<u>COMPUTER GRAPHICS</u>

<u>UNIT-I</u>

<u>INTRODUCTION TO COMPUTER GRAPHICS:</u>

Computer graphics is a field of computer science and technology that deals with the creation, manipulation, and representation of visual images and animations using computers. It involves the use of algorithms, mathematics, and computer hardware to generate and display visual content. Computer graphics has a wide range of applications in various industries, including entertainment, design, science, engineering, and more.

<u>Here are some key components and concepts within computer graphics:</u>

1. Graphics Hardware: Graphics hardware includes components like graphics cards (GPUs), monitors, and input devices (such as mice and tablets). GPUs are specially designed to accelerate graphics rendering and are essential for real-time rendering and complex graphical tasks.

2. Graphics Software: Graphics software encompasses the programs and algorithms used to create, manipulate, and render images. This includes 2D and 3D modeling software, rendering engines, and graphics libraries like OpenGL and DirectX.

3. Rendering: Rendering is the process of generating a visual representation from a digital model. It involves calculating how light interacts with surfaces in a 3D scene to produce a 2D image. Rendering techniques can be categorized as real-time (for video games and simulations) or offline (for movies and pre-rendered animations).

4. Computer-Generated Imagery (CGI): CGI is the use of computer graphics to create visual elements in film, television, video games, and other media. It includes 3D modeling, texturing, animation, and rendering to produce realistic or stylized visual effects.

5. Interactive Graphics: Interactive graphics allow users to manipulate and interact with visual content in real-time. Examples include video games, simulations, and user interfaces.

APPLICATIONS OF COMPUTER GRAPHICS:

Computer graphics find applications in numerous fields, enhancing productivity, creativity, and communication.

Here's an overview of key areas where computer graphics are extensively used:

1. Entertainment and Media:

- Video Games: Computer graphics play a fundamental role in creating immersive 3D worlds and characters in video games.
- Movies and Animation: CGI is used extensively in film and animation production, allowing for breathtaking visual effects and animated storytelling.
- Virtual Reality (VR) and Augmented Reality (AR): Graphics are essential in creating realistic VR and AR experiences.

2. Design and Visualization:

- Architectural Visualization: Architects and designers use 3D graphics to create virtual representations of buildings and interiors.
- Product Design: Computer-aided design (CAD) software enables engineers and designers to model and simulate products before physical prototypes are built.
- Data Visualization: Graphics help convey complex data in an understandable and visually appealing manner.

3. Science and Medicine:

- Medical Imaging: Computer graphics assist in the visualization and analysis of medical scans like MRI and CT scans.
- Scientific Simulation: Researchers use computer graphics for simulating physical phenomena, weather patterns, and more.

4. Education and Training:

- Educational Software: Graphics are used in educational software to make learning more engaging and interactive.
- Flight and Military Simulators: Pilots and military personnel use simulators that rely heavily on computer graphics for training.

5. Web and User Interfaces:

- Web Design: Web developers use graphics to create visually appealing websites and user interfaces.

- User Experience (UX) Design: Graphics enhance the usability and aesthetics of software interfaces.

6. Art and Visualization:

- Digital Art: Artists create digital paintings, sculptures, and animations using graphics software.

- Data Art: Artists use data visualization techniques to create artistic representations of information.

7. Gaming and Simulation:

- Video Games: Graphics are the core of video game development, providing visual realism and immersion.

- Training Simulators: Simulators are used for training in aviation, military, healthcare, and more.

8. Advertising and Marketing:

- Advertisement Creation: Graphics are used to design advertising materials, including banners, posters, and promotional videos.

- Product Visualization: Graphics help showcase products in marketing materials and e-commerce.

9. Geographic Information Systems (GIS): GIS uses graphics to map and analyze geographical data, aiding in urban planning, environmental management, and navigation.

NON-INTERACTIVE GRAPHICS:

Non-interactive graphics refer to visual content or images that are static and do not respond to user input or interactions in real-time. These graphics are pre-rendered or fixed, meaning they remain unchanged unless explicitly modified by an external process.

Here are some key characteristics and examples of non-interactive graphics:

1. Static Nature: Non-interactive graphics are typically static images, illustrations, or visual representations that do not change unless a new version of the graphic is created.

2. No User Control: Users cannot manipulate or interact with non-interactive graphics. They are passive viewers of the visual content.

3. Common Examples:

- Photographs: Digital or analog photographs captured by cameras are non-interactive images. They depict a scene or subject as it appeared at a particular moment.
- Icons and Logos: Icons and logos used in branding and user interfaces are usually non-interactive. They serve as visual identifiers and do not respond to user interactions.
- Illustrations: Static illustrations, diagrams, and charts used in books, reports, or presentations fall under this category.

4. Creation Process: Non-interactive graphics are often created using traditional graphic design tools such as Adobe Photoshop or Illustrator. They can also be generated using data visualization software to create static charts and graphs.

5. Use Cases: Non-interactive graphics are commonly used for visual communication, documentation, branding, and any scenario where the visual content does not need to change based on user input.

INTERACTIVE GRAPHICS:

Interactive graphics, on the other hand, refer to visual content that responds to user input or interactions in real-time. These graphics allow users to actively engage with and manipulate the visual elements.

Here are key characteristics and examples of interactive graphics:

1. Real-Time Interaction: Interactive graphics respond to user actions or input immediately, enabling users to control and modify visual content on-the-fly.

2. User Engagement: Users can interact with elements within the graphic, such as clicking, dragging, zooming, or manipulating objects.

3. Common Examples:

- Video Games: Interactive graphics are a fundamental component of video games. Players can control characters, objects, and the game environment in real-time.

- Web Applications: Many web applications use interactive graphics for features like maps (e.g., Google Maps), data visualization (e.g., interactive charts), and user interfaces with buttons and sliders.

- Simulations: Interactive simulations, such as flight simulators or medical training simulations, respond to user actions and provide a dynamic learning experience.

- Virtual Reality (VR): VR environments are highly interactive, allowing users to explore and interact with virtual spaces and objects.

4. Creation Process: Interactive graphics are created using specialized software and programming languages that enable interactivity. Game engines like Unity or graphics libraries like WebGL are often used to develop interactive graphics.

5. Use Cases: Interactive graphics are employed in various fields for training, education, entertainment, user interfaces, and simulations, where user engagement and real-time interaction are essential.

CONCEPTUAL FRAMEWORK FOR INTERACTIVE GRAPHICS:

A conceptual framework for interactive graphics refers to a structured approach or set of principles that guide the design, development, and understanding of interactive graphical systems and applications. This framework provides a foundation for creating effective and user-friendly interactive graphics by addressing key concepts and considerations.

Components and elements of a conceptual framework for interactive graphics in detail:

1. User-Centered Design:

- User Needs and Goals: Understanding the users' needs, goals, and expectations is paramount. This involves conducting user research and user testing to identify what interactions users are looking for and what their objectives are.

- User Feedback: Incorporating feedback from users during the design and development process ensures that interactive graphics align with user preferences and usability requirements.

2. Interactivity Levels:

- Degrees of Interactivity: Define the level of interactivity needed for the application. Some applications may require simple interactions like clicking buttons, while others demand complex interactions like 3D object manipulation or immersive virtual reality experiences.

- Feedback Mechanisms: Establish how the system provides feedback to users during interactions, such as visual cues, sounds, or haptic feedback (e.g., vibration in a controller).

3. Data Visualization and Representation:

- Data Mapping: Determine how data is mapped to visual elements. This involves choosing appropriate visualization techniques and encoding data into graphical representations like charts, graphs, or maps.

- Real-Time Data: If applicable, address the real-time nature of data updates and how they are reflected in the graphics.

4. User Interface Design:

- Interface Elements: Design user interface elements (buttons, sliders, menus) to facilitate interactions. Ensure they are visually consistent, easily accessible, and responsive to user actions.

- Navigation: Plan the navigation structure, enabling users to move seamlessly between different interactive components or views.

5. User Feedback and Guidance:

- Instructions: Provide clear and concise instructions or tooltips to guide users on how to interact with the graphics.

- Error Handling: Design error messages and recovery mechanisms for when users make mistakes or encounter issues.

6. Performance Optimization:

- Hardware Considerations: Account for the capabilities of the target hardware (e.g., devices, GPUs) and optimize graphics and interactions for smooth performance.

- Efficiency: Implement efficient algorithms and data structures for rendering and handling interactions, particularly in resource-intensive applications.

7. Accessibility and Inclusivity:

- Accessibility Features: Ensure that interactive graphics are accessible to individuals with disabilities. This may involve providing alternative text for non-text elements, keyboard navigation, and compatibility with screen readers.

- Inclusivity: Consider cultural, linguistic, and demographic factors to create graphics that are inclusive and relevant to diverse user groups.

8. Cross-Platform and Cross-Device Compatibility:

- Responsive Design: Ensure that interactive graphics work seamlessly on various devices and screen sizes, including desktops, mobile devices, and tablets.

- Cross-Browser Compatibility: Test and optimize the graphics for different web browsers and platforms.

9. Data Security and Privacy:

- Data Handling: Address data security and privacy concerns, especially when interactive graphics involve sensitive or personal information.

- Compliance: Ensure compliance with relevant data protection regulations and standards.

10. Usability Testing and Evaluation:

- Usability Testing: Conduct usability testing to gather user feedback and identify areas for improvement in terms of interactivity, user interface, and overall user experience.

- Iterative Design: Use feedback and evaluation results to make iterative improvements to the interactive graphics.

11. Documentation and Training:

- User Documentation: Provide comprehensive user documentation or help resources to assist users in understanding how to interact with the graphics.

- Training Materials: Create training materials or tutorials, especially for complex interactive systems.

12. Scalability and Maintenance:

- Scalability: Plan for scalability, particularly in applications where the volume of data or complexity of interactions may grow over time.

- Maintenance: Develop strategies for maintaining and updating interactive graphics to keep them relevant and functional.

RASTER SCAN DISPLAY:

A raster scan display, also known as a raster display or raster graphics, is a type of computer graphics display technology that creates images by scanning the screen's surface sequentially, one pixel (picture element) at a time. This scanning process occurs in a systematic, left-to-right and top-to-bottom manner, forming a grid of pixels.

Here are key details about raster scan displays:

1. Pixel-Based: Raster displays are pixel-based, which means that the screen is composed of a grid of discrete picture elements, each representing a single point of color or brightness. The arrangement and density of pixels determine the display's resolution and quality.

2. Cathode Ray Tube (CRT): Historically, most traditional computer monitors, televisions, and older displays used CRT technology for raster scanning. In a CRT monitor, an electron beam is used to illuminate phosphor dots on the screen, creating the image.

3. Resolution: The resolution of a raster display is defined by the number of pixels it can display both horizontally and vertically. For example, 1920x1080 resolution indicates 1920 pixels in width and 1080 pixels in height.

4. Color Depth: Raster displays can support various color depths, from monochrome (black and white) to millions of colors, depending on the display technology and graphics hardware.

5. Refresh Rate: Raster displays refresh the entire screen multiple times per second to maintain the image. The refresh rate is measured in Hertz (Hz) and affects the smoothness of motion on the screen.

6. Response Time: The response time of a raster display refers to how quickly individual pixels can change from one color or brightness to another. Faster response times are important for reducing motion blur, especially in gaming and video applications.

7. Scalability: Raster displays are scalable in terms of resolution, but increasing resolution requires more processing power and higher-quality hardware to maintain image quality.

8. Applications: Raster scan displays are commonly used in a wide range of applications, including computer monitors, televisions, digital cameras, and mobile devices. They are well-suited for displaying static images, videos, and most computer graphics.

<u>RANDOM SCAN DISPLAY:</u>

A random scan display, also known as a vector display or calligraphic display, is a type of computer graphics display technology that creates images by directly drawing and positioning lines or curves on the screen rather than scanning a grid of pixels.

<u>Here are key details about random scan displays:</u>

1. Vector-Based: Random scan displays are vector-based, which means that they represent images using geometric primitives such as lines, curves, and points. These primitives are drawn directly on the screen.

2. Electron Beam Deflection: In random scan displays, an electron beam is used to move rapidly to the desired screen positions, drawing lines and shapes as it goes. The beam can be redirected to any point on the screen almost instantaneously.

3. No Fixed Pixel Grid: Unlike raster displays, random scan displays do not have a fixed pixel grid. Instead, they rely on the precision of the electron beam and the speed of deflection to create smooth, scalable images.

4. Resolution: Random scan displays are resolution-independent, as they can create sharp lines and curves regardless of the screen's size or resolution.

5. Monochrome: Early random scan displays were typically monochrome and used for tasks like computer-aided design (CAD) and scientific visualization. They were known for their high precision in drawing complex shapes.

6. Limited Color Support: Some random scan displays support color, but achieving full-color graphics is more complex and less common compared to raster displays.

7. Response Time: Random scan displays have very fast response times because they can position the electron beam quickly and precisely.

8. Applications: Random scan displays are primarily used in applications where precise line drawing and vector-based graphics are essential, such as CAD systems, scientific visualization, and certain specialized design and simulation applications.

<u>CHARACTERISTICS OF DISPLAY DEVICES EXPLAINED IN DETAIL:</u>

1. Resolution:

- Definition: Resolution refers to the number of pixels (individual picture elements) that a display can render both horizontally and vertically.

- Importance: Higher resolution displays can show more detail, resulting in sharper text and images. Common resolutions include Full HD (1920x1080), 4K (3840x2160), and 8K (7680x4320).

2. Screen Size:

- Definition: Screen size measures the diagonal length of the display area, typically in inches.

- Importance: Larger screens provide a more immersive viewing experience, but they may require more physical space and higher resolution to maintain image quality.

3. Aspect Ratio:

- Definition: The aspect ratio is the ratio of the display's width to its height (e.g., 16:9 or 4:3).

- Importance: Aspect ratio affects the display's suitability for different content types. Widescreen (16:9) is common for video, while 4:3 is more traditional for older computer monitors.

4. Refresh Rate:

- Definition: Refresh rate measures how many times per second the display updates or redraws the image. It is measured in Hertz (Hz).

- Importance: Higher refresh rates, such as 60Hz, 120Hz, or 240Hz, result in smoother motion in videos and games. Gamers often prefer higher refresh rates for reduced motion blur.

5. Color Depth:

- Definition: Color depth, also known as bit depth, specifies the number of distinct colors or shades of gray that a display can produce.

- Importance: Greater color depth provides more accurate and vibrant color representation. Common depths include 8-bit (16.7 million colors) and 10-bit (over a billion colors).

6. Brightness:

- Definition: Brightness measures the intensity of the light emitted by the display, typically in nits or candelas per square meter (cd/m^2).

- Importance: Brightness affects the display's visibility in various lighting conditions. High brightness is essential for outdoor displays, while lower brightness is suitable for indoor use.

7. Contrast Ratio:

- Definition: Contrast ratio quantifies the difference in luminance (brightness) between the brightest white and the darkest black the display can produce.

- Importance: A higher contrast ratio results in better differentiation between light and dark areas, enhancing image quality and readability.

8. Response Time:

- Definition: Response time measures how quickly individual pixels can change from one color to another, typically measured in milliseconds (ms).

- Importance: Lower response times reduce motion blur in fast-paced content like gaming or video playback.

9. Viewing Angle:

- Definition: Viewing angle describes the range of angles from which the display can be viewed without significant color distortion or loss of brightness.

- Importance: Displays with wider viewing angles are suitable for larger audiences and maintain image quality even when viewed from the sides.

10. Panel Type:

- Definition: Panel type refers to the technology used to create the display, such as LCD (Liquid Crystal Display), OLED (Organic Light Emitting Diode), or LED (Light Emitting Diode).

- Importance: Different panel types offer varying advantages in terms of color accuracy, energy efficiency, and viewing angles.

11. Connectivity Options:

- Definition: Connectivity options include the types and number of ports available on the display for connecting to other devices (e.g., HDMI, DisplayPort, USB-C).

- Importance: Sufficient connectivity ensures compatibility with various devices such as computers, gaming consoles, and media players.

12. Touchscreen Capabilities:

- Definition: Some displays are equipped with touchscreen technology, enabling users to interact with the screen directly through touch gestures.

- Importance: Touchscreens are essential for devices like smartphones, tablets, and interactive kiosks.

13. Energy Efficiency:

- Definition: Energy efficiency measures the display's power consumption, which can impact both operating costs and environmental considerations.

- Importance: Energy-efficient displays are preferable to reduce electricity bills and minimize environmental impact.

14. Durability and Build Quality:

- Definition: Durability relates to the display's resistance to physical damage, while build quality considers the materials used and overall construction.

- Importance: Displays used in rugged environments may require higher durability, while premium build quality enhances aesthetics and longevity.

15. Anti-Glare and Blue Light Filters:

- Definition: Some displays feature anti-glare coatings and blue light filters to reduce reflections and minimize eye strain.

- Importance: These features enhance comfort during extended use and reduce glare in brightly lit environments.

16. Adjustability:

- Definition: Adjustability includes features like tilt, swivel, height adjustment, and pivot capabilities to optimize ergonomics.

- Importance: Adjustable displays allow users to customize their viewing experience for comfort and productivity.

17. Built-in Speakers and Audio Features:

- Definition: Some displays have built-in speakers or advanced audio technologies to enhance sound quality.

- Importance: Built-in audio features are convenient for multimedia applications and video conferencing.

ALIASING:

Aliasing is a visual artifact that occurs when a computer graphics system tries to represent a continuous, smooth curve or line using a limited set of discrete points, such as pixels on a screen. This phenomenon can result in jagged or stair-stepped edges, which are particularly noticeable when rendering high-frequency patterns, fine details, or diagonal lines. Aliasing is caused by the Nyquist-Shannon sampling theorem, which states that to accurately represent a signal (in this case, an image), you must sample it at a rate at least twice the highest frequency present in the signal. When the sampling rate is insufficient, aliasing occurs.

Here are the key points about aliasing:

1. Jagged Edges: Aliasing manifests as jagged or "stair-stepped" edges in rendered images, which can make curves and diagonal lines appear rough and unrealistic.

2. High-Frequency Patterns: Aliasing is most noticeable in images with high-frequency patterns or fine details, as these contain a wide range of rapidly changing colors or shades.

3. Pixel Grid: The aliasing effect is often a result of aligning a smooth curve or line with the pixel grid of a screen. Pixels are discrete, so they cannot represent the full range of colors or shades found within a curve or line.

4. Spatial Aliasing: Spatial aliasing occurs when an image's resolution (number of pixels) is insufficient to represent fine details, causing loss of information. This is common in digital photography and computer graphics.

5. Temporal Aliasing: Temporal aliasing occurs in animations or video when the frame rate is too low to capture smooth motion, resulting in a flickering or strobing effect.

6. Antialiasing Solutions: Antialiasing techniques are employed to reduce or eliminate aliasing artifacts, making images appear smoother and more realistic.

ANTIALIASING:

Antialiasing is a set of techniques used in computer graphics to reduce or eliminate aliasing artifacts and create smoother, more visually pleasing images. These techniques work by adding additional information to the image or altering the sampling process to better represent curves, lines, and fine details.

Here are the key points about antialiasing:

1. Super-sampling: Super-sampling is a common antialiasing technique that involves rendering the image at a higher resolution and then down-sampling it to the target resolution. This process captures more detail and averages pixel values to reduce jagged edges.

2. Multisampling: Multisampling is a variation of super-sampling that focuses antialiasing efforts on edges and high-contrast areas, rather than the entire image, which reduces the computational load.

3. FXAA (Fast Approximate Antialiasing): FXAA is a post-processing antialiasing technique that works on the final image. It analyzes the image for high-contrast edges and applies a smoothing filter to them.

4. MSAA (Multisample Anti-Aliasing): MSAA is a common hardware-based antialiasing method that uses multiple samples per pixel. It produces smoother edges without the need for extensive computational power.

5. Temporal Antialiasing (TAA): TAA is used in animations and video games to reduce temporal aliasing. It combines information from multiple frames to create smoother motion and eliminate flickering.

6. Subpixel Rendering: Subpixel rendering takes advantage of the fact that each pixel on an LCD screen is composed of three subpixels (red, green, blue). By manipulating the color of these subpixels, subpixel rendering can enhance the apparent resolution of text and graphics.

7. Mipmapping: Mipmapping is a technique used in texture mapping to reduce aliasing artifacts when textures are applied to 3D objects. It involves precomputing and storing multiple versions of a texture at different resolutions.

8. Polygon Smoothing: In 3D graphics, antialiasing can be applied to polygon edges to reduce aliasing artifacts that occur when rendering three-dimensional scenes.

LED (LIGHT EMITTING DIODE) DISPLAYS:

1. Technology Overview: LED displays use an array of light-emitting diodes to produce images and visuals on a screen. LEDs emit light when an electric current passes through them, and they are used as individual pixels in the display.

2. Key Features:

- Brightness: LED displays are known for their high brightness levels, making them suitable for well-lit environments and outdoor use.
- Energy Efficiency: LEDs are energy-efficient and have a longer lifespan compared to other display technologies, reducing power consumption and maintenance costs.
- Slim Profile: LED displays can be made very thin, allowing for sleek and space-saving designs.
- Color Accuracy: LED displays offer excellent color accuracy and a wide color gamut, making them ideal for applications like digital signage and high-quality television screens.
- Versatility: LEDs can be used in a variety of display types, including LED TVs, LED monitors, and LED video walls.

3. Applications:

- Consumer Electronics: LED displays are commonly used in televisions, computer monitors, and laptops.
- Digital Signage: LED displays are popular for outdoor billboards, indoor advertising screens, and information displays due to their brightness and visibility.
- Sports Stadiums: LED video boards are frequently used in sports arenas to provide high-resolution live feeds and replays to the audience.
- Outdoor Displays: LEDs are suitable for outdoor displays, such as highway signs and building facades, due to their durability and visibility in daylight.

OLED (ORGANIC LIGHT EMITTING DIODE) DISPLAYS:

1. Technology Overview: OLED displays use organic compounds that emit light when an electric current is applied. Unlike LED displays, OLEDs are capable of emitting their own light individually, eliminating the need for a separate backlight.

2. Key Features:

- Self-emissive: Each pixel in an OLED display emits its own light, allowing for true black levels and infinite contrast ratios.

- Thin and Flexible: OLED displays can be manufactured on flexible substrates, enabling curved and foldable display designs.

- Wide Viewing Angles: OLEDs offer wide viewing angles with consistent color and brightness, making them suitable for curved and large-screen displays.

- Fast Response Times: OLEDs have fast response times, making them ideal for gaming and video playback.

- Energy Efficiency: OLEDs are energy-efficient because they only emit light where needed, reducing power consumption.

3. Applications:

- Smartphones: OLED displays are commonly used in high-end smartphones due to their vibrant colors, deep blacks, and power efficiency.

- Television Screens: OLED TVs are known for their exceptional picture quality, including rich colors and high contrast.

- Wearables: Flexible OLED displays are used in smartwatches and other wearable devices.

- Automotive Displays: OLED screens are increasingly used in car dashboards and infotainment systems for their clarity and adaptability to curved surfaces.

CURVED LED DISPLAYS:

1. Technology Overview: Curved LED displays use the same LED technology as flat LED displays but are curved or bent to create a concave or convex screen surface. These displays are designed to immerse viewers in the content by providing a wraparound visual experience.

2. Key Features:

- Immersive Viewing: Curved displays are intended to create a more immersive viewing experience by matching the curvature of the human eye, which can reduce distortion and create a sense of depth.

- Enhanced Field of View: Curved displays are often used in gaming monitors to provide a wider field of view and a more realistic gaming experience.

- Unique Aesthetics: The curved form factor of these displays can add a distinctive and stylish look to a room or space.

3. Applications:

- Gaming: Curved gaming monitors and gaming laptops use curved displays to enhance the gaming experience by increasing immersion and reducing distortion at the edges of the screen.

- Home Theater: Curved TVs are used in home theaters to provide a more engaging cinematic experience.

- Simulators: Curved displays are commonly employed in flight simulators and driving simulators to create a realistic visual environment.

SCAN CONVERSION:

Scan Conversion is a fundamental concept in computer graphics that involves the process of converting geometric shapes and objects represented in vector or analytical form into a pixel-based, rasterized format. This conversion is essential because most display devices, including computer monitors and screens, work with a grid of discrete picture elements or pixels. Scan conversion determines how these geometric objects are displayed on the screen, and it plays a pivotal role in rendering graphics and images in digital applications.

BASIC STEPS OF SCAN CONVERSION:

1. Input Geometry: The process begins with the description of a geometric shape or object in vector or analytical form. This description typically includes details like the coordinates of vertices, equations of curves, and other relevant parameters.

2. Selection of Pixel Grid: Scan conversion involves mapping the continuous geometry onto a discrete grid of pixels. The resolution of this grid (i.e., the number of pixels horizontally and vertically) determines the level of detail and clarity in the final rendered image.

3. Sampling and Rasterization: The geometric shape is sampled and mapped onto the pixel grid, determining which pixels should be part of the object and which should not. This process is known as rasterization. Each pixel is assigned a color or intensity value based on its relationship to the geometry. Pixels that fall inside the object boundaries are often fully colored, while those partially covered may be given partial shading or anti-aliased values to represent smooth transitions.

4. Rendering: Once the pixel grid is filled with the appropriate color values, the rendered image is displayed on the screen or output device. This image is made up of discrete pixels, each representing a part of the original geometric object.

KEY CONCEPTS AND CONSIDERATIONS:

- Aliasing: Aliasing is a common issue in scan conversion where the limited resolution of the pixel grid leads to visual artifacts like jagged edges or moiré patterns. Antialiasing techniques, such as supersampling or smoothing, are used to mitigate these effects and produce smoother images.

- Interpolation: When mapping continuous geometric shapes onto a grid of discrete pixels, interpolation techniques are used to determine the color or intensity of each pixel. This ensures that objects and curves appear smooth and realistic, even when represented with limited pixel data.

- Clipping: Clipping is the process of identifying and removing parts of geometric objects that fall outside the boundaries of the viewing window or screen. It ensures that only the visible portions of objects are scan-converted.

- Efficiency: Efficiency is a critical concern in scan conversion, especially for real-time or interactive graphics applications. Algorithms and techniques are designed to optimize the conversion process and minimize computational overhead.

- Anti-aliasing: As mentioned earlier, anti-aliasing techniques are used to reduce or eliminate aliasing artifacts, ensuring that edges and curves appear smooth and natural, even when represented on a pixel grid.

- 3D Projection: In 3D graphics, scan conversion extends to the process of projecting three-dimensional objects onto a two-dimensional screen or viewport. This involves perspective transformations and depth calculations to determine the position and visibility of objects in the rendered scene.

APPLICATIONS OF SCAN CONVERSION:

Scan conversion is a fundamental concept used in various computer graphics applications, including:

- Rendering 2D Graphics: Converting vector shapes, lines, and curves into pixel-based images for display on screens.

- 3D Rendering: Projecting and rendering three-dimensional objects and scenes onto 2D screens.

- Computer-Aided Design (CAD): Converting vector-based designs and models into raster images for visualization.

- Video Games: Transforming game objects, characters, and environments into pixel-based graphics for real-time rendering.

- Medical Imaging: Converting medical scan data (e.g., CT scans, MRIs) into visual representations.

- Animation: Scan conversion plays a crucial role in rendering animated sequences and special effects in films and video production.

DDA (DIGITAL DIFFERENTIAL ANALYZER) ALGORITHM:

The DDA algorithm is a straightforward method for scan converting lines. It calculates the coordinates of pixels along the line using linear interpolation.

Here's how it works:

Step 1: Input

- Given two endpoints: (x1, y1) and (x2, y2) that define the line.

Step 2: Determine the Number of Steps

- Calculate the number of steps needed to move from one endpoint to the other. Find the maximum difference between x-coordinates and y-coordinates: steps = max(|x2 - x1|, |y2 - y1|).

Step 3: Calculate Increments

- Determine the increments in x and y for each step:

  - Increment in x (dx) = (x2 - x1) / steps

  - Increment in y (dy) = (y2 - y1) / steps

Step 4: Initialize Starting Point

- Initialize a starting point (x, y) at (x1, y1).

Step 5: Scan Conversion

- For each step from 1 to the total number of steps:
  - Plot the pixel at coordinates (x, y).
  - Update the coordinates for the next step:
    - x = x + dx
    - y = y + dy

Example of DDA Line Drawing Algorithm:

Let's draw a line between points A(1, 1) and B(7, 4) using the DDA algorithm.

- Given endpoints: A(1, 1) and B(7, 4)
- Calculate steps = max(|7 - 1|, |4 - 1|) = 6
- Calculate increments:
  - dx = (7 - 1) / 6 = 1
  - dy = (4 - 1) / 6 = 1/2
- Initialize starting point: (x, y) = (1, 1)

Now, we perform the scan conversion step by step:

- Step 1: Plot pixel at (1, 1) (starting point).
- Step 2: Update (x, y) to (2, 1.5).
- Step 3: Plot pixel at (2, 2).
- Step 4: Update (x, y) to (3, 2.5).

- Step 5: Plot pixel at (3, 3).
- Step 6: Update (x, y) to (4, 3.5).
- Step 7: Plot pixel at (4, 4).

The DDA algorithm has drawn a line from point A(1, 1) to point B(7, 4).

BRESENHAM'S LINE DRAWING ALGORITHM:

Bresenham's algorithm is another widely used method for scan converting lines. It is particularly efficient because it works with integer arithmetic only. Here's how it works:

Step 1: Input

- Given two endpoints: (x1, y1) and (x2, y2) that define the line.

Step 2: Initialize Parameters

- Initialize parameters for the decision-making process:
  - dx = |x2 - x1|
  - dy = |y2 - y1|
  - P = 2 dy - dx (decision parameter)
  - x = x1 (initial x-coordinate)
  - y = y1 (initial y-coordinate)

Step 3: Scan Conversion

- For each x from x1 to x2:
  - Plot the pixel at coordinates (x, y).
  - If P < 0, increment x and update P as P = P + 2 dy.
  - If P ≥ 0, increment x and decrement y, and update P as P = P + 2 dy - 2 dx.

Example of Bresenham's Line Drawing Algorithm:

Let's draw a line between points A(1, 2) and B(7, 5) using Bresenham's algorithm.

- Given endpoints: A(1, 2) and B(7, 5)

- Initialize parameters:
  - dx = |7 - 1| = 6
  - dy = |5 - 2| = 3
  - P = 2 3 - 6 = 0 (initial decision parameter)
  - x = 1 (initial x-coordinate)
  - y = 2 (initial y-coordinate)

Now, we perform the scan conversion step by step:

- At x = 1, plot pixel at (1, 2).
- At x = 2, plot pixel at (2, 2).
- At x = 3, plot pixel at (3, 3).
- At x = 4, plot pixel at (4, 3).
- At x = 5, plot pixel at (5, 4).
- At x = 6, plot pixel at (6, 4).
- At x = 7, plot pixel at (7, 5).

Bresenham's algorithm has drawn a line from point A(1, 2) to point B(7, 5).

BRESENHAM'S CIRCLE-DRAWING ALGORITHM:

Scan converting circles using Bresenham's algorithm is a fundamental technique in computer graphics for rendering circles on a pixel grid efficiently. The Bresenham circle-drawing algorithm is particularly useful because it only uses integer arithmetic and is capable of generating accurate circle approximations.

The Bresenham circle-drawing algorithm draws a circle by selecting a set of points (pixels) that are approximately equidistant from the center of the circle. It uses a decision parameter to determine the position of each point on the circle's circumference.

Step 1: Input

- Given the center of the circle: (x0, y0) and its radius: r.

Step 2: Initialize Parameters

- Initialize parameters for the decision-making process:
  - x = r
  - y = 0
  - P = 1 - r (decision parameter)

Step 3: Plot Initial Points

- Plot the initial points, taking symmetry into account. Typically, you plot points in the eight octants of the circle to minimize computation and take advantage of symmetry.
  - Plot points (x0 + x, y0 + y) and (x0 - x, y0 + y).
  - Plot points (x0 + x, y0 - y) and (x0 - x, y0 - y).
  - Plot points (x0 + y, y0 + x) and (x0 - y, y0 + x).
  - Plot points (x0 + y, y0 - x) and (x0 - y, y0 - x).

Step 4: Scan Conversion

- For each step from 1 to x:
  - If P < 0, update P as P = P + 2 (x + 1) + 1.
  - If P ≥ 0, decrement x and update P as P = P + 2 (x + 1) - 2 (y - 1) + 1.
  - Increment y.
  - Plot points as in Step 3 for the current values of x and y.

<u>**Example of Bresenham's Circle-Drawing Algorithm:**</u>

Let's draw a circle with a center at (5, 5) and a radius of 4 using Bresenham's algorithm.

- Given center: (x0, y0) = (5, 5)
- Given radius: r = 4

Step 2: Initialize Parameters

- Initialize parameters:
  - x = 4
  - y = 0

  - P = 1 - 4 = -3

Step 3: Plot Initial Points

- Plot points: (9, 5), (1, 5), (5, 9), (5, 1), (9, 9), (1, 9), (9, 1), (1, 1)

Step 4: Scan Conversion

- For each step from 1 to x (from 4 to 1):
  - At x = 4:
    - P < 0, so update P as P = -3 + 2 (4 + 1) + 1 = 4
    - Increment y to 1.
    - Plot points: (9, 6), (1, 6), (6, 9), (6, 1)
  - At x = 3:
    - P ≥ 0, so decrement x to 3.
    - Update P as P = 4 + 2 (3 + 1) - 2 (1 - 1) + 1 = 8
    - Increment y to 2.
    - Plot points: (8, 7), (2, 7), (7, 8), (7, 2)
  - Continue this process until x becomes 1.

The Bresenham's circle-drawing algorithm has drawn a circle with a center at (5, 5) and a radius of 4.

This algorithm is efficient and suitable for generating circle approximations with integer coordinates, making it widely used in computer graphics for rendering circles.

UNIT-II

CLIPPING:

Clipping is a fundamental concept in computer graphics that involves removing portions of lines, polygons, or other geometric objects that lie outside a specified region or window. Two popular algorithms for clipping lines and segments are the Cohen-Sutherland algorithm and the Cyrus-Beck algorithm.

COHEN-SUTHERLAND ALGORITHM:

The Cohen-Sutherland algorithm is a line-clipping algorithm that uses a region-based approach to determine which portions of a line segment lie inside a rectangular clipping window. It divides the entire space into nine regions based on a binary encoding of points' positions relative to the clipping window's boundaries (left, right, top, and bottom).

Step 1: Initialize Region Codes

- Assign binary region codes to the line's endpoints and the clipping window boundaries based on their relative positions. For example:

  - Left of the window: 1001
  - Right of the window: 0100
  - Above the window: 0010
  - Below the window: 0001

Step 2: Perform Clipping

- Check the region codes of both endpoints. Based on these codes, decide whether the line segment is completely inside, completely outside, or partially inside the clipping window.

- If both endpoints have region code 0000 (inside the window), the entire line segment is inside, and no clipping is needed.

- If both endpoints have a non-zero bitwise AND operation result (region code), the line segment is entirely outside the window and can be discarded.

- If neither of the above conditions is met, calculate the intersection of the line with the window boundaries (using the parametric equation of the line). Update the endpoints to the intersection points while adjusting their region codes accordingly.

Example of Cohen-Sutherland Algorithm:

Let's clip a line segment from point A(20, 40) to point B(70, 10) using a rectangular clipping window with corners P1(30, 20) and P2(60, 50).

- Assign region codes to endpoints A and B:
  - A: 1000 (left of the window)
  - B: 0100 (right of the window)

Step 2: Perform Clipping

- Both A and B have non-zero region codes, indicating they are outside the window. The line segment is completely outside the window and can be discarded.

CYRUS-BECK ALGORITHM:

The Cyrus-Beck algorithm is a line-clipping algorithm that operates in parametric space. It works by determining the intersection points between a line segment and a clipping polygon (defined by its edges) in parametric form. The algorithm is particularly useful for convex polygons.

Step 1: Define the Clipping Polygon

- Define the clipping polygon as a set of edges, each represented by a pair of points.

Step 2: Calculate Parametric Intersection

- For each edge of the clipping polygon, calculate the parametric values (t values) where the line intersects the edge using parametric equations. The parametric equation of a line is: P(t) = P1 + t (P2 - P1), where P(t) is the point on the line at parameter t.

- Determine the range of t values (t_min and t_max) that represent the portion of the line inside the clipping polygon based on the intersections with all clipping edges.

Step 3: Perform Clipping

- If t_max < t_min, there is no intersection, and the entire line is outside the polygon; discard the line.

- If t_max < 0 or t_min > 1, the line is entirely outside the polygon; discard the line.

- If 0 ≤ t_min ≤ 1 and 0 ≤ t_max ≤ 1, the line is partially inside the polygon. Calculate the intersection points P1' and P2' using the values of t_min and t_max.

- Clip the line segment between P1' and P2' based on the calculated parameters.

Example of Cyrus-Beck Algorithm:

Let's clip a line segment from point A(1, 2) to point B(6, 6) using a convex clipping polygon defined by the edges: (1, 1) to (4, 0), (4, 0) to (7, 3), and (7, 3) to (3, 7).

Step 2: Calculate Parametric Intersection

- Calculate parametric values for each intersection:
  - Intersection with edge 1: t1 ≈ 0.2857
  - Intersection with edge 2: t2 ≈ 1.25
  - Intersection with edge 3: t3 ≈ 0.625

Step 3: Perform Clipping

- Determine t_min = max(0, max(t1, t2, t3)) ≈ 0.625 and t_max = min(1, min(t1, t2, t3)) ≈ 0.625.

- Since 0 ≤ t_min ≤ t_max ≤ 1, the line segment is partially inside the polygon.

- Calculate the intersection points: P1' ≈ (2.375, 3.625) and P2' ≈ (4.625, 4.375).

- The clipped line segment is from P1' to P2': (2.375, 3.625) to (4.625, 4.375).

The Cyrus-Beck algorithm has clipped the line segment within the specified convex clipping polygon.

GEOMETRICAL TRANSFORMATIONS:

Geometrical transformations in computer graphics refer to operations that modify the shape, size, position, or orientation of objects or images in a 2D or 3D space. These transformations are fundamental for creating and manipulating visual elements in digital graphics. Geometrical transformations include translation, rotation, scaling, reflection, shearing, and more.

1. Translation:

- Translation moves an object from one location to another by adding a constant value to its coordinates.
- Example: Consider a point A(2, 3). Translate it by (dx, dy) = (3, 2), and the new location is A'(5, 5).

2. Rotation:

- Rotation changes the orientation of an object by a specified angle around a fixed point (often the origin).
- Example: Rotate a point A(3, 2) by 45 degrees counterclockwise around the origin to get A'(0.707, 3.536).

3. Scaling:

- Scaling changes the size of an object by multiplying its coordinates by scaling factors (sx, sy) in the x and y directions, respectively.
- Example: Scale a point A(2, 3) by (sx, sy) = (2, 0.5) to get A'(4, 1.5).

4. Reflection:

- Reflection mirrors an object across an axis, often the x-axis or y-axis.
- Example: Reflect a point A(2, 3) across the x-axis to get A'(2, -3).

5. Shearing:

- Shearing distorts an object along one axis, usually by altering its y-coordinates based on x-coordinates or vice versa.
- Example: Shear a point A(2, 3) along the x-axis by a shear factor k to get A'(2 + k 3, 3).

6. Affine Transformations:

- Affine transformations combine translation, rotation, scaling, and shearing operations.
- Example: Apply an affine transformation to a rectangle to change its size, position, and orientation.

7. Homogeneous Coordinates:

- In 3D graphics, homogeneous coordinates are used to represent transformations. They include a fourth coordinate, w, which can be used for perspective transformations.

- Example: Transform a 3D point (x, y, z, w) using a 4x4 transformation matrix.

8. Composition of Transformations:

- Multiple transformations can be combined by multiplying their corresponding transformation matrices.

- Example: Translate an object, then rotate it, and finally scale it using matrix multiplication.

9. Interpolation and Bézier Curves:

- Geometrical transformations can also include interpolations and curve constructions.

- Example: Use Bézier curves to create smooth paths between two points.

10. 3D Transformations:

- In 3D graphics, transformations are extended to include 3D translations, rotations, scalings, and projections.

- Example: Rotate and translate a 3D object in a 3D space.

11. Non-Uniform Scaling:

- Scaling factors can be different along each axis, leading to non-uniform scaling.

- Example: Scale a rectangle differently in the x and y directions.

12. Hierarchical Transformations:

- Transformations can be applied hierarchically to objects within a scene, allowing for complex animations and simulations.

- Example: Transform a robot's limbs individually, and then transform the entire robot's position and orientation.

13. 3D Camera Transformations:

- In 3D computer graphics, camera transformations are used to set the viewpoint and projection parameters.

- Example: Adjust the camera's position, orientation, and field of view to capture a scene from a specific perspective.

2D TRANSFORMATIONS:

2D transformations in computer graphics are operations that manipulate the position, orientation, size, and shape of 2D objects or images in a two-dimensional coordinate system. These transformations are fundamental for creating and manipulating visual elements in digital graphics. Common 2D transformations include translation, rotation, scaling, reflection, shearing, and more.

1. Translation:

- Description: Translation moves an object from one location to another by adding a constant value to its x and y coordinates.
- Mathematical Representation: If the translation vector is (dx, dy), the new coordinates of a point (x, y) after translation become (x + dx, y + dy).
- Example: If you have a point A(2, 3) and you translate it by (dx, dy) = (3, 2), the new location is A'(5, 5).

2. Rotation:

- Description: Rotation changes the orientation of an object by a specified angle θ around a fixed point (often the origin).
- Mathematical Representation: The new coordinates of a point (x, y) after rotation become (x' , y') as follows:
  - $x' = x \ \cos(\theta) - y \ \sin(\theta)$
  - $y' = x \ \sin(\theta) + y \ \cos(\theta)$
- Example: Rotate a point A(2, 3) by 45 degrees counterclockwise ($\theta = \pi/4$ radians) around the origin to get A'(0.707, 3.536).

3. Scaling:

- Description: Scaling changes the size of an object by multiplying its x and y coordinates by scaling factors (sx, sy).
- Mathematical Representation: The new coordinates of a point (x, y) after scaling become (x' , y') as follows:
  - $x' = x \ sx$

  - y' = y sy

- Example: Scale a point A(2, 3) by (sx, sy) = (2, 0.5) to get A'(4, 1.5).

4. Reflection:

- Description: Reflection mirrors an object across an axis, often the x-axis or y-axis.

- Mathematical Representation: For reflection about the x-axis, the new coordinates of a point (x, y) become (x' , y') as follows:

  - x' = x

  - y' = -y

- Example: Reflect a point A(2, 3) across the x-axis to get A'(2, -3).

5. Shearing:

- Description: Shearing distorts an object along one axis based on the values of the other axis.

- Mathematical Representation: The new coordinates of a point (x, y) after shearing become (x' , y') as follows:

  - x' = x + k y

  - y' = y + k x

- Example: Shear a point A(2, 3) along the x-axis by a shear factor k = 2 to get A'(4, 7).

6. Affine Transformations:

- Description: Affine transformations combine translation, rotation, scaling, and shearing operations.

- Mathematical Representation: Affine transformations are represented using matrices, and multiple transformations can be combined by matrix multiplication.

- Example: Apply an affine transformation to a rectangle to change its size, position, and orientation.

7. Composite Transformations:

- Description: Multiple transformations can be combined to perform complex operations on objects.

- Mathematical Representation: Composite transformations are achieved by applying a sequence of individual transformations in a specific order.

- Example: Translate, then rotate, and finally scale an object using a sequence of transformation operations.

HOMOGENEOUS COORDINATES AND MATRIX REPRESENTATION OF 2D TRANSFORMATIONS:

Homogeneous Coordinates and Matrix Representation are fundamental concepts in computer graphics that simplify the handling and concatenation of 2D transformations. They are particularly useful when combining multiple transformations and performing operations like translation, rotation, scaling, and shearing

HOMOGENEOUS COORDINATES:

Homogeneous coordinates are an extension of the regular Cartesian coordinates (x, y) to include an additional coordinate (w). In 2D homogeneous coordinates, a point is represented as (x, y, w). The use of homogeneous coordinates simplifies the representation of transformations and allows for the representation of points at infinity, making it valuable in computer graphics.

Homogeneous Coordinates Conversion:

To convert between regular Cartesian coordinates (x, y) and homogeneous coordinates (x, y, w), you use the following conversions:

- From Cartesian to Homogeneous:
  - (x, y) → (x, y, 1)

- From Homogeneous to Cartesian:
  - (x, y, w) → (x/w, y/w)

The third coordinate, w, is typically set to 1 for finite points, while for points at infinity (e.g., direction vectors), w can be set to 0.

MATRIX REPRESENTATION OF 2D TRANSFORMATIONS:

Matrix representation simplifies the application of 2D transformations, as it allows you to concatenate multiple transformations efficiently. In the context of 2D transformations, transformation matrices are 3x3 matrices that include the 2D transformation along with the homogeneous coordinate w. Here are some common transformation matrices:

1. Translation Matrix (T):

   - Translates a point by (dx, dy).

```
T = | 1 0 dx |
    | 0 1 dy |
    | 0 0  1 |
```

2. Rotation Matrix (R):

   - Rotates a point by an angle θ counterclockwise.

```
R = | cos(θ) -sin(θ) 0 |
    | sin(θ)  cos(θ) 0 |
    |   0       0      1 |
```

3. Scaling Matrix (S):

   - Scales a point by factors (sx, sy).

```
S = | sx  0  0 |
    |  0 sy  0 |
    |  0  0  1 |
```

4. Shearing Matrix (Sh):

   - Shears a point in the x or y direction.

```
Sh_x = | 1  k  0 |
       | 0  1  0 |
       | 0  0  1 |
Sh_y = | 1  0  0 |
       | k  1  0 |
       | 0  0  1 |
```

Concatenation of Transformations:

You can combine multiple transformations by multiplying their corresponding matrices together. For example, to apply a sequence of transformations T1, T2, and T3 to a point P, you calculate:

```
P' = T3 * T2 * T1 * P
```

Example: Applying Multiple Transformations using Matrices:

Let's say you want to apply the following transformations to a point A(2, 3):

1. Translate it by (3, 2).

2. Rotate it by 45 degrees counterclockwise around the origin.

3. Scale it by (2, 0.5).

First, create the transformation matrices for each operation:

1. Translation Matrix (T):

```
T = | 1 0 3 |
    | 0 1 2 |
    | 0 0 1 |
```

2. Rotation Matrix (R):

```
R = | 0.707 -0.707 0 |
    | 0.707  0.707 0 |
    |   0      0    1 |
```

3. Scaling Matrix (S):

```
S = | 2  0  0 |
    | 0 0.5 0 |
    | 0  0  1 |
```

Now, apply these transformations to the point A(2, 3):

```
P' = S * R * T * A
   = | 2  0  0 | * | 0.707 -0.707 0 | * | 1 0 3 | * | 2 |
     | 0 0.5 0 |   | 0.707  0.707 0 |   | 0 1 2 |   | 3 |
     | 0  0  1 |   |   0      0
```

COMPOSITION OF 2D TRANSFORMATIONS:

Composition of 2D Transformations is a fundamental concept in computer graphics that involves combining multiple individual transformations to create a single composite transformation. This allows you to apply a series of transformations sequentially to achieve complex effects on an object or image. The order in which transformations are applied matters, as it can lead to different results.

In 2D graphics, the typical transformations include translation, rotation, scaling, reflection, and shearing, among others.

The general procedure for composing 2D transformations is as follows:

1. Identify the transformations you want to apply and their order.

2. Create transformation matrices for each individual transformation.

3. Multiply the transformation matrices in the desired order to create a composite transformation matrix.

4. Apply the composite transformation matrix to the object or points you want to transform.

Example of Composition of 2D Transformations:

Suppose you want to perform the following sequence of transformations on a rectangle:

1. Translate it by (2, 3) units.

2. Rotate it by 45 degrees counterclockwise around the origin.

3. Scale it by a factor of 2 in both the x and y directions.

Step 1: Identify Transformations and Order

We have identified three transformations in the given order: translation, rotation, and scaling.

Step 2: Create Transformation Matrices

Let's create the transformation matrices for each transformation:

1. Translation Matrix (T):

```
T = | 1 0 2 |
    | 0 1 3 |
    | 0 0 1 |
```

2. Rotation Matrix (R):

```
R = | 0.707 -0.707 0 |
    | 0.707  0.707 0 |
    |   0      0    1 |
```

3. Scaling Matrix (S):

```
S = | 2  0  0 |
    | 0 2  0 |
    | 0  0  1 |
```

Step 3: Compose Transformation Matrix

To create a composite transformation matrix (C), multiply the matrices in the order you want to apply the transformations:

```
C = T * R * S
```

Step 4: Apply Composite Transformation

Now, apply the composite transformation matrix (C) to the rectangle. Let's take one of the rectangle's vertices as an example, say A(1, 1):

```
A' = C * A
```

To find A', perform the matrix multiplication:

```
| x' |    | 1 0 2 |    | 0.707 -0.707 0 |    | 2 |
| y' | = | 0 1 3 | * | 0.707  0.707 0 | * | 1 |
| 1  |    | 0 0 1 |    |   0      0    1 |    | 1 |
```

Simplify the multiplication:

```
x' = 3.414
y' = 6.414
1  = 1
```

So, the transformed point A' is approximately (3.414, 6.414).

WINDOW-TO-VIEWPORT TRANSFORMATION:

Window-to-Viewport Transformation, also known as viewport transformation, is a fundamental concept in computer graphics that involves mapping the coordinates of objects or scenes from a virtual or abstract coordinate system (known as the "window" or "world" coordinates) to the actual display or screen coordinates (the "viewport"). This transformation is essential for displaying graphics on computer screens or other output devices with different resolutions, aspect ratios, and sizes.

Key Components of Window-to-Viewport Transformation:

1. Window Coordinates (World Coordinates): These are the coordinates that represent the virtual or abstract space in which your graphical objects or scenes are defined. They often have arbitrary units and may not match the actual screen size or aspect ratio.

2. Viewport Coordinates: These are the coordinates of the physical display or screen on which you want to render your graphics. Viewport coordinates are usually measured in pixels and correspond to the actual screen's dimensions and resolution.

3. Window-to-Viewport Transformation Matrix: This matrix represents the transformation that maps points from the window coordinates to the viewport coordinates. It typically includes scaling, translation, and possibly rotation operations to adjust the size, position, and orientation of the graphics within the viewport.

Steps in Window-to-Viewport Transformation:

The transformation from window coordinates to viewport coordinates involves the following steps:

1. Scaling and Translation: Initially, you may need to scale and translate the window coordinates to match the size and position of the viewport. Scaling ensures that the graphics fit within the viewport, while translation adjusts their position.

2. Aspect Ratio Correction: If the aspect ratio of the window does not match that of the viewport, you may need to apply an aspect ratio correction to avoid distortion. This correction scales the graphics non-uniformly in the x and y directions.

3. Clipping: After the scaling, translation, and aspect ratio correction, some portions of the graphics may fall outside the viewport. Clipping is the process of discarding or truncating any parts of the graphics that are outside the viewport boundaries.

Example:

Let's say you have a window coordinate system that ranges from (-10, -10) to (10, 10), and you want to display this within a viewport of size 800x600 pixels, with the origin (0,0) at the center of the viewport.

1. Scaling and Translation: To map the window coordinates to viewport coordinates, you would need to perform both scaling and translation operations. The scaling operation would stretch the window coordinates to fit the viewport. For example, you might scale the window coordinates by a factor of 40 in both x and y directions. Additionally, you would translate the coordinates by (400, 300) pixels to place the origin at the center of the viewport.

2. Aspect Ratio Correction: If the aspect ratio of the window and viewport differs, you would apply a non-uniform scaling operation to correct the aspect ratio. This ensures that the graphics don't appear stretched or compressed in one direction.

3. Clipping: After applying scaling, translation, and aspect ratio correction, you may need to clip any portions of the graphics that fall outside the viewport boundaries. This ensures that only the visible parts are displayed on the screen.

The resulting transformation matrix represents these operations, and it can be applied to all the points, lines, or polygons in your graphical scene to map them from window coordinates to viewport coordinates, allowing you to display them accurately on the screen.

<u>MATRIX REPRESENTATION OF 3D TRANSFORMATIONS OF TRANSLATION:</u>

Matrix Representation of 3D Transformations, including translation, is a fundamental concept in computer graphics that enables the manipulation of 3D objects in a three-dimensional space. In this representation, transformations like translation, rotation, and scaling are performed using matrices, which simplifies the application of these transformations to points or objects in 3D space.

In a 3D coordinate system, translation involves moving objects or points along the x, y, and z axes. Matrix representation simplifies this process by using a 4x4 transformation matrix. This matrix can perform both translation and maintain the position in homogeneous coordinates, where the fourth coordinate (w) is usually set to 1 for finite points.

The general 3D translation matrix (T) for translating an object or point by (dx, dy, dz) units along the x, y, and z axes, respectively, is as follows:

```
T = | 1 0 0 dx |
    | 0 1 0 dy |
    | 0 0 1 dz |
    | 0 0 0  1 |
```

Here, (dx, dy, dz) represents the translation vector.

<u>Example of Matrix Representation for 3D Translation:</u>

Let's consider an example where we want to translate a 3D point P(2, 3, 4) by (dx, dy, dz) = (1, -2, 3) units.

1. Original Point: P(2, 3, 4)

2. Translation Matrix (T):

```
T = | 1 0 0 1 |
    | 0 1 0 -2 |
    | 0 0 1 3 |
    | 0 0 0 1 |
```

3. Homogeneous Coordinates: Convert the 3D point to homogeneous coordinates by adding a w-coordinate of 1.

```
P(2, 3, 4, 1)
```

4. Applying the Translation:

To apply the translation, you perform a matrix multiplication between the translation matrix (T) and the point in homogeneous coordinates (P).

```
P' = T * P
```

Perform the matrix multiplication:

```
| x' |     | 1 0 0 1 |    | 2 |
| y' |  =  | 0 1 0 -2 | * | 3 |
| z' |     | 0 0 1 3  |    | 4 |
| w' |     | 0 0 0 1  |    | 1 |
```

Simplify the multiplication:

```
x' = 3
y' = 1
z' = 7
w' = 1
```

5. Conversion from Homogeneous Coordinates:

To get the final translated point in regular 3D coordinates, divide the x', y', and z' values by w':

```
x' = 3 / 1 = 3
y' = 1 / 1 = 1
z' = 7 / 1 = 7
```

So, the translated point is P'(3, 1, 7).

SCALING:

Scaling is a transformation that changes the size of an object or image. It can be uniform, where the object is scaled uniformly in all directions, or non-uniform, where different scaling factors are applied along different axes (x, y, and z in 3D).

Uniform Scaling:

In uniform scaling, an object is resized while maintaining its shape. All dimensions are scaled by the same factor.

Example: Consider a square with side length 2 units. If you uniformly scale it by a factor of 2, the resulting square will have side length 4 units. The shape remains a square, but it's now twice as large in each dimension.

Non-Uniform Scaling:

Non-uniform scaling allows you to change the dimensions of an object independently along each axis.

Example: Imagine a rectangle with width 4 units and height 2 units. If you non-uniformly scale it to have a width of 6 units and a height of 3 units, the rectangle's proportions change, and it becomes taller and wider.

ROTATION:

Rotation is a transformation that changes the orientation or angle of an object or image. In 2D, objects are rotated counterclockwise (or clockwise with a negative angle) around a specified point, often the origin.

Example: Take a square and rotate it 45 degrees counterclockwise around its center. The result is a diamond shape with one corner pointing upwards.

<u>3D Rotation:</u>

In 3D graphics, rotation involves changing the orientation of an object in three-dimensional space. Rotations can occur around various axes (x, y, z) and at different angles.

Example: Consider a 3D cube. You can rotate it around its vertical (y) axis to make it spin. This rotation changes the cube's orientation without altering its shape or size.

UNIT-III

REPRESENTING CURVES:

Representing curves in computer graphics is essential for creating smooth and realistic shapes and designs. Curves are used extensively in graphic design, computer-aided design (CAD), animation, and various other fields to create complex shapes and paths. There are several methods for representing curves in computer graphics, each with its own advantages and use cases.

1. Parametric Curves:

Parametric curves represent points on a curve as functions of one or more parameters. The most commonly used parametric curves in computer graphics include:

- Bezier Curves: Bezier curves are defined by control points that influence the shape of the curve. The simplest form is the quadratic Bezier curve, which uses three control points, and the cubic Bezier curve, which uses four control points. Bezier curves are widely used for creating smooth paths and shapes.

- B-spline Curves: B-spline (basis spline) curves are defined by a set of control points and basis functions. They offer greater flexibility than Bezier curves and are often used in CAD and modeling applications.

- Hermite Curves: Hermite curves are defined by endpoints and tangent vectors at those endpoints. They are useful for creating curves with precise control over their direction and tangents.

- Cardinal Splines: Cardinal splines are similar to cubic Bezier curves but use a tension parameter to adjust the shape of the curve. They are useful for creating curves that interpolate a set of control points.

2. Interpolating Curves:

Interpolating curves pass through a set of given points, ensuring that the curve precisely touches those points. Some common interpolating curves include:

- Lagrange Interpolation: Lagrange interpolation constructs a polynomial that passes through a set of given points. It's a simple way to create interpolating curves, but it can be sensitive to point distribution.

- Hermite Interpolation: Hermite interpolation, also known as Hermite splines, constructs curves based on given endpoints and tangent vectors. It's useful for ensuring that the curve has specific slopes at the endpoints.

- Piecewise Linear Interpolation: This method constructs linear segments between consecutive points, creating a piecewise linear curve.

3. Non-parametric Curves:

Non-parametric curves do not rely on predefined functions or equations. Instead, they are defined by a set of discrete data points or sampled values. Examples include:

- Polygonal Chains: These are made up of line segments connecting consecutive data points. They are simple and efficient for representing curves but may not provide smoothness.

- Point Clouds: Curves can also be represented as collections of discrete points in 2D or 3D space. This representation is often used in point cloud data obtained from 3D scans.

4. Implicit Curves:

Implicit curves are defined by an equation in which the curve points are determined by satisfying the equation. For example, a circle can be represented implicitly as the equation of a circle.

5. NURBS (Non-Uniform Rational B-spline):

NURBS curves are an extension of B-spline curves and allow for greater flexibility in representing complex shapes. They are widely used in computer-aided design and modeling applications.

6. Bezier Surfaces and Splines:

Similar to Bezier curves, Bezier surfaces and splines are used to represent complex 3D shapes. They are extensions of the Bezier curve concepts into three dimensions.

7. Subdivision Curves:

Subdivision curves and surfaces use recursive subdivision algorithms to create smooth curves and surfaces by successively subdividing control points.

INTRODUCTION TO POLYGON MESHES:

A polygon mesh is a digital representation of a 3D object's surface, composed of a network of interconnected polygons. Each polygon is defined by its vertices and edges, and these polygons are stitched together to create a surface that approximates the sha pe of the object. The vertices store information about the object's position in 3D space and may also include additional data such as texture coordinates and vertex normals.

Polygon meshes are versatile and offer advantages such as efficient rendering, simplicity in computation, and ease of storage and manipulation. They are the foundation of 3D computer graphics and are used in various applications, including video games, com puter-aided design (CAD), computer-generated imagery (CGI), and virtual reality.

Types of Polygon Meshes:

There are several types of polygon meshes, depending on their structure and how the polygons are connected. The most common types include:

1. Triangular Meshes:

In a triangular mesh (also known as a triangulated mesh), every polygon is a triangle. Triangular meshes are widely used due to their simplicity and efficiency. They are the preferred choice for real-time rendering in video games and interactive applications.

2. Quadrilateral Meshes:

Quadrilateral meshes consist of polygons with four sides, typically squares or rectangles. While less common than triangular meshes, they are used in some 3D modeling applications and computer-aided design (CAD) software.

3. Mixed Meshes:

Mixed meshes combine both triangles and quadrilaterals within the same mesh. This allows for more flexibility in modeling complex shapes. Mixed meshes are used in situations where precise control over polygon placement is required.

4. Regular and Irregular Meshes:

Regular meshes have a consistent pattern of polygons, making them easy to work with and optimize for rendering. Irregular meshes have varying polygon sizes and shapes and are often used for modeling complex and organic objects.

5. Manifold and Non-Manifold Meshes:

- Manifold Meshes: Manifold meshes have well-defined topological properties. Every edge is shared by exactly two faces, and every vertex is shared by an even number of edges. Manifold meshes are suitable for most modeling and rendering tasks.

- Non-Manifold Meshes: Non-manifold meshes have more complex topologies. They can have vertices shared by an odd number of edges or edges shared by more than two faces. Non-manifold meshes are used in specialized applications, such as modeling objects with multiple materials or creating complex, interconnected shapes.

6. Regular Grids:

Regular grids are a specific type of polygon mesh in which polygons are arranged in a regular grid pattern. They are used for terrain modeling and other structured grid-based simulations.

7. Subdivision Surfaces:

Subdivision surfaces start with a coarse mesh and iteratively refine it to create smoother surfaces. They are used for modeling highly detailed and organic shapes.

8. Parametric Surfaces:

Parametric surfaces are defined mathematically using equations or functions. These surfaces are represented using polygons for rendering purposes. Examples include surfaces of revolution, Bézier patches, and NURBS surfaces.

PARAMETRIC CUBIC CURVES:

Parametric Cubic Curves are a class of curves used in computer graphics and computer-aided design (CAD) that are defined by parametric equations involving cubic polynomials. These curves are versatile and widely used due to their smoothness and ability to represent complex shapes.

Parametric cubic curves are defined by a set of parametric equations, one for each coordinate (x, y, z in 3D), and they are typically expressed in terms of a parameter t. A common representation for a cubic curve in 2D is:

```
x(t) = a*t^3 + b*t^2 + c*t + d
y(t) = e*t^3 + f*t^2 + g*t + h
```

Here, a, b, c, d, e, f, g, and h are coefficients that determine the shape of the curve. These coefficients can be adjusted to control the position, curvature, and behavior of the curve.

PARAMETRIC CONTINUITY:

Parametric continuity refers to the smoothness and continuity of the curve with respect to the parameter t. There are three levels of parametric continuity:

1. C0 (Positional Continuity): Two curves are C0 continuous if they meet at their endpoints. In other words, they share a common position at the junction point.

2. C1 (Tangential Continuity): Two curves are C1 continuous if they are C0 continuous and also have the same tangent direction at the junction point. This ensures that the curves connect smoothly without a sharp corner.

3. C2 (Curvature Continuity): Two curves are C2 continuous if they are C1 continuous and also have the same curvature at the junction point. This means that not only do they connect smoothly, but they also have the same rate of change of curvature, resulting in a smooth transition between the curves.

GEOMETRIC CONTINUITY:

Geometric continuity, often denoted as G0, G1, and G2, refers to the smoothness and continuity of the curve in the 2D or 3D space. Geometric continuity considers the physical shape and appearance of the curve, regardless of the parameterization.

1. G0 (Positional Continuity): Two curves are G0 continuous if they meet at the junction point without any gaps or overlaps in their physical appearance.

2. G1 (Tangential Continuity): Two curves are G1 continuous if they are G0 continuous and also have the same tangent direction at the junction point. This ensures a smooth visual connection between the curves.

3. G2 (Curvature Continuity): Two curves are G2 continuous if they are G1 continuous and also have the same curvature at the junction point. This provides a visually smooth transition between the curves, and there are no abrupt changes in shape or curvature.

Example:

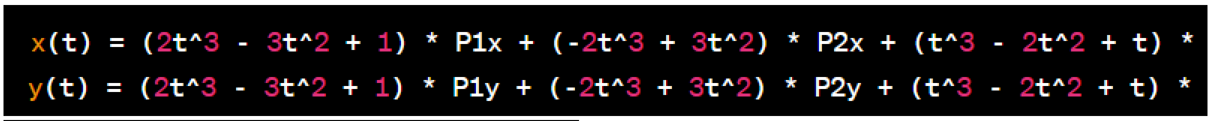
Consider a composite curve formed by connecting two parametric cubic curves. To achieve C1 parametric continuity, you need to adjust the coefficients of the two curves to ensure that they have the same tangent direction at the junction point. To achieve C2 parametric continuity, you need to ensure that they have the same curvature at the junction point.

To achieve G1 or G2 geometric continuity, you may need to adjust the coefficients further to ensure that the visual appearance of the curve is smooth at the junction point, with no visible gaps or discontinuities.

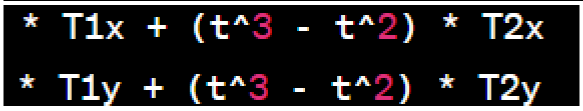
HERMITE CURVES:

Hermite curves are defined by specifying two endpoints and their tangent vectors (derivatives) at those endpoints. These tangent vectors determine the direction and rate of change of the curve at the endpoints. Hermite curves are useful when you need precise control over the shape and direction of the curve.

The general form of a 2D Hermite curve is:



```
x(t) = (2t^3 - 3t^2 + 1) * P1x + (-2t^3 + 3t^2) * P2x + (t^3 - 2t^2 + t) *
y(t) = (2t^3 - 3t^2 + 1) * P1y + (-2t^3 + 3t^2) * P2y + (t^3 - 2t^2 + t) *
```



```
* T1x + (t^3 - t^2) * T2x
* T1y + (t^3 - t^2) * T2y
```

where (P1x, P1y) and (P2x, P2y) are the endpoints, and (T1x, T1y) and (T2x, T2y) are the tangent vectors at those endpoints.

Example:

Suppose you want to create a Hermite curve that smoothly connects two points, P1(0, 0) and P2(4, 2), with tangent vectors T1(1, 1) and T2(-1, 1) at the respective endpoints. This will result in a curve that smoothly transitions between the two points with the specified tangent directions.

BEZIER CURVES:

Bezier curves are defined by a set of control points. The simplest form is the quadratic Bezier curve, which uses three control points, and the more commonly used cubic Bezier curve, which uses four control points. Bezier curves offer a high degree of flexibility and are easy to work with, making them popular in many design and modeling applications.

The general form of a cubic Bezier curve is:

```
 B(t) = (1 - t)^3 * P0 + 3 * (1 - t)^2 * t * P1 + 3 * (1 - t) * t^2 * P2 +
t^3 * P3
```
„

where P0, P1, P2, and P3 are the control points, and t varies between 0 and 1.

Example:

Consider a cubic Bezier curve with control points P0(0, 0), P1(1, 2), P2(3, 2), and P3(4, 0). This curve smoothly interpolates between P0 and P3 while being influenced by the positions of control points P1 and P2.

B-SPLINE CURVES:

B-Spline (basis spline) curves are defined by a set of control points and a set of basis functions. These curves offer greater flexibility than Bezier curves and can be used to model complex and smooth shapes. B-spline curves are often used in CAD and modeling applications.

The general form of a cubic B-spline curve is:

```
 B(t) = (1/6) * (1 - t)^3 * P0 + (1/6) * (3t^3 - 6t^2 + 4) * P1 + (1/6) *
(-3t^3 + 3t^2 + 3t + 1) * P2 + (1/6) * t^3 * P3
```

where P0, P1, P2, and P3 are control points, and t varies between 0 and 1.

Example:

Suppose you have a cubic B-spline curve with control points P0(0, 0), P1(1, 1), P2(3, 2), and P3(4, 0). This curve smoothly interpolates between the control points while being influenced by the positions of all control points.

SURFACE RENDERING:

Surface rendering is the process of generating 2D images or frames from 3D scenes or objects. It aims to create visually convincing images by simulating the interaction of light with surfaces, taking into account various factors such as lighting, shading, and materials.

Surface rendering can be broadly divided into two categories: flat shading and smooth shading.

Flat Shading: In flat shading, each polygon (usually a triangle) is treated as a single flat surface with uniform color and lighting values. This approach is computationally less expensive but results in a faceted appearance, as the shading is constant across the entire polygon.

Smooth Shading: Smooth shading, on the other hand, calculates shading values at every point on a surface, resulting in a smoother and more realistic appearance. It considers the variation in surface normals across the object's surface, leading to the simulation of curved surfaces with different lighting conditions.

BASIC ILLUMINATION:

Basic illumination models are used in surface rendering to simulate how light interacts with surfaces. These models consider several factors, including the position and intensity of light sources, the orientation of the surface, and the properties of materials.

Two fundamental basic illumination models are the Lambertian reflectance model and the Phong reflectance model.

1. Lambertian Reflectance Model:

The Lambertian reflectance model, also known as diffuse reflection, describes how light interacts with matte or non-shiny surfaces. It assumes that light is reflected uniformly in all directions. The intensity of reflected light at a point on the surface depends on the angle between the incident light direction (L) and the surface normal (N). The Lambertian reflectance model is defined by Lambert's cosine law:

```
I = Kd * L * N
```

- I: Intensity of reflected light
- Kd: Diffuse reflection coefficient (surface's reflectivity)
- L: Unit vector representing the direction of the incident light
- N: Unit vector representing the surface normal

The Lambertian model produces a soft and matte appearance, suitable for simulating materials like paper, cloth, and unpolished surfaces.

2. Phong Reflectance Model:

The Phong reflectance model is more complex and versatile than the Lambertian model. It accounts for specular reflection, which is the reflection of light in a concentrated and reflective manner, as seen on shiny surfaces. The Phong model combines three components:

- Ambient Reflection (Ka * Ia): Simulates the light that scatters in all directions due to multiple reflections in the environment.

- Diffuse Reflection (Kd * Id * max(0, N · L)): Models the Lambertian reflection for matte surfaces.

- Specular Reflection (Ks * Is * max(0, R · V)^n): Accounts for the highlight on shiny surfaces due to the reflection of light sources.

- Ka, Kd, and Ks: Coefficients controlling the intensity of ambient, diffuse, and specular reflections.
- Ia, Id, and Is: Intensities of ambient, diffuse, and specular light sources.
- N: Surface normal
- L: Unit vector representing the direction of the incident light
- R: Reflected light direction
- V: Viewer or camera direction
- n: Shininess exponent, controlling the sharpness of specular highlights

The Phong model can produce a wide range of surface appearances, from matte to highly glossy, and is commonly used for realistic rendering of various materials.

<u>EFFECT OF AMBIENT LIGHTING:</u>

Ambient lighting is a fundamental component of lighting models in computer graphics and computer-generated imagery (CGI). It represents the indirect, uniform, and omnidirectional light that objects receive from all directions in a scene. Ambient light does not have a specific source or direction but provides overall illumination, simulating the way light bounces and scatters throughout an environment.

Key characteristics and effects of ambient lighting:

1. Uniform Illumination: Ambient lighting uniformly affects all surfaces in a scene, regardless of their orientation or position relative to light sources. It helps eliminate overly dark or completely shadowed areas, ensuring that objects are visible even in low-light conditions.

2. Global Brightness: Ambient lighting contributes to the overall brightness of a scene. It sets a baseline level of illumination that serves as a foundation upon which other light sources, such as directional lights and point lights, can build.

3. Reduced Contrast: By providing a constant level of illumination from all directions, ambient lighting reduces the contrast between illuminated and shadowed areas. This can make scenes appear less realistic if used excessively, as it may result in a flat and uniform appeara nce.

4. Simulating Indirect Light: Ambient lighting approximates the indirect illumination that occurs when light bounces off surfaces and indirectly lights objects. While it doesn't model complex light interactions as accurately as global illumination techniques, it is a computationally efficient way to approximate the effect.

EFFECT OF DISTANCES:

Distances play a crucial role in computer graphics, affecting how objects are perceived in a 3D environment. Understanding distances and their impact is essential for creating realistic scenes and managing issues related to perspective and depth.

Key aspects and effects of distances in computer graphics:

1. Perspective: The perception of depth and perspective is heavily influenced by distances in a scene. Objects that are farther away appear smaller, and parallel lines seem to converge toward a common point known as the vanishing point, creating the illusion of depth.

2. Depth Cueing: Distance can be used as a depth cue in rendering to make objects in the background appear less detailed, lighter in color, or blurrier compared to closer objects. This technique enhances the perception of depth and distance in a scene.

3. Z-Buffering: In 3D rendering, a Z-buffer (or depth buffer) is used to keep track of the depth or distance of objects from the camera. This allows for proper rendering of occluded objects and ensures that objects closer to the camera appear in front of distant objects.

4. Fog and Atmospheric Effects: Simulating atmospheric effects like fog, haze, or dust in a scene can alter the perception of distances. Objects in the distance may appear faded or obscured due to these atmospheric effects.

5. Clipping: Objects that are too far from the camera may be clipped or omitted from the rendering process to optimize performance and reduce rendering artifacts.

SHADING MODELS:

Shading models, including Gouraud Shading and the Phong Shading Model, are integral components of computer graphics used to simulate the way light interacts with surfaces in 3D rendering. These models play a crucial role in creating realistic lighting and shading effects in computer-generated images.

GOURAUD SHADING:

Gouraud Shading, named after its inventor Henri Gouraud, is an interpolation-based shading model used to calculate vertex colors in a 3D scene and then interpolate these colors across the surface of polygons. It is also known as smooth shading because it aims to produce smooth transitions of color across the surface of objects.

Key Characteristics and Steps in Gouraud Shading:

1. Vertex Colors: In Gouraud Shading, the color (intensity) at each vertex of a polygon is computed. This is typically done by applying a lighting model (e.g., Phong or Lambertian reflection) to each vertex, taking into account factors like the position of the light sources, surface normals, and material properties.

2. Interpolation: Once the vertex colors are calculated, they are interpolated across the surface of the polygon using linear interpolation. The colors at points between the vertices are determined based on their proximity to the vertices.

3. Smoothing: The interpolated colors result in a smooth gradient across the polygon's surface, creating the illusion of smooth shading. This gradient represents the varying intensities of light across the surface.

4. Performance: Gouraud Shading is computationally less intensive than some other shading models (e.g., Phong shading) because it performs most of the shading calculations only at the vertices, rather than at every pixel on the surface. As a result, it is often used in real-time applications like video games.

Advantages and Limitations of Gouraud Shading:

Advantages:

- Efficiency: Gouraud Shading is computationally efficient, making it suitable for real-time rendering.
- Smooth Appearance: It produces smooth shading, providing a visually appealing result.

Limitations:

- Loss of Detail: Gouraud Shading can lose fine details in the shading, especially when polygons are large or when there are rapid changes in surface curvature.
- Specular Highlights: It may not handle specular highlights (shiny spots) as effectively as the Phong model.

PHONG SHADING MODEL:

The Phong Shading Model, developed by Bui Tuong Phong, is a more sophisticated shading model that aims to provide accurate simulations of surface reflection and shading. It is widely used for achieving realistic and detailed lighting effects, especially in 3D rendering and computer graphics applications.

Key Characteristics and Steps in the Phong Shading Model:

1. Vertex Normals: Like Gouraud Shading, the Phong model calculates the lighting at each vertex of a polygon. Additionally, it computes a normal vector (surface normal) at each vertex, which is essential for simulating accurate shading.

2. Interpolation: Rather than interpolating colors, the Phong model interpolates normals across the surface of the polygon. This ensures that the normals are smoothly interpolated, maintaining the integrity of surface details.

3. Per-Pixel Shading: Unlike Gouraud Shading, the Phong model calculates the shading of each pixel on the polygon's surface, taking into account the interpolated normals and lighting conditions. This allows for precise handling of specular highlights and other intricate shading effects.

4. Light Reflection Model: The Phong model uses the Phong reflection model to calculate the color of each pixel, considering ambient, diffuse, and specular reflection components. This model considers factors such as the position of light sources, the view direction, material properties, and the shininess of surfaces.

Advantages and Limitations of the Phong Shading Model:

Advantages:

- Realistic Highlights: The Phong model is capable of accurately simulating specular highlights and other complex lighting effects, resulting in highly realistic shading.
- Surface Details: It preserves surface details and provides a high level of visual fidelity.

Limitations:

- Computational Intensity: Phong shading is more computationally demanding compared to Gouraud shading, making it less suitable for real-time applications without optimization.
- Lack of Global Effects: It does not account for global illumination effects, such as indirect lighting and color bleeding, which require more advanced techniques like ray tracing or radiosity.

UNIT-IV

THREE-DIMENSIONAL (3D) VIEWING:

Three-Dimensional (3D) Viewing refers to the process of rendering and displaying three-dimensional objects or scenes in a way that accurately represents their spatial characteristics, depth, and perspective on a two-dimensional (2D) surface, such as a computer screen or a piece of paper. Achieving a convincing 3D view is essential in various fields, including computer graphics, computer-aided design (CAD), video games, virtual reality, and simulation.

Key Concepts in Three-Dimensional Viewing:

1. Projection: In 3D viewing, the first step is projection, which involves transforming the 3D coordinates of objects into 2D coordinates on the screen. There are two primary types of projections used:

- Perspective Projection: This type of projection simulates the way objects appear in the real world by taking into account their distance from the viewer. Objects that are farther away appear smaller, and parallel lines converge toward a vanishing point, creating a sense of depth and perspective.

- Orthographic Projection: In orthographic projection, objects are projected without considering their distance from the viewer. Parallel lines remain parallel, and there is no convergence. This projection is often used in technical drawings and architectural plans.

2. Viewing Frustum: The viewing frustum is a geometric shape that defines the region of space that is visible in the final 2D image. It resembles a pyramid with the top cut off. The near and far planes of the frustum determine the range of distances that are visible in the scene.

3. Camera Parameters: To accurately represent 3D scenes, the viewing process often involves specifying camera parameters, including the camera's position, orientation (view direction), and field of view. These parameters affect how the scene is projected onto the 2D plane.

Steps in 3D Viewing:

The process of 3D viewing typically involves several steps:

1. Modeling: Objects in the 3D scene are created and defined in a 3D coordinate system. These objects are composed of vertices, edges, and faces, and may have material properties such as colors, textures, and reflectance properties.

2. Transformation: Before rendering, objects may undergo various transformations, including translation, rotation, and scaling. These transformations position and orient objects within the 3D world.

3. Projection: Objects in the 3D scene are projected onto a 2D plane (the screen) using either perspective or orthographic projection, as determined by the chosen viewing setup.

4. Clipping: Portions of objects that fall outside the viewing frustum are clipped or removed to ensure that only visible parts are rendered.

5. Rasterization: The projected objects are converted into a series of pixels on the 2D screen. This involves determining which pixels are covered by each object and interpolating colors or textures across the pixels.

6. Depth Sorting: In scenes with multiple objects, the depth (Z-coordinate) of each pixel is used to determine the order in which objects should be drawn. This ensures that objects in the background do not obscure objects in the foreground.

7. Rendering: The final 2D image is generated by combining the colors of the objects in the correct order, taking into account lighting, shading, and material properties.

Rendering Techniques: Various rendering techniques are employed to enhance the realism of 3D views, including shading models (e.g., Phong shading), texture mapping, shadow mapping, and techniques for simulating reflections and refractions.

<u>REPRESENTATION OF THREE-DIMENSIONAL OBJECTS:</u>

Representing 3D objects involves describing their geometry and attributes in a way that allows for visualization, manipulation, and rendering.

<u>There are several common approaches to representing 3D objects:</u>

1. Wireframe Models: Wireframe models represent objects as a collection of interconnected lines or edges that outline the object's shape. These models are simple and primarily convey the object's structure without any surface or shading information. For example, a wireframe representation of a cube consists of 12 edges forming its outline.

2. Polygonal Models: In polygonal modeling, objects are approximated using flat polygons (typically triangles or quadrilaterals) that make up their surfaces. Each polygon consists of

vertices connected by edges, and the collection of these polygons defines the object's shape. For instance, a cube can be represented using six quadrilateral polygons.

3. Surface Models: Surface models provide a more detailed representation of object surfaces. They describe the geometry and attributes of the object's surfaces explicitly. These models can be created using mathematical functions or by sampling points on the object's surface to create a mesh of connected triangles.

4. Volumetric Models: Volumetric models represent objects as a collection of 3D data points within a volume. These models are often used in medical imaging and scientific simulations, where the interior properties of an object are important.

5. Point Clouds: Point cloud models consist of a set of individual points in 3D space, each with position and sometimes additional attributes like color or intensity. These models are used in laser scanning, LiDAR, and 3D reconstruction from depth sensors.

PROJECTIONS:

Projections are methods for transforming 3D objects or scenes into 2D representations. They are crucial for rendering 3D graphics on 2D screens or surfaces.

There are two primary types of projections: perspective projection and orthographic projection.

1. Perspective Projection:

Perspective projection simulates the way objects appear in the real world, considering their distance from the viewer. It creates a sense of depth and perspective by making objects that are farther away appear smaller and converge toward a vanishing point. Perspective projection is commonly used for realistic rendering.

The perspective projection of a 3D point (X, Y, Z) onto a 2D plane (X', Y') is given by:

```
X' = X * (focal_length / Z)
Y' = Y * (focal_length / Z)
```

Where:

- (X, Y, Z): 3D coordinates of the point.

- (X', Y'): 2D coordinates of the projected point.

- “focal_length”: The focal length of the virtual camera.

Example: Imagine a 3D scene with a cube placed at varying distances from the camera. When using perspective projection, the cube's size in the 2D image decreases as it moves farther from the camera, creating the illusion of depth.

2. Orthographic Projection:

Orthographic projection projects objects onto a 2D plane without considering their distance from the viewer. In this projection, parallel lines in the 3D scene remain parallel in the 2D image, and there is no convergence. Orthographic projection is often used in engineering drawings, architectural plans, and some 3D modeling applications.

The orthographic projection of a 3D point (X, Y, Z) onto a 2D plane (X', Y') is given by:

```
X' = X
Y' = Y
```

Where:

- (X, Y, Z): 3D coordinates of the point.

- (X', Y'): 2D coordinates of the projected point.

Example: When using orthographic projection, a cube placed at varying distances from the camera appears the same size in the 2D image, regardless of its depth, as parallel lines do not converge.

<u>**ORTHOGRAPHIC PROJECTIONS (PARALLEL PROJECTION):**</u>

Orthographic projections are a type of parallel projection that projects 3D objects onto a 2D plane without accounting for the distance between objects and the viewer. In orthographic projections, parallel lines in the 3D space remain parallel in the 2D projection, and there is no perspective or foreshortening.

Mathematical Representation:

The orthographic projection of a 3D point (X, Y, Z) onto a 2D plane (X', Y') is given by:

```
X' = X
Y' = Y
```

- (X, Y, Z): 3D coordinates of the point.

- (X', Y'): 2D coordinates of the projected point.

Characteristics:

- Parallel lines remain parallel in the projection.

- Objects at varying distances from the viewer have the same size in the projection.

Example:

Imagine you have a 3D cube with its faces parallel to the coordinate axes. When projected orthographically onto a 2D plane, all the lines that are parallel in 3D space, such as the edges of the cube, remain parallel in the 2D projection. The cube appears exactly the same size, regardless of its depth in 3D space.

OBLIQUE PROJECTIONS (PARALLEL PROJECTION):

Oblique projections are a subset of parallel projections where objects are projected onto a 2D plane at an angle other than 90 degrees. Unlike orthographic projections, oblique projections introduce an element of perspective and foreshortening.

Mathematical Representation:

The oblique projection of a 3D point (X, Y, Z) onto a 2D plane (X', Y') can be represented using a matrix transformation:

```
X' = X + a * Z
Y' = Y + b * Z
```

- (X, Y, Z): 3D coordinates of the point.

- (X', Y'): 2D coordinates of the projected point.

- 'a' and 'b' are coefficients that control the obliqueness of the projection.

Characteristics:

- Parallel lines in the 3D space may or may not remain parallel in the projection, depending on the chosen oblique angle and coefficients 'a' and 'b.'

- Objects may experience perspective and foreshortening, making them appear differently sized based on their distance from the viewer.

Example:

Suppose you have a 3D rectangular box (e.g., a TV) oriented at an angle to the projection plane. When projected using an oblique projection, the lines representing the edges of the box may no longer be parallel in the 2D projection, and the box's dimensions may appear distorted based on the chosen coefficients 'a' and 'b.'

PERSPECTIVE PROJECTION:

Perspective projection simulates the way objects appear in the real world by accounting for their distance from the viewer. In perspective projection, objects farther from the viewer appear smaller, and parallel lines converge toward a vanishing point, creating a sense of depth and perspective.

Mathematical Representation:

The perspective projection of a 3D point (X, Y, Z) onto a 2D plane (X', Y') is given by:

```
X' = X * (focal_length / Z)
Y' = Y * (focal_length / Z)
```

- (X, Y, Z): 3D coordinates of the point.

- (X', Y'): 2D coordinates of the projected point.

- “focal_length”: The focal length of the virtual camera.

Characteristics:

- Parallel lines in the 3D space converge in the projection, creating a sense of depth and perspective.

- Objects appear smaller as they move farther from the viewer.

Example:

Imagine you have a 3D scene with a road that extends into the distance. When projected using perspective projection, the road's edges appear to converge toward a vanishing point on the horizon, creating the illusion of depth and distance.

HIDDEN SURFACE REMOVAL:

Hidden Surface Removal is a crucial step in computer graphics that ensures that only the visible surfaces of 3D objects are rendered, while occluded or hidden surfaces are correctly discarded. Two common methods for hidden surface removal are the Depth-Buffer (Z-Buffer) Method and the Depth-Sorting Method (Painter's Algorithm).

1. Depth-Buffer (Z-Buffer) Method:

The Depth-Buffer Method, also known as the Z-Buffer Method, is a hardware-based or software-based technique used to determine which surfaces are visible in a 3D scene. It employs a data structure called the Z-buffer, which is an image-sized array that stores the depth (Z-coordinate) value for each pixel in the frame buffer.

How the Depth-Buffer Method Works:

1. Initialization: Initialize the Z-buffer with a large value (typically the farthest depth value).

2. Rendering: For each pixel on the screen, calculate the depth (Z-coordinate) of the corresponding 3D point using the projection matrix and the depth of the object. Compare this depth value with the current Z-buffer value for that pixel.

3. Depth Testing: If the calculated depth is less than the value stored in the Z-buffer, update the Z-buffer with the new depth value and color the pixel with the color of the object. Otherwise, discard the pixel.

Example:

Consider a simple 3D scene with two cubes—one in front of the other. The Depth-Buffer Method ensures that only the visible cube is rendered in the final image.

- When rendering the front cube, the depth values for each pixel are compared to the Z-buffer. Pixels corresponding to the front cube are drawn, and their depth values are updated in the Z-buffer.

- When rendering the rear cube, some pixels may be discarded because their depth values are greater (farther away) than the values in the Z-buffer, indicating that they are occluded by the front cube.

Advantages:

- Efficient and suitable for real-time rendering.

- Handles complex scenes with overlapping objects.

Limitations:

- Requires additional memory for the Z-buffer.

- May not handle transparent or semitransparent objects well without additional techniques.

<u>2. Depth-Sorting Method (Painter's Algorithm):</u>

The Depth-Sorting Method, often referred to as Painter's Algorithm, is a software-based approach that involves sorting objects or polygons in a 3D scene based on their depth (Z-coordinate) from the viewer's perspective. Objects are rendered in back-to-front order to ensure that nearer objects occlude farther ones.

How the Depth-Sorting Method Works:

1. Depth Calculation: Calculate the depth (Z-coordinate) of each object's centroid or reference point in the scene.

2. Sorting: Sort the objects in the scene based on their calculated depths, from farthest to nearest.

3. Rendering: Render the objects in the sorted order, starting with the farthest object and progressing to the nearest one.

Example:

Imagine a 3D scene with multiple objects—a tree, a house, and a mountain. Using the Depth-Sorting Method, the objects are sorted based on their depth values. The mountain, being farthest, is rendered first, followed by the house and the tree. This ensures that objects are drawn in the correct overlapping order.

Advantages:

- Can handle complex scenes with varying levels of depth complexity.
- Provides accurate control over rendering order.

Limitations:

- Sorting objects can be computationally expensive, especially in scenes with numerous objects.
- May not handle intersecting or transparent objects well without additional techniques.