

UNIT I

Introduction

Computer graphics concerns the pictorial synthesis of real or imaginary objects from their computer based models. Computer graphics comprises the creation and representation of simple graphical elements and images, as well as modern techniques for rendering a virtual reality.

- Computer graphics generally means creation, storage and manipulation of models and images.

- Such models come from diverse and expanding set of fields including physical, mathematical, artistic, biological, and even conceptual (abstract) structures
- William Fetter coined term “computer graphics” in 1960 to describe new design methods he was pursuing at Boeing. He created a series of widely reproduced images on pen plotter exploring cockpit design, using 3D model of human body.
- Refers to any computer device or program that makes a computer capable of displaying and manipulating pictures.
- Allows communication through pictures, charts and diagrams.
- Computer Graphics is a study of technique to improve communication between human and machine.

Application areas of Computer Graphics

Computer Graphics is used in diverse areas as advertising, entertainment, medicine, education, science, engineering, navigation, etc.

Some computer applications of Computer Graphics are:

Computer Aided Design (CAD)

1. Used to design the buildings, automobiles, aircraft, watercraft, spacecraft, computers, textiles and many other products.
2. CG is a useful tool for generating the architects, drawing and visualizing structures
3. A computer takes the data about building and makes various images of the building from different angles
4. Animations are used in CAD applications
5. Real time animations using wire frame displays on a video monitor are useful for testing performance of vehicles and also to see the interior of the vehicle and to watch the behavior of inner components during motion.

Presentation Graphics

1. Used to produce illustrations for reports
2. Commonly used to summarize financial, statistical, mathematical, scientific and economic data for research reports, managerial reports, consumer information bulletin and other reports.
3. Typical examples are bar charts, line graphs, surface graphs, pie charts and other display showing relationship between multiple parameters.
4. Time charts and task networks layouts are used in project management to schedule and monitor the progress of projects.

Computer Art

1. CG is used in both fine arts and commercial arts applications
2. Artists use various computer methods such as special-purpose hardware, artist’s paintbrush program(Lumena), other paint packages(Pixelpaint, SuperPaint), symbolic mathematics packages(Mathematica), CAD packages, desktop publishing software’s, animation packages that provide facilities for designing object shapes and specifying object motions.
3. Fine artists use other computer technologies to produce images. He uses a combination of three-dimensional modeling packages, texture mapping, drawing programs and CAD software.
4. These methods are also applied in commercial art for logos and other designs, page layouts combining text and graphics, TV advertising spots and other areas.
5. A common graphics method employed in many commercials is morphing, where one object is transformed into another. This method is used in TV commercials to turn oil can into automobile engine, etc.

Entertainment

1. Computer Graphics methods are commonly used in making motion pictures, music videos, television shows and cartoon animation films.
2. Many TV series regularly employ computer graphics methods.
3. Music videos use graphics in several ways. Graphics objects can be combined with live action, or graphics and image processing techniques can be used to produce a transformation of one person or object into another(morphing)

Education and Training

1. Computer generated models like physical systems, financial systems and economic systems used as education aids
2. Models of physical system, physiological system, population trends or equipment, can help trainees to understand the operation of the system.
3. Examples of some specialized systems are the simulators for practice sessions or training of ship captains, aircraft pilots, heavy-equipment operators, and air traffic-control personnel.
4. Various educational pictures with animations are used to present better understanding for learning with animations

Image processing

1. Image processing applies techniques to modify or interpret existing pictures, such as photographs and TV scans.
2. Two principle applications
 - (1) Improving picture quality
 - (2) Machine perception of visual information used in robotics
3. To digitize the shading and color, interesting, sharpen, improve the contrasting scanning image and to transfer them to monitor or screen or visual display unit.
4. These techniques are used in commercial art applications and to analyze satellite photos of the earth and photos of galaxies.
5. Medical applications also make extensive use of image-processing techniques
 - (1) Tomography is a technique of X-ray photograph that allows cross-sectional views of physiological system to be displayed.
 - (2) Computerized Axial Tomography (CAT) is used to compose the 3D model of the brain by taking x-ray of it, which can be used to detect problems like brain tumor etc
 - (3) Ultrasonic are used to generate digital data.
 - (4) Nuclear medicine scanners collect digital data from radiation emitted from ingested radionuclide and plot color-coded images.
 - (5) Computer-aided surgery

Graphical User Interface

1. A major component of a graphical interface is a window manager that allows a user a display multiple-window area.
2. Each window can contain a different process that can contain graphical or nongraphical displays.
3. Interfaces also display menus and icons for selection of processing options or parameter values.
 - (1) An icon is a graphical symbol that is designed to look like the processing option it represents.
 - (2) Menus contain lists of textual descriptions and icons.

The challenge to computer graphics is to make that virtual world look real, sound real, move and respond to interaction in real time, and even feels real.

Overview of Graphics Systems

- Computer graphics is a complex and diversified technology. This can be achieved by considering the end product of Computer Graphics i.e., a picture.
- The picture may be used for a large variety of purposes such as engineering drawing.

- The picture is a fundamental cohesive concept in Computer Graphics. Therefore the following are considered.
 - (1) Representing pictures in Computer Graphics
 - (2) Preparing pictures for presentation
 - (3) Presenting previously prepared pictures
 - (4) Interacting with pictures
- Here the picture means any collection of lines, points, text, etc displayed on a graphics device.

Representing picture

- Although many algorithms accept pictures data as polygons or edges, each polygon can in turn be represented by points.
- Points are fundamental building blocks of picture representation.
- For example, the unit square can be represented by its four corner points.

$P_1 (0, 0)$

$P_2 (1, 0)$

$P_3 (1, 1)$

$P_4 (0, 1)$

- An associated algorithm description is as
Connect $P_1P_2P_3P_4P_1$

- The unit square can also be described by its four edges

$E_1 \equiv P_1P_2$

$E_2 \equiv P_2P_3$

$E_3 \equiv P_3P_4$

$E_4 \equiv P_4P_1$

- Here the algorithm description is as
Connect $E_1E_2E_3E_4$ in sequence

The fundamental building blocks i.e., points can be represented as either pairs or triplets of numbers, depending on whether the data are two-or-three dimensional space.

- Thus (x_1, y_1) or (x_1, y_1, z_1) represent a point in either two-or three dimensional space.
- Two points represent a line or edge and a collection of three or more points a polygon.
- The representation of curved lines is usually taken by approximating them by connected short straight line segments.
- The representation of textual material is quite complex, involving in many cases curved lines or dot matrices.
- Fundamentally, textual material is again represented by collection of lines and points and an organizing algorithm.

Preparing Pictures

- Pictures ultimately consist of points and a drawing algorithm to display them.
- This information is generally stored in a file before it is used to present the picture; this file is called a database.
- Very complex pictures require very complex databases, which require a complex algorithm to access them.

- These complex databases contain data organized in various ways. E.g.: ring structures, B-tree structure, quad tree structures, etc.
- The database itself contains pointers, substructures and other nongraphic data.
- Many computer graphics application involve much simpler pictures, for which the user can readily invert simple data structure which can be easily accessed, such as linear list. There are quite adequate for many reasonably complex pictures.
- The three fundamental operations for manipulating these points are:
 - Move the beam, pen, cursor, plotting head invisibly to the point.
 - Draw a visible line to a point from an initial point.
 - Display a dot at that point.

Hence there are two ways to specify the position of a point: absolute coordinates or relative (incremental) coordinates. In relative or incremental coordinates, the position of a point is defined by giving the displacement of the point with respect to the previous point. All computer graphics software is based on these fundamental operations.

Presenting previous Prepared Pictures

- The data used to present the picture is frequently called a displaying file.
- The display file represents some position, view or scene of the picture represented by the total database.
- The displayed picture is usually formed by rotating, translating, scaling and performing various projections on data.
- These basic orientation or viewing preparations are generally performed using a 4x4 transformation matrix, operating on the data represented in homogenous coordinates.
- Hidden line or hidden surface removal, shading, transparency, texture or color effects may be added before final presentation of the picture.
- If the picture represented by the entire database is not required to be presented, the appropriate portion must be selected. This is called clipping, which may be two or three dimensional.
- Two important concepts associated with presenting a picture are windows and viewports.
 - Windowing is the process of extracting a portion of a database by clipping the database to the boundaries of the window.
 - A viewport is an area of the display device on which the window data is presented.

Interacting with Pictures

- Interacting with pictures permits higher user-computer interaction. This significantly enhances the ability to understand data, to perceive trends and to visualize real or imaginary objects.

Video-Display devices

1. Cathode Ray Tubes
2. Raster scan Display
3. Vector scan/Random scan Display
4. Color CRT monitors
5. Direct View Storage Tubes
6. Flat panel Display
7. Three-Dimensional Viewing Devices

Cathode Ray Tube

Typically the primary output device in a graphics system is a video monitor. The operation of most video monitors is based on the standard Cathode-ray-tube design. Basic operation is as shown on Cathode Ray Tubes

- A beam of electrons emitted by an electron gun, pass through focusing and deflection systems that direct the beam towards specified positions on the phosphor coated screen

- The phosphor then emits a small spot of light at each position contacted by the electron beam
- The light emitted by phosphorous fades very rapidly to keep glowing is done by redraw the picture repeatedly by quickly directing the electron beam back over the same points is called as “refresh CRT”
- Heated metal cathode and Control grid are the main components of electron gun (figure 1)

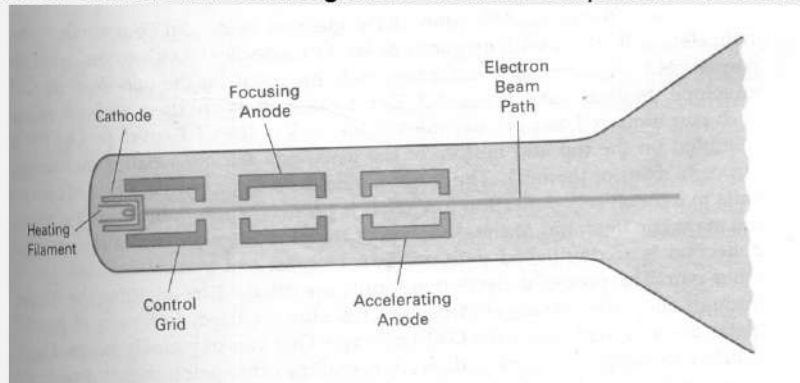


Figure 1: Operation of an electron gun with an accelerating anode.

- The heat is supplied to the cathode through current passing in coil of wire called filament
- This makes electrons to be “boiled off” the hot cathode surface
- The free negatively charged electrons inside CRT are accelerated towards the Phosphorous coated by high positive voltage generated by positively charged metal coating on the inside of CRT
- Intensity of electron beam is controlled by setting voltage levels in the control grid, which is fit over the cathode
- A high negative voltage applied to the control grid will shut off the beam by repelling electrons and stopping them from passing through the small hole at the end of the control grid structure. A smaller negative voltage on the control grid simply decreases the number of electrons passing through.
- Since the amount of light emitted by the phosphor coating depends on the number of electrons striking the screen, the brightness of a display is controlled by varying the voltage on the control grid. The intensity level is specified for individual screen positions with graphics software commands.
- A beam of electrons emitted by an electron gun, pass through focusing and deflection systems that direct the beam towards specified positions on the phosphor-coated screen. The phosphor then emits a small spot of light at each position contacted by the electron beam. The light emitted by phosphorous fades very rapidly to keep glowing is done by redraw the picture repeatedly by quickly directing the electron beam back over the same points is called as “refresh CRT”.
- The focusing system in a CRT is needed to force the electron beam to converge into a small spot as it strikes the phosphor. Otherwise, the electrons would repel each other, and the beam would spread out as it approaches the screen. Focusing is performed with either electric or magnetic fields.

Focusing	Meaning
Electro static focusing	With electro static focusing, the beam pass through positively charged metal cylinder that forms electrostatic and it focus the beam at the centre of the screen. (as shown in figure 2)
Magnetic focusing	With magnetic field set up by a coil mounted around the outside of the CRT envelope. (as shown in figure 3)

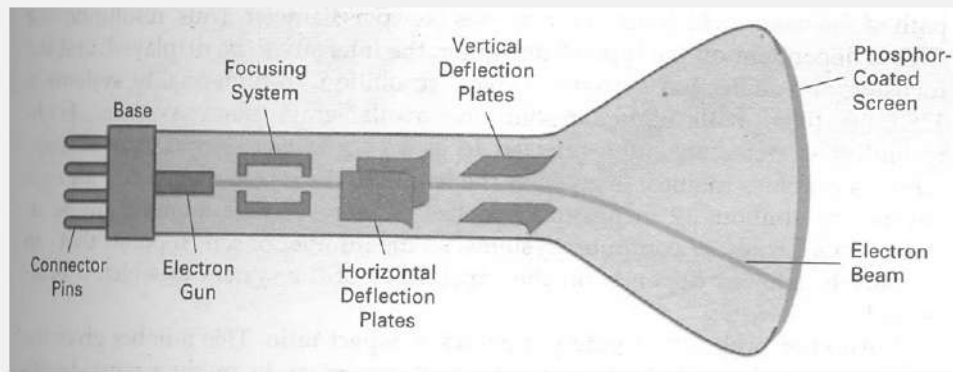


Figure 2: Electrostatic deflection of the electron beam in a CRT.

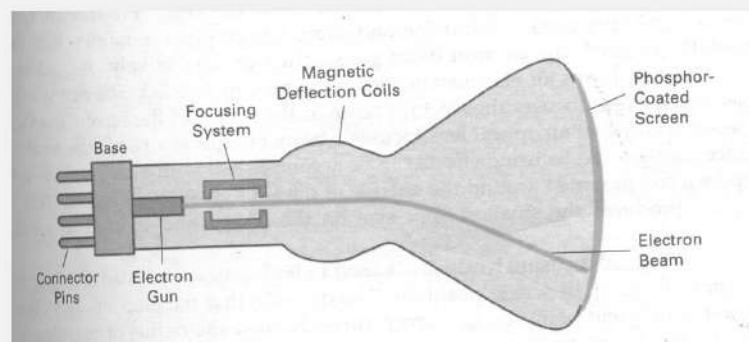


Figure 3: Basic design of a magnetic deflection CRT

Different kinds of phosphors are available for use in a CRT. The properties with which the phosphors vary are:

Term	Description
Persistence	Persistence is defined as the time it takes the emitted light from the screen to decay to one tenth of its original intensity.
Resolution	The maximum number of points displayed on the CRT screen or the number of points/centimeter that can be plotted horizontally and vertically.
Aspect ratio	The ratio of vertical points to horizontal points necessary to produce equal length line in both directions on the screen.

Raster-Scan Display

- The most common type of graphics monitor employing a CRT is the Raster-scan displays, based on television technology
- JPG images are raster based
- Light occurs when an electron beam stimulates a phosphor.

- In Raster scan, the electron beam from electron gun is swept horizontally across the phosphor one row at a time from top to bottom.(figure 4)

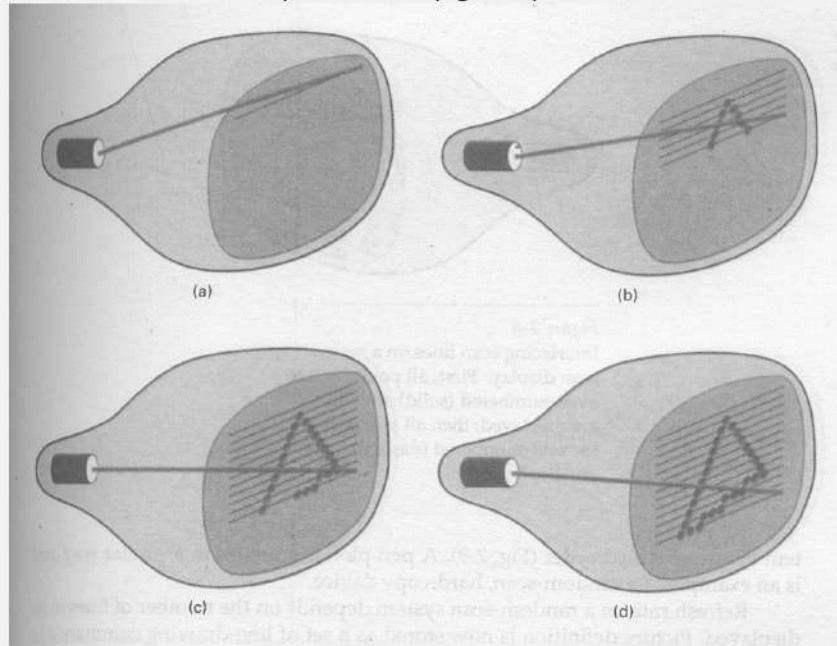


Figure 4: A raster-scan system displays an object as a set of discrete points across
Each scan line.

- After each horizontal sweep the beam is moved.
- After the bottom line is swept, the beam returns to the top and the sweep process begins again.
- As the electron beam moves across each row, the beam intensity is turned on and off to create a pattern of illuminated spots
- Picture definition is stored in a memory area called the refresh buffer or frame buffer
- Each screen point is called as "pixel"
- This memory area holds the set of intensity values for all the screen points
- This is part of the system memory
- The stored intensity values are then retrieved from frame buffer and painted on the screen one row at a time
- Intensity range for pixel position depends on capability of the raster system
- In black and white system, the point on screen is either on or off
- Only one bit is needed to control the intensity of the screen
- In case of color systems, 2 bits are required
- One to represent ON (1), another one is OFF (0).
- Refreshing on raster scan is carried out at the rate of 60 to 80 frames per seconds

Term	Description
Bit map	On a black and white system with one bit per pixel, the frame buffer is called bit map.
Pix map	The frame buffer with multiple bits per pixel (for color display). Refreshing of raster scan displays done by rate of 60 to 80 frames per second. The units used sometimes are Hertz or cycles/second.
Horizontal retrace	Horizontal retrace of the electron beam means the return to the left of the screen. After refreshing each Sean line.
Vertical retrace	If it returns to the top left corner of the screen to begin in the next frame called "vertical retrace".

- On some raster-scan systems (and in TV sets), each frame is displayed in two passes using an interlaced refresh procedure. In the first pass, the beam sweeps across every other scan line from top to bottom. Then after the vertical retrace, the beam sweeps out the remaining scan lines (Figure 5).

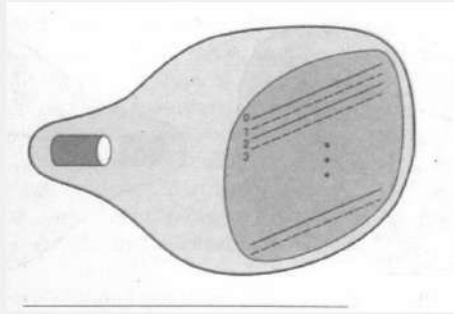


Figure 5: Interlacing scan lines on a raster scan display. First, all points on the even-numbered (solid) scan lines are displayed; and then all points along the odd-numbered (dashed) lines are displayed.

- Interlacing of the scan lines in this way allows us to see the entire screen displayed in one-half the time it would have taken to sweep across all the lines at once from top to bottom.
- Interlacing is primarily used with slower refreshing rates. On an older, 30 frame per-second, noninterlaced display, for instance, some flicker is noticeable. But with interlacing, each of the two passes can be accomplished in 1/60th of a second, which brings the refresh rate nearer to 60 frames per second. This is an effective technique for avoiding flicker, providing that adjacent scan lines contain similar display information.
- Frame buffer size = to store bits per pixel* resolution

Advantages

- || High degree realism is achieved in picture with the aid of advanced shading and hidden surface technique.
- || Decreasing memory costs have made raster systems popular.
- || Computer monitors and TVs use this method

Disadvantages

- || Raster displays have less resolution.
- || The lines produced are ziz-zag as the plotted values are discrete.

Random-Scan Display

In a Random scan system, also called vector, stroke writing, or calligraphic the electron beam directly draws the picture.

- A pen plotter operates in a similar way and is an example of a random-scan, hard-copy device.
- When operated as a random-scan display unit, a CRT has the electron beam directed only to the parts of the screen where a picture is to be drawn.
- Random scan monitors draw a picture one line at a time and for this reason are also referred to as vector displays (or stroke-writing or calligraphic displays).
- Here the electron gun of a CRT illuminates points and / or straight lines in any order.
- Refresh rate on a random-scan system depends on the number of lines to be displayed.
- Picture definition stored as a set of line drawing commands in an area of memory called "refresh display file" or also called as display list or display program or refresh buffer
- This displays to draw all the component lines of picture 30 to 60 frames/second
- This system is designed for line drawing applications
- Vector displays produces smooth line drawings but raster produces jagged lines that are plotted points

- To display a given picture, the system cycles through the set of commands in the display file, drawing each component line in turn
- After all line drawing commands have been processed, the system cycles back to the first line drawing command in the list
- Random scan suitable for applications like engineering and scientific drawings
- Graphics patterns are displayed by directing the electron beam along the component lines of the picture
- A scene is then drawn one line at a time by positioning the beam to fill in the line between specified end points

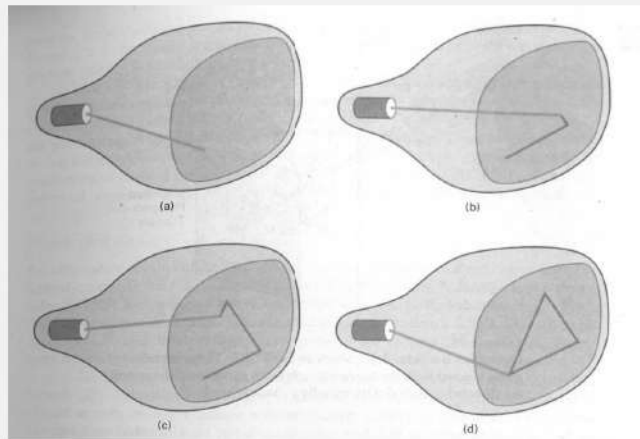


Figure 6: A random-scan system draws the component lines of an object in any order specified.

Advantages

- || Very high resolution, limited only by monitor.
- || Easy animation, just draw at different position.
- || Requires little memory (just enough to hold the display program).

Disadvantages

- || Requires intelligent electron beam, i.e., processor controlled.
- || Limited screen density before have flicker, can't draw a complex image.
- || Limited color capability (very expensive).

Color CRT Monitors

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

Beam-penetration method

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

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

Shadow-mask methods

- Shadow-mask methods are commonly used in raster scan systems (including color TV) because they produce a much wider range of colors than the beam penetration method.
- A shadow-mask CRT has three phosphor color dots at each pixel position.
- One phosphor dot emits a red light, another emits a green light, and the third emits a blue light.
- This type of CRT has three electron guns, one for each color dot, and a shadow-mask grid just behind the phosphor-coated screen.

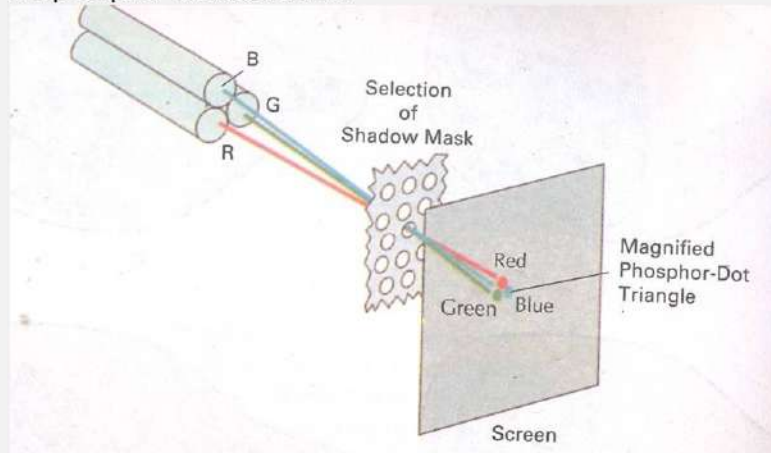


Figure 7: Operation of a delta-delta, shadow-mask CRT. Three electrons

Guns, aligned with the triangular color dot patterns on the screen,

Are directed to each dot triangle by a shadow mask.

- Figure 7 illustrates the delta-delta shadow-mask method, commonly used in color CRT systems.
 - The three electron beams are deflected and focused as a group onto the shadow mask, which contains a series of holes aligned with the phosphor-dot patterns.
 - When the three beams pass through a hole in the shadow mask, they activate a dot triangle, which appears as a small color spot on the screen.
 - The phosphor dots in the triangles are arranged so that each electron beam can activate only its corresponding color dot when it passes through the shadow mask.
- Another configuration for the three electron guns is an in-line arrangement in which the three electron guns, and the corresponding red-green-blue color dots on the screen, are aligned along one scan line instead of in a triangular pattern.
- This in-line arrangement of electron guns is easier to keep in alignment and is commonly used in high-resolution color CRTs.
- Color variations are obtained 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, and cyan shows up when blue and green are activated equally.

- In some low-cost systems, the electron beam can only be set to on or off, limiting displays to eight colors. More sophisticated systems can set intermediate intensity levels for the electron beams, allowing several million different colors to be generated.

Color graphics systems can be designed to be used with several types of CRT display devices. Some inexpensive home-computer systems and video games are designed for use with a color TV set and an RF (radio-frequency) modulator.

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

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

Direct-View Storage Tubes

- A direct-view Storage Tube (DVST) stores the picture information as a charge distribution just behind the phosphor-coated screen.
- Two electron guns are used in a DVST.
 - One, the primary gun, is used to store the picture pattern;
 - the second, the flood gun, maintains the picture display.
- An alternative method for maintaining a screen image is to store the picture information inside the CRT instead of refreshing the screen.

A DVST monitor has both disadvantages and advantages compared to the refresh CRT.

Advantages

- It has a flat screen.
- Refreshing of screen is not required.
- Because no refreshing is needed, very complex pictures can be displayed at very high resolutions without flicker.

Disadvantages

- This has poor contrast.
- Performance is inferior to the refresh CRT.
- Selective or part erasing of screen is not possible.
- They ordinarily do not display color and that selected parts of a picture cannot be erased. To eliminate a picture section, the entire screen must be erased and the modified picture redrawn. The erasing and redrawing process can take several seconds for a complex picture. For these reasons, storage displays have been largely replaced by raster systems.

Flat-Panel Displays

- The term Flat-panel display refers to a class of video devices that have reduced volume, weight, and power requirements compared to a CRT.
- A significant feature of flat-panel displays is that they are thinner than CRTs, and we can hang them on walls or wear them on our wrists.

- Current uses for flat-panel displays include small TV monitors, calculators, pocket video games, laptop computers, armrest viewing of movies on airlines, as advertisement boards in elevators, and as graphics displays in applications requiring rugged, portable monitors.
- Flat-panel displays are of two categories:
 - Emissive displays e.g. Plasma panel, LEDs etc
 - Nonemissive displays. e.g. LCD

The **emissive displays** (or emitters) are devices that convert electrical energy into light. Plasma panels, thin-film electroluminescent displays, and Light-emitting diodes are examples of emissive displays. Flat CRTs have also been devised, in which electron beams are accelerated parallel to the screen, then deflected 90° to the screen. But flat CRTs have not proved to be as successful as other emissive devices.

Nonemissive displays (or nonemitters) use optical effects to convert sunlight or light from some other source into graphics patterns. The most important example of a Nonemissive flat-panel display is a liquid-crystal device.

- Plasma panels, also called gas-discharge displays, are constructed by filling the region between two glass plates with a mixture of gases that usually includes neon.
- A series of vertical conducting ribbons is placed on one glass panel, and a set of horizontal ribbons is built into the other glass panel (Figure 8).

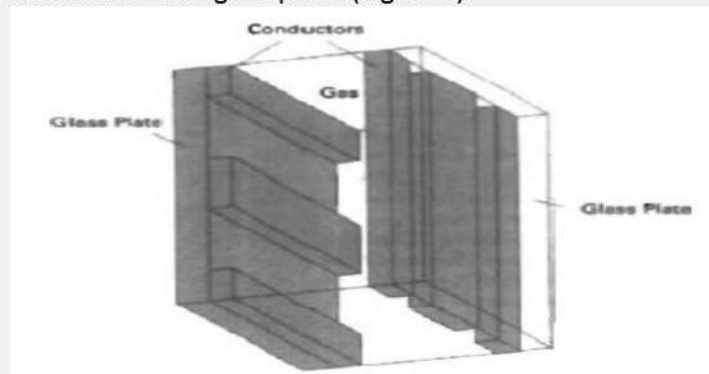


Figure 8: Basic design of a plasma-panel display device.

- Firing voltages applied to a pair of horizontal and vertical conductors cause the gas at the intersection of the two conductors to break down into glowing plasma of electrons and ions.
- Picture definition is stored in a refresh buffer, and the firing voltages are applied to refresh the pixel positions (at the intersections of the conductors) 60 times per second.
- Alternating-current methods are used to provide faster application of the firing voltages, and thus brighter displays.
- Separation between pixels is provided by the electric field of the conductors.
- One disadvantage of plasma panels has been that they were strictly monochromatic devices, but systems have been developed that are now capable of displaying color and grayscale.
- Thin-film electroluminescent displays are similar in construction to a plasma panel.
- The difference is that the region between the glass plates is filled with a phosphor, such as zinc sulfide doped with manganese, instead of a gas (Figure 9).

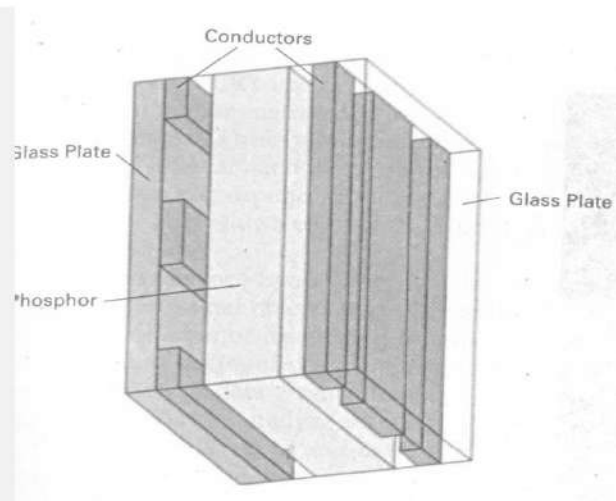


Figure 9: Basic design of a thin-film electroluminescent display device.

- When a sufficiently high voltage is applied to a pair of crossing electrodes, the phosphor becomes a conductor in the area of the intersection of the two electrodes.
- Electrical energy is then absorbed by the manganese atoms, which then release the energy as a spot of light similar to the glowing plasma effect in a plasma panel.
- Electroluminescent displays require more power than plasma panels, and good color and gray scale displays are hard to achieve.
- A third type of emissive device is the light-emitting diode (LED). A matrix of diodes is arranged to form the pixel positions in the display, and picture definition is stored in a refresh buffer.
- As in scan-line refreshing of a CRT, information is read from the refresh buffer and converted to voltage levels that are applied to the diodes to produce the light patterns in the display.
- LCDs are commonly used in small systems, such as calculators and portable, laptop computers.
- These Nonemissive devices produce a picture by passing polarized light from the surroundings or from an internal light source through a liquid-crystal material that can be aligned to either block or transmit the light.

Three-Dimensional Viewing Devices

- Graphics monitors for the display of three-dimensional scenes have been devised using a technique that reflects a CRT image from a vibrating, flexible mirror.
- As the varifocal mirror vibrates, it changes focal length. These vibrations are synchronized with the display of an object on a CRT so that each point on the object is reflected from the mirror into a spatial position corresponding to the distance of that point from a specified viewing position. This allows us to walk around an object or scene and view it from different sides.
- Such systems have been used in medical applications to analyze data from ultrasonography and CAT scan devices, in geological applications to analyze topological and seismic data, in design applications involving solid objects, and in three-dimensional simulations of systems, such as molecules and terrain.
- Another technique for representing three-dimensional objects is displaying stereoscopic views. This method does not produce true three-dimensional images, but it does provide a three-dimensional effect by presenting a different view to each eye of an observer so that scenes do appear to have depth.

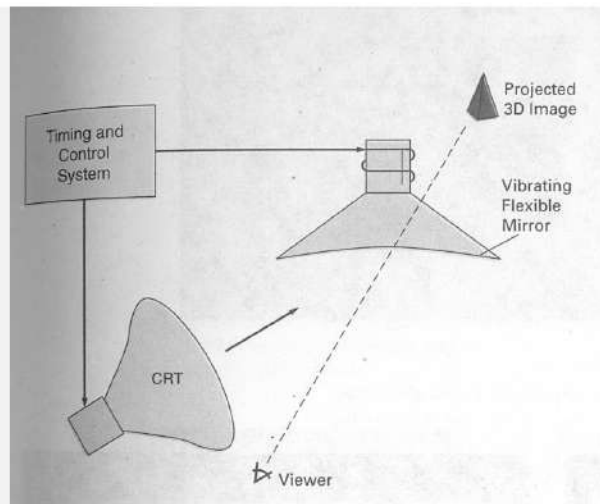


Figure 10: Operation of a three-dimensional display system using a vibrating mirror that changes focal length to match the depth of points in a scene.

- To obtain a stereoscopic projection, we first need to obtain two views of a scene generated from a viewing direction corresponding to each eye (left and right). We can construct the two views as computer-generated scenes with different viewing positions, or we can use a stem camera pair to photograph some object or scene. When we simultaneous look at the left view with the left eye and the right view with the right eye, the two views merge into a single image and we perceive a scene with depth.
- Stereoscopic viewing is also a component in virtual-reality systems, where users can step into a scene and interact with the environment. A headset containing an optical system to generate the stereoscopic views is commonly used in conjunction with interactive input devices to locate and manipulate objects in the scene. A sensing system in the headset keeps track of the viewer's position, so that the front and back of objects can be seen as the viewer "walks through" and interacts with the display.
- An interactive virtual-reality environment can also be viewed with stereoscopic glasses and a video monitor, instead of a headset. This provides a means for obtaining a lower cost virtual-reality system.

Raster-Scan Systems

- Raster scan or raster scanning: is the pattern of image detection and reconstruction in television, and is the pattern of image storage and transmission used in most computer image 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 11

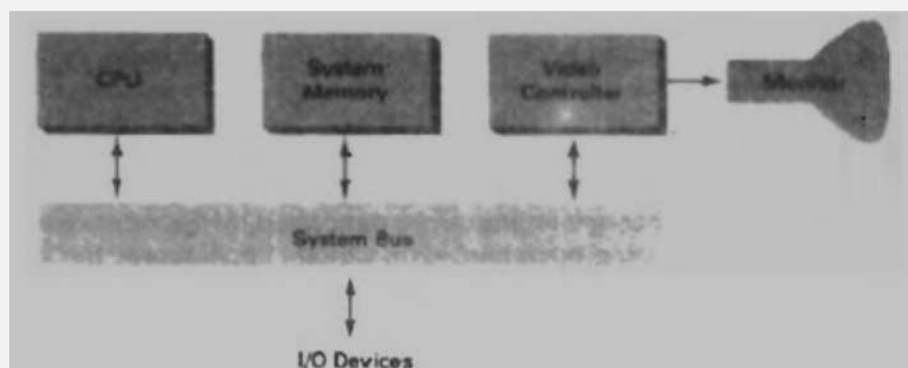


Figure 11: Architecture of a simple raster graphics system.

- Here, the frame buffer can be anywhere in the system memory, and the video controller accesses the frame buffer to refresh the screen.
- In addition to the video controller, more sophisticated raster systems employ other processors as co-processors and accelerators to implement various graphics operations.

Video Controller

Figure 12 shows a commonly used organization for raster systems.

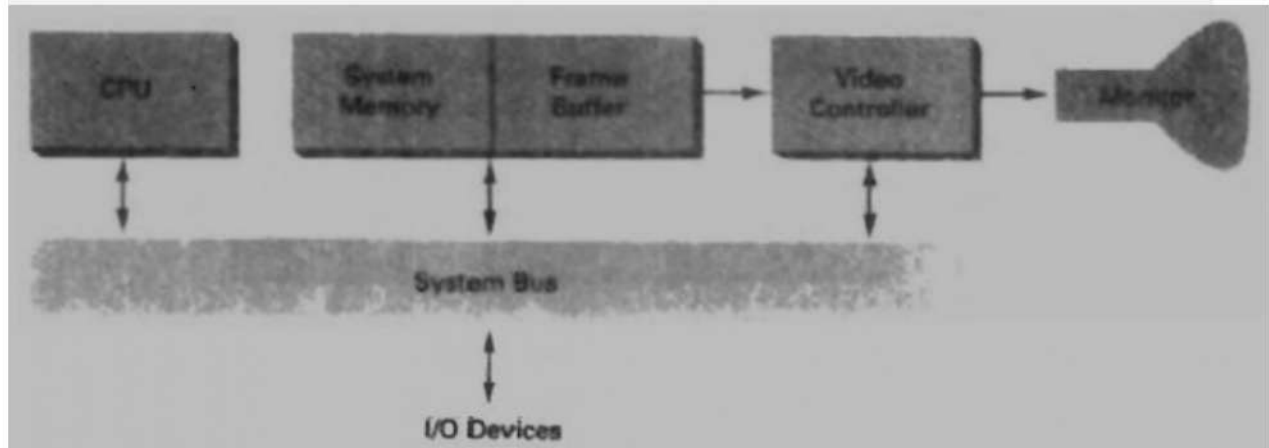


Figure 12: Architecture of a raster system with a fixed portion of the system memory reserved for the frame buffer.

- A fixed area of the system memory is reserved for the frame buffer, and the video controller is given direct access to the frame-buffer memory.
- Frame-buffer locations, and the corresponding screen positions, are referenced in Cartesian coordinates.
- For many graphics monitors, the coordinate origin is defined at the lower left screen corner (Figure13).

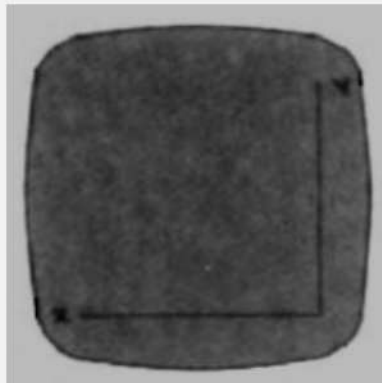


Figure 13: The origin of the coordinate system for identifying screen positions is usually specified in the lower-left corner.

- The screen surface is then represented as the first quadrant of a two-dimensional system, with positive x values increasing to the right and positive y values increasing from bottom to top. (On some personal computers, the coordinate origin is referenced at the upper left corner of the screen, so the y values are inverted.)
- Scan lines are then labeled from y_{max} at the top of the screen to 0 at the bottom. Along each scan line, screen pixel positions are labeled from 0 to x_{max} .
- In Figure 14, the basic refresh operations of the video controller are diagrammed. Two registers are used to store the coordinates of the screen pixels.

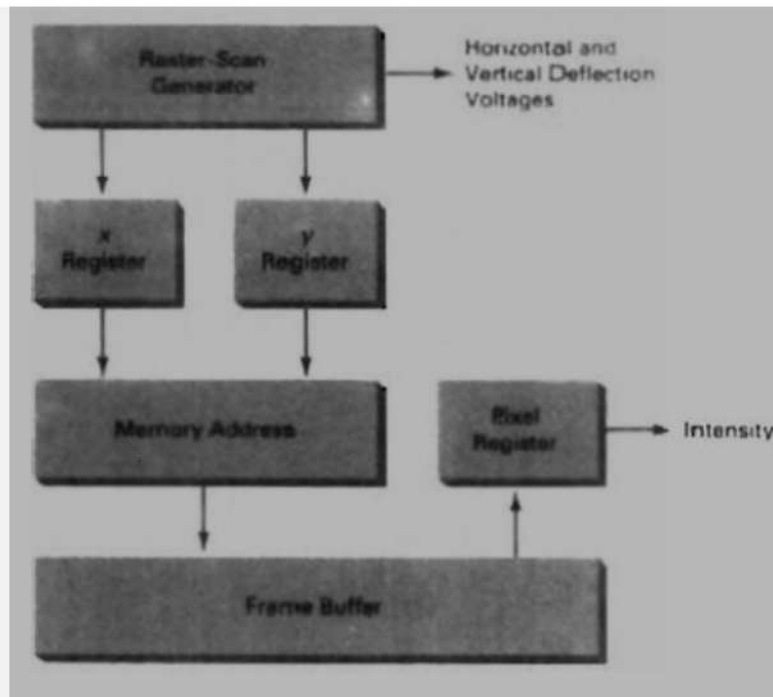


Figure 14: Basic video-controller refresh operations.

- Initially, the x register is set to 0 and the y register is set to y₀.
- The value stored in the frame buffer for this pixel position is then retrieved and used to set the intensity of the CRT beam.
- Then the x register is incremented by 1, and the process repeated for the next pixel on the top scan line.
- This procedure is repeated for each pixel along the scan line.
- After the last pixel on the top scan line has been processed, the x register is reset to 0 and the y register is decremented by 1.
- Pixels along this scan line are then processed in turn, and the procedure is repeated for each successive scan line.
- After cycling through all pixels along the bottom scan line (y = 0), 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 the rate of 60 frames per second, the simple procedure illustrated in above figure cannot be accommodated by typical RAM chips. The cycle time is too slow. To speed up pixel processing, video controllers can retrieve multiple pixel values from the refresh buffer on each pass. The multiple pixel intensities are then stored in a separate register and used to control the CRT beam intensity for a group of adjacent pixels. When that group of pixels has been processed, the next block of pixel values is retrieved from the frame buffer.
- A number of other operations can be performed by the video controller, besides the basic refreshing operations.
- For various applications, the video controller can retrieve pixel intensities from different memory areas on different refresh cycles.
- In high-quality systems, for example, two frame buffers are often provided so that one buffer can be used for refreshing while the other is being filled with intensity values. Then the two buffers can switch roles. This provides a fast mechanism for generating real-time animations, since different views of moving objects can be successively loaded into the refresh buffers.
- Also, some transformations can be accomplished by the video controller. Areas of the screen can be enlarged, reduced, or moved from one location to another during the refresh cycles.
- In addition, the video controller often contains a lookup table, so that pixel values in the frame buffer are used to access the lookup table instead of controlling the CRT beam intensity directly. This provides a fast method for changing screen intensity values.

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

Raster-Scan Display Processor

- Figure 15 shows one way to set up the organization of a raster system containing a separate display processor, sometimes referred to as a graphics controller or a display coprocessor.

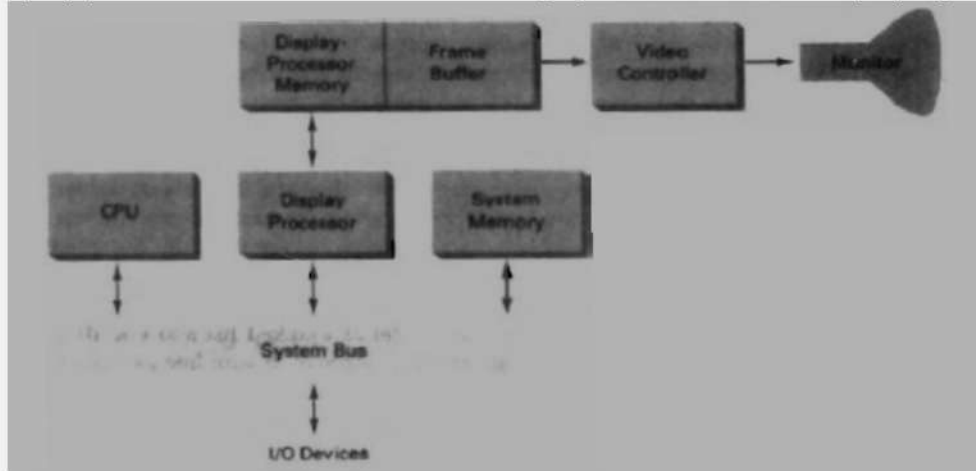


Figure 15: Architecture of a raster-graphics system with a display processor.

- The purpose of the display processor is to free the CPU from the graphics chores.
- In addition to the system memory, a separate display processor memory area can also be provided.
- A major task of the display processor is digitizing a picture definition given in an application program into a set of pixel-intensity values for storage in the frame buffer.
- This digitization process is called scan conversion.
- Graphics commands specifying straight lines and other geometric objects are scan converted into a set of discrete intensity points.
- Scan converting a straight-line segment, for example, means that we have to locate the pixel positions closest to the line path and store the intensity for each position in the frame buffer.
- Similar methods are used for scan converting curved lines and polygon outlines.
- Characters can be defined with rectangular grids, as in Fig. 2-30, or they can be defined with curved outlines, as in Fig. 2-31.

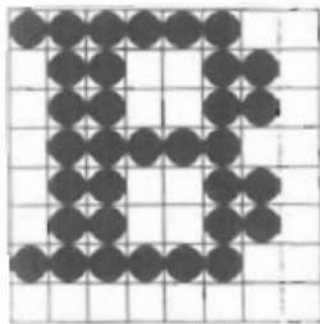


Figure 2-30
A character defined as a rectangular grid of pixel positions.

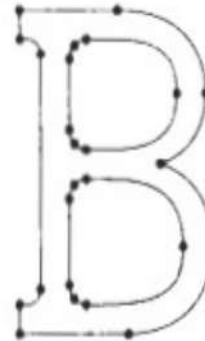


Figure 2-31
A character defined as a curve outline.

- The array size for character grids can vary from about 5 by 7 to 9 by 12 or more for higher quality displays.
- A character grid is displayed by superimposing the rectangular grid pattern into the frame buffer at a specified coordinate position.
- With characters that are defined as curve outlines, character shapes are scan converted into the frame buffer.
- Display processors are also designed to perform a number of additional operations.
- These functions include generating various line styles (dashed, dotted, or solid), displaying colour areas, and performing certain transformations and manipulations on displayed objects.
- Also, display processors are typically designed to interface with interactive input devices, such as a mouse.
- In an effort to reduce memory requirements in raster systems, methods have been devised for organizing the frame buffer as a linked list and encoding the intensity information.
- One way to do this is to store each scan line as a set of integer pairs.
- One number of each pair indicates an intensity value, and the second number specifies the number of adjacent pixels on the scan line that are to have that intensity.
- This technique, called run-length encoding, can result in a considerable saving in storage space if a picture is to be constructed mostly with long runs of a single colour each.
- A similar approach can be taken when pixel intensities change linearly.
- Another approach is to encode the raster as a set of rectangular areas (cell encoding).
- The disadvantages of encoding runs are that intensity changes are difficult to make and storage requirements actually increase as the length of the runs decreases.
- In addition, it is difficult for the display controller to process the raster when many short runs are involved.

Random-Scan Systems

- The organization of a simple random-scan (vector) system is shown in Figure 18

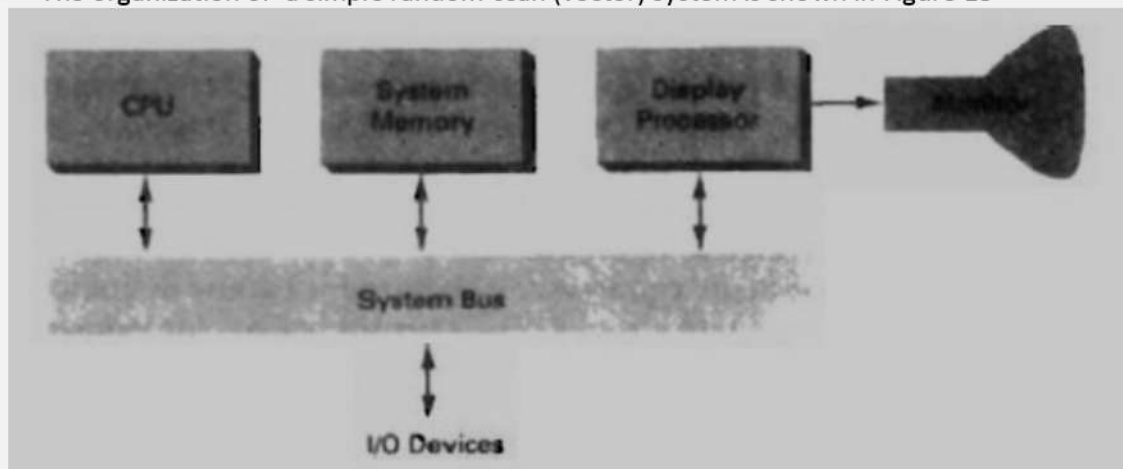


Figure 2-18: Architecture of a simple random scan system

- 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.
- 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.

Graphics monitors and work stations

- Most graphics monitors operate as raster-scan displays.
- Graphics systems range from small general-purpose computer systems with graphics capabilities to sophisticated full-color systems that are designed specifically for graphics applications.
- A typical screen resolution for personal computer systems, such as the Apple Quadra is 640 by 480, although screen resolution and other system capabilities vary depending on the size and cost of the system.
- Diagonal screen dimensions for general-purpose personal computer systems can range from 12 to 21 inches, and allowable color selections range from 16 to over 32,000.
- For workstations specifically designed for graphics applications, typical screen resolution is 1280 by 1024, with a screen diagonal of 16 inches or more.
- Graphics workstations can be configured with from 8 to 24 bits per pixel (full-color systems), with higher screen resolutions, faster processors, and other options available in high-end systems.
- A high-definition graphics monitor used in applications such as air traffic control, simulation, medical imaging, and CAD.
- This system has a diagonal screen size of 27 inches, resolutions ranging from 2048 by 1536 to 2560 by 2048, with refresh rates of 80 Hz or 60 Hz noninterlaced.
- A multiscreen system called the Media Wall, provides a large "wall-sized display area. This system is designed for applications that require large area displays in brightly lighted environments, such as at trade shows, conventions, retail stores, museums, or passenger terminals.
- Media Wall operates by splitting images into a number of Sections and distributing the sections over an array of monitors or projectors using a graphics adapter and satellite control units.
- An array of up to 5 by 5 monitors, each with a resolution of 640 by 480, can be used in the Media Wall to provide an overall resolution of 3200 by 2400 for either static scenes or animations.
- Scenes can be displayed behind mullions, or the mullions can be eliminated to display a continuous picture with no breaks between the various sections.
- Many graphics workstations are configured with two monitors. One monitor can be used to show all features of an object or scene, while the second monitor displays the detail in some part of the picture. Another use for dual-monitor systems is to view a picture on one monitor and display graphics options (menus) for manipulating the picture components on the other monitor.

Input devices

An input device is that is used to interact with or provide data to the computer. Various devices are available for data input on graphics workstations. The most common input devices are the mouse and keyboard. However, additional devices like trackball, spaceball, joysticks, digitizers, button boxes are specially designed for interactive input. Some other input devices used in particular applications are data gloves, touch panels, image scanners and voice systems.

KEYBOARD

1. An alphanumeric keyboard on a graphics system is used primarily as a device for entering text strings
2. Keyboard is an efficient device for inputting such as non graphic data as picture labels associated with a graphic display

3. Cursor keys and function keys are common features on general purpose keyboards
4. Function keys allow users to enter frequently used operations in a single keystroke
5. Cursor control keys can be used to select displayed objects or coordinate positions by positioning the screen cursor
6. Other types of cursor pointing devices such as track ball or joystick are included on some keyboards
7. A numeric keypad is often included on the keyboard for fast entry of numeric data
8. For specialized applications, input to a graphics application may come from a set of buttons, dials or switches that select data values or customized graphics operations
9. Buttons and switches are often used to input predefined functions and dials are common devices for entering scalar values
10. Real numbers within some defined range are selected for input with dial rotations
11. Potentiometers are used to measure dial rotations, which are then converted to deflection voltages for cursor movement

MOUSE

1. 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
2. Another method for detecting mouse motion is with an optical sensor
3. For these systems, the mouse is moved over a special mouse pad that has a grid of horizontal and vertical lines
4. The optical sensor detects movement across the lines in the grid
5. 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
6. 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
7. Additional devices can be included in the basic mouse design to increase the number of allowable input parameters
8. The z mouse includes 3 buttons, a thumb wheel on the side, a track ball on the top and a standard mouse ball underneath
9. With z mouse, we can pick up an object, rotate it and move it in any direction or we can navigate our viewing position and orientation through a 3-D scene
10. Applications of z-mouse include virtual reality, CAD and animation

Track ball and Space ball

1. A track ball is a ball that can be rotated with fingers or palm of the hand to produce screen cursor movement
2. Potentiometers, attached to the ball measure the amount and direction of rotation
3. Track balls are often mounted on keyboards or other devices such as the z-mouse
4. While a track ball is a 2-D positioning device, a space ball provides six degrees of freedom
5. Unlike the track ball, space ball does not actually move strain gauges measure the amount of pressure applied to the space ball to provide input for spatial positioning and orientation as the ball is pushed or pulled in various directions
6. Space balls are used for 3-D positioning and selection operations in virtual reality systems, modeling, animation, CAD and other applications.

Joystick

1. A Joystick has a small, vertical lever (called the stick) mounted on the base and used to steer the screen cursor around
2. It consists of two potentiometers attached to a single lever
3. Moving the lever changes the settings on the potentiometers
4. The left or right movement is indicated by one potentiometer and forward or back movement is indicated by other potentiometer

Data Glove

1. The data glove is used to grasp a virtual object
2. It is constructed with a series of sensors that detect hand and finger motions
3. Each sensor is a short length of fiber optic cable, with a light-emitting diode at one end and a phototransistor at the other end
4. The surface of a cable is roughened in the area where it is to be sensitive to bending
5. When the cable is fixed, some of the LED's light is lost, so less light is received by the phototransistor

Digitizers

1. A common device for drawing, painting, or interactively selecting coordinate positions on an object is a digitizer.
2. These devices can be used to input coordinate values in either a two-dimensional or a three-dimensional space.
3. Typically, a digitizer is used to scan over a drawing or object and to input a set of discrete coordinate positions, which can be joined with straight-line segments to approximate the curve or surface shapes.
4. 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.
5. The artist's digitizing system uses electromagnetic resonance to detect the three-dimensional position of the stylus. 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 44 by 60 inches or larger for floor models.
6. 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.
7. Many graphics tablets are constructed with a rectangular grid of wires embedded in the tablet surface.
8. 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.
9. Depending on the technology, signal strength, coded pulses, or phase shifts can be used to determine the position on the tablet.
10. Acoustic (or sonic) tablets use sound waves to detect a stylus position. Either strip microphones or point microphones can be used to detect the sound emitted by an electrical spark from a stylus tip.

Image Scanners

1. Drawings, graphs, color and black-and-white photos, or text can be stored for computer processing with an image scanner by passing an optical scanning mechanism over the information to be stored.
2. The gradations of gray scale or color are then recorded and stored in an array.
3. Once we have the internal representation of a picture, we can apply transformations to rotate, scale, or crop the picture to a particular screen area. We can also apply various image-processing methods to modify the array representation of the picture. For scanned text input, various editing operations can be performed on the stored documents.
4. Some scanners are able to scan either graphical representations or text, and they come in a variety of sizes and capabilities.

Touch Panels

1. Touch panels allow displayed objects or screen positions to be selected with the touch of a finger.
2. A typical application of touch panels is for the selection of processing options that are represented with graphical icons.
3. Some systems, such as the plasma panels, are designed with touch screens.
4. Other systems can be adapted for touch input by fitting a transparent device with a touch

sensing mechanism over the video monitor screen.

5. Touch input can be recorded using optical, electrical, or acoustical methods.
6. Optical touch panels employ a line of infrared light-emitting diodes (LEDs) along one vertical edge and along one horizontal edge of the frame.
7. The opposite vertical and horizontal edges contain light detectors. These detectors are used to record which beams are interrupted when the panel is touched.
8. 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 $\frac{1}{4}$ inch
9. 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.
10. The LEDs operate at infrared frequencies, so that the light is not visible to a user.
11. 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.
12. In acoustical touch panels, high-frequency sound waves are generated in the horizontal and vertical directions across a glass plate.
13. Touching the screen causes part of each wave to be reflected from the finger to the emitters.
14. 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

1. Such pencil-shaped devices are used to select screen positions by detecting the light coming from points on the CRT screen.
2. They are sensitive to the short burst of light emitted from the phosphor coating at the instant the electron beam strikes a particular point.
3. Other Light sources, such as the background light in the room, are usually not detected by a light pen.
4. 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.
5. As with cursor-positioning devices, recorded Light-pen coordinates can be used to position an object or to select a processing option.
6. Disadvantages:
 - When a light pen is pointed at the screen, part of the screen image is obscured by the hand and pen.
 - Prolonged use of the light pen can cause arm fatigue.
 - Light pens require special implementations for some applications because they cannot detect positions within black areas.
 - To be able to select positions in any screen area with a light pen, we must have some nonzero intensity assigned to each screen pixel.
 - Light pens sometimes give false readings due to background lighting in a room.

Voice Systems

1. Speech recognizers are used in some graphics workstations as input devices to accept voice commands
2. The voice-system input can be used to initiate graphics operations or to enter data.
3. These systems operate by matching an input against a predefined dictionary of words and phrases.
4. A dictionary is set up for a particular operator by having the operator speak the command words to be used into the system.
5. 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.

6. Later, when a voice command is given, the system searches the dictionary for a frequency-pattern match.
7. Voice input is typically spoken into a microphone mounted on a headset
8. The microphone is designed to minimize input of other background sounds.
9. If a different operator is to use the system, the dictionary must be reestablished with that operator's voice patterns.
10. 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.

Output primitives

- A picture can be described in several ways.
 - In a raster display, a picture is completely specified by the set of intensities for the pixel positions in the display.
 - At the other extreme, a picture can be described as a set of complex objects, such as trees and terrain or furniture and walls, positioned at specified coordinate locations within the scene.
- Shapes and colors of the objects can be described internally with pixel arrays or with sets of basic geometric structures, such as straight line segments and polygon color areas.
- The scene is then displayed either by loading the pixel arrays into the frame buffer or by scan converting the basic geometric-structure specifications into pixel patterns.
- Graphics programming packages provide functions to describe a scene in terms of these basic geometric structures, referred to as output primitives, and to group sets of output primitives into more complex structures.
- Each output primitive is specified with input coordinate data and other information about the way that object is to be displayed.
- Points and straight line segments are the simplest geometric components of pictures.
- Additional output primitives that can be used to construct a picture include circles and other conic sections, quadric surfaces, spline curves and surfaces, polygon color areas, and character strings.

Points and lines

POINT	It is the position in a plane and can be specified with an ordered pair of coordinates (x, y) where x is Horizontal distance from the origin and y is vertical distance from origin.
LINE	Two points will specify a line. A line is specified by equations such that if a point (x, y) satisfy the equation. Then the point is said to be lying on the line. Example: If two points are used to specify a line are (x₁, y₁) and (x₂, y₂), then equation of the line is $(y_2 - y_1)(x - x_1) = (x_2 - x_1)(y - y_1)$ $y = y_1 + (y_2 - y_1)(x - x_1)/(x_2 - x_1)$ This is in the form of $y = mx + c$ where $m = (y_2 - y_1)/(x_2 - x_1)$ and $c = y_1 - mx_1$

Point Plotting

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

Line Drawing

- The line is the most fundamental drawing primitive with many uses - charts, engineering drawings, illustrations, 2D pencil-based animation and curve approximation.
- Desired properties of line drawing algorithms
 - || Lines should appear sharp and straight
 - || Line should terminate accurately
 - || Line should have constant density
 - || Line density should be independent of line, length and angle.
 - || Efficient
- Line drawing is accomplished by calculating intermediate positions along directed to fill in these positions between the endpoint positions. An output device is then directed to fill in these positions between the endpoints.
- For analog devices, such as a vector pen plotter or a random-scan display, a straight line can be drawn smoothly from one endpoint to the other. Linearly varying horizontal and vertical deflection voltages are generated that are proportional to the required changes in the x and y directions to produce smooth lines.
- Digital devices display a straight line segment by plotting discrete points between two endpoints. Discrete coordinate positions along the line path are calculated from the equation of the line.
- For a raster video display, the line color (intensity) is then loaded into the frame buffer at the corresponding pixel coordinates. Reading from the frame buffer, the video controller then "plots" the screen pixels. Screen locations are referenced with integer values, so plotted positions may only approximate actual Line positions between two specified endpoints. A computed line position of (10.48,20.51), for example, would be converted to pixel position (10,21). Thus rounding of coordinate values to integers causes lines to be displayed with a stairstep appearance ("the jaggies"), as represented in Figure 2-1.

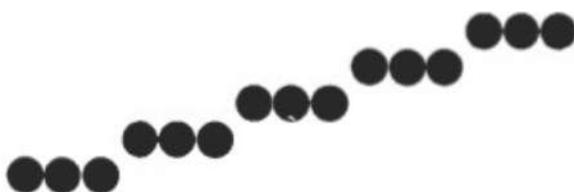


Figure 2-1 Stairstep effect (jaggies) produced when a line is generated as a series of pixel positions.

- The characteristic stairstep shape of raster lines is particularly noticeable on systems with low resolution, their appearance is improved by displaying them on high-resolution systems.
- More effective techniques for smoothing raster lines are based on adjusting pixel intensities along the line paths.
- For the raster-graphics device-level algorithms, object positions are specified directly in integer device coordinates. The pixel positions are referenced according to scan-line number and column number (pixel position across a scan line). This addressing scheme is shown in Figure 2-2.
- Scan lines are numbered consecutively from 0, starting at the bottom of the screen; and pixel columns are numbered from 0, left to right across each scan line.
- To load a specified color into the frame buffer at a position corresponding to column x along scan line y , a low-level procedure set Pixel of the form is used

set Pixel(x,y)

- To retrieve the current frame buffer intensity setting for a specified location the low-level function get Pixel is used

get Pixel(x,y)

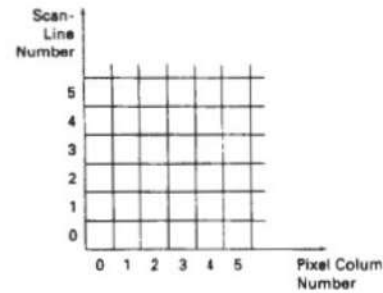


Figure 2-2: Pixel positions referenced by scan line number and column number.

Line Drawing algorithms

- Straight line segments are used greatly in computer generated pictures. They occur in block diagrams, bar charts and graphics, civil and mechanical engineering drawings, logical schematics, and architectural plans are created and used in computer graphics.

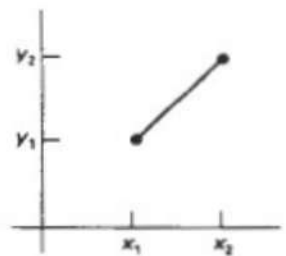


Figure 2.3 Line path between endpoint positions (x_1, y_1) and (x_2, y_2).

The Cartesian slope-intercept equation for a straight line is

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

with m representing the slope of the line and b as the y intercept. Given that the two endpoints of a line segment are specified at positions (x_1, y_1) and (x_2, y_2), as shown in Figure. 2-3, values for the slope m and y intercept b are determined with the following calculations:

$$m = \frac{y_2 - y_1}{x_2 - x_1} \quad - (2)$$

$$b = y_1 - m \cdot x_1 \quad - (3)$$

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

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

$$\Delta y = m \Delta x \quad - (4)$$

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

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

These equations form the basis for determining deflection voltages in analog devices.

- For lines with slope magnitudes $|m| < 1$, Δx can be set proportional to a small horizontal deflection voltage and the corresponding vertical deflection is then set proportional to Δy as calculated from Eq. (4).

- For lines whose slopes have magnitudes $|m| > 1$, Δy can be set proportional to a small vertical deflection voltage with the corresponding horizontal deflection voltage set proportional to Δx , calculated from Eq. (5).
- For lines with $m = 1$, $\Delta x = \Delta y$ and the horizontal and vertical deflections voltages are equal. In each case, a smooth line with slope m is generated between the specified endpoints.

On raster systems, lines are plotted with pixels, and step sizes in the horizontal and vertical directions are constrained by pixel separations. That is, a line is sampled at discrete positions and determines the nearest pixel to the line at each sampled position. This scan-conversion process for straight lines is illustrated in Fig. 2-Fig 2-4, for a near horizontal line with discrete sample positions along the x axis.

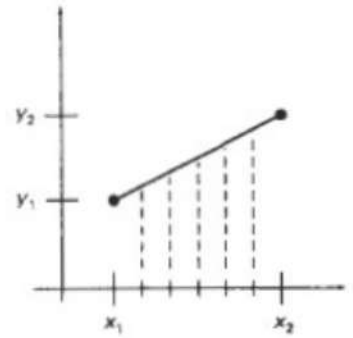


Figure 2-4: Straight line segment with five sampling positions along the x axis between x_1 and x_2 .

There are three types of line drawing algorithms:

- Digital differential Algorithm (DDA)
- Bresenham's Line drawing Algorithm.
- Parallel Line Algorithms

DDA Algorithm

- The digital differential analyzer (DDA) is a scan-conversion line algorithm based on calculating either Δy or Δx , using Eq. (4) or Eq. (5).
- The line is sampled at unit intervals in one coordinate and corresponding integer values nearest the line path for the other coordinate are determined.
- For line with positive slope(slope is less than or equal to 1), we sample at unit x intervals ($\Delta x = 1$) and compute each successive y value as

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

- Subscript k takes integer values starting from 1, for the first point, and increases by 1 until the final endpoint is reached.
- Since m can be any real number between 0 and 1, the calculated y values must be rounded to the nearest integer.

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

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

- Equations (6) and (7) are based on the assumption that lines are to be processed from the left endpoint to the right endpoint (Fig. 2-3).
- If this processing is reversed, so that the starting endpoint is at the right, then either we have $\Delta x = -1$ and

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

- or (when the slope is greater than 1) we have $\Delta y = -1$ with

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

- Equations (6) through (9) can also be used to calculate pixel positions along a line with negative slope.
- If the absolute value of the slope is less than 1 and the start endpoint is at the left, we set $\Delta x = 1$ and calculate y values with Eq. (6).
- When the start endpoint is at the right (for the same slope), we set $\Delta x = -1$ and Output Primitives obtain y positions from Eq. (8).

- Similarly, when the absolute value of a negative slope is greater than 1, we use $\Delta y = -1$ and Eq. (9) or we use $\Delta y = 1$ and Eq. (7).

Algorithm:

Step1

Input two end point pixel positions (x_1, y_1) and (x_2, y_2)

Step2

Find horizontal and vertical difference between the end points

$$dx = x_2 - x_1$$

$$dy = y_2 - y_1$$

Step3

The difference with the greater magnitude determines the value of parameter steps

If $\text{abs}(dx) > \text{abs}(dy)$ then

$$\text{steps} = \text{abs}(dx)$$

Else

$$\text{steps} = \text{abs}(dy)$$

Step4

Starting with pixel position (x_1, y_1) be determined offset needed at each step to generate next pixel along the line path.

$$x\text{Increment} = dx / \text{steps}$$

$$y\text{Increment} = dy / \text{steps}$$

Step5

Assign the values of x_1, y_1 to x, y

$$x = x_1$$

$$y = y_1$$

Step6

Plot the pixel at (x, y) position on screen set pixel (round(x), round(y), 1). Here '1' is the intensity of pixel i.e., intensity with which picture is illuminated.

step7

Calculate the values of x and y for the next pixel position.

$$x = x + x\text{Increment}$$

$$y = y + y\text{Increment}$$

Step8

Plot the pixel at (x, y) position, set pixel (round(x), round(y), 1)

Step9

Repeat the steps 7 and 8 until the value of l i.e., start from 1 to steps.



EXAMPLE: Draw A line between the points (20,10) and (30,18)

Step 1: $x_1 = 20$, $x_2 = 30$, $y_1 = 10$, $y_2 = 18$.

Step 2: $dx = 30 - 20 = 10$, $dy = 18 - 10 = 8$.

Step 3: $dx > dy \Rightarrow 10 > 8$, steps = 10.

Step 4: x increment = $10/10 = dx/steps = 1$.

y increment = $8/10 = dy/steps = 0.8$.

Step 5: $x = 20$, $y = 10$.

Step 6: pixel (20,10) .

Step 7: $x = x + x$ increment.

$y = y + y$ increment.

l	x	y	(round(x),round(y))
1	$20+1=21$	$10+0.8=10.8$	(21,11)
2	$21+1=22$	$10.8+0.8=11.6$	(22,12)
3	$22+1=23$	$11.6+0.8=12.4$	(23, 12)
4	$23+1=24$	$12.4+0.8=13.2$	(24,13)
5	$24+1=25$	$13.2+0.8=14$	(25,14)
6	$25+1=26$	$14+0.8=14.8$	(26,15)
7	$26+1=27$	$14.8+0.8=15.6$	(27,16)
8	$27+1=28$	$15.6+0.8=16.4$	(28,16)
9	$28+1=29$	$16.4+0.8=17.2$	(29,17)
10	$29+1=30$	$17.2+0.8=18$	(30,18)

Advantages:

- The DDA algorithm is a faster method for calculating pixel positions than the direct use of Eqs.-(1).
- It eliminates the multiplication in Eq. (1) by making use of raster characteristics, so that appropriate increments are applied in the x or y direction to step to pixel positions along the line path.
- Does not calculate coordinates based on the complete equation (uses offset method)

Disadvantages:

- Round-off errors are accumulated, thus line diverges more and more from straight line
- Round-off operations take time
- Perform integer arithmetic by storing float as integers in numerator and denominator and performing integer arithmetic

Bresenham's Line Algorithm

- An accurate and efficient raster line-generating algorithm, developed by Bresenham, scans converts lines using only incremental integer calculations that can be adapted to display circles and other curves.
- Figures 2-5 and 2-6 illustrate sections of a display screen where straight line segments are to be drawn.
- The vertical axes show scan-line positions, and the horizontal axes identify pixel columns.
- Sampling at unit x intervals in these examples, it is decided which of two possible pixel positions is closer to the line path at each sample step.
- Starting from the left endpoint shown in Fig. 2-5, the next sample position whether to plot the pixel at position (11, 11) or the one at (11, 12) is determined
- Similarly, Fig. 2-6 shows a negative slope-line path starting from the left endpoint at pixel position (50, 50). In this, the next pixel position as (51, 50) or as (51, 49) is decided
- Bresenham's line algorithm is used here, by testing the sign of an integer parameter, whose value is proportional to the difference between the separations of the two pixel positions from the actual line path.
- To illustrate Bresenham's approach, first the scan-conversion process is considered for lines with positive slope less than 1.
- Pixel positions along a line path are then determined by sampling at unit x intervals.
- Starting from the left endpoint (x_0, y_0) of a given line, we step to each successive column (x position) and plot the pixel whose scan-line y value is closest to the line path.
- Figure 2-7 demonstrates the k th step in this process. Assuming that the pixel at (x_k, y_k) is to be displayed are determined, it is to needed to decide which pixel to plot in column x_{k+1} . The choices are the pixels at positions (x_{k+1}, y_k) and (x_{k+1}, y_{k+1}) .
- At sampling position x_{k+1} , vertical pixel separations from the mathematical line path are labeled as d_1 , and d_2 (Fig. 2-8). The y coordinate on the mathematical line at pixel column position x_{k+1} is calculated as

$$y = m(x_k + 1) + b \quad -(10)$$

- Then

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

- And

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

- The difference between these two separations is

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

- A decision parameter p_k for the k th step in the line algorithm can be obtained by rearranging Eq. (11) so that it involves only integer calculations.

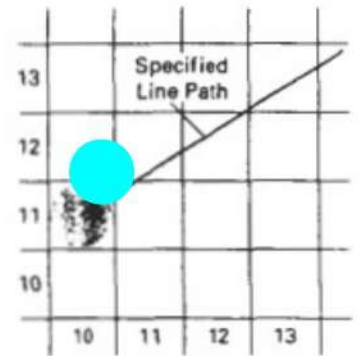


Figure 2-5: Section of a display screen where a straight line segment is to be plotted, starting from the pixel at

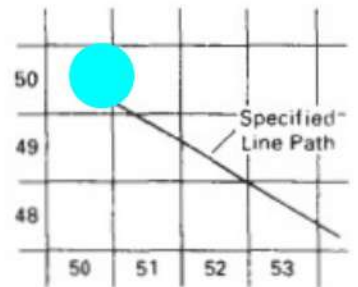


Figure 2-6: Section of a display screen where a negative slope line segment is to be plotted, starting from the

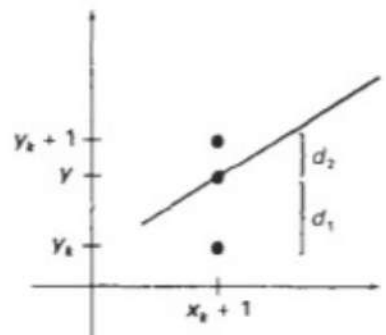
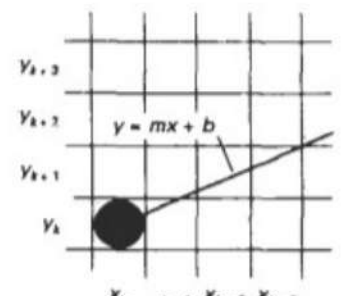


Figure 2-8: Distances between pixel positions and the line y coordinate at sampling position x_{k+1}

- This is achieved by substituting $m = \Delta y / \Delta x$, where Δy and Δx are the vertical and horizontal separations of the endpoint positions, and defining:

$$p_k = \Delta x(d_1 + d_2) \\ = 2\Delta y \cdot x_k - 2\Delta x \cdot y_k + c \quad \text{---(12)}$$

- The sign of p_k , is the same as the sign of $d_1 - d_2$, since $\Delta x > 0$.
- For example here, Parameter c is constant and has the value $2\Delta y + \Delta x (2b - 1)$, which is independent of pixel position and will be eliminated in the recursive calculations for p_k .
- If the pixel at y_k is closer to the line path than the pixel at y_{k+1} (that is, $d_1 < d_2$), then decision parameter p_k is negative. In that case, the lower pixel is plotted; otherwise, the upper pixel is plotted.
- Coordinate changes along the line occur in unit steps in either the x or y directions.
- Therefore, we can obtain the values of successive decision parameters using incremental integer calculations.
- At step $k + 1$, the decision parameter is evaluated from Eq. 3-12 as

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

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

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

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

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

- where the term $y_{k+1} - y_k$ is either 0 or 1, depending on the sign of parameter p_k .
- This recursive calculation of decision parameters is performed at each integer x position, starting at the left coordinate endpoint of the line.
- The first parameter, p_0 is evaluated from Eq. 3-12 at the starting pixel position (x_0, y_0) and with m evaluated as $\Delta y / \Delta x$:

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

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

Algorithm:

Step1

Input the two line end points and store the left endpoints in (x_0, y_0)

Step2

Load (x_0, y_0) into the frame buffer; plot the first point.

Step3

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

$$\Delta x = x_n - x_0 \text{ and } \Delta y = y_n - y_0$$

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

Step4

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

$$p_{k+1} = p_k + 2\Delta y - 2\Delta x$$

Step5

Repeat step 4 Δx times.

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

EXAMPLE: Draw A line between the points (20,10) and (30,18)

This line has a slope of 0.8, with

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

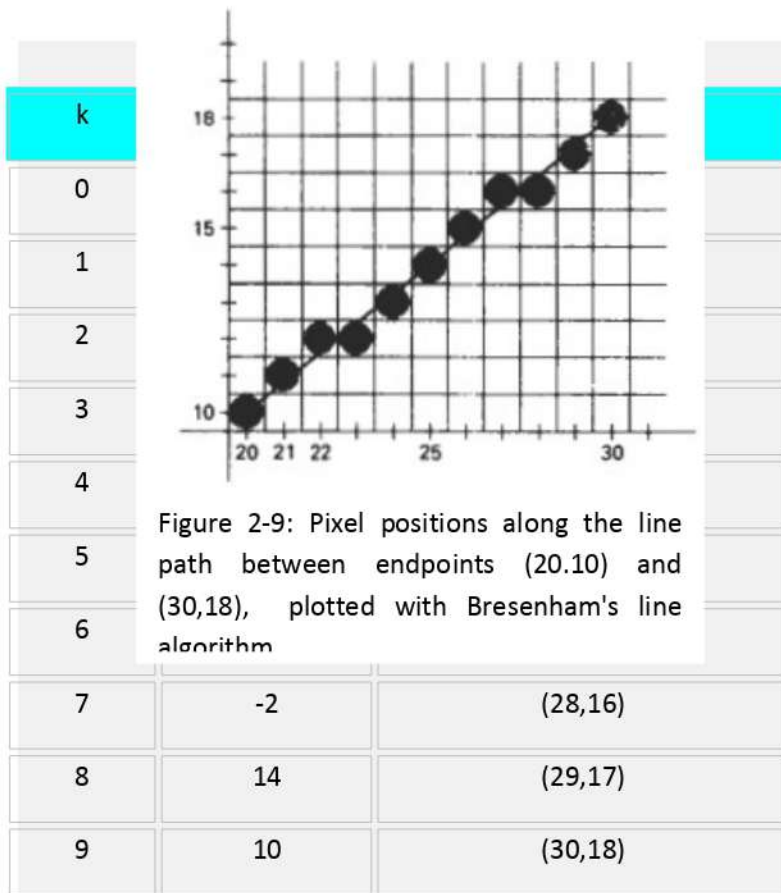
The initial decision parameter has the value

$$\begin{aligned} p_0 &= 2\Delta y - \Delta x \\ &= 6 \end{aligned}$$

and the increments for calculating successive decision parameters are

$$\begin{aligned} 2\Delta y &= 16 \\ 2\Delta y - 2\Delta x &= -4 \end{aligned}$$

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



A plot of the pixels generated along this line path is shown in Fig. 2-9.

Advantages:

- Bresenham's algorithm generally faster than DDA
- Bresenham's algorithm uses integer arithmetic
- Constants need to be

computed only once

Parallel Line Algorithms

- The line-generating algorithms such as DDA or Bresenham's determine pixel positions sequentially.
- With a parallel computer, pixel positions are along a line path simultaneously by partitioning the computations among the various processors available.
- Approaches to the partitioning problem
 - to adapt an existing sequential algorithm to take advantage of multiple processors.
 - look for other ways to set up the processing so that pixel positions can be calculated efficiently in parallel.
- Devising a parallel algorithm is to balance the processing load among the available processors.

Parallel Bresenham's algorithm

- Given n , processors, a parallel Bresenham's line algorithm is by subdividing the line path into n , partitions and simultaneously generating line segments in each of the subintervals.
- For a line with slope $0 < m < 1$ and left endpoint coordinate position (x_0, y_0) , the line is partitioned along the positive x direction.
- The distance between beginning x positions of adjacent partitions can be calculated as

$$-(15)$$

- where Δx is the width of the line, and the value for partition width Δx is computed using integer division.
- Numbering the partitions, and the processors as $0, 1, 2$, up to $n_p - 1$, the starting x coordinate for the k_{th} partition is calculated as

$$x_k = x_0 + k\Delta x_p \quad -(16)$$

- **Example:** Suppose $\Delta x = 15$ and we have $n_p = 4$ processors.

- Then the width of the partitions is 4 and the starting x values for the partitions are x_0, x_0+4, x_0+8 and x_0+12 .
- To apply Bresenham's algorithm over the partitions, the initial value for the y coordinate is required and the initial value for the decision parameter in each partition. The change Δy , in the y direction over each partition is calculated from the line slope m and partition width Δx_p .

$$\Delta y_p = m \Delta x_p \quad -(17)$$
- At the k_{th} partition, the starting y coordinate is then

$$y_k = y_0 + \text{round}(k \Delta y_p) \quad -(18)$$
- The initial decision parameter for Bresenham's algorithm at the start of the k_{th} subinterval is obtained from Eq. 3-12:

$$p_k = (k \Delta x_p) (2 \Delta x_p) - \text{round}(k \Delta x) (2 \Delta x) + 2 \Delta y - \Delta x \quad -(18)$$
- Each processor then calculates pixel positions over its assigned subinterval using the starting decision parameter value for that subinterval and the starting coordinates (x_k, y_k) .
- The floating-point calculations to integer arithmetic in the computations are reduced for starting values y_k and p_k by substituting $m = \Delta y / \Delta x$ and rearranging terms.
- The extension of the parallel Bresenham's algorithm to a line with slope greater than 1 is achieved by partitioning the line in the y direction and calculating beginning x values for the partitions.
- For negative slopes, we increment coordinate values in one direction and decrement in the other.

Advantages:

- With this partitioning scheme, the width of the last(rightmost) subinterval will be smaller than the others in some cases.
- If the line endpoints are not integers, truncation errors can result in variable width partitions along the length of the line.

Other Approach

- Other Approach to set up Parallel algorithms on raster systems is to assign each processor to a particular group of screen pixels.
- With a sufficient number of processors (such as a Connection Machine CM-2 with over 65,000 processors), we can assign each processor to one pixel within some screen region.
- This approach can be adapted to line display by assigning one processor to each of the pixels within the limits of the line coordinate extents (bounding rectangle) and calculating pixel distances from the line path.
- The number of pixels within the bounding box of a line is $\Delta x, \Delta y$ (Fig. 2-10).
- Perpendicular distance d from the line in Fig. 2-10 to a pixel with coordinates (x, y) is obtained with the calculation

$$d = Ax + By + C \quad -(20)$$

- here

$$A = \frac{-\Delta y}{\text{linelength}}$$

$$B = \frac{\Delta x}{\text{linelength}}$$

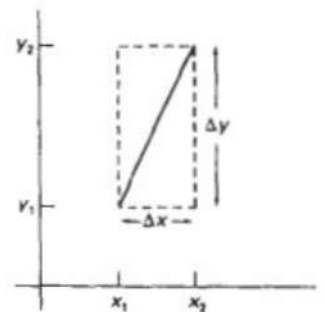


Figure 2-10: Bounding box for a line with coordinate extents

$$C = \frac{x_0 \Delta y - y_0 \Delta x}{\text{linelength}}$$

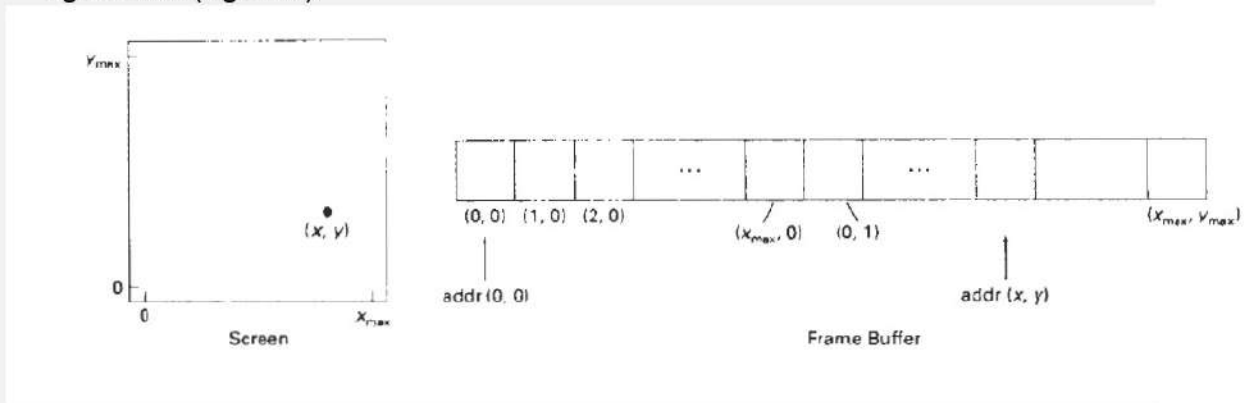
- with

$$\text{linelength} = \sqrt{\Delta x^2 + \Delta y^2}$$

- Once the constants A, B, and C have been evaluated for the line, each processor needs to perform two multiplications and two additions to compute the pixel distance d.
- A pixel is plotted if d is less than a specified line-thickness parameter.
- Instead of partitioning the screen into single pixels; we can assign to each processor either a scan line or a column of pixels depending on the line slope.
- Each processor then calculates the intersection of the line with the horizontal row or vertical column of pixels assigned that processor.
- For a line with slope $|m| < 1$, each processor simply solves the line equation for y, given an x column value.
- For a line with slope magnitude greater than 1, the line equation is solved for x by each processor, given a scan-line y value.
- Such direct methods, although slow on sequential machines, can be performed very efficiently using multiple processors

Loading the Frame Buffer

- When straight line segments and other objects are scan converted for display with a raster system, frame-buffer positions must be calculated.
- This is accomplished with the set pixel procedure, which stores intensity values for the pixels at corresponding addresses within the frame-buffer array.
- Scan-conversion algorithms generate pixel positions at successive unit intervals.
- Hence incremental methods are used to calculate frame-buffer addresses.
- As a specific example, suppose the frame-buffer array is addressed in row major order and that pixel positions vary from (0,0) at the lower left screen corner to $(x_{\max}, y_{\max})$ at the top right corner (Fig. 2-11).



- For a bi-level system (1 bit per pixel), the frame-buffer bit address for pixel position (x, y) is calculated as

$$\text{addr}(x,y) = \text{addr}(0,0) + y(x_{\max}+1) + x \quad -(21)$$
- Moving across a scan line, the frame-buffer address for the pixel at $(x + 1, y)$ is calculated as the following offset from the address for position (x, y):

$$\text{addr}(x+1,y) = \text{addr}(x,y) + 1 \quad -(22)$$
- Stepping diagonally up to the next scan line from (x, y), frame buffer address of $(x + 1, y+1)$ is calculated as

$$\text{addr}(x+1,y+1) = \text{addr}(x,y) + x_{\max} + 2 \quad -(23)$$

- Similar incremental calculations can be obtained from Eq. 21 for unit steps in the negative x and y screen directions. Each of these address calculations involves only a single integer addition.
- Methods for implementing the set pixel procedure to store pixel intensity values depend on the capabilities of a particular system and the design requirements of the software package.
- With systems that can display a range of intensity values for each pixel, frame-buffer address calculations would include pixel width (number of bits), as well as the pixel screen location.

Mid-Point circle and Ellipse algorithms

Mid-Point circle Algorithm

- As in the raster line algorithm, unit intervals are taken and the closest pixel position is determined to the specified circle path at each step.
- For a given radius r and screen center position (x_c, y_c) , the algorithm is first setup to calculate pixel positions around a circle path centered at the coordinate origin $(0,0)$.
- Then each calculated position (x, y) is moved to its proper screen position by adding x_c to x and y_c to y .
- Along the circle section from $x = 0$ to $x = y$ in the first quadrant, the slope of the curve varies from 0 to -1.
- Therefore, unit steps are taken in the positive x direction over this octant and a decision parameter is used to determine which of the two possible y positions is closer to the circle path at each step.
- Positions in the other seven octants are then obtained by symmetry.
- To apply the midpoint method, a circle function is defined:

$$f_{\text{circle}}(x,y) = x^2 + y^2 - r^2 \quad -(24)$$

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

$$f_{\text{circle}}(x, y) \begin{cases} < 0, & \text{if } (x, y) \text{ is inside the circle boundary} \\ = 0, & \text{if } (x, y) \text{ is on the circle boundary} \\ > 0, & \text{if } (x, y) \text{ is outside the circle boundary} \end{cases} \quad -(25)$$

- The circle-function tests in Eqs. (25) are performed for the midpositions between pixels near the circle path at each sampling step.
- Thus, the circle function is the decision parameter in the midpoint algorithm, and incremental calculations for this function are similar to the line algorithm.
- Figure 2-12 shows the midpoint between the two candidate pixels at sampling position x_k+1 .
- Assuming the pixel at (x_k, y_k) is plotted, next the pixel at position (x_k+1, y_k) or the one at position $(x_k+1, y_k - 1)$ is closer to the circle is determined.
- Decision parameter is the circle function (25) is evaluated at the midpoint between these two pixels:

$$\begin{aligned} p_k &= f_{\text{circle}}\left(x_k + 1, y_k - \frac{1}{2}\right) \\ &= (x_k + 1)^2 + \left(y_k - \frac{1}{2}\right)^2 - r^2 \end{aligned} \quad -(26)$$

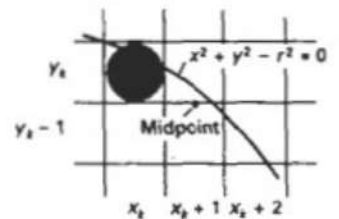


Figure 2-12: Midpoint between candidate pixels at

- If $p_k < 0$, this midpoint is inside the circle and the pixel on scan line y_k is closer to the circle boundary.
- Otherwise, the midposition is outside or on the circle boundary, and the pixel on scan line $y_k + 1$ is selected.
- Successive decision parameters are obtained using incremental calculations.
- A recursive expression for the next decision parameter is determined by evaluating the circle function at sampling position $x_{k+1}+1 = x_k+2$:

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

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

or

$$p_{k+1} = p_k + 2(x_k + 1) + (y_{k+1}^2 - y_k^2) - (y_{k+1} - y_k) + 1 \quad -(27)$$

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

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

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

- At the start position $(0, r)$, these two terms have the values 0 and $2r$, respectively.
- Each successive value is obtained by adding 2 to the previous value of $2x$ and subtracting 2 from the previous value of $2y$.
- The initial decision parameter is obtained by evaluating the circle function at the start position $(x_o, y_o) = (0, r)$:

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

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

- or

$$p_0 = \frac{5}{4} - r \quad -(28)$$

- If the radius r is specified as an integer, p_0 is rounded to $p_0 = 1 - r$ (for r an integer)
- since all increments are integers.
- As in Bresenham's line algorithm, the midpoint method calculates pixel positions along the circumference of a circle using integer additions and subtractions, assuming that the circle parameters are specified in integer screen coordinates.

Algorithm:

Step 1

Input radius r and circle center (x_c, y_c) and obtain the first point on the circumference of a circle

centered on the origin as

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

Step 2

Calculate the initial value of the decision parameter as

$$p_0 = 5/4 - r$$

Step 3

At each x_k position starting at $k=0$, perform the following test:

If $p_k < 0$, the next point along the circle centered on $(0, 0)$ is (x_{k+1}, y_k) and

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

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

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

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

Step 4

Determine symmetry points in the other seven octants.

Step 5

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

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

Step 6

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

Example:

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

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

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

$$2x_0 = 0 \text{ and } 2y_0 = 20$$

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

k	p
0	-9
1	-6
2	-1
3	6
4	-3
5	8
6	5

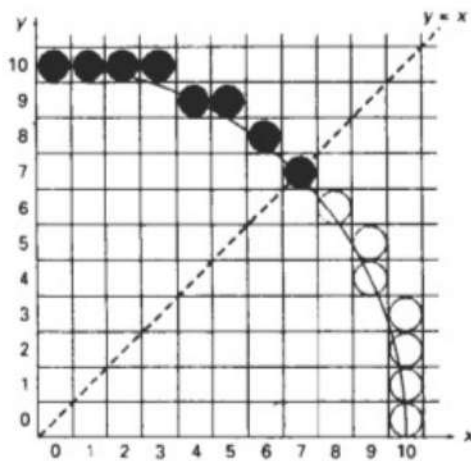


Figure 2-13: Selected pixel positions(solid circles) along a circle path with radius $r=10$ centered on the origin, using the midpoint circle algorithm. Open circles show the symmetry positions in the first quadrant.

- A plot of the generated pixel positions in the first quadrant is shown in Fig. 2-13.

Mid-point Ellipse algorithms

- Approach is similar to that used in displaying a raster circle.
- Given parameters d_x , r_y , and (x_c, y_c) , the points (x, y) for an ellipse in standard position centered on the origin are determined, and then shifted to the points so the ellipse is centered at (x_c, y_c) .
- The midpoint ellipse method is applied throughout the first quadrant in two parts.
- Figure 2-14 shows the division of the first quadrant according to the slope of an ellipse with $d_x < r_y$.
- This quadrant is processed taking unit steps in the x direction where the slope of the curve has a magnitude less than 1, and taking unit steps in the direction where the slope has a magnitude greater than 1.
- Regions 1 and 2 (Fig. 2-14), can be processed in various ways.
- One can start at position $(0, r_y)$ and step clockwise along the elliptical path in the first quadrant, shifting from unit steps in x to unit steps in y when the slope becomes less than -1.
- Alternatively, one can start at $(r_x, 0)$ and select points in a counterclockwise order, shifting from unit steps in y to unit steps in x when the slope becomes greater than -1.
- With parallel processors, one could calculate pixel positions in the two regions simultaneously.
- As an example of a sequential implementation of the midpoint algorithm, we take the start position at $(0, r_y)$ and step along the ellipse path in clockwise order throughout the first quadrant.
- an ellipse function is defined with $(x_c, y_c) = (0, 0)$ as

$$\text{fellipse}(x, y) = r_y^2 x^2 + r_x^2 y^2 - r_x^2 r_y^2 \quad (29)$$
- which has the following properties

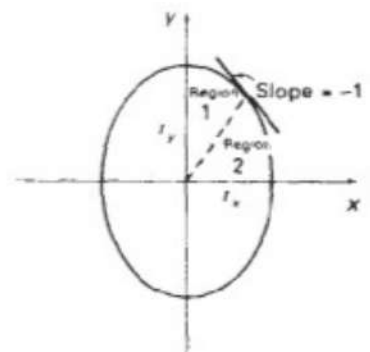


Figure 2-14: Ellipse processing regions, Over region 1, the magnitude of the ellipse slope is less than 1; over region 2, the magnitude of the slope is

$$f_{\text{ellipse}}(x, y) \begin{cases} < 0, & \text{if } (x, y) \text{ is inside the ellipse boundary} \\ = 0, & \text{if } (x, y) \text{ is on the ellipse boundary} \\ > 0, & \text{if } (x, y) \text{ is outside the ellipse boundary} \end{cases} \quad -(30)$$

- Thus, the ellipse function $f_{\text{ellipse}}(x, y)$ serves as the decision parameter in the midpoint algorithm.
- At each sampling position, the next pixel is selected along the ellipse path according to the sign of the ellipse function evaluated at the midpoint between the two candidate pixels.
- Starting at $(0, r_y)$ unit steps are taken in the x direction until the boundary between region 1 and region 2 is reached.
- Then switched to unit steps in the y direction over the remainder of the curve in the first quadrant.
- At each step, the value of the slope of the curve is tested.
- The ellipse slope is calculated from Eq. (29) as

$$\frac{dy}{dx} = -\frac{2r_y^2 x}{2r_x^2 y} \quad -(31)$$

- At the boundary between region 1 and region 2, $dy/dx = -1$ and $2r_y^2 x = 2r_x^2 y$

- Therefore, we move out of region 1 whenever $2r_y^2 x \geq 2r_x^2 y$ - (32)

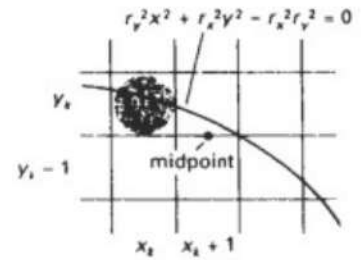


Figure 2-15: Midpoint between candidate pixels at sampling position $x_k + 1$ along an elliptical

- Figure 2-15 shows the midpoint between the two candidate pixels at sampling position $x_k + 1$ in the first region.
- Assuming position (x_k, y_k) has been selected at the previous step, the next position along the ellipse path is determined by evaluating the decision parameter (that is, the ellipse function (29)) at this midpoint:

$$\begin{aligned} p1_k &= f_{\text{ellipse}}\left(x_k + 1, y_k - \frac{1}{2}\right) \\ &= r_y^2(x_k + 1)^2 + r_x^2\left(y_k - \frac{1}{2}\right)^2 - r_x^2 r_y^2 \end{aligned} \quad -(33)$$

- If $p1_k < 0$, the midpoint is inside the ellipse and the pixel on scan line y_k is closer to the ellipse boundary.
- Otherwise, the midpositions is outside or on the ellipse boundary, and the pixel on scan line $y_k - 1$ is selected.
- At the next sampling position $(x_{k+1} + 1 = x_k + 2)$, the decision parameter for region 1 is evaluated as

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

or

$$p1_{k+1} = p1_k + 2r_y^2(x_k + 1) + r_y^2 + r_x^2\left[\left(y_{k+1} - \frac{1}{2}\right)^2 - \left(y_k - \frac{1}{2}\right)^2\right] \quad -(34)$$

- where y_{k+1} is either y_k or $y_k - 1$, depending on the sign of $p1_k$.
- Decision parameters are incremented by the following amounts:

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

- As in the circle algorithm, increments for the decision parameters can be calculated using only addition and subtraction, since values for the terms $2r_y^2x$ and $2r_x^2y$ can also be obtained incrementally.

- At the initial position $(0, r_y)$, the two terms evaluate to

$$2r_y^2x = 0 \quad -(35)$$

$$2r_x^2y = 2r_x^2r_y \quad -(36)$$

- As x and y are incremented, updated values are obtained by adding $2r_y^2$ to (35) and subtracting $2r_x^2$ from (36).
- The updated values are compared at each step, and we move from region 1 to region 2 when condition (32) is satisfied.
- In region 1, the initial value of the decision parameter is obtained by evaluating the ellipse function at the start position $(x_o, y_o) = (0, r_y)$:

$$\begin{aligned} p_{10} &= f_{\text{ellipse}}\left(1, r_y - \frac{1}{2}\right) \\ &= r_y^2 + r_x^2\left(r_y - \frac{1}{2}\right)^2 - r_x^2r_y^2 \end{aligned}$$

or

$$p_{10} = r_y^2 - r_x^2r_y + \frac{1}{4}r_x^2 \quad -(37)$$

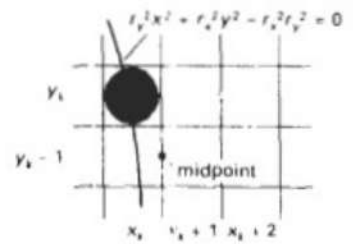


Figure 2-16 Midpoint between candidate pixels at sampling position $y_k - 1$ along an elliptical

- Over region 2, we sample at unit steps in the negative y direction, and the midpoint is now taken between horizontal pixels at each step (Fig. 2-16).
- For this region, the decision parameter is evaluated as

$$\begin{aligned} p_{2k} &= f_{\text{ellipse}}\left(x_k + \frac{1}{2}, y_k - 1\right) \\ &= r_y^2\left(x_k + \frac{1}{2}\right)^2 + r_x^2(y_k - 1)^2 - r_x^2r_y^2 \end{aligned} \quad -(38)$$

- If $p_{2k} > 0$, the midposition is outside the ellipse boundary, and the pixel at x_k is selected.
- If $p_{2k} \leq 0$, the midpoint is inside or on the ellipse boundary, and pixel position at x_{k+1} is selected.
- To determine the relationship between successive decision parameters in region 2, the ellipse function at the next sampling step $y_{k+1} - 1 = y_k - 2$ is evaluated as:

$$\begin{aligned} p_{2k+1} &= f_{\text{ellipse}}\left(x_{k+1} + \frac{1}{2}, y_{k+1} - 1\right) \\ &= r_y^2\left(x_{k+1} + \frac{1}{2}\right)^2 + r_x^2[(y_k - 1) - 1]^2 - r_x^2r_y^2 \end{aligned} \quad -(39)$$

- or

$$p_{2k+1} = p_{2k} - 2r_x^2(y_k - 1) + r_x^2 + r_y^2\left[\left(x_{k+1} + \frac{1}{2}\right)^2 - \left(x_k + \frac{1}{2}\right)^2\right] \quad -(40)$$

- with x_{k+1} , set either to x_k or to $x_k + 1$, depending on the sign of p_{2k} .
- When we enter region 2, the initial position (x_o, y_o) is taken as the last position selected in region 1 and the initial decision parameter in region 2 is then

$$\begin{aligned} p_{20} &= f_{\text{ellipse}}\left(x_o + \frac{1}{2}, y_o - 1\right) \\ &= r_y^2\left(x_o + \frac{1}{2}\right)^2 + r_x^2(y_o - 1)^2 - r_x^2r_y^2 \end{aligned} \quad -(41)$$

- To simplify the calculation of $p2_0$, pixel positions are selected in counterclockwise order starting at $(r_k, 0)$.
- Unit steps would then be taken in the positive y direction up to the last position selected in region 1.
- The midpoint algorithm can be adapted to generate an ellipse in nonstandard position using the ellipse function Eq. (42) and calculating pixel positions over the entire elliptical path.

$$Ax^2 + By^2 + Cxy + Dx + Ey + F = 0 \quad - (42)$$
- Assuming (x_k, y_k) and the ellipse center are given in integer screen coordinates, incremental integer calculations to values for the decision parameters are determined in the midpoint ellipse algorithm.
- The increments r_x^2 , r_y^2 , $2r_x^2$ and $2r_y^2$ are evaluated once at the beginning of the procedure.

Algorithm:

Step 1

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

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

Step 2

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

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

Step 3

At each x_k position in region 1, starting at $k=0$, perform the following test:

If $p1_k < 0$, the next point along the ellipse centered on $(0, 0)$ is (x_{k+1}, y_k) and

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

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

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

where $2r_y^2 x_{k+1} = 2r_y^2 x_k + 2r_y^2$ and $2r_x^2 y_{k+1} = 2r_x^2 y_k - 2r_x^2$

and continue until $2r_y^2 x \geq 2r_y^2 x$

Step 4

Calculate the initial value of the decision parameter in region 2 using the last point (x_0, y_0) calculated in region 1 as

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

Step 5

At each y_k position in region 2, starting at $k = 0$, perform the following test: If $p2_k > 0$, the next

point along the ellipse centered on (0, 0) is

If $p_{2k} < 0$, the next point along the ellipse centered on (0, 0) is (x_k, y_{k+1}) and

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

Otherwise, the next point along the circle is (x_{k+1}, y_{k+1}) and

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

using the same incremental calculations for x and y as in region 1.

Step 6

Determine symmetry points in the other three quadrants.

Step 7

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

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

Step 8

Repeat the steps for region 1 until $2r_y^2 x \geq 2r_x^2 y$

Example:

- Given input ellipse parameters $d_x = 8$ and $r_y = 6$, the steps in the midpoint ellipse algorithm are illustrated by determining raster positions along the ellipse path in the first quadrant.
- Initial values and increments for the decision parameter calculations are

$$2r_y^2 x = 0 \quad (\text{with increment } 2r_y^2 = 72)$$

$$2r_x^2 y = 2r_x^2 r_y \quad (\text{with increment } -2r_x^2 = -128)$$

- For region 1: The initial point for the ellipse centered on the origin is $(x_o, y_o) = (0, 6)$, and the initial decision parameter value is 1

$$p_{10} = r_y^2 - r_x^2 r_y + \frac{1}{4} r_x^2 = -332$$

- Successive decision parameter values and positions along the ellipse path are calculated

k	p_{1k}	(x_{k+1}, y_{k+1})	$2r_y^2 x_{k+1}$	$2r_x^2 y_{k+1}$
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

using the midpoint method as

- We now move out of region 1, since $2r_y^2x > 2r_x^2y$
- For region 2, the initial point is $(x_o, y_o) = (7,3)$ and the initial decision parameter is

$$p2_0 = f\left(7 + \frac{1}{2}, 2\right) = -151$$

shown in Fig. 2-17

Filled area primitives

- Area is filled with a certain color
- Most often the shape is that of a polygon
- Boundaries are linear as they are polygons
- Standard graphics objects are objects made of a set of polygon surface patches.
- There are two basic approaches to area filling on raster systems.
 - To fill an area is to determine the overlap intervals for scan lines that cross the area.
 - Area filling is to start from a given interior position and paint outward from this point until we encounter the specified boundary conditions.
- The scan-line approach is typically used in general graphics packages to fill polygons, circles, ellipses, and other simple curves.
- All methods starting from an interior point are useful with more complex boundaries and in interactive painting systems.

Scan line polygon fill algorithm

- Figure 2-18 illustrates the scan-line procedure for solid tilling of polygon areas.
- For each scan line crossing a polygon, the area-fill algorithm locates the intersection points of the scan line with the polygon edges.
- These intersection points are then sorted from left to right, and the corresponding frame-buffer positions between each intersection pair are set to the specified fill color.
- In the example of Figure 2-18, the four pixel intersection positions with the polygon boundaries define two stretches of interior pixels from $x=10$ to $x=14$ and from $x=18$ to $x=24$.
- Some scan-line intersections at polygon vertices require special handling. A scan line passing through a vertex intersects two edges at that position, adding two points to the list of intersections for the scan line.
- Figure 2-19 shows two scan lines at positions y and y' that intersect edge endpoints.
- Scan line y intersects five polygon edges. Scan line y' intersects an even number of edges although it also passes through a vertex.
- Intersection points along scan line y' correctly identify the interior pixel spans. But with scan line y , some additional processing is required to determine the correct interior points.

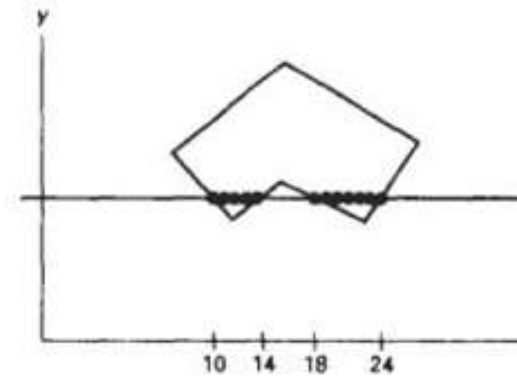


Figure 2-18: Interior pixels along a scan line

- The topological difference between scan line y and scan line y' in Fig. 2-19 is identified by noting the position of the intersecting edges relative to the scan line.
- For scan line y , the two intersecting edges sharing a vertex are on opposite sides of the scan line.
- But for scan line y' , the two intersecting edges are both above the scan line.
- Thus, the vertices that require additional processing are those that have connecting edges on opposite sides of the scan line.
- These vertices are identified by tracing around the polygon boundary either in clockwise or counterclockwise order and observing the relative changes in vertex y coordinates as we move from one edge to the next.
- If the endpoint y values of two consecutive edges monotonically increase or decrease, we need to count the middle vertex as a single intersection point for any scan line passing through that vertex.
- Otherwise, the shared vertex represents a local extremum (minimum or maximum) on the polygon boundary, and the two edge intersections with the scan line passing through that vertex can be added to the intersection list.
- One way to resolve the question as to whether we should count a vertex as one intersection or two is to shorten some polygon edges to split those vertices that should be counted as one intersection.
- We can process nonhorizontal edges around the polygon boundary in the order specified, either clockwise or counter-clockwise.
- As we process each edge, we can check to determine whether that edge and the next nonhorizontal edge have either monotonically increasing or decreasing endpoint y values.
- If so, the lower edge can be shortened to ensure that only one intersection point is generated for the scan line going through the common vertex joining the two edges.
- Figure 2-20 illustrates shortening of an edge.

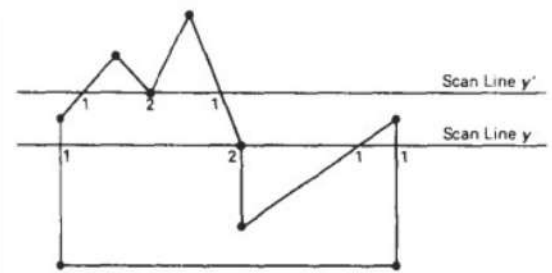


Figure 2-19: Intersection points along scan lines that intersect polygon vertices. Scan line y generates an odd number of intersections, but scan line y' generates an even number of intersections that can be paired to identify

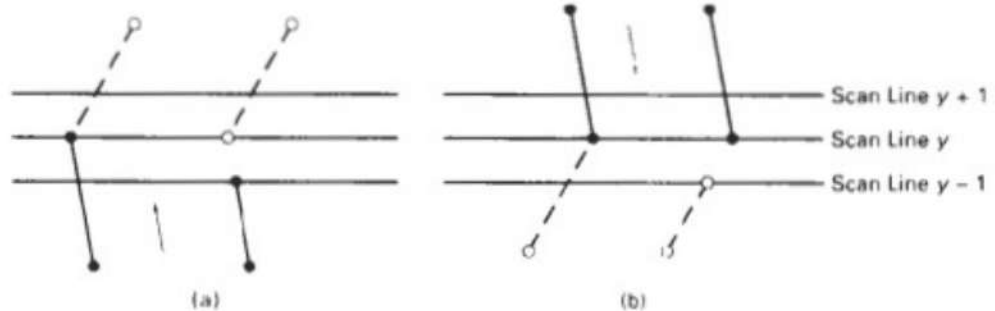


Figure 2-20: Adjusting endpoint y values for a polygon, as we process edges in order around the polygon perimeter. The edge currently being processed is indicated as a solid line. In (a), they coordinate of the upper endpoint of the current edge is decreased by 1. In (b), they coordinate of

- When the endpoint y coordinates of the two edges are increasing, the y value of the upper endpoint for the current edge is decreased by 1, as in Fig. 2-20(a).
- When the endpoint y values are monotonically decreasing, as in Fig. 2-20(b), we decrease the y coordinate of the upper endpoint of the edge following the current edge.
- Calculations performed in scan-conversion and other graphics algorithms typically take advantage of various coherence properties of a scene that is to be displayed.
- Coherence means that the properties of one part of a scene are related in some way to other parts of the scene so that the relation-ship can be used to reduce processing.
- Coherence methods often involve incremental calculations applied along a single scan line or between successive scan lines.
- In determining edge intersections, we can set up incremental coordinate calculations along any edge by exploiting the fact that the slope of the edge is constant from one scan line to the next.
- Figure 2-20 shows two successive scan lines crossing a left edge of a polygon.
- The slope of this polygon boundary line can be expressed in terms of the scan-line intersection coordinates:

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

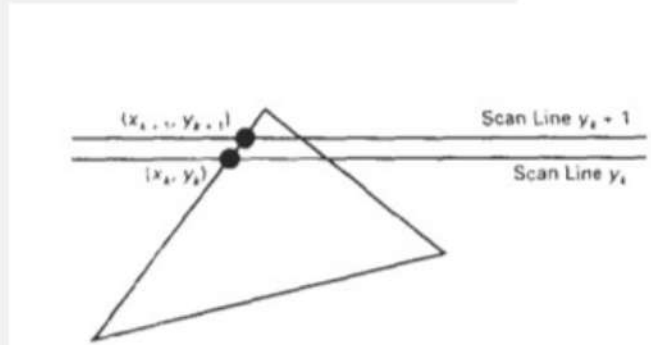


Figure 2-20: Two successive scan lines intersecting a

- Since the change in y coordinates between the two scan lines is simply

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

- the x-intersection value x_{k+1} on the upper scan line can be determined from the x-intersection value x_k on the preceding scan line as

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

- Each successive x intercept can thus be calculated by adding the inverse of the slope and rounding to the nearest integer.
- Parallel implementation of the fill algorithm is to assign each scan line crossing the polygon area to a separate processor.
- Edge-intersection calculations are then performed independently.
- Along an edge with slope m, the intersection x_k value for scan line k above the initial scan line can be calculated as

$$x_k = x_0 + \frac{k}{m}$$

- In a sequential fill algorithm, the increment of x values by the amount $1/m$ along an edge can be accomplished with integer operations

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

- where Δx and Δy are the differences between the edge endpoint x and y coordinate values. Thus, incremental calculations of x intercepts along an edge for successive scan lines can be expressed as

$$x_{k+1} = x_k + \frac{\Delta x}{\Delta y}$$

- Using this equation, integer evaluation of the x intercepts is done by initializing a counter to 0, then incrementing the counter by the value of Δx each time we move up to a new scan line.
- Whenever the counter value becomes equal to or greater than Δy , the current x intersection value is incremented by 1, and decrease the counter by the value Δy .

- To efficiently perform a polygon fill, the polygon boundary are stored in a sorted edge table that contains all the information necessary to process the scan lines efficiently.
- Proceeding around the edges in either a clockwise or a counterclockwise order, sorted on the smallest y value of each edge, in the correct scan-line positions.
- Only nonhorizontal edges are entered into the sorted edge table.
- Each entry in the table for a particular scan line contains the maximum y value for that edge, the x-intercept value (at the lower vertex) for the edge, and the inverse slope of the edge.
- For each scan line, the edges are in sorted order from left to right. Figure 2-21 shows a polygon and the associated sorted edge table.

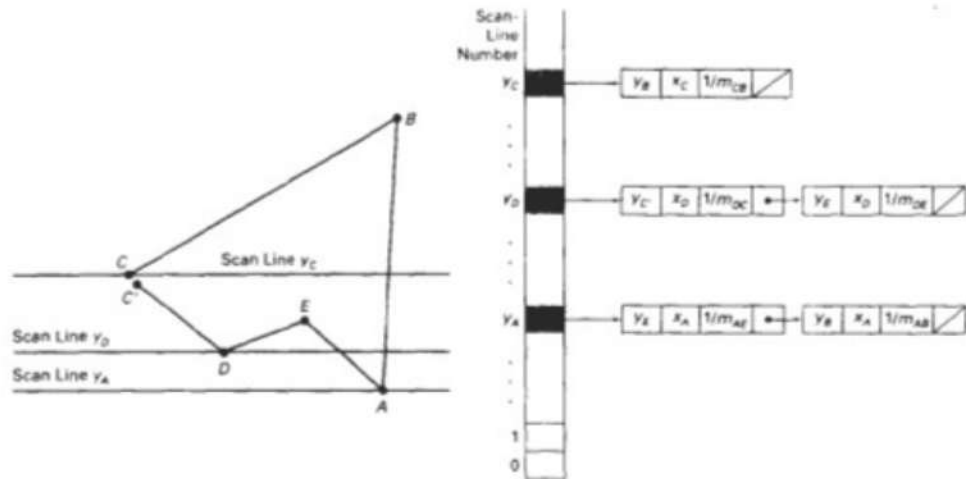


Figure 2-21: A polygon and its sorted edge table, with edge DC shortened by one unit in the y direction

- Next, the scan lines are processed from the bottom of the polygon to its top, producing an active edge list for each scan line crossing the polygon boundaries.
- The active edge list for a scan line contains all edges crossed by that scan line, with iterative coherence calculations used to obtain the edge intersections.
- Implementation of edge-intersection calculations is by storing Δx and Δy values in the sorted edge table.
- For each scan line, the pixel spans are filled for each pair of x-intercepts starting from the leftmost x-intercept value and ending at one position before the rightmost x-intercept.
- And each polygon edge can be shortened by one unit in the y direction at the top endpoint.
- These measures also guarantee that pixels in adjacent polygons will not overlap each other.

Algorithm:

Step 1

Read n , the number of vertices of polygon

Step 2

Read x and y co-ordinates of all vertices in array $x[n]$ and $y[n]$

Step 3

Find $y_{\min}$ and $y_{\max}$

Step 4

Store the initial x value x_1 and x_2 and y value y_1 and y_2 for two endpoints and x increment Δx from scan line to scan line for each edge in the array $edges[][]$. [If $y_1 < y_2$, then interchange y_1 and y_2 and corresponding x_1 and x_2]

Step 5

Sort the rows of array, $edges[][]$ in descending order of y_1 , descending order of y_2 , ascending order of x_2

Step 6

Set $y = y_{\max}$

Step 7

Find the active edges and update active edge list:

```
if ( $y > y_2$  and  $y \leq y_1$ )
    [edge is active]
else
    [edge is not active]
```

Step 8

Compute the x intersects for all active edges for current y value [initially x -intersect is x_1]

$$x_{i+1} \leftarrow x_i + \Delta x$$

Step 9

If x intersect is vertex i.e., $x=x_1$ and $y=y_1$ then apply vertex test to check whether to consider one intersect or two intersects. Store all x intersects in the $x\text{-intersect}[]$ array

Step 10

Sort $x\text{-intersect}[]$ array in ascending order

Step 11

Extract pairs of intersects from the sorted $x\text{-intersect}[]$ array

Step 12

Pass pairs of x values to line drawing routing to draw corresponding line segments

Step 13

Set $y=y-1$

Step 14

Repeat steps 7 through 13 until $y \geq y_{\min}$

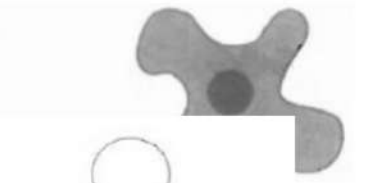
boundary-fill and flood-fill algorithms

Boundary-fill Algorithm

- In Boundary fill algorithm, area-filling is started at a point inside a region and paint the interior towards the boundary
- If the boundary is specified in a single color, the fill algorithm proceeds outward pixel by pixel until the boundary color is encountered
- This method is particularly useful in interactive painting packages, where interior points are easily selected
- Using a graphics tablet or other interactive device, an artist or designer can sketch a figure outline, select a fill color or pattern from a color menu, and pick an interior point
- The system then paints the figure interior
- To display a solid color region (with no border), the designer can choose the fill color to be the same as the boundary color
- A boundary fill procedure accepts as input the coordinates of an interior point (x, y) , a fill color, and a boundary color
- Starting from (x, y) , the procedure tests neighboring positions to determine whether they are of the boundary color. If not, they are painted with the fill color, and their neighbors are tested
- This process continues until all pixels up to the boundary color for the area has been tested
- Both inner and outer boundaries can be set up to specify an area, and some examples of defining regions for boundary fill are shown in Fig. 2-22.
- Figure 2-23 shows two methods for proceeding to neighboring pixels from the current test position.
- In Fig. 2-23(a), four neighboring points are tested. These are the pixel positions that are right, left, above, and below the current pixel. Areas filled by this method are called 4-connected.
- The second method, shown in Fig. 2-23(b), is used to fill more complex figures. Here the set of neighboring positions to be tested includes the four diagonal pixels. Fill methods using this approach are called 8-connected.
-
- An 8-connected boundary-fill algorithm would correctly fill the interior of the area defined in Fig. 2-44, but a 4-connected boundary-fill algorithm produces the partial fill shown.



Figure 2-22: Example



lary-fill

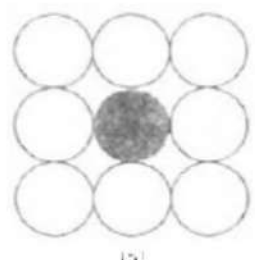
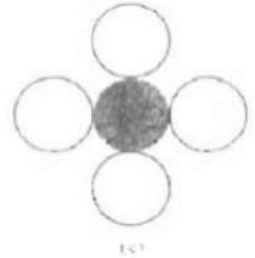


Figure 2-23: Fill methods applied to a 4-connected area (a) and to an 8-connected area (b). Open circles represent pixels to be tested from the current test position, shown as a solid color

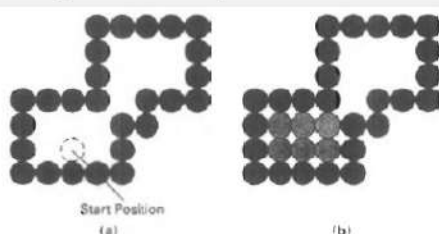


Figure 2-24: The area defined within the color boundary (a) is only partially

Procedure for boundary fill algorithm:

```
void boundaryFill4(int x, int y, int fill, int boundary: integer)
{
    int current;

    current = get pixel(x, y) ;
    if (current != boundary) and (current != fill)
    {
        setColor (fill);

        set pixel(x, y);
        boundaryFill4 (x+1, y, fill, boundary);
        boundaryFill4 (x-1, y, fill, boundary);
        boundaryFill4 (x, y+1, fill, boundary);
        boundaryFill4 (x, y-1, fill, boundary);
    }
} // { end of procedure boundaryFill4}.
```

Flood-fill algorithm

- Flood-Fill Algorithm is used to fill in (or recolor) an area that is not defined within a single color boundary.
- Figure 2-25 shows an area bordered by several different color regions.
- We can paint such areas by replacing a specified interior color instead of searching for a boundary color value.
- We start from a specified interior point (x, y) and reassign all pixel values that are currently set to a given interior color with the desired fill color.
- The process will be a recursive one and stopped when there is no neighboring pixels which can be colored.
- If the area we want to paint has more than one interior color, we can first reassign pixel values so that all interior points have the same color.
- Using either a 4-connected or 8-connected approach, we then step through pixel positions until all interior points have been repainted.

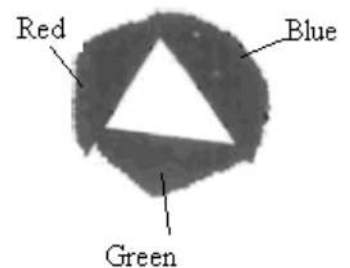


Figure 2-25: An area defined within

Procedure for flood fill algorithm:

```
void floodFill4 (int x, int y, int fillcolor, int oldcolor)
{
```

```
if (getpixel(x, y) == oldcolor)
{
    set pixel(x, y, fill color);
    floodFill4 (x+1, y, fill color, old color);
    floodFill4 (x-1, y, fill color, old color);
    floodFill4 (x, y+1, fill color, old color);
    floodFill4 (x, y-1, fill color, old color);
}
} // {end of procedurefloodFill4}
```

TWO DIMENSIONAL TRANSFORMATIONS

INTRODUCTION

Transformations are a fundamental part of computer graphics. Transformation means changing some graphics into something else by applying rules. Transformations are used to position objects, to shape objects, to change viewing positions, and even to change how something is viewed. There are 5 types of transformations that one can perform in two dimensions.

1. Translation
2. Scaling
3. Rotation
4. Reflection
5. Shearing

1. Translation

A translation is applied to an object by repositioning it along a straight-line path from one coordinate location to another.

We translate a two-dimensional point by adding translation distances, t_x & t_y to the original coordinate position (x, y) to move the point to a new position (x', y') as in the above figure.

$$x' = x + t_x$$

$$y' = y + t_y$$

The translation distance pair (t_x, t_y) is called a **Translation vector** or **Shift vector**.

Sometimes matrix-transformation equations are expressed in terms of coordinate row vectors instead of column vectors. In this case, we would write the matrix representations as $P = [x, y]$ & $T = [t_x, t_y]$

Translation is a Rigid-body transformation that moves object without deformation.

- A straight-line segment is translated by applying the transformation equation (1) to each of the line endpoints and redraws the line between the new endpoint positions.
- Polygons are translated by adding the translating vector to the coordinate position of each vertex and regenerating the polygons using the new set of vertex coordinates.
- If t_x, t_y value is higher than the width value then there will be an error.

2. Rotation

A two-dimensional rotation is applied to an object by repositioning it along a circular path in the xy plane. To generate a rotation, we specify a rotation angle θ (theta) and the position (x_r, y_r) of the rotation point (or pivot point) about which the object is to be rotated as given in the figure.

Positive values for the rotation angle define counterclockwise rotations about the pivot point, as in figure, and negative values rotate objects in the clockwise direction. This transformation can also be described as a rotation about a rotation axis that is perpendicular to the xy plane and passes through the pivot point.

We first determine the transformation equations for rotation of a point position P when the pivot point is at the coordinate origin. The angular and coordinate relationships of the original and transformed point positions are shown in the below figure. In this figure, r is the constant distance of the point from the origin, angle ϕ is the original angular position of the point from the horizontal, and θ is the rotation angle.

we can express the transformed coordinates in terms of angles θ and ϕ as

$$\begin{aligned}x' &= x \cos \theta - y \sin \theta \\y' &= x \sin \theta + y \cos \theta\end{aligned}$$

3. Scaling

- Scaling is used to alter the size of an object
- In the scaling process, you either expand or compress the dimensions of the object. Scaling can be achieved by multiplying the original coordinates of the object with the scaling factor to get the desired result.

Let us assume that the original coordinates are (X, Y) , the scaling factors are (S_x, S_y) , and the produced coordinates are (X', Y') . This can be mathematically represented as shown below,

$$\begin{aligned}x' &= x \cdot s_x \\y' &= y \cdot s_y\end{aligned}\text{-----(1)}$$

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

Matrix Representations and Homogeneous Coordinates

We have seen that basic transformations can be expressed in matrix form. But many graphic applications involve sequences of geometric transformations. Hence, we need a general form of matrix to represent such transformations. This can be expressed as:

$$\mathbf{P}' = \mathbf{M}_1 \cdot \mathbf{P} + \mathbf{M}_2\text{-----(1)}$$

Where P and P' - represent the column vectors.

T_1 - is a 2 by 2 array containing multiplicative factors.

T_2 - is a 2 element row matrix containing translation terms.

For translation, M_1 is the identity matrix.

For rotation or scaling, M_2 contains the translational terms associated with the pivot point or scaling fixed point.

To produce a sequence of transformations with these equations, such as scaling followed by rotation then translation, we must calculate the transformed coordinates one step at a time. First, coordinate positions are scaled, then these scaled coordinates are rotated, and finally the rotated coordinates are translated. A more efficient approach would be to combine the transformations so that the final coordinate positions are obtained directly from the initial coordinates, thereby eliminating the calculation of intermediate coordinate values. To be able to do this, we need to reformulate Eq. (1) to eliminate the matrix addition associated with the translation terms in M_2 .

We can combine multiplicative and translational terms for 2D geometric transformations into a single matrix representation by expanding the 2 by 2 matrix representations to 3 by 3 matrices. This allows us to express all transformation equations as matrix multiplications, providing that we also expand the matrix representations for coordinate positions. To express any 2D transformations as a matrix multiplication, we represent each Cartesian coordinate position (x , y) with the homogeneous coordinate triple (X_h , Y_h , h), such that

$$X = X_h / h, \quad Y = Y_h / h \text{ ----- (2)}$$

For Translation, we have

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

which we can write in the abbreviated form

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

Similarly, **for Rotation transformation**, we have

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

or

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

Finally, **for Scaling transformation**, we have

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

or

$$P' = S(s_x, s_y) \cdot P$$

COMPOSITE TRANSFORMATIONS

A number of transformations or sequence of transformations can be combined into single one called as composition. The resulting matrix is called as composite matrix. The process of combining is called as concatenation.

Translations

If two successive translation vectors (t_{x1} , t_{y1}) and (t_{x2} , t_{y2}) are applied to a coordinate position P, the final transformed location P' is calculated as

$$P' = M_2 \cdot M_1 \cdot P$$

$$P' = M \cdot P$$

- Apply two successive translations, T_1 and T_2

$$T_1 = T(t_{1x}, t_{1y})$$

$$T_2 = T(t_{2x}, t_{2y})$$

$$P' = T_2 \cdot (T_1 \cdot P) = (T_2 \cdot T_1) \cdot P$$

Where P and P' are represented as homogeneous coordinates column vectors. So the composite matrix form of two-dimensional transformation is

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

Rotations

Two successive rotations applied to point P produce the transformed position

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

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

Multiply two rotation matrices to get composite transformation matrix

$$R(\theta_1) \cdot R(\theta_2) = R(\theta_1 + \theta_2)$$

Scaling

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

Concatenating transformation matrices for two successive scaling operations produce the following composite scaling matrix:

$$S(s_{x2}, s_{y2}) \cdot S(s_{x1}, s_{y1}) = S(s_{x1} \cdot s_{x2}, s_{y1} \cdot s_{y2})$$

The resulting matrix in this case indicates that successive scaling operations are multiplicative. That is, if we were to triple the size of an object twice in succession, the final size would be nine times that of the original.

General Pivot-Point Rotation

With a graphics package that only provides a rotate function for revolving objects about the coordinate origin, we can generate rotations about any selected pivot point (x, y) by performing the following sequence of translate-rotate-translate operations:

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

The composite transformation matrix for this sequence is obtained with the concatenation.

$$\begin{bmatrix} 1 & 0 & x_r \\ 0 & 1 & y_r \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & -x_r \\ 0 & 1 & -y_r \\ 0 & 0 & 1 \end{bmatrix} \\ = \begin{bmatrix} \cos \theta & -\sin \theta & x_r(1 - \cos \theta) + y_r \sin \theta \\ \sin \theta & \cos \theta & y_r(1 - \cos \theta) - x_r \sin \theta \\ 0 & 0 & 1 \end{bmatrix}$$

which can be expressed in the form

$$T(x_r, y_r) \cdot R(\theta) \cdot T(-x_r, -y_r) = R(x_r, y_r, \theta)$$

General Fixed-point scaling

Transformation sequence to produce scaling with respect to a selected fixed position (x_f, y_f) using a scaling function that can only scale relative to the coordinate origin.

Step1: The object is kept at desired location as shown in fig (a)

Step2: The object is translated so that its center coincides with origin as shown in fig (b)

Step3: Scaling of object by keeping object at origin is done as shown in fig (c)

Step4: Again translation is done. This translation is called as reverse translation.

Thus transformation is automatically generated on systems that provide a scale function that accepts coordinates for the fixed point.

General scaling directions

Parameters s_x and s_y scale objects along the x and y directions. We can scale an object in other directions by rotating the object to align the desired scaling directions with the coordinate axes before applying the scaling transformation.

To accomplish the scaling without changing the orientation of the object, we first perform a rotation so that the directions for s_1 and s_2 coincide with the x and y axes, respectively. Then the scaling transformation is applied, followed by an opposite rotation to return points to their original orientations. The composite matrix resulting from the product of these three transformations is

$$\mathbf{R}^{-1}(\theta) \cdot \mathbf{S}(s_1, s_2) \cdot \mathbf{R}(\theta) = \begin{bmatrix} s_1 \cos^2 \theta + s_2 \sin^2 \theta & (s_2 - s_1) \cos \theta \sin \theta & 0 \\ (s_2 - s_1) \cos \theta \sin \theta & s_1 \sin^2 \theta + s_2 \cos^2 \theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

As an example of this scaling transformation, we turn a unit square into a parallelogram as in the below figure by stretching it along the diagonal from (0, 0) to (1, 1). We rotate the diagonal onto the y axis and double its length with the transformation parameters $\theta = 45^\circ$, $s_1 = 1$, and $s_2 = 2$.

OTHER TRANSFORMATIONS

1. Reflection

A reflection is a transformation that produces a mirror image of an object. The mirror image for 2-D reflection is generated relative to an axis of reflection by rotating the object 180° about the reflection axis.

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

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

(ii) If we flip both x and y coordinates of a point by reflecting relative to an axis that is perpendicular to the xy plane and that through the coordinate origin. Matrix representation is

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

(iv) Reflection axis as the diagonal line $y=x$ the reflection matrix is

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

(v) To obtain a transformation matrix for reflection about the diagonal $y = -x$, we could concatenate matrices for the transformation sequence:

- (1) clockwise rotation by 45° angle
- (2) reflection about the y axis, and
- (3) counter clockwise rotation by 45° angle.

The resulting transformation matrix is

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

2. Shear

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

1) A x-direction shear relative to the x-axis

$$\begin{bmatrix} 1 & sh_x & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \dots\dots\dots (1)$$

Which transforms coordinate positions as

$$\begin{aligned} x' &= x + sh_x \cdot y \\ y' &= y \dots\dots\dots(2) \end{aligned}$$

Any real number can be assigned to the shear parameter sh_x . A coordinate position (x, y) is then shifted horizontally by an amount proportional to its distance (y value) from the x axis ($y = 0$). Setting sh_x to 2, for example, changes the square in figure into a parallelogram. Negative values for sh_x shift coordinate positions to the left.

with coordinate positions transformed as

$$x' = x + Sh_x (y - y_{ref})$$

$$y' = y$$

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

$$\begin{bmatrix} 1 & 0 & 0 \\ sh_y & 1 & -sh_y \cdot x_{\text{ref}} \\ 0 & 0 & 1 \end{bmatrix} \dots\dots\dots (5)$$

Two-Dimensional Viewing

The Viewing Pipeline

A world-coordinate area selected for display is called a **window**. An area on a display device to which a window is mapped is called a **viewport**. The window defines what is to be viewed; the viewport defines where it is to be displayed.

Often, windows and viewports are rectangles in standard position, with the rectangle edges parallel to the coordinate axes. Other window or viewport geometries, such as general polygon shapes and circles, are used in some applications, but these shapes take longer to process. In general, the mapping of a part of a world-coordinate scene to device coordinates is referred to as a **viewing transformation**.

Viewing transformation in several steps:

First, we construct the scene in world coordinate using the output primitives and attributes. To obtain a particular orientation, we can set up a 2-D viewing coordinate system in the window coordinate plane and define a window in viewing coordinates system.

Once the viewing frame is established, we then transform description in world coordinates to viewing coordinates. Then, we define viewport in normalized coordinates (range from 0 to 1) and map the viewing coordinates description of the scene to normalized coordinates. At the final step, all parts of the picture that (i.e., outside the viewport are clipped, and the contents are transferred to device coordinates).

By changing the position of the viewport, we can view objects at different locations on the display area of an output device as shown in figure.

Also, by varying the size of viewports, we can change the size and proportions of displayed objects. We can achieve zooming effects by successively mapping different-sized windows on a fixed-size viewport. As the windows are made smaller, we zoom in on some part of a scene to view details that are not shown with larger windows.

WINDOW-TO-VIEWPORT COORDINATE TRANSFORMATION

Once object description has been transmitted to the viewing reference frame, we choose the window extends in viewing coordinates and selects the viewport limits in normalized coordinates. Object descriptions are then transferred to normalized device coordinates. We do this thing using a transformation that maintains the same relative placement of an object in normalized space as they had in viewing coordinates. If a coordinate position is at the center of the viewing window: It will display at the center of the viewport.

Figure shows the window to viewport mapping. A point at position (x_w, y_w) in window mapped into position (x_v, y_v) in the associated viewport. To maintain the same relative placement in the viewport as in the window, we require that

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

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

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

where the scaling factors are

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

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

Equations can also be derived with a set of transformations that converts the window area into the viewport area. The conversion is performed with the following sequence of transformations.

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

Relative proportions of objects are maintained if scaling factor are the same ($Sx = Sy$). Otherwise world objects will be stretched or contracted in either the x or y direction when displayed on output device.

For normalized coordinates, object descriptions are mapped to various display devices. Any number of output devices can be open in particular application and another window view port transformation can be performed for each open output device. This mapping called the work station transformation is accomplished by selecting a window area in normalized space and a view port are in coordinates of display device.

Clipping

Any procedure that identifies those portions of a picture that are either inside or outside of a specified region of space is referred to as a **clipping algorithm, or simply clipping**. The region against which an object is to clipped is called a **clip window**.

Line Clipping

In line clipping, we will cut the portion of line which is outside of window and keep only the portion that is inside the window.

Cohen Sutherland Algorithm is a line clipping algorithm that cuts lines to portions which are within a rectangular area. It eliminates the lines from a given set of lines and rectangle area of interest (view port) which belongs outside the area of interest and clip those lines which are partially inside the area of interest.

Cohen-Sutherland Line Clipping

1. Visible: If a line lies within the window, i.e., both endpoints of the line lies within the window. A line is visible and will be displayed as it is.

2. Not Visible: If a line lies outside the window it will be invisible and rejected. Such lines will not display. If any one of the following inequalities is satisfied, then the line is considered invisible.

x_{min} , x_{max} are coordinates of the window.

y_{min} , y_{max} are also coordinates of the window.

3. Clipping Case: If the line is neither visible case nor invisible case. It is considered to be clipped case. First of all, the category of a line is found based on nine regions given below. All nine regions are assigned codes. Each code is of 4 bits. If both endpoints of the line have end bits zero, then the line is considered to be visible. The center area is having the code, 0000, i.e., region 5 is considered a rectangle window.

The algorithm divides a two-dimensional space into nine regions and then efficiently determines the lines and portions of lines that are visible in the central region of interest (the viewport).

Four Bit code is calculated by comparing extreme end point of given line (x,y) by four co-ordinates x_{min} , x_{max} , y_{min} , y_{max} which are the coordinates of the area of interest (0000).

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, or bottom.

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

Bit 1: left

Bit 2: right

Bit 3: below

Bit 4: above

A value of 1 in any bit position indicated that the point is in that relative position, otherwise, the bit position is set to 0. If a point is within the clipping rectangle, the region code is 0000. A point that is below & to the left of the rectangle has a region code of 0101.

Bit values in the region code are determined by comparing endpoint coordinate values (x, y) to the clip boundaries. Bit 1 is set 1 if $x < x_{min}$. The other three bit values can be determined using similar comparisons.

Once we have established region codes for all line endpoints, we can quickly determine which lines are completely inside the clip window & which are clearly outside. Any lines that are completely contained within the window boundaries have a region code of 0000 for both endpoints.

Calculate the four-bit code as follows:

- Set First Bit if 1 Points lies to left of window ($x < x_{\min}$)
- Set Second Bit if 1 Points lies to right of window ($x > x_{\max}$)
- Set Third Bit if 1 Points lies to bottom of window ($y < y_{\min}$)
- Set Fourth Bit if 1 Points lies to top of window ($y > y_{\max}$)

There are 3 possibilities for the line:

- Line can be completely inside the window (This line should be accepted).
- Line can be completely outside of the window (This line will be completely removed from the region).
- Line can be partially inside the window (We will find intersection point and draw only that portion of line that is inside region).

The more efficient Cohen-Sutherland Algorithm performs initial tests on a line to determine whether intersection calculations can be avoided.

Midpoint Subdivision Line Clipping Algorithm:

The Sutherland Cohen subdivision line clipping algorithm requires the calculation of the intersection of the line with the window edge. These calculations can be avoided by repetitively subdividing the line at its midpoint.

Midpoint Subdivision Algorithm :

1. Read two endpoints of the line say $P_1(x_1, y_1)$ and $P_2(x_2, y_2)$.
2. Read two corners (left-top and right-bottom) of the window, say (Wx_1, Wy_1) and (Wx_2, Wy_2) .
3. Assign region codes for two end points using following steps :
 - Initialize code with bits 0000
 - Set Bit 1 - if ($x < Wx_1$)
 - Set Bit 2 - if ($x > Wx_2$)
 - Set Bit 3 - if ($y < Wy_1$)
 - Set Bit 4 - if ($y > Wy_2$)
4. Check for visibility of line
 - a) If region codes for both endpoints are zero then the line is completely visible. Hence draw the line and go to step 6.
 - b) If region codes for endpoints are not zero and the logical ANDing of them is also nonzero then the line is completely invisible, so reject the line and go to step 6.
 - c) If region codes for two endpoints do not satisfy the conditions in 4a) and 4b) the line is partially visible.
5. Divide the partially visible line segment in equal parts and repeat steps 3 through 5 for both subdivided line segments until you get completely visible and completely invisible line segments.
6. Stop.

UNIT-III

Two Dimensional Viewing

Two-dimensional viewing is the mechanism for displaying views of a picture on an output device. 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. For a 2-D picture, a view is selected by specifying a subarea of the total picture area.

The Viewing Pipeline

- **Window:** Area selected in world-coordinate for display is called window. It defines what is to be viewed.
- **Viewport:** Area on a display device in which window image is display (mapped) is called viewport. It defines where to display.
- In many case window and viewport are rectangle, also other shape may be used as window and viewport.
- In general finding device coordinates of viewport from word coordinates of window is called as **viewing transformation**.
- Sometimes we consider this viewing transformation as window-to-viewport transformation but in general it involves more steps.

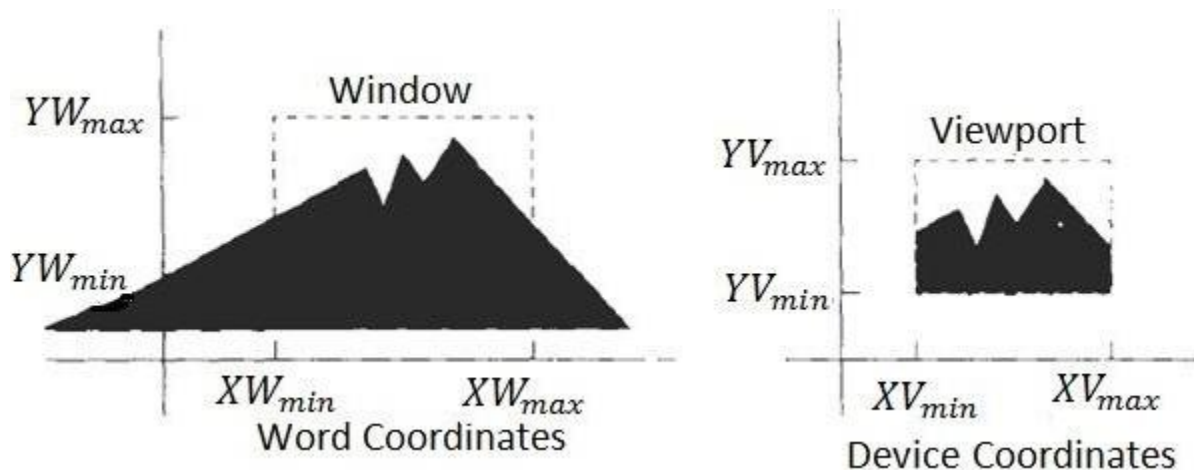


Fig. 3.1: - A viewing transformation using standard rectangles for the window and viewport.

- Now we see steps involved in viewing pipeline.

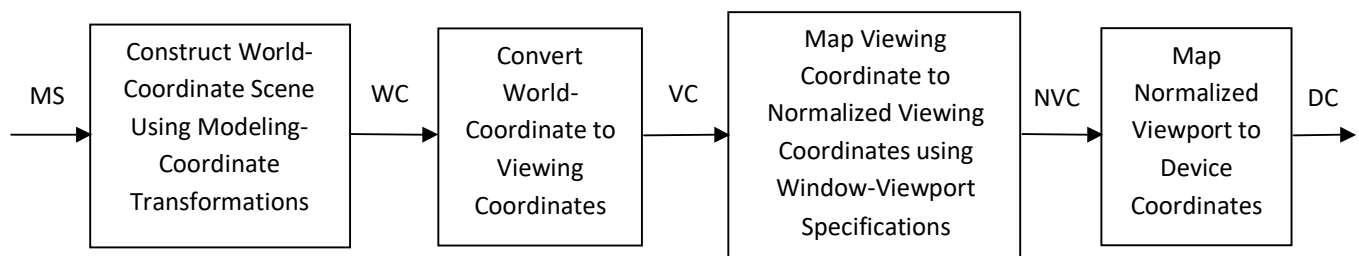


Fig. 3.2: - 2D viewing pipeline.

- As shown in figure above first of all we construct world coordinate scene using modeling coordinate transformation.
- After this we convert viewing coordinates from world coordinates using window to viewport transformation.
- Then we map viewing coordinate to normalized viewing coordinate in which we obtain values in between 0 to 1.
- At last we convert normalized viewing coordinate to device coordinate using device driver software which provide device specification.
- Finally device coordinate is used to display image on display screen.
- By changing the viewport position on screen we can see image at different place on the screen.

- By changing the size of the window and viewport we can obtain zoom in and zoom out effect as per requirement.
- Fixed size viewport and small size window gives zoom in effect, and fixed size viewport and larger window gives zoom out effect.
- View ports are generally defines with the unit square so that graphics package are more device independent which we call as normalized viewing coordinate.

Viewing Coordinate Reference Frame

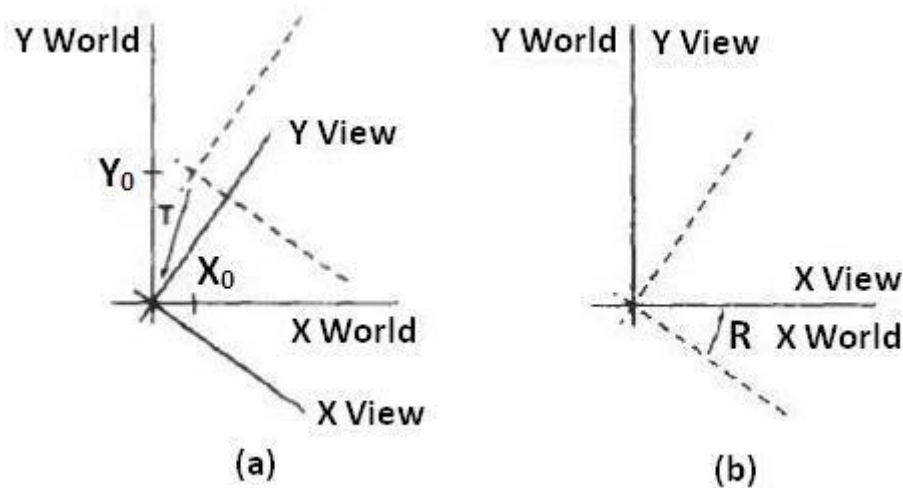


Fig. 3.3: - A viewing-coordinate frame is moved into coincidence with the world frame in two steps: (a) translate the viewing origin to the world origin, and then (b) rotate to align the axes of the two systems.

- We can obtain reference frame in any direction and at any position.
- For handling such condition first of all we translate reference frame origin to standard reference frame origin and then we rotate it to align it to standard axis.
- In this way we can adjust window in any reference frame.
- this is illustrate by following transformation matrix:

$$M_{wc,vc} = RT$$

- Where T is translation matrix and R is rotation matrix.

Window-To-Viewport Coordinate Transformation

- Mapping of window coordinate to viewport is called window to viewport transformation.
- We do this using transformation that maintains relative position of window coordinate into viewport.
- That means center coordinates in window must be remains at center position in viewport.
- We find relative position by equation as follow:

$$\frac{x_v - x_{vmin}}{x_{vmax} - x_{vmin}} = \frac{x_w - x_{wmin}}{x_{wmax} - x_{wmin}}$$

$$\frac{y_v - y_{vmin}}{y_{vmax} - y_{vmin}} = \frac{y_w - y_{wmin}}{y_{wmax} - y_{wmin}}$$

- Solving by making viewport position as subject we obtain:

$$x_v = x_{vmin} + (x_w - x_{wmin})s_x$$

$$y_v = y_{vmin} + (y_w - y_{wmin})S_y$$

- Where scaling factor are :

$$S_x = \frac{x_{vmax} - x_{vmin}}{x_{wmax} - x_{wmin}}$$

$$S_y = \frac{y_{vmax} - y_{vmin}}{y_{wmax} - y_{wmin}}$$

- We can also map window to viewport with the set of transformation, which include following sequence of transformations:
 1. Perform a scaling transformation using a fixed-point position of (x_{wmin}, y_{wmin}) that scales the window area to the size of the viewport.
 2. Translate the scaled window area to the position of the viewport.
- For maintaining relative proportions we take $(s_x = s_y)$. in case if both are not equal then we get stretched or contracted in either the x or y direction when displayed on the output device.
- Characters are handle in two different way one way is simply maintain relative position like other primitive and other is to maintain standard character size even though viewport size is enlarged or reduce.
- Number of display device can be used in application and for each we can use different window-to-viewport transformation. This mapping is called the **workstation transformation**.

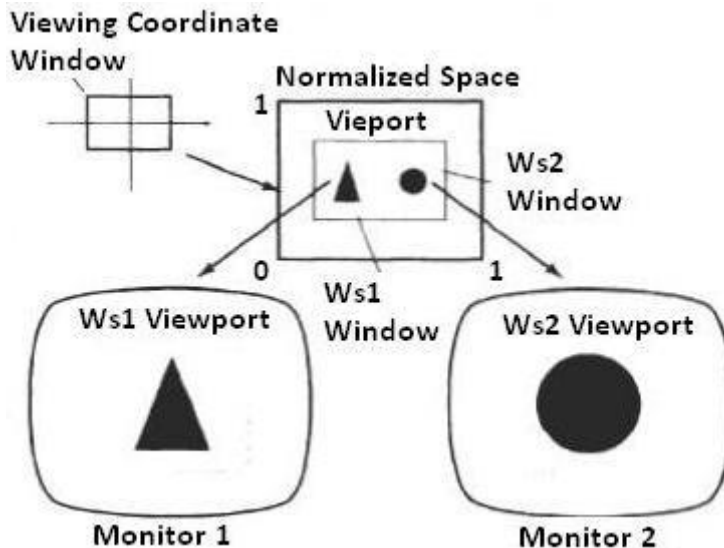


Fig. 3.4: - workstation transformation.

- As shown in figure two different displays devices are used and we map different window-to-viewport on each one.

Clipping Operations

- Generally, any procedure that identifies those portions of a picture that are either inside or outside of a specified region of space is referred to as a **clipping algorithm**, or simply **clipping**. The region against which an object is to clip is called a **clip window**.
- Clip window can be general polygon or it can be curved boundary.

Application of Clipping

- It can be used for displaying particular part of the picture on display screen.

- Identifying visible surface in 3D views.
- Antialiasing.
- Creating objects using solid-modeling procedures.
- Displaying multiple windows on same screen.
- Drawing and painting.

Point Clipping

- In point clipping we eliminate those points which are outside the clipping window and draw points which are inside the clipping window.
- Here we consider clipping window is rectangular boundary with edge $(x_{wmin}, x_{wmax}, y_{wmin}, y_{wmax})$.
- So for finding whether given point is inside or outside the clipping window we use following inequality:

$$x_{wmin} \leq x \leq x_{wmax}$$

$$y_{wmin} \leq y \leq y_{wmax}$$

- If above both inequality is satisfied then the point is inside otherwise the point is outside the clipping window.

Line Clipping

- Line clipping involves several parts. First case is completely inside second case is completely outside and third case include partially inside or partially outside the clipping window.

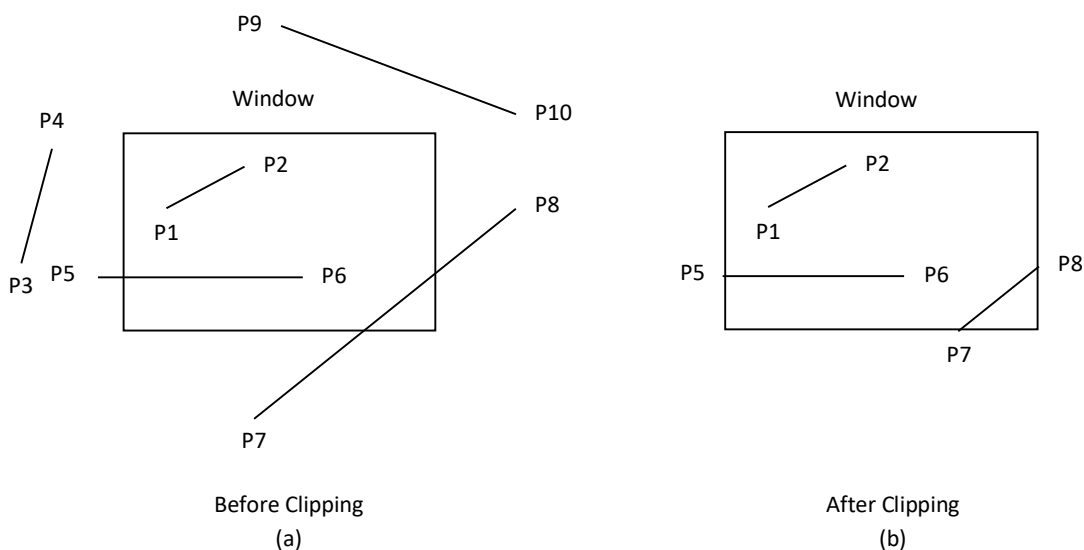


Fig. 3.5: - Line clipping against a rectangular window.

- Line which is completely inside is display completely. Line which is completely outside is eliminated from display. And for partially inside line we need to calculate intersection with window boundary and find which part is inside the clipping boundary and which part is eliminated.
- For line clipping several scientists tried different methods to solve this clipping procedure. Some of them are discuss below.

Cohen-Sutherland Line Clipping

- This is one of the oldest and most popular line-clipping procedures.

- In this we divide whole space into nine region and assign 4 bit code to each endpoint of line depending on the position where the line endpoint is located.

1001	1000	1010
0001	0000	0010
0101	0100	0110

Fig. 3.6: - Workstation transformation.

- Figure 3.6 shows code for line end point which is fall within particular area.
- Code is deriving by setting particular bit according to position of area.
Set bit 1: For left side of clipping window.
Set bit 2: For right side of clipping window.
Set bit 3: For below clipping window.
Set bit 4: For above clipping window.
- All bits as mention above are set means 1 and other are 0.
- After assigning code we check both end points code if both endpoint have code 0000 then line is completely inside.
- If both are not 0000 then we perform logical ending between this two code and check the result if result is non zero line is completely outside the clipping window.
- Otherwise we calculate the intersection point with the boundary one by one and divide the line into two parts from intersection point and perform same procedure for both line segments until we get all segments completely inside or completely outside.
- Draw line segment which are completely inside and eliminate other line segment which found completely outside.
- Now we discuss how to calculate intersection points with boundary.
- For intersection calculation we use line equation " $y = mx + b$ ".
- For left and right boundary x coordinate is constant for left " $x = x_{wmin}$ " and for right " $x = x_{wmax}$ ".
- We calculate y coordinate of intersection for this boundary by putting values of x as follow:

$$y = y_1 + m(x - x_1)$$

- Similarly fro top and bottom boundary y coordinate is constant for top " $y = y_{wmax}$ " and for bottom " $y = y_{wmin}$ ".
- We calculate x coordinate of intersection for this boundary by putting values of y as follow:

$$x = x_1 + \frac{y - y_1}{m}$$

Polygon Clipping

- For polygon clipping we need to modify the line clipping procedure because in line clipping we need to consider about only line segment while in polygon clipping we need to consider the area and the new boundary of the polygon after clipping.

Sutherland-Hodgeman Polygon Clipping

- For correctly clip a polygon we process the polygon boundary as a whole against each window edge.
- This is done by whole polygon vertices against each clip rectangle boundary one by one.
- Beginning with the initial set of polygon vertices we first clip against the left boundary and produce new sequence of vertices.
- Then that new set of vertices is clipped against the right boundary clipper, a bottom boundary clipper and a top boundary clipper, as shown in figure below.

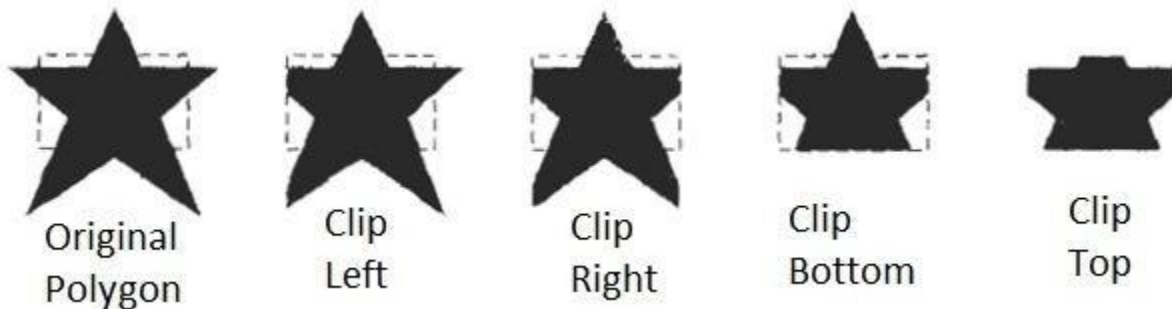


Fig. 3.11: - Clipping a polygon against successive window boundaries.

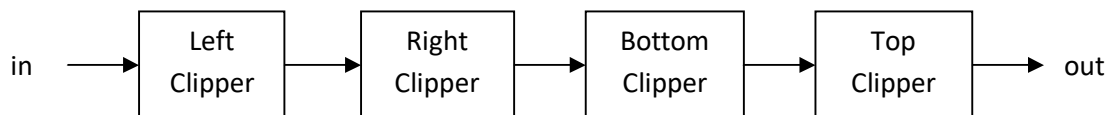


Fig. 3.12: - Processing the vertices of the polygon through boundary clipper.

- There are four possible cases when processing vertices in sequence around the perimeter of a polygon.

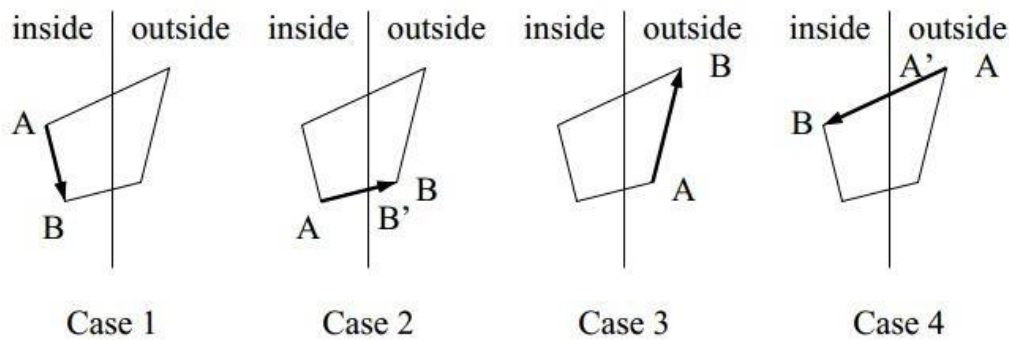


Fig. 3.13: - Clipping a polygon against successive window boundaries.

- As shown in case 1: if both vertices are inside the window we add only second vertices to output list.
- In case 2: if first vertices is inside the boundary and second vertices is outside the boundary only the edge intersection with the window boundary is added to the output vertex list.
- In case 3: if both vertices are outside the window boundary nothing is added to window boundary.
- In case 4: first vertex is outside and second vertex is inside the boundary, then adds both intersection point with window boundary, and second vertex to the output list.
- When polygon clipping is done against one boundary then we clip against next window boundary.
- We illustrate this method by simple example.

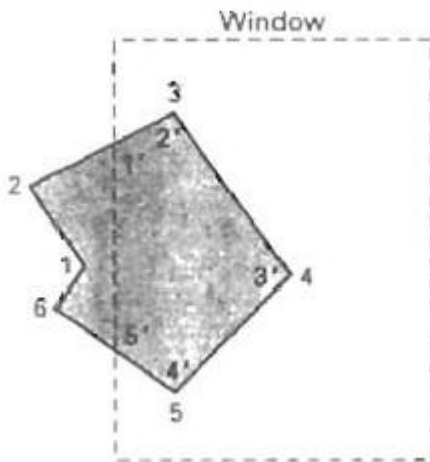
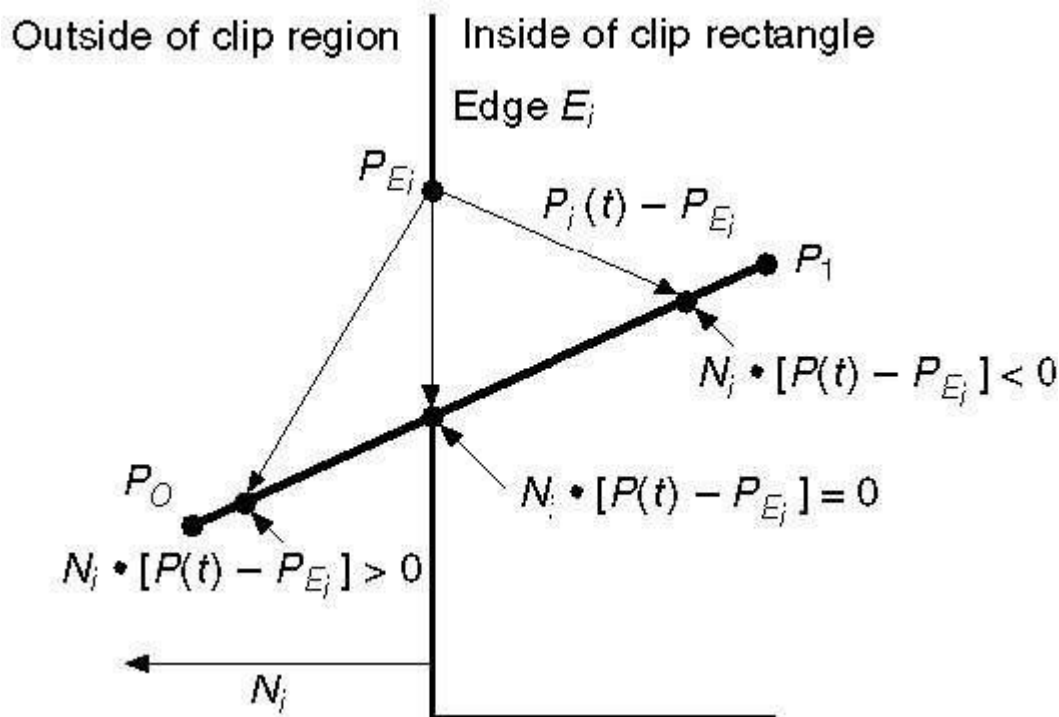


Fig. 3.14: - Clipping a polygon against left window boundaries.

- As shown in figure above we clip against left boundary vertices 1 and 2 are found to be on the outside of the boundary. Then we move to vertex 3, which is inside, we calculate the intersection and add both intersection point and vertex 3 to output list.
- Then we move to vertex 4 in which vertex 3 and 4 both are inside so we add vertex 4 to output list, similarly from 4 to 5 we add 5 to output list, then from 5 to 6 we move inside to outside so we add intersection pint to output list and finally 6 to 1 both vertex are outside the window so we does not add anything.
- Convex polygons are correctly clipped by the Sutherland-Hodgeman algorithm but concave polygons may be displayed with extraneous lines.

Cyrus-Beck Line Clipping Algorithm

This algorithm is more efficient than Cohen-Sutherland algorithm. It employs parametric line representation and simple dot products.



Parametric equation of line is –

$$P_0P_1: P(t) = P_0 + t(P_1 - P_0)$$

Let N_i be the outward normal edge E_i . Now pick any arbitrary point P_{E_i} on edge E_i then the dot product $N_i \cdot [P_t - P_{E_i}]$ determines whether the point P_t is “inside the clip edge” or “outside” the clip edge or “on” the clip edge.

The point P_t is inside if $N_i \cdot [P_t - P_{E_i}] < 0$

The point P_t is outside if $N_i \cdot [P_t - P_{E_i}] > 0$

The point P_t is on the edge if $N_i \cdot [P_t - P_{E_i}] = 0$ Intersection point

$$\begin{aligned} N_i \cdot [P_t - P_{E_i}] &= 0 \\ N_i \cdot [P_0 + t(P_1 - P_0) - P_{E_i}] &= 0 \text{ Replacing } P(t) \text{ with } P_0 + t(P_1 - P_0) \\ N_i \cdot [P_0 - P_{E_i}] + N_i \cdot t[P_1 - P_0] &= 0 \\ N_i \cdot [P_0 - P_{E_i}] + N_i \cdot tD &= 0 \text{ (substituting } D \text{ for } [P_1 - P_0]) \\ N_i \cdot [P_0 - P_{E_i}] &= -N_i \cdot tD \end{aligned}$$

The equation for t becomes,

$$t = \frac{N_i \cdot [P_0 - P_{E_i}]}{N_i \cdot D}$$

It is valid for the following conditions –

- $N_i \neq 0$ error cannot happen $D \neq 0$

UNIT-IV

3D Object Representations

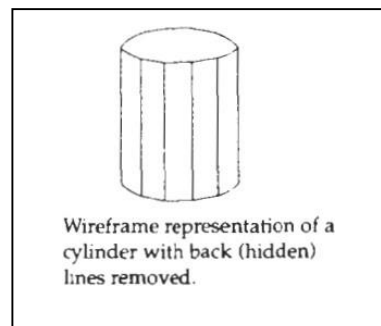
Introduction

Graphics scenes can contain many different kinds of objects; trees, flowers, clouds, water bricks, wood paneling, rubber, paper, marble, steel, glass plastic and cloth. So it is probably not too surprising that there is no one method that we can use to describe objects that will include all characteristics of these materials. And to produce realistic displays of scenes, we need to use representations that accurately model object characteristics.

For representing 3-D surfaces following three most common techniques are as follows:- polygon mesh surfaces, parametric surfaces and quadric surfaces. Parametric surfaces are generalization of the curves.

Polygon surfaces

A polygon mesh is a set of connected polygonally bounded planar surfaces. Open boxes, cabinets and building exteriors can be easily and naturally represented by polygon meshes. A polygon mesh is a collection of edges, vertices and polygons connected such that each edge is shared by at most two polygons. An edge connects two vertices and a polygon is a closed sequence of edges. An edge can be shared by two adjacent polygons and a vertex is shared by at least two edges. A polygon mesh can be represented in several different ways, each with its advantages and disadvantages. Two basic criteria, space and time, can be used to evaluate different representation.

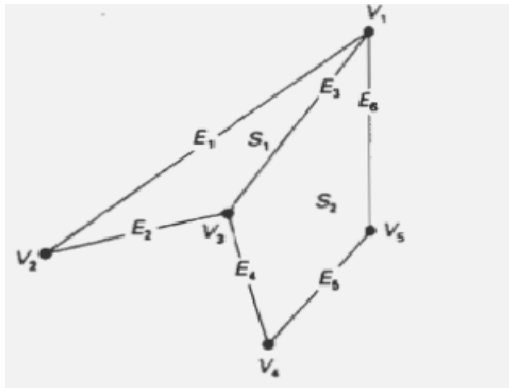


Polygon mesh Representation

We specify a polygon surface with a set of vertex coordinates and associated attribute parameters. As information for each polygon is input, the data are placed into tables that are to be used in the subsequent processing, display, and manipulation of the objects in a scene. Polygon data tables can be organized into two groups: **geometric tables** and **attribute tables**. Geometric data tables contain vertex coordinates and parameters to identify the spatial orientation of the polygon surfaces. Attribute information for an object includes parameters specifying the

degree of transparency of the object and its surface reflectivity and texture characteristics.

A convenient organization for storing geometric data is to create three lists: **a vertex table, an edge table, and a polygon table.** Coordinate values for each vertex in the object are stored in the vertex table. The edge table contains pointers back into the vertex table to identify the vertices for each polygon edge. And the polygon table contains pointers back into the edge table to identify the edges for each polygon.



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

Geometric data table representation for two adjacent polygon surfaces, formed with six edges and five vertices.

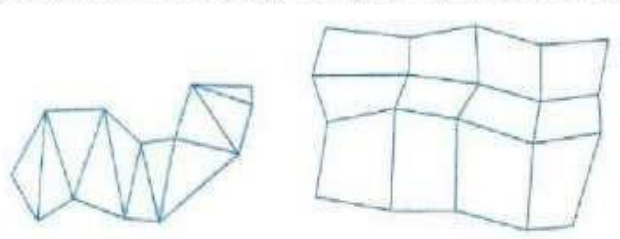
Listing the geometric data in three tables provides a convenient reference to the individual components (vertices, edges, and polygons) of each object. Also, the object can be displayed efficiently by using data from the edge table to draw the component lines. An alternative arrangement to use just two tables: a vertex table and a polygon table. But this scheme is less convenient, and some edges could get drawn twice. Another possibility is to use only a polygon table, but this duplicate coordinate information, since explicit coordinate values are listed for each vertex in each polygon. Also edge Information would have to **be** reconstructed from the vertex listings in the polygon table.

Some consistency checks of the geometric data table:

- Every vertex is listed as an endpoint for at least 2 edges
- Every edge is part of at least one polygon
- Every polygon is closed
- Each polygon has at least one shared edge
- If the edge table contains pointers to polygon every edge referenced by a polygon pointer has a reciprocal back to the polygon.

Types of polygon mesh

Common types of polygon meshes are triangle strip and quadrilateral mesh.



Fast hardware-implemented polygon renderers are capable of displaying up to 1,000,000 or more shaded triangles per second, including the application of surface texture and special lighting effects.

Plane Equations:-

To produce a display of a three-dimensional object, we must process the input data representation for the object through several procedures. These processing steps include transformation of the modeling and world-coordinate descriptions to viewing coordinates, then to device coordinates; identification of visible surfaces; and the application of surface-rendering procedures. For some of these processes, we need information about the spatial orientation of the individual surface components of the object. This information is obtained from the vertex coordinate values and the equations that describes the polygon planes. The equations for a plane surface can be expressed in the form.

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

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. For this purpose, we can select three

successive polygon vertices, (x_1, y_1, z_1) , (x_2, y_2, z_2) and (x_3, y_3, z_3) and solve the following set of equations for the ratio A/D , B/D and C/D .

$$\left(\frac{A}{D}\right)x_k + \left(\frac{B}{D}\right)y_k + \left(\frac{C}{D}\right)z_k = -1 \quad k=1,2,3$$

The solution for this set of equations can be obtained in determinant form using cramer's rule, as

$$A = \begin{vmatrix} 1 & y_1 & z_1 \\ 1 & y_2 & z_2 \\ 1 & y_3 & z_3 \end{vmatrix} \quad B = \begin{vmatrix} x_1 & 1 & z_1 \\ x_2 & 1 & z_2 \\ x_3 & 1 & z_3 \end{vmatrix} \quad C = \begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix} \quad D = - \begin{vmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ x_3 & y_3 & z_3 \end{vmatrix}$$

Expanding the determinant we can write the calculation for the plane coefficient in the form

$$A = y_1(z_2 - z_3) + y_2(z_3 - z_1) + y_3(z_1 - z_2)$$

$$B = z_1(x_2 - x_3) + z_2(x_3 - x_1) + z_3(x_1 - x_2)$$

$$C = x_1(y_2 - y_3) + x_2(y_3 - y_1) + x_3(y_1 - y_2)$$

$$D = -x_1(y_2 z_3 - y_3 z_2) - x_2(y_3 z_1 - y_1 z_3) - x_3(y_1 z_2 - y_2 z_1)$$

Then, the plane equation at the form: $Ax + By + Cz + D = 0$ has the property that:

If we substitute any arbitrary point (x, y) into this equation, then,

$Ax + By + Cz + D < 0$ implies that the point (x, y) is inside the surface, and

$Ax + By + Cz + D > 0$ implies that the point (x, y) is outside the surface.

Polygon Meshes

A polygon mesh is a collection of vertices, edges and faces that defines the shape of a polyhedral object in 3D computer graphics and solid modeling. The faces usually consist of triangles (triangle mesh), quadrilaterals, or other simple convex polygons, since this simplifies rendering, but may also be composed of more general concave polygons, or polygons with holes.

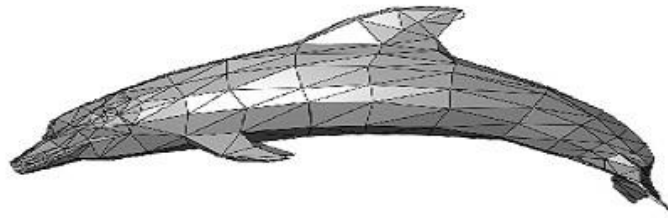


Fig: A Dolphin formed by mesh of triangles

The study of polygon meshes is a large sub-field of computer graphics and geometric modeling. Different representations of polygon meshes are used for different applications and goals.

Polygon meshes may be represented in a variety of ways, using different methods to store the vertex, edge and face data. These include:

1. Face-vertex meshes: A simple list of vertices, and a set of polygons that point to the vertices it uses.
2. Winged-edge meshes, in which each edge points to two vertices, two faces, and the four (clockwise and counterclockwise) edges that touch it. Winged-edge meshes allow constant time traversal of the surface, but with higher storage requirements.
3. Half-edge meshes: Similar to winged-edge meshes except that only half the edge traversal information is used.
4. Quad-edge meshes, which store edges, half-edges, and vertices without any reference to polygons. The polygons are implicit in the representation, and may be found by traversing the structure. Memory requirements are similar to half-edge meshes.
5. Corner-tables, which store vertices in a predefined table, such that traversing the table implicitly defines polygons. This is in essence the triangle fan used in hardware graphics rendering. The representation is more compact, and more efficient to retrieve polygons, but operations to change polygons are slow. Furthermore, corner-tables do not represent meshes completely. Multiple corner-tables (triangle fans) are needed to represent most meshes.
6. Vertex-vertex meshes: A "VV" mesh represents only vertices, which point to other vertices. Both the edge and face information is implicit in the representation. However, the simplicity of the representation does not allow for many efficient operations to be performed on meshes.

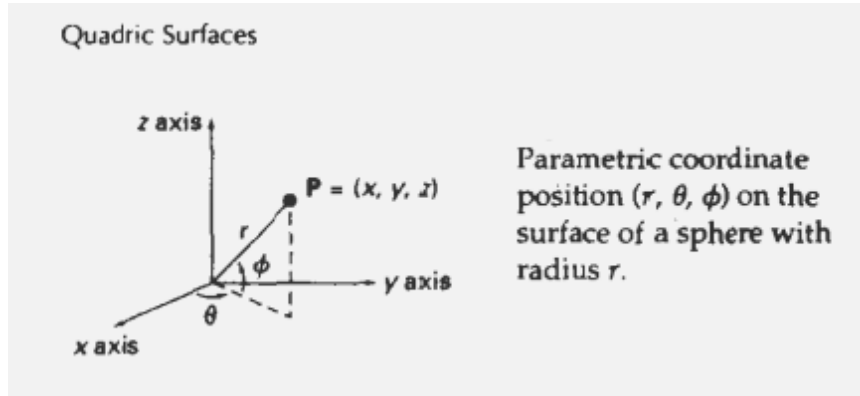
Polyline- a polyline is a chain of connected line segments. It is specified by giving the vertices defining the line segments. The

Curve line and surfaces

Polylines are polygons are first degree, piecewise linear approximations to curves and surfaces. Unless the curves or surfaces being approximated are also piecewise linear, large number of end joint coordinates must be created and stored to achieve accuracy. Interactive manipulation of the data to approximate a shape is tedious, because many points have to be positioned precisely.

Quadric surfaces

A frequently used class of objects are the quadric surfaces, which are described with second-degree equations (quadratics). They include spheres, ellipsoids, tori, paraboloids, and hyperboloids. Quadric surfaces, particularly spheres and ellipsoids, are common elements of graphic scenes and they are often available in graphic packages as primitives from which more complex objects can be constructed.



Sphere

In Cartesian coordinates, a spherical surface with radius r centered on the coordinate origin is defined as the set of points (x, y, z) that satisfy the equation.

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

We can also describe the spherical surface in parametric form, using latitude and longitude angles.

$$x = r \cos \phi \cos \theta, \quad -\pi/2 \leq \phi \leq \pi/2$$

$$y = r \cos \phi \sin \theta, \quad -\pi \leq \theta \leq \pi$$

$$z = r \sin \phi$$

The parametric representation provides a symmetric range for the angular parameters θ and ϕ .

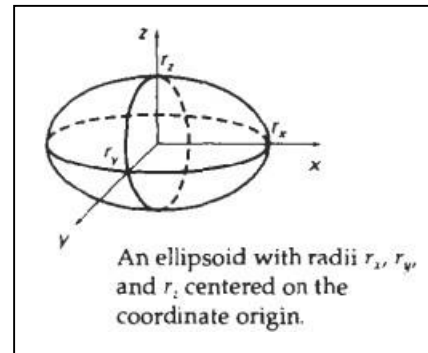
Ellipsoid

An ellipsoidal surface can be described as an extension of a spherical surface, where the radii in three mutually perpendicular directions can have different values. The Cartesian representation for points over the surface of an ellipsoid centered on the origin is-

$$\left(\frac{x}{r_x}\right)^2 + \left(\frac{y}{r_y}\right)^2 + \left(\frac{z}{r_z}\right)^2 = 1$$

And a parametric representation for the ellipsoid in terms of the latitude angle ϕ and the longitude angle θ is as follows:-

$$\begin{aligned} x &= r_x \cos \phi \cos \theta, & -\pi/2 \leq \phi \leq \pi/2 \\ y &= r_y \cos \phi \sin \theta, & -\pi \leq \theta \leq \pi \\ z &= r_z \sin \phi \end{aligned}$$



Torus

A torus is a doughnut-shaped object. It can be generated by rotating a circle or other conic about a specified axis. The Cartesian representation for points over the surface of a torus can be written in the form

$$\left[r - \sqrt{\left(\frac{x}{r_x}\right)^2 + \left(\frac{y}{r_y}\right)^2} \right]^2 + \left(\frac{z}{r_z}\right)^2 = 1$$

Parametric representation for a torus are similar to those for an ellipse except that angle ϕ extends over 360° .

$$\begin{aligned} x &= r_x(r + \cos \phi) \cos \theta, & -\pi \leq \phi \leq \pi \\ y &= r_y(r + \cos \phi) \sin \theta, & -\pi \leq \theta \leq \pi \\ z &= r_z \sin \phi \end{aligned}$$

Superquadrics

This class of objects is a generalization of the quadric representations. **Superquadrics** are formed by incorporating additional parameters into the quadric equations to provide increased flexibility for adjusting object shapes. The number of additional parameters used is equal to the dimension of the object: one parameter for curves and two parameters for surfaces.

Superellipse

We obtain a Cartesian representation for a superellipse from the corresponding equation for an ellipse by allowing the exponent on the x and y terms to be variable. One way to do this is to write the Cartesian superellipse equation in, the form

$$\left(\frac{x}{r_x}\right)^{2/s} + \left(\frac{y}{r_y}\right)^{2/s} = 1$$

Corresponding parametric equation for the superellipse can be expressed as;

$$x = r_x \cos^s \theta, \quad -\pi \leq \theta \leq \pi$$

$$y = r_y \sin^s \theta$$

Superellipsoid

A Cartesian representation for a superellipsoid is obtained from the equation for an ellipsoid by incorporating two exponent parameters:

$$\left[\left(\frac{x}{r_x}\right)^{2/s_2} + \left(\frac{y}{r_y}\right)^{2/s_2}\right]^{s_2/s_1} + \left(\frac{z}{r_z}\right)^{2/s_1} = 1$$

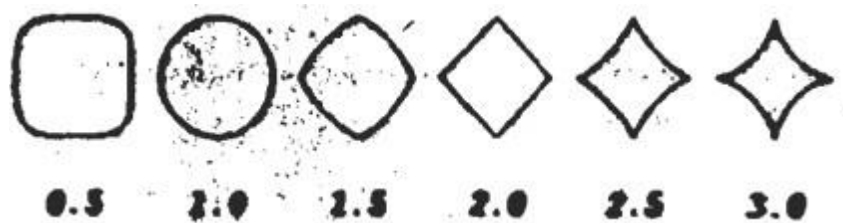
For $s_1=s_2=1$

We can write the corresponding parametric representation for the superellipsoid.

$$x = r_x \cos^{s_1} \phi \cos^{s_2} \theta, \quad -\pi/2 \leq \phi \leq \pi/2$$

$$y = r_y \cos^{s_1} \phi \sin^{s_2} \theta, \quad -\pi \leq \theta \leq \pi$$

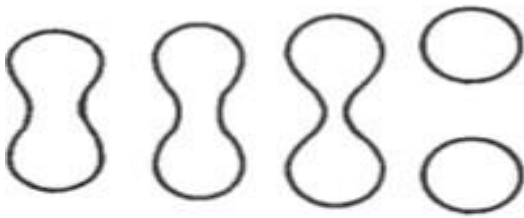
$$z = r_z \sin^{s_1} \phi$$



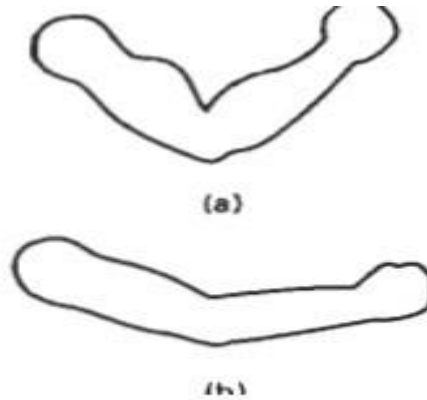
Superellipsoid plotted with different values for parameters and with $r_x = r_y$

Blobby Objects

Some objects don't maintain a fixed shape, but change their surface characteristics in certain motions or when in proximity to other objects. Examples in this class of objects include molecular structures, water droplets and other liquid effects, melting objects, and muscle shapes in the human body. **These** objects can be described as exhibiting "blobbiness" and are often simply referred to as blobby objects, since their shapes show a certain degree of fluidity.



Molecular bonding. As two molecules move away from each other, the surface shapes stretch, snap, and finally contract into spheres.



Blobby muscle shapes in a human arm.

Spline Representation

A spline is a flexible strip **used** to produce a smooth curve through a designated **set** of points. Several small weights are distributed along the length of the strip to hold it in position on the drafting table as the curve is drawn.

The term **spline curve** originally referred to a curve drawn in this manner. We can mathematically **describe** such a curve with a piecewise cubic polynomial function whose first and second derivatives are continuous across the various curve section.

In computer graphics, the term spline curve now refers to any composite curve formed with polynomial sections satisfying specified continuity conditions at the boundary of the pieces. A spline surface can be described with two sets of orthogonal spline curves. There are several different kinds of spline specifications that are used in graphics applications. Each individual specification simply refers to a particular type of polynomial with certain specified boundary conditions.

Control points:- We specify a spline curve by giving a set of coordinate positions, called "**control points**", which indicates the general shape of the curve. These,

control points are then fitted with piecewise continuous parametric polynomial functions in one of two ways:-

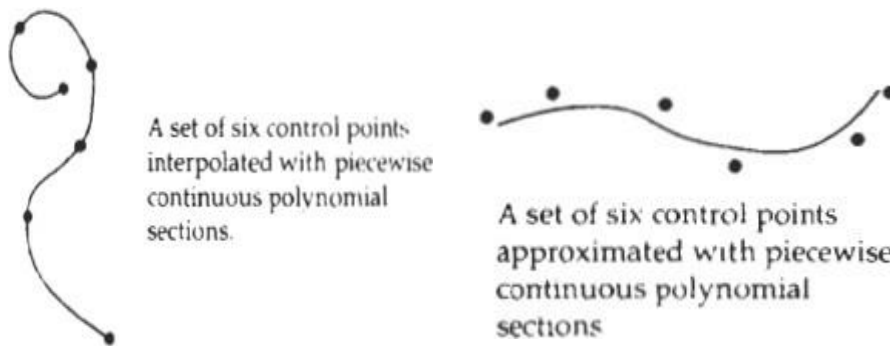
1. Interpolate

When polynomial sections are fitted so that the curve passes through each control point

2. Approximate

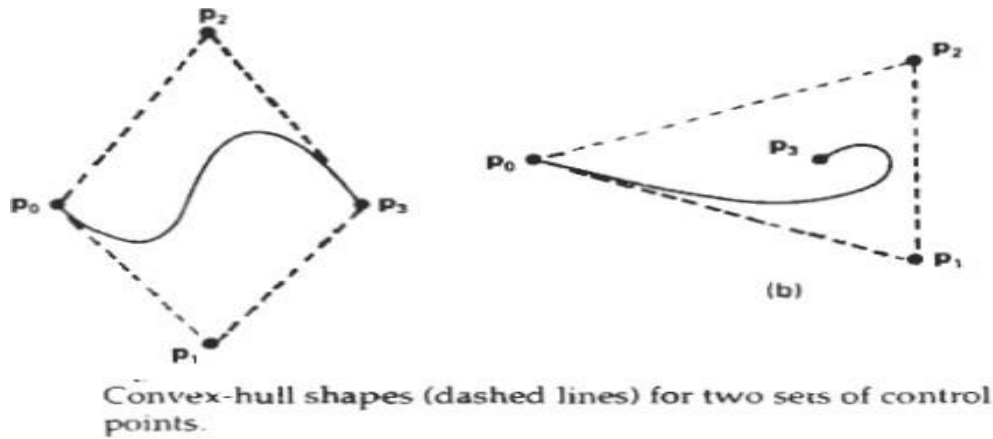
When the polynomial are fitted to the general control-point path without necessarily passing through any control point.

Interpolation curves are commonly used to digitize drawings or to specify animation paths. Approximation curves are primarily used as design tools to structure object surfaces.



A spline curve is defined, modified, and manipulated with operations on the control points.

Convex Hull: - The convex polygon boundary that encloses a set of control points is called the “**convex hull**”. One way envision the shape of a convex hull is to imagine a rubber band stretched around the positions of the control points so that each control point is either on the perimeter of the hull or inside it. Convex hulls provide a measure for the deviation of a curve or surface from the region bounding the control points. Some splines are bounded by the convex hull, thus ensuring that the polynomials smoothly follow the control points without erratic oscillations.



Parametric Continuity Conditions:

To ensure a smooth transition from one section **of** a piecewise parametric **curve** to the next, **we** can impose various **continuity conditions** at the connection points. **If** each **section** of a spline is described with a set of parametric coordinate functions of the **form**.

$$x = x(u), y = y(u), z = z(u), \quad u_1 \leq u \leq u_2 \text{ -----eq(1)}$$

We set parametric continuity by matching the parametric derivatives of adjoin curve sections at their common boundary.

Zero order parametric continuity: - described as C^0 continuity, means simply that the curves meet. That is, the values of **x**, **y**, and **z** evaluated at **u_2** , for the first curve section are equal, respectively, to the values of **x**, **y**, and **z** evaluated at **u_1** , for the next curve section.

First-order parametric continuity:- referred as C^1 , continuity, means that the first parametric derivatives (tangent lines) of the coordinate functions in Eq. 1 for two successive curve sections are equal at their joining point.

Second-order-parametric-continuity:- C^2 continuity, means that both the first and second parametric derivatives of the two curve sections are the same at the intersection, Higher-order parametric continuity conditions are defined similarly.



piecewise construction of a curve by joining two curve segments using different orders of continuity (a) Zero order (b) first order (c) second order

Geometric Continuity: - An alternate method for joining two successive curve sections is to specify conditions for geometric continuity. In this, we require parametric derivation of the two sections to be proportional to each other at their common boundary instead of equal to each other.

Zero order Geometric Continuity: - described as G^0 continuity, is the same as zero-order parametric continuity.

First-order-geometric-continuity - G^1 continuity means that the parametric first derivatives are proportional at the intersection of two successive sections.

Second-order geometric continuity:- G^2 continuity means that both the first and second parametric derivatives of the two curve sections are proportional at their boundary.

Spline Specifications

There are three equivalent methods for specifying a particular spline representation:

- (1) We can state the set of boundary conditions that are imposed on the spline
- (2) We can state the matrix that characterizes the spline; or
- (3) We can state the set of blending functions (or basis functions) that determine how specified geometric constraints on the curve are combined to calculate positions along the curve path.

To illustrate these three equivalent specifications, suppose we have the following parametric cubic polynomial representation for the x coordinate along the path of a spline section:

$$P(u) = \sum_{k=0}^n p_k BEZ_{k,n}(u). \quad 0 \leq u \leq 1$$

$$BEZ_{k,n}(u) = C(n, k) = \frac{n!}{k!(n-k)!} u^k (1-u)^{n-k}$$

$$x(u) = a \underset{x}{u^3} + b \underset{x}{u^2} + c \underset{x}{u} + d \underset{x} \quad 0 \leq u \leq 1$$

Boundary conditions for this curve might be set, for example, on the endpoint coordinate $x(0)$ and $x(1)$ and on the parametric first derivatives at the endpoints

$x'(0)$ and $x'(1)$. These four boundary conditions are sufficient to determine the values of the four coefficients a_x , b_x , c_x and d_x .

From the boundary conditions, we can obtain the matrix that characterizes this spline curve by first rewriting Eq. as the matrix product.

$$x(u) = \begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix} \begin{bmatrix} a_x \\ b_x \\ c_x \\ d_x \end{bmatrix}$$

Where U is the row matrix of powers of parameter u , and C is the coefficient column matrix. Using **Eq.** we can write the boundary conditions in matrix form and solve for the coefficient matrix C as

$$C = M_{spline} \cdot M_{geom}$$

Where M_{geom} is a four-element column matrix containing the geometric constraint values (boundary conditions) on the spline; and M_{spline} is the 4-by-4 matrix that transforms the geometric constraint values to the polynomial coefficients and provides a characterization for the spline curve. Matrix M_{geom} contains control point coordinate values and other geometric constraints that have been specified. Thus, we can substitute the matrix representation for C into Eq. to obtain

$$x(u) = U \cdot M_{spline} \cdot M_{geom}$$

The matrix M_{spline} characterizing a spline representation, sometimes called the basis matrix, is particularly useful for transforming from one spline representation to another.

We can expand the above equation to obtain a polynomial representation for coordinate x in terms of the geometric constraint parameters.

$$x(u) = \sum_{k=0}^3 g_k \cdot BF_k(u)$$

Where g_k are the constraint parameter, such as the control point coordinates and slope of the curve at the control points and $BF_k(u)$ are the polynomial blending functions.

Cubic Splines.

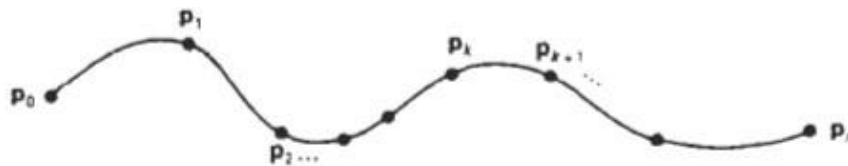
This class of splines is most often used to set up paths for object motions or to provide a representation for an existing object or drawing, but interpolation splines are also used sometimes to design object shapes. Cubic polynomials offer a reasonable compromise between flexibility and speed of computation. Compared to higher-order polynomials, cubic splines require less calculations and memory and they are more stable. Compared to lower-order polynomials, cubic splines are more flexible for modeling arbitrary curve shapes. Given a set of control points, cubic interpolation splines are obtained by fitting the input points with a piecewise cubic polynomial curve that passes through every control point. Suppose we have $n + 1$ control points specified with coordinate

$$P_k = (x_k, y_k, z_k), \quad k=0, 1, 2, \dots, n$$

We can describe the parametric cubic polynomial that is to be fitted between each pair of control points with the following set of equations.

$$\begin{aligned} x(u) &= a_x u^3 + b_x u^2 + c_x u + d_x \\ y(u) &= a_y u^3 + b_y u^2 + c_y u + d_y \\ z(u) &= a_z u^3 + b_z u^2 + c_z u + d_z \end{aligned} \quad 0 \leq u \leq 1$$

For each of these three equations, we need to determine the values of the four coefficients a , b , c , and d in the polynomial representation for each of the n curve sections between the $n + 1$ control points. We do this by setting enough boundary conditions at the "joints" between curve sections so that we can obtain numerical values for all the coefficients.



A piecewise continuous cubic-spline interpolation of $n + 1$ control points.

Hermite Splines.

A Hermite spline is an interpolating piecewise cubic polynomial with a specified tangent at each control point. Unlike the natural cubic splines, Hermite splines can be adjusted locally because each curve section is only dependent on its endpoint constraints.

If $\mathbf{P}(u)$ represents a parametric cubic point function for the curve section between control points $\mathbf{p}_k$ and $\mathbf{p}_{k+1}$, then the boundary conditions that define this Hermite curve section are

$$P(0) = p_k$$

$$P(1) = p_{k+1}$$

$$P'(0) = Dp_k$$

$$P'(1) = Dp_{k+1}$$

With Dp_k and Dp_{k+1} specifying the values for the parametric derivatives at control points p_k and p_{k+1} .

We can rewrite the vector equivalent of above equation for this Hermite curve section as $P(u) = au^3 + bu^2 + cu + d \quad 0 \leq u \leq 1$

Where the x component of P is $P(u) = a_x u^3 + b_x u^2 + c_x u + d_x$ and similarly for the y and z components. The matrix equivalent of the above equation is y and z components. The matrix equivalent of above Eq. is

$$p(u) = \begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix}$$

And derivative of the point function can be expressed as

$$p'(u) = \begin{bmatrix} 3u^2 & 2u & 1 & 0 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix}$$

Substituting the value 0 and 1 for parameter u into the previous two equations, we can express the hermite boundary conditions in the matrix form.

$$\begin{bmatrix} P_k \\ P_{k+1} \\ DP_k \\ DP_{k+1} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix}$$

Solving this equation for the polynomial coefficient, we have

$$\begin{aligned} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} &= \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{bmatrix}^{-1} \cdot \begin{bmatrix} P_k \\ P_{k+1} \\ DP_k \\ DP_{k+1} \end{bmatrix} \\ &= \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} P_k \\ P_{k+1} \\ DP_k \\ DP_{k+1} \end{bmatrix} \\ &= M_H \cdot \begin{bmatrix} P_k \\ P_{k+1} \\ DP_k \\ DP_{k+1} \end{bmatrix} \end{aligned}$$

Where M_H , is the Hermite matrix, is the inverse of the boundary constraint matrix can thus be written in terms of the boundary condition as

$$P(u) = [u^3 \ u^2 \ u \ 1] \cdot M_H \cdot \begin{bmatrix} P_k \\ P_{k+1} \\ DP_k \\ DP_{k+1} \end{bmatrix}$$

Finally we can determine expression for the Hermite blending functions by carrying out the matrix multiplication and collecting coefficients for the boundary constraints to obtain the polynomial form.

$$P(u) = p_k H_0(u) + p_{k+1} H_1(u) + DP_k H_2(u) + DP_{k+1} H_3(u)$$

Hermite polynomial can be useful for some digitizing applications where it may not be too difficult to specify or approximate the curve slope. But for most problems in computer graphic, it is more useful to generate spline curves without requiring input values for curve slopes or other geometric information.

Bezier Curve

This spline approximation method was developed by the French engineer Pierre Bezier for use in the design of Renault automobile bodies. **Bezier** splines have a number of properties that make them highly useful and convenient for curve and surface design. They are also easy to implement. For these reasons, Bezier splines are widely available in various CAD systems, in general graphics packages (such as GL on Silicon Graphics systems), and in assorted drawing and painting packages.

A Bezier curve section can be fitted to any number of control points. The number of control points to be approximated and their relative position determine the degree of the Bezier polynomial. As with the interpolation splines, a Bezier curve can be specified with boundary conditions, with a characterizing matrix, or with blending functions. For general Bezier curves, the blending-function specification is the most convenient. Suppose we are given $n + 1$ control-point positions: $p_k = (x_k, y_k, z_k)$, with k varying from 0 to n . These coordinate points can be blended to produce the following position vector $P(u)$, which describes the path of an approximating Bezier polynomial function between p_0 and p_n .

$$P(u) = \sum_{k=0}^n p_k BEZ_{k,n}(u), \quad 0 \leq u \leq 1$$

The Bezier blending function $BEZ_{k,n}(u)$ are the Barnstein polynomials:

$$BEZ_{k,n}(u) = C(n, k) u^k (1-u)^{n-k}$$

Where the $C(n, k)$ are the binomial coefficients: $C(n, k) = \frac{n!}{k!(n-k)!}$

Equivalently, we can define Bezier blending function with the recursive calculations:-

$$BEZ_{k,n}(u) = (1-u) BEZ_{k,n-1}(u) + u BEZ_{k-1,n-1}(u), \quad n > k \geq 1$$

$$x(u) = \sum_{k=0}^n x_k BEZ_{k,n}(u)$$

$$y(u) = \sum_{k=0}^n y_k BEZ_{k,n}(u)$$

$$z(u) = \sum_{k=0}^n z_k BEZ_{k,n}(u)$$

Properties of Bezier Curves

01. The basis functions are real
02. Bezier curve always passes through the first and last control points. i.e., curve has same endpoints as the guiding polygon
03. The degree of the polynomial defining the curve segment is one less than the number of defining polygon point. Therefore for 4 control points, the degree of the polynomial is 3.
04. The curve generally follows the shape of the defining polygon.
05. The direction of the tangent vector at the end points is the same as that of the vector determined by the first and last segments.
06. The curve lies entirely within the convex hull formed by four control points.
07. The convex hull property for a Bezier curve ensures that the polynomial smoothly follows the control points.
08. The curve exhibits the variation diminishing property. This means that the curve does not oscillate about any straight line more often than the defining polygon.
09. The curve is invariant under an affine transformation.

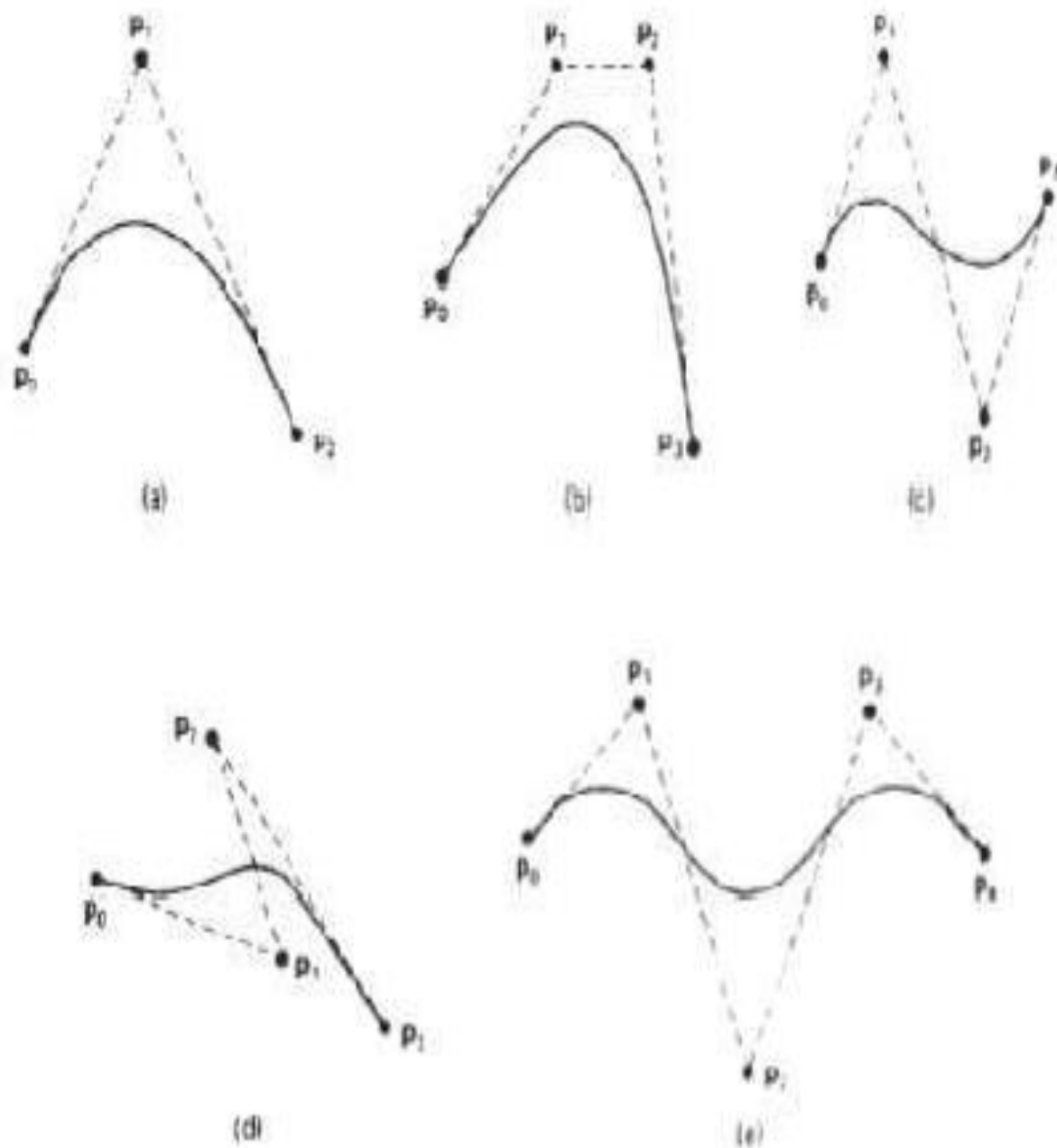


Figure 10.34

Examples of two-dimensional Bézier curves generated from three, four, and five control points. Dashed lines connect the control-point positions.

B-spline Curve

A curve generated by using the vertices of a defining polygon is dependent on some interpolation or approximation scheme to establish the relationship between the curve and the polygon. This scheme is provided by the choice of basis function. The Bezier curve produced by the Bernstein basis function has a limited flexibility. First the number of specified polygon vertices fixes the order of resulting polynomial which defines the curve. For example, polygon with four vertices results a cubic polynomial curve. The only way to reduce the degree of the curve is to reduce the number of vertices, and conversely the only way to increase the degree of the curve is to increase the number of vertices. The second limiting characteristic is that the value of blending function is non-zero for all parameter value over the entire curve. Due to this change in one vertex, changes the entire curve and this eliminates the ability to produce a local change within a curve.

There is another basis function, called the B-spline, which contains the Bernstein basis as a special case. The B-spline is non-global because each vertex affects the shape of the curve only over a range of parameter values where its associated basis function is non-zero. The B-spline basis also allows the order of the basis function and hence the degree of the resulting curves independent on the number of vertices. It is possible to change the degree of the resulting curve without changing the number of vertices of the defining polygon.

If $P(u)$ be the position vectors along the curve as a function of the parameter u , a B-spline curve is given by

$$P(u) = \sum_{k=0}^n P_k B_{kd}(u), \quad u_{\min} \leq u \leq u_{\max}$$

$$2 \leq d \leq n+1$$

Where P_k are an input set of $n+1$ control points.

There are several differences between this B-spline formation and that of Bezier Splines. The range of parameter u now depends on how we choose the B-spline parameters and the B-spline blending function B_{kd} are polynomial of degree $d-1$ where parameters d can be chosen to be any integer value in the range of 2 up to the number of control points, $n+1$.

Blending function for B-spline curve is defined by the Cox-deBoor recursion formulas.

$$B_k(u) = \begin{cases} 1 & \text{if } u_k \leq u \leq u_{k+1} \\ 0 & \text{otherwise} \end{cases}$$

$$B_k(u) = \frac{u - u_{k-d}}{u_{k-d+1} - u_{k-d}} \frac{u_{k+1} - u}{u_{k+1} - u_{k+1-d}} B_{k+1,d}(u)$$

$$B_k(u) = \frac{u - u_{k-d}}{u_{k-d+1} - u_{k-d}} \frac{u_{k+1} - u}{u_{k+1} - u_{k+1-d}} B_{k+1,d}(u)$$

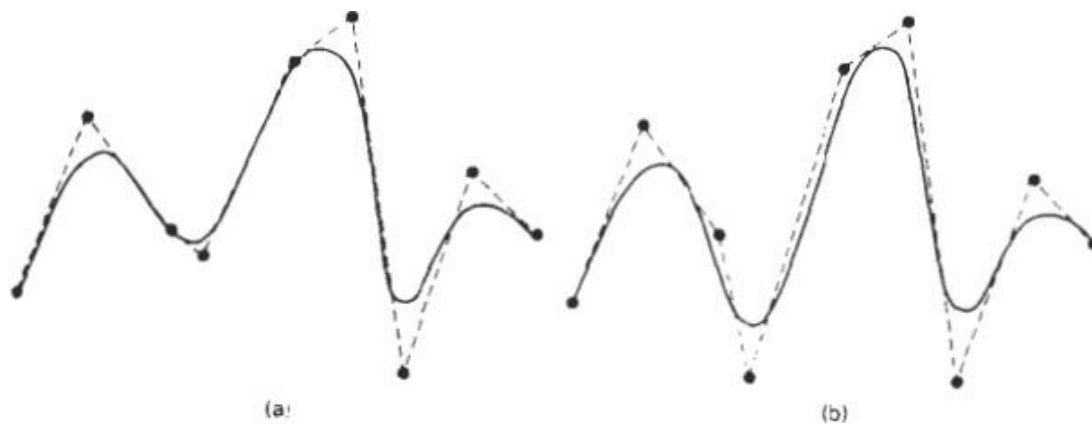
Where each blending function is defined over d subintervals of the total range of u . The selected set of subinterval endpoints u_j , is referred to as a **knot** vector. We can choose any values for the subinterval endpoints satisfying the relation $u_{j-1} \leq u_j \leq u_{j+1}$. Values for u_{min} and u_{max} then depend on the number of control points we select, the value we choose for parameter d , and how we set up the subintervals (knot vector). Since it is possible to choose the elements of the knot vector so that the denominators in the previous

calculations can have a value of 0, this formulation assumes that any terms evaluated as $0/0$ are to be assigned the value 0.

For a cubic B-spline, the degree $d-1 = 3$, so $d=4$ and hence the no of control points $(n+1)$ required is atleast 4.

Properties of B-spline Curve

1. The polynomial curve has degree $d-1$ and C^{d-2} continuity over the range of u .
2. For $n+1$ control points the curve is describe with $n+1$ blending functions.
3. Each blending function $B_{k,d}$ is defined over d sub-intervals of the total range of u , starting at knot value u_k .
4. The range of parameter u is divided into $n+d$ subintervals by the $n + d + 1$ specified in the knot vector.
5. With knot values labeled as $\{u_0, u_1, \dots, u_{n+d}\}$ the resulting B-spline curve is defined only in the interval from knot value u_d to knot value u_{n+1} .
6. Each section of the spline curve is influenced by d control points.
7. Any one control point can affect the shape of at most d curve sections.



Local modification of a B-spline curve. Changing one of the control points in (a) produces curve (b), which is modified only in the neighborhood of the altered control point.

Uniform, periodic B-Splines

When the spacing between knot values is constant, the resulting curve is called a uniform B-Spline. For example, we can set up a uniform knot vector as

$\{-1.5, -1.0, -0.5, 0.0, 0.5, 1.0, 1.5, 2.0\}$

Often knot values are normalized to the range between 0 and 1.

Bezier Surfaces

Introduction: Two sets of orthogonal Bezier curves can be used to design an object surface by specifying by an input mesh of control points. The parametric vector function for the Bezier surface is formed as the Cartesian product of Bezier blending functions:

$$\mathbf{P}(u, v) = \sum_{j=0}^m \sum_{k=0}^n \mathbf{p}_{j,k} \text{BEZ}_{j,m}(v) \text{BEZ}_{k,n}(u)$$

With $\mathbf{p}_{j,k}$, specifying the location of the $(m+1)$ by $(n+1)$ control points.

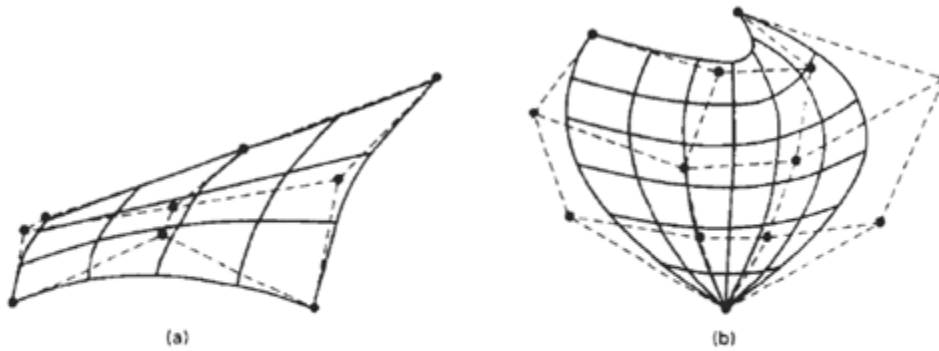


Figure 10-39

Bezier surfaces constructed for (a) $m = 3, n = 3$, and (b) $m = 4, n = 4$. Dashed lines connect the control points.

Figure 10-39 illustrates two Bezier surface plots. The control points are connected by dashed lines, and the solid lines show curves of constant u and constant v . Each curve of constant u is plotted by varying v over the interval from 0 to 1, with u fixed at one of the values in this unit interval. Curves of constant v are plotted similarly.

Bezier surfaces have the same properties as Bezier curves, and they provide a convenient method for interactive design applications. For each surface patch, we can select a mesh of control points in the xy "ground" plane, then we choose elevations above the ground plane for the z -coordinate values of the control points. Patches can then be pieced together using the boundary constraints.

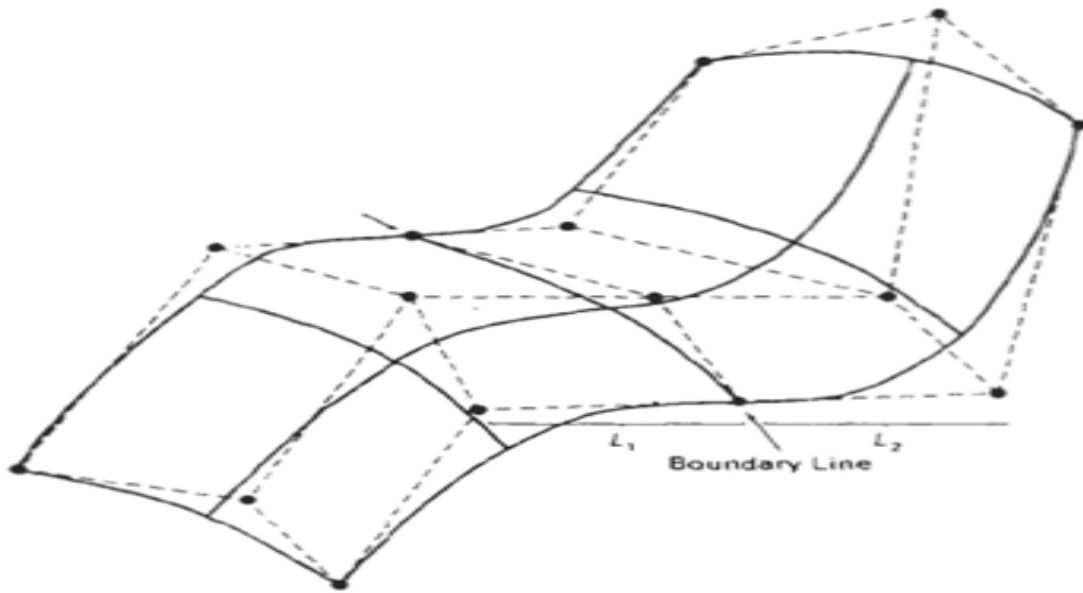


Figure 10-40

A composite Bézier surface constructed with two Bézier sections, joined at the indicated boundary line. The dashed lines connect specified control points. First-order continuity is established by making the ratio of length L_1 to length L_2 constant for each collinear line of control points across the boundary between the surface sections.

Figure 10-40 illustrates a surface formed with two Bezier sections. As with curves, a smooth transition from one section to the other is assured by establishing both zero-order and first-order continuity at the boundary line. Zero-order continuity is obtained by matching control points at the boundary. First-order continuity is obtained by choosing control points along a straight line across the boundary and by maintaining a constant ratio of collinear line segments for each set of specified control points across section boundaries.

B-Spline curves and surfaces

Introduction: These are the most widely used class of approximating splines. B-splines have two advantages over Bezier splines: (1) the degree of a B-spline polynomial can be set independently of the number of control points (with certain limitations), and (2) B-splines allow local control over the shape of a spline curve or surface. The trade-off is that B-splines are more complex than Bezier splines.

B-Spline Curves: We can write a general expression for the calculation of coordinate positions along a B-spline curve in a blending-function formulation as

$$P(u) = \sum_{k=0}^n p_k B_{k,d}(u), \quad u_{\min} \leq u \leq u_{\max}, \quad 2 \leq d \leq n + 1$$

where the p_k are an input set of $n + 1$ control points. There are several differences between this B-spline formulation and that for Bezier splines. The range of parameter u now depends on how we choose the B-spline parameters. And the B-spline blending functions $B_{k,d}$ are polynomials of degree $d - 1$, where parameter d can be chosen to be any integer value in the range from 2 up to the number of control points, $n + 1$. (Actually, we can also set the value of d at 1, but then our "curve" is just a point plot of the control points.) Local control for B-splines is achieved by defining the blending functions over subintervals of the total range of u .

Blending functions for B-spline curves are defined by the Cox-deBoor recursion formulas:

$$B_{k,1}(u) = \begin{cases} 1, & \text{if } u_k \leq u < u_{k+1} \\ 0, & \text{otherwise} \end{cases}$$

$$B_{k,d}(u) = \frac{u - u_k}{u_{k+d-1} - u_k} B_{k,d-1}(u) + \frac{u_{k+d} - u}{u_{k+d} - u_{k+1}} B_{k+1,d-1}(u)$$

where each blending function is defined over d subintervals of the total range of u . The selected set of subinterval endpoints u_j is referred to as a knot vector. We can choose any values for the subinterval endpoints satisfying the relation $u_1 \leq u_2 \leq \dots \leq u_n$. Values for $u_{\max}$ and $u_{\min}$ then depend on the number of control points we select, the value we choose for parameter d , and how we set up the subintervals (knot vector). Since it is possible to choose the elements of the knot vector so that the denominators in the previous calculations can have a value of 0, this formulation assumes that any terms evaluated as $0/0$ are to be assigned the value 0.

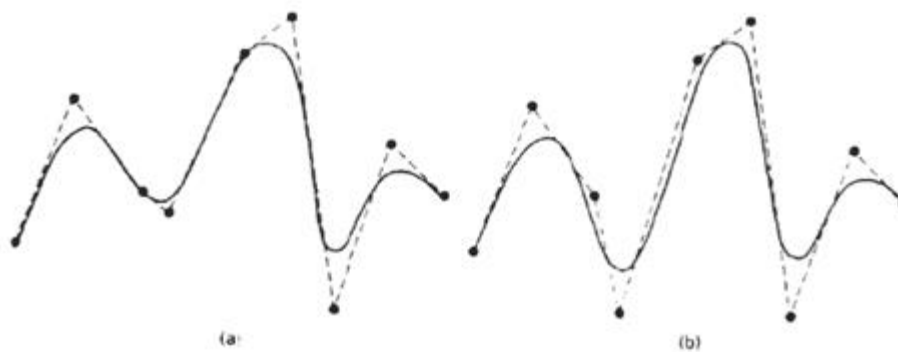


Figure 10-41

Local modification of a B-spline curve. Changing one of the control points in (a) produces curve (b), which is modified only in the neighborhood of the altered control point.

Figure 10-41 demonstrates the local-control characteristics of B splines. In addition to local control, B-splines allow us to vary the number of control points used to design a curve without changing the degree of the polynomial. Also, any number of control points can be added or modified to manipulate curve shapes. Similarly, we can increase the number of values in the knot vector to aid in curve design. When we do this, however, we also need to add control points since the size of the knot vector depends on parameter n .

B-spline curves have the following properties.

1. The polynomial curve has degree $d - 1$ and C^{d-2} Continuity over the range of u .
2. For $n + 1$ control points, the curve is described with $n + 1$ blending functions.
3. Each blending function $B_{k,d}$ is defined over d subintervals of the total range of u , starting at knot value u_k .
4. The range of parameter u is divided into $n - d + 1$ subintervals by the $n - d + 1$ values specified in the knot vector.
5. With knot values labeled as $[u_1 \ u_2 \ \dots \ u_{n-d+1}]$, the resulting B-spline curve is defined only in the interval from knot value u_{d-1} , up to knot value u_n .
6. Each section of the spline curve (between two successive knot values) is influenced by d control points.
7. Any one control point can affect the shape of at most d curve sections.

In addition, a B-spline curve lies within the convex hull of at most $d + 1$ control points, so that B-splines are tightly bound to the input positions. For any value of u in the interval from knot value u_{d-1} to u_n , the sum over all basis functions is 1:

Given the control-point positions and the value of parameter u , we then need to specify the knot values to obtain the blending functions using the recurrence relations 10-55.

Basic Illumination models

1. Ambient light
2. Diffuse reflection
3. Specular reflection and Phong model
4. Combined approach
5. Warn model
6. Intensity Attenuation
7. Color consideration
8. Transparency and Shadows

Basic Illumination

- An illumination model, also called lighting model and sometimes referred to as a shading model, is used to calculate the intensity of light that we should see at a given point on the surface of an object.
- Illumination models in computer graphics are often loosely derived from the physical laws that describe surface light intensities.

• **Illumination Model** - the model for calculating light intensity at a single surface point is called illumination model or a lighting model

• **Surface Rendering** is a procedure for applying a lighting model to obtain pixel intensities for all the projected surface positions in a scene.

LIGHT

- When we view an opaque non uniform object, we see reflected light from the surfaces of the object.
- A surface that is not directly exposed to a light source may still be visible if nearby objects are illuminated.

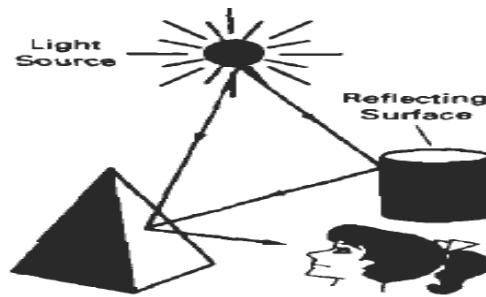


Figure 14-1
Light viewed from an opaque nonluminous surface is in general a combination of reflected light from a light source and reflections of light from other surfaces

Diffuse

- Surfaces that are rough, or grainy, tend to scatter the reflected light in all directions.
- This scattered light is called diffuse reflection.

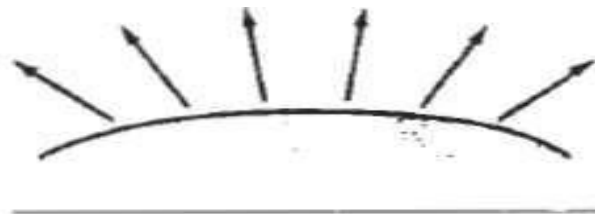
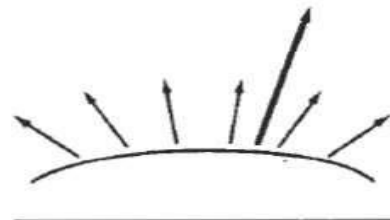


Figure 14-4
Diffuse reflections from a surface.

Specular

- In addition to diffuse reflection, light sources create highlights, or bright spots, called specular reflection.
- This highlighting effect is more pronounced on shiny surfaces than on dull surfaces.



Specular Reflection and Diffuse

Specular and Diffuse Reflection

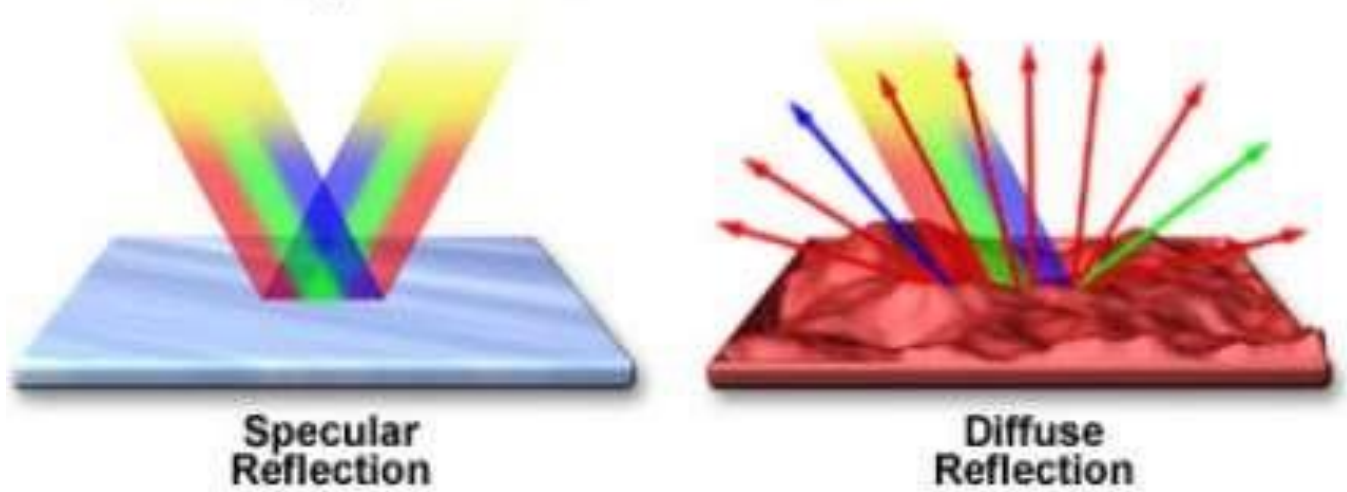


Figure 1

Ambient

- This is a simple way to model the combination of light reflections from various surfaces to produce a uniform illumination called the ambient light, or background light.



Ambient light (I_a)

- Ambient light has no spatial or directional characteristics.
- The amount of ambient light incident on each object is a constant for all surfaces and over all directions.
- The intensity of the reflected light for each surface depends on the optical properties of the surface; that is, how much of the incident energy is to be reflected and how much absorbed.
- We can set the level for the ambient light in a scene with parameter I_a and each surface is then illuminated with this constant value.

Diffuse reflection (kd)

- Diffuse reflections are constant over each surface in a scene, independent of the viewing direction.
- The fractional amount of the incident light that is diffusely reflected can be set for each surface with parameter **kd**, the diffuse-reflection coefficient, or diffuse reflectivity.
- Parameter **kd** is assigned a constant value in the interval 0 to 1, according to the reflecting properties we want the surface to have.

Diffuse

- If a surface is exposed only to ambient light, we can express the intensity of the diffuse reflection at any point on the surface as

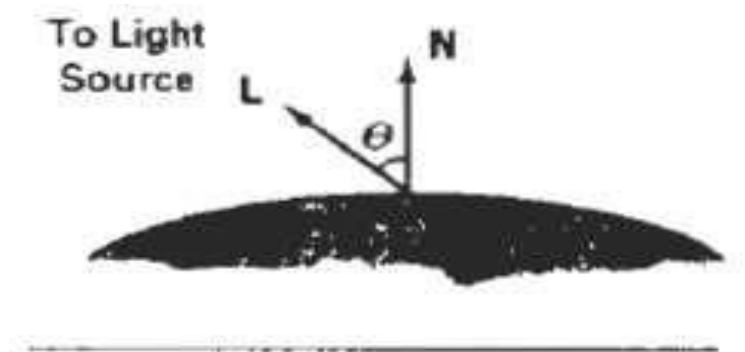
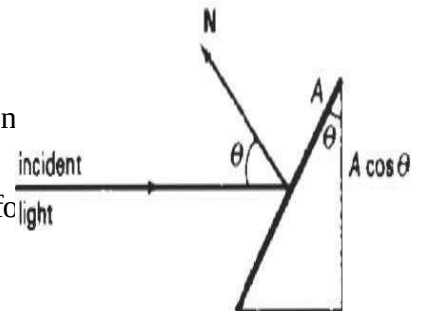
$$I_{\text{ambdiff}} = k_d I_a$$

- (where K_d is the diffuse-reflection coefficient)
- If we denote the angle of incidence between the incoming light direction and the surface normal as θ , then the projected area of a surface patch perpendicular to the light direction is proportional to $\cos\theta$.
- If I_L is the intensity of the point light source, then the diffuse reflection equation for a point on the surface can be written as

$$I_{l,\text{diff}} = K_d I_l \cos\theta$$

- If N is the unit normal vector to a surface and L is the unit direction to the point light source from a position on the surface
- (Fig. 14-9), then $\cos\theta = N \cdot L$ and the diffuse reflection equation for single point-source illumination is

$$I_{l,\text{diff}} = K_d I_l (N \cdot L)$$



Specular reflection

- When we look at an illuminated shiny surface, such as polished metal, an apple, or a person's forehead, we see a **highlight, or bright spot**, at certain viewing directions.

This phenomenon, called specular reflection, is the result of total, or near total, reflection of the incident light in a concentrated region around the specular reflection angle.

- In this figure, we use
- $\mathbf{N}$ unit normal surface vector
- $\mathbf{R}$ to represent the unit vector in the direction of **ideal specular reflection**;
- $\mathbf{L}$ to represent the unit vector directed toward the **point light source**;
- $\mathbf{V}$ as the unit vector pointing to the **viewer** from the surface position.

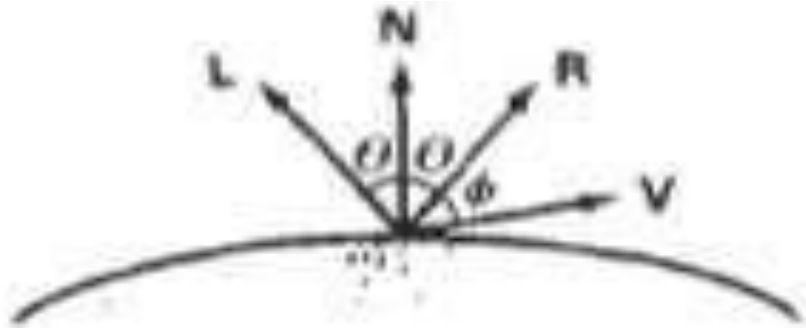
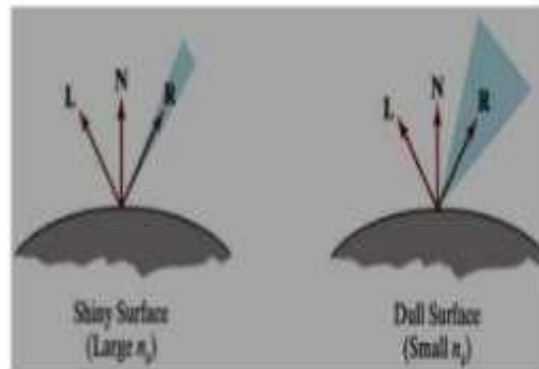


Figure 14-12
Specular-reflection angle
equals angle of incidence θ .

Phong specular-reflection model

- Shiny surfaces have a narrow specular-reflection range, and dull surfaces have a wider reflection range.
- An empirical model for calculating the specular-reflection range, developed by **Phong Bui Tuong** and called the Phong specular-reflection model.

Phong specular-reflection model, or simply the **Phong model**, sets the intensity of specular reflection proportional to $\cos^n \phi$. Angle ϕ can be assigned values in the range 0° to 90° , so that $\cos \phi$ varies from 0 to 1. The value assigned to *specular-reflection parameter* n_s is determined by the type of surface that we want to display. A very shiny surface is modeled with a large value for n_s (say, 100 or more), and smaller values (down to 1) are used for duller surfaces.



- The Warn model provides a method for **simulating studio lighting effects by controlling light intensity in different directions**.
- Intensity in different directions is controlled by selecting values for the Phong exponent. In addition, light controls, such as "barn doors" and spotlighting, used by studio photographers can be simulated in the Warn model.
- It is implemented in PHIGS+



Figure 14-20

Studio lighting effects produced with the Warn model, using five light sources to illuminate a Chevrolet Camaro. (Courtesy of David R. Warn, General Motors Research Laboratories.)

Intensity Attenuation

- As radiant energy from a point light source travels through space, its amplitude is attenuated by the factor $1/d^2$, where d is the distance that the light has traveled.
- a general inverse quadratic attenuation function can be set up as
- A user can then fiddle with the coefficients a_0 , a_1 , and a_2 , to obtain a variety of lighting effects for a scene.

$$f(d) = \frac{1}{a_0 + a_1d + a_2d^2}$$

With a given set of attenuation coefficients, we can limit the magnitude of the attenuation function to 1 with the calculation

$$f(d) = \min\left(1, \frac{1}{a_0 + a_1d + a_2d^2}\right) \quad (14-12)$$

Using this function, we can then write our basic illumination model as

$$I = k_a I_a + \sum_{i=1}^n f(d_i) I_{li} [k_d (\mathbf{N} \cdot \mathbf{L}_i) + k_s (\mathbf{N} \cdot \mathbf{H}_i)^{n_s}] \quad (14-13)$$

where d_i is the distance light has traveled from light source i .

Color

- To incorporate color, we need to write the intensity equation as a function of the color properties of the light **sources** and object surfaces.
- For an RGB description, each color in a scene is expressed in terms of red, green, and blue components.
- The diffuse reflection coefficient vector, for example, would then have RGB components (k_dR, k_dG, k_dB).
- If we want an object to have a blue surface, we select a nonzero value in the range from 0 to 1 for the blue reflectivity component, k_dB
- The red and green reflectivity components are set to zero ($k_dR = k_dG = 0$).
- The intensity calculation for this example reduces to the single expression

$$I_B = k_{dB} I_{dB} + \sum_{i=1}^n f_i(d) I_{iB} [k_{dB} (N \cdot L_i) + k_{sB} (N \cdot H_i)^{n_s}]$$

Transperence

- Realistic transparency effects are modeled by considering light refraction.
- When light is incident upon a transparent surface, part of it is reflected and part is **refracted**.

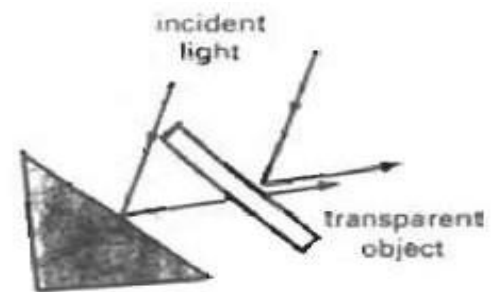


Figure 14-25
Light emission from a transparent surface is in general a combination of reflected and transmitted light.

- **Snell's Law**
- Angle of refraction θ_r is calculated from the angle of incidence θ_i , the index of refraction n_i of the "incident" material (usually air), and the index of refraction n_r of the refracting material according to

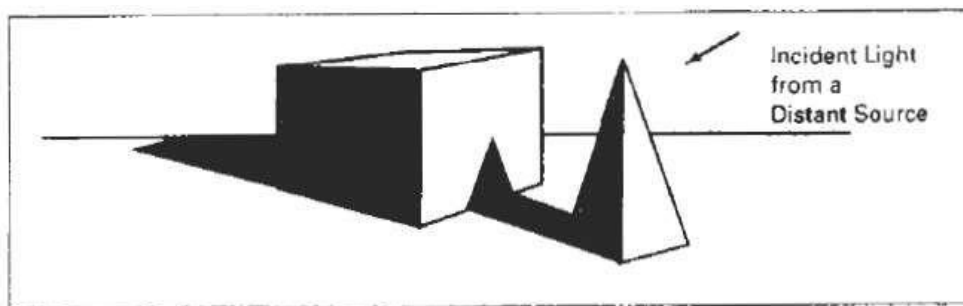
$$\sin \theta_r = \frac{n_i}{n_r} \sin \theta_i$$

From Snell's law and the diagram in Fig. 14-27, we can obtain the unit transmission vector $\mathbf{T}$ in the refraction direction θ_r as

$$\mathbf{T} = \left(\frac{n_i}{n_r} \cos \theta_i - \cos \theta_r \right) \mathbf{N} - \frac{n_i}{n_r} \mathbf{L} \quad (14-18)$$

- A simpler procedure for modeling transparent objects is to ignore the path shifts altogether.
- In effect, this approach assumes there is no change in the index of refraction from one material to another, so that the angle of refraction is always the same as the angle of incidence.

Shadow



- Hidden-surface methods can be used to locate areas where light sources produce shadows.
- By applying a hidden-surface method with a light source at the view position, we can determine which surface sections cannot be "seen" from the light source.
- These are the shadow areas.



Figure 14-26

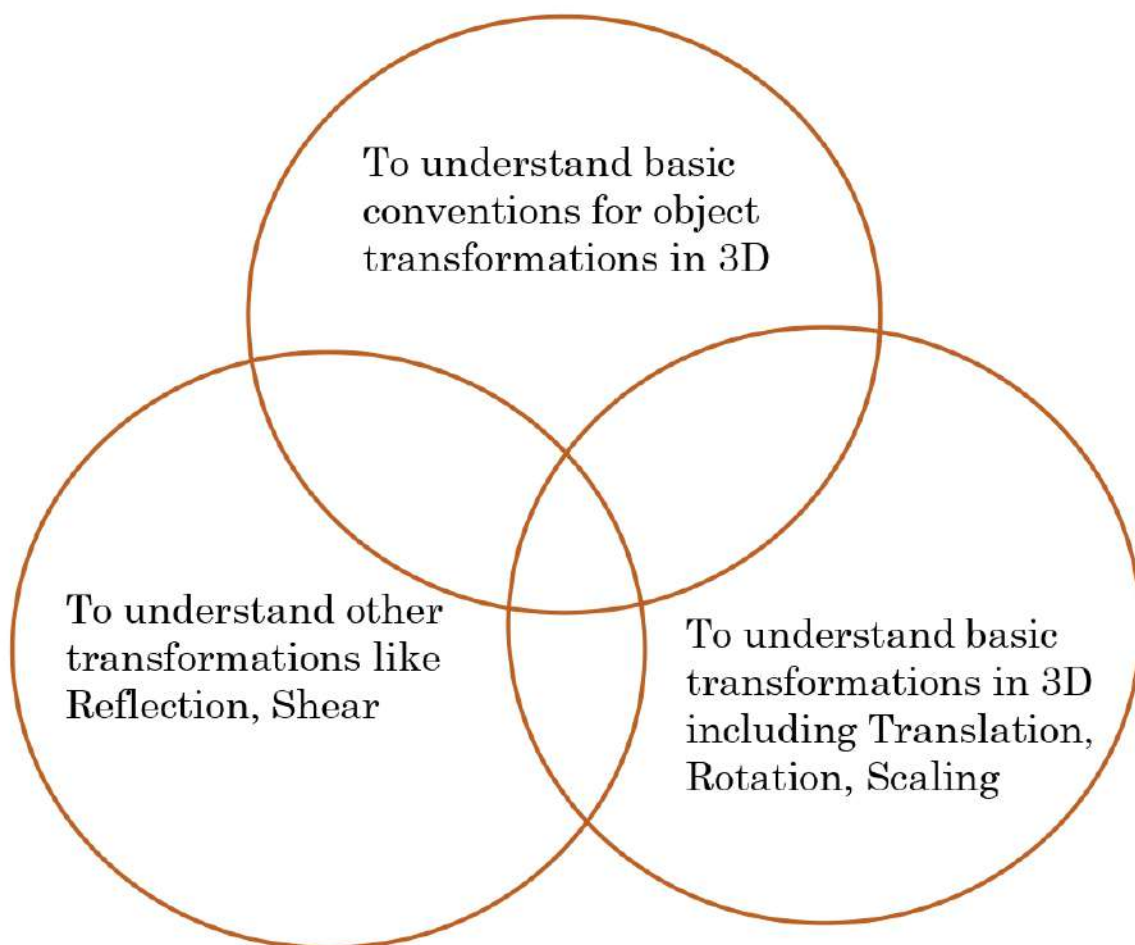
- Once we have determined the shadow areas for all light sources, the shadows could be treated as surface patterns and stored in pattern arrays.
- The scene in Fig. 14-26 shows shadow effects produced by multiple light sources.
- We can display shadow areas with ambient- light intensity only, or we can combine the ambient light with specified surface textures.

CONTENTS

- Transformation
- Types of transformation
- 3-D Transformation
 - 1) Translation
 - 2) Rotation
 - 3) Scaling
 - 4) Reflection
 - 5) Shearing



OBJECTIVE



TRANSFORMATION

- Transformations are a fundamental part of the computer graphics. Transformations are the movement of the object in Cartesian plane .
- TYPES OF TRANSFORMATION
 - ❖ There are two types of transformation in computer graphics.
 - 1) 2D transformation
 - 2) 3D transformation
 - ❖ Types of 2D and 3D transformation
 - 1) Translation 2) Rotation 3) Scaling
 - 4) Shearing 5) reflection



3D TRANSFORMATION

- When the transformation takes place on a 3D plane, it is called 3D transformation
- The translation, scaling and rotation transformations used for 2D can be extended to three dimensions.
- In 3D, each transformation is represented by a 4x4 matrix.
- Using homogeneous coordinates it is possible to represent each type of transformation in a matrix form and integrate transformations into one matrix
- To apply transformations, simply multiply matrices, also easier in hardware and software implementation
- Homogeneous coordinates can represent directions



- Homogeneous coordinates: 4 components
- Transformation matrices: 4×4 elements

$$\begin{bmatrix} a & b & c & p \\ d & e & f & q \\ g & h & i & r \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- **WHY WE USE TRANSFORMATION**

Transformation are used to position objects, to shape object, to change viewing positions, and even how something is viewed.



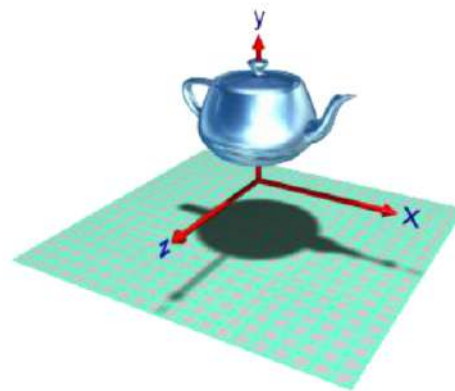
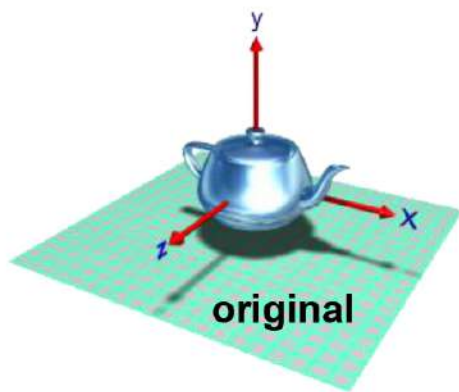
3D TRANSLATION

- Moving of object is called translation.
- In 3 dimensional homogeneous coordinate representation , a point is transformed from position $P = (x, y, z)$ to $P' = (x', y', z')$

This can be written as: Using $P' = T \cdot P$ where

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





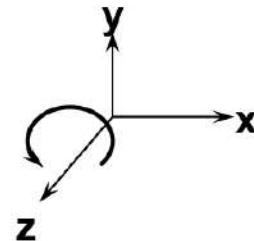
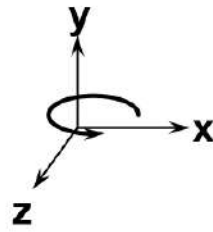
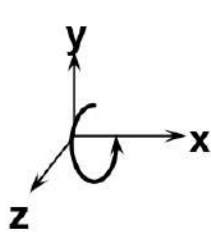
translation along y,
or $V = (0, k, 0)$

TRANSLATION



3-D ROTATIONS

- Rotation needs an angle and an axis.
- Rotation is defined according to the right-hand rule (our convention)
- In 3D, rotation is about a vector, which can be done through rotations about x, y or z axes.
- Positive rotations are anti-clockwise, negative rotations are clockwise, when looking down a positive axis towards the origin



3-D ROTATION TRANSFORMATION MATRIX

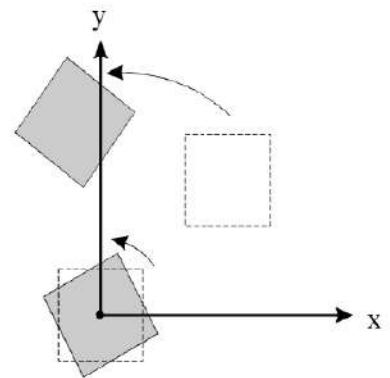
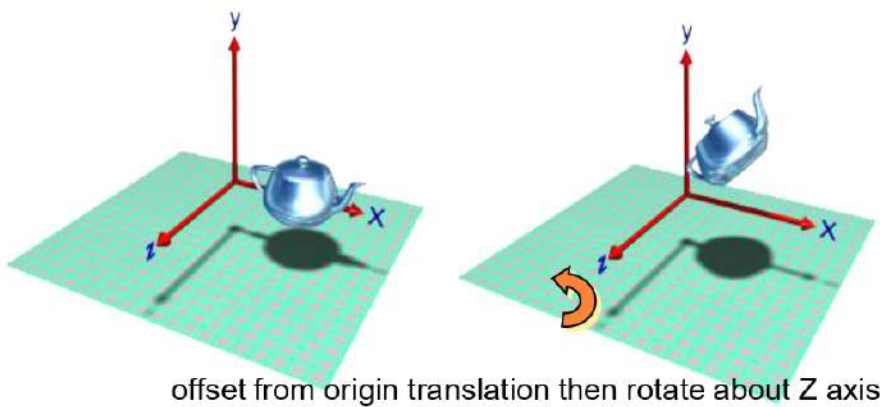
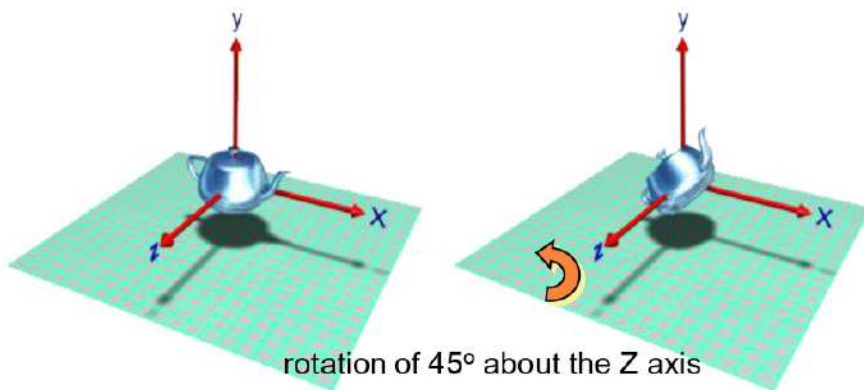
Rotations are orthogonal matrices, preserving distances and angles.

1. Rotation about X-axis $\mathbf{R}_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \theta & \sin \theta & 0 \\ 0 & -\sin \theta & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

2. Rotation about Y-axis $\mathbf{R}_y = \begin{bmatrix} \cos \theta & 0 & -\sin \theta & 0 \\ 0 & 1 & 0 & 0 \\ \sin \theta & 0 & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

3. Rotation about Z-axis $\mathbf{R}_z = \begin{bmatrix} \cos \theta & \sin \theta & 0 & 0 \\ -\sin \theta & \cos \theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$





Rotation



3-D SCALING

- You can change the size of an object using scaling transformation .
- In the scaling process , you either expand or compress the dimensions of the object .
- Scaling can be achieved by multiplying the original coordinates of the object with scaling factor to get the desired result
- The general transformation matrix for scaling is

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



3-D SCALING

- Scaling in x-direction by factor a

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

- Scaling in y-direction by factor e

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

- Scaling in z-direction by factor j

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



3-D SCALING

- Scaling in x,y,z-direction by factor a,e,j-units respectively.

$$\begin{bmatrix} a & 0 & 0 & 0 \\ 0 & e & 0 & 0 \\ 0 & 0 & j & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Uniform scaling in x-direction by factor a, if $a > 1$ then expansion occurs
if $0 < a < 1$ contraction occurs

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

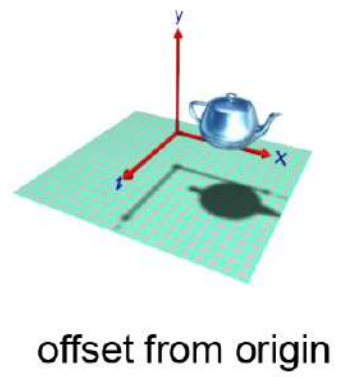
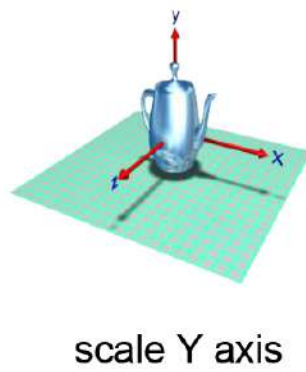
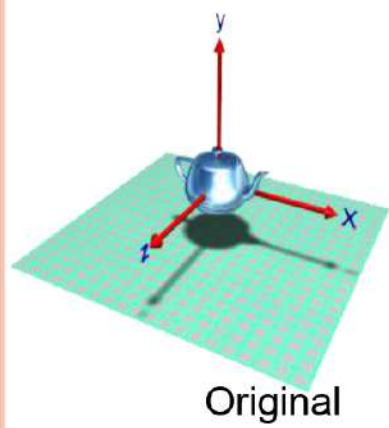
- Overall scaling by s units

Here, $a = 1/s$

if $0 < s < 1$ then expansion occurs

if $s > 1$ then contraction occurs

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



Scaling



3D REFLECTION

- Reflection in computer graphics is used to emulate reflective objects like mirrors and shiny surfaces
- Reflection may be an x-axis y-axis , z-axis. and also in the planes xy-plane,yz-plane , and zx-plane.
- **Note:** Reflection relative to a given Axis are equivalent to 180 Degree rotations



3D REFLECTION

- Reflection about xy-plane(or $z=0$ plane)

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

- Reflection about yz-plane(or $x=0$ plane)

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



3D REFLECTION

- Reflection about zx-plane(or y=0plane)

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



3D SHEARING

- Shearing:

The change in each coordinate is a linear combination of all three

- The general Transformation matrix for shearing in x, y and z-direction is given by,

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & a & b & 0 \\ c & 1 & d & 0 \\ e & f & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$



3D SHEARING

- Shearing in x-direction proportional to y and z-axis by c-units and e-units respectively is

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ c & 1 & 0 & 0 \\ e & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Shearing in y-direction proportional to x and z-axis by a-units and f-units respectively is

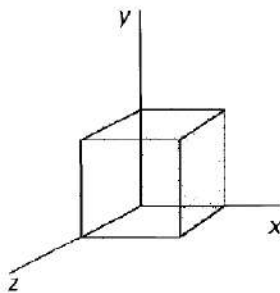
$$\begin{bmatrix} 1 & a & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & f & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



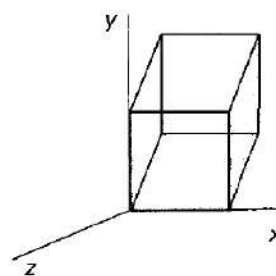
3D SHEARING

- Shearing in z-direction proportional to x and y-axis by b-units and d-units respectively is

$$\begin{bmatrix} 1 & 0 & b & 0 \\ 0 & 1 & d & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



Original object



Shearing in z-direction

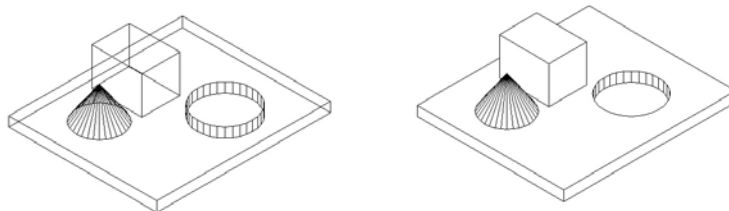
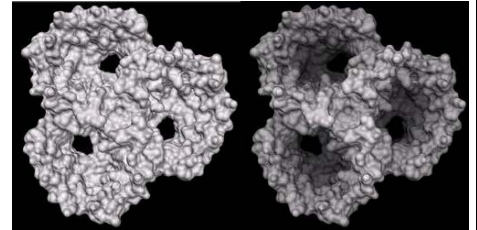
3D-Viewing

■ Concepts of 3D-Viewing

For representing 3D-objects on a 2D-screen in a nice or recognizable way, many techniques are combined. First, the projection has to be defined, which will be described in the next paragraph. After the projection has been set, any of the following qualities can be generated:

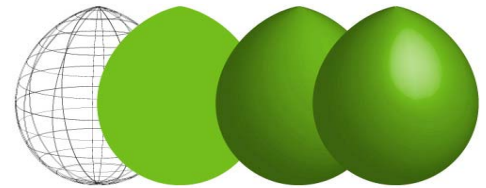
Wire-Frame-Representation = Only the edges of polygons are drawn. Hidden edges may or may not be drawn.

Depth-Cueing = Edges or parts which are nearer to the viewer are displayed with more intensity (brighter, broadened, more saturated), edges which are farther away from the viewer are displayed with less intensity (darker, thinner, grayed out).

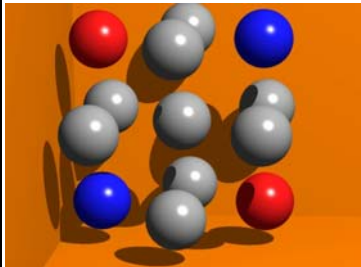


Correct Visibility = Surface-elements (edges, polygons), which are occluded by other surface-elements, are not drawn so that only visible areas are shown.

Shading = Depending on the angle of view or the angle of incident light, surfaces are colored brighter or darker.

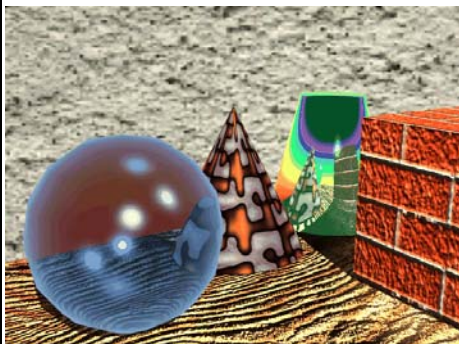


Illumination Models = Physical simulation of lighting conditions and propagation and their influence on the appearance of surfaces.



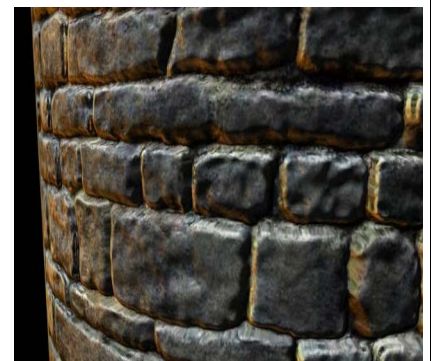
Shadows = Areas which have no line of sight to the light-source are displayed darker.

Reflections, Transparency = Reflecting objects show mirror-images, and through transparent objects the background can be seen.



Textures = Patterns or samples are „painted“ on surfaces to give the objects a more complex look (looks much more realistic then).

Surface Details = Small geometric structures on surfaces (like orange peel, bark, cobblestones, tire profiles) are simulated using tricks.



Stereo Images = A separate image is created and presented (with various techniques) for each eye to generate a 3D-impression.



■ 3D-Viewing-Pipeline

The viewing-pipeline in 3 dimensions is almost the same as the 2D-viewing-pipeline. Only after the definition of the viewing direction and orientation (i.e., of the camera) an additional projection step is done, which is the reduction of 3D-data onto a projection plane:

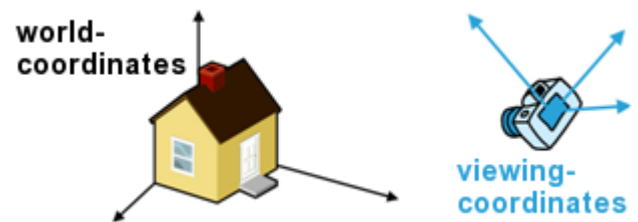


This projection step can be arbitrarily complex, depending on which 3D-viewing concepts should be used.

■ Viewing-Coordinates

Similar to photography there are certain degrees of freedom when specifying the camera:

1. Camera position in space
2. Viewing direction from this position
3. Orientation of the camera (view-up vector)
4. Size of the display window (corresponds to the focal length of a photo-camera)



With these parameters the *camera-coordinate system* is defined (viewing coordinates). Usually the xy-plane of this viewing-coordinate system is orthogonal to the main viewing direction and the viewing direction is *in the direction of the negative z-axis*.

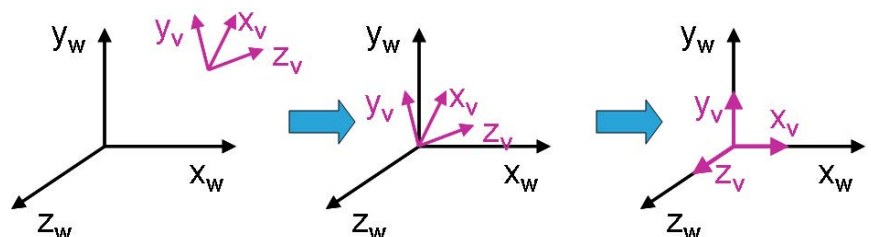
Based on the camera position the usual way to define the viewing-coordinate system is:

1. Choose a camera position (also called eye-point, or view-point).
2. Choose a viewing direction = Choose the z^- direction of the viewing-coordinates.
3. Choose a direction „upwards“. From this, the x-axis and y-axis can be calculated: the image-plane is orthogonal to the viewing direction. The parallel projection of the „view-up vector“ onto this image plane defines the y-axis of the viewing coordinates.
4. Calculate the x-axis as vector-product of the z- and y-axis.
5. The distance of the image-plane from the eye-point defines the viewing angle, which is the size of the scene to be displayed.

In animations the camera-definition is often automatically calculated according to certain conditions, e.g. when the camera moves around an object or in flight-simulations, such that desired effects can be achieved in an uncomplicated way.

To convert world-coordinates to viewing-coordinates a series of simple transformations is needed: mainly a translation of the coordinate origins onto each other and afterwards 3 rotations, such that the coordinate-axes also coincide (two rotations for the first axis, one for the second axis, and the third axis is already correct then). Of course, all these transformations can be merged by multiplication into one matrix, which looks about like this:

$$M_{WC,VC} = R_z \cdot R_y \cdot R_x \cdot T$$



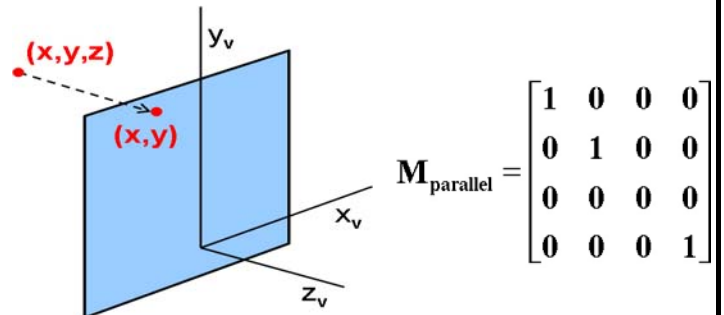
■ Projections

In geometry several types of projections are known. However, in computer graphics mainly only the parallel and the perspective projection are of interest.

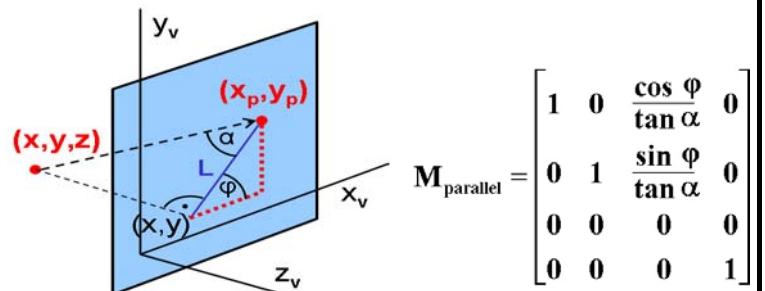
Parallel Projection

A parallel projection can either be done orthogonally to an image-plane, or obliquely (e.g. when casting shadows).

The orthogonal parallel projection is very simple. Assuming that the direction of projection is the z -axis, simply omit the z -value of a point, i.e. set its z -value to zero. The corresponding matrix is therefore also very simple, because the z -coordinate is just replaced by zero.



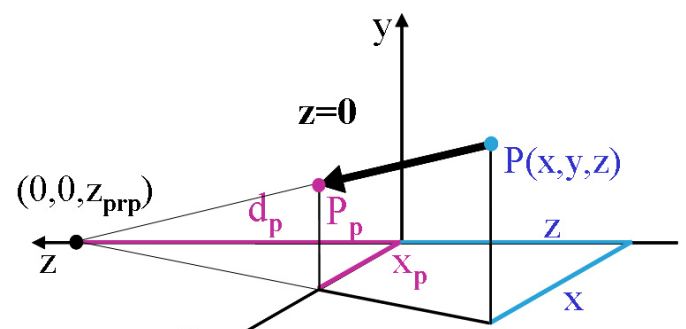
An oblique parallel projection, which is defined by two angles α and ϕ , can be done as follows: for any point the distance L is the distance between two points, the one orthogonally projected onto the z -plane, which results in $(x, y, (0))$, and the result of the oblique projection, which results in $(x_p, y_p, (0))$. This distance L is then one cathetus of the right-angled triangle with vertices (x, y, z) , $(x, y, 0)$ and $(x_p, y_p, 0)$, and at the same time L is the hypotenuse of the also right-angled triangle in the image-plane with the cathetus $(x_p - x)$ and the cathetus $(y_p - y)$. From this $L = z / \tan \alpha$, and $x_p = x + L \cdot \cos \phi$ and $y_p = y + L \cdot \sin \phi$. If L is substituted into the other two equations this produces the given matrix.



Perspective Projection

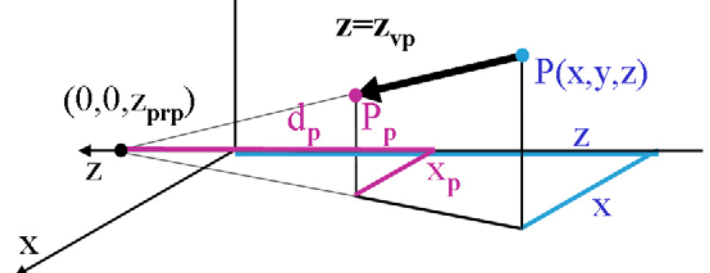
With a perspective projection some laws of affine transformations are not valid anymore (e.g. parallel lines may not be parallel anymore after the perspective projection has been applied). Therefore, since it's not an affine transformation anymore, it can't be described by a 3×3 matrix anymore. Luckily, again homogeneous coordinates can help in this case. However, this is the sole case where the homogeneous component h is not equal to 1 in the result. Therefore, in a subsequent step a division by this value is needed.

Let us first examine the case where the projected image lies on the plane $z=0$. The image of a point $P(x, y, z)$ is the point, where the line through the point itself and the center of projection $(0, 0, z_{\text{prp}})$ cuts the plane $z=0$. When viewed from above, similar right-angled triangles can be found with the catheti d_p and x_p on one hand, and $d_p - z$ (because z is on the negative side of the z -axis) and x on the other hand.



This yields $x_p : x = d_p : (d_p - z)$ or $x_p = x \cdot d_p / (d_p - z)$
and in analogy $y_p : y = d_p : (d_p - z)$ or $y_p = y \cdot d_p / (d_p - z)$
and of course $z_p = 0$.

If we project onto any other plane $z = z_{\text{vp}} \neq 0$ not much has to be changed: only d_p is replaced by the z -value of the center of projection z_{prp} :



$$\begin{aligned}
x_p : x &= d_p : (z_{prp} - z) \text{ or } x_p = x \cdot d_p / (z_{prp} - z) \\
y_p : y &= d_p : (z_{prp} - z) \text{ or } y_p = y \cdot d_p / (z_{prp} - z) \\
z_p &= z_{vp}.
\end{aligned}$$

This can be written as a homogeneous matrix also:

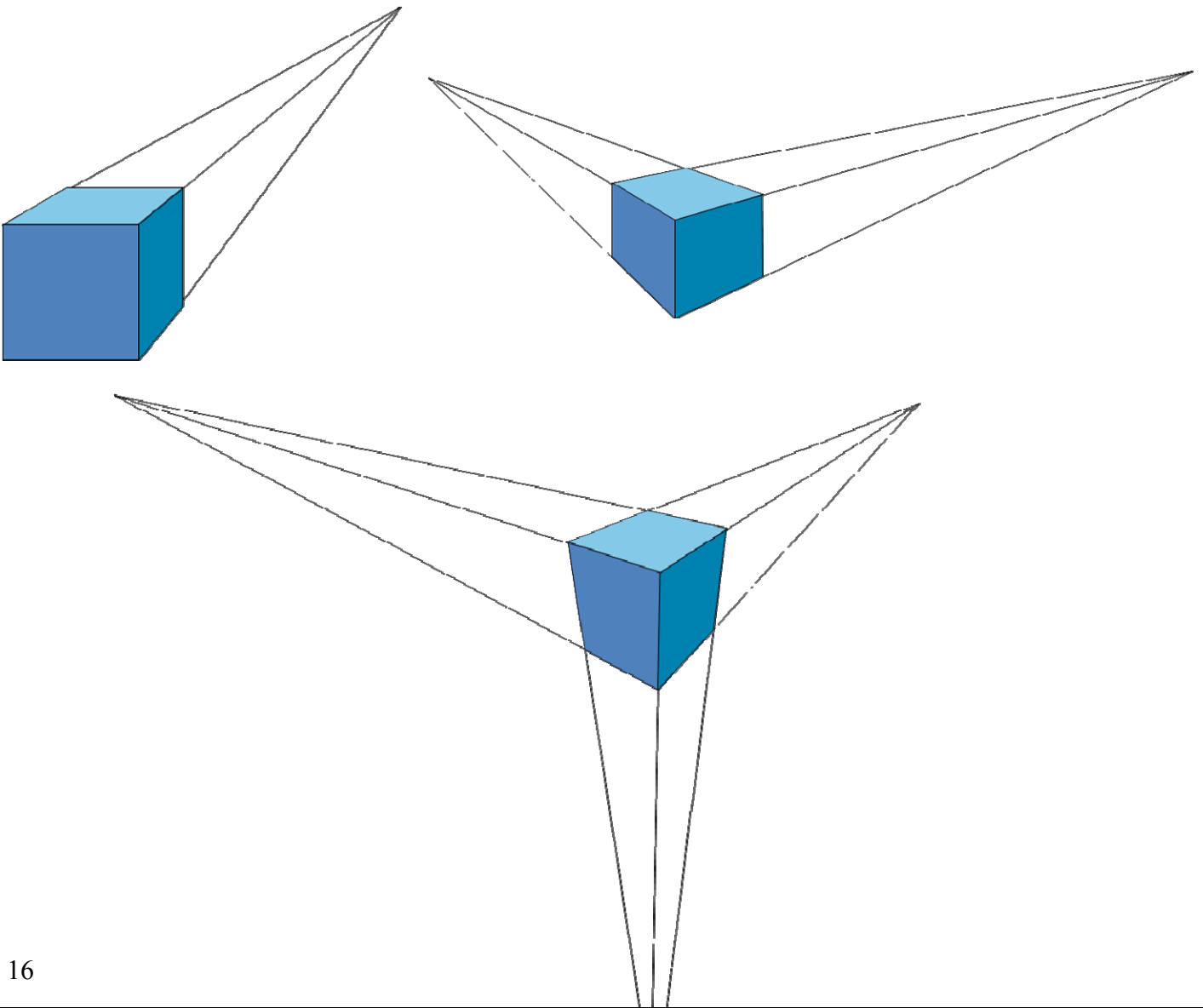
$$\begin{bmatrix} x_h \\ y_h \\ z_h \\ h \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -z_{vp}/d_p & z_{vp}(z_{prp}/d_p) \\ 0 & 0 & -1/d_p & z_{prp}/d_p \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

In general, the value of h of the resulting vector will not be equal to 1 (h is calculated as $(z_{prp} - z)/d_p$). This is the reason why the resulting vector has to be divided by h (in other words, it is scaled with $1/h$) to get the correct resulting point:

$$x_p = x_h/h, y_p = y_h/h \text{ and } z_p \text{ results in } z_{vp}.$$

By representing projections in matrix form, it is possible to formulate the whole transformation from model-coordinates to device-coordinates by multiplying the single matrices to one matrix. Only if a perspective projection is involved, it must not be forgotten to divide the result by the homogeneous component h .

Finally, it should be mentioned that the number of *principal vanishing points* is dependent on how the axes of the coordinate system are positioned against the image plane. If 2 coordinate axes are parallel to the image plane it is called *1-point perspective projection*, if only one is parallel to the image plane we call it *2-point perspective projection*, and if none of the three axes is parallel to the image plane it is called *3-point perspective projection* (because then there are 3 principal vanishing points).



Composite Transformation:

A number of transformations or sequence of transformations can be combined into single one called as composition. The resulting matrix is called as composite matrix. The process of combining is called as concatenation.

Suppose we want to perform rotation about an arbitrary point, then we can perform it by the sequence of three transformations

1. Translation
2. Rotation
3. Reverse Translation

The ordering sequence of these numbers of transformations must not be changed. If a matrix is represented in column form, then the composite transformation is performed by multiplying matrix in order from right to left side. The output obtained from the previous matrix is multiplied with the new coming matrix.

Example showing composite transformations:

The enlargement is with respect to center. For this following sequence of transformations will be performed and all will be combined to a single one

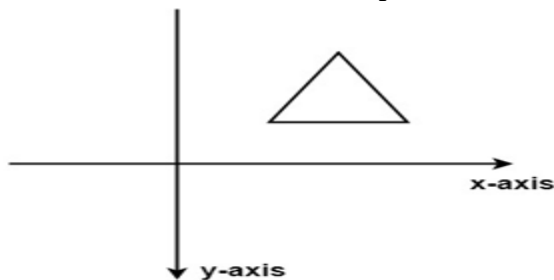
Step1: The object is kept at its position as in fig (a)

Step2: The object is translated so that its center coincides with the origin as in fig (b)

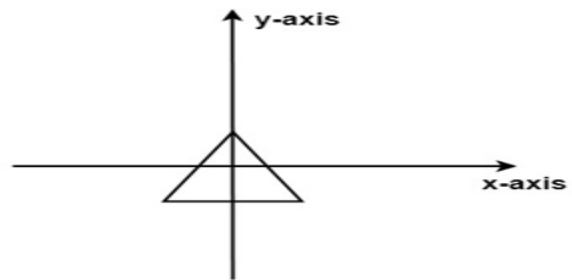
Step3: Scaling of an object by keeping the object at origin is done in fig (c)

Step4: Again translation is done. This second translation is called a reverse translation. It will position the object at the origin location.

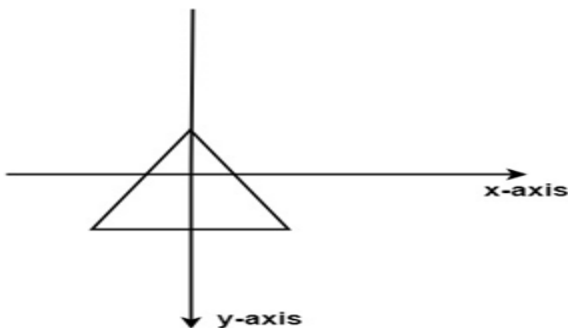
Above transformation can be represented as $T_v \cdot ST_v^{-1}$



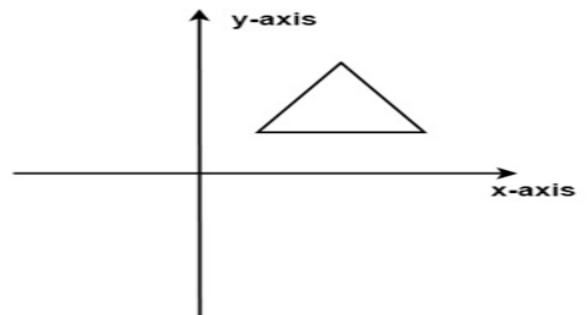
(original position of object)
Fig:(a)



(object is translated to origin)
Fig:(b)



(object is enlarged)
Fig:(c)



(object is retranslated to original position)
Fig:(d)

Row Method

1. Translation

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ t_x & t_y & 1 \end{bmatrix}$$

2. Scaling

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ S_x & S_y & 1 \end{bmatrix}$$

Column Method

1. Translation

$$\begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

2. Scaling

$$\begin{bmatrix} 1 & 0 & S_x \\ 0 & 1 & S_y \\ 0 & 0 & 1 \end{bmatrix}$$

Advantage of composition or concatenation of matrix:

1. It transformations become compact.
2. The number of operations will be reduced.
3. Rules used for defining transformation in form of equations are complex as compared to matrix.

Composition of two translations:

Let t_1 t_2 t_3 t_4 are translation vectors. They are two translations P_1 and P_2 . The matrix of P_1 and P_2 given below. The P_1 and P_2 are represented using Homogeneous matrices and P will be the final transformation matrix obtained after multiplication.

$$P_1 = \begin{bmatrix} 1 & 0 & t_1 \\ 0 & 1 & t_2 \\ 0 & 0 & 1 \end{bmatrix} \quad P_2 = \begin{bmatrix} 1 & 0 & t_3 \\ 0 & 1 & t_4 \\ 0 & 0 & 1 \end{bmatrix}$$

$$P = \begin{bmatrix} 1 & 0 & t_1 + t_2 \\ 0 & 1 & t_3 + t_4 \\ 0 & 0 & 1 \end{bmatrix}$$

Above resultant matrix show that two successive translations are additive.

Composition of two Rotations: Two Rotations are also additive

Composition of two Scaling: The composition of two scaling is multiplicative. Let S_{11} and S_{12} are matrix to be multiplied.

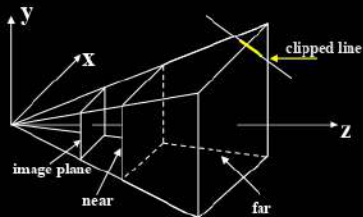
$$S_{11} = \begin{bmatrix} S_1 & 0 & 0 \\ 0 & S_2 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$S_{12} = \begin{bmatrix} S_3 & 0 & 0 \\ 0 & S_4 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$S = S_{11} * S_{12} = \begin{bmatrix} S_1 * S_3 & 0 & 0 \\ 0 & S_2 * S_4 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Clipping

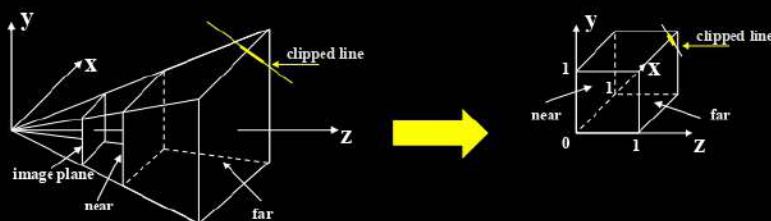
- There is something missing between projection and viewing...
- Before projecting, we need to eliminate the portion of scene that is outside the viewing frustum



- Need to clip objects to the frustum (truncated pyramid)
- Now in a canonical position but it still seems kind of tricky...

Normalizing the Viewing Frustum

- Solution: transform frustum to a cube before clipping



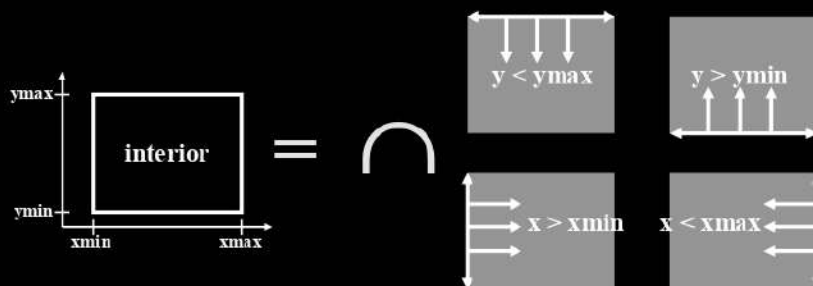
- Converts perspective frustum to orthographic frustum
- This is yet another homogeneous transform!

Clipping to a Cube

- Determine which parts of the scene lie within cube
- We will consider the 2D version: clip to rectangle
- This has its own uses (viewport clipping)
- Two approaches:
 - clip during scan conversion (rasterization) - check per pixel or end-point
 - clip before scan conversion
- We will cover
 - clip to rectangular viewport before scan conversion

Line Clipping

- Modify endpoints of lines to lie in rectangle
- How to define “interior” of rectangle?
- Convenient definition: intersection of 4 half-planes
 - Nice way to decompose the problem
 - Generalizes easily to 3D (intersection of 6 half-planes)



INTRODUCTION

- ▮ The term **computer animation** generally refers to any time sequence of visual changes in a scene.
- ▮ In addition to changing object position with translations or rotations, a computer-generated animation could display time variations in object size, color, transparency, or surface texture.
- ▮ Computer animations can be generated by changing camera parameters, such as position, orientation, and focal length.
- ▮ Computer animations can also be produced by changing lighting effects.

DESIGN OF ANIMATION SEQUENCES

An animation sequence is designed with the following steps:

- ❑ Storyboard layout
- ❑ Object definitions
- ❑ Key-frame specifications
- ❑ Generation of in-between frames

STORYBOARD LAYOUT

- ▮ It is an outline of the action.
- ▮ It defines the motion sequence as a set of basic events that are to take place.
- ▮ Depending on the type of animation to be produced, the storyboard could consist of a set of rough sketches or it could be a list of the basis ideas for the motion

OBJECT DEFINATION

- ▮ An object definition is given for each participant in the action.
- ▮ Objects can be defined in terms of basic shapes, such as polygons or splines.
- ▮ Along with the shape, the associated movements for each object are specified.

KEY FRAME SPECIFICATION

- ▮ A key frame is a detailed drawing of the scene at a certain time in the animation sequence.
- ▮ Within each key frame, each object is positioned according to the time for that frame.
- ▮ Some key frames are chosen at extreme positions in the action; others are spaced so that the time interval between key frames is not too great.

IN-BETWEEN FRAMES

- ▮ It is a process of generating intermediate frames between two images to give appearance that the first image evolves smoothly into the second image. In-betweens are the drawing between the key frames which help to create the illusion of motion.

- ▮ In-between are the intermediate frames between the key frames
- ▮ The number of in-betweens needed is determined by the media to be used to display the animation. Film requires 24 frames per second, and graphics terminals are refreshed at the rate of 30 to 60 frames per second
- ▮ Time intervals for the motion are set up so that there are from three to five in-betweens for each pair of key frames.

There are several other tasks that may be required, depending on the application. They include:

- ❑ Motion verification
- ❑ Editing
- ❑ Production and synchronization of a soundtrack

GENERAL COMPUTER ANIMATION FUNCTIONS

- ▮ Some steps included the development of an animation sequence are:
 - ❑ Object manipulations and rendering
 - ❑ Camera motions
 - ❑ Generation of in-betweens
- ▮ Animation packages, *such as Wave-front*, provide special functions for designing the animation and processing individual objects.
- ▮ In animation packages, one function is provided *to store and manage the object database*. Object shapes are stored and updated in the database.
- ▮ Other object functions include those *for motion generation and object rendering*. Motions can be generated according to specified constraints using two-dimensional or three-dimensional transformations.
- ▮ Another function *simulates camera movements*. Standard motions are zooming, panning, and tilting.

GENERAL COMPUTER-ANIMATIONFUNCTIONS

- Animation packages ,such as wave front , provide special functions for designing the animation and processing individual objects.
- Some steps included in the development of animation sequence are-
 - Object manipulation and rendering
 - Camera motions .
 - Generation of in-betweens .

- **ONE FUNCTION AVAILABLE IN ANIMATION PACKAGES IS PROVIDED TO STORE AND MANAGE THE OBJECT DATABASE.**
(OBJECT SHAPES AND ASSOCIATED PARAMETERS ARE STORED AND UPDATED IN THE DATABASE).

- **Other object functions include:-**

- **Object motion generation(2-D or 3-D transformations)**
- **Object rendering** .

- **One function to stimulate Camera Movements:-**

- **Zooming**
- **Panning (rotating horizontally or vertically)**
- **Tilting** .

RASTER ANIMATIONS

On Raster systems ,we generate real-time animation in limited application Using raster operation.

Such as 2-D or 3-D transformations.

We can also animate objects along 2D motion paths using the color-table transformations.

The pixel value at successive positions along the motion path of an object are stored in color-table and the pixel at 1st pixel is set on , we set the pixel at the other object positions to the background color.

KEY-FRAME SYSTEMS

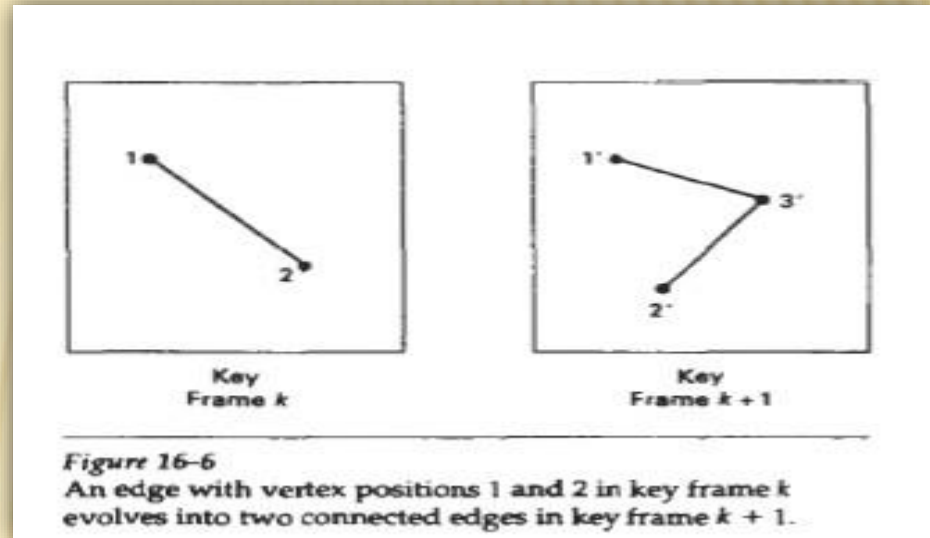
A key-frame in animation is a drawing that defines the starting and ending points of any smooth transition. The drawings are called "frames" because their position in time is measured in frames on a strip of film. A sequence of key frames defines which movement the viewer will see, whereas the position of the key frames on the film, video or animation defines the timing of the movement. Because only two or three key frames over the span of a second do not create the illusion of movement, the remaining frames are filled with in-betweens.

With complex object transformations, the shapes of objects may change over time. Examples are clothes, facial features, magnified detail, evolving shapes, exploding or disintegrating objects, and transforming one object into another object.

If all surfaces are described with polygon meshes, then the number of edges per polygon can change from one frame to the next. Thus, the total number of line segments can be different in different frames.

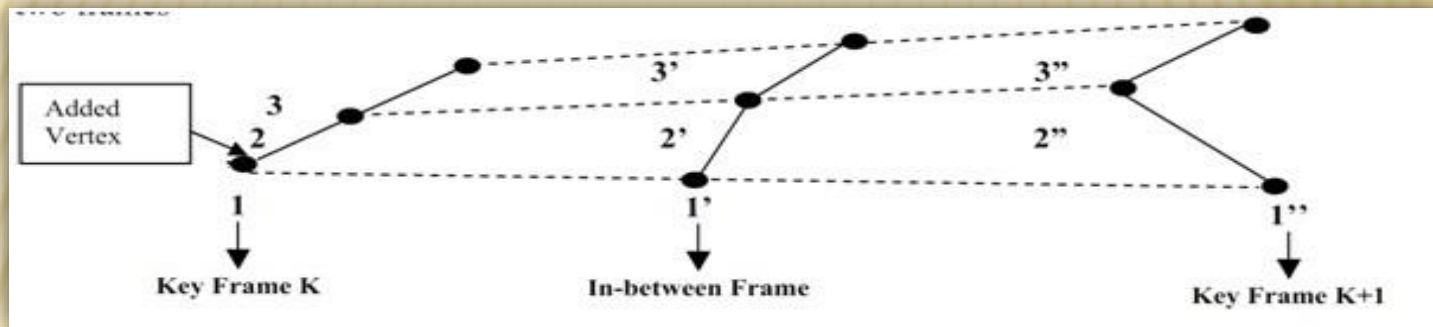
MORPHING

Transformation of object shapes from one form to the other is termed as morphing as short form of metamorphism. This method can be applied to any of motion or transition relating a change in shape.

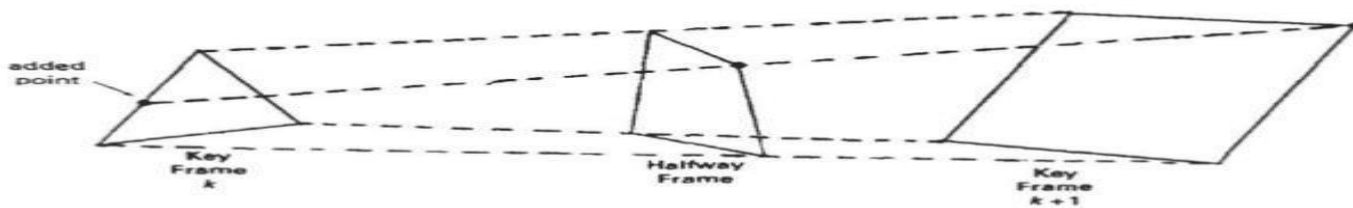


Given two key frames for an object transformation, we first adjust the object specification in one of the frames so that the number of polygon edges (or the number of vertices) is the same for the two frames

A straight-line segment in key frame k is transformed into two line segments in key frame $k + 1$. Since key frame $k + 1$ has an extra vertex, we add a vertex between vertices 1 and 2 in key frame k to balance the number of vertices (and edges) in the two key frames. Using linear interpolation to generate the in-betweens, we transition the added vertex in key frame k into vertex $3'$ along the straight-line path shown in Fig.



Morphing



An example of a triangle linearly expanding into a quadrilateral given in Fig

The general pre processing rules for equalizing key frames in terms of either the number of vertices to be added to a key frame.

Suppose we equalize the **edge count** and parameters L_k and L_{k+1} denote the number of line segments in two consecutive frames. We define,

$$L_{\max} = \max (L_k, L_{k+1})$$

$$L_{\min} = \min(L_k, L_{k+1})$$

$$N_e = L_{\max} \bmod L_{\min}$$

$$N_s = \text{int} (L_{\max}/L_{\min})$$

The pre processing is accomplished by

- Dividing N_e edges of key frame_{min} into N_s+1 section.
- Dividing the remaining lines of key frame_{min} into N_s sections.

For example, if $L_k = 15$ and $L_{k+1} = 11$, we divide 4 lines of key-frame_{k+1} into 2 sections each. The remaining lines of keyframe_{k+1} are left intact.

If we equalize the **vertices counts** then the parameters V_k and V_{k+1} are used to denote the number of vertices in the two consecutive frames. In this case we define:

$$V_{\max} = \max(V_k, V_{k+1})$$

$$V_{\min} = \min(V_k, V_{k+1}) \text{ and}$$

$$N_{ls} = (V_{\max} - 1) \bmod (V_{\min} - 1)$$

$$N_p = \text{int}((V_{\max} - 1) / (V_{\min} - 1))$$

Preprocessing using vertex count is performed by:

- Adding N_p points to N_{ls} line section of key-frame_{min}.
- Adding $N_p - 1$ points to the remaining edges of key-frame_{min}.

For the triangle-to quadrilateral example, $V_k = 3$ and $V_{k+1} = 4$.

Both N_{ls} and N_p are 1, so we would add one point to one edge of key-frame_k

No points would be added to the remaining lines of keyframe_{k+1}.

SIMULATING ACCELERATIONS

Curve-fitting techniques are often used to specify the animation paths between key frames. Given the vertex positions at the key frames, we can fit the positions with linear or nonlinear paths.

This figure illustrates a nonlinear fit of key-frame positions. To simulate accelerations, we can adjust the time spacing for the in-betweens.

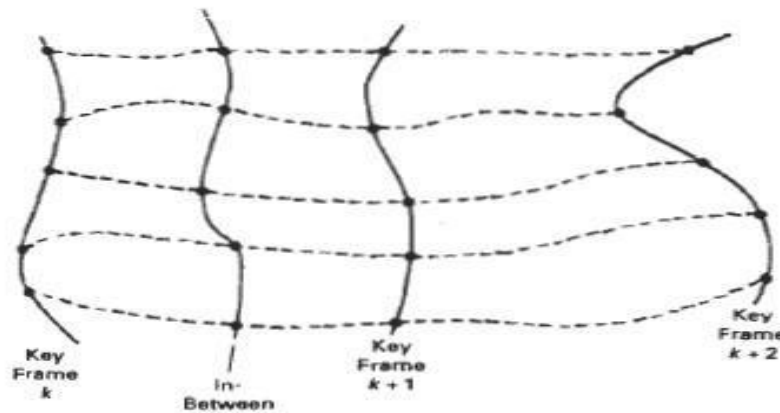


Figure 16-11
Fitting key-frame vertex positions with nonlinear splines.

For constant speed (zero acceleration), we use equal-interval time spacing for the in-betweens. Suppose we want n in-betweens for key frames at times t_1 and t_2 . The time interval between key frames is then divided into $n + 1$ subintervals, yielding an in-between spacing of

$$\Delta t = \frac{t_2 - t_1}{n + 1}$$

We can calculate the time for any in-between as

$$tB_j = t_1 + j \Delta t, \quad j = 1, 2, \dots, n$$

Nonzero accelerations are used to produce realistic displays of speed changes, particularly at the beginning and end of a motion sequence. We can model the start-up and slowdown portions of an animation path with spline or trigonometric functions. Parabolic and cubic time functions have been applied to acceleration modelling, but trigonometric functions are more commonly used in animation packages.

To model increasing speed (positive acceleration), we want the time spacing between frames to increase so that greater changes in position occur as the object moves faster. We can obtain an increasing interval size with the function

$$1 - \cos\theta, \quad 0 < \theta < \pi/2$$

For n in-betweens, the time for the j th in-between would then be calculated as:

where Δt the time difference between the two key frames

$$tB_j = t_1 + \Delta t \left[1 - \cos \frac{j\pi}{2(n+1)} \right], \quad j = 1, 2, \dots, n$$

This figure gives a plot of the trigonometric acceleration function and the in-between spacing for $n = 5$.

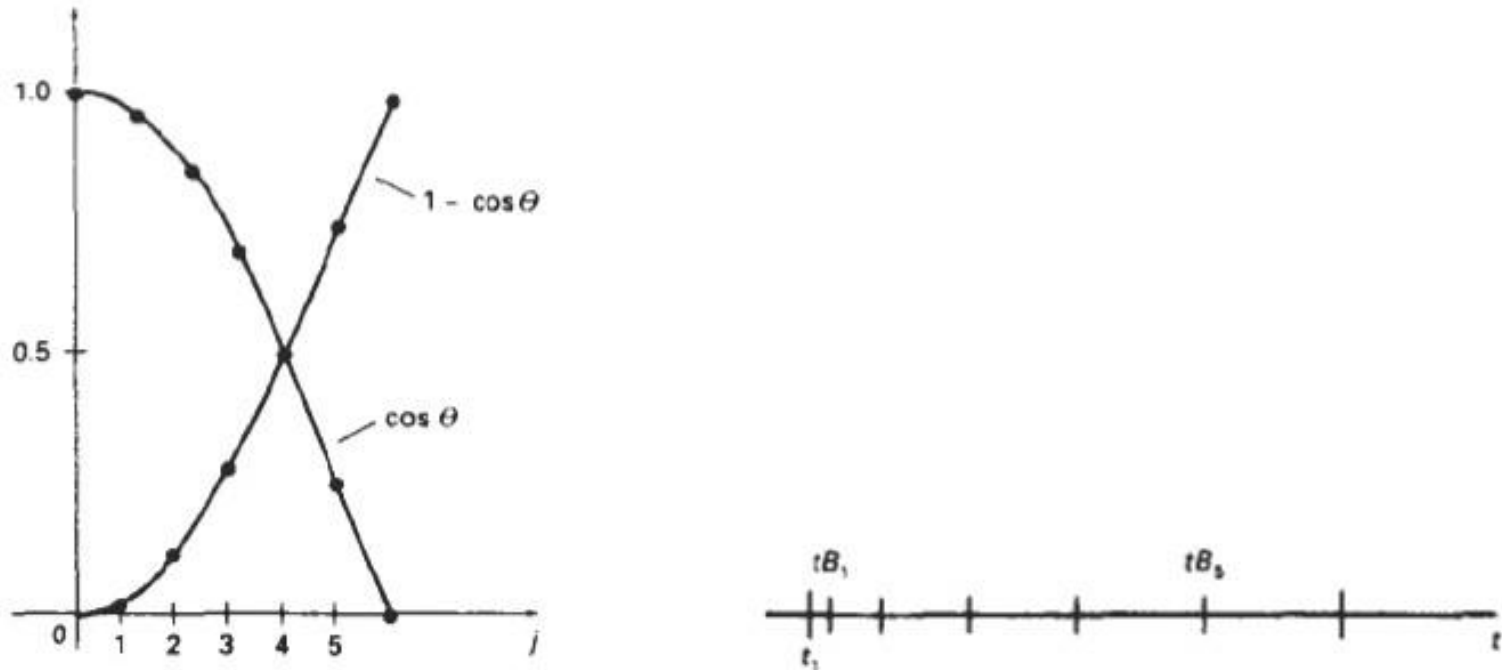


Figure 16-13

A trigonometric acceleration function and the corresponding in-between spacing for $n = 5$ and $\theta = j\pi/12$ in Eq. 16-7, producing increased coordinate changes as the object moves through each time interval.

Often, motions contain both speed-ups and slow-downs. We can model a combination of increasing-decreasing speed by first increasing the in-between time spacing, then we decrease this spacing. A function to accomplish these time changes is

$$\frac{1}{2}(1 - \cos \theta), \quad 0 < \theta < \pi/2$$

The time for the j th in-between is now calculated as:

$$tB_j = t_1 + \Delta t \left\{ \frac{1 - \cos[j\pi/(n+1)]}{2} \right\}, \quad j = 1, 2, \dots, n$$

with Δt denoting the time difference for the two key frames. Time intervals for the moving object first increase, then the time intervals decrease, as shown in next figure:

We can model decreasing speed (deceleration) with $\sin\theta$ in the range $0 < \theta < \pi/2$. The time position of an in-between is now defined as:

$$tB_j = t_1 + \Delta t \sin \frac{j\pi}{2(n+1)}, \quad j = 1, 2, \dots, n$$

A plot of this function and the decreasing size of the time intervals is shown in the next figure for five in-betweens

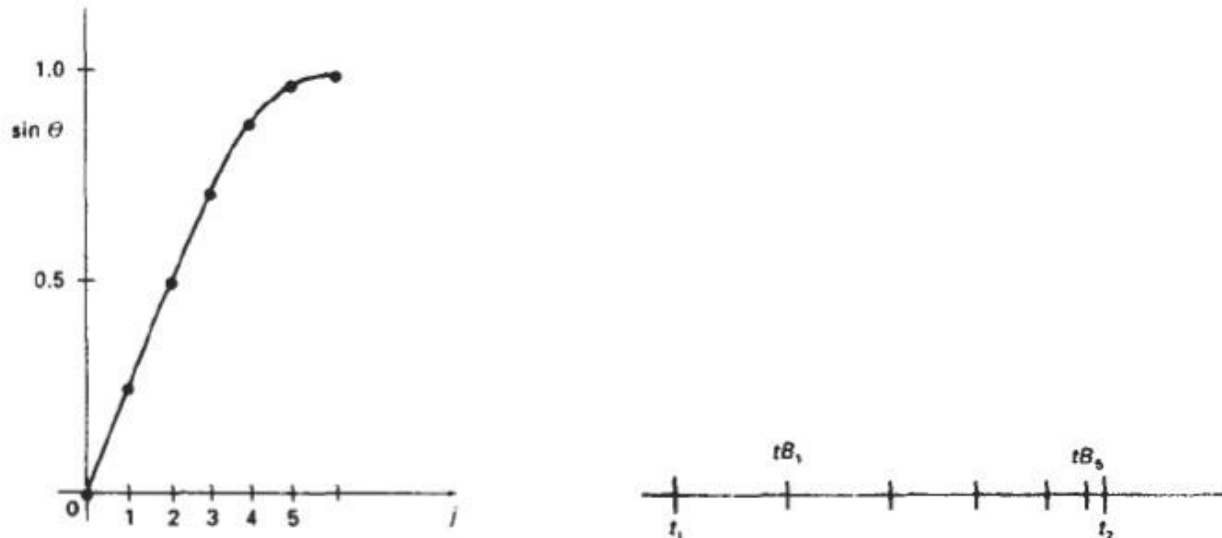


Figure 16-14

A trigonometric deceleration function and the corresponding in-between spacing for $n = 5$ and $\theta = j\pi/12$ in Eq. 16-8, producing decreased coordinate changes as the object moves through each time interval.

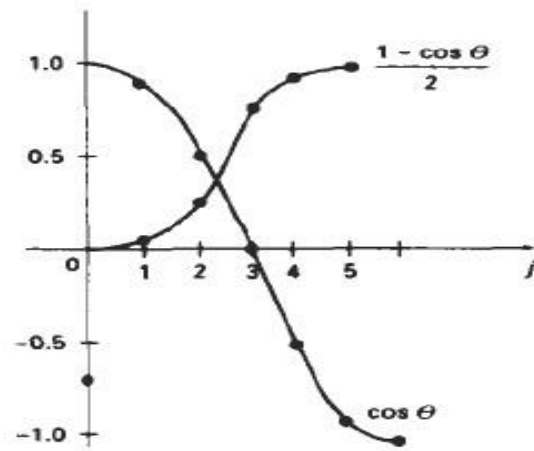


Figure 16-15

A trigonometric accelerate-decelerate function and the corresponding in-between spacing for $n = 5$ in Eq. 16-9.