

Table of Contents

Interactive Computer Graphics:.....	7
Non-Interactive Computer Graphics:	7
Advantages of CG.....	8
1. Enhanced Visual Communication:.....	8
2. Creation of Complex Designs:	8
3. Improved Realism:	8
4. 3D Modeling and Animation:	8
5. Accessibility:.....	8
6. Productivity and Efficiency:.....	8
Image Processing	9
Cathode Ray Tube (CRT).....	10
Features of CRT.....	11
Components of a CRT:.....	12
Types of CRT.....	12
Raster Scan Display	12
Random Scan Display/ vector	14
Colored CRT	17
How it Works:	17
Light Emitting Diode (LED).....	20
Liquid Crystal Display (LCD)	21
1. Frame Buffer	22
2. Pixel Density	23
3. Persistence.....	23
4. Resolution.....	23
5. Megapixel.....	23
6. Refresh Rate.....	24
7. Bit Depth	24
8. Aspect Ratio.....	24

9. Scanline	25
10. Bitmap	25
11. Pixel Map	25
Emissive Displays	25
Non-Emissive Displays	25
Types of graphics softwares.....	26
General-Purpose Graphics Software	26
Special-Purpose Graphics Software	26
Software standards	27
1. GKS (Graphical Kernel System).....	27
2. PHIGS (Programmer's Hierarchical Interactive Graphics System)	28
3. PHIGS+ (PHIGS Plus).....	28
Scan Conversion Algorithm	29
Why Scan Conversion is Needed.....	29
Key Scan Conversion Algorithms.....	29
a) Line Drawing Algorithms	29
b) Circle Drawing Algorithms.....	29
c) Ellipse Drawing Algorithms.....	29
d) Polygon Filling Algorithms.....	29
1. Line Drawing Algorithms.....	29
a) Digital Differential Analyzer (DDA).....	29
DDA algorithm	30
b) Bresenham's Line Algorithm.....	30
Bresenham Line Drawing Algorithm	31
2. Circle Drawing Algorithms.....	32
1. Midpoint Circle Algorithm.....	32
Midpoint Circle Drawing Algorithm	33
1. Bresenham's Circle Algorithm.....	33
Bresenham Circle Algorithm.....	34

3. Ellipse Drawing Algorithms.....	36
a) Midpoint Ellipse Algorithm.....	36
Midpoint Ellipse Algorithm	36
4. Polygon Filling Algorithms	37
a) Scan-Line Polygon Fill Algorithm	37
Scan line Polygon fill Algorithm.....	37
b) Flood Fill and Boundary Fill.....	38
1. Flood Fill	38
Flood fill Algorithm	38
2. Boundary Fill Algorithm.....	39
Boundary Fill Algorithm	39
Homogeneous Coordinates.....	46
Homogeneous Matrix	46
2D viewing pipeline	48
Window to viewport.....	49
Clipping	50
Cohen Sutherland line clipping algorithm	51
Sutherland Hodgman algorithm.....	51
Visible Surface Detection.....	53
Example of Visible Surface Detection.....	53
Object-Space Methods.....	53
Features.....	53
Advantages	54
Disadvantages	54
Types	54
Basic Working of object space.....	54
Image-Space Methods	54
Features.....	54
Advantages	55

Disadvantages	55
Types	55
Basic working.....	55
1. Backface Detection.....	55
Z buffer algorithm.....	57
A Buffer	60
Why linked lists ?	61
Depth Sorting (Painter's Algorithm).....	63
All in one link for the previous algorithm.....	65
scanline method	65
Refresh Rate	67
Electromagnetic Waves and Human Vision	67
Angle deviation from eyes and visual change.....	68
Components of Light	69
What Light is Made Of	70
Depiction of Monochromatic Light	70
Additive Color Model	71
Subtractive Color Model.....	71
HSLvs CMYK vs RGB	73
HSL color wheel.....	74
Rgb COLOR CUBE	75
RGB to HSL conversion	75
HSL to RGB conversion	76
YIQ color model	76
Animation.....	78
12 principles of animation.....	78
1. Squash and Stretch	78
2. Anticipation.....	78
3. Staging	79

4. Straight Ahead Action and Pose-to-Pose.....	79
5. Follow Through and Overlapping Action	79
6. Slow In and Slow Out	79
7. Arc.....	80
8. Secondary Action.....	80
9. Timing	80
10. Exaggeration.....	80
11. Solid Drawing	80
12. Appeal	81
Types of animation	81
Vector art vs bitmap art	83
Frame by frame animation	84
Tweening.....	84
1. Shape Tweens	84
2. Motion Tweens.....	85
Frame-by-Frame Animation vs Tweens	86
Virtual Reality	86
VR vs AR.....	87
Elements of VR	87
Types of VR	88
a. Non-Immersive VR:.....	88
b. Semi-Immersive VR:	88
c. Fully Immersive VR:	88
d. Augmented Virtuality:	88
Components of VR.....	89
1. Hardware Components.....	89
a. Head-Mounted Display (HMD):	89
b. Motion Tracking Sensors:	89
c. Controllers:.....	89

d. Haptic Feedback Devices:	89
e. Audio Devices:	89
f. Input Devices:	89
g. Computing System:	90
2. Software Components.....	90
a. VR Content:	90
b. Simulation Engine:	90
c. Middleware:	90
Working of VR headset	90
a. Display of Virtual Environment:	90
b. Head Tracking:	90
c. Position Tracking:	90
d. Lens Adjustment:	91
e. Field of View (FOV):	91
f. Interactivity:	91
g. Audio Feedback:	91
h. Rendering and Latency:	91

Interactive Computer Graphics:

- **Definition:** Users can interact with the graphics in real-time using input devices like a mouse, keyboard, or touchscreen.
- **Examples:** Video games, virtual reality applications, and interactive simulations.
- **Characteristics:**
 - Two-way communication between the user and the computer.
 - Real-time feedback and updates based on user input.
 - Dynamic and changeable content.

Non-Interactive Computer Graphics:

- **Definition:** Graphics that do not allow user interaction; the content is static and pre-rendered.
- **Examples:** Pre-rendered animations, digital images, and video playback.
- **Characteristics:**
 - One-way communication from the computer to the user.
 - No real-time interaction or feedback.
 - Static and unchangeable content.

Period	Key Developments
1950s	Early experiments with CRT displays and light pens.
1960s	Development of Sketchpad by Ivan Sutherland, introduction of vector graphics.
1970s	Introduction of raster graphics, development of frame buffers, and early 3D graphics.
1980s	Advances in rendering techniques, introduction of GPUs, and the rise of CGI in films.
1990s	Development of real-time 3D graphics, widespread use in video games and simulations.

Period	Key Developments
2000s	Advances in photorealistic rendering, virtual reality, and augmented reality.
2010s-Present	Growth of interactive graphics, AI-driven graphics, and real-time ray tracing.

Advantages of CG

1. Enhanced Visual Communication:

- Creates visually striking and effective images and videos.
- Facilitates the clear and engaging presentation of ideas and information.

2. Creation of Complex Designs:

- Enables intricate and detailed designs that are difficult to achieve manually.
- Useful in fields like architecture, engineering, and product design.

3. Improved Realism:

- Produces photorealistic images and videos that closely mimic real-world objects and scenes.
- Enhances the visual appeal and accuracy of digital content.

4. 3D Modeling and Animation:

- Allows for the creation of 3D models and animations used in films, video games, and simulations.
- Provides a realistic and immersive experience for users.

5. Accessibility:

- Makes visual content more accessible to people with visual impairments through options like text-to-speech and high contrast modes.

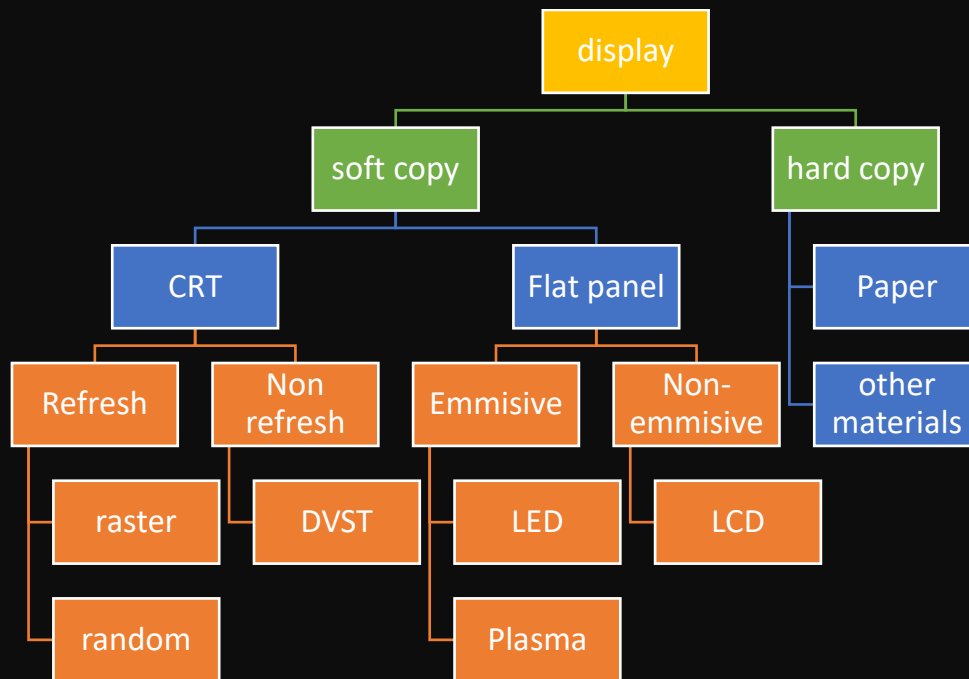
6. Productivity and Efficiency:

- Reduces the time and cost associated with design and analysis.
- Increases productivity by automating repetitive tasks and providing precise results.

Image Processing

Image processing involves the manipulation and analysis of digital images to enhance their quality or extract useful information. This field encompasses a variety of techniques, including image enhancement, restoration, encoding, and compression. The primary goal of image processing is to improve the visual appearance of an image or to prepare it for further analysis. Applications of image processing are vast and include medical imaging, satellite image analysis, digital photography, and computer vision. By converting images into a digital form and applying algorithms, image processing can correct distortions, enhance features, and even recognize patterns within the images

Computer Graphics	Image Processing
Creation and manipulation of visual content using computers.	Manipulation and analysis of existing images to enhance quality or extract information.
To generate visual content such as images, animations, and simulations.	To improve image quality or extract meaningful information from images.
Video games, animated movies, virtual reality.	Medical imaging, satellite imagery, digital photo enhancement.
Rendering, modeling, shading, texturing.	Filtering, transformation, segmentation, compression.
New visual content created from scratch or modified existing content.	Enhanced or analyzed version of the original image.
Often interactive, allowing real-time changes and feedback.	Generally non-interactive, focusing on processing the image data.
Entertainment, design, simulation, virtual reality.	Medical diagnostics, remote sensing, digital photography, surveillance.



Cathode Ray Tube (CRT)

A CRT is a vacuum tube containing one or more electron guns, which emit electron beams. These beams are directed towards a phosphorescent screen to create images. Here's a step-by-step breakdown:

1. **Electron Emission:** The electron gun at the back of the CRT emits electrons through thermionic emission, where a heated cathode releases electrons.
2. **Acceleration and Focusing:** These electrons are accelerated and focused into a narrow beam by anodes.
3. **Deflection:** The electron beam is deflected by magnetic or electrostatic deflection coils to scan across the screen in a raster pattern.
4. **Phosphor Screen:** When the electron beam strikes the phosphor-coated screen, it emits light, creating visible images. Different phosphors emit different colors, allowing for color displays

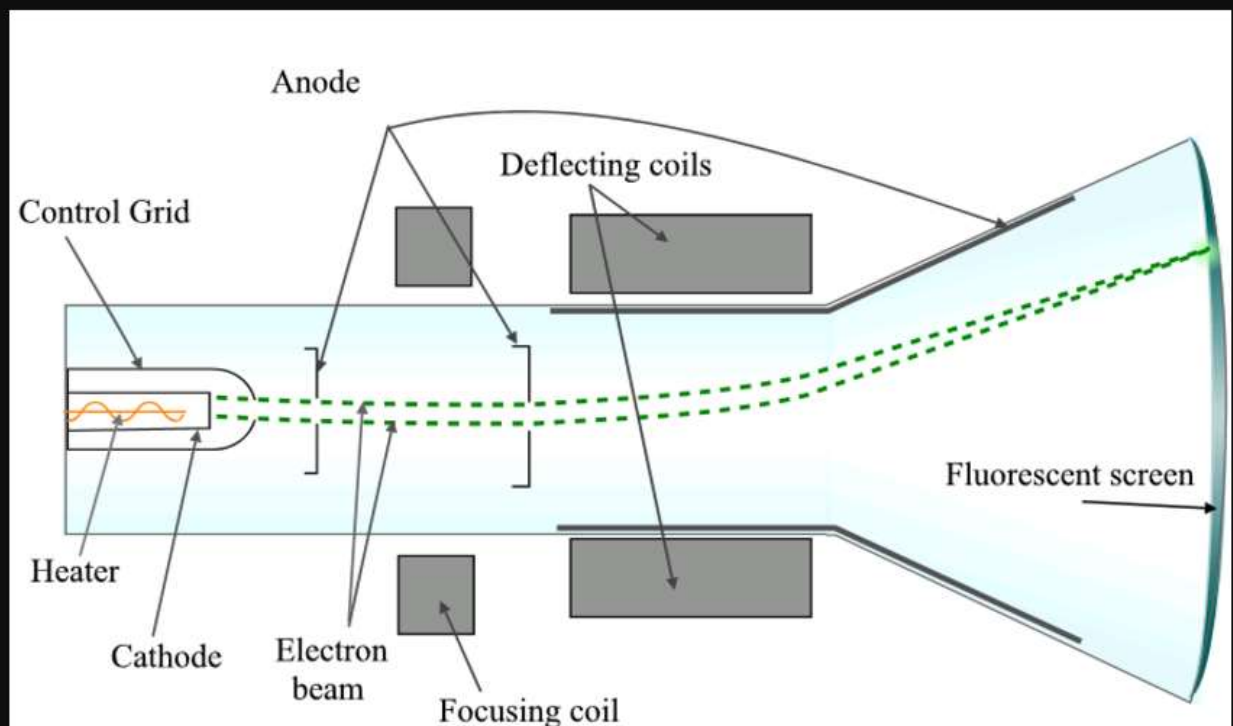


Fig: CRT tube

Features of CRT

1. Travel in Straight Lines:

- Cathode rays move in straight lines from the cathode to the anode.
- They cast sharp shadows when objects are placed in their path.

2. Negative Charge:

- Cathode rays are deflected by electric and magnetic fields, indicating they carry a negative charge.
- This negative charge is due to the electrons emitted from the cathode.

3. High Velocity:

- Cathode rays travel at high speeds.
- This high velocity contributes to their ability to excite the phosphor coating on the screen, producing visible light.

4. Cause Fluorescence:

- When cathode rays strike a phosphor-coated surface, they cause it to emit light.
- This property is utilized in CRT displays to produce images.

5. Ionize Gases:

- Cathode rays can ionize gases, meaning they can knock electrons off gas atoms, creating ions.
- This property is used in various scientific instruments and applications.

6. Mechanical Effects:

- Cathode rays can exert a mechanical force on objects they strike.
- This property was historically used in early forms of radiometry.

7. Independent of the Material of the Cathode:

- The properties of cathode rays remain the same regardless of the material used for the cathode.
- This suggests that cathode rays are a fundamental property of matter.

Components of a CRT:

1. **Electron Gun:** Emits a focused beam of electrons.
2. **Deflection System:** Controls the direction of the electron beam.
3. **Phosphor Screen:** Converts the kinetic energy of the electrons into light.
4. **Evacuated Glass Envelope:** Maintains a vacuum to prevent interference with the electron beam

Types of CRT

Raster Scan Display

A **raster scan display** is the most common type of CRT display used in televisions and computer monitors. In this system, the electron beam scans the screen in a series of horizontal lines from top to bottom. Each line is drawn from left to right, and after reaching the end of a line, the beam moves to the beginning of the next line. This process continues until the entire screen is covered. The image is refreshed many times per second to create a stable picture. Raster scan displays are suitable for displaying complex images and realistic scenes

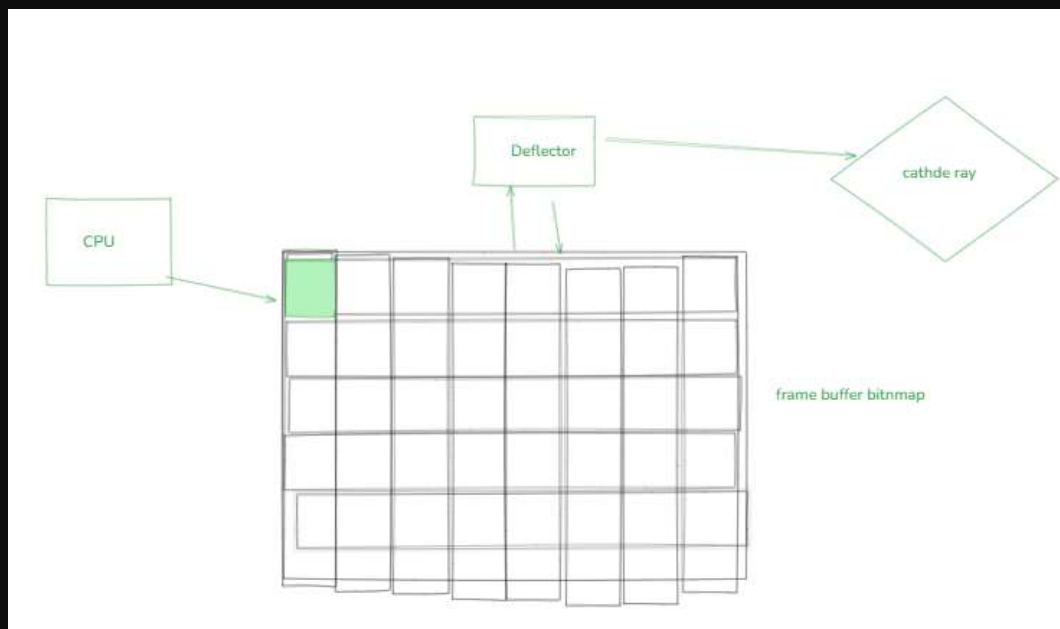
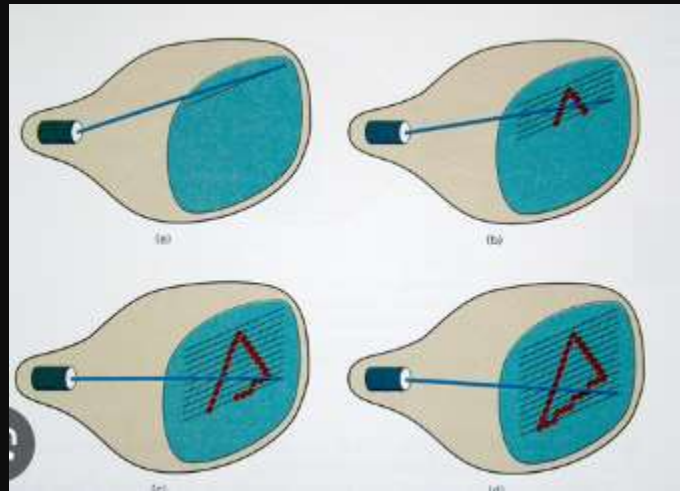
Advantages:

- Capable of displaying complex and realistic images.
- Supports a wide range of colors and shades.

Disadvantages:

- Lower resolution compared to random scan displays.

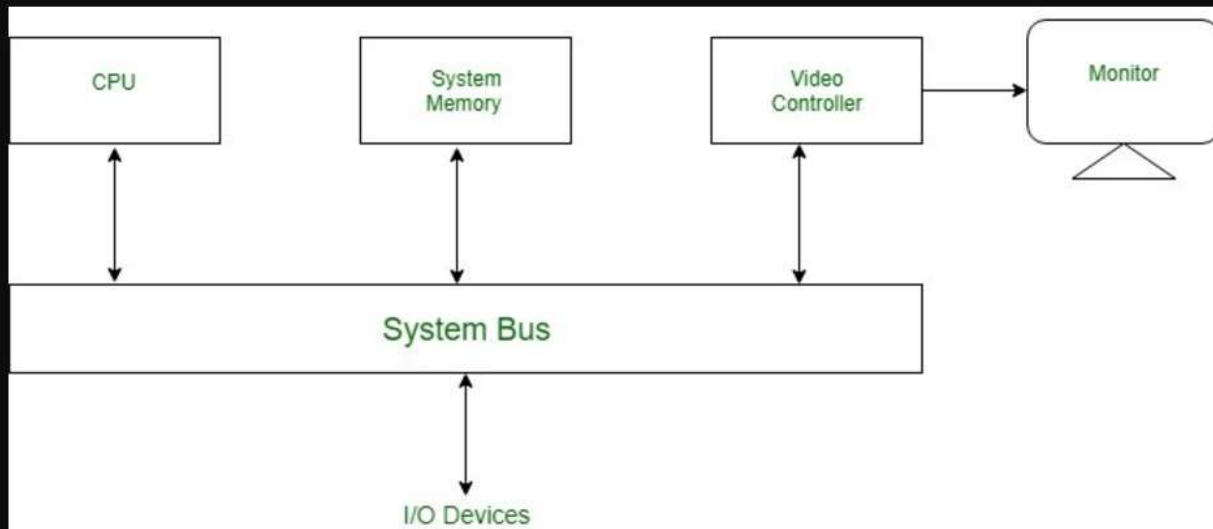
- Can suffer from flickering if the refresh rate is too low.



Working of Raster scan system

- **Electron Beam:** An electron beam is emitted from an electron gun within the display tube.
- **Deflection System:** This system controls the direction of the electron beam, moving it across the screen in a horizontal line.
- **Phosphor Coating:** The inside of the screen is coated with a phosphor material. When struck by the electron beam, the phosphor emits light.
- **Raster Scanning:** The electron beam scans the screen line by line, from top to bottom. Each line is called a scan line.

- **Intensity Modulation:** The intensity of the electron beam is modulated to control the brightness of the pixels. This modulation is based on the image data stored in the display's memory.
- **Refresh Rate:** To maintain a stable image, the entire screen is scanned multiple times per second. This is called the refresh rate, typically measured in Hertz (Hz). A higher refresh rate reduces flicker and improves image quality.



Types under Raster display

1. Horizontal
2. Vertical

Random Scan Display/ vector

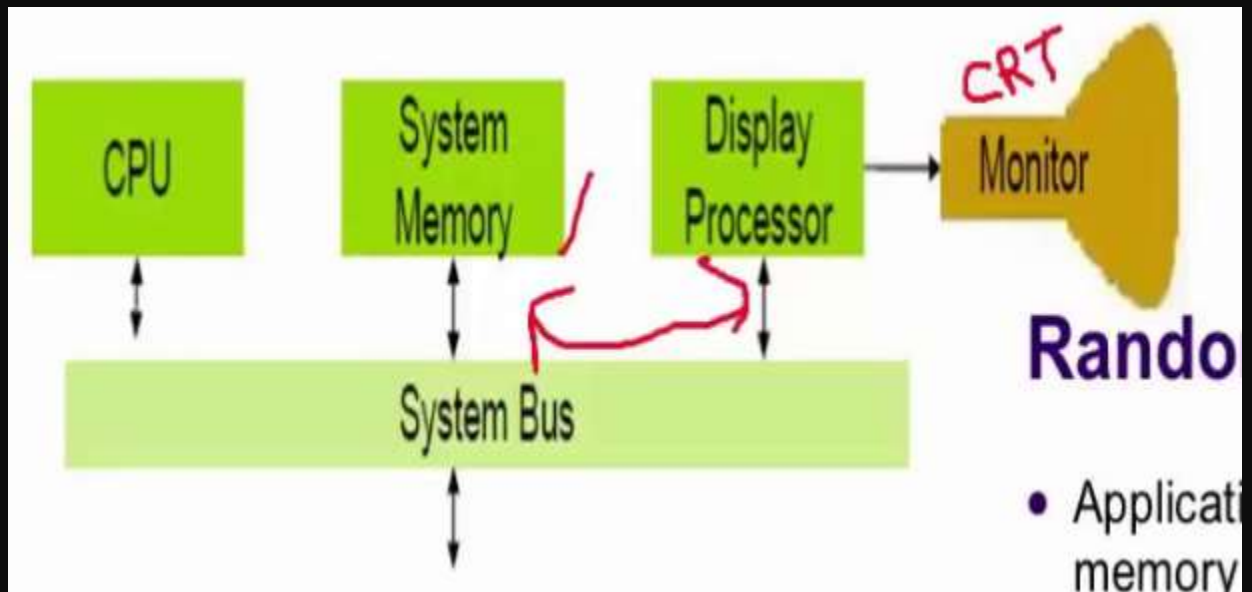
A **random scan display**, also known as a vector display or stroke-writing display, uses an electron beam that directly draws the image on the screen by moving to the specific coordinates of each line segment. This method is more efficient for drawing images composed of lines, such as engineering drawings or wireframe models. The beam only illuminates the parts of the screen where the image is to be drawn, resulting in higher resolution for line drawings.

Advantages:

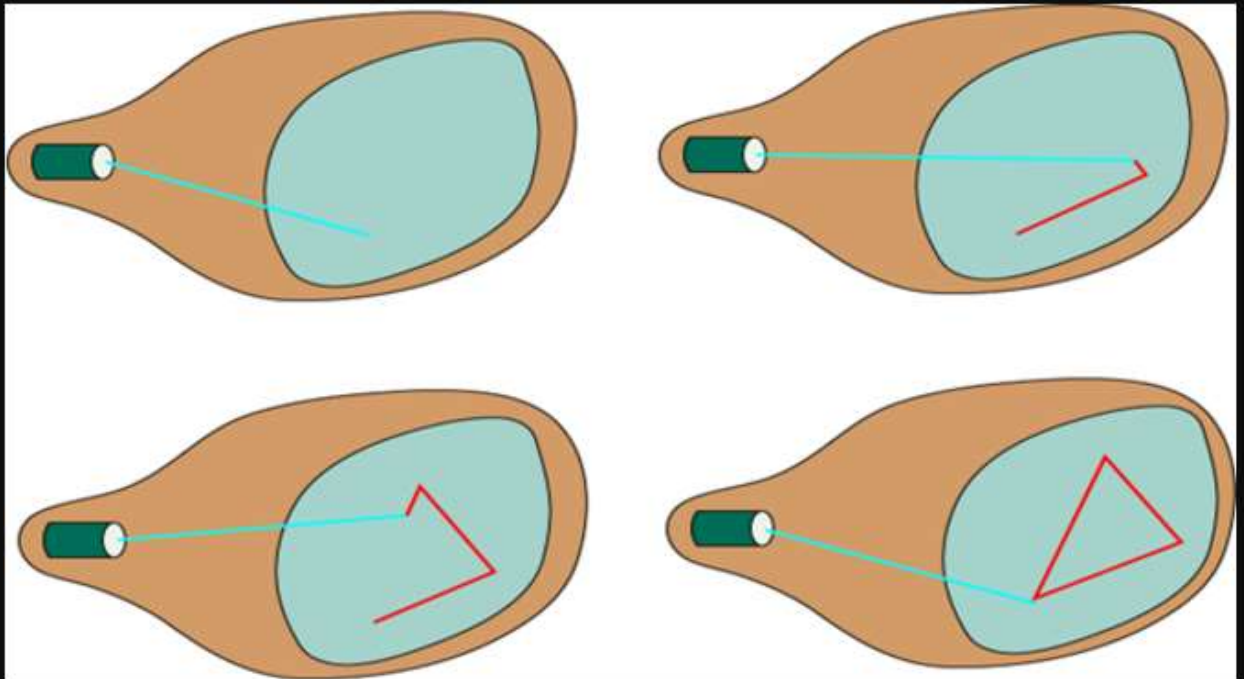
- High resolution for line drawings.
- Efficient for displaying vector graphics.

Disadvantages:

- Not suitable for displaying complex images with many colors.
- More expensive and complex to implement.



- **Display File:** A display file stores the image definition as a set of line-drawing commands. Each command specifies the starting and ending coordinates of a line.
- **Display Processor:** The display processor reads the commands from the display file and controls the electron beam to draw the lines on the screen.
- **Electron Beam:** The electron beam is directed to the starting point of a line, then it is deflected to the ending point, drawing the line directly on the screen.
- **Refresh Cycle:** The display processor continuously cycles through the display file, refreshing the image on the screen.



Feature	Random Scan	Raster Scan
Image Creation	Draws lines and curves directly	Scans the screen line by line, pixel by pixel
Image Storage	Display file stores line and curve definitions	Frame buffer stores pixel intensity values
Refresh Rate	Lower refresh rate (30-60 Hz)	Higher refresh rate (60-80 Hz)
Image Quality	High-quality lines and curves, but limited color and shading	Lower-quality lines and curves, but better for complex images and realistic scenes
Complexity of Images	Best suited for simple graphics	Best suited for complex images and realistic scenes
Efficiency	Efficient for simple graphics	Efficient for complex images
Cost	More expensive	Less expensive

Common Applications	CAD systems, scientific visualization	Televisions, computer monitors
----------------------------	---------------------------------------	--------------------------------

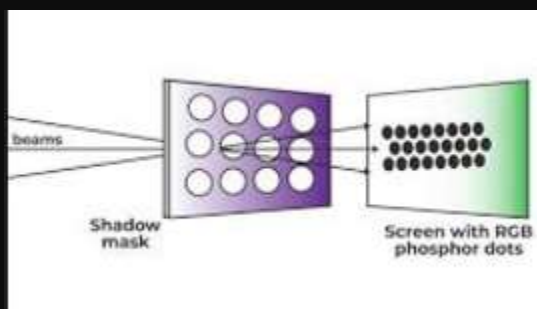
Colored CRT

A color CRT (Cathode-Ray Tube) is a type of display technology that uses three separate electron guns to produce red, green, and blue colors on the screen.

By combining these primary colors in varying intensities, a wide range of colors can be displayed

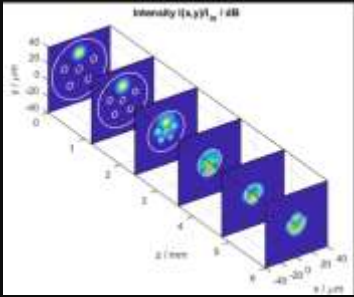
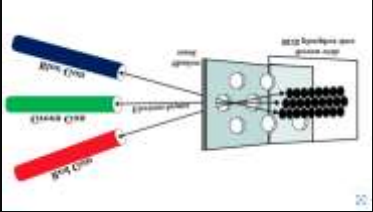
How it Works:

1. **Electron Guns:** Three electron guns emit beams of electrons, one for each primary color (red, green, and blue).
2. **Deflection System:** The electron beams are deflected horizontally and vertically by magnetic fields, allowing them to scan the entire screen.
3. **Phosphor Coating:** The inside of the screen is coated with phosphor dots, which emit light when struck by the electron beams. Each dot is sensitive to a specific color (red, green, or blue).
4. **Shadow Mask:** A shadow mask, a perforated metal plate, is placed behind the phosphor screen to ensure that each electron beam strikes the correct color phosphor dot.
5. **Color Formation:** By controlling the intensity of each electron beam, different colors can be produced. For example, combining red and green produces yellow, red and blue produces magenta, and green and blue produces cyan.



Difference between two popular methods in producing color in CRT Beam Penetration and Shadow Mask :

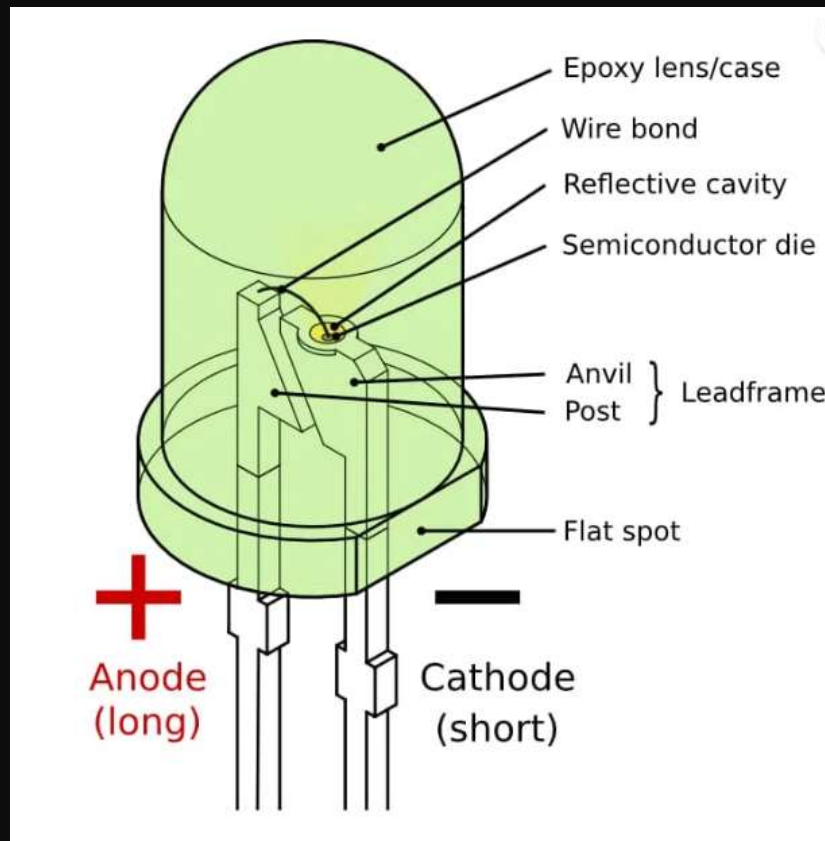
Basis	Beam Penetration	Shadow Mask
Colors produced	In this method, there is the production of only four colors i.e., red, green, yellow, orange.	In this method, there is the production of millions of colors.
Color dependency	As in this method only four colors are produced it is because of the speed of the electron gun.	As in this method millions of colors are produced because it depends upon the intensity value of the three available guns.
Number of electron guns used.	In this method, only one electron gun is used.	In this methods, three electron guns are used; i.e red, green and blue.
Picture quality	As we know in this different colors and shades are not possible. So, it's picture quality is poor.	As we know in this different colors and shades are possible. So, it's picture quality is quite good.
Realistic view	This method is not suitable for providing the realistic view.	This method is suitable for providing the realistic view.
Resolution	This method provides high resolution.	Whereas, this method does not able to provide high resolution.
Cost	It is cheaper than shadow mask method.	It is an expensive method.
Application	It is used in random scan system to display color.	It is used in raster scan system to display color.

Basis	Beam Penetration	Shadow Mask
		
Implications	<p>Advantages:</p> <ul style="list-style-type: none"> • Produces realistic images with a wide range of colors. • Capable of generating millions of different colors. • Suitable for detailed and shaded scenes <p>Disadvantages:</p> <ul style="list-style-type: none"> • Relatively expensive compared to monochrome CRTs. • Lower resolution compared to some other methods. • Potential convergence problems where electron beams may not align perfectly 	<p>Advantages:</p> <ul style="list-style-type: none"> • Inexpensive to implement. • Simple construction and operation <p>Disadvantages:</p> <ul style="list-style-type: none"> • Limited to only four colors (red, green, orange, and yellow). • Poorer image quality and resolution compared to shadow mask. • Not suitable for producing realistic images or detailed shading

Light Emitting Diode (LED)

LEDs are semiconductor devices that emit light when an electric current passes through them. Here's how they work:

1. **Semiconductor Material:** LEDs are made from materials like Gallium Arsenide (GaAs) that have a direct band gap.
2. **Electron-Hole Recombination:** When a forward voltage is applied, electrons and holes recombine at the junction, releasing energy in the form of photons (light). This process is called electroluminescence.
3. **Color Emission:** The color of the emitted light depends on the semiconductor material's band gap. For example, GaAs emits infrared light, while Gallium Phosphide (GaP) emits green light



LED diode

Feature	CRT (Cathode Ray Tube)	LCD (Liquid Crystal Display)
Image Formation	Electron beam scans phosphor-coated screen	Liquid crystals modulate light from a backlight
Size and Weight	Larger, bulkier, and heavier	Slim, lightweight, and compact
Power Consumption	Higher power consumption	Lower power consumption
Color Accuracy	Excellent color accuracy and contrast	Good color accuracy, improving with technology
Resolution	Generally lower resolution	Higher resolution
Viewing Angle	Limited viewing angles	Wide viewing angles
Response Time	Fast response time	Slower response time, but improving
Image Flickering	Can experience flickering	Minimal flickering
Burn-in Risk	No burn-in risk	Potential burn-in risk (especially in OLEDs)
Magnetic Interference	Susceptible to magnetic fields	Not affected by magnetic fields
Cost	Generally lower cost	Higher cost

Liquid Crystal Display (LCD)

LCDs use liquid crystals to modulate light and create images. Here's a detailed look at their operation:

1. **Liquid Crystals:** LCDs contain liquid crystals that can change their orientation when an electric field is applied.

2. **Polarizing Filters:** Two polarizing filters are placed at right angles to each other with liquid crystals sandwiched between them.
3. **Light Modulation:** When an electric current is applied, the liquid crystals align in such a way that they either block or allow light to pass through the filters.
4. **Backlight:** Most LCDs use a backlight to illuminate the display. The light passes through the liquid crystals and polarizing filters to create the desired image

Feature	LCD (Liquid Crystal Display)	LED (Light Emitting Diode)
Backlighting	Uses Cold Cathode Fluorescent Lamps (CCFL)	Uses LED backlighting
Energy Efficiency	Less energy-efficient	More energy-efficient
Brightness	Lower brightness	Higher brightness
Contrast Ratio	Lower contrast ratio	Higher contrast ratio
Thickness	Thicker due to CCFL backlighting	Thinner due to LED backlighting
Color Accuracy	Good color accuracy	Better color accuracy and vibrancy
Lifespan	Shorter lifespan	Longer lifespan
Local Dimming	Limited or no local dimming	Supports local dimming for better blacks
Viewing Angle	Narrower viewing angles	Wider viewing angles
Cost	Generally cheaper	Generally more expensive

1. Frame Buffer

A **frame buffer** is a portion of RAM containing a bitmap that drives a video display. It stores the pixel data that will be displayed on the screen. The frame buffer holds

color values for every pixel, which can be in various formats like 1-bit binary (monochrome), 4-bit palettized, 8-bit palettized, 16-bit high color, and 24-bit true color. Modern video cards have frame buffer circuitry that converts this data into a video signal

$$\text{Memory(MB)} = \frac{x_{\text{resolution}} * y_{\text{resolution}} * \text{bit}_{\text{per_pixel}}}{8 * 1024^2}$$

2. Pixel Density

Pixel density is measured in pixels per inch (PPI) and indicates how many pixels are packed into a given inch of the display. Higher pixel density means sharper images and more detailed visuals. For example, a 24-inch monitor with a resolution of 1920x1080 has a pixel density of about 92 PPI, while a 27-inch monitor with the same resolution has about 82 PPI

$$\text{pixel density} = \frac{\text{sqrt}(\text{width}^2 + \text{length}^2)}{\text{screen}_{\text{size}}(\text{diagonal})}$$

$$\text{pixel density} = \frac{\text{diagonal}_{\text{resolution}}}{\text{display}_{\text{size}}}$$

3. Persistence

Persistence refers to how long a pixel remains illuminated after being activated. Lower persistence reduces motion blur in fast-moving images, which is crucial for applications like gaming and virtual reality. High persistence can cause ghosting effects, where previous frames linger on the screen

4. Resolution

Resolution is the number of distinct pixels in each dimension that can be displayed. It is often described in terms of width and height, such as 1920x1080 (Full HD) or 3840x2160 (4K). Higher resolution means more detail and clarity in the image. For example, a 4K display has four times the number of pixels as a Full HD display

5. Megapixel

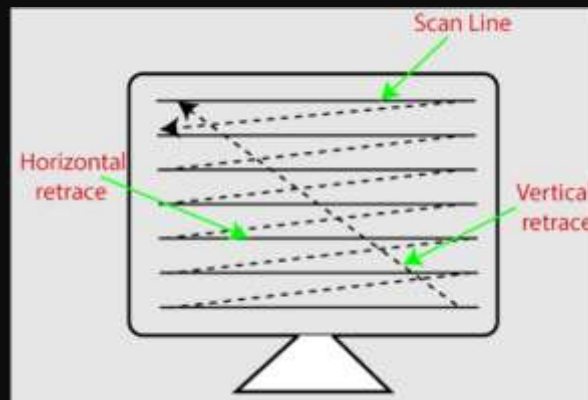
A **megapixel** equals one million pixels and is commonly used to describe the resolution of digital cameras and displays. For instance, a 12-megapixel camera can capture images

with 12 million pixels, providing high detail and clarity. The number of megapixels affects the size and quality of the images that can be printed or displayed

$$\text{megapixel} = \frac{\text{resolution(like 1080x1920)}}{1,000,000}$$

6. Refresh Rate

Refresh rate is measured in Hertz (Hz) and indicates how many times per second the display updates with new information. A higher refresh rate, such as 120Hz or 144Hz, results in smoother motion and is particularly beneficial for gaming and fast-moving video content. Standard refresh rates include 60Hz, 120Hz, and 144Hz number of times or second image is redrawn to give feeling of non flickering pictures is called refresh rate



7. Bit Depth

Bit depth refers to the number of bits used to represent the color of a single pixel. Higher bit depth allows for more colors and better color accuracy. For example, an 8-bit depth can display 256 colors per channel, while a 10-bit depth can display 1024 colors per channel. This results in smoother gradients and more detailed images

8. Aspect Ratio

Aspect ratio is the ratio of the width to the height of the display. Common aspect ratios include 16:9 (widescreen), 4:3 (traditional TV), and 21:9 (ultrawide). The aspect ratio affects how content is displayed and can influence the viewing experience. For example, a 16:9 aspect ratio is ideal for most modern video content, while 4:3 is more suited for older TV shows and some computer monitors

$$\text{aspect ratio of image} = \frac{\text{width}_{\text{image}}}{\text{height}_{\text{image}}}$$

9. Scanline

A **scanline** is a horizontal line of pixels that the display's electron beam draws on the screen. In raster graphics, the image is composed of multiple scanlines, each representing a row of pixels. The electron beam moves from left to right, drawing each pixel in the line, then moves down to the next line and repeats the process until the entire screen is filled

10. Bitmap

A **bitmap** is a type of memory organization or image file format used to store digital images. The bitmap consists of a matrix of pixels, where each pixel represents a single point in the image. The color of each pixel is stored as a binary value, with different bit depths determining the number of colors that can be represented. For example, a 1-bit bitmap can represent two colors (black and white), while a 24-bit bitmap can represent over 16 million colors

11. Pixel Map

A **pixel map** (or **pixmap**) is similar to a bitmap but typically refers to images with more color depth. While bitmaps are often associated with monochrome or limited color images, pixel maps can handle full-color images. Each pixel in a pixel map has multiple bits to represent the color, allowing for a wide range of colors and more detailed images

Emissive Displays

Emissive displays generate their own light. Examples include:

- **Plasma Panels:** High contrast, wide viewing angles.
- **LED:** Bright, energy-efficient, long lifespan.
- **OLED:** Excellent color, deep blacks, flexible screens.

Advantages: High brightness, contrast, and fast response times. **Disadvantages:** More expensive, potential burn-in (especially OLEDs).

Non-Emissive Displays

Non-emissive displays use external light sources. Examples include:

- **LCD:** Uses a backlight, common in monitors and TVs.
- **E-Paper:** Reflects ambient light, used in e-readers.

Advantages: Lower power consumption, no burn-in risk, generally cheaper. **Disadvantages:** Limited viewing angles, requires a backlight (for LCDs).

Types of graphics softwares

General-Purpose Graphics Software

General-purpose graphics software provides a wide range of graphics functions that can be used in various programming languages like C or FORTRAN. These packages are designed for developers and programmers who need extensive control over graphics operations. They typically include functions for:

- **Drawing Primitives:** Basic shapes like lines, circles, and polygons.
- **Attributes:** Properties like color, line style, and fill patterns.
- **Transformations:** Operations like translation, rotation, and scaling.
- **Input Handling:** Functions to manage user input from devices like a mouse or keyboard.

Examples:

- **OpenGL:** A cross-language, cross-platform API for rendering 2D and 3D vector graphics.
- **DirectX:** A collection of APIs for handling tasks related to multimedia, especially game programming and video.

Special-Purpose Graphics Software

Special-purpose graphics software is designed for specific tasks and is often user-friendly, requiring no programming knowledge. These applications are tailored for non-programmers to create and manipulate graphics easily. They include:

- **Painting Programs:** Used for creating and editing bitmap images. Examples include Adobe Photoshop and GIMP.
- **Drawing Programs:** Focus on vector graphics, allowing users to create scalable images. Examples include Adobe Illustrator and CorelDRAW.

- **CAD (Computer-Aided Design) Programs:** Used for precision drawing and design, often in engineering and architecture. Examples include AutoCAD and SolidWorks.
- **Animation Software:** Used for creating animations and visual effects. Examples include Adobe After Effects and Blender.

Examples:

- **Adobe Photoshop:** A powerful tool for photo editing and graphic design.
- **CorelDRAW:** A vector graphics editor for creating illustrations, logos, and complex designs.
- **AutoCAD:** A CAD software application for 2D and 3D design and drafting.

Software standards

1. GKS (Graphical Kernel System)

GKS was the first international standard for computer graphics, established by the International Standards Organization (ISO) and the American National Standards Institute (ANSI). It provides a set of functions for creating 2D graphics, including:

- **Drawing Primitives:** Functions to draw basic shapes like lines, circles, and polygons.
- **Attributes:** Functions to set properties like color, line style, and fill patterns.
- **Transformations:** Functions to perform operations like translation, rotation, and scaling.
- **Input Handling:** Functions to manage user input from devices like a mouse or keyboard

Advantages:

- Standardized, ensuring portability across different systems.
- Provides a comprehensive set of 2D graphics functions.

Disadvantages:

- Limited to 2D graphics, with extensions needed for 3D.

2. PHIGS (Programmer's Hierarchical Interactive Graphics System)

PHIGS is a standard for 3D graphics, developed to provide more advanced capabilities than *GKS*. It includes features for:

- **Object Modeling:** Creating and manipulating complex 3D objects.
- **Hierarchical Structures:** Organizing objects in a hierarchical manner for efficient rendering.
- **Color Specifications:** Defining colors and shading for objects.
- **Surface Rendering:** Techniques for rendering surfaces with different properties

Advantages:

- Supports complex 3D graphics and hierarchical structures.
- Provides advanced modeling and rendering capabilities.

Disadvantages:

- More complex to implement and use compared to *GKS*.
- Initially lacked some advanced rendering features, which were later added in *PHIGS+*.

3. PHIGS+ (PHIGS Plus)

PHIGS+ is an extension of *PHIGS* that adds more advanced 3D graphics capabilities, including:

- **Surface Shading:** Techniques for realistic shading of 3D surfaces.
- **Lighting Models:** Methods for simulating light sources and their effects on objects.
- **Advanced Primitives:** Support for more complex shapes like Non-Uniform Rational B-Splines (NURBS)

Scan Conversion Algorithm

Scan conversion is a process in computer graphics that involves converting geometric primitives (like lines, circles, ellipses, and polygons) into a pixel-based representation on a raster display. This process determines which pixels should be illuminated to best approximate the shape of the primitive.

Why Scan Conversion is Needed

Raster displays consist of a grid of pixels, and geometric shapes like lines and circles must be mapped onto this grid. Since the shapes are mathematically continuous and the display is discrete, scan conversion is required to approximate these shapes on the screen.

Key Scan Conversion Algorithms

a) Line Drawing Algorithms

- Digital Differential Analyzer (DDA) Algorithm
- Bresenham's Line Algorithm

b) Circle Drawing Algorithms

- Midpoint Circle Algorithm
- Bresenham's Circle Algorithm

c) Ellipse Drawing Algorithms

- Midpoint Ellipse Algorithm

d) Polygon Filling Algorithms

- Scan-Line Polygon Fill Algorithm
- Flood Fill and Boundary Fill Algorithms

1. Line Drawing Algorithms

These algorithms find the pixels that form the best approximation of a line between two points.

a) Digital Differential Analyzer (DDA)

- Uses floating-point calculations.
- It incrementally plots points along the line by calculating the next point based on the slope.
- Slower due to floating-point operations but simple to implement.

- Otherwise, increment y by 1 and calculate x .

DDA algorithm

Given two endpoints of the line (x_0, y_0) and (x_1, y_1) :

1. Calculate the Differences:

$$\Delta x = x_1 - x_0$$

$$\Delta y = y_1 - y_0$$

2. Determine the Number of Steps: The number of steps required to draw the line is determined by the greater difference (in either the x or y direction):

$$\text{Steps} = \max(|\Delta x|, |\Delta y|)$$

3. Calculate the Increments: Determine the increment in the x and y coordinates for each step:

$$\text{X Increment} = \frac{\Delta x}{\text{Steps}}$$

$$\text{Y Increment} = \frac{\Delta y}{\text{Steps}}$$

4. Initialize Starting Point: Set the starting point (x, y) to (x_0, y_0) .

5. Iterate and Plot Points: For each step from 1 to the total number of steps:

- Plot the point $(\text{round}(x), \text{round}(y))$.
- Update the coordinates:

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

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

b) Bresenham's Line Algorithm

- Uses integer arithmetic, making it faster and more efficient than DDA.
- Determines which pixel is closer to the line path using a decision parameter.
- Works well for all types of slopes.

Key Idea:

- It starts at one endpoint and iteratively determines the next pixel to plot based on the error value (or decision parameter).

Bresenham Line Drawing Algorithm

Given two endpoints (x_0, y_0) and (x_1, y_1) :

1. Initialize Variables:

- Calculate differences:

$$\Delta x = x_1 - x_0$$

$$\Delta y = y_1 - y_0$$

- Initialize the starting point $(x, y) = (x_0, y_0)$.

- Determine the **decision parameter**:

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

2. Iterate Over Pixels:

- For each x from x_0 to x_1 :

- Plot the current point (x, y) .

- If the decision parameter $p < 0$:

- Move **horizontally**: $x = x + 1$

- Update the decision parameter:

$$p = p + 2\Delta y$$

- Else (if $p \geq 0$):

- Move **diagonally**: $x = x + 1, y = y + 1$

- Update the decision parameter:

$$p = p + 2\Delta y - 2\Delta x$$

S.NO	DDA Line Algorithm	Bresenham line Algorithm
1.	DDA stands for Digital Differential Analyzer.	While it has no full form.
2.	DDA algorithm is less efficient than Bresenham line algorithm.	While it is more efficient than DDA algorithm.
3.	The calculation speed of DDA algorithm is less than Bresenham line algorithm.	While the calculation speed of Bresenham line algorithm is faster than DDA algorithm.

S.NO	DDA Line Algorithm	Bresenham line Algorithm
4.	DDA algorithm is costlier than Bresenham line algorithm.	While Bresenham line algorithm is cheaper than DDA algorithm.
5.	DDA algorithm has less precision or accuracy.	While it has more precision or accuracy.
6.	In DDA algorithm, the complexity of calculation is more complex.	While in this, the complexity of calculation is simple.
7.	In DDA algorithm, optimization is not provided.	While in this, optimization is provided.

2. Circle Drawing Algorithms

These algorithms find the pixels that form the best approximation of a circle.

1. Midpoint Circle Algorithm

- Uses symmetry to reduce computations by calculating only the points in one octant and reflecting them.
- Based on the decision parameter to determine the next pixel.

Steps:

1. Start at the topmost point $(0, r)(0, r)(0, r)$.
2. Use the decision parameter $p = 1 - r_p = 1 - r_p = 1 - r$ to decide whether to move horizontally or diagonally.
3. Reflect the calculated points in the other seven octants.

Midpoint Circle Drawing Algorithm

Algorithm Steps

1. Initialize Variables:

- Start at the top of the circle: $(x, y) = (0, r)$.
- Calculate the initial decision parameter:

$$p = 1 - r$$

2. Iterate Over Octant:

- While $x \leq y$:
 - Plot the points (x, y) and their reflections in the other octants.
 - If $p < 0$:
 - Increment x : $x = x + 1$
 - Update the decision parameter:
$$p = p + 2x + 1$$
 - Else:
 - Increment x and decrement y : $x = x + 1, y = y - 1$
 - Update the decision parameter:
$$p = p + 2x - 2y + 1$$

1. Bresenham's Circle Algorithm

- An optimized version of the Midpoint Circle Algorithm using only integer calculations.
- It avoids floating-point arithmetic, making it efficient.

Bresenham Circle Algorithm

1. Initialize Variables:

- Start at the top of the circle: $(x, y) = (0, r)$.
- Set the initial decision parameter:

$$p = 3 - 2r$$

2. Iterate Over the First Octant:

- While $x \leq y$:
 - Plot the points in all 8 octants using symmetry:
 - $(x, y), (y, x), (-x, y), (-y, x), (-x, -y), (-y, -x), (x, -y), (y, -x)$
 - If $p < 0$:
 - Increment x : $x = x + 1$
 - Update the decision parameter:

$$p = p + 4x + 6$$
 - Else:
 - Increment x and decrement y : $x = x + 1, y = y - 1$
 - Update the decision parameter:

$$p = p + 4x - 4y + 10$$

Criteria	Bresenham's Circle	Midpoint Circle	General Circle
Basic Idea	Uses integer calculations and a decision parameter to determine the next point.	Uses a decision parameter to choose the next point based on midpoint testing.	Uses the general equation $x^2 + y^2 = r^2$ for calculations.
Calculations	Integer arithmetic only, no floating-point calculations.	Integer arithmetic with simple updates using midpoint checks.	Requires floating-point operations (square root).

Criteria	Bresenham's Circle	Midpoint Circle	General Circle
Efficiency	High efficiency due to integer-only calculations.	High efficiency but slightly less than Bresenham's due to conditional checks.	Least efficient due to floating-point arithmetic.
Symmetry Utilized	Uses 8-way symmetry (octant symmetry).	Uses 8-way symmetry (octant symmetry).	May not explicitly use symmetry, usually calculates full circle points.
Initialization	$p = 3 - 2r$ (initial decision parameter).	$p = 1 - r$ (initial decision parameter).	Uses the radius r and calculates $y = \sqrt{r^2 - x^2}$ for each x .
Decision Parameter Update	p is updated based on integer comparisons and additions.	p is updated using additions and subtractions, avoiding multiplications.	Direct computation for each point; no decision parameter.
Accuracy	High accuracy with consistent spacing of points.	High accuracy with consistent spacing of points.	Prone to rounding errors, resulting in gaps or uneven spacing.
Ease of Implementation	Moderate, requires careful management of decision parameter updates.	Moderate, simpler but involves additional conditional checks.	Easy, follows the basic equation of a circle but requires floating-point math.
Use Cases	Best for raster graphics where speed and accuracy are critical.	Suitable for most graphics applications, balances simplicity and efficiency.	Suitable for simple graphics applications or when precision is not critical.
Complexity	$O(r)$ with integer operations.	$O(r)$ with integer operations.	$O(r)$ with floating-point operations (slower).

3. Ellipse Drawing Algorithms

These algorithms plot an ellipse using a similar approach to circle algorithms but account for different radii along the x and y axes.

a) Midpoint Ellipse Algorithm

- Divides the ellipse into two regions: one where the slope is less than 1 and another where it is greater than 1.
- Uses a decision parameter to determine the next point and reflects it across all four quadrants.

Midpoint Ellipse Algorithm

1. Initialize Starting Point:

- Start at $(x, y) = (0, ry)$.
- Calculate the initial decision parameter for Region 1:

$$p_1 = ry^2 - rx^2 \cdot ry + \frac{rx^2}{4}$$

2. Region 1 (Slope > -1):

- Iterate while $2 \cdot ry^2 \cdot x \leq 2 \cdot rx^2 \cdot y$:
 - Plot the points (x, y) and their reflections in all four quadrants:
 - $(x, y), (-x, y), (x, -y), (-x, -y)$
 - If $p_1 < 0$:
 - Increment x : $x = x + 1$
 - Update the decision parameter:

$$p_1 = p_1 + 2 \cdot ry^2 \cdot x + ry^2$$
 - Else:
 - Increment x and decrement y : $x = x + 1, y = y - 1$
 - Update the decision parameter:

$$p_1 = p_1 + 2 \cdot ry^2 \cdot x - 2 \cdot rx^2 \cdot y + ry^2$$

3. Region 2 (Slope ≤ -1):

- Calculate the initial decision parameter for Region 2:

$$p_2 = ry^2 \cdot (x + 0.5)^2 + rx^2 \cdot (y - 1)^2 - rx^2 \cdot ry^2$$
- Iterate while $y \geq 0$:
 - Plot the points (x, y) and their reflections in all four quadrants:
 - $(x, y), (-x, y), (x, -y), (-x, -y)$
 - If $p_2 > 0$:
 - Decrement y : $y = y - 1$
 - Update the decision parameter:

$$p_2 = p_2 - 2 \cdot rx^2 \cdot y + rx^2$$
 - Else:
 - Increment x and decrement y : $x = x + 1, y = y - 1$
 - Update the decision parameter:

$$p_2 = p_2 + 2 \cdot ry^2 \cdot x - 2 \cdot rx^2 \cdot y + rx^2$$

4. Polygon Filling Algorithms

These algorithms are used to fill the interior of polygons.

a) Scan-Line Polygon Fill Algorithm

- For each horizontal scan line, determine intersections with polygon edges.
- Sort intersections and fill the pixels between each pair of intersections.

Scan line Polygon fill Algorithm**1. Sort the Polygon Edges:**

- First, the edges of the polygon are sorted based on their **y-coordinates** (vertical position). This is necessary for processing the scanlines from top to bottom.

2. Create Active Edge Table (AET):

- As the scanline progresses, a set of active edges (edges that intersect the current scanline) is maintained. These edges are stored in an **Active Edge Table (AET)**.
- Each entry in the AET contains information about the current edge (e.g., its starting and ending y-coordinates, slope, etc.).

3. Process Each Scanline:

- For each scanline (y-coordinate), the algorithm does the following:

- **Add New Edges to AET:** Any edges starting at the current scanline are added to the AET.
- **Remove Edges from AET:** Any edges ending at the current scanline are removed from the AET.
- **Sort the AET:** Sort the active edges based on the x-coordinate of the intersection point with the scanline. This ensures that the edges are processed in the correct order from left to right.

4. Fill Pixels Between Intersections:

- Once the active edges are sorted, the algorithm fills pixels between pairs of intersection points on the scanline (i.e., the area inside the polygon).
- For every pair of edges in the AET, the algorithm draws a horizontal line between the two intersection points.

5. Update the AET:

- The algorithm increments the scanline (i.e., move to the next y-coordinate) and updates the AET by adjusting the edge slopes for the next scanline.

b) Flood Fill and Boundary Fill

1. Flood Fill

Flood Fill: Starts at a seed point inside the polygon and recursively fills neighboring pixels with the desired color until a boundary is reached.

Flood fill Algorithm

1. **Start at a given point (seed point):**
 - This point should be inside the area that needs to be filled.
2. **Check the pixel color at the seed point:**
 - Compare the color of the pixel at the seed point with the color that needs to be replaced (the current color).
3. **Fill the region:**
 - If the current pixel color is the same as the target color (or an unfilled region), change the color to the new color.

- Then, recursively or iteratively check the neighboring pixels (based on the type of flood fill you are using: 4-way or 8-way).
4. **Repeat for neighboring pixels:**
 - Repeat the process for all neighboring pixels that have the same original color, until all reachable pixels in the region have been filled.

2. Boundary Fill Algorithm

Boundary Fill: Similar to Flood Fill but stops filling when it encounters a boundary color.

Boundary Fill Algorithm

1. **Start at a given point (seed point):**
 - This point should be inside the region that needs to be filled.
2. **Check the pixel color at the seed point:**
 - If the pixel color is not the same as the boundary color (or already filled), the pixel gets filled with the new color.
3. **Recursively fill the neighboring pixels:**
 - The algorithm then checks the 4 or 8 neighboring pixels (depending on whether you're using 4-connected or 8-connected neighbors).
 - If a neighboring pixel's color is not the boundary color or the fill color, the algorithm will continue to fill that pixel.
4. **Stop when the boundary color is encountered:**
 - The algorithm stops filling when it reaches a pixel with the boundary color.

Types of Boundary Fill:

1. **4-connected Boundary Fill:** This uses 4 neighboring pixels: up, down, left, right.
2. **8-connected Boundary Fill:** This uses 8 neighboring pixels: up, down, left, right, and the four diagonals.

1. Odd-Even Rule

Description:

The **Odd-Even Rule** is a method of determining whether a point is inside a polygon based on how many times a ray starting from that point crosses the edges of the polygon. The rule works as follows:

- A ray is drawn from the point in question in any direction.
- The point is inside the polygon if the ray crosses an odd number of edges.
- The point is outside the polygon if the ray crosses an even number of edges.

Process:

- Start from the given point and extend a ray in any direction.
- Count how many times the ray intersects the edges of the polygon.
- If the number of intersections is odd, the point is inside the polygon.
- If the number of intersections is even, the point is outside the polygon.

This rule works for both convex and concave polygons, and the ray can be drawn in any direction as long as it is consistent.

2. Non-Zero Winding Rule

Description:

The **Non-Zero Winding Rule** determines whether a point is inside a polygon based on the total winding number of the polygon's edges as seen from the point. The winding number is a count of how many times the polygon's edges wind around the point in a counterclockwise or clockwise direction.

Process:

- Traverse each edge of the polygon.
- For each edge, determine whether the point is to the left or right of the edge.
- Keep a running tally of the "winding number" (i.e., the number of times the edges go around the point).
- The winding number is incremented when the edge crosses from right to left, and decremented when the edge crosses from left to right.
- If the total winding number is non-zero, the point is inside the polygon. If it is zero, the point is outside the polygon.

This rule handles more complex polygons, including self-intersecting polygons, and is particularly useful for concave polygons.

3. Scan-Line Algorithm

Description:

The **Scan-Line Algorithm** is an efficient method for filling the area of a polygon by drawing horizontal lines across the entire polygon from top to bottom, scanning from left to right. The idea is to find the intersections of the polygon's edges with each scan line and then fill the pixels between pairs of intersections.

Process:

- Start with the scan line at the topmost point of the polygon.
- For each scan line, find all the intersections of the polygon's edges with the line.
- Sort the intersection points in order from left to right.
- Fill the pixels between each consecutive pair of intersections.
- Move the scan line downwards and repeat the process until the bottom of the polygon is reached.

This method is widely used in graphics hardware and for polygon rasterization.

4. Filling with Curved Boundaries

Filling polygons with curved boundaries (e.g., circles, ellipses) follows similar principles but requires specialized algorithms like **Bresenham's Circle Algorithm**, **Midpoint Circle Algorithm**, and others to handle the curvature efficiently.

The scan-line method can also be extended for curved boundaries by approximating the curves with small linear segments and using the same filling approach for intersections of scan lines with these segments.

5. Boundary Fill Algorithm: 4-Connected and 8-Connected Pixels

The **Boundary Fill Algorithm** can be used to fill an enclosed area by starting at a given seed point and expanding outward until a boundary color is encountered. It uses **4-connected** or **8-connected** pixel checking to determine the neighbors of the current pixel to fill.

4-Connected Pixels:

- A **4-connected pixel** refers to the set of pixels that are adjacent to the current pixel in the four main directions: up, down, left, and right.
- In this case, only those pixels are considered that share an edge with the current pixel.

8-Connected Pixels:

- An **8-connected pixel** includes not only the four main directions (up, down, left, right) but also the diagonal directions (top-left, top-right, bottom-left, bottom-right).
- This allows for a larger region of pixels to be filled in each step, making the fill more comprehensive.

Boundary Fill Algorithm (General Steps):

1. **Start at a seed point** within the enclosed area to be filled.
2. **Check the pixel color:**
 - If it matches the boundary color or the fill color, do nothing (this is the stopping condition).
3. **Fill the current pixel** with the fill color.
4. **Recursively check and fill adjacent pixels** using either 4-connected or 8-connected neighbors:
 - For **4-connected**, check only the pixels above, below, left, and right.
 - For **8-connected**, check all 8 neighboring pixels (including diagonals).
5. **Repeat the process** until all pixels in the area are filled.

6. Convex vs. Concave Polygon Area Filling

The distinction between convex and concave polygons mainly affects the complexity of determining whether a point is inside the polygon, which is critical for many area-filling algorithms.

- **Convex Polygon:** A convex polygon has the property that any line drawn between two points inside the polygon will lie entirely inside the polygon. For convex polygons, the **Odd-Even Rule** and **Non-Zero Winding Rule** both work effectively and are computationally simpler.

- **Concave Polygon:** A concave polygon has at least one interior angle greater than 180 degrees, meaning there are regions of the polygon where the boundary curves inward. This makes determining whether a point is inside the polygon more challenging. For concave polygons, the **Non-Zero Winding Rule** is often preferred, as it can handle such cases more accurately, especially when polygons have self-intersections.

7. Comparison of Different Area Filling Techniques

Technique	Convex Polygon	Concave Polygon	Efficiency	Usage
Odd-Even Rule	Works well for simple polygons	Works well, but can be slower for complex polygons	Moderate	General filling, simple polygons
Non-Zero Winding Rule	Effective for complex convex shapes	Handles complex and self-intersecting polygons	Higher computational cost	Complex shapes, self-intersecting polygons
Scan-Line Algorithm	Efficient for convex polygons	Works for concave polygons as well	High	Polygon rasterization
Boundary Fill (4-connected)	Simple, quick fill	Slower for concave polygons	Low to Moderate	Simple regions
Boundary Fill (8-connected)	Comprehensive fill	Effective for concave polygons	Moderate to High	More complete fill, curved boundaries

Filling Algorithm

Feature	Boundary Fill Algorithm	Scanline Fill Algorithm	Flood Fill Algorithm
Method	Fills from a seed point to the boundary.	Fills between pairs of intersections on horizontal scan lines.	Fills from a seed point to the boundary, similar to boundary fill but generally uses different pixel connectivity.

Feature	Boundary Fill Algorithm	Scanline Fill Algorithm	Flood Fill Algorithm
Type of Connectivity	4-connected or 8-connected	Works with scan lines, usually checks for edge intersections.	4-connected or 8-connected (based on the implementation).
Starting Point	Starts from a seed point inside the area.	Starts from scan lines and finds intersections with edges.	Starts from a seed point inside the area.
Boundary Condition	Stops when it encounters a boundary pixel color.	Fills between intersections on scan lines, stopping at edges.	Stops when the boundary color is encountered.
Efficiency	Less efficient for large regions due to recursion.	Efficient for large and complex polygons (requires sorting of intersections).	Moderate, can be inefficient for large areas due to recursion or excessive checks.
Polygon Type	Works for both convex and concave polygons.	Works well for convex and simple concave polygons.	Works for both convex and concave polygons.
Memory Usage	Can lead to stack overflow with recursion for large areas.	Requires additional memory to store edge intersections for each scan line.	Recursion can lead to memory issues in large regions.
Fill Direction	Fills outward from the seed point.	Fills from left to right along scan lines.	Fills outward from the seed point.
Application	Useful for small or moderately complex polygons.	More efficient for large polygons or rasterization tasks.	Useful for filling smaller areas or regions enclosed by boundaries.

Feature	Boundary Fill Algorithm	Scanline Fill Algorithm	Flood Fill Algorithm
Speed/Complexity	Slower for larger areas due to recursive checking.	Faster for complex polygons due to line-by-line processing.	Slower for large areas, especially when there are many recursive calls.
Handling Curved Boundaries	Can handle curved boundaries if they are approximated as line segments.	Often adapted for rasterized curves by approximating curves as edges.	Can fill curved areas if the boundary color is well-defined.
Implementation Complexity	Relatively simple to implement.	More complex to implement, especially sorting intersections.	Simple to implement.

Homogeneous Coordinates

Homogeneous coordinates extend traditional Cartesian coordinates by adding an extra dimension. For a point $((x, y))$ in 2D Cartesian coordinates, the homogeneous coordinates are $((x, y, w))$, where $(w \neq 0)$. Typically, (w) is set to 1 for simplicity, so $((x, y, 1))$ represents the same point. This system allows for more flexible and unified handling of geometric transformations, including translations, which are not linear in Cartesian coordinates

Homogeneous Matrix

A homogeneous matrix is used to perform transformations in homogeneous coordinates. For 2D transformations, it's a 3×3 matrix, and for 3D transformations, it's a 4×4 matrix. These matrices can represent translations, rotations, scaling, and perspective projections. The key advantage is that multiple transformations can be combined into a single matrix, allowing for efficient computation

Why Use Homogeneous Coordinates and Matrices?

1. **Unified Transformations:** All transformations can be represented as matrix multiplications, simplifying the math involved
2. **Points at Infinity:** They allow for the representation of points at infinity, which is useful in projective geometry
3. **Efficiency:** Combining multiple transformations into a single matrix multiplication is computationally efficient

Feature	Raster Graphics	Vector Graphics
Definition	Made up of pixels (tiny squares of color)	Made up of paths defined by mathematical equations
File Types	JPEG, PNG, GIF, BMP, TIFF	SVG, AI, EPS, PDF
Scalability	Loses quality when scaled up	Can be scaled infinitely without losing quality
Best Use Cases	Photographs, detailed images	Logos, icons, illustrations

Feature	Raster Graphics	Vector Graphics
File Size	Generally larger due to pixel data	Generally smaller due to mathematical data
Editing	More complex and detailed editing possible	Easier to edit shapes and lines
Resolution	Fixed resolution, can become pixelated	Resolution-independent, always sharp
Storage	Takes up more storage space	Takes up less storage space

Yaw

- **Yaw** is the rotation around the vertical axis.
- Imagine turning your head left and right to say "no." That's yaw.

Pitch

- **Pitch** is the rotation around the side-to-side axis.
- Think of nodding your head up and down to say "yes." That's pitch.

Roll

- **Roll** is the rotation around the front-to-back axis.
- Picture tilting your head to touch your ear to your shoulder. That's roll.

Visualizing It

- **Yaw:** Turning left or right.
- **Pitch:** Tilting forward or backward.
- **Roll:** Rotating side to side

1. Modeling Coordinate System (MCS)

- **Definition:** The local coordinate system used to define the shape and geometry of individual objects.

- **Purpose:** Allows designers to create and manipulate objects independently of their position in the overall scene.

2. World Coordinate System (WCS)

- **Definition:** A global coordinate system where all objects are placed and transformed.
- **Purpose:** Provides a common reference frame for all objects in the scene, allowing for positioning and transformations relative to each other

3. Viewing Coordinate System (VCS)

- **Definition:** A coordinate system that defines the viewpoint or camera position.
- **Purpose:** Helps in setting up the view by specifying what part of the world coordinate system will be visible

4. Normalized Viewing Coordinate System (NVCS)

- **Definition:** A standardized coordinate system where coordinates are normalized to a range (usually between 0 and 1).
- **Purpose:** Ensures device-independent representation of the scene, making it easier to map to different display devices

5. Device or Screen Coordinate System (DCS)

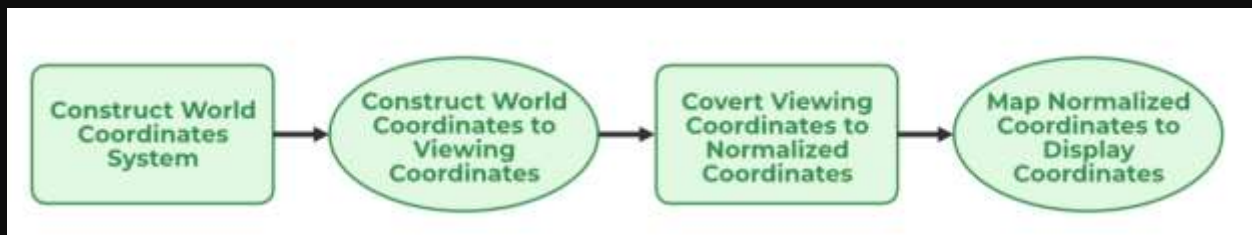
- **Definition:** The coordinate system specific to the output device (e.g., monitor, printer).
- **Purpose:** Converts normalized coordinates to actual pixel coordinates on the screen or device, ensuring the scene is displayed correctly

2D viewing pipeline

The 2D viewing transformation pipeline involves several steps to transform and display a 2D scene on a screen. Here are the main stages:

1. **Modeling Coordinates (MC):** The local coordinate system used to define the shape and geometry of individual objects.
2. **World Coordinates (WC):** A global coordinate system where all objects are placed and transformed.

3. **Viewing Coordinates (VC):** A coordinate system that defines the viewpoint or camera position.
4. **Clipping Coordinates (CC):** Coordinates used to clip the scene to the desired viewing area.
5. **Normalized Viewing Coordinates (NVC):** A standardized coordinate system where coordinates are normalized to a range (usually between 0 and 1).
6. **Device Coordinates (DC):** The coordinate system specific to the output device (e.g., monitor, printer).



- **Window:** A rectangular region in the world coordinate system that defines what part of the scene will be visible. It's essentially the "what we want to see" area.
- **Viewport:** A rectangular area on the display device where the contents of the window will be mapped. It's the "where we want to see" area

Window to viewport

This transformation maps the contents of the window to the viewport. The process involves scaling and translating the coordinates from the window to fit into the viewport. Here's how it works:

Define the Window: Specify the boundaries of the window in world coordinates (e.g., $(x_{wmin}, x_{wmax}, y_{wmin}, y_{wmax})$).

Define the Viewport: Specify the boundaries of the viewport in device coordinates (e.g., $(x_{vmin}, x_{vmax}, y_{vmin}, y_{vmax})$).

Use the formula for wtv scaling and transformation and apply it on actual coordinates

Clipping

Clipping in computer graphics refers to the process of removing parts of objects that lie outside a defined viewing area, such as a window or viewport. Essentially, it ensures that only the visible portions of objects are rendered, improving both performance and visual accuracy

Why is Clipping Needed?

1. **Performance Optimization:** By discarding parts of objects that are not visible, clipping reduces the number of calculations and rendering operations, leading to faster rendering times
2. **Memory Efficiency:** It helps in conserving memory by not storing or processing parts of objects that won't be displayed
3. **Visual Accuracy:** Ensures that only the relevant parts of objects are shown, providing a cleaner and more accurate representation of the scene

Applications of Clipping in Computer Graphics

1. **Real-time Rendering in Video Games:** Clipping improves frame rates by rendering only the visible portions of game objects
2. **CAD and Architectural Design:** Helps in displaying complex 3D models accurately, allowing designers to focus on visible components
3. **Virtual Reality (VR) and Augmented Reality (AR):** Ensures that virtual or augmented objects are rendered realistically within the user's field of view
4. **Zooming Features:** Enables zooming in on specific parts of an image or scene without rendering the entire scene
5. **Drawing Operations:** Used in graphic design software to ensure that only the parts of objects within the drawing area are displayed
6. **Solid Modeling:** Assists in creating and manipulating 3D objects by focusing on visible parts
7. **Text Clipping:** Ensures that text is displayed only within a defined region, commonly used in web design and document processing

Cohen Sutherland line clipping algorithm

1. **Assign Region Codes:** Each endpoint of the line is assigned a 4-bit code (region code) based on its position relative to the clipping window. The bits represent:
 - Bit 1: Above the window
 - Bit 2: Below the window
 - Bit 3: Right of the window
 - Bit 4: Left of the window
2. **Initial Checks:**
 - **Trivially Accept:** If both endpoints have a region code of 0000, the line is completely inside the window and is accepted.
 - **Trivially Reject:** If the logical AND of the region codes of both endpoints is not 0000, the line is completely outside the window and is rejected.
3. **Clipping:**
 - If the line is neither trivially accepted nor rejected, it is partially inside the window. The algorithm then iteratively clips the line against the window boundaries.
 - For each endpoint outside the window, calculate the intersection point with the window boundary and update the endpoint to this intersection point.
 - Reassign the region code for the updated endpoint and repeat the process until the line is either accepted or rejected.

Sutherland Hodgman algorithm

- **Initialize:** Start with the list of vertices of the polygon to be clipped.
- **Process Each Edge of the Clipping Window:**
- For each edge of the clipping window, process all vertices of the polygon.
- Determine the relationship of each vertex to the clipping edge (inside or outside).
- **Generate Output Vertices:**
 - **Case 1:** If both vertices of an edge are inside the clipping window, add the second vertex to the output list.

- Case 2: If the first vertex is inside and the second vertex is outside, add the intersection point of the edge with the clipping window to the output list.
 - Case 3: If the first vertex is outside and the second vertex is inside, add both the intersection point and the second vertex to the output list.
 - Case 4: If both vertices are outside, do nothing.
- Repeat for All Edges: Use the output list from one edge as the input list for the next edge of the clipping window.
- Final Output: After processing all edges of the clipping window, the final output list contains the vertices of the clipped polygon.

Visible Surface Detection

Visible surface detection, also known as hidden surface removal, is a process in computer graphics used to determine which surfaces and parts of surfaces are visible from a particular viewpoint. This is essential for rendering realistic images, as it ensures that only the visible parts of objects are drawn, while hidden parts are not displayed.

Example of Visible Surface Detection

Consider a 3D scene with multiple overlapping objects. Using the Z-Buffer method, the algorithm would:

1. Initialize a depth buffer to store the depth of the closest object at each pixel.
2. For each object, calculate the depth of each pixel it covers.
3. Compare the depth of the current object with the stored depth in the buffer.
4. Update the pixel color and depth buffer if the current object is closer.

Applications of Visible Surface Detection

- **Video Games:** Enhances performance by rendering only visible parts of the scene.
- **CAD Systems:** Provides accurate visualizations of complex models.
- **Virtual Reality:** Ensures realistic rendering of 3D environments.
- **Medical Imaging:** Helps in visualizing internal structures by removing occluded parts.

Object-Space Methods

Definition: Object-space methods determine visibility by comparing objects and parts of objects within the scene. These methods operate in the physical coordinate system of the objects.

Features

- **Geometric Precision:** Operate on the geometric properties of objects, making them resolution-independent.
- **Comparison-Based:** Compare objects and parts of objects to determine visibility.
- **View-Independent:** The visibility determination is independent of the viewpoint.

Advantages

- **High Precision:** Suitable for applications requiring high geometric accuracy.
- **Resolution Independence:** Not affected by the resolution of the display device.
- **Efficient for Small Scenes:** More efficient for scenes with a small number of objects.

Disadvantages

- **Complexity:** Can become computationally expensive for scenes with many objects.
- **Memory Usage:** Requires significant memory to store geometric data.

Types

- **Back-Face Culling:** Removes faces of objects that are facing away from the viewer.
- **BSP Trees (Binary Space Partitioning):** Divides the scene into convex sets by hyperplanes and organizes these sets into a tree structure.
- **Octree:** Divides the 3D space into smaller cubes, making it easier to manage and render complex scenes.

Basic Working of object space

1. **Calculate Geometric Properties:** Compute properties like normals and bounding volumes for each object.
2. **Visibility Determination:** Use geometric comparisons to determine which objects or parts of objects are visible.
3. **Render Objects:** Render the visible parts of objects in the correct order.

Image-Space Methods

Definition: Image-space methods determine visibility at each pixel on the screen. These methods operate in the screen coordinate system.

Features

- **Pixel-Based:** Operate on individual pixels or groups of pixels.
- **View-Dependent:** The visibility determination depends on the viewpoint.
- **Resolution-Dependent:** The accuracy depends on the resolution of the display device.

Advantages

- **Simplicity:** Easier to implement for simple scenes.
- **Direct Rendering:** Directly determines the color of each pixel, making it suitable for raster devices.
- **Handles Complex Scenes:** Can handle complex scenes with many objects.

Disadvantages

- **Computationally Intensive:** Requires significant processing power for high-resolution displays.
- **Memory Usage:** Requires large amounts of memory for depth and frame buffers.

Types

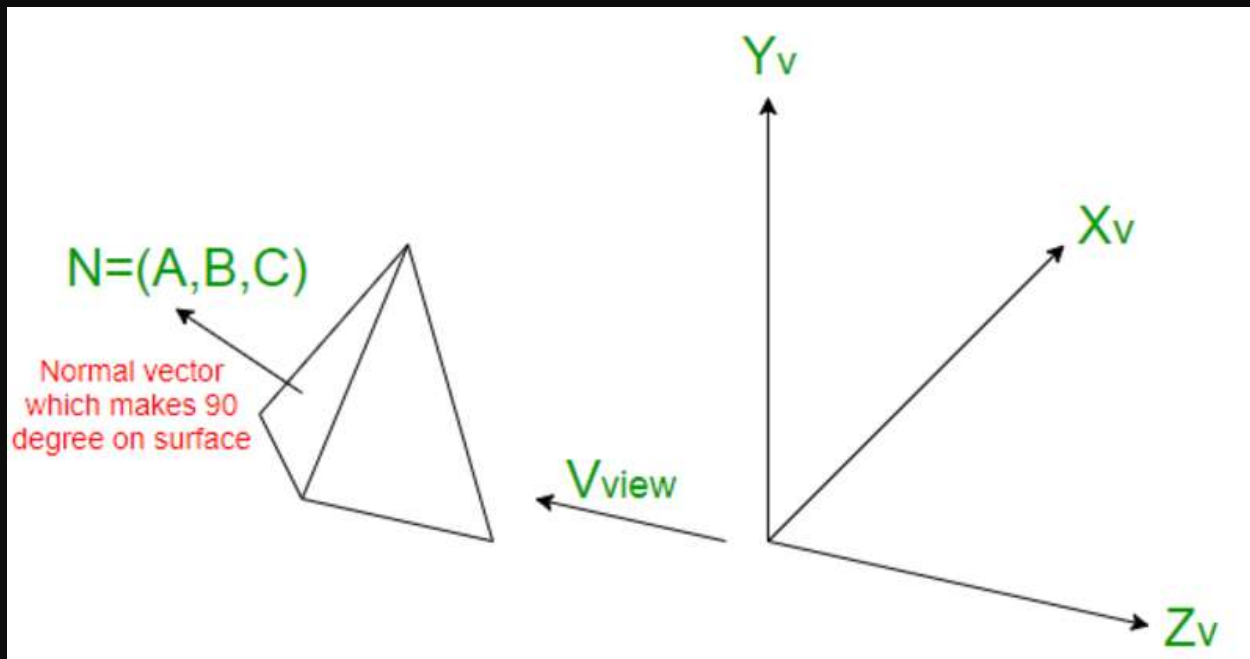
- **Z-Buffer (Depth-Buffer) Method:** Keeps track of the depth of every pixel on the screen and updates the pixel color only if a closer object is found.
- **Scan-Line Method:** Processes each scan line of the image, determining which surfaces intersect the line and which are visible.
- **Ray Casting:** Casts rays from the viewpoint through each pixel to determine the visible surfaces.

Basic working

1. **Initialize Buffers:** Set up buffers to store pixel colors and depths.
2. **Object Projection:** Project objects onto the screen to determine which pixels they cover.
3. **Depth Comparison:** Compare the depth of each object at each pixel to determine visibility.
4. **Update Pixels:** Update the pixel color and depth buffers with the closest object's information.

1. Backface Detection

Definition: Back-face detection is a technique used in computer graphics to determine which faces of a 3D object are not visible from a given viewpoint. These faces are typically those that are oriented away from the viewer and can be culled (not rendered) to improve rendering efficiency.



Algorithm

The back-face detection algorithm works as follows:

1. **Calculate Normal Vectors:** For each polygon in the 3D object, calculate the normal vector. The normal vector is perpendicular to the surface of the polygon.
Equation is $Ax + By + Cz + D = 0$
2. **Determine Visibility:**
 - Compute the dot product of the normal vector of the polygon and the view vector (a vector from the viewer to the polygon).
 - If the dot product is **greater than or equal to zero**, the polygon is facing away from the viewer and is considered a back face.
 - If the dot product is **less than zero**, the polygon is facing towards the viewer and is considered a front face.
3. **Cull Back Faces:** Discard all polygons identified as back faces. Only the front faces are rendered.

Advantages:

- **Performance Improvement:** By culling back faces, the number of polygons that need to be processed and rendered is reduced, leading to faster rendering times

- **Memory Efficiency:** Reduces the amount of memory required to store and process polygons, as only the visible faces are considered

Disadvantages:

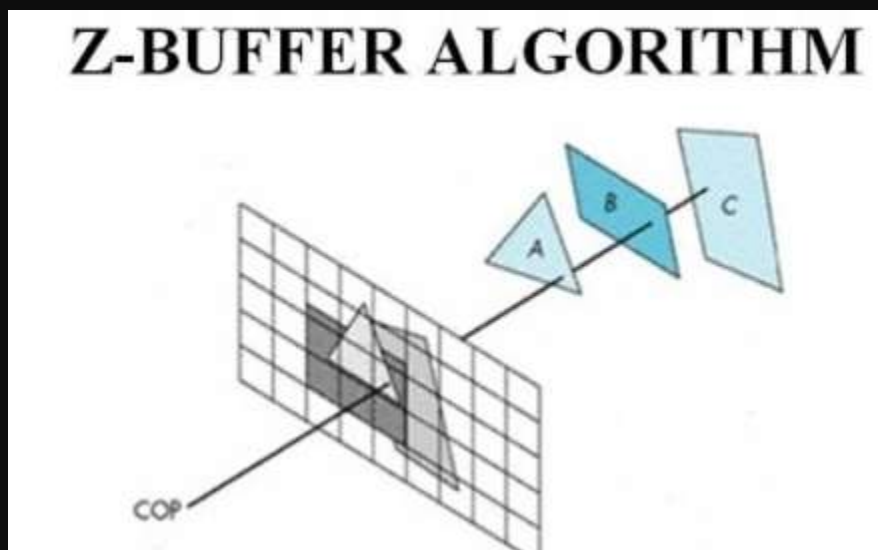
- **Limited to Convex Objects:** This method works best with convex objects. For concave objects, additional techniques may be needed to handle visibility correctly
- **Does Not Handle Inter-Object Occlusion:** Back-face detection only considers individual objects and does not account for occlusion between different objects

Applications

- **Real-Time Rendering:** Widely used in video games and simulations to improve rendering performance.
- **CAD Systems:** Helps in visualizing complex models by removing hidden surfaces.
- **Virtual Reality:** Ensures efficient rendering of 3D environments by culling non-visible surfaces.

Z buffer algorithm

The Z-buffer algorithm, also known as the depth-buffer algorithm, is a technique used in computer graphics to handle hidden surface determination. It ensures that only the visible surfaces of objects are rendered by keeping track of the depth of every pixel on the screen.



Here A is closest so surface intensity of A is saved. Two types of buffer are there: depth buffer and refresh buffer.

$$AX + BY + CZ + D = 0$$

$$Z = \frac{-AX - BY - D}{C}$$

$$Z' = \frac{-A(X + 1) - BY - D}{C}$$

$$Z' = Z - A/C$$

Algorithm

1. Initialization:

- Create a depth buffer (Z-buffer) and a frame buffer. The depth buffer stores the depth (z-value) of the closest object at each pixel, while the frame buffer stores the color information.
- Initialize the depth buffer with a maximum depth value (e.g., 1.0 for normalized coordinates) and the frame buffer with the background color.

2. Processing Each Polygon:

- For each polygon in the scene, determine the pixels it covers when projected onto the screen.
- For each pixel $((x, y))$ covered by the polygon:
 1. **Calculate Depth:** Compute the depth (z) of the polygon at $((x, y))$.
 2. **Depth Comparison:** Compare the calculated depth (z) with the current value in the depth buffer at $((x, y))$.
 3. **Update Buffers:** If (z) is less than the current depth buffer value (indicating the polygon is closer to the viewer), update the depth buffer with (z) and the frame buffer with the polygon's color.

3. Final Image:

- After processing all polygons, the frame buffer contains the color values of the visible surfaces, and the depth buffer contains the depth values of the closest surface.

Pseudocode

- First of all, initialize the depth of each pixel.
- i.e, $d(i, j) = \text{infinite (max length)}$
- Initialize the color value for each pixel
- as $c(i, j) = \text{background color}$
- for each polygon, do the following steps :
 - for (each pixel in polygon's projection)
 - {
 - find depth i.e, z of polygon
 - at (x, y) corresponding to pixel (i, j)
 - if $(z < d(i, j))$
 - {
 - $d(i, j) = z;$
 - $c(i, j) = \text{color};$
 - }
 - }

Example

Consider a scene with multiple overlapping polygons. The Z-buffer algorithm will:

1. Initialize the depth buffer to a maximum value (e.g., 1.0) and the frame buffer to the background color.
2. For each polygon, calculate the depth of each pixel it covers.
3. Compare the depth with the current depth buffer value and update the buffers if the polygon is closer.

Advantages

- **Simplicity:** Easy to implement and understand
- **Handles Complex Scenes:** Can handle scenes with many overlapping objects without sorting them

- **Hardware Support:** Widely supported by graphics hardware, making it efficient for real-time applications

Disadvantages

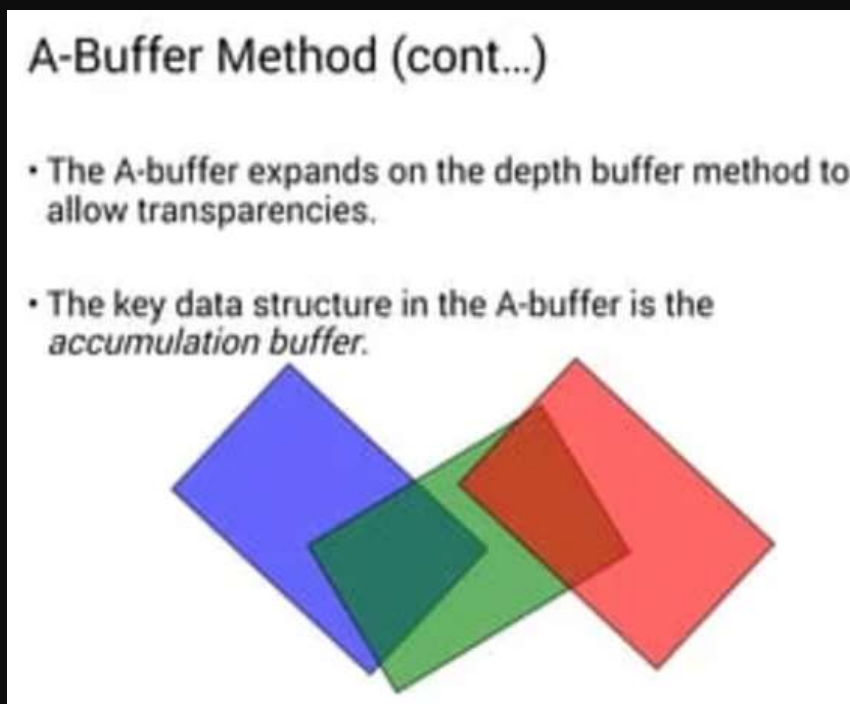
- **Memory Usage:** Requires additional memory for the depth buffer
- **Precision Issues:** Limited precision can lead to artifacts like "z-fighting" when objects are very close to each other

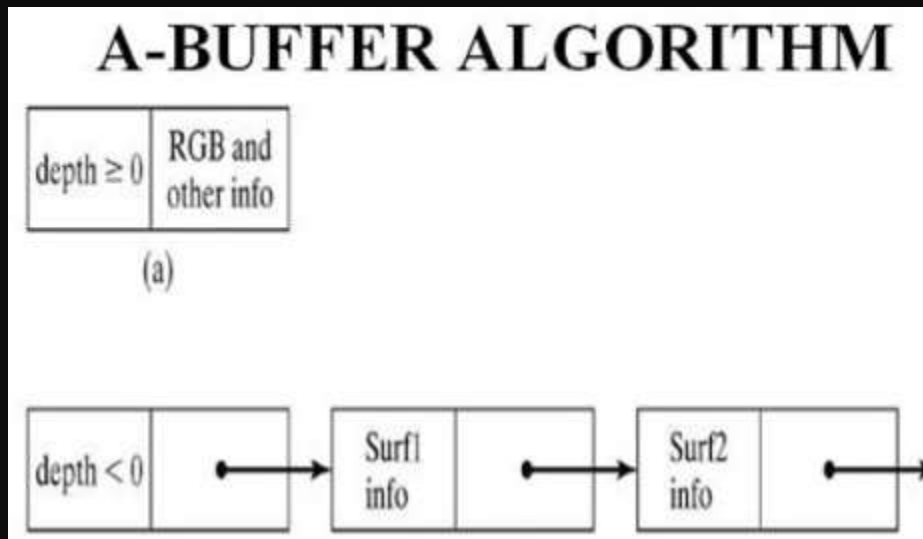
Applications

- **Video Games:** Ensures fast and accurate rendering of 3D scenes.
- **CAD Systems:** Provides accurate visualizations of complex models.
- **Virtual Reality:** Ensures realistic rendering of 3D environments by handling hidden surfaces efficiently.

A Buffer

Definition: The A-buffer, also known as the accumulation buffer, is an advanced hidden surface removal technique used in computer graphics. It extends the Z-buffer method to handle transparency and anti-aliasing, making it suitable for scenes with complex visibility issues, such as overlapping transparent objects





Why linked lists ?

- **Dynamic Storage:** Linked lists allow for flexible and efficient storage of varying numbers of fragments per pixel, which is crucial for handling transparency and anti-aliasing.
- **Efficient Operations:** They enable easy insertion and deletion of fragments, which is important for maintaining the correct order of overlapping transparent surfaces.
- **Handling Multiple Fragments:** Linked lists can store multiple fragments at each pixel, each with its own depth, color, and coverage information, allowing for accurate blending of transparent surfaces.

Algorithm

1. Initialization:

- Create an accumulation buffer to store information about each pixel, including depth, color, and coverage (a bitmask indicating which parts of the pixel are covered by different fragments).

2. Processing Each Polygon:

- For each polygon, determine the pixels it covers when projected onto the screen.
- For each pixel $((x, y))$ covered by the polygon:
 1. **Calculate Depth and Coverage:** Compute the depth (z) and the coverage bitmask for the polygon at $((x, y))$.

2. **Update Accumulation Buffer:** Store the depth, color, and coverage information in the accumulation buffer. If multiple fragments cover the same pixel, combine their coverage bitmasks and average their colors based on coverage.

3. **Final Image:**

- After processing all polygons, the accumulation buffer contains the combined color and depth information for each pixel, taking into account transparency and anti-aliasing.

Features

- **Handles Transparency:** Unlike the Z-buffer, the A-buffer can handle transparent surfaces by combining colors based on coverage
- **Anti-Aliasing:** Provides anti-aliasing by averaging colors of overlapping fragments, resulting in smoother edges
- **Complex Visibility:** Can resolve visibility among intersecting and overlapping objects, including transparent ones

Advantages

- **High Quality:** Produces high-quality images with smooth edges and correct handling of transparency
- **Versatile:** Can handle a wide range of geometric primitives and complex scenes

Disadvantages

- **Memory Usage:** Requires more memory than the Z-buffer due to the additional information stored for each pixel
- **Computational Cost:** More computationally intensive, making it slower than simpler methods like the Z-buffer

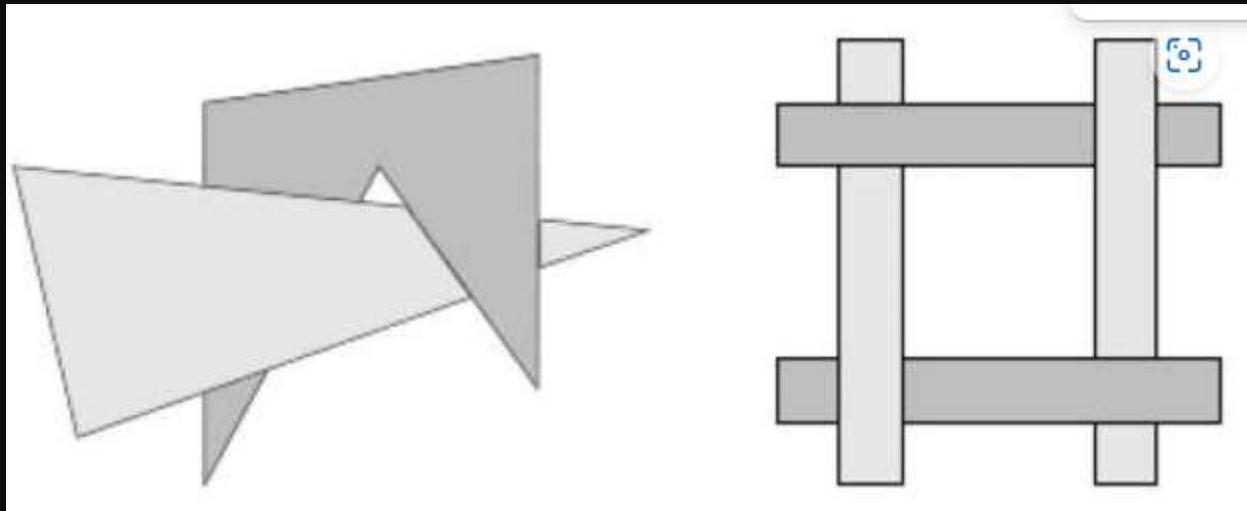
Applications

- **Rendering Systems:** Used in high-quality rendering systems where accurate handling of transparency and anti-aliasing is crucial.
- **Visual Effects:** Suitable for creating complex visual effects in movies and simulations.

- **Scientific Visualization:** Used in applications requiring precise visualization of overlapping transparent structures.

Depth Sorting (Painter's Algorithm)

Definition: Depth sorting, also known as the Painter's Algorithm, is a technique used in computer graphics to determine the visibility of surfaces in a 3D scene. It works by sorting polygons based on their depth from the viewer and rendering them in back-to-front order.



Here image where they are partially or fully obscuring one another alternatively we use priority of object surface low priority gets obscured

Algorithm

1. Sort Polygons by Depth:

- Sort all polygons in the scene based on their depth (z-coordinate) from the viewer. The farthest polygons are rendered first, and the closest polygons are rendered last.

2. Render Polygons:

- Render each polygon in the sorted order. Polygons that are farther away are drawn first, and closer polygons are drawn over them.

Steps in Detail

1. Sort Polygons:

- Calculate the depth of each polygon. This can be done by averaging the z-coordinates of the polygon's vertices.
- Sort the polygons in descending order of their depth values.

2. Render in Order:

- For each polygon in the sorted list, render it to the screen. This ensures that polygons closer to the viewer overwrite those that are farther away.

Example

Consider a scene with three overlapping polygons. The algorithm will:

1. Calculate the depth of each polygon.
2. Sort the polygons by depth.
3. Render the polygons in back-to-front order, ensuring correct visibility.

Advantages

- **Simplicity:** Easy to implement and understand
- **Handles Transparency:** Can handle transparent surfaces by blending colors as polygons are rendered

Disadvantages

- **Overdraw:** May render pixels multiple times, leading to inefficiency
- **Cyclic Overlap:** Struggles with cyclic overlap where polygons overlap in a circular manner, requiring additional handling
- **Piercing Polygons:** Issues with polygons that intersect each other, which may need to be split

Applications

- **Basic Rendering:** Suitable for simple scenes where depth sorting is sufficient for visibility determination.
- **Transparency Handling:** Useful in scenes with transparent objects where blending is required.

All in one link for the previous algorithm

https://www.tutorialspoint.com/computer_graphics/visible_surface_detection.htm

scanline method

The scanline method is an image-space algorithm used in computer graphics for visible surface determination. It processes one horizontal line (scanline) at a time, rather than one pixel or polygon at a time

Algorithm**1. Initialization:**

- Create an edge table (ET) that contains all the edges of the polygons sorted by their minimum y-coordinate.
- Create an active edge list (AEL) that will store edges intersecting the current scanline.

2. Process Each Scanline:

- For each scanline from the top to the bottom of the screen:
 1. **Update AEL:** Add edges from the edge table to the AEL if their minimum y-coordinate matches the current scanline. Remove edges from the AEL if their maximum y-coordinate is less than the current scanline.
 2. **Sort AEL:** Sort the edges in the AEL by their x-coordinates.
 3. **Fill Pixels:** For each pair of edges in the AEL, fill the pixels between them on the current scanline.
 4. **Increment Edges:** Update the x-coordinates of the edges in the AEL based on their slopes.

3. Repeat:

- Repeat the process for each scanline until the entire screen is processed.

Features

- **Row-by-Row Processing:** Processes the image one row at a time, which can be more efficient than pixel-by-pixel processing

- **Edge Coherence:** Utilizes the coherence of edges across scanlines to reduce the number of calculations

Advantages

- **Efficiency:** Reduces the number of comparisons needed by processing edges row-by-row
- **Memory Usage:** Only requires storing edges that intersect the current scanline, reducing memory usage compared to methods that store all polygons

Disadvantages

- **Complexity:** Requires careful management of the edge table and active edge list
- **Handling Intersections:** Can be complex to handle intersections and overlapping polygons

Applications

- **Rendering Systems:** Used in rendering systems to efficiently determine visible surfaces.
- **Raster Graphics:** Suitable for raster graphics where images are processed row-by-row.
- **CAD Systems:** Helps in visualizing complex models by processing one scanline at a time

Pseudocode for scanline

Initialize Edge Table (ET) and Active Edge List (AEL)

For each scanline from top to bottom:

 Update AEL:

- Add edges from ET to AEL if their minimum y-coordinate matches the current scanline
- Remove edges from AEL if their maximum y-coordinate is less than the current scanline

Sort AEL by x-coordinates

For each pair of edges in AEL:

For each pixel between the pair of edges:

Calculate the depth (z-value) of the current polygon at the pixel

If the current polygon is closer than the stored depth in the depth buffer:

Update the depth buffer with the new depth

Update the frame buffer with the polygon's color

Increment x-coordinates of edges in AEL based on their slopes

End For

Refresh Rate

The refresh rate of a display is the number of times per second that the screen is redrawn. It is measured in Hertz (Hz). For example, a refresh rate of 60 Hz means the screen is refreshed 60 times per second

Flickering and Stability:

- **Flickering:** If the refresh rate is too low (typically below 60 Hz), the display may flicker. This flickering can cause eye strain and headaches because the human eye can detect the rapid changes in brightness
- **Stability:** A higher refresh rate (above 60 Hz) generally results in a more stable and smoother display. Modern monitors often have refresh rates of 120 Hz, 144 Hz, or even higher, which reduces flickering and provides a better viewing experience

Electromagnetic Waves and Human Vision

Electromagnetic Spectrum: The electromagnetic spectrum includes all types of electromagnetic radiation, ranging from very long radio waves to very short gamma rays

Visible Light:

- **Range:** The human eye can detect electromagnetic waves in the range of approximately 380 to 700 nanometers. This range is known as visible light and includes all the colors of the rainbow, from red to violet

- **Colors:** Different wavelengths within this range correspond to different colors. Red has the longest wavelength and lowest energy, while violet has the shortest wavelength and highest energy

Invisible Electromagnetic Waves:

- **Radio Waves:** These have the longest wavelengths and are used for communication, such as radio and television broadcasts
- **Microwaves:** Used in microwave ovens and for certain communication technologies
- **Infrared Radiation:** Felt as heat and used in remote controls and thermal imaging
- **Ultraviolet (UV) Light:** Beyond the violet end of the visible spectrum, UV light can cause sunburn and is used in sterilization
- **X-Rays:** Used in medical imaging to view inside the body
- **Gamma Rays:** Have the shortest wavelengths and highest energies, used in cancer treatment and emitted by radioactive materials

Sensitivity of Human Eyes

- **Visible Light Sensitivity:** Human eyes are most sensitive to green light, around 555 nanometers, which is why green appears brighter compared to other colors at the same intensity
- **Invisible Light:** Human eyes cannot detect wavelengths outside the visible spectrum. However, some animals can see ultraviolet or infrared light, which is beyond human capability


Angle deviation from eyes and visual change

1. **Eye Strain and Tears:** When you stare at something for an extended period, your eyes can become strained and may start to water. This can cause the eyes to lose their alignment temporarily.
2. **Angle Deviation:** This misalignment is referred to as angle deviation. It means that the eyes are not perfectly coordinated, causing each eye to see slightly different images.

3. **Visual Perception:** Normally, our brain merges the images from both eyes into a single, coherent picture. However, when angle deviation occurs, the brain might struggle to merge these images correctly.
4. **Multiple Images:** As a result, you might see multiple images or a blurred image. If the deviation is significant, the brain might combine these images into one larger, distorted image.

This phenomenon is similar to what happens when you cross your eyes or when you experience double vision. It's a temporary condition that usually resolves once you rest your eyes.

Wavelength of color



Color	Wavelength (nm)	Frequency (THz)	Photon energy (eV)
violet	380–450	670–790	2.75–3.26
blue	450–485	620–670	2.56–2.75
cyan	485–500	600–620	2.48–2.56
green	500–565	530–600	2.19–2.48
yellow	565–590	510–530	2.10–2.19
orange	590–625	480–510	1.98–2.10
red	625–750	400–480	1.65–1.98

Components of Light

1. **Electromagnetic Waves:** Light is an electromagnetic wave, which means it consists of oscillating electric and magnetic fields that propagate through space.
2. **Photons:** Light can also be described as being made up of particles called photons. These are packets of energy that exhibit both wave-like and particle-like properties

3. **Wavelength and Frequency:** Light has a specific wavelength and frequency, which determine its color and energy. The visible spectrum ranges from about 400 nm (violet) to 700 nm (red)

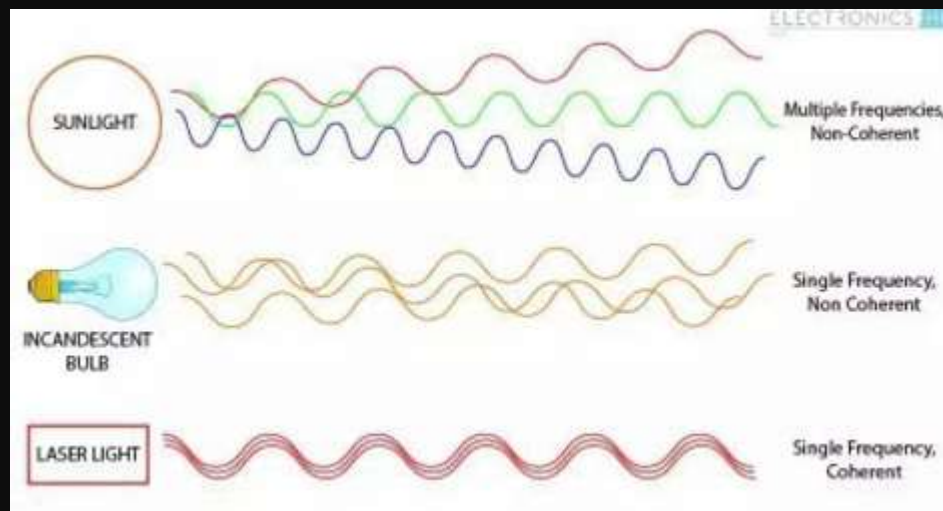
What Light is Made Of

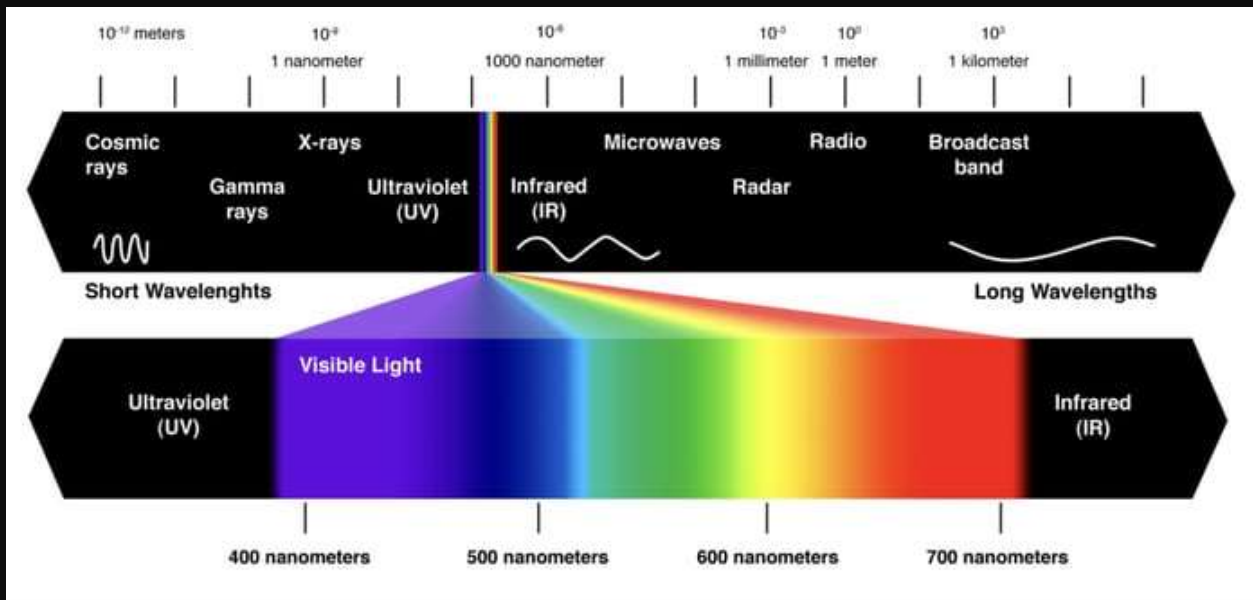
Light is essentially made of photons, which are massless particles that carry energy. These photons travel at the speed of light and can interact with matter in various ways, such as being absorbed, reflected, or refracted

Depiction of Monochromatic Light

Monochromatic light refers to light that has a single wavelength and thus a single color. It can be depicted in several ways:

1. **Single Wavelength:** Monochromatic light is represented by a single, well-defined wavelength. This means it appears as a single color, such as the light from a laser
2. **Waveform:** In diagrams, monochromatic light can be shown as a sine wave with a constant frequency and amplitude
3. **Spectral Line:** In a spectrum, monochromatic light appears as a single line, indicating its specific wavelength





Visible spectrum

Additive Color Model

The additive color model is based on the way light mixes to create colors. It uses three primary colors: red, green, and blue (RGB). When these colors of light are combined in various ways, they produce a wide spectrum of colors. Here's how it works:

- Primary Colors: Red, Green, Blue
- Secondary Colors: When two primary colors are combined, they create secondary colors:
 - Red + Green = Yellow
 - Green + Blue = Cyan
 - Blue + Red = Magenta
- White Light: When all three primary colors are combined in equal intensity, they produce white light.

This model is used in devices that emit light, such as computer monitors, televisions, and cameras

Subtractive Color Model

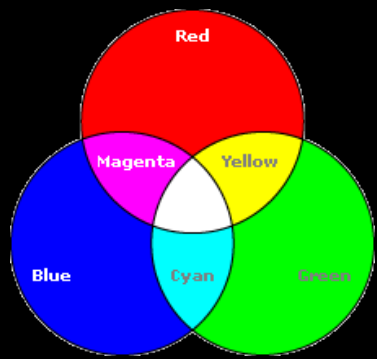
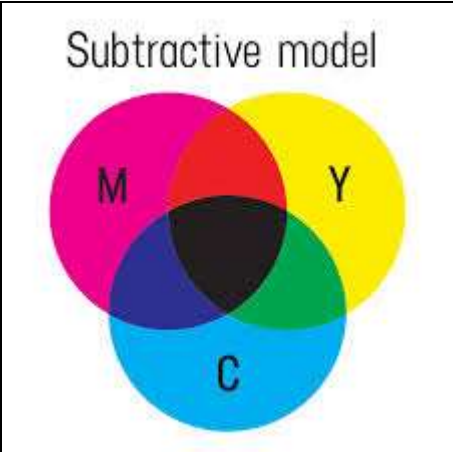
The subtractive color model is used for mixing pigments, dyes, and inks. It works by subtracting (absorbing) certain wavelengths of light and reflecting others. The primary colors in this model are cyan, magenta, and yellow (CMY). Here's how it works:

- Primary Colors: Cyan, Magenta, Yellow

- **Secondary Colors:** When two primary colors are combined, they create secondary colors:
- Cyan + Magenta = Blue
- Magenta + Yellow = Red
- Yellow + Cyan = Green

Black: In practical applications, black (K) is added to the CMY model to create the CMYK model, which is used in color printing. This is because combining cyan, magenta, and yellow inks doesn't produce a perfect black, so black ink is added to achieve deeper blacks and improve contrast.

Feature	Additive Color Model (RGB)	Subtractive Color Model (CMY/CMYK)
Primary Colors	Red, Green, Blue	Cyan, Magenta, Yellow (Black in CMYK)
Secondary Colors	Yellow (Red + Green), Cyan (Green + Blue), Magenta (Blue + Red)	Red (Magenta + Yellow), Green (Yellow + Cyan), Blue (Cyan + Magenta)
Combination Result	Combining all primary colors produces white light	Combining all primary colors ideally produces black
Used In	Devices that emit light (e.g., screens, cameras)	Printing, painting, and other pigment-based media
Color Creation	By adding light of different colors	By subtracting (absorbing) light of different colors
Example	Computer monitors, TVs, stage lighting	Color printing, painting, dyeing fabrics

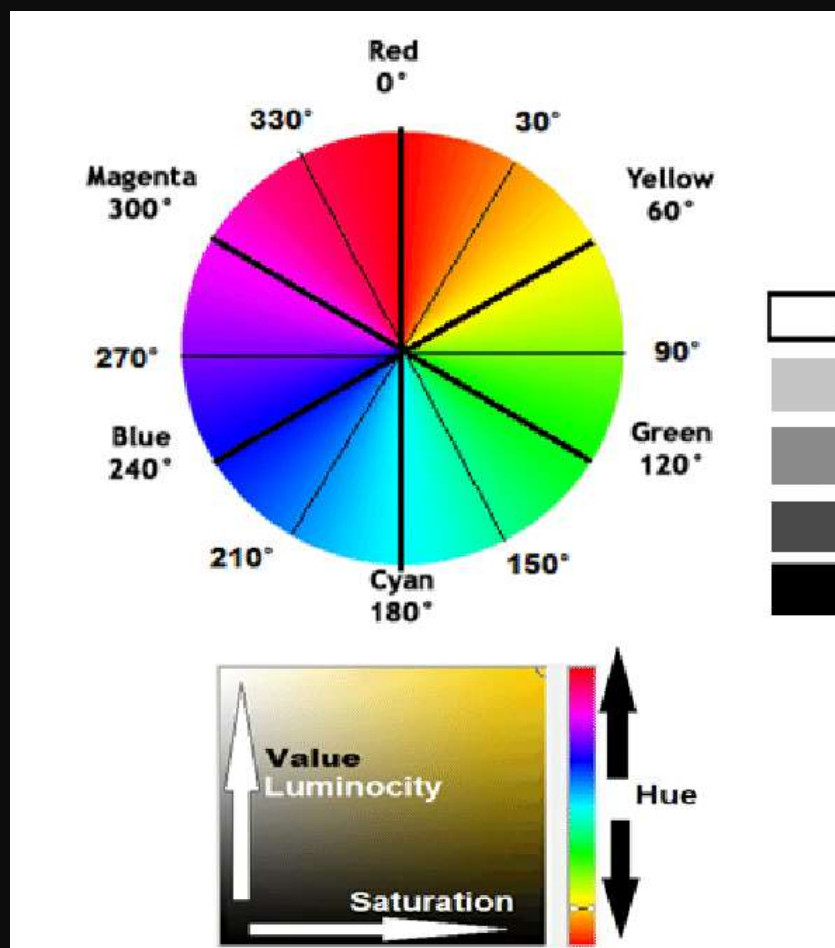
Feature	Additive Color Model (RGB)	Subtractive Color Model (CMY/CMYK)
Venn representation		

HSLvs CMYK vs RGB

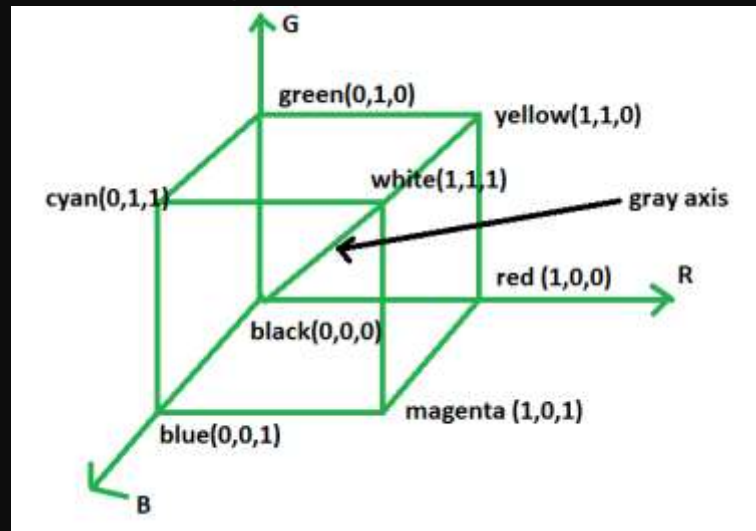
Feature	RGB (Red, Green, Blue)	CMYK (Cyan, Magenta, Yellow, Black)	HSL (Hue, Saturation, Lightness)
Type	Additive	Subtractive	Cylindrical (perceptual)
Primary Colors	Red, Green, Blue	Cyan, Magenta, Yellow (Black in CMYK)	Hue (0°-360°), Saturation (0%-100%), Lightness (0%-100%)
Used In	Digital screens, cameras, lighting	Printing, painting, dyeing	Digital design, color selection tools
Color Creation	By adding light of different colors	By subtracting (absorbing) light of different colors	By adjusting hue, saturation, and lightness
Combination Result	Combining all primary colors produces white light	Combining all primary colors ideally produces black	Adjusting hue changes color, saturation adjusts intensity, lightness adjusts brightness

Feature	RGB (Red, Green, Blue)	CMYK (Cyan, Magenta, Yellow, Black)	HSL (Hue, Saturation, Lightness)
Advantages	Wide color gamut, suitable for digital displays	Accurate color reproduction in printing	Intuitive for human perception, easy to manipulate
Disadvantages	Not suitable for printing, colors may not match print	Limited color gamut compared to RGB	Not as precise for color matching as RGB or CMYK

HSL color wheel



Rgb COLOR CUBE



RGB to HSL conversion

1. Normalize RGB values:

Convert the RGB values (assume R, G, B are in the range 0–255) to the range 0–1:

$$r = \frac{R}{255}, \quad g = \frac{G}{255}, \quad b = \frac{B}{255}$$

2. Find the maximum and minimum values:

$$C_{max} = \max(r, g, b), \quad C_{min} = \min(r, g, b)$$

$$\Delta = C_{max} - C_{min}$$

3. Compute Hue (H):

- If $\Delta = 0$, $H = 0$ (undefined hue, set to 0 by convention).
- Otherwise:

$$H = \begin{cases} 60 \times \left(\frac{g-b}{\Delta} \bmod 6 \right) & \text{if } C_{max} = r \\ 60 \times \left(\frac{b-r}{\Delta} + 2 \right) & \text{if } C_{max} = g \\ 60 \times \left(\frac{r-g}{\Delta} + 4 \right) & \text{if } C_{max} = b \end{cases}$$

- If $H < 0$, add 360 to make it positive.

4. Compute Saturation (S):

$$S = \begin{cases} 0 & \text{if } C_{max} = 0 \\ \frac{\Delta}{C_{max}} & \text{if } C_{max} \neq 0 \end{cases}$$

5. Compute Value (V):

$$V = C_{max}$$

6. Convert back to degrees and percentages (if required):

H in degrees (0–360), S and V are multiplied by 100 to express them as percentages.

HSL to RGB conversion

1. Normalize HSV values:

H is in degrees (0–360), S and V are in percentages (0–100). Normalize:

$$H' = \frac{H}{60}, \quad S' = \frac{S}{100}, \quad V' = \frac{V}{100}$$

Determine the sector:

$i = \lfloor H' \rfloor$ (integer part), $f = H' - i$ (fractional part).

2. Intermediate calculations:

$$p = V' \times (1 - S'), \quad q = V' \times (1 - f \times S'), \quad t = V' \times (1 - (1 - f) \times S')$$

3. Compute RGB values:

Based on the sector i :

$$(r', g', b') = \begin{cases} (V', t, p) & \text{if } i = 0 \\ (q, V', p) & \text{if } i = 1 \\ (p, V', t) & \text{if } i = 2 \\ (p, q, V') & \text{if } i = 3 \\ (t, p, V') & \text{if } i = 4 \\ (V', p, q) & \text{if } i = 5 \end{cases}$$

4. Convert back to the 0–255 range:

$$R = r' \times 255, \quad G = g' \times 255, \quad B = b' \times 255$$

YIQ color model

The **YIQ color model** is primarily used in the analog NTSC color television system, which was widely used in the United States. Here's a breakdown of its components and how it works:

Components of the YIQ Color Model

1. **Y (Luminance):** Represents the brightness or grayscale information of the image. This component is the same as in black-and-white television and is used to ensure compatibility with monochrome TV sets.
2. **I (In-phase):** Represents the chrominance information related to the orange-blue color axis. It carries the color information that the human eye is most sensitive to.
3. **Q (Quadrature):** Represents the chrominance information related to the purple-green color axis. It carries the color information that the human eye is less sensitive to.

How YIQ Works

- **Luminance (Y):** This component is crucial for black-and-white TVs, as it contains all the brightness information. It is calculated as a weighted sum of the RGB components: [$Y = 0.299R + 0.587G + 0.114B$]
- **Chrominance (I and Q):** These components carry the color information. They are derived from the RGB values using the following formulas: [$I = 0.596R - 0.275G - 0.321B$] [$Q = 0.212R - 0.523G + 0.311B$]

Conversion from YIQ to RGB

To convert back from YIQ to RGB, the following formulas are used: [$R = Y + 0.956I + 0.621Q$] [$G = Y - 0.272I - 0.647Q$] [$B = Y - 1.106I + 1.703Q$]

Applications and Advantages

- **Broadcasting:** The YIQ model was designed to optimize the transmission of color information while maintaining compatibility with black-and-white broadcasts. This was particularly important during the transition period when both color and monochrome TVs were in use.
- **Human Vision:** The model takes advantage of the human eye's sensitivity to different colors, allocating more bandwidth to the I component (orange-blue) than the Q component (purple-green), which helps in efficient transmission and better color reproduction

Animation

Animation is the process of creating the illusion of movement by displaying a series of static images or frames in rapid succession. This technique can be applied to drawings, models, or even digital images. The human eye perceives these rapidly changing images as continuous motion due to a phenomenon known as persistence of vision

Types of Animation

1. **Traditional Animation:** Hand-drawn frames on celluloid sheets.
2. **2D Animation:** Digital version of traditional animation, often using vector graphics.
3. **3D Animation:** Creating animated scenes in a three-dimensional space using computer software.
4. **Stop Motion:** Physical objects are moved in small increments between individually photographed frames.
5. **Motion Graphics:** Animated graphic design elements, often used in multimedia projects

12 principles of animation

1. Squash and Stretch

Definition: This principle gives objects a sense of weight and flexibility. When an object moves, it stretches or compresses based on its action. For example, a bouncing ball squashes when it hits the ground and stretches as it ascends.

Example:

A water balloon squashes when it lands on the ground but stretches when thrown in the air.

2. Anticipation

Definition: Preparation for an action. Anticipation gives the audience a clue about what's coming next. This enhances realism and builds interest.

Example:

Before a character jumps, they bend their knees and move downward slightly.

3. Staging

Definition: The presentation of an idea clearly so the audience understands the scene. This involves placement of characters, actions, and camera angles.

Example:

A suspenseful moment might focus the camera on a character's hand reaching for a doorknob, leaving the rest of the frame blurry.

4. Straight Ahead Action and Pose-to-Pose

Definition:

- **Straight Ahead:** Animating frame by frame from start to finish for fluid motion.
- **Pose-to-Pose:** Creating keyframes first, then filling in intermediate frames for control.

Example:

Straight ahead: A fire's flicker.

Pose-to-pose: A gymnast performing a flip.

5. Follow Through and Overlapping Action

Definition: When a character stops, different parts of their body continue to move to show inertia. Overlapping action means that not all parts of a character move simultaneously.

Example:

When a dog runs and stops, its tail keeps moving briefly.

6. Slow In and Slow Out

Definition: Movements start slow, accelerate, and then decelerate before stopping to create realism.

Example:

A car gradually accelerates before cruising and decelerates when braking.

7. Arc

Definition: Most natural movements follow an arc rather than straight lines, giving animations a lifelike feel.

Example:

A pendulum swinging follows an arc.

8. Secondary Action

Definition: Supporting actions that emphasize the main action to enrich the scene.

Example:

A character walking while swinging their arms and whistling.

9. Timing

Definition: The number of frames used in a motion determines the speed, weight, and emotion of the animation.

Example:

A heavy box takes more frames to lift than a feather.

10. Exaggeration

Definition: Exaggerating features or actions for comedic or dramatic effect while keeping believability.

Example:

A character's jaw dropping to the floor in surprise.

11. Solid Drawing

Definition: Ensuring that characters have proper volume, weight, and balance in their designs, even in motion.

Example:

A character walking with proper perspective and proportions.

12. Appeal

Definition: Characters and objects should be visually interesting and easy to relate to, even villains.

Example:

Mickey Mouse's rounded features make him appealing and friendly.

Types of animation

1. Traditional Animation (2D Animation)

- **Description:** The original style of animation, where each frame is drawn by hand on paper or cells, and then photographed and displayed in sequence.
- **Method:** Artists draw each movement frame by frame.
- **Examples:** Classic Disney movies like *Snow White and the Seven Dwarfs* and *The Lion King (1994)*.
- **Use:** Television, movies, and artistic projects.

2. 2D Vector-Based Animation

- **Description:** A digital form of traditional animation created using vector graphics. Instead of redrawing frames, software allows manipulation of parts of characters.
- **Tools:** Adobe Animate, Toon Boom Harmony.
- **Examples:** *Rick and Morty*, *Adventure Time*.
- **Use:** Web animations, advertisements, and TV shows.

3. 3D Animation (CGI)

- **Description:** Computer-generated imagery (CGI) creates three-dimensional objects and environments, allowing realistic movement and depth.
- **Method:** Artists create models in 3D space, then animate them using rigs and motion paths.
- **Examples:** *Toy Story*, *Frozen*, *Avatar*.

- **Use:** Movies, video games, simulations, and VR.

4. Motion Graphics

- **Description:** Animated graphic designs with text as the main component. It's used to convey ideas or messages.
- **Tools:** After Effects, Cinema 4D.
- **Examples:** Title sequences, explainer videos, advertisements.
- **Use:** Marketing, branding, and educational videos.

5. Stop Motion Animation

- **Description:** Objects are physically manipulated in small increments and photographed frame by frame.
- **Types:**
 - **Claymation:** Using clay (e.g., *Wallace and Gromit*).
 - **Puppetry:** Using puppets (e.g., *The Nightmare Before Christmas*).
 - **Object Animation:** Using everyday objects.
- **Examples:** *Coraline*, *Shaun the Sheep*.
- **Use:** Films, artistic expressions, and advertisements.

6. Rotoscope Animation

- **Description:** Tracing over live-action footage frame by frame to create realistic movement.
- **Method:** Live-action footage is recorded, and each frame is traced or digitally rendered.
- **Examples:** *A Scanner Darkly*.
- **Use:** Stylized movies, commercials, and special effects.

7. Mechanical Animation

- **Description:** Animating technical and mechanical parts for industrial or engineering purposes.
- **Method:** Often created using CAD software to simulate real-world operations.
- **Examples:** Animated car engines or machinery demonstrations.
- **Use:** Engineering presentations and instructional vide

Vector art vs bitmap art

Feature	Vector Art	Bitmap Art
Composi tion	Made up of mathematical equation s and geometric shapes	Made up of pixels (tiny dots)
Resolut ion	Resolution-independent (scalable without loss of quality)	Resolution-dependent (quality decreases when scaled)
File Si ze	Generally smaller file size	Generally larger file size
Scalabi lity	Infinitely scalable without losi ng quality	Not scalable without losing qu ality (pixelation)
Editing	Easy to edit and modify	Difficult to edit without losi ng quality
Complex ity	Best suited for simple graphics and illustrations	Can handle complex images with fine details
Color S upport	Supports limited colors	Supports millions of colors
File Fo rmats	SVG, AI, EPS, PDF	JPEG, PNG, GIF, BMP
Image Q uality	High-quality images with smooth curves and lines	May suffer from pixelation or loss of quality

Feature	Vector Art	Bitmap Art
Usage	Ideal for logos, icons, and illustrations	Commonly used for photographs and complex images

Frame by frame animation

In frame-by-frame animation, every frame of an animation sequence is individually created or drawn. This allows for complete control over the movement and look of the animation but can be time-consuming.

How it works:

- Each frame is a unique drawing or adjustment of an object.
- When played in sequence, the frames create the illusion of movement.

Uses:

- Complex, fluid movements like in traditional hand-drawn animation.
- Situations where subtle details or unique motions are required.

Example:

A character running, where every step and arm swing is drawn manually.

Tweening

Definition:

Tweening (short for "in-betweening") is the process of generating intermediate frames between two keyframes to create smooth transitions. Modern software automates this process, saving time and effort compared to frame-by-frame animation.

1. Shape Tweens

Shape tweens are used to morph one shape into another over time. This works with vector shapes (not symbols) and is common for abstract animations or transitions.

How it works:

- You define the start and end shapes (keyframes).

- The software calculates and generates intermediate frames to transform the shape.

Uses:

- Transforming a circle into a star.
- Morphing one object into another (e.g., a logo animation).

Example:

A cloud shape morphing into a sun.

Tools:

Adobe Animate, Blender, and similar software.

2. Motion Tweens

Motion tweens are used to move an object (symbol) along a path from one position to another. It animates properties like position, rotation, scale, and transparency over time.

How it works:

- Create a motion path for the object to follow.
- Define keyframes for start and end positions, and the software generates the movement.

Uses:

- Moving a car from left to right across the screen.
- Animating a bouncing ball or a flying bird.

Example:

A logo sliding into the frame or a character moving across a scene.

Tools:

Adobe Animate, After Effects.

Frame-by-Frame Animation vs Tweens

Aspect	Frame-by-Frame	Tweens
Flexibility	Total control over every frame	Limited to transformations
Time Consumption	Very time-consuming	Time-efficient
Usage	Complex and unique movements	Repeated, smooth transformations
Automation	No automation, fully manual	Automated by software

Virtual Reality

Virtual Reality (VR) is a computer-generated simulation that immerses the user in a fully digital, 3D environment. By using VR headsets or devices, users experience the illusion of being in a completely different world, disconnected from their physical surroundings.

Key Features:

- **Immersion:** The user is entirely enveloped in a digital environment.
- **Interaction:** Users can interact with the virtual environment using controllers, gloves, or motion tracking.
- **Hardware:** Requires VR headsets (e.g., Oculus Rift, HTC Vive), sensors, and sometimes additional equipment like haptic gloves.

Applications:

1. **Gaming:** Immersive video games where players can interact within a virtual world.
2. **Training:** Flight simulations for pilots, medical surgery training, or military exercises.
3. **Education:** Virtual field trips to historical locations or space exploration.
4. **Therapy:** Treating phobias or PTSD through controlled simulations.

VR vs AR

Aspect	Virtual Reality (VR)	Augmented Reality (AR)
Definition	Fully immersive, computer-generated environment.	Enhances the real world with digital overlays.
Environment	Entirely virtual, disconnects users from reality.	Combines virtual elements with the physical world.
Hardware	VR headsets (e.g., Oculus, HTC Vive) and controllers.	AR-enabled devices like smartphones or AR glasses.
Interaction	Interaction is within the virtual world.	Interaction occurs in the real world with added data.
Immersion	Completely immersive; user cannot see the real world.	Partially immersive; user remains aware of reality.
Applications	Gaming, simulations, training, and therapy.	Gaming, retail, architecture, and navigation.
Examples	Playing <i>Beat Saber</i> in VR.	Using AR to see furniture in your room (IKEA app).
Purpose	Simulates an entirely new reality.	Enhances real-world experiences.

Elements of VR

Immersion:

- The degree to which the user feels "present" in the virtual environment.
- Achieved through visual, auditory, and sometimes haptic feedback that replaces real-world sensory input.

Interaction:

- The ability of users to interact with the virtual environment using controllers, gloves, or gestures.
- Interaction makes VR dynamic and engaging, simulating real-world responses.

Sensory Feedback:

- VR incorporates sensory inputs (e.g., vision, hearing, and touch) to make experiences realistic.
- Advanced VR systems include haptic feedback for tactile sensations.

Virtual Environment:

- A computer-generated 3D environment designed to simulate real or imaginary scenarios.
- These environments are often interactive and dynamic.

Presence:

- A psychological state where users feel like they are "inside" the virtual world.
- Strong immersion and interaction contribute to a greater sense of presence.

Types of VR

a. Non-Immersive VR:

- **Description:** Interacts with a virtual environment through a computer screen, but the real world remains dominant.
- **Examples:** Video games or simulations on desktop monitors.
- **Use Case:** Simulations in education or training.

b. Semi-Immersive VR:

- **Description:** Provides a partial sense of immersion using large screens or projection systems. The user remains aware of the real world.
- **Examples:** Flight simulators or interactive museum exhibits.
- **Use Case:** Training pilots, interactive exhibits.

c. Fully Immersive VR:

- **Description:** Offers complete immersion using VR headsets, haptic feedback, and 3D environments.
- **Examples:** Games like *Beat Saber*, simulations for surgery, or military training.
- **Use Case:** Gaming, high-stakes training scenarios, therapy.

d. Augmented Virtuality:

- **Description:** Merges real-world elements into a predominantly virtual environment.

- **Examples:** Virtual worlds with live video feeds.
- **Use Case:** Architecture or real-time collaboration.

Components of VR

1. Hardware Components

a. Head-Mounted Display (HMD):

- Displays the virtual environment to the user, typically with two lenses for a stereoscopic view.
- Examples: Oculus Quest, HTC Vive, PlayStation VR.

b. Motion Tracking Sensors:

- Track the user's head, body, and hand movements to synchronize the virtual environment accordingly.
- Types of tracking:
 - **Positional tracking** (e.g., cameras and infrared sensors).
 - **Orientation tracking** (e.g., gyroscopes, accelerometers).

c. Controllers:

- Handheld devices for interaction within the virtual environment (e.g., Oculus Touch, Vive Controllers).

d. Haptic Feedback Devices:

- Provide tactile sensations, such as vibrations or resistance, to simulate touch.
- Examples: Haptic gloves, vests.

e. Audio Devices:

- Deliver 3D spatial sound for enhanced immersion. Often integrated into the HMD.

f. Input Devices:

- Include joysticks, keyboards, gesture recognition systems, or voice commands.

g. Computing System:

- High-performance PCs, consoles, or standalone devices to process and render virtual environments.
- Requirements: GPUs, CPUs, and sufficient memory for smooth performance.

2. Software Components**a. VR Content:**

- The 3D virtual environment, scenarios, or simulations created using specialized software or engines.

b. Simulation Engine:

- Software like Unity or Unreal Engine is used to develop interactive VR content.

c. Middleware:

- Facilitates communication between hardware and software for smooth functionality.

Working of VR headset**a. Display of Virtual Environment:**

- The HMD projects two slightly different images to each eye.
- This creates a stereoscopic effect, giving depth perception to the virtual world.

b. Head Tracking:

- Sensors like gyroscopes, accelerometers, and magnetometers in the headset track head orientation and movements.
- Example: Turning your head in real life causes the virtual world to shift accordingly.

c. Position Tracking:

- External sensors (e.g., infrared cameras or laser systems) or built-in cameras track the user's physical position in a room.
- Example: Walking forward in the real world moves your virtual avatar forward.

d. Lens Adjustment:

- Lenses inside the headset focus and warp the images to match how your eyes naturally perceive the world.
- Adjustable lenses allow users to calibrate for comfort and clarity.

e. Field of View (FOV):

- Most VR headsets offer a wide FOV, typically between 90°-120°, to mimic peripheral vision.

f. Interactivity:

- Handheld controllers or motion-tracking gloves allow the user to interact with objects in the virtual space.
- Example: Picking up a virtual sword using a motion controller.

g. Audio Feedback:

- Spatial audio systems simulate sound from different directions to match the environment.
- Example: Hearing footsteps behind you feels realistic.

h. Rendering and Latency:

- The computing system renders the virtual environment at high frame rates (typically 90 FPS or higher) to prevent motion sickness.
- Low latency ensures real-time responsiveness between movements and visual updates