Assignmen-1

Date
Page 01

Q-1 List and describe the basic graphics primitives. How are they fundamental to computer graphics?

Ans. Graphics primitives are the fundamental building blocks used to create complex images, shapes, and graphical content in computer graphics. These primitives represent the simplest geometric forms and are essential for rendering all kinds of visual elements in software and hardware graphics systems. Below is a list of the most common graphics primitives along with detailed descriptions of their roles in computer graphics:

1.] Point

- Definition : A point is a single pixel or dot in the coordinate space defined by (x, y) in 2D or (x, y, z) in 3D.
- Usage : Points are used to mark locations, vertices of shapes, and as reference coordinates in various applications.
- Significance : They are the simplest form of graphics primitives and serve as building blocks for more complex shapes.

2.] Line

- Definition : A line is defined by two endpoints, (x1, y1) and (x2, y2), in 2D space

- Usage : Lines are essential for drawing contours, ~~and~~ wireframes, and outlining shapes in graphics applications.

- Significance : They form the foundation for creating edges in polygons and polyline shapes.

* Fundamental Importance in Compute Graphics :

1. Building Complex Shapes :
-> All complex images and models are composed of basic primitives.

2. Efficient Rendering :
-> Primitives form the basis of efficient rendering techniques in graphics pipelines.

3. Hardware Optimization:
-> Graphics hardware (GPUs) is optimized to process and render these primitives efficiently.

4. Mathematical Simplicity:
-> They have straightforward mathematical representations, which are essential for transformations, scaling, and other graphics operations.

5. Interactivity:
-> Primitives provide essential data for detecting collisions and interactions in graphical applications.

Date
Page 05

Q-2 Explain the differences between a raster-scan system and a random-scan system. Provide examples to illustrate their working principles.

Aspect	Raster-Scan System	Random-Scan System
Basic Principle	Scans the entire screen row by row to generate the display	Draws graphics directly by tracing line segments between specified points.
Display Element	Works with pixels	Works with geometric primitives like lines and curves
Image Representation	Images are stored as bitmaps (pixel arrays).	Images are stored as a set of drawing instructions

Screen Refresh	Refreshes the entire screen at a fixed rate	Refreshes only the necessary lines or primitives
Image Quality	Better for complex and shaded images, but may suffer from jagged lines	Better for simple line drawings with sharp and smooth edges.
Complexity	Easier to implement and widely used in modern displays	More complex to implement; used for specialized applications.
Speed	May be slower for drawing geometric primitives due to pixel-by-pixel updates	Faster for line drawings as it directly traces vector paths.

Date
Page 07

* Working Principles with Examples.

1. Raster - Scan System.

- How It Works: The screen is divided into a grid of pixels. The system sequentially updates each pixel from left to right, row by row, to form the image.

- Example: Modern computer monitors, televisions, and smartphones use raster-scan techniques.

- Illustration: Consider a photo viewer application. When you load a high-resolution photo, the raster system updates each pixel to render the entire image on the screen.

Date
Page 08

2. Random-Scan System

- **How It Works**: The electron ~~br~~beam moves only to the required points and traces the geometric primitives directly without scanning all pixels.
- **Example**: Early oscilloscopes and vector-based arcade games
- **Illustration**: In a CAD software for architectural design, vector graphics allow ~~pro~~precise line drawings without rendering the entire canvas pixel by pixel.

Q-3 Write and explain the steps of the Bresenham's line drawing algorithm for the line drawing. Illustrate its applications with an example for a line between two given points.

Ans. Bresenham's algorithm efficiently draws a straight line between two points in a raster grid (like on a screen) by calculating the next pixel based on decision parameters, avoiding floating-point calculations.

* Steps :-

1. Initialize : Start with the two endpoints (x_0, y_0) and (x_1, y_1).

- Calculate differences : $dx = x_1 - x_0$,
  $dy = y_1 - y_0$

2. Decision Parameter : Compute the initial decision parameter :

$$p_0 = 2dy - dx$$

3. Iterate : For each step from (x_0, y_0) to (x_1, y_1):

- If $p_k < 0$, move horizontally $(x+1)$.
- If $p_k \geq 0$, move diagonally $(x+1, y+1)$.
- Update the decision parameter.

4. Repeat until the endpoint (x_1, y_1) is reached.

* Example :

For points (2,3) to (10,7)

Initialize.

- $x_0 = 2$, $y_0 = 3$
- $x_1 = 10$, $y_1 = 7$

- $dx = x_1 - x_0 = 10 - 2 = 8$
- $dy = y_1 - y_0 = 7 - 3 = 4$

$p_0 = 2dy - dx = 2(4) - 8 = 8 - 8 = 0$

Iterate and Decide :-

- Point (2, 3) : $p_0 = 0$, move horizontally to (3, 3)
- Point (3, 3) : $p_1 = 8$, move diagonally to (4, 4)
- Point (4, 4) : $p_2 = 0$, move horizontally to (5, 4)
- Point (5, 4) : $p_3 = 8$, move diagonally to (6, 5)
- Point (6, 5) : $p_4 = 0$, move horizontally to (7, 5)
- Point (7, 5) : $p_5 = 8$, move diagonally to (8, 6)
- Point (8, 6) : $p_6 = 0$, move horizontally to (9, 6)
- Point (9, 6) : $p_7 = 8$, move diagonally to (10, 7)

* Applications :

- Used in computer graphics to draw lines.
- Essential for scan conversion in raster graphics.

Q-4 Explain the difference between scan-line, boundary-fill, and flood-fill algorithms. Provides scenarios where each algorithm would be most suitable.

1. Scan-line Fill :-

How it works :

- The scan-line algorithm works by moving horizontally across the screen.
- It fills an enclosed area by checking each horizontal line and determining where the line intersects the boundary of the shape.
- It then fills the region between the intersection points for each scan line.

Applications :

- Suitable for polygon filling, especially for convex polygons or complex shapes.
- It's efficient for large and complex shapes because it operates line by line, filling areas iteratively from top to bottom.

Scenario :

- Filling a complex, non-convex polygon, such as a 2D terrain or irregular objects, where traditional flood fill methods might be slow.

2. Boundary-fill :-

How it works :-

- The boundary-fill algorithm fills an area by starting from a seed point inside the enclosed region and expanding outward.
- It works by filling pixels starting from the seed and checking each adjacent pixel. If the adjacent pixel is not part of the boundary, it gets filled.
- It stops when it encounters a boundary color or a predetermined boundary limit.

Applications :

- Best suited for simple shapes, objects with well-defined boundaries
- It's not ideal for large, complex regions because it can be slow, particularly with irregular or large boundaries.

Scenario :

- Filling a small enclosed area like the interior of a button or any simple shape where the boundary is well defined, such as a circle inside a rectangle.

3. Flood-fill :-

How it works :

- Flood-fill is similar to the boundary-fill algorithm, but instead of checking for a boundary color, it fills all pixels that match the initial "seed" color.

- It uses a queue or recursive approach to check and fill all adjacent pixels that have the same color as the starting point, expanding outward untill it reaches a different color.

Applications :

- Used for region filling in more general cases, especially when the boundaries are not clearly defined or not easily specified.
- Effective for irregular shapes and large regions where boundaries may not be strictly constrained.

Scenario :

- Filling regions in image editing software, like coloring an area in a photo that may have irregular edges or undefined boundaries.

Q-5. Write the steps to implement a boundary-fill algorithm. Provide an example where it ~~can~~ can be applied to fill a specific region in a graphical environment.

Ans.

Scenario: You want to fill the interior of a circle with a blue color while keeping the black boundary intact.

Steps:

1. Initialize:

- Starting point: Choose a point inside the circle, e.g., (5,5)
- Boundary color: The color of the circle's boundary. (e.g., black)
- Fill color: The color you want to fill the interior of the circle with (e.g., blue).

2. Check Boundary Conditions:-

- If the pixel at the current point is the boundary color (black) or out of bounds, stop the algorithm for that pixel.

3. Fill the Pixel:-

- If the current pixel is not the boundary color (black) or already filled with the fill color (blue), color the pixel with the fill color (blue).

4. Expand Recursively:

- For the current pixel, check the 4 neighboring pixels
- If any of the neighboring pixels are not boundary color (black), repeat the filling process for them.

5. Repeat

- Continue recursively filling neighbouring pixels untils the entire enclose area is filled with blue and the boundary (black) remains intact.

* Example:

Step 1: Start at (5, 5), inside the circle.
- The pixel is not black, so it is filled with blue.

Step 2: Check the 4 neighboring pixels of (5,5):
- (6, 5), (4, 5), (5, 6), (5, 4) - none of them are black, so fill them with blue.

Step 3: Continue expanding by checking the 4 neighbors of each newly filled pixel.
- (6, 5) has neighbors (7,5), (5,5), (6,6), (6,4). None are black, so fill them with blue.

Step 4: Continue this process untill the entire interior of the circle is filled.

Final Result:

- The area inside the circle will be blue, and the black boundary will remain intact, marking the edge of the filled region.

Q-6 List and describe the different two-dimensional geometric transformations. Provide brief examples for each transformation.

Ans.

1. Translation

- Description : Moves an object by a specified distance.
- Formula : $(x', y') = (x + t_x, y + t_y)$
- Example : Move point (2, 3) by (3, -2) results in (5, 1).

2. Scaling

- Description : Changes the size of an object
- Formula : $(x', y') = (x \cdot s_x, y \cdot s_y)$
- Example : Scale point (2, 3) by 2 results in (4, 6).

3. Rotation

- Description: Rotates an object by an angle around the origin.
- Formula : $(x', y') = (x \cdot \cos(\theta) - y \cdot \sin(\theta), x \cdot \sin(\theta) + y \cdot \cos(\theta))$
- Example : Rotate $(1,0)$ by $90°$ results in $(0,1)$.

4. Shearing

- Description: Slants an object along an axis.
- Formula : $(x', y') = (x + sh_x \cdot y, y)$
- Example : Shear $(1,2)$ along the X-axis by a factor of 2 results in $(5,2)$.

5. Reflection

Description : Flips an object over an axis.

Formula : Reflection across Y-axis :

$$(x', y') = (-x, y)$$

Example : Reflect $(3, 4)$ across the Y-axis results in $(-3, 4)$.