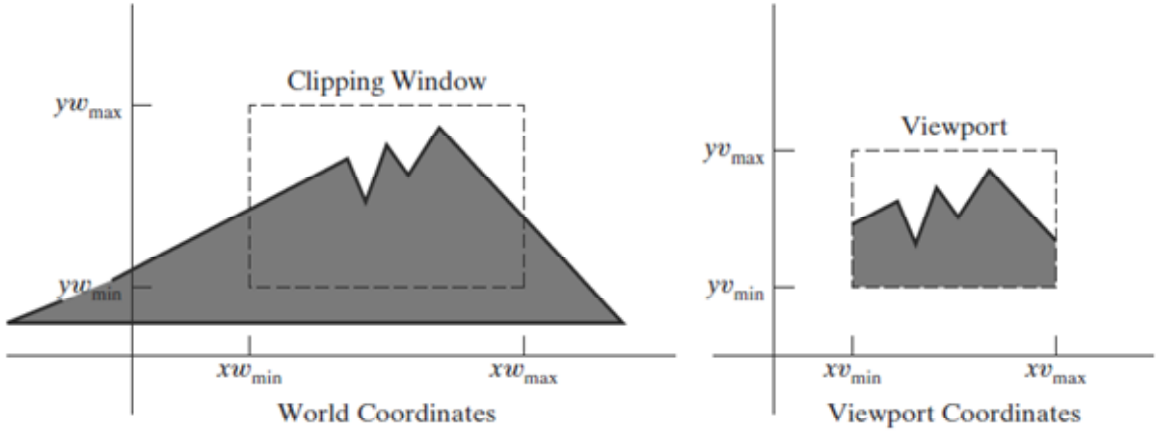


Figure 1: A clipping window and associated viewport, specified as rectangles aligned with the coordinate axes.

5.4 Two-dimensional viewing transformation

The mapping of a two-dimensional, world-coordinate scene description to device coordinates is called a two-dimensional viewing transformation. Sometimes this transformation is simply referred to as the window-to-viewport transformation or the windowing transformation. But, in general, viewing involves more than just the transformation from clipping-window coordinates to viewport coordinates. In analogy with three-dimensional viewing, we can describe the steps for two-dimensional viewing as indicated in Figure 2. Once a world-coordinate scene has been constructed, we could set up a separate two-dimensional, viewing coordinate reference frame for specifying the clipping window. But the clipping window is often just defined in world coordinates, so viewing coordinates for two-dimensional applications are the same as world

coordinates. (For a three dimensional scene, however, we need a separate viewing frame to specify the parameters for the viewing position, direction, and orientation.)

To make the viewing process independent of the requirements of any output device, graphics systems convert object descriptions to normalized coordinates and apply the clipping routines. Some systems use normalized coordinates in the range from 0 to 1, and others use a normalized range from -1 to 1. Depending upon the graphics library in use, the viewport is defined either in normalized coordinates or in screen coordinates after the normalization process. At the final step of the viewing transformation, the contents of the viewport are transferred to positions within the display window. Clipping is usually performed in normalized coordinates. This allows us to reduce computations by first concatenating the various transformation matrices.

Clipping procedures are of fundamental importance in computer graphics. They are used not only in viewing transformations, but also in window-manager systems, in painting and drawing packages to erase picture sections, and in many other applications.

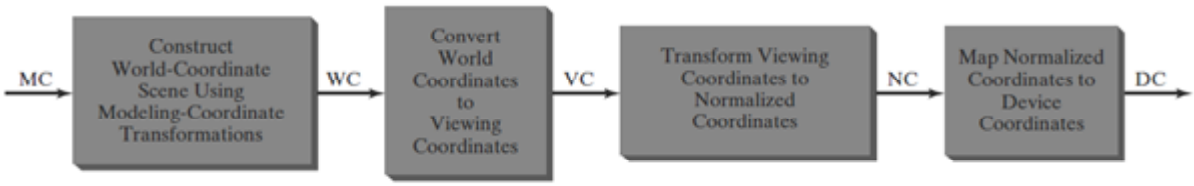


Figure 2: Two-dimensional viewing-transformation pipeline.

5.5 The Clipping Window

To achieve a particular viewing effect in an application program, we could design our own clipping window with any shape, size, and orientation we choose. For example, we might like to use a star pattern, an ellipse, or a figure with spline boundaries as a clipping window. But clipping a scene using a concave polygon or a clipping window with nonlinear boundaries requires more processing than clipping against a rectangle.We need to perform more computations to determine where an object intersects a circle than to find out where it intersects a straight line. The simplest window edges to clip against are straight lines that are parallel to the coordinate

axes. Therefore, graphics packages commonly allow only rectangular clipping windows aligned with the x and y axes.

If we want some other shape for a clipping window, then we must implement our own clipping and coordinate-transformation algorithms, or we could just edit the picture to produce a certain shape for the display frame around the scene.For example, we could trim the edges of a picture with any desired pattern by overlaying polygons that are filled with the background color. In this way, we could generate any desired border effects or even put interior holes in the picture.

Rectangular clipping windows in standard position are easily defined by giving the coordinates of two opposite corners of each rectangle. If we would like to get a rotated view of a scene, we could either define a rectangular clipping window in a rotated viewing-coordinate frame or, equivalently, we could rotate the world-coordinate scene. Some systems provide options for selecting a rotated, two-dimensional viewing frame, but usually the clipping window must be specified in world coordinates.

Viewing-Coordinate Clipping Window

A general approach to the two-dimensional viewing transformation is to set up a viewing-coordinate system within the world-coordinate frame. This viewing frame provides a reference for specifying a rectangular clipping window with any selected orientation and position, as in Figure 3. To obtain a view of the world coordinate scene as determined by the clipping window of Figure 3, we just need to transfer the scene description to viewing coordinates. Although many graphics packages do not provide functions for specifying a clipping window in a two-dimensional viewing-coordinate system, this is the standard approach for defining a clipping region for a three-dimensional scene.

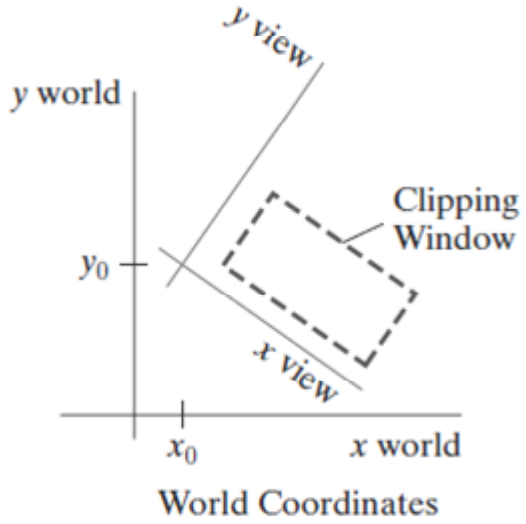


Figure 3: A rotated clipping window defined in viewing coordinates.

We choose an origin for a two-dimensional viewing-coordinate frame at some world position $P_0 = (x_0, y_0)$, and we can establish the orientation using a world vector V that defines the y_{view} direction. Vector V is called the two-dimensional view up vector. An alternative method for specifying the orientation of the viewing frame is to give a rotation angle relative to either the x or y axis in the world frame. From this rotation angle, we can then obtain the view up vector. Once we have established the parameters that define the viewing-coordinate frame, we transform the scene description to the viewing system. This involves a sequence of transformations equivalent to superimposing the viewing frame on the world frame.

The first step in the transformation sequence is to translate the viewing origin to the world origin. Next, we rotate the viewing system to align it with the world frame. Given the orientation vector **V**, we can calculate the components of unit vectors **v** = (v_x, v_y) and **u** = (u_x, u_y) for the y_{view} and x_{view} axes, respectively. These unit vectors are used to form the first and second rows of the rotation matrix **R** that aligns the viewing $x_{view}y_{view}$ axes with the world $x_w y_w$ axes.

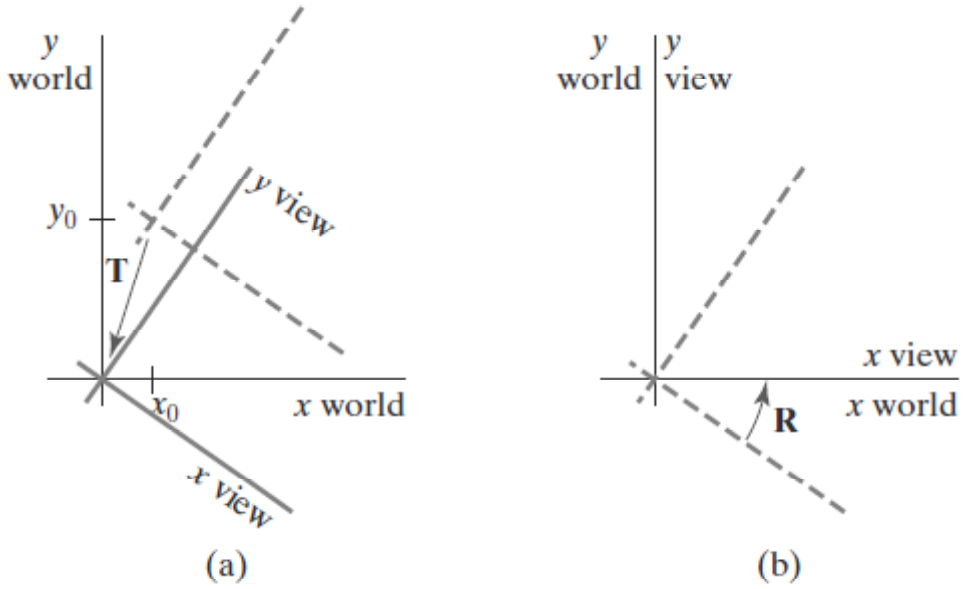


Figure 4: A viewing-coordinate frame is moved into coincidence with the world frameby (a) applying a translation matrix T to move the viewing origin to the world origin, then (b) applying a rotation matrix R to align the axes of the two systems.

Object positions in world coordinates are then converted to viewing coordinates with the composite two-dimensional transformation matrix.

$$\mathbf{M}_{WC,VC} = \mathbf{R} \cdot \mathbf{T}$$

Where **T** is the translation matrix that takes the viewing origin $\mathbf{P}_0$ to the world origin, and **R** is the rotation matrix that rotates the viewing frame of reference into coincidence with the world-coordinate system. Figure 4 illustrates the steps in this coordinate transformation.

World-Coordinate Clipping Window

A routine for defining a standard, rectangular clipping window in world coordinates is typically provided in a graphics-programming library. We simply specify two world-coordinate positions, which are then assigned to the two opposite corners of a standard rectangle. Once the clipping window has been established, the scene description is processed through the viewing routines to the output device.

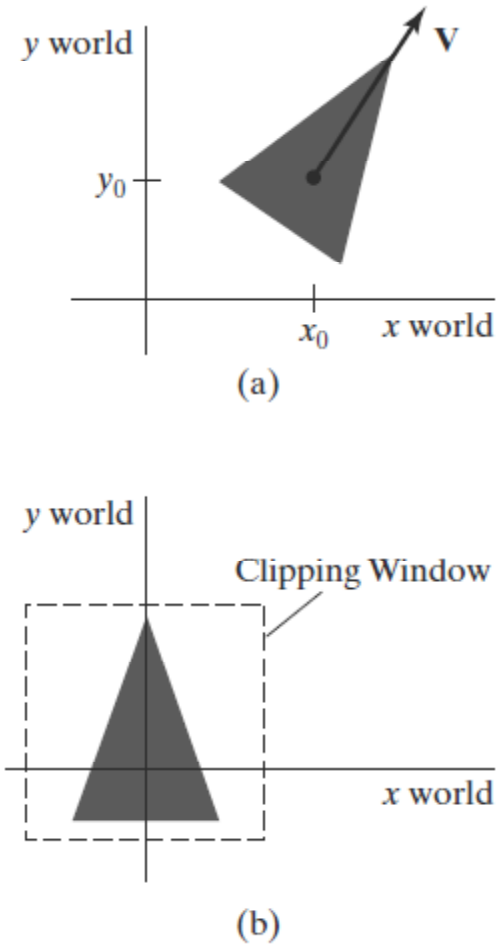


Figure 5: A triangle (a), with a selected reference point and orientation vector, is translated and rotated to position (b) within a clipping window.

If we want to obtain a rotated view of a two-dimensional scene, as discussed in the previous section, we perform exactly the same steps as described there, but without considering a viewing frame of reference. Thus, we simply rotate (and possibly translate) objects to a desired position and set up the clipping window all in world coordinates. For example, we could display a rotated view of the triangle in Figure 5(a) by rotating it into the position we want and setting up a standard clipping rectangle. In analogy with the coordinate transformation described in the previous section, we could also translate the triangle to the world origin and define a clipping window around the triangle. In that case, we define an orientation vector and choose a reference point such as the triangle's centroid.

Then we translate the reference point to the world origin and rotate the orientation vector onto the yworld axis using transformation matrix 1. With the triangle in the desired orientation, we can use a standard clipping window in world coordinates to capture the view of the rotated triangle. The transformed position of the triangle and the selected clipping window are shown in Figure 5(b).

5.6 Normalization and Viewport Transformations

With some graphics packages, the normalization and window-to-viewport transformations are combined into one operation. In this case, the viewport coordinates are often given in the range from 0 to 1 so that the viewport is positioned within a unit square. After clipping, the unit square containing the viewport is mapped to the output display device. In other systems, the normalization and clipping routines are applied before the viewport transformation. For these systems, the viewport boundaries are specified in screen coordinates relative to the display window position.

Mapping the Clipping Window into a Normalized Viewport

To illustrate the general procedures for the normalization and viewport transformations, we first consider a viewport defined with normalized coordinate values between 0 and 1. Object descriptions are transferred to this normalized space using a transformation that maintains the same relative placement of a point in the viewport as it had in the clipping window. If a coordinate position is at the center of the clipping window, for instance, it would be mapped to the center of the viewport. Figure 6 illustrates this window-to-viewport mapping. Position (xw, yw) in the clipping window is mapped to position (xv, yv) in the associated viewport.

To transform the world-coordinate point into the same relative position within the viewport, we require that

$$\frac{xv - xv_{\min}}{xv_{\max} - xv_{\min}} = \frac{xw - xw_{\min}}{xw_{\max} - xw_{\min}}$$
$$\frac{yv - yv_{\min}}{yv_{\max} - yv_{\min}} = \frac{yw - yw_{\min}}{yw_{\max} - yw_{\min}}$$

Solving these expressions for the viewport position (*xv*, *yv*), we have

$$xv = s_x xw + t_x$$
$$yv = s_y yw + t_y$$

where the scaling factors are

$$s_x = \frac{xv_{\max} - xv_{\min}}{xw_{\max} - xw_{\min}}$$
$$s_y = \frac{yv_{\max} - yv_{\min}}{yw_{\max} - yw_{\min}}$$

and the translation factors are

$$t_x = \frac{xw_{\max} xv_{\min} - xw_{\min} xv_{\max}}{xw_{\max} - xw_{\min}}$$
$$t_y = \frac{yw_{\max} yv_{\min} - yw_{\min} yv_{\max}}{yw_{\max} - yw_{\min}}$$

We could obtain the transformation from world coordinates to viewport coordinates with the following sequence:

1. Scale the clipping window to the size of the viewport using a fixed-point position of (xw_{min}, yw_{min}).

2. Translate (xw_{min}, yw_{min}) to (xv_{min}, yv_{min}).

The window-to-viewport transformation maintains the relative placement of object descriptions. An object inside the clipping window is mapped to a corresponding position inside the viewport.

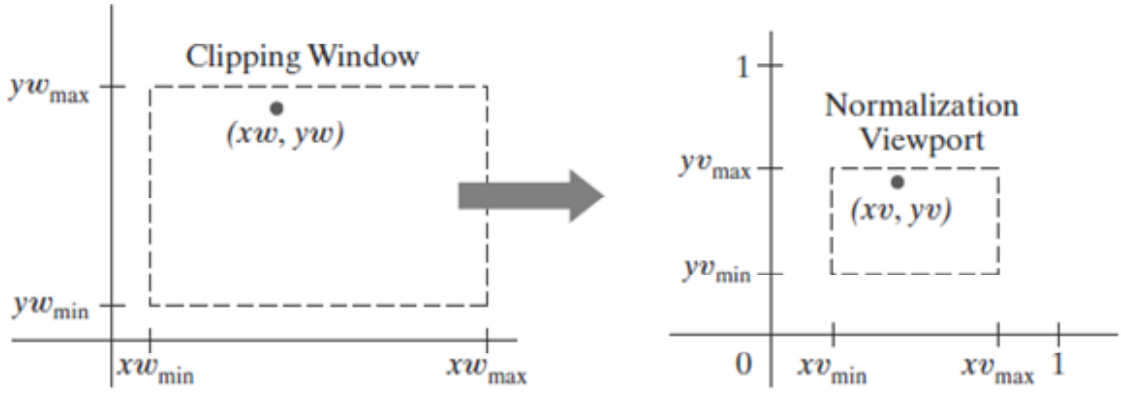


Figure 6: A point (xw, yw) in a world-coordinate clipping window is mapped to viewport coordinates (xv, yv), within a unit square, so that the relative positions of the two points in their respective rectangles are the same.

The scaling transformation in step (1) can be represented with the two-dimensional matrix

$$\mathbf{S} = \begin{bmatrix} s_x & 0 & xw_{\min}(1 - s_x) \\ 0 & s_y & yw_{\min}(1 - s_y) \\ 0 & 0 & 1 \end{bmatrix}$$

The two-dimensional matrix representation for the translation of the lower-left corner of the clipping window to the lower-left viewport corner is

$$\mathbf{T} = \begin{bmatrix} 1 & 0 & xv_{\min} - xw_{\min} \\ 0 & 1 & yv_{\min} - yw_{\min} \\ 0 & 0 & 1 \end{bmatrix}$$

And the composite matrix representation for the transformation to the normalized viewport is

$$\mathbf{M}_{\text{window, normviewp}} = \mathbf{T} \cdot \mathbf{S} = \begin{bmatrix} s_x & 0 & t_x \\ 0 & s_y & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

Any other clipping-window reference point, such as the top-right corner or the window center, could be used for the scale–translate operations. Alternatively, we could first translate any clipping window position to the corresponding location in the viewport, and then scale relative to that viewport location.

The window-to-viewport transformation maintains the relative placement of object descriptions. An object inside the clipping window is mapped to a corresponding position inside the viewport. Similarly, an object outside the clipping window is outside the viewport. Relative proportions of objects, on the other hand, are maintained only if the aspect ratio of the viewport is the same as the aspect ratio of the clipping window. In other words, we keep the same object proportions if the scaling factors s_x and s_y are the same. Otherwise, world objects will be stretched or contracted in either the x or y directions (or both) when displayed on the output device.

The clipping routines can be applied using either the clipping-window boundaries or the viewport boundaries. After clipping, the normalized coordinates are transformed into device coordinates. And the unit square can be mapped onto the output device using the same procedures as in the window-to-viewport transformation, with the area inside the unit square transferred to the total display area of the output device.

Mapping the Clipping Window into a Normalized Square

Another approach to two-dimensional viewing is to transform the clipping window into a normalized square, clip in normalized coordinates, and then transfer the scene description to a viewport specified in screen coordinates. This transformation is illustrated in Figure 7 with normalized coordinates in the range from -1 to 1. The clipping algorithms in this transformation sequence are now standardized so that objects outside the boundaries x = ±1 and y = ±1 are detected and removed from the scene description. At the final step of the viewing transformation, the objects in the viewport are positioned within the display window.

We transfer the contents of the clipping window into the normalization square using the same procedures as in the window-to-viewport transformation. The matrix for the normalization transformation is obtained by substituting -1 for xv_{min} and yv_{min} and substituting +1 for xv_{max} and yv_{max}.

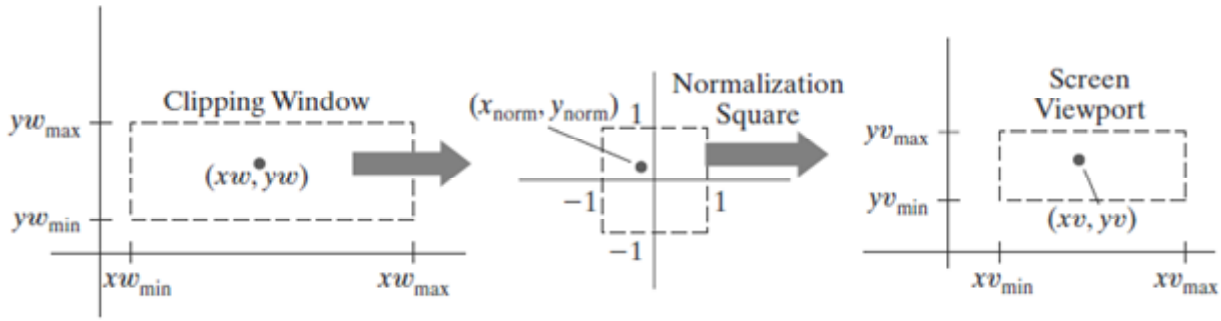


Figure 7: A point (xw, yw) in a clipping window is mapped to a normalized coordinate position (x_{norm}, y_{norm}), then to a screen-coordinate position (xv, yv) in a viewport. Objects are clipped against the normalization square before the transformation to viewport coordinates occurs.

Making these substitutions in the expressions for t_x, t_y, s_x, and s_y, we have

$$\mathbf{M}_{\text{window, normsquare}} = \begin{bmatrix} \frac{2}{xw_{\max} - xw_{\min}} & 0 & -\frac{xw_{\max} + xw_{\min}}{xw_{\max} - xw_{\min}} \\ 0 & \frac{2}{yw_{\max} - yw_{\min}} & -\frac{yw_{\max} + yw_{\min}}{yw_{\max} - yw_{\min}} \\ 0 & 0 & 1 \end{bmatrix}$$

Similarly, after the clipping algorithms have been applied, the normalized square with edge length equal to 2 is transformed into a specified viewport. This time, we get the transformation matrix from Equation 8 by substituting -1 for xw_{min} and yw_{min} and substituting +1 for xw_{max} and yw_{max}:

$$\mathbf{M}_{\text{normsquare, viewport}} = \begin{bmatrix} \frac{xv_{\max} - xv_{\min}}{2} & 0 & \frac{xv_{\max} + xv_{\min}}{2} \\ 0 & \frac{yv_{\max} - yv_{\min}}{2} & \frac{yv_{\max} + yv_{\min}}{2} \\ 0 & 0 & 1 \end{bmatrix}$$

The last step in the viewing process is to position the viewport area in the display window. Typically, the lower-left corner of the viewport is placed at a coordinate position specified relative to the lower-left corner of the display window. Figure 8 demonstrates the positioning of a viewport within a display window.

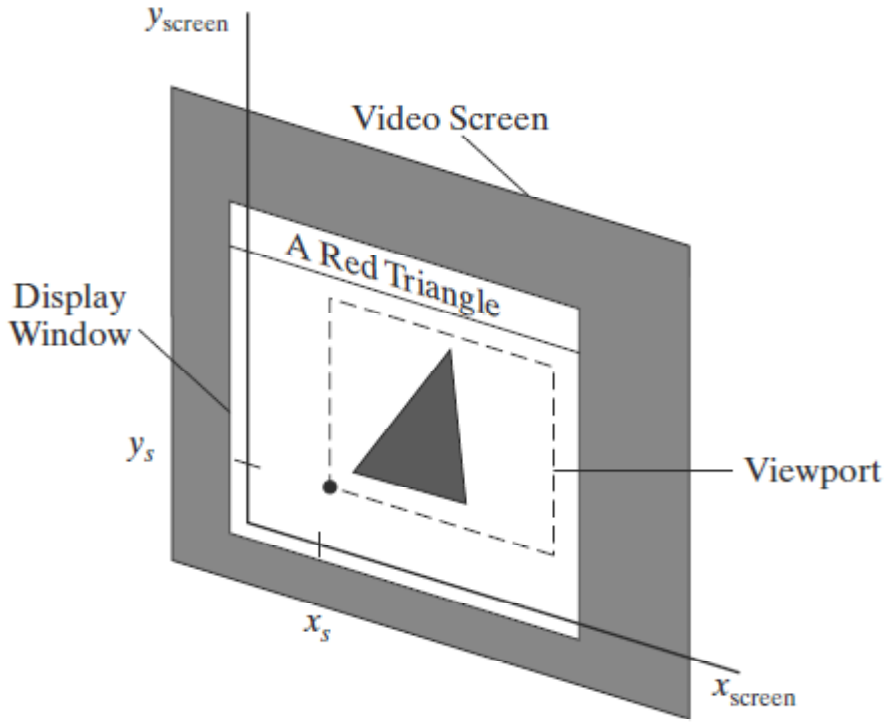


Figure 8: A viewport at coordinate position (x_s , y_s) within a display window

5.7 Summary

The two-dimensional viewing-transformation pipeline is a series of operations that result in the display of a world-coordinate picture that has been defined in the xy plane. After we construct the scene, it can be mapped to a viewing-coordinate reference frame, then to a normalized coordinate system where clipping routines can be applied. Finally, the scene is transferred to device coordinates for display.

Normalized coordinates can be specified in the range from 0 to 1 or in the range from -1 to 1.

We select part of a scene for display on an output device using a clipping window, which can be described in the world-coordinate system or in a viewing coordinate frame defined relative to world coordinates.

The contents of the clipping window are transferred to a viewport for display on an output device.

The clipping window and viewport are rectangles whose edges are parallel to the coordinate axes. An object is mapped to the viewport so that it has the same relative position in the viewport as it has in the clipping window.

To maintain the relative proportions of an object, the viewport must have the same aspect ratio as the corresponding clipping window.

5.8 Model Question

1. Define viewing transformation.
2. Explain in detail about window to viewport coordinate transformation.
3. What is a clipping window?
4. Define world coordinate and device coordinate.

LESSON – 6
CLIPPING ALGORITHM

Structure of the lesson

6.1 Introduction

Generally, any procedure that eliminates those portions of a picture that are either inside or outside a specified region of space is referred to as a clipping algorithm or simply clipping. Usually a clipping region is a rectangle in standard position, although we could use any shape for a clipping application. The most common application of clipping is in the viewing pipeline, where clipping is applied to extract a designated portion of a scene (either two-dimensional or three-dimensional) for display on an output device. Clipping methods are also used to antialias object boundaries, to construct objects using solid-modeling methods, to manage a multiwindow environment, and to allow parts of a picture to be moved, copied, or erased in drawing and painting programs.

Clipping algorithms are applied in two-dimensional viewing procedures to identify those parts of a picture that are within the clipping window. Everything outside the clipping window is

then eliminated from the scene description that is transferred to the output device for display. An efficient implementation of clipping in the viewing pipeline is to apply the algorithms to the normalized boundaries of the clipping window. This reduces calculations, because all geometric and viewing transformation matrices can be concatenated and applied to a scene description before clipping is carried out. The clipped scene can then be transferred to screen coordinates for final processing.

6.2 Objective of this Lesson

In the following sections, we explore two-dimensional algorithms for

- Point clipping
- Line clipping (straight-line segments)
- Fill-area clipping (polygons)
- Curve clipping
- Text clipping

The students will get better understanding about the clipping procedure.

6.3 Point Clipping

Two-dimensional point P = (x, y) is saved or clipped if the following inequalities are satisfied:

$$xw_{\min} \leq x \leq xw_{\max}$$
$$yw_{\min} \leq y \leq yw_{\max}$$

If any of these four inequalities is not satisfied, the point is clipped.

6.4 Line Clipping

A line-clipping algorithm processes each line in a scene through a series of tests and intersection calculations to determine whether the entire line or any part of it is to be saved. The expensive part of a line-clipping procedure is in calculating the intersection positions of a line with the window edges. We must perform the following tests

- Whether a line segment is completely inside.
- Whether a line segment is completely outside.
- Whether any part of the line crosses the window interior.

It is easy to determine whether a line is completely inside a clipping window, but it is difficult to identify all lines that are entirely outside the window. If a line is not completely inside or completely outside a clipping rectangle, we must then perform intersection calculations to determine whether any part of the line crosses the window interior.

To determine if a line segment is completely inside or outside a selected clipping-window edge perform the following. When both endpoints of a line segment are inside all four clipping boundaries, such as the line fromP_1 to P_2 in Figure 1, the line is completely inside the clipping window and we save it. And when both endpoints of a line segment are outside any one of the four boundaries (as with line $P_3 P_4$ in Figure 1), that line is completely outside the window and it is eliminated from the scene description. But if both these tests fail, the line segment intersects at least one clipping boundary and it may or may not cross into the interior of the clipping window.

One way to formulate the equation for a straight-line segment is to use the following parametric representation, where the coordinate positions (x_0, y_0) and (x_{end}, y_{end}) designate the two line endpoints:

$$x = x_0 + u(x_{\text{end}} - x_0)$$
$$y = y_0 + u(y_{\text{end}} - y_0) \qquad 0 \le u \le 1$$

We can use this parametric representation to determine where a line segment crosses each clipping-window edge by assigning the coordinate value for that edge to either *x* or *y* and solving for parameter *u*. For example, the left window boundary is at position xw_{min}, so we substitute this value for *x*, solve for *u*, and calculate the corresponding *y*-intersection value. If this value of *u* is outside the range from 0 to 1, the line segment does not intersect that window border line.

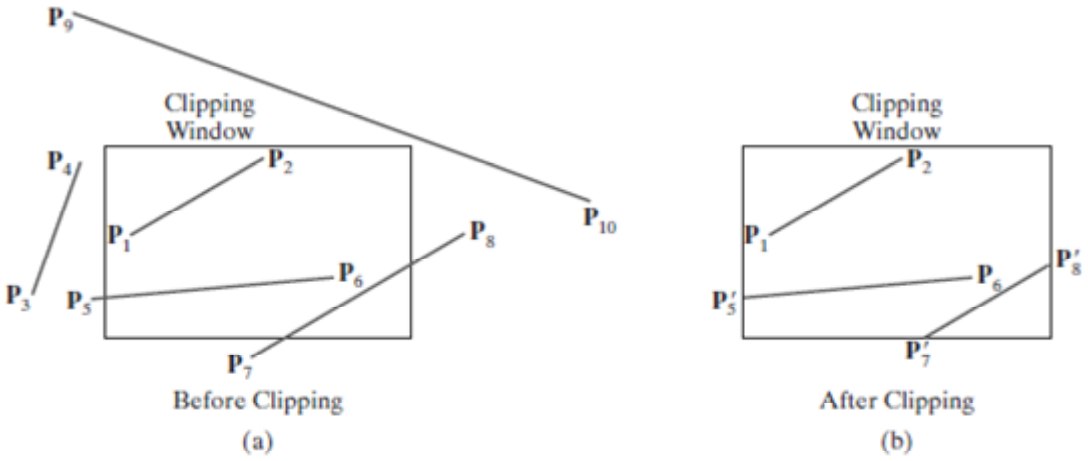


Figure 1: Clipping straight-line segments using a standard rectangular clipping window.

A major goal for any line-clipping algorithm is to minimize the intersection calculations. A number of faster line clippers have been developed.

Major Line clipping algorithm

1. Cohen-Sutherland Line Clipping
2. Liang-Barsky Line Clipping
3. Nicholl-Lee-Nicholl Line Clipping

Cohen-Sutherland Line Clipping

One of the earliest algorithms to be developed for fast line clipping. Cohen-Sutherland method performs more tests before proceeding to the intersection calculations. Initially, every line endpoint in a picture is assigned a four-digit binary value, called a **region code,** and each bit position is used to indicate whether the point is inside or outside one of the clipping-window boundaries.

Figure 2 illustrates one possible ordering with the bit positions numbered 1 through 4 from right to left. The rightmost position (bit 1) references the left clipping-window boundary, and the leftmost position (bit 4) references the top window boundary. A value of 1 (or *true*) in any bit position indicates that the endpoint is outside that window border. Similarly, a value of 0 (or

false) in any bit position indicates that the endpoint is not outside (it is inside or on) the corresponding window edge. The four window borders create nine regions, and Figure 3 lists the value for the binary code in each of these regions. Thus, an endpoint that is below and to the left of the clipping window is assigned the region code 0101, and the region-code value for any endpoint inside the clipping window is 0000.

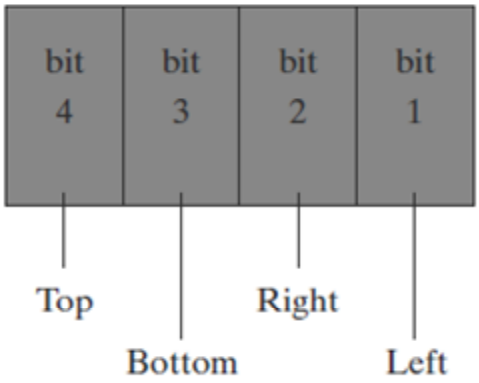


Figure 2: A possible ordering for the clipping window boundaries corresponding to the bit positions in the Cohen- Sutherland endpoint region code.

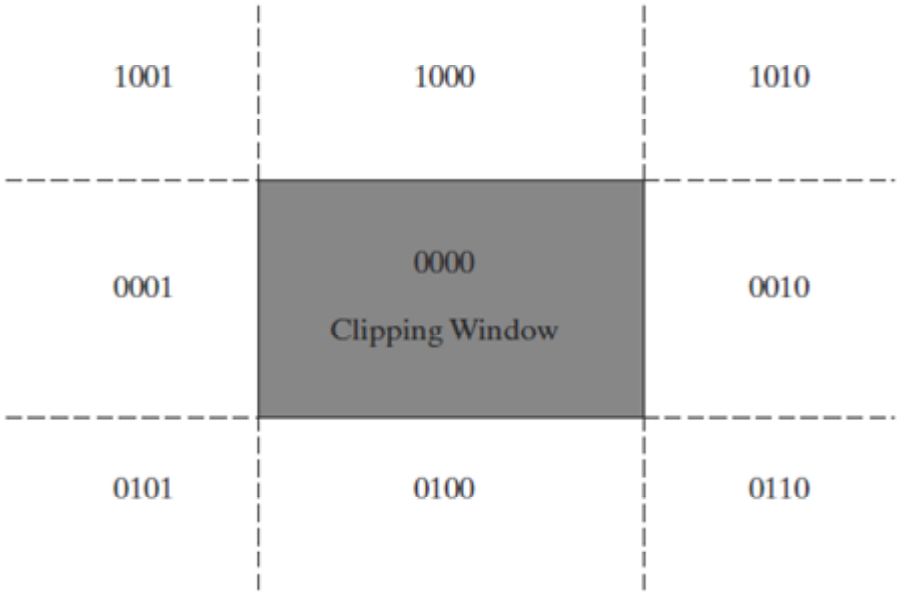


Figure 3: The nine binary region codes

We can determine the values for a region-code more efficiently using bit-processing operations and the following two steps:

(1) Calculate differences between endpoint coordinates and clipping boundaries.

(2) Use the resultant sign bit of each difference calculation to set the corresponding value in the region code.

bit 1 is the sign bit of $x - xw_{min}$;

bit 2 is the sign bit of $xw_{max} - x$;

bit 3 is the sign bit of $y - yw_{min}$; and

bit 4 is the sign bit of $yw_{max} - y$.

The inside-outside tests for line segments using logical operators. When the *or* operation between two endpoint region codes for a line segment is *false* (0000), the line is inside the clipping window. Therefore, we save the line and proceed to test the next line in the scene description. When the *and* operation between the two endpoint region codes for a line is *true* (not 0000), the line is completely outside the clipping window, and we can eliminate it from the scene description.

The following figure (Figure 4) illustrates two line segments that cannot be identified immediately as completely inside or completely outside the clipping window. The region codes for the line from $\mathbf{P}_1$ to $\mathbf{P}_2$ are 0100 and 1001. Thus, $\mathbf{P}_1$ is inside the left clipping boundary and $\mathbf{P}_2$ is outside that boundary. We then calculate the intersection position $\mathbf{P}'_2$, and we clip off the line section from $\mathbf{P}_2$ to $\mathbf{P}'_2$. Endpoint $\mathbf{P}_1$ is below the bottom clipping edge and $\mathbf{P}'_2$ is above it, so we determine the intersection position at this boundary ($\mathbf{P}'_1$).We eliminate the line section from $\mathbf{P}_1$ to $\mathbf{P'}_1$ and proceed to the top window edge. There we determine the intersection position to be $\mathbf{P}'_2$. The final step is to clip off the section above the top boundary and save the interior segment from $\mathbf{P}'_1$ to $\mathbf{P}'_2$. For the second line, we find that point $\mathbf{P}_3$ is outside the left boundary and $\mathbf{P}_4$ is inside. Thus, we calculate the intersection position $\mathbf{P}'_3$ and eliminate the line section from $\mathbf{P}_3$ to $\mathbf{P}'_3$. By checking region codes for the endpoints $\mathbf{P}'_3$ and $\mathbf{P}_4$.

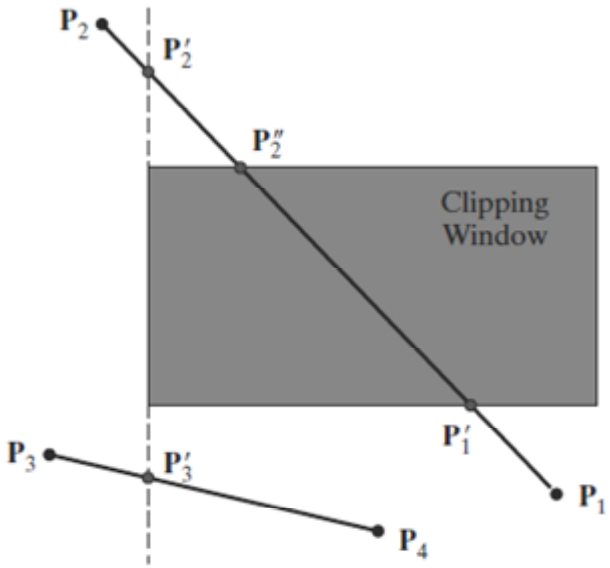


Figure 4: Two line segments with all possible cases.

To determine a boundary intersection for a line segment, we can use the slope intercept form of the line equation. For a line with endpoint coordinates (x_0, y_0) and (x_{end}, y_{end}), the y coordinate of the intersection point with a vertical clipping border line can be obtained with the calculation

$$y = y_0 + m(x - x_0)$$

where the x value is set to either xw_{min} or xw_{max}, and the slope of the line is calculated as $m = (y_{end} - y_0)/(x_{end} - x_0)$. Similarly, for the intersection with a horizontal border, the x coordinate can be calculated as

$$x = x_0 + \frac{y - y_0}{m}$$

Liang-Barsky Line Clipping

Liang-Barsky Line Clipping is a form of the parametric line-clipping algorithm. The working procedure is as follows:

For a line segment with endpoints (x_0, y_0) and (x_{end}, y_{end}), we can describe the line with the parametric form

$$x = x_0 + u\Delta x$$
$$y = y_0 + u\Delta y \qquad 0 \le u \le 1$$

where $\Delta x = x_{end} - x_0$ and $\Delta y = y_{end} - y_0$. In the Liang-Barsky algorithm, the parametric line equations are combined with the point-clipping conditions to obtain the inequalities

$$xw_{\min} \le x_0 + u\Delta x \le xw_{\max}$$
$$yw_{\min} \le y_0 + u\Delta y \le yw_{\max}$$

which can be expressed as

$$u\ p_k \le q_k, \qquad k = 1, 2, 3, 4$$

where parameters p and q are defined as

$$p_1 = -\Delta x, \qquad q_1 = x_0 - xw_{\min}$$
$$p_2 = \Delta x, \qquad q_2 = xw_{\max} - x_0$$
$$p_3 = -\Delta y, \qquad q_3 = y_0 - yw_{\min}$$
$$p_4 = \Delta y, \qquad q_4 = yw_{\max} - y_0$$

Any line that is parallel to one of the clipping-window edges has $pk = 0$ for the value of k corresponding to that boundary, where $k = 1$, 2, 3, and 4 correspond to the left, right, bottom, and top boundaries, respectively.

For that value of k, we also find $qk < 0$, then the line is completely outside the boundary and can be eliminated from further consideration.

If $qk \geq 0$, the line is inside the parallel clipping border.

When $pk < 0$, the infinite extension of the line proceeds from the outside to the inside.

If $pk > 0$, the line proceeds from the inside to the outside.

Nicholl-Lee-Nicholl Line Clipping

By creating more regions around the clipping window, the Nicholl-Lee-Nicholl (NLN) algorithm avoids multiple line-intersection calculations. Initial testing to determine whether a line segment is completely inside the clipping window or outside the window limits can be accomplished with region code.

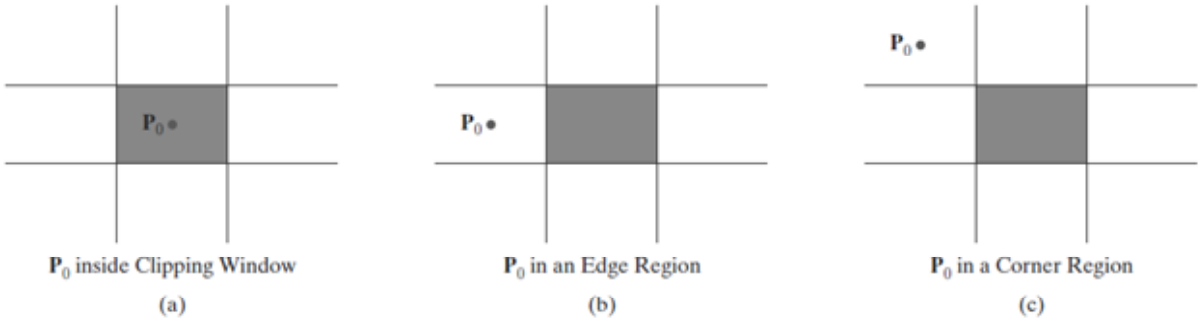


Figure 5: Three possible positions for a line endpoint P_0 in the NLN line-clipping algorithm.

For a line with endpoints P_0 and P_{end}, we first determine the position of point P_0 for the nine possible regions relative to the clipping window. Only the three regions shown in Figure 5 need be considered. If P_0 lies in any one of the other six regions, we can move it to one of the three regions using a symmetry transformation. The region directly above the clip window can be transformed to the region left of the window using a reflection about the line y = -x, or we could use a 90º counter clockwise rotation.

Assuming that P_0 and P_{end} are not both inside the clipping window, we next determine the position of P_{end} relative toP_0.To do this, we create some new regions in the plane, depending on the location of P_0. Boundaries of the new regions are semi-infinite line segments that start at the position of P_0 and pass through the clipping-window corners. If P_0 is inside the clipping

window, we set up the four regions. Then, depending on which one of the four regions (L, T, R, or B) contains P_{end}, we compute the line-intersection position with the corresponding window boundary.

If P_0 is in the region to the left of the window, we set up the four regions labelled L, LT, LR, and LB in Figure 7. These four regions again determine a unique clipping-window edge for the line segment, relative to the position of P_{end}. If P_{end} is in any one of the three regions labeled L, we clip the line at the left window border and save the line segment from this intersection point to P_{end}.

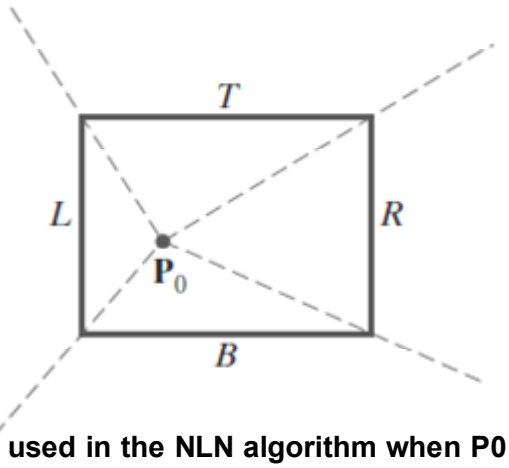


Figure 6: The four regions used in the NLN algorithm when P0 is inside the clipping window and Pend is outside.

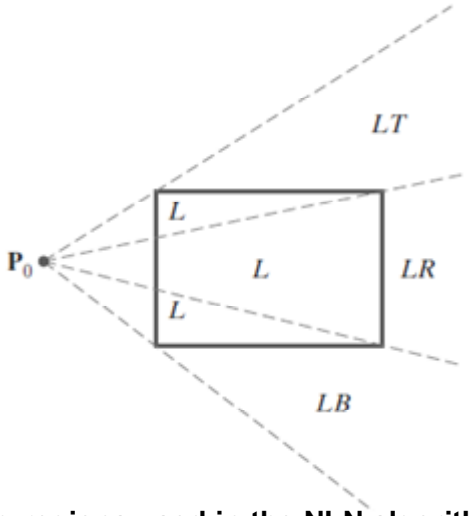


Figure 7: The four clipping regions used in the NLN algorithm when P0 is directly to the left of the clip window.

For example, if P_0 is left of the clipping window (Figure 7), then P_{end} is in region LT if

$$\text{slope}\overline{P_0P_{TR}} < \text{slope}\overline{P_0P_{\text{end}}} < \text{slope}\overline{P_0P_{TL}}$$

or

$$\frac{y_T - y_0}{x_R - x_0} < \frac{y_{\text{end}} - y_0}{x_{\text{end}} - x_0} < \frac{y_T - y_0}{x_L - x_0}$$

We clip the entire line if

$$(y_T - y_0)(x_{\text{end}} - x_0) < (x_L - x_0)(y_{\text{end}} - y_0)$$

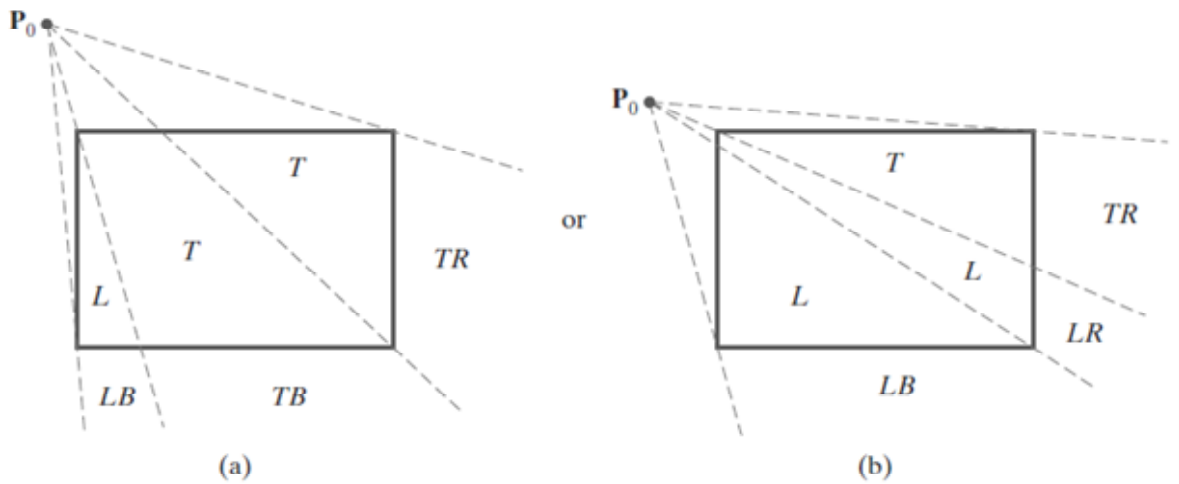


Figure 8: The two possible sets of clipping regions used in the NLN algorithm when P0 is above and to the left of the clipping window.

6.5 Polygon Clipping

We can process a polygon fill area against the borders of a clipping window using the same general approach as in line clipping. We can clip a polygon fill area by determining the new shape for the polygon as each clipping-window edge is processed, as demonstrated in Figure 10.

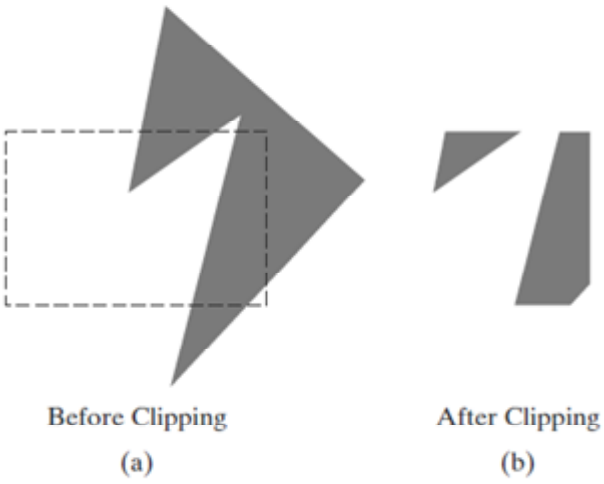


Figure 9: Display of a correctly clipped polygon fill area.

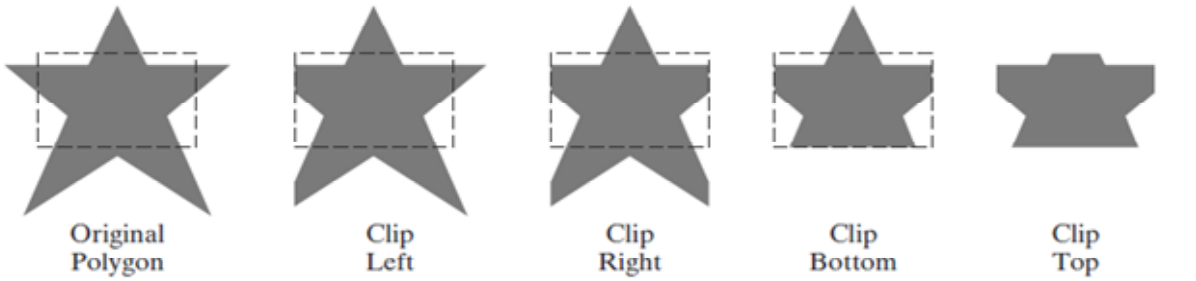


Figure 10: Processing a polygon fill area against successive clipping-window boundaries.

Sutherland—Hodgman Polygon Clipping

An efficient method for clipping a convex-polygon fill area, developed by Sutherland and Hodgman, is to send the polygon vertices through each clipping stage so that a single clipped vertex can be immediately passed to the next stage. The general strategy in this algorithm is to send the pair of endpoints for each successive polygon line segment through the series of clippers (left, right, bottom, and top).

There are four possible cases that need to be considered when processing a polygon edge against one of the clipping boundaries. One possibility is that the first edge endpoint is

outside the clipping boundary and the second endpoint is inside. Or, both endpoints could be inside this clipping boundary. Another possibility is that the first endpoint is inside the clipping boundary and the second endpoint is outside. And, finally, both endpoints could be outside the clipping boundary.

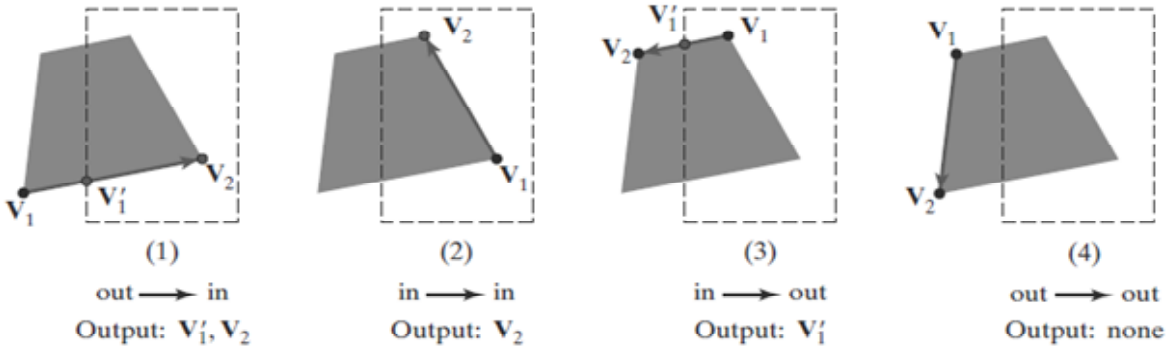


Figure 11: Four possible outputs generated by the left clipper

As each successive pair of endpoints is passed to one of the four clippers, an output is generated for the next clipper according to the results of the following tests:

1. If the first input vertex is outside this clipping-window border and the second vertex is inside, both the intersection point of the polygon edge with the window border and the second vertex are sent to the next clipper.
2. If both input vertices are inside this clipping-window border, only the second vertex is sent to the next clipper.
3. If the first vertex is inside this clipping-window border and the second vertex is outside, only the polygon edge-intersection position with the clipping-window border is sent to the next clipper.
4. If both input vertices are outside this clipping-window border, no vertices are sent to the next clipper.

Figure 12 provides an example of the Sutherland-Hodgman polygon clipping algorithm for a fill area defined with the vertex set {1, 2, 3}. As soon as a clipper receives a pair of endpoints, it determines the appropriate output.

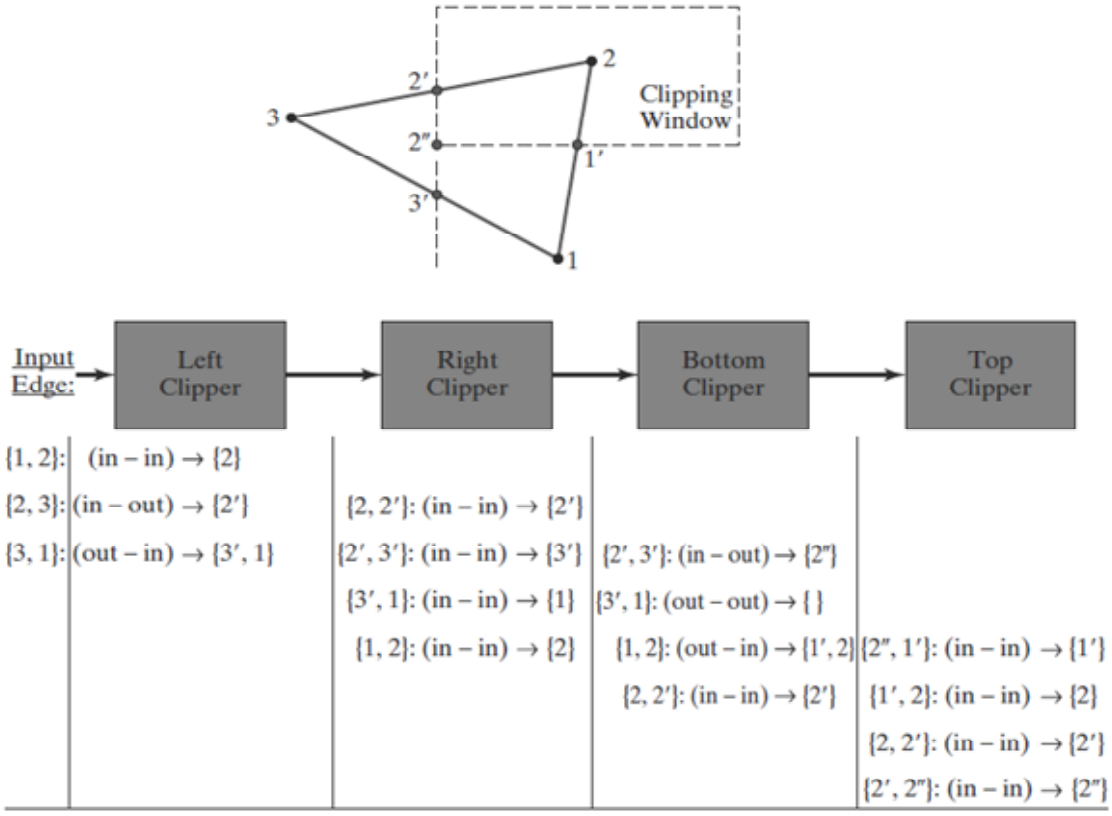


Figure 12: Processing a set of polygon vertices, {1, 2, 3}, through the boundary clippers using the Sutherland-Hodgman algorithm.

Weiler-Atherton Polygon Clipping

This algorithm provides a general polygon-clipping approach that can be used to clip a fill area that is either a convex polygon or a concave polygon. we could also use this approach to clip any polygon fill area against a clipping window with any polygon shape. Weiler-Atherton algorithm traces around the perimeter of the fill polygon searching for the borders that enclose a clipped fill region. In this way, multiple fill regions can be identified and displayed as separate, unconnected polygons. To find the edges for a clipped fill area, we follow a path (either counter clockwise or clockwise) around the fill area that detours along a clipping-window boundary whenever a polygon edge crosses to the outside of that boundary. The direction of a detour at a clipping-window border is the same as the processing direction for the polygon edges.

Determine whether the processing direction is counter clockwise. In most cases, the vertex list is specified in a counter clockwise order or clockwise.

For a counter clockwise traversal of the polygon fill-area vertices, we apply the following Weiler-Atherton procedures:

1. Process the edges of the polygon fill area in a counter clockwise order until an inside-outside pair of vertices is encountered for one of the clipping boundaries;
2. Follow the window boundaries in a counter clockwise direction from the exit-intersection point to another intersection point with the polygon.
3. Form the vertex list for this section of the clipped fill area.
4. Return to the exit-intersection point and continue processing the polygon edges in a counter clockwise order.

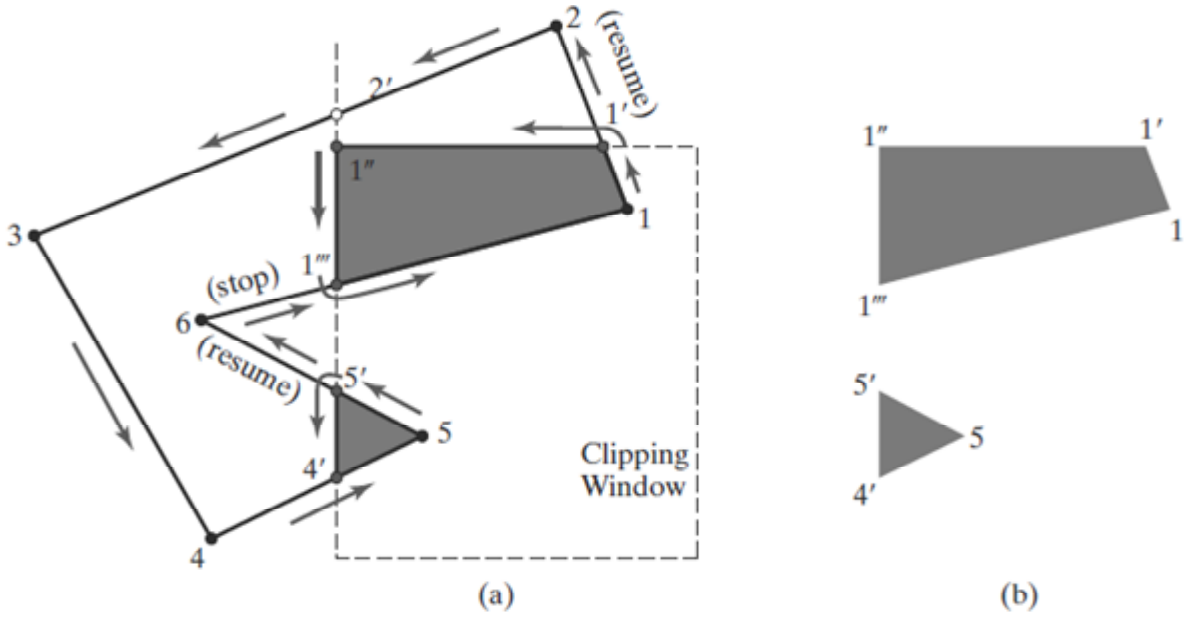


Figure 13: A concave polygon (a), defined with the vertex list {1, 2, 3, 4, 5, 6}, is clipped using the Weiler-Atherton algorithm to generate the two lists {1, 1', 1'', 1'''} and {4', 5, 5'}

6.6 Curve Clipping

Areas with curved boundaries can be clipped with polygon clipping methods. If the boundary is not a straight line section, the clipping procedures involve nonlinear equations. We can first test the coordinate extents of an object against the clipping boundaries to determine whether it is possible to accept or reject the entire object trivially. If not, we could check for object symmetries that we might be able to exploit in the initial accept/reject tests.

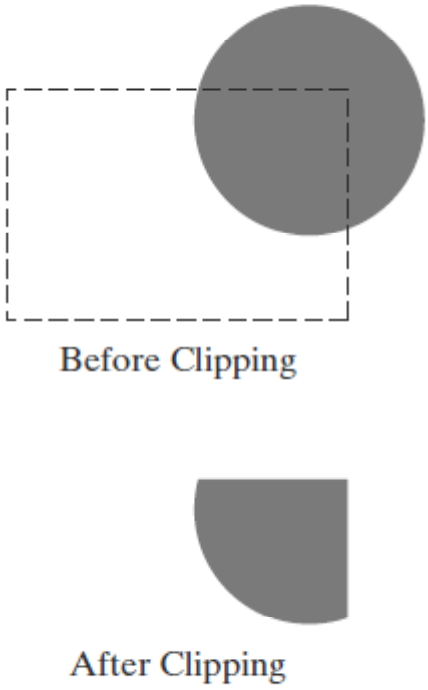


Figure 14: Clipping a circle fill area.

An intersection calculation involves substituting a clipping-boundary position (xw_{min}, xw_{max}, yw_{min}, or yw_{max}) in the nonlinear equation for the object boundary and solving for the other coordinate value. Once all intersection positions have been evaluated, the defining positions for the object can be stored for later use by the scan-line fill procedures.

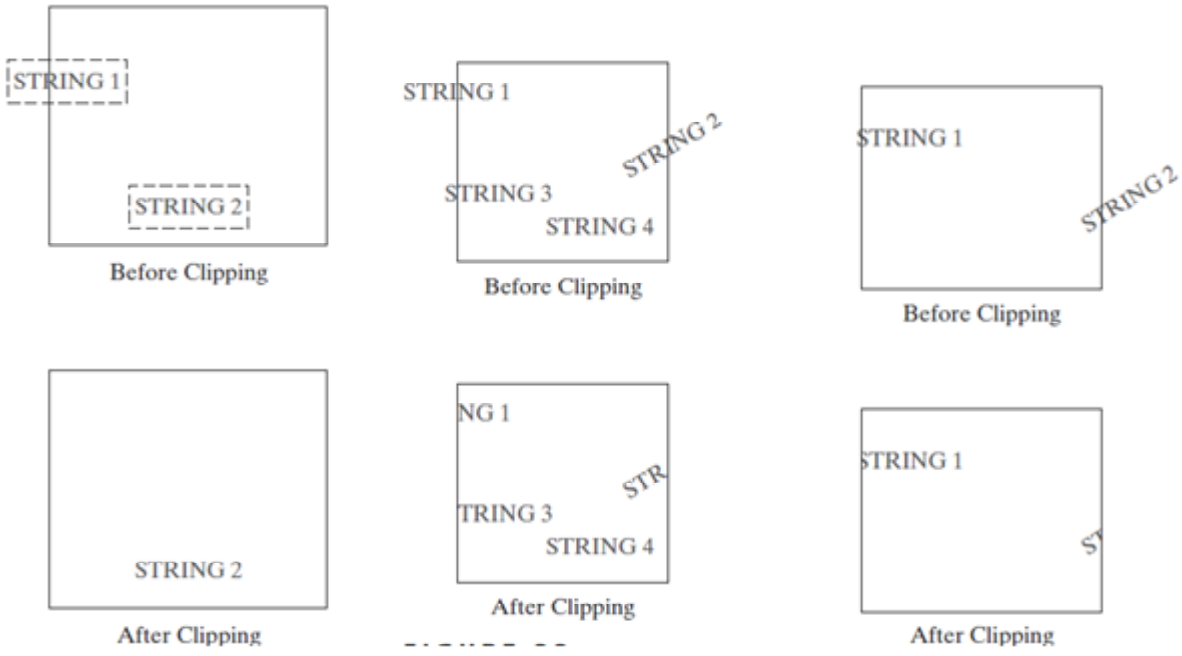
6.7 Text Clipping

Text Clipping is a procedure that identifies a portion of a string that are either inside or outside of the clipping window. Several techniques can be used to provide text clipping.

The simplest method for processing character strings relative to the limits of a clipping window is to use the all-or-none string-clipping strategy shown in Figure 15 (a). If all of the string is inside the clipping window, we display the entire string. Otherwise, the entire string is eliminated.

An alternative is to use the all-or-none character-clipping strategy. Here we eliminate only those characters that are not completely inside the clipping window (Figure 15 b). In this case, the coordinate extents of individual characters are compared to the window boundaries. Any character that is not completely within the clipping-window boundary is eliminated.

A third approach to text clipping is to clip the components of individual characters. This provides the most accurate display of clipped character strings, but it requires the most processing.



a) All-or-none string -clipping b) all-or-none character-clipping c) clip individual characters

Figure 15: Text Clipping

6.8 Exterior Clipping

In some cases, we want to clip a picture to the exterior of a specified region. The picture parts to be saved are those that are outside the region. This is referred as exterior clipping.

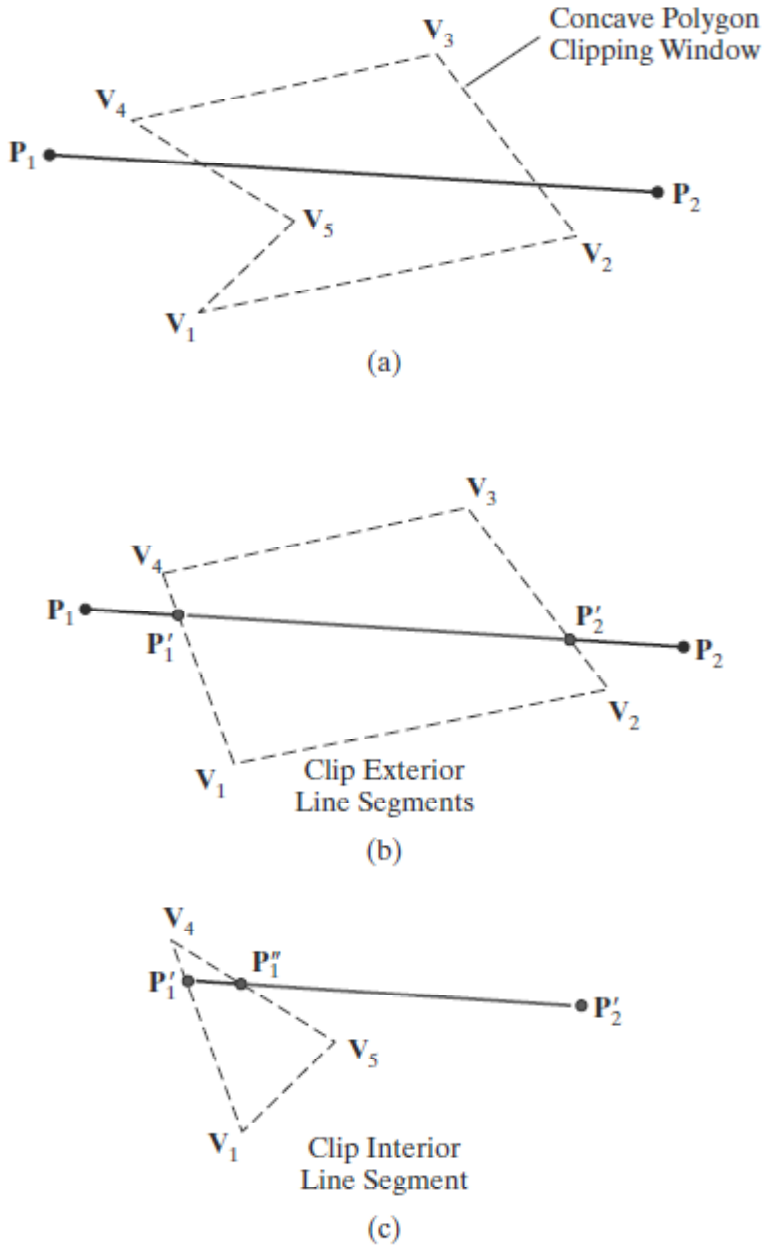


Figure 16: Clipping line $\overline{P_1P_2}$ to the interior of a concave polygon with vertices V_1,V_2,V_3,V_4,V_5 (a) convex polygon V_1,V_2,V_3,V_4 (b) V_1,V_3,V_4 (c) Clipped line $\overline{P_1''9_2'}$

6.9 Summary

Clipping algorithms are usually implemented in normalized coordinates.

All graphics packages include routines for clipping straight-line segments and polygon fill areas.

Because the clipping calculations are time-consuming, the development of improved clipping algorithms continues to be an area of major concern in computer graphics.

Cohen and Sutherland developed a line-clipping algorithm uses a region code to identify the position of a line endpoint relative to the clipping window boundaries.

Liang and Barsky developed a faster line-clipping algorithm that represents line segments with parametric equations.

The Nicholl-Lee-Nicholl (NLN) algorithm further reduces intersection calculations by using more region testing in the xy plane.

NLN approach applies only to two-dimensional line segments.

For polygon clipping, one approach is to split a concave clipping window into a set of convex polygons and apply the parametric line-clipping methods.

Another approach is to add edges to the concave window to modify it to a convex shape. Then a series of exterior and interior clipping operations can be performed to obtain the clipped line segment.

Sutherland-Hodgman algorithm, pairs of fill-area vertices are processed by each boundary clipper in turn, and clipping information for that edge is passed immediately to the next clipper, which allows the four clipping routines (left, right, bottom, and top) to be operating in parallel.

Both convex and concave fill areas can be clipped correctly with the Weiler-Atherton algorithm, which uses a boundary-traversal approach.

The fastest text-clipping method is the all-or-none strategy.

We could clip a text string by eliminating only those characters in the string that are not completely inside the clipping window.

6.10 Model Question

1. Define Clipping
2. What is meant by point clipping?
3. Write short notes on line clipping.
4. Explain Cohen-Sutherland Line Clipping algorithm.
5. Explain Liang-Barsky Line Clipping algorithm.
6. Explain Nicholl-Lee-Nicholl Line Clipping algorithm.
7. Explain Sutherland—Hodgman Polygon Clipping algorithm.
8. Explain Curve clipping in detail.
9. What is Text clipping?
10. What is Exterior Clipping?

LESSON – 7
THREE DIMENSIONAL TRANSFORMATIONS

Structure of the lesson

7.1 Introduction

Methods for geometric transformations in three dimensions are extended from two-dimensional methods by including considerations for the z coordinate. In most cases, this extension is relatively straight forward. When we discussed two-dimensional rotations in the xy plane, we needed to consider only rotations about axes that were perpendicular to the xy plane. In three-dimensional space, we can now select any spatial orientation for the rotation axis. Some graphics packages handle three-dimensional rotation as a composite of three rotations, one for each of the three Cartesian axes. Alternatively, we can set up general rotation equations, given the orientation of a rotation axis and the required rotation angle.

A three-dimensional position, expressed in homogeneous coordinates, is represented as a four-element column vector. Thus, each geometric transformation operator is now a 4 × 4 matrix, which premultiplies a coordinate column vector. In addition, as in two dimensions, any

sequence of transformations is represented as a single matrix, formed by concatenating the matrices for the individual transformations in the sequence. Each successive matrix in a transformation sequence is concatenated to the left of previous transformation matrices.

7.2 Objective of this Lesson

The objective of this lesson is to learn and understand

- Three-Dimensional Translation
- Three-Dimensional Rotation
- Three-Dimensional Scaling
- Composite Three-Dimensional Transformations
- Other Three-Dimensional

7.3 Three-Dimensional Translation

A position P = (*x, y, z*) in three-dimensional space is translated to a location $P' = (x', y', z')$ by adding translation distances t_x, t_y, and t_z to the Cartesian coordinates of P:

$$x' = x + t_x, \qquad y' = y + t_y, \qquad z' = z + t_z$$

We can express these three-dimensional translation operations in matrix form. But now the coordinate positions, P and P', are represented in homogeneous coordinates with four-element column matrices, and the translation operator T is a 4 × 4 matrix:

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

or

$$\mathbf{P}' = \mathbf{T} \cdot \mathbf{P}$$

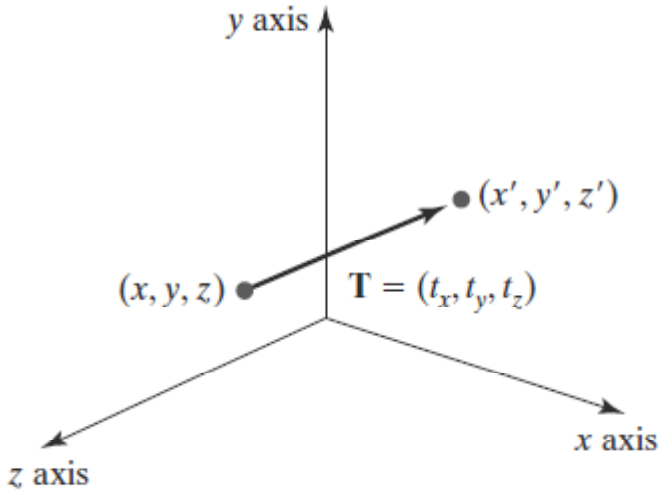


Figure 1: Three dimensional translation.

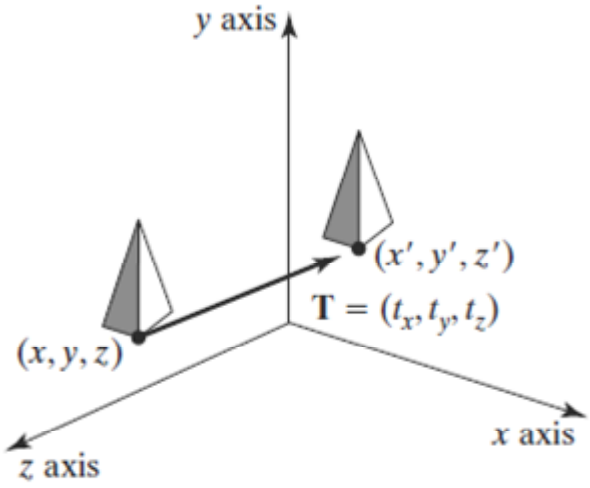


Figure 2: Three-dimensional object using translation vector T

An object is translated in three dimensions by transforming each of the defining coordinate positions for the object, then reconstructing the object at the new location. For an object represented as a set of polygon surfaces, we translate each vertex for each surface (Figure 2) and redisplay the polygon facets at the translated positions.

7.4 Three-Dimensional Rotation

We can rotate an object about any axis in space, but the easiest rotation axes to handle are those that are parallel to the Cartesian-coordinate axes. Also, we can use combinations of coordinate-axis rotations to specify a rotation about any other line in space. Positive rotation angles produce counter clockwise rotations about a coordinate axis.

Three-Dimensional Coordinate-Axis Rotations

The two-dimensional z-axis rotation equations are easily extended to three dimensions, as follows:

$$x' = x\cos\theta - y\sin\theta$$
$$y' = x\sin\theta + y\cos\theta$$
$$z' = z$$

Parameter θ specifies the rotation angle about the z axis, and z-coordinate values are unchanged by this transformation. In homogeneous-coordinate form, the three-dimensional z-axis rotation equations are

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Or $\mathbf{P}' = \mathbf{R}_z(\theta) \cdot \mathbf{P}$

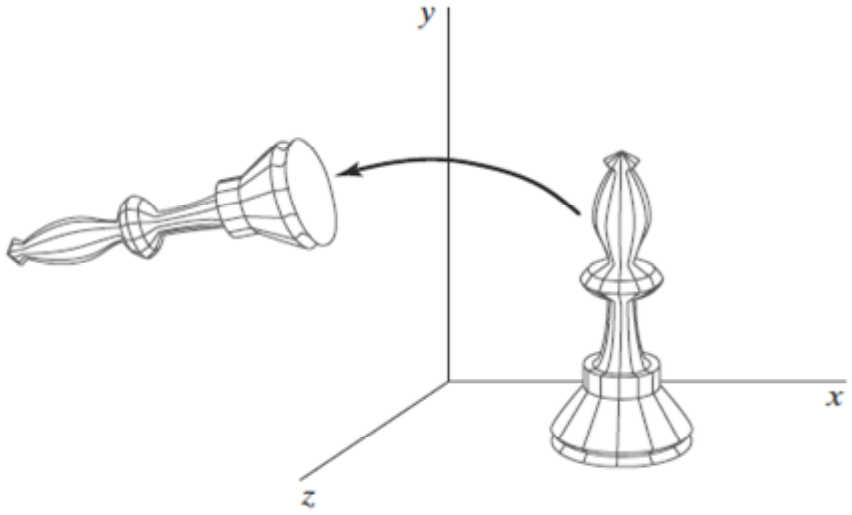


Figure 3: Rotation of an object about the z axis.

Transformation equations for rotations about the other two coordinate axes can be obtained with a cyclic permutation of the coordinate parameters x, y, and z in z-axis rotation equations. $x \rightarrow y \rightarrow z \rightarrow x$

Thus, to obtain the x-axis and y-axis rotation transformations, we cyclically replace x with y, y with z, and z with x.

Substituting permutations 7 into z-axis rotation equations, we get the equations for an x-axis rotation:

$$y' = y\cos\theta - z\sin\theta$$
$$z' = y\sin\theta + z\cos\theta$$
$$x' = x$$

A cyclic permutation of coordinates in Equations 8 gives us the transformation equations for a y-axis rotation:

$$z' = z\cos\theta - x\sin\theta$$
$$x' = z\sin\theta + x\cos\theta$$
$$y' = y$$

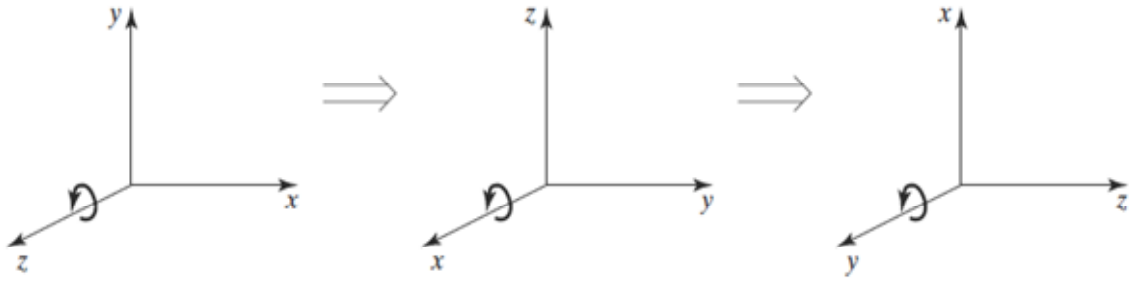


Figure 4: Cyclic permutation of the Cartesian-coordinate axes to produce the three sets of coordinate-axis rotation equations.

General Three-Dimensional Rotations

A rotation matrix for any axis that does not coincide with a coordinate axis can be set up as a composite transformation involving combinations of translations and the coordinate-axis rotations. We first move the designated rotation axis onto one of the coordinate axes. Then we apply the appropriate rotation matrix for that coordinate axis. The last step in the transformation sequence is to return the rotation axis to its original position.

In the special case where an object is to be rotated about an axis that is parallel to one of the coordinate axes, we attain the desired rotation with the following transformation sequence:

1. Translate the object so that the rotation axis coincides with the parallel coordinate axis.
2. Perform the specified rotation about that axis.
3. Translate the object so that the rotation axis is moved back to its original position.

A coordinate position P is transformed with the sequence shown in this figure as

$$\mathbf{P}' = \mathbf{T}^{-1} \cdot \mathbf{R}_x(\theta) \cdot \mathbf{T} \cdot \mathbf{P}$$

where the composite rotation matrix for the transformation is

$$\mathbf{R}(\theta) = \mathbf{T}^{-1} \cdot \mathbf{R}_x(\theta) \cdot \mathbf{T}$$

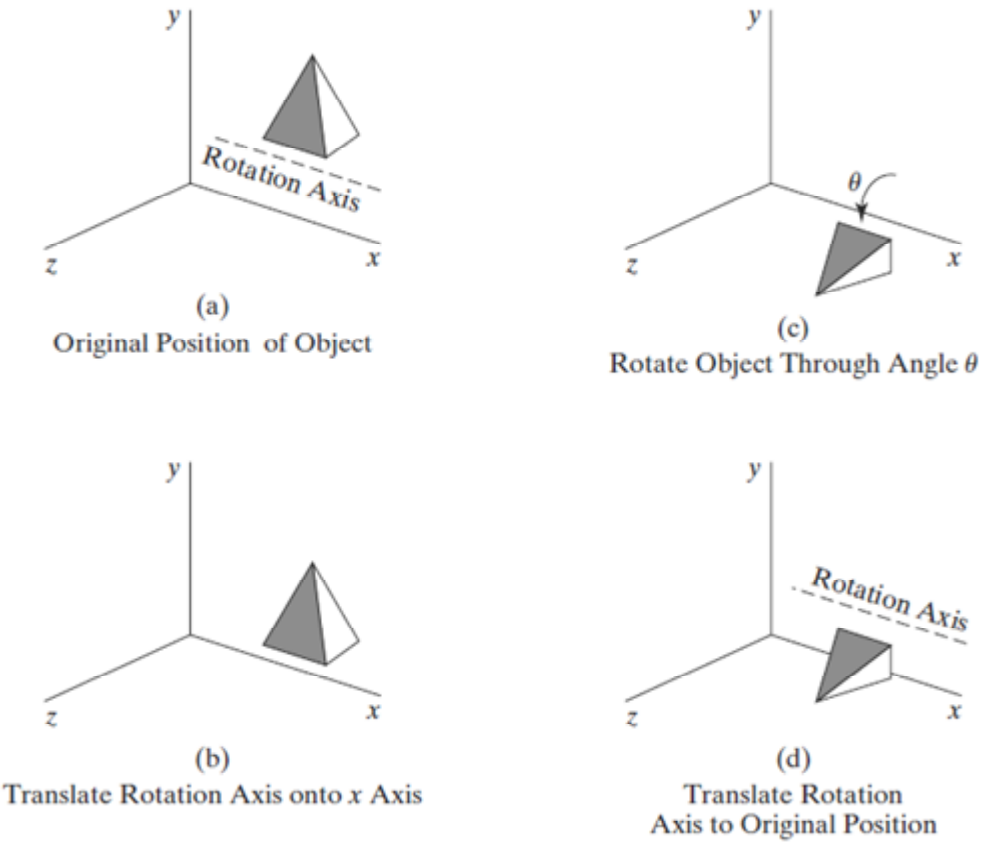


Figure 5: Sequence of transformations for rotating an object about an axis that is parallel to the x axis.

Given the specifications for the rotation axis and the rotation angle, we can accomplish the required rotation in five steps:

1. Translate the object so that the rotation axis passes through the coordinate origin.
2. Rotate the object so that the axis of rotation coincides with one of the coordinate axes.
3. Perform the specified rotation about the selected coordinate axis.

4. Apply inverse rotations to bring the rotation axis back to its original orientation.

5. Apply the inverse translation to bring the rotation axis back to its original spatial position.

A rotation axis can be defined with two coordinate positions, or with one coordinate point and direction angles (or direction cosines) between the rotation axis and two of the coordinate axes. The direction of rotation is to be counter clockwise when looking along the axis from P_2 to P_1. The components of the rotation-axis vector are then computed as

$$\begin{aligned}\mathbf{V} &= \mathbf{P}_2 - \mathbf{P}_1 \\ &= (x_2 - x_1, y_2 - y_1, z_2 - z_1)\end{aligned}$$

The unit rotation-axis vector **u** is

$$\mathbf{u} = \frac{\mathbf{V}}{|\mathbf{V}|} = (a, b, c)$$

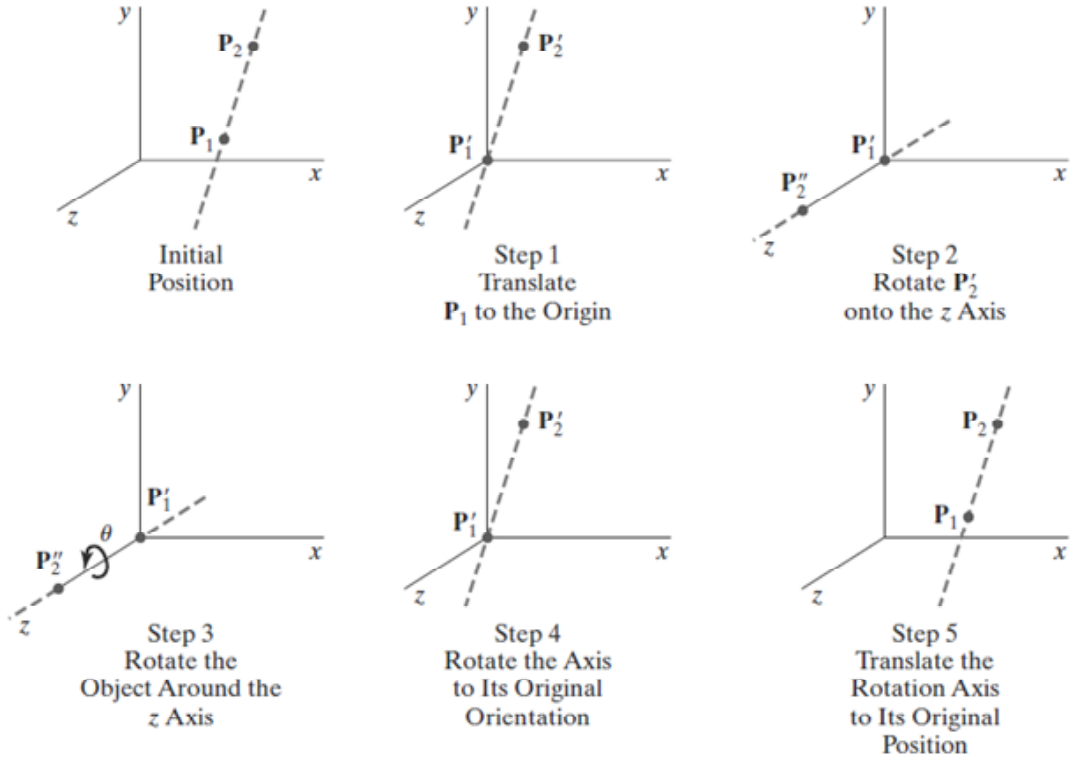


Figure 6: Five transformation steps

where the components a, b, and c are the direction cosines for the rotation axis:

$$a = \frac{x_2 - x_1}{|\mathbf{V}|}, \qquad b = \frac{y_2 - y_1}{|\mathbf{V}|}, \qquad c = \frac{z_2 - z_1}{|\mathbf{V}|}$$

The first step in the rotation sequence is to set up the translation matrix that repositions the rotation axis so that it passes through the coordinate origin.

Because we want a counter clockwise rotation when viewing along the axis from P_2 to P_1, we move the point P_1 to the origin This translation matrix is

$$\mathbf{T} = \begin{bmatrix} 1 & 0 & 0 & -x_1 \\ 0 & 1 & 0 & -y_1 \\ 0 & 0 & 1 & -z_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Next, we formulate the transformations that will put the rotation axis onto the z axis. We establish the transformation matrix for rotation around the x axis by determining the values for the sine and cosine of the rotation angle necessary to get u into the xz plane. This rotation angle is the angle between the projection of u in the yz plane and the positive z axis.

The next step in the formulation of the transformation sequence is to determine the matrix that will swing the unit vector in the xz plane counter clockwise around the y axis onto the positive z axis.

Quaternion Methods for Three-Dimensional Rotations

A more efficient method for generating a rotation about an arbitrarily selected axis is to use a quaternion representation for the rotation transformation. Quaternions, which are extensions of two-dimensional complex numbers, are useful in a number of computer-graphics procedures, including the generation of fractal objects.

One way to characterize a quaternion is as an ordered pair, consisting of a scalar part and a vector part:

$$q = (s, \mathbf{v})$$

A rotation about any axis passing through the coordinate origin is accomplished by first setting up a unit quaternion with the scalar and vector parts as follows:

$$s = \cos\frac{\theta}{2}, \qquad \mathbf{v} = \mathbf{u}\sin\frac{\theta}{2}$$

where u is a unit vector along the selected rotation axis and θ is the specified rotation angle about this axis.

Any point position P that is to be rotated by this quaternion can be represented in quaternion notation as

$$\mathbf{P} = (0, \mathbf{p})$$

with the coordinates of the point as the vector part p = (x, y, z). The rotation of the point is then carried out with the quaternion operation.

$$\mathbf{P}' = q\mathbf{P}q^{-1}$$

where q^{-1} = (s, -v) is the inverse of the unit quaternion q with the scalar and vector parts.

The second term in this ordered pair is the rotated point position p', which is evaluated with vector dot and cross-products as

$$\mathbf{p}' = s^2\mathbf{p} + \mathbf{v}(\mathbf{p}\cdot\mathbf{v}) + 2s(\mathbf{v}\times\mathbf{p}) + \mathbf{v}\times(\mathbf{v}\times\mathbf{p})$$

7.5 Three-Dimensional Scaling

The matrix expression for the three-dimensional scaling transformation of a position P = (x, y, z) relative to the coordinate origin is a simple extension of two-dimensional scaling. We just include the parameter for z-coordinate scaling in the transformation matrix:

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

The three-dimensional scaling transformation for a point position can be represented as

$$\mathbf{P}' = \mathbf{S} \cdot \mathbf{P}$$

where scaling parameters s_x, s_y, and s_z are assigned any positive values. Explicit expressions for the scaling transformation relative to the origin are

$$x' = x \cdot s_x, \qquad y' = y \cdot s_y, \qquad z' = z \cdot s_z$$

Scaling an object with transformation changes the position of the object relative to the coordinate origin. A parameter value greater than 1 moves a point farther from the origin in the corresponding coordinate direction. A parameter value greater than 1 moves a point farther from the origin in the corresponding coordinate direction. Similarly, a parameter value less than 1 moves a point closer to the origin in that coordinate direction. Also, if the scaling parameters are not all equal, relative dimensions of a transformed object are changed. We preserve the original shape of an object with a uniform scaling: $s_x = s_y = s_z$.

We can always construct a scaling transformation with respect to any selected fixed position (x_f , y_f , z_f) using the following transformation sequence:

1. Translate the fixed point to the origin.
2. Apply the scaling transformation relative to the coordinate origin
3. Translate the fixed point back to its original position.

The matrix representation for an arbitrary fixed-point scaling can then be expressed as the concatenation of these translate-scale-translate transformations

$$\mathbf{T}(x_f, y_f, z_f) \cdot \mathbf{S}(s_x, s_y, s_z) \cdot \mathbf{T}(-x_f, -y_f, -z_f) = \begin{bmatrix} s_x & 0 & 0 & (1-s_x)x_f \\ 0 & s_y & 0 & (1-s_y)y_f \\ 0 & 0 & s_z & (1-s_z)z_f \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

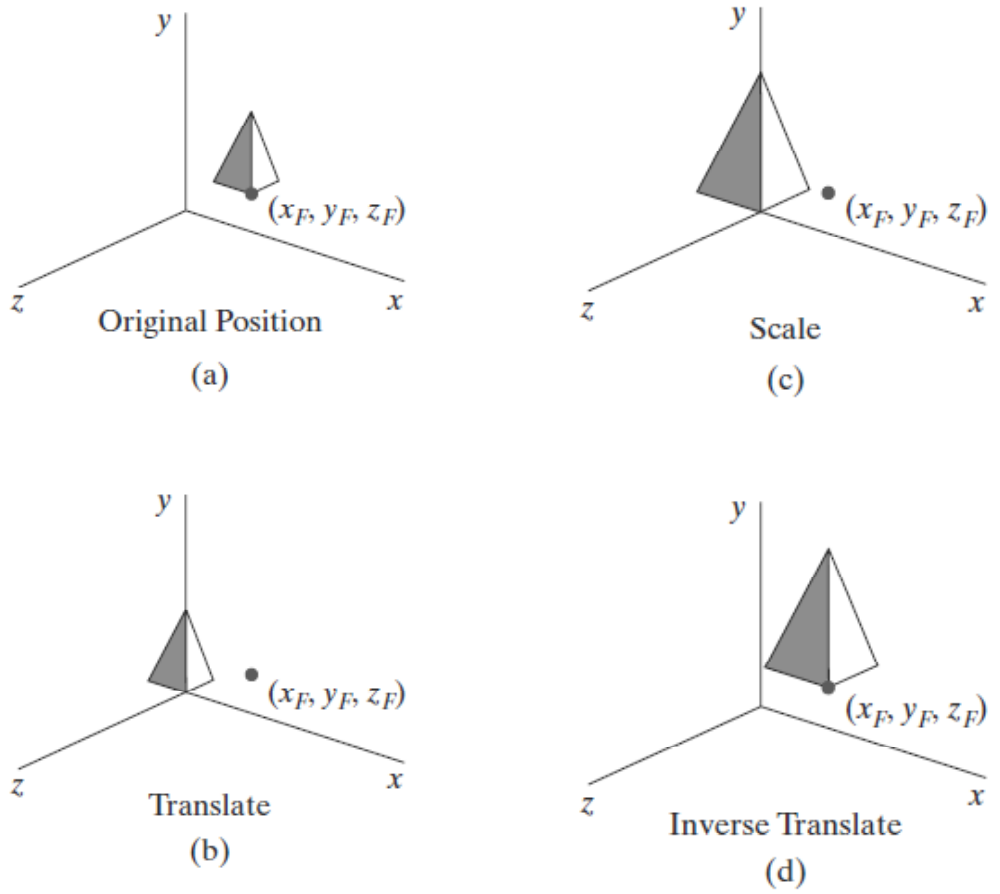


Figure 7: sequence of transformations for scaling an object

7.6 Composite transformation

A composite three dimensional transformation can be formed by multiplying the matrix representations for the individual operations in the transformation sequence. Any of the two dimensional transformation sequences, such as scaling in non-coordinate directions, can be carried out in three-dimensional space. We can implement a transformation sequence by concatenating the individual matrices from right to left or from left to right, depending on the order in which the matrix representations are specified.

We need to use this ordering for the matrix product because coordinate positions are represented as four-element column vectors, which are premultiplied by the composite 4 × 4 transformation matrix. The three basic geometric transformations are combined in a selected order to produce a single composite matrix, which is initialized to the identity matrix. We choose a left-to-right evaluation of the composite matrix so that the transformations are called in the order that they are to be applied. Thus, as each matrix is constructed, it is concatenated on the left of the current composite matrix to form the updated product matrix.

In addition to translation, rotation, and scaling, the other transformations discussed for two-dimensional applications are also useful in many three dimensional situations. These additional transformations include reflection, shear, and transformations between coordinate-reference frames.

7.7 Three-Dimensional Reflections

A reflection in a three-dimensional space can be performed relative to a selected reflection axis or with respect to a reflection plane. In general, three-dimensional reflection matrices are set up similarly to those for two dimensions. Reflections relative to a given axis are equivalent to 180æ% rotations about that axis.

The matrix representation for this reflection relative to the xy plane is

$$M_{\text{zreflect}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Transformation matrices for inverting x coordinates or y coordinates are defined similarly, as reflections relative to the yz plane or to the xz plane, respectively. Reflections about other planes can be obtained as a combination of rotations and coordinate-plane reflections.

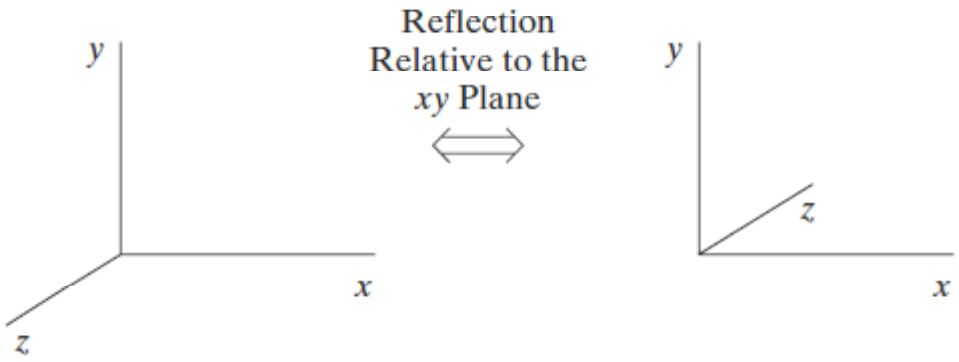


Figure 8: Conversion of coordinate specifications between a right-handed and a left-handed system

7.8 Three-Dimensional Shears

These transformations can be used to modify object shapes, just as in two dimensional applications. They are also applied in three-dimensional viewing transformations for perspective projections. For three-dimensional we can also generate shears relative to the z axis. A general z-axis shearing transformation relative to a selected reference position is produced with the following matrix:

$$M_{\text{zshear}} = \begin{bmatrix} 1 & 0 & \text{sh}_{zx} & -\text{sh}_{zx} \cdot z_{\text{ref}} \\ 0 & 1 & \text{sh}_{zy} & -\text{sh}_{zy} \cdot z_{\text{ref}} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Shearing parameters sh_{zx} and sh_{zy} can be assigned any real values. The effect of this transformation matrix is to alter the values for the x and y coordinates by an amount that is proportional to the distance from z_{ref}, while leaving the z coordinate unchanged. Plane areas that are perpendicular to the z axis are thus shifted by an amount equal to $z-z_{ref}$. An example of the effect of this shearing matrix on a unit cube is shown in Figure 9 for shearing values $sh_{zx} = sh_{zy} = 1$ and a reference position $z_{ref} = 0$. Three-dimensional transformation matrices for an x-axis shear and a y-axis shear are similar to the two-dimensional matrices. We just need to add a row and a column for the z-coordinate shearing parameters.

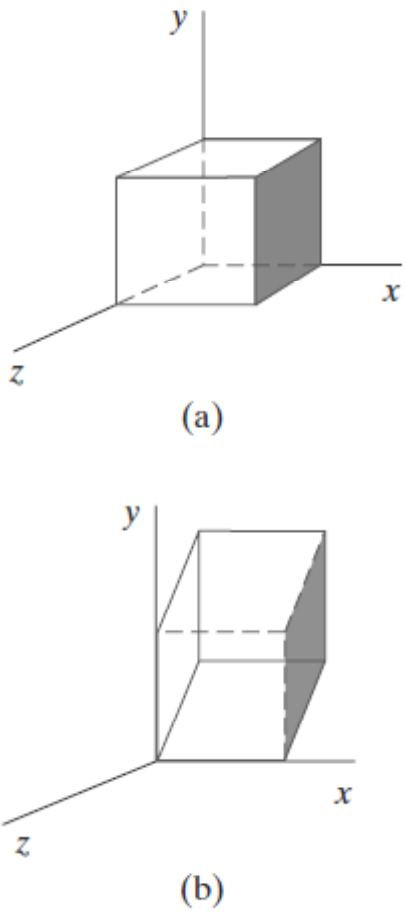


Figure 9: A unit cube (a) is sheared relative to the origin (b).

7.9 Summary

We can express three-dimensional transformations as 4 × 4 matrix operators.

We use a four element column matrix (vector) representation for three-dimensional coordinate points, representing them using a homogeneous coordinate representation.

We can create composite transformations through matrix multiplications of translation, rotation, scaling, and other transformations.

Transformations between Cartesian coordinate systems in three dimensions are accomplished with a sequence of translate-rotate transformations. We specify the coordinate origin and axis vectors for one reference frame relative to the original coordinate reference frame. The transfer of object descriptions from the original coordinate system to the second system is calculated as the matrix product of a translation that moves the new origin to the old coordinate origin and a rotation to align the two sets of axes.

7.10 Model Question

1. What is transformation?
2. Explain three dimensional translation.
3. Explain three dimensional rotation.
4. Explain three dimensional scaling.
5. Explain three dimensional composite transformation.
6. Explain 3D shearing.
7. What is 3D reflection?

LESSON – 8
THREE DIMENSIONAL VIEWING

Structure of the lesson

8.1 Introduction

In two dimensional graphics applications, viewing operations transfer positions from the world coordinate plane to pixel positions in the plane of the output device. A two dimensional package maps the world coordinate to device coordinates and clips the scene against the four boundaries of the viewport. For three dimensional graphics applications, Viewing involves the following considerations: - We can view an object from any spatial position like,

- In front of an object,
- Behind the object,
- In the middle of a group of objects,
- Inside an object.

3D descriptions of objects must be projected onto the flat viewing surface of the output device.

Three-dimensional viewing operations, however, are more involved, because we now have many more choices as to how we can construct a scene and how we can generate views of the scene on an output device.

In this chapter, we explore the general operations needed to produce views of a three dimensional scene.

8.2 Objective of this Lesson:

The objective of this lesson is to learn and understand

- Three dimensional Viewing
- Projection and its types

8.3 Overview of Three-Dimensional Viewing Concepts

When we model a three-dimensional scene, each object in the scene is typically defined with a set of surfaces that form a closed boundary around the object interior. And, for some applications, we may need also to specify information about the interior structure of an object. In addition to procedures that generate views of the surface features of an object, graphics packages sometimes provide routines for displaying internal components or cross-sectional views of a solid object. Viewing functions process the object descriptions through a set of procedures that ultimately project a specified view of the objects onto the surface of a display device. Many processes in three-dimensional viewing, such as the clipping routines, are similar to those in the two-dimensional viewing pipeline. But three-dimensional viewing involves some tasks that are not present in two-dimensional viewing. For example, projection routines are needed to transfer the scene to a view on a planar surface, visible parts of a scene must be identified, and, for a realistic display, lighting effects and surface characteristics must be taken into account.

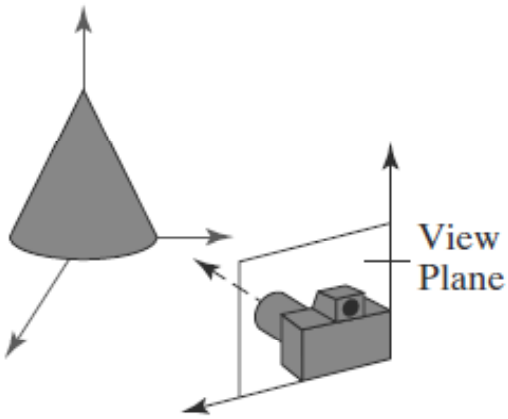


Figure 1: Coordinate reference for obtaining a selected view of a three-dimensional scene.

Viewing a Three-Dimensional Scene

To obtain a display of a three-dimensional world-coordinate scene, we first set up a coordinate reference for the viewing, or "camera," parameters. This coordinate reference defines the position and orientation for a view plane (or projection plane) that corresponds to a camera film plane. Object descriptions are then transferred to the viewing reference coordinates and projected onto the view plane. We can generate a view of an object on the output device in wireframe (outline) form, or we can apply lighting and surface-rendering techniques to obtain a realistic shading of the visible surfaces.

Projections

Unlike a camera picture, we can choose different methods for projecting a scene onto the view plane. One method for getting the description of a solid object onto a view plane is to project points on the object surface along parallel lines. This technique, called parallel projection, is used in engineering and architectural drawings to represent an object with a set of views that show accurate dimensions of the object, as in Figure 2.

Another method for generating a view of a three-dimensional scene is to project points to the view plane along converging paths. This process, called a perspective projection, causes objects farther from the viewing position to be displayed smaller than objects of the same size

that are nearer to the viewing position. A scene that is generated using a perspective projection appears more realistic, because this is the way that our eyes and a camera lens form images. Parallel lines along the viewing direction appear to converge to a distant point in the background, and objects in the background appear to be smaller than objects in the foreground.

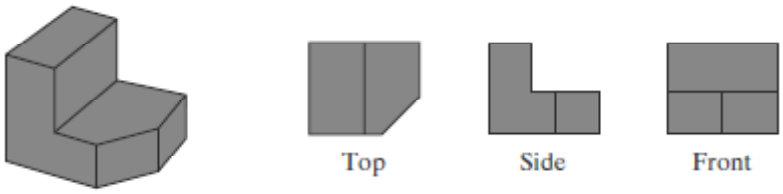


Figure 2: Three parallel-projection views of an object, showing relative proportions from different viewing positions.

Depth Cueing

With few exceptions, depth information is important in a three-dimensional scene so that we can easily identify, for a particular viewing direction, which is the front and which is the back of each displayed object.

A simple method for indicating depth with wire-frame displays is to vary the brightness of line segments according to their distances from the viewing position.

Another application of depth cuing is modeling the effect of the atmosphere on the perceived intensity of objects. More distant objects appear dimmer to us than nearer objects due to light scattering by dust particles, haze, and smoke.

Identifying Visible Lines and Surfaces

When a realistic view of a scene is to be a produced, back part of the objects are completely eliminated so that only the visible surfaces are displayed. In this case, surface-rendering procedures are applied so that screen pixels contain only the color patterns for the front surfaces.

Surface Rendering

Added realism is attained in displays by rendering object surfaces using the lighting conditions in the scene and the assigned surface characteristics. Surface properties of objects

include whether a surface is transparent or opaque and whether the surface is smooth or rough.

Three-Dimensional and Stereoscopic Viewing

Other methods for adding a sense of realism to a computer-generated scene include three-dimensional displays and stereoscopic views. Three-dimensional views can be obtained by reflecting a raster image from a vibrating, flexible mirror.

Stereoscopic devices present two views of a scene: one for the left eye and the other for the right eye. Stereoscopic devices present two views of a scene: one for the left eye and the other for the right eye. The viewing positions correspond to the eye positions of the viewer. These two views are typically displayed on alternate refresh cycles of a raster monitor. When we view the monitor through special glasses that alternately darken first one lens and then the other, in synchronization with the monitor refresh cycles, we see the scene displayed with a three-dimensional effect.

8.4 Three-Dimensional Viewing Pipeline

Procedures for generating a computer-graphics view of a three-dimensional scene are somewhat analogous to the processes involved in taking a photograph. First of all, we need to choose a viewing position corresponding to where we would place a camera. We choose the viewing position according to whether we want to display a front, back, side, top, or bottom view of the scene. We could also pick a position in the middle of a group of objects or even inside a single object, such as a building or a molecule. Then we must decide on the camera orientation (Figure 3). Which way do we want to point the camera from the viewing position, and how should we rotate it around the line of sight to set the “up” direction for the picture? Finally, when we snap the shutter, the scene is cropped to the size of a selected clipping window, which corresponds to the aperture or lens type of a camera, and light from the visible surfaces is projected onto the camera film.

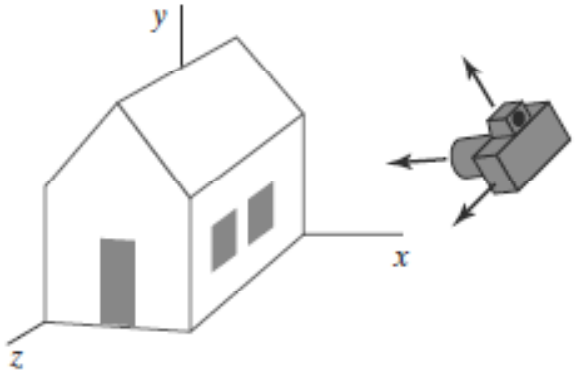


Figure 3: Photographing a scene involves selection of the camera position and orientation.

We need to keep in mind, however, that the camera analogy can be carried only so far, because we have more flexibility and many more options for generating views of a scene with a computer-graphics program than we do with a real camera. We can choose to use either a parallel projection or a perspective projection, we can selectively eliminate parts of a scene along the line of sight, we can move the projection plane away from the “camera” position, and we can even get a picture of objects in back of our synthetic camera.

Some of the viewing operations for a three-dimensional scene are the same as, or similar to, those used in the two-dimensional viewing pipeline. A two-dimensional viewport is used to position a projected view of the three-dimensional scene on the output device, and a two-dimensional clipping window is used to select a view that is to be mapped to the viewport. In addition, we set up a display window in screen coordinates, just as we do in a two-dimensional application. Clipping windows, viewports, and display windows are usually specified as rectangles with their edges parallel to the coordinate axes. In three-dimensional viewing, however, the clipping window is positioned on a selected view plane, and scenes are clipped against an enclosing volume of space, which is defined by a set of clipping planes. The viewing position, view plane, clipping window, and clipping planes are all specified within the viewing-coordinate reference frame.

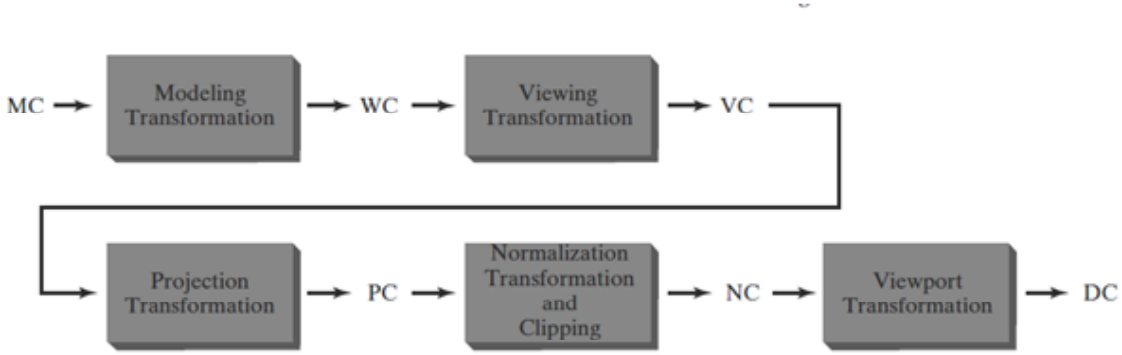


Figure 4: General three-dimensional transformation

Figure 4 shows the general processing steps for creating and transforming a three-dimensional scene to device coordinates. Once the scene has been modelled in world coordinates, a viewing-coordinate system is selected and the description of the scene is converted to viewing coordinates. The viewing coordinate system defines the viewing parameters, including the position and orientation of the projection plane (view plane), which we can think of as the camera film plane. A two-dimensional clipping window, corresponding to a selected camera lens, is defined on the projection plane, and a three-dimensional clipping region is established. This clipping region is called the view volume, and its shape and size depends on the dimensions of the clipping window, the type of projection we choose, and the selected limiting positions along the viewing direction. Projection operations are performed to convert the viewing-coordinate description of the scene to coordinate positions on the projection plane. Objects are mapped to normalized coordinates, and all parts of the scene outside the view volume are clipped off. The clipping operations can be applied after all device-independent coordinate transformations (from world coordinates to normalized coordinates) are completed. In this way, the coordinate transformations can be concatenated for maximum efficiency.

As in two-dimensional viewing, the viewport limits could be given in normalized coordinates or in device coordinates. In developing the viewing algorithms, we will assume that the viewport is to be specified in device coordinates and that normalized coordinates are transferred to viewport coordinates, following the clipping operations. There are also a few other tasks that must be performed, such as identifying visible surfaces and applying the surface-rendering procedures.

The final step is to map viewport coordinates to device coordinates within a selected display window. Scene descriptions in device coordinates are sometimes expressed in a left-handed reference frame so that positive distances from the display screen can be used to measure depth values in the scene.

Three-Dimensional Viewing-Coordinate Parameters

We first select a world-coordinate position $P_0 =(x_0, y_0, z_0)$ for the viewing origin, which is called the view point or viewing position. And we specify a view-up vector V, which defines the y_{view} direction. We also need to assign a direction for one of the remaining two coordinate axes. This is typically accomplished with a second vector that defines the z_{view} axis, with the viewing direction along this axis.

View-Plane Normal Vector

Because the viewing direction is usually along the z_{view} axis, the view plane, also called the projection plane, is normally assumed to be perpendicular to this axis. Thus, the orientation of the view plane, as well as the direction for the positive z_{view} axis, can be defined with a view-plane normal vector N. An additional scalar parameter is used to set the position of the view plane at some coordinate value z_{vp} along the z_{view} axis N to be in the direction from a reference point P_{ref} to the viewing origin P_0. We could also define the view-plane normal vector, and other vector directions, using direction angles. These are the three angles, α, β, and γ , that a spatial line makes with the x, y, and z axes, respectively.

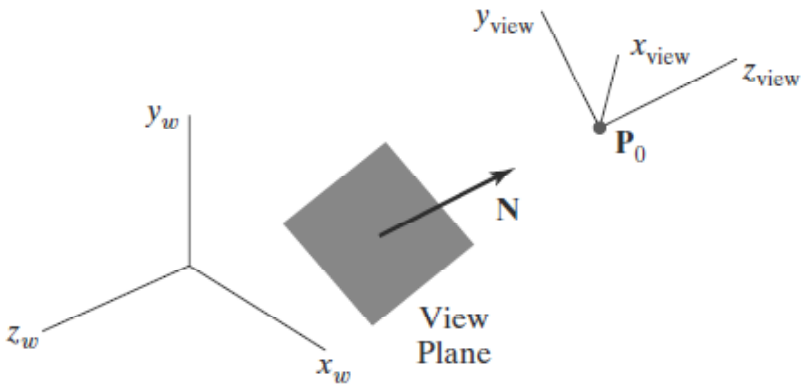


Figure 5: Orientation of the view plane and view-plane normal vector N.

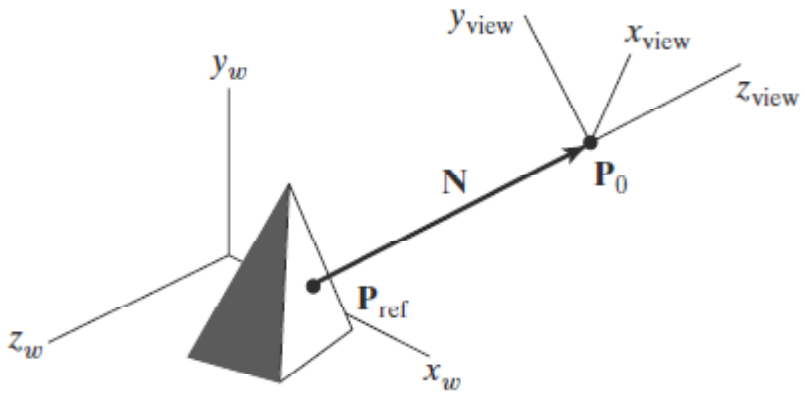


Figure 6: View-plane normal vector N reference point P_{ref} to the viewing-coordinate origin P_0

Transformation from World to Viewing Coordinates

In the three-dimensional viewing pipeline, the first step after a scene has been constructed is to transfer object descriptions to the viewing-coordinate reference frame. We can accomplish this conversion using the methods for transforming between coordinate system:

1. Translate the viewing-coordinate origin to the origin of the world coordinate system.

2. Apply rotations to align the x_{view}, y_{view}, and z_{view} axes with the world x_w, y_w, and z_w axes, respectively.

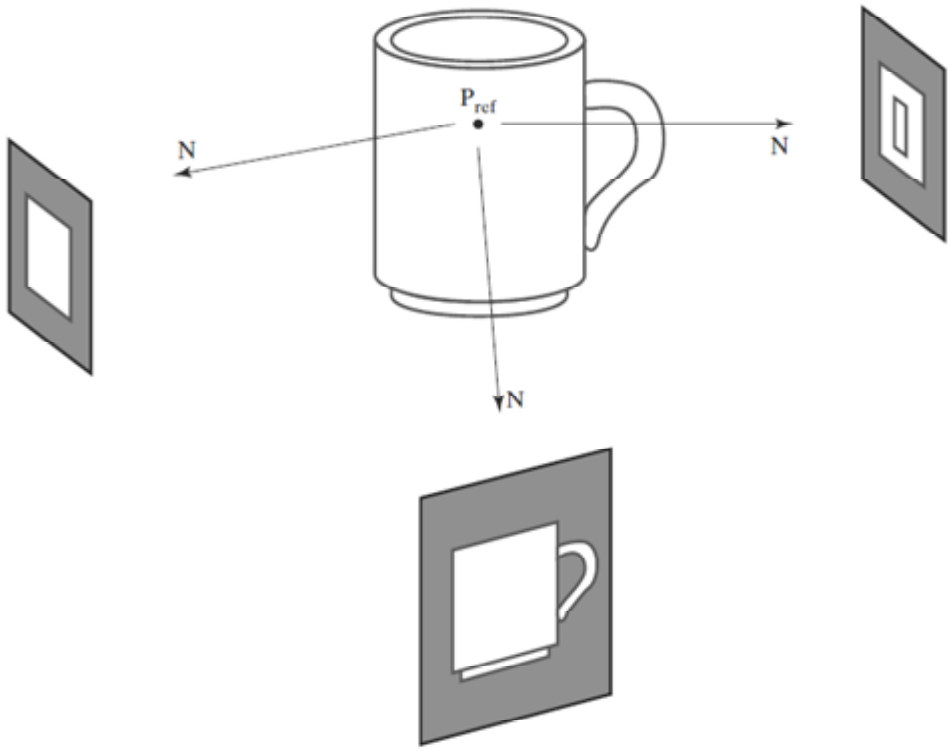


Figure 7: Viewing an object from different directions using a fixed reference point.

The viewing-coordinate origin is at world position $P_0 = (x_0, y_0, z_0)$. Therefore, the matrix for translating the viewing origin to the world origin is

$$\mathbf{T} = \begin{bmatrix} 1 & 0 & 0 & -x_0 \\ 0 & 1 & 0 & -y_0 \\ 0 & 0 & 1 & -z_0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

For the rotation transformation, we can use the unit vectors u, v, and n to form the composite rotation matrix that superimposes the viewing axes onto the world frame. This transformation matrix is

$$\mathbf{R} = \begin{bmatrix} u_x & u_y & u_z & 0 \\ v_x & v_y & v_z & 0 \\ n_x & n_y & n_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

where the elements of matrix R are the components of the u_{vn} axis vectors.

The coordinate transformation matrix is then obtained as the product of the preceding translation and rotation matrices:

$$\mathbf{M}_{WC,\,VC} = \mathbf{R} \cdot \mathbf{T} = \begin{bmatrix} u_x & u_y & u_z & -\mathbf{u} \cdot \mathbf{P}_0 \\ v_x & v_y & v_z & -\mathbf{v} \cdot \mathbf{P}_0 \\ n_x & n_y & n_z & -\mathbf{n} \cdot \mathbf{P}_0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Translation factors in this matrix are calculated as the vector dot product of each of the u, v, and n unit vectors with P0, which represents a vector from the world origin to the viewing origin.

These matrix elements are evaluated as

$$\begin{aligned} -\mathbf{u} \cdot \mathbf{P}_0 &= -x_0 u_x - y_0 u_y - z_0 u_z \\ -\mathbf{v} \cdot \mathbf{P}_0 &= -x_0 v_x - y_0 v_y - z_0 v_z \\ -\mathbf{n} \cdot \mathbf{P}_0 &= -x_0 n_x - y_0 n_y - z_0 n_z \end{aligned}$$

8.5 Projection

In the next phase of the three-dimensional viewing pipeline, after the transformation to viewing coordinates, object descriptions are projected to the view plane. Two types of projection namely parallel projection and perspective projection.

In a parallel projection, coordinate positions are transferred to the view plane along parallel lines. A parallel projection preserves relative proportions of objects, and this is the method used in computer aided drafting and designs to produce scale drawings of three-dimensional objects. All parallel lines in a scene are displayed as parallel when viewed with a parallel projection. There are two general methods for obtaining a parallel-projection view of an object: We can project along lines that are perpendicular to the view plane, or we can project at an oblique angle to the view plane.

For a perspective projection, object positions are transformed to projection coordinates along lines that converge to a point behind the view plane. Unlike a parallel projection, a perspective projection does not preserve relative proportions of objects. But perspective views of a scene are more realistic because distant objects in the projected display are reduced in size.

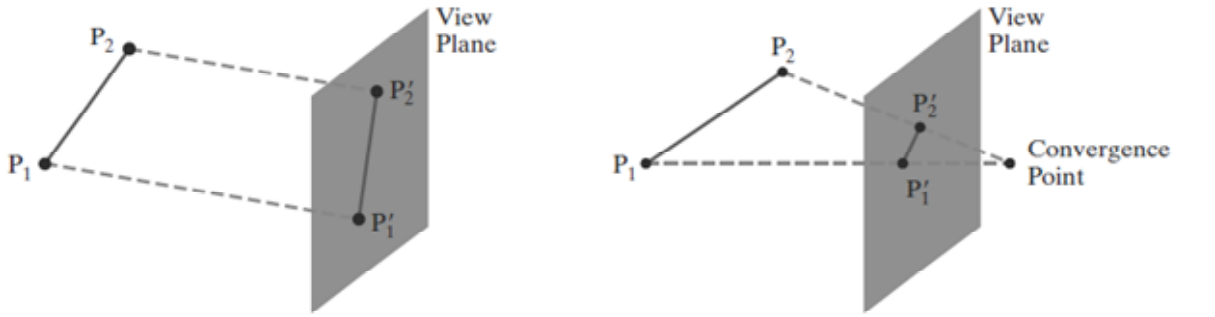


Figure 8: parallel and perspective projections

8.6 Orthogonal Projections

A transformation of object descriptions to a view plane along lines that are all parallel to the view-plane normal vector N is called an orthogonal projection or an orthographic projection. This produces a parallel-projection transformation in which the projection lines are perpendicular to the view plane. Orthogonal projections are most often used to produce the front, side, and top views of an object, as shown in Figure 9. Front, side, and rear orthogonal projections of an object are called elevations; and a top orthogonal projection is called a plan view. Engineering and architectural drawings commonly employ these orthographic projections, because lengths and angles are accurately depicted and can be measured from the drawings.

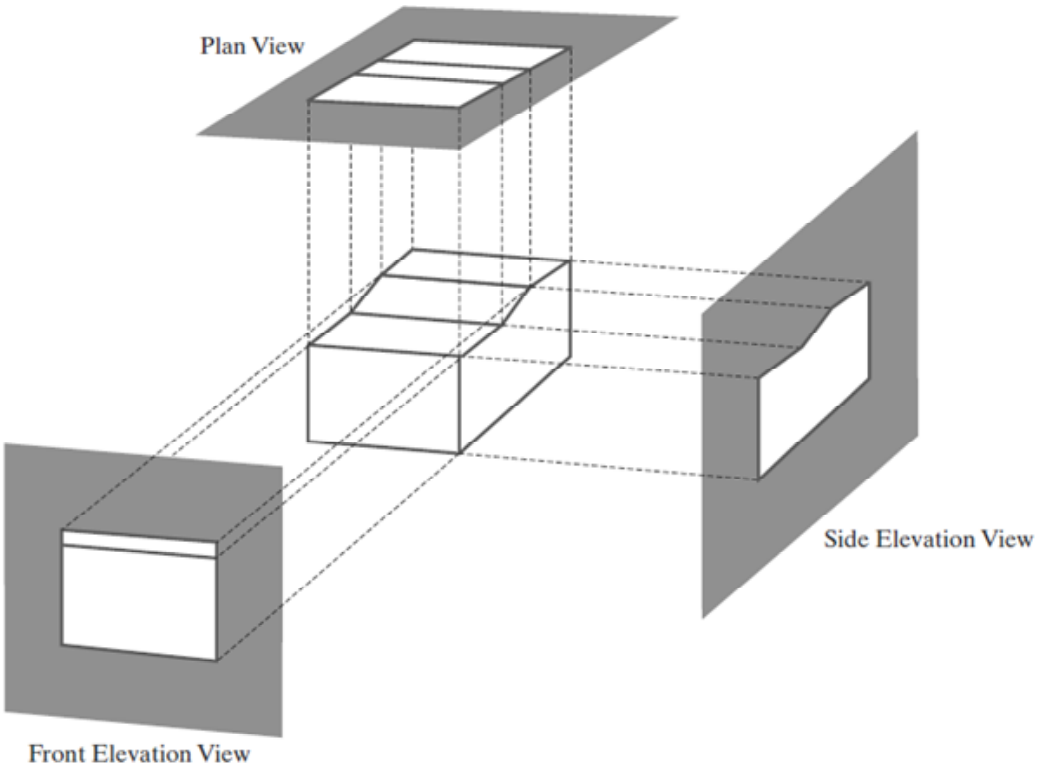


Figure 9: Orthogonal projections of an object, displaying plan and elevation views.

Axonometric and Isometric Orthogonal Projections

We can also form orthogonal projections that display more than one face of an object. Such views are called axonometric orthogonal projections. The most commonly used axonometric projection is the isometric projection, which is generated by aligning the projection plane (or the object) so that the plane intersects each coordinate axis in which the object is defined, called the principal axes, at the same distance from the origin. We can obtain the isometric projection shown in this figure by aligning the view plane normal vector along a cube diagonal. There are eight positions, one in each octant, for obtaining an isometric view. All three principal axes are foreshortened equally in an isometric projection, so that relative proportions are maintained. This is not the case in a general axonometric projection, where scaling factors may be different for the three principal directions.

Orthogonal Projection Coordinates

With the projection direction parallel to the z_{view} axis, the transformation equations for an orthogonal projection are trivial. For any position (x, y, z) in viewing coordinates, the projection coordinates are $x_p = x, \qquad y_p = y$

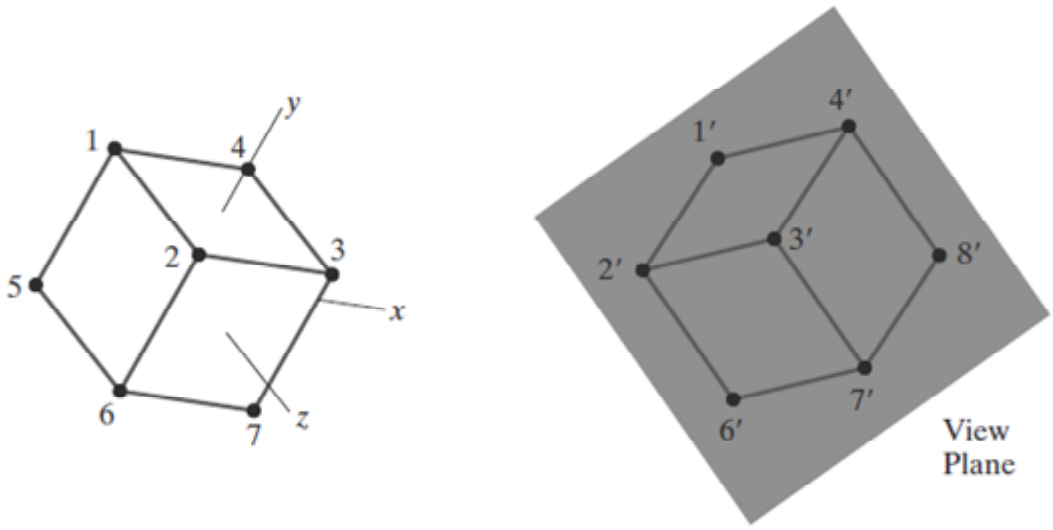


Figure 10: An isometric projection of a cube.

Clipping Window and Orthogonal-Projection View Volume

For three-dimensional viewing, the clipping window is positioned on the view plane with its edges parallel to the x_{view} and y_{view} axes, as shown in Figure 11.

The edges of the clipping window specify the x and y limits for the part of the scene that we want to display. These limits are used to form the top, bottom, and two sides of a clipping region called the orthogonal-projection view volume. Because projection lines are perpendicular to the view plane, these four boundaries are planes that are also perpendicular to the view plane and that pass through the edges of the clipping window to form an infinite clipping region.

We can limit the extent of the orthogonal view volume in the z_{view} direction by selecting positions for one or two additional boundary planes that are parallel to the view plane. These two planes are called the near-far clipping planes, or the front-back clipping planes.

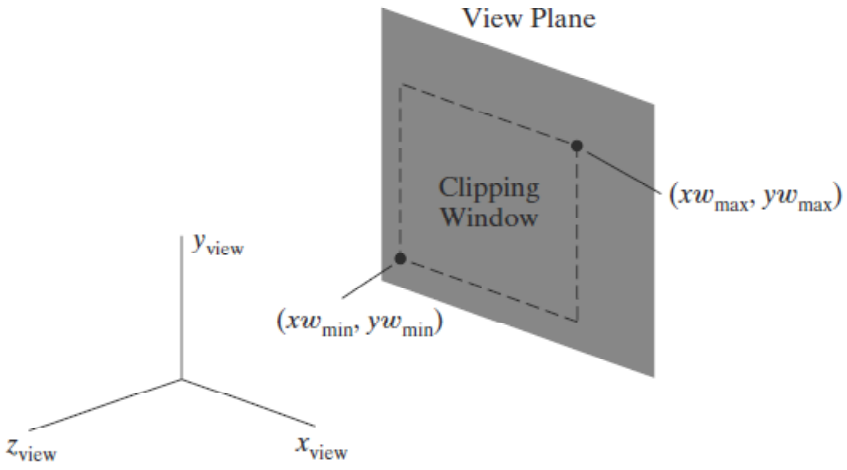


Figure 11: Clipping window on the view plane.

8.7 Oblique Parallel Projections

In general, a parallel-projection view of a scene is obtained by transferring object descriptions to the view plane along projection paths that can be in any selected direction relative to the view-plane normal vector. When the projection path is not perpendicular to the view plane, this mapping is called an oblique parallel projection. Using this projection, we can produce combinations such as a front, side, and top view of an object. Oblique parallel projections are defined by a vector direction for the projection lines, and this direction can be specified in various ways.

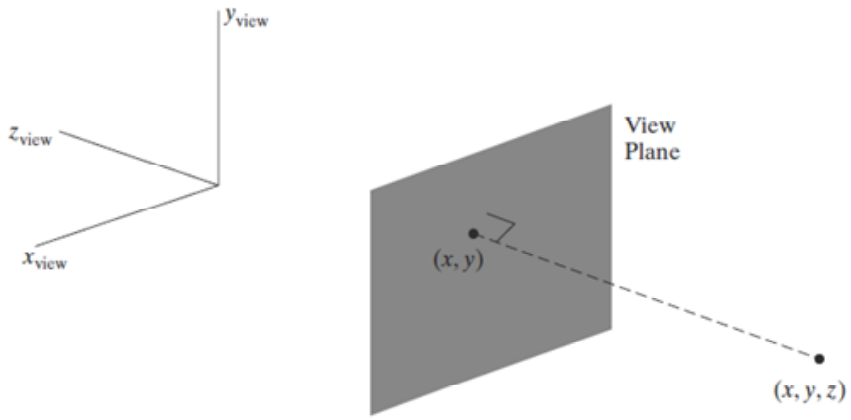


Figure 12: An orthogonal projection of a spatial position onto a view plane.

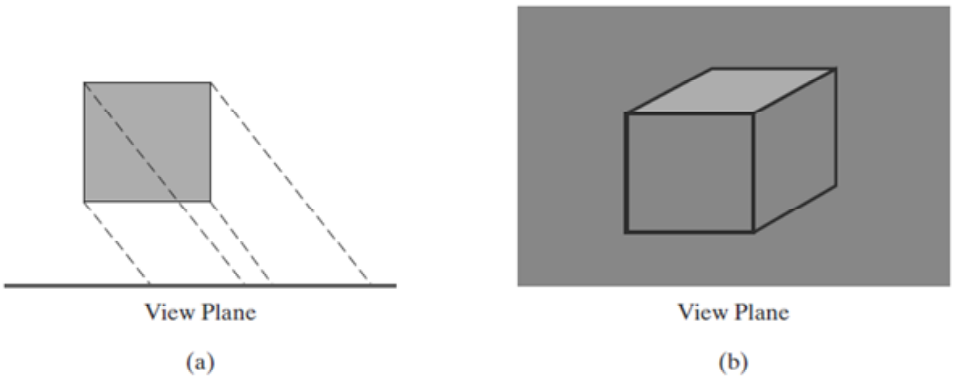


Figure 13: Oblique parallel projection of a cube

An oblique parallel projection is often specified with two angles, α and φ. A spatial position (x, y, z), in this illustration, is projected to (x_p, y_p, z_{vp}) on a view plane, which is at location z_{vp} along the viewing z axis. Position (x, y, z_{vp}) is the corresponding orthogonal-projection point. The oblique parallel projection line from (x, y, z) to (x_p, y_p, z_{vp}) has an intersection angle α with the line on the projection plane that joins (x_p, y_p, z_{vp}) and (x, y, z_{vp}). This view-plane line, with length L, is at an angle φ with the horizontal direction in the projection plane. Angle α can be assigned a value between 0 and 90°, and angle φ can vary from 0 to 360°.

We can express the projection coordinates in terms of x, y, L, and φ as

$$x_p = x + L\cos\phi$$
$$y_p = y + L\sin\phi$$

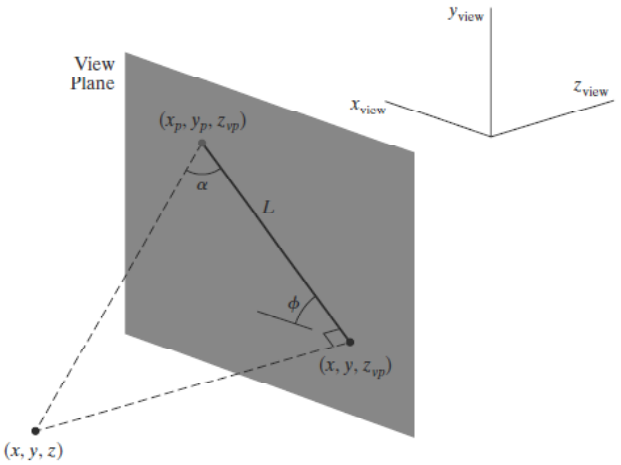


Figure 14: An oblique parallel projection of coordinate position (x , y, z) to position (x_p, y_p, z_{vp}) on a projection plane at position z_{vp} along the z view axis.

Length L depends on the angle α and the perpendicular distance of the point (x, y, z) from the view plane:

$$\tan\alpha = \frac{z_{vp} - z}{L}$$

$$L = \frac{z_{vp} - z}{\tan\alpha}$$
$$= L_1(z_{vp} - z)$$

where $L_1 = \cot\alpha$, which is also the value of L when z_{vp}-z = 1.We can then write the oblique parallel projection equations as

$$x_p = x + L_1(z_{vp} - z)\cos\phi$$
$$y_p = y + L_1(z_{vp} - z)\sin\phi$$

An orthogonal projection is obtained when $L_1 = 0$ (which occurs at the projection angle $\alpha = 90^o$).

Orthogonal and oblique parallel projection

8.8 Summary

Viewing procedures for three-dimensional scenes follow the general approach used in two-dimensional viewing.

First create a world-coordinate scene, either from the definitions of objects in modeling coordinates or directly in world coordinates. Then set up a viewing-coordinate reference frame and transfer object descriptions from world coordinates to viewing coordinates.

Unlike two-dimensional viewing, however, three-dimensional viewing requires projection routines to transform object descriptions to a viewing plane before the transformation to device coordinates.

Either parallel-projection or perspective-projection methods can be used to transfer object descriptions to the view plane.

Orthographic parallel projections that display more than one face of an object are called axonometric projections.

An isometric view of an object is obtained with an axonometric projection that foreshortens each principal axis by the same amount.

Commonly used oblique projections are the cavalier projection and the cabinet projection. Perspective projections of objects are obtained with projection lines that meet at the projection reference point.

Perspective projections cause parallel lines to appear to converge to a vanishing point, provided the lines are not parallel to the view plane.

Engineering and architectural displays can be generated with one-point, two-point, or three-point perspective projections, depending on the number of principal axes that intersect the view plane.

An oblique perspective projection is obtained when the line from the projection reference point to the center of the clipping window is not perpendicular to the view plane.

8.9 Model Question

1. Define projection.
2. Write short notes on parallel projection.
3. What is a view plane?
4. Explain three-dimensional transformation pipeline.
5. Explain the procedure for the Transformation from World to Viewing Coordinates.
6. Explain Orthogonal Projections.
7. Explain Axonometric and Isometric Orthogonal Projections in detail.

LESSON – 9
THREE DIMENSIONAL VIEWING-CLIPPING

Structure of the lesson

9.1 Introduction

In the next phase of the three-dimensional viewing pipeline, after the transformation to viewing coordinates, object descriptions are projected to the view plane. Graphics packages generally support both parallel and perspective projections. Although a parallel-projection view of a scene is easy to generate and preserves relative proportions of objects, it does not provide a realistic representation. For a perspective projection, object positions are transformed to projection coordinates along lines that converge to a point behind the view plane.

Previously, we discussed the advantages of using the normalized boundaries of the clipping window in two-dimensional clipping algorithms. Similarly, we can apply three-dimensional clipping algorithms to the normalized boundaries of the view volume.

9.2 Objective of this Lesson

In this chapter the students will learn and understand

- Perspective Projections
- Three-Dimensional Clipping Algorithms
- Three-Dimensional Point and Line Clipping
- Three-Dimensional Polygon Clipping
- Three-Dimensional Curve Clipping

9.3 Perspective Projections

A perspective projection causes objects farther from the viewing position to be displayed smaller than objects of the same size that are nearer to the viewing position. A scene that is generated using a perspective projection appears more realistic, because this is the way that our eyes and a camera lens form images. We can approximate this geometric-optics effect by projecting objects to the view plane along converging paths to a position called the projection reference point (or center of projection). Objects are then displayed with foreshortening effects, and projections of distant objects.

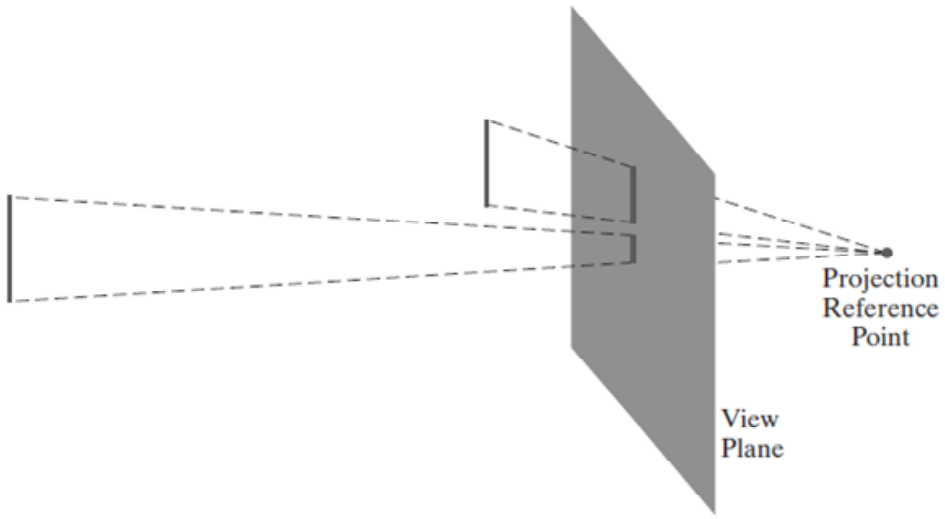


Figure 1: A perspective projection of two equal-length line segments at different distances from the view plane.

Perspective-Projection Transformation Coordinates

We can sometimes select the projection reference point as another viewing parameter in a graphics package, but some systems place this convergence point at a fixed position, such as at the view point. Figure 2 shows the projection path of a spatial position (x, y, z) to a general projection reference point at (x_{prp}, y_{prp}, z_{prp}). The projection line intersects the view plane at the coordinate position (x_p, y_p, z_{vp}), where z_{vp} is some selected position for the view plane on the z_{view} axis. We can write equations describing coordinate positions along this perspective-projection line in parametric form as

$$\begin{aligned} x' &= x - (x - x_{prp})u \\ y' &= y - (y - y_{prp})u \qquad 0 \le u \le 1 \\ z' &= z - (z - z_{prp})u \end{aligned}$$

Coordinate position (x', y', z') represents any point along the projection line. When u = 0, we are at position P = (x, y, z). At the other end of the line, u = 1 and we have the projection reference-point coordinates (x_{prp}, y_{prp}, z_{prp}). On the view plane, z'= z_{vp} and we can solve the z' equation for parameter u at this position along the projection line:

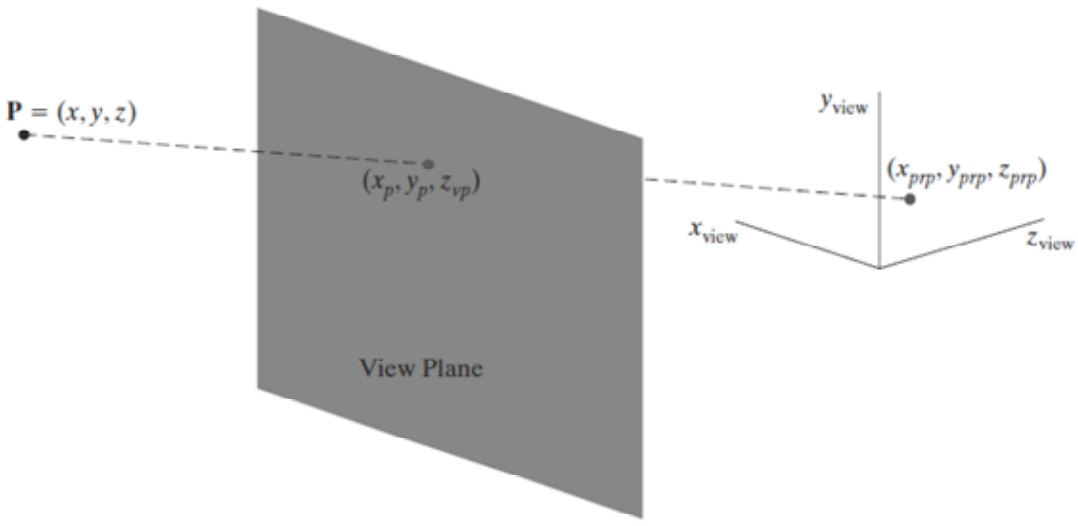


Figure 2: A perspective projection of a point P with coordinates (x , y, z) to a selected projection reference point. The intersection position on the view plane is (x_p, y_p,z_{vp}).

Substituting this value of *u* into the equations for x' and , y' we obtain the general perspective-transformation equations

Perspective-Projection Equations:

To simplify the perspective calculations, the projection reference point could be limited to positions along the zview axis, then

1. $x_{prp} = y_{prp} = 0$:

$$x_p = x\left(\frac{z_{prp} - z_{vp}}{z_{prp} - z}\right), \qquad y_p = y\left(\frac{z_{prp} - z_{vp}}{z_{prp} - z}\right)$$

Sometimes the projection reference point is fixed at the coordinate origin, and

2. $(x_{prp}, y_{prp}, z_{prp}) = (0, 0, 0)$:

$$x_p = x\left(\frac{z_{vp}}{z}\right), \qquad y_p = y\left(\frac{z_{vp}}{z}\right)$$

If the view plane is the uv plane and there are no restrictions on the placement of the projection reference point, then we have

3. $z_{vp} = 0$:

$$x_p = x\left(\frac{z_{prp}}{z_{prp} - z}\right) - x_{prp}\left(\frac{z}{z_{prp} - z}\right)$$
$$y_p = y\left(\frac{z_{prp}}{z_{prp} - z}\right) - y_{prp}\left(\frac{z}{z_{prp} - z}\right)$$

With the uv plane as the view plane and the projection reference point on the zview axis, the perspective equations are

4. $x_{prp} = y_{prp} = z_{vp} = 0$:

$$x_p = x\left(\frac{z_{prp}}{z_{prp} - z}\right), \qquad y_p = y\left(\frac{z_{prp}}{z_{prp} - z}\right)$$

The view plane is usually placed between the projection reference point and the scene, but, in general, the view plane could be placed anywhere except at the projection point.

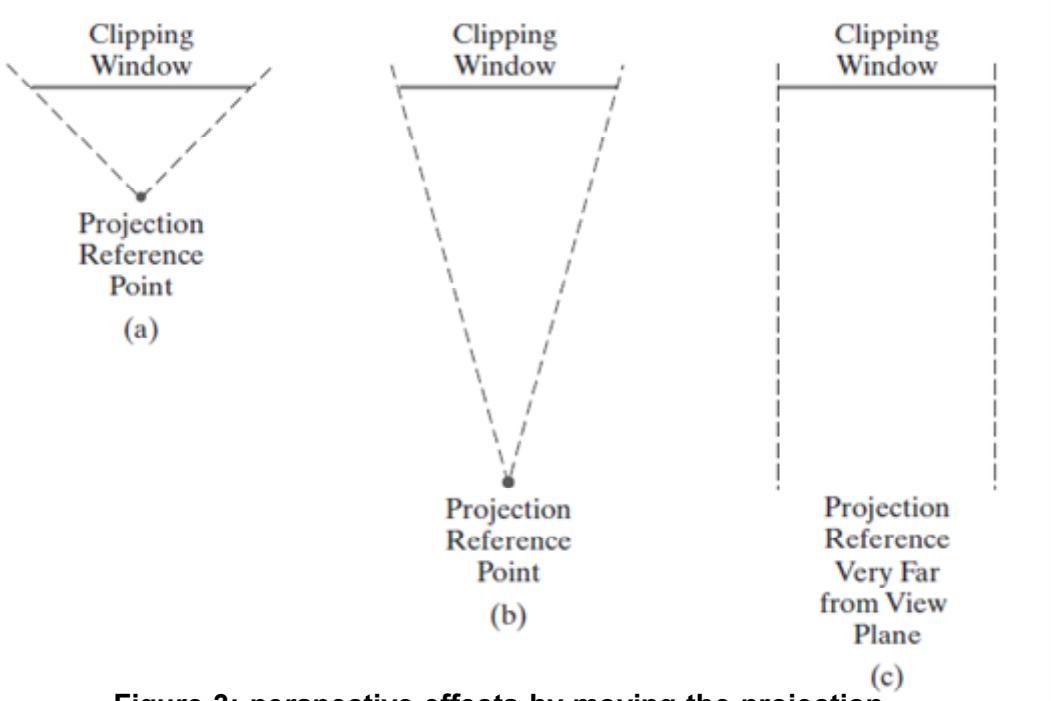


Figure 3: perspective effects by moving the projection reference point away from the view plane

Perspective effects also depend on the distance between the projection reference point and the view plane.

Vanishing Points for Perspective Projections

When a scene is projected onto a view plane using a perspective mapping, lines that are parallel to the view plane are projected as parallel lines. But any parallel lines in the scene that are not parallel to the view plane are projected into converging lines. The point at which a set of projected parallel lines appears to converge is called a vanishing point.

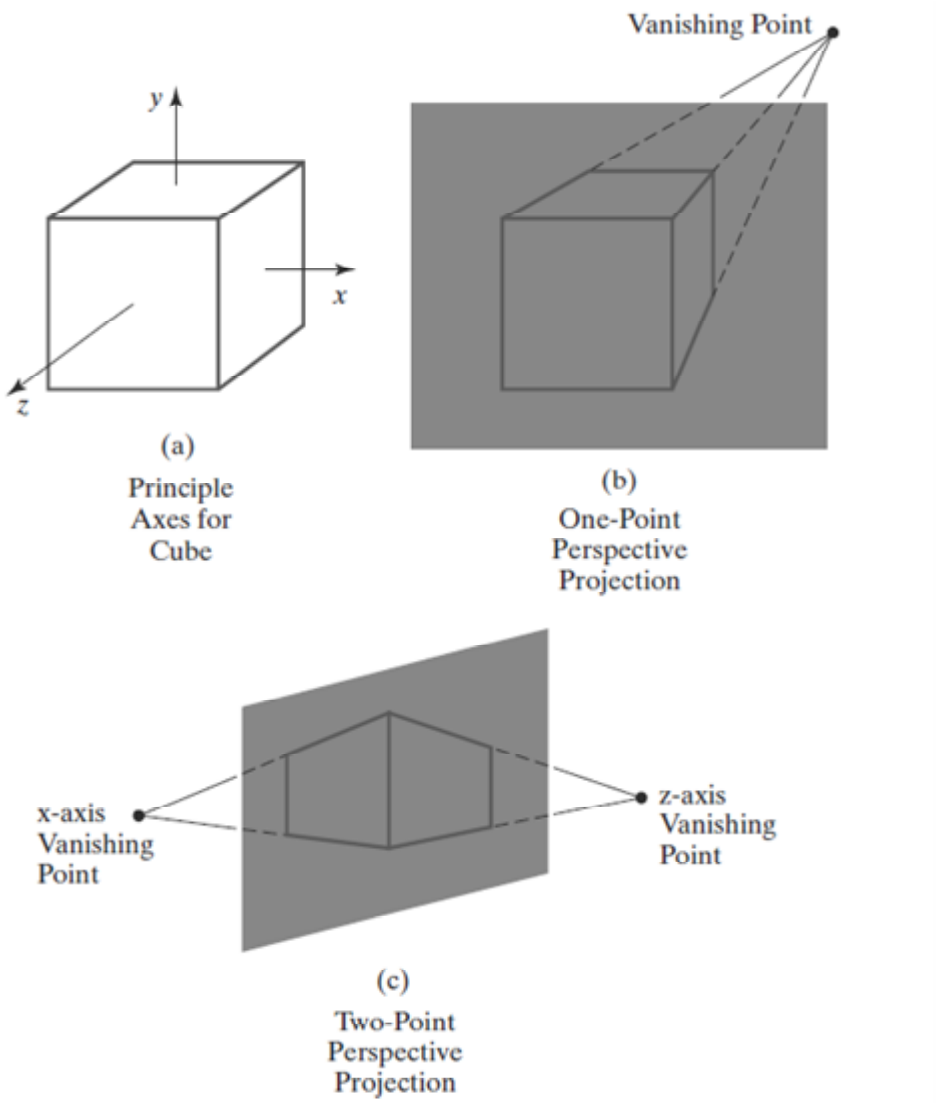


Figure 4: Vanishing Point at one-point and two-point perspective projection.

For a set of lines that are parallel to one of the principal axes of an object, the vanishing point is referred to as a principal vanishing point.

The number of principal vanishing points in a projection is equal to the number of principal axes that intersect the view plane. Perspective projections are accordingly classified as one-point, two-point, or three-point projections.

Perspective-Projection Transformation Matrix

We can use a three-dimensional, homogeneous-coordinate representation to express the perspective-projection equations in the form

$$x_p = \frac{x_h}{h}, \qquad y_p = \frac{y_h}{h}$$

where the homogeneous parameter has the value

$$h = z_{prp} - z$$

$$x_h = x(z_{prp} - z_{vp}) + x_{prp}(z_{vp} - z)$$
$$y_h = y(z_{prp} - z_{vp}) + y_{prp}(z_{vp} - z)$$

The perspective-projection transformation of a viewing-coordinate position is then accomplished in two steps. First, we calculate the homogeneous coordinates using the perspective-transformation matrix:

$$\mathbf{P}_h = \mathbf{M}_{\text{pers}} \cdot \mathbf{P}$$

where P_h is the column-matrix representation of the homogeneous point (x_h, y_h, z_h, h) and P is the column-matrix representation of the coordinate position (x, y, z, 1).

Second, after other processes have been applied, such as the normalization transformation and clipping routines, homogeneous coordinates are divided by parameter h to obtain the true transformation-coordinate positions.

The following matrix gives one possible way to formulate a perspective-projection matrix.

$$\mathbf{M}_{\text{pers}} = \begin{bmatrix} z_{prp} - z_{vp} & 0 & -x_{prp} & x_{prp}z_{prp} \\ 0 & z_{prp} - z_{vp} & -y_{prp} & y_{prp}z_{prp} \\ 0 & 0 & s_z & t_z \\ 0 & 0 & -1 & z_{prp} \end{bmatrix}$$

Parameters s_z and t_z are the scaling and translation factors for normalizing the projected values of z-coordinates. Specific values for s_z and t_z depend on the normalization range we select.

9. 4 Three-Dimensional Clipping Algorithms

We can apply three-dimensional clipping algorithms to the normalized boundaries of the view volume. This allows the viewing pipeline and the clipping procedures to be implemented in a highly efficient way. For the symmetric cube, the equations for the three-dimensional clipping planes are

$$\begin{aligned} xw_{\min} &= -1, & xw_{\max} &= 1 \\ yw_{\min} &= -1, & yw_{\max} &= 1 \\ zw_{\min} &= -1, & zw_{\max} &= 1 \end{aligned}$$

The x and y clipping boundaries are the normalized limits for the clipping window, and the z clipping boundaries are the normalized positions for the near and far clipping planes.

Clipping algorithms for three-dimensional viewing identify and save all object sections within the normalized view volume for display on the output device. All parts of objects that are outside the view-volume clipping planes are eliminated.

Clipping in Three-Dimensional Homogeneous Coordinates

Computer-graphics libraries process spatial positions as four-dimensional homogeneous coordinates so that all transformations can be represented as 4 by 4 matrices. As each coordinate position enters the viewing pipeline, it is converted to a four-dimensional representation:

$$(x, y, z) \rightarrow (x, y, z, 1)$$

After a position has passed through the geometric, viewing, and projection transformations, it is now in the homogeneous form

$$\begin{bmatrix} x_h \\ y_h \\ z_h \\ h \end{bmatrix} = \mathbf{M} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

where matrix M represents the concatenation of all the various transformations from world coordinates to normalized, homogeneous projection coordinates, and the homogeneous parameter h may no longer have the value 1.

Three-Dimensional Region Codes

We extend the concept of a region code to three dimensions by simply adding a couple of additional bit positions to accommodate the near and far clipping planes. Thus, we now use a six-bit region code. Bit positions in this region-code example are numbered from right to left, referencing the left, right, bottom, top, near, and far clipping planes, in that order.

For a three-dimensional scene, we need to apply the clipping routines to the projection coordinates, which have been transformed to a normalized space. After the projection transformation, each point in a scene has the four-component representation P (x_h, y_h, z_h, h).

A point is inside this normalized view volume if the projection coordinates of the point satisfy the following six inequalities:

$$-1 \leq \frac{x_h}{h} \leq 1, \qquad -1 \leq \frac{y_h}{h} \leq 1, \qquad -1 \leq \frac{z_h}{h} \leq 1$$

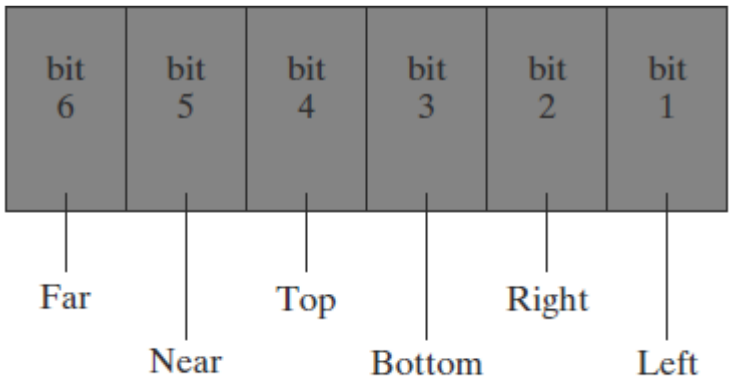


Figure 5: Region-code bit positions.

The homogeneous parameter can be either positive or negative. Therefore, assuming h ‘“0, we can write the preceding inequalities in the form

$$\begin{aligned} -h \le x_h \le h, \quad & -h \le y_h \le h, \quad & -h \le z_h \le h \quad & \text{if } h > 0 \\ h \le x_h \le -h, \quad & h \le y_h \le -h, \quad & h \le z_h \le -h \quad & \text{if } h < 0 \end{aligned}$$

In most cases h > 0, and we can then assign the bit values in the region code for a coordinate position according to the tests:

$$\begin{aligned} &\text{bit } 1 = 1 \quad \text{if } h + x_h < 0 \quad \text{(left)} \\ &\text{bit } 2 = 1 \quad \text{if } h - x_h < 0 \quad \text{(right)} \\ &\text{bit } 3 = 1 \quad \text{if } h + y_h < 0 \quad \text{(bottom)} \\ &\text{bit } 4 = 1 \quad \text{if } h - y_h < 0 \quad \text{(top)} \\ &\text{bit } 5 = 1 \quad \text{if } h + z_h < 0 \quad \text{(near)} \\ &\text{bit } 6 = 1 \quad \text{if } h - z_h < 0 \quad \text{(far)} \end{aligned}$$

We simply use the sign bit of one of the calculations $h \pm x_h$, $h \pm y_h$, or $h \pm z_h$ to set the corresponding region-code bit value.

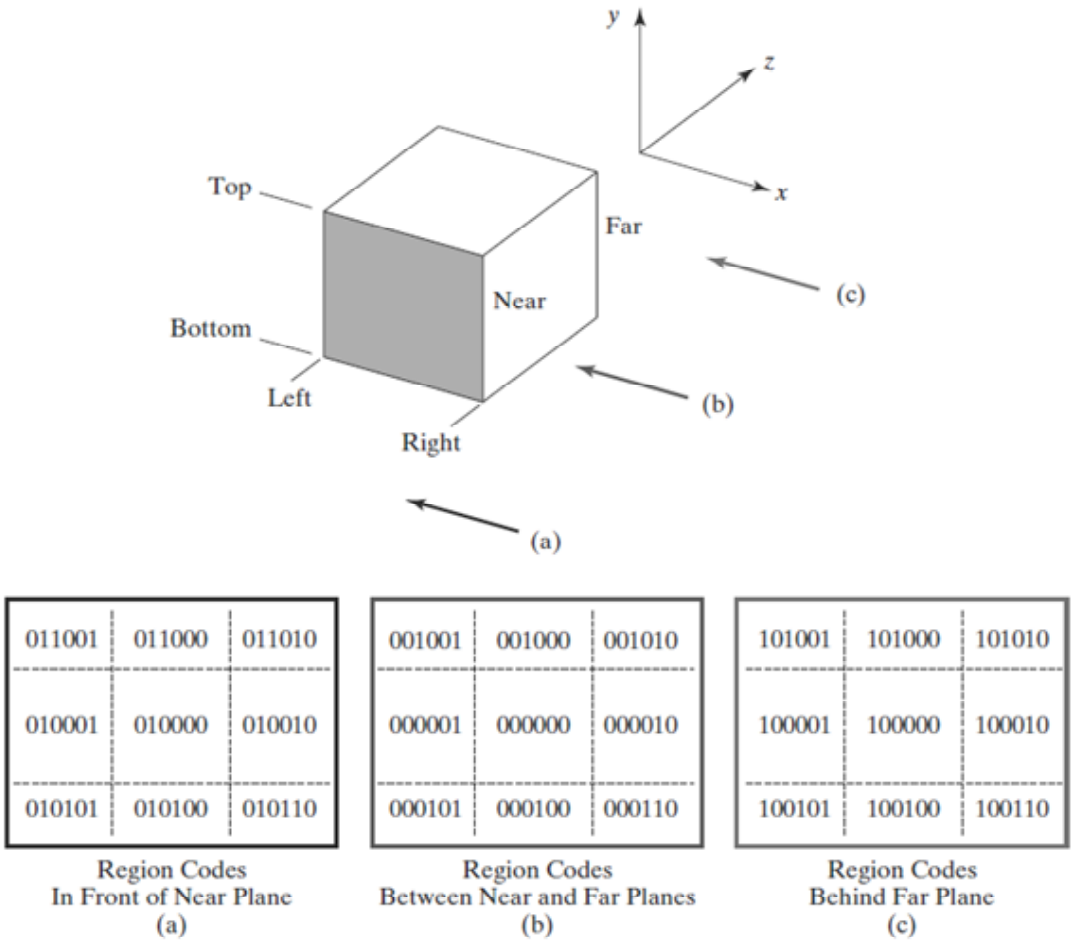


Figure 6: Values for the three-dimensional, six-bit region code

9. 5 Three-Dimensional Point and Line Clipping

For standard point positions and straight-line segments that are defined in a scene that is not behind the projection reference point, all homogeneous parameters are positive and the region codes can be established using the conditions. Then, once we have set up the region code for each position in a scene, we can easily identify a point position as outside the view volume or inside the view volume. For point clipping, we simply eliminate any individual point whose region code is not 000000. In other words, if any one of the tests is negative, the point is outside the view volume.

Methods for three-dimensional line clipping are essentially the same as for two-dimensional lines. We can first test the line endpoint region codes for trivial acceptance or rejection of the line. If the region code for both endpoints of a line is 000000, the line is completely inside the view volume. Equivalently, we can trivially accept the line if the logical or operation on the two endpoint region codes produces a value of 0. And we can trivially reject the line if the logical and operation on the two endpoint region codes produces a value that is not 0. This nonzero value indicates that both endpoint region codes have a 1 value in the same bit position, and hence the line is completely outside one of the clipping planes.

Equations for three-dimensional line segments are conveniently expressed in parametric form

For a line segment with endpoints $P_1 = (x_{h1}, y_{h1}, z_{h1}, h1)$ and $P_2 = (x_{h2}, y_{h2}, z_{h2}, h_2)$, we can write the parametric equation describing any point position along the line as

$$\mathbf{P} = \mathbf{P}_1 + (\mathbf{P}_2 - \mathbf{P}_1)u \qquad 0 \le u \le 1$$

When the line parameter has the value u = 0, we are at position P_1. And u = 1 brings us to the other end of the line, P_2. Writing the parametric line equation explicitly, in terms of the homogeneous coordinates, we have

$$\begin{aligned} x_h &= x_{h1} + (x_{h2} - x_{h1})u \\ y_h &= y_{h1} + (y_{h2} - y_{h1})u \\ z_h &= z_{h1} + (z_{h2} - z_{h1})u \\ h &= h_1 + (h_2 - h_1)u \end{aligned} \qquad 0 \le u \le 1$$

Using the endpoint region codes for a line segment, we can first determine which clipping planes are intersected. If one of the endpoint region codes has a 0 value in a certain bit position while the other code has a 1 value in the same bit position, then the line crosses that clipping boundary.

Therefore, we determine the intersection value for parameter u by setting the x-projection coordinate equal to 1:

$$x_p = \frac{x_h}{h} = \frac{x_{h1} + (x_{h2} - x_{h1})u}{h_1 + (h_2 - h_1)u} = 1$$

Solving for parameter u, we obtain

$$u = \frac{x_{h1} - h_1}{(x_{h1} - h_1) - (x_{h2} - h_2)}$$

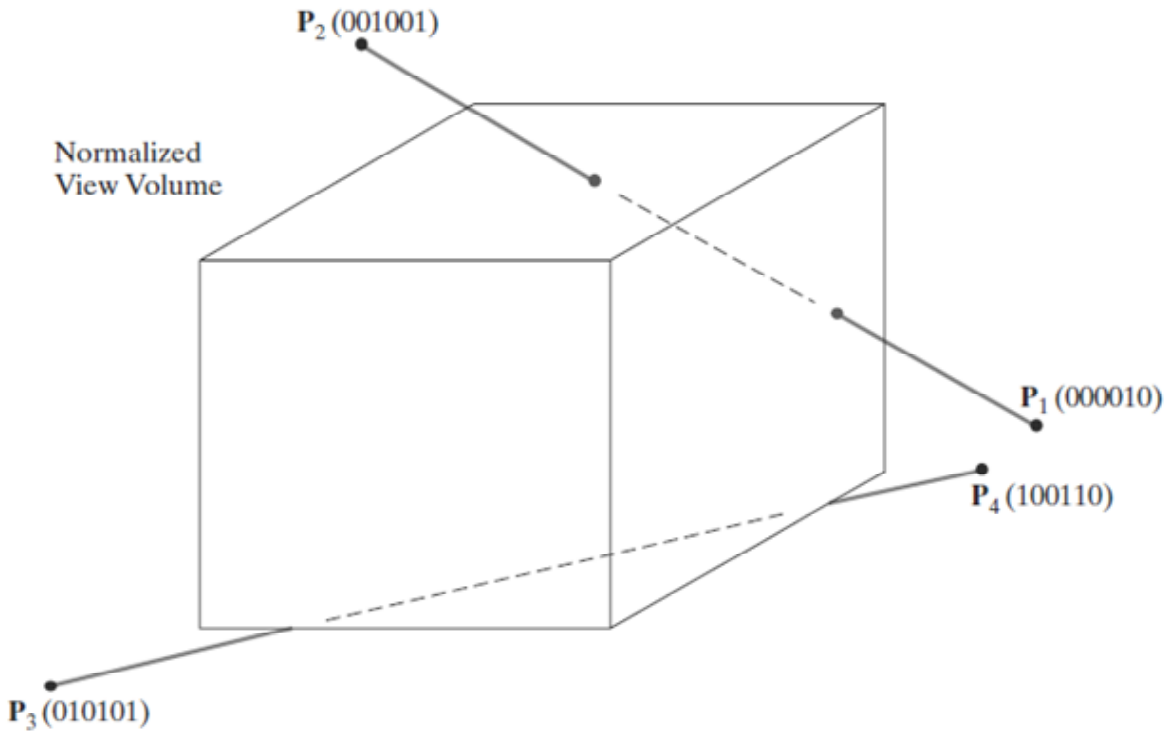


Figure 7: Three-dimensional region codes for two line segments.

Next, we determine the values y_p and z_p on this clipping plane, using the calculated value for u. In this case, the y_p and z_p intersection values are within the ±1 boundaries of the view

volume and the line does cross into the view-volume interior. So we next proceed to locate the intersection position with the top clipping plane. That completes the processing for this line segment, because the intersection points with the top and right clipping planes identify the part of the line that is inside the view volume and all the line sections that are outside the view volume.

9.6 Three-Dimensional Polygon Clipping

We can first test a polyhedron for trivial acceptance or rejection using its coordinate extents, a bounding sphere, or some other measure of its coordinate limits. If the coordinate limits of the object are inside all clipping boundaries, we save the entire object. If the coordinate limits are all outside any one of the clipping boundaries, we eliminate the entire object.

When we cannot save or eliminate the entire object, we can next process the vertex lists for the set of polygons that define the object surfaces. Applying methods similar to those in two-dimensional polygon clipping, we can clip edges to obtain new vertex lists for the object surfaces. We may also need to create some new vertex lists for additional surfaces that result from the clipping operations. And the polygon tables are updated to add any new polygon surfaces and to revise the connectivity and shared-edge information about the surfaces.

To simplify the clipping of general polyhedra, polygon surfaces are often divided into triangular sections and described with triangle strips. We can then clip the triangle strips using the Sutherland-Hodgman approach. Each triangle strip is processed in turn against the six clipping planes to obtain the final vertex list for the strip.

9.7 Three-Dimensional Curve Clipping

We first check to determine whether the coordinate extents of a curved object, such as a sphere or a spline surface, are completely inside the view volume. Then we can check to determine whether the object is completely outside any one of the six clipping planes.

If the trivial rejection-acceptance tests fail, we locate the intersections with the clipping planes. To do this, we solve the simultaneous set of surface equations and the clipping-plane equation. In most graphics packages curved surfaces are approximated as a set of polygon patches, and the objects are then clipped using polygon-clipping routines.

9.8 Summary

Perspective projections of objects are obtained with projection lines that meet at the projection reference point.

Parallel projections maintain object proportions, but perspective projections decrease the size of distant objects.

Perspective projections cause parallel lines to appear to converge to a vanishing point, provided the lines are not parallel to the view plane.

Engineering and architectural displays can be generated with one-point, two-point, or three-point perspective projections, depending on the number of principal axes that intersect the view plane. An oblique perspective projection is obtained when the line from the projection reference point to the center of the clipping window is not perpendicular to the view plane.

Objects in a three-dimensional scene can be clipped against a view volume to eliminate unwanted sections of the scene.

The top, bottom, and sides of the view volume are formed with planes that are parallel to the projection lines and that pass through the clipping-window edges.

Near and far planes (tgalso called front and back planes) are used to create a closed view volume.

Clipping is generally carried out in graphics packages in four-dimensional homogeneous coordinates following the projection and view-volume normalization transformations.

Homogeneous coordinates are converted to three-dimensional, Cartesian projection coordinates.

Additional clipping planes, with arbitrary orientations, can also be used to eliminate selected parts of a scene or to produce special effects.

9.9 Model Questions

1. Briefly explain about the basic transformations performed on three dimensional objects.

2. Write short notes on parallel and perspective projections.

3. Explain in detail about three dimensional display methods.

4. Explain in detail about the boundary representation of three dimensional objects.

5. Explain in detail about the three dimensional transformations.

LESSON - 10
VISIBLE SURFACE DETECTION METHODS - I

Structure of the lesson

10.1 Introduction

A major consideration in the generation of realistic graphics displays is determining what is visible within a scene from a chosen viewing position. There are a number of approaches we can take to accomplish this, and numerous algorithms have been devised for efficient identification and display of visible objects for different types of applications. Some methods require more memory, some involve more processing time, and some apply only to special types of objects. Which method we select for a particular application can depend on such factors as the complexity of the scene, type of objects to be displayed, available equipment, and whether static or animated displays are to be generated. The various algorithms are referred to as visible-surface detection methods. Sometimes these methods are also referred to as hidden-surface elimination methods, although there can be subtle differences between identifying visible surfaces and eliminating hidden surfaces.

10.2 Objective of this Lesson

In this lesson we will have a detail study of various Visible-Surface Detection Algorithms.

- Back-Face Detection
- Depth-Buffer Method
- A-Buffer Method
- Scan-Line Method

10.3 Classification of Visible-Surface Detection Algorithms

We can broadly classify visible-surface detection algorithms according to whether they deal with the object definitions or with their projected images. These two approaches are called object-space methods and image-space methods, respectively.

Object Space

An object-space method compares objects and parts of objects to each other within the scene definition to determine which surfaces, as a whole, we should label as visible.

Image Sspace

In an image-space algorithm, visibility is decided point by point at each pixel position on the projection plane. Most visible-surface algorithms use image-space methods, although object-space methods can be used effectively to locate visible surfaces in some cases.

Although there are major differences in the basic approaches taken by the various visible-surface detection algorithms, most use sorting and coherence methods to improve performance. Sorting is used to facilitate depth comparisons by ordering Although there are major differences in the basic approaches taken by the various visible-surface detection algorithms, most use sorting and coherence methods to improve performance. Sorting is used to facilitate depth comparisons by ordering the individual surfaces in a scene according to their distance from the view plane. Coherence methods are used to take advantage of regularities in a scene the individual surfaces in a scene according to their distance from the view plane. Coherence methods are used to take advantage of regularities in a scene.

10.4 Back-Face Detection

A fast and simple object-space method for locating the back faces of a polyhedron is based on front-back tests. A point (x, y, z) is behind a polygon surface if

$$Ax + By + Cz + D < 0$$

where A, B,C, and Dare the plane parameters for the polygon. When this position is along the line of sight to the surface, we must be looking at the back of the polygon. Therefore, we could use the viewing position to test for back faces.

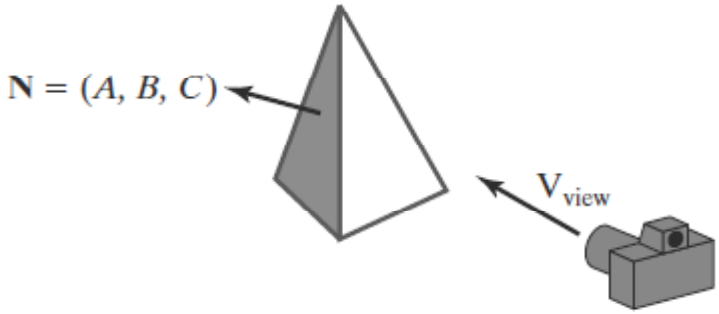


Figure 1: A surface normal vector N and the viewing-direction vector V_{view}.

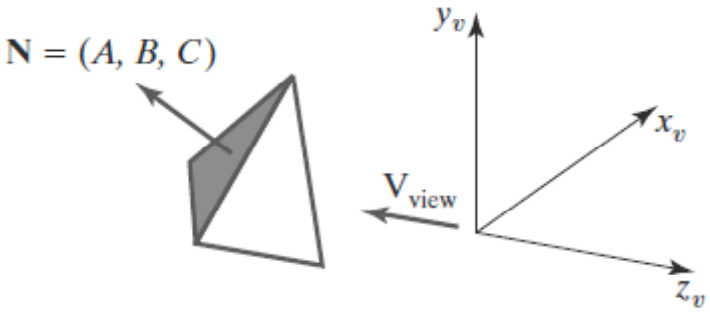


Figure 2: A polygon surface with plane parameter C < 0 in a right-handed viewing coordinate system is identified as a back face when the viewing direction is along the negative z v axis.

We can simplify the back-face test by considering the direction of the normal vector N for a polygon surface. If V_{view} is a vector in the viewing direction from our camera position, as shown in Figure 1, then a polygon is a back face if

$$\mathbf{V}_{view} \cdot \mathbf{N} > 0$$

If object descriptions have been converted to projection coordinates and our viewing direction is parallel to the viewing zv axis, then we need to consider only the z component of the normal vector N. In a right-handed viewing system with the viewing direction along the negative z_v axis (Figure 2), a polygon is a back face if the z component, C, of its normal vector N satisfies C < 0. Also, we cannot see any face whose normal has z component C = 0, because our viewing direction is grazing that polygon. Thus, in general, we can label any polygon as a back face if its normal vector has a z component value that satisfies the inequality

$$C \leq 0$$

Similar methods can be used in packages that employ a left-handed viewing system. In these packages, plane parameters A, B, C, and D can be calculated from polygon vertex coordinates specified in a clockwise direction. Inequality 1 then remains a valid test for points behind the polygon. Also, back faces have normal vectors that point away from the viewing position and are identified by C e" 0 when the viewing direction is along the positive z_v axis.

By examining parameter C for the different plane surfaces describing an object, we can immediately identify all the back faces. For a single convex polyhedron, such as the pyramid in Figure 2, this test identifies all the hidden surfaces in the scene, because each surface is either completely visible or completely hidden. Also, if a scene contains only nonoverlapping convex polyhedra, then again all hidden surfaces are identified with the back-face method.

For other objects, such as the concave polyhedron more tests must be carried out to determine whether there are additional faces that are totally or partially obscured by other faces. A general scene can be expected to contain overlapping objects along the line of sight, and we then need to determine where the obscured objects are partly or completely hidden by other objects.

In general, back-face removal can be expected to eliminate about half of the polygon surfaces in a scene from further visibility tests.

10.5 Depth-Buffer Method

A commonly used image-space approach for detecting visible surfaces is the depth-buffer method, which compares surface depth values throughout a scene for each pixel position on the projection plane. Each surface of a scene is processed separately, one pixel position at a time, across the surface. The algorithm is usually applied to scenes containing only polygon surfaces, because depth values can be computed very quickly and the method is easy to implement. But we could also apply the same procedures to nonplanar surfaces. This visibility-detection approach is also frequently alluded to as the z-buffer method, because object depth is usually measured along the z axis of a viewing system. However, rather than using actual z coordinates within the scene, depth-buffer algorithms often compute a distance from the view plane along the z axis.

Figure 3 shows three surfaces at varying distances along the orthographic projection line from position (x, y) on a view plane. These surfaces can be processed in any order. As each surface is processed, its depth from the view plane is compared to previously processed surfaces. If a surface is closer than any previously processed surfaces, its surface color is calculated and saved, along with its depth.

The visible surfaces in a scene are represented by the set of surface colors that have been saved after all surface processing is completed. Implementation of the depth-buffer algorithm is typically carried out in normalized coordinates, so that depth values range from 0 at the near clipping plane (the view plane) to 1.0 at the far clipping plane.

As implied by the name of this method, two buffer areas are required. A depth buffer is used to store depth values for each (x, y) position as surfaces are processed, and the frame buffer stores the surface-color values for each pixel position. Initially, all positions in the depth buffer are set to 1.0 (maximum depth), and the frame buffer (refresh buffer) is initialized to the background color. Each surface listed in the polygon tables is then processed, one scan line at a time, by calculating the depth value at each (x, y) pixel position. This calculated depth is compared to the value previously stored in the depth buffer for that pixel position.

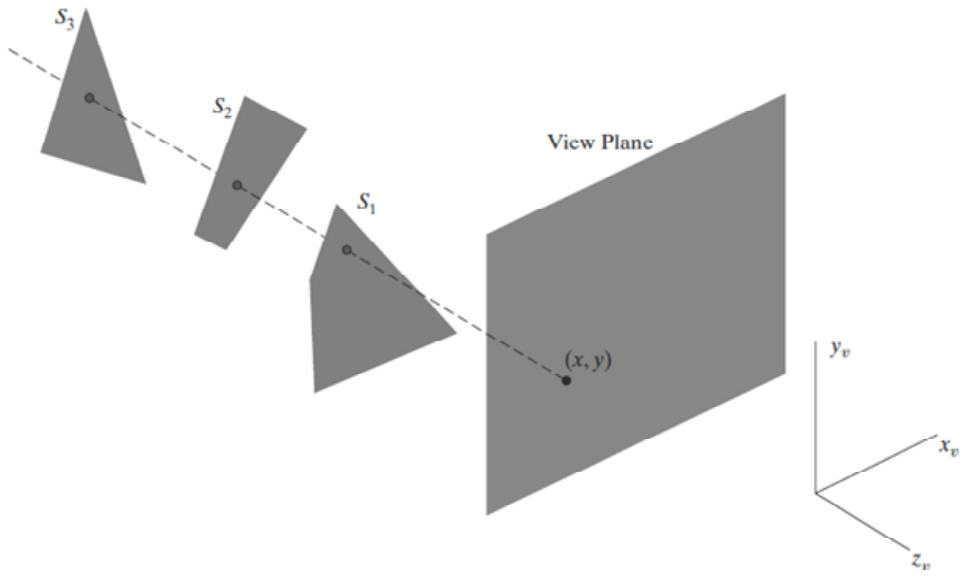


Figure 3: Three surfaces overlapping pixel position (x , y) on the view plane. The visible surface, S_1, has the smallest depth value.

If the calculated depth is less than the value stored in the depth buffer, the new depth value is stored. Then the surface color at that position is computed and placed in the corresponding pixel location in the frame buffer.

The depth-buffer processing steps are summarized in the following algorithm.

This algorithm assumes that depth values are normalized on the range from 0.0 to 1.0 with the view plane at depth= 0 and the farthest depth= 1.We can also apply this algorithm for any other depth range, and some graphics packages allow the user to specify the depth range over which the depth-buffer algorithm is to be applied. Within the algorithm, the variable z represents the depth of the polygon (that is, its distance from the view plane along the negative z axis).

Depth-Buffer Algorithm

1. Initialize the depth buffer and frame buffer so that for all buffer positions (x, y),

   depthBuff (x, y) = 1.0, frameBuff (x, y) = backgndColor

2. Process each polygon in a scene, one at a time, as follows:

- For each projected (x, y) pixel position of a polygon, calculate the depth z (if not already known).
- If z < depthBuff (x, y), compute the surface color at that position and set

  depthBuff (x, y) = z, frameBuff (x, y) = surfColor (x, y)

After all surfaces have been processed, the depth buffer contains depth values for the visible surfaces and the frame buffer contains the corresponding color values for those surfaces.

Given the depth values for the vertex positions of any polygon in a scene, we can calculate the depth at any other point on the plane containing the polygon.

At surface position (x, y), the depth is calculated from the plane equation as

$$z = \frac{-Ax - By - D}{C}$$

For any scan line (Figure 4), adjacent horizontal x positions across the line differ by ±1, and vertical y values on adjacent scan lines differ by ±1. If the depth of position (x, y) has been determined to be z, then the depth z of the next position (x + 1, y) along the scan line is the above equation.

$$z' = \frac{-A(x+1) - By - D}{C}$$

or

$$z' = z - \frac{A}{C}$$

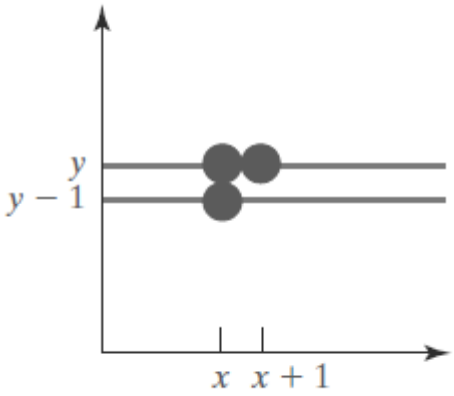


Figure 4: From position (x , y) on a scan line, the next position across the line has coordinates (x + 1, y) , and the position immediately below on the next line has coordinates (x , y - 1) .

The ratio -A/C is constant for each surface, so succeeding depth values across a scan line are obtained from preceding values with a single addition.

Processing pixel positions from left to right across each scan line, we start by calculating the depth on a left polygon edge that intersects that scan line. For each successive position across the scan line, we then calculate the depth value. We can implement the depth-buffer algorithm by starting at a top vertex of the polygon. Then, we could recursively calculate the x-coordinate values down a left edge of the polygon. The x value for the beginning position on each scan line can be calculated from the beginning (edge) x value of the previous scan line as

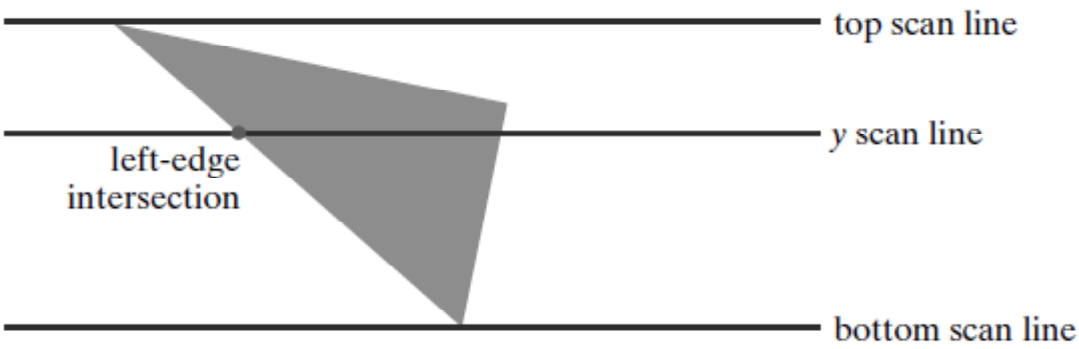


Figure 5: Scan lines intersecting a polygon surface.

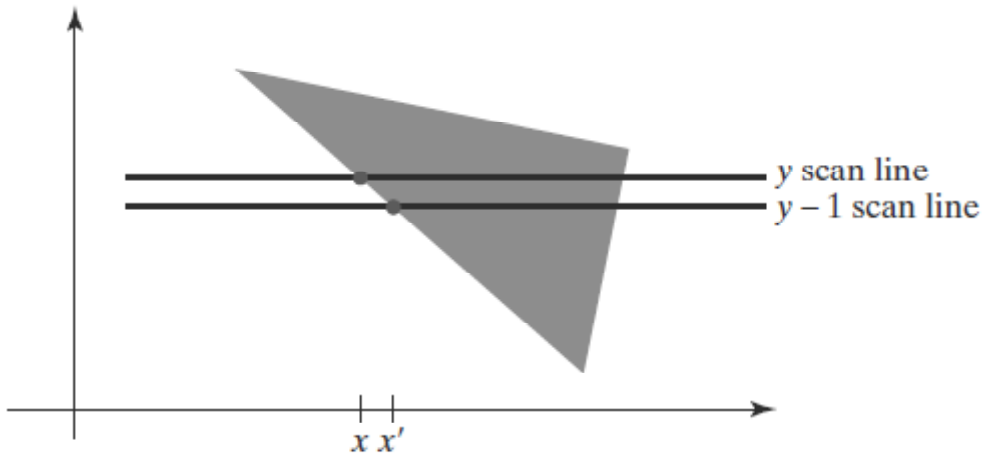


Figure 6: Intersection positions on successive scan lines along a left polygon edge.

$$x' = x - \frac{1}{m}$$

where m is the slope of the edge (Figure 6). Depth values down this edge are obtained recursively as

$$z' = z + \frac{A/m + B}{C}$$

If we are processing down a vertical edge, the slope is infinite and the recursive calculations reduce to

$$z' = z + \frac{B}{C}$$

One slight complication with this approach is that while pixel positions are at integer (x, y) coordinates, the actual point of intersection of a scan line with the edge of a polygon may not be. As a result, it may be necessary to adjust the intersection point by rounding its fractional part up or down, as is done in scan-line polygon fill algorithms.

An alternative approach is to use a midpoint method or Bresenham-type algorithm for determining the starting x values along edges for each scan line. The method can be applied to curved surfaces by determining depth and color values at each surface projection point.

For polygon surfaces, the depth-buffer method is very easy to implement, and it requires no sorting of the surfaces in a scene. But it does require the availability of a second buffer in addition to the refresh buffer. A system with a resolution of 1280 × 1024, for example, would require over 1.3 million positions in the depth buffer, with each position containing enough bits to represent the number of depth increments needed. One way to reduce storage requirements is to process one section of the scene at a time, using a smaller depth buffer. After each view section is processed, the buffer is reused for the next section.

10.6 A-Buffer Method

An extension of the depth-buffer ideas is the A-buffer procedure (at the other end of the alphabet from "z-buffer," where z represents depth). This depth-buffer extension is an antialiasing, area-averaging, visibility-detection method developed at Lucasfilm Studios for inclusion in the surface-rendering system called REYES (an acronym for "Renders Everything You Ever Saw"). The buffer region for this procedure is referred to as the accumulation buffer, because it is used to store a variety of surface data, in addition to depth values.

A drawback of the depth-buffer method is that it identifies only one visible surface at each pixel position. In other words, it deals only with opaque surfaces and cannot accumulate color values for more than one surface, as is necessary if transparent surfaces are to be displayed (Figure 7). The A-buffer method expands the depth-buffer algorithm so that each position in the buffer can reference a linked list of surfaces. This allows a pixel color to be computed as a combination of different surface colors for transparency or antialiasing effects.

Each position in the A-buffer has two fields:

- Depth field: Stores a real-number value (positive, negative, or zero).
- Surface data field: Stores surface data or a pointer.

If the depth field is nonnegative, the number stored at that position is the depth of a surface that overlaps the corresponding pixel area. The surface data field then stores various surface information, such as the surface color for that position and the percent of pixel coverage, as illustrated in Figure 8(a). If the depth field for a position in the A-buffer is negative, this indicates multiple-surface contributions to the pixel color. The color field then stores a pointer to a linked list of surface data, as in Figure 8(b). Surface information in the A-buffer includes

- RGB intensity components
- Opacity parameter (percent of transparency)
- Depth
- Percent of area coverage
- Surface identifier
- Other surface-rendering parameters

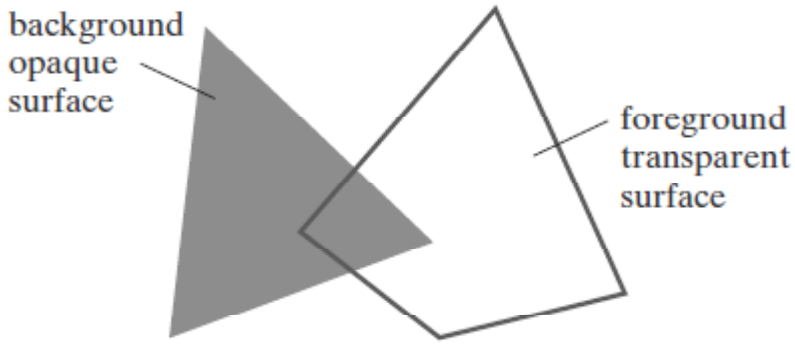


Figure 7: Viewing an opaque surface through a transparent surface requires multiple color inputs and the application of color-blending operations.

The A-buffer visibility-detection scheme can be implemented using methods similar to those in the depth-buffer algorithm. Scan lines are processed to determine how much of each surface covers each pixel position across the individual scan lines. Surfaces are subdivided into a polygon mesh and clipped against the pixel boundaries. Using the opacity factors and percent of surface coverage, the rendering algorithms calculate the color for each pixel as an average of the contributions from the overlapping surfaces.

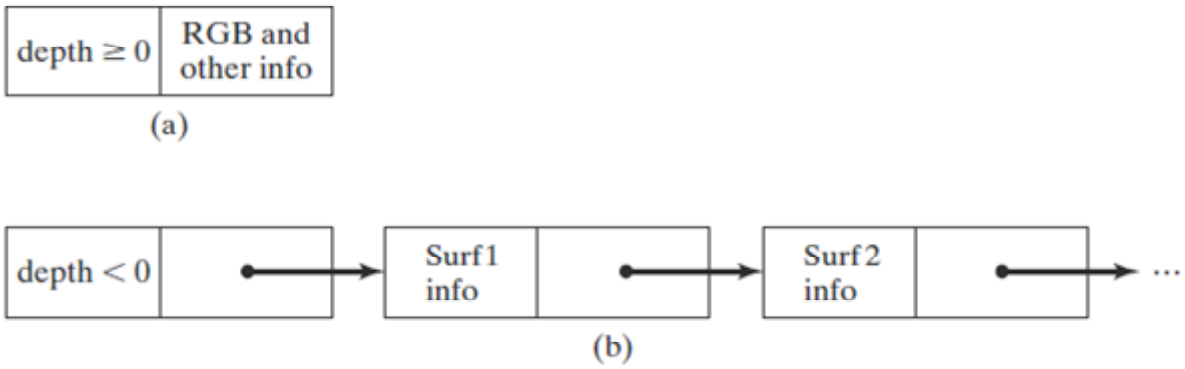


Figure 8: Two possible organizations for surface information in an A-buffer representation for a pixel position. When a single surface overlaps the pixel, the surface depth, color, and other information are stored as in (a). When more than one surface overlaps the pixel, a linked list of surface data is stored as in (b).

10.7 Scan-Line Method

This image-space method for identifying visible surfaces computes and compares depth values along the various scan lines for a scene. As each scan line is processed, all polygon surface projections intersecting that line are examined to determine which are visible. Across each scan line, depth calculations are performed to determine which surface is nearest to the view plane at each pixel position. When the visible surface has been determined for a pixel, the surface color for that position is entered into the frame buffer.

Surfaces are processed using the information stored in the polygon tables. The edge table contains coordinate endpoints for each line in the scene, the inverse slope of each line, and pointers into the surface-facet table to identify the surfaces bounded by each line. The surface-facet table contains the plane coefficients, surface material properties, other surface data, and possibly pointers into the edge table. To facilitate the search for surfaces crossing a given scan line, an active list of edges is formed for each scan line as it is processed. The active edge list contains only those edges that cross the current scan line, sorted in order of increasing x. In addition, we define a flag for each surface that is set to “on” or “off” to indicate whether a position along a scan line is inside or outside the surface. Pixel positions across each scan line are processed from left to right.

At the left intersection with the surface projection of a convex polygon, the surface flag is turned on; at the right intersection point along the scan line, it is turned off. For a concave polygon, scan-line intersections can be sorted from left to right, with the surface flag set to “on” between each intersection pair.

Figure 9 illustrates the scan-line method for locating visible portions of surfaces for pixel positions along a scan line. The active list for scan line 1 contains information from the edge table for edges AB, BC, EH, and FG. For positions along this scan line between edges AB and BC, only the flag for surface S_1 is on. Therefore, no depth calculations are necessary, and color values are calculated from the surface properties and lighting conditions for surface S_1. Similarly, between edges EH and FG, only the flag for surface S_2 is on. No other positions along scan line 1 intersect surfaces, so the color for those pixels is the background color, which could be loaded into the frame buffer as part of the initialization routine. For scan lines 2 and 3, the active edge list contains edges AD, EH, BC, and FG. Along scan line 2 from edge AD to edge EH, only the flag for surface S_1 is on. But between edges EH and BC, the flags for both surfaces are on.

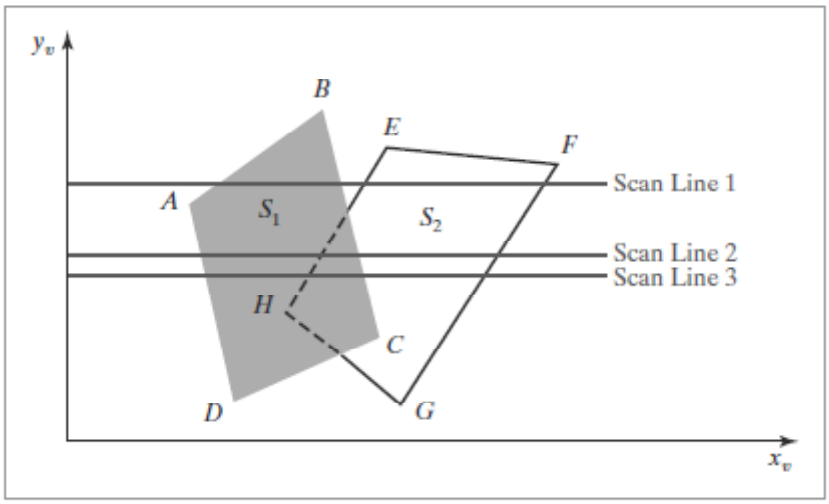


Figure 9: Scan lines crossing the view-plane projection of two surfaces, S_1 and S_2.

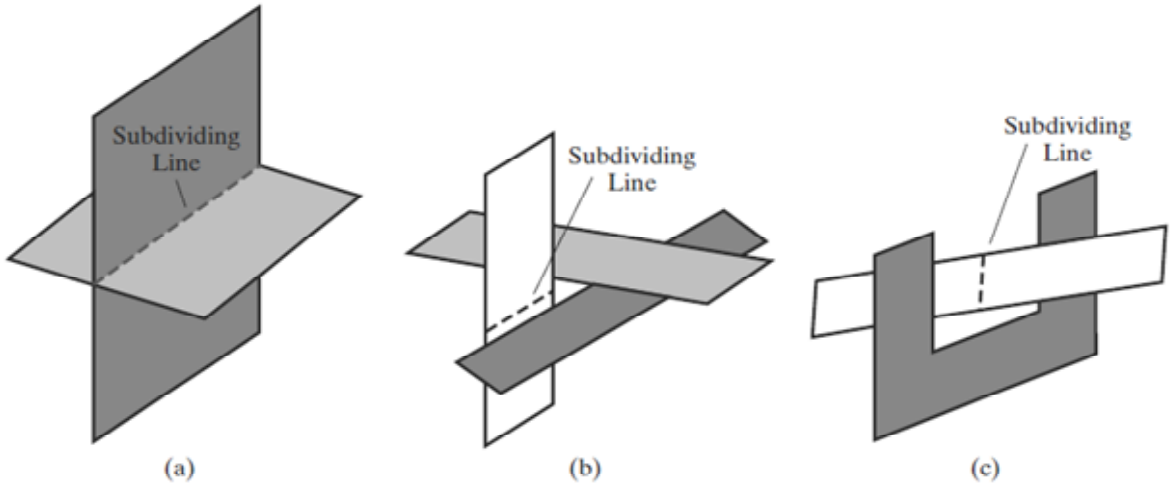


Figure 10: Intersecting and cyclically overlapping surfaces that alternately obscure one another.

Therefore, a depth calculation is necessary, using the plane coefficients for the two surfaces, when we encounter edge EH. For this example, the depth of surface S_1 is assumed to be less than that of S_2, so the color values for surface S_1 are assigned to the pixels across the scan line until boundary BC is encountered. Then the surface flag for S_1 goes off, and the colors for surface S_2 are stored up to edge FG. No other depth calculations are necessary, because we assume that surface S_2 remains behind S_1 once we have determined the depth relationship at edge EH.

We can take advantage of coherence along the scan lines as we pass from one scan line to the next. Scan line 3 has the same active list of edges as scan line 2. No changes have occurred in line intersections, so it is again unnecessary to make depth calculations between edges EH and BC. The two surfaces must be in the same orientation as determined on scan line 2, so the colors for surface S_1 can be entered without further depth calculations.

Any number of overlapping polygon surfaces can be processed with this scan line method. Flags for the surfaces are set to indicate whether a position is inside or outside, and depth calculations are performed only at the edges of overlapping surfaces. This procedure works correctly only if surfaces do not cut through or otherwise cyclically overlap each other (Figure 10). If any kind of cyclic overlap is present in a scene, we can divide the surfaces to eliminate

the overlaps. The dashed lines in this figure indicate where planes could be subdivided to form two distinct surfaces, so that the cyclic overlaps are eliminated.

10.8 Summary

The simplest visibility test is the back-face detection algorithm, which is fast and effective as an initial screening to eliminate many polygons from further visibility tests.

For a single convex polyhedron, back-face detection eliminates all hidden surfaces but, in general, back-face detection cannot completely identify all hidden surfaces.

A commonly used method for identifying all visible surfaces in a scene is the depth-buffer algorithm.

In depth-buffer method two buffers are needed: one to store pixel colors and one to store the depth values for the pixel positions.

Fast, incremental, scan-line methods are used to process each polygon in a scene to calculate surface depths. As each surface is processed, the two buffers are updated.

An extension of the depth-buffer approach is the A-buffer, which provides additional information for displaying antialiased and transparent surfaces.

The scanline method processes all surfaces at once for each scan line.

10.9 Model Questions

1. What is meant by Visible-Surface Detection?
2. Define object-space method.
3. Define image-space methods.
4. Explain how Back-Face Detection algorithm detects Visible-Surface Detection.
5. Explain Depth-Buffer Method.
6. Explain A-Buffer Method and its significance in detail.
7. Explain Scan-Line Method.

LESSON - 11
VISIBLE SURFACE DETECTION METHODS - II

Structure of the lesson

11.1 Introduction

In the previous lesson we have learnt few Visible-Surface Detection Algorithms namely Back-Face Detection, Depth-Buffer Method, A-Buffer Method, Scan-Line Method. Several other visibility-detection methods have been devised.

In the lesson we are going to discuss visible surface detection algorithms Depth-Sorting Method, BSP-Tree Method, Area-Subdivision Method, Octree Methods. Visible surfaces can also be detected using raycasting methods, which project lines from the pixel plane into a scene to determine object intersection positions along these projected lines.

11.2 Objective of this Lesson

In this lesson we are going to learn few visible surface detection algorithms namely Depth-Sorting Method, BSP-Tree Method, Area-Subdivision Method, Octree Methods.

We will have a comparative study on Visible-Surface Detection algorithm and know the pros and cons.

Will also know which algorithm will be suitable for which image.

11.3 Depth-Sorting Method

Using image-space and object-space operations, the depth-sorting method performs the following basic functions:

1. Surfaces are sorted in order of decreasing depth.
2. Surfaces are scan-converted in order, starting with the surface of greatest depth.

Sorting operations are carried out in both image and object space, and the scan conversion of the polygon surfaces is performed in image space.

This visibility-detection method is often referred to as the painter's algorithm. In creating an oil painting, an artist first paints the background colors. Next, the most distant objects are added, then the nearer objects, and so forth. At the final step, the foreground is painted on the canvas over the background and the more distant objects. Each color layer covers up the previous layer.

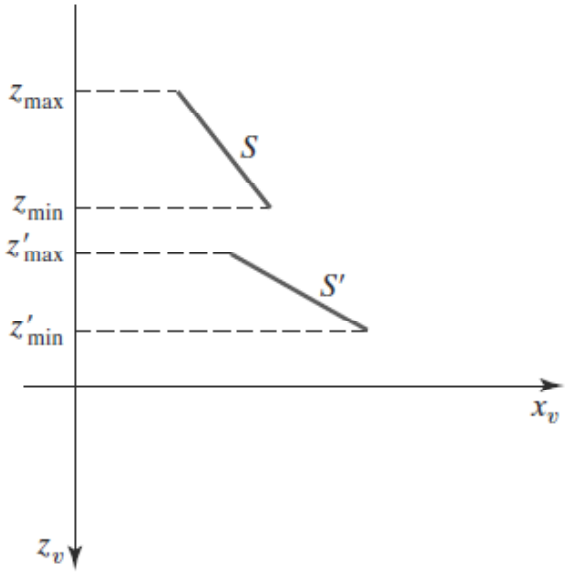


Figure 1: Two surfaces with no depth overlap.

Using a similar technique, we first sort surfaces according to their distance from the view plane. The color values for the farthest surface can then be entered into the refresh buffer. Taking each succeeding surface in turn (in decreasing depth order), we “paint” the surface onto the frame buffer over the colors of the previously processed surfaces.

Painting polygon surfaces into the frame buffer according to depth is carried out in several steps. Assuming we are viewing along the z direction, surfaces are ordered on the first pass according to the smallest z value on each surface. The surface S at the end of the list (with the greatest depth) is then compared to the other surfaces in the list to determine whether there are any depth overlaps. If no depth overlaps occur, S is the most distant surface and it is scan-converted. Figure 1 shows two surfaces that overlap in the xy plane but have no depth overlap. This process is then repeated for the next surface in the list. So long as no overlaps occur, each surface is processed in depth order until all have been scan-converted. If a depth overlap is detected at any point in the list, we need to make some additional comparisons to determine whether any of the surfaces should be reordered.

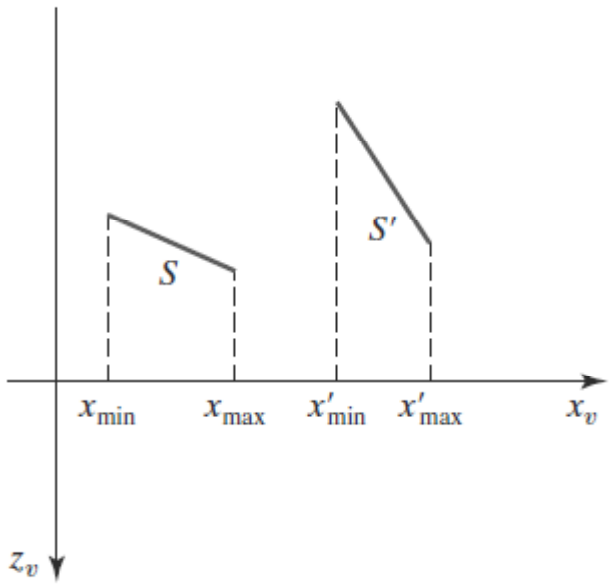


Figure 2: Two surfaces with depth overlap but no overlap in the x direction.

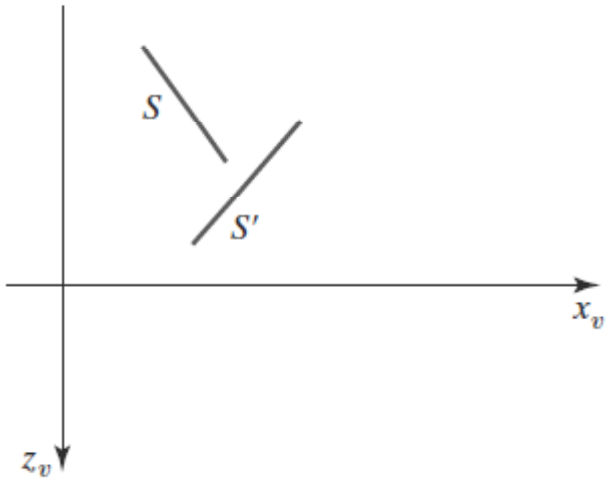


Figure 3: Surface S is completely behind the overlapping surface S'.

We make the following tests for each surface that has a depth overlap with S. If any one of these tests is true, no reordering is necessary for S and the surface being tested

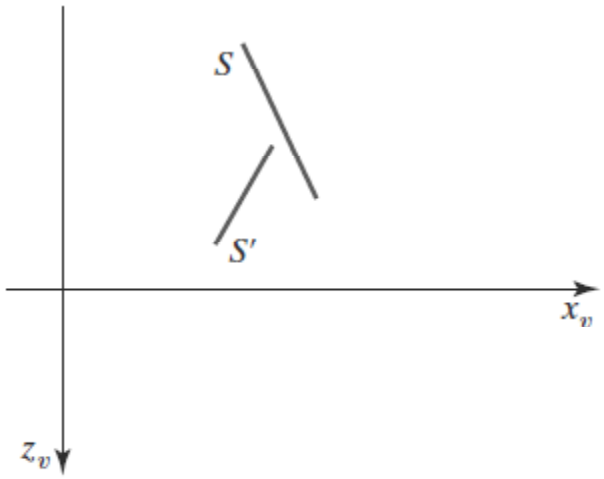


Figure 4: Overlapping surface S' is completely in front of surface S, but S is not completely behind S'.

The tests are listed in order of increasing difficulty:

1. The bounding rectangles (coordinate extents) in the xy directions for the two surfaces do not overlap.

2. Surface S is completely behind the overlapping surface relative to the viewing position.
3. The overlapping surface is completely in front of S relative to the viewing position.
4. The boundary-edge projections of the two surfaces onto the view plane do not overlap.

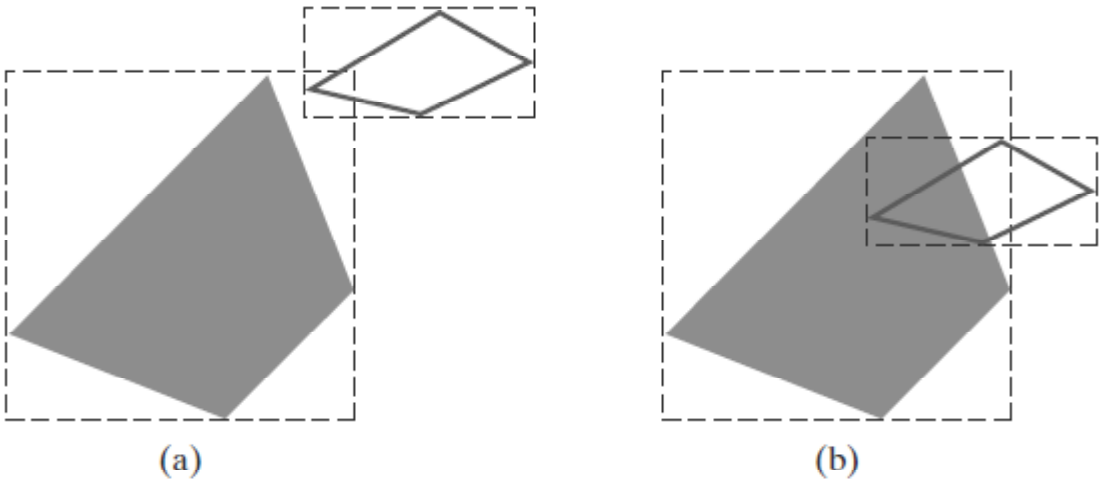


Figure 5: Two polygon surfaces with overlapping bounding rectangles in the x y plane.

We perform these tests in the order listed and proceed to the next overlapping surface as soon as we find that one of the tests is true. If all the overlapping surfaces pass at least one of these tests, then S is the most distant surface. No reordering is then necessary, therefore, and S is scan-converted.

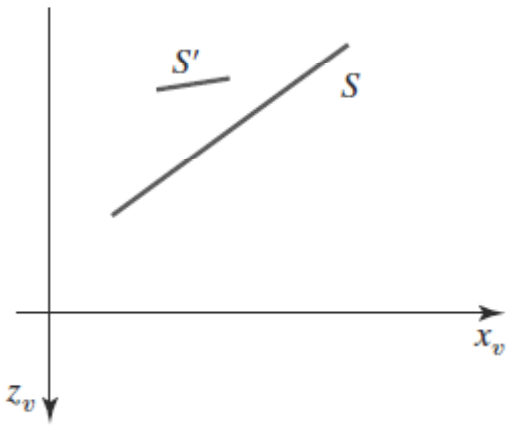


Figure 6: Surface S extends to a greater depth, but it obscures surface S'

Test 1 is performed in two parts. We check for overlap first in the x direction, then in the y direction. If there is no surface overlap in either of these directions, the two planes cannot obscure one other. An example of two surfaces that overlap in the z direction but not in the x direction is shown in Figure 2.

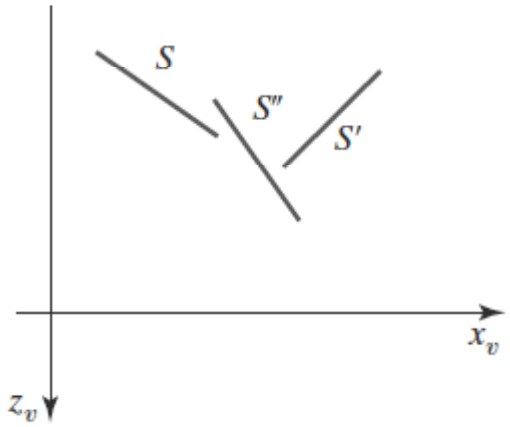


Figure 7: Three surfaces that have been entered into the sorted surface list in the order S, S', S'', should be reordered as S', S'',S

We can perform tests 2 and 3 using back-front polygon tests. That is, we substitute the coordinates for all vertices of S into the plane equation for the overlapping surface and check the sign of the result. If the plane equations are set up so that the front of the surface is toward the viewing position, then S is behind S' if all vertices of S are in back of S' (Figure 3). Similarly, S' is completely ahead of S if all vertices of S^1 are in front of S' . Figure 4 shows S' an overlapping surface S' that is completely in front of S, but surface S is not completely behind S' (test 2 is not true).

If tests 1 through 3 have all failed, we perform test 4 to determine whether the two surface projections overlap. As demonstrated in Figure 5, two surfaces may or may not intersect even though their coordinate extents overlap.

Should all four tests fail for an overlapping surface S', we interchange surfaces S and S' in the sorted list. An example of two surfaces that would be reordered with this procedure is given in Figure 6. At this point, we still do not know for certain that we have found the farthest surface from the view plane. Figure 7 illustrates a situation in which we would first interchange

S and S''. However, S'' obscures part of S', so we need to interchange S'' and S' to get the three surfaces into the correct depth order. Therefore, we need to repeat the testing process for each surface that is reordered in the list.

It is possible for the algorithm just outlined to get into an infinite loop if two or more surfaces alternately obscure each other. In such situations, the algorithm would continually rearrange the ordering of the overlapping surfaces. To avoid such loops, we can flag any surface that has been reordered to a farther depth position so that it cannot be moved again. If an attempt is made to switch the surface a second time, we divide it into two parts to eliminate the cyclic overlap. The original surface is then replaced by the two new surfaces, and we continue processing as before.

11.4 BSP-Tree Method

A binary space-partitioning (BSP) tree is an efficient method for determining object visibility by painting surfaces into the frame buffer from back to front, as in the painter's algorithm. The BSP tree is particularly useful when the view reference point changes, but the objects in a scene are at fixed positions.

Applying a BSP tree to visibility testing involves identifying surfaces that are behind or in front of the partitioning plane at each step of the space subdivision, relative to the viewing direction. Figure 8 illustrates the basic concept in this algorithm. With plane P_1, we first partition the space into two sets of objects. One set of objects is in back of plane P_1 relative to the viewing direction, and the other set is in front of P_1. Because one object is intersected by plane P_1, we divide that object into two separate objects, labeled A and B. Objects A and C are in front of P_1, and objects B and D are behind P_1. Because each object list contains more than one object, we partition the space again with plane P_2, recursively processing the front and back object list

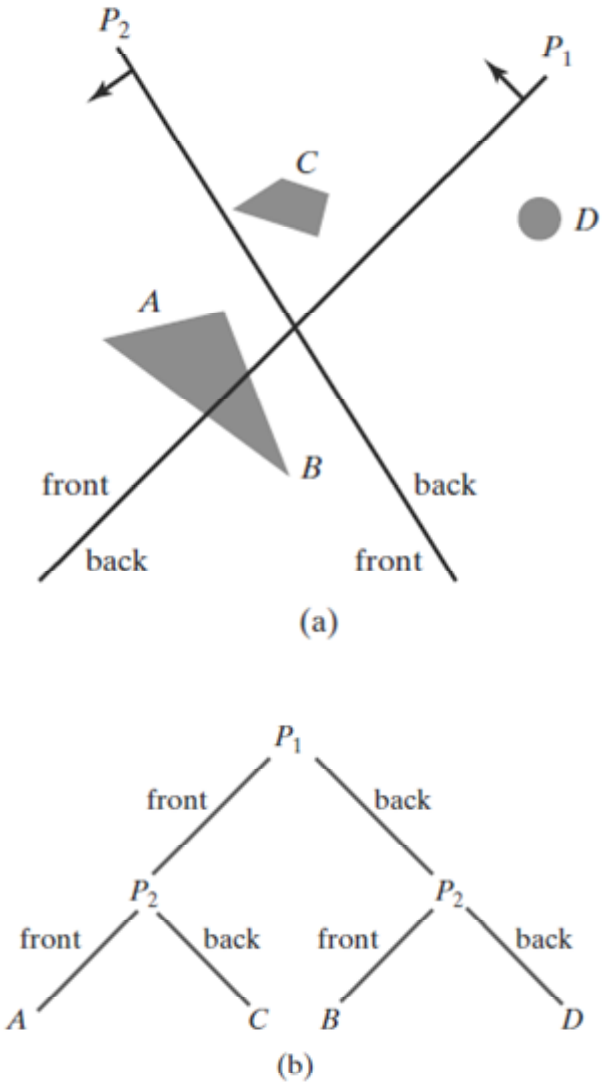


Figure 8: A region of space (a) is partitioned with two planes P_1 and P_2 to form the BSP tree representation shown in (b).

This process continues until all object lists contain no more than one object. This partitioning can be easily represented using a binary tree such as the one shown in Figure 8(b). In this tree, the objects are represented as terminal nodes, with front objects occupying the left branches and back objects occupying the right branches. The location of an object in the tree exactly represents its position relative to each of the partitioning planes.

For objects described with polygon facets, we often choose the partitioning planes to coincide with polygon-surface planes. The polygon equations are then used to identify back and front polygons, and the tree is constructed with one partitioning plane for each polygon face. Any polygon intersected by a partitioning plane is split into two parts.

When the BSP tree is complete, we interpret the tree relative to the position of our viewpoint, beginning at the root node. If the viewpoint is in front of that partitioning plane, we recursively process the back subtree, then recursively process the front subtree. If the viewpoint is behind the partitioning plane, we reverse this, and process the front subtree followed by the back subtree. Thus, the surfaces are generated for display in the order back to front, so that foreground objects are painted over the background objects. Fast hardware implementations for constructing and processing BSP trees are used in some systems.

11.5 Area-Subdivision Method

This technique for hidden-surface removal is essentially an image-space method, but object-space operations can be used to accomplish depth ordering of surfaces.

The area-subdivision method takes advantage of area coherence in a scene by locating those projection areas that represent part of a single surface. We apply this method by successively dividing the total view-plane area into smaller and smaller rectangles until each rectangular area contains the projection of part of a single visible surface, contains no surface projections, or the area has been reduced to the size of a pixel.

To implement this method, we need to establish tests that can quickly identify the area as part of a single surface or tell us that the area is too complex to analyze easily. Starting with the total view, we apply the tests to determine whether we should subdivide the total area into smaller rectangles. If the tests indicate that the view is sufficiently complex, we subdivide it. Next, we apply the tests to each of the smaller areas, subdividing these if the tests indicate that visibility of a single surface is still uncertain. We continue this process until the subdivisions are easily analyzed as belonging to a single surface or until we have reached the resolution limit. An easy way to do this is to successively divide the area into four equal parts at each step, as shown in Figure 9. This approach is similar to that used in constructing a quadtree. A viewing

area with a pixel resolution of 1024 × 1024 could be subdivided ten times in this way before a subarea is reduced to the size of a single pixel.

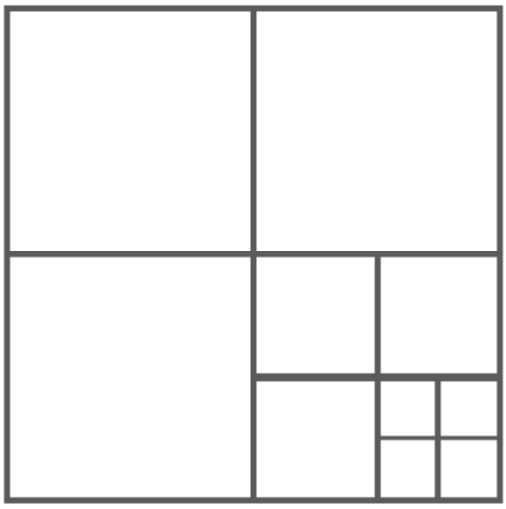


Figure 9: Dividing a square area into equal-sized quadrants at each step.

There are four possible relationships that a surface can have with an area of the subdivided view plane. We can describe these relative surface positions using the following classifications

Surrounding Surface: A surface that completely encloses the area.

Overlapping Surface: A surface that is partly inside and partly outside the area.

Inside Surface: A surface that is completely inside the area.

Outside Surface: A surface that is completely outside the area.

The tests for determining surface visibility within a rectangular area can be stated in terms of the four surface classifications illustrated in Figure 10.No further subdivisions of a specified area are needed if one of the following conditions is true.

Condition 1: An area has no inside, overlapping, or surrounding surfaces (all surfaces are outside the area).

Condition 2: An area has only one inside, overlapping, or surrounding surface.

Condition 3: An area has one surrounding surface that obscures all other surfaces within the area boundaries.

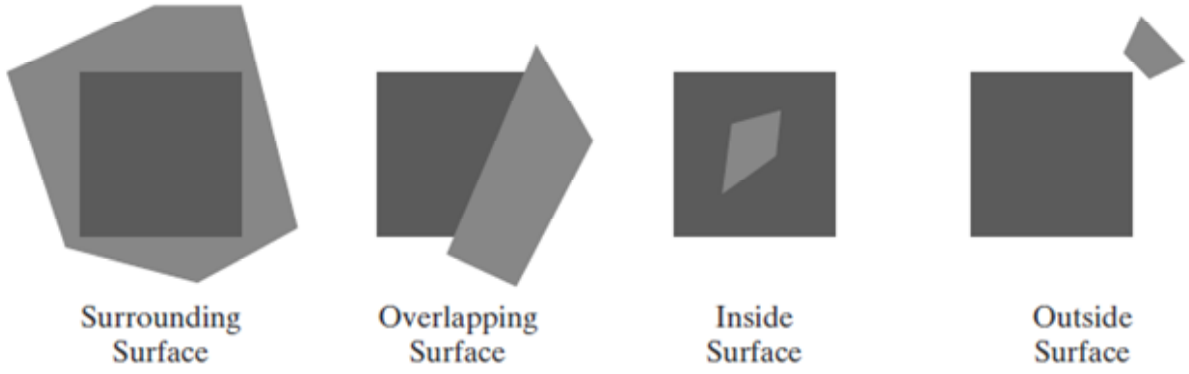


Figure 10: Possible relationships between polygon surfaces and a rectangular section of the viewing plane.

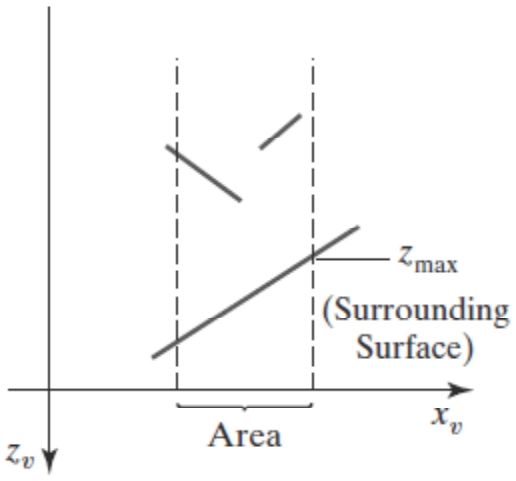


Figure 11: Within a specified area, a surrounding surface with a maximum depth of z_{max} obscures all surfaces that have a minimum depth beyond z_{max}.

Initially, we can compare the coordinate extents of each surface with the area boundaries. This will identify the inside and surrounding surfaces, but overlapping and outside surfaces usually require intersection tests. If a single bounding rectangle intersects the area in some way, additional checks are used to determine whether the surface is surrounding, overlapping, or outside. Once a single inside, overlapping, or surrounding surface has been identified, the surface color values are stored in the frame buffer.

One method for testing condition 3 is to sort the surfaces according to minimum depth from the view plane. For each surrounding surface, we then compute the maximum depth within the area under consideration. If the maximum depth of one of these surrounding surfaces is closer to the view plane than the minimum depth of all other surfaces within the area, condition 3 is satisfied. Figure 11 illustrates this situation.

Another method for testing condition 3 that does not require depth sorting is to use plane equations to calculate depth values at the four vertices of the area for all surrounding, overlapping, and inside surfaces. If all four depths for one of the surrounding surfaces are less than the calculated depths for all other surfaces, condition 3 is satisfied. Then the area can be displayed with the colors for that surrounding surface.

For some situations, the previous two testing methods may fail to identify correctly a surrounding surface that obscures all the other surfaces. Further testing could be carried out to identify the single surface that covers the area, but it is faster to subdivide the area than to continue with more complex testing. Once a surface has been identified as an outside or surrounding surface for an area, it will remain in that category for all subdivisions of the area. Furthermore, we can expect to eliminate some inside and overlapping surfaces as the subdivision process continues, so that the areas become easier to analyze. In the limiting case, when a subdivision the size of a pixel is produced, we simply calculate the depth of each relevant surface at that point and assign the color of the nearest surface to that pixel.

As a variation on the basic subdivision process, we could subdivide areas along surface boundaries instead of dividing them in half. If the surfaces have been sorted according to minimum depth, we can use the surface of smallest depth value to subdivide a given area. Figure 12 illustrates this method for subdividing areas. The projection of the boundary of surface S is used to partition the original area into the subdivisions A_1 and A_2. Surface S is then a surrounding surface for A_1, and visibility conditions 2 and 3 can be tested to determine whether further subdividing is necessary. In general, fewer subdivisions are required using this approach, but more processing is needed to subdivide areas and to analyze the relation of surfaces to the subdivision boundaries.

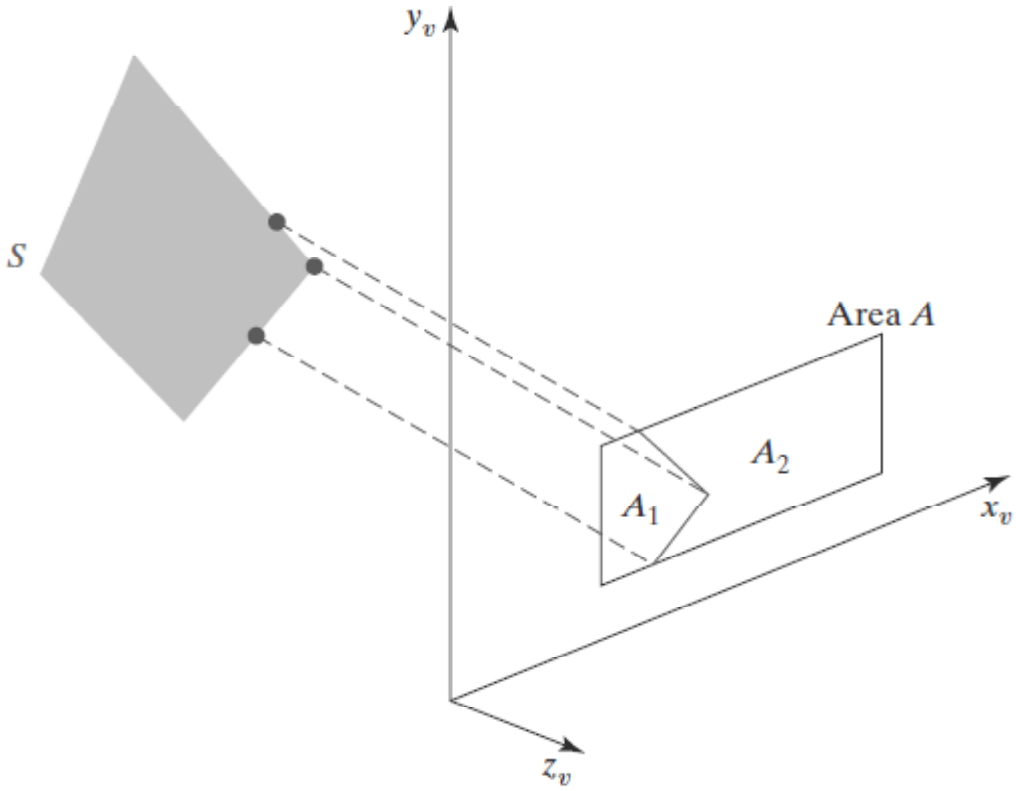


Figure 12: Area A is subdivided into A_1 and A_2 using the boundary of surface S on the view plane.

11.6 Octree Methods

When an octree representation is used for the viewing volume, visible-surface identification is accomplished by searching octree nodes in a front-to-back order. In Figure 13, the foreground of a scene is contained in octants 0, 1, 2, and 3. Surfaces in the front of these octants are visible to the viewer. Any surfaces toward the rear of the front octants or in the back octants (4, 5, 6, and 7) may be hidden by the front surfaces.

We can process the octree nodes of Figure 13 in the order 0, 1, 2, 3, 4, 5, 6, 7. This results in a depth-first traversal of the octree, where the nodes for the four front suboctants of octant 0 are visited before the nodes for the four back suboctants. The traversal of the octree continues in this order for each octant subdivision.

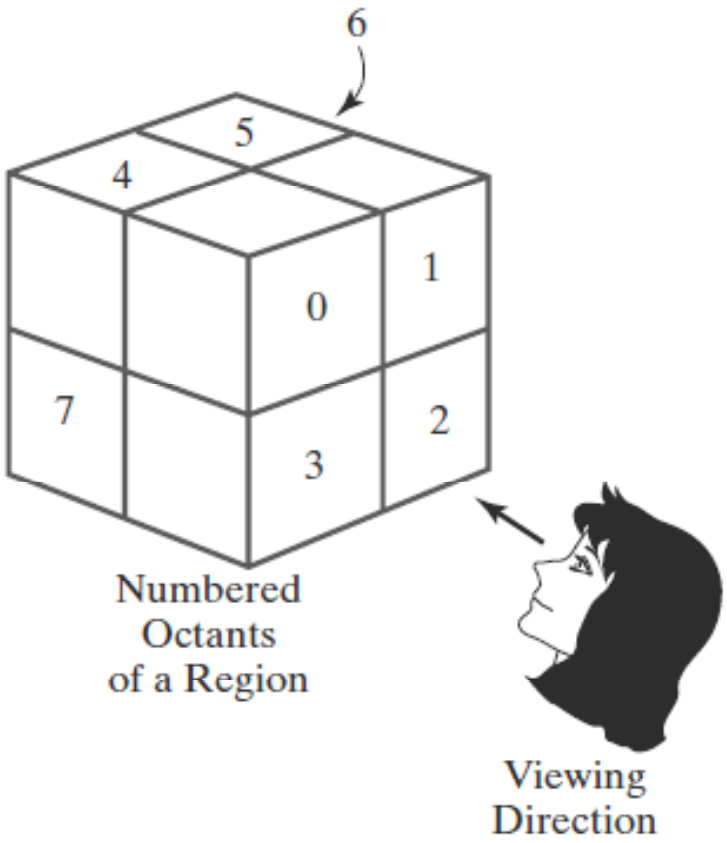
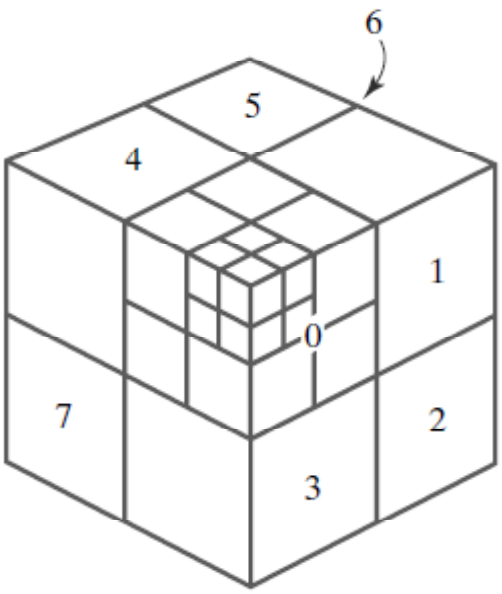
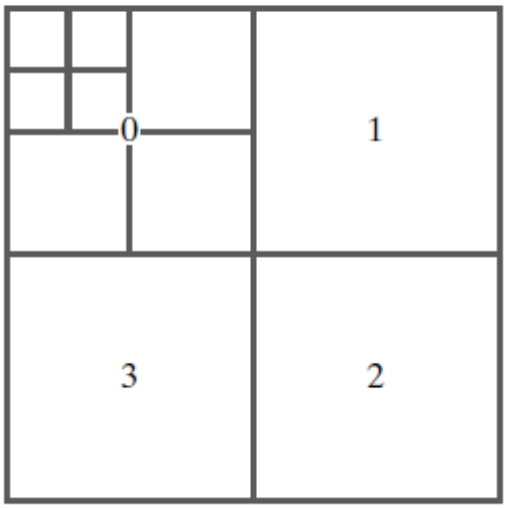


Figure 13: Objects in octants 0, 1, 2, and 3 obscure objects in the back octants (4, 5, 6, 7) when the viewing direction is as shown.

When a color value is encountered in an octree node, that color is saved in the quadtree only if no values have previously been saved for the same area. In this way, only the front colors are saved. Nodes that have the value “void” are ignored. Any node that is completely obscured is eliminated from further processing, so that its subtrees are not accessed. Figure 14 depicts the octants in a region of space and the corresponding quadrants on the view plane. Contributions to quadrant 0 come from octants 0 and 4. Color values in quadrant 1 are obtained from surfaces in octants 1 and 5, and values in each of the other two quadrants are generated from the pairs of octants aligned with each of these quadrants.



Octants in Space



Quadrants for
the View Plane

Figure 14: Octant divisions for a region of space and the corresponding quadrant plane.

Effective octree visibility testing is carried out with recursive processing of octree nodes and the creation of a quadtree representation for the visible surfaces. In most cases, both a

front and a back octant must be considered in determining the correct color values for a quadrant. But if the front octant is homogeneously filled with some color, we do not process the back octant. For heterogeneous regions, a recursive procedure is called, passing as new arguments the child of the heterogeneous octant and a newly created quadtree node. If the front is empty, it is necessary only to process the child of the rear octant. Otherwise, two recursive calls are made: one for the rear octant and one for the front octant.

Different views of objects represented as octrees can be obtained by applying transformations to the octree representation that reorient the object according to the view selected. Octants can then be renumbered so that the octree representation is always organized with octants 0, 1, 2, and 3 as the front face.

11.7 Other Methods

In the following section we will have a study on some popular visible surface detection methods.

Ray-Casting Method

If we consider the line of sight from a pixel position on the view plane through a scene, as in Figure 15, we can determine which objects in the scene (if any) intersect this line. After calculating all ray-surface intersections, we identify the visible surface as the one whose intersection point is closest to the pixel. This visibility-detection scheme uses ray casting procedures.

Ray casting, as a visibility-detection tool, is based on geometricoptics methods, which trace the paths of light rays. Because there are an infinite number of light rays in a scene and we are interested only in those rays that pass through pixel positions, we can trace the light-ray paths backward from the pixels through the scene. The ray-casting approach is an effective visibility-detection method for scenes with curved surfaces, particularly spheres.

We can think of ray casting as a variation on the depth-buffer method. In the depth-buffer algorithm, we process surfaces one at a time and calculate depth values for all projection points over the surface. The calculated surface depths are then compared to previously stored

depths to determine visible surfaces at each pixel. In ray casting, we process pixels one at a time and calculate depths for all surfaces along the projection path to that pixel.

Ray casting is a special case of ray-tracing algorithms that trace multiple raypaths to pick up global reflection and refraction contributions from multiple objects in a scene. With ray casting, we only follow a ray out from each pixel to the nearest object. Efficient ray-surface intersection calculations have been developed for common objects, particularly spheres.

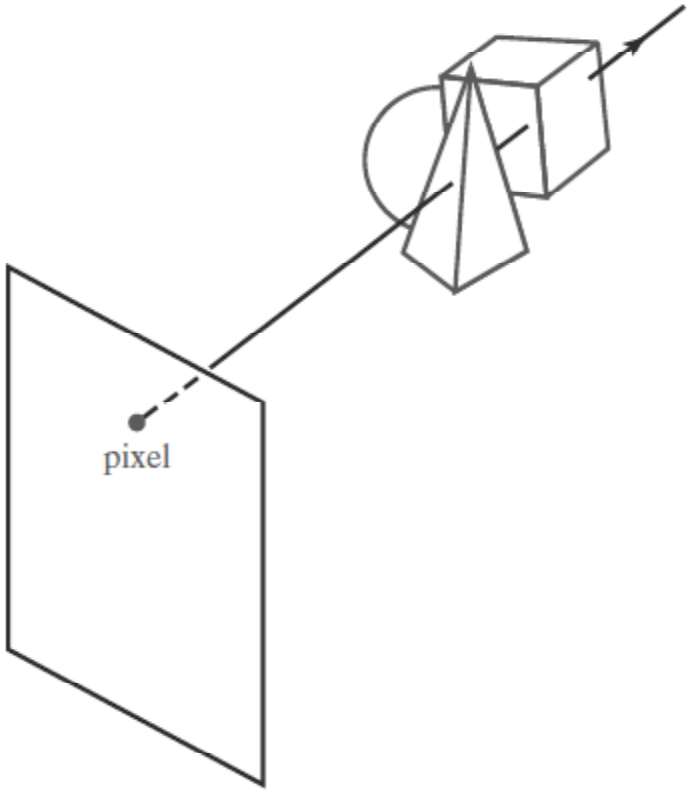


Figure 15: A ray along the line of sight from a pixel position through a scene.

Curved Surfaces

Effective methods for determining the visibility of objects with curved surfaces include ray casting and octree methods. With ray casting, we calculate ray-surface intersections and locate the smallest intersection distance along the pixel ray. With octrees, we simply search the nodes from front to back to locate the surface color values. Once an octree representation has been established from the input definition of the objects, all visible surfaces are identified with the

same processing procedures. No special considerations need be given to different kinds of surfaces, curved or otherwise.

A curved surface can also be approximated as a polygon mesh, and we can then use one of the visible-surface identification methods previously discussed. But for some objects, such as spheres, it could be more efficient as well as more accurate to use ray casting and the equations describing the curved surface.

Wire-Frame Visibility Methods

Scenes usually do not contain isolated line sections, unless we are displaying a graph, diagram, or network layout. But often we want to view a three-dimensional scene in an outline form to obtain a quick display of the object features. The fastest way to generate a wire-frame view of a scene is to display all object edges.

However, it may be difficult to determine the front and back features of the objects in such a display. One solution to this problem is to apply depth cueing, so that the displayed intensity of a line is a function of its distance from the viewer. Alternatively, we can apply visibility tests, so that hidden line sections can be either eliminated or displayed differently from the visible edges. Procedures for determining visibility of object edges are referred to as wire-frame visibility methods. They are also called visible-line detection methods or hidden-line detection methods. In addition, some of the visible-surface methods discussed in preceding sections can be used to test for edge visibility.

Wire-Frame Surface-Visibility Algorithms

A direct approach to identifying visible line sections is to compare edge positions with the positions of the surfaces in a scene. This process involves the same methods used in line-clipping algorithms. That is, we test the position of line endpoints with respect to the boundaries of a specified area, but, for visibility testing, we also need to compare edge and surface depth values. When the projected edge endpoints of a line segment are both within the projected area of a surface, we compare the depth of the endpoints to the surface depth at those (x, y) positions. If both endpoints are behind the surface, we have a hidden edge. If both endpoints are in front of the surface, the edge is visible with respect to that surface. Otherwise, we must calculate

intersection positions and determine the depth values at those intersection points. If the edge has greater depth than the surface at the perimeter intersections, part of the edge is hidden by the surface. Another possibility is that an edge has greater depth at one boundary intersection and less depth than the surface at the other boundary intersection. In that case, we need to determine where the edge penetrates the surface interior. Once we have identified a hidden section of an edge, we could eliminate it, display it as a dashed line, or display it in some other way to distinguish it from the visible sections.

Some of the visible-surface detection methods are readily adapted to wireframe visibility testing of object edges. Using a back-face method, we could identify all the back surfaces of an object and display only the boundaries for the visible surfaces. With depth sorting, surfaces can be painted into the refresh buffer so that surface interiors are in the background color while boundaries are in the foreground color. By processing the surfaces from back to front, hidden lines are erased by the nearer surfaces. An area-subdivision method can be adapted to hidden-line removal by displaying only the boundaries of visible surfaces. And scan-line methods can be used to display the scan-line intersection positions at the boundaries of visible surfaces.

Wire-Frame Depth-Cueing Algorithm

Another method for displaying visibility information is to vary the brightness of objects in a scene as a function of distance from the viewing position. This depth-cueing method is typically applied using the linear function

$$f_{\text{depth}}(d) = \frac{d_{\max} - d}{d_{\max} - d_{\min}}$$

where d is the distance of a point from the viewing position. Values for minimum and maximum depth, d_{min} and d_{max}, can be set to convenient values for a particular application, or the minimum and maximum depths can be set to the normalization depth range: d_{min} =0.0 and d_{max} =1.0. As each pixel position is processed, its color is multiplied by $f_{depth}(d)$. Thus, nearer points are displayed with higher intensities, and the points at the maximum depth have an intensity equal to 0. The depth-cueing function can be implemented with various options. In some graphics libraries, a general atmosphere function is available, which can combine depth

cueing with atmospheric effects to simulate smoke or haze, for example. Thus, an objects color could be modified by the depth-cueing function and then combined with the atmosphere color.

11.8 Summary

With the depth sorting method (painter's algorithm), objects are "painted" into the refresh buffer according to their distances from the viewing position.

Subdivision schemes for identifying visible parts of a scene include the BSP-tree method, area subdivision, and octree representations.

Visible surfaces can also be detected using ray casting methods, which project lines from the pixel plane into a scene to determine object intersection positions along these projected lines.

The area-subdivision method takes advantage of area coherence in a scene by locating those projection areas that represent part of a single surface.

Effective methods for determining the visibility of objects with curved surfaces include ray casting and octree methods.

Ray casting, as a visibility-detection tool, is based on geometricoptics methods, which trace the paths of light rays.

Ray-casting methods are an integral part of ray-tracing algorithms, which allow scenes to be displayed with global-illumination effects.

11.9 Model Question

1. Illustrate Depth-Sorting Method in hidden surface elimination.
2. What is hidden surface elimination?
3. Explain BSP-Tree Method in detail.
4. What is Area-Subdivision Method?
5. Explain Octree Methods and its significance in visible surface detection.
6. Explain Ray-Casting Method.

LESSON – 12
COMPUTER ANIMATION

Structure of the lesson

12.1 Introduction

Computer-graphics methods are now commonly used to produce animations for a variety of applications, including entertainment (motion pictures and cartoons), advertising, scientific and engineering studies, and training and education. Although we tend to think of animation as implying object motion, the term computer animation generally refers to any time sequence of visual changes in a picture. In addition to changing object positions using translations or rotations, a computer-generated animation could display time variations in object size, color, transparency, or surface texture. Advertising animations often transition one object shape into another: for example, transforming a can of motor oil into an automobile engine. We can also generate

computer animations by varying camera parameters, such as position, orientation, or focal length, and variations in lighting effects or other parameters and procedures associated with illumination and rendering can be used to produce computer animations.

Another consideration in computer-generated animation is realism. Many applications require realistic displays. An accurate representation of the shape of a thunderstorm or other natural phenomena described with a numerical model is important for evaluating the reliability of the model. Similarly, simulators for training aircraft pilots and heavy-equipment operators must produce reasonably accurate representations of the environment. Entertainment and advertising applications, on the other hand, are sometimes more interested in visual effects. Thus, scenes may be displayed with exaggerated shapes and unrealistic motions and transformations.

However, there are many entertainment and advertising applications that do require accurate representations for computer-generated scenes. Also, in some scientific and engineering studies, realism is not a goal. For example, physical quantities are often displayed with pseudo-colors or abstract shapes that change over time to help the researcher understand the nature of the physical process.

Two basic methods for constructing a motion sequence are real-time animation and frame-by-frame animation. In a real-time computer-animation, each stage of the sequence is viewed as it is created. Thus the animation must be generated at a rate that is compatible with the constraints of the refresh rate. For a frame-by-frame animation, each frame of the motion is separately generated and stored. Later, the frames can be recorded on film, or they can be displayed consecutively on a video monitor in “real-time playback” mode. Simple animation displays are generally produced in real time, while more complex animations are constructed more slowly, frame by frame. However, some applications require real-time animation, regardless of the complexity of the animation.

A flight-simulator animation, for example, is produced in real time because the video displays must be generated in immediate response to changes in the control settings. In such cases, special hardware and software systems are often developed to allow the complex display sequences to be developed quickly.

12.2 Objective of this Lesson

From this lesson the learner will know about animation and its need. They will also learn Animation Techniques, Computer-Animation Functions, Computer-Animation Languages, Motion Specifications and Character Animation.

12.3 Raster Methods for Computer Animation

Most of the time, we can create simple animation sequences in our programs using real-time methods. In general, though, we can produce an animation sequence on a raster-scan system one frame at a time, so that each completed frame could be saved in a file for later viewing. The animation can then be viewed by cycling through the completed frame sequence, or the frames could be transferred to film. If we want to generate an animation in real time, however, we need to produce the motion frames quickly enough so that a continuous motion sequence is displayed.

For a complex scene, one frame of the animation could take most of the refresh cycle time to construct. In that case, objects generated first would be displayed for most of the frame refresh time, but objects generated toward the end of the refresh cycle would disappear almost as soon as they were displayed. For very complex animations, the frame construction time could be greater than the time to refresh the screen, which can lead to erratic motion and fractured frame displays.

Because the screen display is generated from successively modified pixel values in the refresh buffer, we can take advantage of some of the characteristics of the raster screen-refresh process to produce motion sequences quickly.

Double Buffering

One method for producing a real-time animation with a raster system is to employ two refresh buffers. Initially, we create a frame for the animation in one of the buffers. Then, while the screen is being refreshed from that buffer, we construct the next frame in the other buffer. When that frame is complete, we switch the roles of the two buffers so that the refresh routines use the second buffer during the process of creating the next frame in the first buffer. This alternating buffer process continues throughout the animation. Graphics libraries that permit

such operations typically have one function for activating the double buffering routines and another function for interchanging the roles of the two buffers.

When a call is made to switch two refresh buffers, the interchange could be performed at various times. The most straight forward implementation is to switch the two buffers at the end of the current refresh cycle, during the vertical retrace of the electron beam. If a program can complete the construction of a frame within the time of a refresh cycle, say $\frac{1}{60}$ of a second, each motion sequence is displayed in synchronization with the screen refresh rate. However, if the time to construct a frame is longer than the refresh time, the current frame is displayed for two or more refresh cycles while the next animation frame is being generated. For example, if the screen refresh rate is 60 frames per second and it takes $\frac{1}{50}$ of a second to construct an animation frame, each frame is displayed on the screen twice and the animation rate is only 30 frames each second. Similarly, if the frame construction time is $\frac{1}{25}$ of a second, the animation frame rate is reduced to 20 frames per second because each frame is displayed three times.

Irregular animation frame rates can occur with double buffering when the frame construction time is very nearly equal to an integer multiple of the screen refresh time. As an example of this, if the screen refresh rate is 60 frames per second, then an erratic animation frame rate is possible when the frame construction time is very close to $\frac{1}{60}$ of a second, or $\frac{2}{60}$, or $\frac{3}{60}$ of a second, and so forth. Because of slight variations in the implementation time for the routines that generate the primitives and their attributes, some frames could take a little more time to construct and some a little less time. Thus, the animation frame rate can change abruptly and erratically. One way to compensate for this effect is to add a small time delay to the program. Another possibility is to alter the motion or scene description to shorten the frame construction time.

Generating Animations Using Raster Operations

We can also generate real-time raster animations for limited applications using block transfers of a rectangular array of pixel values. This animation technique is often used in game-playing programs. A simple method for translating an object from one location to another in the xy plane is to transfer the group of pixel values that define the shape of the object to the new location. Two-dimensional rotations in multiples of 90º are also simple to perform, although we

can rotate rectangular blocks of pixels through other angles using antialiasing procedures. For a rotation that is not a multiple of 90º, we need to estimate the percentage of area coverage for those pixels that overlap the rotated block. Sequences of raster operations can be executed to produce realtime animation for either two-dimensional or three-dimensional objects, so long as we restrict the animation to motions in the projection plane. Then no viewing or visible-surface algorithms need be invoked.

We can also animate objects along two-dimensional motion paths using color table transformations. Here we predefine the object at successive positions along the motion path and set the successive blocks of pixel values to color-table entries. The pixels at the first position of the object are set to a foreground color, and the pixels at the other object positions are set to the background color. The animation is then accomplished by changing the color-table values so that the object color at successive positions along the animation path becomes the foreground color as the preceding position is set to the background color.

12.4 Design of Animation Sequences

Constructing an animation sequence can be a complicated task, particularly when it involves a story line and multiple objects, each of which can move in a different way. A basic approach is to design such animation sequences using the following development stages:

- Storyboard layout
- Object definitions
- Key-frame specifications
- Generation of in-between frames

Storyboard

The storyboard is an outline of the action. It defines the motion sequence as a set of basic events that are to take place. Depending on the type of animation to be produced, the storyboard could consist of a set of rough sketches, along with a brief description of the movements, or it could just be a list of the basic ideas for the action. Originally, the set of motion sketches was attached to a large board that was used to present an overall view of the animation project. Hence, the name “storyboard.”

Object

An object definition is given for each participant in the action. Objects can be defined in terms of basic shapes, such as polygons or spline surfaces. In addition, a description is often given of the movements that are to be performed by each character or object in the story.

Key Frame

A key frame is a detailed drawing of the scene at a certain time in the animation sequence. Within each key frame, each object (or character) is positioned according to the time for that frame. Some key frames are chosen at extreme positions in the action; others are spaced so that the time interval between key frames is not too great. More key frames are specified for intricate motions than for simple, slowly varying motions. Development of the key frames is generally the responsibility of the senior animators, and often a separate animator is assigned to each character in the animation.

In-Between

In-betweens are the intermediate frames between the key frames. The total number of frames, and hence the total number of in-betweens, needed for an animation is determined by the display media that is to be used. Film requires 24 frames per second, and graphics terminals are refreshed at the rate of 60 or more frames per second. Typically, time intervals for the motion are set up so that there are from three to five in-betweens for each pair of key frames. Depending on the speed specified for the motion, some key frames could be duplicated. As an example, a 1-minute film sequence with no duplication requires a total of 1,440 frames. If five in-betweens are required for each pair of key frames, then 288 key frames would need to be developed.

There are several other tasks that may be required, depending on the application. These additional tasks include motion verification, editing, and the production and synchronization of a soundtrack. Many of the functions needed to produce general animations are now computer-generated.

12.5 Traditional Animation Techniques

Film animators use a variety of methods for depicting and emphasizing motion sequences. These include object deformations, spacing between animation frames, motion anticipation and follow-through, and action focusing.

One of the most important techniques for simulating acceleration effects, particularly for nonrigid objects, is squash and stretch. Figure 1 shows how this technique is used to emphasize the acceleration and deceleration of a bouncing ball. As the ball accelerates, it begins to stretch. When the ball hits the floor and stops, it is first compressed (squashed) and then stretched again as it accelerates and bounces upwards.

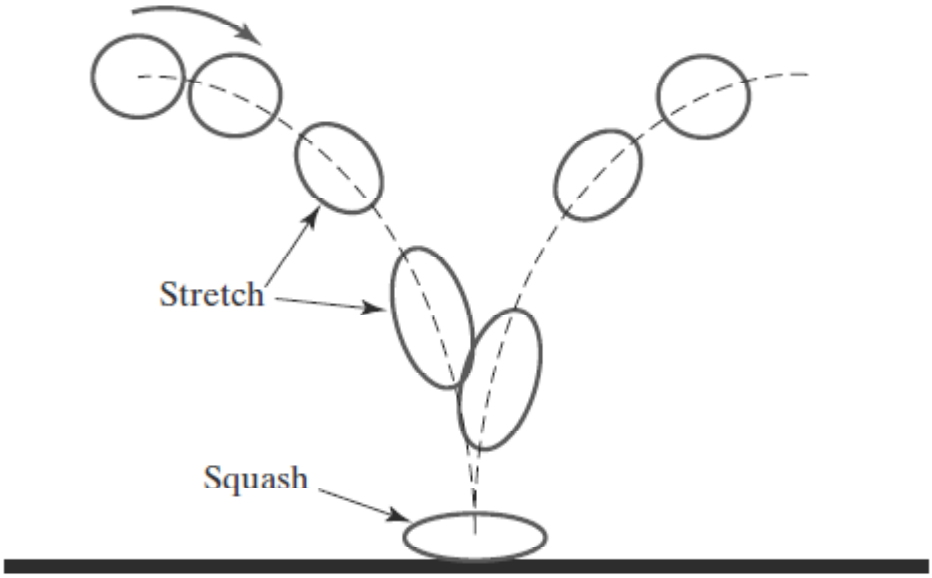


Figure 1: A bouncing-ball illustration of the "squash and stretch" technique for emphasizing object acceleration.

Another technique used by film animators is timing, which refers to the spacing between motion frames. A slower moving object is represented with more closely spaced frames, and a faster moving object is displayed with fewer frames over the path of the motion. This effect is illustrated in Figure 2, where the position changes between frames increase as a bouncing ball moves faster.

Object movements can also be emphasized by creating preliminary actions that indicate an anticipation of a coming motion. For example, a cartoon character might lean forward and rotate its body before starting to run; or a character might perform a “windup” before throwing a ball. Similarly, follow-through actions can be used to emphasize a previous motion. After throwing a ball, a character can continue the arm swing back to its body; or a hat can fly off a character that is stopped abruptly. An action also can be emphasized with staging, which refers to any method for focusing on an important part of a scene, such as a character hiding something.

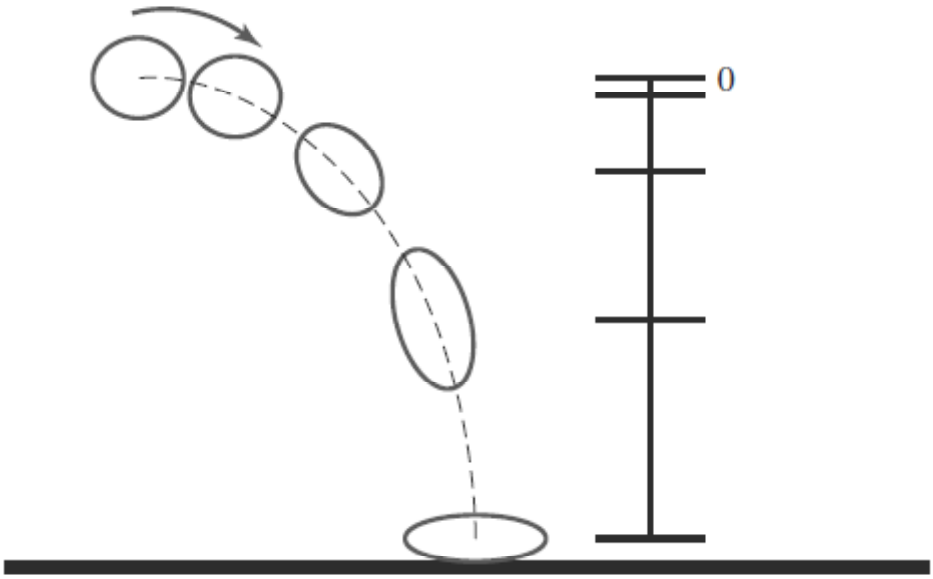


Figure 2: The position changes between motion frames for a bouncing ball increase as the speed of the ball increases.

12.6 General Computer-Animation Functions

Many software packages have been developed either for general animation design or for performing specialized animation tasks. Typical animation functions include managing object motions, generating views of objects, producing camera motions, and the generation of in-between frames. Some animation packages, such as Wavefront for example, provide special functions for both the overall animation design and the processing of individual objects. Others are special-purpose packages for particular features of an animation, such as a system for generating in-between frames or a system for figure animation.

A set of routines is often provided in a general animation package for storing and managing the object database. Object shapes and associated parameters are stored and updated in the database. Other object functions include those for generating the object motion and those for rendering the object surfaces. Movements can be generated according to specified constraints using two-dimensional or three-dimensional transformations. Standard functions can then be applied to identify visible surfaces and apply the rendering algorithms.

Another typical function set simulates camera movements. Standard camera motions are zooming, panning, and tilting. Finally, given the specification for the key frames, the in-betweens can be generated automatically.

12.7 Computer-Animation Languages

We can develop routines to design and control animation sequences within a general-purpose programming language, such as C, C++, Lisp, or Fortran, but several specialized animation languages have been developed. These languages typically include a graphics editor, a key-frame generator, an in-between generator, and standard graphics routines. The graphics editor allows an animator to design and modify object shapes, using spline surfaces, constructive solid geometry methods, or other representation schemes.

An important task in an animation specification is scene description. This includes the positioning of objects and light sources, defining the photometric parameters (light-source intensities and surface illumination properties), and setting the camera parameters (position, orientation, and lens characteristics). Another standard function is action specification, which involves the layout of motion paths for the objects and camera. We need the usual graphics routines: viewing and perspective transformations, geometric transformations to generate object movements as a function of accelerations or kinematic path specifications, visible-surface identification, and the surface-rendering operations.

Key-frame systems were originally designed as a separate set of animation routines for generating the in-betweens from the user-specified key frames. Now, these routines are often a component in a more general animation package. In the simplest case, each object in a scene is defined as a set of rigid bodies connected at the joints and with a limited number of

degrees of freedom. As an example, the single-armed robot in Figure 3 has 6 degrees of freedom, which are referred to as arm sweep, shoulder swivel, elbow extension, pitch, yaw, and roll. We can extend the number of degrees of freedom for this robot arm to 9 by allowing three-dimensional translations for the base (Figure 3). If we also allow base rotations, the robot arm can have a total of 12 degrees of freedom. The human body, in comparison, has more than 200 degrees of freedom.

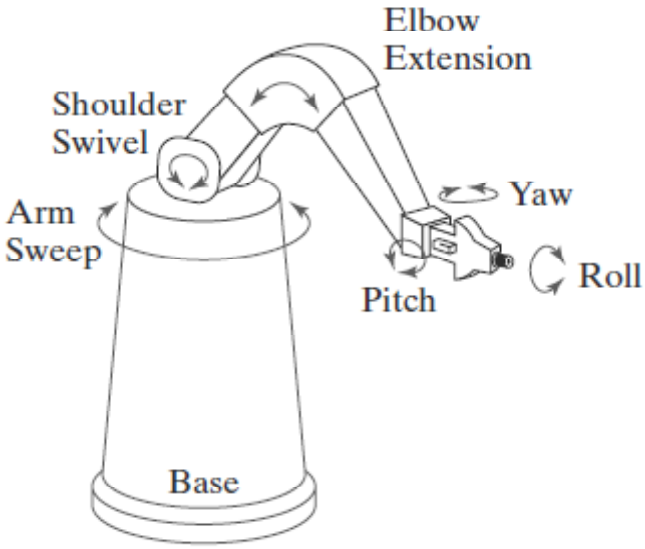


Figure 3: Degrees of freedom for a stationary, single-armed robot.

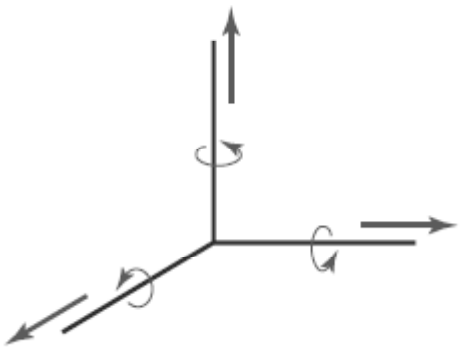


Figure 4: Translational and rotational degrees of freedom for the base of the robot arm.

Parameterized systems

Parameterized systems allow object motion characteristics to be specified as part of the object definitions. The adjustable parameters control such object characteristics as degrees of freedom, motion limitations, and allowable shape changes.

Scripting Systems

Scripting systems allow object specifications and animation sequences to be defined with a user-input script. From the script, a library of various objects and motions can be constructed.

12.8 Key-Frame Systems

A set of in-betweens can be generated from the specification of two (or more) key frames using a key-frame system. Motion paths can be given with a kinematic description as a set of spline curves, or the motions can be physically based by specifying the forces acting on the objects to be animated.

For complex scenes, we can separate the frames into individual components or objects called cels (celluloid transparencies). This term developed from cartoon animation techniques where the background and each character in a scene were placed on a separate transparency. Then, with the transparencies stacked in the order from background to foreground, they were photographed to obtain the completed frame. The specified animation paths are then used to obtain the next cel for each character, where the positions are interpolated from the key-frame times.

With complex object transformations, the shapes of objects may change over time. Examples are clothes, facial features, magnified detail, evolving shapes, and exploding or disintegrating objects. For surfaces described with polygon meshes, these changes can result in significant changes in polygon shape such that the number of edges in a polygon could be different from one frame to the next. These changes are incorporated into the development of the in-between frames by adding or subtracting polygon edges according to the requirements of the defining key frames.

Morphing

Transformation of object shapes from one form to another is termed morphing, which is a shortened form of “metamorphosing.” An animator can model morphing by transitioning polygon shapes through the in-betweens from one key frame to the next.

Given two key frames, each with a different number of line segments specifying an object transformation, we can first adjust the object specification in one of the frames so that the number of polygon edges (or the number of polygon vertices) is the same for the two frames. This preprocessing step is illustrated in Figure 5. A straight-line segment in key frame k is transformed into two line segments in key frame k +1. Because key frame k +1 has an extra vertex, we add a vertex between vertices 1 and 2 in key frame k to balance the number of vertices (and edges) in the two key frames.

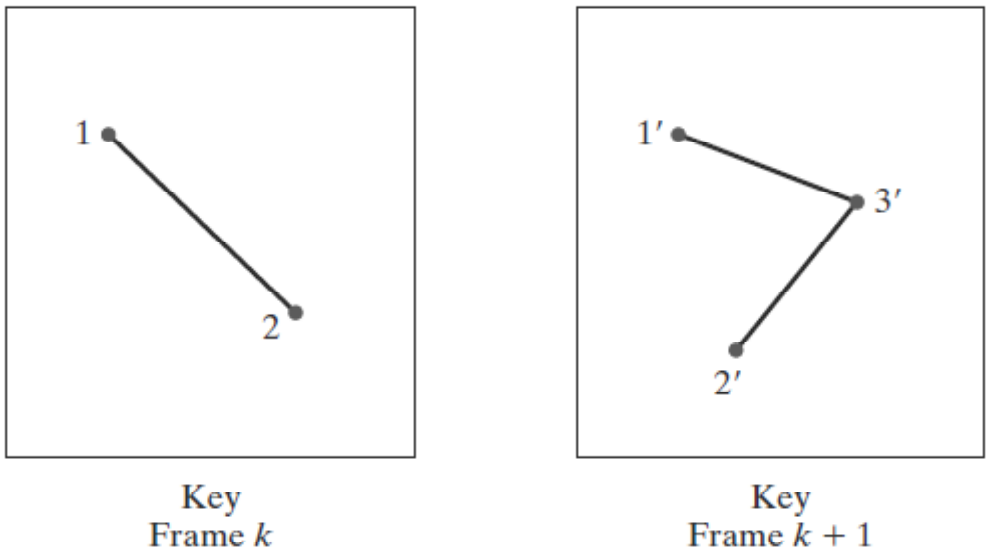


Figure 5: An edge with vertex positions 1 and 2 in key frame k evolves into two connected edges in key frame k + 1.

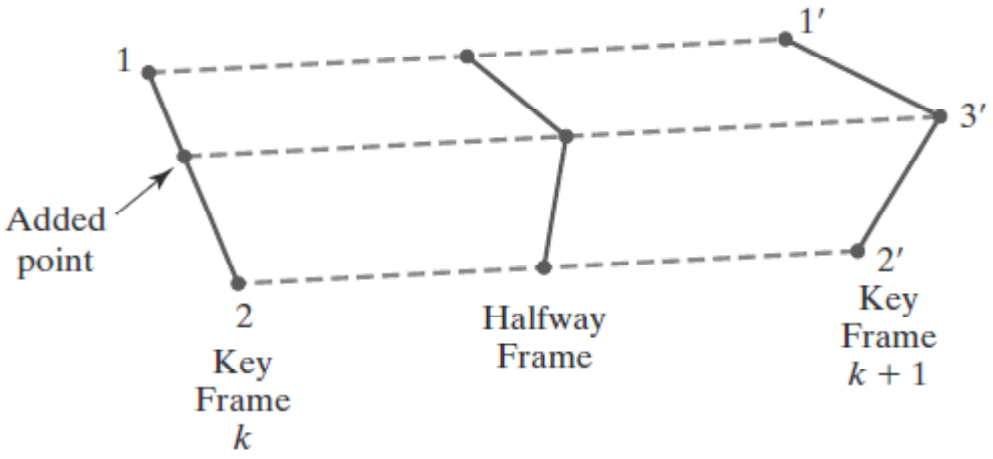


Figure 6: Linear interpolation for transforming a line segment in key frame k into two connected line segments in key frame k + 1.

Using linear interpolation to generate the in-betweens, we transition the added vertex in key frame k into vertex 3' along the straight-line path shown in Figure 6. An example of a triangle linearly expanding into a quadrilateral is given in Figure 7.

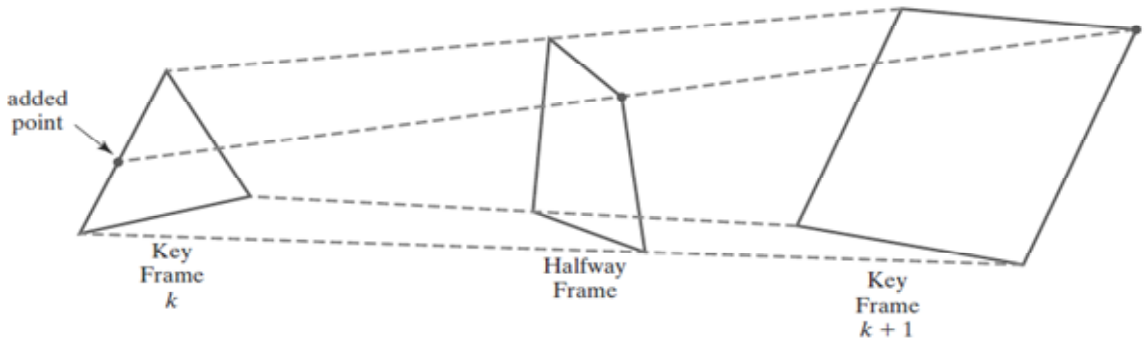


Figure 7: Linear interpolation for transforming a triangle into a quadrilateral.

We can state general preprocessing rules for equalizing key frames in terms of either the number of edges or the number of vertices to be added to a key frame. We first consider equalizing the edge count, where parameters L_k and L_{k+1} denote the number of line segments in two consecutive frames. The maximum and minimum number of lines to be equalized can be determined as

$$L_{\max} = \max(L_k, L_{k+1}), \qquad L_{\min} = \min(L_k, L_{k+1})$$

Next we compute the following two quantities:

$$N_e = L_{\max} \bmod L_{\min}$$
$$N_s = int\left(\frac{L_{\max}}{L_{\min}}\right)$$

The preprocessing steps for edge equalization are then accomplished with the following two procedures:

1. Divide N_e edges of $keyframe_{min}$ into N_s + 1 sections.

2. Divide the remaining lines of $keyframe_{min}$ into N_s sections.

As an example, if L_k = 15 and L_k+1 = 11,we would divide four lines of $keyframe_{k+1}$ into two sections each. The remaining lines of $keyframe_{k+1}$ are left intact. If we equalize the vertex count, we can use parameters V_k and V_{k+1} to denote the number of vertices in the two consecutive key frames. In this case, we determine the maximum and minimum number of vertices as

$$V_{\max} = \max(V_k, V_{k+1}), \qquad V_{\min} = \min(V_k, V_{k+1})$$

Then we compute the following two values:

$$N_{ls} = (V_{\max} - 1) \bmod (V_{\min} - 1)$$
$$N_p = int\left(\frac{V_{\max} - 1}{V_{\min} - 1}\right)$$

These two quantities are then used to perform vertex equalization with the following procedures:

1. Add N_p points to N_{ls} line sections of $keyframe_{min}$.

2. Add N_p - 1 points to the remaining edges of $keyframe_{min}$.

For the triangle-to-quadrilateral example, $V_k = 3$ and $V_k+1 = 4$. Both N_{ls} and N_p are 1, so we would add one point to one edge of $keyframe_k$. No points would be added to the remaining lines of $keyframe_k$.

Simulating Accelerations

Curve-fitting techniques are often used to specify the animation paths between key frames. Given the vertex positions at the key frames, we can fit the positions with linear or nonlinear paths. Figure 8 illustrates a nonlinear fit of keyframe positions. To simulate accelerations, we can adjust the time spacing for the in-betweens. If the motion is to occur at constant speed (zero acceleration), we use equal interval time spacing for the in-betweens. For instance, with n in-betweens and

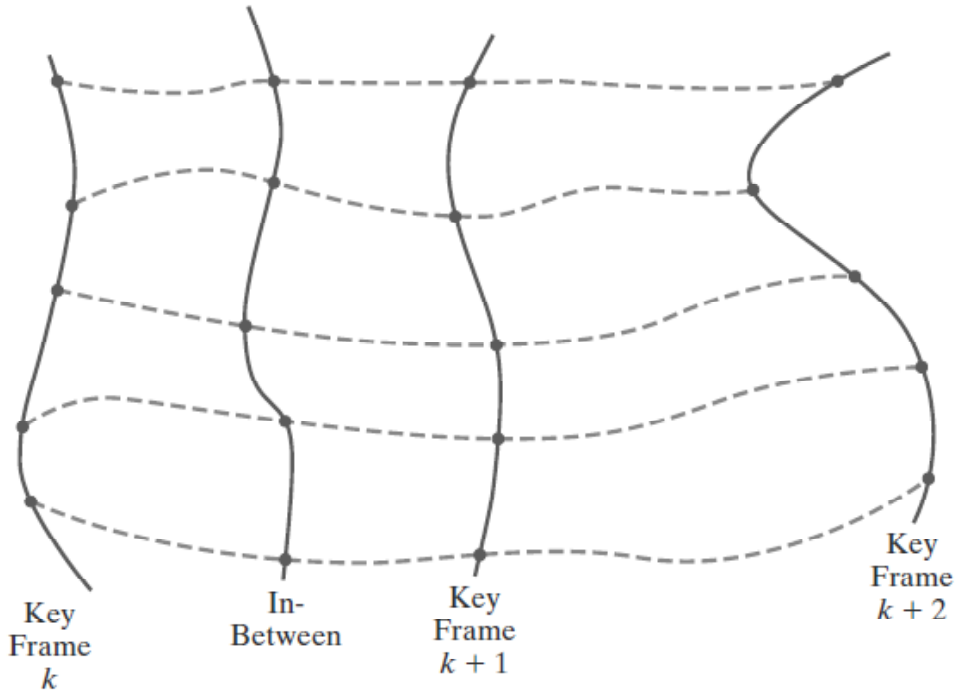


Figure 8: Fitting key-frame vertex positions with nonlinear splines.

key-frame times of t_1 and t_2 (Figure 9), the time interval between the key frames is divided into n+1 equal subintervals, yielding an in-between spacing of

$$\Delta t = \frac{t_2 - t_1}{n+1}$$

The time for the jth in-between is

$$tB_j = t_1 + j\Delta t, \qquad j = 1, 2, \ldots, n$$

and this time value is used to calculate coordinate positions, color, and other physical parameters for that frame of the motion.

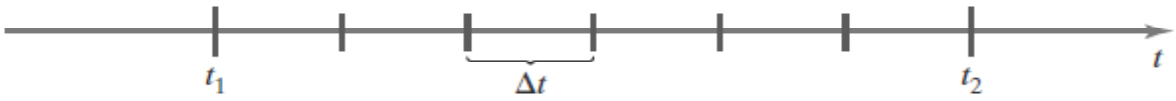


Figure 9: In-between positions for motion at constant speed.

Speed changes (nonzero accelerations) are usually necessary at some point in an animation film or cartoon, particularly at the beginning and end of a motion sequence. The startup and slowdown portions of an animation path are often modeled with spline or trigonometric functions, but parabolic and cubic time functions have been applied to acceleration modeling. Animation packages commonly furnish trigonometric functions for simulating accelerations.

To model increasing speed (positive acceleration), we want the time spacing between frames to increase so that greater changes in position occur as the object moves faster. We can obtain an increasing size for the time interval with the function

$$1 - \cos\theta, \qquad 0 < \theta < \pi/2$$

For n in-betweens, the time for the jth in-between would then be calculated as

$$tB_j = t_1 + \Delta t \left[1 - \cos\frac{j\pi}{2(n+1)}\right], \qquad j = 1, 2, \ldots, n$$

where Δt is the time difference between the two key frames. Figure 10 gives a plot of the trigonometric acceleration function and the in-between spacing for n = 5.

We can model decreasing speed (deceleration) using the function $\sin\theta$, with $0 < \theta < \pi/2$. The time position of an in-between is then determined as

$$tB_j = t_1 + \Delta t \sin\frac{j\pi}{2(n+1)}, \qquad j = 1, 2, \ldots, n$$

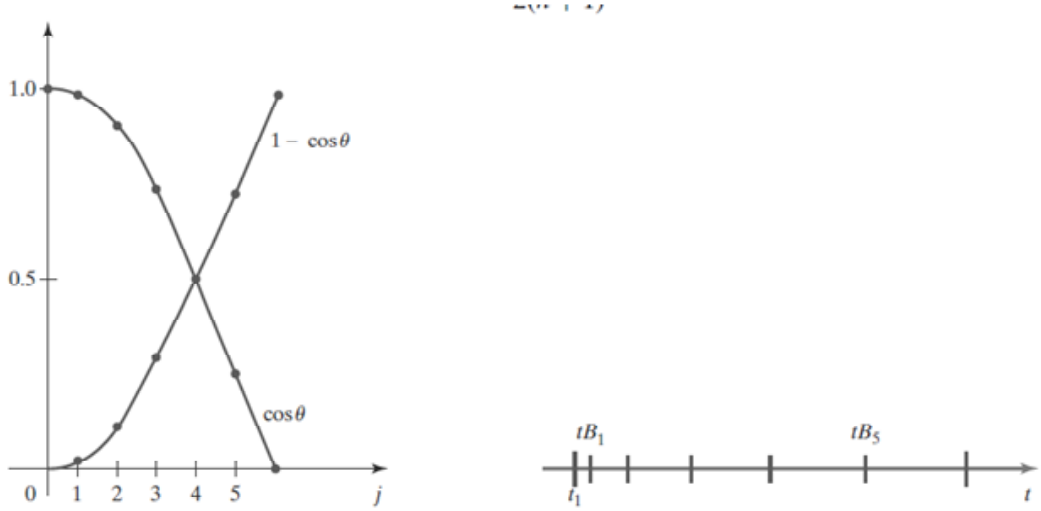


Figure 10: A trigonometric acceleration function and the corresponding in-between spacing for n = 5 and θ = j π/12 (Positive acceleration)

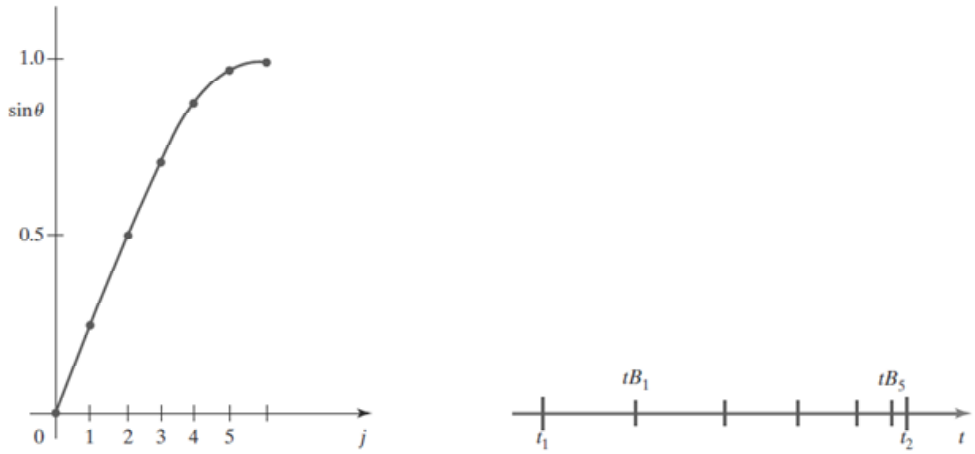


Figure 11: A trigonometric deceleration function and the corresponding in-between spacing for n = 5 and θ = j π/12 (deceleration)

Often, motions contain both speedups and slowdowns. We can model a combination of increasing–decreasing speed by first increasing the in-between time spacing and then decreasing this spacing. A function to accomplish these time changes is

$$\frac{1}{2}(1 - \cos\theta), \qquad 0 < \theta < \pi/2$$

The time for the jth in-between is now calculated as

$$tB_j = t_1 + \Delta t \left\{ \frac{1 - \cos[j\pi/(n+1)]}{2} \right\}, \qquad j = 1, 2, \ldots, n$$

with Δt denoting the time difference between the two key frames. Time intervals for a moving object first increase and then decrease, as shown in Figure 12.

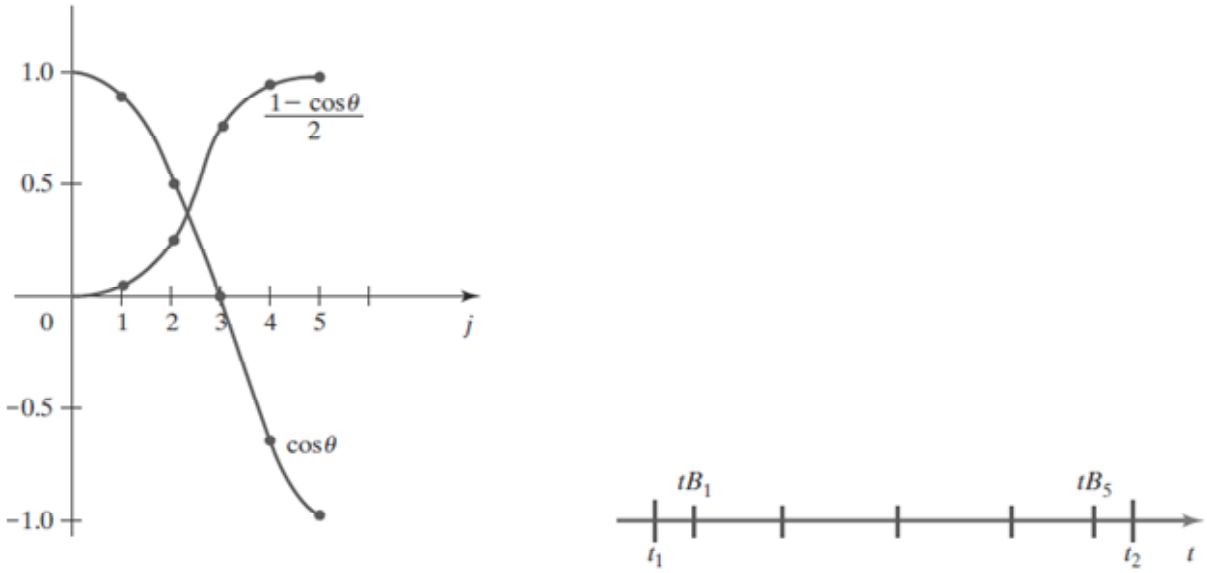


Figure 12: The trigonometric accelerate–decelerate function (1 “ cos θ)/2 and the corresponding in-between spacing for n = 5

Processing the in-betweens is simplified by initially modeling “skeleton” (wire-frame) objects so that motion sequences can be interactively adjusted. After the animation sequence is completely defined, objects can be fully rendered.

12.9 Motion Specifications

General methods for describing an animation sequence range from an explicit specification of the motion paths to a description of the interactions that produce the motions. Thus, we could define how an animation is to take place by giving the transformation parameters, the motion path parameters, the forces that are to act on objects, or the details of how objects interact to produce motion.

Direct Motion Specification

The most straightforward method for defining an animation is direct motion specification of the geometric-transformation parameters. Here, we explicitly set the values for the rotation angles and translation vectors. Then the geometric transformation matrices are applied to transform coordinate positions. Alternatively, we could use an approximating equation involving these parameters to specify certain kinds of motions. We can approximate the path of a bouncing ball, for instance, with a damped, rectified, sine curve (Figure 13):

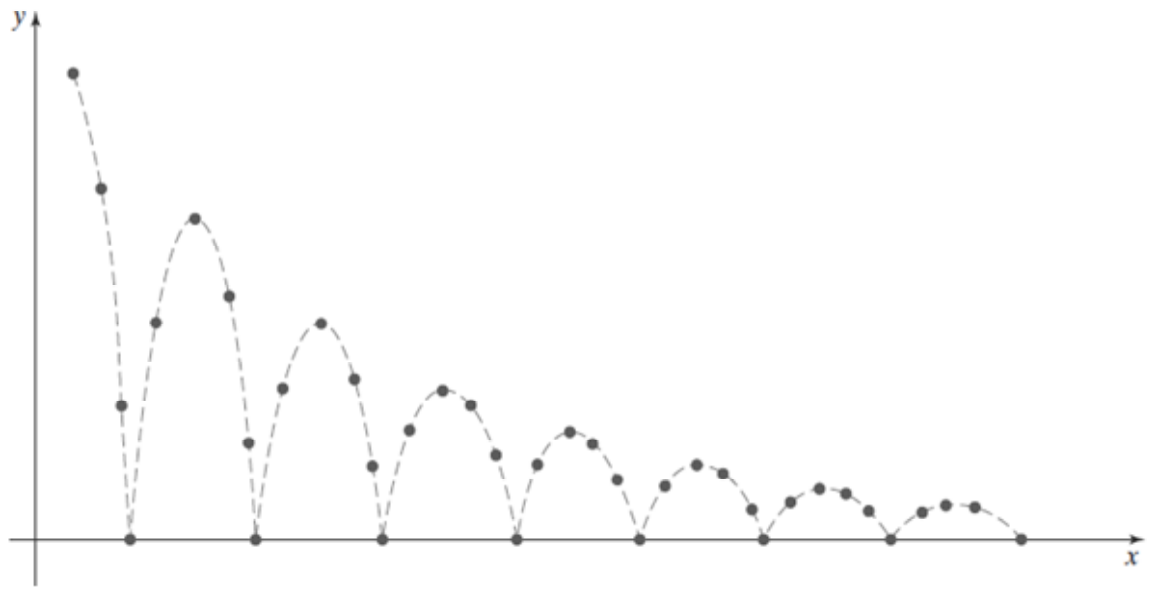


Figure 13: Approximating the motion of a bouncing ball with a damped sine function

$$y(x) = A|\sin(\omega x + \theta_0)|e^{-kx}$$

where A is the initial amplitude (height of the ball above the ground), ω is the angular frequency, θ_0 is the phase angle, and k is the damping constant.

This method for motion specification is particularly useful for simple user programmed animation sequences.

Goal-Directed Systems

At the opposite extreme, we can specify the motions that are to take place in general terms that abstractly describe the actions in terms of the final results. In other words, an animation is specified in terms of the final state of the movements. These systems are referred to as goal-directed, since values for the motion parameters are determined from the goals of the animation. For example, we could specify that we want an object to “walk” or to “run” to a particular destination; or we could state that we want an object to “pick up” some other specified object. The input directives are then interpreted in terms of component motions that will accomplish the described task. Human motions, for instance, can be defined as a hierarchical structure of submotions for the torso, limbs, and so forth. Thus, when a goal, such as “walk to the door” is given, the movements required of the torso and limbs to accomplish this action are calculated.

Kinematics and Dynamics

We can also construct animation sequences using kinematic or dynamic descriptions. With a kinematic description, we specify the animation by giving motion parameters (position, velocity, and acceleration) without reference to causes or goals of the motion. For constant velocity (zero acceleration), we designate the motions of rigid bodies in a scene by giving an initial position and velocity vector for each object. For example, if a velocity is specified as (3, 0, -4) km per sec, then this vector gives the direction for the straight-line motion path and the speed (magnitude of velocity) is calculated as 5 km per sec. If we also specify accelerations (rate of change of velocity), we can generate speedups, slowdowns, and curved motion paths. Kinematic specification of a motion can also be given by simply describing the motion path. This is often accomplished using spline curves.

An alternate approach is to use inverse kinematics. Here, we specify the initial and final positions of objects at specified times and the motion parameters are computed by the system. For example, assuming zero acceleration, we can determine the constant velocity that will accomplish the movement of an object from the initial position to the final position. This method is often used with complex objects by giving the positions and orientations of an end node of an object, such as a hand or a foot. The system then determines the motion parameters of other nodes to accomplish the desired motion.

Dynamic descriptions, on the other hand, require the specification of the forces that produce the velocities and accelerations. The description of object behavior in terms of the influence of forces is generally referred to as physically based modeling.

Examples of forces affecting object motion include electromagnetic, gravitational, frictional, and other mechanical forces.

Object motions are obtained from the force equations describing physical laws, such as Newton's laws of motion for gravitational and frictional processes, Euler or Navier-Stokes equations describing fluid flow, and Maxwell's equations for electromagnetic forces.

For example, the general form of Newton's second law for a particle of mass m is

$$\mathbf{F} = \frac{d}{dt}(m\mathbf{v})$$

where F is the force vector and v is the velocity vector. If mass is constant, we solve the equation F = ma, with a representing the acceleration vector. Otherwise, mass is a function of time, as in relativistic motions or the motions of space vehicles that consume measurable amounts of fuel per unit time. We can also use inverse dynamics to obtain the forces, given the initial and final positions of objects and the type of motion required.

Applications of physically based modeling include complex rigid-body systems and such nonrigid systems as cloth and plastic materials. Typically, numerical methods are used to obtain the motion parameters incrementally from the dynamical equations using initial conditions or boundary values.

12.10 Character Animation

Animation of simple objects is relatively straightforward. When we consider the animation of more complex figures such as humans or animals, however, it becomes much more difficult to create realistic animation. Consider the animation of walking or running human (or humanoid) characters. Based upon observations in their own lives of walking or running people, viewers will expect to see animated characters move in particular ways. If an animated character's movement doesn't match this expectation, the believability of the character may suffer. Thus, much of the work involved in character animation is focused on creating believable movements.

Articulated Figure Animation

A basic technique for animating people, animals, insects, and other critters is to model them as articulated figures, which are hierarchical structures composed of a set of rigid links that are connected at rotary joints (Figure 14). In less formal terms, this just means that we model animate objects as moving stick figures, or simplified skeletons, that can later be wrapped with surfaces representing skin, hair, fur, feathers, clothes, or other outer coverings.

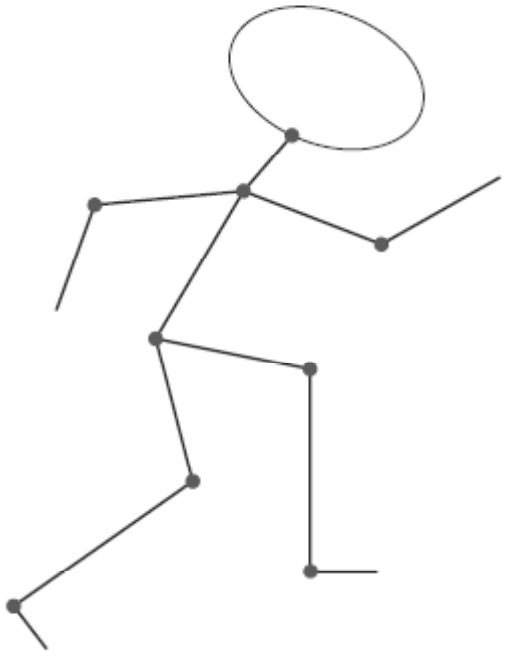


Figure 14: A simple articulated figure with nine joints and twelve connecting links, not counting the oval head.

The connecting points, or hinges, for an articulated figure are placed at the shoulders, hips, knees, and other skeletal joints, which travel along specified motion paths as the body moves. For example, when a motion is specified for an object, the shoulder automatically moves in a certain way and, as the shoulder moves, the arms move. Different types of movement, such as walking, running, or jumping, are defined and associated with particular motions for the joints and connecting links.

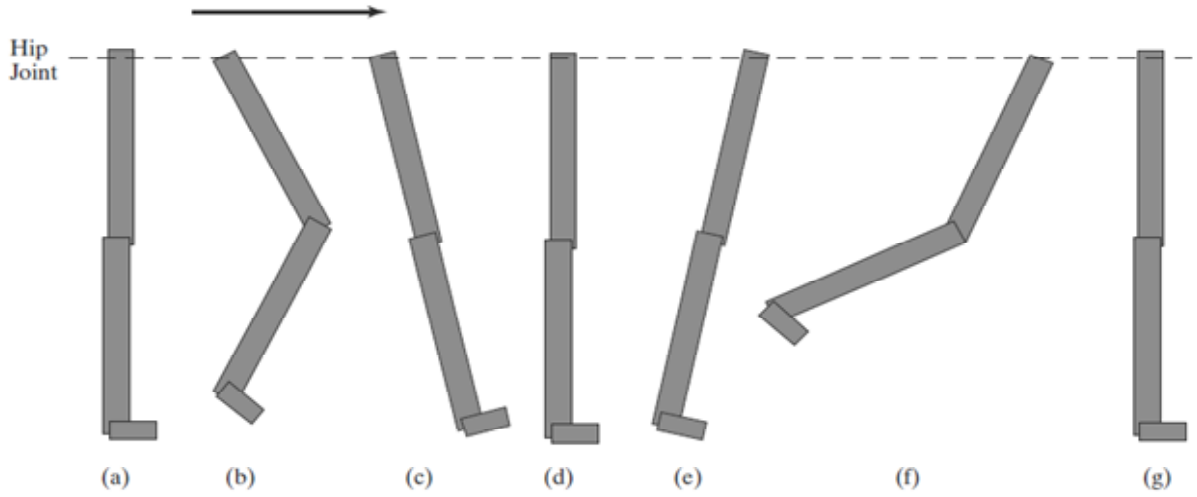


Figure 15: Possible motions for a set of connected links representing a walking leg.

A series of walking leg motions, for instance, might be defined as in Figure 15. The hip joint is translated forward along a horizontal line, while the connecting links perform a series of movements about the hip, knee, and angle joints. Starting with a straight leg [Figure 15(a)], the first motion is a knee bend as the hip moves forward [Figure 15(b)]. Then the leg swings forward, returns to the vertical position, and swings back, as shown in Figures 15(c), (d), and (e). The final motions are a wide swing back and a return to the straight vertical position, as in Figures 15(f) and (g). This motion cycle is repeated for the duration of the animation as the figure moves over a specified distance or time interval.

As a figure moves, other movements are incorporated into the various joints. A sinusoidal motion, often with varying amplitude, can be applied to the hips so that they move about on the torso. Similarly, a rolling or rocking motion can be imparted to the shoulders, and the head can bob up and down.

Both kinematic-motion descriptions and inverse kinematics are used in figure animations. Specifying the joint motions is generally an easier task, but inverse kinematics can be useful for producing simple motion over arbitrary terrain. For a complicated figure, inverse kinematics may not produce a unique animation sequence: Many different rotational motions may be possible for a given set of initial and final conditions. In such cases, a unique solution may be possible by adding more constraints, such as conservation of momentum, to the system.

Motion Capture

An alternative to determining the motion of a character computationally is to digitally record the movement of a live actor and to base the movement of an animated character on that information. This technique, known as motion capture or mo-cap, can be used when the movement of the character is predetermined (as in a scripted scene). The animated character will perform the same series of movements as the live actor.

The classic motion capture technique involves placing a set of markers at strategic positions on the actor's body, such as the arms, legs, hands, feet, and joints. It is possible to place the markers directly on the actor, but more commonly they are affixed to a special skintight body suit worn by the actor. The actor is them filmed performing the scene. Image processing techniques are then used to identify the positions of the markers in each frame of the film, and their positions are translated to coordinates. These coordinates are used to determine the positioning of the body of the animated character. The movement of each marker from frame to frame in the film is tracked and used to control the corresponding movement of the animated character.

To accurately determine the positions of the markers, the scene must be filmed by multiple cameras placed at fixed positions. The digitized marker data from each recording can then be used to triangulate the position of each marker in three dimensions. Typical motion capture systems will use up to two dozen cameras, but systems with several hundred cameras exist.

Optical motion capture systems rely on the reflection of light from a marker into the camera. These can be relatively simple passive systems using photo reflective markers that reflect illumination from special lights placed near the cameras, or more advanced active systems in

which the markers are powered and emit light. Active systems can be constructed so that the markers illuminate in a pattern or sequence, which allows each marker to be uniquely identified in each frame of the recording, simplifying the tracking process.

Non-optical systems rely on the direct transmission of position information from the markers to a recording device. Some non-optical systems use inertial sensors that provide gyroscope-based position and orientation information. Others use magnetic sensors that measure changes in magnetic flux. A series of transmitters placed around the stage generate magnetic fields that induce current in the magnetic sensors; that information is then transmitted to receivers.

Some motion capture systems record more than just the gross movements of the parts of the actor's body. It is possible to record even the actor's facial movements. Often called performance capture systems, these typically use a camera trained on the actor's face and small light-emitting diode (LED) lights that illuminate the face. Small photoreflective markers attached to the face reflect the light from the LEDs and allow the camera to capture the small movements of the muscles of the face, which can then be used to create realistic facial animation on a computer-generated character.

12.11 Periodic Motions

When we construct an animation with repeated motion patterns, such as a rotating object, we need to be sure that the motion is sampled frequently enough to represent the movements correctly. In other words, the motion must be synchronized with the frame-generation rate so that we display enough frames per cycle to show the true motion. Otherwise, the animation may be displayed incorrectly.

A typical example of an under sampled periodic-motion display is the wagon wheel in a Western movie that appears to be turning in the wrong direction. Figure 16 illustrates one complete cycle in the rotation of a wagon wheel with one red spoke that makes 18 clockwise revolutions per second. If this motion is recorded on film at the standard motion-picture projection rate of 24 frames per second, then the first five frames depicting this motion would be as shown in Figure 17. Because the wheel completes $\frac{3}{4}$ of a turn every $\frac{1}{24}$ of a second, only one animation frame is generated per cycle, and the wheel thus appears to be rotating in the opposite (counterclockwise) direction.

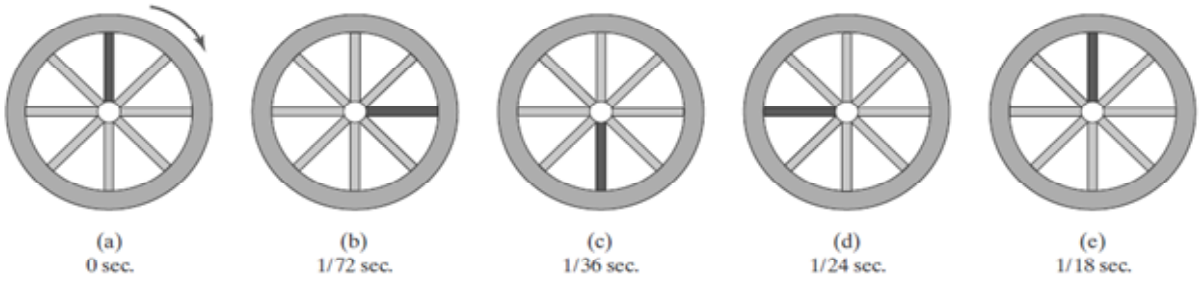


Figure 16: Five positions for a red spoke during one cycle of a wheel motion that is turning at the rate of 18 revolutions per second.

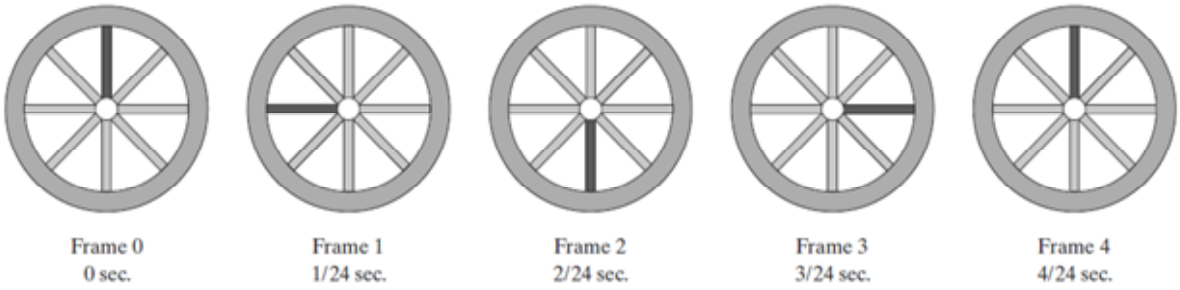


Figure 17: The first five film frames of the rotating wheel in Figure 16 produced at the rate of 24 frames per second.

In a computer-generated animation, we can control the sampling rate in a periodic motion by adjusting the motion parameters. For example, we can set the angular increment for the motion of a rotating object so that multiple frames are generated in each revolution. Thus, a 3^o increment for a rotation angle produces 120 motion steps during one revolution, and a 4^o increment generates 90 steps.

For faster motion, larger rotational steps could be used, so long as the number of samples per cycle is not too small and the motion is clearly displayed. When complex objects are to be animated, we also must take into account the effect that the frame construction time might have on the refresh rate. The motion of a complex object can be much slower than we want it to be if it takes too long to construct each frame of the animation.

Another factor that we need to consider in the display of a repeated motion is the effect of round-off in the calculations for the motion parameters. We can reset parameter values periodically to prevent the accumulated error from producing erratic motions. For a continuous rotation, we could reset parameter values once every cycle (360º).

12.12 Summary

An animation sequence can be constructed frame by frame, or it can be generated in real time.

Animations involving complex scenes and motions are commonly produced one frame at a time, while simpler motion sequences are displayed in real time.

On a raster system, double-buffering methods can be used to facilitate motion displays. One buffer is used to refresh the screen, while a second buffer is being loaded with the screen values for the next frame of the motion.

Translations are accomplished by a simple move of a rectangular block of pixel colors from one frame-buffer position to another.

Rrotations in 90æ% increments can be performed with combinations of translations and row-column interchanges within the pixel array.

Color-table methods can be used for simple raster animations.

By rapidly interchanging the foreground and background color values stored in the color table, we can display the object at various screen positions.

Several developmental stages can be used to produce an animation, starting with the storyboard, object definitions, and specification of key frames.

The storyboard is an outline of the action, and the key frames define the details of the object motions for selected positions in the animation.

Various techniques have been developed for simulating and emphasizing motion effects.

Squash and stretch effects are standard methods for stressing accelerations, and the timing between motion frames can be varied to produce speed variations.

Trigonometric functions are typically used to generate the time spacing for in-between frames when the motions involve accelerations.

Morphing effects, in which one object shape is transformed into another.

Motions in an animation can be described with direct motion specifications or they can be goal-directed.

Motions can be described with equations or with kinematic or dynamic parameters.

Kinematic motion descriptions specify positions, velocities, and accelerations; dynamic motion descriptions are given in terms of the forces acting on the objects in a scene.

Motion capture techniques provide an alternative to computed character motion.

The sampling rate for periodic motions should produce enough frames per cycle to display the animation correctly.

12.13 Model Questions

1. What is Double Buffering?
2. Explain the development stages in designing animation sequences.
3. Define storyboard.
4. What is a key frame?
5. Describe Traditional Animation Techniques and its basics in detail.
6. Explain key-frame system and its importance in animation.
7. What is morphing? Explain the various techniques available for morphing.
8. Explain Character Animation.
9. What is Motion Capture?

LESSON – 13
THREE-DIMENSIONAL OBJECT REPRESENTATIONS

Structure of the lesson

13.1 Introduction

Polygon and quadric surfaces provide precise descriptions for simple Euclidean objects such as polyhedrons and ellipsoids. They are examples of boundary representations (B-reps), which describe a three-dimensional object as a set of surfaces that separate the object interior from the environment. In this chapter, we consider the features of these types of representation schemes and how they are used in computer-graphics applications.

The term spline curve originally referred to a curve drawn in this manner. We can mathematically describe such a curve with a piecewise cubic polynomial function whose first

and second derivatives are continuous across the various curve sections. In computer graphics, the term spline curve now refers to any composite curve formed with polynomial sections satisfying any specified continuity conditions at the boundary of the pieces. A spline surface can be described with two sets of spline curves. There are several different kinds of spline specifications that are used in computer-graphics applications. Each individual specification simply refers to a particular type of polynomial with certain prescribed boundary conditions.

13.2 Objective of the Lesson

In this chapter, we learn different types of representation schemes and how they are used in computer-graphics applications. We will also explore the following terms related to three-dimensional object representations

Spline representation

Bézier Spline Curves

Bézier Surfaces

B-Spline Curves

B-Spline Surfaces

13.3 Polyhedra

The most commonly used boundary representation for a three-dimensional graphics object is a set of surface polygons that enclose the object interior. Many graphics systems store all object descriptions as sets of surface polygons. This simplifies and speeds up the surface rendering and display of objects because all surfaces are described with linear equations. For this reason, polygon descriptions are often referred to as standard graphics objects. In some cases, a polygonal representation is the only one available, but many packages also allow object surfaces to be described with other schemes, such as spline surfaces, which are usually converted to polygonal representations for processing through the viewing pipeline.

To describe an object as a set of polygon facets, we give the list of vertex coordinates for each polygon section over the object surface. The vertex coordinates and edge information for

the surface sections are then stored in tables along with other information, such as the surface normal vector for each polygon. Some graphics packages provide routines for generating a polygon-surface mesh as a set of triangles or quadrilaterals. This allows us to describe a large section of an object's bounding surface, or even the entire surface, with a single command.

And some packages also provide routines for displaying common shapes, such as a cube, sphere, or cylinder, represented with polygon surfaces. Sophisticated graphics systems use fast hardware-implemented polygon renderers that have the capability for displaying a million or more shaded polygons (usually triangles) per second, including the application of surface texture and special lighting effects.

13.4 Curved Surfaces

Equations for objects with curved boundaries can be expressed in either a parametric or a nonparametric form. The various objects that are often useful in graphics applications include quadric surfaces, superquadrics, polynomial and exponential functions, and spline surfaces. These input object descriptions typically are tessellated to produce polygon-mesh approximations for the surfaces.

13.5 Quadric Surfaces

A frequently used class of objects are the quadric surfaces, which are described with second-degree equations (quadratics). They include spheres, ellipsoids, tori, paraboloids, and hyperboloids. Quadric surfaces, particularly spheres and ellipsoids, are common elements of graphics scenes, and routines for generating these surfaces are often available in graphics packages. Also, quadric surfaces can be produced with rational spline representations.

Sphere

In Cartesian coordinates, a spherical surface with radius r centered on the coordinate origin is defined as the set of points (x, y, z) that satisfy the equation

$$x^2 + y^2 + z^2 = r^2$$

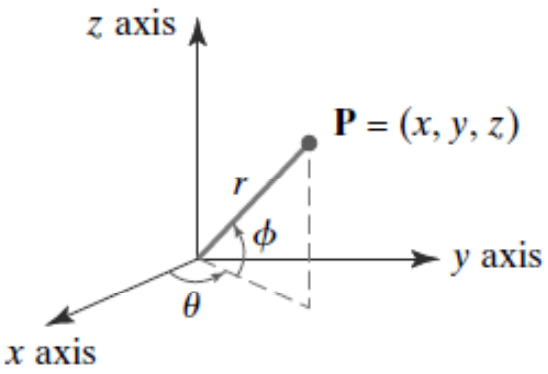


Figure 1: Parametric coordinate position (r , θ, φ) on the surface of a sphere with radius r .

We can also describe the spherical surface in parametric form, using latitude and longitude angles

$$x = r\cos\phi\cos\theta, \qquad -\pi/2 \le \phi \le \pi/2$$
$$y = r\cos\phi\sin\theta, \qquad -\pi \le \theta \le \pi$$
$$z = r\sin\phi$$

The parametric representation provides a symmetric range for the angular parameters θ and φ. Alternatively, we could write the parametric equations using standard spherical coordinates, where angle φ is specified as the colatitude (Figure 2). Then, φ is defined over the range $0 \le \varphi \le \pi$, and θ is often taken in the range $0 \le \theta \le 2\pi$.We could also set up the representation using parameters u and v defined over the range from 0 to 1 by substituting $\varphi = \pi u$ and $\theta = 2\pi v$.

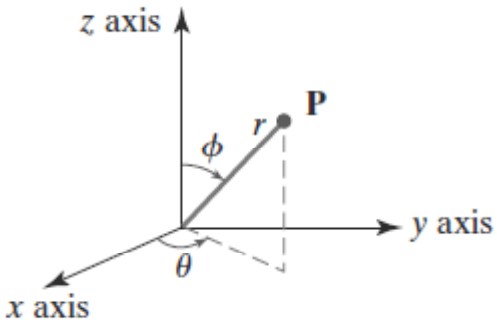


Figure 2: Spherical coordinate parameters (r , θ, φ) , using colatitude for angle φ.

Ellipsoid

An ellipsoidal surface can be described as an extension of a spherical surface where the radii in three mutually perpendicular directions can have different values (Figure 3). The Cartesian representation for points over the surface of an ellipsoid centered on the origin is

$$\left(\frac{x}{r_x}\right)^2 + \left(\frac{y}{r_y}\right)^2 + \left(\frac{z}{r_z}\right)^2 = 1$$

And a parametric representation for the ellipsoid in terms of the latitude angle φand the longitude angle θ is

$$\begin{aligned} x &= r_x \cos\phi \cos\theta, \qquad -\pi/2 \le \phi \le \pi/2 \\ y &= r_y \cos\phi \sin\theta, \qquad -\pi \le \theta \le \pi \\ z &= r_z \sin\phi \end{aligned}$$

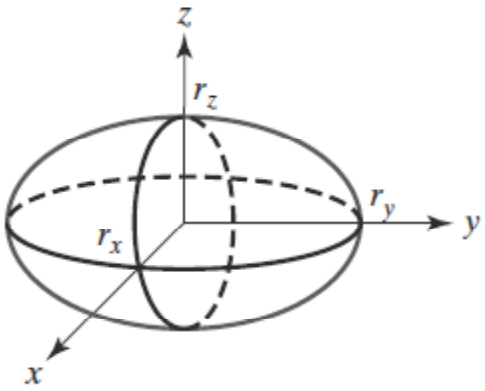


Figure 3: An ellipsoid with radii r_x , r_y , and r_z , centered on the coordinate origin.

Torus

A doughnut-shaped object is called a torus or anchor ring. Most often it is described as the surface generated by rotating a circle or an ellipse about a coplanar axis line that is external to the conic. The defining parameters for a torus are then the distance of the conic center from the rotation axis and the dimensions of the conic.

A torus generated by the rotation of a circle with radius r in the yz plane about the z axis is shown in Figure 4. With the circle center on the y axis, the axial radius, r_{axial}, of the resulting torus is equal to the distance along the y axis to the circle center from the z axis (the rotation axis); and the cross-sectional radius of the torus is the radius of the generating circle.

The equation for the cross-sectional circle shown in the side view of Figure 4 is

$$(y - r_{\text{axial}})^2 + z^2 = r^2$$

Rotating this circle about the z axis produces the torus whose surface positions are described with the Cartesian equation

$$\left(\sqrt{x^2 + y^2} - r_{\text{axial}}\right)^2 + z^2 = r^2$$

The corresponding parametric equations for the torus with a circular cross-section are

$$\begin{aligned} x &= (r_{\text{axial}} + r\cos\phi)\cos\theta, \qquad -\pi \le \phi \le \pi \\ y &= (r_{\text{axial}} + r\cos\phi)\sin\theta, \qquad -\pi \le \theta \le \pi \\ z &= r\sin\phi \end{aligned}$$

We could also generate a torus by rotating an ellipse, instead of a circle, about the z axis. For an ellipse in the yz plane with semimajor and semiminor axes denoted as r_y and r_z, we can write the ellipse equation as

$$\left(\frac{y - r_{\text{axial}}}{r_y}\right)^2 + \left(\frac{z}{r_z}\right)^2 = 1$$

where r_{axial} is the distance along the y axis from the rotation z axis to the ellipse center. This generates a torus that can be described with the Cartesian equation

$$\left(\frac{\sqrt{x^2+y^2}-r_{\text{axial}}}{r_y}\right)+\left(\frac{z}{r_z}\right)^2=1$$

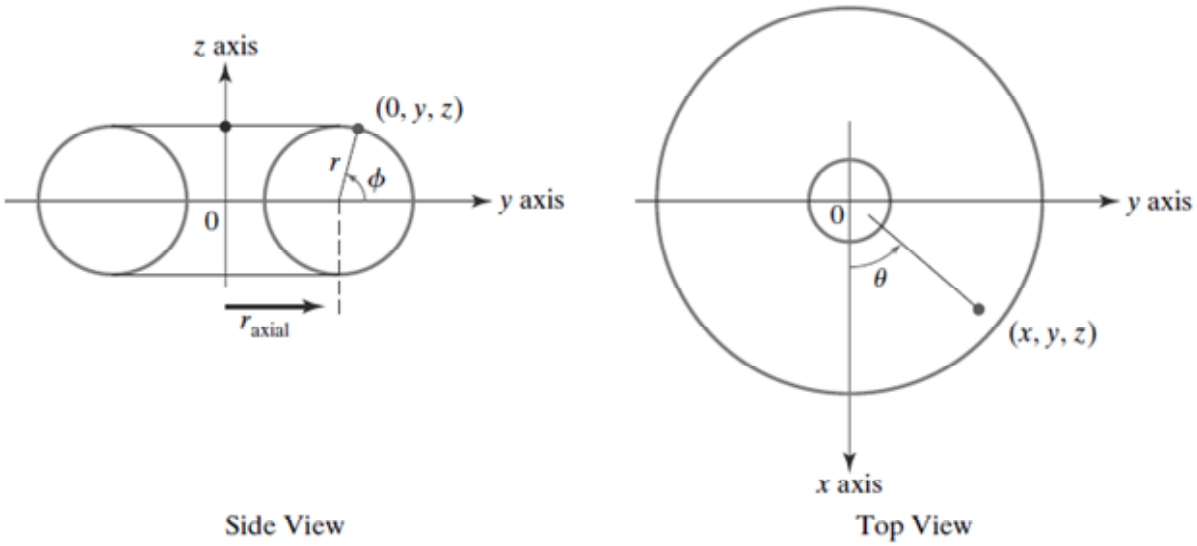


Figure 4: A torus, centered on the coordinate origin, with a circular cross-section and with the torus axis along the z axis.

The corresponding parametric representation for the torus with an elliptical cross section is

$$\begin{aligned} x &= (r_{\text{axial}} + r_y \cos\phi)\cos\theta, \qquad -\pi \le \phi \le \pi \\ y &= (r_{\text{axial}} + r_y \cos\phi)\sin\theta, \qquad -\pi \le \theta \le \pi \\ z &= r_z \sin\phi \end{aligned}$$

Other variations on the preceding torus equations are possible. For example, we could generate a torus surface by rotating either a circle or an ellipse along an elliptical path around the rotation axis.

13.6 Superquadrics

The class of objects called Superquadrics is a generalization of the quadric representations. Superquadrics are formed by incorporating additional parameters into the quadric equations to provide increased flexibility for adjusting object shapes. One additional parameter is added to curve equations, and two additional parameters are used in surface equations.

Superellipse

We obtain a Cartesian representation for a superellipse from the corresponding equation for an ellipse by allowing the exponent on the x and y terms to be variable. One way to do this is to write the Cartesian superellipse equation in the form

$$\left(\frac{x}{r_x}\right)^{2/s} + \left(\frac{y}{r_y}\right)^{2/s} = 1$$

where parameter s can be assigned any real value. When s =1, we have an ordinary ellipse.

Corresponding parametric equations for the superellipse can be expressed as

$$x = r_x \cos^s \theta, \qquad -\pi \le \theta \le \pi$$
$$y = r_y \sin^s \theta$$

Figure 5 illustrates superellipse shapes that can be generated using various values for parameter s.

Figure 5: Superellipses plotted with values for parameter s ranging from 0.5 to 3.0 and with $r_x=r_y$.

Superellipsoid

A Cartesian representation for a superellipsoid is obtained from the equation for an ellipsoid by incorporating two exponent parameters as follows:

$$\left[\left(\frac{x}{r_x}\right)^{2/s_2} + \left(\frac{y}{r_y}\right)^{2/s_2}\right]^{s_2/s_1} + \left(\frac{z}{r_z}\right)^{2/s_1} = 1$$

For s1 = s2 = 1, we have an ordinary ellipsoid. We can then write the corresponding parametric representation for the superellipsoid as

$$\begin{aligned} x &= r_x \cos^{s_1}\phi \cos^{s_2}\theta, & -\pi/2 \le \phi \le \pi/2 \\ y &= r_y \cos^{s_1}\phi \sin^{s_2}\theta, & -\pi \le \theta \le \pi \\ z &= r_z \sin^{s_1}\phi \end{aligned}$$

13.7 Spline representation

Interpolation and Approximation Splines

We specify a spline curve by giving a set of coordinate positions, called control points, which indicate the general shape of the curve. These coordinate positions are then fitted with piecewise-continuous, parametric polynomial functions in one of two ways. When polynomial

sections are fitted so that all the control points are connected, as in Figure 6, the resulting curve is said to interpolate the set of control points. On the other hand, when the generated polynomial curve is plotted so that some, or all, of the control points are not on the curve path, the resulting curve is said to approximate the set of control points (Figure 7). Similar methods are used to construct interpolation or approximation spline surfaces.

Figure 6: A set of six control points interpolated with piecewise continuous polynomial sections.

Interpolation methods are commonly used to digitize drawings or to specify animation paths. Approximation methods are used primarily as design tools to create object shapes.

A spline curve or surface is defined, modified, and manipulated with operations on the control points.

A set of control points forms a boundary for a region of space that is called the convex hull.

Figure 7: A set of six control points approximated with piecewise continuous polynomial sections.

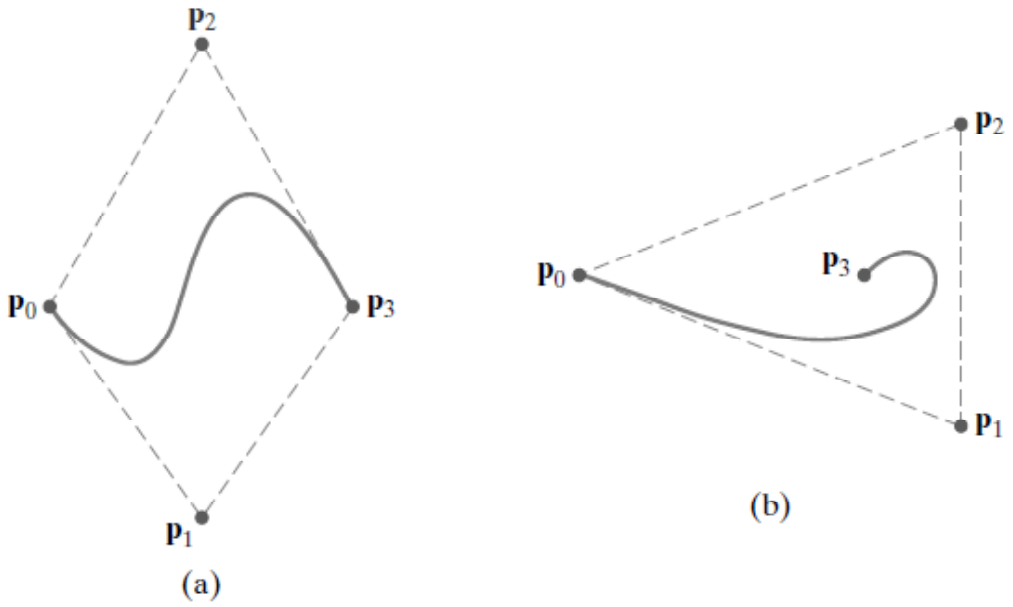


Figure 8: Convex-hull shapes (dashed lines) for two sets of control points in the x y plane.

A polyline connecting the sequence of control points for an approximation spline curve is usually displayed to remind a designer of the control-point positions and ordering. This set of connected line segments is called the control graph for the curve.

To ensure a smooth transition from one section of a piecewise parametric spline to the next, we can impose various continuity conditions at the connection points.

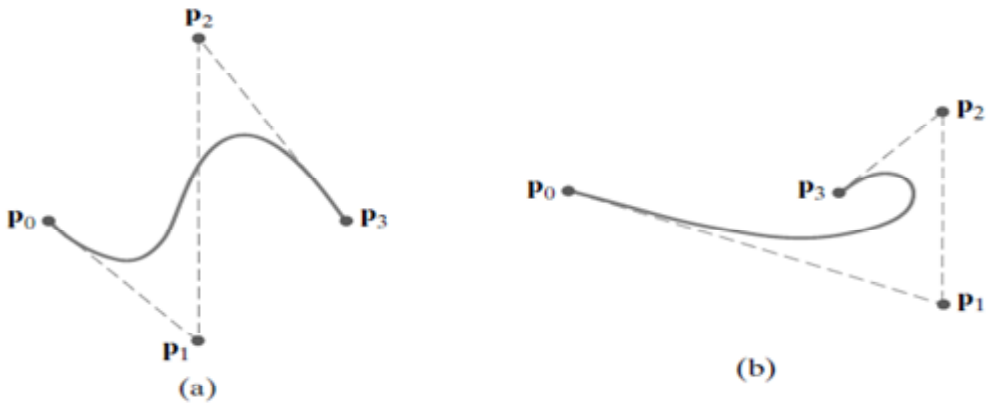


Figure 9: Control-graph shapes (dashed lines) for two sets of control points in the x y plane.

If each section of a spline curve is described with a set of parametric coordinate functions of the form

$$x = x(u), \qquad y = y(u), \qquad z = z(u), \qquad u_1 \le u \le u_2$$

we set parametric continuity by matching the parametric derivatives of adjoining curve sections at their common boundary.

Zero-order parametric continuity, represented as C^0 continuity, means simply that the curves meet. That is, the values of x, y, and z evaluated at u_2 for the first curve section are equal, respectively, to the values of x, y, and z evaluated at u_1 for the next curve section. First-order parametric continuity, referred to as C^1continuity, means that the first parametric derivatives (tangent lines) of the coordinate functions for two successive curve sections are equal at their joining point. Second-order parametric continuity, or C^2 continuity, means that both the first and second parametric derivatives of the two curve sections are the same at the intersection.

Geometric Continuity

Geometric Continuity Conditions

Another method for joining two successive curve sections is to specify conditions for geometric continuity. In this case, we require only that the parametric derivatives of the two sections are proportional to each other at their common boundary, instead of requiring equality.

Zero-order geometric continuity, described as G^0 continuity, is the same as zero-order parametric continuity. That is, two successive curve sections must have the same coordinate position at the boundary point. First-order geometric continuity, or G^1 continuity, means that the parametric first derivatives are proportional at the intersection of two successive sections. If we denote the parametric position on the curve as P(u), the direction of the tangent vector P'(u), but not necessarily its magnitude, will be the same for two successive curve sections at their common point under G^1 continuity. Second-order geometric continuity, or G^2 continuity, means that both the first and second parametric derivatives of the two curve sections are proportional at their boundary. Under G^2 continuity, curvatures of two curve sections will match at the joining position.

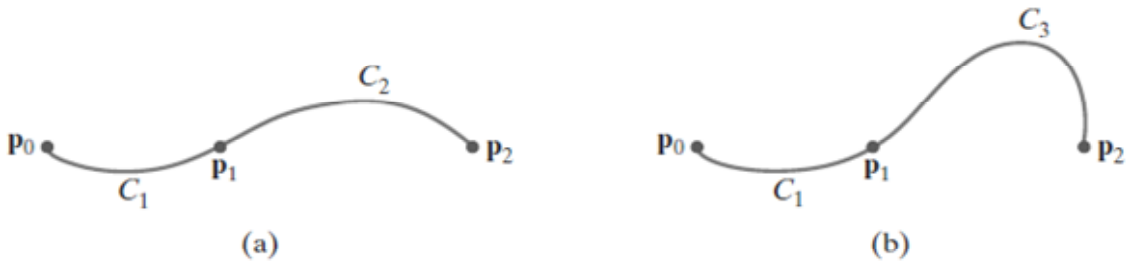


Figure 10: Three control points fitted with two curve sections joined with (a) parametric continuity and (b) geometric continuity.

Spline Specifications

There are three equivalent methods for specifying a particular spline representation, given the degree of the polynomial and the control-point positions:

(1) We can state the set of boundary conditions that are imposed on the spline; or

(2) We can state the matrix that characterizes the spline; or

(3) We can state the set of blending functions (or basis functions) that determine how specified constraints on the curve are combined to calculate positions along the curve path.

To illustrate these three equivalent specifications, suppose we have the following parametric cubic polynomial representation for the x coordinate along the path of a spline-curve section:

$$x(u) = a_x u^3 + b_x u^2 + c_x u + d_x, \qquad 0 \le u \le 1$$

Boundary conditions for this curve can be set for the endpoint coordinate positions x(0) and x(1) and for the parametric first derivatives at the endpoints: $x'(0)$ and $x'(1)$. These four boundary conditions are sufficient to determine the values of the four coefficients a_x, b_x, c_x, and d_x.

We can write the boundary conditions in matrix form and solve for the coefficient matrix C as

$$\mathbf{C} = \mathbf{M}_{\text{spline}} \cdot \mathbf{M}_{\text{geom}}$$

Where M_{geom} is a four-element column matrix containing the geometric constraint values (boundary conditions) on the spline, and M_{spline} is the 4 by 4 matrix that transforms the geometric constraint values to the polynomial coefficients and provides a characterization for the spline curve. Matrix M_{geom} contains control point coordinate values and other geometric constraints that have been specified.

To obtain a polynomial representation for coordinate x in terms of the geometric constraint parameters g_k , such as the control-point coordinates and slope of the curve at the control points:

$$x(u) = \sum_{k=0}^{3} g_k \cdot \mathrm{BF}_k(u)$$

The polynomials $BF_k(u)$, for k = 0, 1, 2, 3, are called blending functions or basis functions because they combine (blend) the geometric constraint values to obtain coordinate positions along the curve.

Spline Surfaces

The usual procedure for defining a spline surface is to specify two sets of spline curves using a mesh of control points over some region of space. If we denote the control-point positions as $p_{ku,kv}$, then any point position on the spline surface can be computed as the product of the spline-curve blending functions as follows:

$$\mathbf{P}(u, v) = \sum_{k_u, k_v} \mathbf{p}_{k_u, k_v} \mathrm{BF}_{k_u}(u) \mathrm{BF}_{k_v}(v)$$

Surface parameters u and v often vary over the range from 0 to 1, but this range depends on the type of spline curves we use. One method for designating the three-dimensional control-point positions is to select height values above a two-dimensional mesh of positions on a ground plane.

13.8 Bézier Spline Curves

This spline approximation method was developed by the French engineer Pierre Bézier for use in the design of Renault automobile bodies. Bézier splines have a number of properties that make them highly useful and convenient for curve and surface design. They are also easy to implement. For these reasons, Bézier splines are widely available in various CAD systems, in general graphics packages, and in assorted drawing and painting packages.

In general, a Bézier curve section can be fitted to any number of control points, although some graphic packages limit the number of control points to four. The degree of the Bézier polynomial is determined by the number of control points to be approximated and their relative position. As with the interpolation splines, we can specify the Bézier curve path in the vicinity of the control points using blending functions, a characterizing matrix, or boundary conditions. For general Bézier curves, with no restrictions on the number of control points, the blending function specification is the most convenient representation.

Bézier Curve Equations

We first consider the general case of n + 1 control-point positions, denoted as $p_k = (x_k , y_k , z_k)$, with k varying from 0 to n. These coordinate points are blended to produce the following position vector P(u), which describes the path of an approximating Bézier polynomial function between p_0 and p_n:

$$\mathbf{P}(u) = \sum_{k=0}^{n} \mathbf{p}_k \, \mathrm{BEZ}_{k,n}(u), \qquad 0 \le u \le 1$$

The Bézier blending functions $BEZ_{k,n}(u)$ are the Bernstein polynomials

$$\mathrm{BEZ}_{k,n}(u) = C(n, k)u^k(1 - u)^{n-k}$$

where parameters C(n, k) are the binomial coefficients

$$C(n, k) = \frac{n!}{k!(n - k)!}$$

Equation P(u) represents a set of three parametric equations for the individual curve coordinates:

$$x(u) = \sum_{k=0}^{n} x_k \, \mathrm{BEZ}_{k,n}(u)$$

$$y(u) = \sum_{k=0}^{n} y_k \, \mathrm{BEZ}_{k,n}(u)$$

$$z(u) = \sum_{k=0}^{n} z_k \, \mathrm{BEZ}_{k,n}(u)$$

In most cases, a Bézier curve is a polynomial of a degree that is one less than the designated number of control points: Three points generate a parabola, four points a cubic curve, and so forth. Figure 11 demonstrates the appearance of some Bézier curves for various selections of control points in the xy plane (z = 0). With certain control-point placements, however, we obtain degenerate Bézier polynomials. For example, a Bézier curve generated with three collinear control points is a straight-line segment; and a set of control points that are all at the same coordinate position produce a Bézier “curve” that is a single point.

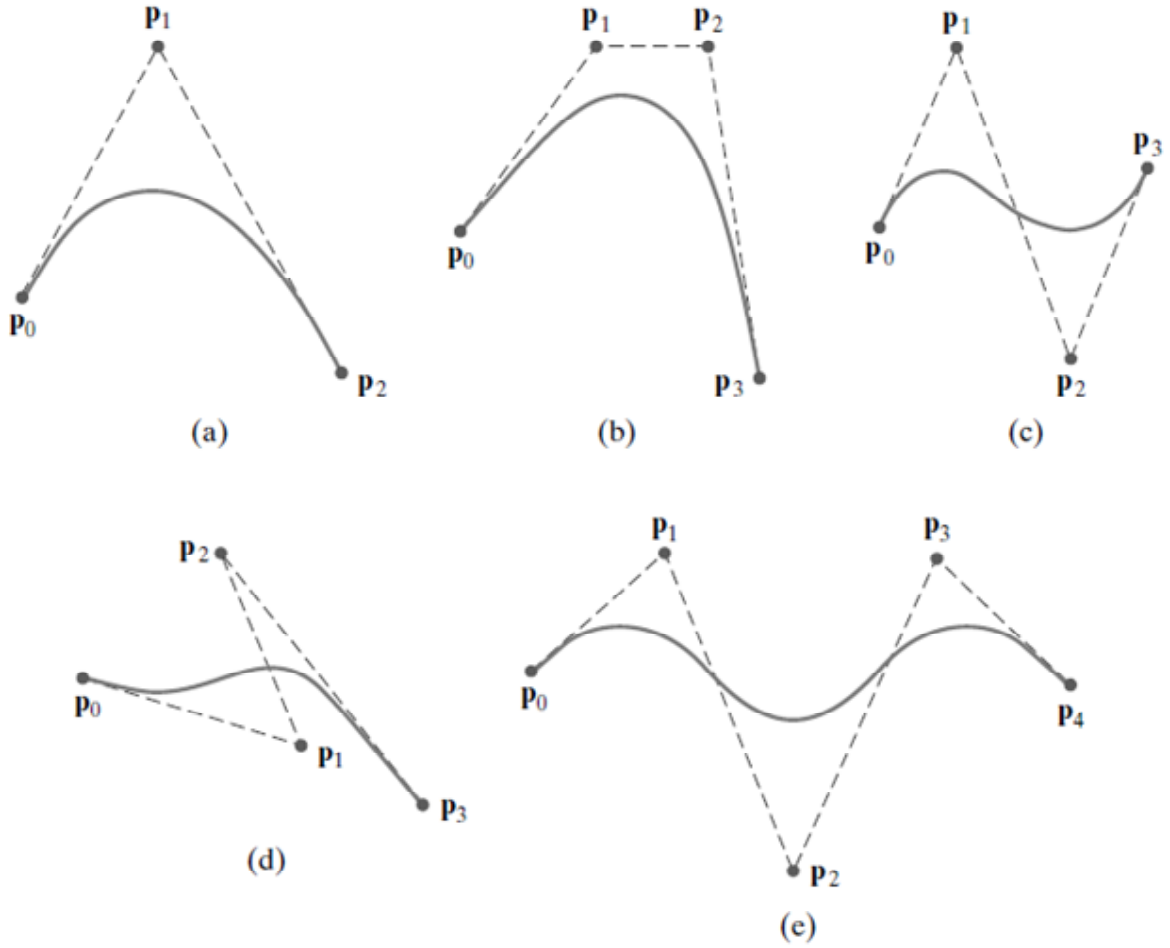


Figure 11: Examples of two-dimensional Bézier curves generated with three, four, and five control points. Dashed lines connect the control-point positions.

Recursive calculations can be used to obtain successive binomial-coefficient values as

$$C(n,k) = \frac{n-k+1}{k} C(n,k-1)$$

for n ≥ k. Also, the Bézier blending functions satisfy the recursive relationship

$$\mathrm{BEZ}_{k,n}(u) = (1-u)\mathrm{BEZ}_{k,n-1}(u) + u\,\mathrm{BEZ}_{k-1,n-1}(u), \qquad n > k \geq 1$$

with $\mathrm{BEZ}_{k,k} = u^k$ and $\mathrm{BEZ}_{0,k} = (1-u)^k$.

Properties of Bézier Curves

A very useful property of a Bézier curve is that the curve connects the first and last control points. Thus, a basic characteristic of any Bézier curve is that

$$\mathbf{P}(0) = \mathbf{p}_0$$
$$\mathbf{P}(1) = \mathbf{p}_n$$

Values for the parametric first derivatives of a Bézier curve at the endpoints can be calculated from control-point coordinates as

$$\mathbf{P}'(0) = -n\mathbf{p}_0 + n\mathbf{p}_1$$
$$\mathbf{P}'(1) = -n\mathbf{p}_{n-1} + n\mathbf{p}_n$$

From these expressions, we see that the slope at the beginning of the curve is along the line joining the first two control points, and the slope at the end of the curve is along the line joining the last two endpoints. Similarly, the parametric second derivatives of a Bézier curve at the endpoints are calculated as

$$\mathbf{P}''(0) = n(n-1)[(\mathbf{p}_2 - \mathbf{p}_1) - (\mathbf{p}_1 - \mathbf{p}_0)]$$
$$\mathbf{P}''(1) = n(n-1)[(\mathbf{p}_{n-2} - \mathbf{p}_{n-1}) - (\mathbf{p}_{n-1} - \mathbf{p}_n)]$$

Another important property of any Bézier curve is that it lies within the convex hull (convex polygon boundary) of the control points. This follows from the fact that the Bézier blending functions are all positive and their sum is always 1:

$$\sum_{k=0}^{n} \mathrm{BEZ}_{k,n}(u) = 1$$

so that any curve position is simply the weighted sum of the control-point positions. The convex-hull property for a Bézier curve ensures that the polynomial smoothly follows the control points without erratic oscillations.

13.9 Bézier Surfaces

Two sets of orthogonal Bézier curves can be used to design an object surface. The parametric vector function for the Bézier surface is formed as the tensor product of Bézier blending functions:

$$\mathbf{P}(u, v) = \sum_{j=0}^{m} \sum_{k=0}^{n} \mathbf{p}_{j,k} \, \mathrm{BEZ}_{j,m}(v) \, \mathrm{BEZ}_{k,n}(u)$$

with $p_{j,k}$ specifying the location of the (m + 1) by (n + 1) control points.

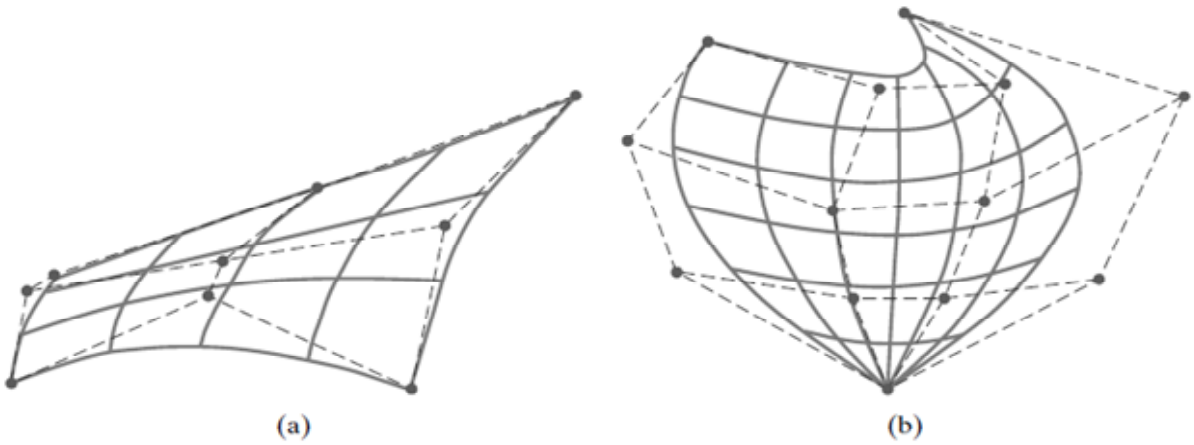


Figure 12: Wire-frame Bézier surfaces constructed with (a) 9 control points arranged in a 3 × 3 mesh and (b) 16 control points arranged in a 4 × 4 mesh. Dashed lines connect the control points.

Figure 12 illustrates two Bézier surface plots. The control points are connected by dashed lines, and the solid lines show curves of constant u and constant v. Each curve of constant u is plotted by varying v over the interval from 0 to 1, with u fixed at one of the values in this unit interval. Curves of constant v are plotted similarly.

Bézier surfaces have the same properties as Bézier curves, and they provide a convenient method for interactive design applications. To specify the three-dimensional coordinate positions for the control points, we could first construct a rectangular grid in the xy “ground” plane. We then choose elevations above the ground plane at the grid intersections as the z-coordinate values for the control points.

13.10 B-Spline Curves

This spline category is the most widely used, and B-spline functions are commonly available in CAD systems and many graphics-programming packages. Like Bézier splines, B-splines are generated by approximating a set of control points.

But B-splines have two advantages over Bézier splines:

(1) the degree of a B-spline polynomial can be set independently of the number of control points (with certain limitations), and

(2) B-splines allow local control over the shape of a spline. The tradeoff is that B-splines are more complex than Bézier splines.

B-Spline Curve Equations

We can write a general expression for the calculation of coordinate positions along a B-spline curve using a blending-function formulation as

$$\mathbf{P}(u) = \sum_{k=0}^{n} \mathbf{p}_k B_{k,d}(u), \qquad u_{\min} \le u \le u_{\max}, \quad 2 \le d \le n+1$$

where p_k is an input set of n + 1 control points. There are several differences between this B-spline formulation and the expression for a Bézier spline curve.

The range of parameter u now depends on how we choose the other B-spline parameters. And the B-spline blending functions $B_{k,d}$ are polynomials of degree d - 1, where d is the degree parameter. (Sometimes parameter d is alluded to as the “order” of the polynomial, but this can

be misleading because the term order is also often used to mean simply the degree of the polynomial.) The degree parameter d can be assigned any integer value in the range from 2 up to the number of control points (n + 1). Actually, we could also set the value of the degree parameter at 1, but then our "curve" is just a point plot of the control points. Local control for B-splines is achieved by defining the blending functions over subintervals of the total range of u.

Blending functions for B-spline curves are defined by the Cox-deBoor recursion formulas:

$$B_{k,1}(u) = \begin{cases} 1 & \text{if } u_k \le u \le u_{k+1} \\ 0 & \text{otherwise} \end{cases}$$

$$B_{k,d}(u) = \frac{u - u_k}{u_{k+d-1} - u_k} B_{k,d-1}(u) + \frac{u_{k+d} - u}{u_{k+d} - u_{k+1}} B_{k+1,d-1}(u)$$

where each blending function is defined over d subintervals of the total range of u. Each subinterval endpoint uj is referred to as a knot, and the entire set of selected subinterval endpoints is called a knot vector. We can choose any values for the subinterval endpoints, subject to the condition $u_j \le u_{j+1}$. Values for u_{min} and u_{max} then depend on the number of control points we select, the value we choose for the degree parameter d, and how we set up the subintervals (knot vector). Because it is possible to choose the elements of the knot vector so that some denominators in the Cox-deBoor calculations evaluate to 0, this formulation assumes that any terms evaluated as 0/0 are to be assigned the value 0.

Figure 13 demonstrates the local-control characteristics of B-splines. In addition to local control, B-splines allow us to vary the number of control points used to design a curve without changing the degree of the polynomial. Also, we can increase the number of values in the knot vector to aid in curve design. When we do this, however, we must add control points because the size of the knot vector depends on parameter n.

B-spline curves have the following properties:

- The polynomial curve has degree d -1 and C^{d-2} continuity over the range of u.

- For n+1 control points, the curve is described with n+1 blending functions.
- Each blending function $B_{k,d}$ is defined over d subintervals of the total range of u, starting at knot value u_k .
- The range of parameter u is divided into n+d subintervals by the n+d +1 values specified in the knot vector.
- With knot values labeled as $\{u_0, u_1, \ldots, u_{n+d}\}$, the resulting B-spline curve is defined only in the interval from knot value $u_{d"1}$ up to knot value u_{n+1}.
- Each section of the spline curve (between two successive knot values) is influenced by d control points.
- Any one control point can affect the shape of at most d curve sections.

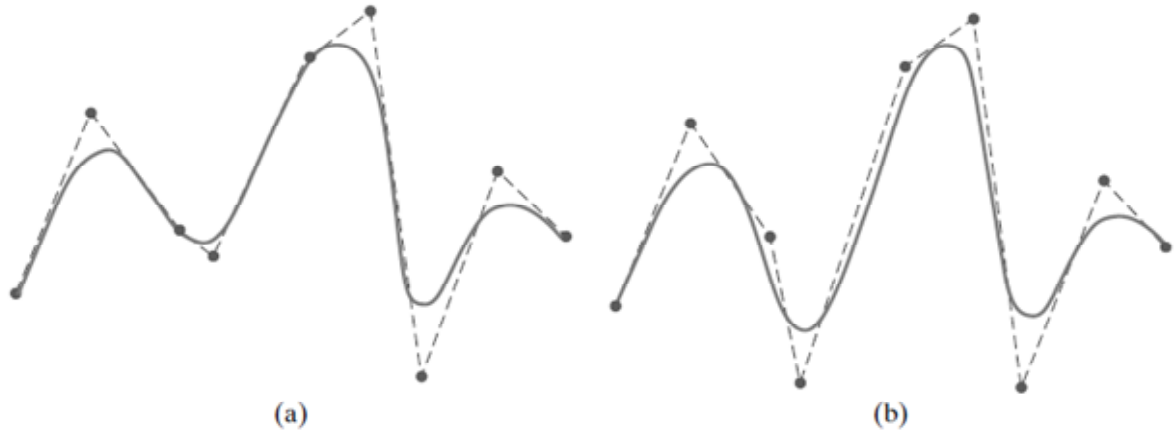


Figure 13: Local modification of a B-spline curve. Changing one of the control points in (a) produces curve (b), which is modified only in the neighborhood of the altered control point.

In addition, a B-spline curve lies within the convex hull of at most d + 1 control points, so that B-splines are tightly bound to the input positions. For any value of u in the interval from knot value u_{d-1} to u_{n+1}, the sum over all basis functions is 1, as follows:

$$\sum_{k=0}^{n} B_{k,d}(u) = 1$$

Given the control-point positions and the value of the degree parameter d, we then need to specify the knot values to obtain the blending functions. There are three general classifications for knot vectors: uniform, open uniform, and nonuniform. B-splines are commonly described according to the selected knot vector class.

Uniform Periodic B-Spline Curves

When the spacing between knot values is constant, the resulting curve is called a uniform B-spline. Uniform B-splines have periodic blending functions. That is, for given values of n and d, all blending functions have the same shape. Each successive blending function is simply a shifted version of the previous function:

$$B_{k,d}(u) = B_{k+1,d}(u + \Delta u) = B_{k+2,d}(u + 2\Delta u)$$

where Δu is the interval between adjacent knot values.

Cubic Periodic B-Spline Curves

Because cubic periodic B-splines are commonly used in graphics packages, we consider the formulation for this class of splines. Periodic splines are particularly useful for generating certain closed curves.

To derive the curve equations for a periodic, cubic B-spline, we consider an alternate formulation by starting with the boundary conditions and obtaining the blending functions normalized to the interval $0 \leq u \leq 1$. Using this formulation, we can also obtain the characteristic matrix easily. The boundary conditions for periodic cubic B-splines with four control points, labeled p_0, p_1, p_2, and p_3, are

$$\mathbf{P}(0) = \frac{1}{6}(\mathbf{p}_0 + 4\mathbf{p}_1 + \mathbf{p}_2)$$

$$\mathbf{P}(1) = \frac{1}{6}(\mathbf{p}_1 + 4\mathbf{p}_2 + \mathbf{p}_3)$$

$$\mathbf{P}'(0) = \frac{1}{2}(\mathbf{p}_2 - \mathbf{p}_0)$$

$$\mathbf{P}'(1) = \frac{1}{2}(\mathbf{p}_3 - \mathbf{p}_1)$$

These boundary conditions are similar to those for cardinal splines: Curve sections are defined with four control points, and parametric derivatives (slopes) at the beginning and end of each curve section are parallel to the chords joining adjacent control points. The B-spline curve section starts at a position near p_1 and ends at a position near p_2.

A matrix formulation for a cubic periodic B-spline with four control points can then be written as

$$\mathbf{P}(u) = [u^3 \quad u^2 \quad u \quad 1] \cdot \mathbf{M}_B \cdot \begin{bmatrix} \mathbf{p}_0 \\ \mathbf{p}_1 \\ \mathbf{p}_2 \\ \mathbf{p}_3 \end{bmatrix}$$

Open Uniform B-Spline Curves

This class of B-splines is a cross between uniform B-splines and nonuniform B-splines. Sometimes it is treated as a special type of uniform B-spline, and sometimes it is considered to be in the nonuniform B-spline classification. For the open uniform B-splines, or simply open B-splines, the knot spacing is uniform except at the ends, where knot values are repeated d times.

For any values of parameters d and n, we can generate an open uniform knot vector with integer values using the calculations

$$u_j = \begin{cases} 0 & \text{for } 0 \le j < d \\ j - d + 1 & \text{for } d \le j \le n \\ n - d + 2 & \text{for } j > n \end{cases}$$

for values of j ranging from 0 to n+d.With this assignment, the first d knots are assigned the value 0, and the last d knots have the value n - d + 2.

Nonuniform B-Spline Curves

For this class of splines, we can specify any values and intervals for the knot vector. With nonuniform B-splines, we can choose multiple internal knot values and unequal spacing between the knot values. Nonuniform B-splines provide increased flexibility in controlling a curve shape. With unequally spaced intervals in the knot vector, we obtain different shapes for the blending functions in different intervals, which can be used in designing the spline features. By increasing knot multiplicity, we can produce subtle variations in the curve path and introduce discontinuities. Multiple knot values also reduce the continuity by 1 for each repeat of a particular value.

13.11 B-Spline Surfaces

Formulation of a B-spline surface is similar to that for Bézier splines. We can obtain a vector point function over a B-spline surface using the tensor product of B-spline blending functions in the form

$$\mathbf{P}(u, v) = \sum_{k_u=0}^{n_u} \sum_{k_v=0}^{n_v} \mathbf{p}_{k_u,k_v} B_{k_u,d_u}(u) B_{k_v,d_v}(v)$$

where the vector values for p_{ku},k_v specify the positions of the $(n_u + 1)$ by $(n_v + 1)$ control points.

B-spline surfaces exhibit the same properties as those of their component B-spline curves. A surface can be constructed from selected values for degree parameters d_u and d_v, which set the degrees for the orthogonal surface polynomials at d_u -1 and d_v-1. For each surface parameter u and v, we also select values for the knot vectors, which determines the parameter range for the blending functions.

A generalization of B-splines are the beta-splines, also referred to as β-splines, that are formulated by imposing geometric continuity conditions on the first and second parametric derivatives. The continuity parameters for beta-splines are called β parameters.

13.12 Summary

A three-dimensional object representation is rendered by a software package as a standard graphics object, whose surfaces are displayed as a polygon mesh.

Functions for displaying some common quadric surfaces, such as spheres and ellipsoids, are often available in graphics packages.

Extensions of the quadrics, called superquadrics, provide additional parameters for creating a wider variety of object shapes.

The most widely used methods for representing objects in CAD applications are the spline representations, which are piecewise continuous polynomial functions.

A spline curve or surface is defined with a set of control points and the boundary conditions on the spline sections.

Lines connecting the sequence of control points forms the control graph.

A spline surface can be described with the tensor product of two polynomials.

Cubic polynomials are commonly used for the interpolation representations, which include the Hermite, cardinal, and Kochanek-Bartels splines.

Bézier splines provide a simple and powerful approximation method for describing curved lines and surfaces.

B-splines, which include Bézier splines as a special case, are a more versatile approximation representation, but they require the specification of a knot vector.

13.13 Model Questions

1. What are Interpolation and Approximation Splines?
2. Explain the properties of Bézier Curves.
3. Explain B-splines and its properties.
4. State the advantage of B-splines over Bézier Spline Curves
5. Explain the different types of B-splines.
6. What is Uniform Periodic B-Spline Curves?

LESSON – 14
COLOR MODELS AND COLOR APPLICATIONS

Structure of the lesson

14.1 Introduction

Basically the primary color components red, green, and blue (RGB) are used for generating displays on video monitors. Several other color descriptions are useful as well in computer-graphics applications. Some methods are used to describe color output on printers and plotters, some are used for transmitting and storing color information, and others are used to provide a more intuitive color-parameter interface to a program.

14.2 Objective of this Lesson

In this lesson we are going to learn the Properties of Light, Color Models Standard Primaries and the Chromaticity Diagram. We will also discuss in detail about various color models namely RGB Color Model, YIQ Color Models, CMY and CMYK Color Models, HSV Color Model and HLS Color Model. Finally we conclude with the Color Selection procedure and Applications of color model.

14.3 Properties of Light

Light exhibits many different characteristics, and we describe the properties of light in different ways in different contexts. Physically, we can characterize light as radiant energy, but we also need other concepts to describe our perception of light.

The Electromagnetic Spectrum

In physical terms, color is electromagnetic radiation within a narrow frequency band. Some of the other frequency groups in the electromagnetic spectrum are referred to as radio waves, microwaves, infrared waves, and X-rays. Figure 1 shows the approximate frequency ranges for these various aspects of electromagnetic radiation.

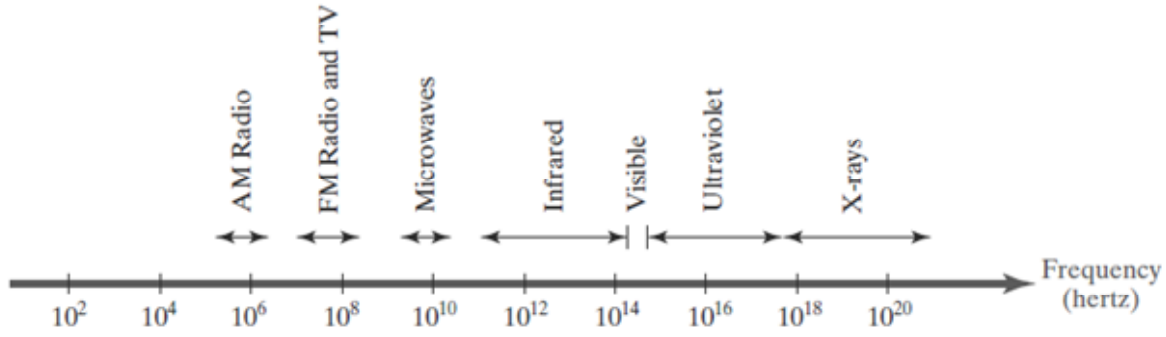


Figure 1: Electromagnetic spectrum.

Each frequency value within the visible region of the electromagnetic spectrum corresponds to a distinct spectral color. At the low-frequency end (approximately 3.8×1014 hertz) are the red colors, and at the high-frequency end (approximately 7.9 × 1014 hertz) are the violet colors.

Actually, the human eye is sensitive to some frequencies into the infrared and ultraviolet bands. Spectral colors range from shades of red through orange and yellow, at the low-frequency end, to shades of green, blue, and violet at the high end.

In the wave model of electromagnetic radiation, light can be described as oscillating transverse electric and magnetic fields propagating through space. The electric and magnetic fields are oscillating in directions that are perpendicular to each other and to the direction of propagation. For each spectral color, the rate of oscillation of the field magnitude is given by the frequency *f*. Figure 2 illustrates the time-varying oscillations for the magnitude of the electric field within one plane. The time between any two consecutive positions on the wave that have the same amplitude is called the period (T) of the wave, which is the inverse of the frequency (i.e., T =1/ f). And the distance that the wave has traveled from the beginning of one oscillation to the beginning of the next oscillation is called the wavelength (λ). For one spectral color (a monochromatic wave), the wavelength and frequency are inversely proportional to each other, with the proportionality constant as the speed of light (c): $c = \lambda f$

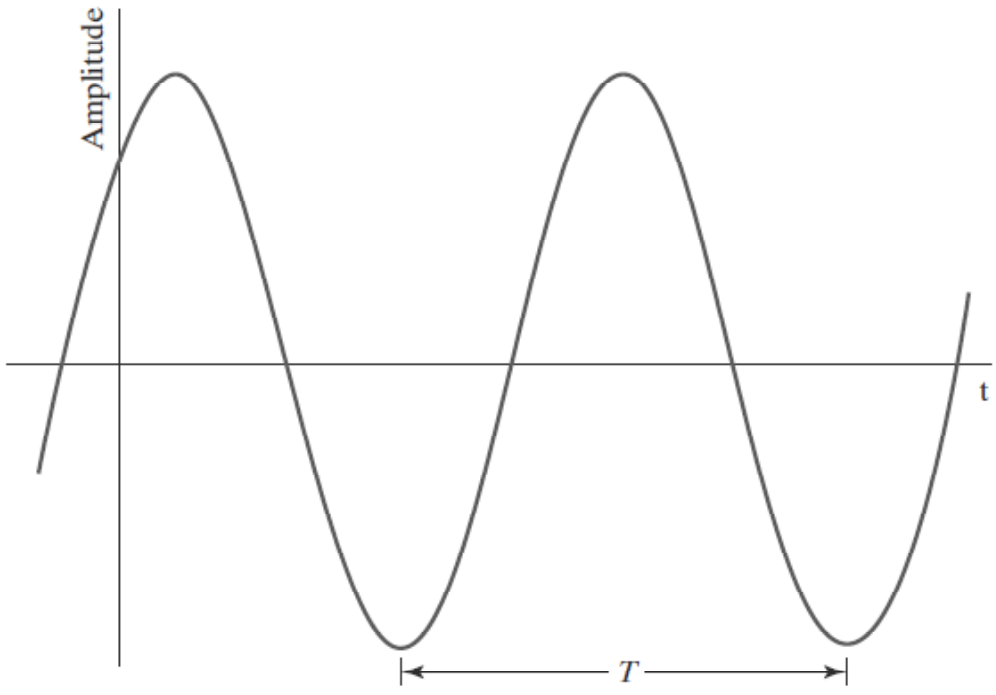


Figure 2: Time variations for the amplitude of the electric field for one frequency component of a plane-polarized electromagnetic wave.

Frequency for each spectral color is a constant for all materials, but the speed of light and the wavelength are material dependent. In a vacuum, the speed of light is very nearly c =3×1010 cm/sec. Light wavelengths are very small, so length units for designating spectral colors are usually given in angstroms (1 Å =10-8 cm) or in nanometers (1nm = 10-7 cm). An equivalent term for nanometer is millimicron. Light at the low-frequency end of the spectrum (red) has a wavelength of approximately 780 nanometers (nm), and the wavelength at the other end of the spectrum (violet) is about 380 nm. Because wavelength units are somewhat more convenient to deal with than frequency units, spectral colors are typically specified in terms of the wavelength values in a vacuum.

A light source such as the sun or a standard household light bulb emits all frequencies within the visible range to produce white light. When white light is incident upon an opaque object, some frequencies are reflected and some are absorbed. The combination of frequencies present in the reflected light determines what we perceive as the color of the object. If low frequencies are predominant in the reflected light, the object is described as red. In this case, we say that the perceived light has a dominant frequency (or dominant wavelength) at the red end of the spectrum. The dominant frequency is also called the hue, or simply the color, of the light.

Characteristics of Color

Other properties besides frequency are needed to characterize our perception of light. When we view a source of light, our eyes respond to the color (or dominant frequency) and two other basic sensations. One of these we call the brightness, which corresponds to the total light energy and can be quantified as the luminance of the light. The third perceived characteristic is called the purity, or the saturation, of the light. Purity describes how close a light appears to be to a pure spectral color, such as red. Pastels and pale colors have low purity (low saturation) and they appear to be nearly white. Another term, chromaticity, is used to refer collectively to the two properties describing color characteristics: purity and dominant frequency (hue).

14.4 Color Models

Any method for explaining the properties or behavior of color within some particular context is called a color model. No single model can explain all aspects of color, so we make use of different models to help describe different color characteristics.

Primary Colors

When we combine the light from two or more sources with different dominant frequencies, we can vary the amount (intensity) of light from each source to generate a range of additional colors. This represents one method for forming a color model. The hues that we choose for the sources are called the primary colors, and the color gamut for the model is the set of all colors that we can produce from the primary colors. Two primaries that produce white are referred to as complementary colors. Examples of complementary color pairs are red and cyan, green and magenta, and blue and yellow.

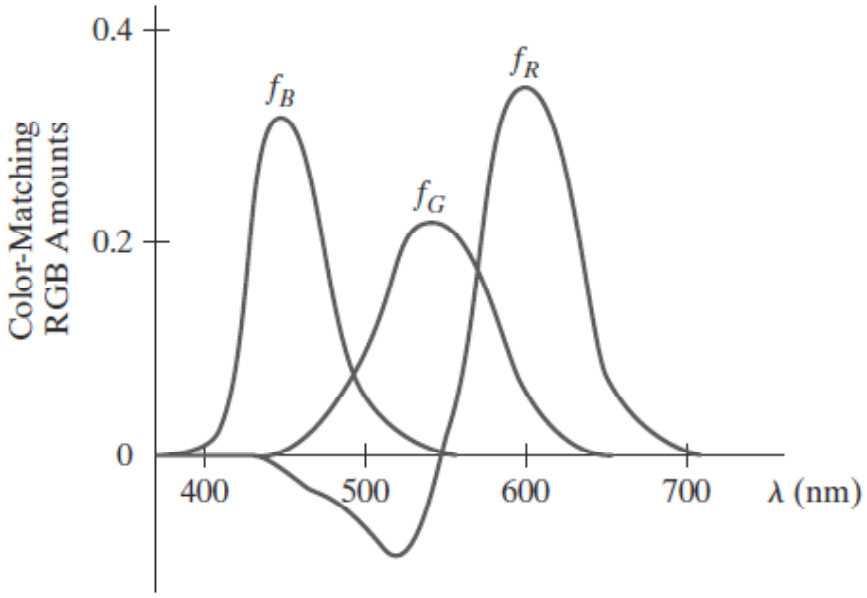


Figure 3: Three color-matching functions for displaying spectral frequencies within the approximate range from 400 nm to 700 nm.

No finite set of real primary colors can be combined to produce all possible visible colors. Nevertheless, three primaries are sufficient for most purposes, and colors not in the color gamut for a specified set of primaries can still be described using extended methods. Given a set of three primary colors, we can characterize any fourth color using color-mixing processes. Thus, a mixture of one or two of the primaries with the fourth color can be used to match some combination of the remaining primaries. In this extended sense, a set of three primary colors can be considered to describe all colors. Figure 3 shows a set of color-matching functions for three primaries and the amount of each needed to produce any spectral color.

The curves plotted in Figure 3 were obtained by averaging the judgments of a large number of observers. Colors in the vicinity of 500 nm can be matched only by "subtracting" an amount of red light from a combination of blue and green lights. This means that a color around 500 nm is described only by combining that color with an amount of red light to produce the blue-green combination specified in the diagram. Thus, an RGB color monitor cannot display colors in the neighborhood of 500 nm.

Intuitive Color Concepts

An artist creates a color painting by mixing color pigments with white and black pigments to form the various shades, tints, and tones in the scene. Starting with the pigment for a "pure color" ("pure hue"), the artist adds a black pigment to produce different shades of that color. The more black pigment, the darker the shade. Similarly, different tints of the color are obtained by adding a white pigment to the original color, making it lighter as more white is added. Tones of the color are produced by adding both black and white pigments.

14.5 Standard Primaries and the Chromaticity Diagram

Because no finite set of light sources can be combined to display all possible colors, three standard primaries were defined in 1931 by the International Commission on Illumination, referred to as the CIE (Commission Internationale de l' Eclairage). The three standard primaries are imaginary colors. They are defined mathematically with positive color-matching functions that specify the amount of each primary needed to describe any spectral color. This provides an international standard definition for all colors, and the CIE primaries eliminate negative-value color-matching and other problems associated with selecting a set of real primaries.

The XYZ Color Model

The set of CIE primaries is generally referred to as the XYZ color model, where parameters X, Y, and Z represent the amount of each CIE primary needed to produce a selected color. Thus, a color is described with the XYZ model in the same way that we described a color using the RGB model.

In the three-dimensional XYZ color space, we represent any color $C(\lambda)$ as

$$C(\lambda) = (X,\ Y,\ Z)$$

where X, Y, and Z are calculated from the color-matching functions:

$$X = k \int_{\text{visible } \lambda} f_X(\lambda)\, I(\lambda)\, d\lambda$$

$$Y = k \int_{\text{visible } \lambda} f_Y(\lambda)\, I(\lambda)\, d\lambda$$

$$Z = k \int_{\text{visible } \lambda} f_Z(\lambda)\, I(\lambda)\, d\lambda$$

Parameter k in these calculations has the value 683 lumens/watt, where lumen is a unit of measure for light radiation per unit solid angle from a "standard" point light source (once called a candle). The function $I(\lambda)$ represents the spectral radiance, which is the selected light intensity in a particular direction, and the color-matching function f_Y is chosen so that parameter Y is the luminance for that color. Luminance values are normally adjusted to the range from 0 to 100.0, where 100.0 represents the luminance of white light.

Any color can be represented in the XYZ color space as an additive combination of the primaries using unit vectors X, Y, Z. Thus, we can write it as

$$C(\lambda) = X\mathbf{X} + Y\mathbf{X} + Z\mathbf{X}$$

Normalized XYZ Values

In discussing color properties, it is convenient to normalize the amounts in color-matching functions against the sum X + Y + Z, which represents the total light energy. Normalized amounts are thus calculated as

$$x = \frac{X}{X+Y+Z}, \qquad y = \frac{Y}{X+Y+Z}, \qquad z = \frac{Z}{X+Y+Z}$$

Because x+y+z = 1, any color can be represented with just the x and y amounts. Also, we have normalized against total energy, so parameters x and y depend only on hue and purity and are called the chromaticity values. However, the x and y values alone do not allow us to describe all properties of the color completely, and we cannot obtain the amounts X, Y, and Z. Therefore, a complete description of a color is typically given with three values: x, y, and the luminance Y. The remaining CIE amounts are then calculated as

$$X = \frac{x}{y}Y, \qquad Z = \frac{z}{y}Y$$

where z = 1 - x - y. Using chromaticity coordinates (x, y), we can represent all colors on a two-dimensional diagram.

The CIE Chromaticity Diagram

When we plot the normalized amounts x and y for colors in the visible spectrum, we obtain the tongue-shaped curve shown in Figure 4. This curve is called the CIE chromaticity diagram. Points along the curve are the spectral colors (pure colors).

The line joining the red and violet spectral points, referred to as the purple line, is not part of the spectrum. Interior points represent all possible visible color combinations. Point C in the diagram corresponds to the white-light position. Actually, this point is plotted for a white light source known as illuminant C, which is used as a standard approximation for average daylight.

Luminance values are not available in the chromaticity diagram because of normalization. Colors with different luminance but with the same chromaticity map to the same point. The chromaticity diagram is useful for:

- Comparing color gamuts for different sets of primaries.
- Identifying complementary colors.
- Determining purity and dominant wavelength for a given color.

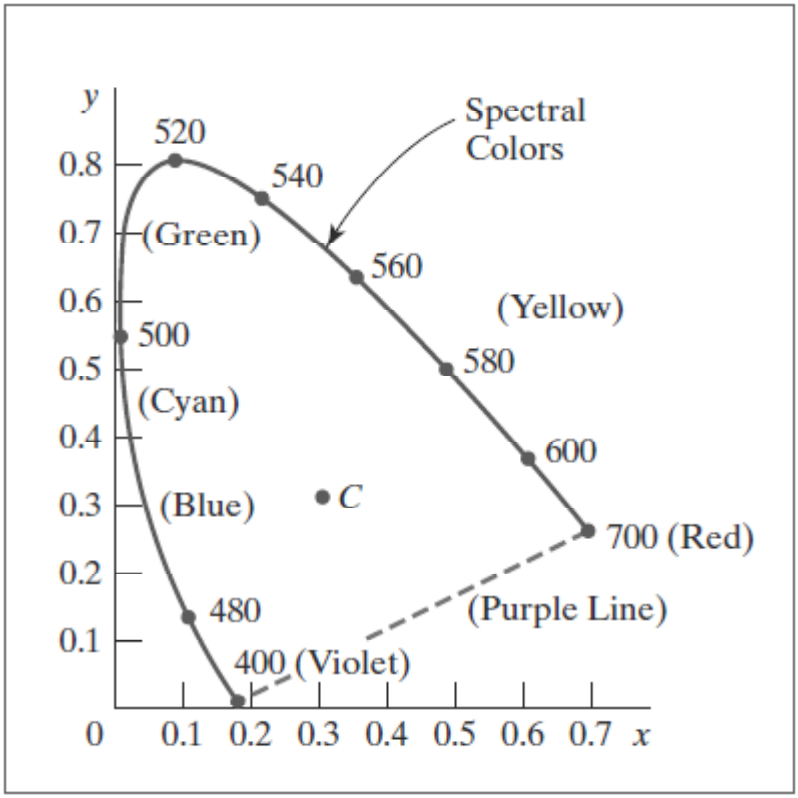


Figure 4: CIE chromaticity diagram for the spectral colors from 400 nm to 700 nm.

Color Gamuts

We identify color gamuts on the chromaticity diagram as straight-line segments or polygon regions. All colors along the straight line joining points C_1 and C_2 in Figure 5 can be obtained by mixing appropriate amounts of the colors C_1 and C_2. If a greater proportion of C_1 is used, the resultant color is closer to C_1 than to C_2 . The color gamut for three points, such C_3, C_4, and C_5 in Figure 5, is a triangle with vertices at the three color positions. These three primaries can generate only the colors inside or on the bounding edges of the triangle. Thus, the chromaticity diagram helps us to understand why no set of three primaries can be additively combined to generate all colors, because no triangle within the diagram can encompass all colors. Color gamuts for video monitors and hard-copy devices are compared conveniently on the chromaticity diagram.

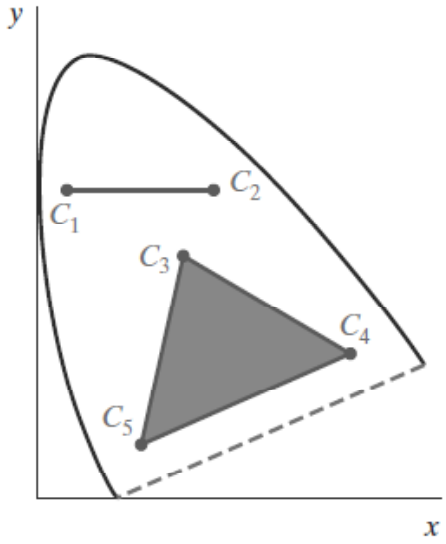


Figure 5: Color gamuts defined on the chromaticity diagram for a two-color and a three-color system of primaries.

Complementary Colors

Because the color gamut for two points is a straight line, complementary colors must be represented on the chromaticity diagram as two points on opposite sides of C and collinear with C, as in Figure 6. The distances of the two colors $C_1$and C_2 to C determine the amounts of each needed to produce white light.

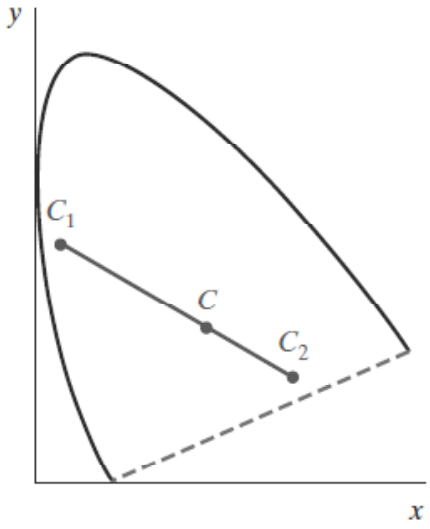


Figure 6: Representing complementary colors on the chromaticity diagram.

Dominant Wavelength

To determine the dominant wavelength of a color, we draw a straight line from C through that color point to a spectral color on the chromaticity curve. The spectral color Cs in Figure 7 is the dominant wavelength for color C_1 in this diagram. Thus, color C_1 can be represented as a combination of white light C and the spectral color C_s . This method for determining dominant wavelength will not work for color points that are between C and the purple line. Drawing a line from C through point C_2 in Figure 7 takes us to point C_p on the purple line, which is not in the visible spectrum. In this case, we take the compliment of C_p on the spectral curve, which is the point C_{sp}, as the dominant wavelength. Colors such as C_2 in this diagram have spectral distributions with subtractive dominant wavelengths. We can describe such colors by subtracting the spectral dominant wavelength from white light.

Purity

For a color point such as C_1 in Figure 7, we determine the purity as the relative distance of C_1 from C along the straight line joining C to C_s. If d_{c1} denotes the distance from C to C_1 and d_{cs} is the distance from C to C_s , we can represent purity as the ratio d_{c1}/d_{cs} . Color C_1 in this figure is about 25 percent pure, because it is situated at about one-fourth the total distance from C to C_s. At position C_s, the color point would be 100 percent pure.

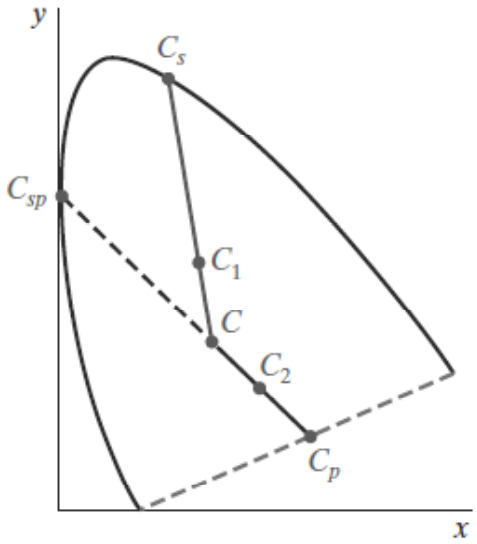


Figure 7: Determining dominant wavelength and purity using the chromaticity diagram.

14.6 The RGB Color Model

According to the tristimulus theory of vision, our eyes perceive color through the stimulation of three visual pigments in the cones of the retina. One of the pigments is most sensitive to light with a wavelength of about 630 nm (red), another has its peak sensitivity at about 530 nm (green), and the third pigment is most receptive to light with a wavelength of about 450 nm (blue). By comparing intensities in a light source, we perceive the color of the light. This theory of vision is the basis for displaying color output on a video monitor using the three primaries red, green, and blue, which is referred to as the RGB color model. We can represent this model using the unit cube defined on R, G, and B axes, as shown in Figure 8. The origin represents black and the diagonally opposite vertex, with coordinates (1, 1, 1), is white.Vertices of the cube on the axes represent the primary colors, and the remaining vertices are the complementary color points for each of the primary colors. As with the XYZ color system, the RGB color scheme is an additive model.

Each color point within the unit cube can be represented as a weighted vector sum of the primary colors, using unit vectors R, G, and B:

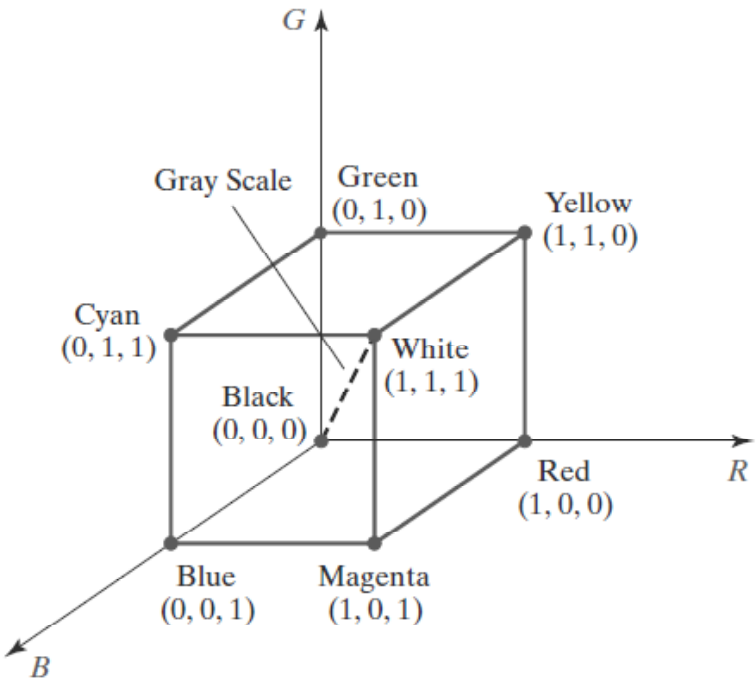


Figure 8: The RGB color model.

Table 1:

RGB (x, y) Chromaticity Coordinates

	NTSC Standard	CIE Model	Approx. Color Monitor Values
R	(0.670, 0.330)	(0.735, 0.265)	(0.628, 0.346)
G	(0.210, 0.710)	(0.274, 0.717)	(0.268, 0.588)
B	(0.140, 0.080)	(0.167, 0.009)	(0.150, 0.070)

where parameters R, G, and B are assigned values in the range from 0 to 1.0. For example, the magenta vertex is obtained by adding maximum red and blue values to produce the triple (1, 0, 1), and white at (1, 1, 1) is the sum of the maximum values for red, green, and blue. Shades of gray are represented along the main diagonal of the cube from the origin (black) to the white vertex. Points along this diagonal have equal contributions from each primary color, and a gray shade halfway between black and white is represented as (0.5, 0.5, 0.5). Chromaticity coordinates for the National Television System Committee (NTSC) standard RGB phosphors are listed in Table 1. Also listed are the RGB chromaticity coordinates within the CIE color model and the approximate values used for phosphors in color monitors. Figure 9 shows the approximate color gamut for the NTSC standard RGB primaries.

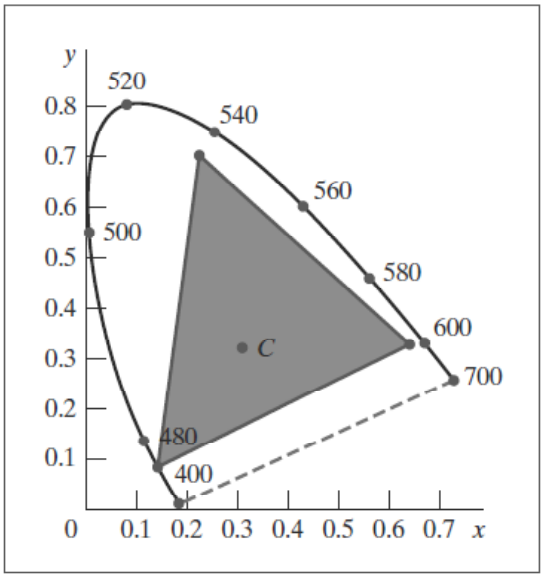


Figure 9: The RGB color gamut for NTSC chromaticity coordinates.

14.7 The YIQ and Related Color Models

In the YIQ color model, parameter Y is the same as the Y component in the CIE XYZ color space. Luminance (brightness) information is conveyed by the Y parameter, while chromaticity information (hue and purity) is incorporated into the I and Q parameters. A combination of red, green, and blue is chosen for the Y parameter to yield the standard luminosity curve. Because Y contains the luminance information, black-and-white television monitors use only the Y signal. Parameter I contains orange-cyan color information that provides the flesh-tone shading, and parameter Q carries green-magenta color information.

The NTSC composite color signal is designed to provide information in a form that can be received by black-and-white television monitors, which obtain grayscale information for a picture within a 6-MHz bandwidth. Thus, the YIQ information is also encoded within a 6-MHz bandwidth, but the luminance and chromaticity values are encoded on separate analog signals. In this way, the luminance signal is unchanged for black-and-white monitors, and the color information is simply added within the same bandwidth. Luminance information, the Y value, is conveyed as an amplitude modulation on a carrier signal with a bandwidth of about 4.2 MHz. Chromaticity information, the I and Q values, is combined on a second carrier signal that has a bandwidth of about 1.8 MHz. The parameter names I and Q refer to the modulation methods used to encode the color information on this carrier. An amplitude-modulation encoding (the “in-phase” signal) transmits the I value, using about 1.3 MHz of the bandwidth. And a phase-modulation encoding (the “quadrature” signal), using about 0.5 MHz, carries the Q value.

Luminance values are encoded at a higher precision in the NTSC signal (4.2 MHz bandwidth) than the chromaticity values (1.8 MHz bandwidth), because we can detect small brightness changes more easily compared to small color changes. However, the lower precision for the chromaticity encoding does result in some degradation of the color quality for an NTSC picture.

We can calculate the luminance value for an RGB color. One method for producing chromaticity values is to subtract the luminance from the red and blue components of the color. Thus,

$$Y = 0.299\,R + 0.587\,G + 0.114\,B$$
$$I = R - Y$$
$$Q = B - Y$$

Transformations Between RGB and YIQ Color Spaces

An RGB color is converted to a set of YIQ values using an NTSC encoder that implements the calculations and modulates the carrier signals.

The conversion from RGB space to YIQ space is accomplished using the following transformation matrix:

$$\begin{bmatrix} Y \\ I \\ Q \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ 0.701 & -0.587 & -0.114 \\ -0.299 & -0.587 & 0.886 \end{bmatrix} \cdot \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

Conversely, an NTSC video signal is converted to RGB color values using an NTSC decoder, which first separates the video signal into the YIQ components, and then converts the YIQ values to RGB values. The conversion from YIQ space to RGB space is accomplished with the inverse of transformation matrix:

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1.000 & 1.000 & 0.000 \\ 1.000 & -0.509 & -0.194 \\ 1.000 & 0.000 & 1.000 \end{bmatrix} \cdot \begin{bmatrix} Y \\ I \\ Q \end{bmatrix}$$

14.8 The CMY and CMYK Color Models

A video monitor displays color patterns by combining light that is emitted from the screen phosphors, which is an additive process. However, hard-copy devices, such as printers and plotters, produce a color picture by coating a paper with color pigments. We see the color patterns on the paper by reflected light, which is a subtractive process.

The CMY Parameters

A subtractive color model can be formed with the three primary colors cyan, magenta, and yellow. As we have noted, cyan can be described as a combination of green and blue. Therefore, when white light is reflected from cyan colored ink, the reflected light contains only the green and blue components, and the red component is absorbed, or subtracted, by the ink. Similarly, magenta ink subtracts the green component from incident light, and yellow subtracts the blue component. A unit cube representation for the CMY model is illustrated in Figure 10.

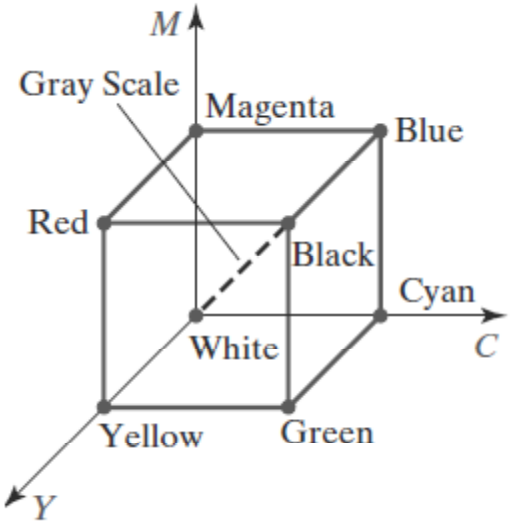


Figure 10: The CMY color model.

In the CMY model, the spatial position (1, 1, 1) represents black, because all components of the incident light are subtracted. The origin represents white light. Equal amounts of each of the primary colors produce shades of gray along the main diagonal of the cube. A combination of cyan and magenta ink produces blue light, because the red and green components of the incident light are absorbed. Similarly, a combination of cyan and yellow ink produces green light, and a combination of magenta and yellow ink yields red light.

The CMY printing process often uses a collection of four ink dots, which are arranged in a close pattern somewhat as an RGB monitor uses three phosphor dots. Thus, in practice, the CMY color model is referred to as the CMYK model, where K is the black color parameter. One ink dot is used for each of the primary colors (cyan, magenta, and yellow), and one ink dot is black. A black dot is included because reflected light from the cyan, magenta, and yellow inks typically produce only shades of gray. Some plotters produce different color combinations

Transformations Between CMY and RGB Color Spaces

We can express the conversion from an RGB representation to a CMY representation using the following matrix transformation:

$$\begin{bmatrix} C \\ M \\ Y \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

where the white point in RGB space is represented as the unit column vector. And we convert from a CMY color representation to an RGB representation using the matrix transformation

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} C \\ M \\ Y \end{bmatrix}$$

In this transformation, the unit column vector represents the black point in the CMY color space. For the conversion from RGB to the CMYK color space, we first set K = max(R, G, B). Then K is subtracted from each of C, M, and Y. Similarly, for the transformation from CMYK to RGB, we first set K =min(R,G, B). Then K is subtracted from each of R, G, and B. In practice, these transformation equations are often modified to improve the printing quality for a particular system.

14.9 The HSV Color Model

Interfaces for selecting colors often use a color model based on intuitive concepts, rather than a set of primary colors. We can give a color specification in an intuitive model by selecting a spectral color and the amounts of white and black that are to be added to that color to obtain different shades, tints, and tones.

The HSV Parameters

Color parameters in this model are called hue (H), saturation (S), and value (V). We derive this three-dimensional color space by relating the HSV parameters to the directions in the RGB cube. If we imagine viewing the cube along the diagonal from the white vertex to the origin (black), we see an outline of the cube that has the hexagon shape shown in Figure 11. The boundary of the hexagon represents the various hues, and it is used as the top of the HSV hexcone (Figure 12).

In HSV space, saturation S is measured along a horizontal axis, and the value parameter V is measured along a vertical axis through the center of the hexcone. Hue is represented as an angle about the vertical axis, ranging from 0^o at red through 360^o. Vertices of the hexagon are separated by 60^o intervals. Yellow is at 60^o, green at 120^o, and cyan (opposite the red point) is at $H = 180^o$. Complementary colors are 180^o apart. Saturation parameter S is used to designate the purity of a color. A pure color (spectral color) has the value S=1.0, and decreasing S values tend toward the grayscale line (S = 0) at the center of the hexcone.

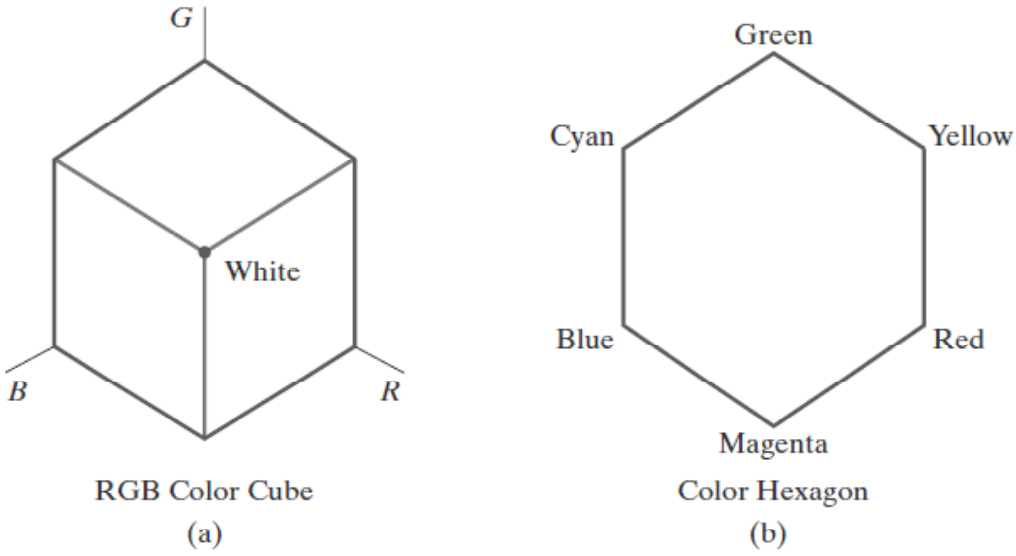


Figure 11: When the RGB color cube (a) is viewed along the diagonal from white to black, the color-cube outline is a hexagon (b).

Value V varies from 0 at the apex of the hexcone to 1.0 at the top plane. The apex of the hexcone is the black point. At the top plane, colors have their maximum intensity. When V = 1.0 and S = 1.0, we have the pure hues. Parameter values for the white point are V = 1.0 and S = 0.

For most users, this is a more convenient model for selecting colors. Starting with a selection for a pure hue, which specifies the hue angle H and sets V = S = 1.0, we describe the color we want in terms of adding either white or black to the pure hue. Adding black decreases the setting for V while S is held constant. To get a dark blue, for instance, V could be set to 0.4 with S = 1.0 and H = 240°.

Similarly, when white is to be added to the selected hue, parameter S is decreased while keeping V constant. A light blue could be designated with S = 0.3 while V = 1.0 and H = 240°. By adding some black and some white, we decrease both V and S. An interface for this model typically presents the HSV parameter choices in a color palette containing sliders and a color wheel.

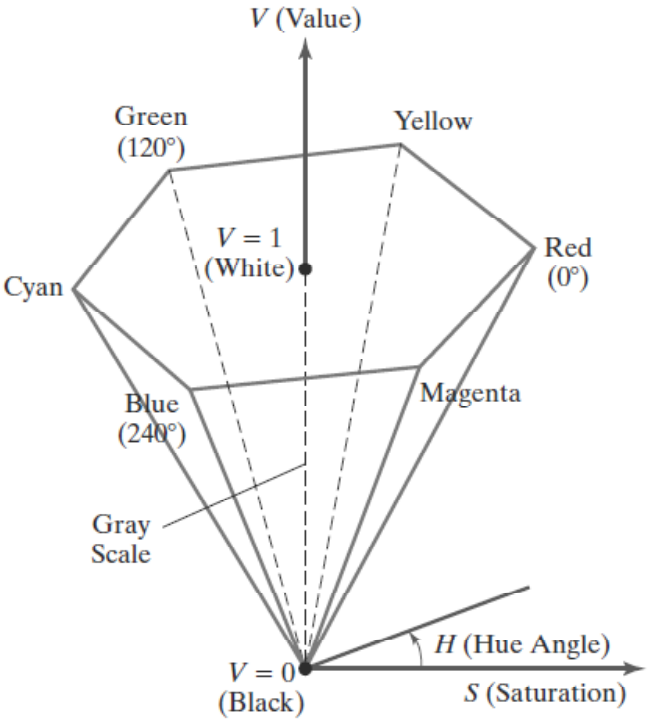


Figure 12: The HSV hexcone.

Selecting Shades, Tints, and Tones

Color regions for selecting shades, tints, and tones are represented in the cross sectional plane of the HSV hexcone shown in Figure 12. Adding black to a spectral color decreases V along the side of the hexcone toward the black point. Thus, various shades are represented with the values S=1.0 and $0.0 \leq V \leq 1.0$. Adding white to spectral colors produces the tints across the top plane of the hexcone, where parameter values are V = 1.0 and $0 \leq S \leq 1.0$. Various tones are obtained by adding both black and white to spectral colors, which generates color points within the triangular cross-sectional area of the hexcone. The human eye can distinguish about 128 different hues and about 130 different tints (saturation levels). For each of these, a number of shades (value settings) can be detected, depending on the hue selected. About 23 shades are discernible with yellow colors, and about 16 different shades can be seen at the blue end of the spectrum. This means that we can distinguish about 128×130×23 = 382,720 different colors. For most graphics applications, 128 hues, 8 saturation levels, and 16 value settings are sufficient. With this range of parameters in the HSV color model, 16,384 colors are available to a user. These color values can be stored in 14 bits per pixel, or we could use color-lookup tables and fewer bits per pixel.

Transformations between HSV and RGB Color Spaces

To determine the operations required for the transformations between the HSV and RGB spaces, we first consider how the HSV hexcone can be constructed from the RGB cube. The diagonal of the RGB cube from black (the origin) to white corresponds to the V axis of the hexcone. Also, each subcube of the RGB cube corresponds to a hexagonal cross-sectional area of the hexcone. At any cross section, all sides of the hexagon and all radial lines from the V axis to any vertex have the value V. Thus, for any set of RGB values, V is equal to the value of the maximum RGB component. The HSV point corresponding to this set of RGB values lies on the hexagonal cross section at value V. Parameter S is then determined as the relative distance of this point from the V axis. Parameter H is determined by calculating the relative position of the point within each sextant of the hexagon.

14.10 The HLS Color Model

Another model based on intuitive color parameters is the HLS system used by the Tektronix Corporation. This color space has the double-cone representation shown in Figure 13. The three parameters in this color model are called hue (H), lightness (L), and saturation (S). Hue has the same meaning as in the HSV model. It specifies an angle about the vertical axis that locates a hue (spectral color). In this model, H = 0° corresponds to blue. The remaining colors are specified around the perimeter of the cone in the same order as in the HSV model. Magenta is located at H =60°, red is at H =120°, and cyan is at H =300°. Again, complementary colors are 180° apart on the double cone.

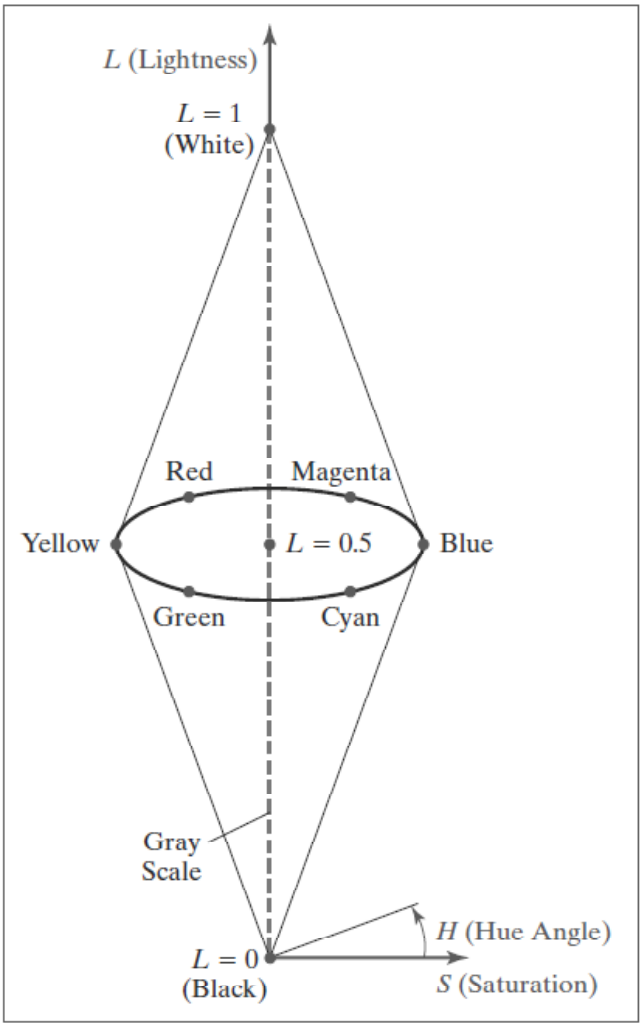


Figure 13: The HLS double cone.

The vertical axis in this model is called lightness, L. At L = 0, we have black, and at L = 1.0, we have white. Grayscale values are along the L axis, and the pure colors lie on the L = 0.5 plane. Saturation parameter S again specifies the purity of a color. This parameter varies from 0 to 1.0, and pure colors are those for which S = 1.0 and L = 0.5. As S decreases, more white is added to a color. The grayscale line is at S = 0.

To specify a color, we begin by selecting hue angle H. Then a particular shade, tint, or tone for that hue is obtained by adjusting parameters L and S. We obtain a lighter color by increasing L, and we obtain a darker color by decreasing L. When S is decreased, the spatial color point moves toward the grayscale line.

14.11 Color Selection and Applications

A graphics package can provide color capabilities in a way that aids us in making color selections. For example, an interface can contain sliders and color wheels instead of requiring that all color specifications be provided as numerical values for the RGB components. In addition, some aids can be provided for choosing harmonious color combinations and for basic color selection guidelines.

One method for obtaining a set of coordinating colors is to generate the color combinations from a small subspace of a color model. If colors are selected at regular intervals along any straight line within the RGB or CMY cube, for example, we can expect to obtain a set of well-matched colors. Randomly selected hues can be expected to produce harsh and clashing color combinations. Another consideration in color displays is the fact that we perceive colors at different depths. This occurs because our eyes focus on colors according to their frequency. Blues, in particular, tend to recede. Displaying a blue pattern next to a red pattern can cause eye fatigue, because we continually need to refocus when our attention is switched from one area to the other. This problem can be reduced by separating these colors or by using colors from one-half or less of the color hexagon in the HSV model. With this technique, a display contains either blues and greens or reds and yellows.

As a general rule, the use of a smaller number of colors produces a better looking display than one with a large number of colors. Also, tints and shades tend to blend better than the pure

hues. For a background, gray or the complement of one of the foreground colors is usually best.

14.12 Summary

Light can be described as electromagnetic radiation.

We quantify our perceptions of a light source using terms such as dominant frequency (hue), luminance (brightness), and purity (saturation).

Hue and purity are referred to collectively as the chromaticity properties of a color.

The set of colors that can be generated by a set of primaries is called a color gamut.

Two colors that combine to produce white light are called complementary colors.

In 1931, the International Commission on Illumination (CIE) adopted a set of three hypothetical color-matching functions as a standard. This set of colors is referred to as the XYZ model.

Other color models based on a set of three primaries are the RGB, YIQ, and CMY models.

RGB model to describe colors that are displayed on a video monitor.

The YIQ model is used to describe the composite video signal for television broadcasting.

The CMY model is used to describe color on hard-copy devices.

The HSV and HLS models, specify a color as a mixture of a selected hue and certain amounts of white and black.

Adding black produces color shades, adding white produces tints, and adding both black and white produces tones.

Color selection is an important factor in the design of effective displays

As a general rule, a small number of color combinations formed with tints and shades, rather than pure hues, results in a more harmonious color display.

14.13 Model Question

1. What is hue?
2. What are color gamuts?
3. Define the term chromaticity.
4. Define complementary colors.
5. Define primary colors.
6. State the use of chromaticity diagram.
7. Explain in detail about XYZ color model.
8. Explain RGB color model.
9. Explain in detail about YIQ color model.
10. Explain in detail about CMY and CMYK Color Models.
11. Explain HSV color model and its properties.
12. Explain HLS color model.

MODEL QUESTION PAPER

MASTER OF COMPUTER APPLICATIONS

SECOND YEAR - THIRD SEMESTER

CORE PAPER - XII

COMPUTER GRAPHICS

Time: 3 Hrs. Max Marks : 80

Section – A (10 x 2 = 20Marks)

Answer any TEN Questions.

Each question carries 2 Marks.

1. Define the term Computer Graphics
2. What are output primitives?
3. What is meant by transformation?
4. Define scaling in 2D object.
5. What is point clipping?
6. Differentiate translation and scaling.
7. What is reflection?
8. What is meant by visible surface detection?
9. Write down the significance of octree method.
10. What are complementary colors
11. List the applications of computer animation.
12. State any two general rules for color selection.

Section – B **(5 x 6 = 30 Marks)**

Answer any FIVE Questions.

Each question carries 6 Marks.

13. Explain how Raster Scan Systems visualize the output.

14. Describe two-dimensional rotation in detail.

15. Explain polygon clipping.

16. Write short notes on parallel projection.

17. Explain composite transformation.

18. Give a brief note on backface detection method.

19. Explain RGB color model.

Section – C **(3 x 10 = 30 Marks)**

Answer any THREE Questions

Each question carries 10 Marks.

20. Explain DDA line drawing algorithm.

21. Illustrate the procedure in window to view port mapping.

22. Explain Cohen-Sutherland line clipping algorithm.

23. Explain scan-line method for visible surface detection.

24. Describe Spline and its properties in detail.

———