UNIT - I

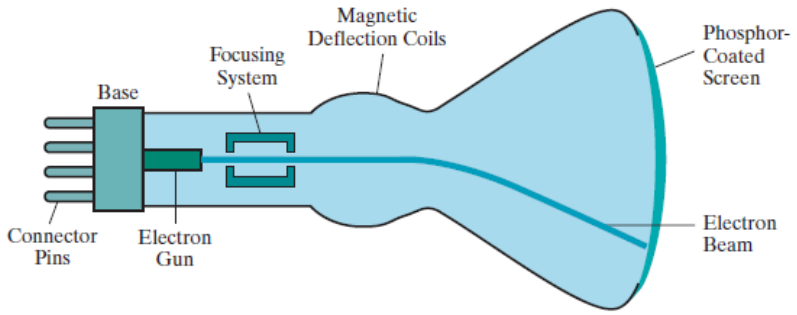
Overview of Graphics System

Video Display Devices

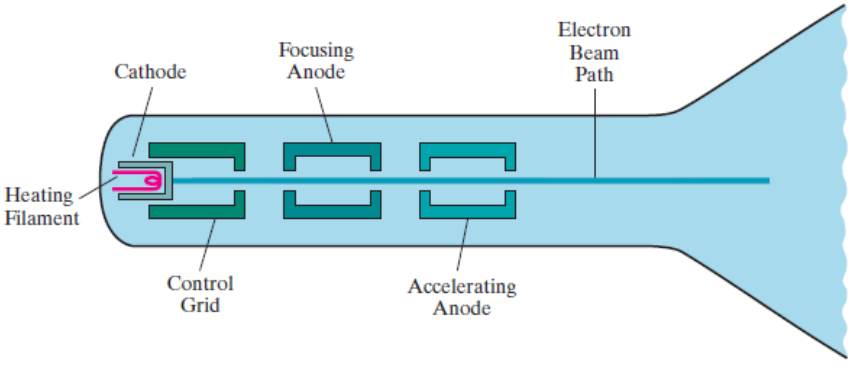
The Primary output device in Computer Graphics is a Monitor which operates on the standard cathode-ray tube(CRT) design and a few more technological hardware have also come into the concept. Computer graphics is a complex and diversified technology.

Refresh Cathode-Ray Tubes

The following figure illustrates the basic operation of how does a CRT work. An electron beam comes from the electron gun, passes through focus and deflection systems that send the beam towards directed positions on the phosphor-coated screen. The phosphor in return emits a small spot of light at each position where ever the electron beam makes contact. As the light which is emitted by the phosphor fades very easily, some mechanism is required for managing the picture on the screen. One method to make the phosphor glowing is to keep on redrawing the picture in a repeated manner by quickly projecting the electron beam over the same points again and again.



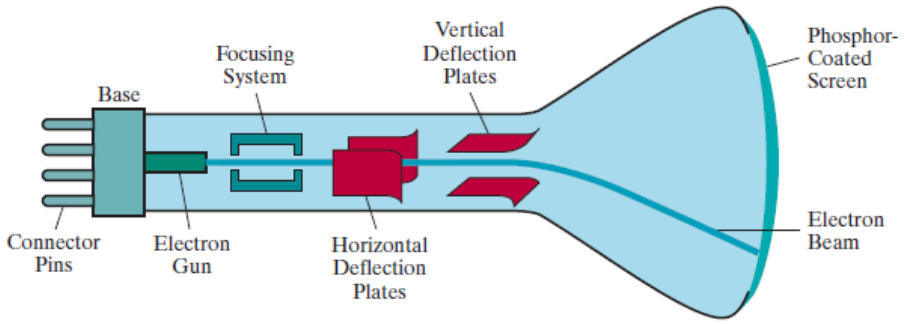
Basic design of a magnetic-deflection CRT.



Operation of an electron gun with an accelerating anode.

The heated metal cathode and a control grid are the key components of an electron gun in a CRT. Through the coil of wire, called the filament, inside the cylindrical cathode structure, heat is supplied to the cathode by directing a current which makes electrons to be 'boiled off" the hot cathode surface. The free, negatively charged electrons are then accelerated toward the phosphor coating by a high positive voltage, in the vacuum inside the CRT envelope. 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 can be used, as in Figure. Most of the times the electron gun is meant to contain the accelerating anode and focusing system within the same unit. Intensity of the electron beam is maintained by keeping voltage levels on the control grid, which is a metallic cylinder and that fits over the shape of the cathode. A high negative voltage applied to the control grid shuts off the beam as it repels electrons and stops them from passing through the small hole at the end of the control grid structure. A smaller negative voltage on the control grid all-together decreases the total number of electrons passing through it. Since the amount of light emitted by the phosphor coating depends on the number of electrons which strike the screen, the brightness of a display can be controlled by changing the voltage on the control grid. In the electron beam, electrons spread all over the screen as a result of repulsion among them. To make the electron beam strike at one point, focusing anode is present in the CRT. Hence our focusing mechanism makes the electron beam to strike the phosphor screen at a small spot and focusing is following by usage of magnetic and electric field. Magnetic deflected is carried out by using 2 pairs of magnetic coils within the CRT. One pair is on the top and down position and the other one pair is on the opposite sides of CRT as it is shown in the Figure Magnetics field thus produced by each pair creates a transverse deflection force, perpendicular to the way of magnetic field and to the direction in which the electron bean is travelling. Moreover Horizontal deflection of electron bean is accomplished by one pair of coils and vertical deflection is carried out by the others.

Electric deflection is carried out by using two pairs of deflecting plates inside CRT, the two pairs are mounted vertically and horizontally.



Electrostatic deflection of the electron beam in a CRT.

Horizontal deflecting plates provide vertical deflection to the electron beam and vertical deflecting plates provide horizontal deflection to the electron beam. Important terminologies in CRT are as follows:

Refresh rate: It denotes the number of images which are displayed every second, or we can say that it is the number of times the images is remapped per second. And It is also known as vertical scan rate and is expressed in Hertz (Hz).

Resolution : It denotes the number of pixels per surface unit and can be abbreviated as DPI or dots per inches and is calculated both vertically and horizontally. A resolution of 200dpi means that 200 columns and 200 rows of pixels per square

Size : It is calculated by taking the dimension of the diagonal of the screen and is expressed

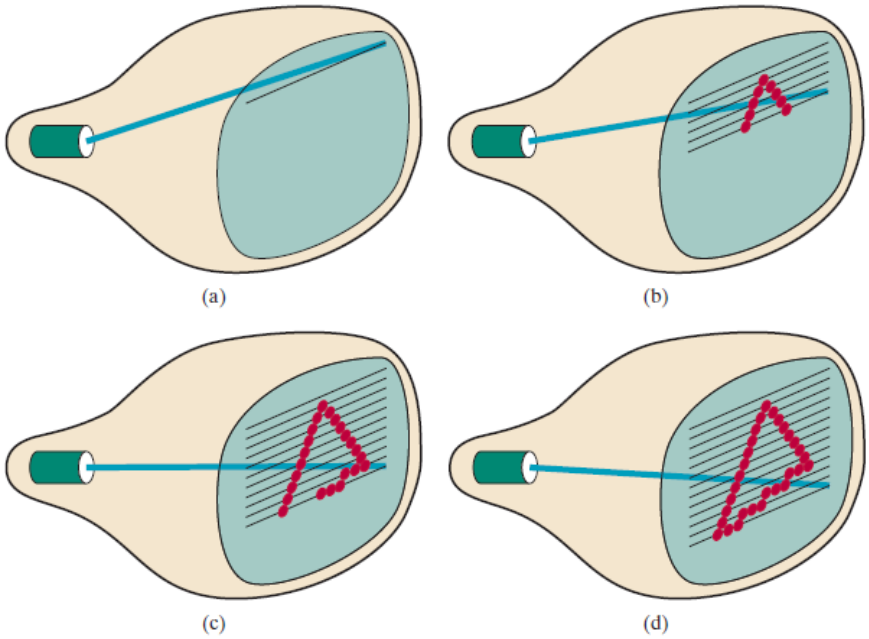
Aspect Ratio : It is termed as the ratio of vertical points to horizontal points.

Raster scan Displays

Our home television sets use Raster scan technologies. In this sort of Display Mechanism, an electron beam scans every row of the screen display row by row starting from top to the bottom. Each screen point represents the intensity value either 0 or 1 and the intensity value is kept in refresh buffer or frame buffer. Thus, each pixel value or screen point keeps on changing from 0 to 1 or from 1 to 0 depending on its intensity value in refresh buffer. And this is the way the screen is painted one row at a time. And this is shown in the Figure.

The range of the intensity depends upon the system capabilities. We can plot only two different colors or intensities if it is a black and white system. In this case one bit per pixel is enough, 1 for white intensity and bit value 0 for black intensity. More bits can be used to display color and intensity for colors. So, bitmap is the term used for frame buffer for black and white systems and Pixmap is the term which is used for Frame buffer which stores multiple bits per pixel.

For raster system the refresh rate is generally 60 to 80 frames per second, it can be higher for some systems. After it scans one row and it returns to the left of the screen for scanning next row, it is called horizontal retrace. After it has refreshed each scan line, it moves to the top left corner of the display and again starts the refreshing process and this is called vertical retrace.



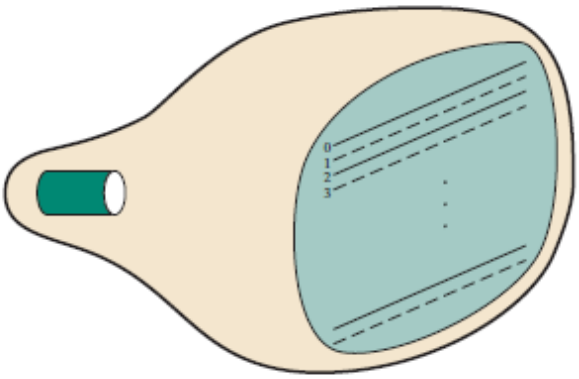
A raster-scan system displays an object as a set of discrete points across each scan line.

Raster scan systems are much more capable than the random systems. As it stores the intensity values for each screen position, it is capable of displaying the color variations and shade which is not possible with random systems. But raster system has lower resolution as compared to random system. This is because, random system follows the line path to be drawn and line drawing commands are stored in refresh buffer. For raster system, intensity values are stored for each screen.

Random Scan Displays

The arrangement of a simple random scan system is shown in the following figure. System stores and application program in the system memory along with a graphics package. With the help of graphics package the Graphics command in the application program are converted into a display file stored in the system memory. And this file helps the system to refresh the screen. When operated as a **random-scan** display unit, a CRT has the electron

beam directed only to the parts of the screen where a picture is to be drawn. Random-scan monitor draw a picture one line at a time and for this reason are also referred to as **vector** displays (or **stroke-writing** of **calligraphic** displays).

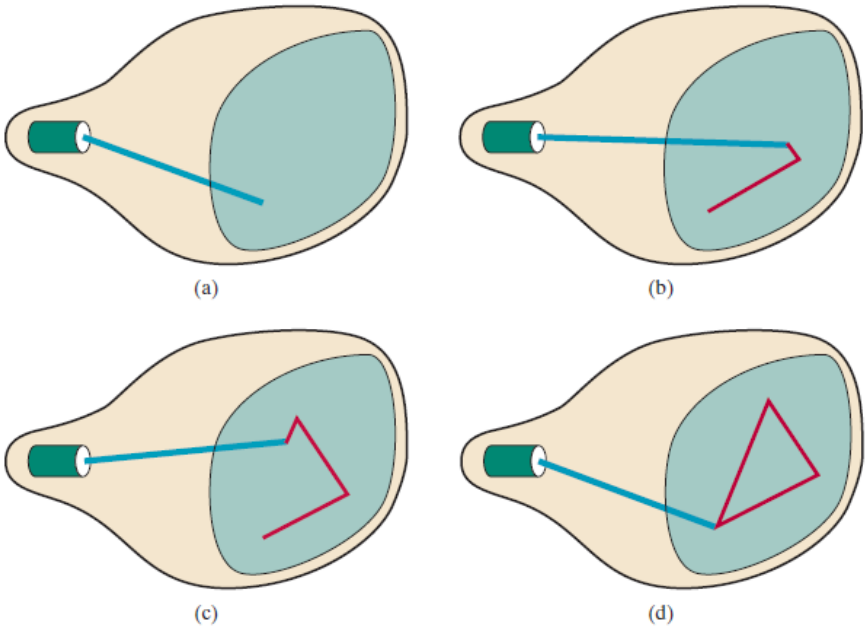


Interlacing scan lines on a raster-scan display. First, all points on the even-numbered (solid) scan lines are displayed; then all points along the odd-numbered (dashed) lines are displayed.

The component lines of a picture can be drawn and refreshed by a random-scan system in any specified order Figure. A pen plotter in a similar way and is an example of a random-scan, hard-copy device.

Refresh rate on a random-scan system depends on the number of lines to be displayed. Picture definition is now stored as a set of line-drawing commands in an area of memory 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 cycle back to the first line command in the list. Random-scan displays are designed to draw all the component lines of a picture 30 to 60 times each second.

High-quality vector systems are capable of handling approximately 100,000 “short” lines at this refresh rate. When a small set of lines is to be displayed, each refresh cycle is delayed to avoid refresh rates greater than 60 frames per second. Otherwise, faster refreshing of the set of lines could burn out the phosphor. Random-scan systems are designed for line-drawing applications and can-not display realistic shaded scenes.



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

Since picture definition is stored as a set of line-drawing instruction and not as a set of intensity values for all screen points, vector displays generally have higher resolution then raster system. Also, vector displays produce smooth line drawings because the CRT beam directly follows the line path. A raster system, in contrast, produces jagged lines that are plotted as discrete point sets.

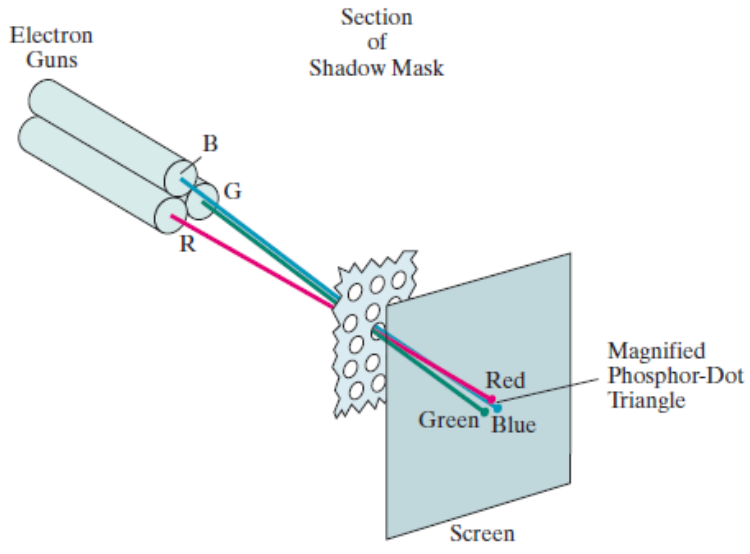
COLOR CRT

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

The **beam-penetration** method for displaying color pictures has been used with random-scan monitors. Two layers of phosphor, usually red and green, are coated onto the inside of the CRT screen, and the displayed color depends on how far the electron beam penetrates into the phosphor layers. A beam of slow electrons excites only the outer red layer. A beam of very fast electron penetrates through the red layer and excites the inner green layer. At intermediate beam speeds, combinations of red and green light are emitted to show two additional colors, orange and yellow. The speed of the electrons, and hence the screen color at any point, is controlled by the beam-

acceleration voltage. Beam penetration has been an inexpensive way to produce color in random-scan monitor, but only four colors are possible, and the quality of picture is not as good as with other methods.

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



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

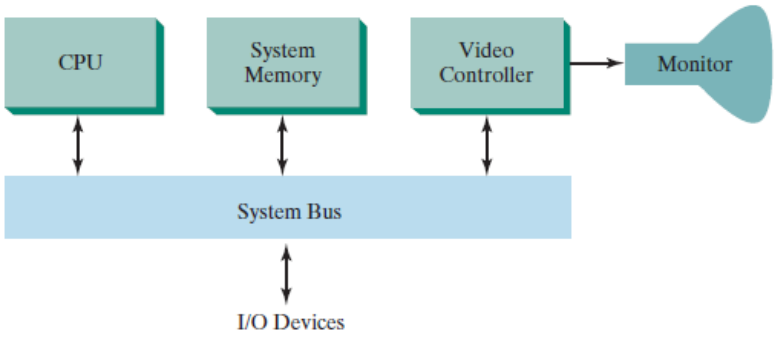
We obtain color variations in a shadow-mask CRT by varying the intensity levels of the three electron beams. By turning off the red and green guns, we get only the color coming from the blue phosphor. Other combinations of beam intensities produce a small light spot for each pixel position, since our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

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

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

RASTER-SCAN SYSTEMS

Interactive raster-graphics systems typically employ several processing units. In addition to the central processing unit, or CPU, a special-purpose processor, called the video controller or display controller, is used to control the operation of the display device. Organization of a simple raster system is shown in Figure. Here, the frame buffer can be anywhere in the system memory, and the video controller accesses the frame buffer to refresh the screen. In addition to the video controller, more sophisticated raster systems employ other processors as coprocessors and accelerators to implement various graphics operations.

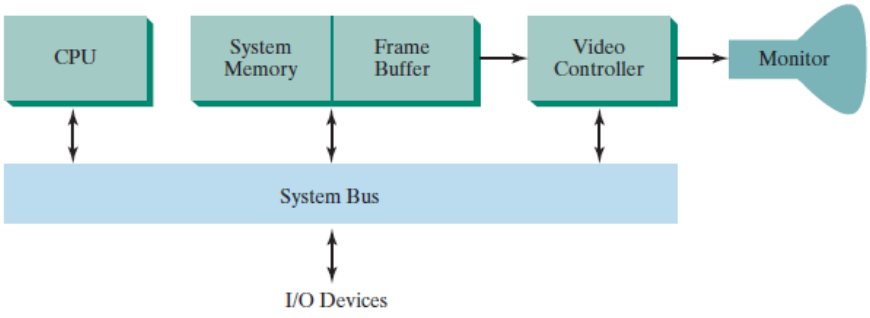


Architecture of a simple raster-graphics system.

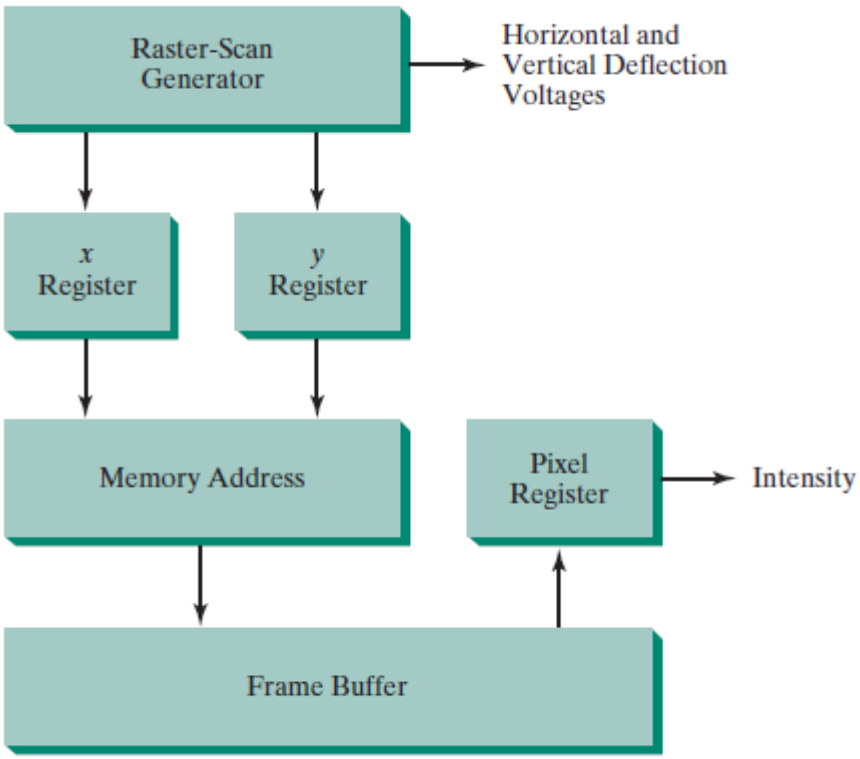
Video Controller

Figure shows a commonly used organization for raster systems. A fixed area of the system memory is reserved for the frame buffer, and the video controller is given direct access to the frame-buffer memory.

Frame-buffer locations, and the corresponding screen positions, are referenced in Cartesian coordinates.



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



Basic video-controller refresh operations.

The basic refresh operations of the video controller are diagrammed. Two registers are used to store the coordinate values for the screen pixels. Initially, the *x* register is set to 0 and the *y* register is set to the value for the top scan line. The contents of the frame buffer at this pixel position are then retrieved and used to set the intensity of the CRT beam. Then the *x* register is incremented by 1, and the process is repeated for the next pixel on the top scan line. This procedure continues for each pixel along the top scan line.

After the last pixel on the top scan line has been processed, the *x* register is reset to 0 and the *y* register is set to the value for the next scan line down from the top of the screen. Pixels along this scan line are then processed in turn, and the procedure is repeated for each successive scan line. After cycling through all pixels along the bottom scan line, the video controller resets the registers to the first pixel position on the top scan line and the refresh process starts over. Since the screen must be refreshed at a rate of at least 60 frames per second, the simple procedure illustrated in Figure may not be accommodated by typical RAM chips if the cycle time is too slow.

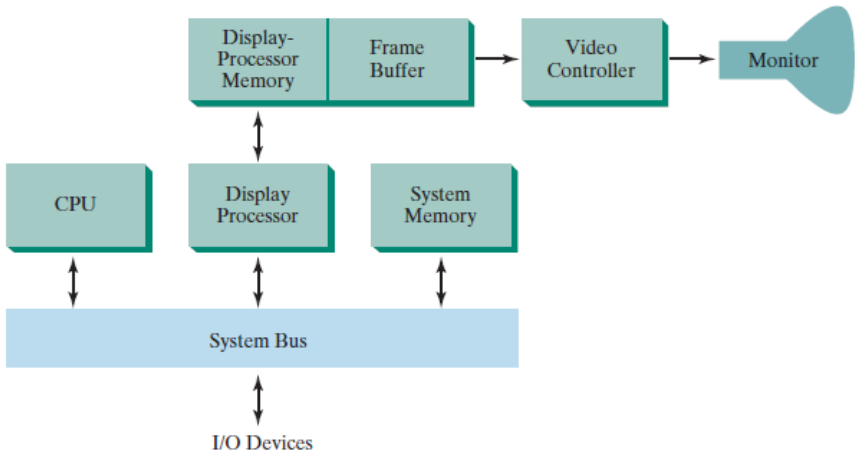
To speed up pixel processing, video controllers can retrieve multiple pixel values from the refresh buffer on each pass.

The multiple pixel intensities are then stored in a separate register and used to control the CRT beam intensity for a group of adjacent pixels. When that group of pixels has been processed, the next block of pixel values is retrieved from the frame buffer.

Raster-Scan Display Processor

Figure shows one way to organize the components of a raster system that contains a separate **display processor**, sometimes referred to as a **graphics controller** or a **display coprocessor**. The purpose of the display processor is to free the CPU from the graphics chores. In addition to the system memory, a separate display-processor memory area can be provided.

A major task of the display processor is digitizing a picture definition given in an application program into a set of pixel values for storage in the frame buffer.

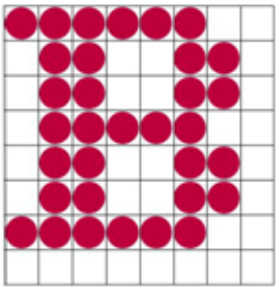


Architecture of a raster-graphics system with a display processor.

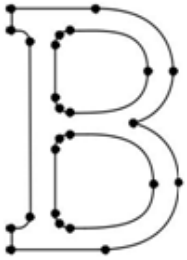
This digitization process is called **scan conversion**. Graphics commands specifying straight lines and other geometric objects are scan converted into a set of discrete points, corresponding to screen pixel positions. Scan converting a straight-line segment, for example, means that we have to locate the pixel positions closest to the line path and store the color for each position in the frame buffer. Similar methods are used for scan converting other objects in a picture definition. Characters can be defined with rectangular pixel grids, as in Figure. or they can be defined with outline shapes, as in Figure. The array size for character grids can vary from about 5 by 7 to 9 by 12 or

more for higher-quality displays. A character grid is displayed by superimposing the rectangular grid pattern into the frame buffer at a specified coordinate position. For characters that are defined as outlines, the shapes are scan converted into the frame buffer by locating the pixel positions closest to the outline.

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



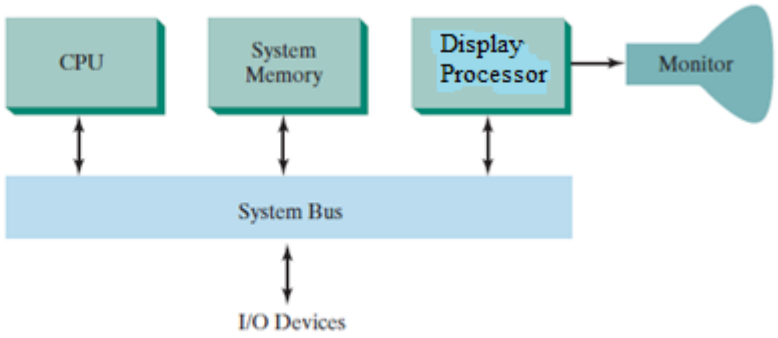
A character defined as a rectangular grid of pixel positions.



A character defined as an outline shape

RANDOM-SCAN SYSTEMS

The organization of a simple random-scan (vector) system is shown in Figure. An application program is input and stored in the system memory along with a graphics package. Graphics commands in the application program are translated by the graphics package into a display file stored in the system memory. This display file is then accessed by the display processor to refresh the screen. The display processor cycles through each command in the display file program once during every refresh cycle. Sometimes the display processor in a random-scan system is referred to as a display processing unit or a graphics controller.



Architecture of a simple random scan system.

Graphics patterns are drawn on a random-scan system by directing the electron beam along the component lines of the picture. Lines are defined by the values for their coordinate endpoints, and these input coordinate values are converted to x and y deflection voltages. A scene is then drawn one line at a time by positioning the beam to fill in the line between specified endpoints.

INTERACTIVE INPUT DEVICES

Introduction

As hardware cost is plummeting, which is considered as the major bottleneck for the progress; now communication devices is more listened for better development. For that reason, techniques for developing high-quality user interfaces are moving to the forefront in computer science and are becoming the "last frontier" in providing computing to a wide variety of users—as other aspects of technology continue to improve, but the human users remain the same. Interest in the quality of user-computer interfaces is a recent part of the formal study of computers. The emphasis until the early 1980s was on optimizing two scarce hardware resources, computer time and memory. Program efficiency was the highest goal. With today's plummeting hardware costs and powerful graphics-oriented personal computing environments the focus turns to optimizing user efficiency rather than computer efficiency. Thus, although many of the ideas presented in this chapter require additional CPU cycles and memory space, the potential rewards in user productivity and satisfaction well outweigh the modest additional cost of these resources.

The quality of the user interface often determines whether users enjoy or despise a system, whether the designers of the system are praised or damned, whether a system succeeds or fails in the market. Actually, a poor user interface such as in air traffic control or in nuclear power plant monitoring can lead to catastrophic consequences.

The desktop user-interface metaphor, with its windows, icons, and pull-down menus, all making heavy use of raster graphics, is popular because it is easy to learn and requires little typing skill. Most users of such systems are not computer programmers and have little sympathy for the old-style difficult-to-learn keyboard-oriented command-language interfaces that many programmers take for granted. The designer of an interactive graphics application must be sensitive to users' desire for easy-to-learn yet powerful interfaces. In this chapter, we discuss the three basic low-level elements of user interfaces: input devices, interaction techniques, and interaction tasks. **Interaction techniques are the primitive building blocks from which a user interface is crafted.**

We focus in this chapter on input devices—those pieces of hardware by which a user enters information into a computer system. Input devices for the earliest computers were switches and knobs, jumper wires placed in patch boards, and punched cards. These were followed by the teletype, the text-only forerunner of today's interactive terminals. The mouse and keyboard now predominate, but a wide variety of input devices can be used.

An interaction task is the entry of a unit of information by the user. Basic interaction tasks are *position*, *text*, *select*, and *quantify*. The unit of information that is input in a position interaction task is of course a position; the text task yields a text string; the select task yields an object identification; and the quantify task yields a numeric value. A designer begins with the interaction tasks necessary for a particular application.

For each such task, the designer chooses an appropriate interaction device and interaction technique. Many different *interaction techniques* can be used for a given interaction task, and there may be several different ways of using the same device to perform the same task. For instance, a selection task can be carried out by using a mouse to select items from a menu, using a keyboard to enter the name of the selection, pressing a function key, circling the desired command with the mouse, or even writing the name of the command with the mouse. Similarly, a single device can be used for different tasks: A mouse is often used for both positioning and selecting.

Interaction tasks are defined by *what* the user accomplishes, whereas logical input devices categorize *how* that task is accomplished by the application program and the graphics system. Interaction tasks are user-centered, whereas logical input devices are a programmer and graphics-system concept. By analogy with a natural language, single actions with input devices are similar to the individual letters of the alphabet from which

words are formed. The sequence of input-device actions that makes up an interaction technique is analogous to the sequence of letters that makes up a word. A word is a unit of meaning; just as several interaction techniques can be used to carry out the same interaction task, so too words that are synonyms convey the same meaning. An interactive dialogue is made up of interaction-task sequences, just as a sentence is constructed from word sequences.

Concept of Positioning and Pointing

Most display terminals provide the user with an alphanumeric keyboard with which to type commands and enter data for the program. For some applications, however, the keyboard is inconvenient or inadequate. For example, the user may wish to indicate one of a number of symbols on the screen, in order to erase the symbol. If each symbol is labeled, he can do so by typing the symbol's name; by pointing at the symbol, however, he may be able to erase more rapidly, and the extra clutter of labels can be avoided.

Another problem arises if the user has to add lines or symbols to the picture on the screen. Although he can identify an items's position by typing coordinates he can do so even better by pointing at the screen, particularly if what matters most is the items's position relative to the rest of the picture.

These two examples illustrate the two basic types of graphical interaction: pointing at items already on the screen and positioning new items. The need to interact in these ways has stimulated the developed of a number of different types of graphical input device,some of which are described in this chapter.

Ideally a graphical input device should lend itself both to pointing and to positioning. In reality there are no devices with this versatility. Most devices are much better at positioning than at pointing; one device, the light pen, is the exact opposite. Fortunately, however we can supplement the deficiencies of these devices by software and in this way produce hardware-software system that has both capabilities. Nevertheless the distinction between pointing and positioning capability is extremely important.

Another important distinction is between devices that can be used directly on the screen surface and devices that cannot. The latter might appear to be less useful, but this is far from true. Radar operators and air-traffic controllers have for years used devices like the joystick and the tracker ball neither of which can be pointed at the screen. The effectiveness of these input devices depends on the use of visual feedback: the x and y

outputs of the device control the movement of a small cross, or cursor, displayed on the screen. The user of the device steers the cursor around the screen as if it were a toy boat
on the surface of a pond. Although this operation sounds as if it requires a lot of skill, it is in fact very easy.
The use of visual feedback has an additional advantage: just as in any control system, it compensates for any lack of linearity in the device. A linear input device is one that faithfully increases or decreases the input coordinate value in exact proportion to the user's hand movement. If the device is being used to trace a graph or a map. Linearity is important. A cursor, however, can be controlled quite easily even if the device behaves in a fairly nonlinear fashion. For example, the device may be much less sensitive near the left – hand region of its travel: a 1 – inch hand movement may change the x value by only 50 units, whereas the same movement elsewhere may change x by 60 units. The user will simply change his hand movement to compensate, often without even noticing the no linearity. This phenomenon has allowed simple, inexpensive devices like the mouse to be used very successfully for graphical input.

Interactive Graphic Devices

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

Keyboards

The well-known QWERTY keyboard has been with us for many years. It is ironic that this keyboard was originally designed to *slow down* typists, so that the typewriter hammers would not be so likely to jam. Studies have shown that the newer Dvorak
keyboard , which places vowels and other high-frequency characters under the home positions of the fingers, is somewhat faster than is the QWERTY design. It has not been widely accepted. Alphabetically organized keyboards are sometimes used when many of the users are non typists. But more and more people are being exposed to QWERTY keyboards, and experiments have shown no advantage of alphabetic over QWERTY keyboards .In recent years, the chief force serving to displace the keyboard has been the shrinking size of computers, with laptops, notebooks, palmtops, and personal digital assistants. The typewriter keyboard is becoming the largest component of such pocketsized devices, and often the main component standing in the way of reducing its overall size. The *chord keyboard* has five keys similar to piano keys, and is operated with one

hand, by pressing one or more keys simultaneously to "play a chord." With five keys, 31 different chords can be played. Learning to use a chord keyboard (and other similar stenographer style keyboards) takes longer than learning the QWERTY keyboard, but skilled users can type quite rapidly, leaving the second hand free for other tasks. This increased training time means, however, that such keyboards are not suitable substitutes for general use of the standard alphanumeric keyboard. Again, as computers become smaller, the benefit of a keyboard that allows touch typing with only five keys may come to outweigh the additional difficulty of learning the chords. Other keyboard-oriented considerations, involving not hardware but software design, are arranging for a user to enter frequently used punctuation or correction characters without needing simultaneously to press the control or shift keys, and assigning dangerous actions (such as delete) to keys that are distant from other frequently used keys.

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 with graphical icons. Other systems can be adapted for touch input by fitting a transparent device with 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. The opposite vertical and horizontal edges contain light detectors. 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 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 the 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

The pencil-shaped devices 's are used to select screen positions by detecting the light coming from point on the CRT screen. They are sensitive to the short burst of light emitted from the phosphor coating at the instant the electron beam strikes a particular point. Other light sources, such as the background light in the room, are usually not detected by a light pen. An activated light pen, pointed at a spot on the screen as the electron beam lights up that spot, generates an electrical pulse that causes the coordinate position of the electron beam to be recorded. As with cursor-positioning devices,recorded light-pen coordinates can be used to position an object or to select a processing option. Although light pens are still with us, they are not as popular as they once were since they have several disadvantages compared to other input devices that have been developed. For one, when a light pen is pointed at the screen, part of the screen image is obscured by the hand and pen. And prolonged use of the light pen can cause arm fatigue.

Also, light pens require special implementation 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 intensity assigned to each screen pixel.

In addition, light pens sometime give false readings due to background lighting in a room.

Graphics Tablets

One type of digitizer is the graphics tablet (also referred to as a data tablet), which is used to input two-dimensional coordinates by activating a hand cursor or stylus at selected positions on a flat surface. A hand cursor contains cross hairs for sighting positions, while a stylus is a pencil-shaped device that is pointed at positions on the tablet. This allows an artist to produce different brush strokes with different pressures on the tablet surface.

Tablet size varies from 12 by 12 inches for desktop models to 4 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 wire 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, a their signal strength, coded pulses, or phase shifts can be used to determine the position on the tablet.

Joysticks

A joystick consists of a small, vertical lever (called the stick) mounted on a base that is used to steer the screen cursor around. Most joysticks select screen positions with actual stick movement; others respond to pressure on the stick. The distance that the stick is moved in any direction from its center position corresponds to screen-cursor movement in that direction. Potentiometers mounted at the base of the joystick measure the amount of movement, and springs return the stick to the center position when it is released. One or more buttons can be programmed to act as input switchs to signal certain actions once a screen position has been selected.

Mouse

A mouse is small hand-held box used to position the screen cursor. Wheels or rollers on the bottom of the mouse can be used to record the amount and direction of movement.

Another method for detecting mouse motion is with an optical sensor. For these systems, the mouse is moved over a special mouse pad that has a grid of horizontal and vertical lines. The optical sensor detects movement across the lines in the grid.

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, or three buttons are usually included on the top of the mouse for signaling the execution of some operation, such as recording cursor position or invoking a function. Most general-purpose graphics systems now include a mouse and a keyboard as the major input devices.

Voice Systems

Speech recognizers are used in some graphics workstations as input devices to accept 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 for a particular operator by having the operator speak the command words to be used into the system. Each word is spoken several times, and the system analyzes the word and establishes a frequency pattern for that word in the dictionary along with the corresponding function to be performed. Later, when a voice command is given, the system

searches the dictionary for a frequency-pattern match. Voice input is typically spoken into a microphone mounted on a headset. The microphone is designed to minimize input of other background sounds. If a different operator is to use the system, the dictionary must be reestablished with that operator's voice patterns. Voice systems have some advantage over other input devices, since the attention of the operator does not have to be switched from one device to another to enter a command.

Logical Input Devices

Some APIs (PHIGS, GKS, Direct xx) supports 6 classes of logical input devices – OpenGL does not take this approach

Two older APIs (GKS, PHIGS) defined six types of logical input

Locator: return a position:
Pick: return ID of an object:
Keyboard: return strings of characters:
Stroke: return array of positions:
Valuator: return floating point number:
Choice: return one of n items

String – logical device providing ASCII strings – keyboard

Locator – provides a position in world coordinates – usually implemented via pointing device– mouse, trackball. OpenGL provides similar but conversion from screen coordinates to world coordinates must be made by a user

Pick – returns identifier of an object – in OpenGL process called selection can be used to accomplish picking

Choice – allows the user to select on of a discrete number of options – in OpenGL various widgets provided by the window system can be used; widget is a graphical interactive device provided by window system or a toolkit (menu with n selections etc.)

Dial – provides analog input to the user program – slide bars etc.

Stroke – device returns an array of locations – different implementations – usually: mouse button down, transfer data to an array with different positions, release button – ends the transfer

Input Modes
Input devices contain a trigger which can be used to send a signal to the operating system Button on mouse Pressing or releasing a key

When triggered, input devices return information (their measure) to the system)Mouse returns position information Keyboard returns ASCII code

Request Mode
Input provided to program only when user triggers the device

Typical of keyboard input Can erase (backspace), edit, correct until enter (return) key (the trigger) is depressed

Event Mode
Most systems have more than one input device, each of which can be triggered at an arbitrary time by a user

Each trigger generates an event whose measure is put in an event queue which can be examined by the user program

Event Types
Window: resize, expose, iconify
Mouse: click one or more buttons
Motion: move mouse
Keyboard: press or release a key
Idle: nonevent
Define what should be done if no other event is in queue

OUTPUT PRIMITIVES
Output primitives are the geometric structures such as straight line segments (pixel array) and polygon color areas, used to describe the shapes and colors of the objects. Points and straight line segments are the simplest geometric components of pictures. Additional output primitive includes: circles and other conic sections, quadric surfaces, spline curves and surfaces, polygon color areas and character strings. Here, we discuss picture generation algorithm by examining device-level algorithms for displaying two-dimensional output primitives, with emphasis on scan-conversion methods for raster graphics system.

Points and Lines
Point plotting is done in CRT monitor by turning on the electron beam to illuminate at the screen phosphor at the selected location.

Random-scan systems: stores point plotting instructions in the display list and co-ordinate values in these instructions are converted into deflection voltages that position the electron beam at selected location.

B/W raster system: Within frame buffer, bit value is set to 1 for specified screen position. Electron beam then sweeps across each horizontal scan line, it emits a burst of electrons (plots a point) whenever value of 1 is encountered in the frame buffer.
RGB raster system: Frame buffer is loaded with the color codes for the intensities that are to be displayed at the screen pixel positions.

Line drawing is accomplished by calculating intermediate positions along the line path between two specified endpoint positions. An output device is then directed to fill in these positions between the endpoints.

For analog devices (vector-pen plotter and random-scan display), a straight line can be drawn smoothly between two points.
Reason: linearly varying horizontal and vertical deflection voltages are generated that are proportional to the required changes in the x and y directions.

Digital devices display a straight line segment by plotting discrete points between two end-points. Discrete integer coordinates are calculated from the equation of the line. Since rounding of coordinate values occur [(4.48, 48.51) would be converted to (4, 49)], line is displayed with stair step appearance.

SOFTWARE STANDARDS
Primary goal of standardized graphics software is portability. When packages are designed with standard graphics functions, software can he moved easily from one hardware system to another and used in different implementations and applications.
International and national standards planning organizations in many countries have cooperated in an effort to develop a generally accepted standard for computer graphics.

After considerable effort, this work led to following standards:
GKS (Graphical Kernel System): This system was adopted as the first graphics software standardby the International Standards Organization (ISO) and American National Standards Institute (ANSI). Although GKS was originally designed as a two-

dimensional graphics package, a three-dimensional GKS extension was subsequently developed.

PHIGS (Programmer's Hierarchical Interactive Graphics Standard): Extension to GKS, Increased Capabilities for object modeling, color specifications, surface rendering and picture manipulations are provided. Subsequently, an extension of PHIGS, called PHIGS+, was developed to provide three-dimensional surface-shading capabilities not available in PHIGS.

Although PHIGS presents a specification for basic graphics functions, it does not provide a standard methodology for a graphics interface to output devices (i.e. still machine dependent). Nor does it specify methods for storing and transmitting pictures. Separate standards have been developed for these areas:

CGI (Computer Graphics interface):Standardization for device interface

CGM (Computer Graphics Metafile):Standards for archiving and transporting pictures

GRAPHICS SOFTWARE

There are two general categories of graphics software

General programming packages:

Provides extensive set of graphics functions for high level languages (FORTRAN, C etc).

Basic functions include those for generating picture components (straight lines, polygons, circles, and other figures), setting color and intensity values, selecting views, and applying transformations.

Example: GL(Graphics Library)

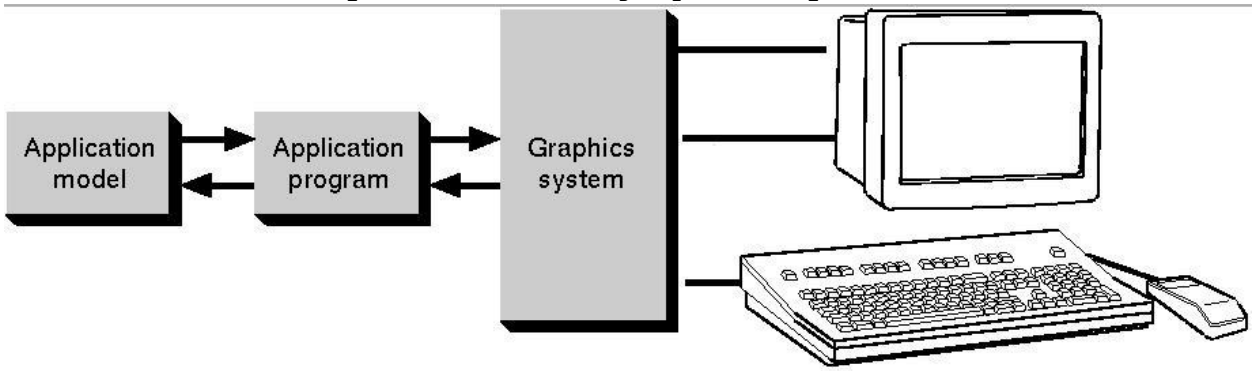
Special-purpose application packages: Designed for nonprogrammers, so that users can generate displays without worrying about how graphics operations work.

The interface to the graphics routines in such packages allows users to communicate with the programs in their own terms.

Example: artist's painting programs and various business, medical, and CAD systems.

CONCEPTUAL FRAMEWORK FOR INTERACTIVE GRAPHICS SYSTEM

The high-level conceptual framework shown here can be used to describe almost any interactive graphics system.



The three major parts of the framework are:

Application Modeling

Calculating what is to be displayed

Displaying the Model

Calling the graphics API routines

Interaction Handling

Handling user interaction, which will change the model, and therefore the display.

typically an event driven loop

- Graphics Library - Between application and display hardware there is graphics library / API.

- Application Program - An application program maps all application objects to images by invoking graphics.

- Graphics System – An interface that interacts between Graphics library and Hardware.

- Modifications to images are the result of user interaction.

UNIT I

Line Drawing Algorithms

1) Digital Differential Analyzer (DDA) Algorithm:

Straight Line Equation in the form of Slope intercept is as follows:

$$y = m x + b \qquad \text{----------1}$$

where m represents the slope of the line and b as the y intercept which it makes with the Y axis. The two end point of a line segment are denoted by the positions (x_1, y_1) and (x_2, y_2) as shown in the following diagram. Using this Equation we can determine values for the slope m and y intercept b using the following calculations.

$$M = \frac{y_2 - y_1}{x_2 - x_1} \qquad \text{----2}$$

$$b = y_1 - m x_1 \qquad \text{-----3}$$

Value of y is calculated

$$\Delta y - m \cdot \Delta x \qquad \text{\_ \_ \_ (4)}$$

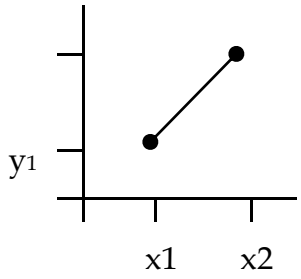


Figure. (1) Line Path between endpoint position (x_1, y_1) & (x_2, y_2)

Similarly we can obtain Δ x interval

$$\Delta x = \frac{y}{m}$$

For lines with slope magnitude $m > 1$, Δy can be set proportional to a small deflection voltage with the corresponding horizontal deflection voltage set proportional to Δx.

For lines with m = 1 Δx = Δy.

DDA Algorithm : Also Called as Digital Differential Analyzer (DDA) performs scan. It is a Scan conversion line algorithm. It is also called an incremental algorithm, as it increments the value of x or y by 1 depending on the slope value. It tries to decrease the computation burden and increase the sped of computing. Conversion line Algorithm based on calculating either Δy or Δx using equation (4) & (5).

We sample the line at unit intervals in one coordinate and determine corresponding integer values nearest. The line paths for the other

coordinate. Now consider first a line with positive slope, as shown in Figure.(1). If the slope is less than one or equal to 1. We sample at unit x intervals ($\Delta x = 1$) compute each successive y values as :

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

Value k takes integer values starting form 1, for the first point & gets incremented by 1 until the final end point is reached.

For lines with positive slope greater than 1, we reverse the role of x and y. That is we sample at unit y intervals ($\Delta y = 1$) and calculate each succeeding x value as :

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

Equation (6) and (7) are based on assumption that lines are to be processed form left end point to the right end point.

If this processing is reversed the sign is changed

$\Delta x = -1$ & $\Delta y = -1$

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

$$x_{k+1} = x_k - \frac{1}{m} \qquad \_ \_ \_ (9)$$

Equations (6) to (9) are used to calculate pixel position along a line with negative slope.

When the start endpoint is at the right we set $\Delta x = -1$ and obtain y position from equation (7) similarly when Absolute value of Negative slope is greater than 1, we use $\Delta y = -1$ & eq.(9) or we use $\Delta y = 1$ & eq.(7).

DDA Example:

(0, 0) to (8, 5)

x1 = 0 y1 = 0

x2 = 8 y2 = 4

$m = 5 - 0 / 8 - 0 = 5/8 = 0.6 < 1$

so, x = 0+1 y=0+0.6 (Round off y values)

i	x	y	(x,y)
	0	**0**	
1	1.0	0.6	(1,1)
2	2.0	1.2	(2,1)
3	3.0	1.8	(3,2)
4	4.0	2.4	(4,2)
5	5.0	3.0	(5,3)
6	6.0	3.6	(6,4)
7	7.0	4.2	(7,4)
8	8.0	4.8	(8,5)

Plot the graph with these (x,y) points.

2) Bresenham's Line Drawing Algorithm :

This is very efficient and faster line drawing algorithm. It scan converts lines and uses only incremental integer calculations. Thus we can use this algorithm for drawing circles and similar other curves also. An accurate and efficient raster line generating Algorithm, developed by Bresenham, scan concerts line using only incremental integer calculations that can be adapted to display circles and other curves. The vertical axes show scan-line position, & the horizontal axes identify pixel columns as shown in Figure. (5) & (6) . This algorithm follows the closeness theory to implement line plotting.

As we did for DDA algorithm, here also we start with I octant where slope , m<1. Since m<1, we move in x-direction by sampling at unit x intervals.

Thus we start plotting from initial, say(x0-y0) and take steps in success x-columns an plot the point whose y-values is closest to the ideal line path.

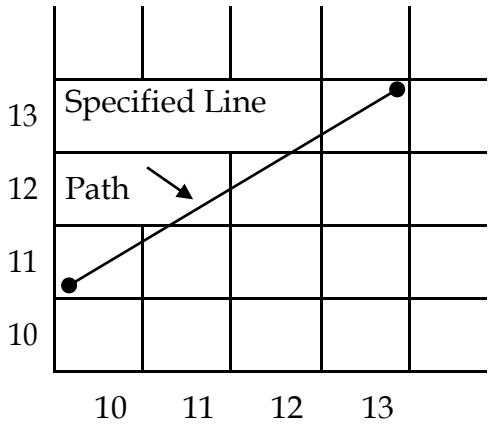


Figure.5

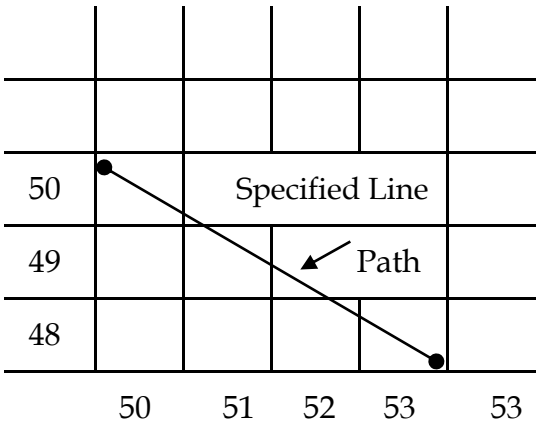


Figure.6

We first consider the scan conversion process for lines with positive slope less than 1 to illustrate Bresenham's approach. Pixel position along a line path are then determined by sampling at unit x intervals starting form left and point (x_0 , y_0) of a given line, we step at each successive column (x position) & plot the pixel whose scan line y is closest to the line path. Now assuming we have to determine that the pixel at (x_k , y_k) is to be displayed, we next need to divide which pixel to plot in column x_{k+1}. Preference would be at the position (x_{k+1} , y_k) and (x_{k+1} , y_{k+1}). At sampling position x_{k+1}, we label vertical pixel separations from the mathematical line path as d_1 and d_2 Figure.(8).

The y coordinate on the mathematical line at pixel column position x_{k+1} is calculated as :

$$y = m(x_k + 1) + b \qquad \_ \_ \_ (10)$$

Then $d_1 = y - y_k = m(x_k + 1) + b - y_k$

$$d_2 = (y_k + 1) - y = y_k + 1 - m(x_k + 1) - b$$

The difference can be define between these two separations as

$$d_1 - d_2 = 2m(x_k + 1) - 2y_k + 2b - 1 \quad \_ \_ \_ (11)$$

The deciding Parameter P_k for the K_{th} step in the line algorithm can be obtained by making some rearrangements in eq.(11) so that it involves sort of integer calculation. We accomplish this by substituting $m = \Delta y/\Delta x$. where Δy & Δx are the vertical & horizontal separation of the endpoint positions & defining. The sign of P_k remains same as that of the sign of $d_1 - d_2$.

$$P_k = \Delta x (d_1 - d_2) = 2\Delta y . x_k - 2\Delta x\, y_k + c \qquad \_ \_ \_ (12)$$

Since $\Delta x > 0$ for our example Parameter C is constant & has the value $2\Delta y + \Delta x (2b - 1)$, which is independent of pixel position.

If the pixel position at y_k is closer to line path than the pixel at y_{k+1} (that is $d_1 < d_2$), then decision Parameter P_k is Negative. In that case we plot the lower pixel otherwise we plot the upper pixel. Coordinate changes along the line owner in unit steps in either the x or directions. Therefore we can obtain the values of successive decision Parameter using incremental integer calculations. At step k = 1, the decision Parameter is evaluated form eq.(12) as :

$$P_{k+1} = 2\Delta y . x_{k+1} - 2\Delta x . y_{k+1} + C$$

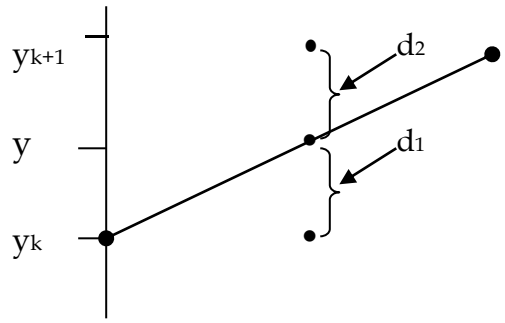


Figure.8

Subtracting eq.(12) from the preceding equation we have

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

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

So that, $P_{k+1} = P_k + 2\Delta y - 2\Delta x\,(y_{k+1} - y_k)$ _ _ _ (13)

The term $y_{k+1} - y_k$ either results into 0 or 1, depending on sign of Parameter P_k.This recursive calculation of decision Parameter is performed each integer x position, starting at left coordinate endpoint of the line. The first parameter P_0 is evaluated from equation (12) at starting pixel position (x_0, y_0) and with m evaluated as $\Delta y/\Delta x$.

$$P_0 = 2\Delta y - \Delta x \qquad \text{_ _ _ (14)}$$

The following lines express how does Bresenham's Line Drawing Algorithm work for $|m|<1$:

Take input for two endpoints of a line & store the left end point in (x_0 , y_0). Load (x_0 , y_0) into frame buffer that is plot the first point. Calculate

constants Δx, Δy, $2\Delta y$ and $2\Delta y - 2\Delta x$ and obtain the starting value for the decision parameter as : $P_0 = 2\Delta y - \Delta x$. 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 $P_{k+1} = P_k + 2\Delta y - 2\Delta x$. Repeat step 4 Δx times.

Example :

Digitize the line with end points (20, 10) & (30, 18) using Bresenham's Line Drawing Algorithm.

slope of line, $m = \frac{y_2 - y_1}{x_2 - x_1} = \frac{18 - 10}{30 - 20} = \frac{8}{10} = 0.8$

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

Initial decision parameter has the value

$P_0 = 2\Delta y - \Delta x = 2x8 - 10 = 6$

Since $P_0 > 0$, so next point is $(x_k + 1, y_k + 1)$ (21, 11)

Now k = 0, $P_{k+1} = P_k + 2\Delta y - 2\Delta x$

$P_1 = P_0 + 2\Delta y - 2\Delta x$

$= 6 + (-4)$

$= 2$

Since $P_1 > 0$, Next point is (22, 12)

Now k = 1, $P_{k+1} = P_k + 2\Delta y - 2\Delta x$

$P_2 = 2 + (-4)$

$= -2$

Since $P_2 < 0$, Next point is (23, 12)

Now k = 2 $P_{k+1} = P_k + 2\Delta y$ ∴

$P_2 = -2 + 16$

$= 14$

Since $P_3 > 0$, Next point is (24, 13)

Now k = 3 $P_{k+1} = P_k + 2\Delta y - 2\Delta x$

$P_4 = 14 - 4$

$= 10$

Since $P_4 > 0$, Next point is (25, 14)

Now k = 4 $P_{k+1} = P_k + 2\Delta y - 2\Delta x$

$P_5 = 10 - 4$

$= 6$

Since $P_5 > 0$, Next point is (26, 15)

Now k = 5 $P_{k+1} = P_k + 2\Delta y - 2\Delta x$

$P_6 = 6 - 4$

$= 2$

Since $P_6 > 0$, Next point is (27, 16)

Now k = 6 $P_{k+1} = P_k + 2\Delta y - 2\Delta x$

$P_7 = 2 + (-4)$

$= -2$

Since $P_7 < 0$, Next point is (28, 16)

Now k = 7 $P_{k+1} = P_k + 2\Delta y$

$P_8 = -2 + 16$

$= 14$

Since $P_8 > 0$, Next point is (29, 17)

Now k = 8 $P_{k+1} = P_k + 2\Delta y - 2\Delta x$

$P_9 = 14 - 4$

$= 10$

Since $P_9 > 0$, Next point is (30, 18)

K	**P_k**	**(x_{k+1}, y_{k+1})**
0	6	(21, 11)
1	2	(22, 12)

2	-2	(23, 12)
3	14	(24, 13)
4	10	(25, 14)
5	6	(26, 15)
6	2	(27, 16)
7	-2	(28, 16)
8	14	(29, 17)
9	10	(30, 18)

Plot the graph with these points.

Circle Drawing Algorithms

1) Mid Point Circle Algorithm

The equation of a circle can be given as follows, where (x_c,y_c) represents the centre coordinates.

$$(x - x_c)^2 + (y - y_c)^2 - r^2 = 0$$

In the following way the calculation is made for the position of points along the circlular path by moving in the x direction from $(x_c - r)$ to $(x_c + r)$ and determining the corresponding y values as :

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

As it requires heavy computation this method is not the best method to calculate the circle point coordinates. Moreover spacing between the

points is not uniform. Another method that can be used by calculating the polar coordinates r and θ where

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

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

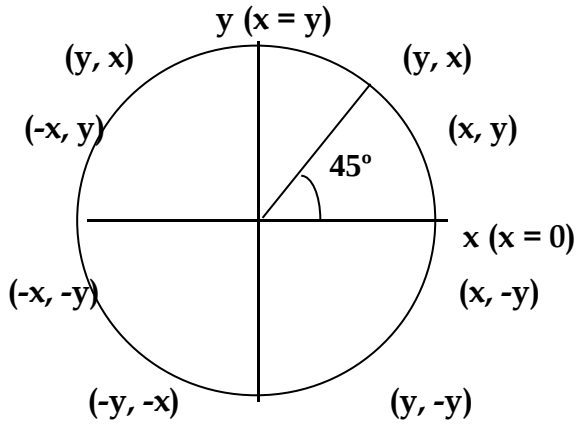
It requires heavy computation but this method results in equal spacing between the points. The efficient method is incremental calculation of decision parameter.

Mid Point Algorithm :

We assume that we are working in II octant of the circle .The concept behind Mid point circle is that, a midpoint M lies between two points and we have to decide if M lies between two points and we have to decide if M lies inside or outside the circle. This would tell the next point to be plotted along the circumference of the circle. We move in unit steps in the x-direction and calculate the closed pixel position along the circle path at each step.

For a given radius r & screen center position (x_c, y_c). We first set our Algorithm to calculate the position of points along the coordinate position (x_0, y_0). These calculated positions are then placed at this proper screen position by adding x_c to x and y_c to y. For a circle from x = 0 to x = y in first quadrant, the slope varies from 0 to 1.

We move in the positive x direction and determine the decision parameter to find out the possible two y values along the circle path. And the Points calculation in other 7 octants is done using the symmetry pattern.

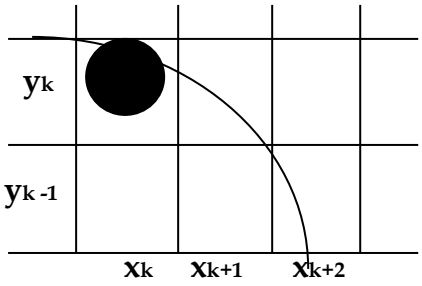


The following function is used for the implementation of this method :

$$f_{circle}(x, y) = x^2 + y^2 - r^2 \qquad \_\_\_ (1)$$

Any point (x, y) on the boundary of the circle with radius r satisfies the equation of $f_{circle}(x, y) = 0$. The relative position of any point (x, y) can be determined by checking the sign of circle function.

$$f_{circle}(x, y) \begin{cases} < 0 \text{ if (x, y) denotes it inside circle boundary.} \\ = 0 \text{ if (x, y) denotes it on circle boundary.} \_\_> (2) \\ 0 \text{ if (x, y) denotes it outside circle boundary.} \end{cases}$$



Taking an assumption that we have just plotted a pixel at (x_k, y_k). We next need to determine whether the pixel (x_{k+1}, y_k) or (x_{k+1}, y_{k-1}) is closer. Our decision parameter is the circle function evaluated at the mid point between these two pixels.

$$P_k = f_{circle}(x_k + 1, y_k - ½)$$

Or $$P_k = (x_k + 1)^2 + (y_k - ½)^2 - r^2 \quad \_ \_ \_ (3)$$

This denotes that If $P_k < 0$, Mid point is inside the circle boundary and the pixel on the scan line y_k is closer to the circle boundary. Otherwise, Mid point is on or outside the circle boundary and the point on the scan line $y_k - 1$ is closer. Successive decision parameters are obtained by incremental calculations. Again the next deciding parameter is calculate the position at next sampling position by taking the next position.

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

$$P_{k+1} = f_{circle}(x_{k+1} + 1, y_{k+1} - ½)$$

Or $$P_{k+1} = [(x_k + 1) + 1]^2 + (y_{k+1} - ½)^2 - r^2$$

Or $$P_{k+1} = P_k + 2(x_k + 1) + (y_{k+1}^2 - y_k^2) - (y_{k+1} - y_k) + 1 \quad \_ \_ \_ (4)$$

Successive increment for P_k is $2x_{k+1} + 1$ (If $P_k < 0$) otherwise $(2x_{k+1} + 1 - 2y_{k+1})$ where

$$2x_{k+1} = 2x_k + 2 \quad \& \quad 2y_{k+1} = 2y_k - 2$$

Initial decision parameter P_0 is obtained as $(0, r) = (x_0, y_0)$

$$P_0 = f_{circle}(x, y) = f_{circle}(1, r - ½) = 1 + (r - ½)^2 - r^2$$

Or $$P_0 = \frac{5}{4} - r$$

If r is a integer then $P_0 = 1 - r$

Algorithm for this can be defined in the following steps for calculating the Mid Point:

(1) Input radius r and circle center (x_c, y_c) and obtain the first point on circumference of a circle centered on origin $(x_0, y_0) = (0, r)$

(2) Calculate the initial value of the decision parameter as : $P_0 = 5/4 - r$

(3) At each x_k position, starting at k = 0 if $P_k < 0$ the next point along the circle 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 $2x_{k+1} = 2x_k + 2$ & $2y_{k+1} = 2y_k - 2$.

(1) Determine symmetry points in other seven octants.

(2) Move each calculated pixel position (x, y) onto the circular path centered on (x_c, y_c) & plot coordinate values $x = x + x_c$ & $y = y + y_c$.

(3) Repeat step (3) through (5) until $x \geq y$.

Example: Demonstrate the Mid Point Circle Algorithm with circle radius, r = 10.

$P_0 = 1 - r = 1 - 10 = -9$

Now the initial point $(x_0, y_0) = (0, 10)$ and initial calculating terms for calculating decision parameter are

$2x_0 = 0$, $2y_0 = 20$	Since $P_k < 0$,	Next point is (1, 10)
$P_1 = -9 + 3 = -6$	Now $P_1 < 0$,	Next point is (2, 10)
$P_2 = -6 + 5 = -1$	Now $P_2 < 0$,	Next point is (3, 10)
$P_3 = -1 + 7 = 6$	Now $P_3 > 0$,	Next point is (4, 9)
$P_4 = 6 + 9 - 18 = -3$	Now $P_4 < 0$,	Next point is (5, 9)
$P_5 = -3 + 11 = 8$	Now $P_5 > 0$,	Next point is (6, 8)
$P_6 = 8 + 13 - 16 = 5$	Now $P_6 > 0$,	$\therefore$ Next point is (7, 7)

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

Plot the graph with these points.

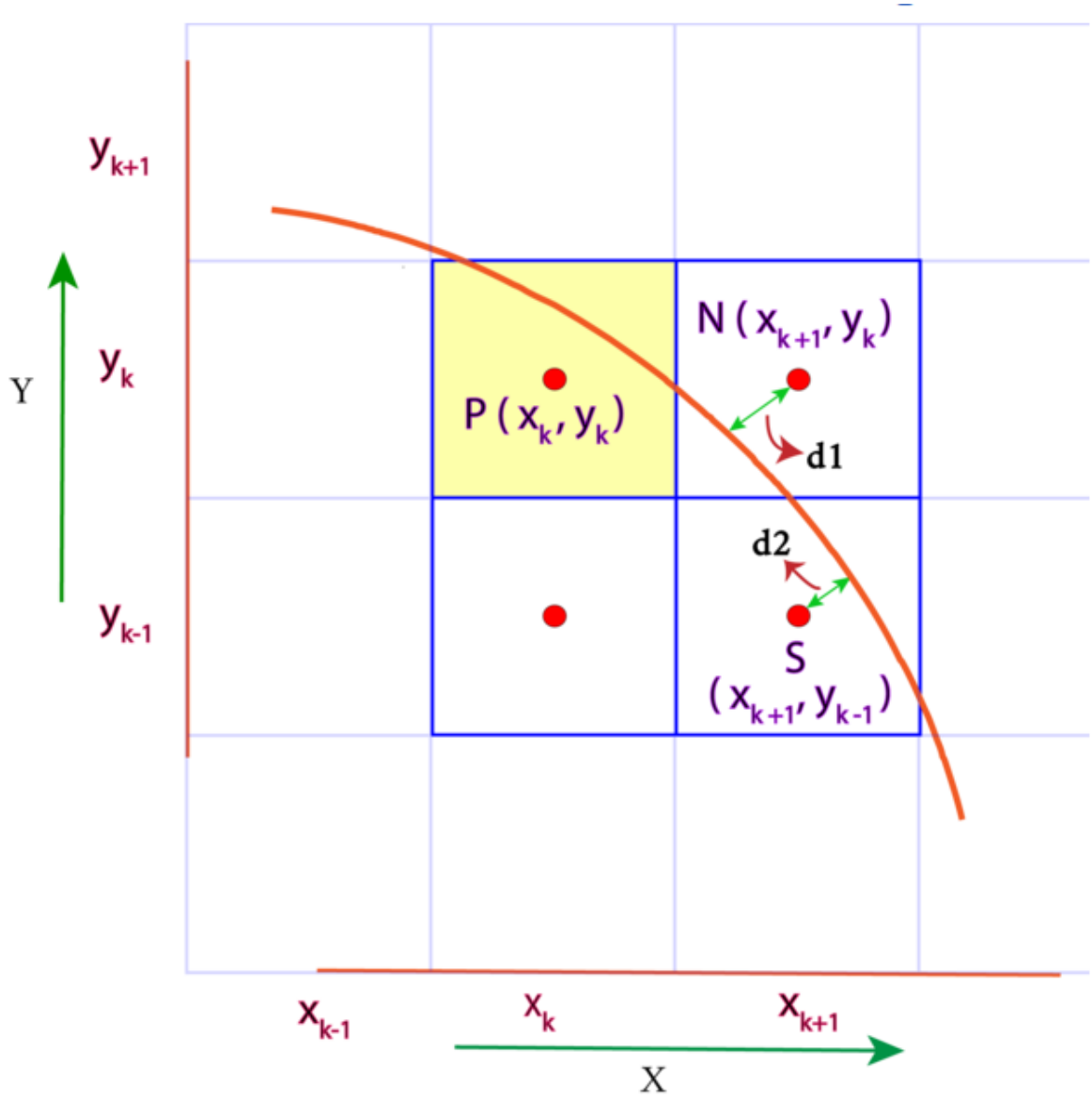
2) Bresenham's Circle Drawing Algorithm:

A continuous arc cannot be displayed in the raster. Hence nearest pixel position is chosen for completing the arc.

It is observed from the following illustration that the pixel is put at (X, Y) location and to decide where to put the next pixel at **N (X+1, Y)** or at S **(X+1, Y-1).**

This can be decided by the decision parameter d.

- If d <= 0, then **N(X+1, Y)** is to be chosen as next pixel.
- If d > 0, then **S(X+1, Y-1)** is to be chosen as the next pixel.



Let's say our circle is at some random pixel P whose coordinates are (x_k, y_k).
Now we need to find out our next pixel.

Note- This is octet 2 so here x can never be decremented as per properties of a circle but y either needs to decremented or to be kept same. y is needed to be decided.

Here it needs to decide whether go with N or S.

For this bresenham's circle drawing algorithm will help us to decide by calculating the difference between radius and the coordinates of the next pixels.

The shortest of d1 and d2 will help us Decide our next pixel.

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

As x_{k+1} **is the next consecutive pixel of** x_k

similarly

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

Equation of Circle with Radius r

$$(x - h)^2 + (y - k)^2 = r^2$$

When coordinates of centre are at Origin i.e., (h=0, k=0)

$$x^2 + y^2 = r^2$$ **(Pythagoras theorem)**

Function of Circle Equation

$$F(C) = x^2 + y^2 - r^2$$

Function of Circle at N

$$F(N) = (x_{k+1})^2 + (y_k)^2 - r^2$$ **(Positive)**

Here the value of F(N) will be positive because N is out-side the circle that makes $(x_{k+1})^2 + (y_k)^2$ **Greater than** r^2

Function of Circle at S

$$F(S) = (x_{k+1})^2 + (y_{k-1})^2 - r^2$$ *(Negative)*

Here the value of F(S) will be Negative because S is in-side the circle that makes $(x_{k+1})^2 + (y_{k-1})^2$ *Less than* r^2

Now we need a decision parameter which help us decide the next pixel

Say D_k

And , $D_k = F(N) + F(S)$

Here either we will get the positive or negative value of D_k

So if $D_k < 0$

that means the negative F(S) is bigger then the positive F(N), that implies Point N is closer to the circle than point S. So we will select pixel N as our next pixel.

and if $D_k > 0$

that means positive F(N) is bigger and S is more closer as F(S) is smaller. So we will Select S as our next pixel.

Now lets find D_k

$$D_k = (x_{k+1})^2 + (y_k)^2 - r^2 + (x_{k+1})^2 + (y_{k-1})^2 - r^2$$

(replacing x_{k+1} with $x_k + 1$ and y_{k-1} with $y_k - 1$)

$$= (x_k + 1)^2 + (y_k)^2 - r^2 + (x_k + 1)^2 + (y_k - 1)^2 - r^2$$

$$= 2(x_k + 1)^2 + (y_k)^2 + (y_k - 1)^2 - 2r^2 \quad \text{----- } (i)$$

Now lets find D_{k+1}

(Replacing every k with k+1)

$$D_{k+1} = 2(x_{k+1} + 1)^2 + (y_{k+1})^2 + (y_{k+1} - 1)^2 - 2r^2$$

$$= 2(x_{k+1} + 1)^2 + (y_{k+1})^2 + (y_{k+1} - 1)^2 - 2r^2$$

(Replacing x_{k+1} with $x_k + 1$ but now we can't replace y_{k+1} because we don't know the exact value of y_k)

$$= 2(x_k + 1 + 1)^2 + (y_{k+1})^2 + (y_{k+1} - 1)^2 - 2r^2$$

$$= 2(x_k + 2)^2 + (y_{k+1})^2 + (y_{k+1} - 1)^2 - 2r^2 \quad \text{----- } (ii)$$

Now to find out the decision parameter of next pixel i.e. D_{k+1}
We need to find

$$D_{k+1} - D_k = (ii) - (i)$$

$$= 2(x_k + 2)^2 + (y_{k+1})^2 + (y_{k+1} - 1)^2 - 2r^2$$

$$- [\, 2(x_k + 1)^2 + (y_k)^2 + (y_k - 1)^2 - 2r^2]$$

$$= 2(x_k)^2 + 8x_k + 8 + (y_{k+1})^2 + (y_{k+1})^2 - 2y_{k+1} + 1 - 2r^2$$

$$- 2x_k - 4x_k - 2 - (y_k)^2 - (y_k)^2 + 2y_k - 1 + 2r^2$$

$$= 4x_k + 2(y_{k+1})^2 - 2y_{k+1} - 2(y_k)^2 - 2y_k + 6$$

$$D_{k+1} = D_k + 4x_k + 2(y_{k+1})^2 - 2y_{k+1} - 2(y_k)^2 - 2y_k + 6 \quad \text{----- } (iii)$$

If ($D_k < 0$) then we will choose N point as discussed.

i.e. (x_{k+1}, y_k)

that means our next x coordinate is x_{k+1} and next y coordinate is y_k i.e. $y_{k+1} = y_k$, putting y_k in *(iii)* in replace of y_{k+1}

now,

$$D_{k+1} = D_k + 4x_k + 2(y_k)^2 - 2y_k - 2(y_k)^2 - 2y_k + 6$$

$$= D_k + 4x_k + 6$$

If ($D_k > 0$) then we will choose S point.

i.e. (x_{k+1}, y_{k-1})

that means our next x coordinate is x_{k+1} and next y coordinate is y_k i.e. $y_{k+1} = y_{k-1}$ putting y_{k-1} in *(iii)* in replace of y_{k+1}

now,

$$D_{k+1} = D_k + 4x_k + 2(y_{k-1})^2 - 2y_{k-1} - 2(y_k)^2 - 2y_k + 6$$

Now we know

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

therefore,

$$D_{k+1} = D_k + 4x_k + 2(y_k - 1)^2 - 2(y_k - 1) - 2(y_k)^2 - 2y_k + 6$$

$$= D_k + 4x_k + 2(y_k)^2 + 2 - 4y_k - 2y_k + 2 - 2(y_k)^2 - 2y_k + 6$$

$$= D_k + 4x_k - 4y_k + 10$$

$$= D_k + 4(x_k - y_k) + 10$$

Now to find initial decision parameter means starting point that is (0,r) ,value of y is r

Putting (0,r) in *(i)*

$$D_k = 2(x_k + 1)^2 + (y_k)^2 + (y_k - 1)^2 - 2r^2$$

$$D_0 = 2(0 + 1)^2 + r^2 + (r - 1)^2 - 2r^2$$

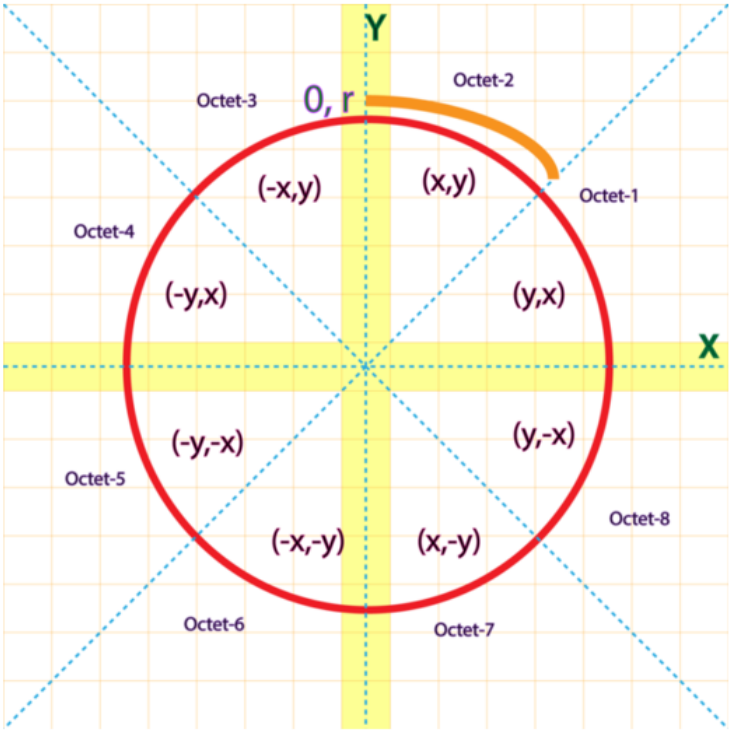
$$= 2 + r^2 + r^2 + 1 - 2r - 2r^2$$

$$= 3 - 2r$$

A circle is made up of 8 Equal Octets so we need to find only coordinates of any one octet rest we can conclude using that coordinates.

We took octet-2. Where X and Y will represent the pixel

Let us make a function Circle() with parameters coordinates of Centre (Xc,Yc) and pixel point (X,Y) that will plot the pixel on screen.



We will find pixels assuming that Centre is at Origin (0,0) then we will add the coordinates of centre to corresponding X and Y while drawing circle on screen.

Circle (Xc,Yc,X,Y) {

Plot (Y+Xc , X+Yc) ……Octet-1
Plot (X+Xc , Y+Yc) ……Octet-2
Plot (-X+Xc , Y+Yc) ……Octet-3
Plot (-Y+Xc , X+Yc) …..Octet-4
Plot (-Y+Xc , -X+Yc) ……Octet-5
Plot (-X+Xc , -Y+Yc) ……Octet-6
Plot (X+Xc , -Y+Yc) ……Octet-7
Plot (Y+Xc , -X+Yc) ……Octet-8
}

Now,
Each plot function is for different octet and will construct the circle while in loop.

Step 1: Get the Radius of Circle R
And Coordinates of centre of circle (Xc,Yc).

Step 2: X and Y are going to be plotted points
Set X=0 and Y=R

Step 3: D = 3-2R (Initial decision Parameter)

Step 4: Plot Circle (Xc,Yc,X,Y)

Step 5: if D < 0 Then

D = D + 4X + 6
X=X+1
Y=Y

Else

D=D+4(X-Y)+10
X=X+1
Y=Y-1

Step 6: Check, if X=Y

Goto Step 7

Else

Goto Step 4

Step 7: Stop/Exit.

Example: Demonstrate the Circle Algorithm with circle radius, r = 8 and mid point (xc,yc)=(30,40).

$P_0 = 3 - 2r = 3 - 16 = -13$

Now the initial point $(x_0, y_0) = (0,8)$ and initial calculating terms for calculating decision parameter are

$P_1 = -13 + 4*1 + 6 = -3$ Now $P_1 < 0$,

$P_2 = -3 + 4*2 + 6 = 11$ Now $P_2 < 0$,

$P_3 = 11 + 4*(3-7) + 10 = 5$ Now $P_3 > 0$,

$P_4 = 5 + 4*(4-6) + 10 = 7$ Now $P_4 > 0$.

X+30	Y+40	P	(X+Xc, Y+Yc)
0	8	−13	(30,48)
1	8	−3	(31,48)
2	8	11	(32,48)
3	7	5	(33,47)
4	6	7	(34,46)
5	5	(X=Y)	(35,45)

Plot the graph with these points.

MID POINT ELLIPSE ALGORITHM

The midpoint ellipse drawing algorithm uses the four way symmetry of the ellipse to generate it. The figure(a) shows the four-way symmetry of ellipse. This approach is similar to that used in displaying a raster circle. Here, the quadrant of the ellipse is divided into two regions. The figure(b) shows the division of the first quadrant according to the slope of an ellipse with **rx < ry**. As ellipse is drawn from 90 to 0 degrees ,the x moves in the Positive direction and y moves in the negative direction, and ellipse passes through two regions. It is important to note that while processing first quadrant we have to take steps in the x direction where the slope of the curve has a magnitude less than 1(for region 1) and to take steps in the y direction where the slope has a magnitude greater than 1 (for region 2).

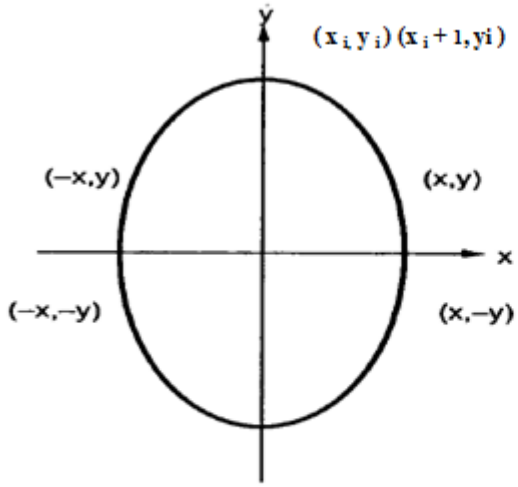
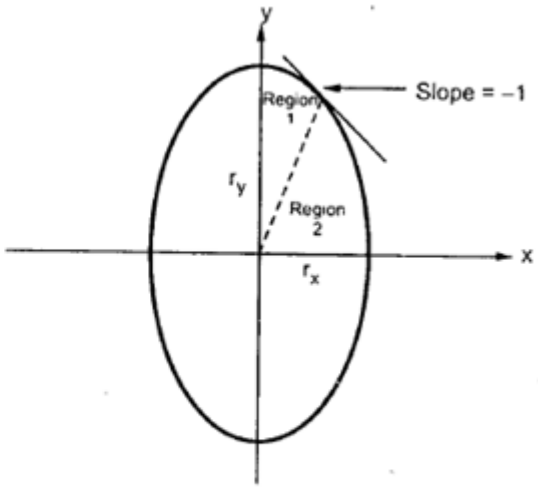


Fig (a). Four way symmetry of Ellipse



Fig(b). Ellipse processing regions

Like circle function, the ellipse function,

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

serves as the decision parameter in the midpoint algorithm. At each sampling position, the next pixel along the ellipse Path is selected according to the sign of the ellipse function evaluated at midpoint between the two candidate pixels (xi + 1 , yi or xi + 1, yi -1 for region 1 and xi, yi - 1 or xi + l, yi - l for region2).

Starting at (0, ry) we have to take unit steps in the x direction until we reach the boundary between region 1 and region 2. Then we have to switch to unit steps in the y direction over the remainder of the curve in the first quadrant. To check for boundary point between region 1 and region 2 we have to test the value of the slope of the curve at each step. The slope of the ellipse at each step is given as

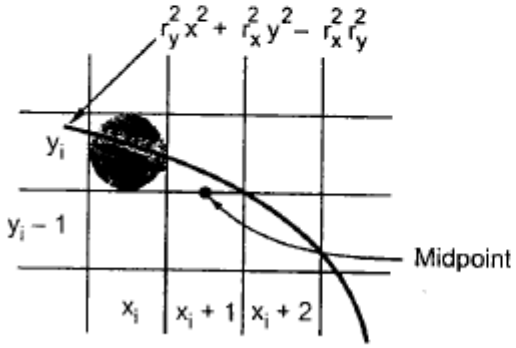
$$\frac{dy}{dx} = -\frac{2r_y^2 x}{2r_x^2 y}$$

At the boundary point between region 1 and region 2, $dy/dx = -1$ and

$$2r_y^2 x = 2r_x^2 y$$

Therefore, when

$$2r_y^2 x \geq 2r_x^2 y$$



Fig(c)

we have to switch to unit steps in the y direction over the remainder of the curve in the first quadrant. The figure(c) shows the midpoint between the two candidate pixels at sampling position xi + 1 in the first region. The next position along the ellipse path can be evaluated by decision parameter at this midpoint.

$$d_{1i} = f_{ellipse}\left(x_i + 1, y_i - \frac{1}{2}\right)$$

$$= r_y^2 (x_i + 1)^2 + r_x^2\left(y_i - \frac{1}{2}\right)^2 - r_x^2 r_y^2$$

If **d1i < 0**, the midpoint is inside the ellipse and the pixel on scan line yi is closer to the ellipse boundary.

If **d1i ≥ 0**, the midpoint is outside or on the ellipse boundary and the pixel on the scan line **yi - 1** is closer to the ellipse boundary.

The incremental calculation of decision parameter of region 1 can be given as

$$d_{1i+1} = f_{ellipse}\left(x_{i+1}+1, y_{i+1}-\frac{1}{2}\right)$$

$$= r_y^2\left[(x_i+1)+1\right]^2 + r_x^2\left(y_{i+1}-\frac{1}{2}\right)^2 - r_x^2\, r_y^2$$

Substituting value of d1i in above expression we get.

$$d_{1i+1} = d_{1i} + 2r_y^2\,(x_i+1) + r_y^2 + r_x^2\left[\left(y_{i+1}-\frac{1}{2}\right)^2 - \left(y_i-\frac{1}{2}\right)^2\right]$$

where y_{i+1} is either y_i or $y_i - 1$, depending on the sign of d_{1i}.

If d_{1i} is negative, i.e. $d_{1i} < 0$, $y_{i+1} = y_i$

$$\therefore \quad d_{1i+1} = d_{1i} + 2r_y^2\, x_{i+1} + r_y^2$$

If d_{1i} is positive or zero, i.e. $d_{1i} \geq 0$, $y_{i+1} = y_i - 1$

$$\therefore \quad d_{1i+1} = d_{1i} + 2r_y^2\, x_{i+1} + r_y^2 - 2r_x^2\, y_{i+1}$$

The terms $2r_y^2 x$ and $2r_x^2 y$ can be incrementally calculated as

$$2r_y^2\, x_{i+1} = 2r_y^2 x_i + 2r_y^2 \text{ and}$$

$$2r_x^2 y_{i+1} = 2r_x^2 y_i - 2r_x^2$$

In region 1, the initial value of the decision parameter can be obtained by evaluating the ellipse function at the start position **(x0, y0) = (0, ry)**.

$$d_{10} = f_{ellipse}\left(1, r_y - \frac{1}{2}\right)$$

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

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

For region 2, we sample at unit steps in the negative y direction, and the midpoint is now taken between horizontal pixels, at each step, as shown in the figure(c). For this region, the decision Parameter is evaluated as

$$d_{2i} = f_{ellipse}\left(x_i + \frac{1}{2}, y_i - 1\right)$$

$$= r_y^2\left(x_i + \frac{1}{2}\right)^2 + r_x^2\,(y_i - 1)^2 - r_x^2\, r_y^2$$

If **d2i > 0**, the midpoint is outside the ellipse boundary, and we select the pixel at **xi**.

If **d2i ≤ 0**, the midpoint is inside or on the ellipse boundary, and we select pixel Position **xi +1** . The incremental decision parameters for region 2 can be given as

$$d_{2i+1} = f_{ellipse}\left(x_{i+1} + \frac{1}{2}, y_{i+1} - 1\right)$$

$$= r_y^2\left(x_{i+1} + \frac{1}{2}\right)^2 + r_x^2\left[(y_i - 1) - 1\right]^2 - r_x^2\, r_y^2$$

Substituting value of expression d2i in above expression we get,

$$d_{2i+1} = d_{2i} - 2r_x^2(y_i - 1) + r_x^2 + r_y^2\left[\left(x_{i+1} + \frac{1}{2}\right)^2 - \left(x_i + \frac{1}{2}\right)^2 2\right]$$

where **x i +1** set either to xi or to **xi + 1** depending on the sign of **d2i**

In region 2, the initial value of the decision parameter can be obtained by evaluating the ellipse function at the last position in the region 1.

$$d_{20} = f_{ellipse}\left(x_0 + \frac{1}{2}, y_0 - 1\right)$$

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

ALGORITHM

Calculating the Initial Decision Parameter

In region **1** the initial value of a decision parameter is obtained by giving starting position = **$(0,r_y)$**.

i.e. $p1_0=r_y^2+1/4r_x^2-r_x^2r_y$

When we enter into a region **2** the initial position is taken as the last position selected in region **1** and the initial decision parameter in region 2 is then:

$p2_0=r_y^2(x_0+1/2)^2+r_x^2(y_0-1)^2-r_x^2r_y^2$

1. Take the input and ellipse centre and obtain the first point on an ellipse centered on the origin as a $(x,y_0)= (0,r_y)$.
2. Now calculate the initial decision parameter in region 1 as:
   $p1_0=r_y^2+1/4r_x^2-r_x^2r_y$
3. At each **x_k** position in region **1** perform the following task. If **$p1_k<0$** then the next point along the ellipse centered on **$(0,0)$** is **(x_k+1,y_k)**.
   i.e. $p1_{k+1}=p1_k+^2r_y^2x_{k+1}+r_y^2$
   Otherwise the next point along the circle is $(x_{k+1},y_k -1)$
   i.e. $p1_{k+1}=p1_k+2r_y^2x_{k+1} - 2r_x^2y_{k+1}+r_y^2$
4. Now, again calculate the initial value in region 2 using the last point **(x_0,y_0)** calculated in a region 1 as : **$p2_0=r_y^2(x_0+1/2)^2+r_x^2(y_0-1)^2-r_x^2r_y^2$**

5. At each y_k position in region 2 starting at **k =0** perform the following task. If $p2_k<0$ the next point along the ellipse centered on **(0,0)** is (x_k , y_{k-1})
   i.e. $p2_{k+1}=p2_k-2r_x^2y_{k+1}+r_x^2$
   Otherwise the next point along the circle will be $(x_{k+1},y_k -1)$
   i.e. $p2_{k+1} =p2_k+2r_y^2x_{k+1} -2r_x^2y_{k+1}+r_x^2$
6. Now determine the symmetric points in another three quadrants.
7. Plot the coordinate value as: $x=x+x_c , y=y+y_c$
8. Repeat the steps for region 1 until $2r_y^2x>=2r_x^2y$.

Example: $r_x = 8$ and $r_y = 6$

- $2r_y^2x = 0$ (with increment $2r_y^2 = 72$)
- $2r_x^2y = 2r_x^2r_y$ (with increment $-2r_x^2 = -128$)
- For region 1: $(x_0, y_0) = (0,6)$
- $P1_0 = r_y^2 - r_x^2r_y + \frac{1}{4}r_x^2 = -332$

$2r_y^2x > 2r_x^2y$

Initial coord for region 2 is (7, 3)

K	$P1_k$	(x_{k+1}, y_{k-1})	$2r_y^2x_{k+1}$	$2r_x^2xy_{k+1}$
0	-332	(1, 6)	72	768
1	-224	(2, 6)	144	768
2	-44	(3, 6)	216	768
3	208	(4, 5)	288	640
4	-108	(5, 5)	360	640
5	288	(6, 4)	432	512
6	244	(7, 3)	504	384

If P1k >= 0 (Xk+1,Yk−1), If P1k < 0 (Xk+1,Yk)

If P2k >= 0 (Xk,Yk−1), If P2k < 0 (Xk+1,Yk−1)

Initial decision parameter for Region 2 is, $P2_0 = -23 < 0$

K	**(Xk,Yk)**	**P2k**	**(Xk+1,YK+1)**
7	(7,3)	−23	(8,2)
P2k+1 = 361 > 0			
8	(8,2)	361	(8,1)
P2k+1 = 297 > 0			
9	(8,1)	297	(8,0)

UNIT II
Scan Converting Lines

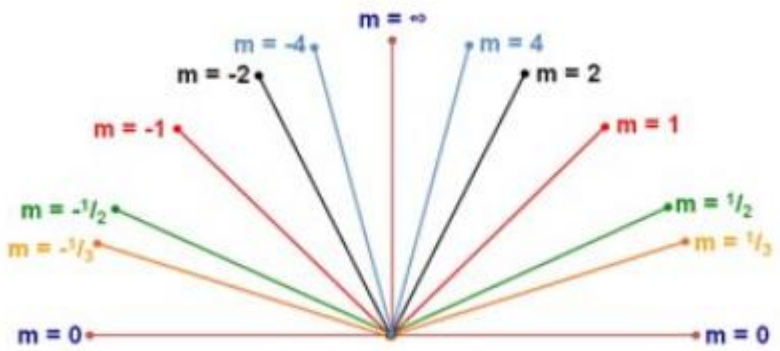
Scan converting a line means to draw pixels on an integer coordinate system in such a way that these pixels are as close to the actual line as possible. To draw a line we need two points, the starting point (x1,y1) of the line and the ending point (x2,y2). x1, y1,x2,y2 represents the coordinates of the points. Mathematically a line can be represented by the following slope intercept form: $y = m x + c$

This equation is used to find any point on the line and is true for all the points on the line. Here

x is the x coordinate of the point.
y is the y coordinate of the point.
m is the slope of the line.
c is the y intercept.

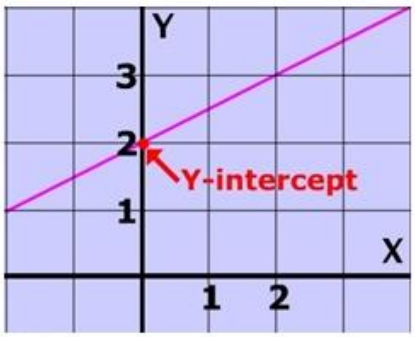
- **m** which is the **slope of the line** represents the angle, the line forms with the horizontal axis. If the angle of incidence is 45 degree then the slope (m) is 1. All lines parallel to x-axis have the m =0, whereas the lines parallel to y-axis have m as infinity (∞).

  m = dy / dx = (y2-y1) / (x2-x1)



Lines whose starting point is to the left of ending point have a positive (+ve) slope i.e. m > 0. And the lines with starting point to the right of ending point have –ve slope i.e. m<0

c which is the y intercept is the point on the y-axis where the line will intersect with this axis. c can be calculated as : c = y1 – m x1. The **c** of the following line is 2.



Polynomial method (Direct method)

This method makes the use of the equation of the line (y = mx +c) to draw a line segment whose end points are A and B. Coordinates of A are (x1,y1) and coordinates of B are (x2,y2). For this line:

Slope m = (y2-y1) / (x2-x1).
y-interept c = y1 – m*x1

For the lines with the value of **m** as **0 <= m <= 1** (Angle of incidence between 0 and 45), we step the value of x by 1 and calculate the corresponding value of y.

$x_{i+1} = x_i + 1$
$y_{i+1} = (m * x_{i+1}) + c$

The pixel (x_{i+1} , round(y_{i+1})) is turned on. This is done while the value of x is <= x2.

For the lines with the value of **m** as **1 < m <** ∞ (Angle of incidence between 46 and 90), we step the value of y by 1 and calculate the corresponding value of x.

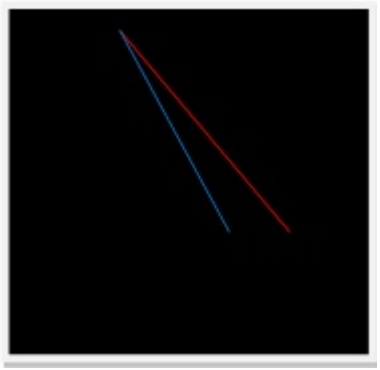
$y_{i+1} = y_i + 1$
$x_{i+1} = (y_{i+1} - c) / m$

The pixel (round(x_{i+1}) , y_{i+1})) is turned on. This is done while the value of y is <= y2.

Disadvantages

This method involves floating point multiplications and division, this takes considerably more time then addition. This makes the method slow.

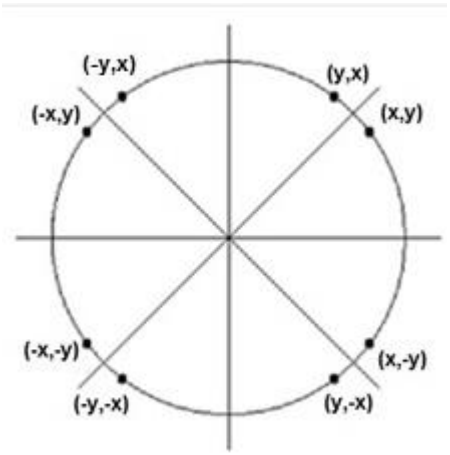
Accumulation of the round off errors may make the line drift away from the actual line. Thus accurate lines may not be produced. In the following picture Blue is the actual line, whereas the line in Red colour is the line drawn with the polynomial method.



Scan Converting a circle

A circle is a geometric figure which is round, and can be divided into 360 degrees. A circle is a symmetrical figure which follows 8-way symmetry.

8-Way symmetry: Any circle follows 8-way symmetry. This means that for every point (x,y) 8 points can be plotted. These (x,y), (y,x), (-y,x), (-x,y), (-x,-y), (-y,-x), (y,-x), (x,-y).
For any point (x+a, y+b), points (x ± a, y ± b) and (y ± a, x ± b) also lie on the same circle. So it is sufficient to compute only 1/8 of a circle, and all the other points can be computed from it.



Drawing a circle: To draw a circle we need two things, the coordinates of the centre and the radius of the circle.

Radius: The radius of a circle is the length of the line from the centre to any point on its edge.

Equation of the circle: For any point on the circle (x,y) and the centre at point (x_c,y_c), the equation of the circle is

$$(x-x_c)^2+ (y-y_c)^2- r^2 = 0$$

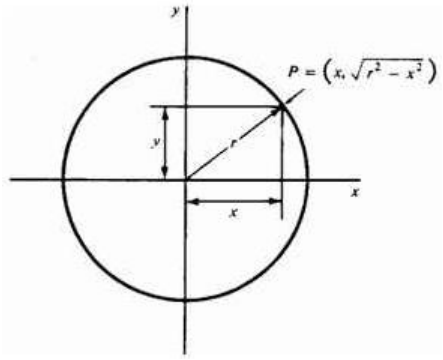
Here **r** is the radius of the circle. If the circle has origin (0,0) as its centre then the above equation can be reduced to

$$x^2 + y^2 = r^2$$

Using polar coordinate system the point on the circle can be represented as:

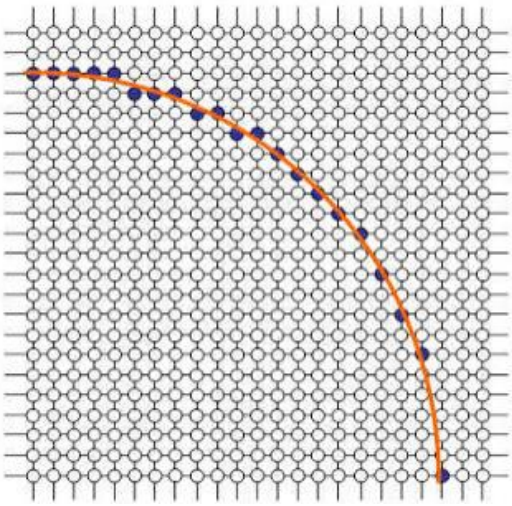
$x = r * \cos \theta$
$y = r * \sin \theta$

θ is the angle which the point makes With x-axis.



<u>Direct Method (Polynomial method) of Scan converting a circle :</u>

From equation of the circle $x^2 + y^2 = r^2$ we can derive that the value of y.
If we increment the value of x from 0 to r, and find the corresponding value of y for each x, we can draw a circle very easily, by plotting the pixels (x,y), (-x,y), (-x,-y), (x,-y).



Advantages :

- This method is easy to understand.
- Fairly easy to implement.

Disadvantages:

- Inefficient method as time required for calculation of square and square root is very large.
- Does not take full advantage of 8-way symmetry.
- Value of y needs to be converted to integer every time.
- The resulting circle has large gaps where the slope approaches the vertical. i.e. Increase in x direction is uniform but the gap in y is not uniform, it increases as the circle approaches the x-axis.

Polar Equations method :

The polar equations of the circle are $x = r * \cos\theta$ $y = r * \sin\theta$

In this method we increment the value of θ from 0 to 2π , and we find the corresponding values of x and y. For every value of (x,y) thus calculated we can plot 7 other points taking advantage of 8-way symmetry.

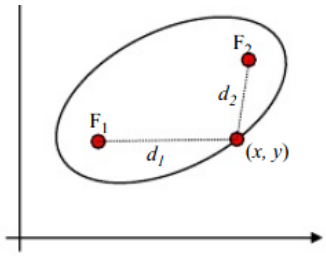
Disadvantages:

- The main issue with this method is that calculation of cos and sin at each step take a lot of time.
- This method is also inefficient, as it takes a lot of time.

Scan Converting Ellipse

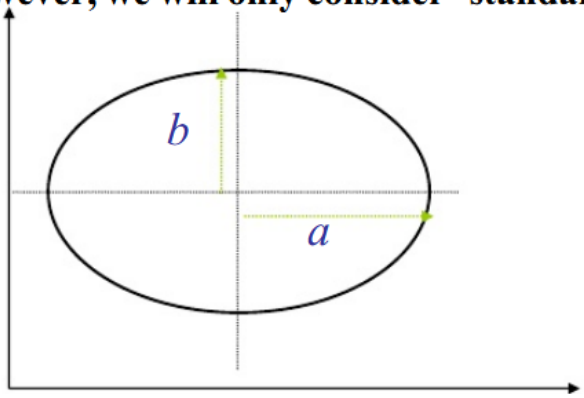
General equation of an ellipse:

$d_1 + d_2 = constant$



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

However, we will only consider 'standard' ellipse:



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

– Formally An ellipse is defined as the locus of points satisfying following equation: r from a fixed point *(h,k)*. This distance is described In the Pythagoras theorem as :

$$\frac{(x-h)^2}{a^2}+\frac{(y-k)^2}{b^2}=1$$

where *(h,k)* is the centre, a and b are the length of major and minor axis respectively

– The standard equation for the ellipse at centre (0,0) is:

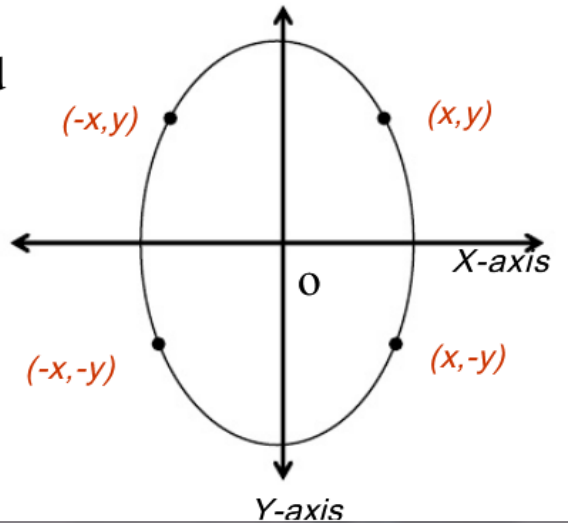
$$\frac{x^2}{a^2}+\frac{y^2}{b^2}=1$$

– So, we can write a simple circle drawing algorithm by solving the equation for y at unit x intervals using:

$$y=b.(\pm\sqrt{1^2-(x/a)^2})$$

Four-Way Symmetry

- Like circle, ellipse centred at (*0, 0*) follows symmetry. But now it is *four symmetry*.
- Thus ellipse points are computed in the first quadrant, rest three of the ellipse points are found by symmetry.
- Centre can be shifted while plotting



So we have to determine the point where the slope is 1. Differentiating the implicit equation $0 = F(x,y) = b^2x^2 + a^2y^2 - a^2b^2$ gives

$$0 = F_x(x,y) + F_y(x,y)\,y' = 2b^2\,x + 2a^2\,y\,y',$$

so we have $y' = 1$ when $a^2\,y + b^2\,x = 0$ and inserting this into the implicit equation gives $a^2b^2x^2 + b^4\,x^2 - a^4b^2 = 0$, i.e. $x = \pm\frac{a^2}{\sqrt{a^2+b^2}}$.

AREA FILLING ALGORITHMS

1) **FLOOD FILL ALGORITHM**
2) **BOUNDARY FILL ALGORITHM**

1) Flood fill algorithm

Flood fill algorithm helps in visiting each and every point in a given area. It determines the area connected to a given cell in a multi-dimensional array.

Implementation of flood fill algorithm: **Bucket Fill in Paint**

Flood fill algorithm fills new color until the old color match.

Flood fill algorithm:-

```
// A recursive function to replace previous
// color 'oldcolor' at  '(x, y)' and all
// surrounding pixels of (x, y) with new
// color 'newcolor' and
floodfill(x, y, newcolor, oldcolor)
1) If x or y is outside the screen, then
   return.
2) If color of getpixel(x, y) is same as
   oldcolor, then
3) Recur for top, bottom, right and left.
    floodFill(x+1, y, newcolor, oldcolor);
    floodFill(x-1, y, newcolor, oldcolor);
    floodFill(x, y+1, newcolor, oldcolor);
    floodFill(x, y-1, newcolor, oldcolor);
```

2) Boundary Fill Algorithm

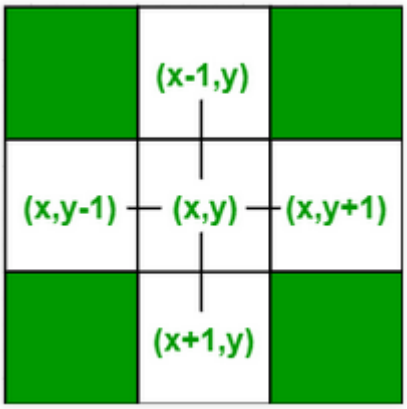
Introduction : Boundary Fill Algorithm starts at a pixel inside the polygon to be filled and paints the interior proceeding outwards towards the boundary. This algorithm works **only if** the color with which the region has to be filled and the color of the boundary of the region are different. If the boundary is of one single color, this approach proceeds outwards pixel by pixel until it hits the boundary of the region.

Boundary Fill Algorithm is recursive in nature. It takes an interior point(x, y), a fill color, and a boundary color as the input. The algorithm starts by checking the color of (x, y). If it's color is not equal to the fill color and the boundary color, then it is painted with the fill color and the function is called for all the neighbours of (x, y). If a point is found to be of fill color or of boundary color, the function does not call its neighbours and returns. This process continues until all points up to the boundary color for the region have been tested.

The boundary fill algorithm can be implemented by 4-connected pixels or 8-connected pixels.

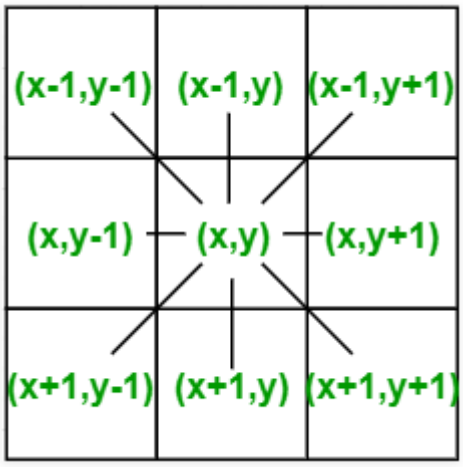
4-connected pixels : After painting a pixel, the function is called for four neighboring points. These are the pixel positions that are right, left, above and below the current pixel. Areas filled by this method are called 4-connected. Below given is the algorithm :

```
void boundaryFill4(int x, int y, int fill_color,int boundary_color)
{
    if(getpixel(x, y) != boundary_color &&
       getpixel(x, y) != fill_color)
    {
        putpixel(x, y, fill_color);
        boundaryFill4(x + 1, y, fill_color, boundary_color);
        boundaryFill4(x, y + 1, fill_color, boundary_color);
        boundaryFill4(x - 1, y, fill_color, boundary_color);
        boundaryFill4(x, y - 1, fill_color, boundary_color);
    }
}
```

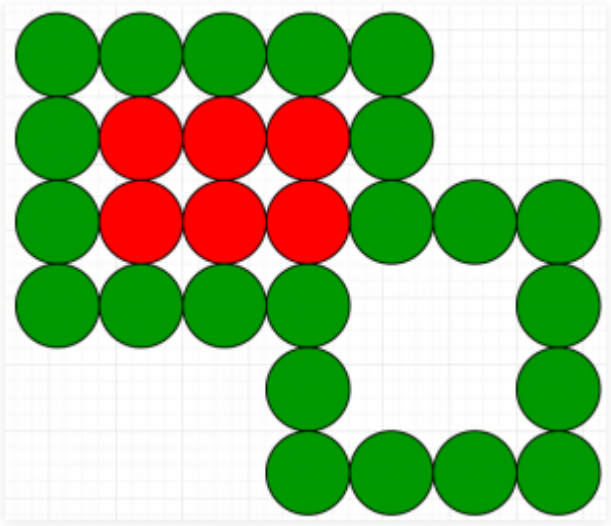


8-connected pixels : More complex figures are filled using this approach. The pixels to be tested are the 8 neighboring pixels, the pixel on the right, left, above, below and the 4 diagonal pixels. Areas filled by this method are called 8-connected. Below given is the algorithm :

```
void boundaryFill8(int x, int y, int fill_color,int boundary_color)
{
    if(getpixel(x, y) != boundary_color &&
       getpixel(x, y) != fill_color)
    {
        putpixel(x, y, fill_color);
        boundaryFill8(x + 1, y, fill_color, boundary_color);
        boundaryFill8(x, y + 1, fill_color, boundary_color);
        boundaryFill8(x - 1, y, fill_color, boundary_color);
        boundaryFill8(x, y - 1, fill_color, boundary_color);
        boundaryFill8(x - 1, y - 1, fill_color, boundary_color);
        boundaryFill8(x - 1, y + 1, fill_color, boundary_color);
        boundaryFill8(x + 1, y - 1, fill_color, boundary_color);
        boundaryFill8(x + 1, y + 1, fill_color, boundary_color);
    }
}
```



4-connected pixels Vs 8-connected pixels :
Let us take a figure with the boundary color as GREEN and the fill color as RED. The 4-connected method fails to fill this figure completely. This figure will be efficiently filled using the 8-connected technique.



Flood fill Vs Boundary fill :
Though both Flood fill and Boundary fill algorithms color a given figure with a chosen color, they differ in one aspect. In Flood fill, all the connected pixels of a selected color get replaced by a fill color. On the other hand, in Boundary fill, the program stops when a given color boundary is found.

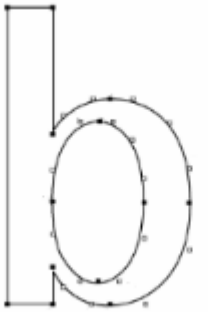
CHARACTER GENERATION

There are three basic methods to generate characters on a computer screen:

(1) hardware-based (2) vector-based and (3) bit map-based methods.

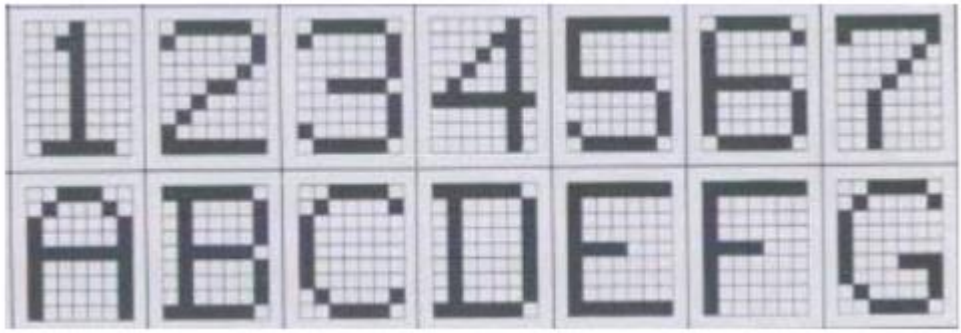
In the hardware-based method, the logic for generating character is built into the graphics terminal. Though the generation time is less the typefaces* are limited due to hardware restrictions.

In the vector-based method the characters are developed using a set of polylines and splines that approximates the character outline. This form of character representation is completely device-independent; memory requirement is less as boldface, italics or different size can be produced by manipulating the curves outlining the character shapes – it doesn't require separate memory blocks for each variation.



Vector-Based Font Generated with Line and Spline Passing Through Control Points

In the bitmap based method small rectangular bitmap called character mask (containing binary values 1 and 0) is used to store pixel representation of each character in a framebuffer area known as font cache. Relative pixel locations corresponding to a character bitmap are marked depending on the size, face and style (font) of character. Size of each character masks range from 5 × 7 to 10 × 12. A single font in 10 different font sizes and 4 faces (normal, bold, italic, bold italic) would require 40 font caches. Characters are actually generated on display by copying the appropriate bitmaps from the frame buffer to the desired screen positions. A mask is referenced by the coordinate of the origin (lower left corner) of the mask w.r.t frame buffer addressing system.



Bitmapped Font

In bit map based method a bold face character is obtained by writing the corresponding ‘normal’ character mask in consecutive frame buffer x-locations. Italics character is produced by necessary skewing of ‘normal’ character mask while being written in the frame buffer. In fact a typeface designer can create from scratch new fonts using a program like Windows Paint. The overall design style (font and face) for a set of characters is called a typeface.

2D Transformations

Introduction

Transformations are fundamental part of computer graphics. In order to manipulate object in two dimensional space, we must apply various transformation functions to object. This allows us to change the position, size, and orientation of the objects. Transformations are used to position objects, to shape objects, to change viewing positions, and even to change how something is viewed.

There are two complementary points of view for describing object movement. The first is that the object itself is moved relative to a stationary coordinate system or background. The mathematical statement of this viewpoint is described by geometric transformations applied to each point of the object. The second point of view holds that the object is held stationary while the coordinate system is moved relative to the object. This effect is attained through the application of coordinate transformations. An example involves the motion of an automobile against a scenic background. We can also keep the automobile fixed while moving the backdrop fixed (a geometric transformation). We can also keep the automobile fixed while moving the backdrop scenery (a coordinate transformation). In some situations, both methods are employed.

Coordinate transformations play an important role in the instancing of an object – the placement of objects, each of which is defined in its own coordinate system, into an overall picture or design defined with respect to a master coordinate system.

Geometric Transformations

An object in the plane is represented as a set of points (vertices). Let us impose a coordinate system on a plane. An object Obj in the plane can be considered as a set of points. Every object point P has coordinates (x, y), and so the object is the sum total of all its coordinate points. If the object is moved to a new position, it can be regarded as a new object Obj' , all of whose coordinate point P' can be obtained from the original points P by the application of a geometric transformation.

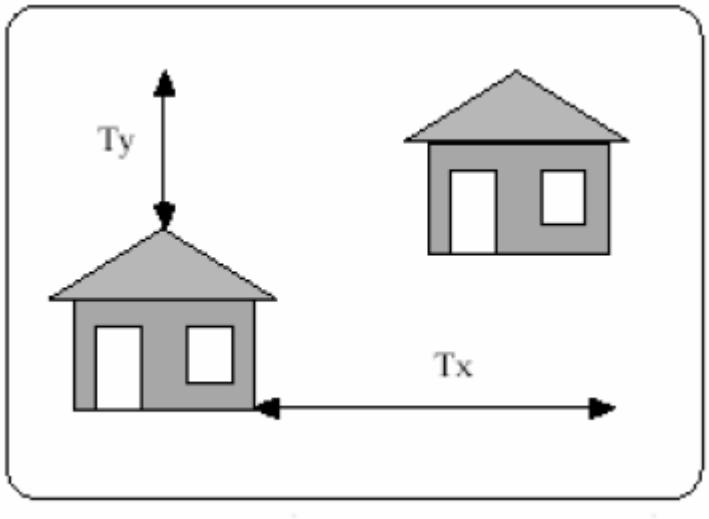


Figure 4.1

Points in 2-dimensional space will be represented as column vectors:
We are interested in three types of transformation:

- Translation
- Scaling
- Rotation
- Mirror Reflection

4.2.1 Translation

In translation, an object is displaced a given and direction from its original position. If the displacement is given by the vector $v\ t\ l\ t\ J_{x\,y} = +$, the new object point P'(x' ,y') can be found by applying the transformation T_v to P(x, y) (see Fig. 4.1).

$P'\ T\ (P)_v =$ where $_x x' = x + t$ and $_y y' = y + t$.

4.2.2 Rotation about the origin

In rotation, the object is rotated θ° about the origin. The convention is that the direction of rotation is counterclockwise if θ is a positive angle and clockwise if θ is a negative angle (see Fig. 4.2). The transformation of rotation $_\theta R$ is

$P'\ R\ (P)_\theta =$

where $x' = x\cos(\theta) - y\sin(\theta)$ and $y' = x\sin(\theta) + y\cos(\theta)$

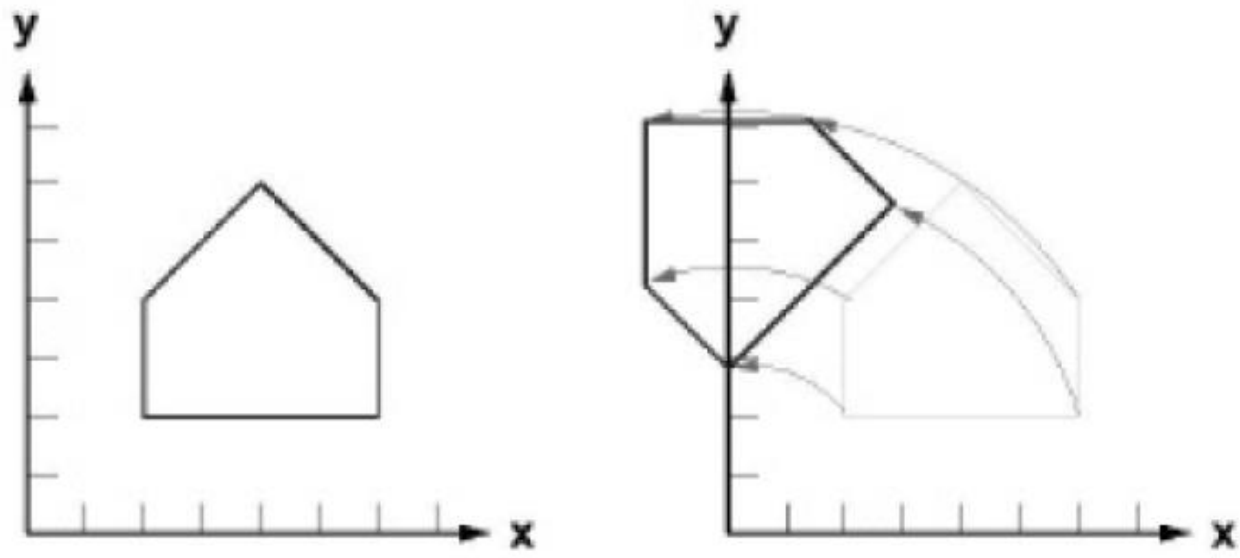


Figure 4.2

4.2.3 Scaling with Respect to the origin

Scaling is the process of expanding or compressing the dimension of an object. Positive scaling constants $_x$S and $_y$S , are used to describe changes in length with respect to the x direction and *y* direction, respectively. A scaling constant greater than one indicates an expansion of length, and less than one, compression of length. The scaling transformation $_{sx\,sy}$S is given by *P*' *S* (*P*) $_{sx\,sy}$ = where x' s .x $_x$ = and y' s .y $_x$ = . Notice that after a scaling transformation is performed, the new object is located at a different position relative to the origin. In fact, in a scaling transformation the only point that remains fixed is the origin (Figure 4.3).

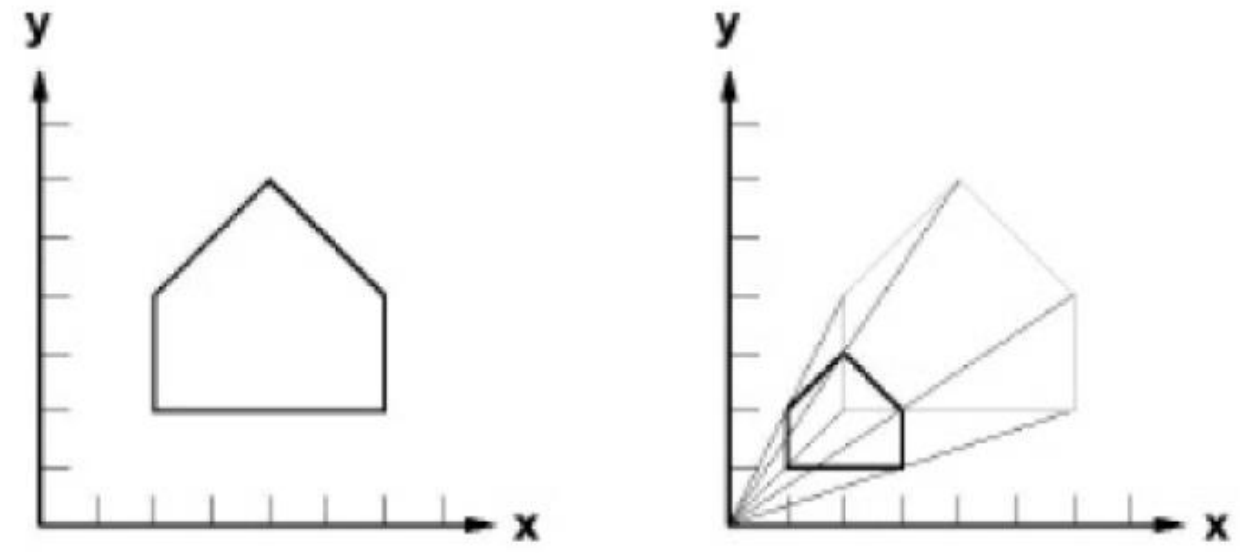


Figure 4.3

If both scaling constants have the same value *s*, the scaling transformation is said to be homogeneous. Furthermore, if s > 1, it is a magnification and for s < 1, a reduction

4.2.4 Mirror Reflection about an Axis

If either the x and y axis is treated as a mirror, the object has a mirror image or reflection. Since the reflection P' of an object point P is located the same distance from the mirror as P (Fig. 4.4), the mirror reflection transformation $_x$M about the x-axis is given by

P' M (P) $_x$ =

where $x' = x$ and $y' = -y$.

Similarly, the mirror reflection about the y-axis is

P' M (P) $_y$ =

where $x' = -x$ and $y' = y$.

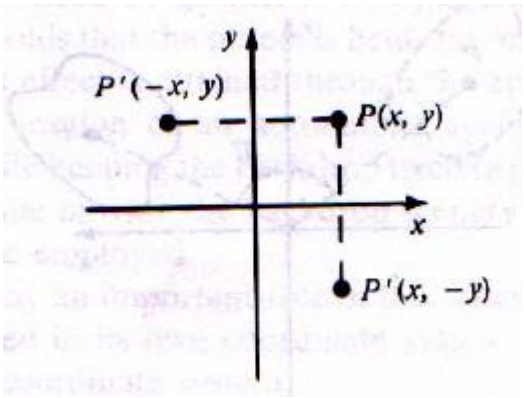


Figure 4.4

4.2.5 Inverse Geometric Transformation

Each geometric transformation has an inverse, which is described by the opposite operation performed by the transformation.

Translation: $T_v^{-1} = T_{-v}$ or translation in the opposite direction

Rotation: $R_\theta^{-1} = R_{-\theta}$ or rotation in the opposite direction

Scaling: $S_{s_x,s_y}^{-1} = S_{1/s_x,1/s_y}$

Mirror reflection: $M_x^{-1} = M_x$ and $M_y^{-1} = M_y$

4.3 Coordinate Transformations

Suppose that we have two coordinate systems in the plane. The first system is located at origin O and has coordinate axes xy figure 4.6. The second coordinate system is located at origin O' and has coordinate axes x' y' Now each point in the plane has two coordinate descriptions: (x, y) or (x', y') , depending on which coordinate system is used. If we think of the second system x'y' as arising from a transformation applied to the first system xy, we say that a coordinate transformation has been applied. We can describe this transformation by determining how the (x' ,y') coordinates of a point P are related to the (x, y) coordinates of the same point.

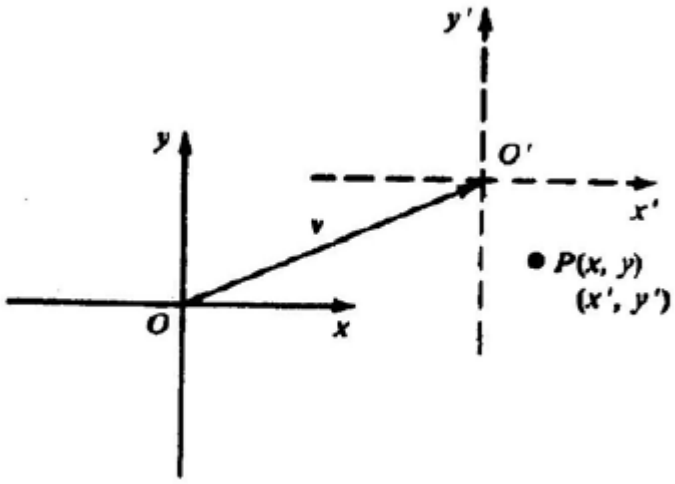


Figure 4.5

4.3.1 Translation

If the xy coordinate system is displaced to a new position, where the direction and distance of the displacement is given by the vector v I J $_{xy} = t + t$, the coordinates of a point in both systems are related by the translation transformation T_v :

(x' ,y') = T_v (x,y)

where $_x x' = x - t$ and $_y$ y'= y − t

4.3.2 Rotation about the Origin

The *xy* system is rotated by θ ° about the origin figure 4.6. Then the coordinates of a point in both systems are related by the rotation transformation R_θ :

(x' ,y') = R_θ(x,y)

where $x' = x\cos(\theta) + y'\sin(\theta)$ and $y' = -x\sin(\theta) + y\cos(\theta)$.

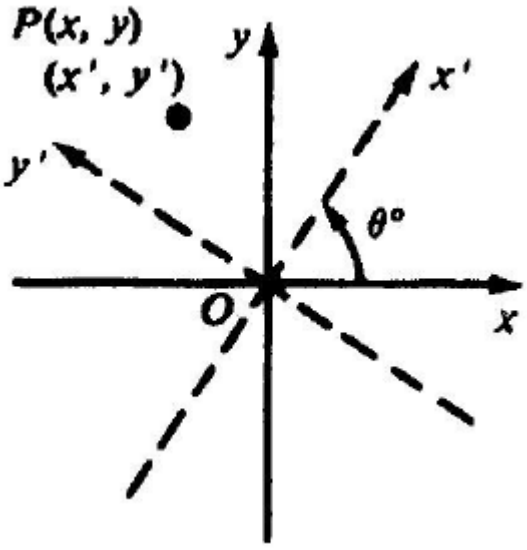


Figure 4.6

4.3.3 Scaling with Respect to the Origin

Suppose that a new coordinate system is formed by leaving the origin and coordinate axes unchanged, but introducing different units of measurement along the x and y axes. If the new units are obtained from the old units by a scaling of $_y$s units along the y-axis, the coordinates in the new system are related to coordinates in the old system through the scaling transformation $S_{s x, sy}$:

(x' ,y') S , (x,y) = s x sy

where $_x$' = 1/ *s x* . *x* and y' 1/ sy .y = . Figure 4.7 shows coordinate scaling transformation using scaling factors $s_x = 2$ and $s_y = 1/2$.

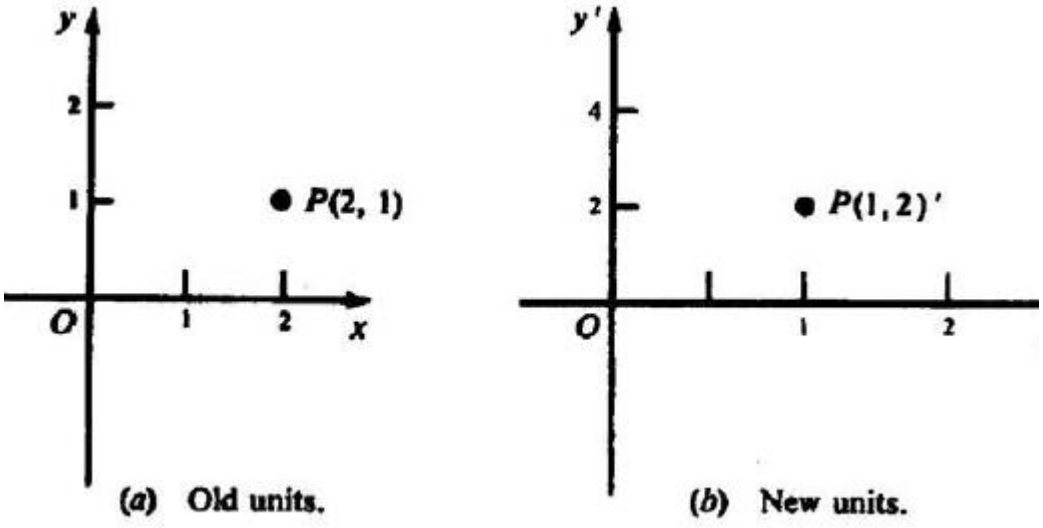


Figure 4.7

4.3.4 Mirror Reflection about an Axis

If the new coordinate system is obtained by reflecting the old system about either *x* or *y* axis, the relationship between coordinates is given by the coordinate transformations M_x and M_y: For reflection about the x axis (figure 4.8 (a))

(x' ,y') = M_x (x,y)

where *x*'= *x* and y'= −y . For reflection about the *y* axis [figure 4.8(b)]

(x' ,y') = M_y (x,y)

where *x*'= −*x* and y'= y .

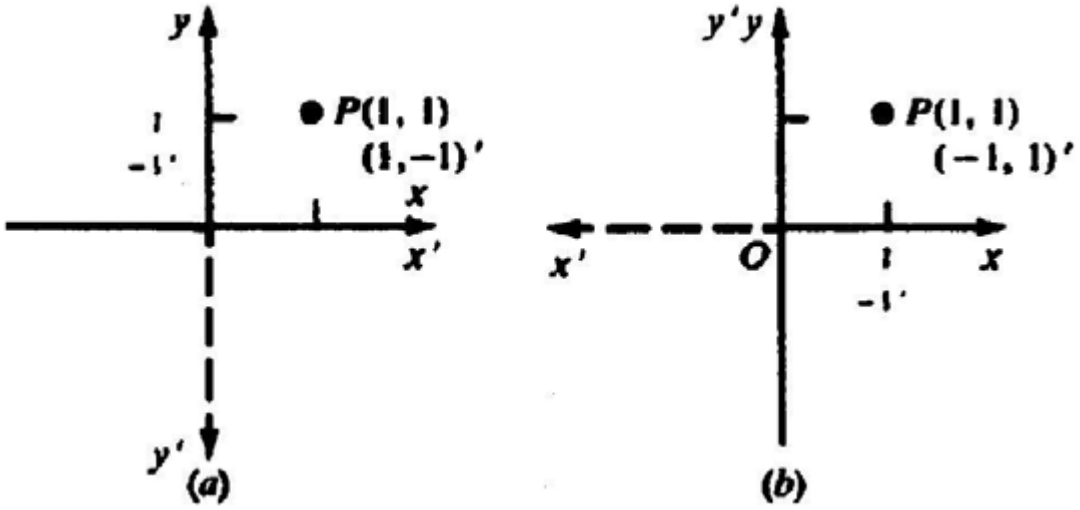


Figure 4.8

Notice that the reflected coordinate system is left-handed; thus reflection changes the orientation of the coordinate system.

4.3.5 Inverse Coordinate Transformation

Each coordinate transformation has an inverse which can be found by applying the opposite transformation:

Translation : $\overline{T}_v^{-1} = \overline{T}_{-v}$ translation in the opposite direction

Rotation: $\overline{R}_\theta^{-1} = \overline{R}_{-\theta}$ rotation in the opposite direction

Scaling : $\overline{S}_{s_x, s_y}^{-1} = \overline{S}_{1/s_x, 1/s_y}$

Mirror reflection: $M_x = \overline{M}_x$ and $M_y = \overline{M}_y$

4.4 Composite Transformations

More complex geometric and coordinate transformations can be built from the basic transformations described above by using the process of composition of functions. For example, such operations as rotation about a point other than the origin or reflection about lines other than the axes can be constructed from the basic transformations.

Matrix Description of the Basic Transformations

The transformations of rotation, scaling, and reflection can be represented as matrix functions:

Geometric Transformations	Coordinate Transformations
$R_\theta = \begin{pmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{pmatrix}$	$\overline{R}_\theta = \begin{pmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{pmatrix}$
$S_{s_x, s_y} = \begin{pmatrix} s_x & 0 \\ 0 & s_y \end{pmatrix}$	$\overline{S}_{s_x, s_y} = \begin{pmatrix} \frac{1}{s_x} & 0 \\ 0 & \frac{1}{s_y} \end{pmatrix}$
$M_x = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$	$\overline{M}_x = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$
$M_y = \begin{pmatrix} -1 & 0 \\ 1 & 1 \end{pmatrix}$	$\overline{M}_y = \begin{pmatrix} -1 & 0 \\ 0 & 1 \end{pmatrix}$

The translation transformation cannot be expressed as a 2 x 2 matrix function. However, a certain artifice allows us to introduce a 3 x 3 matrix function, which performs the translation transformation.

We represent the coordinate pair (x, y) of a point P by the triple (x, y, 1). This is simply the homogeneous representation of P. Then translation in the direction v I J $_{xy}$ = t + t can be expressed by the matrix function.

$$T_v = \begin{pmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{pmatrix}$$

Then

$$\begin{pmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} x + t_x \\ y + t_y \\ 1 \end{pmatrix}$$

From this we extract the coordinate pair $(x + t_x, y + t_y)$.

4.5 Shear Transformation

The shear transformation distorts an object by scaling one coordinate using the other. If distorts the shape of an object in such a way as if the object were composed of internal layers that has been caused to slide over each other is called shear. Two common shearing transformations are those that shift coordinate x values and those that shift y values.

2D Shear along X-direction

Shear in **X** direction is represented by the following set of equations.

$$Sh_x = P_x + hP_y$$
$$Sh_y = P_y$$

where **h** is the negative or positive fraction of **Y** coordinate of **P** to be added to the **X** coordinate. Sh_x can be any real number.

The matrix of form of shear in x-direction is given by

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

(0.1)

2D Shear along Y Direction

Similarly, shear along y-direction is given by

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

(0.2)

Combining the shear in **X** and **Y** directions,

$$Sh_x = P_x + hP_y$$
$$Sh_y = gP_y + P_y$$

where **g** is a non-zero fraction of to be added to the **Y** coordinate

General matrix form of shear

The general matrix form of shear is

$$\begin{bmatrix} 1 & g \\ h & 1 \end{bmatrix}$$

(0.3)

Shear will reduce to a pure shear in the y-direction, when **h=0**.

The inverse of Shear is given by

$$\left[Sh^{-1} = frac11 - gh\begin{bmatrix} 1 & -g \\ -h & 1 \end{bmatrix} \right]$$

(0.4)

For Example,

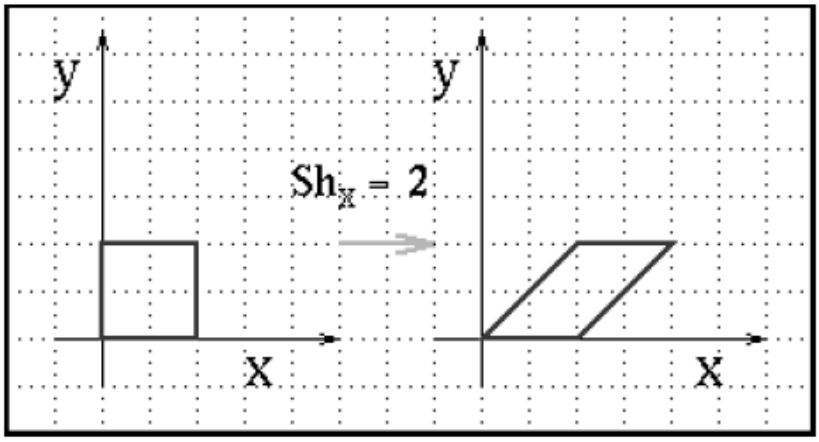
If h=0,5 g=0.8, then shear along X direction of the point P : (8,9) is obtained by substituting these values in (0.3).

$$Sh_x = 8 + (0.5 \times 9)9 = 12.5, 9 \qquad (0.5)$$

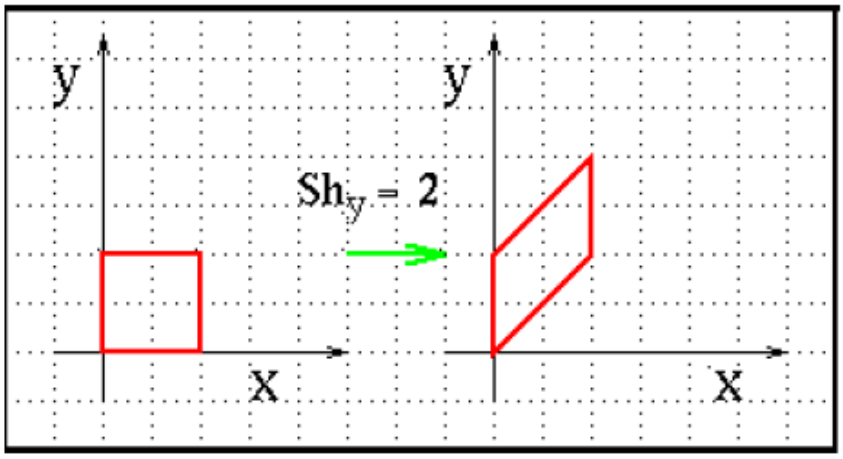
Shear in **Y** direction is

$$Sh_y = 8 + (0.5 \times 9)(0.8 \times 8) + 9 = 12.5, 15.4 \qquad (0.6)$$

$$Sh_x = 2, (2)\ Sh_y = 2$$



2 Consider a square of side = 2. Show the effect of shear when (1)



4.6 Summary

- Transformation is a process carried out by means of transformation to these object or changing the orientation of the object or may be combination of these.
- In translation, an object is displaced a given and direction from its original position
- If the new coordinate system is obtained by reflecting the old system about either *x* or *y* axis, the relationship between coordinates is given by the coordinate transformations
- Scaling is the process of expanding or compressing the dimension of an object
- Multiplying the basic matrix transformations can do complex transformations
- Shear transformation distorts an object by scaling one coordinate using the other in such a way as if the object were composed of internal layers that has been caused to slide over each other.

WINDOW VIEWPORT TRANSFORMATIONS

5.1 Introduction

In very basic two dimensional graphics usually use device coordinates. If any graphics primitive lies partially or completely outside the window then the portion outside will not be drawn. It is clipped out of the image. In many situations we have to draw objects whose dimensions are given in units completely incompatible with the screen coordinates system. Programming in device coordinates is not very convenient since the programmer has to do any required scaling from the coordinates natural to the application to device coordinates. This has led to two dimensional packages being developed which allow the application programmer to work directly in the coordiate system which is natural to the application. These user coordinates are usually called **World Coordinates (WC).** The packages then coverts the coordinates to **Device Coordinates (DC)** automatically. The transformation form the WC to DC is often carried out in tow steps. First using the **Normalisation Transformation** and then the **Workstation Transformation**. The **Viewing Transformation** is the process of going form a window in World coordinates to viewport in **Physical Device Coordinates (PDC).**

5.2 Window-to-Viewport Mapping

A window is specified by four world coordinates : wx_{min}, wx_{max}, wy_{min},and wy_{max} (see Fig. 5.1) Similarly, a viewport is described by four normalized device coordinates: vx_{min},vx_{max},vy_{min}, and vy_{max}. The objective of window – to – viewport mapping is to convert the world coordinates (wx, wy) of an arbitrary point to its corresponding normalized device coordinates (vx,vy). In order to maintain the same relative placement of the point in the viewport as in the window, we require:

$$\frac{wx - wx_{min}}{wx_{max} - wx_{min}} = \frac{vx - vx_{min}}{vx_{max} - vx_{min}} \quad \text{and} \quad \frac{wy - wy_{min}}{wy_{max} - wy_{min}} = \frac{vy - vy_{min}}{vy_{max} - vy_{min}}$$

Thus

$$\begin{cases} vx = \dfrac{vx_{max} - vx_{min}}{wx_{max} - wx_{min}}(wx - wx_{min}) + vx_{min} \\ vy = \dfrac{vy_{max} - vy_{min}}{wy_{max} - wy_{min}}(wy - wy_{min}) + vy_{min} \end{cases}$$

Since the eight coordinate values that define the window and the viewport are just constants, we can express these two formulas for computing (vx, vy) from (wx, wy) in terms of a translate-scale-translate transformation N

$$\begin{pmatrix} vx \\ vy \\ 1 \end{pmatrix} = N. \begin{pmatrix} wx \\ wy \\ 1 \end{pmatrix}$$

where

$$N = \begin{pmatrix} 1 & 0 & vx_{min} \\ 0 & 1 & vy_{min} \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \frac{vx_{max} - vx_{min}}{wx_{max} - wx_{min}} & 0 & 0 \\ 0 & \frac{vy_{max} - vy_{min}}{wy_{max} - wy_{min}} & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & -wx_{min} \\ 0 & 1 & -wy_{min} \\ 0 & 0 & 1 \end{pmatrix}$$

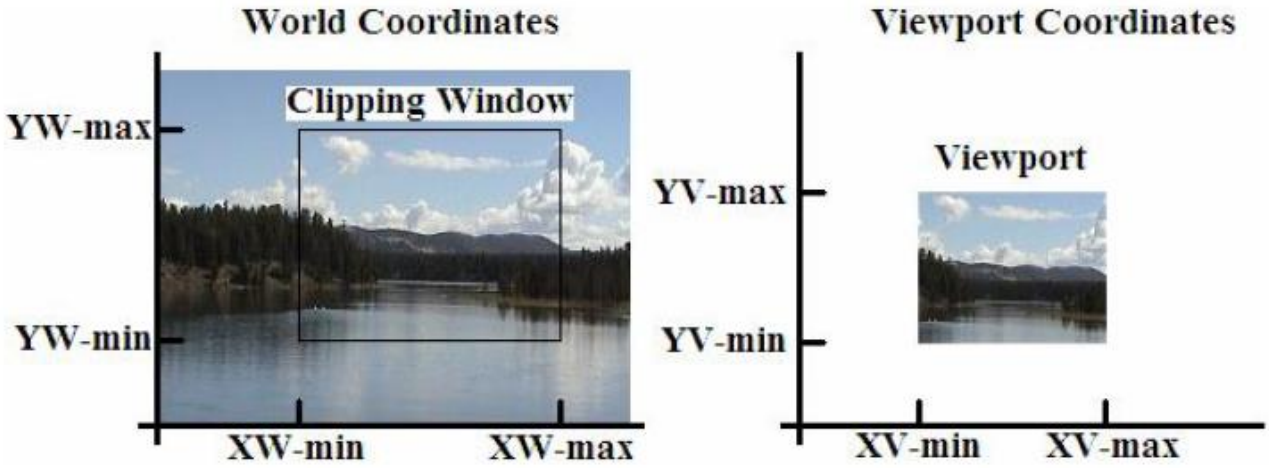


Fig. 5.1: Window-to-viewport mapping

Note that geometric distortions occur (e.g. squares in the window become rectangles in the viewport) whenever the two scaling constants differ.

5.3 Two – Dimensional Viewing and Clipping

Much like what we see in real life through a small window on the wall or the viewfinder of a camera, a Computer-generated image often depicts a partial view of a large scene. Objects are placed into the scene by modeling transformations to a master coordinate system, commonly referred to as the world coordinate system (WCS). A rectangular window with its edge parallel to the axes of the WCS is used to select the portion of the scene for which an image is to be generated (see Fig. 5.2). Sometimes an additional coordinate system called the viewing coordinate system is introduced to simulate the effect of moving and / or tilting the camera.

On the other hand, an image representing a view often becomes part of a larger image, like a photo on an album page, which models a computer monitor's display area. Since album pages vary and monitor sizes differ from one system to another, we want to introduce a device-independent tool to describe the display area. This tool is called the normalized device coordinate system (NDCS) in which a unit (1 x 1) square whose lower left corner is at the origin of the coordinate system defines the display area of a virtual display device. A rectangular viewport with its edges parallel to the axes of the NDCS is used to specify a sub-region of the display area that embodies the image.

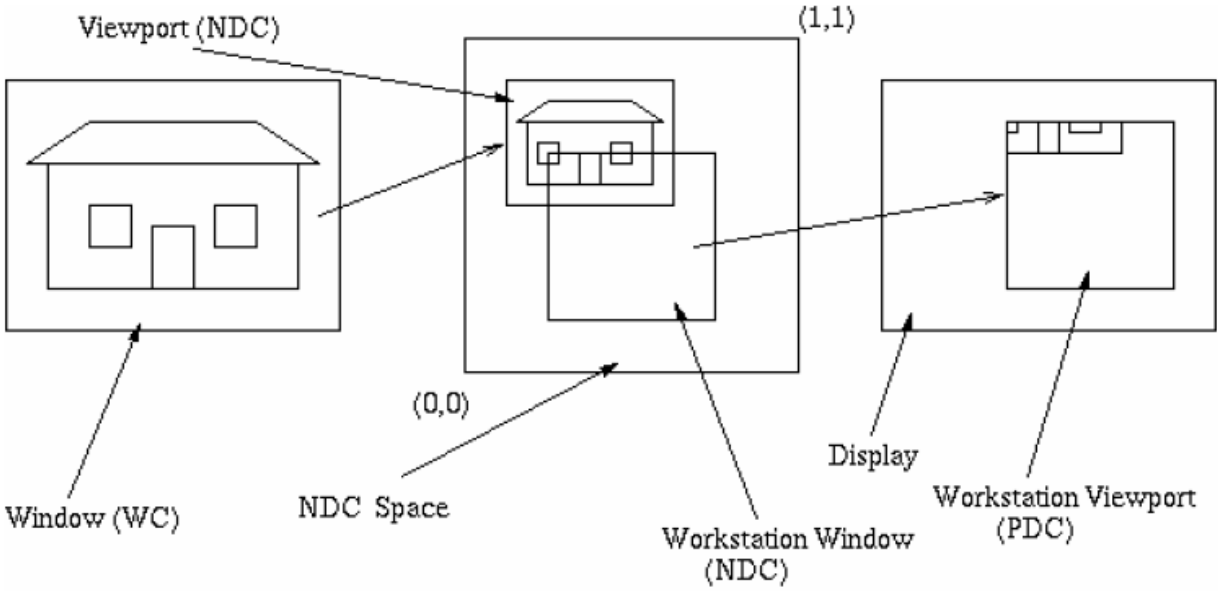


Fig. 5.2: Viewing transformation

The process that converts object coordinates in WCS to normalized device coordinate is called window–to– viewport mapping or normalization transformation. The process that maps normalized device coordinates to Physical Device Co-ordinates (PDC) / image coordinates is called work, station transformation, which is essentially a second windowto-viewport mapping., with a workstation window in the normalized device coordinate system and a workstation viewport in the device coordinate window in the normalized device coordinate system and a workstation viewport in the device coordinate system. Collectively, these two coordinate mapping operations are referred to as viewing transformation.

Workstation transformation is dependent on the resolution of the display device/frame buffer. When the whole display area of the virtual device is mapped to a physical device that does not have a 1/1 aspect ratio, it may be mapped to a square sub-region (see fig. 5.2) so as to avoid introducing unwanted geometric distortion.

Along with the convenience and flexibility of using a window to specify a localized view comes the need for clipping, since objects in the scene may be completely inside the window, completely outside the window, or partially visible through the window. The clipping operation eliminates objects or portions of objects that are not visible through the window to ensure the proper construction of the corresponding image.

Note that clipping may occur in the world coordinate or viewing coordinate space, where the window is used to clip the objects; it may also occur in the normalized device coordinate space, where the viewport/workstation window is used to clip. In either case we refer to the window or the viewport/workstation window as the clipping window.

5.8 Window-To-Viewport Coordinate Transformation

Once object descriptions have been transferred to the viewing reference frame, we choose the window extents in viewing coordinates and select the viewport limits in normalized coordinates. Object descriptions are then transferred to normalized device coordinates. We do this using a transformation that maintains the same relative placement of objects in normalized space as they had in viewing coordinates. If a coordinate position is at the center of the viewing window, for instance, it will be displayed at the center of the viewport.

$$\frac{xv - xv_{min}}{xv_{max} - xv_{min}} = \frac{xw - xw_{min}}{xw_{max} - xw_{min}}$$

$$\frac{yv - yv_{min}}{yv_{max} - yv_{min}} = \frac{yw - yw_{min}}{yw_{max} - yw_{min}}$$

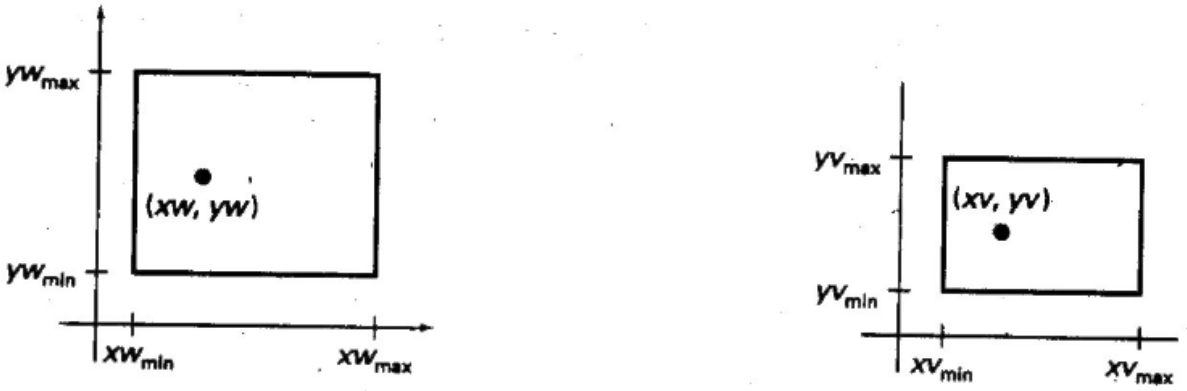


Figure 5.12: A point at position (xm, yw) in a designated window is mapped to viewport coordinates

(xv, yv) so that relative positions in the two areas are the same.

Figure 5.12 illustrates the window-to-viewport mapping. A point at position (xw, yw) in the window is mapped into position (xv, yv) in the associated view-port. To maintain the same relative placement in the viewport as in the window, we require that

$$xv = xv_{min} + (xw - xw_{min})sx$$

$$yv = yv_{min} + (yw - yw_{min})sy$$

where the scaling factors are

$$sx = \frac{xv_{max} - xv_{min}}{xw_{max} - xw_{min}}$$

$$sy = \frac{yv_{max} - yv_{min}}{yw_{max} - yw_{min}}$$

Above equations can also be derived with a set of transformations that converts the window area into the viewport area. This conversion is performed with the following sequence of transformations:

1. Perform a scaling transformation using a fixed-point position of (xw_{min}, yw_{min}) that scales the window area to the size of the viewport.
2. Translate the scaled window area to the position of the viewport.

Relative proportions of objects are maintained if the scaling factors are the same (sx = sy). Otherwise, world objects will be stretched or contracted in either x or y direction when displayed on the output device.
Character strings can be handled in two ways when they are mapped to a viewport. The simplest mapping maintains a constant character size, even though the viewport area may be enlarged or reduced relative to the window. This method would be employed when text is formed with standard character fonts that cannot be changed. In systems that allow .for changes in character size, string definitions can be windowed the same as other primitives. For characters formed with line segments, the mapping to the viewport can be carried out as a sequence of line transformations.
From normalized coordinates, object descriptions are mapped to the viewport display devices. Any number of output devices can be open in a particular application, and another window-to-viewport transformation can be performed for each open output device. This mapping, called the workstation transformation, is accomplished by selecting a window area in normalized space and a viewport area in the coordinates of the display device. With the workstation transformation, we gain some additional control over the positioning of parts of a scene on individual output devices. As illustrated in Fig. 5.13, we can use work station transformations to partition a view so that different parts of normalized space can be displayed on different output devices.

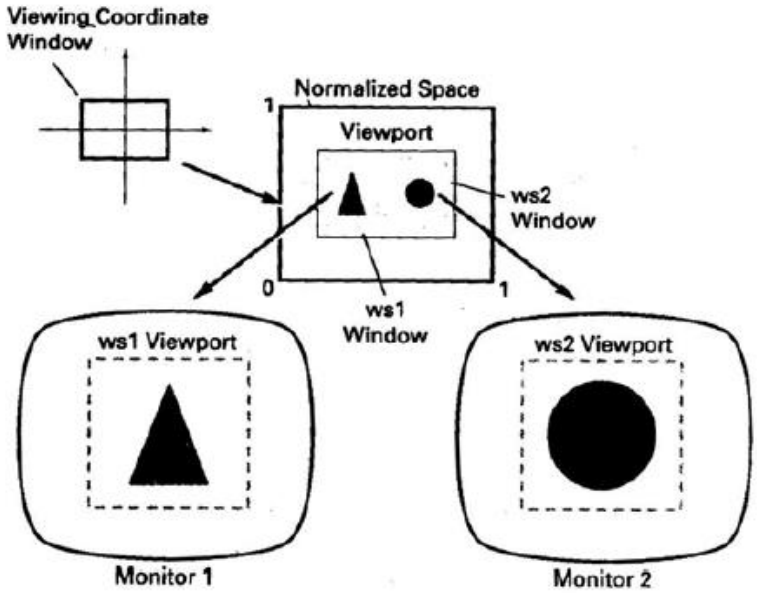


Figure 5.13: Mapping selected parts of a scene in normalized coordinated to different video monitors with workstation transformations.

Two Dimensional Concepts
Clipping Algorithms

Clipping:

Clipping is defined as the identification of the objects of the view which are outside the clipping region and which can be removed or clipped from the viewing window. Any procedure that identifies those portion of a picture that are either inside or outside of a specified region of space is referred to as a clipping Algorithm or clipping.

The region against which the object needs to be clipped is known as clip window. We will assume here that clip window is a rectangular window but it might be a polygon shaped as well and can have boundaries in the curved form as well. Hence the objects which are not inside and are outside the rectangular clip window are hence discarded. The various clipping algorithms which are involved in the process of clipping are as follows:

Point Clipping

Line Clipping

Polygon clipping

Text Clipping

Curve Clipping

Here are a few examples of the application of the clipping concepts which are as follows:

(1) Creating objects using solid-modeling procedures.

(2) Drawing and painting operations.

(3) Identifying visible surface in three dimensional views.

(4) Antialising line segments or object boundaries.

(5) Extracting parts of defined scene for viewing.

(6) Displaying multi window environment.

1) Point Clipping.

Assuming that a point P(x,y) is to be displayed on the screen we need to determine if this point lies within the clip window or not. Assuring that

clip window is a rectangle in standard Position, we save a point P = (x, y) for display if following inequalities are satisfied. And we have to compare the point coordinates with window coordinates.

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

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

then the point (x,y) lies within the view window and can be displayed, otherwise it needs to be discarded. Where the edges of clip window (xw_{min}, xw_{max}, yw_{min}, yw_{max}) can be either coordinate window boundaries or view port boundaries. If any one of these inequalities is not satisfied the point is clipped.

Application of Point Clipping: Point clipping can be applied to scenes involving explosions or sea foam that are modeled with particles (points) distributed in some region of the scene.

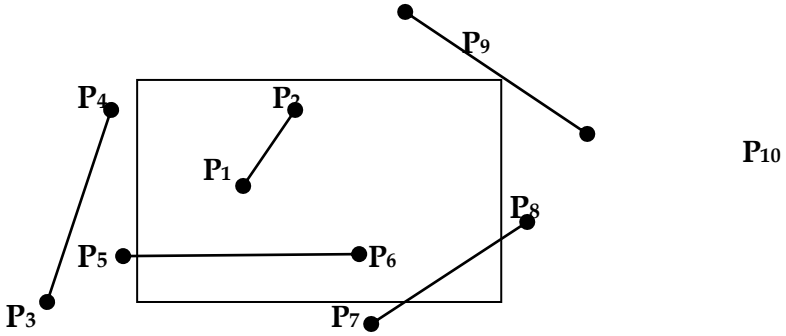
2) Line Clipping:

A line consists of a sequence of number of points arranged between the two end points. Here we just have to consider only the endpoints for clipping purpose, and we would not consider the points between the endpoints. A line clipping procedure involves several parts First, we can test a given line segment to determine whether it lies completely inside the clipping window.

If the two endpoints of a line fall within the clip window, it is accepted. And if in case one of the line end point falls inside and the other end point goes out of the clip window, we need to make calculation of the

intersection of the line with the edges of the rectangular window. If does not, we try to determine whether it lies completely outside the window.

Finally if we can not identify a line as completely inside or completely outside we must perform intersection calculation with one or more clipping boundaries. We process lines through inside-outside tests by checking the line end points.



A line with both end points outside any one of the clip boundaries (line P_3, P_4 in Figure.) is outside the window.

A line with both end points inside all clipping boundaries such as line form P_1 to P_2 is saved.

And all the other lines that cross one or more clipping boundaries and may require calculation of multiple intersection point.

For a line segment with end points (x_1, y_1) and (x_2, y_2) and one or both end points outside clipping rectangle, the parametric representation.Could be used to determine values of parameter u for intersections with the clipping boundary coordinates.

$$x = x_1 + u(x_2 - x_1)$$

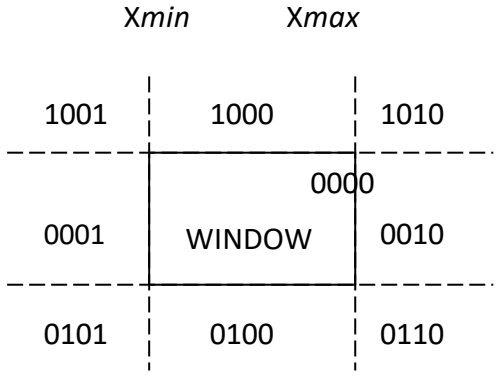
$$y = y_1 + u(y_2 - y_1),\ 0 \le u \le 1$$

If the value of u for an intersection with a rectangle boundary edge is outside the range 0 to 1, the line does not enter the interior of the window at that boundary. If the value of u is within the range from 0 to 1, the line segment does indeed cross into the clipping area.

Cohen Sutherland Line Clipping:

This algorithm also reduces calculations by the identification of the lines which can be trivially discarded or accepted. And this can be done by making comparison with the endpoints with the window coordinates (Xmin, Ymin) and (Xmax, Ymax). This is one of the oldest and most popular line clipping procedures. Generally, the method speeds up the processing of line segments by performing initial test that reduces the number of intersections that must be calculated.

Every line end point in a picture is assigned a four digit binary code called, a region code, that identifies the location of the points relative to the boundaries of the clipping rectangle. For this purpose we assign 4-digit binary code to each endpoint of the line. As shown in the following diagram, we have extended the window to get a plane of nine regions.



Code for inside the window region is 0000. Each regions is defined or represented by 1 bit. First bit from left i.e., MSB is for the region above the top edge. If this bit is 1, it means point is above the top edge or $y > y_{max.}$ Each bit position in the region code is used to indicate one of the four relative coordinate positions of the point with respect to the clip window: to the left, right, top and bottom.

Second bit from left is for the region below the bottom

By numbering the bit position in the region code as 1 through 4 right to left, the coordinate regions can be correlated with the bit positions as :

bit 1 : left ; bit 2 : right ; bit 3 : below ; bit 4 : above

A value of 1 in any bit position indicates that point is in that relative position otherwise the bit position is set to 0.

Now here bit values in the region are determined by comparing end point coordinate values (x, y) to the clip boundaries.

Bit 1 is set to 1 if $x < xw_{min}$

Bit 2 is set to 1 it $xw_{max} < x$

Now

Bit 1 sign bit of $x - xw_{min}$

Bit 2 sign bit of $xw_{max} - x$

Bit 3 sign bit of $y - yw_{min}$

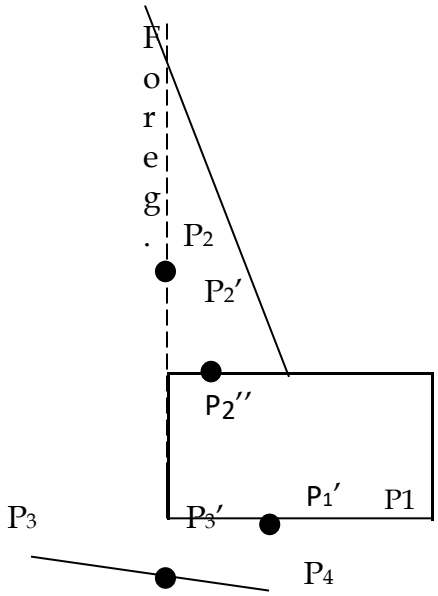
Bit 4 sing bit of $yw_{max} - y$

(1) Any lines that are completely inside the clip window have a region code 0000 for both end points few points to be kept in mind while checking.

(2) Any lines that have 1 in same bit position for both end points are considered to be completely outside.

(3) Now here we use AND operation with both region codes and if result is not 0000 then line is completely outside.

Now for lines that cannot be identified as completely inside or completely outside the window by this test are checked by intersection with window boundaries.



We check P_1 against left right we find it is below window. We find intersection point P_1' and then discard line section P_1 to P_1' & Now we have P_1 to P_2. Now we take P_2 we find it in left position outside window then we take an intersection point P_2''. But we find it outside window then we again calculate final intersection point P_2''. Now we discard line P_2 to P_2''. We finally get a line P_2'' to P_1' inside window similarly check for line P_3 to P_4.

For end points (x_1, y_1) (x_2, y_2) y–coordinate with vertical boundary can be calculated as

$y = y_1 + m(x - x_1)$ where x is set to xw_{min} to xw_{max}

Liang-Barsky Line Clipping

Express line segment in parametric form:

x = x1 + (x2 - x1) * m = x1 + dx * m 0.0 < m < 1.0

y = y1 + (y2 - y1) * m = y1 + dy * m

when m = 0.0 => x1, y1

when m = 1.0 => x2, y2

Any point P(x, y) on line segment which is inside window satisfies:

Xwmin <= x <=xwmax and Ywmin <= y <= ywmax

(Note that if Xwmin = x implies an intersection of the line with left boundary)

or in Parametric form

(1) Xwmin <= x1 + dx*m <= xwmax

(2) Ywmin <= y1 + dy*m <= ywmax

Now (1) can be rewritten as

-dx * m <= x1 xwmin left boundary

dx * m < xwmax x1 right boundary

similarly (2) can be written as

-dy * m < y1 ywmin bottom boundary

dy * m < ywmax y1 top boundary

Above are all of the form:

Pi * m <qi i="1," 2, 3, 4

where:

P1 = -dx q1 = x1 - Xwmin -- Left

P2 = dx q2 = Xwmax - x1 -- Right

P3 = -dy q3 = y1 - Ywmin -- Bottom

P4 = dy q4 = Ywmax - y1 -- Top

ASIDE

Now note that if a line is parallel to Left / Right boundary then

dx = 0 -> P1 = P2 = 0

Similarly if a line is parallel to Top / Bottom then

dy = 0 -> P3 = P4 = 0

Now if P1 = 0

if (q1 = x1 - Xwmin) <0 then x1 < xwmin and line is outside of window

then reject

Similarly:

if P2 = 0 and (q2 = Xwmax - x1) < 0 then x1> Xwmax -> reject.

if P3 = 0 and (q3 = y1 - Ywmin) < 0 then y1 <& ywmin> reject.

if P4 = 0 and (q4 = Ywmax - y1) < 0 then y1> Ywmax -> reject.

So as a general rule: if Pi = 0 and qi < 0 reject the lines, else retain lines for further consideration.

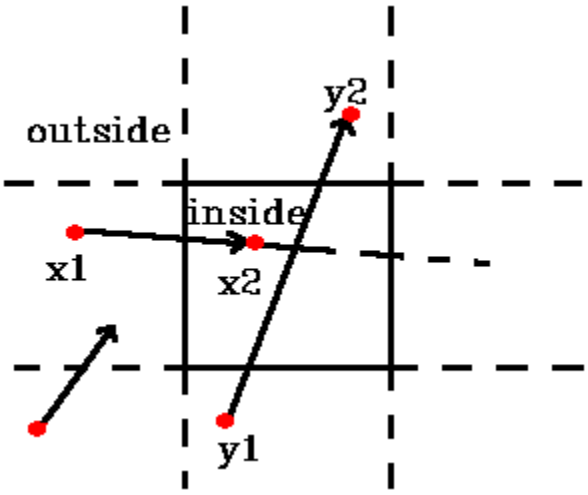
Look at case

P1 = -dx < 0

-dx = - (x2 - x1) < 0

x1 - x2 < 0 x1 < x2

so if extend line segment it goes from Left to Right or from outside of Left boundary to inside (see figure)



Now if P1 < 0 then p2> 0, so extended line segment goes from inside of Right boundary to outside. Similarly if P3 = -dy = -(y2 - y1) = (y1 - y2) <0 then y1 < y2 and line goes from outside of bottom boundary to inside and if P4 > 0, line goes from inside of top boundary to outside.

Another way of looking at above.

General inequality :

Pi * m < qi> mi < qi / pi

if Pi < 0 (outside> inside) -> mi > qi / |Pi|

so point of intersection (mi = qi / Pi) is the minimum value for which the line is on the visible side of the boundary. Since m increases along line, the

direction of the line is from the OUTSIDE (invisible) to the INSIDE (visible).

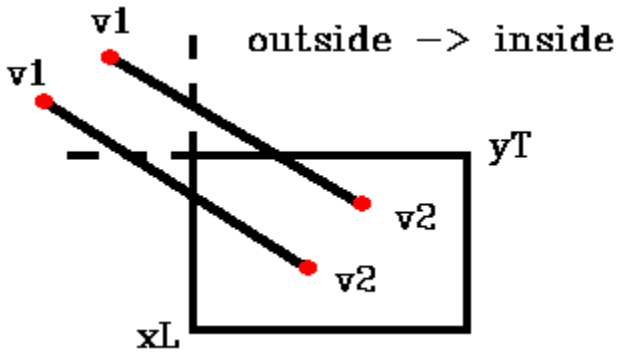
Similarly if Pi > 0 -> mi <* qi / pi or point of intersection is maximum value of mi. then the direction of line is from inside to outside.

Now remember that for our line segment: 0.0 < m < 1.0

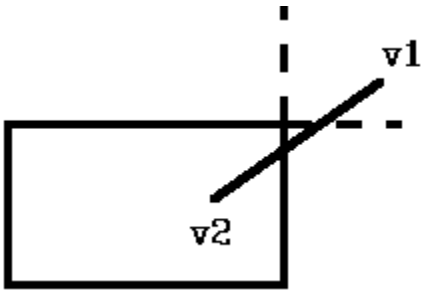
Now look at lines proceeding from outside to inside

Compute intersection at XL, YT

The visible part of line starts at largest such value of m.



same is true, visible portion starts at largest value of m



Remember: m > 0.0 so can express as (for cases of Pi < 0 i.e. outside>inside)

m1 = MAX ({qi / Pi | Pi < 0, i="1," 2, 3, 4} U {0})

Reverse above endpoints for Pi > 0 and get

m2 = MIN ({qi / Pi | Pi > 0, i = 1, 2, 3, 4} U {1})

Now if there is a visible segment it corresponds to the parametric interval

m1 <= m <= m2 and m1 <= m2

So if m1 > m2 reject line else compute visible endpoints from m1, m2.

EXAMPLE :

Let P1 (-1, -2), P2 (2, 4)

XL = 0, XR = 1, YB = 0, YT = 1

dx = 2 - (-1) = 3 dy = 4 - (-2) = 6

P1 = -dx = -3 q1 = x1 - XL = -1 - 0 = -1 q1 / P1 = 1/3

P2 = dx = 3 q2 = XR - x1 = 1 - (-1) = 2 q2 / P2 = 2/3

P3 = -dy = -6 q3 = y1 - YR = -2 q3 / P3 = 1/3

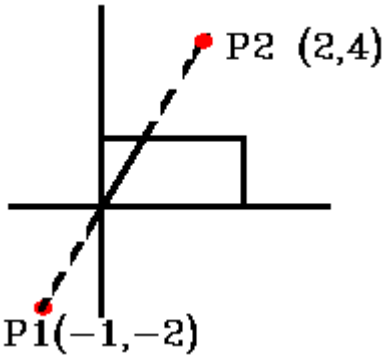
P4 = dy = 6 q4 = YT - y1 = 3 q4 / P4 = 1/2

for (Pi < 0) t1="MAX" (1 / 3, 1 / 3, 0)="1" / 3

for (Pi > 0) t2 = MIN (2 / 3, 1 / 2, 1) = 1 / 2

Since t1 < t2 there is a visible section

compute new endpoints
t1 = 1 / 3 x1' = x1 + dx . t1
= -1 + (3 . 1 / 3) = 0
y1' = y1 + dy . t1 = -2 + (6 .
1 / 3) = 0
t2 = 1 / 2 x2' = x1 + dx . t2
= -1 + (3 . 1 / 2) = 1 / 2
y2' = y1 + dy . t2 = -1 + (6 .
1 / 2) = 1



Polygon-Clipping Algorithms

Sutherland-Hodgman Polygon Clipping

One of the earliest polygon-clipping algorithms is the Sutherland-Hodgman algorithm. It is based on clipping the entire subject polygon against an edge of the window (more precisely, the half plane determined by that edge which contains the clip polygon), then clipping the new polygon against the next edge of the window, and so on, until the polygon has been clipped against all of the four edges. An important aspect of their algorithm is that one can avoid generating a lot of intermediate data.

Representing a polygon as a sequence of vertices P1, P2, . . ., Pn, suppose that we want to clip against a single edge e. The algorithm considers the input vertices Pi one at a time and generates a new sequence Q1, Q2, . . ., Qm. Each Pi generates 0, 1, or 2 of the Qj, depending on the position of the input vertices with respect to e. If we consider each input vertex P, except the first, to be the terminal vertex of an edge, namely the edge defined by P and the immediately preceding input vertex, call it S, then the Q's generated by P depend on the relationship between the edge [S,P] and the line L determined by e. There are four possible cases. See Figure a) . The window side of the line is marked as "inside." The circled vertices are those that are output. Figure 3.10 shows an example of how the clipping works. Clipping the polygon with vertices labeled Pi against edge e1 produces the polygon with vertices Qi. Clipping the new polygon against edge e2 produces the polygon with vertices Ri.
Note that we may end up with some bogus edges. For example, the edge R5R6 in Figure b) is not a part of the mathematical intersection of the subject polygon with the clip polygon.

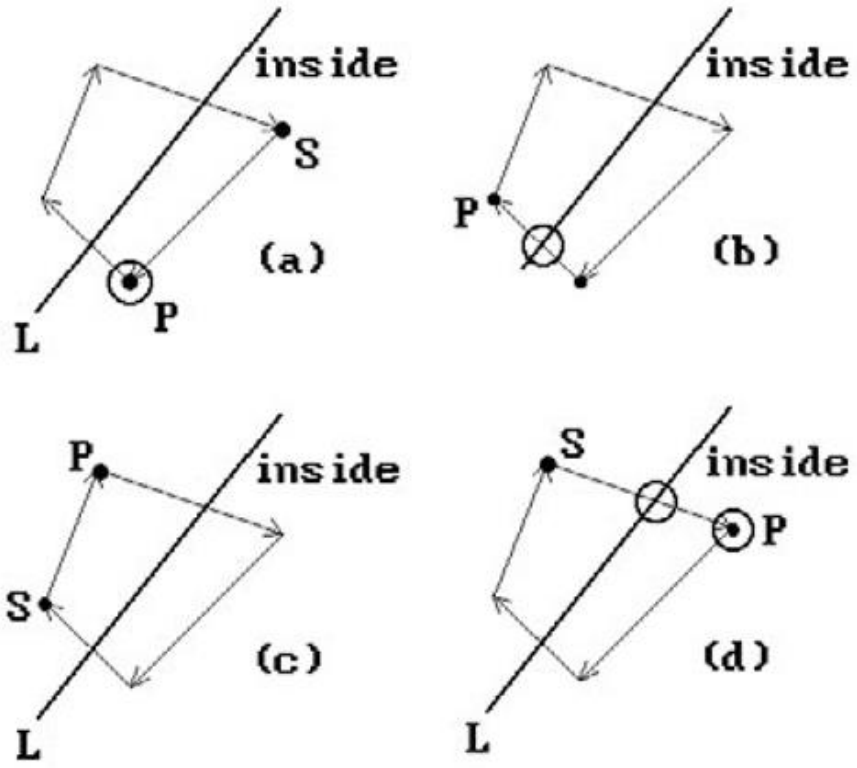


Figure a). The four cases in Sutherland- Hodgman polygon clipping.

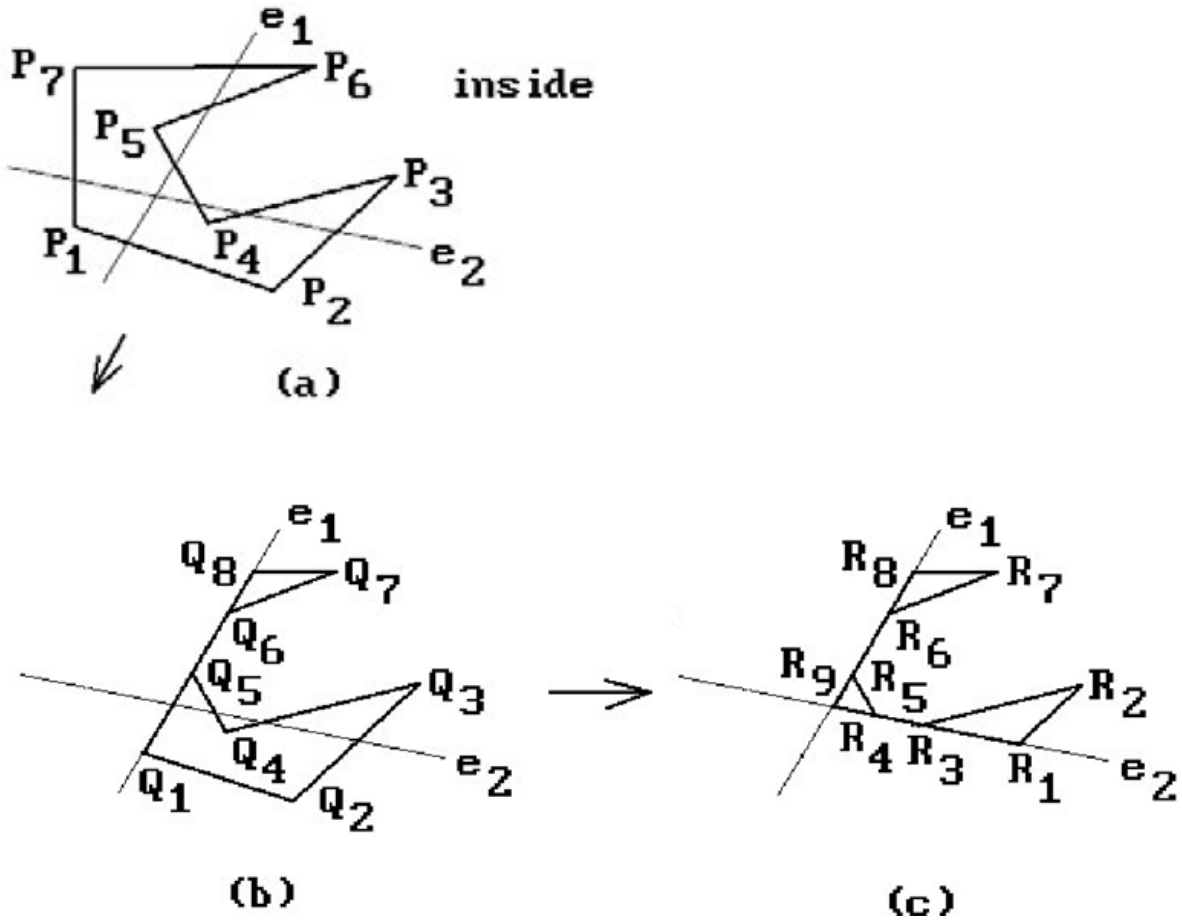


Figure b). A Sutherland-Hodgman polygon-clipping example.

Eliminating such edges from the final result would be a nontrivial effort, but normally they do not cause any problems. We run into this bogus edge problem with other clipping algorithms also.

Weiler Artherton Polygon Clipping

Weiler and Atherton needed a new algorithm because the Sutherland-Hodgman algorithm would have created too many auxiliary polygons.

Here is a very brief description of the algorithm:

The boundaries of polygons are assumed to be oriented so that the inside of the polygon is always to the right as one traverses the boundary. Note that intersections of the subject and clip polygon, if any, occur in pairs: one where the subject enters the inside of the clip polygon and one where it leaves.

Step 1: Compare the borders of the two polygons for intersections. Insert vertices into the polygons at the intersections.

Step 2: Process the nonintersecting polygon borders, separating those contours that are outside the clip polygon and those that are inside.

Step 3: Separate the intersection vertices found on all subject polygons into two lists. One is the entering list, consisting of those vertices where the polygon edge enters the clip polygon. The other is the leaving list, consisting of those vertices where the polygon edge leaves the clip polygon. Step 4: Now clip.

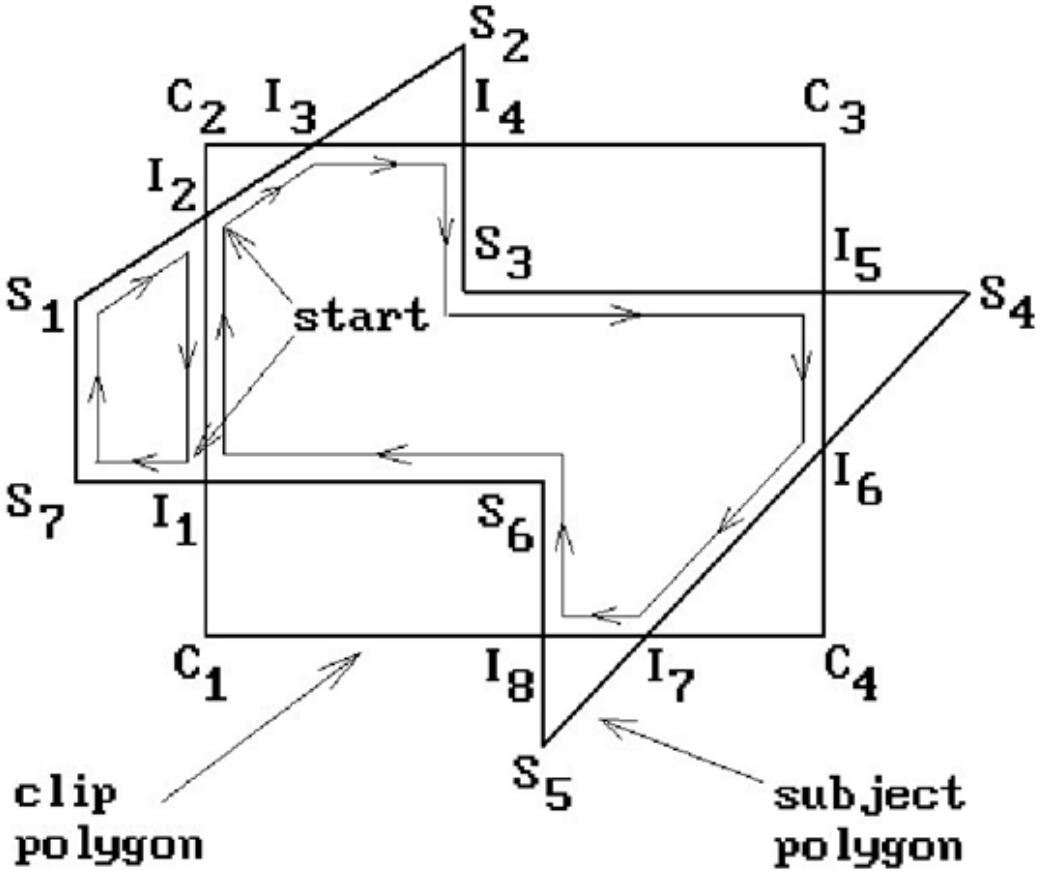


Figure 3.11. Weiler polygon clipping.

(a) Remove an intersection vertex from the entering list. If there is none, then we are done.

(b) Follow the subject polygon vertices to the next intersection.

(c) Jump to the clip polygon vertex list.

(d) Follow the clip polygon vertices to the next intersection.

(e) Jump back to the subject polygon vertex list.

(f) Repeat (b)-(e) until we are back to the starting point.

This process creates the polygons inside the clip polygon. To get those that are outside, one repeats the same steps, except that one starts with a vertex from the leaving list and the clip polygon vertex list is followed in the reverse direction. Finally, all holes are attached to their associated exterior contours.

Example. Consider the polygons in Figure 3.11. The subject polygon vertices are labeled Si, those of the clip polygon are labeled Ci, and the intersections are labeled Ii. The entering list consists of I2, I4, I6, and I8. The leaving list consists of I1, I3, I5, and I7. Starting Step 4(a) with the vertex I2 will generate the inside contour

$$\mathbf{I_2 I_3 I_4 S_3 I_5 I_6 I_7 I_8 S_6 I_1 I_2}.$$

Starting Step 4(a) with vertices $\mathbf{I_1}$, $\mathbf{I_3}$, $\mathbf{I_5}$, and $\mathbf{I_7}$ will generate the outside contours

$$\mathbf{I_1 S_7 S_1 I_2 I_1}, \mathbf{I_3 S_2 I_4 I_3}, \mathbf{I_5 S_4 I_6 I_5}, \text{ and } \mathbf{I_7 S_5 I_8 I_7}.$$

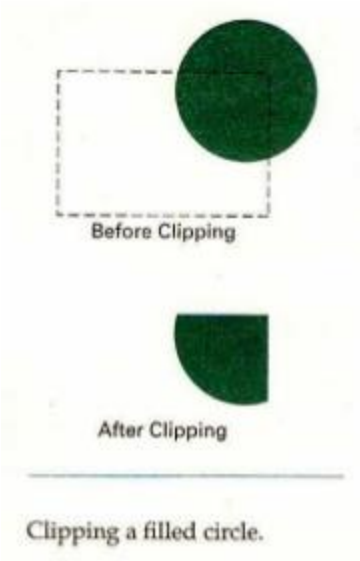
Curve Clipping

Curve-clipping procedures will involve nonlinear equations and this requires more processing than for objects with linear boundaries. The bounding rectangle for a circle or other curved object can be used first to test for overlap with a rectangular clip window.

If the bounding rectangle for the object is completely inside the window, we save the object.

If the rectangle is determined to be completely outside window, we discard the object. In either case, there is no further computation necessary. But if the bounding rectangle test fails, we can look for other computation-saving approaches.

For a circle, we can use the coordinate extents of individual quadrants and then octants for preliminary testing before calculating curve-window intersections.



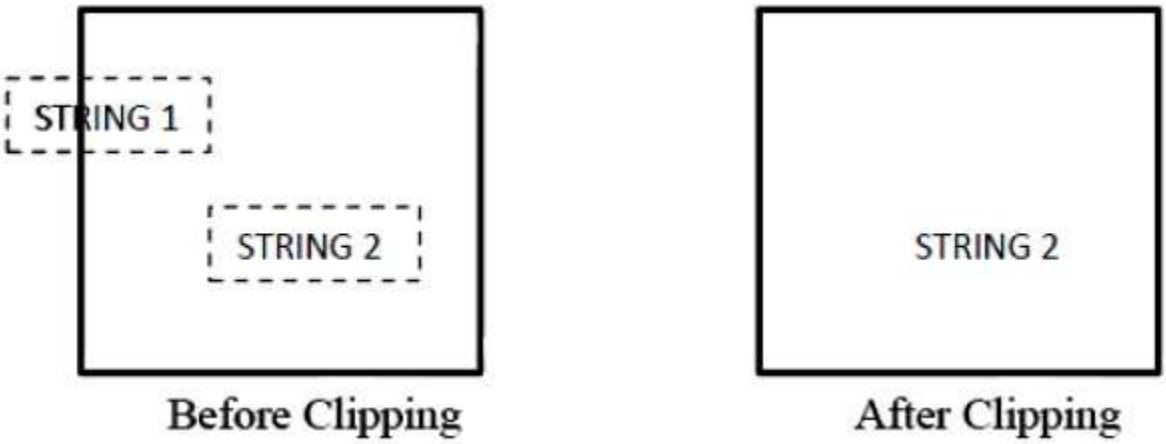
Clipping a filled circle.

Text Clipping

Various techniques are used to provide text clipping in a computer graphics. It depends on the methods used to generate characters and the requirements of a particular application. There are three methods for text clipping which are listed below :

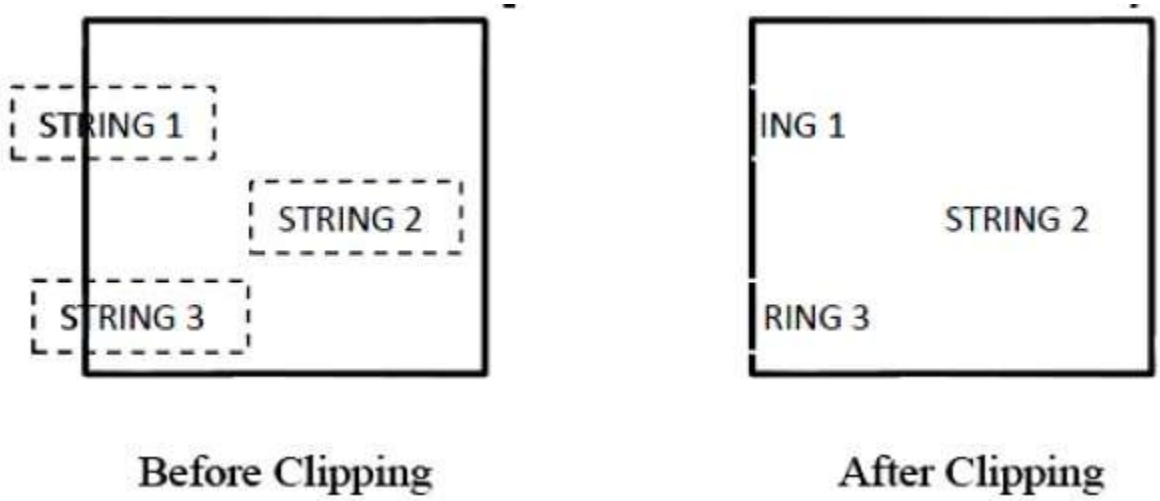
- All or none string clipping
- All or none character clipping
- Text clipping

The following figure shows all or none string clipping −



In all or none string clipping method, either we keep the entire string or we reject entire string based on the clipping window. As shown in the above figure, STRING2 is entirely inside the clipping window so we keep it and STRING1 being only partially inside the window, we reject.

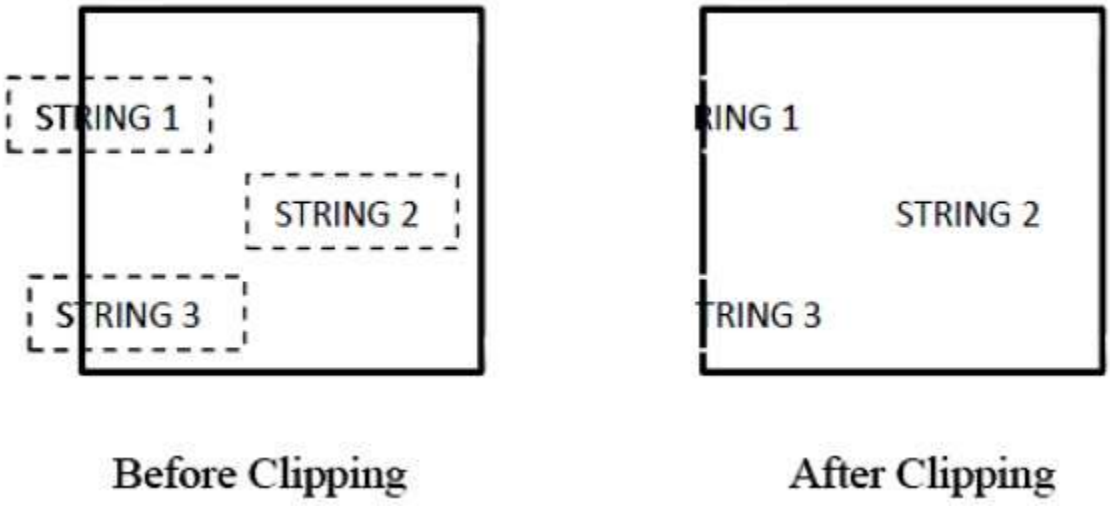
The following figure shows all or none character clipping −



This clipping method is based on characters rather than entire string. In this method if the string is entirely inside the clipping window, then we keep it. If it is partially outside the window, then −

- You reject only the portion of the string being outside
- If the character is on the boundary of the clipping window, then we discard that entire character and keep the rest string.

The following figure shows text clipping −



This clipping method is based on characters rather than the entire string. In this method if the string is entirely inside the clipping window, then we keep it. If it is partially outside the window, then

- You reject only the portion of string being outside.
- If the character is on the boundary of the clipping window, then we discard only that portion of character that is outside of the clipping window.

UNIT−III
3D TRANSFORMATIONS

Three-Dimensional Viewing

Viewing in 3D involves the following considerations:

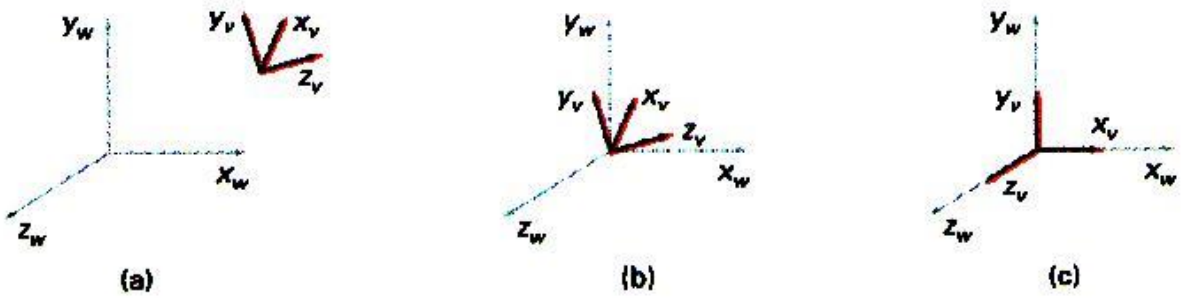
- We can view an object from any spatial position, eg. In front of an object, Behind the object, In the middle of a group of objects, Inside an object, etc.
- 3D descriptions of objects must be projected onto the flat viewing surface of the output device.
- The clipping boundaries enclose a volume of space

Modelling Transformation and Viewing Transformation can be done by 3D transformations. The viewing-coordinate system is used in graphics packages as a reference for specifying the observer viewing position and the position of the projection plane. Projection operations convert the viewing-coordinate description (3D) to coordinate positions on the projection plane (2D). (Usually combined with clipping, visual-surface identification, and surface-rendering)Workstation transformation maps the coordinate positions on the projection plane to the output device

Viewing Transformation

Conversion of objection descriptions from world to viewing coordinates is equivalent to a transformation that superimposes the viewing reference frame onto the world frame using the basic
geometric translate-rotate operations:

1. Translate the view reference point to the origin of the world-coordinate system.
2. Apply rotations to align the xv, yv, and zv axes (viewing coordinate system) with the world xw, yw,
zw axes, respectively.

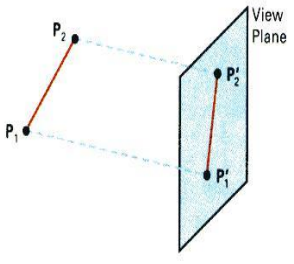


Projections

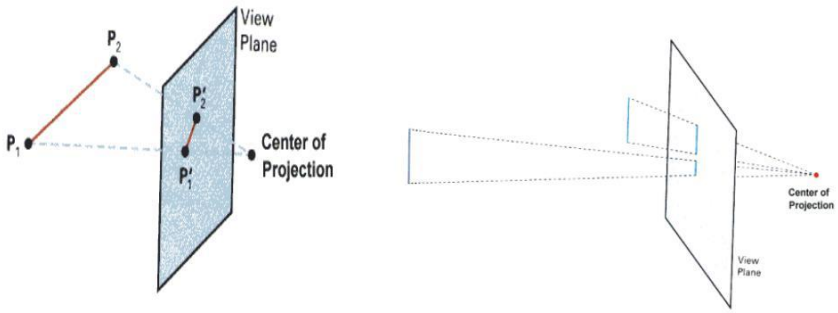
Projection operations convert the viewing-coordinate description (3D) to coordinate positions on the
projection plane (2D). There are 2 basic projection methods:

1. Parallel Projection transforms object positions to the view plane along parallel lines.

A parallel projection preserves relative proportions of objects. Accurate views of the various sides of
an object are obtained with a parallel projection. But not a realistic representation



- Perspective Projection transforms object positions to the view plane while converging to a center
point of projection. Perspective projection produces realistic views but does not preserve relative proportions. Projections of distant objects are smaller than the projections of objects of the same size that are closer to the
projection plane.



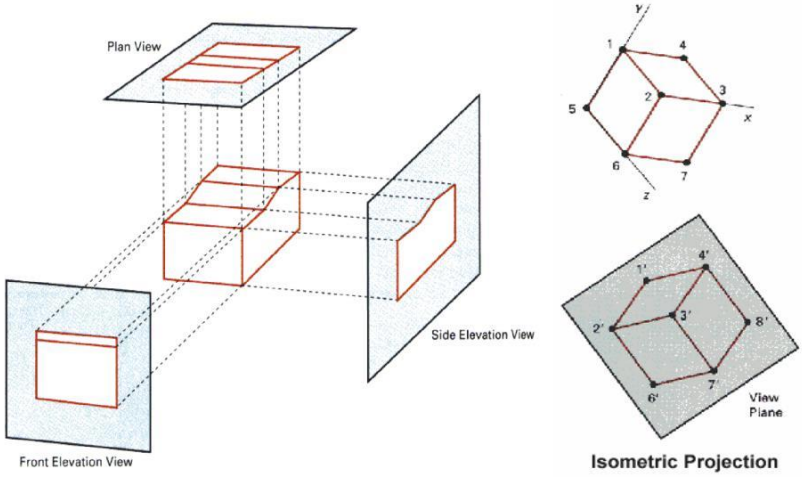
Parallel Projection

Classification:

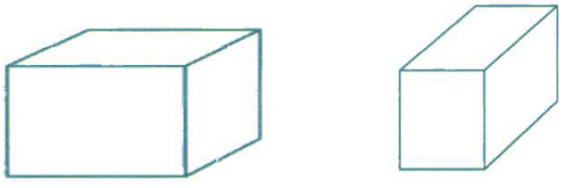
Orthographic Parallel Projection and Oblique Projection:

Orthographic parallel projections are done by projecting points along parallel lines that are perpendicular to the projection plane.

Oblique projections are obtained by projecting along parallel lines that are NOT perpendicular to the
projection plane. Some special Orthographic Parallel Projections involve Plan View (Top projection), Side Elevations, and Isometric Projection:



The following results can be obtained from oblique projections of a cube:



Perspective Projection

Perspective projection is done in 2 steps: Perspective transformation and Parallel projection. These
steps are described in the following section.

Perspective Transformation and Perspective Projection To produce perspective viewing effect, after Modelling Transformation, Viewing Transformation is carried out to transform objects from the world coordinate system to the viewing coordinate system. Afterwards, objects in the scene are further processed with Perspective Transformation: the view volume in the shape of a frustum becomes a regular parallelepiped. The transformation equations are shown as follows and are applied to every vertex of each object:

$x' = x * (d/z)$,

$y' = y * (d/z)$,

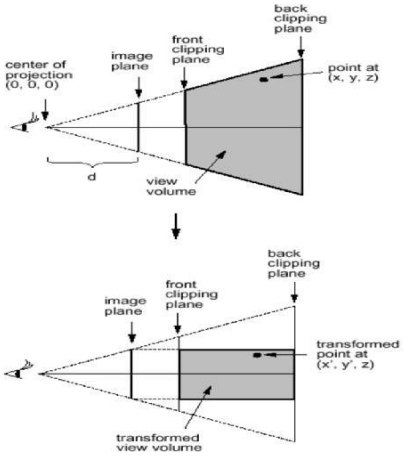
$z' = z$

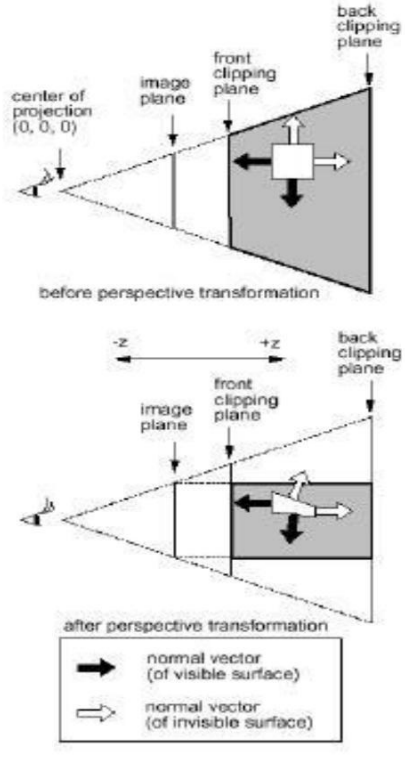
Where (x,y,z) is the original position of a vertex, (x',y',z') is the transformed position of the vertex,
and d is the distance of image plane from the center of projection.

Note that:

Perspective transformation is different from perspective projection: Perspective projection projects a
3D object onto a 2D plane perspectively. Perspective transformation converts a 3D object into a deformed 3D object. After the transformation, the depth value of an object remains unchanged. Before the perspective transformation, all the projection lines converge to the center of projection.

After the transformation, all the projection lines are parallel to each other. Finally we can apply parallel projection to project the object onto a 2D image plane. Perspective Projection = Perspective Transformation + Parallel Projection

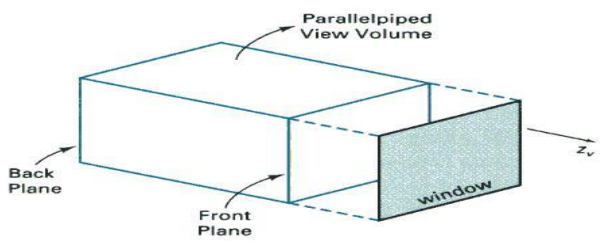




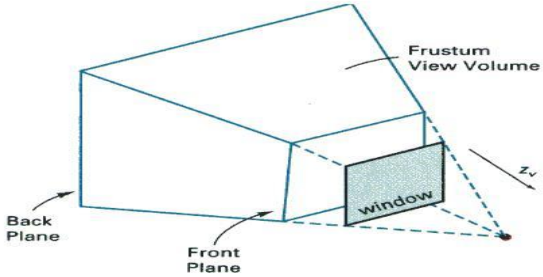
View Volumes

View window - A rectangular area in the view plane which controls how much of the scene is viewed.

The edges of the view window are parallel to the xv and yv viewing axes. View volume - formed by the view window and the type of projection to be used. Only those objects within the view volume will appear in the generated display. So we can exclude objects that are beyond the view volume when we render the objects in the scene. A finite view volume is obtained by bounding with front plane and back plane (or the near plane and the far plane). Hence a view volume is bounded by 6 planes => rectangular parallelepiped or a frustum, for parallel projection and perspective projection respectively. Some



Parallel Projection



Perspective Projection

Some facts:

Perspective effects depend on the positioning of the center point of projection. If it is close to the view plane, perspective effects are emphasized, ie. closer objects will appear larger than more distant

objects of the same size. The projected size of an object is also affected by the relative position of the object and the view plane.

'Viewing' a static view:

The view plane is usually placed at the viewing-coordinate origin and the center of projection is positioned to obtain the amount of perspective desired.

'Viewing' an animation sequence:

Usually the center of projection point is placed at the viewing-coordinate origin and the view plane is

placed in front of the scene. The size of the view window is adjusted to obtain the amount of scene

desired. We move through the scene by moving the viewing reference frame (ie. the viewing coordinate system).

Some facts:

Perspective effects depend on the positioning of the center point of projection. If it is close to the view plane, perspective effects are emphasized, ie. closer objects will appear larger than more distant

objects of the same size. The projected size of an object is also affected by the relative position of the object and the view

plane.

'Viewing' a static view:

The view plane is usually placed at the viewing-coordinate origin and the center of projection is positioned to obtain the amount of perspective desired.

'Viewing' an animation sequence:

Usually the center of projection point is placed at the viewing-coordinate origin and the view plane is
placed in front of the scene. The size of the view window is adjusted to obtain the amount of scene
desired. We move through the scene by moving the viewing reference frame (ie. the viewing coordinate system).

Depth Cueing

With few exceptions, depth information is important ***so*** that we can easily identify, for a particular viewing direction, which is the front and which is the back of displayed objects. Figure 9-5 illustrates the ambiguity that can result when a wireframe object is displayed without depth information. There are several ways in which we can include depth information in the two-dimensional representation of solid objects.

A simple method for indicating depth with wireframe displays is to vary the intensity of objects according to their distance from the viewing position. Figure 9-6 shows a wireframe object displayed with depth ***cueing.*** The lines closest to

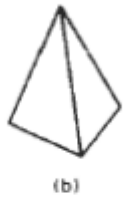


Figure 9-5
The wireframe representation of the pyramid in (a) contains no depth information to indicate whether the viewing direction is (b) downward from a position above the apex or (c) upward from a position below the base.

the viewing position are displayed with the highest intensities, and lines farther away are displayed with decreasing intensities. Depth cueing is applied by choosing maximum and minimum intensity (or color) values and a range of distances over which the intensities are to vary.

Another application of depth cueing 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. Some atmospheric effects can change the perceived color of an object, and we can model these effects with depth cueing.

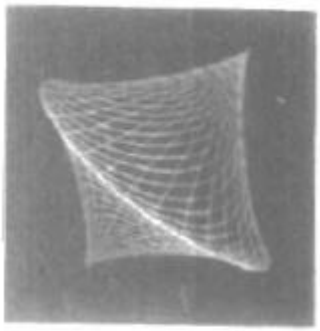


Figure 9-6
A wireframe object displayed with depth cueing, so that the intensity of lines decreases from the front to the back of the object.

Visible Line and Surface Identification

We can also clarify depth lat ti on ships in a wireframe display by identifying visible lines in some way. The simplest method is to highlight the visible lines or to display them in a different color. Another technique, commonly used for engineering drawings, is to display the nonvisible lines as dashed lines. Another approach is to simply remove the nonvisible lines, as in Figs. 9-5(b) and 9-5(c). But removing the hidden lines also removes information about the shape of the back surfaces of an object. These visible-line methods also identify the visible surfaces
of objects.

When objects are to be displayed with color or shaded surfaces, we apply surface-rendering procedures to the visible surfaces so that the hidden surfaces are obscured. Some visible surface algorithms establish visibility pixel by pixel across the viewing plane; other algorithms determine visibility for object surfaces as a whole.

Surface Rendering

Added realism is attained in displays by **`setting`** the surface intensity of objects according to the lighting conditions in the scene and according to assigned surface characteristics. **Lighting** specifications include the intensity and positions of light sources and the general background illumination required for a scene. Surface properties of objects include degree of transparency and how rough or smooth the surfaces are to be. Procedures can then be applied to generate the correct illumination and shadow regions for the scene. In Fig. 9-7, surface-rendering methods are combined with perspective and visible-surface identification to generate a degree of realism in a displayed scene.

Exploded and Cutaway Views

Many graphics packages allow objects to be defined as hierarchical structures, so that internal details can be stored. Exploded and cutaway views of such objects can then be used to show the internal structure and relationship of the object parts. Figure 9-8 shows several kinds of exploded displays for a mechanical design.

An alternative to exploding an objects into its component parts is the cutaway view (Fig. 9-9, which removes part of the visible surfaces to show internal structure.

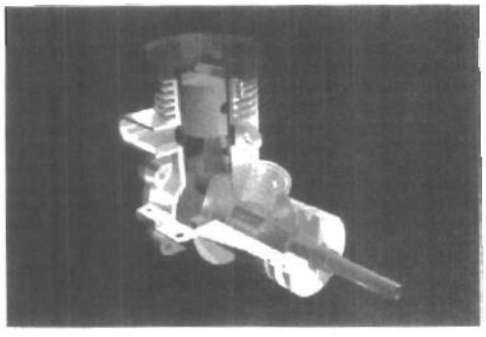


Figure 9-9
Color-coded cutaway view of a lawn mower engine showing the structure and relationship of internal components. (*Courtesy of Autodesk, Inc.*)

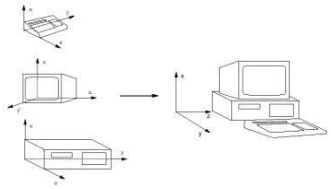
Visible-Surface Detection Methods

More information about Modelling and Perspective Viewing:

Before going to visible surface detection, we first review and discuss the followings:

Modeling Transformation:

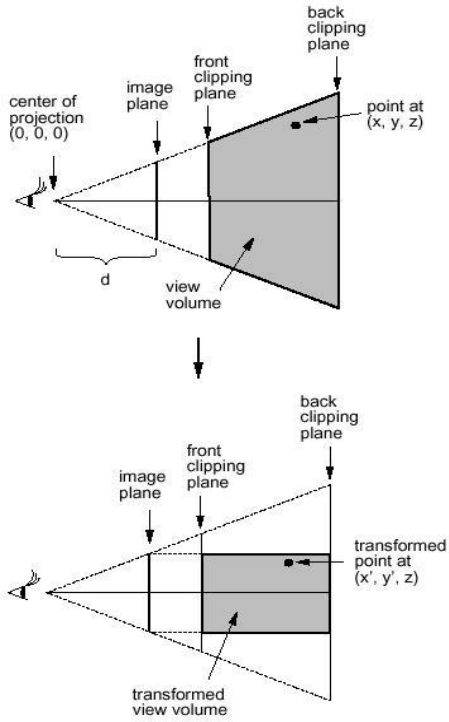
In this stage, we transform objects in their local modelling coordinate



systems into a common coordinate system called the world coordinates.

Perspective Transformation (in a perspective viewing system):

After Modelling Transformation, Viewing Transformation is carried out to transform objects from the world coordinate system to the viewing



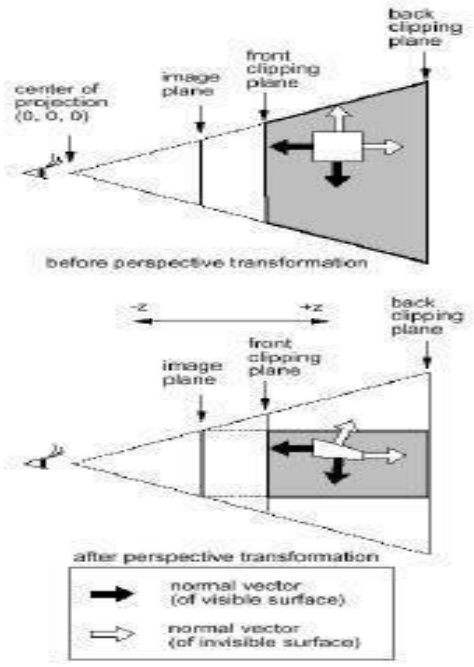
coordinate system. Afterwards, objects in the scene are further perspectively transformed. The effect of such an operation is that after the transformation, the view volume in the shape of a frustum becomes a regular parallelepiped. The transformation equations are shown as follows and are applied to every vertex of each object:

$x' = x * (d/z)$,

$y' = y * (d/z)$,

$z' = z$

Where (x,y,z) is the original position of a vertex, (x',y',z') is the transformed position of the vertex, and d is the distance of image plane



from the center of projection.

Note that:

Perspective transformation is different from perspective projection:

Perspective projection projects a 3D object onto a 2D plane perspectively.

Perspective transformation converts a 3D object into a deformed 3D object.

After the transformation, the depth value of an object remains unchanged.

Before the perspective transformation, all the projection lines converge to the center of projection. After the transformation, all the projection lines are parallel to each others.

Perspective Projection = Perspective Transformation + Parallel Projection

In 3D clipping, we remove all objects and parts of objects which are outside of the view volume. Since we have done perspective transformation, the 6 clipping planes,

which form the parallelepiped, are parallel to the 3 axes and hence clipping is straight forward. Hence the clipping operation can be performed in 2D. For example, we may first perform the clipping operations on the x-y plane and then on the x-z plane.

Problem definition of Visible-Surface Detection Methods:

To identify those parts of a scene that are visible from a chosen viewing position.

Surfaces which are obscured by other opaque surfaces along the line of sign (projection) are invisible to the viewer.

Characteristics of approaches:

- Require large memory size?
- Require long processing time?
- Applicable to which types of objects?

Considerations:

- Complexity of the scene
- Type of objects in the scene
- Available equipment
- Static or animated?

Classification of Visible-Surface Detection Algorithms:

Object-space Methods

Compare objects and parts of objects to each other within the scene definition to determine which
surfaces, as a whole, we should label as visible:

For each object in the scene do

Begin

3. Determine those part of the object whose view is unobstructed by other parts of it or any other object with respect to the viewing specification.
4. Draw those parts in the object color.

- Compare each object with all other objects to determine the visibility of the object parts.
- If there are n objects in the scene, complexity = O(n2)
- Calculations are performed at the resolution in which the objects are defined (only limited by the
computation hardware).
- Process is unrelated to display resolution or the individual pixel in the image and the result of the
process is applicable to different display resolutions.
- Display is more accurate but computationally more expensive as compared to image space methods because step 1 is typically more complex, eg. Due to the possibility of intersection between surfaces.
- Suitable for scene with small number of objects and objects with simple relationship with each other.

Image-space Methods (Mostly used)

Visibility is determined point by point at each pixel position on the projection plane.

For each pixel in the image do

Begin

1. Determine the object closest to the viewer that is pierced by the projector through the pixel
2. Draw the pixel in the object colour.

- For each pixel, examine all n objects to determine the one closest to the viewer.
- If there are p pixels in the image, complexity depends on n and p (O(np)).
- Accuarcy of the calculation is bounded by the display resolution.
- A change of display resolution requires re-calculation

Application of Coherence in Visible Surface Detection Methods:
- Making use of the results calculated for one part of the scene or image for other nearby parts.
- Coherence is the result of local similarity
- As objects have continuous spatial extent, object properties vary smoothly within a small local region in the scene. Calculations can then be made incremental.

Types of coherence:

1. **Object Coherence:**

Visibility of an object can often be decided by examining a circumscribing solid (which may be of
simple form, eg. A sphere or a polyhedron.)

2. Face Coherence:

Surface properties computed for one part of a face can be applied to adjacent parts after small incremental modification. (eg. If the face is small, we sometimes can assume if one part of the face is
invisible to the viewer, the entire face is also invisible).

3. Edge Coherence:

The Visibility of an edge changes only when it crosses another edge, so if one segment of an nonintersecting edge is visible, the entire edge is also visible.

4. Scan line Coherence:

Line or surface segments visible in one scan line are also likely to be visible in adjacent scan lines.
Consequently, the image of a scan line is similar to the image of adjacent scan lines.

5. Area and Span Coherence:

A group of adjacent pixels in an image is often covered by the same visible object. This coherence is
based on the assumption that a small enough region of pixels will most likely lie within a single polygon. This reduces computation effort in searching for those polygons which contain a given

screen area (region of pixels) as in some subdivision algorithms.

6. Depth Coherence:

The depths of adjacent parts of the same surface are similar.

7. Frame Coherence:

Pictures of the same scene at successive points in time are likely to be similar, despite small changes

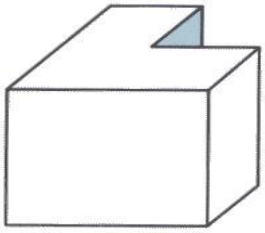
in objects and viewpoint, except near the edges of moving objects. Most visible surface detection methods make use of one or more of these coherence properties of a scene. To take advantage of regularities in a scene, eg. Constant relationships often can be established between objects and surfaces in a scene.

Back-Face Detection

In a solid object, there are surfaces which are facing the viewer (front faces) and there are surfaces

which are opposite to the viewer (back faces). These back faces contribute to approximately half of the total number of surfaces. Since we cannot see these surfaces anyway, to save processing time, we can remove them before the clipping process with a simple test. Each surface has a normal vector. If this vector is pointing in the direction of the center of projection, it is a front face and can be seen by the viewer. If it is pointing away from the center of projection, it is a back face and cannot be seen by the viewer. The test is very simple, if the z component of the normal vector is positive, then, it is a back face. If the z component of the vector is negative, it is a front face. Note that this technique only caters well for non overlapping convex polyhedral. For other cases where there are concave polyhedra or

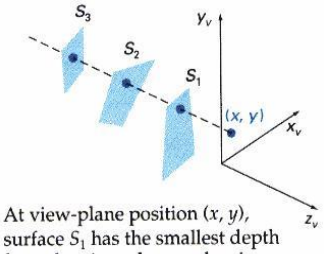
overlapping objects, we still need to apply other methods to further determine where the obscured faces are partially or completely



hidden by other objects (eg.Using Depth-Buffer Method or Depth-sort Method).

Depth-Buffer Method (Z-Buffer Method)

This approach compare surfac
e depths at each pixel

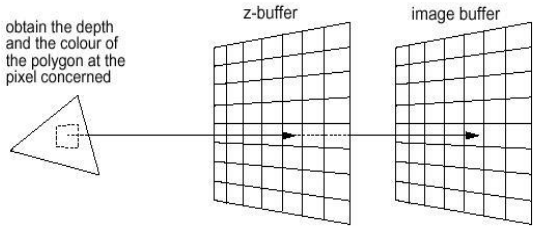


At view-plane position (x, y), surface S_1 has the smallest depth from the view plane and so is visible at that position.

position on the projection plane.

Object depth is usually measured from the view plane

along the z axis of a viewing system. This method requires 2 buffers: one is the image buffer and the other is called the z-buffer (or the depth buffer). Each of these buffers has the same resolution as the image to be



captured. As surfaces are processed, the image buffer is used to store the color values of each pixel position and the z-buffer is used to store the depth values for each (x,y) position.

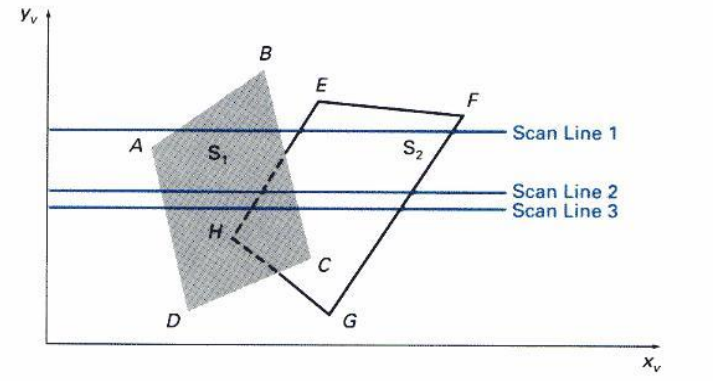
Algorithm:

1. Initially each pixel of the z-buffer is set to the maximum depth value (the depth of the back clipping plane).
2. The image buffer is set to the background color.
3. Surfaces are rendered one at a time.
4. For the first surface, the depth value of each pixel is calculated.
5. If this depth value is smaller than the corresponding depth value in the z-buffer (ie. it is closer to the view point), both the depth value in the z-buffer and the color value in the image buffer are replaced by the depth value and the color value of this surface calculated at the pixel position.
6. Repeat step 4 and 5 for the remaining surfaces.
7. After all the surfaces have been processed, each pixel of the image buffer represents the color of a visible surface at that pixel. This method requires an additional buffer (if compared with the Depth-Sort Method) and the overheads involved in updating the buffer. So this method is less attractive in the cases where only a few objects in the scene are to be rendered.

- Simple and does not require additional data structures.
- The z-value of a polygon can be calculated incrementally.
- No pre-sorting of polygons is needed.
- No object-object comparison is required.
- Can be applied to non-polygonal objects.
- Hardware implementations of the algorithm are available in some graphics workstation.
- For large images, the algorithm could be applied to, eg., the 4 quadrants of the image separately, so as to reduce the requirement of a large additional buffer

Scan-Line Method

In this method, as each scan line is processed, all polygon surfaces intersecting that line are examined to determine which are visible. Across each scan line, depth calculations are made for each overlapping surface to determine which is nearest to the view plane. When the visible surface has been determined, the intensity value for that position is entered into the image buffer.



Scan lines crossing the projection of two surfaces, S_1 and S_2, in the view plane. Dashed lines indicate the boundaries of hidden surfaces.

For each scan line do

Begin

For each pixel (x,y) along the scan line do ---------- Step 1

Begin

z_buffer(x,y) = 0

Image_buffer(x,y) = background_color

End

For each polygon in the scene do ---------- Step 2

Begin

For each pixel (x,y) along the scan line that is covered by the polygon do

2a. Compute the depth or z of the polygon at pixel location (x,y).

2b. If z < z_buffer(x,y) then

Set z_buffer(x,y) = z

Set Image_buffer(x,y) = polygon's colour

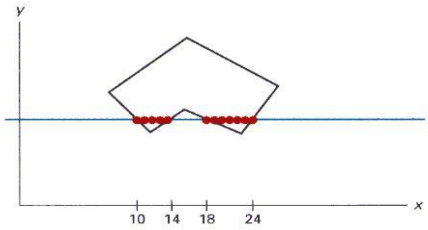
End

End

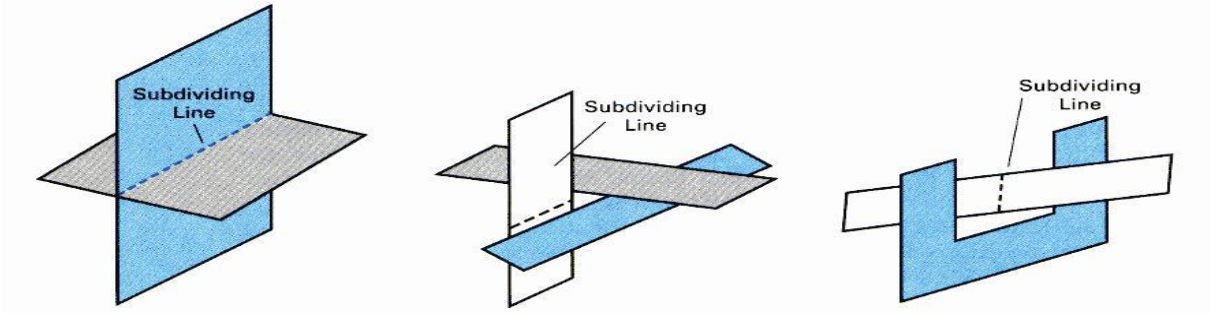
End

- Step 2 is not efficient because not all polygons necessarily intersect with the scan line.
- Depth calculation in 2a is not needed if only 1 polygon in the scene is mapped onto a segment of
the scan line.
- To speed up the process:

Recall the basic idea of polygon filling: For each scan line crossing a polygon, this algorithm locates the intersection points of the scan line with the polygon edges. These intersection points are sorted from left to right. Then, we fill the pixels between each intersection pair.



With similar idea, we fill every scan line span by span. When polygon overlaps on a scan line, we perform depth calculations at their edges to determine which polygon should be visible at which span. Any number of overlapping polygon surfaces can be processed with this method. Depth calculations are performed only when there are polygons overlapping. We can take advantage of coherence along the scan lines as we pass from one scan line to the next. If no changes in the pattern of the intersection of polygon edges with the successive scan lines, it is not necessary to do depth calculations. This works only if surfaces do not cut through or otherwise cyclically overlap each other. If cyclic overlap happens, we can divide the surfaces to eliminate the overlaps.



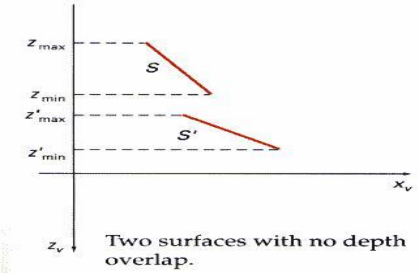
- The algorithm is applicable to non-polygonal surfaces (use of surface and active surface table, zvalue
is computed from surface representation).
- Memory requirement is less than that for depth-buffer method.
- Lot of sortings are done on x-y coordinates and on depths.

Depth-Sort Method

1. Sort all surfaces according to their distances from the view point.
2. Render the surfaces to the image buffer one at a time starting from the farthest surface.
3. Surfaces close to the view point will replace those which are far away.
4. After all surfaces have been processed, the image buffer stores the final image.

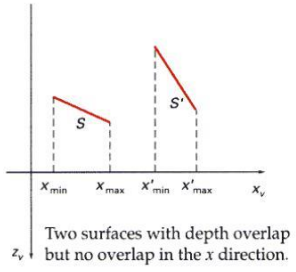
The basic idea of this method is simple. When there are only a few objects in the scene, this method can be very fast. However, as the number of objects increases, the sorting process can become very complex and time consuming.

Example: Assuming we are viewing along the z axis. Surface S with the greatest depth is then compared to other surfaces in the list to determine whether there are any overlaps in depth. If no depth

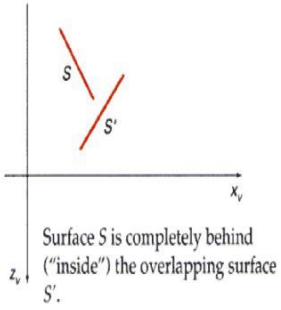


Two surfaces with no depth overlap.

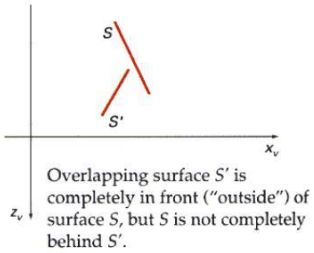
overlaps occur, S can be scan converted. This process is repeated for the next surface in the list. However, if depth overlap is detected, we need to make some additional comparisons to determine whether any of the surfaces should be reordered.



Two surfaces with depth overlap but no overlap in the x direction.



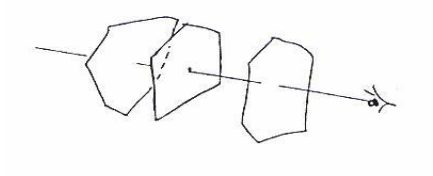
Surface S is completely behind ("inside") the overlapping surface S'.

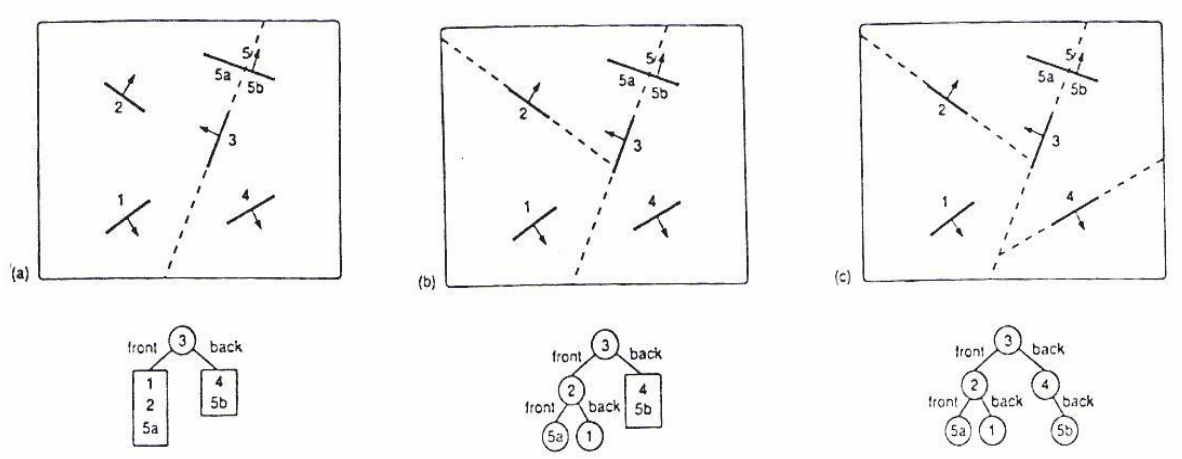


Overlapping surface S' is completely in front ("outside") of surface S, but S is not completely behind S'.

Binary Space Partitioning

- suitable for a static group of 3D polygon to be viewed from a number of view points
- based on the observation that hidden surface elimination of a polygon is guaranteed if all polygons on the other side of it as the viewer is painted first, then itself, then all polygons on the same side of it as the viewer





1.The algorithm first build the BSP tree:

- a root polygon is chosen (arbitrarily) which divides the region into 2 half-spaces (2 nodes => front and back)
- a polygon in the front half-space is chosen which divides the half-space into another 2 halfspaces

- the subdivision is repeated until the half-space contains a single polygon (leaf node of the tree)
- the same is done for the back space of the polygon.

2.To display a BSP tree:

- see whether the viewer is in the front or the back half-space of the root polygon.
- if front half-space then first display back child (subtree) then itself, followed by its front child / subtree
- the algorithm is applied recursively to the BSP tree.

BSP Algorithm

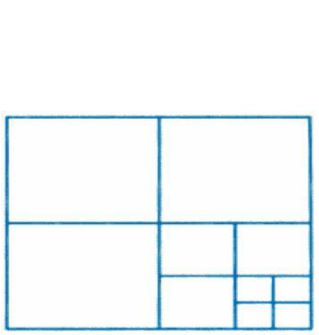
```
Procedure DisplayBSP(tree: BSP_tree)
Begin
If tree is not empty then
If viewer is in front of the root then
Begin
DisplayBSP(tree.back_child)
displayPolygon(tree.root)
DisplayBSP(tree.front_child)
End
Else
Begin
DisplayBSP(tree.front_child)
displayPolygon(tree.root)
DisplayBSP(tree.back_child)
End
End
```

Discussion:

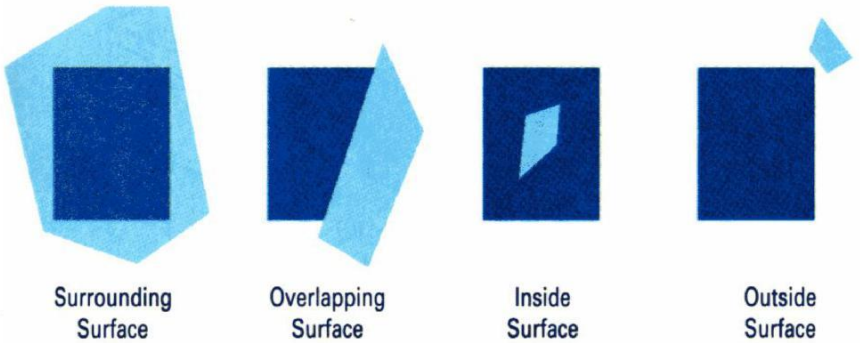
- Back face removal is achieved by not displaying a polygon if the viewer is located in its back half-space
- It is an object space algorithm (sorting and intersection calculations are done in object space precision)
- If the view point changes, the BSP needs only minor re-arrangement.
- A new BSP tree is built if the scene changes
- The algorithm displays polygon back to front (cf. Depth-sort)

Area Subdivision Algorithms

The area-subdivision method takes advantage of area coherence in a scene by locating those view areas that represent part of a single surface. The total viewing area is successively divided into smaller and smaller rectangles until each small area is simple, ie. it is a single pixel, or is covered wholly by a part of a single visible surface or no surface at all.



Dividing a square area into equal-sized quadrants at each step.



Possible relationships between polygon surfaces and a rectangular area.

The procedure to determine whether we should subdivide an area into smaller rectangle is:

1. We first classify each of the surfaces, according to their relations with the area: Surrounding surface - a single surface completely encloses the area Overlapping surface - a single surface that is partly inside and partly outside the area Inside surface - a single surface that is completely inside the area Outside surface - a single surface that is completely outside the area. To improve the speed of classification, we can make use of the bounding rectangles of surfaces for early confirmation or rejection that the surfaces should be belong to that type.

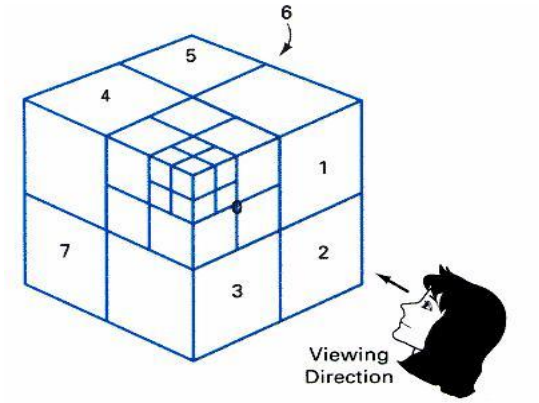
2. Check the result from 1., that, if any of the following condition is true, then, no subdivision of this area is needed.

a. All surfaces are outside the area.

b. Only one surface is inside, overlapping or surrounding surface is in the area. c. A surrounding surface obscures all other surfaces within the area boundaries. For cases b and c, the color of the area can be determined from that single surface.

Octree Methods

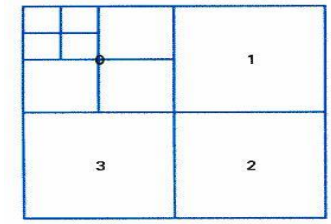
In these methods, octree nodes are projected onto the viewing surface in a front-to-back order. Any surfaces toward the rear of the front octants (0,1,2,3) or in the back octants (4,5,6,7) may be hidden by the front surfaces.



Octants in Space

With the numbering method (0,1,2,3,4,5,6,7), nodes representing octants 0,1,2,3 for the entire region are visited before the nodes representing octants 4,5,6,7. Similarly the nodes for the front four suboctants of octant 0 are visited before the nodes

for the four back suboctants.When a colour is encountered in an octree node, the corresponding



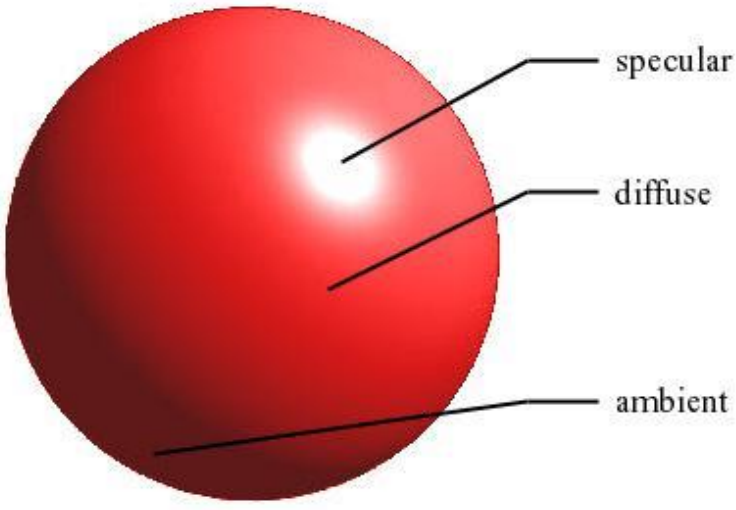
Quadrants for
the View Plane

pixel in the frame buffer is painted only if no previous color has been loaded into the same pixel position. 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.
- If the front is empty, it is necessary only to process the rear octant.
- If the front octant has heterogeneous regions, it has to be subdivided and the sub-octants are handled recursively.

ILLUMINATION MODELS:

- The important components are:
  - – Diffuse reflection
  - – Specular reflection
  - – Ambient light



- The total reflected light from a surface is the sum of the contributions from light sources and reflected light
- Ambient light

- Also called background light
- Not created by any light source
- A constant lighting from all directions
- Contributed by scattered light in a surrounding
- When used alone, does not produce very interesting

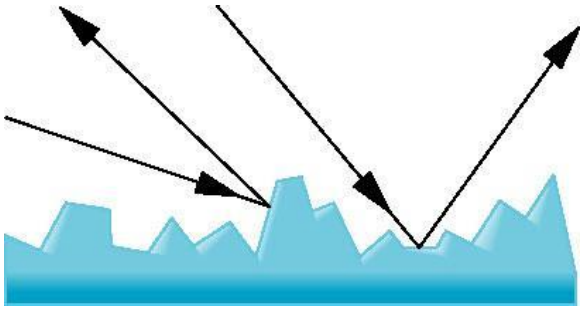
Pictures

- incorporate background light we simply set a general brightness level I_a for a scene
- surfaces may reflect different amount of ambient light, based on their reflectance properties. We model this by a constant factor for each surface:

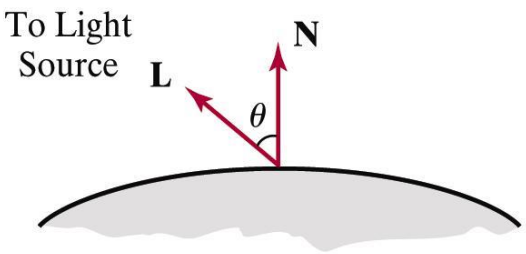
$$k_a \times I_a$$

Reflection

- Light scattered with equal intensity in all directions (ideal diffuse reflection)
- Light from a point is independent on viewing direction (equally bright in all directions)



- angle between the incoming light direction and a surface normal is referred to as the **angle of incidence,** denoted θ.

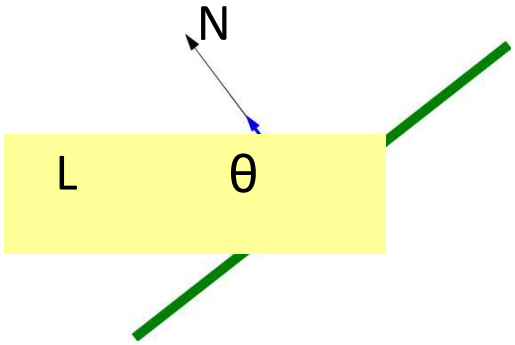


L = unit vector
to light source

N = unit vector
normal to surface

- Law of reflection: the angle of incidence equals the angle of reflection, and L, N and R(eflection) directions are co-planar.

- If surface has brightness I when facing light, it has brightness I*cos(θ) when tilted at angle θ.



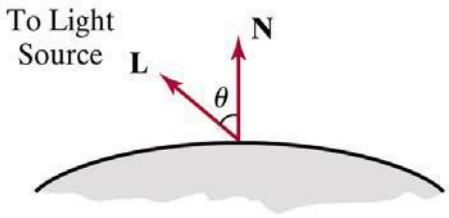
N·L = cos θ

(N, L are unit vectors)

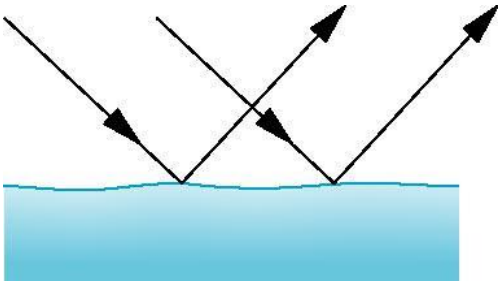
- You will see the brightness written as I(N·L)

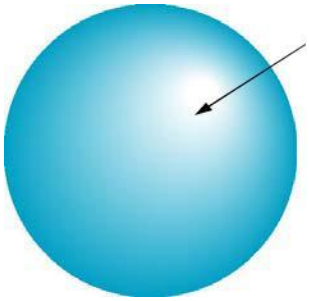
- A parameter k_d set for each surface determines the fraction of incident light scattered as diffuse reflections from that surface
- This parameter is known as the diffuse reflection coefficient or the diffuse reflectivity
- k_d is assigned a value between 0.0 and 1.0
  - 0.0 for dull surface that absorbs almost all light
  - 1.0 for shiny surface that reflects almost all light

- Diffuse reflections:

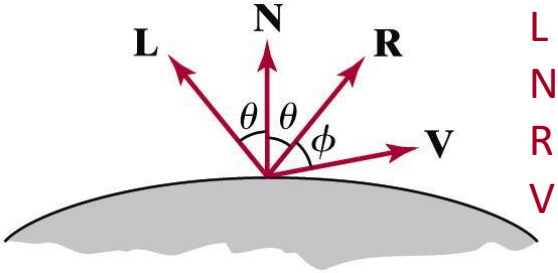


- depends on where the viewer is!



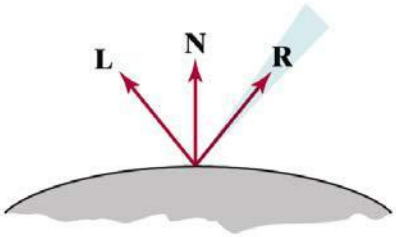


- The white specular highlight is the reflection of white light from the source in the direction of the viewer
- The bright spot that we see on a shiny surface is the result of incident light reflected in a concentrated region around the specular reflection angle
- The specular reflection angle equals the angle of the incident light

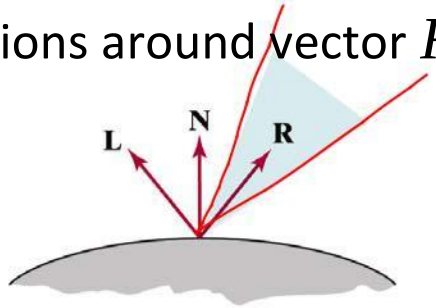


L = vector to light source
N = vector normal to surface
R = direction of reflected light
V = vector to viewer

- perfect mirror reflects light only in the specular-reflection direction
- Other objects exhibit specular reflections over a finite range of viewing positions around vector R



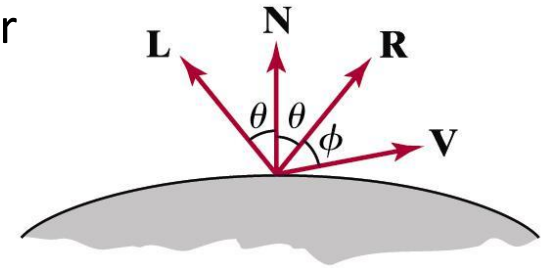
Shiny Surface



Dull Surface

Phong Specular Reflection Model

- Phong model sets the intensity of specular reflection as proportional to the angle φ between the viewing vector and the specular reflection vector:



$$I_s = I \times k_s \times \cos^{\alpha} \varphi$$

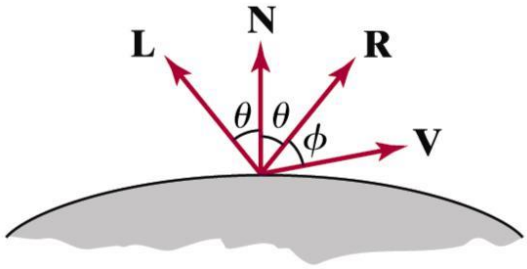
α = shineness exponent

k_S = specular reflectivity of material

$$I_s = I \times k_s \times \cos^{\alpha} \varphi$$

α = shininess

k_S = reflectivity



- The shineness α is determined by the type of surface we want to display
  - Shiny surfaces have a very large value (>100)
  - Rough surfaces would have a value near 1
- The larger the α, the more concentrated the light is around R. For mirrors, α ⇗ infinity.

$$I_s = I \times k_s \times \cos^{\alpha} \varphi$$

α = shininess

k_S = reflectivity

- Recall that R·V = cos φ

$$k_S I (V \cdot R)^{\alpha}$$

I_s =0.0

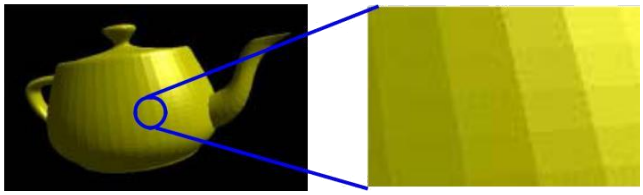
Polygonal Shading

- to render solid surfaces
- Determines how surfaces will be filled
- Process for computing the color intensity value for each pixel contained in a polygon
- The most common shading techniques are:
  - *Flat Shading*
  - *Gauraud Shading*
  - *Phong Shading*

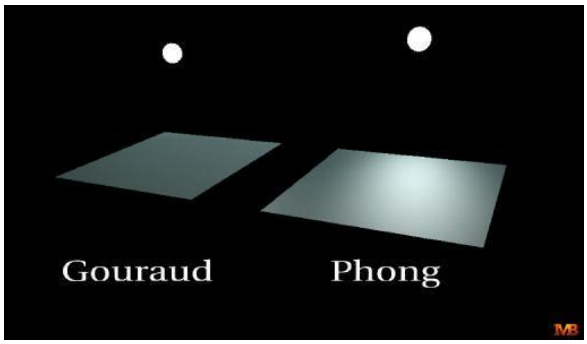
Flat Shading

- Simplest, Cheapest, Fastest Shading Method
  - Works well for objects really made of flat faces.
  - Appearance depends on number of polygons for curved surface objects.
- Fills an entire polygon with one color intensity

- This model is only valid (realistic) if:
  - *The light source is imagined to be at infinity*
  - *The viewer is at infinity*
  - *The polygon is not an approximation to a curved surface*
- Flat shading suffers from “mach band effect”
  - human eyes accentuate the discontinuity at the boundary



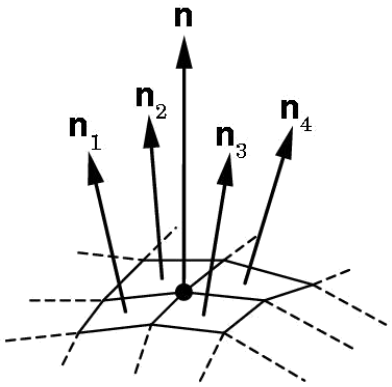
- Fix the mach band effect – remove edge discontinuity
- Compute lighting for more points on each face
- Two popular methods:
  - Gauraud shading (used by OpenGL)
  - Phong shading (better specular highlight, not in OpenGL)



Gaurad Shading

- Per-vertex lighting calculation
- Normal is needed for each vertex
- Per-vertex normal can be computed by averaging the adjacent face normals:

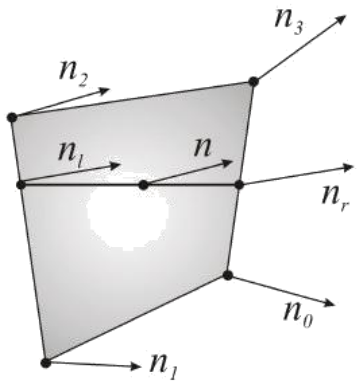
$$n = \frac{n_1}{} + \frac{n_2}{} + \frac{n_3}{} + \frac{n_4}{} \; 4$$



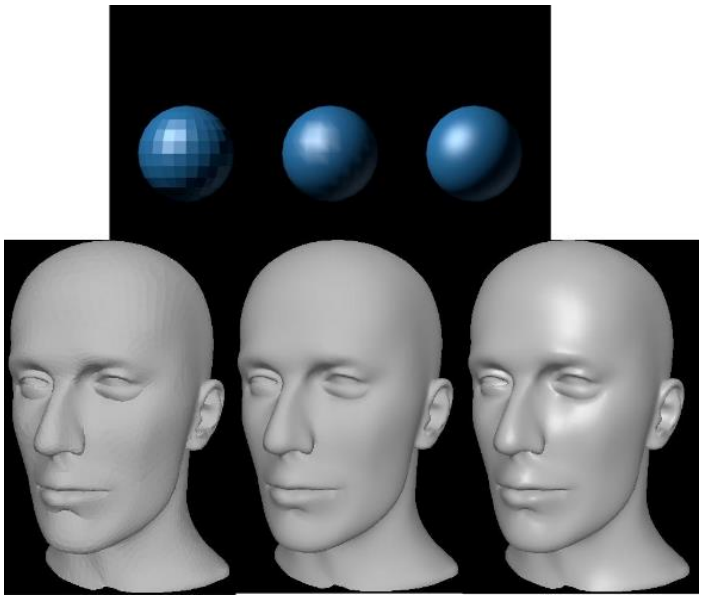
- Requires knowledge about adjacent faces

Phong Shading

- Instead of interpolation, we calculate lighting for each pixel inside the polygon (per pixel lighting)
- Need normals for all the pixels – not provided by user
- Phong shading algorithm interpolates the normals and compute lighting for each pixel
- Over Normal Vector, NOT Vertex Color:



FLAT, GOURAUD AND PHONG SHADING example



UNIT IV

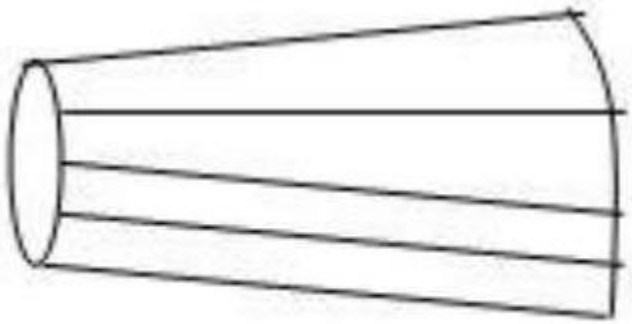
Polygon Surfaces

Objects are represented as a collection of surfaces. 3D object representation is divided into two categories.

- **Boundary Representations (B-reps)** − It describes a 3D object as a set of surfaces that separates the object interior from the environment.
- **Space–partitioning representations** − It is used to describe interior properties, by partitioning the spatial region containing an object into a set of small, non-overlapping, contiguous solids (usually cubes).

The most commonly used boundary representation for a 3D graphics object is a set of surface polygons that enclose the object interior. Many graphics system use this method. Set of polygons are stored for object description. This simplifies and speeds up the surface rendering and display of object since all surfaces can be described with linear equations.

The polygon surfaces are common in design and solid-modeling applications, since their **wireframe display** can be done quickly to give general indication of surface structure. Then realistic scenes are produced by interpolating shading patterns across polygon surface to illuminate.



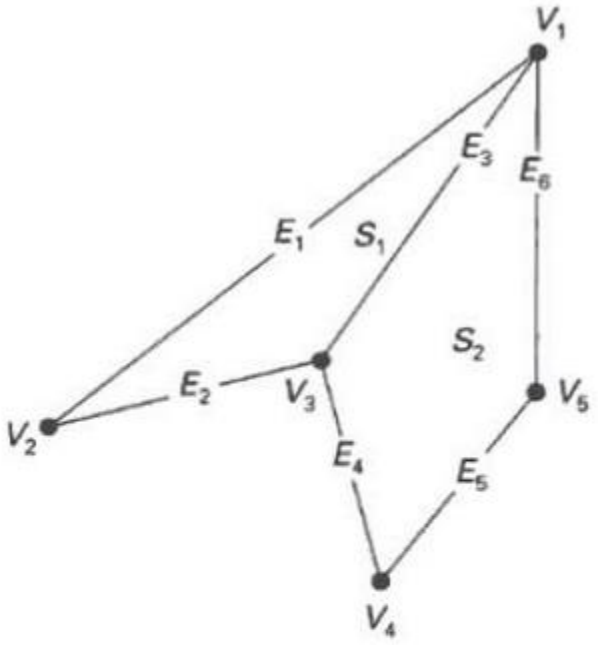
A 3D object represented by polygons

Polygon Tables

In this method, the surface is specified by the set of vertex coordinates and associated attributes. As shown in the following figure, there are five vertices, from v_1 to $v_{5v5.}$

- Each vertex stores x, y, and z coordinate information which is represented in the table as v_1: x_1, y_1, z_1.

- The Edge table is used to store the edge information of polygon. In the following figure, edge E_1 lies between vertex v_1 and v_2 which is represented in the table as E_1: v_1, v_2.
- Polygon surface table stores the number of surfaces present in the polygon. From the following figure, surface S_1 is covered by edges E_1, E_2 and E_3 which can be represented in the polygon surface table as S_1: E_1, E_2, and E_3.



VERTEX TABLE	
V_1:	x_1, y_1, z_1
V_2:	x_2, y_2, z_2
V_3:	x_3, y_3, z_3
V_4:	x_4, y_4, z_4
V_5:	x_5, y_5, z_5

EDGE TABLE	
E_1:	V_1, V_2
E_2:	V_2, V_3
E_3:	V_3, V_1
E_4:	V_3, V_4
E_5:	V_4, V_5
E_6:	V_5, V_1

POLYGON-SURFACE TABLE	
S_1:	E_1, E_2, E_3
S_2:	E_3, E_4, E_5, E_6

Plane Equations

The equation for plane surface can be expressed as −

$$Ax + By + Cz + D = 0$$

Where (x, y, z) is any point on the plane, and the coefficients A, B, C, and D are constants describing the spatial properties of the plane. We can obtain the values of A, B, C, and D by solving a set of three plane equations using the coordinate values for three non collinear points in the plane. Let us assume that three vertices of the plane are (x_1, y_1, z_1), (x_2, y_2, z_2) and (x_3, y_3, z_3).

Let us solve the following simultaneous equations for ratios A/D, B/D, and C/D. You get the values of A, B, C, and D.

$$(A/D)\ x_1 + (B/D)\ y_1 + (C/D)\ z_1 = -1$$

$$(A/D)\ x_2 + (B/D)\ y_2 + (C/D)\ z_2 = -1$$

$$(A/D)\ x_3 + (B/D)\ y_3 + (C/D)\ z_3 = -1$$

To obtain the above equations in determinant form, apply Cramer's rule to the above equations.

$$A = \begin{bmatrix} 1 & y_1 & z_1 \\ 1 & y_2 & z_2 \\ 1 & y_3 & z_3 \end{bmatrix} B = \begin{bmatrix} x_1 & 1 & z_1 \\ x_2 & 1 & z_2 \\ x_3 & 1 & z_3 \end{bmatrix} C = \begin{bmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{bmatrix} D = - \begin{bmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ x_3 & y_3 & z_3 \end{bmatrix}$$

For any point (x, y, z) with parameters A, B, C, and D, we can say that −

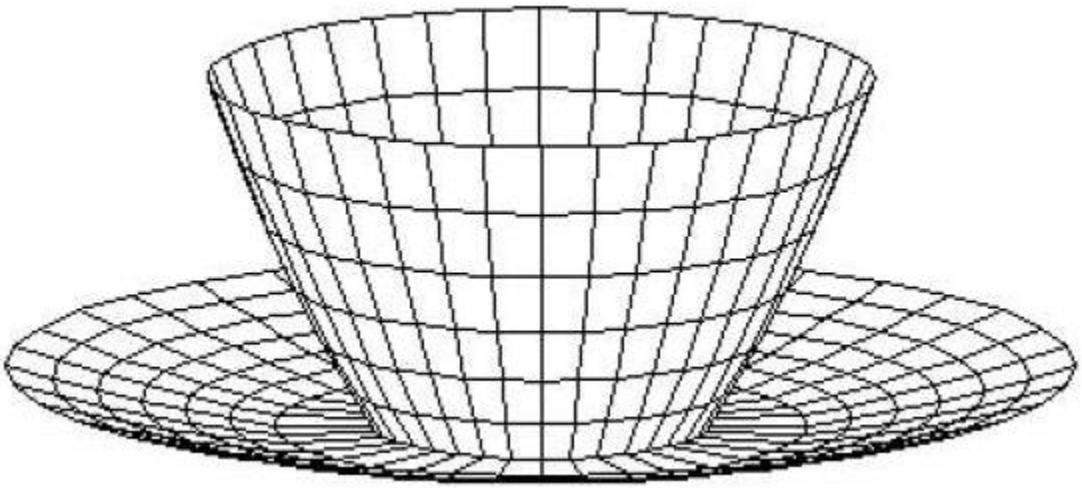
- $Ax + By + Cz + D \neq 0$ means the point is not on the plane.
- $Ax + By + Cz + D < 0$ means the point is inside the surface.
- $Ax + By + Cz + D > 0$ means the point is outside the surface.

Polygon Meshes

3D surfaces and solids can be approximated by a set of polygonal and line elements. Such surfaces are called **polygonal meshes**. In polygon mesh, each edge is shared by at most two polygons. The set of polygons or faces, together form the "skin" of the object.

This method can be used to represent a broad class of solids/surfaces in graphics. A polygonal mesh can be rendered using hidden surface removal algorithms. The polygon mesh can be represented by three ways −

- Explicit representation
- Pointers to a vertex list
- Pointers to an edge list



Advantages

- It can be used to model almost any object.
- They are easy to represent as a collection of vertices.
- They are easy to transform.
- They are easy to draw on computer screen.

Disadvantages

- Curved surfaces can only be approximately described.
- It is difficult to simulate some type of objects like hair or liquid.

In computer graphics, we often need to draw different types of objects onto the screen. Objects are not flat all the time and we need to draw curves many times to draw an object.

Types of Curves

A curve is an infinitely large set of points. Each point has two neighbors except endpoints. Curves can be broadly classified into three categories − **explicit, implicit,** and **parametric curves**.

Implicit Curves

Implicit curve representations define the set of points on a curve by employing a procedure that can test to see if a point in on the curve. Usually, an implicit curve is defined by an implicit function of the form −

$$f(x, y) = 0$$

It can represent multivalued curves (multiple y values for an x value). A common example is the circle, whose implicit representation is

$$\mathbf{x^2 + y^2 - R^2 = 0}$$

Explicit Curves

A mathematical function y = f(x) can be plotted as a curve. Such a function is the explicit representation of the curve. The explicit representation is not general, since it cannot represent vertical lines and is also single-valued. For each value of x, only a single value of y is normally computed by the function.

Parametric Curves

Curves having parametric form are called parametric curves. The explicit and implicit curve representations can be used only when the function is known. In practice the parametric curves are used. A two-dimensional parametric curve has the following form −

$$P(t) = f(t), g(t) \text{ or } P(t) = x(t), y(t)$$

The functions f and g become the (x, y) coordinates of any point on the curve, and the points are obtained when the parameter t is varied over a certain interval [a, b], normally [0, 1].

Bezier Curves

Bezier curve is discovered by the French engineer **Pierre Bézier**. These curves can be generated under the control of other points. Approximate tangents by using control points are used to generate curve. The Bezier curve can be represented mathematically as −

$$\sum_{k=0}^{n} P_i B_{ni}(t)$$

Where p_i

is the set of points and $B_{ni}(t)$

represents the Bernstein polynomials which are given by −

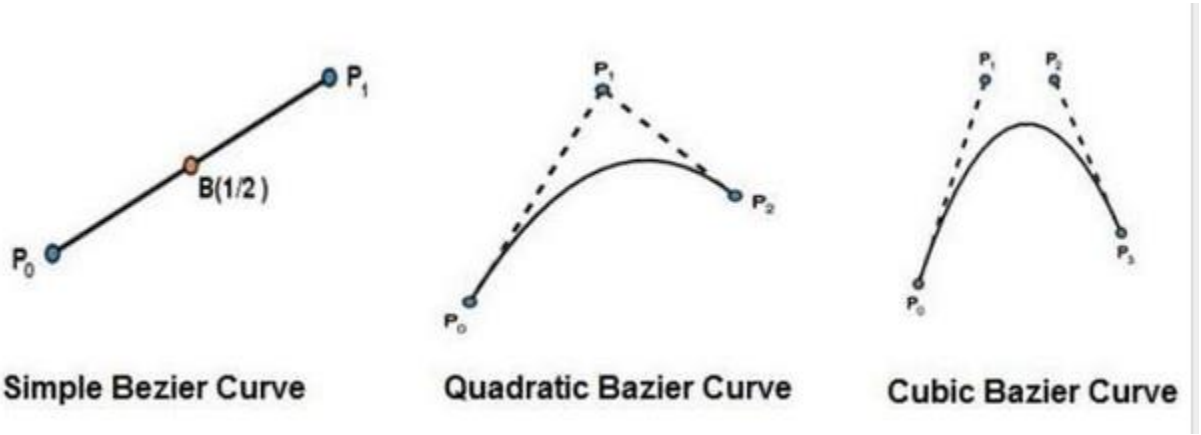
$$B_{ni}(t) = \binom{n}{i}(1-t)^{n-i}t^i$$

Where **n** is the polynomial degree, **i** is the index, and **t** is the variable.

The simplest Bézier curve is the straight line from the point P_0

to P_1

. A quadratic Bezier curve is determined by three control points. A cubic Bezier curve is determined by four control points.



Properties of Bezier Curves

Bezier curves have the following properties −

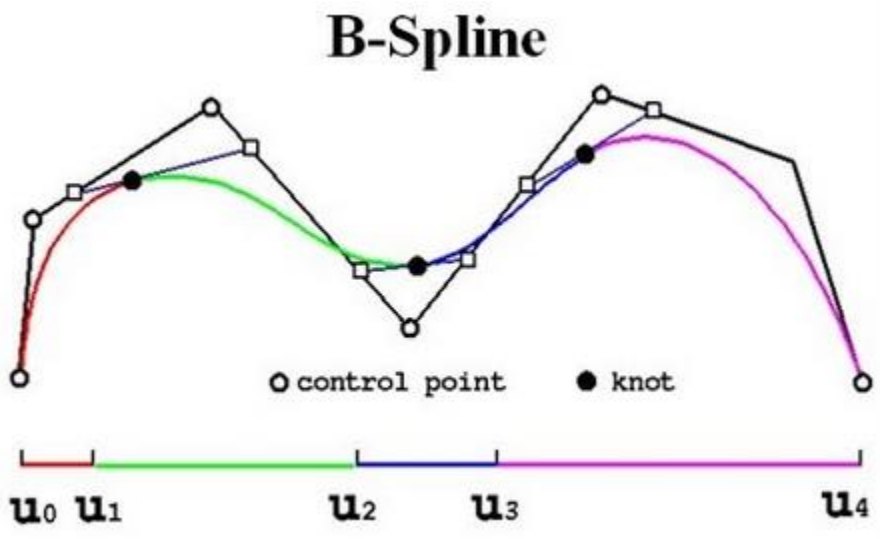
- They generally follow the shape of the control polygon, which consists of the segments joining the control points.
- They always pass through the first and last control points.
- They are contained in the convex hull of their defining control points.
- The degree of the polynomial defining the curve segment is one less that the number of defining polygon point. Therefore, for 4 control points, the degree of the polynomial is 3, i.e. cubic polynomial.
- A Bezier curve generally follows the shape of the defining polygon.
- The direction of the tangent vector at the end points is same as that of the vector determined by first and last segments.
- The convex hull property for a Bezier curve ensures that the polynomial smoothly follows the control points.
- No straight line intersects a Bezier curve more times than it intersects its control polygon.
- They are invariant under an affine transformation.
- Bezier curves exhibit global control means moving a control point alters the shape of the whole curve.
- A given Bezier curve can be subdivided at a point t=t0 into two Bezier segments which join together at the point corresponding to the parameter value t=t0.

B-Spline Curves

The Bezier-curve produced by the Bernstein basis function has limited flexibility.

- First, the number of specified polygon vertices fixes the order of the resulting polynomial which defines the curve.
- The second limiting characteristic is that the value of the blending function is nonzero for all parameter values over the entire curve.

The B-spline basis contains the Bernstein basis as the special case. The B-spline basis is non-global.



A B-spline curve is defined as a linear combination of control points Pi and B-spline basis function $N_{i,}$

k (t) given by

$$C(t)=\sum_{ni=0} P_i N_{i,k}(t),$$

$$n \geq k-1,\ t \epsilon [tk-1, tn+1]$$

Where,

- $\{p_i$: i=0, 1, 2….n} are the control points

- k is the order of the polynomial segments of the B-spline curve. Order k means that the curve is made up of piecewise polynomial segments of degree k - 1,

- the $N_{i,k}(t)$

  - are the "normalized B-spline blending functions". They are described by the order k and by a non-decreasing sequence of real numbers normally called the "knot sequence".

$$t_i : i = 0, \ldots n + K$$

The N_i, k functions are described as follows −

$$N_{i,1}(t) = \{1, 0, if u \epsilon [t_i, t_{i+1}) Otherwise$$

and if k > 1,

$$N_{i,k}(t) = t - t_i t_{i+k-1} N_{i,k-1}(t) + t_{i+k} - t t_{i+k} - t_{i+1} N_{i+1,k-1}(t)$$

and

$$t \epsilon [t_{k-1}, t_{n+1})$$

Properties of B-spline Curve

B-spline curves have the following properties −

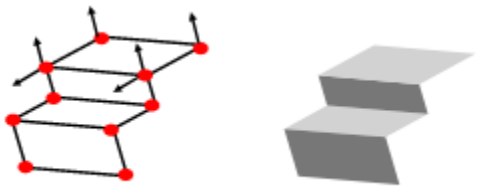
- The sum of the B-spline basis functions for any parameter value is 1.
- Each basis function is positive or zero for all parameter values.
- Each basis function has precisely one maximum value, except for k=1.
- The maximum order of the curve is equal to the number of vertices of defining polygon.
- The degree of B-spline polynomial is independent on the number of vertices of defining polygon.
- B-spline allows the local control over the curve surface because each vertex affects the shape of a curve only over a range of parameter values where its associated basis function is nonzero.
- The curve exhibits the variation diminishing property.
- The curve generally follows the shape of defining polygon.
- Any affine transformation can be applied to the curve by applying it to the vertices of defining polygon.
- The curve line within the convex hull of its defining polygon.

SOLID MODELING

Solid modeling is the most advanced method of geometric modeling in three dimensions. Solid modeling is the representation of the solid parts of the object on your computer. The typical geometric model is made up of wire frames that show the object in the form of wires. This wire frame structure can be two dimensional, two and half dimensional or three dimensional. Providing surface representation to the wire three dimensional views of geometric models makes the object appear solid on the computer screen and this is what is called as solid modeling.

Polygon Meshes

•A polygon meshis a collection of polygons, along with a normal vector associated to each polygon vertex :– An edge connects two vertices– A polygon is a closed sequence of edges– An edge can be shared by two adjacent polygons– A vertex is shared by at least two edges– A normal vector pointing "outside" is associated with each polygon vertex



Properties:

•Connectedness:A mesh is connectedif thereis an path of edges between any two vertices •Simplicity:A mesh is simpleif the mesh has noholes in it•Planarity:A mesh is planarif every face of it is a planar polygon•Convexity:The mesh is convexif the line connecting any two points in the mesh belongs to the mesh

ADVANTAGES

Solid modeling is one of the most important applications of the CAD software and it has been becoming increasingly popular of late. The solid modeling CAD software helps the designer to see the designed object as if it were the real manufactured product. It can be seen from various directions and in various views. This helps the designer to be sure that the object looks exactly as they wanted it to be. It also gives additional vision to the designer as to what more changes can be done in the object.

COLOR MODELS

The purpose of a color model is to facilitate the specification of colors in some standard generally accepted way. In essence, a color model is a specification of a 3-D coordinate system and a subspace within that system where each color is represented by a single point.

RGB Color Model

In the RGB model, each color appears as a combination of red, green, and blue. This model is called additive, and the colors are called primary colors. The primary colors can be added to produce the

secondary colors of light (see Figure "Primary and Secondary Colors for RGB and CMYK Models") - magenta (red plus blue), cyan (green plus blue), and yellow (red plus green). The combination of red, green, and blue at full intensities makes white.

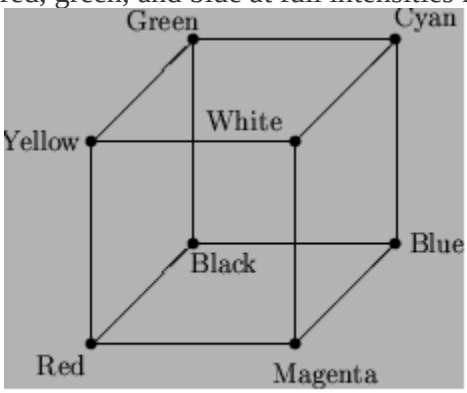


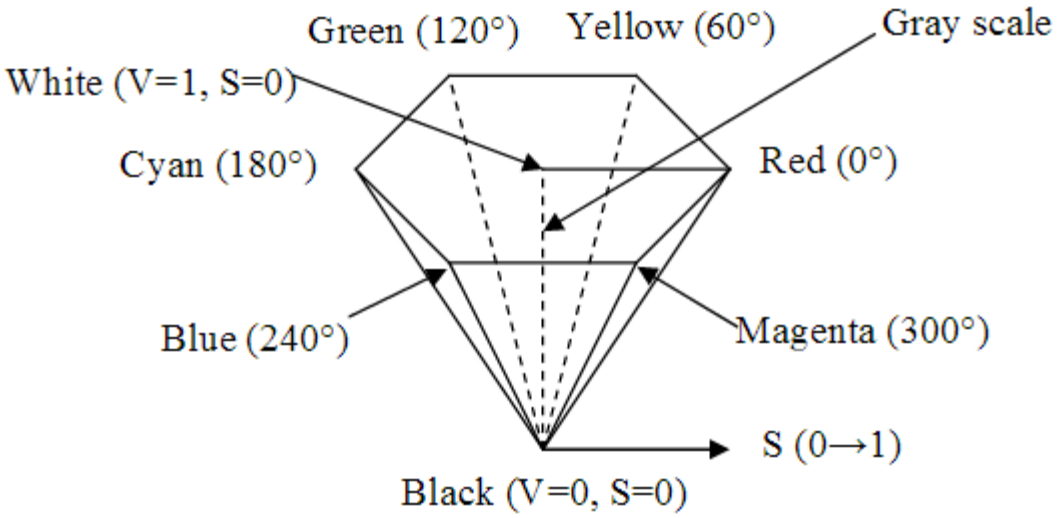
Figure: The RGB-cube

The color subspace of interest is a cube, in which RGB values are at three corners; cyan, magenta, and yellow are the three other corners, black is at their origin; and white is at the corner farthest from the origin.
The gray scale extends from black to white along the diagonal joining these two points. The colors are the points on or inside the cube, defined by vectors extending from the origin.
Thus, images in the RGB color model consist of three independent image planes, one for each primary color.

The importance of the RGB color model is that it relates very closely to the way that the human eye perceives color. RGB is a basic color model for computer graphics because color displays use red, green, and blue to create the desired color. Therefore, the choice of the RGB color space simplifies the architecture and design of the system. Besides, a system that is designed using the RGB color space can take advantage of a large number of existing software routines, because this color space has been around for a number of years.

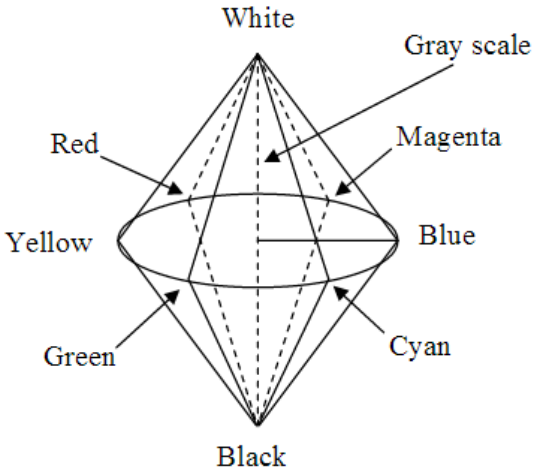
HSV Color Model

The HLS (hue, lightness, saturation) and HSV (hue, saturation, value) color models were developed to be more “intuitive” in manipulating with color and were designed to approximate the way humans perceive and interpret color. *Hue* defines the color itself.
The values for the hue axis vary from 0 to 360 beginning and ending with red and running through green, blue and all intermediary colors. *Saturation* indicates the degree to which the hue differs from a neutral gray. The values run from 0, which means no color saturation, to 1, which is the fullest saturation of a given hue at a given illumination. Intensity component - *lightness* (HLS) or *value* (HSV), indicates the illumination level.
Both vary from 0 (black, no light) to 1 (white, full illumination). The difference between the two is that maximum saturation of hue (S=1) is at *value* V=1 (full illumination) in the HSV color model, and at *lightness* L=0.5 in the HLS color model.



HLS COLOR MODEL

This model, Hue, Lightness, and Saturation, was popularized by Tektronix who used it to define the color effects on its monitors. It uses a double cone, as shown below:

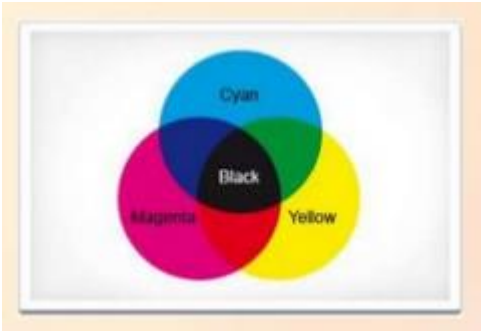


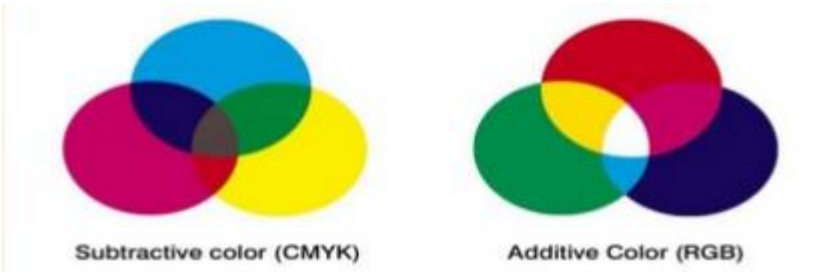
The hues are specifies by angles, as they were for HSV, but in this model Blue is at 0°, Magenta is at 60°, Red is at 120°, Yellow is at 180°, Green is at 240°, and Cyan is at 300°. So the order on which the colors appear is the same as before, and complementary colors are still on opposite sides of the circle, separated by 180°, but the color sequence begins with blue instead of red. The angle is measured from above, as before, beginning at the line shown from medium gray to blue. The hue definitions now lie on a circle, as compared to the hexagon that was used for HSV. This is much easier to deal with since full saturation of any hue will now have an S value of 1.0, as compared to, for example, the $\sqrt{3}/2$ that we had to use for the S value for orange using HSV. Once again, gray scales appear on the center line of symmetry, with L= 0 at the bottom and L= 1 at the top. In this mode l the line is twice as long as in HSV.

Pure colors have an L value of 0.5. So, for example, pure orange is at an HLS triple of (150°, 0.5, 1.0).
Overall HSV seems to be the preferred method for interactive selection of colors.

CMYK Color Model

The CMYK color model is a subset of the RGB model and is primarily used in color print production. CMYK is an acronym for cyan, magenta, and yellow along with black (noted as K). The CMYK color space is subtractive, meaning that cyan, magenta yellow, and black pigments or inks are applied to a white surface to subtract some color from white surface to create the final color. Cyan is white minus red, magenta is white minus green, and yellow is white minus blue. Subtracting all colors by combining the CMY at full saturation should, in theory, render black. However, impurities in the existing CMY inks make full and equal saturation impossible, and some RGB light does filter through, rendering a muddy brown color. Therefore, the black ink is added to CMY.





$$\begin{pmatrix} C \\ M \\ Y \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} - \begin{pmatrix} R \\ G \\ B \end{pmatrix}$$

CIE XYZ Color Model

The XYZ color space is an international standard developed by the CIE (Commission Internationale de l'Eclairage). This model is based on three hypothetical primaries, XYZ,and all visible colors can be represented by using only positive values of X, Y, and Z. The CIE XYZ primaries are hypothetical because they do not correspond to any real light wavelengths. The Y primary is intentionally defined to match closely to luminance, while X and Z primaries give color information. The main advantage of the CIE XYZ space (and any color space based on it) is that this space is completely device-independent.

Intel IPP functions use the following basic equations, to convert between gamma corrected

R’G’B’ and CIE XYZ models:
X = 0.412453*R’ + 0.35758 *G’ + 0.180423*B’
Y = 0.212671*R’ + 0.71516 *G’ + 0.072169*B’
Z = 0.019334*R’ + 0.119193*G’ + 0.950227*B’

The equations for X,Y,Z calculation are given on the assumption that R’,G’, and B’ values are normalized to the range [0..1].
R’ = 3.240479 * X - 1.53715 * Y - 0.498535 * Z
G’ = -0.969256 * X + 1.875991 * Y + 0.041556 * Z
B’ = 0.055648 * X - 0.204043 * Y + 1.057311 * Z

The equations for R’,G’, and B’ calculation are given on the assumption that X,Y, and Z values are in the range [0..1].

YIQ Color Model

YIQ is the system used for US TV broadcast (PAL is the most common system used in other countries). The primary goals of the system were to provide a signal that could be directly displayed by black and white TVs, while also providing easy coding and decoding of RGB signals.

The conversions from RGB to YIQ and back are given by the matrices:

$$\begin{bmatrix} Y \\ I \\ Q \end{bmatrix} = \begin{bmatrix} .299 & .587 & .114 \\ .701 & -.587 & -.114 \\ -.299 & -.587 & .886 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

and

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1.000 & 1.000 & .000 \\ 1.000 & -.509 & -.194 \\ 1.000 & .000 & 1.000 \end{bmatrix} \begin{bmatrix} Y \\ I \\ Q \end{bmatrix}$$

where obviously the two matrices are inverses. The Y component, which is the same as the Y value in the CIE system, is the signal that is used directly by black and white TVs.

Y is said to convey the luminance information and is transmitted on a separate carrier signal from the chromaticity components, I and Q.
.
This encoding is of far more importance for film & TV people than it is for computer graphics people.

Ray Tracing-

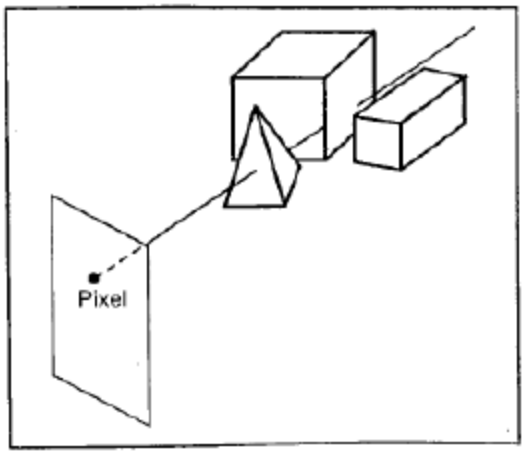


Fig. 10.22 A ray along the line of sight from a pixel position through a scene

If we consider the line of sight from a pixel position on the view plane through a scene, as in Fig. 10.22, we can determine which objects in the scene (if any) intersect this line. From the intersection points with different object, we can identify the visible surface as the one whose intersection point is closest to the pixel. **Ray tracing** is an extension of this basic idea. Here, instead of identifying for the visible surface for each pixel, we continue to bounce the ray around the picture. This is illustrated in Fig. 10.23. When the ray is bouncing from one surface to another surface, it contributes the intensity for that surfaces. This is a simple and powerful rendering technique for obtaining global reflection and transmission effects.

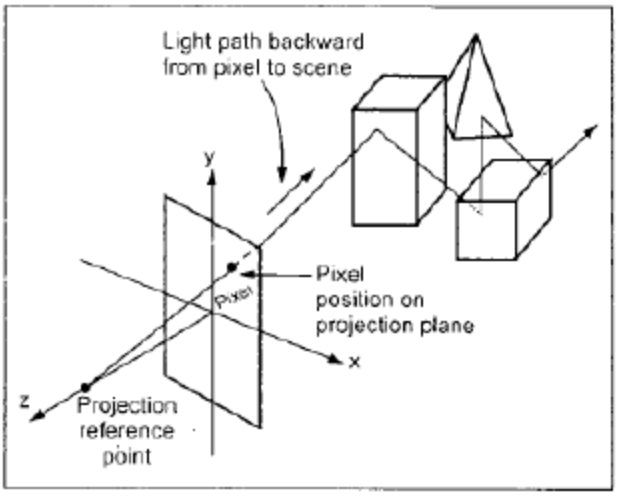


Fig. 10.23 Bouncing of ray around the scene

As shown in the Fig. 10.23, usually pixel positions are designated in the xy plane and projection reference point lie on the z axis, i.e. the pixel screen area is centered on viewing coordinate origin. With this coordinate system the contributions to a pixel is determined by tracing a light path backward from the pixel to the picture.

For each pixel ray, each surface is tested in the picture to determine if it is intersected by the ray. If surface is intersected, the distance from the pixel to the surface intersection point is calculated. The smallest calculated intersection distance identifies the visible surface for that pixel. Once the visible surface is identified the ray is reflected off the visible surface along a specular path where the angle reflection equals angle of incidence. If the surface is transparent, the ray is passed through the surface in the refraction direction. The ray reflected from the visible surface or passed through the transparent surface in the refraction direction is called **secondary ray.** The ray after reflection or refraction strikes another visible surface. This process is repeated recursively to produce the next generations of reflection and refraction paths. These paths are represented by **ray tracing tree** as shown in the Fig. 10.24.

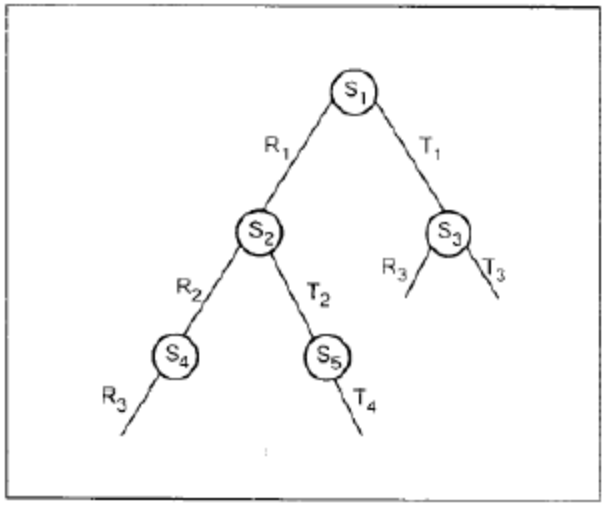


Fig. 10.24 Binary ray-tracing tree

As shown in the Fig. 10.24, the left branches in the binary ray tracing tree are used to represent reflection paths, and right branches are used to represent transmission paths. The recursion depth for ray tracing tree is determined by the amount of storage available, or by the user. The ray path is terminated when predetermined depth is reached or if ray strikes a light source. As we go from top to bottom of the tree, surface intensities are attenuated by distance from the parent surface. The surface intensities of all the nodes are added traversing the ray tree from bottom to top to determine the intensity of the pixel.

If pixel ray does not intersect to any surface then the intensity value of the background is assigned to the pixel. If a pixel ray intersects a nonreflecting light source, the pixel can be assigned the intensity of the source, although light sources are usually placed beyond the path of the initial rays.

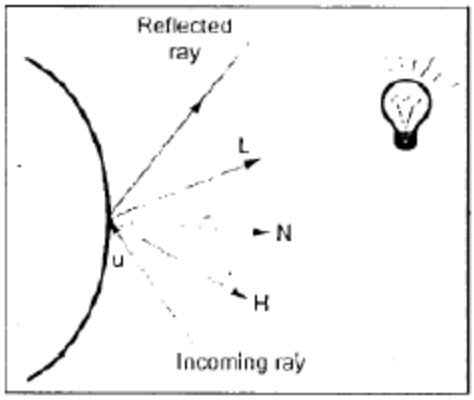


Fig. 10.25 Surface intersected by a ray and the unit vectors

The Fig. 10.25 shows a surface intersected by a ray and unit vectors needed for the reflected light-intensity calculations. Here, u is the unit vector in the direction of the ray path, N is the unit surface normal, R is the unit reflection vector, L is the unit vector pointing to the light source, and H is the unit vector halfway between V (viewer) and L (light source).

If any object intersects the path along L between the surface and the point light source, the surface is in shadow with respect to that source. Hence a path along L is referred to as **shadow ray**. Ambient light at the surface is given as $K_a I_a$, diffuse reflection due to the surface is proportional to K_d (N.L), and the specular-reflection component is proportional to K_s $(H.N)^{n_s}$. We know that, the specular reflection direction for R depends on the surface normal and the incoming ray direction. It is given as

$$R = u - (2u \,.\, N)\, N$$

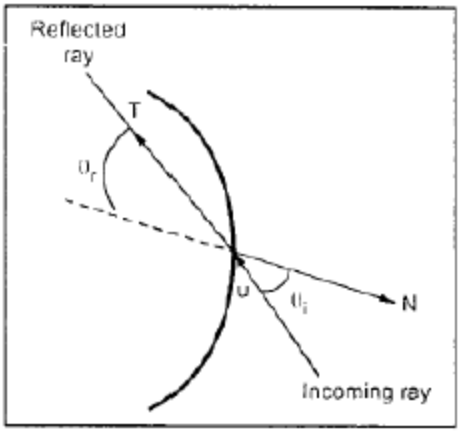


Fig. 10.26 Refracted ray through the transparent material

In a transparent material light passes through the material and we have to calculate intensity contributions from light transmitted through the material. Referring the Fig. 10.26, we can obtain the unit transmission vector from vectors u and N as

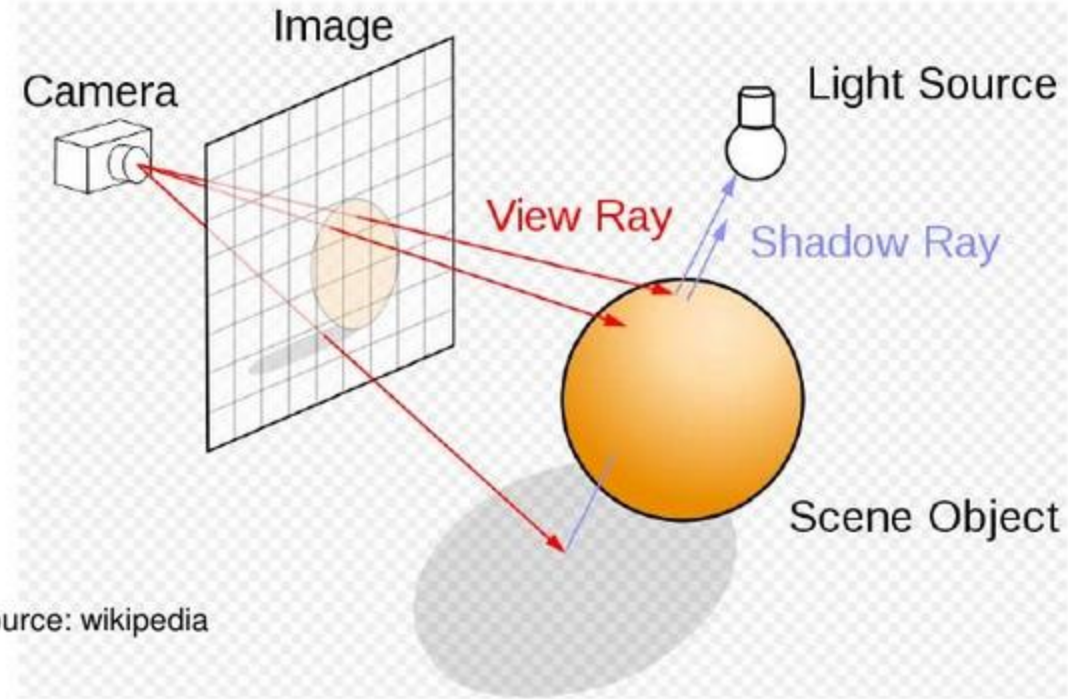
$$T = \frac{\eta_i}{\eta_r} u - (\cos\theta_r - \frac{\eta_i}{\eta_r} \cos\theta_i) N$$

where η_i and η_r are the indices of relfection in the incident material and the refracting meterial, respectively. The angle of refraction θ_r is given by Snell's law

$$\cos\theta_r = \sqrt{1 - \left(\frac{\eta_i}{\eta_r}\right)^2 (1 - \cos^2\theta_i)}$$

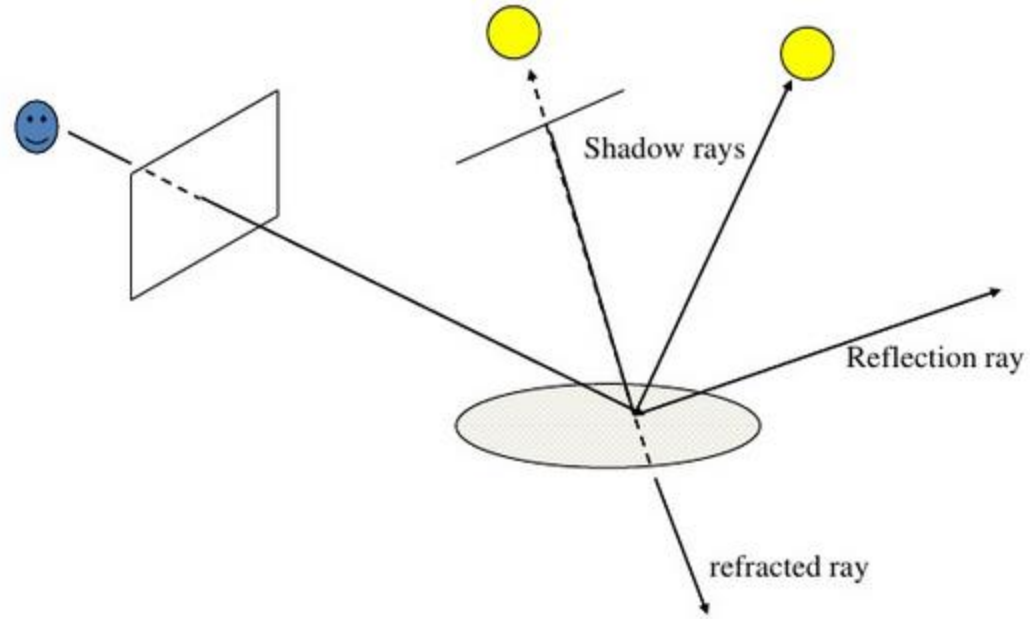
- Ray tracing is a technique for rendering three-dimensional graphics with very complex light interactions. This means you can create pictures full of mirrors, transparent surfaces, and shadows, with stunning results.
- A very simple method to both understand and implement.
- It is based on the idea that you can model reflection and refraction by recursively following the path that light takes as it bounces through an environment

Ray Tracing Model



ource: wikipedia

Ray tracing



Ray tracing algorithm

- Builds the image pixel by pixel
- Cast additional rays from the hit point to determine the pixel color
  - Shoot rays toward each light. If they hit something, the object is shadowed from that light, otherwise use “standard model” for the light
  - Reflection rays for mirror surfaces, to see what should be reflected in the mirror
  - Refraction rays to see what can be seen through transparent objects
  - Sum all the contributions to get the pixel color

Recursive ray tracing

- When a reflected or refraction ray hits a surface, repeat the whole process from that point
  - Send more out shadow rays
  - Send out new reflected rays (if required)
  - Send out a new refracted ray (if required)
  - Generally, reduce the weight of each additional ray when computing the contributions to the surface color
  - Stop when the contribution from a ray is too small to notice or maximum recursion level has been reached

Common Image File Formats

There are numerous image file types out there so it can be hard to know which file type best suits your image needs. Some image types such a TIFF are great for printing while others, like JPG or PNG, are best for web graphics.

The list below outlines some of the more common file types and provides a brief description, how the file is best used, and any special attributes the file may have.

TIFF (.tif, .tiff)

TIFF or Tagged Image File Format are lossless images files meaning that they do not need to compress or lose any image quality or information (although there are options for compression), allowing for very high-quality images but also larger file sizes.

Compression: Lossless - no compression. Very high-quality images.
Best For: High quality prints, professional publications, archival copies
Special Attributes: Can save transparencies

Bitmap (.bmp)

BMP or Bitmap Image File is a format developed by Microsoft for Windows. There is no compression or information loss with BMP files which allow images to have very high quality, but also very large file sizes. Due to BMP being a proprietary format, it is generally recommended to use TIFF files.

Compression: None
Best For: High quality scans, archival copies

JPEG (.jpg, .jpeg)

JPEG, which stands for Joint Photographic Experts Groups is a “lossy” format meaning that the image is compressed to make a smaller file. The compression does create a loss in quality but this loss is generally not noticeable. JPEG files are very common on the Internet and JPEG is a popular format for digital cameras - making it ideal for web use and non-professional prints.

Compression: Lossy - some file information is compressed or lost
Best For: Web Images, Non-Professional Printing, E-Mail, Powerpoint
Special Attributes: Can choose amount of compression when saving in image editing programs like Adobe Photoshop or GIMP.

GIF (.gif)

GIF or Graphics Interchange Format files are widely used for web graphics, because they are limited to only 256 colors, can allow for transparency, and can be animated. GIF files are typically small is size and are very portable.

Compression: Lossless - compression without loss of quality
Best For: Web Images
Special Attributes: Can be Animated, Can Save Transparency

PNG (.png)

PNG or Portable Network Graphics files are a lossless image format originally designed to improve upon and replace the gif format. PNG files are able to handle up to 16 million colors, unlike the 256 colors supported by GIF.

Compression: Lossless - compression without loss of quality
Best For: Web Images
Special Attributes: Save Transparency

EPS (.eps)

An EPS or Encapsulated PostScript file is a common vector file type. EPS files can be opened in many illustration applications such as Adobe Illustrator or CorelDRAW.

Compression: None - uses vector information
Best For: Vector artwork, illustrations
Special Attributes: Saves vector information

RAW Image Files (.raw, .cr2, .nef, .orf, .sr2, and more)

RAW images are images that are unprocessed that have been created by a camera or scanner. Many digital SLR cameras can shoot in RAW, whether it be a .raw, .cr2, or .nef. These RAW images are the equivalent of a digital negative, meaning that they hold a lot of image information, but still need to be processed in an editor such as Adobe Photoshop or Lightroom.

Compression: None
Best For: Photography
Special Attributes: Saves metadata, unprocessed, lots of information

UNIT V
USER INTERFACE DESIGN

6.1 Introduction

As hardware cost is plummeting, which is considered as the major bottleneck for the progress; now communication devices is more listened for better development. For that reason, techniques for developing high-quality user interfaces are moving to the forefront in computer science and are becoming the "last frontier" in providing computing to a wide variety of users—as other aspects of technology continue to improve, but the human users remain the same. Interest in the quality of user-computer interfaces is a recent part of the formal study of computers. The emphasis until the early 1980s was on optimizing two scarce hardware resources, computer time and memory. Program efficiency was the highest goal. With today's plummeting hardware costs and powerful graphics-oriented personal computing environments the focus turns to optimizing user efficiency rather than computer efficiency. Thus, although many of the ideas presented in this chapter require additional CPU cycles and memory space, the potential rewards in user productivity and satisfaction well outweigh the modest additional cost of these resources. The quality of the user interface often determines whether users enjoy or despise a system, whether the designers of the system are praised or damned, whether a system succeeds or fails in the market. Actually, a poor user interface such as in air traffic control or in nuclear power plant monitoring can lead to catastrophic consequences.

The desktop user-interface metaphor, with its windows, icons, and pull-down menus, all making heavy use of raster graphics, is popular because it is easy to learn and requires little typing skill. Most users of such systems are not computer programmers and have little sympathy for the old-style difficult-to-learn keyboard-oriented command-language interfaces that many programmers take for granted. The designer of an interactive graphics application must be sensitive to users' desire for easy-to-learn yet powerful interfaces. In this chapter, we discuss the three basic low-level elements of user interfaces: input devices, interaction techniques, and interaction tasks. **Interaction techniques are the primitive building blocks from which a user interface is crafted.**

We focus in this chapter on input devices—those pieces of hardware by which a user enters information into a computer system. Input devices for the earliest computers were switches and knobs, jumper wires placed in patch boards, and punched cards. These were followed by the teletype, the text-only forerunner of today's interactive

terminals. The mouse and keyboard now predominate, but a wide variety of input devices can be used. **An interaction task is the entry of a unit of information by the user. Basic interaction tasks are *position, text, select,* and *quantify*.** The unit of information that is input in a position interaction task is of course a position; the text task yields a text string; the select task yields an object identification; and the quantify task yields a numeric value. A designer begins with the interaction tasks necessary for a particular application. For each such task, the designer chooses an appropriate interaction device and interaction technique. Many different *interaction techniques* can be used for a given interaction task, and there may be several different ways of using the same device to perform the same task. For instance, a selection task can be carried out by using a mouse to select items from a menu, using a keyboard to enter the name of the selection, pressing a function key, circling the desired command with the mouse, or even writing the name of the command with the mouse. Similarly, a single device can be used for different tasks: A mouse is often used for both positioning and selecting.

Interaction tasks are defined by *what* the user accomplishes, whereas logical input devices categorize *how* that task is accomplished by the application program and the graphics system. Interaction tasks are user-centered, whereas logical input devices are a programmer and graphics-system concept. By analogy with a natural language, single actions with input devices are similar to the individual letters of the alphabet from which words are formed. The sequence of input-device actions that makes up an interaction technique is analogous to the sequence of letters that makes up a word. A word is a unit of meaning; just as several interaction techniques can be used to carry out the same interaction task, so too words that are synonyms convey the same meaning. An interactive dialogue is made up of interaction-task sequences, just as a sentence is constructed from word sequences.

6.2 Concept of Positioning and Pointing

Most display terminals provide the user with an alphanumeric keyboard with which to type commands and enter data for the program. For some applications, however, the keyboard is inconvenient or inadequate. For example, the user may wish to indicate one of a number of symbols on the screen, in order to erase the symbol. If each symbol is labeled, he can do so by typing the symbol’s name; by pointing at the

symbol, however, he may be able to erase more rapidly, and the extra clutter of labels can be avoided.

Another problem arises if the user has to add lines or symbols to the picture on the screen. Although he can identify an items's position by typing coordinates he can do so even better by pointing at the screen, particularly if what matters most is the items's position relative to the rest of the picture.

These two examples illustrate the two basic types of graphical interaction: pointing at items already on the screen and positioning new items. The need to interact in these ways has stimulated the developed of a number of different types of graphical input device, some of which are described in this chapter.

Ideally a graphical input device should lend itself both to pointing and to positioning. In reality there are no devices with this versatility. Most devices are much better at positioning than at pointing; one device, the light pen, is the exact opposite. Fortunately, however we can supplement the deficiencies of these devices by software and in this way produce hardware-software system that has both capabilities. Nevertheless the distinction between pointing and positioning capability is extremely important.

Another important distinction is between devices that can be used directly on the screen surface and devices that cannot. The latter might appear to be less useful, but this is far from true. Radar operators and air-traffic controllers have for years used devices like the joystick and the tracker ball neither of which can be pointed at the screen. The effectiveness of these input devices depends on the use of visual feedback: the x and y outputs of the device control the movement of a small cross, or cursor, displayed on the screen. The user of the device steers the cursor around the screen as if it were a toy boat on the surface of a pond. Although this operation sounds as if it requires a lot of skill, it is in fact very easy.

The use of visual feedback has an additional advantage: just as in any control system, it compensates for any lack of linearity in the device. A linear input device is one that faithfully increases or decreases the input coordinate value in exact proportion to the user's hand movement. If the device is being used to trace a graph or a map. Linearity is important. A cursor, however, can be controlled quite easily even if the device behaves in a fairly nonlinear fashion. For example, the device may be much less

sensitive near the left – hand region of its travel: a 1 – inch hand movement may change the x value by only 50 units, whereas the same movement elsewhere may change x by 60 units. The user will simply change his hand movement to compensate, often without even noticing the no linearity. This phenomenon has allowed simple, inexpensive devices like the mouse to be used very successfully for graphical input.

6.3 Interactive Graphic Devices

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

6.3.1 Keyboards

The well-known QWERTY keyboard has been with us for many years. It is ironic that this keyboard was originally designed to *slow down* typists, so that the typewriter hammers would not be so likely to jam. Studies have shown that the newer Dvorak keyboard , which places vowels and other high-frequency characters under the home positions of the fingers, is somewhat faster than is the QWERTY design. It has not been widely accepted. Alphabetically organized keyboards are sometimes used when many of the users are non typists. But more and more people are being exposed to QWERTY keyboards, and experiments have shown no advantage of alphabetic over QWERTY keyboards .In recent years, the chief force serving to displace the keyboard has been the shrinking size of computers, with laptops, notebooks, palmtops, and personal digital assistants. The typewriter keyboard is becoming the largest component of such pocket-sized devices, and often the main component standing in the way of reducing its overall size. The *chord keyboard* has five keys similar to piano keys, and is operated with one hand, by pressing one or more keys simultaneously to "play a chord." With five keys, 31 different chords can be played. Learning to use a chord keyboard (and other similar stenographer style keyboards) takes longer than learning the QWERTY keyboard, but skilled users can type quite rapidly, leaving the second hand free for other tasks. This increased training time means, however, that such keyboards are not suitable substitutes for general use of the standard alphanumeric keyboard. Again, as computers become smaller, the benefit of a

keyboard that allows touch typing with only five keys may come to outweigh the additional difficulty of learning the chords. Other keyboard-oriented considerations, involving not hardware but software design, are arranging for a user to enter frequently used punctuation or correction characters without needing simultaneously to press the control or shift keys, and assigning dangerous actions (such as delete) to keys that are distant from other frequently used keys.

6.3.2 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 with graphical icons. Other systems can be adapted for touch input by fitting a transparent device with 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. The opposite vertical and horizontal edges contain light detectors. 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 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 the 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.

6.3.3 Light pens

The pencil-shaped devices 's are used to select screen positions by detecting the light coming from point on the CRT screen. They are sensitive to the short burst of light emitted from the phosphor coating at the instant the electron beam strikes a particular point. Other light sources, such as the background light in the room, are usually not detected by a light pen. An activated light pen, pointed at a spot on the screen as the electron beam lights up that spot, generates an electrical pulse that causes the coordinate position of the electron beam to be recorded. As with cursor-positioning devices, recorded light-pen coordinates can be used to position an object or to select a processing option. Although light pens are still with us, they are not as popular as they once were since they have several disadvantages compared to other input devices that have been developed. For one, when a light pen is pointed at the screen, part of the screen image is obscured by the hand and pen. And prolonged use of the light pen can cause arm fatigue. Also, light pens require special implementation 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 intensity assigned to each screen pixel. In addition, light pens sometime give false readings due to background lighting in a room.

6.3.4 Graphics Tablets

One type of digitizer is the graphics tablet (also referred to as a data tablet), which is used to input two-dimensional coordinates by activating a hand cursor or stylus at selected positions on a flat surface. A hand cursor contains cross hairs for sighting positions, while a stylus is a pencil-shaped device that is pointed at positions on the tablet. This allows an artist to produce different brush strokes with different pressures on the tablet surface. Tablet size varies from 12 by 12 inches for desktop models to 4 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 wire 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, a their signal strength, coded pulses, or phase shifts can be used to determine the position on the tablet.

6.3.5 Joysticks

A joystick consists of a small, vertical lever (called the stick) mounted on a base that is used to steer the screen cursor around. Most joysticks select screen positions with actual stick movement; others respond to pressure on the stick. The distance that the stick is moved in any direction from its center position corresponds to screen-cursor movement in that direction. Potentiometers mounted at the base of the joystick measure the amount of movement, and springs return the stick to the center position when it is released. One or more buttons can be programmed to act as input switchs to signal certain actions once a screen position has been selected.

6.3.6 Mouse

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

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, or three buttons are usually included on the top of the mouse for signaling the execution of some operation, such as recording cursor position or invoking a function. Most general-purpose graphics systems now include a mouse and a keyboard as the major input devices.

6.3.7 Voice Systems

Speech recognizers are used in some graphics workstations as input devices to accept 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 for a particular operator by having the operator speak the command words to be used into the system. Each word is spoken several times, and the system analyzes the word and establishes a frequency pattern for that word in the dictionary along with the corresponding function to be performed. Later, when a voice command is given, the system searches the dictionary for a frequency-pattern match. Voice input is typically spoken into a microphone mounted on a headset. The microphone is designed to minimize input of other background sounds. If a different operator is to use the system, the dictionary must be reestablished with that operator's voice patterns. Voice systems have some advantage over other input devices, since the attention of the operator does not have to be switched from one device to another to enter a command.

6.3.8 Other Devices

Here we discuss some of the less common, and in some cases experimental, 2D interaction devices. Voice recognizers, which are useful because they free the user’s hands for other uses, apply a pattern-recognition approach to the waveforms created when we speak a word. The waveform is typically separated into a number of different frequency bands, and the variation over time of the magnitude of the waveform. in each band forms the basis for the pattern matching. However, mistakes can occur in the pattern matching, so it is especially important that an application using a recognizer provide convenient correction capabilities. Voice recognizers differ in whether or not they must be trained to recognize the waveforms of a particular speaker, and whether they can recognize connected speech as opposed to single words or phrases. Speaker-independent recognizers have very limited vocabularies—typically, they include only the ten digits and 50 to 100 words. Some discrete word recognizers can recognize vocabularies of thousands of different words after appropriate training. But if the user has a cold, the recognizer must be retrained. The user of a discrete word recognizer must pause for a fraction of a second after each word to cue the system that a word end has occurred. The more difficult task of recognizing connected speech from a limited vocabulary can now be performed by off-the-shelf hardware and software, but with somewhat less accuracy. As the vocabulary becomes larger, however, artificial-intelligence techniques are needed to

exploit the context and meaning of a sequence of sentences to remove ambiguity. A few systems with vocabularies of 20,000 or more words can recognize sentences such as "Write Mrs. Wright a letter right now!" Voice synthesizers create waveforms that approximate, with varying degrees of realism, spoken words. The simplest synthesizers use *phonemes*, the basic sound units that form words. This approach creates an artificial-sounding, inflection-free voice. More sophisticated phoneme-based systems add inflections. Other systems actually play back digitized spoken words or phrases. They sound realistic, but require more memory to store the digitized speech. Speech is best used to augment rather than to replace visual feedback, and is most effective when used sparingly. For instance, a training application could show a student a graphic animation of some process, along with a voice narration describing what is being seen. See for additional guidelines for the effective application of speech recognition and generation in user-computer interfaces, and for an introduction to speech interfaces, and for speech recognition technology. The data tablet has been extended in several ways. Many years ago, Herot and Negroponte used an experimental pressure-sensitive stylus : High pressure and a slow drawing speed implied that the user was drawing a line with deliberation, in which case the line was recorded exactly as drawn; low pressure and fast speed implied that the line was being drawn quickly, in which case a straight line connecting the endpoints was recorded. Some commercially available tablets sense not only stylus pressure but orientation as well. The resulting 5 degrees of freedom reported by the tablet can be used in various creative ways. For example, Bleser, Sibert, and McGee implemented the GWPaint system to simulate various artist’s tools, such as an italic pen, that are sensitive to pressure and orientation. An experimental touch tablet, developed by Buxton and colleagues, can sense multiple finger positions simultaneously, and can also sense the area covered at each point of contact The device is essentially a type of touch panel, but is used as a tablet on the work surface, not as a touch panel mounted over the screen. The device can be used in a rich variety of ways . Different finger pressures correlate with the area covered at a point of contact, and are used to signal user commands: a light pressure causes a cursor to appear and to track finger movement; increased pressure is used, like a button-push on a mouse or puck, to begin feedback such as dragging of an object; decreased pressure causes the dragging to stop.

6.4 Interactive Graphical Techniques

There are several techniques that are incorporated into graphics packages to aid the interactive construction of pictures. Various input options can be provided, so that coordinate information entered with locator and stroke devices can be adjusted or interpreted according to a selected option. For example, we can restrict all lines to be either horizontal or vertical. Input coordinates can establish the position or boundaries for objects to be drawn, or they can be used to rearrange previously displayed objects.

6.4.1 Basic Positioning Methods

Coordinate values supplied by locator input are often used with positioning methods to specify a location for displaying an object or a character string. We interactively select coordinate positions with a pointing device, usually by positioning the screen cursor. Just how the object or text-string positioning is performed depends on the selected options. With a text string, for example, the screen point could be taken as the center string position, or the start or end position of the string, or any of the other string-positioning options. For lines, straight line segments can be displayed between two selected screen positions:

As an aid in positioning objects, numeric values for selected positions can be echoed on the screen. Using the echoed coordinate values as a guide, we can make adjustments in the selected location to obtain accurate positioning.

6.4.2 Constraints

With some applications, certain types of prescribed orientations or object alignments are useful. A constraint is a rule for altering input-coordinate values to produce a specified orientation or alignment of the displayed coordinates. There are many kinds of constraint functions that can be specified, but the most common constraint is a horizontal and vertical alignment of straight lines. This type of constraint, shown in Figs. 6.1 and 6.2, is useful in forming network layouts. With this constraint, we can create horizontal and vertical lines without worrying a-bout precise specification of endpoint coordinates.

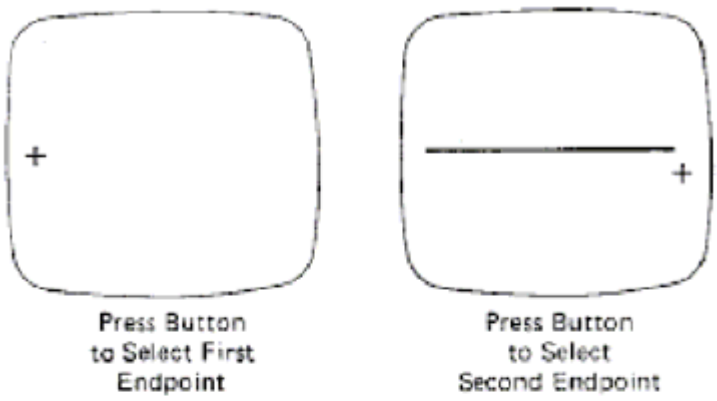


Figure 6.1: Horizontal line constraint

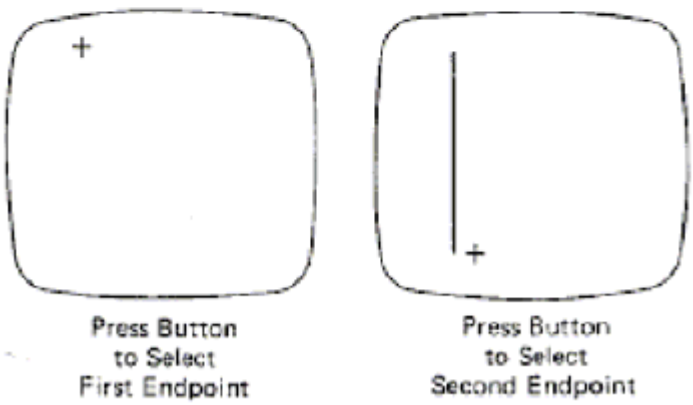


Figure 6.2: Vertical line constraint

A horizontal or vertical constraint is implemented by determining whether any two input coordinate endpoints are more nearly horizontal or more near vertical. If the difference in the y values of the two endpoints is smaller than the difference in x values, a horizontal line is displayed. Otherwise, a vertical line is drawn. Other kinds of constraints can be applied to input coordinates to produce a variety of alignments. Lines could be constrained to have a particular slant, such as 45°, and input coordinates could be constrained to lie along predefined paths, such as circular arcs.

6.4.3 Grids

Another kind of constraint is a grid of rectangular lines displayed in some part of the screen area. When a grid is used, any input coordinate position is rounded to the nearest intersection of two grid lines. Figure 6.3 illustrates line drawing with grid. Each of the two cursor positions is shifted to the nearest grid intersection point, and the line is drawn between these grid points. Grids facilitate object constructions,

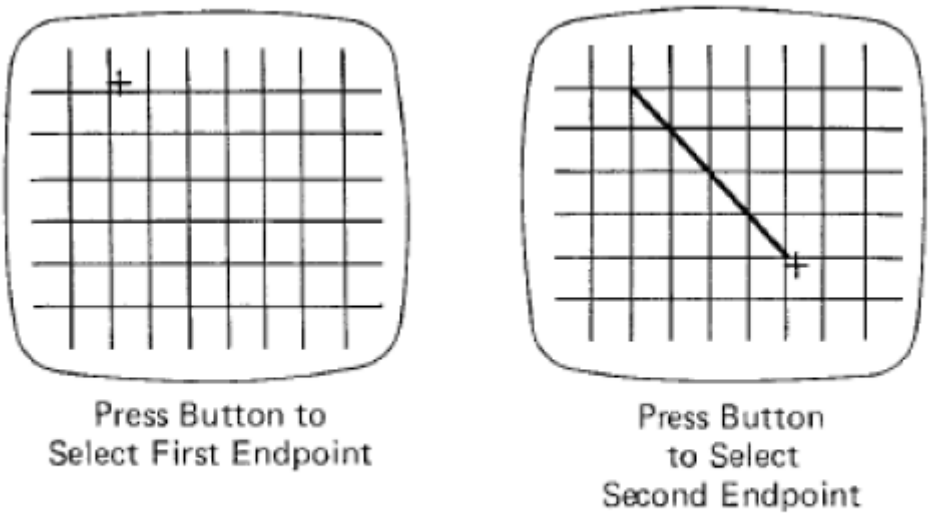


Figure 6.3: Line drawing using a grid

because a new line can be joined easily to a previously drawn line by selecting any position near the endpoint grid intersection of one end of the displayed line.

Figure 6.3: Line drawing using a grid

Spacing between grid lines is often an option that can be set by the user. Similarly, grids can be turned on and off, and it is sometimes possible to use partial grids and grids of different sizes in different screen areas.

6.4.4 Gravity Field

In the construction of figures, we sometimes need to connect lines at positions between endpoints. Since exact positioning of the screen cursor at the connecting point can be difficult, graphics packages can be designed to convert any input position near a line to a position on the line.

This conversion of input position is accomplished by creating a gravity field area around the line. Any selected position within the gravity field of a line is moved ("gravitated") to the nearest position on the line. A gravity field area around a line is illustrated with the shaded boundary shown in Fig. 6.4. Areas around the endpoints are enlarged to make it easier for us to connect lines at their endpoints. Selected positions in one of the circular areas of the gravity field are attracted to the endpoint in that area. The size of gravity fields is chosen large enough to aid positioning, but small enough to reduce chances of overlap with other lines. If many lines are displayed, gravity areas can overlap, and it may be difficult to specify points correctly. Normally, the boundary for the gravity field is not displayed.

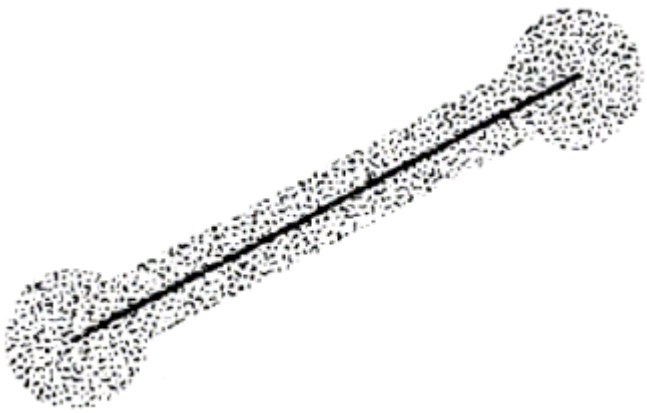


Figure 6.4: Gravity field around a line. Any selected point in the shaded area is shifted to a position on the line

6.4.5 Rubber-Band Methods

Straight lines can be constructed and positioned using rubber-band method which stretch out a line from a starting position as the screen cursor is move Figure 6.5 demonstrates the rubber-band method. We first select a screen position for one endpoint of the line. Then, as the cursor moves around, the line displayed from the start position to the current position of the cursor. When we finally select a second screen position, the other line endpoint is set.

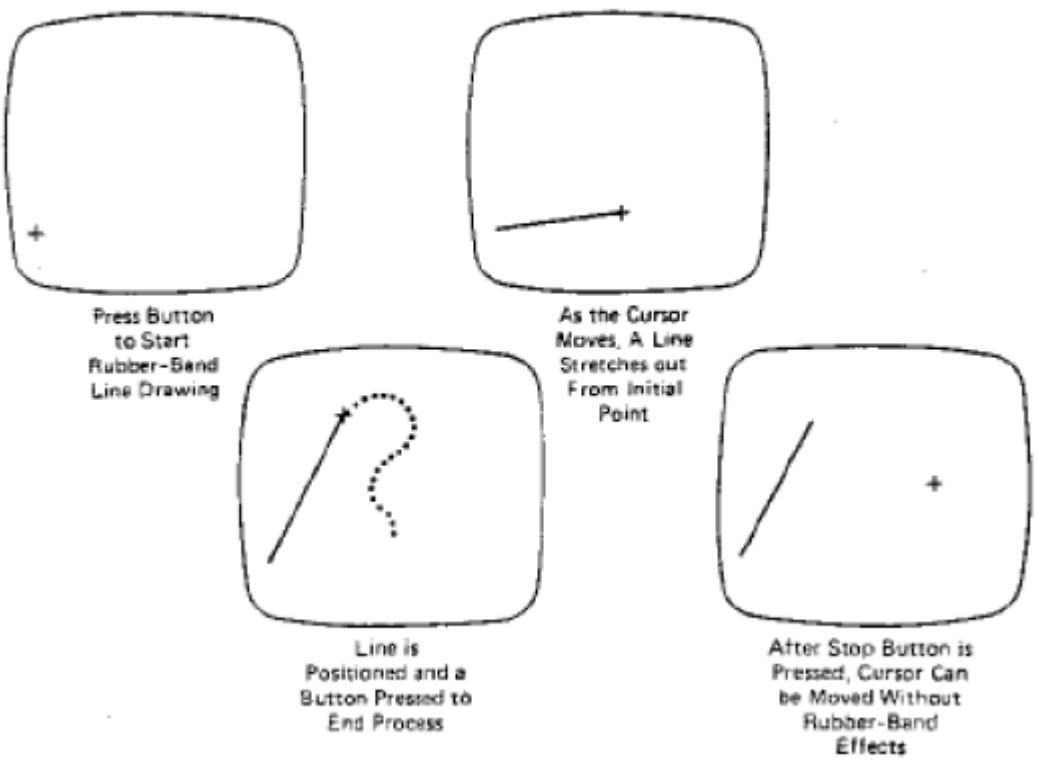


Figure 6.5: Rubber-band method for drawing and positioning a straight line segment

Rubber-band methods are used to construct and position other objects besides straight lines. Figure 6.6 demonstrates rubber-band construction of a rectangle, and Fig. 6.7 shows a rubber-band circle construction.

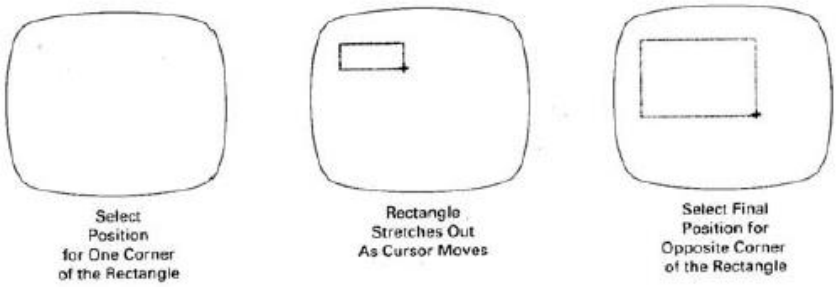


Figure 6.6: Rubber-band method for constructing a rectangle

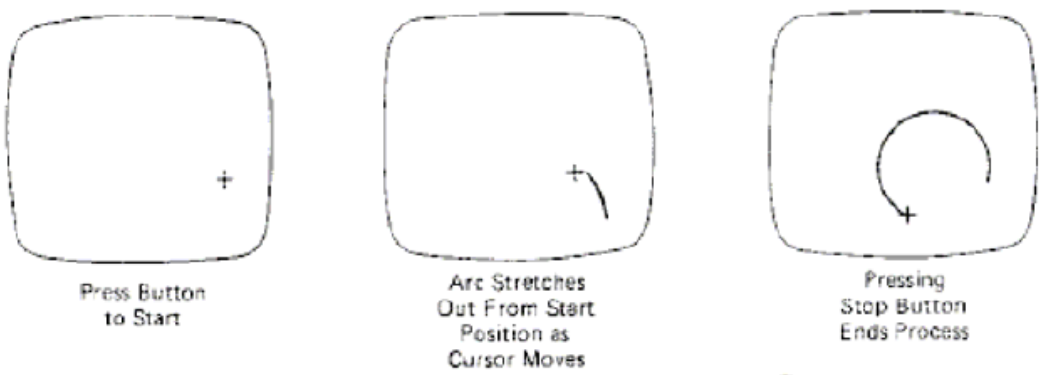


Figure 6.7: Constructing a circle using a rubber-band method

6.4.6 Sketching

Options for sketching, drawing, and painting come in a variety of forms. Straight lines, polygons, and circles can be generated with methods discussed in the previous sections. Curve-drawing options can be provided using standard curve shapes, such as circular arcs and splines, or with freehand sketching procedures. Splines are interactively constructed by specifying a set of discrete screen points that give the general shape of the curve. Then the system fits the set of points with a polynomial curve. In freehand drawing, curves are generated by following the path of a stylus on a graphics tablet or the path of the screen cursor on a video monitor. Once a curve is displayed, the designer can alter the curve shape by adjusting the positions of selected points along the curve path.

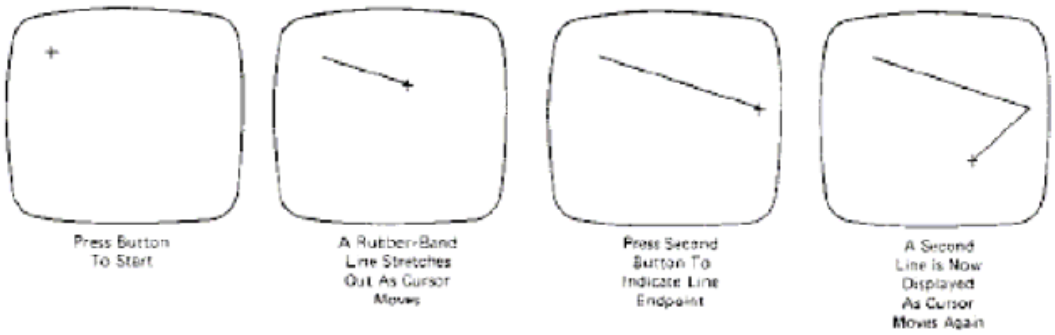


Figure 6.8 Uses rubber band methods to create objects consisting of connected line segments

Line widths, line styles, and other attribute options are also commonly found in painting and drawing packages. Various brush styles, brush patterns, color combinations, object shapes, and surface-texture patterns are also available on many systems, particularly those designed as artist's workstations. Some paint systems vary the line width and brush strokes according to the pressure of the artist's hand on the stylus.

6.4.7 Dragging

A technique that is often used in interactive picture construction is to move objects into position by dragging them with the screen cursor. We first select an object, then move the cursor in the direction we want the object to move, and the selected object follows the cursor path. Dragging objects to various positions in scene is useful in applications where we might want to explore different possibilities before selecting a final location.

6.4.8 Inking and Painting

If we sample the position of a graphical input device at regular intervals and display a dot at each sampled position, a trail will be displayed of the movement of the device. This technique, which closely simulates the effect of drawing on paper, is called inking. For many years the main use of inking has been in conjunction with on-line character-recognition programs. With the advent of high-quality raster displays the technique has found wider use for painting purposes.

6.4.9 Painting

A raster display incorporating a random-access frame buffer, can be treated as a painting surface for interactive purposes. As the user moves the cursor around, a trace of its path can be left on the screen. The user can build up freehand drawings of surprisingly good quality.

It is possible to provide a range of tools for painting on a raster display: these tools take the form of brushes that lay down trails of different thick nesses and colors. For example, instead of depositing a single dot at each sampled input position, the program can insert a group of dots so as to fill in a square or circle: the result will be a much thicker trace. On a "black-and-white display the user needs brushes that paint in both black and white, so that information can be both added and removed (Figure 6.9). When a color display is used for painting, a menu of different colors can be provided.

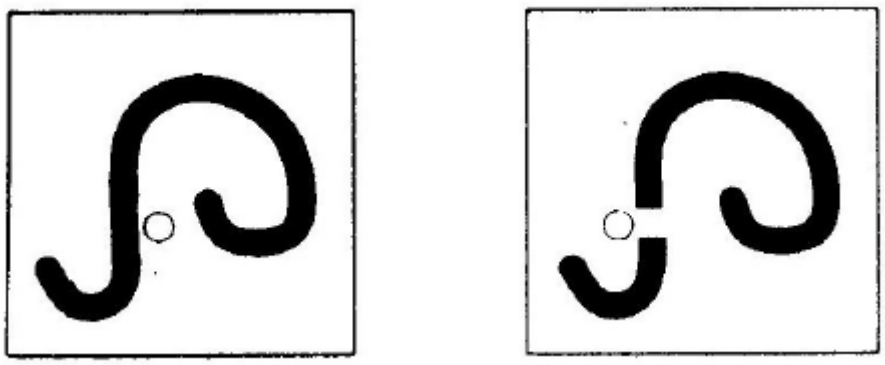


Figure 6.9: Erasing with a white brush

6.5 Summary

• An interaction technique is a way of using a physical input/output device to perform a generic interaction task in a human-computer dialogue. It represents an abstraction of some common class of interactive task, for example, choosing one of several objects shown on a display screen, so it is not bound to a single application.

• The basic interaction tasks for interactive graphics are positioning, selecting, entering text, and entering numeric quantities.

• Input functions available in a graphics package can be defined in three input modes. Request mode places input under the control of the application program. Sample mode allows the input devices and program to operate concurrently. Event mode allows input devices to initiate data entry and control processing of data. Once a mode has been chosen for a logical device class and the particular physical device to be used to enter this class of data, input functions the program are used to enter data values into the program. An application program can make simultaneous use of several physical input devices operating in different modes.

• Interactive picture-construction methods are commonly used in a variety applications, including design and painting packages. These methods provide users with the capability to position objects, to constrain figures to predefine orientations or alignments, to sketch figures, and to drag objects around the screen. Grids, gravity fields, and rubber-band methods are used to aid in positioning and other picture-construction operations.

INPUT AND OUTPUT HANDLING IN WINDOWS SYSTEM :

Input Handling:

The GS is also responsible for handling input from the user, since it sits between the application and the devices.
Again, think of this as a mapping/transformation - we're taking the physical input and mapping it to logical devices (remember that?) so that applications can make sense of it.
How does the GS know what to do with an input? Applications usually have to explicitly express interest in something (e.g. a right mouse button click) - then the GS will pass it on to that app (assuming that other conditions are met - such as window was in focus, etc.)
This leads to a model of programming called event-driven programming (anyone have any experience with this?) -different from normal sequential program -structure: init, run event loop, quit -the GS basically renders the scene, waits for an event, passes it on, and re-renders the scene Any problems with this setup (e.g. for animation/VR - can't wait on events)

Output Handling:

Once the application has specified primitives and attributes, the GS is responsible for realizing those primitives in terms of output on the screen.
Again, you can think of this as a mapping or transformation - from the more abstract primitive descriptions to actual pixel values. Called rendering of primitives
The idea of different spaces also comes up here - we're mapping from model/world space into screen space (different coordinate systems!)
A final way to think of this job is providing the user a certain view into the model (a window on the internal world of the application) - that is often the point of computer graphics - visualization of something that otherwise is only present in bits.

Window management:

Window managers/window systems are usually separate entities from the graphics package.
Their job is to manage the available screen space and mediate this space between multiple applications
This is where logical output devices come in - each application only sees its canvas(es) and doesn't need to worry about everyone else.
The window manager takes care of saving portions of windows that get covered up, dividing the space among windows, deciding the size and position of windows, etc. (GS-Graphics System)