**Code No: 155AM** **R18**

# JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY HYDERABAD

**B. Tech III Year I Semester Examinations, January/February - 2023**

**COMPUTER GRAPHICS**

**(Common to CSE, IT, CSIT, CSE(AIML), CSE(DS))**

**Time: 3 Hours** **Max.Marks:75**

**Note:** i) Question paper consists of Part A, Part B.
ii) Part A is compulsory, which carries 25 marks. In Part A, Answer all questions.
iii) In Part B, Answer any one question from each unit. Each question carries 10 marks and may have a, b as sub questions.

## PART - A

**(25 Marks)**

1. a) Explain applications for large screen displays. [2]
b) Write a short note on video-display device. [3]
c) What is viewing functions? [2]
d) Explain the 2 D transformation matrix for Translation. [3]
e) What is the role of parametric functions in curve generation? [2]
f) Write an algorithm for the generation of B-spline? [3]
g) Derive the transformation matrix for rotation about y-axis in 3D. [2]
h) Derive the matrix form for the Translation operation in 3-D graphics. [3]
i) Write about depth-sort algorithm. [2]
j) What are the steps in design of animation sequence? [3]

## PART - B

**(50 Marks)**

2. a) What are the steps involved in DDA algorithm for line drawing.
b) Write a short note on boundary-fill algorithm. [5+5]

**OR**

3. a) Briefly explain about mid-point ellipse algorithms with example.
b) Discuss about raster-scan systems. [5+5]

4. a) Describe the Cohen-Sutherland algorithm.
b) What is reflection? Discuss with example? [5+5]

**OR**

5. a) Explain the stages in viewing pipeline in 2-D graphics.
b) Derive mathematically the transformation that rotates an object point $0^0$ anti-clockwise about the origin. What the matrix representation for this rotation. [5+5]

6. a) Write a short note on Hermite curve.
b) Discuss about quadric surfaces. [5+5]

**OR**

7. a) Write a short note on Bezier curve.
b) Discuss about polygon rendering methods. [5+5]

8. a) Derive the matrix form for Rotation in 3-D graphics.
b) Explain about the approaches followed for clipping in 3-D space. [5+5]

**OR**

9. a) Briefly explain about 3-D composite transformations.
b) Write a short note on shear transformations in 3-D. [5+5]

10. a) Discuss about the graphical languages followed to achieve animation.
b) Explain in detail about depth-buffer algorithm. [5+5]

**OR**

11. a) Describe linear list notation of animation languages.
b) Write a short note on BSP-trees. [5+5]

**---ooOoo---**

## ANSWER KEY

### PART - A

**1. a) Explain applications for large screen displays.**

Large screen displays are used in various applications such as digital billboards, stadium scoreboards, conference rooms, control rooms, and public information displays. They are also utilized in retail stores for advertising, in airports and train stations for flight and train schedules, and in classrooms and auditoriums for presentations and lectures.

**b) Write a short note on video-display device.**

A video-display device is an electronic device that displays visual information. Common types include CRTs, LCDs, LEDs, and plasma screens. These devices convert electrical signals into visible images and are used in televisions, computer monitors, digital signage, and video walls. They are essential for visual communication, entertainment, and information dissemination.

**c) What is viewing functions?**

Viewing functions define how a scene is projected onto a display screen. They involve transformations that map 3D objects to 2D screen coordinates, including perspective and orthographic projections. These functions control the viewing volume, the position, and the orientation of the camera, determining how the final image is rendered.

**d) Explain the 2 D transformation matrix for Translation.**

2D Transformation Matrix for Translation: Translation in 2D involves shifting a point (x, y) by a distance (tx, ty). The transformation matrix for translation is:

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

Applying this matrix to a point [x y 1] translates it to [x + tx, y + ty, 1].

**e) What is the role of parametric functions in curve generation?**

Role of Parametric Functions in Curve Generation: Parametric functions describe curves by expressing the coordinates of the points as functions of a parameter, usually denoted as t. This allows for flexible and precise representation of complex shapes and curves, such as Bezier curves and B-splines, which are widely used in computer graphics for modeling and animation.

**f) Write an algorithm for the generation of B-spline?**

Algorithm for Generation of B-spline:

1. Define the control points.
2. Choose the degree of the B-spline.
3. Create a knot vector.
4. For each value of the parameter t:

Calculate the basis functions.

Compute the corresponding point on the B-spline curve using the basis functions and control points.

5. Plot the computed points to form the B-spline curve.

**g) Derive the transformation matrix for rotation about y-axis in 3D.**

Transformation Matrix for Rotation about Y-axis in 3D:

The matrix for rotating a point around the y-axis by an angle θ is:

$$\begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

**h) Derive the matrix form for the Translation operation in 3-D graphics.**

Matrix Form for Translation Operation in 3D Graphics:

The matrix for translating a point (x, y, z) by distances (tx, ty, tz) in 3D is:

$$\begin{bmatrix} 1 & 0 & 0 & tx \\ 0 & 1 & 0 & ty \\ 0 & 0 & 1 & tz \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Applying this matrix to a point [x y z 1] translates it to [x + tx, y + ty, z + tz, 1].

**i) Write about depth-sort algorithm.**

Depth-Sort Algorithm: The depth-sort algorithm, also known as the painter's algorithm, is used in computer graphics to handle visibility of overlapping objects. Steps include:

1. Sort all polygons by their depth (z-values) from the viewer.
2. Resolve any depth conflicts.
3. Render polygons in the sorted order, from farthest to nearest.

This ensures proper occlusion and visibility in 3D scenes.

**j) What are the steps in design of animation sequence?**

Steps in Design of Animation Sequence:

1. Concept Development: Define the storyline, characters, and overall visual style.
2. Storyboard Creation: Sketch key frames and sequences to outline the animation.
3. Design and Modeling: Create detailed designs and 3D models of characters and environments.
4. Animation: Develop the motion sequences using keyframing or motion capture.
5. Rendering: Generate the final frames or scenes with appropriate lighting and textures.
6. Post-Production: Add sound effects, music, and any additional visual effects.
7. Final Review and Editing: Make necessary adjustments and compile the final animation.

## PART - B

**2. a) What are the steps involved in DDA algorithm for line drawing.**

Steps Involved in DDA Algorithm for Line Drawing:

1. Initialize Variables: Set the starting point (x0, y0) and the ending point (x1, y1) of the line. Calculate the differences dx = x1 - x0 and dy = y1 - y0.
2. Calculate Steps: Determine the number of steps needed to draw the line. This is the maximum absolute difference between dx and dy: steps = max(|dx|, |dy|).
3. Compute Increment Values: Calculate the increment values for x and y

coordinates for each step. These are computed as:
x_inc = dx/steps, y_inc = dy/steps
4. Set Initial Point: Initialize the starting point as the first point to be plotted: x = x0, y = y0.
5. Plot the Initial Point: Plot the first point (round(x), round(y)).
6. Iterate Over Steps: Loop through the number of steps calculated.
For each step, increment the x and y values by x_inc and y_inc, respectively.
Plot the new point (round(x), round(y)).
7. Repeat: Continue the iteration until all steps are completed, ensuring that the line is drawn from the starting to the ending point.
This algorithm is efficient for drawing lines as it avoids floating-point operations and uses only incremental integer calculations.

**b) Write a short note on boundary-fill algorithm.**
The boundary-fill algorithm is used in computer graphics to fill a closed area with a particular color. It works by starting at a seed point inside the area and spreading outwards until it encounters a boundary color. Here are the key steps:
1. Identify Seed Point: Choose a starting point (seed) inside the boundary to begin filling.
2. Check Boundary Condition: Ensure that the seed point is not on the boundary and is within the area to be filled.
3. Fill Algorithm: Recursively or iteratively color the pixel at the seed point with the fill color if it is not the boundary color and not already filled.
4. Recursive Process: Move to adjacent pixels (left, right, top, bottom, and diagonally) and repeat the fill process for each.
5. Edge Detection: Stop filling when a pixel with the boundary color or the fill color is encountered, ensuring the fill does not go beyond the desired region.
6. Efficiency: The recursive method can be memory-intensive due to stack depth, so an iterative approach using a stack or queue is often preferred to avoid overflow issues.
7. Applications: Boundary-fill is commonly used in areas like painting programs, where a user can click a region to fill it with a chosen color.
This algorithm is effective for regions with well-defined boundaries and is simple to implement, making it a staple in basic computer graphics operations.

**3. a) Briefly explain about mid-point ellipse algorithms with example.**
The mid-point ellipse algorithm is used to draw an ellipse by determining the points in the ellipse's boundary using a midpoint approach, which involves calculating the midpoint between pixels to decide the next point to plot. Here are the steps:
1. Initialization: Define the ellipse's parameters, including its radii (Rx, Ry) and center (xc, yc).

2. Starting Point: Start from the topmost point of the ellipse (0, Ry).
3. Region 1 (initial region):
Calculate the initial decision parameter p1 = Ry^2 - (Rx^2 * Ry) + ¼Rx^2 .
Move in the x-direction while checking the decision parameter to determine the next point.
Update the decision parameter \( p1 \) accordingly and plot the symmetric points.
4. Transition to Region 2:
Switch to the second region when the slope becomes greater than or equal to -1.
5. Region 2 (remaining region):
Calculate the decision parameter p2 = Ry^2 (x + (½))^2 + Rx^2 (y-1)^2 - Rx^2 * Ry^2 .
Move in the y-direction while updating the decision parameter and plotting the symmetric points.
6. Symmetry: Plot points in all four quadrants by mirroring the calculated points.
Example:
For an ellipse centered at (0, 0) with radii Rx = 5 and Ry = 3:
Start at (0, 3) and calculate the initial decision parameter \( p1 \).
Use the decision parameters to move horizontally until transition to Region 2.
Then, use the second decision parameter \( p2 \) to move vertically, plotting symmetric points around the center until the ellipse is complete.

**b) Discuss About raster-scan systems.**
Raster-scan systems are the most common method for displaying images on screens, such as in CRTs, LCDs, and LED monitors. Here are the key points:
1. Definition: A raster-scan system displays images by illuminating pixels in a systematic sequence, row by row, from top to bottom and then returning to the top (known as scanning).
2. Frame Buffer: The image to be displayed is stored in a frame buffer, a memory area where each pixel's color and intensity are recorded.
3. Scanning Process: The electron beam (in CRT) or pixel matrix (in modern displays) moves across each row (scanline), illuminating pixels according to the data in the frame buffer.
4. Refresh Rate: This process repeats many times per second (typically 60-120 Hz) to create a stable image without flickering.
5. Resolution: The resolution of a raster-scan system is determined by the number of pixels in the horizontal and vertical directions (e.g., 1920x1080).
6. Color Depth: Each pixel can display a range of colors, determined by the color depth (e.g., 24-bit color depth allows 16.7 million colors).
7. Interlaced and Progressive Scanning: Older systems used interlaced scanning, where even and odd lines are refreshed alternately. Modern systems use progressive scanning, where all lines are refreshed in sequence.

8. Applications: Raster-scan systems are used in televisions, computer monitors, and any display devices that require the rendering of images and videos.
This method is efficient and versatile, allowing for the display of complex images and videos with a wide range of colors and resolutions.

## 4. a) Describe the Cohen-Sutherland algorithm.

The Cohen-Sutherland algorithm is used for line clipping in computer graphics, specifically to determine which portions of a line segment are visible within a rectangular clipping window. The algorithm works as follows:
1. Define Clipping Window: Specify the coordinates of the rectangular clipping window (xmin, ymin, xmax, ymax).
2. Assign Region Codes: Each endpoint of the line is assigned a 4-bit region code (or outcode) based on its position relative to the clipping window. The bits represent the top, bottom, right, and left regions.
3. Initial Check: If both endpoints of the line have a region code of 0000, the line is entirely inside the clipping window and is accepted.
4. Trivial Rejection: If the logical AND of the region codes of the two endpoints is not 0000, the line is entirely outside the clipping window and is rejected.
5. Clipping Process:
If the line is partially inside, the algorithm iteratively clips the line segment by intersecting it with the clipping window edges (left, right, top, bottom) based on the non-zero bits of the region codes.
Update the region codes and endpoints accordingly.
Repeat until the line is either accepted or rejected.
Example: For a line segment with endpoints (x1, y1) and (x2, y2):
Assign region codes based on the clipping window.
Perform clipping calculations if needed, adjusting the endpoints to fit within the clipping window.
The resulting segment within the window is the visible portion of the line.

## b) What is reflection? Discuss with example?

Reflection in computer graphics is a transformation that produces a mirror image of an object relative to a specified line or plane. It is used to create symmetrical shapes or simulate mirror effects. The process involves flipping the object's coordinates across the reflection axis.
Example:
1. Reflection Across the X-axis: To reflect a point (x, y) across the x-axis, the y-coordinate is negated, resulting in (x, -y). The transformation matrix is:

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

2. Reflection Across the Y-axis: To reflect a point (x, y) across the y-axis, the x-coordinate is negated, resulting in (-x, y). The transformation matrix is:

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

3. Reflection Across the Origin: To reflect a point (x, y) across the origin, both coordinates are negated, resulting in (-x, -y). The transformation matrix is:

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

4. Reflection Across an Arbitrary Line: To reflect across a line y = mx + c, a series of transformations is applied: translating the line to the origin, rotating the line to align with the axis, performing the reflection, and applying the inverse transformations.
Example Application:
If a triangle with vertices at (1, 2), (3, 4), and (5, 6) is reflected across the x-axis, the new vertices would be (1, -2), (3, -4), and (5, -6), creating a mirror image below the x-axis.
Reflection is a fundamental operation in graphics used to create symmetry, design patterns, and mirror effects in various applications.

**5. a) Explain the stages in viewing pipeline in 2-D graphics.**
a) Stages in Viewing Pipeline in 2-D Graphics:
The viewing pipeline in 2-D graphics involves several stages to transform and project objects onto a 2-D display. Here are the key stages:
1. Modeling Transformation: Transform objects from their local coordinates to world coordinates. This includes scaling, rotation, and translation to place objects in the correct position and orientation in the scene.
2. World Coordinate System: Define the scene using a world coordinate system, which serves as a reference for all objects.

3. Viewing Transformation: Convert world coordinates to viewing coordinates based on the position and orientation of the camera or viewing window.
4. Clipping: Clip the objects against a predefined clipping window. Only the parts of objects within the clipping window are retained for further processing, removing any portions outside the viewable area.
5. Window-to-Viewport Transformation: Map the clipped objects from the window (viewing coordinates) to the viewport (device coordinates). This involves scaling and translating to fit the objects within the display area.
6. Projection: Project the transformed coordinates onto the 2-D view plane. In 2-D graphics, this step is straightforward as it usually involves direct mapping.
7. Device Coordinates: Finally, the coordinates are converted to device-specific coordinates (pixel coordinates) for rendering on the screen.
This pipeline ensures that objects are correctly transformed, clipped, and projected for accurate display on the 2-D screen.

**b) Derive mathematically the transformation that rotates an object point $0^0$ anti-clockwise about the origin. What the matrix representation for this rotation.**

Mathematically Deriving the Transformation for Rotation:
To rotate an object point (x, y) counterclockwise by an angle θ about the origin, the following steps are performed:
1. Rotation Formulas: The new coordinates (x', y') after rotation are given by:
x' = xcosθ - ysinθ]
y' = xsinθ + ycosθ]
2. Matrix Representation: The rotation can be represented using a transformation matrix. The point (x, y) can be expressed in homogeneous coordinates as [x y 1]. The rotation matrix for counterclockwise rotation by θ is:

$$\begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

3. Applying the Matrix: Multiplying the point [x y 1] by the rotation matrix:

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

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

4. Result: The resulting coordinates (x', y') are the new position of the point after rotation.
This matrix representation simplifies the rotation operation, allowing it to be easily combined with other transformations such as translation and scaling.

**6. a) Write a short note on Hermite curve.**
A Hermite curve is a type of cubic polynomial curve commonly used in computer graphics and geometric modeling. It is characterized by its smooth transitions between points, which are controlled by the curve's endpoints and the tangents at these points.
1. Definition: A Hermite curve is defined by two endpoints and the tangents (direction and steepness) at those endpoints.
2. Control: The shape of the Hermite curve is influenced by adjusting these tangents, allowing for precise control over the curve's shape.
3. Continuity: Hermite curves are designed to be C1 continuous, ensuring smooth transitions between connected segments.
4. Applications: They are used in animation paths, modeling complex shapes, and CAD systems for creating smooth and flexible curves.
5. Flexibility: The ability to adjust tangents provides great flexibility in designing curves that fit specific requirements.
6. Usage: Designers and engineers utilize Hermite curves for tasks that require smooth and easily controllable curves.

**b) Discuss about quadric surfaces.**

Quadric surfaces are a family of surfaces in three-dimensional space, characterized by their smooth and continuous shapes. They are defined by second-degree polynomial equations and encompass a variety of geometric shapes.

1. Types: Common types of quadric surfaces include ellipsoids, hyperboloids, paraboloids, and cones. Each type has unique geometric properties.
2. Ellipsoid: Resembles an elongated or flattened sphere, often used to model objects like eggs or planets.
3. Hyperboloid: Can have one or two sheets, with shapes that appear like a saddle or an hourglass.
4. Paraboloid: Can be either elliptic or hyperbolic, used in applications like satellite dishes and reflectors.
5. Cone: A surface that extends infinitely in both directions, forming a shape like an ice cream cone or a megaphone.
6. Applications: Widely used in computer graphics, CAD systems, and physical simulations to model natural and man-made objects.
7. Geometric Properties: Quadric surfaces exhibit various symmetries and can be visualized by plotting their equations or using parametric representations.
These surfaces are integral to geometric modeling and computer graphics, offering a wide range of shapes for creating detailed and accurate 3D models.

**7. a) Write a short note on Bezier curve.**
A Bezier curve is a parametric curve used extensively in computer graphics and related fields for modeling smooth curves. They are particularly popular due to their simplicity and ease of implementation.
1. Definition: A Bezier curve is defined by a set of control points. The simplest Bezier curve is the linear Bezier curve, defined by two points. Quadratic and cubic Bezier curves, defined by three and four points respectively, are commonly used in practice.
2. Control Points: The curve is influenced by its control points, which do not necessarily lie on the curve itself. These points determine the shape and direction of the curve.
3. Properties: Bezier curves are affine invariant, meaning their shape does not change under affine transformations (translation, rotation, scaling). They also exhibit the property of local control, where moving a control point affects only the portion of the curve near that point.
4. Applications: They are used in vector graphic design (e.g., Adobe Illustrator), animation (e.g., motion paths), and font design (TrueType fonts).
5. Advantages: Bezier curves offer a good balance between simplicity and flexibility, allowing for the easy creation of smooth, complex shapes with relatively few control points.
6. Ease of Use: The simplicity of the Bezier curve's mathematical formulation makes it easy to implement and use in various software applications for graphical modeling and animation.

**b) Discuss about polygon rendering methods.**
Polygon rendering methods are techniques used to convert a polygonal

representation of a 3D object into a 2D image on a screen. These methods focus on how to fill the polygons with appropriate colors, textures, and shading to create a realistic or stylized image.
1. Flat Shading: The simplest shading method where a single color is used for each polygon. This color is typically determined by the polygon's orientation relative to a light source. It produces a faceted look, emphasizing the polygonal structure of the model.
2. Gouraud Shading: This method interpolates vertex colors across the surface of the polygon. The color at each vertex is calculated using a lighting model, and these colors are blended smoothly across the polygon's surface. This technique reduces the faceted look and creates smoother transitions between polygons.
3. Phong Shading: Similar to Gouraud shading but with improved accuracy. Instead of interpolating colors, Phong shading interpolates the surface normals across the polygon and calculates the color at each pixel. This produces more realistic highlights and a smoother appearance.
4. Texture Mapping: Involves applying a 2D image (texture) onto the surface of a polygon. Texture mapping adds details without increasing the polygon count, enhancing the visual complexity of the rendered image.
5. Bump Mapping: A technique used to simulate small surface details by perturbing the surface normals before the lighting calculation. This creates the illusion of a textured surface without increasing the geometric complexity.
6. Ray Tracing: A more advanced rendering technique that simulates the way light interacts with objects. Ray tracing can produce highly realistic images with accurate reflections, refractions, and shadows but is computationally intensive.
7. Rasterization: The most common method for real-time rendering. It converts the 3D vertices of polygons into 2D screen coordinates and fills the polygons by interpolating colors and textures. It is highly efficient and widely used in interactive applications like video games.
Polygon rendering methods are essential for producing visually appealing and realistic images in computer graphics, each method offering a different balance between computational complexity and visual quality.

**8. a) Derive the matrix form for Rotation in 3-D graphics.**
Derive the Matrix Form for Rotation in 3-D Graphics:
Rotation in 3-D graphics involves rotating an object around one of the coordinate axes (x, y, or z). Each rotation has a specific matrix representation. Here are the derivations for each axis:

1. Rotation about the X-axis:
The rotation angle is θ.
The coordinates (y, z) transform according to the 2D rotation formulas, while x

remains unchanged.
The rotation matrix R_x is:

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

2. Rotation about the Y-axis:
The rotation angle is θ.
The coordinates (x, z) transform according to the 2D rotation formulas, while y remains unchanged.
The rotation matrix R_y is:

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

3. Rotation about the Z-axis:
The rotation angle is θ.
The coordinates (x, y) transform according to the 2D rotation formulas, while z remains unchanged.
The rotation matrix R_z is:

$$R_z = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

These matrices can be combined to perform rotations around arbitrary axes by multiplying the corresponding matrices together.

**b) Explain about the approaches followed for clipping in 3-D space.**
Approaches Followed for Clipping in 3-D Space:
Clipping in 3-D space involves determining which parts of a 3D object are inside a defined viewing volume and should be rendered. Several approaches

are used for 3-D clipping:
1. Cohen-Sutherland 3D Clipping:
An extension of the 2D Cohen-Sutherland algorithm.
Objects are clipped against a 3D rectangular parallelepiped (viewing volume).
Each point is assigned a 6-bit outcode representing its position relative to the clipping planes.
The algorithm uses logical operations to determine if a line segment is entirely inside, outside, or needs to be clipped.
2. Sutherland-Hodgman Clipping:
Typically used for clipping polygons.
The polygon vertices are successively clipped against each clipping plane.
The process generates new vertices where the edges intersect the clipping planes, resulting in a polygon entirely within the viewing volume.
3. Liang-Barsky Algorithm:
Uses parametric line equations to find intersections with the clipping volume.
It is more efficient than Cohen-Sutherland because it calculates the intersection points directly.
This method works well for line segments and is often preferred for its computational efficiency.
4. Weiler-Atherton Clipping:
Designed for complex polygons, especially those that may result in multiple disjointed pieces after clipping.
The algorithm handles both convex and concave polygons by maintaining lists of vertices inside and outside the clipping volume.
It constructs new polygons from the clipped segments, ensuring accurate rendering of complex shapes.
5. View Frustum Culling:
Determines if entire objects are outside the view frustum and can be excluded from rendering.
This approach uses bounding volumes (like bounding boxes or spheres) to quickly test against the viewing frustum.
It helps in reducing the number of polygons to be processed, improving rendering performance.
Each approach has its advantages and use cases, with the choice depending on the specific requirements of the rendering system and the complexity of the scenes being processed.

**9. a) Briefly explain about 3-D composite transformations.**
a) 3-D Composite Transformations:
3-D composite transformations involve applying a series of basic transformations—such as translation, rotation, scaling, and shearing—in sequence to achieve a desired effect on a 3D object. These transformations are represented by matrices, and the composite transformation is obtained by

multiplying these matrices together.
1. Combination of Transformations: Composite transformations allow for the combination of multiple transformations into a single matrix. For example, to first scale an object, then rotate it, and finally translate it, you multiply the individual matrices in the correct order.
2. Matrix Multiplication: The order of multiplication is crucial because matrix multiplication is not commutative. The resulting transformation matrix is applied to the object's coordinates to achieve the combined effect.
3. Efficiency: Using a single composite matrix reduces computational overhead, as it avoids applying multiple transformations individually to each vertex of the object.
4. Hierarchical Modeling: Composite transformations are used in hierarchical modeling, where complex objects are constructed from simpler parts. Each part can have its transformations, and composite transformations can define the relationships between parts.
5. Homogeneous Coordinates: To facilitate composite transformations, objects are often represented in homogeneous coordinates, allowing translation to be included in the transformation matrix.
6. Application: Composite transformations are widely used in computer graphics for animating objects, creating complex scenes, and simulating real-world motions.
By combining transformations, complex effects such as rotation around an arbitrary axis or scaling relative to a specific point can be achieved more easily.

**b) Write a short note on shear transformations in 3-D.**
Shear Transformations in 3-D:
Shear transformations in 3-D graphics distort the shape of an object such that its sides become non-parallel, effectively "sliding" one face over another along a particular axis.
1. Definition: A shear transformation changes the coordinates of points in an object so that the shape is skewed in one or more directions. In 3D, this can occur along the x, y, or z axes, or in a combination of these.
2. Types of Shear:
XY-Shear: Alters the x and y coordinates as a function of the z coordinate.
XZ-Shear: Alters the x and z coordinates as a function of the y coordinate.
YZ-Shear: Alters the y and z coordinates as a function of the x coordinate.
3. Matrix Representation: Shear transformations are represented by matrices. For example, an xy-shear would have the form:

$$\begin{bmatrix} 1 & sh_{xy} & sh_{xz} & 0 \\ sh_{yx} & 1 & sh_{yz} & 0 \\ sh_{zx} & sh_{zy} & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

where sh_xy, sh_xz, sh_yx, sh_yz, sh_zx, sh_zy are the shear factors.
4. Effect: Shearing distorts the shape by shifting coordinates. For example, in an xy-shear, each point's x and y coordinates are modified based on its z coordinate, producing a slanted effect.
5. Applications: Shear transformations are useful in simulating realistic deformations, such as the effect of wind on a flag or the leaning of a building. They are also used in character animation and modeling to create more dynamic and flexible movements.
6. Combined Shearing: Multiple shear transformations can be combined to produce complex distortions. The order of application affects the final result, similar to other composite transformations.
Shear transformations add flexibility to 3D modeling, allowing for more intricate and realistic modifications to objects.

**10. a) Discuss about the graphical languages followed to achieve animation.**
Graphical languages are specialized programming languages or frameworks used to create animations in computer graphics. These languages provide tools and functionalities to define, manipulate, and control the motion and appearance of objects over time.
1. OpenGL: A widely-used graphics library that provides a low-level interface for rendering 2D and 3D vector graphics. It allows for the creation of complex animations through shaders and programmable pipelines.
2. WebGL: A JavaScript API for rendering interactive 3D graphics within web browsers without the need for plug-ins. It is based on OpenGL ES and is used for web-based animations and visualizations.
3. Blender: An open-source 3D creation suite that supports the entire 3D pipeline, including modeling, rigging, animation, simulation, rendering, compositing, and motion tracking. Blender uses Python for scripting and automating tasks.
4. Three.js: A JavaScript library that simplifies the creation of 3D graphics in the web browser using WebGL. It provides an easy-to-use interface for creating and animating 3D objects, scenes, and cameras.
5. Maya: A professional 3D computer graphics application used for creating interactive 3D applications, including video games, animated films, TV series, and visual effects. Maya uses MEL (Maya Embedded Language) and Python for scripting.
6. Unity: A cross-platform game engine used for developing video games and

simulations. Unity uses C# for scripting and provides powerful tools for creating and controlling animations.
7. Adobe Animate: A multimedia authoring and computer animation program used for creating vector graphics and animations for TV shows, online video, websites, web applications, and video games. It uses ActionScript for scripting.
8. Processing: A flexible software sketchbook and a language for learning how to code within the context of the visual arts. It is used to create visual art and animations with simple syntax and powerful capabilities.
These graphical languages and tools provide a range of functionalities for creating animations, from low-level control over rendering to high-level scripting and automation, making them essential for various applications in entertainment, education, and visualization.

**b) Explain in detail about depth-buffer algorithm.**

The depth-buffer algorithm, also known as the z-buffer algorithm, is a widely-used method for hidden surface removal in 3D computer graphics. It ensures that only the visible surfaces of objects are rendered, creating a correct representation of the scene from the viewer's perspective.
1. Depth Buffer Initialization: The depth buffer is a 2D array that stores the depth values (z-values) of every pixel in the scene. It is initialized to a maximum depth value, representing the farthest possible distance.
2. Rendering Process: For each pixel of every polygon in the scene, the algorithm calculates the depth value. This depth value is compared to the current value stored in the depth buffer at the corresponding pixel location.
3. Depth Comparison: If the calculated depth value is less than the stored value (indicating the pixel is closer to the viewer), the depth buffer is updated with the new depth value, and the color buffer is updated with the pixel's color.
4. Pixel Overwriting: If the calculated depth value is greater than or equal to the stored value, the pixel is not rendered, ensuring that only the nearest (visible) surfaces are displayed.
5. Advantages:
Simplicity: The depth-buffer algorithm is straightforward to implement and integrates well with modern graphics hardware.
Efficiency: It handles complex scenes efficiently, allowing for real-time rendering in applications like video games and simulations.
Versatility: It can handle overlapping objects, transparency, and complex surface geometries.
6. Limitations:
Memory Usage: The depth buffer requires additional memory proportional to the screen resolution.
Precision Issues: Limited precision of depth values can cause artifacts, especially in scenes with objects at vastly different distances.
7. Applications: The depth-buffer algorithm is used in most 3D rendering

pipelines, including those in gaming, simulation, and virtual reality, due to its ability to produce accurate and realistic images.
The depth-buffer algorithm is a fundamental technique in computer graphics, providing a reliable way to manage visibility and rendering of complex 3D scenes.

**11. a) Describe linear list notation of animation languages.**
a) Linear List Notation of Animation Languages:
Linear list notation is a method used in animation languages to represent the sequence of actions and transformations that constitute an animation. It organizes these actions into a straightforward, sequential format, making it easy to understand and manipulate the animation process.
1. Definition: Linear list notation is a simple, ordered list of animation commands that describe the sequence of transformations and operations applied to objects over time.
2. Sequence of Actions: Each entry in the list represents an action or transformation, such as translation, rotation, scaling, or changing an object's properties (color, opacity, etc.).
3. Time Control: Time is an essential aspect of linear list notation, with each action associated with a specific time or frame number. This ensures that the actions are executed in the correct order and at the correct intervals.
4. Simplicity: The linear structure makes it easy to read, write, and modify animations. Animators can quickly add, remove, or rearrange actions to change the animation sequence.
5. Example: An animation of a bouncing ball might include actions like:
Translate ball to initial position
Apply downward force (gravity)
Detect collision with the ground
Reverse direction and apply upward force
Repeat
6. Applications: Linear list notation is used in various animation languages and systems, providing a straightforward way to define and manage animations. It is particularly useful in keyframe animation, where specific frames are defined, and intermediate frames are interpolated.
7. Flexibility: While simple, linear list notation can be extended to include conditional statements and loops, allowing for more complex animations and interactions.
Linear list notation provides a clear and structured way to define animations, making it accessible for both simple and complex animation tasks.

**b) Write a short note on BSP-trees.**

BSP-Trees:

Binary Space Partitioning (BSP) trees are a data structure used in computer graphics to efficiently manage and render complex scenes. They are particularly useful for hidden surface determination, collision detection, and ray tracing.
1. Definition: A BSP tree recursively subdivides a space into convex sets by using hyperplanes. Each node in the tree represents a partition of space, and the leaves represent the subdivided regions.
2. Construction: The construction of a BSP tree involves choosing a polygon as a partitioning plane and dividing the remaining polygons into two sets: those in front of the plane and those behind it. This process is repeated recursively for each subset.
3. Rendering: BSP trees facilitate efficient rendering by allowing the scene to be rendered back-to-front or front-to-back. This ensures that polygons are drawn in the correct order, solving visibility issues.
4. Hidden Surface Removal: By traversing the BSP tree, it is possible to determine which surfaces are visible from a given viewpoint and efficiently remove hidden surfaces.
5. Collision Detection: BSP trees are also used in collision detection algorithms, allowing for quick determination of potential collisions in a 3D space.
6. Advantages:
Efficiency: BSP trees provide efficient ways to handle complex scenes, especially in real-time rendering and interactive applications.
Versatility: They can be used for various tasks, including visibility determination, ray tracing, and physics simulations.
7. Disadvantages:
Preprocessing Time: Building a BSP tree can be time-consuming, especially for very complex scenes.
Dynamic Scenes: BSP trees are less suited for dynamic scenes where objects frequently move or change, as the tree would need frequent updates.
8. Applications: BSP trees are widely used in computer games, simulations, and CAD systems for managing and rendering 3D environments efficiently.
BSP trees are a powerful tool in computer graphics, providing a robust method for managing spatial information and rendering complex scenes accurately and efficiently.