COMPUTER GRAPHICS

Introduction

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

- Computer graphics generally means creation, storage and manipulation of models and
- mages. 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 methds he was pursuing at Boeing. He created a series of widely reproduced images on pen plotte 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 various fields such as advertising, entertainment, medicine, education, science, engineering and navigation. It helps in creating, modifying and displaying pictures using computers.

1. Computer-Aided Design (CAD)

Computer-Aided Design (CAD) is used to design buildings, automobiles, aircraft, ships, spacecraft, computers, textiles and many other products. It is a useful tool for architects to prepare drawings and visualize building structures. A computer can generate images of a building from different angles by using the given design data.

Animations are also used in CAD applications. Real-time animations using wireframe displays on a video monitor help engineers test the performance of vehicles, view their interiors and observe the behaviour of internal components during motion.

2. Presentation Graphics

Presentation Graphics is used to prepare illustrations and diagrams for reports. It is commonly used to summarize financial, statistical, mathematical, scientific and economic data for research reports, managerial reports and consumer information.

Typical examples of presentation graphics are bar charts, line graphs, surface graphs and pie charts. These graphical representations help users understand the relationship between different parameters. Time charts and task network layouts are also used in project management to schedule activities and monitor project progress.

3. Computer Art

Computer Graphics is widely used in both fine arts and commercial arts. Artists use special-purpose hardware, painting programs, drawing software, mathematical packages, CAD packages, desktop publishing software and animation packages to create different designs.

Fine artists use three-dimensional modelling, texture mapping, drawing programs and CAD software to produce attractive images. These techniques are also used in commercial art for designing logos, page layouts, advertisements, television commercials and other graphical designs.

One important technique used in commercial art is morphing, in which one object or image is gradually transformed into another. For example, a television advertisement may show an oil can transforming into an automobile engine.

4. Entertainment

Computer Graphics plays an important role in entertainment, including motion pictures, music videos, television programmes and animated films. It is used to create special effects, animated characters, realistic backgrounds and attractive visual scenes.

Computer Graphics and image-processing techniques are used to combine graphical objects with live-action videos. Morphing techniques are also used to transform one person or object into another. Many films and television series use Computer Graphics to improve visual quality and provide realistic effects.

5. Medical Applications

Computer Graphics and image-processing techniques are widely used in the medical field to examine and analyse the human body. They help improve the quality, brightness, contrast and colour of medical images.

Tomography is a technique used to obtain cross-sectional images of internal body structures. Computerized Axial Tomography (CAT) uses X-rays and computer processing to produce images of internal organs, such as the brain, and helps doctors detect problems like brain tumours.

Ultrasonic scanning uses sound waves to generate digital images of internal body structures. Nuclear medicine scanners collect digital data from radiation emitted by radioactive substances introduced into the body and produce colour-coded images. Computer-aided surgery also uses computer-based graphical systems to assist doctors in planning and performing surgical procedures.

6. Graphical User Interface (GUI)

A Graphical User Interface (GUI) allows users to interact with a computer through graphical elements instead of typing commands for every operation. An important component of a GUI is the window manager, which allows multiple windows to be displayed on the screen.

Each window can contain a different process and display graphical or non-graphical information. GUI systems also use menus and icons to help users select commands and processing options.

An icon is a graphical symbol designed to represent a program, file, command or operation. A menu contains a list of textual descriptions, icons or commands from which the user can select the required option.

The main challenge of Computer Graphics is to create a virtual world that looks and sounds real, moves naturally, responds to user interaction in real time and provides a realistic experience.

7. Education and Training

Computer Graphics is used as an educational aid to explain complex concepts through pictures, animations and computer-generated models. Models of physical systems, financial systems, economic systems, physiological systems and population trends help students understand how different systems work.

Specialized simulation systems are used to train ship captains, aircraft pilots, heavy-equipment operators and air traffic-control personnel. These simulators provide a realistic training environment without requiring the actual equipment. Educational animations and graphical presentations also make learning easier and more effective.

8. Image Processing

Image Processing involves applying techniques to modify, improve or interpret existing pictures, photographs and television scans. It is used to improve picture quality and to extract useful information from images.

The two principal applications of image processing are improving picture quality and enabling machine perception of visual information. In robotics, image-processing techniques help computers and robots analyse visual data and recognize objects. These techniques are also used in medical imaging, satellite photography and the analysis of photographs of the Earth and galaxies.

OVERVIEW OF GRAPHICS SYSTEMS

Computer Graphics is a complex and diversified technology used to create, store, modify and display pictures. The final output of a Computer Graphics system is generally a picture, which may be used for various purposes such as engineering drawings, scientific visualisation, animation and education.

The picture is the fundamental concept in Computer Graphics. The major activities involved in Computer Graphics are representing pictures, preparing pictures for presentation, presenting previously prepared pictures and interacting with pictures.

A picture may consist of points, lines, text, polygons and other graphical elements displayed on a graphics device. These elements are used to create simple as well as complex images

Representing Pictures

A unit square can be represented by its four corner points P_1, P_2, P_3, P_4. The associated algorithm connects these points in sequence to form the square.

The unit square can also be represented by its four edges E_1, E_2, E_3, E_4. In this method, the algorithm connects the edges in sequence to complete the square.

The fundamental building blocks of Computer Graphics are points. Points can be represented by pairs or triplets of numbers, depending on whether the data belongs to two-dimensional (2D) or three-dimensional (3D) space. Thus, (X,Y) represents a point in 2D space, whereas (X,Y,Z) represents a point in 3D space.

Two points represent a line or edge, while a collection of three or more points can represent a polygon. Curved lines are generally represented by approximating them with connected short straight-line segments.

The representation of text is more complex because characters may contain curved lines or dot matrices. Fundamentally, text can also be represented using collections of lines and points, together with an organizing algorithm.

Preparing Pictures

Pictures consist of points and a drawing algorithm used to display them. The information required to create a picture is generally stored in a file called a database.

Complex pictures require large databases and algorithms to access and process the stored information. These databases may organize data using different structures, such as ring structures, B-tree structures and quadtree structures. A database may also contain pointers, substructures and other non-graphical information.

Many Computer Graphics applications use simpler pictures that can be represented using simple data structures, such as a linear list. These structures are easy to access and are sufficient for many reasonably complex pictures.

The three fundamental operations used to manipulate points are:

1. Move: Move the beam, pen, cursor or plotting head invisibly to a specified point.
2. Draw: Draw a visible line from the initial point to another point.
3. Display: Display a dot at a specified point.

The position of a point can be specified using two methods: absolute coordinates and relative (incremental) coordinates. In absolute coordinates, the exact position of a point is specified. In relative coordinates, the position is defined by giving the displacement from the previous point. These fundamental operations form the basis of Computer Graphics software.

Presenting Previously Prepared Pictures

The data used to display a picture is frequently called a display file. A display file represents a particular position, view or scene of the picture stored in the complete database.

The displayed picture is usually generated by performing operations such as rotation, translation, scaling and projection on the stored data. These operations are generally performed using a 4 × 4 transformation matrix that operates on data represented in homogeneous coordinates.

Additional effects, such as hidden-line removal, hidden-surface removal, shading, transparency, texture and colour, may be applied before the final picture is displayed.

If the entire picture stored in the database is not required, only the necessary portion is selected. This process is called clipping, and it can be performed in both two-dimensional and three-dimensional graphics.

Two important concepts related to picture presentation are windows and viewports.

Windowing: Windowing is the process of selecting a required portion of a picture by clipping the database to the boundaries of a window.

Viewport: A viewport is the area of the display device on which the selected window contents are displayed.

Interacting with Pictures

Interacting with pictures allows users to communicate with a computer graphics system and modify displayed graphical objects. This improves the user's ability to understand data, identify trends and visualize real or imaginary objects.

Users can perform operations such as selecting, moving, resizing, rotating and editing graphical objects. These interactions are widely used in CAD systems, graphical user interfaces, simulations and design applications.

Video Display Devices

Video display devices are output devices used to display pictures, text and graphical information generated by a computer. The important types of video display devices are:

1. Cathode Ray Tubes (CRT)
2. Raster-Scan Displays
3. Vector-Scan or Random-Scan Displays
4. Colour CRT Monitors
5. Direct-View Storage Tubes (DVST)
6. Flat-Panel Displays
7. Three-Dimensional Viewing Devices

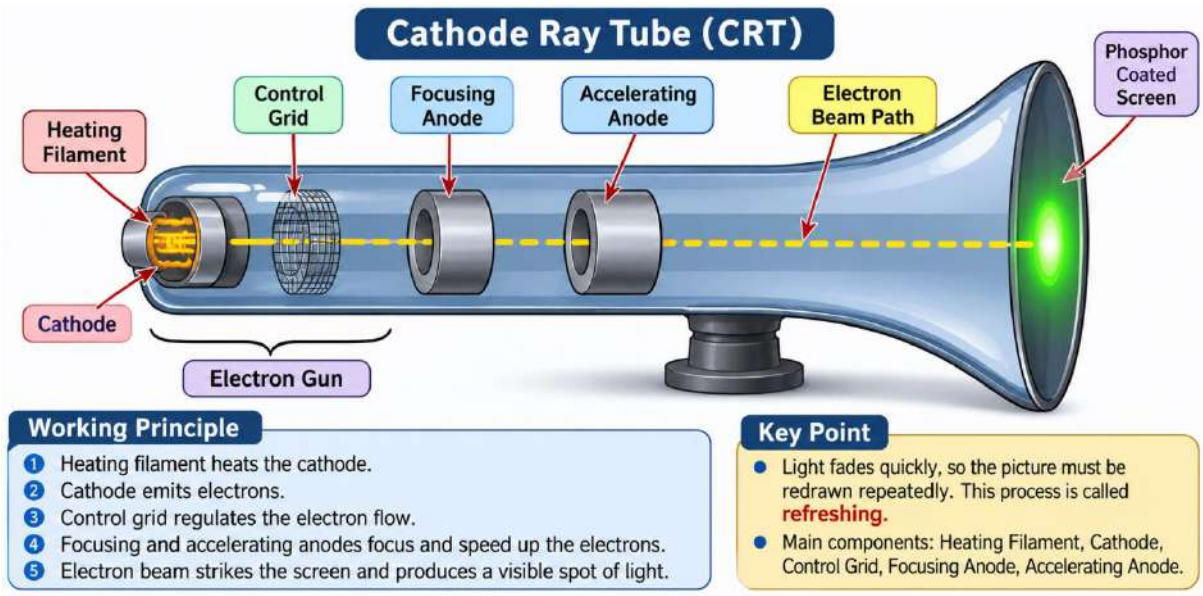
Cathode Ray Tube (CRT)

A Cathode Ray Tube (CRT) is a display technology traditionally used in computer graphics systems and video monitors. Its operation is based on an electron beam that strikes a phosphor-coated screen to produce visible light.

The electron gun emits a beam of electrons, which passes through focusing and deflection systems. These systems direct the beam towards specific positions on the screen. When the electron beam strikes the phosphor coating, a small spot of light is produced.

The light emitted by the phosphor fades rapidly. Therefore, the picture must be redrawn repeatedly by directing the electron beam over the same points to maintain the display. This process is called refreshing, and the display is known as a refresh CRT.

The main components of an electron gun are the heated metal cathode and the control grid. The cathode produces electrons when heated, while the control grid controls the intensity of the electron beam.



Working of Cathode Ray Tube (CRT)

The cathode is heated by passing an electric current through a coil of wire called the filament. Due to this heat, electrons are emitted from the surface of the cathode. This process is called thermionic emission.

The free negatively charged electrons inside the CRT are accelerated towards the phosphor-coated screen by a high positive voltage. The positively charged metal coating inside the CRT helps accelerate the electrons towards the screen.

The intensity of the electron beam is controlled by adjusting the voltage applied to the control grid, which is fitted near the cathode. A high negative voltage applied to the control grid repels the electrons and stops the beam from passing through its small opening. A lower negative voltage allows more electrons to pass through the grid.

The brightness of the display depends on the number of electrons striking the phosphor coating. When more electrons hit the screen, the spot becomes brighter. Therefore, the brightness is controlled by varying the voltage applied to the control grid. Graphics software specifies the intensity level for different screen positions.

An electron beam emitted by the electron gun passes through the focusing and deflection systems, which direct it towards specific positions on the phosphor-coated screen. When the beam strikes the screen, the phosphor emits a small spot of light.

Since the light emitted by the phosphor fades quickly, the picture must be redrawn repeatedly by directing the electron beam over the same points. This process is called refreshing, and the display is known as a refresh CRT.

The focusing system is used to converge the electron beam into a small spot when it strikes the phosphor-coated screen. Without proper focusing, the beam spreads out because the electrons repel one another. Focusing is achieved by using either electric fields or magnetic fields.

Type of Focusing	Meaning
1. Electrostatic Focusing	In electrostatic focusing, the electron beam passes through a positively charged metal cylinder that produces an electrostatic field. This field focuses the beam at the required point on the screen.
2. Magnetic Focusing	In magnetic focusing, a magnetic field is produced by a coil mounted around the outside of the CRT envelope. This magnetic field focuses the electron beam onto the screen.

Electrostatic Deflection of Electron Beam in CRT

This diagram shows the arrangement of an electron gun, focusing system, horizontal deflection plates, vertical deflection plates and phosphor-coated screen in a Cathode Ray Tube (CRT).

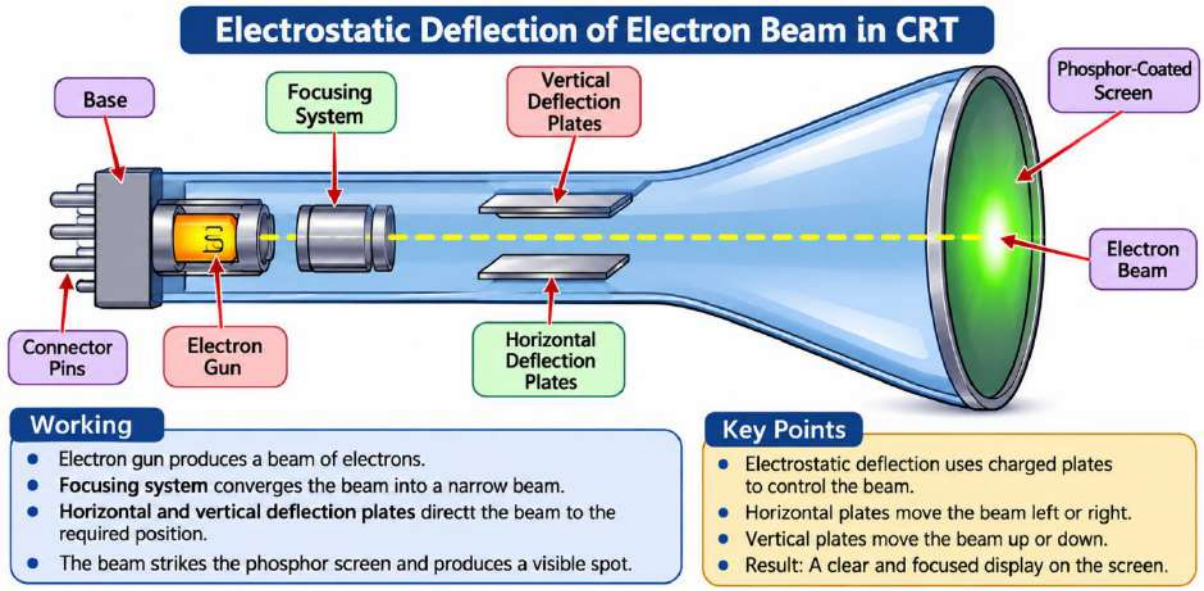
Explanation:

The electron gun produces a beam of electrons. The focusing system converges the electrons into a narrow beam. The horizontal and vertical deflection plates control the direction of the electron beam and guide it towards the required position on the screen. When the electron beam strikes the phosphor-coated screen, it produces a visible spot of light.

Main components:

1. Base: Supports the connections of the CRT.
2. Connector Pins: Provide electrical connections to the CRT.
3. Electron Gun: Produces the electron beam.
4. Focusing System: Focuses the electron beam into a narrow beam.
5. Horizontal Deflection Plates: Deflect the electron beam horizontally.

6. Vertical Deflection Plates: Deflect the electron beam vertically.
7. Phosphor-Coated Screen: Produces visible light when struck by the electron beam.



Properties of Phosphors in CRT

Different types of phosphors are available for use in a Cathode Ray Tube (CRT). The main properties in which phosphors vary are persistence, resolution, and aspect ratio.

Term	Description
1. Persistence	Persistence is the time required for the emitted light from the screen to decay to one-tenth (1/10) of its original intensity.
2. Resolution	Resolution refers to the maximum number of points that can be displayed on the CRT screen. It can also be expressed as the number of points per centimetre that can be plotted horizontally and vertically.
3. Aspect Ratio	Aspect ratio is the ratio of vertical points to horizontal points required to produce lines of equal length in both directions on the screen.

Raster-Scan Display

A Raster-Scan Display is the most common type of graphics monitor that uses a Cathode Ray Tube (CRT). It is based on television display technology.

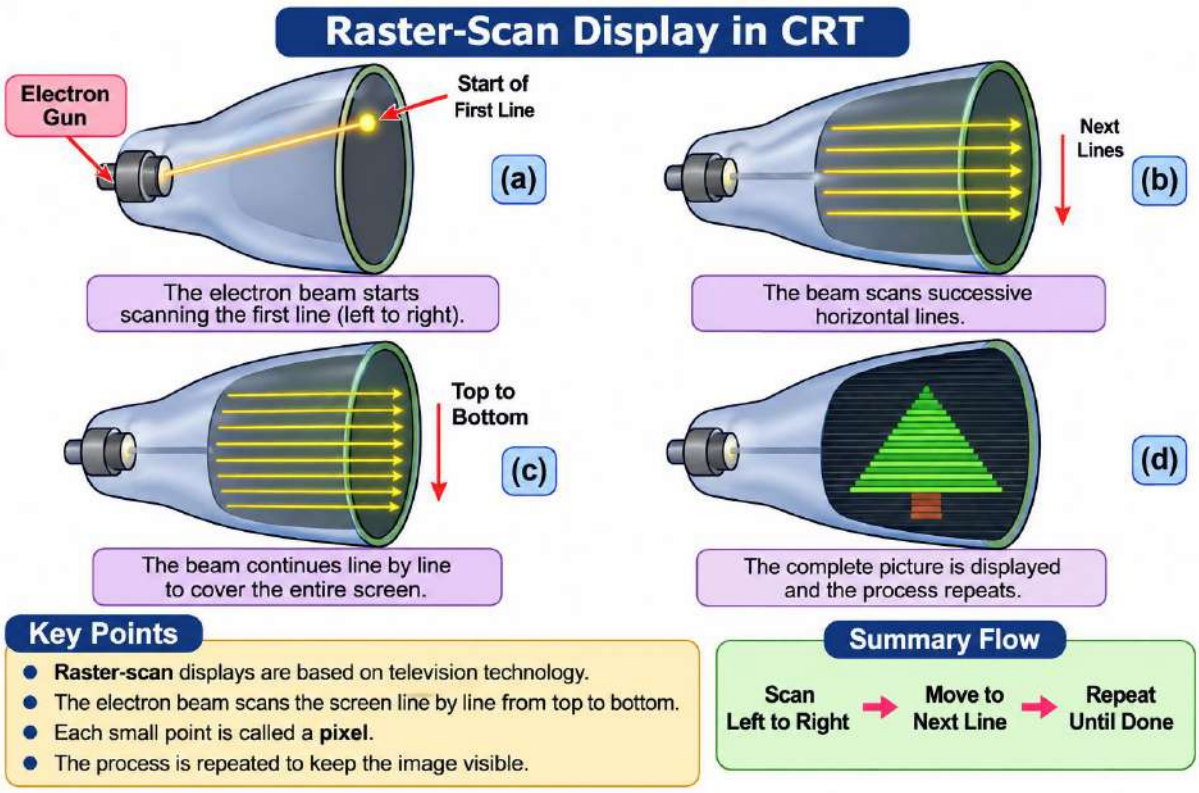
In a raster-scan system, the electron beam from the electron gun moves horizontally across the phosphor-coated screen, one row at a time, from top to bottom. When the electron beam strikes the phosphor, it produces light and displays the required picture.

The screen is divided into small picture elements called pixels. The intensity and colour of each pixel are controlled to form the complete image.

The electron beam scans the screen line by line. After completing one row, it moves to the next row until the entire screen is covered. This process is repeated continuously to maintain the displayed image.

Raster graphics are commonly used in televisions, computer monitors, digital photographs and images such as JPEG files.

Example: A digital photograph displayed on a computer screen is a raster image made up of many pixels.



Working of Raster-Scan Display

In a raster-scan display, the electron beam moves horizontally across the screen, one row at a time. Each horizontal movement is called a scan line. After completing one horizontal sweep, the beam moves to the next row and continues scanning from top to bottom.

After the bottom line is scanned, the beam returns to the top of the screen, and the scanning process begins again. This process is called refreshing.

As the electron beam moves across each row, its intensity is switched on and off to create a pattern of illuminated spots that forms the picture.

The picture information is stored in a memory area called the refresh buffer or frame buffer. Each screen point is called a pixel (picture element). The frame buffer stores the intensity values of all the pixels and is part of the computer's memory. These stored values are retrieved and displayed on the screen one row at a time.

The intensity range of each pixel depends on the capabilities of the raster system. In a black-and-white system, each pixel is either ON or OFF. Therefore, one bit per pixel is sufficient to represent the two states: 1 for ON and 0 for OFF.

In colour systems, more bits are generally required to represent different colours and intensity levels. The number of bits depends on the colour depth of the system.

Raster-scan displays are refreshed continuously, typically at a rate of 60 to 80 frames per second in traditional systems, to maintain a stable and visible picture.

Terms Used in Raster-Scan Display

Term	Description
1. Bitmap	In a black-and-white system with one bit per pixel, the frame buffer is called a bitmap. Each pixel can have one of two values: ON (1) or OFF (0).
2. Pixmap	A frame buffer that stores multiple bits per pixel, usually for colour displays, is called a pixmap. It can represent different colours and intensity levels.
3. Horizontal Retrace	Horizontal retrace is the process in which the electron beam returns to the left side of the screen after completing each scan line, ready to scan the next line.
4. Vertical Retrace	Vertical retrace is the process in which the electron beam returns from the bottom of the screen to the top-left corner to begin displaying the next frame.

Interlaced Refreshing

Interlacing the scan lines allows the entire screen to be displayed in two passes. Instead of scanning all the lines in a single pass, the electron beam scans alternate lines in the first pass and the remaining lines in the second pass.

Interlacing is mainly used in systems with slower refresh rates. For example, an older non-interlaced display operating at 30 frames per second may show noticeable flickering. With interlacing, the two passes occur at different times, making the screen update more frequently and helping reduce flicker.

This technique is effective when adjacent scan lines contain similar display information.

Frame Buffer Size

The frame buffer size depends on the screen resolution and the number of bits used to represent each pixel.

Formula:

Frame buffer size = Horizontal resolution × Vertical resolution × Bits per pixel

For example, for a display resolution of 800 × 600 pixels with 8 bits per pixel:

Frame buffer size = 800 × 600 × 8 = 3,840,000 bits

Therefore, the frame buffer requires 480,000 bytes (approximately 469 KB).

Advantages of Raster-Scan Displays

1. A high degree of realism can be achieved by using advanced shading and hidden-surface removal techniques.
2. Decreasing memory costs have made raster-scan systems more economical.
3. Raster-scan technology is widely used in computer monitors and television sets.
4. It is suitable for displaying filled areas, coloured images and realistic pictures.

Disadvantages of Raster-Scan Displays

1. Raster displays may have limited resolution compared with some vector displays.
2. Lines and edges may appear zigzagged because pixels are discrete points on the screen. This effect is called aliasing.
3. High-quality images require more memory to store pixel information.
4. Enlarging raster images can reduce image quality and make individual pixels visible.

Random-Scan Display

A Random-Scan Display is a type of graphics display system also known as a Vector Display, Stroke-Writing Display, or Calligraphic Display. In this system, the electron beam directly draws the picture by moving along the required lines.

A pen plotter works in a similar way and is an example of a random-scan hard-copy device.

In a random-scan display, the electron beam is directed only towards the portions of the screen where the picture is to be drawn. The electron gun illuminates points and draws straight lines in any required order. The picture is drawn one line at a time by moving the beam between specified endpoints.

The picture information is stored as a set of line-drawing commands in a memory area called the Refresh Display File. It is also known as a display list, display program, or refresh buffer.

To display a picture, the system executes the commands in the display file and draws each component line in sequence. After all the commands have been processed, the system returns to the first command and repeats the process to refresh the picture.

The refresh rate depends on the number of lines to be displayed. Typically, random-scan systems refresh simple pictures about 30 to 60 times per second. These displays are mainly designed for line-drawing applications, such as engineering drawings and scientific diagrams.

Random-scan displays produce smooth lines because the electron beam follows the required line paths directly. In contrast, raster-scan displays represent lines using individual pixels, which may produce jagged edges.

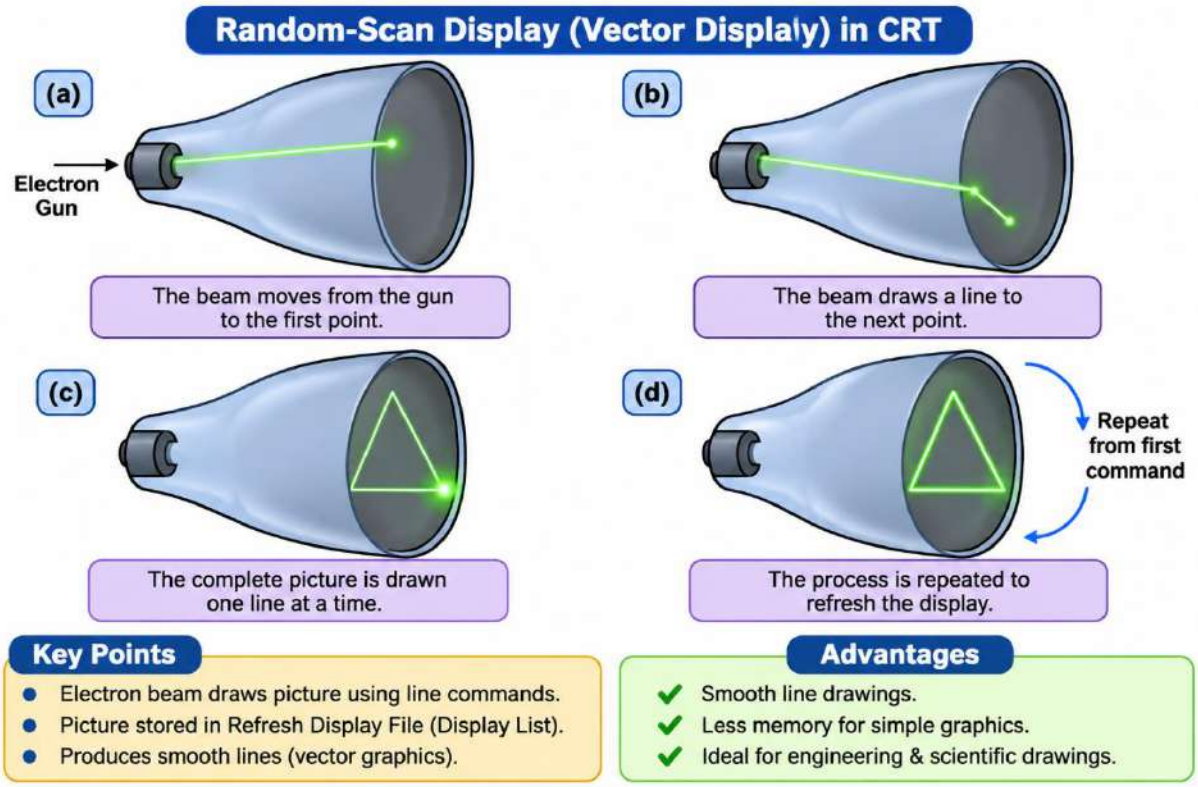
Advantages of Random-Scan Displays

1. Produce smooth and high-quality line drawings.

2. Require less memory for pictures containing a small number of lines.
3. Suitable for engineering and scientific drawings.
4. Provide high-quality output for wireframe models.

Disadvantages of Random-Scan Displays

1. Not suitable for displaying realistic shaded images and complex filled areas.
2. Refreshing becomes slower when the picture contains many lines.
3. Colour and shading capabilities are limited in traditional systems.
4. They are less suitable for photographs and detailed images than raster-scan displays.



Color CRT Monitors

Color pictures can be displayed on a CRT monitor by using phosphors that emit different colors of light. By combining the light emitted by different phosphors, a wide range of colors can be produced.

The two basic techniques used to produce color displays in CRT monitors are the Beam-Penetration Method and the Shadow-Mask Method.

1. Beam-Penetration Method

The beam-penetration method is mainly used in random-scan monitors to display color pictures. In this method, two layers of phosphor, usually red and green, are coated on the inside of the CRT screen. The color displayed depends on how deeply the electron beam penetrates these phosphor layers.

A beam of slow-moving electrons excites only the outer red phosphor layer and produces red light. A beam of very fast-moving electrons penetrates the red layer and excites the inner green phosphor layer, producing green light.

At intermediate electron-beam speeds, combinations of red and green light produce two additional colors: orange and yellow. The speed of the electrons is controlled by the beam-acceleration voltage.

The beam-penetration method is an inexpensive way to produce colors in random-scan monitors. However, it can produce only a limited number of colors, typically four, and the picture quality is lower than that of other color-display methods.

2. Shadow-Mask Method

The shadow-mask method is commonly used in raster-scan systems, including color television sets, because it can produce a much wider range of colors than the beam-penetration method.

In a shadow-mask CRT, each pixel position contains three phosphor dots: red, green, and blue (RGB). Each phosphor dot emits its corresponding color when struck by an electron beam.

This type of CRT uses three electron guns, one for each primary color. A metal sheet containing tiny holes, called the shadow mask, is placed just behind the phosphor-coated screen. The shadow mask directs each electron beam toward its corresponding red, green, or blue phosphor dot.

By varying the intensities of the red, green, and blue beams, different colors can be produced. When all three colors are combined at maximum intensity, they produce white light. By adjusting their intensities, the monitor can display millions of different colors.

The shadow-mask method provides better color reproduction and is suitable for displaying realistic pictures, photographs, and detailed graphics.

Difference Between Beam-Penetration and Shadow-Mask Methods

Beam-Penetration Method	Shadow-Mask Method
Mainly used in random-scan monitors.	Commonly used in raster-scan monitors.
Uses two phosphor layers, usually red and green.	Uses red, green, and blue phosphor dots.
Uses electron-beam speed to control color.	Uses three electron guns and controls their intensities.
Produces a limited number of colors.	Produces a wide range of colors.
Inexpensive and has lower picture quality.	Provides better color reproduction and picture quality.

Shadow-Mask Method

The shadow-mask method is commonly used in raster-scan display systems, including color television, because it can produce a much wider range of colors than the beam-penetration method.

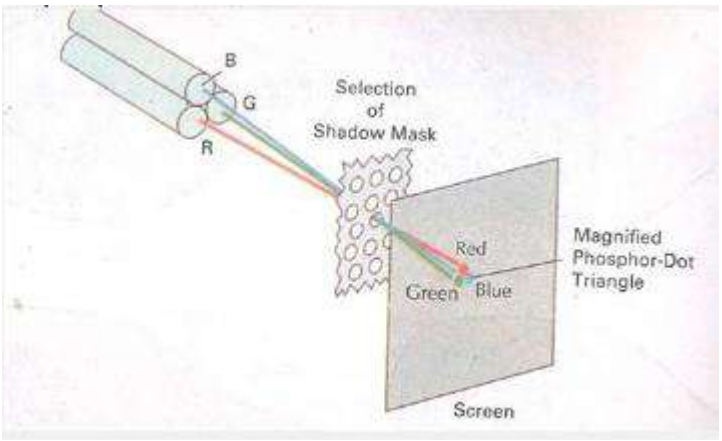
In a shadow-mask CRT, each pixel position contains three phosphor dots. One dot emits red light, the second emits green light, and the third emits blue light. These three colors are known as the RGB primary colors.

A shadow-mask CRT consists of three electron guns, one for each color: red, green, and blue. A metal sheet containing tiny holes, called the shadow mask, is placed just behind the phosphor-coated screen.

The shadow mask ensures that each electron beam strikes only its corresponding phosphor dot. When the electron beams hit the phosphors, they produce red, green, and blue light.

By varying the intensity of these three beams, different colors can be produced. When red, green, and blue light are combined at maximum intensity, they produce white light. By changing their intensities, a wide range of colors can be displayed.

Therefore, the shadow-mask method is suitable for displaying color pictures, photographs, and realistic graphics.



Shadow-Mask Method (Continued)

In a shadow-mask CRT, the three electron beams are deflected together toward the shadow mask. The shadow mask contains a series of tiny holes aligned with the phosphor-dot patterns on the screen.

When the three electron beams pass through a hole in the shadow mask, they strike a group of three phosphor dots arranged in a triangle. This group is called a phosphor-dot triangle. The three dots emit red, green, and blue light, respectively, producing a small color spot on the screen.

The phosphor dots are arranged so that each electron beam activates only its corresponding color dot. This arrangement ensures that the correct colors are produced on the screen.

In-Line Arrangement

Another arrangement of the three electron guns is called the in-line arrangement. In this method, the three electron guns and their corresponding red, green, and blue phosphor dots are arranged in a straight line instead of a triangular pattern.

This arrangement is easier to keep in alignment and is commonly used in high-resolution color CRT monitors.

Color Generation in Shadow-Mask CRTs

Different colors are produced by varying the intensity of the three electron beams. Each beam controls the intensity of its corresponding red, green, or blue phosphor.

For example, if the red and green electron guns are turned off, only the blue phosphor emits light, producing blue. Similarly, different combinations of red, green, and blue intensities produce different colors. Our eyes combine the light from the three phosphor dots and perceive it as a single color.

The following table shows how different colors are produced by combining the RGB primary colors.

Color	Red	Green	Blue
Red	On	Off	Off
Green	Off	On	Off
Blue	Off	Off	On
Yellow	On	On	Off
Magenta	On	Off	On
Cyan	Off	On	On
White	On	On	On
Black	Off	Off	Off

When all three phosphor dots are activated with equal intensity, they produce white or gray, depending on the intensity level. By changing the intensity of each beam independently, a wide range of colors can be generated.

In some low-cost systems, each electron beam can only be switched on or off. This allows a maximum of eight color combinations. More advanced systems support intermediate intensity levels, enabling them to display millions of colors.

Video Display Devices

Color graphics systems can be designed to work with different types of CRT display devices, such as color television sets, composite monitors, and RGB monitors.

1. Color Television Sets

Some inexpensive home-computer systems and video games are designed to use a color television set as the display device. An RF (Radio-Frequency) modulator converts the computer's video output into a signal that can be transmitted through the television's antenna input.

However, the picture quality may be lower than that of a dedicated computer monitor.

2. Composite Monitors

Composite monitors are adapted television displays that allow the computer to bypass the television's broadcast-reception circuitry.

In these monitors, picture information is combined into a single signal called a composite video signal. The monitor separates this signal to display the picture. Although composite monitors are convenient and relatively inexpensive, their picture quality is generally lower than that of RGB monitors.

3. RGB Monitors

RGB monitors are specially designed for computer graphics systems. They use the shadow-mask method and receive separate intensity signals for the red, green, and blue electron guns directly from the computer.

Since the three color signals are handled separately, RGB monitors generally provide better color reproduction and picture quality than composite monitors.

High-quality raster-graphics systems commonly use 24 bits per pixel in the frame buffer. This provides 8 bits for each of the three primary colors.

- Red: $2^8 = 256$ intensity levels
- Green: $2^8 = 256$ intensity levels
- Blue: $2^8 = 256$ intensity levels

The total number of possible color combinations is:

$$256 \times 256 \times 256 = 16,777,216$$

1. Direct-View Storage Tubes (DVST)

A Direct-View Storage Tube (DVST) is a type of CRT display that stores picture information inside the tube instead of refreshing the screen repeatedly. The picture information is stored as a charge distribution behind the phosphor-coated screen.

A DVST uses two electron guns: the primary gun and the flood gun. The primary gun is used to store the picture pattern, while the flood gun helps maintain the displayed picture.

In a conventional refresh CRT, the picture must be redrawn repeatedly to keep it visible. In a DVST, the picture information is stored inside the CRT, so continuous refreshing is not required.

Advantages of DVST

1. Flat screen: It provides a relatively flat viewing surface.
2. No refreshing required: The picture remains visible without repeated refreshing.
3. High resolution: Very complex pictures can be displayed at high resolutions.
4. No flicker: Since continuous refreshing is not required, screen flicker is avoided.

Disadvantages of DVST

1. Poor contrast: The displayed picture generally has lower contrast.
2. Slower performance: Its performance is inferior to that of refresh CRT displays for interactive applications.
3. No selective erasing: A particular part of the picture cannot be erased independently.
4. Limited color support: Traditional DVSTs generally do not support color displays.
5. Slow modification: To remove or modify part of a picture, the entire screen must be erased and the modified picture redrawn. This process may take several seconds for complex pictures.

Because of these limitations, DVST displays have largely been replaced by raster-scan display systems.

2. Flat-Panel Displays

A flat-panel display is a type of video display device designed to have less volume, weight, and power consumption than a traditional CRT monitor. Flat-panel displays are thin and lightweight, making them suitable for wall mounting and portable electronic devices.

Flat-panel displays are used in television monitors, calculators, handheld video games, laptop computers, airline seat-back entertainment systems, elevator advertising boards, and portable graphics displays.

Flat-panel displays are classified into two main categories: emissive displays and non-emissive displays.

2.1 Emissive Displays

Emissive displays are devices that convert electrical energy directly into light to produce images. Examples include plasma panels, thin-film electroluminescent displays, and light-emitting diodes (LEDs).

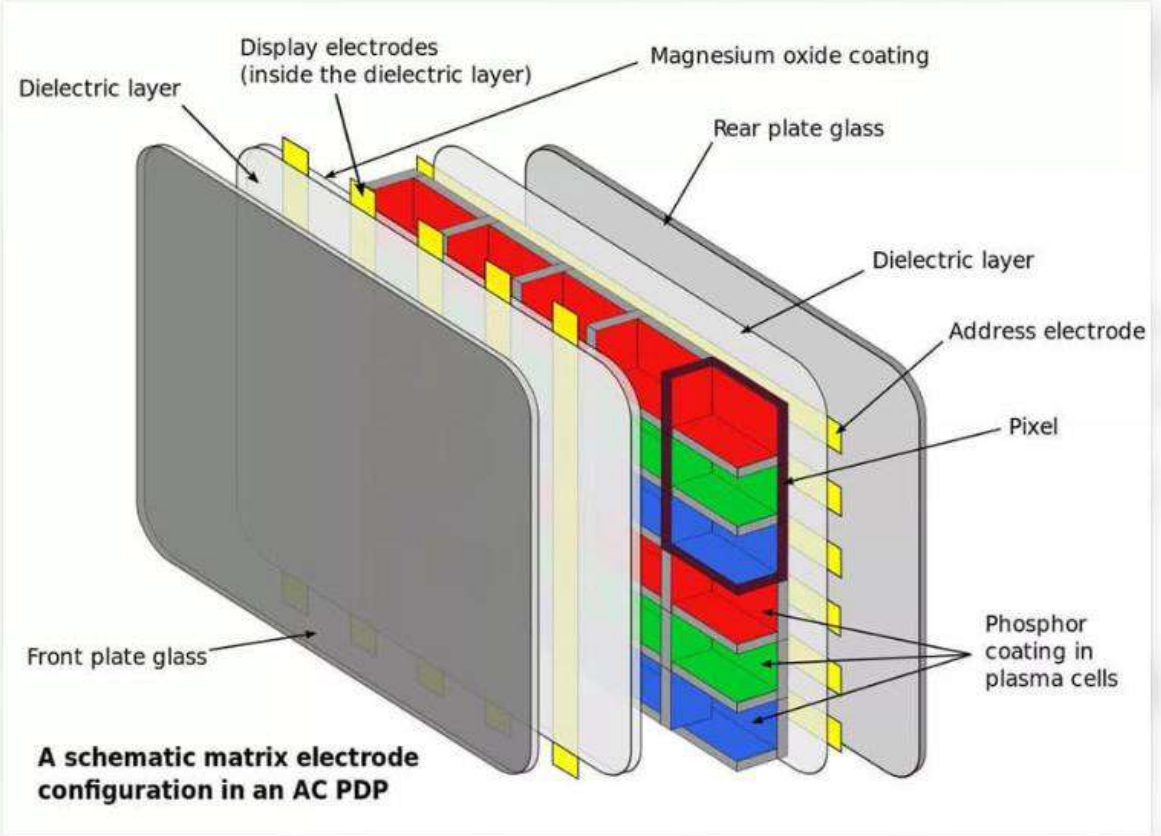
Flat CRTs have also been developed. In these displays, electron beams are accelerated parallel to the screen and then deflected by 90 degrees toward it. However, flat CRTs have not been as successful as other emissive display technologies.

2.2 Non-Emissive Displays

Non-emissive displays do not produce light directly. Instead, they use optical effects to control or modify light from an external source, such as sunlight or a backlight, to create graphics and images.

The most important example of a non-emissive flat-panel display is the Liquid Crystal Display (LCD). LCDs are widely used in laptops, calculators, monitors, and other electronic devices because of their thin design and low power consumption.

2.3 Plasma Panels



Plasma panels, also called gas-discharge displays, are constructed by filling the space between two glass plates with a mixture of gases, usually including neon and other suitable gases.

A series of vertical conducting ribbons is placed on one glass plate, while a set of horizontal conducting ribbons is placed on the other glass plate. These conducting ribbons act as electrodes.

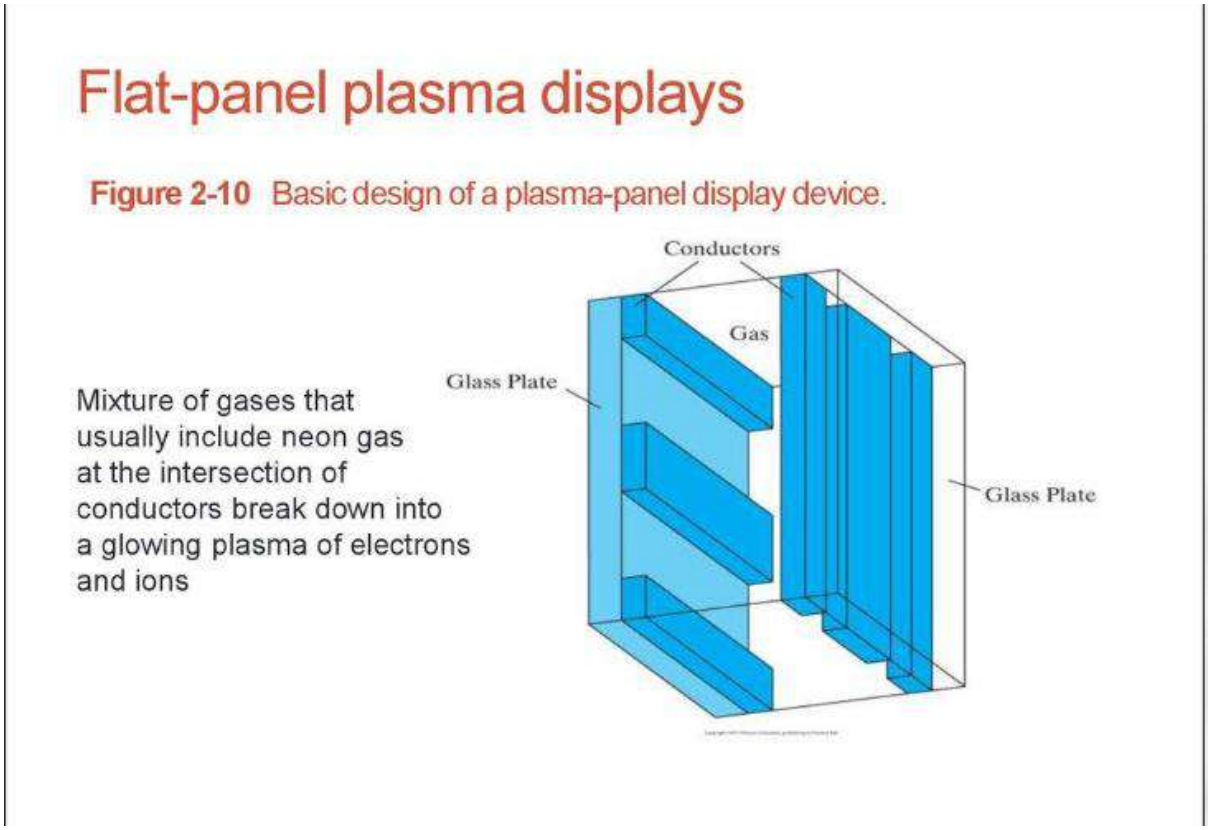
When a suitable voltage is applied to selected horizontal and vertical electrodes, the gas in the corresponding cell becomes ionized and produces an electrical discharge. This discharge generates ultraviolet radiation, which excites phosphor material and causes it to emit visible light.

By controlling the electrodes and the intensity of light produced by the cells, the display forms images on the screen.

Plasma-Panel Displays

Plasma panels, also called gas-discharge displays, are constructed by filling the space between two glass plates with a mixture of gases, usually including neon.

A series of vertical conducting ribbons is placed on one glass plate, while a set of horizontal conducting ribbons is placed on the other glass plate. These conductors are arranged to form a grid, as shown in Figure 8.



When firing voltages are applied to a selected horizontal conductor and a vertical conductor, the gas at their intersection becomes ionized and forms glowing plasma consisting of electrons and ions. This produces light at the corresponding pixel position on the screen.

The picture information is stored in a refresh buffer. The firing voltages are applied to the required pixel positions to refresh the display. In traditional plasma displays, the refresh process is repeated approximately 50 times per second.

Alternating-current methods can be used to apply firing voltages more effectively, helping to produce brighter displays. The electric fields associated with the conductors help control the discharge at individual pixel positions.

One disadvantage of early plasma panels was that they were mainly monochromatic, meaning they displayed only one color. Later developments made it possible for plasma displays to support color and grayscale images.

Thin-Film Electroluminescent Displays

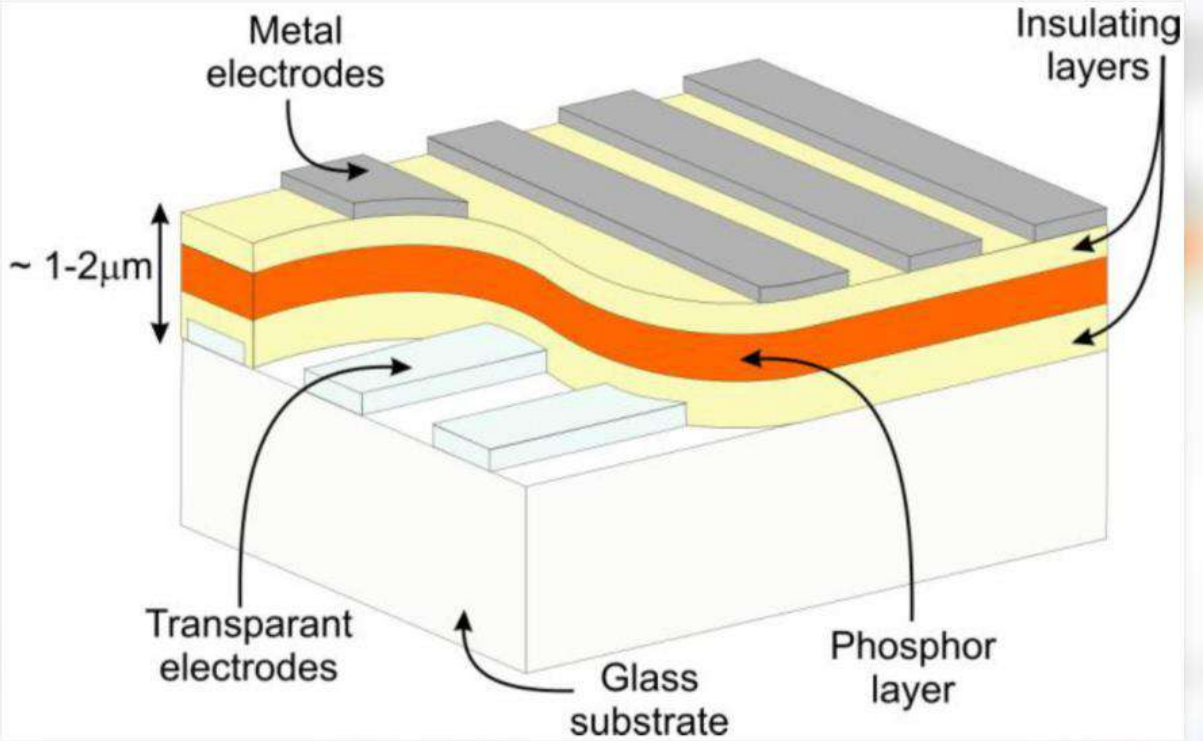
Thin-film electroluminescent displays are similar in construction to plasma panels. The main difference is the material placed between the two glass plates.

In a plasma panel, the space between the plates contains a gas mixture. In a thin-film electroluminescent display, this region contains a thin layer of phosphor material, such as zinc sulfide doped with manganese.

When an appropriate electric field is applied, the phosphor material emits light. By controlling the electrical signals applied to the display, images can be formed on the screen.

These displays are examples of emissive flat-panel displays because they produce their own light rather than depending entirely on an external light source.

1. Thin-Film Electroluminescent Displays



When a sufficiently high voltage is applied to a pair of crossing electrodes, the phosphor material emits light in the region where the electrodes intersect.

Electrical energy excites the manganese atoms present in the phosphor material. These atoms release the absorbed energy in the form of light, producing a glowing spot similar to the light produced in a plasma panel.

Electroluminescent displays require more power than plasma panels in some implementations. Producing high-quality color and grayscale images has also been a challenge for this display technology.

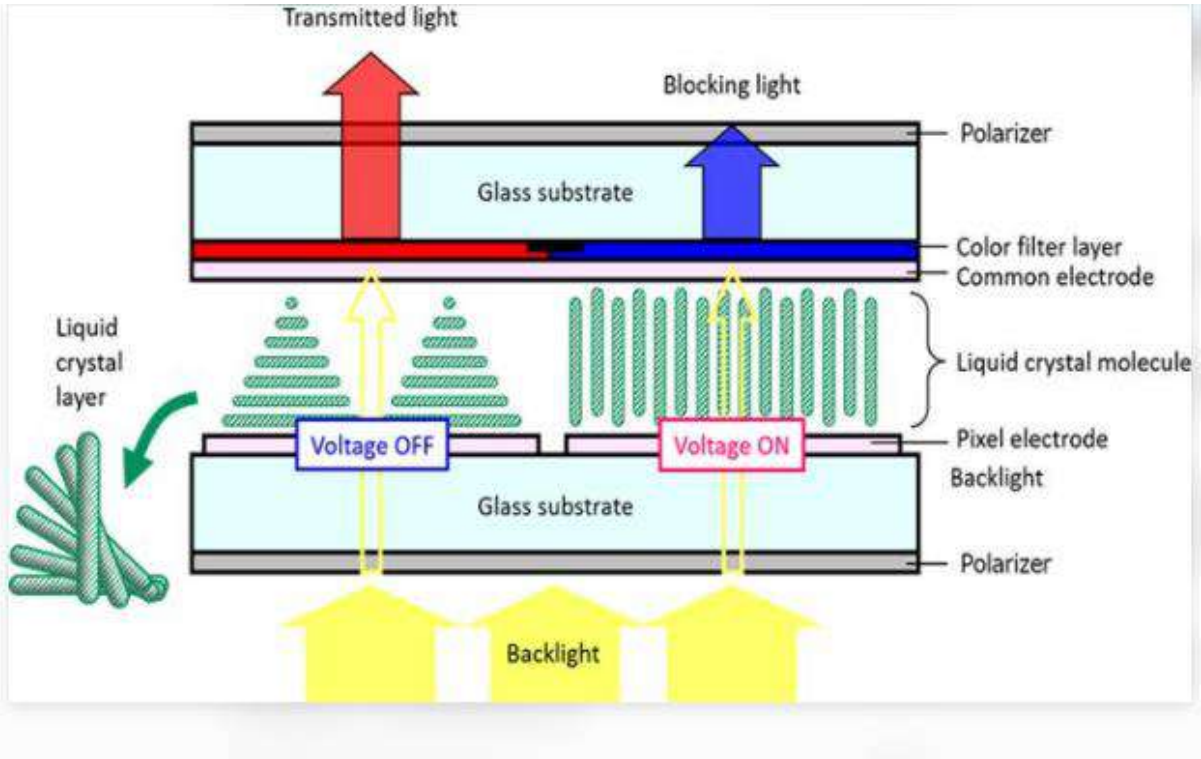
2. Light-Emitting Diode (LED) Displays

A Light-Emitting Diode (LED) display is another type of emissive flat-panel display. In this display, a matrix of light-emitting diodes is arranged to form the pixel positions on the screen.

The picture information is stored in a memory area called the refresh buffer. During display refreshing, the information is read from the refresh buffer and converted into suitable electrical signals. These signals are applied to the LEDs to produce the required light patterns.

By controlling the LEDs at different pixel positions, the display forms text, graphics, and images. LED displays are widely used in electronic signboards, digital displays, and large display screens.

3. Liquid-Crystal Displays (LCDs)



A Liquid-Crystal Display (LCD) is a non-emissive flat-panel display commonly used in calculators, laptops, computer monitors, and other portable electronic devices.

LCDs produce images by controlling the passage of polarized light through a liquid-crystal material. The light may come from an external source or from an internal backlight.

The liquid-crystal material changes its alignment when an electric field is applied. This controls whether light is transmitted through or blocked by the display. By controlling the light at individual pixel positions, LCDs form the required images.

LCDs are thin, lightweight, and generally consume less power than traditional CRT displays, making them suitable for portable devices.

Input Devices

An input device is a hardware device used to interact with a computer or provide data to it. Various input devices are available on graphics workstations. The most commonly used input devices are the keyboard and mouse. Other devices, such as trackballs, spaceballs, joysticks, digitizers, and button boxes, are specially designed for interactive input. Data gloves, touch panels, image scanners, and voice systems are also used in particular applications.

1. Keyboard

An alphanumeric keyboard on a graphics system is primarily used to enter text strings. It is also an efficient device for entering non-graphical data, such as picture labels associated with a graphics display.

General-purpose keyboards commonly include cursor keys and function keys. Function keys allow users to perform frequently used operations with a single keystroke. Cursor control keys are used to select displayed objects, select coordinate positions, and position the screen cursor.

Some keyboards also include other cursor-pointing devices, such as a trackball or joystick. A numeric keypad is often provided for the quick entry of numerical data.

In specialized graphics applications, input may be provided through buttons, dials, or switches that select data values or perform customized graphics operations. Buttons and switches are generally used to enter predefined functions, while dials are used to enter scalar values.

Real numbers within a defined range can be selected by rotating a dial. Potentiometers measure the amount of dial rotation and convert it into electrical voltages that can be used for cursor movement.

2. Mouse

A mouse is a small, handheld device used to position the screen cursor. Wheels or rollers located underneath the mouse detect the amount and direction of its movement.

Another method of detecting mouse movement uses an optical sensor. In such systems, the mouse is moved over a special mouse pad containing a grid of horizontal and vertical lines. The optical sensor detects movement across these lines and determines the direction and amount of movement.

A mouse can be picked up and placed at another position without changing the relative movement of the screen cursor. Therefore, it is useful for making relative changes to the cursor's position.

One, two, or three buttons are usually provided on the top of a mouse. These buttons are used to execute operations, record cursor positions, or invoke particular functions.

Additional controls can be incorporated into the basic mouse design to increase the number of input parameters. One example is the Z mouse, which includes three buttons, a thumb wheel on the side, a trackball on the top, and a standard mouse ball underneath.

Using a Z mouse, users can pick up an object, rotate it, and move it in any direction. They can also navigate through a three-dimensional (3D) scene by changing the viewing position and orientation.

The main applications of the Z mouse include virtual reality, Computer-Aided Design (CAD), and computer animation.

3. Trackball and Spaceball

A trackball is an input device consisting of a ball that can be rotated using the fingers or palm of the hand to move the screen cursor. Potentiometers attached to the ball measure the amount and direction of its rotation. Trackballs are often mounted on keyboards or other devices, such as the Z mouse.

A trackball is a two-dimensional (2D) positioning device, whereas a spaceball provides six degrees of freedom.

Unlike a trackball, a spaceball does not actually move. Strain gauges measure the amount of pressure applied to the spaceball when it is pushed or pulled in different directions. This information is used to determine spatial position and orientation.

Spaceballs are used for 3D positioning and selection in virtual reality systems, modeling, animation, Computer-Aided Design (CAD), and other applications.

4. Joystick

A joystick is an input device consisting of a small vertical lever, called a stick, mounted on a base. It is used to steer the screen cursor around the display.

A joystick consists of two potentiometers attached to a single lever. Moving the lever changes the settings of these potentiometers.

One potentiometer detects left or right movement, while the other detects forward or backward movement. These signals are used to control the cursor's position on the screen.

5. Data Glove

A data glove is an input device used to grasp and interact with virtual objects. It is constructed with a series of sensors that detect hand and finger movements.

Each sensor consists of a short length of fibre-optic cable, with a light-emitting diode (LED) at one end and a phototransistor at the other end. The surface of the cable is roughened in the area where it needs to be sensitive to bending.

When the cable bends, some of the light from the LED is lost. As a result, less light reaches the phototransistor. This change in light is used to detect the bending of fingers and the movement of the hand.

6. Digitizers

A digitizer is a common input device used for drawing, painting, and interactively selecting coordinate positions on an object. Digitizers can be used to input coordinate values in either two-dimensional (2D) or three-dimensional (3D) space.

Typically, a digitizer is used to scan a drawing or object and input a set of discrete coordinate positions. These positions can be joined using straight-line segments to approximate curves or surface shapes.

Graphics Tablet

One type of digitizer is the graphics tablet, also called a data tablet. It is used to input 2D coordinates by activating a hand cursor or stylus at selected positions on a flat surface.

A hand cursor contains crosshairs for accurately identifying positions, while a stylus is a pointed device used to indicate positions on the tablet.

An artist's digitizing system uses electromagnetic resonance to detect the 3D position of the stylus. This allows an artist to produce different brush strokes by applying different amounts of pressure to the tablet surface.

Tablet sizes vary from 12 × 12 inches for desktop models to 44 × 60 inches or larger for floor models. Graphics tablets provide highly accurate coordinate selection, with accuracy ranging from approximately 0.2 mm on desktop models to about 0.05 mm or less on larger models.

Working of Graphics Tablets

Many graphics tablets are constructed using a rectangular grid of wires embedded in the tablet surface. Electromagnetic pulses are generated sequentially along these wires. An electrical signal is induced in a wire coil inside an activated stylus or hand cursor to record its position on the tablet.

Depending on the technology used, signal strength, coded pulses, or phase shifts can determine the exact position of the stylus.

Acoustic or sonic tablets use sound waves to detect the position of a stylus. Strip microphones or point microphones detect the sound produced by an electrical spark at the tip of the stylus. The detected sound is used to determine the stylus position.

7. Image Scanners

An image scanner is an input device used to store drawings, graphs, colour and black-and-white photographs, and text in a computer. It works by passing an optical scanning mechanism over the information to be stored.

The scanner records the different shades of grey or colour and stores them in an array in the computer's memory.

Once the picture is stored in an internal digital representation, various transformations can be applied. For example, the picture can be rotated, scaled, or cropped to fit a particular screen area. Image-processing methods can also be used to modify the stored image.

When text is scanned, editing operations can be performed on the stored document. Some scanners can scan both graphical images and text. Scanners are available in different sizes and have different capabilities.

8. Touch Panels

Touch panels are input devices that allow users to select displayed objects or screen positions by touching the screen with a finger. A typical application of touch panels is the selection of processing options represented by graphical icons.

Some systems, such as plasma panels, are designed with touch screens. Other systems can be adapted for touch input by fitting a transparent touch-sensing device over the video monitor screen.

Touch input can be recorded using three methods: optical, electrical, and acoustical.

Optical Touch Panels

Optical touch panels use infrared light-emitting diodes (LEDs) arranged along one vertical edge and one horizontal edge of the frame. The opposite vertical and horizontal edges contain light detectors that record which light beams are interrupted when the panel is touched.

The two interrupted beams identify the horizontal and vertical coordinates of the selected screen position. With closely spaced LEDs, two horizontal or two vertical beams may be interrupted simultaneously. In such cases, the average position between the two interrupted beams is recorded.

The position can be detected with an accuracy of approximately a quarter of an inch, according to the provided notes. The LEDs operate at infrared frequencies, so their light is invisible to the user.

Electrical Touch Panels

An electrical touch panel consists of two transparent plates separated by a small distance. One plate is coated with a conducting material, while the other is coated with a resistive material.

When the outer plate is touched, it is pressed into contact with the inner plate. This contact creates a voltage drop across the resistive plate. The voltage is then converted into the coordinate values of the selected screen position.

Acoustical Touch Panels

Acoustical touch panels generate high-frequency sound waves in horizontal and vertical directions across a glass plate.

When a user touches the screen, part of each sound wave is reflected from the finger toward the emitters. The position of contact is calculated by measuring the time interval between the transmission of each wave and the reception of its reflection.

9. Light Pens

A light pen is a pencil-shaped input device used to select screen positions by detecting light emitted from points on a CRT screen.

A light pen is sensitive to the short burst of light emitted by the phosphor coating when the electron beam strikes a particular point on the screen. Other light sources, such as normal room lighting, are usually not detected.

When the light pen is pointed at a spot on the screen as the electron beam illuminates that spot, it generates an electrical pulse. This pulse causes the coordinate position of the electron beam to be recorded.

The recorded coordinates can be used to position an object or select a processing option.

Disadvantages of Light Pens

A light pen has several disadvantages. When it is pointed at the screen, the user's hand and the pen may obscure part of the screen image. Prolonged use can also cause arm fatigue.

Light pens cannot detect positions within black areas unless special arrangements are made. To select positions in any screen area, a nonzero intensity must be assigned to every screen pixel. Light pens may also produce false readings because of background lighting in a room.

10. Voice Systems

Voice systems use speech recognizers as input devices in graphics workstations. They accept voice commands to initiate graphics operations or enter data.

These systems work by matching spoken input against a predefined dictionary of words and phrases. The dictionary is created for a particular operator by asking the operator to speak the command words that will be used.

Each word is spoken several times. The system analyses the spoken word and establishes a frequency pattern for it in the dictionary, along with the corresponding function to be performed.

When a voice command is given later, the system searches the dictionary for a matching frequency pattern and performs the associated function.

Voice input is typically provided through a microphone mounted on a headset. The microphone is designed to minimize the input of background sounds. If a different operator uses the system, the dictionary must be re-established using that operator's voice patterns.

The main advantage of voice systems is that the operator does not need to shift attention from one input device to another to enter a command. This makes voice input useful for hands-free interaction with graphics applications.

Output Primitives

A picture can be described in several ways. In a raster display, a picture is completely specified by the set of intensity values assigned to the pixel positions on the display.

At the other extreme, a picture can be described as a collection of complex objects, such as trees, terrain, furniture, and walls, positioned at specified coordinate locations within a scene.

The shapes and colours of these objects can be represented internally using pixel arrays or basic geometric structures, such as straight-line segments and polygon colour areas.

The scene is displayed either by loading the pixel arrays into the frame buffer or by converting the basic geometric specifications into pixel patterns through a process called scan conversion.

Graphics programming packages provide functions to describe a scene using these basic geometric structures, which are called output primitives. They also provide facilities to group output primitives into more complex structures.

Each output primitive is specified using input coordinate data and other information that determines how the object should be displayed.

Points and straight-line segments are the simplest geometric components of pictures. Other output primitives used to construct pictures include circles, other conic sections, quadric surfaces, spline curves and surfaces, polygon colour areas, and character strings.

1. Point

A point is the simplest geometric output primitive in computer graphics. It represents a specific position in a coordinate system. A point is usually specified using its coordinates, such as \((x,y)\) in two-dimensional graphics or \((x,y,z)\) in three-dimensional graphics.

2. Line

A line segment is a basic output primitive used to connect two points in a graphics system. It is specified by the coordinates of its two endpoints. Line segments are used to construct shapes, boundaries, diagrams, and other graphical objects.

Line Drawing

A line is the most fundamental drawing primitive in Computer Graphics. Lines are used in charts, engineering drawings, illustrations, pencil-based animation, and curve approximation.

1. Desired Properties of Line Drawing Algorithms

A good line drawing algorithm should have the following properties:

1. Sharp and straight: Lines should appear sharp and straight on the screen.
2. Accurate termination: The line should terminate accurately at the specified endpoints.
3. Constant density: The plotted points should maintain uniform density along the line.
4. Independent of length and angle: Line density should not depend on the length or angle of the line.
5. Efficiency: The algorithm should draw lines quickly and use minimum computation.

2. Line Drawing Methods

Line drawing is accomplished by calculating intermediate positions along a line between its two endpoints. The output device is then directed to plot these positions.

A. Analog Devices

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 in proportion to the required changes in the X and Y directions. This produces smooth lines.

B. Digital Devices

Digital devices display a straight line segment by plotting discrete points between two endpoints.

The coordinate positions along the line are calculated using the equation of the line. Since digital screens use discrete pixel positions, the displayed line only approximates the actual line.

3. Line Drawing on a Raster Display

In a raster video display, the colour or intensity of the line is loaded into the frame buffer at the corresponding pixel coordinates.

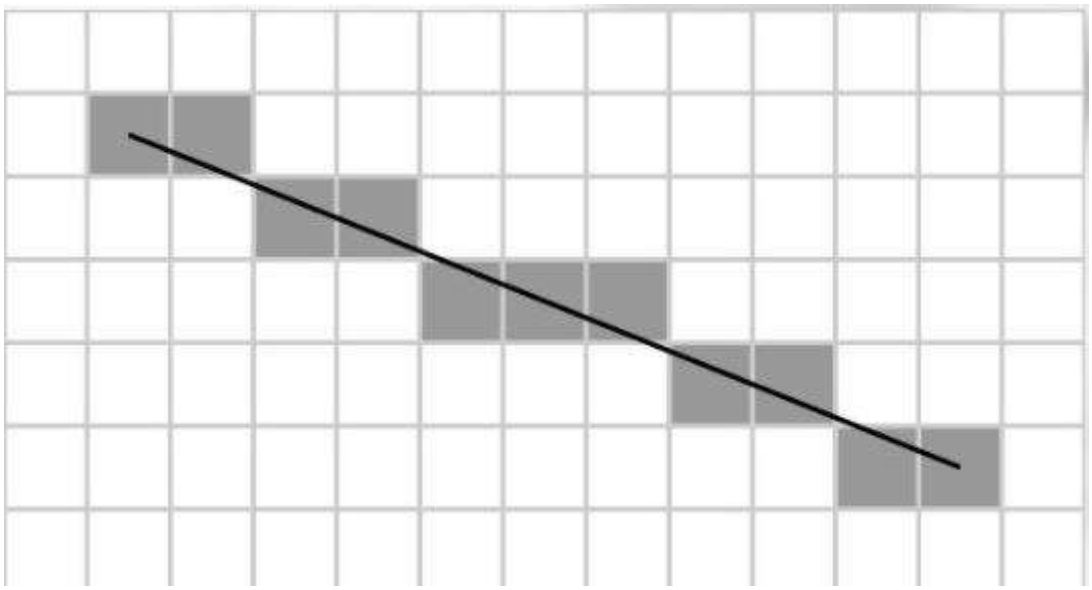
The video controller reads the frame buffer and plots the pixels on the screen.

Screen locations are represented by integer coordinates. Therefore, calculated positions may need to be rounded to the nearest integer pixel.

For example, a calculated line position of $(10.48, 20.51)$ may be converted to the pixel position $(10, 21)$.

4. Staircase Appearance (Jaggies)

Because a raster display uses discrete pixels, a diagonal line may appear as a series of small steps rather than a perfectly smooth line. This effect is called the staircase appearance or jaggies.



The staircase effect occurs because continuous line coordinates are converted into integer pixel coordinates.

Conclusion: Line drawing algorithms aim to produce lines that are accurate, smooth, uniform in density, and efficient to draw.

Raster Lines and Pixel Addressing

The characteristic staircase shape of raster lines is particularly noticeable on systems with low resolution. The appearance of these lines improves when they are displayed on high-resolution systems.

More effective techniques for smoothing raster lines are based on adjusting the pixel intensities along the line paths.

1. Pixel Positioning in Raster Graphics

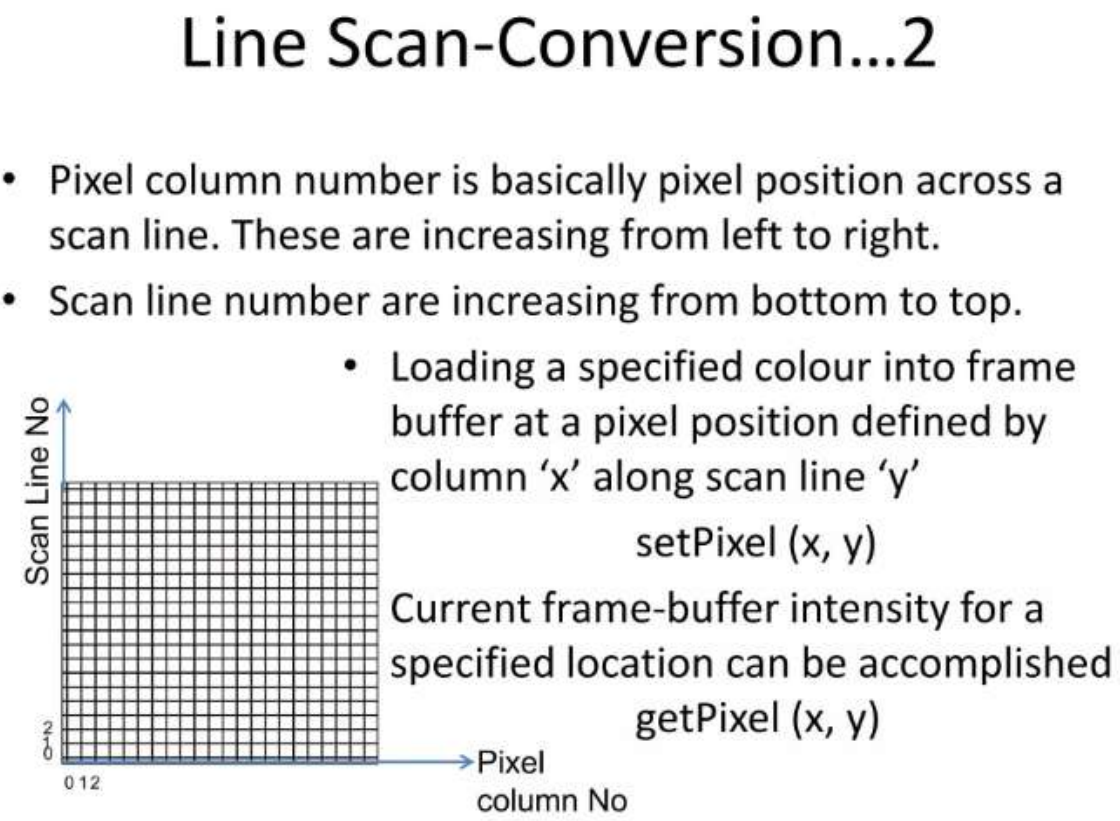
In raster-graphics device-level algorithms, object positions are specified directly using integer device coordinates.

Pixel positions are referenced using two values:

- Scan-line number (Y): Identifies the horizontal row of pixels.
- Column number (X): Identifies the pixel position across a scan line.

Scan lines are numbered consecutively from 0, starting at the bottom of the screen. Pixel columns are numbered from 0, moving from left to right across each scan line.

2. Pixel Addressing



A. `setPixel(x, y)`

The low-level procedure `setPixel(x, y)` is used to load a specified colour into the frame buffer at the pixel corresponding to column `x` along scan line `y`.

Example: `setPixel(5, 3)` sets the pixel at column 5 on scan line 3 to the specified colour.

B. `getPixel(x, y)`

The low-level function `getPixel(x, y)` is used to retrieve the current intensity setting stored in the frame buffer at a specified pixel location.

Example: `getPixel(5, 3)` returns the current intensity value of the pixel at column 5 on scan line 3.

Summary

Function	Purpose
`setPixel(x, y)`	Sets the colour of a specified pixel.
`getPixel(x, y)`	Retrieves the current intensity of a specified pixel.

These functions are basic operations used by raster graphics algorithms to draw and examine pixels on the screen.

DDA Line Drawing Algorithm

DDA (Digital Differential Analyzer) is a line-drawing algorithm in Computer Graphics used to draw a straight line between two given endpoints by calculating intermediate pixel positions.

1. Formulae

Step 1: Calculate differences

$$dx = x_2 - x_1$$

$$dy = y_2 - y_1$$

Step 2: Calculate number of steps

$$Steps = \max(|dx|, |dy|)$$

Step 3: Calculate increments

$$x_{inc} = \frac{dx}{Steps}$$

$$y_{inc} = \frac{dy}{Steps}$$

2. Algorithm Steps

2. Algorithm Steps

1. Start.
2. Read the two endpoints (x_1, y_1) and (x_2, y_2).
3. Calculate $dx = x_2 - x_1$ and $dy = y_2 - y_1$.
4. Calculate $Steps = \max(|dx|, |dy|)$.
5. Calculate $x_{inc} = dx/Steps$ and $y_{inc} = dy/Steps$.
6. Set $x = x_1$ and $y = y_1$.
7. Plot the pixel at $(round(x), round(y))$.
8. Add x_{inc} to x and y_{inc} to y.
9. Repeat until all steps are completed.
10. Stop.

3. Example

Draw a line from $(2, 3)$ to $(10, 8)$.

Step 1: Calculate differences

$$dx = 10 - 2 = 8$$

$$dy = 8 - 3 = 5$$

Step 2: Calculate steps

$$Steps = \max(8, 5) = 8$$

Step 3: Calculate increments

$$x_{inc} = \frac{8}{8} = 1$$

$$y_{inc} = \frac{5}{8} = 0.625$$

Step	X	Y	Plotted pixel
0	2.000	3.000	(2, 3)
1	3.000	3.625	(3, 4)
2	4.000	4.250	(4, 4)
3	5.000	4.875	(5, 5)
4	6.000	5.500	(6, 6)
5	7.000	6.125	(7, 6)
6	8.000	6.750	(8, 7)
7	9.000	7.375	(9, 7)
8	10.000	8.000	(10, 8)

Note: Pixel coordinates are rounded to the nearest integer. At exact half values, rounding conventions may differ between implementations.

4. Advantages

- Simple and easy to implement.

- Uses basic arithmetic operations.
- Can draw lines with different slopes.

5. Disadvantages

- Uses floating-point calculations.
- Rounding errors may occur.
- Generally slower than integer-based algorithms such as Bresenham's line algorithm.

Bresenham’s Line Drawing Algorithm

Definition: Bresenham’s Line Drawing Algorithm is a line-drawing algorithm in Computer Graphics used

Given two endpoints (x_1, y_1) and (x_2, y_2), assuming $0 \leq m \leq 1$:

Step 1: Calculate differences

$$dx = x_2 - x_1$$

$$dy = y_2 - y_1$$

Step 2: Calculate initial decision parameter

$$p_0 = 2dy - dx$$

Step 3: Select the next pixel

- If $p_k < 0$, choose the **E (East)** pixel.

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

- If $p_k \geq 0$, choose the **NE (North-East)** pixel.

$$p_{k+1} = p_k + 2dy - 2dx$$

to draw a straight line between two endpoints using only integer arithmetic.

2. Algorithm Steps

1. Start.
2. Read the endpoints (x_1, y_1) and (x_2, y_2).
3. Calculate $dx = x_2 - x_1$ and $dy = y_2 - y_1$.
4. Calculate $p_0 = 2dy - dx$.
5. Plot the starting point (x_1, y_1).
6. If $p_k < 0$, select E and update $p_k = p_k + 2dy$.
7. Otherwise, select NE and update $p_k = p_k + 2dy - 2dx$.
8. Repeat until the endpoint is reached.
9. Stop.

3. Example

Draw a line from $(2, 2)$ to $(9, 6)$.

Step 1: Calculate differences

$$dx = 9 - 2 = 7$$

$$dy = 6 - 2 = 4$$

Step 2: Calculate initial decision parameter

$$p_0 = 2dy - dx$$

$$p_0 = 2(4) - 7 = 1$$

Step	Current pixel	Decision parameter	Next pixel
0	(2, 2)	1	(3, 3)
1	(3, 3)	-5	(4, 3)
2	(4, 3)	3	(5, 4)
3	(5, 4)	-3	(6, 4)
4	(6, 4)	5	(7, 5)
5	(7, 5)	-1	(8, 5)
6	(8, 5)	7	(9, 6)

The plotted pixels are:

$$(2,2),(3,3),(4,3),(5,4),$$

$$(6,4),(7,5),(8,5),(9,6)$$

4. Advantages

1. Uses integer arithmetic.
2. Faster than DDA in many implementations.
3. Reduces floating-point calculations.
4. Suitable for raster displays.

Important for exams: Remember the initial decision parameter \(p_0=2dy-dx\) and the two update formulas. These are for lines with slope between 0 and 1; other slopes require corresponding adjustments.

1. Midpoint Circle Drawing Algorithm

Definition: The Midpoint Circle Drawing Algorithm is used to draw a circle by selecting the nearest pixels using a decision parameter.

Given centre (x_c, y_c) and radius r:

Step 1: Initialize

$$x = 0, \quad y = r$$

Step 2: Initial decision parameter

$$p_0 = 1 - r$$

Step 3: Update the decision parameter

- If $p_k < 0$, choose the East pixel:

$$p_{k+1} = p_k + 2x + 3$$

- If $p_k \geq 0$, choose the South-East pixel:

$$p_{k+1} = p_k + 2x - 2y + 5$$

For the second case, decrease y by 1. In both cases, increase x by 1.

Algorithm Steps

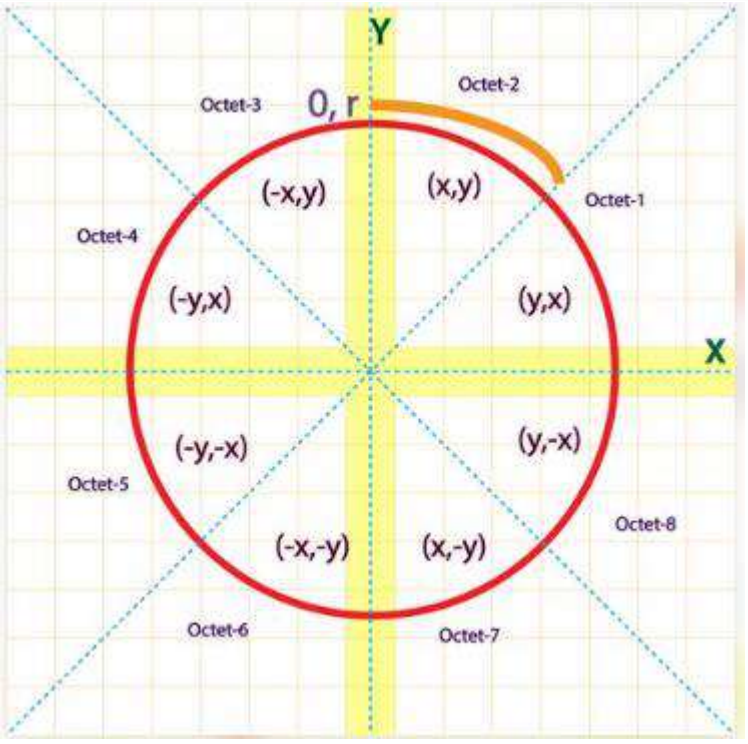
1. Start.
2. Input the radius r and circle centre (x_c, y_c).
3. Set $x = 0,\ y = r$.
4. Calculate $p = 1 - r$.
5. Plot the eight symmetrical points of the circle.
6. If $p < 0$, update $p = p + 2x + 3$.
7. Otherwise, update $p = p + 2x - 2y + 5$ and decrease y by 1.
8. Increase x by 1.
9. Repeat while $x \leq y$.
10. Stop.

Eight-Way Symmetry

For each calculated point (x, y), plot these eight points:

$$(x_c \pm x,\ y_c \pm y)$$

$$(x_c \pm y,\ y_c \pm x)$$



2. Midpoint Ellipse Drawing Algorithm

Definition: The Midpoint Ellipse Drawing Algorithm is used to draw an ellipse by selecting pixels using a decision parameter and dividing the first quadrant into two regions.

Formulae

Step 1: Initialize

$$x = 0, \quad y = r_y$$

$$r_x^2 = (r_x)^2, \quad r_y^2 = (r_y)^2$$

Step 2: Initial decision parameter for Region 1

$$p_1 = r_y^2 - r_x^2 r_y + \frac{1}{4} r_x^2$$

Region 1: Continue while

$$2r_y^2 x < 2r_x^2 y$$

- If $p_1 < 0$, increase x by 1:

$$p_1 = p_1 + 2r_y^2 x + r_y^2$$

- Otherwise, increase x by 1 and decrease y by 1:

$$p_1 = p_1 + 2r_y^2 x - 2r_x^2 y + r_y^2$$

Step 3: Initial decision parameter for Region 2

After reaching the end of Region 1:

$$p_2 = r_y^2(x + \tfrac{1}{2})^2 + r_x^2(y - 1)^2 - r_x^2 r_y^2$$

Region 2: Continue until $y < 0$.

- If $p_2 > 0$, decrease y by 1:

$$p_2 = p_2 - 2r_x^2 y + r_x^2$$

- Otherwise, increase x by 1 and decrease y by 1:

$$p_2 = p_2 + 2r_y^2 x - 2r_x^2 y + r_x^2$$

The decision parameter updates use the new x, y values after the corresponding coordinate changes.

The decision parameter updates use the new \(x,y\) values after the corresponding coordinate changes.

Algorithm Steps

1. Start.
2. Input the centre (x_c, y_c) and radii r_x, r_y.
3. Initialize $x = 0,\ y = r_y$.
4. Calculate the initial decision parameter for Region 1.
5. Plot four symmetrical points.
6. Calculate and update the decision parameter until Region 1 ends.
7. Calculate the initial decision parameter for Region 2.
8. Continue plotting and updating until $y < 0$.
9. Stop.

Four-Way Symmetry

For each calculated point (x, y), plot:

$$(x_c + x, y_c + y)$$

$$(x_c - x, y_c + y)$$

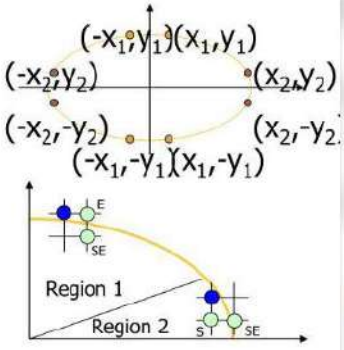
$$(x_c + x, y_c - y)$$

$$(x_c - x, y_c - y)$$

Midpoint Ellipse Algorithm

- Implicit equation is:

  $F(x,y) = b^2x^2 + a^2y^2 - a^2b^2 = 0$
- We have only 4-way symmetry
- There exists two regions
  - In **Region 1** dx > dy
    - Increase x at each step
    - y may decrease
  - In **Region 2** dx < dy
    - Decrease y at each step
    - x may increase



Difference Between Circle and Ellipse Algorithms

Midpoint Circle	Midpoint Ellipse
Uses one radius \(r\).	Uses two radii \(r_x,r_y\).
Uses eight-way symmetry.	Uses four-way symmetry.
Uses one decision parameter formula.	Uses two regions with separate decision parameters.
Draws a circle.	Draws an ellipse.

Exam preparation tip: Learn the circle's initial parameter \(p_0=1-r\), and the ellipse's two-region method. Practice one numerical example of each to understand how the pixels are selected.

Unit 2

Filled Area Primitives

Definition: Filled area primitives are graphical shapes filled with a particular colour.

- The most common filled shape is a polygon.
- Polygon boundaries are made up of straight lines.
- Standard graphics objects are often constructed using a set of polygon surface patches.

Two Basic Approaches to Area Filling

1. Scan-Line Approach

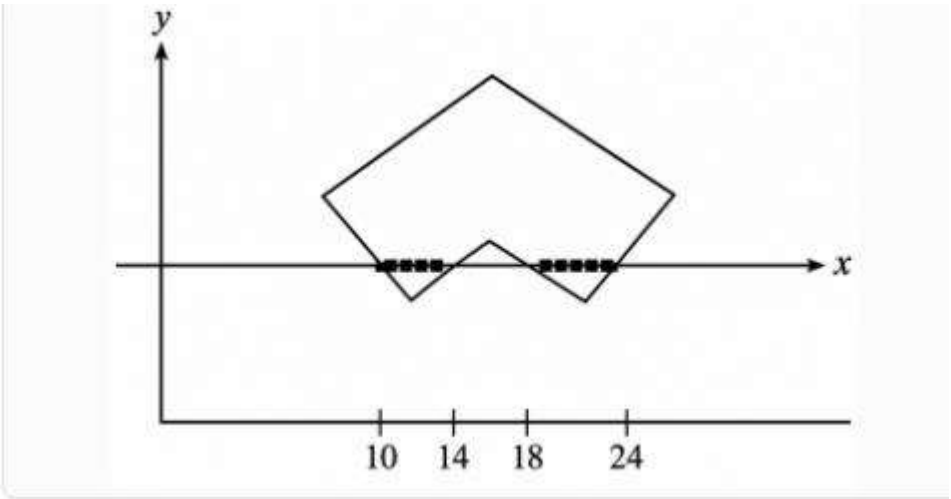
This method determines the intervals where scan lines intersect the area and fills the pixels between the boundary intersections.

It is commonly used in graphics packages to fill polygons, circles, ellipses, and other simple curves.

2. Interior-Point Approach

This method starts from a point inside the area and fills outward until the specified boundary conditions are encountered.

It is useful for areas with complex boundaries and in interactive painting systems.



Scan-Line Polygon Fill Algorithm

The scan-line polygon fill algorithm is used to fill the interior of a polygon with a specified colour on a raster display.

1. Working Principle

For each scan line crossing a polygon, the algorithm finds the intersection points between the scan line and the polygon edges.

1. Find the intersection points of the scan line with the polygon edges.
2. Sort the intersection points from left to right.
3. Identify pairs of intersection points.
4. Fill the pixels between each pair with the specified colour.
5. Repeat the process for all scan lines crossing the polygon.

Example: If the intersection points are $x=10,14,18,24$, fill the pixel spans from 10 to 14 and from 18 to 24.

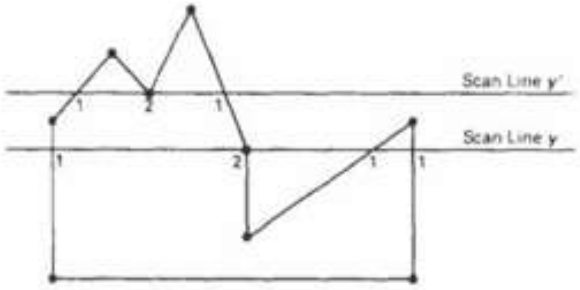
2. Handling Polygon Vertices

Special handling is required when a scan line passes through a polygon vertex because two edges meet at that point.

- Case 1: If the two connected edges lie on opposite sides of the scan line, the vertex should be counted as one intersection.
- Case 2: If both connected edges lie above the scan line or both lie below it, the vertex is a local minimum or maximum. It may need to be counted as two intersections.

These cases help prevent incorrect filling caused by an odd number of intersection points.

3. Edge Shortening Method



One method of handling vertices is to shorten selected polygon edges slightly.

- Process the non-horizontal edges in clockwise or counterclockwise order.
- Compare the Y-coordinates of consecutive edges.
- If their endpoint Y-coordinates increase or decrease monotonically, shorten the appropriate lower edge so that the shared vertex produces only one intersection for a scan line passing through it.

This avoids counting the same vertex incorrectly.

4. Advantages

- Efficient for filling polygon areas.
- Suitable for raster graphics systems.
- Can fill complex polygons by processing scan lines one at a time.

Conclusion: The scan-line polygon fill algorithm fills a polygon by finding edge intersections, sorting them, and colouring the pixels between intersection pairs. Special vertex handling ensures that the correct interior regions are filled.

Scan-Line Polygon Fill Algorithm – Continuation Notes

1. Handling Increasing Y-Coordinates

When the endpoint Y-coordinates of two edges are increasing, the Y-coordinate of the upper endpoint of the current edge is decreased by 1.

This prevents the common vertex from being counted twice when a scan line passes through it.

2. Handling Decreasing Y-Coordinates

When the endpoint Y-coordinates are monotonically decreasing, the Y-coordinate of the upper endpoint of the following edge is decreased by 1.

This also helps avoid duplicate intersections at shared vertices.

3. Coherence

Coherence means that the properties of one part of a scene are related to the properties of other parts of the same scene. This relationship can be used to reduce processing.

In Computer Graphics, coherence methods use calculations from one scan line to the next instead of repeating all calculations from the beginning.

4. Incremental Calculations for Edge Intersections

When determining edge intersections, incremental coordinate calculations can be performed along an edge because the slope of the edge remains constant from one scan line to the next.

Consider two successive scan lines,

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

Since the change in Y-coordinates between successive scan lines is:

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

The slope becomes:

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

Rearranging:

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

Therefore, the next X-coordinate can be calculated from the previous X-coordinate using the inverse slope.

5. Advantages of Coherence

- Reduces repeated calculations.
- Improves the efficiency of scan conversion.
- Helps calculate edge intersections quickly.
- Uses information from successive scan lines.

Conclusion: Coherence and incremental calculations improve the efficiency of the scan-line polygon fill algorithm by reusing information from one scan line to the next.

Scan-Line Polygon Fill Algorithm – Continuation

1. Calculation of X-Intersection

The change in Y-coordinates between two successive scan lines is 1.

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

The X-intersection value on the upper scan line can be calculated from the X-intersection value on the preceding scan line:

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

Each successive X-intersection is calculated by adding the inverse of the slope and rounding to the nearest integer.

2. Parallel Implementation

In the parallel implementation of the fill algorithm, each scan line crossing the polygon is assigned to a separate processor.

The edge-intersection calculations are then performed independently for each scan line.

3. Integer Calculation of X-Intersections

In a sequential fill algorithm, the increment of X-values by \(1/m\) along an edge can be performed using integer operations.

Let:

- Δx = difference between the edge endpoint X-coordinates.
- Δy = difference between the edge endpoint Y-coordinates.

$$\frac{1}{m} = \frac{\Delta x}{\Delta y}$$

The X-intersections can be calculated incrementally using integer operations.

Procedure:

1. Initialize a counter to 0.
2. Increment the counter by \(\Delta x\) whenever moving to the next scan line.
3. When the counter becomes equal to or greater than \(\Delta y\), increment the current X-intersection by 1.
4. Subtract \(\Delta y\) from the counter.
5. Repeat for successive scan lines.

This method reduces floating-point calculations.

4. Sorted Edge Table (SET)

To perform polygon filling efficiently, the polygon boundaries are stored in a Sorted Edge Table (SET).

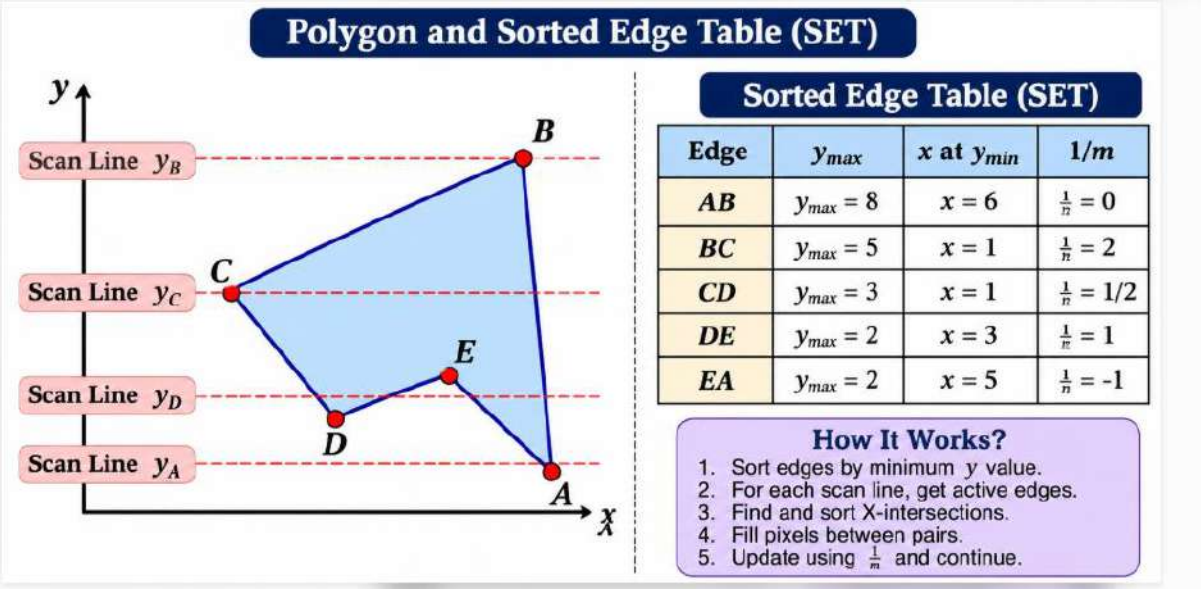
The table contains the information required to process scan lines efficiently.

Important points:

- Polygon edges are processed in clockwise or counterclockwise order.
- Edges are sorted according to the smallest Y-coordinate of each edge.
- Only non-horizontal edges are included in the table.
- Each edge entry contains:

1. Maximum Y-coordinate of the edge.
2. X-intersection value at the lower vertex.
3. Inverse slope of the edge, \(1/m\).

- For each scan line, the active edges are arranged from left to right.



Scan-Line Polygon Fill Algorithm

Algorithm:

1. Start.
2. Input the polygon vertices.
3. Create a Sorted Edge Table (SET) containing all non-horizontal edges, sorted by minimum Y-coordinate.
4. Initialize the scan line at the minimum Y-coordinate of the polygon.
5. Add the edges that begin at the current scan line to the Active Edge Table (AET).
6. Remove edges whose maximum Y-coordinate has been reached.
7. Calculate the X-intersection points of the active edges.
8. Sort the intersection points in ascending order of X-coordinate.
9. Fill the pixels between each pair of intersection points with the specified colour.
10. Move to the next scan line and update the X-intersections using the inverse slope \(1/m\).
11. Repeat Steps 5–10 until all scan lines have been processed.
12. Stop.

1. Boundary-Fill Algorithm

Definition: The Boundary-Fill Algorithm is a computer graphics algorithm used to fill the interior of a closed shape with a specified colour until the boundary colour is reached.

Algorithm Steps

1. Start.
2. Select a point inside the closed region.
3. Check the colour of the current pixel.
4. If the pixel colour is neither the boundary colour nor the fill colour, colour the pixel with the fill colour.
5. Apply the same procedure to the neighbouring pixels.
6. Continue until the boundary is reached.
7. Stop.

Types

- 4-connected boundary fill: Checks left, right, top and bottom neighbouring pixels.
- 8-connected boundary fill: Checks all eight neighbouring pixels, including diagonal neighbours.

Example

If a closed shape has a black boundary and a white interior, the algorithm fills the white interior with red, stopping when it reaches the black boundary.

2. Flood-Fill Algorithm

Definition: The Flood-Fill Algorithm is used to replace all connected pixels of the same initial colour with a new fill colour.

Algorithm Steps

1. Start.
2. Select a starting pixel inside the region.
3. Identify the original colour of the pixel.
4. If the current pixel has the original colour, replace it with the new fill colour.
5. Apply the same procedure to neighbouring pixels.
6. Continue until all connected pixels of the original colour have been replaced.
7. Stop.

Types

- 4-connected flood fill: Checks left, right, top and bottom neighbouring pixels.
- 8-connected flood fill: Checks all eight neighbouring pixels, including diagonal neighbours.

Example

If a region contains connected white pixels, the algorithm replaces those white pixels with red while leaving pixels of other colours unchanged.

Difference Between Boundary-Fill and Flood-Fill

Boundary-Fill	Flood-Fill
Stops when it reaches the specified boundary colour.	Replaces pixels matching the original interior colour.
Requires a boundary colour and a fill colour.	Requires an original colour and a new fill colour.
Suitable for closed shapes with a clearly defined boundary.	Suitable for replacing connected regions of the same colour.
Stops when the boundary colour is encountered.	Stops when neighbouring pixels no longer match the original colour.

Exam tip: Boundary-fill checks the boundary colour, whereas flood-fill checks the original pixel colour.

1. Boundary-Fill Algorithm

Definition

In the Boundary-Fill Algorithm, area filling starts at a point inside a region and paints the interior towards the boundary.

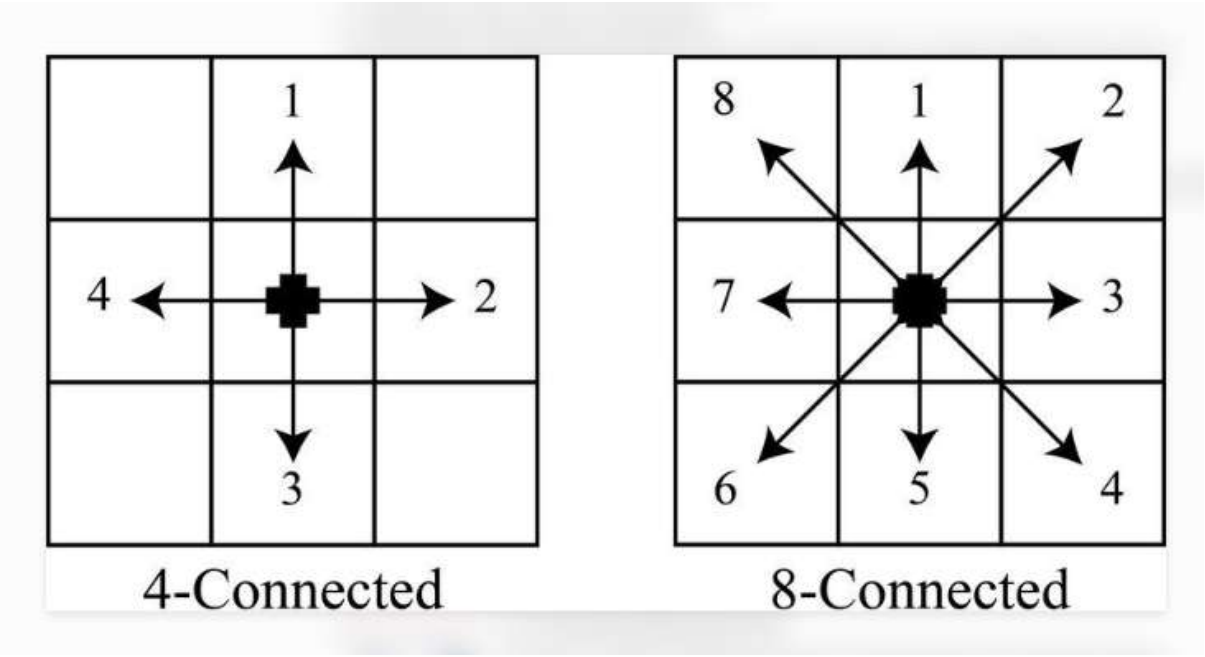
If the boundary is specified in a single colour, the algorithm fills the region pixel by pixel until the boundary colour is encountered.

Working Principle

1. Select an interior point \((x,y)\).
2. Specify the fill colour and boundary colour.
3. Check the colour of the current pixel.
4. If the pixel is neither the boundary colour nor the fill colour, paint it with the fill colour.
5. Check the neighbouring pixels and repeat the process.
6. Stop when the boundary is reached and all interior pixels have been tested.

This algorithm is useful in interactive painting packages. An artist can draw a figure, select a colour and click inside the figure to fill its interior.

Types of Boundary-Fill Algorithm

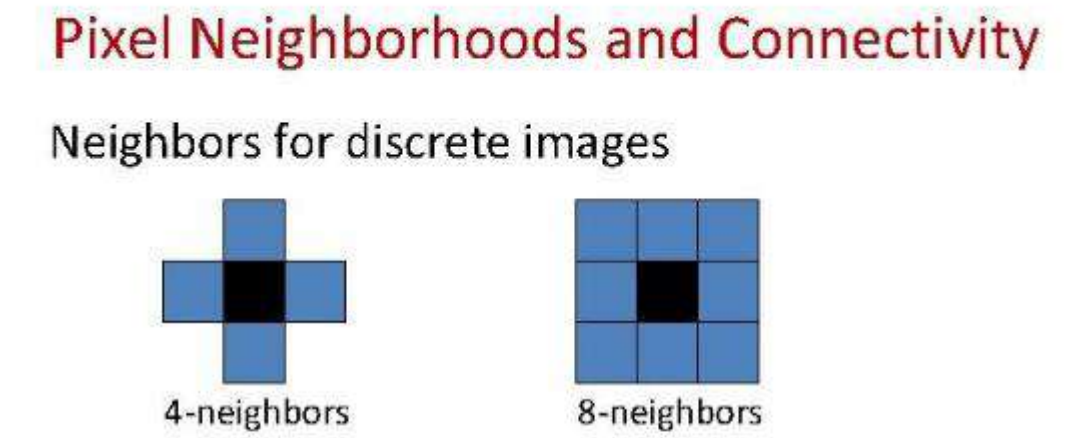


1. 4-Connected Boundary Fill

Checks four neighbouring pixels: right, left, above and below.

$$(x+1, y),\ (x-1, y)$$

$$(x, y+1),\ (x, y-1)$$



2. 8-Connected Boundary Fill

Checks all eight neighbouring pixels, including the four diagonal pixels.

It is useful for filling more complex figures where diagonal connections matter.

Procedure for 4-Connected Boundary Fill

```
boundaryFill4(x, y, fill, boundary)
{
   current = getPixel(x, y);

   if (current != boundary AND current != fill)
   {
      setPixel(x, y, fill);

      boundaryFill4(x+1, y, fill, boundary);
      boundaryFill4(x-1, y, fill, boundary);
      boundaryFill4(x, y+1, fill, boundary);
      boundaryFill4(x, y-1, fill, boundary);
   }
}
```

Explanation: The procedure checks the current pixel. If its colour is neither the boundary colour nor the fill colour, it fills that pixel and recursively checks its four neighbours.

2. Flood-Fill Algorithm

Definition

The Flood-Fill Algorithm is used to fill or recolour an area that is not defined by a single boundary colour.

Instead of searching for a boundary colour, it replaces a specified original interior colour with the desired fill colour.

Working Principle

1. Select an interior point \((x,y)\).
2. Specify the old colour and the new fill colour.
3. Check whether the current pixel has the old colour.
4. If it matches, replace it with the fill colour.
5. Check the neighbouring pixels and repeat the process.
6. Stop when no connected neighbouring pixels have the old colour.

If the area contains multiple interior colours, the interior pixels can first be changed to a common colour before applying the algorithm.

Procedure for 4-Connected Flood Fill

```
floodFill4(x, y, fillColor, oldColor)
{
```

```
   if (getPixel(x, y) == oldColor)
   {
      setPixel(x, y, fillColor);

      floodFill4(x+1, y, fillColor, oldColor);
      floodFill4(x-1, y, fillColor, oldColor);
      floodFill4(x, y+1, fillColor, oldColor);
      floodFill4(x, y-1, fillColor, oldColor);
   }
}
```

Explanation: If the current pixel matches the old colour, it is replaced with the fill colour. The procedure then checks the four neighbouring pixels recursively.

Difference Between Boundary-Fill and Flood-Fill

Boundary-Fill	Flood-Fill
Uses a boundary colour.	Uses an old or original colour.
Stops when the boundary colour is encountered.	Stops when connected pixels no longer match the old colour.
Fills the interior of a closed boundary.	Recolours a connected region of the same colour.
Requires fill colour and boundary colour.	Requires fill colour and old colour.

Remember for exams: Boundary-fill checks the boundary colour, whereas flood-fill checks the original interior colour. Both can use 4-connected or 8-connected neighbouring pixels.

Two-Dimensional Transformations – Translation

Your notes explain the first type of 2D transformation: Translation. Here is a corrected and simplified explanation for easy understanding.

1. Introduction

Transformation is a fundamental concept in computer graphics.

Transformation means changing an object into another position, size, shape, or orientation by applying certain rules.

Transformations are used to:

- Position objects.
- Change the size of objects.
- Rotate objects.
- Change the shape of objects.
- Change the viewing position.

There are five types of 2D transformations:

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

2. Translation

Definition: Translation is the process of moving an object from one coordinate position to another along a straight-line path without changing its shape or size.

In translation, we move an object by adding translation distances txt_x and tyt_y to its original coordinates (x,y)(x,y).

The new coordinates are:

$$x' = x + t_x$$

$$y' = y + t_y$$

Where:

- x, y = Original coordinates.
- x', y' = New coordinates.
- t_x = Translation distance along the X-axis.
- t_y = Translation distance along the Y-axis.

Translation vector: The pair of translation distances (tx,ty)(t_x,t_y) is called a translation vector or shift vector.

Example

Suppose an object has a point at (2, 3).

Translation distances are:

- $t_x = 4$
- $t_y = 2$

New coordinates:

$$x' = 2 + 4 = 6$$

$$y' = 3 + 2 = 5$$

Therefore, the new point is (6, 5).

Important points

1. Translation is a rigid-body transformation.
2. It moves an object without changing its shape or size.
3. A straight-line segment is translated by applying the translation equations to both endpoints.
4. A polygon is translated by adding the translation vector to the coordinates of each vertex.
5. The object is redrawn using the new coordinates.

What happens if translation distance is greater than width?

If txt_x or tyt_y is greater than the object's width or height, it does not automatically produce an error.

The object simply moves by that distance. However, it may move outside the visible screen or viewing window.

Exam tip Remember: Translation changes only the position of an object, not its shape or size.

I can also explain the remaining four transformations — Scaling, Rotation, Reflection and Shearing — in the same simple, exam-friendly format.

3. Scaling
Definition: Scaling is the process of increasing or decreasing the size of an object.

- Scaling is used to enlarge or reduce the size of an object.
- It changes the dimensions (width and height) of an object.
- Scaling is done by multiplying the original coordinates by a scaling factor.

Scaling Formula
Suppose the original coordinates are (x,y)(*x*,*y*), and the scaling factors are (Sx,Sy)(*Sx*,*Sy*).

The new coordinates are:

$$x' = x \times S_x$$

$$y' = y \times S_y$$

Where:

- x, y = Original coordinates.
- x', y' = New coordinates.
- S_x = Scaling factor along X-axis.
- S_y = Scaling factor along Y-axis.

Where:

- x, y = Original coordinates.
- x', y' = New coordinates.
- S_x = Scaling factor along X-axis.
- S_y = Scaling factor along Y-axis.

Types of Scaling

1. **Scaling factor less than 1:** Object becomes smaller.
2. **Scaling factor greater than 1:** Object becomes bigger.
3. **Scaling factor equal to 1:** Object remains the same size.

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:

$$\mathbf{P' = M_1 \cdot P + M_2} \quad \text{---------- (1)}$$

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

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

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

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

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

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^1 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

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

Scaling

Concatenating transformation matrices for two successive scaling operations producesthe 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.

4. Rotation (Pivot-Point Rotation)

Definition: Rotation is the process of turning an object around a fixed point called the pivot point by a given angle.

Normally, rotation is performed around the origin (0,0). But sometimes, we need to rotate an object around another point. This point is called the pivot point.

Steps for Pivot-Point Rotation

To rotate an object around a selected pivot point (xr,yr)(x_r,y_r), follow these three steps:

1. Translate: Move the object so that the pivot point comes to the origin (0,0).
2. Rotate: Rotate the object around the origin by the required angle θ\theta.
3. Translate back: Move the object back to its original position.

Rotation Formula

For rotation around the origin:

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

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

Pivot-Point Rotation Matrix

The combined transformation matrix is:

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

It can also be written as:

$$T(x_r, y_r) \times R(\theta) \times T(-x_r, -y_r)$$

Where:

- x_r, y_r = Pivot-point coordinates.
- θ = Angle of rotation.
- T = Translation matrix.
- R = Rotation matrix.

5. Fixed-Point Scaling

Definition: Fixed-point scaling is the process of increasing or decreasing the size of an object while keeping one selected point fixed.

This fixed point is called the reference point or fixed point (xf,yf)(x_f,y_f).

Steps for Fixed-Point Scaling

1. Step 1: Keep the object at its original position.
2. Step 2: Translate the object so that the fixed point comes to the origin (0,0).
3. Step 3: Apply scaling to increase or decrease the size of the object.
4. Step 4: Translate the object back to its original position.

In simple words:

Move to origin → Apply scaling → Move back.

The object changes its size, but the fixed point remains in the same position.

6. General Scaling Directions

Definition: General scaling direction means scaling an object in a direction other than the X-axis and Y-axis.

Normally, scaling is done along the X-axis and Y-axis. But sometimes, we need to scale an object in a different direction.

Steps for General Scaling Directions

1. Step 1: Rotate the object to align the required scaling direction with the X and Y axes.
2. Step 2: Apply the scaling transformation.
3. Step 3: Rotate the object back to its original direction.

Formula

The combined transformation matrix is:

$$R^{-1}(\theta) \times S(S_x, S_y) \times R(\theta)$$

Where:

- $R(\theta)$ = Rotation matrix.
- $R^{-1}(\theta)$ = Reverse rotation matrix.
- S_x = Scaling factor along X-axis.
- S_y = Scaling factor along Y-axis.
- θ = Angle of rotation.

Important Points for Exams

Fixed-Point Scaling:

- Changes the size of an object.
- Keeps one selected point fixed.
- Uses translation, scaling and reverse translation.

General Scaling Directions:

- Scaling is performed in a direction other than the coordinate axes.
- First rotate the object.
- Apply scaling.
- Rotate it back.

Easy trick to remember

- Fixed-point scaling: Translate → Scale → Translate back.
- General scaling: Rotate → Scale → Reverse Rotate.

7. General Scaling Example

In this example, a unit square is converted into a parallelogram by stretching it along a diagonal direction.

- The diagonal is from (0,0) to (1,1).
- The object is rotated by 45°.
- Scaling factors are $S_1 = 1$ and $S_2 = 2$.
- The length is doubled in the required direction.

8. Reflection

Definition: Reflection is a transformation that produces a mirror image of an object.
It is similar to looking at an object in a mirror. The object is flipped around a fixed line called the **reflection axis**.

Types of Reflection

1. Reflection about X-axis

In this reflection, the object is flipped vertically around the X-axis.

The X-coordinate remains the same, but the Y-coordinate changes its sign.
Formula:

$$x' = x$$

$$y' = -y$$

Transformation Matrix:

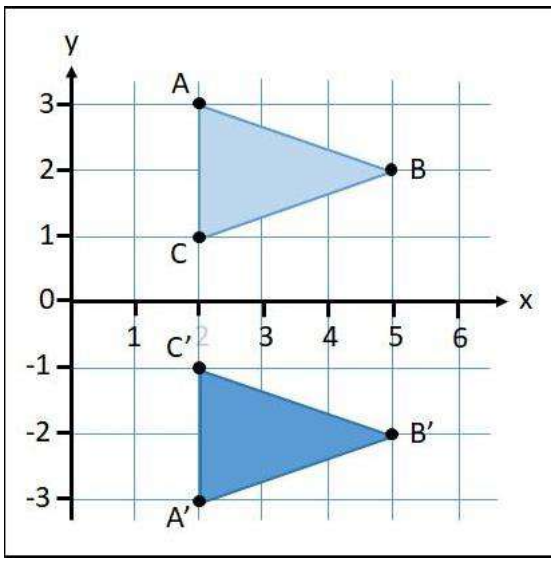
$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Example

Original point = (2, 3)

After reflection:
$x' = 2, y' = -3$ $x' = 2, y' = -3$

New point = **(2, -3)**.



Computer Graphics: Reflection and Shear

1. Reflection

Definition

Reflection is a 2D transformation that produces a mirror image of an object with respect to a reference axis or line.

The reflected object has the same shape and size as the original object, but its position and orientation change.

Types of Reflection and Transformation Matrices

We use homogeneous coordinates to represent a 2D transformation matrix as a 3x3 matrix.

(i) Reflection about the X-axis

The X-coordinate remains unchanged, and the Y-coordinate changes its sign.

x'=x, y'=-y

Transformation matrix:

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

Example: (2, 3) becomes (2, −3).

(ii) Reflection about the Y-axis

The Y-coordinate remains unchanged, and the X-coordinate changes its sign.

$$x' = -x, \qquad y' = y$$

Transformation matrix:

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

Example: (2, 3) becomes (−2, 3).

(iii) Reflection about the origin

Both the X-coordinate and Y-coordinate change their signs.

$$x' = -x, \qquad y' = -y$$

Transformation matrix:

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

Example: (2, 3) becomes (−2, −3).

(iv) Reflection about the diagonal line $y = x$

The X-coordinate and Y-coordinate are interchanged.

$$x' = y, \qquad y' = x$$

Transformation matrix:

$$R_{y=x} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

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 defineswhere 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 **viewingtransformation.**
- Sometimes we consider this viewing transformation as window-to-viewport transformation but in general itinvolves more steps.

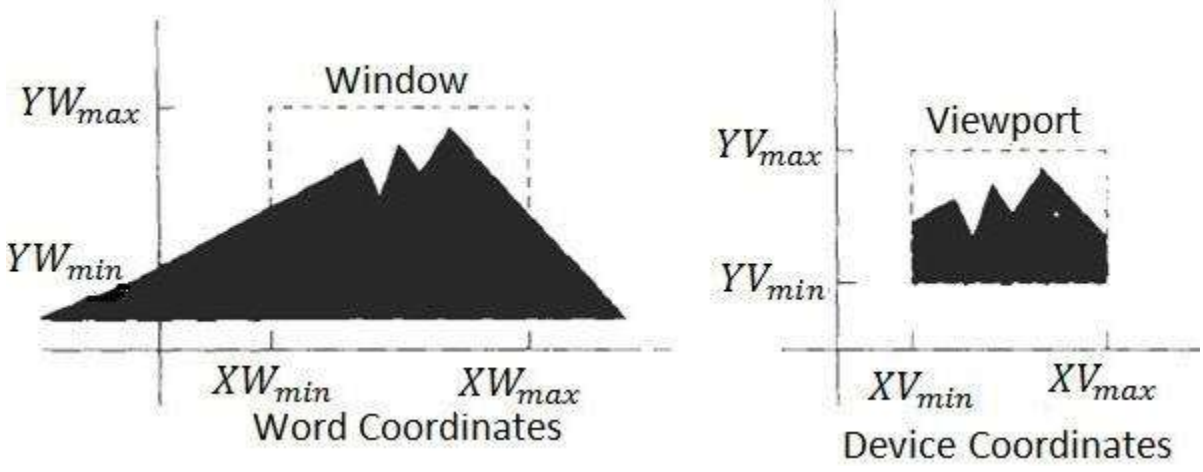


Fig. 3.1: - A viewing transformation using standard rectangles for the window and viewport.

- Now we see steps involved in viewing pipeline.

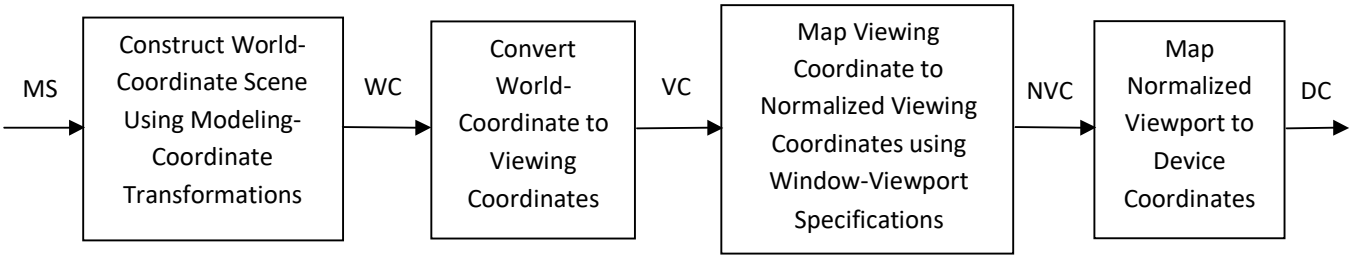


Fig. 3.2: - 2D viewing pipeline.

- As shown in figure above first of all we construct world coordinate scene using modeling coordinatetransformation.
- 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 whichprovide 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

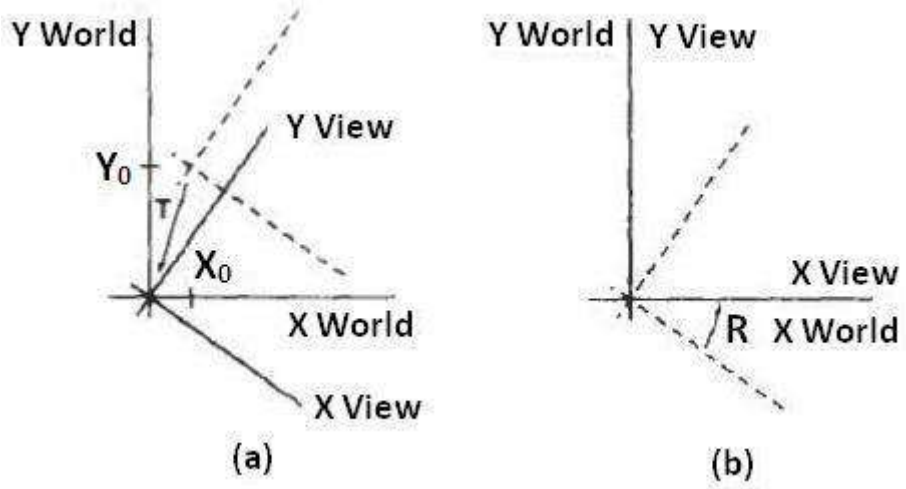


Fig. 3.3: - A viewing-coordinate frame is moved into coincidence with the world frame in two steps: (a) translate the viewing origin to the world origin, and then (b) rotate to align the axes of the two systems.

- 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.**

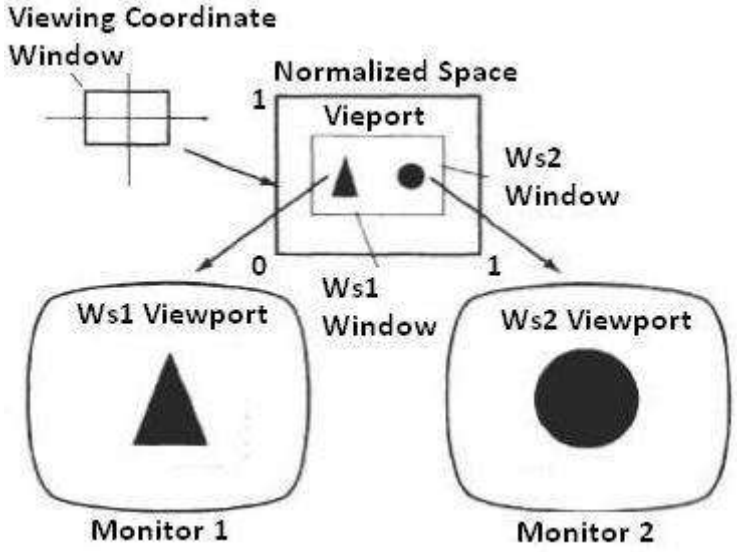


Fig. 3.4: - workstation transformation.

- 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

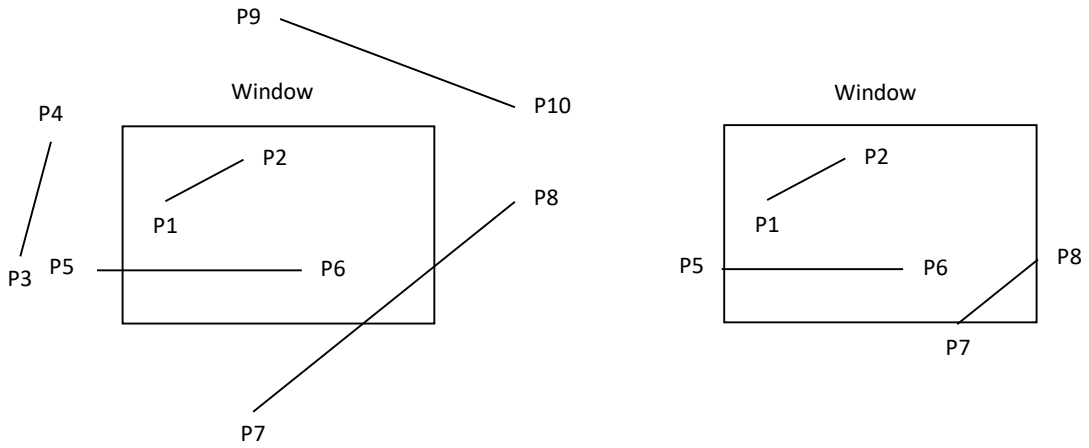
- In point clipping we eliminate those points which are outside the clipping window and draw points which are inside the clipping window.
- Here we consider clipping window is rectangular boundary with edge (x_{wmin},x_{wmax},y_{wmin},y_{wmax}).
- So for finding wether given point is inside or outside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

$$x_{wmin} \leq x \leq x_{wamx}$$
$$y_{wmin} \leq y \leq y_{wamx}$$

- 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.



Before Clipping
(a)

After Clipping
(b)

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 bttom boundary clipper and a top boundary clipper, as shown in figure below.

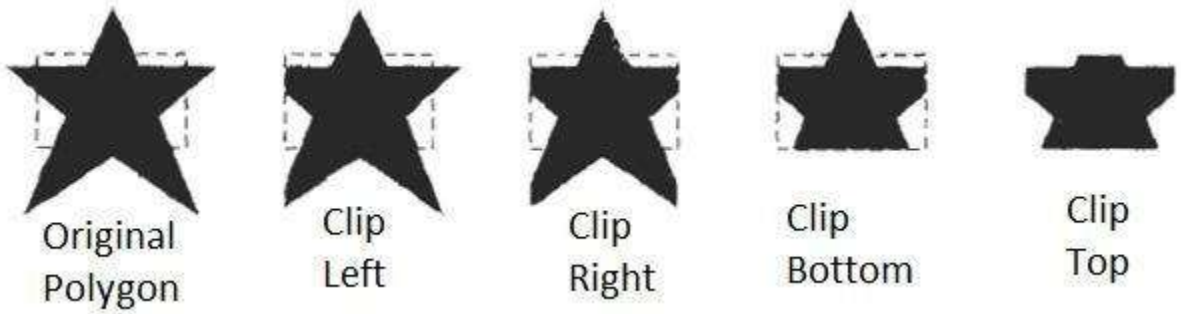


Fig. 3.11: - Clipping a polygon against successive window boundaries.

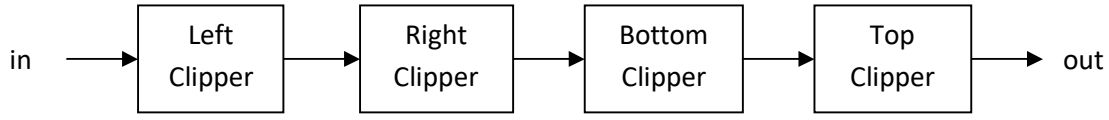


Fig. 3.12: - Processing the vertices of the polygon through boundary clipper.

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

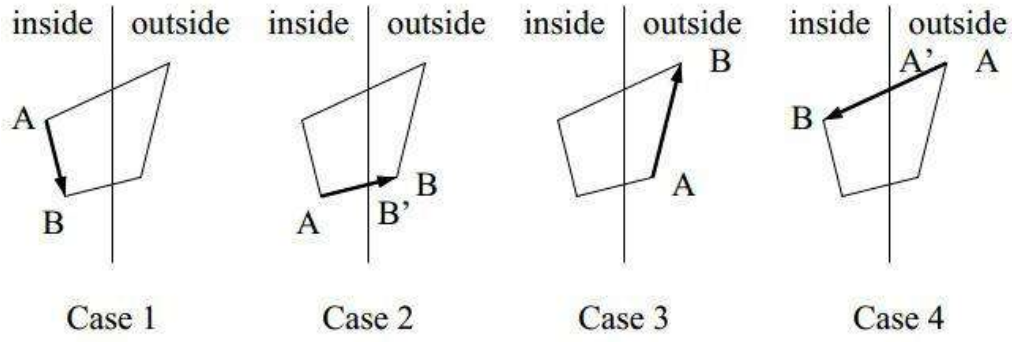


Fig. 3.13: - Clipping a polygon against successive window boundaries.

- 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.

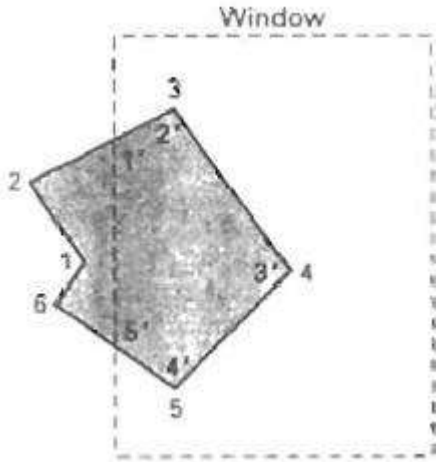
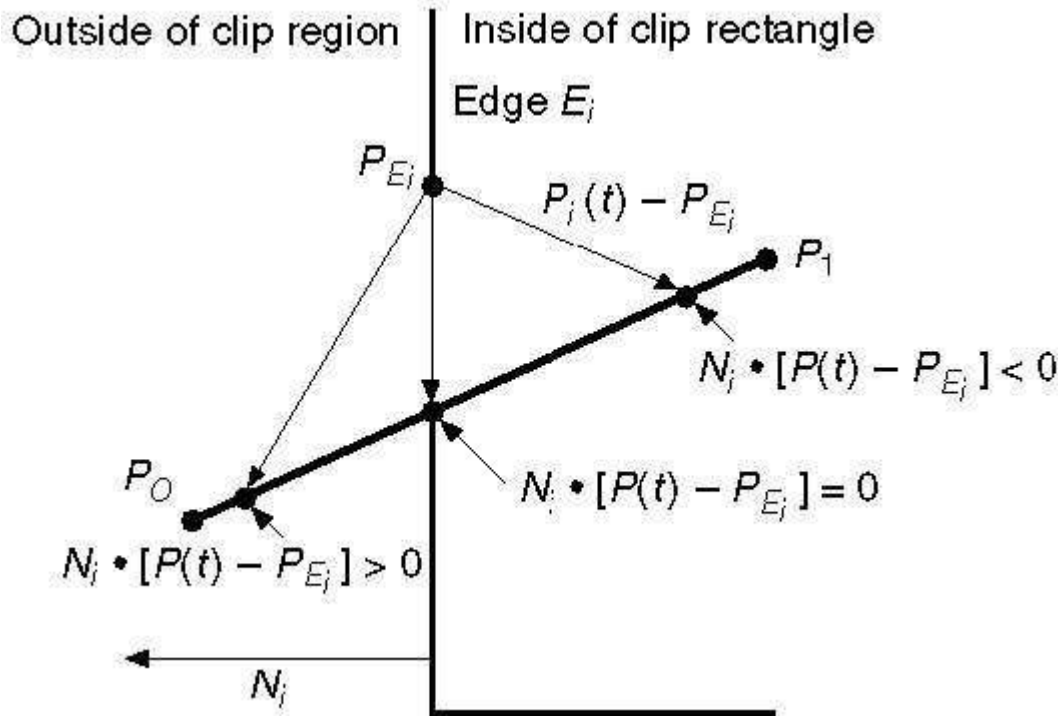


Fig. 3.14: - Clipping a polygon against left window boundaries.

- 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 –

```
P0P1:P(t) = P0 + t(P1 - P0)
```

Let N_i be the outward normal edge E_i. Now pick any arbitrary point P_{Ei} on edge E_i then the dot product $N_i.[Pt - P_{Ei}]$ 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 $N_i.[Pt - P_{Ei}] < 0$

The point P*t* is outside if $N_i.[Pt - P_{Ei}] > 0$

The point P*t* is on the edge if $N_i.[Pt - P_{Ei}] = 0$ *Intersectionpoint*

$N_i.[Pt - P_{Ei}] = 0$

$N_i.[P_0 + t(P_1 - P_0) - P_{Ei}] = 0$ *ReplacingP*(*t* with $P_0 + t(P_1 - P_0)$)

$N_i.[P_0 - P_{Ei}] + N_i.t[P_1 - P_0] = 0$

$N_i.[P_0 - P_{Ei}] + N_i \cdot tD = 0$ (substituting D for $[P_1 - P_0]$)

$N_i.[P_0 - P_{Ei}] = - N_i \cdot tD$

The equation for t becomes,

$$t = \frac{N_i . [P_0 - P_{Ei}]}{N_i . D}$$

It is valid for the following conditions –

- $N_i \neq 0$ *errorcannothappen* D!=0

unit 4

3D Object Representations – Introduction

3D Object Representation is a method of representing three-dimensional objects in computer graphics. These objects have three dimensions: length, width, and height. 3D representations are used to create realistic objects in computer games, movies, engineering, and architecture.

Example: Cube, sphere, cylinder, and cone.

Polygon Surfaces

A polygon mesh is a collection of connected polygonal planar surfaces used to represent 3D objects in computer graphics.

- Vertex: A point where two or more edges meet.
- Edge: A line connecting two vertices.
- Polygon: A closed shape formed by connected edges.
- Polygon Mesh: A collection of vertices, edges, and polygons connected to form a 3D object.

Example: Boxes, cabinets, and building exteriors can be represented using polygon meshes.

Polygon mesh representations can be evaluated based on two criteria:

1. Space: The amount of memory required.
2. Time: The time required to create, store, and process the representation.

Polygon Mesh Representation

A polygon mesh representation is a method of representing a 3D object using a collection of vertices, edges, and polygons.

There are three basic elements:

1. Vertex: A point in 3D space.
2. Edge: A line connecting two vertices.
3. Polygon: A closed surface formed by connecting edges.

Example: A cube can be represented using 8 vertices, 12 edges, and 6 polygonal faces.

Polygon mesh representations are commonly used in computer graphics, 3D modelling, animation, and video games.



1. Vertex Table

Stores the coordinates of each vertex in 3D space.

Vertex	Coordinates
V1	x1, y1, z1
V2	x2, y2, z2
V3	x3, y3, z3
V4	x4, y4, z4
V5	x5, y5, z5

2. Edge Table

Stores the two vertices connected by each edge.

Edge	Connected vertices
E1	V1, V2
E2	V2, V3
E3	V3, V4
E4	V4, V1
E5	V4, V5
E6	V5, V1

3. Polygon-Surface Table

Stores the edges that form each polygon surface.

- Surface S1: E1, E2, E3, E4
- Surface S2: E3, E5, E6, E4

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 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 describe 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 points.

successive polygon vertices, 4), h, 4) and (x3, h, 4) and solve the following set Of equations for the ratio A/D, B/D and C/D

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

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 < 1 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.

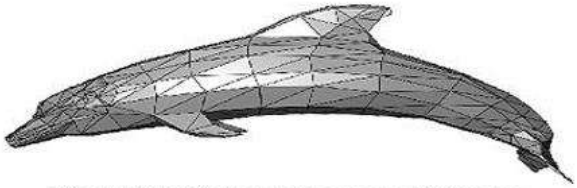


Fig: A Dolphin formed by mesh of triangles

The study of meshes is a large sub-field of computer graphics and geometric modeling. Different representations of 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 this representation, 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: These store 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: These store vertices in a predefined table, such that traversing the table implicitly defines polygons. This is essentially the triangle fan in hardware graphics rendering. The representation is more compact and more efficient for retrieving 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 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 that define the line segments.

Curve Lines and Surfaces

Polylines and polygons are first-degree, piecewise-linear approximations to curves and surfaces. Unless the curves or surfaces being approximated are also piecewise linear, a large number of endpoint coordinates must be created and stored to achieve accuracy.

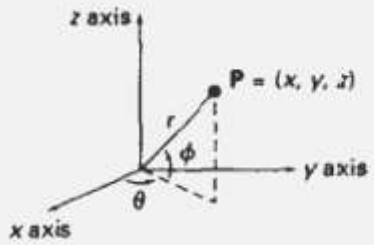
Interactive manipulation of the data to approximate a shape is tedious because many points have to be positioned precisely.

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 graphics packages as preconstructed objects.

Quadric Surfaces



Parametric coordinate position (r, θ, ϕ) on the surface of a sphere with radius r.

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.

$$x = r\cos\phi\cos\theta, \quad -\pi/2 \le \phi \le \pi/2$$
$$y = r\cos\phi\sin\theta, \quad -\pi \le \theta \le \pi$$
$$z = r\sin\phi$$

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

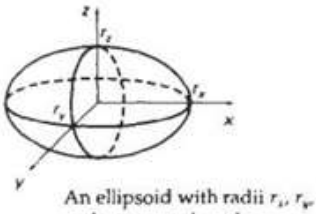
Ellipsoid

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:-

$$x = r_x \cos\phi \cos\theta, \quad -\pi/2 \le \phi \le \pi/2$$
$$y = r_y \cos\phi \sin\theta, \quad -\pi \le \theta \le \pi$$
$$z = r_z \sin\phi$$



An ellipsoid with radii r_x, r_y, and r_z centered on the coordinate origin.

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 360^0.

Parametric representation for a torus are similar to those for an ellipse except that angle extends over 3600.

$$x = r_x(r + \cos\phi)\cos\theta, \quad -\pi \le \phi \le \pi$$
$$y = r_y(r + \cos\phi)\sin\theta, \quad -\pi \le \theta \le \pi$$
$$z = r_z \sin\phi$$

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:

$$\left|\frac{x}{a}\right|^n + \left|\frac{y}{b}\right|^n = 1$$

Where:

- x, y = Coordinates of points on the superellipse.
- a, b= Semi-major and semi-minor axis parameters.
- n= Shape parameter that controls the shape of the curve.

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

Corresponding parametric equation for the superellipse can be expressed as;

$$x = r_x \cos^s \theta, \qquad -\pi \le \theta \le \pi$$
$$y = r_y \sin^s \theta$$

Superellipsoid

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

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

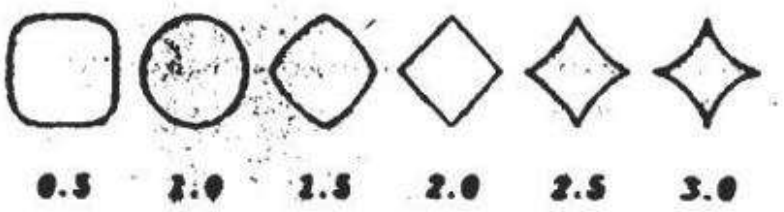
For $s_1=s_2=1$

We can write the corresponding parametric representation for the superellipsoid.

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

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

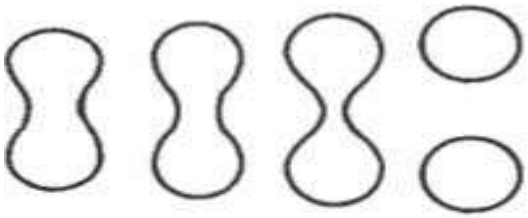
$$z = r_z \sin^{s_1}\phi$$



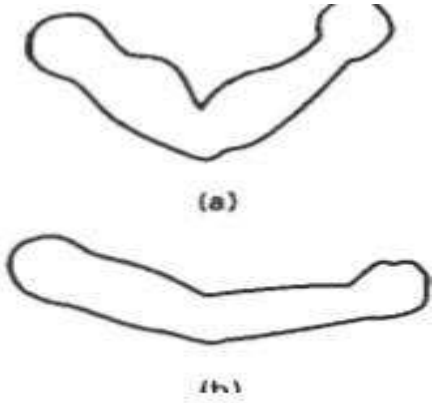
Superellipsoid plotted with different values for parameters and with $r_x = r_y$

Blobby Objects

Some objects donot 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.

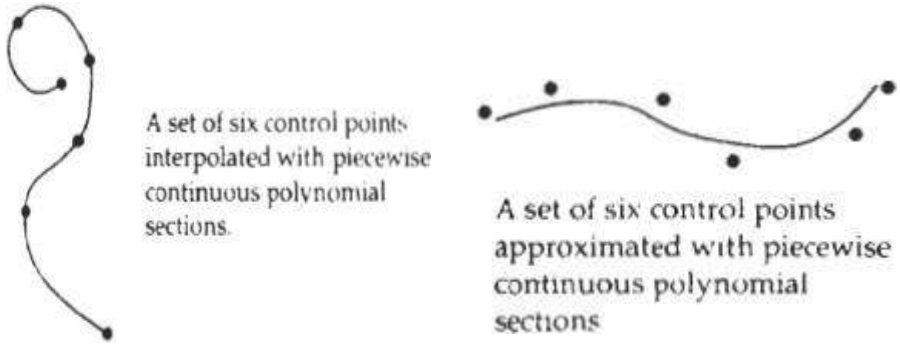
In computer graphics, the term spline curve now refers to any composite curve formed with polynomial sections satisfying specified continuity conditions at the boundary of the pieces. **A** spline surface can be described with two sets of orthogonal spline curves. There are several different kinds of 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 pararnetric polynomial functions in one of two ways:-

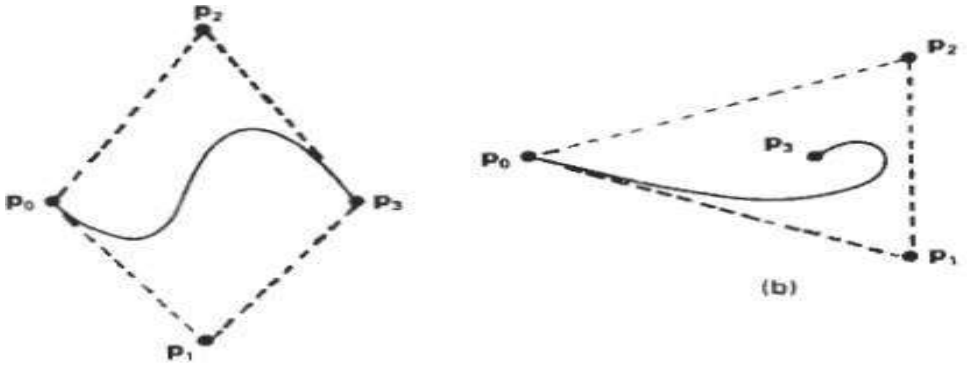
1. **Interpolate**
   When polynomial sections are fitted so that the curve passes through each control point
2. **Approximate**
   When the polynomial are fitted to the general control-point path without necessarily passing through any control point.

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.



Convex-hull shapes (dashed lines) for two sets of control points.

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.**

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

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

Zero order parametric continuity: - described as C^0continuity, 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^2continuity, 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^0continuity, 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:

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

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

$$x(u) = a_x u^3 + b_x u^2 + c_x u + d_x \qquad 0 \le u \le 1$$

Boundary conditions for this curve might be set, for example, on the endpoint coordinate x(0) and x(1) and on the parametric first derivaties 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.

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

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

$$C = M_{spline}.M_{geom}$$

Where M_{geom} is a four-element column matrix containing the geometric constraint values (boundary conditions) on the spline; and $\mathbf{M_{spline}}$ 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 $\mathbf{M_{geom}}$ 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

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

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.

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

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.

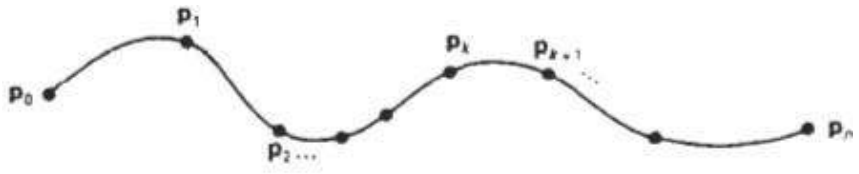
This class of splines is most often used to set up paths for object motions or to provide a representation for an existing object or drawing, but interpolation splines are also used sometimes to design object shapes. Cubic polynomials offer a reasonable compromise between flexibility and speed of computation. 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 + 1control points specified with coordinate

$P_k=(x_k,y_k,z_k)$, k=0, 1,2,…..... ,n

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

$$x(u) = a_x u^3 + b_x u^2 + c_x u + d_x$$
$$y(u) = a_y u^3 + b_y u^2 + c_y u + d_y \qquad 0 \le u \le 1$$
$$z(u) = a_z u^3 + b_z u^2 + c_z u + d_z$$

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 splints can be adjusted locally because each curve section is only dependent on its endpoint constraints.

If **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 $P(u) = au^3 + bu^2 + cu + d \qquad 0 \le u \le 1$

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

$$\begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{bmatrix}^{-1} \cdot \begin{bmatrix} p_k \\ p_{k+1} \\ Dp_k \\ Dp_{k+1} \end{bmatrix}$$

$$= \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} p_k \\ p_{k+1} \\ Dp_k \\ Dp_{k+1} \end{bmatrix}$$

$$= \mathbf{M}_H \cdot \begin{bmatrix} p_k \\ p_{k+1} \\ Dp_k \\ Dp_{k+1} \end{bmatrix}$$

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

$$P(u) = [u^3\ u^2\ u\ 1] \cdot \mathbf{M}_H \cdot \begin{bmatrix} p_k \\ p_{k+1} \\ Dp_k \\ Dp_{k+1} \end{bmatrix}$$

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.

$P(u)=p_kH_0(u)+ p_{k+1}H_1(u)+ Dp_kH_2(u)+ Dp_{k+1}H_3(u)$

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.

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

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: $C(n,k) = \dfrac{n!}{k!(n-k)!}$

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

$$BEZ_{k,n}(u) = (1-u)\,BEZ_{k,n-1}(u) + uBEZ_{k-1,n-1}(u), \qquad n > k \ge 1$$

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

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

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

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.

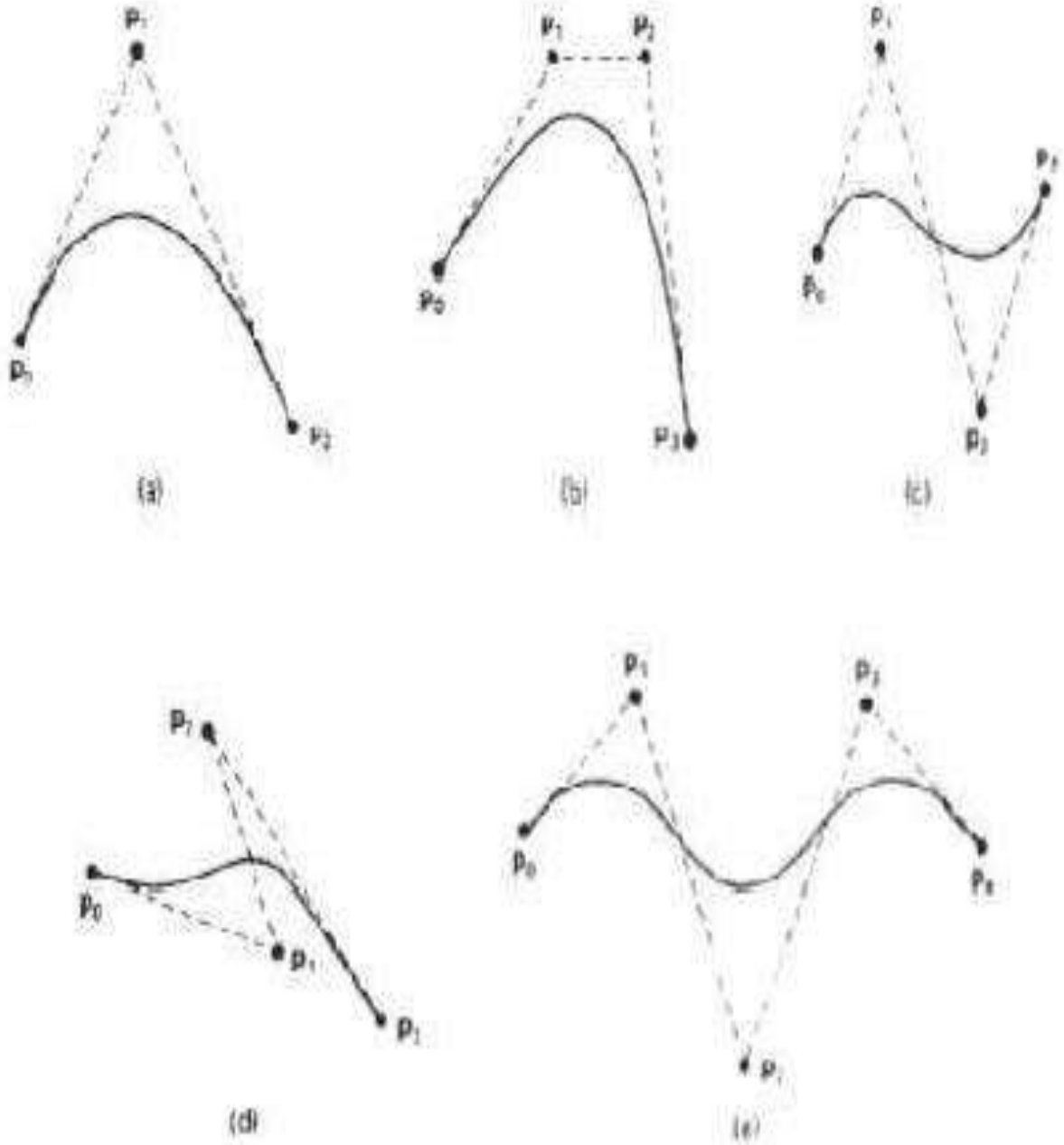


Figure 10-34
Examples of two-dimensional Bézier curves generated from three, four, and five control points. Dashed lines connect the control-point positions.

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 Barnstein 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

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

$$2 \le d \le n+1$$

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.

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

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

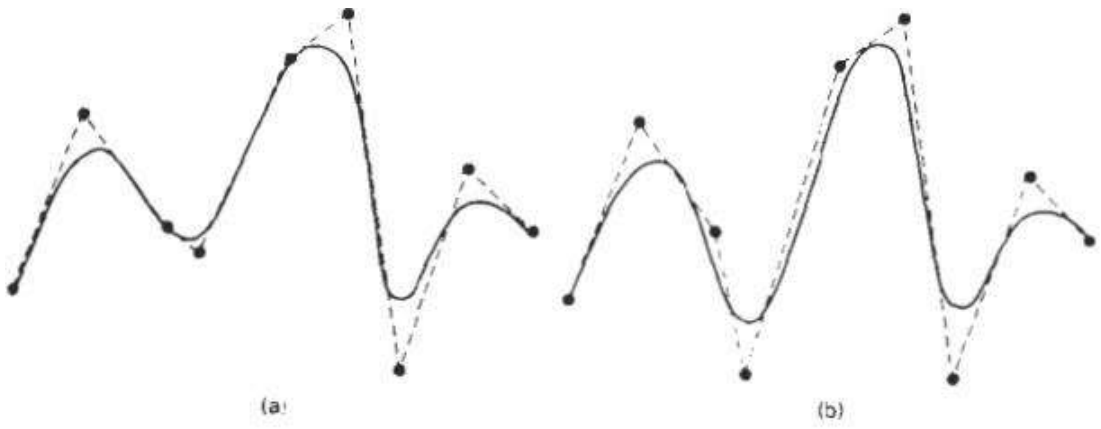
Where each blending function is defined over d subintervals of the total range of ***u.*** The selected set of subinterval endpoints u_j, is referred to as a **knot** vector. We can choose any values for the subinterval endpoints satisfying the relation $u_j \le u_{j+1}$. Values for u_{min} and u_{max} then depend on the number of control points we select, the value we choose for parameter d, and how we set u p 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 cublic 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+1control 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 $\{u_0,u_1\ldots\ldots,u_{n-d}\}$ the resulting B-spline curve is defined only in the interval from knot value u_{d-1} upto 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:

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

With $p_{j,k}$, specifying the location of the (m 1) by (n I) control points.

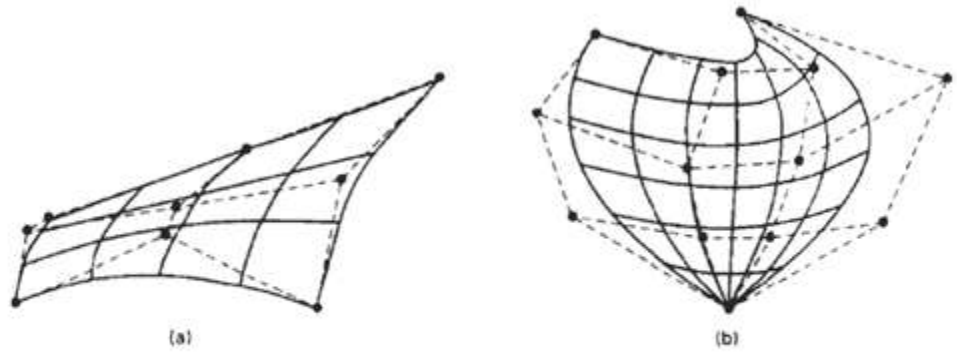


Figure 10-39
Bézier surfaces constructed for (a) $m = 3$, $n = 3$, and (b) $m = 4$, $n = 4$. Dashed lines connect the control points.

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.

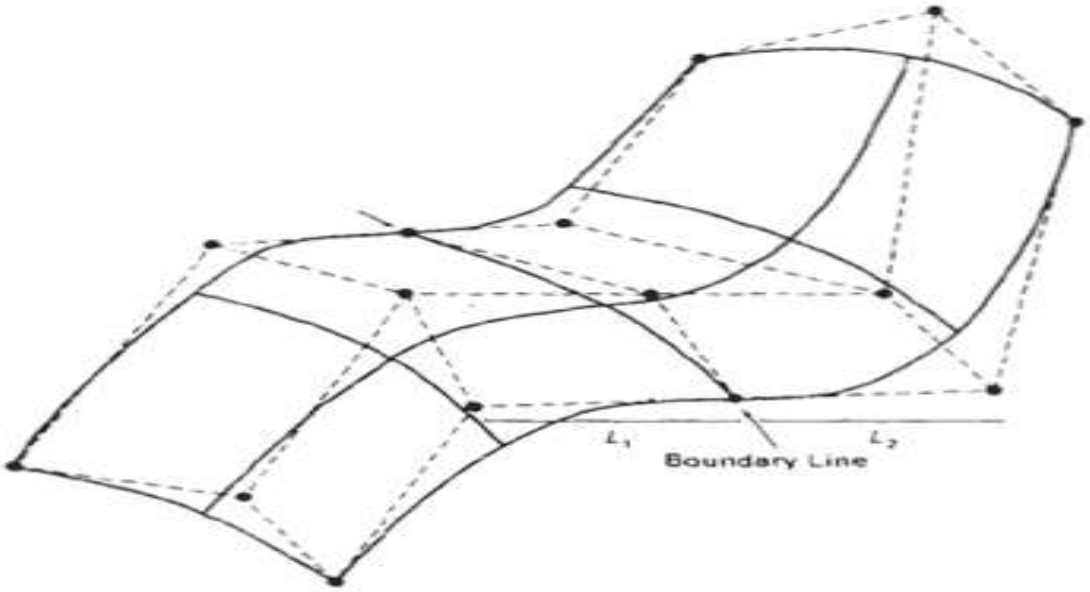


Figure 10-40
A composite Bézier surface constructed with two Bézier sections, joined at the indicated boundary line. The dashed lines connect specified control points. First-order continuity is established by making the ratio of length L_1 to length L_2 constant for each collinear line of control points 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 &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

$$\mathbf{P}(u) = \sum_{k=0}^{n} \mathbf{p}_k B_{k,d}(u), \qquad u_{min} \le u \le u_{max}, \qquad 2 \le d \le n+1$$

where the pk 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 Bspline parameters. And the B spline blending functions Bbd 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:

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

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

where each blending function is defined over d subintervals of the total range of u. The selected set of subinterval endpoints u, is referred to as a knot vector. We can choose any values for the subinterval endpoints satisfying the relation u1 ≤ uj 1,.Values for umax, and umin, 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.

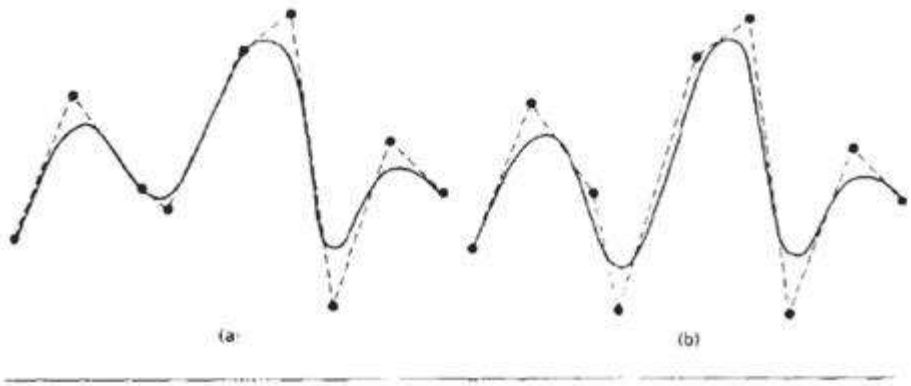


Figure 10-41
Local modification of a B-spline curve. Changing one of the control points in (a) produces curve (b), which is modified only in the neighborhood of the altered control point.

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 hanging 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 uk.
4. The range of parameter u 1s divided into n d subintervals by the n d 1 values specified in the knot vector.
5. With knot values labeled as [u1 u1 . . . . . $u_{n\ 1}$,] the resulting B-spline curve is defined only in the interval from knot value u_{d-1} , up to knot value $u_{n-1.}$
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 $u_{n\ 1}$the sum over all basis functions is 1:

Given the control-point positions and the value of parameter d, 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 objectsare illuminated.

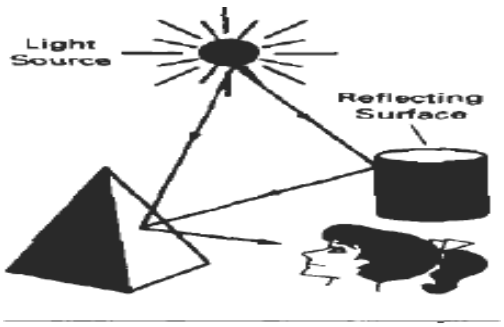


Figure 14-1
Light viewed from an opaque nonluminous surface is in general a combination of reflected light from a light source and reflections of light reflections from other surfaces

Diffuse

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

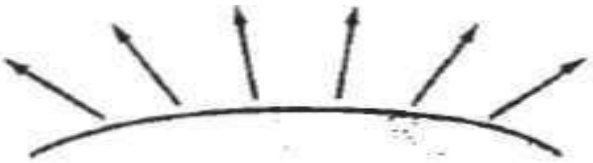
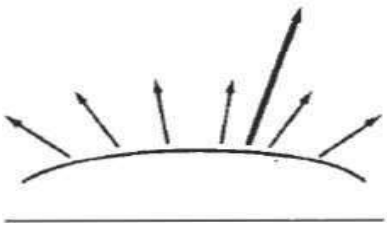


Figure 14-4
Diffuse reflections from a surface.

Specular

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





Specular Reflection and Diffuse

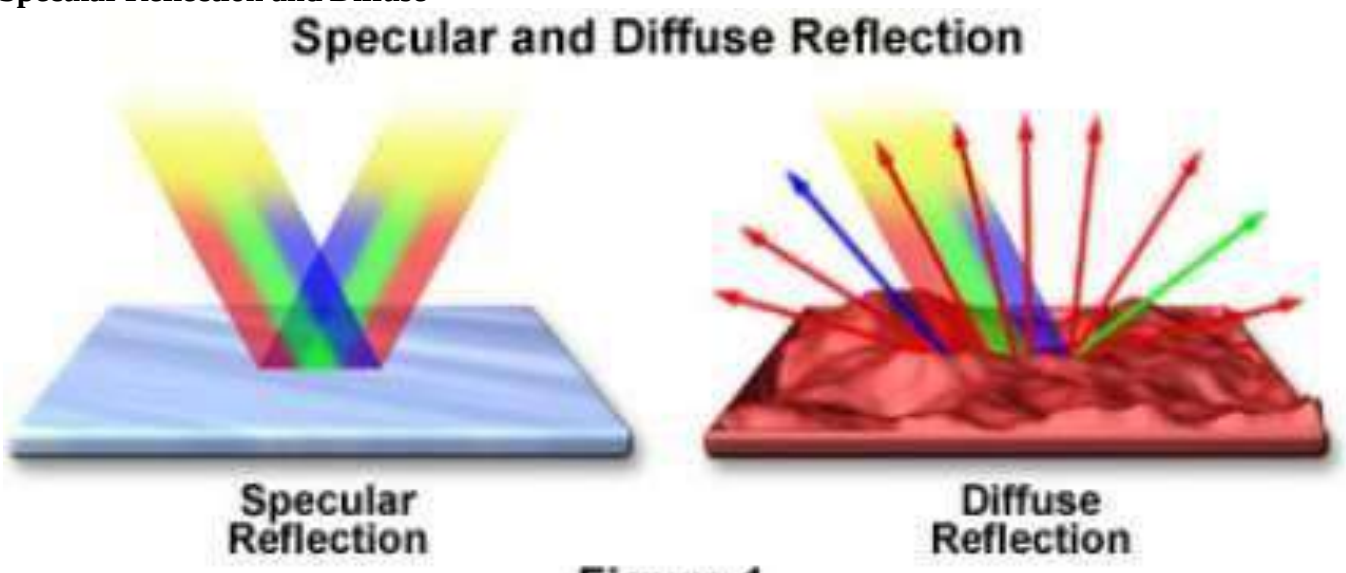


Figure 1

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 mount 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.

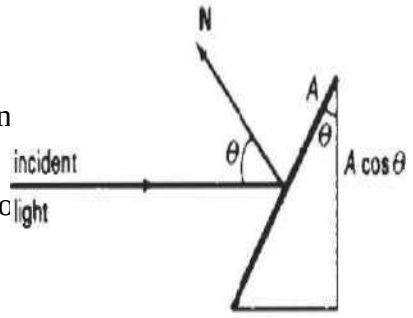
Diffuse

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

$$I_{ambdiff} = k_d I_a$$

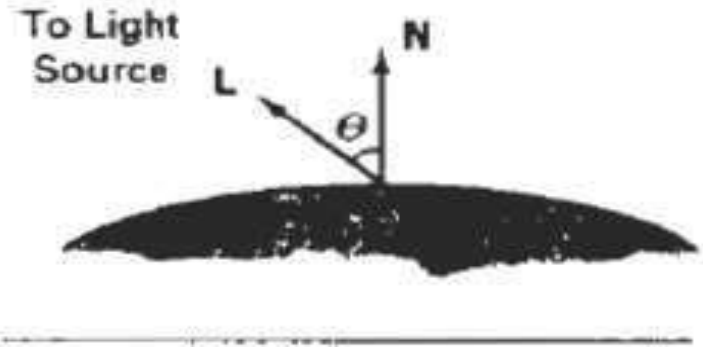
- (where Kd is the diffuse-reflection coefficient)
- If we denote the angle of incidence between the incoming light direction and the surface normal as □ , then the projected area of a surface patch perpendicular to the light direction is proportional to cos□.
- If I_L is the intensity of the point light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

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



- 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□ = N.**L** and the diffuse reflection equation fo single point-source illumination is

$$I_{l,diff} = K_d I_l (N.L)$$



Specular reflection

- When we look at an illuminated shinysurface, **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 specularreflection, 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
- **N** unit normal surface vector
- R to represent the unit vector in the direction of **ideal specularreflection**;
- L to represent the unit vector directed toward the **point light source**;

- **V** as the unit vector pointing to the **viewer** from the surface position.

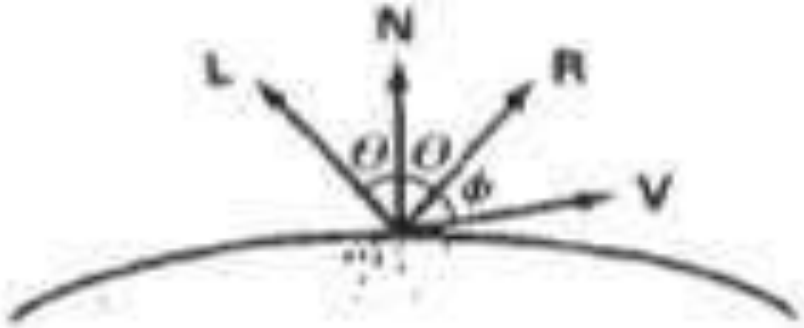


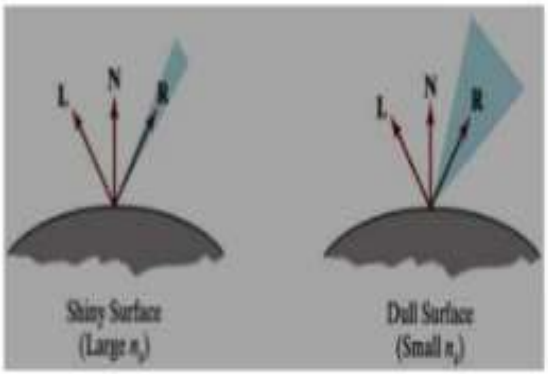
Figure 14-12
Specular-reflection angle equals angle of incidence θ.

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 $\cos^{n_s}\phi$. 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 a0, a1, and a2, to obtain avariety of lighting effects for a scene.

$$f(d) = \frac{1}{a_0 + a_1 d + a_2 d^2}$$

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

$$f(d) = \min\left(1, \frac{1}{a_0 + a_1 d + a_2 d^2}\right) \qquad (14\text{-}12)$$

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

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

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

Activate

Color

- To incorporate color, we need to write the intensity equation as afunction 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 reflation coefficient vector, for example, would then have RGBcomponents (kdR,kdG, kdB).
- 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, kdB
  - The red and green reflectivity components are set to zero(kdR = kdG = 0).
- The intensity calculation for this example reduces to the single expression

$$I_B = k_{aB} I_{aB} + \sum_{i=1}^{n} f_i(d) I_{lBi} [k_{dB}(\mathbf{N} \cdot \mathbf{L}_i) + k_{sB}(\mathbf{N} \cdot \mathbf{H}_i)^{n_s}]$$

Transperence

- Realistic transparency effects are modeledby considering light refraction.
- When light is incident upon a transparentsurface, part of it is reflected and part is **refracted.**

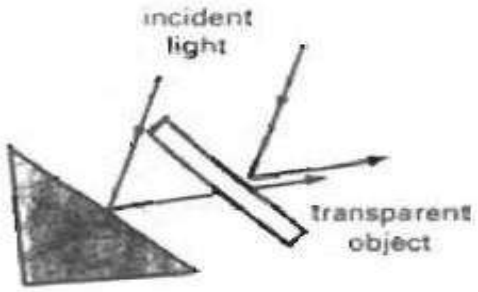


Figure 14-25
Light emission from a transparent surface is in general a combination of reflected and transmitted light.

- ***Snell's Law***
- Angle of refraction **r,** is calculated from the angle of incidence **i,** the index of refraction **ni** of the "incident" material (usually air), and the index of refraction **nr** of the refracting material according to

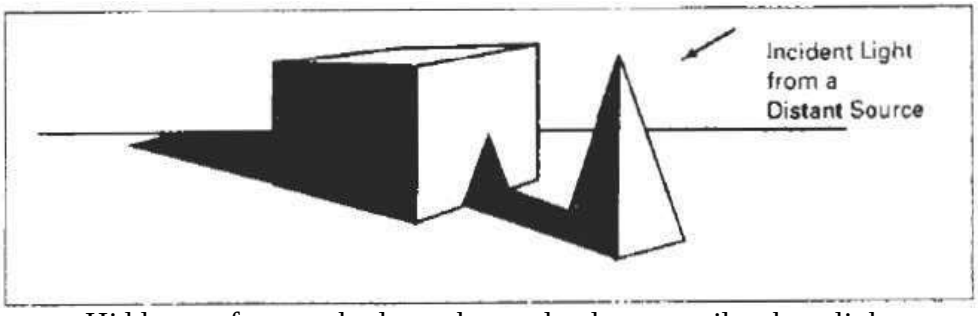
$$\sin \theta_r = \frac{\eta_i}{\eta_r} \sin \theta_i$$

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

$$\mathbf{T} = \left(\frac{\eta_i}{\eta_r} \cos \theta_i - \cos \theta_r \right) \mathbf{N} - \frac{\eta_i}{\eta_r} \mathbf{L} \qquad (14\text{-}18)$$

- **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 areils where lightsources 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.