Computer Graphics

Syllabus

Course Title: **Computer Graphics** Full Marks: 60 + 20 + 20
Course no: CSC209 Pass Marks: 24 + 8 + 8
Nature of the Course: Theory + Lab Credit Hrs: 3
***Semester:* III**

Course Description: The course covers concepts of graphics hardware, software, and applications, data structures for representing 2D and 3D geometric objects, drawing algorithms for graphical objects, techniques for representing and manipulating geometric objects, illumination and lighting models, and concept of virtual reality.

Course Objectives: The objective of this course is to understand the theoretical foundation as well as the practical applications of 2D and 3D graphics.

Course Contents:

Unit 1: Introduction of Computer Graphics **(3 Hrs.)**

1.1 A Brief Overview of Computer Graphics, Areas of Applications.

1.2 Graphics Hardware: Display Technology, Architecture of Raster-Scan Displays, Vector Displays, Display Processors, Hard copy device. Input Devices.

1.3 Graphics Software: Software standards, Need of machine independent graphics language.

Unit 2: Scan Conversion Algorithm **(6 Hrs.)**

2.1 Scan Converting a Point and a straight Line: DDA Line Algorithm, Bresenham's Line Algorithm

2.2 Scan Converting Circle and Ellipse: Mid Point Circle and Ellipse Algorithm

2.3 Area Filling: Scan Line Polygon fill Algorithm, Inside-outside Test, Scan line fill of Curved Boundary area, Boundary-fill and Flood-fill algorithm

Unit 3: Two-Dimensional Geometric Transformations **(5 Hrs.)**

3.1 Two-Dimensional translation, Rotation, Scaling, Reflection and Shearing

3.2 Homogeneous Coordinate and 2D Composite Transformations. Transformation between Co-ordinate Systems.

3.3 Two Dimensional Viewing: Viewing pipeline, Window to viewport coordinate transformation

3.4 Clipping: Point, Lines (Cohen Sutherland line clipping, Liang-Barsky Line Clipping), Polygon Clipping(Sutherland Hodgeman polygon clipping)

Unit 4: Three-Dimensional Geometric Transformation **(5 Hrs.)**

4.1 Three-Dimensional translation, Rotation, Scaling, Reflection and Shearing

4.2 Three-Dimensional Composite Transformations

4.3 Three-Dimensional Viewing: Viewing pipeline, world to screen viewing transformation, Projection concepts (Orthographic, parallel, perspective projections)

Unit 5: 3D Objects Representation **(7 Hrs.)**

5.1 Representing Surfaces: Boundary and Space partitioning

5.1.1 Polygon Surface: Polygon tables, Surface normal and Spatial orientation of surfaces, Plane equations, Polygon meshes

5.1.2 Wireframe Representation

5.1.3 Blobby Objects

5.2 Representing Curves: Parametric Cubic Curves, Spline Representation, Cubic spline interpolation, Hermite Curves, Bezier and B-spline Curve and surface

5.3 Quadric Surface: Sphere and Ellipsoid

Unit 6: Solid Modeling (4 Hrs.)

6.1 Sweep, Boundary and Spatial-Partitioning Representation
6.2 Binary Space Partition Trees (BSP)
6.3 Octree Representation

Unit 7: Visible Surface Detections (5 Hrs.)

7.1 Image Space and Object Space Techniques
7.2 Back Face Detection, Depth Buffer (Z-buffer), A-Buffer and Scan-Line Algorithms.
7.3 Depth Sorting Method (Painter's Algorithm)
7.4 BSP tree Method, Octree and Ray Tracing

Unit 8: Illumination Models and Surface Rendering Techniques (5 Hrs.)

8.1 Basic Illumination Models: Ambient light, Diffuse reflection, Specular reflection and Phong model
8.2 Intensity attenuation and Color consideration, Transparency, Shadows
8.3 Polygon Rendering Methods: Constant intensity shading, Gouraud shading, Phong Shading and Fast Phong Shading

Unit 9: Introduction to Virtual Reality (2 Hrs.)

9.1 Concept of Virtual reality
9.2 Virtual Reality Components of VR System, Types of VR System, 3D Position Trackers, Navigation and Manipulation Interfaces
9.3 Application of VR

Unit 10: Introduction to OpenGL (3 Hrs.)

1.1 Introduction, Callback functions, Color commands, Drawings pixels, lines, polygons using OpenGL, Viewing and Lighting

Laboratory Works:

The laboratory course consists of implementing following algorithms using high level languages and OpenGL.

1. DDA Line Algorithm
2. Bresenham's line drawing algorithm
3. Mid Point Circle Algorithm
4. Mid Point Ellipse Algorithm
5. Basic transformation on 2D including Translation, Rotation and Scaling
6. Simple 3D Object with basic transformations including Translation, Rotation and Scaling
7. Clipping
8. Hidden surface removal
9. Basic Drawing Techniques in OpenGL

Text Books:

1. Donald Hearne and M. Pauline Baker, "Computer Graphics, C Versions." Prentice Hall

Reference Books:

1. J.D. Foley, S.K. Feiner and J.F. Hughes, "Computer Graphics – Principles and Practises" (Second Edition in C)
2. R.K. Maurya, "Computer Graphics with Virtual Reality", Wiley India
3. F.S. Hill, Stephen M.Kelley, "Computer Graphics using Open GL" Prentice Hall

TRIBHUVAN UNIVERSITY

Institution of Science and Technology

Course Title: Computer Graphics
Course No: CSC 209
Nature of the Course: Theory + Lab
Semester: III

Full Marks: 60
Pass Marks: 24
Time: 3 hrs

Computer Graphics

MODEL QUESTIONS-ANSWERS

Section A

Long Answer Questions

Attempt any *TWO* questions [2 × 10 = 20]

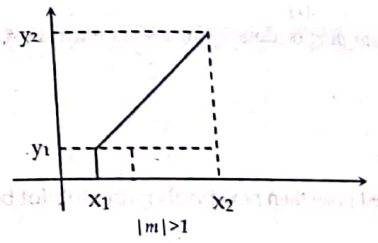
1. Explain the working details of DDA algorithm? Explain. Digitize a line with end points A(6, 12) and B(10, 5) using Bresenham's line drawing algorithm. [5 +5]

Ans: The digital differential analyzer (DDA) is a scan conversion line drawing algorithm based on calculating either Δx or Δy form the equation,

$m = \Delta y / \Delta x$

We sample the line at unit intervals in one co-ordinate and determine the corresponding integer values nearest to the line path for the other co-ordinates. Here following cases occur:

Case 1: If m is positive (m>0) and |m|<=1,



Let (x_k, y_k) be the plotted pixel then next pixels going to plot be,

$x_{k+1} = x_k + \Delta x$

$y_{k+1} = y_k + \Delta y$ Where, Δx and Δy are small incremental distance.

We have m=

$\Delta y = m\Delta x$.....(1)

Also here |m|<=1, sampling be done along x-axis with $\Delta x=1$.

Then,

$x_{k+1} = x_k + 1$

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

Here k takes value from starting point and increase by 1 until final end point. m can be any real value between 0 and 1.

Case 2: If m is positive (m > 0) and $|m| >= 1$,

Let (x_k, y_k) be the plotted pixel then next pixels going to plot be,

$x_{k+1} = x_k + \Delta x$

$y_{k+1} = y_k + \Delta y$

Where Δx and Δy are small incremental distance

We have m=

$\Delta y = m \Delta x$.....(1)

Also here $|m| >= 1$, sampling be done along y-axis with $\Delta y = 1$.

Then, $\Delta x = 1/m$

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

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

The above equations are under the assumption that the lines are processed from left to right. I.e. left end point is starting.

If the processing is from right to left

Case 3: if $|m| <= 1$

Let (x_k, y_k) be the plotted pixel then next pixels going to plot be,

$x_{k+1} = x_k + \Delta x$

$y_{k+1} = y_k + \Delta y$

Where Δx and Δy are small incremental distance

We have m=

$\Delta y = m \Delta x$.....(1)

Also here $|m| <= 1$, sampling be done along x-axis with $\Delta x = -1$.

Then,

$x_{k+1} = x_k - 1$

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

Case 4: If $|m| > 1$

Let (x_k, y_k) be the plotted pixel then next pixels going to plot be,

$x_{k+1} = x_k + \Delta x$

$y_{k+1} = y_k + \Delta y$

Where Δx and Δy are small incremental distance

We have m =

$\Delta y = m \Delta x$.....(1)

Also here $|m| >= 1$, sampling be done along y-axis with $\Delta y = -1$.

Then,

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

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

Therefore, in general,

$\therefore y_{k+1} = y_k \pm m$ For $|m| < 1$ and

$x_{k+1} = x_k \pm \frac{1}{m}$ For $|m| > 1$

Digitize a line with end points (6, 12) and (10, 5) using BSA

Soln: Here, $\Delta y = |5-12| = 7$

$\Delta x = |10-6| = 4$

Since, $|m| = \Delta y / \Delta x \geq 1$, so we sample at y direction i.e., at each step we simply increment y-coordinate by 1 and find appropriate x-coordinate.

First pixel to plot is (6, 12), which is the starting endpoint.

Now, initial decision parameter, $p_0 = 2\Delta x - \Delta y = 2 * 4 - 7 = 1$

We know that, If $p_k < 0$, then we need to set $p_{k+1} = p_k + 2\Delta x$ and plot pixel $(x_k, y_k - 1)$.

And if $p_k \geq 0$, then we need to set $p_{k+1} = p_k + 2\Delta x - 2\Delta y$ then plot pixel $(x_k + 1, y_k - 1)$

Here, $p_0 = 1$, we have i.e., $p_k \geq 0$

So, $p_1 = p_0 + 2\Delta x - 2\Delta y = 1 + 2 * 4 - 2 \times 7 = -5$

$p_2 = p_1 + 2\Delta x = -5 + 2 * 4 = 3$

$p_3 = p_2 + 2\Delta x - 2\Delta y = 3 + 2*4 - 2*7 = -3$

$p_4 = p_3 + 2\Delta x = -3 + 2 * 4 = 5$

$p_5 = p_4 + 2\Delta x - 2\Delta y = 5 + 2*4 - 2*7 = -1$

$p_6 = p_5 + 2\Delta x = -1 + 2*4 = 7$

$p_7 = p_6 + 2\Delta x - 2\Delta y = 7 + 2*4 - 2*7 = 1$

$p_8 = p_7 + 2\Delta x - 2\Delta y = 1 + 2*4 - 2*7 = -5$

$p_9 = p_8 + 2\Delta x = -5 + 2*4 = 3$

Successive pixel positions and decision parameter calculation is shown in the table below,

K	X	Y	p_k	(x_{k+1}, y_{k+1})
0	6	12	1 (>=0)	(7, 11)
1	7	11	-5 < 0	(7, 10)
2	7	10	3	(8, 9)
3	8	9	-3	(8, 8)
4	8	8	5	(9, 7)
5	9	7	-1	(9, 6)
6	9	6	7	(10, 5)

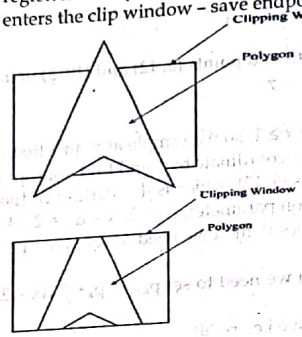
2. How can polygons be clipped? Why is Phong shading also called Normal Vector Interpolation scheme? Explain. **[5 +5]**

Ans: Polygon Clipping

A set of connected lines are considered as polygon; polygons are clipped based on the window and the portion which is inside the window is kept as it is and the outside portions are clipped. The polygon clipping is required to deal different cases. Usually it clips the four edges in the boundary of the clip rectangle. The clip boundary determines the visible and invisible regions of polygon clipping and it is categorized as four.

- Visible region is wholly inside the clip window – saves endpoint
- Visible exits the clip window – Save the interaction

- Visible region is wholly outside the clip window – nothing to save
- Visible enters the clip window – save endpoint and intersection

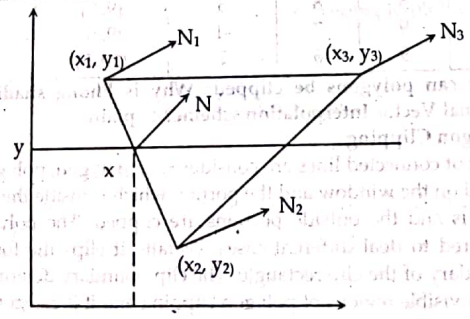


Phong shading

It is the best known shading algorithm, developed by Phong Bui Tuong, is called Phong shading or normal vector interpolation shading. Idea here is to interpolate the normal vectors instead of the light intensity. Then apply the illumination model at each surface point. It provides more accurate calculation of light intensity values and more realistic surface highlights. But it is computationally inefficient.

A more accurate method for rendering a polygon surface is to interpolate normal vector and then apply illumination model to each surface point. This method is called Phong shading or normal vector interpolation method for shading. It displays more realistic highlights and greatly reduces the mach band effect. A polygon surface is rendered with Phong shading by carrying out following calculations.

- Determine the average normal unit vectors at each polygon vertex.
- Linearly interpolate vertex normal over the surface of polygon.
- Apply illumination model along each scan line to calculate the pixel intensities for the surface point.



In figure, N_1, N_2, N_3 are the normal unit vectors at each vertex of polygon surface. For scan-line that intersects an edge, the normal vector N can be obtained by vertically interpolating normal vectors of the vertex on that edge as,

$$N = \frac{y - y_2}{y_1 - y_2} N_1 + \frac{y_1 - y}{y_1 - y_2} N_2$$

Incremental calculations are used to evaluate normal between scan lines and along each individual scan line as in Gouraud shading. Phong shading produces accurate results than the direct interpolation but it requires considerably more calculations.

Algorithm

- Determine the average unit normal vector at each polygon vertex.
- Linearly interpolate the vertex normal over the surface of the polygon.
- Apply an illumination model along each scan line to calculate pixel intensities for the surface points. The intensity value is calculated using the interpolated normal vector.

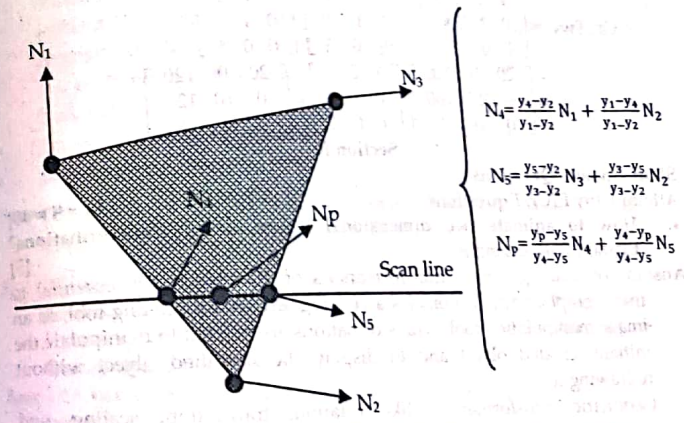


Fig: Phong shading

Advantages

- It displays more realistic highlights on a surface
- It reduces the mach band effect
- It gives more accurate result

Disadvantages

- It requires more calculations
- It greatly increases the cost of shading steeply.

3. Given a window bordered by (1, 2) at the lower left and (16, 12) at the upper right, give the screen coordinates of a triangle with vertices (3, 2), (10, 7.5) and (5, 5) when mapped into a viewport with corners (100, 100) and (400, 200). Provide accurate illustrations of the window, viewport, and the untransformed and transformed triangles with your answer. [10]

Ans: We have the coordinate for windows,

Xwmin=1 Ywmin = 2

Xwmax = 16 Ywmax = 12

Coordinates of view port,

Xvmin = 100 Yvmin = 100

Xvmax = 400 Yvmax = 200

We know that,

$$Sx = \frac{X\text{-}Xvmin}{X\text{-}Xwmin} = \frac{400-100}{16-1} = 300/15 = 20$$

And

$$Sy = \frac{Yvmax}{X\text{-}Ywmin} = \frac{200-100}{12-2} = 100/10 = 10$$

We know the composite transformation matrix is given as,

Twv = T(Xvmin, Yvmin) . S(Sx, Sy) . T(-Xwmin, -Ywmin)

$$\text{Or, Twv} = \begin{bmatrix} 1 & 0 & 100 \\ 0 & 1 & 100 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 20 & 0 & 0 \\ 0 & 10 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 20 & 0 & 100 \\ 0 & 10 & 100 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 20 & 0 & 120 \\ 0 & 10 & 120 \\ 0 & 0 & 1 \end{bmatrix}$$

Section B

Short Answer Questions

Attempt any *EIGHT* questions [8 × 5 = 40]

4. How to animate two dimensional figure using transformations? Explain with example. [5]

Ans: In computer graphics, transformations of 2D objects are essential to many graphics applications (as a viewing aid, as a modeling tool, as an image manipulation tool). Transformations are needed to manipulate the initially created object and to display the modified object without redrawing it.

Geometric transformations like rotation, translation, scaling, and projection can be accomplished with matrix multiplication and matrices are a convenient and efficient way to represent a sequence of transformations. Represent a 2D transformation by a 2×2 matrix

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

Apply the transformation to a point

$x' = ax + by$

$y' = cx + dy$ $\Longrightarrow$ $\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \times \begin{bmatrix} x \\ y \end{bmatrix}$

Image = T.M. × Object

If object and image are represented in row matrix then

$x' = ax + by$

$y' = cx + dy$ $\Longrightarrow$ $[x' \;\; y'] = [x \;\; y] \times \begin{bmatrix} a & b \\ c & d \end{bmatrix}$

Image = Object × T.M.

Transformations can be combined by matrix multiplication

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} e & f \\ g & h \end{bmatrix} \begin{bmatrix} i & j \\ k & l \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

Image = $T.M_1 \times T.M_2 \times T.M_3 \times$ Object

Homogeneous co-ordinate representation of 2D Transformation

The homogeneous co-ordinate system provides a uniform frame-work for handling different geometric transformations, simply as multiplication of matrices. In homogeneous coordinate representation each 2D point (x, y) is represented as homogeneous coordinate triple (x_h, y_h, h), where $x = x_h/h$, $y = y_h/h$ (h is 1 usually for 2D case). The reason why this coordinate system is called 'homogeneous' is because it is possible to transform functions such as $f(x, y)$ into the form $f(x/h, y/h)$ without disturbing the degree of the curve. In the field of projective geometry it is a very powerful concept.

- (x_h, y_h, h) represents a point at location $(x_h/h, y_h/h)$
- $(x_h, y_h, 0)$ represents a point at infinity
- $(0, 0, 0)$ Is not allowed

Examples: 2D point (2, 1) can be represented as (2, 1, 1) or (4, 2, 2) or (6, 3, 3) so on.

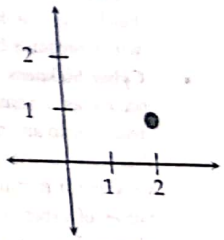


Figure: Possible Homogenous coordinate of a 2D Point (2, 1)

5. What are the key issues prevalent in producing a Virtual reality scene? Describe the Binary Space Partition tree [2 +3]

Ans: The key issues prevalent in producing a virtual reality scene are listed below:

- **Social Impact:** There is high level of concern over the negative influences of interactive VR environments towards social implications. The users who are engage in violence VR video games and television in the virtual world may become desensitized to their violent virtual actions and mimic that behavior in real world. There are other issues like people turning their backs on the real world and wander around the synthetic worlds that fulfill their whims. As of now, violence in VR is nearly inevitable but it is still important to address social issues before they result in crisis or harm.

- **Human Sensory Limitations:** For virtual environment systems to be compatible with their users, it is vital for designers to understand design constraints imposed by human sensory and motor physiology. The physiological and perceptual issues that directly impact the design of virtual environment systems are visual perception, auditory perception, and haptic and kinesthetic perception. The human visual system is very sensitive to any anomalies in perceived imagery and becomes prominent when motion is introduced into a virtual reality. In auditory perception, there is challenge for audio localization to obtain realistic auditory environment. Localization is helps differentiating sound sources and their direction.
- **Direct micro/macroscopic Effect:** Ensuring Health and safety of users are important and challenging issues for VR systems to avoid discomfort, harm or even injury. Developers should ensure that advancement in technology do not come at the expense of human well-being. When experiencing VR, the brain tends to work harder to integrate the unusual stimuli being presented to the different senses. Therefore, VR has power to affect the senses and brain of a user, leading to fatigue or sickness such as dizziness and nausea unlike any other simpler media. It is due to the problems in hardware, low-level software or carelessness of a VR developer who disregards the side effects of the experience on the user.
- **Cyber Sickness:** Cyber sickness is a form of motion sickness that occurs as a result of exposure to VR. It can range from slight headache to an emetic response. Several factors has been identified that may contribute to cyber sickness such as vection, lag, field of view but it is still an undergoing research to identify the specific causes of cyber sickness and to develop methods to alleviate this ailment. Vection is illusion of self-movement in VR which causes conflicts between the visual and vestibular system in the body because the motion is just illusion.

Binary Space Partitioning (BSP)

Many 3D modeling and rendering programs utilize a Binary Space Partition tree (BSP tree) to make rendering go faster. Sometimes these programs will even have customizable settings for the BSP tree. Binary space partitioning is a generic process of recursively dividing a scene into two until the partitioning satisfies one or more requirements. It can be seen as a generalization of other spatial tree structures such as *k*-d trees and quad trees, one where hyper planes that partition the space may have any orientation, rather than being aligned with the coordinate axes as they are in *k*-d trees or quad trees. When used in computer graphics to render scenes composed of planar polygons, the partitioning

planes are frequently chosen to coincide with the planes defined by polygons in the scene.

A BSP tree is a way of grouping data so it can be processed faster. It's used in many computer science applications, not just VFX rendering.BSP trees are often used by 3D video games, particularly first-person shooters and those with indoor environments. Game engines using BSP trees include the Doom, Quake, and Source engines. The grouping process starts with data of some kind, and divides it into two parts. Then the two parts are divided into two parts, and so on until the parts at the end contain the smallest piece of working data. As a very simple example, let's take the word "graphics" and partition it with a BSP tree. For this example, a single letter of the alphabet is the smallest possible element.

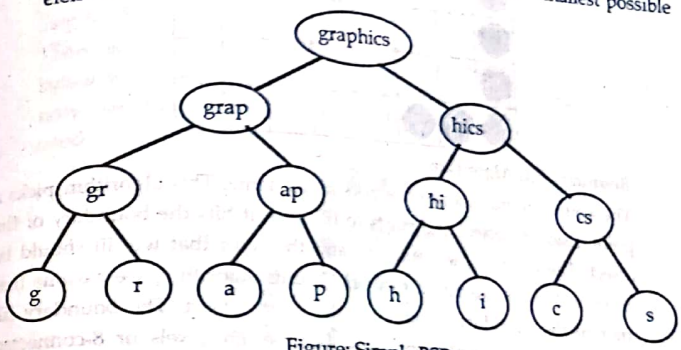


Figure: Simple BSP tree

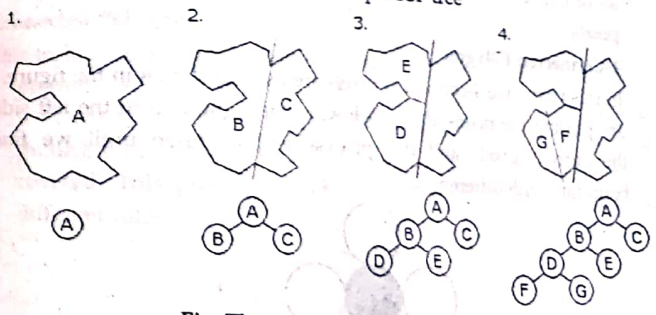
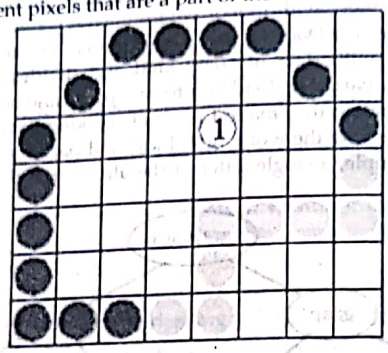


Fig: The process of making a BSP tree

6. **How can a polygon surface be filled using the Flood fill approach Explain.** [5

Ans: Sometimes we come across an object where we want to fill the area ar its boundary with different colors. We can paint such objects with specified interior color instead of searching for particular boundary col as in boundary filling algorithm. Instead of relying on the boundary the object, it relies on the fill color. In other words, it replaces the inter

color of the object with the fill color. When no more pixels of the original interior color exist, the algorithm is completed. Once again, this algorithm relies on the Four-connect or Eight-connect method of filling in the pixels. But instead of looking for the boundary color, it is looking for all adjacent pixels that are a part of the interior.

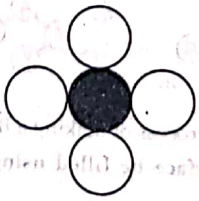


Boundary Fill Algorithm

The boundary fill algorithm works as its name. This algorithm picks a point inside an object and starts to fill until it hits the boundary of the object. The color of the boundary and the color that we fill should be different for this algorithm to work. In this algorithm, we assume that color of the boundary is same for the entire object. The boundary fill algorithm can be implemented by 4-connected pixels or 8-connected pixels.

4-Connected Polygon

In this technique 4-connected pixels are used as shown in the figure. We are putting the pixels above, below, to the right, and to the left side of the current pixels and this process will continue until we find a boundary with different color.



Algorithm

Step 1 – Initialize the value of seed point seedx, seedy seedx, seedy, fcolor and dcol.

Step 2 – Define the boundary values of the polygon.

Step 3 – Check if the current seed point is of default color, then repeat the steps 4 and 5 till the boundary pixels reached.

If getpixel(x, y)= dcol then repeat step 4and5

Step 4 – Change the default color with the fill color at the seed point.

setPixel(seedx, seedy, fcol)

Step 5 – Recursively follow the procedure with four neighborhood points.

FloodFill (seedx – 1, seedy, fcol, dcol)

FloodFill (seedx + 1, seedy, fcol, dcol)

FloodFill (seedx, seedy - 1, fcol, dcol)

FloodFill (seedx – 1, seedy + 1, fcol, dcol)

Step 6 – Exit

There is a problem with this technique. Consider the case as shown below where we tried to fill the entire region. Here, the image is filled only partially. In such cases, 4-connected pixels technique cannot be used.

8-Connected Polygon

In this technique 8-connected pixels are used as shown in the figure. We are putting pixels above, below, right and left side of the current pixels as we were doing in 4-connected technique. In addition to this, we are also putting pixels in diagonals so that entire area of the current pixel is covered. This process will continue until we find a boundary with different color.

Algorithm

Step 1 – Initialize the value of seed point seedx, seedyseedx, seedy fcolor and dcol.

Step 2 – Define the boundary values of the polygon.

Step 3 – Check if the current seed point is of default color then repeat the steps 4 and 5 till the boundary pixels reached

If getpixel(x,y) = dcol then repeat step 4 and 5

Step 4 – Change the default color with the fill color at the seed point.

setPixel(seedx, seedy, fcol)

Step 5 – Recursively follow the procedure with four neighbourhood points

FloodFill (seedx – 1, seedy, fcol, dcol)

FloodFill (seedx + 1, seedy, fcol, dcol)

FloodFill (seedx, seedy - 1, fcol, dcol)

FloodFill (seedx, seedy + 1, fcol, dcol)

FloodFill (seedx – 1, seedy + 1, fcol, dcol)

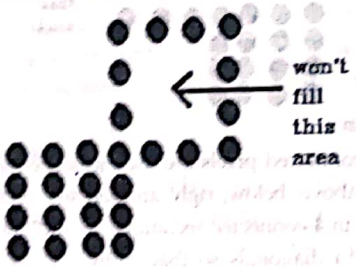
FloodFill (seedx + 1, seedy + 1, fcol, dcol)

FloodFill (seedx + 1, seedy - 1, fcol, dcol)

FloodFill (seedx – 1, seedy - 1, fcol, dcol)

Step 6 – Exit

The 4-connected pixel technique failed to fill the area as marked in the following figure which won't happen with the 8-connected technique.



7. What is the significance of vanishing points in Perspective Projection? Explain. [5]

Ans: A vanishing point is a point on the image plane of a perspective drawing where the two-dimensional perspective projections (or drawings) of mutually parallel lines in three-dimensional space appear to converge.

A vanishing point, or point of convergence, is a key element in many works of art. In a linear perspective drawing, the vanishing point is the spot on the horizon line to which the receding parallel lines diminish. It is what allows us to create drawings, paintings, and photographs that have a three-dimensional look.

The easiest way to illustrate this in real life is to stand in the middle of a straight road. When you do this, you'll notice how the sides of the road and the lines painted on it meet in one spot on the horizon. The center line will go straight for it, and the lines on the side will angle in until all of them intersect. That point of intersection is the vanishing point.

The vanishing point may also be referred to as the 'direction point', as lines having the same directional vector, say D, will have the same vanishing point. Mathematically, let $q \equiv (x, y, f)$ be a point lying on the image plane, where f is the focal length (of the camera associated with the image), and let $v_q \equiv (x/h, y/h, f/h)$ be the unit vector associated with q, where $h = \sqrt{x^2 + y^2 + f^2}$. If we consider a straight line in space S with the unit vector $n_s \equiv (n_x, n_y, n_z)$ and its vanishing point v_s the unit vector associated with v_s is equal to n_s, assuming both are assumed to point towards the image plane.

When the image plane is parallel to two world-coordinate axes, lines parallel to the axis which is cut by this image plane will have images that meet at a single vanishing point. Lines parallel to the other two axes will not form vanishing points as they are parallel to the image plane. This is one-point perspective. Similarly, when the image plane intersects two world-coordinate axes, lines parallel to those planes will meet form two vanishing points in the picture plane. This is called two-point perspective. In three-point perspective the image plane intersects the x, y, and z axes and therefore lines parallel to these axes intersect, resulting in three different vanishing points.

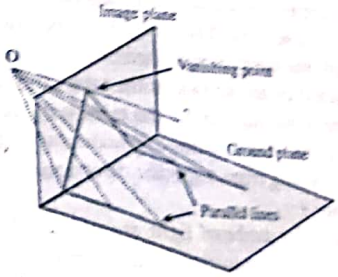


Fig: A 2D construction of perspective viewing, showing the formation of a vanishing point

8. Explain ambient light, diffuse reflection and specular reflection with examples. [5]

Ans:

1. Ambient light

Ambient light means the light that is already present in a scene, before any additional lighting is added. It usually refers to natural light, either outdoors or coming through windows etc. It can also mean artificial lights such as normal room lights. An ambient light source represents a fixed-intensity and fixed-color light source that affects all objects in the scene equally.

Surface that is not exposed directly to a light source still will be visible if nearby objects are illuminated. This light is called ambient light. This is a simple way to model the combination of light reflections from various surfaces to produce a uniform illumination called the ambient light, or background light. Ambient light has no spatial or directional characteristics. The amount of ambient light incident on each object is a constant for all surfaces and over all directions.

If a surface is exposed only to ambient light, then the intensity of the diffuse reflection at any point on the surface is;

$I = k_a I_a$.

Where, I_a is the intensity of the ambient light, and k_a is the ambient reflection coefficient.

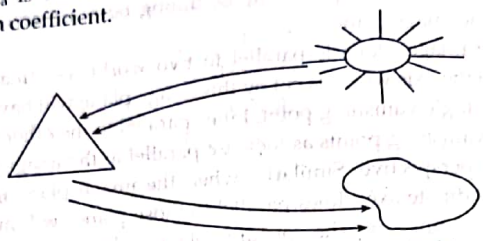


Fig: Object illuminated due to ambient light

Ambient light does not come from a specific light source and has no direction. It represents the light that is more or less everywhere in the scene, originating from multiple reflections of light at various surfaces. In a room with a lamp on a table, it will not be completely dark under the table although the lamp cannot shed its light directly under the table. The light is reflected by the surface of the table, the walls, the ceiling and the floor. Of course, the light under the table will have a lower intensity, but it will still be there with approximately the same intensity everywhere, not coming from a specific direction.

2. **Diffuse reflection**

Objects illuminated by ambient light are uniformly illuminated across their surfaces then the illumination is called diffuse illumination. It is the background light reflected from walls, floor and ceilings. We assume that the reflection is constant over each surface of the object and are independent of the viewing direction. This type of reflection on dull surfaces is called diffuse reflection.

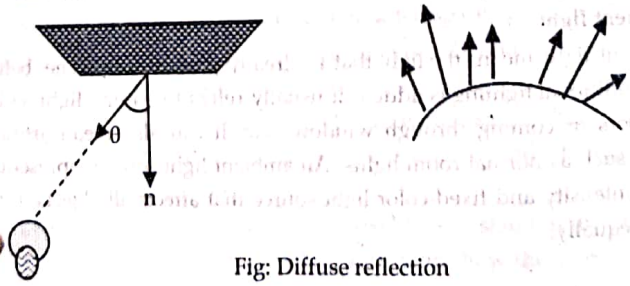


Fig: Diffuse reflection

This effect of light reflection for purely dull surfaces can be computed according to Lambert's cosine law by the illumination equation:

$I = I_L \cdot K_d \cdot \cos\theta$.....(1)

Where, I_L is the intensity of the light hitting the surface, $0 \le k_d \le 1$ is the reflection coefficient of the surface or the material, and θ is the angle between the normal vector **n** to the surface in the considered point and the vector **I** pointing in the direction where the light comes from. The illumination equation (1) for diffuse reflection is only valid for angles θ between $0°$ and $90°$. Otherwise, the light ray hits the surface from the backside so that no reflection occurs. In the case of directional light coming from a light source in infinite distance, the variable I_L in (1) has the same value everywhere. In the case of a point light source, I_L is the intensity of the light source multiplied by the attenuation factor, which depends on the distance of the point on the surface to the light source.

3. **Specular reflection**

Specular light is the white highlight reflection seen on smooth, shiny objects. Specular light is dependent on the direction of the light, the surface normal and the viewer location. When we look at an illuminated shiny surface, such as polished metal, a person's forehead, we see a highlight or bright spot, at certain viewing direction. Such phenomenon is called specular reflection. This phenomenon occurs as a result of total internal reflection of the incident light in a concentrated region around the specular reflection angle. Due to this phenomenon, the surface appears to be not in its original color but of white color.

Diffuse reflection on dull surfaces reflects the light into all directions. **Specular reflection** occurs on shiny surfaces. Such shiny surfaces reflect at least a portion of the light in a similar way as a mirror. In contrast to diffuse reflection, ideal specular reflection takes place only in one direction. The difference between diffuse and specular reflection is illustrated in fig below.

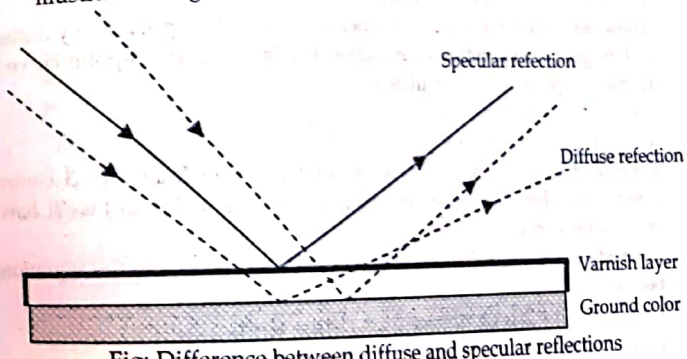


Fig: Difference between diffuse and specular reflections

Shiny surfaces very often have a very thin transparent layer, for instar varnish. When light hits the surface, part of the light penetrates varnish layer and is reflected on the dull surface of the object. This p

of the light is subject to diffuse reflection and the color of the reflected light depends strongly on the ground color of the dull surface. Another part of the light is directly reflected on the transparent layer by specular reflection. Therefore, specular reflection does usually not change the color of the light. The position of the viewer is of no importance for calculating effects coming from diffuse reflection. Whether or where the viewer can see specular reflection depends on his position.

The Phong specular reflection model described by the relation,

$I = I_L . w(0) . \cos(\alpha)^n$.....(1)

Where, I_L is the intensity of the light. The value $0 \leq w(0) \leq 1$ is the fraction of the light which is directly reflected at the shiny surface. n is the specular reflection exponent of the surface. For a perfect mirror, $n \approx \infty$ would hold. A smaller n leads to a less focused specular reflection.

Fig: Specular reflection

9. Compute the midpoint of the Bezier Curve with control points $p_0 = (0, 0, 1)$, $p_1 = (1, 0, 1)$ and $p_2 = (1, 2, 0)$. [5]

Ans: Given the coordinates of control points P_i: the first control point has coordinates $P_1 = (x_1, y_1)$, the second: $P_2 = (x_2, y_2)$, and so on, the curve coordinates are described by the equation that depends on the parameter t from the segment [0,1].

The formula for a 2-points curve:

$P = (1-t)P_1 + tP_2$

For 3 control points:

$P = (1-t)^2P_1 + 2(1-t)tP_2 + t^2P_3$

For 4 control points:

$P = (1-t)^3P_1 + 3(1-t)^2tP_2 + 3(1-t)t^2P_3 + t^3P_4$

These are vector equations. In other words, we can put x and y instead of P to get corresponding coordinates. For instance, the 3-point curve is formed by points (x, y) calculated as:

$x = (1-t)^2x_1 + 2(1-t)tx_2 + t^2x_3$

$y = (1-t)^2y_1 + 2(1-t)ty_2 + t^2y_3$

Instead of x_1, y_1, x_2, y_2, x_3, y_3 we should put coordinates of 3 control points, and then as t moves from 0 to 1, for each value of t we'll have (x, y) of the curve.

For instance, if control points are (0, 0), (0.5, 1) and (1, 0), the equations become:

$x = (1 - t)^2 * 0 + 2(1-t)t * 0.5 + t^2 * 1 = (1-t)t + t^2 = t$

$y = (1 - t)^2 * 0 + 2(1-t)t * 1 + t^2 * 0 = 2(1-t)t = -t^2 + 2t$

Now as t runs from 0 to 1, the set of values (x,y) for each t forms the curve for such control points.

==========

10. How does a polygon can be created in OpenGL? Illustrate with an example. [5]

Ans: Polygons are typically drawn by filling in all the pixels enclosed within the boundary, but you can also draw them as outlined polygons or simply as points at the vertices. A filled polygon might be solidly filled or stippled with a certain pattern. Although the exact details are omitted here, filled polygons are drawn in such a way that if adjacent polygons share an edge or vertex, the pixels making up the edge or vertex are drawn exactly once—they're included in only one of the polygons. This is done so that partially transparent polygons don't have their edges drawn twice, which would make those edges appear darker (or brighter, depending on what color you're drawing with).

A polygon has two sides—front and back—and might be rendered differently depending on which side is facing the viewer. This allows you to have cutaway views of solid objects in which there is an obvious distinction between the parts that are inside and those that are outside. By default, both front and back faces are drawn in the same way. To change this, or to draw only outlines or vertices, use **glPolygonMode()**.

We can draw polygon by following code of segment;

```
glBegin(GL_POLYGON); // Draw A Quad
glVertex3f(-0.5f, 1.0f, 0.0f); // Top Left
glVertex3f(-1.0f, 0.0f, 0.0f); // Left
glVertex3f(-0.5f, 1.0f, 0.0f); // Bottom Left
glVertex3f(0.5f, 1.0f, 0.0f); // Top Right
glVertex3f(1.0f, 0.0f, 0.0f); // Right
glVertex3f(0.5f, 1.0f, 0.0f); // Bottom Right
glEnd();
```

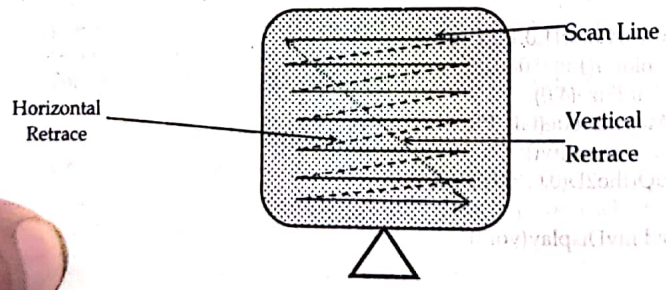
Complete example of drawing a polygon in OpenGL by using c/c++ as below:

```
#include <windows.h>
#include <stdio.h>
#include <conio.h>
#include <iostream.h>
#include <stdlib.h>
#include <glut.h>
void myInit(void)
{
glClearColor(1.0,1.0,1.0,0.0);
glColor3f(1.0f,0.0f,0.0f);
glPointSize(4.0);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluOrtho2D(0.0,800.0,0.0,600.0);
}
void myDisplay(void)
{
```

```
glClear(GL_COLOR_BUFFER_BIT);
glBegin(GL_POLYGON);
glVertex2i(100,100);
glVertex2i(100,300);
glVertex2i(400,300);
glVertex2i(600,150);
glVertex2i(400,100);
glEnd(); glFlush();
}
int main(int argc,char * argv[])
{
glutInit(&argc,argv);
glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
glutInitWindowSize(800,600);
glutInitWindowPosition(100,100);
glutCreateWindow("opengl Window");
glutDisplayFunc(myDisplay);
myInit(); glutMainLoop();
getch();
return 0;
}
```

11. How do a video controller and a frame buffer jointly collaborate to produce graphical display on the screen, in case of a Raster Display?[5]

Ans: In a raster scan system, the electron beam is swept across the screen, one row at a time from top to bottom. As the electron beam moves across each row, the beam intensity is turned on and off to create a pattern of illuminated spots. Picture definition is stored in memory area called the **Refresh Buffer** or **Frame Buffer**. This memory area holds the set of intensity values for all the screen points. Stored intensity values are then retrieved from the refresh buffer and "painted" on the screen one row (scan line) at a time as shown in the following illustration. Each screen point is referred to as a **pixel (picture element)** or **pel**. At the end of each scan line, the electron beam returns to the left side of the screen to begin displaying the next scan line. In comparisons of random scan display method it is more accurate method.



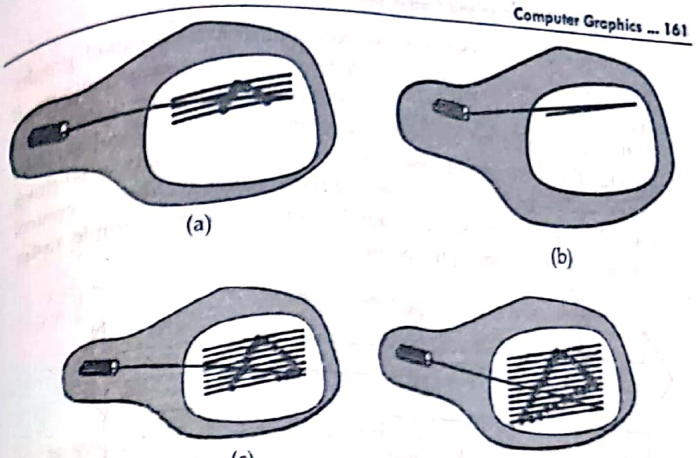


Fig: A raster-scan system displays an object as a set of points across each screen scan line

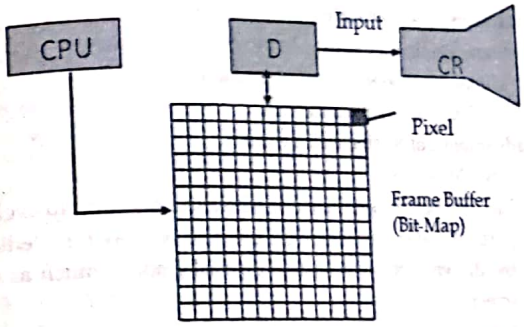


Fig: Raster Scan display system

The stored intensity value is retrieved from frame buffer and painted on the scan line at a time. The stored intensity value is retrieved from frame buffer and painted on the screen. Refreshing on Raster-Scan display is carried out at the rate of 60 or higher frames per second. 60 frames per second is also termed as 60 cycle per second usually used unit Hertz (HZ). Most of display devices are based on this technology. For example, CRT, color CRT, LCD, and LED etc.

Architecture of Raster-Scan System

The raster graphics systems typically consist of several processing units. CPU is the main processing unit of computer systems. Besides CPU, graphics system consists of a special buffer processor called video controller or display controller. It is used to control the operation of the display device. In addition, to the video controller, raster scan systems

can have other processors as co-processors which are called graphic controller or display processors. A fixed area of system memory is reserved for the frame buffer. The video controller has the direct access to the frame buffer for refreshing the screen. The video controller cycles through the frame buffer, one scan line at a time, typically at 60 times per second or higher. The contents of frame buffer are used to control the CRT beam's intensity or color. The organization of simple raster system is shown in the figure below.

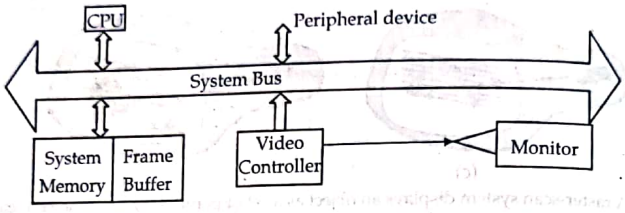


Fig: Architecture of simple Raster-scan system.

Advantages of Raster scale system

- Show Realistic pictures
- Million Unique hues can be produced
- Shadow scenes are conceivable

Disadvantages of Raster scale system

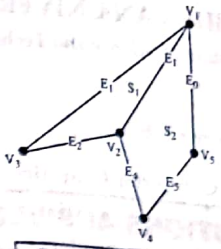
- Low Resolution
- Electron beam coordinated to whole screen not exclusively to those parts of the screen where picture is to be drawn so tedious when the drawn picture estimate is especially not as much as the whole screen.
- Expensive

12. Write short notes on (Any TWO) [2.5 +2.5]

a. Polygon Tables

Ans: In this method, the surface is specified by the set of vertex coordinates and associated attributes. As shown in the following figure, there are five vertices, from v_1 to v_5.

- Each vertex stores x, y, and z coordinate information which is represented in the table as v_1: x_1, y_1, z_1.
- The Edge table is used to store the edge information of polygon. In the following figure, edge E_1 lies between vertex v_1 and v_2 which is represented in the table as E_1: v_1, v_2.
- Polygon surface table stores the number of surfaces present in the polygon. From the following figure, surface S_1 is covered by edges E_1, E_2 and E_3 which can be represented in the polygon surface table as S_1: E_1, E_2, and E_3.



POLYGON-SURFACE TABLE
S_1: E_1, E_2, E_3
S_2: E_3, E_4, E_5, E_6

b. Augmented Reality

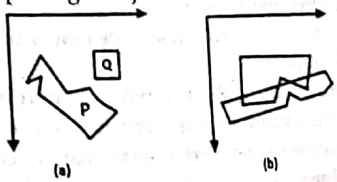
Ans: Augmented reality (AR) adds digital elements to a live view often by using the camera on a smartphone. Examples of augmented reality experiences include Snap chat lenses and the game Pokemon Go. Virtual reality (VR) implies a complete immersion experience that shuts out the physical world.

Augmented reality is defined as "an enhanced version of reality created by the use of technology to add digital information on an image of something."AR is used in apps for smart phones and tablets. AR apps use your phone's camera to show you a view of the real world in front of you, then put a layer of information, including text and/or images, on top of that view. Apps can use AR for fun, such as the game Pokémon GO, or for information, such as the app Layer. The Layer app can show you interesting information about places you visit, using augmented reality. Open the app when you are visiting a site and read information that appears in a layer over your view.

c. Painter's algorithm

Ans: It came under the category of list priority algorithm. It is also called a depth-sort algorithm. In this algorithm ordering of visibility of an object is done. If objects are reversed in a particular order, then correct picture results. Objects are arranged in increasing order to z coordinate. Rendering is done in order of z coordinate. Further objects will obscure near one. Pixels of rear one will overwrite pixels of farther objects. If z values of two overlap, we can determine the correct order from Z value as shown in fig (a).

If z objects overlap each other as in fig (b) this correct order can be maintained by splitting of objects.



TRIBHUVAN UNIVERSITY

Institution of Science and Technology

Course Title: Computer Graphics
Course No: CSC 209
Nature of the Course: Theory + Lab
Semester: III

Full Marks: 60
Pass Marks: 24
Time: 3 hrs

Computer Graphics

TU QUESTIONS-ANSWERS 2075

Section A

Attempt any two questions. (2 ×10 =20)

1. Define window, viewport and viewing transformation. Let ABCD be the rectangular window with A(20, 20), B(90, 20), C(90, 70), D(20, 70). Find the region codes for end points and use Cohen Sutherland algorithm to clip the lines P(10, 20), Q(80, 90). 3+7=10

Ans: Window

A world-coordinate area selected for display is called a window. That is, window is the section of the 2D scene that is selected for viewing. The window defines what is to be viewed.

Viewport

An area on a display device to which a window is mapped is called a viewport. The viewport indicates where on an output device selected part will be displayed. Figure given below illustrates the mapping of a 2D picture that falls within a rectangular window onto a designated rectangular viewpoint.

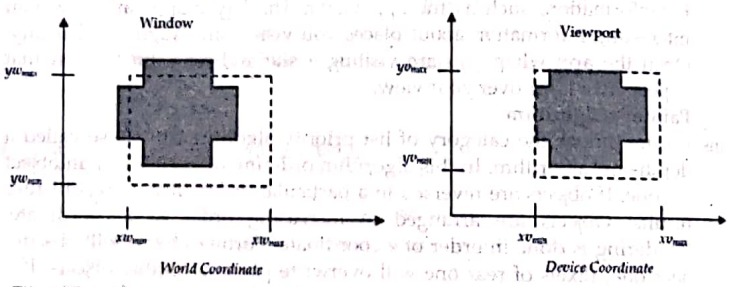


Figure: World and Viewport Coordinates, specified as rectangles aligned with the co-ordinate axes.

Window and viewport are often rectangular in standard positions, because it simplifies the transformation process and clipping process. Other shapes such as polygons, circles take longer time to process.

Viewing transformation

Viewing Transformation is the mapping of coordinates of points and lines that form the picture into appropriate coordinates on the display device. The window-to-viewport transformation can be explained in three steps as below:

1. Translate object along with window such that the lower left corner of the window is at the origin. That is, apply T $(-xw_{min}, -yw_{min})$.

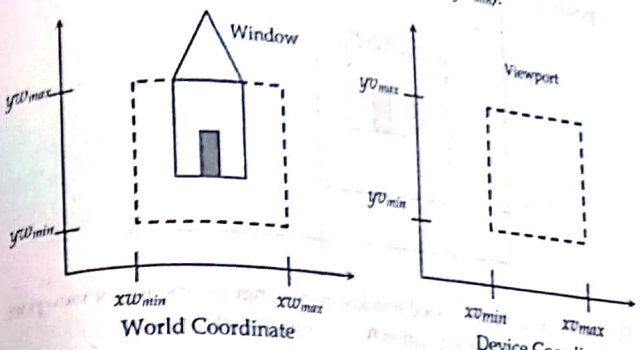


Figure: Window port and viewport before coordinate transformation

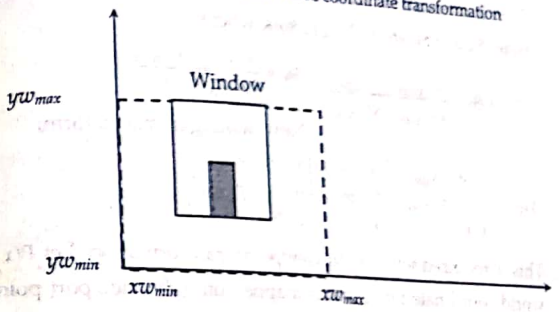


Figure: Translation of window to origin

2. Scale the object and the window such that window has the same dimension as that of a viewport. Simply we can say that we are converting the object into image and window in viewport. That is, apply S (sx, sy).

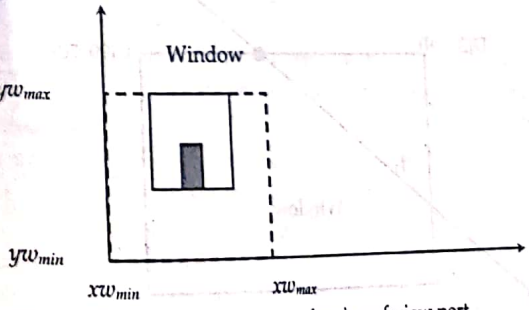


Figure: Scaling the window to the size of view port

3. Finally, apply another translation to move the viewport to its original position on the screen. That is, apply $T(xv_{min}, yv_{min})$.

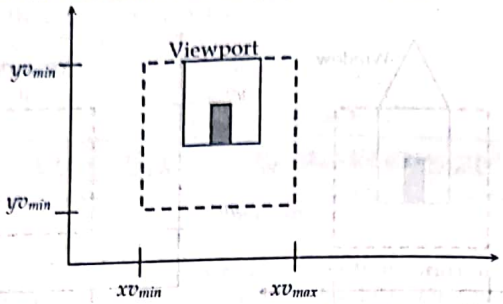


Figure: Translating the scaled window to the original position of viewport.

Therefore, net transformation (Composite Transformation),

$T_{WV} = T(xv_{min}, yv_{min}).\ S(S_x, S_y).T(-xw_{min}, -yw_{min})$ .....(i)

Here, *Sx* and *Sy* are scaling factors, where;

$$Sx = \frac{xv_{max} - xv_{min}}{xw_{max} - xw_{min}} \qquad Sy = \frac{yv_{max} - yv_{min}}{yw_{max} - yw_{min}}$$

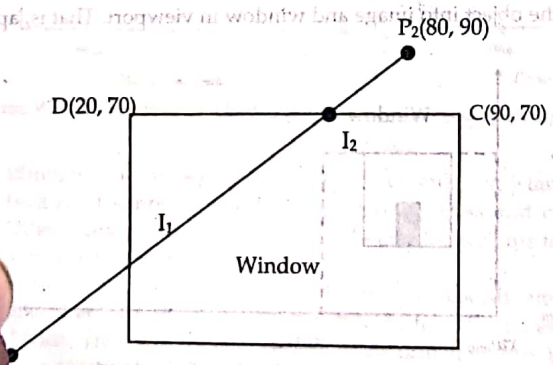
Now, writing in matrix form;

$$Twv = \begin{bmatrix} 1 & 0 & xv_{min} \\ 0 & 1 & yv_{min} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} Sx & 0 & 0 \\ 0 & Sy & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -yw_{min} \\ 0 & 1 & -yw_{min} \\ 0 & 0 & 1 \end{bmatrix}$$

This is required window-to-viewpoint transformation. Let $P(x, y)$ be the world coordinate point that is mapped onto the viewport point $P'(u, v)$, then we must have:

$P' = TWV.P$

Numerical Part



Since the point P_1 lies left outside the window hence its outcode is 0001
Similarly the outcode of P_2=1000
The outcode for I_1 is 0001
And outcode for I_2 is 1000
The Logical AND for P_1P_2 = 0001 AND 1000 = 0000.
Which is zero hence line is partially visible.
Since left edge of window is having x coordinate as 20. The intersection points z-coordinate will also be 20 only. Now we have to find y-coordinate of intersection point. For this we will use slope intercept form of line.
i.e. let's find slope of line P_1P_2,
Hence, m = = = = 0.8

So, slope of line p_1p_2 is 0.8
We know that the slope between two endpoints of a line is same as that of slope between one endpoint and any other point which is lying on the line.
Hence slope between points P1 and intersection point will be,
m =

where, x1 and y1 are related with P_1, x_2 is known i.e. 20, y_2 we have to find.
Hence, $m(x_2 - x_1) = y_2 - y_1$
⇨ $y_2 = m(x_2 - x_1) + y1$
⇨ $y_2 = 0.8\ (20 - 10) + 30$
$= 0.8 * 10 + 30 = 8 + 30 = 38$
⇨ $y_2 = 38$
Therefore the intersection point is (20, 38). Let's call this point as I1. Since I1 point is exactly at the boundary. Its region code will be (0, 0, 0, 0).
Now we have to discard the line segment $P_1 I_1$ since P_1 point is outside window.
Similarly we have to find another intersection point I_2 with respect to top boundary. Now I_2 points region.
Code will be 0000. So we will discard I_2P_2 since P_2 point is above the window.
Now our line segment is I_1I_2. Both I_1 and I_2 's region codes are 0000. So the line I_1I_2 is completely visible and we have to display I_1I_2 line.

2. **List any two advantages of BSP tree method in visible surface detection. Make a compression between Painter's algorithm and A-Buffer algorithm.** **(2+8=10)**

Ans: A binary space partitioning (BSP) tree is an efficient method for determining object visibility by painting surfaces onto the screen from back to front as in the painter's algorithm. The BSP tree is particularly useful when the view reference point changes, but object in a scene are at fixed position. Applying a BSP tree to visibility testing involves

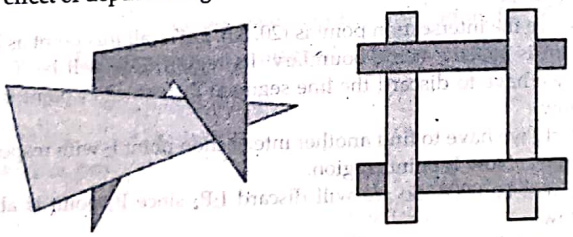
identifying surfaces that are "inside" or "outside" the partitioning plane at each step of space subdivision relative to viewing direction. It is useful and efficient for calculating visibility among a static group of 3D polygons as seen from an arbitrary viewpoint.

Depth Sorting Method

The basic idea of this method is simple. When there are only a few objects in the scene, this method can be very fast. However, as the number of objects increases, the sorting process can become very complex and time consuming. This method uses both object space and image space method. The depth-sorting method performs two basic functions:

- First, the surfaces are sorted in order of decreasing depth.
- Second, the surfaces are scan-converted in order, starting with the surface of greatest depth.

The intensity values for farthest surface are then entered into the refresh buffer. That is farthest polygon is displayed first, then the second farthest polygon, so on, and finally, the closest polygon surface. After all surfaces have been processed, the refresh buffer stores the final intensity values for all visible surfaces. When there are only a few objects in the scene, this method can be very fast. However, as the number of objects increases, the sorting process can become very complex and time consuming. The scan conversion of the polygon surfaces is performed in image space. This method for solving the hidden-surface problem is often referred to as the painter's algorithm. The following figure shows the effect of depth sorting:



In this method, the newly displayed surface is partly or completely obscures the previously displayed surface. Essentially, we are sorting the surface into priority order such that surface with lower priority (lower z, far objects) can be obscured by those with higher priority (high z-value).

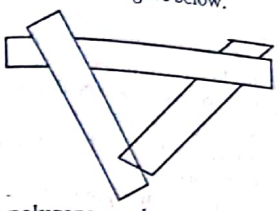
Advantages

- The sorting computation is very fast, so a quick calculation of the image display is possible.
- The finished image data remains object based (sorted polygons), and can be edited in terms of line weight, exploded views, etc.

- It will also print at the highest resolution available on devices such as postscript laser printers.

Limitations

- Certain conditions, such as intersecting polygons or cyclic overlap (A covers part of B, which covers part of C, which covers part of A), are not processed correctly, since parts of each should be visible, but the algorithm will always yield a sorted list. The problem described as shown in figure below.



- Different polygons may have same depth.
- The nearest polygon could also be farthest.
- We cannot use simple depth-sorting to remove the hidden-surfaces in the images.

A-Buffer method

A-Buffer method is an extension of depth buffer method. A drawback of the depth buffer method is that it can only find one visible surface at each pixel position. That means, it deals only with opaque surface and cannot accumulate intensity values for more than one surface, as is necessary if transparent surfaces are to be displayed.

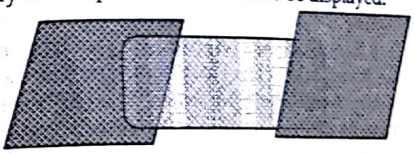


Figure: Viewing two background opaque surfaces through a foreground transparent surface

The A-buffer method expands the depth buffer so that each position in the buffer can reference a linked list of surfaces. Thus more than one surface intensity can be taken into consideration at each pixel position. Each pixel position in the A-buffer has following fields:

- **Depth field:** It stores a positive or negative real numbers
- **Intensity field:** It stores surface intensity information or a pointer value.

If the depth field is positive, the number stored at that position is the depth of a single surface overlapping the corresponding pixel area. If the depth field is negative, this indicates multiple surface contributions to the pixel intensity.

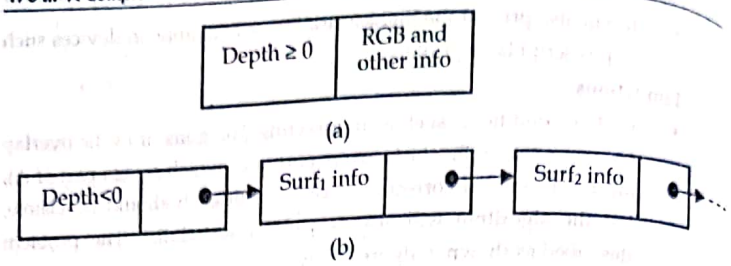


Figure: Organization of an A-buffer pixel position: (a) single-surface (b) multiple surfaces overlap

As shown in the above figure (a), if the value of depth is >= 0, the number stored at that position is the **depth of single surface** overlapping the corresponding pixel area. The 2nd field, i.e., the intensity field then stores the RGB components of the surface color at that point and the percent of pixel coverage.

As shown in the above figure (b), **multiple-surface contributions** to the pixel intensity are indicated by depth < 0. The 2nd field, i.e., the intensity field then stores a pointer to a linked list of surface data.

3. **Describe the architecture of raster scan display. Explain about sweep, octree and boundary representations for solid modeling.** (4+6=10)

Ans: Architecture of Raster-Scan System

The raster graphics systems typically consist of several processing units. CPU is the main processing unit of computer systems. Besides CPU, graphics system consists of a special buffer processor called video controller or display controller. It is used to control the operation of the display device. In addition, to the video controller, raster scan systems can have other processors as co-processors which are called graphic controller or display processors. A fixed area of system memory is reserved for the frame buffer. The video controller has the direct access to the frame buffer for refreshing the screen. The video controller cycles through the frame buffer, one scan line at a time, typically at 60 times per second or higher. The contents of frame buffer are used to control the CRT beam's intensity or color. The organization of simple raster system is shown in the figure below.

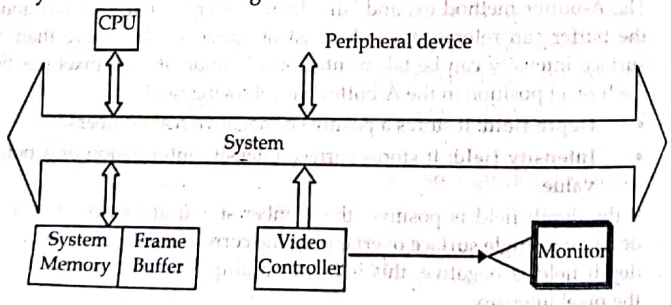
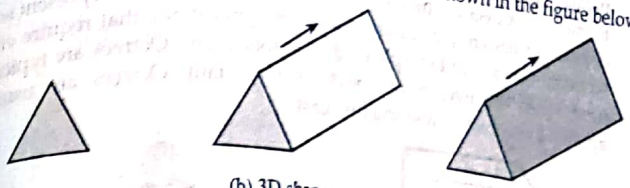


Figure: Architecture of simple Raster-scan system.

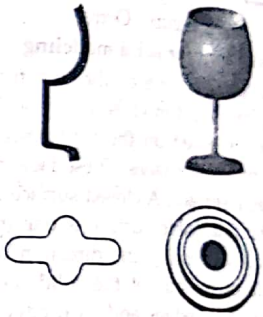
Sweep Representations

Sweep representations are used to construct three dimensional objects from two dimensional shape .There are two ways to achieve sweep: **Translational sweep and Rotational sweep.** In translational sweeps, the 2D shape is swept along a linear path normal to the plane of the area to construct three dimensional objects. To obtain the wireframe representation we have to replicate the 2D shape and draw a set of connecting lines in the direction of shape, as shown in the figure below,



(b) 3D shape (c) Wire frame representation

In rotational sweeps, the 2D shape is rotated about an axis of rotation specified in the plane of 2D shape to produce three dimensional objects. This is illustrated in figure below,



In general we can specify sweep constructions using any path. For translation we can vary the shape or size of the original 2D shape along the sweep path. For rotational sweeps, we can move along a circular path through any angular distance from 0° to 360°. These sweeps whose generating area or volume changes in size, shape or orientation as they are swept and that follow an arbitrary curved trajectory are called general sweeps .General sweeps are difficult to model efficiently for example, the trajectory and object shape may make the swept object intersect itself, making volume calculations complicated. Furthermore, general sweeps do not always generate solids. For example, sweeping a 2D shape in its own plane generates another 2D shape.

Octree

Octree are three dimensional quadtrees, its three dimensions are recursively subdivided into octants and have quadrants. The number of nodes in a quadtree or octree is proportional to the object's perimeter or surface, respectively because subdivision occurs only from the need to represent an object's boundary. Therefore, subdivision only occurs in those quadrants where a boundary passes. Boolean set operators can also apply to both quadtrees and octrees by traversing the two trees in parallel. Individual partitions of 3D space are called Voxels (Volume Elements). Octrees are hierarchical tree structures used to represent solid objects Octrees are particularly useful in applications that require cross sectional views – for example medical applications. Octrees are typically used when the interior of objects is important. Octrees are usually applied in ray casting and shadow casting.

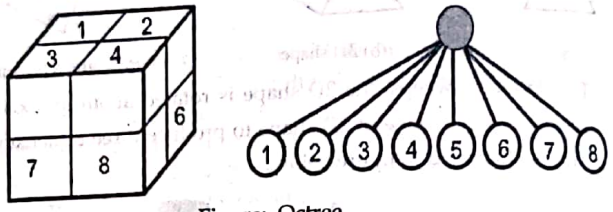


Figure: Octree

Boundary representations for solid modeling

Boundary representation is one of the two most popular and widely used schemes to create solid models of physical objects. A B-rep model or boundary model is based on the topological notion that a physical object is bounded by a set of faces. These faces are regions or subsets of closed and orientable surface. A closed surface is one that is continuous without breaks. An orientable surface is one in which it is possible to distinguish two sides by using the direction of the surface normal to point to the inside or outside of the solid model under construction. Each face is bounded by edges and each edge is bounded by vertices. Thus, topologically, a boundary model of an object is comprised of faces, edges, and vertices of the object linked together in such a way as to ensure the topological consistency of the model.

Section B

Short Answer Questions

Attempt any *EIGHT* questions **[8 × 5 = 40]**

4. Given some basic color model. Give the basic command to draw the pixel and polygon in OpenGL. **(2+3)**

Ans: Basic color model

A color space or color model is a method by which we can specify, create and visualize color. A color model is a method for explaining the properties or behavior of color within particular context. It is a mathematical way of representing a set of colors. A color model is an

orderly system for creating a whole range of colors from a small set of primary colors. There is several established color models used in computer graphics, RGB, CMY, CMYK, YIQ etc. But the two most common are the RGB model (Red-Green-Blue) for computer display and the CMYK model (Cyan-Magenta-Yellow-black) for computer display and Color models, but fall into two broad categories:

- Additive Color Model and
- Subtractive Color Model

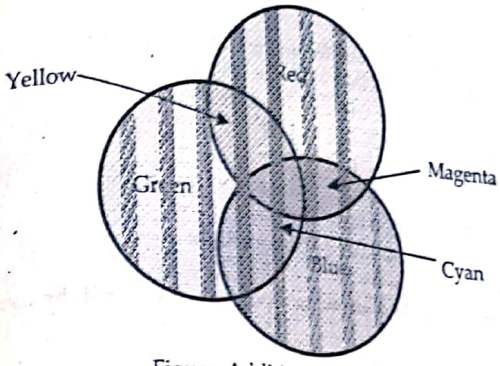


Figure: Additive color model

Points

A point is represented by a set of floating-point numbers called a vertex. All internal calculations are done as if vertices are three-dimensional. Vertices specified by the user as two-dimensional (that is, with only x and y coordinates) are assigned a z coordinate equal to zero by OpenGL. OpenGL works in the homogeneous coordinates of three-dimensional projective geometry, so for internal calculations, all vertices are represented with four floating-point coordinates (x, y, z, w). If w is different from zero, these coordinates correspond to the Euclidean three-dimensional point (x/w, y/w, z/w).

To control the size of a rendered point, use **glPointSize()** and supply the desired size in pixels as the argument.

void **glPointSize**(GLfloat size);

Sets the width in pixels for rendered points; size must be greater than 0.0 and by default is 1.0.

Polygon

Polygons are typically drawn by filling in all the pixels enclosed within the boundary, but you can also draw them as outlined polygons or simply as points at the vertices. A filled polygon might be solidly filled or stippled with a certain pattern. Although the exact details are omitted here, filled polygons are drawn in such a way that if adjacent polygons share an edge or vertex, the pixels making up the edge or vertex are drawn exactly once—they're included in only one of the polygons. This

is done so that partially transparent polygons don't have their edges drawn twice, which would make those edges appear darker (or brighter, depending on what color you're drawing with).
A polygon has two sides—front and back—and might be rendered differently depending on which side is facing the viewer. This allows you to have cutaway views of solid objects in which there is an obvious distinction between the parts that are inside and those that are outside. By default, both front and back faces are drawn in the same way. To change this, or to draw only outlines or vertices, use glPolygonMode().
We can draw polygon by following code of segment;

```
glBegin(GL_POLYGON); // Draw A Quad
glVertex3f(-0.5f, 1.0f, 0.0f); // Top Left
glVertex3f(-1.0f, 0.0f, 0.0f); // Left
glVertex3f(-0.5f, 1.0f, 0.0f); // Bottom Left
glVertex3f(0.5f, 1.0f, 0.0f); // Top Right
glVertex3f(1.0f, 0.0f, 0.0f); // Right
glVertex3f(0.5f, 1.0f, 0.0f); // Bottom Right
glEnd();
```

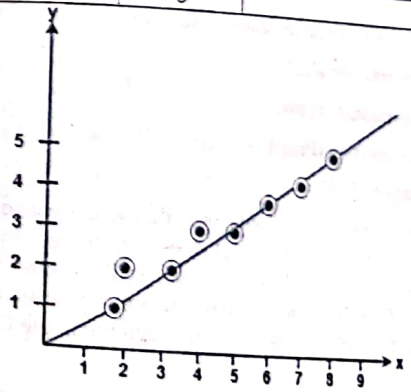
Complete example of drawing a polygon in OpenGL by using c/c++ as below:

```
#include <windows.h>
#include <stdio.h>
include <conio.h>
#include <iostream.h>
#include <stdlib.h>
#include <glut.h>
void myInit(void)
{
glClearColor(1.0,1.0,1.0,0.0);
glColor3f(1.0f,0.0f,0.0f);
glPointSize(4.0);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluOrtho2D(0.0,800.0,0.0,600.0);
}
void myDisplay(void)
{
glClear(GL_COLOR_BUFFER_BIT);
glBegin(GL_POLYGON);
glVertex2i(100,100);
glVertex2i(100,300);
glVertex2i(400,300);
glVertex2i(600,150);
glVertex2i(400,100);
glEnd(); glFlush();
}
int main(int argc,char * argv[])
{
glutInit(&argc,argv);
glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
glutInitWindowSize(800,600);
glutInitWindowPosition(100,100);
glutCreateWindow("opengl Window");
glutDisplayFunc(myDisplay);
myInit(); glutMainLoop();
getch();
return 0;
}
```


5. **Trace the Bresenham's Line drawing algorithm for the end points (1, 1) and (8, 5).** (5)

Ans: x1=1
y1=1
x2=8
y2=5
dx= x2-x1=8-1=7
dy=y2-y1=5-1=4
$I1=2^* \Delta y=2^*4=8$
$I2=2^*(\Delta y-\Delta x)=2^*(4-7)=-6$
d = I1-Δx=8-7=1

x	y	$d=d+I_1$ or I_2
1	1	$d+I_2=1+(-6)=-5$
2	2	$d+I_1=-5+8=3$
3	2	$d+I_2=3+(-6)=-3$
4	3	$d+I_1=-3+8=5$
5	3	$d+I_2=5+(-6)=-1$
6	4	$d+I_1=-1+8=7$
7	4	$d+I_2=7+(-6)=1$
8	5	



6. Derive the relation for tree-dimensional translation and rotation. (5)

Ans: Translation

Translation is used to move a point, or a set of points, linearly in space. Since now we are talking about 3D, therefore each point has 3 coordinates i.e. x, y and z. Similarly, the translation distances can also be specified in any of the 3 dimensions. These Translation Distances are given by t_x, t_y and t_z.

For any point $P(x, y, z)$ after translation we have $P'(x', y', z')$ where

$x' = x + t_x$,

$y' = y + t_y$,

$z' = z + t_z$

And (t_x, t_y, t_z) is Translation vector.

Now this can be expressed as a single matrix equation.

$P' = P + T$

Where,

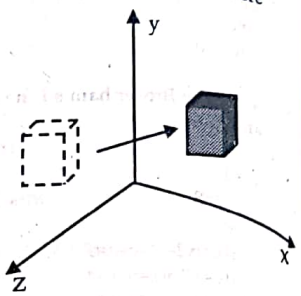


Figure: 3D Translation

$$P = \begin{bmatrix} x \\ y \\ z \end{bmatrix}, \quad P' = \begin{bmatrix} x' \\ y' \\ z' \end{bmatrix}, \text{ and } T = \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix}$$

Rotation

Rotation is the process of moving a point in space in a non-linear manner. More particularly, it involves moving the point from one position on a sphere whose center is at the origin to another position on the sphere. 3D rotation is not same as 2D rotation. In 3D rotation, we have to specify the angle of rotation along with the axis of rotation. We can perform 3D coordinate axes rotation as

- Z-axis rotation (Roll)
- Y-axis rotation (Yaw)
- X-axis rotation (Pitch)

Z-axis rotation (Roll)

If you look closely, you should note that when we rotate around the Z axis, the Z element of the point does not change. In fact, we can just ignore the Z – we already know what it will be after the rotation. If we ignore the Z element, then we have the same case as if we were rotating the two-dimensional point <x, y> through the angle θ. Z-axis rotation is

same as the origin about the 2D for which we have the derived matrices already.

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

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

$z' = z$

Homogeneous representation is:

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

i.e. $P' = R_z(\theta).P$

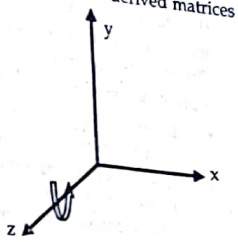


Figure: Roll

Y-axis rotation (Yaw)

The equations for Y-axis rotation

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

$y' = y$

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

Homogeneous representation is:

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

i. e. $P' = R_y(\theta).P$

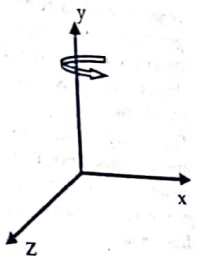


Figure: Yaw

X-axis rotation (Pitch)

The equations for X-axis rotation

$x' = x$

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

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

Homogeneous representation is:

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

i.e. $P' = R_x(\theta).P$

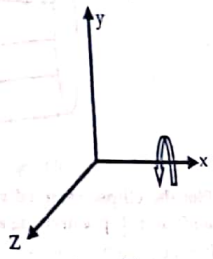


Figure: Pitch

7. What is the purpose of wireframe representation? Describe about boundary and space partitioning. (2+3)

Ans: A wireframe is a three-dimensional model that only includes vertices and lines. It does not contain surfaces, textures, or lighting like a 3D mesh. Instead, a wireframe model is a 3D image comprised of only "wires" that represent three-dimensional shapes.

Wireframes provide the most basic representation of a three-dimensional scene or object. They are often used as the starting point in 3D modeling since they create a "frame" for 3D structures. For example, a 3D graphic designer can create a model from scratch by simply

defining points (vertices) and connecting them with lines (paths). Once the shape is created, surfaces or textures can be added to make the model appear more realistic.

Objects are represented as a collection of surfaces. 3D object representation is divided into two categories.

- Boundary Representations B-reps – It describes a 3D object as a set of surfaces that separates the object interior from the environment.
- Space-partitioning representations – It is used to describe interior properties, by partitioning the spatial region containing an object into a set of small, non-overlapping, contiguous solids usually cubes.

The most commonly used boundary representation for a 3D graphics object is a set of surface polygons that enclose the object interior. Many graphics system use this method. Set of polygons are stored for object description. This simplifies and speeds up the surface rendering and display of object since all surfaces can be described with linear equations.

The polygon surfaces are common in design and solid-modeling applications, since their wireframe display can be done quickly to give general indication of surface structure. Then realistic scenes are produced by interpolating shading patterns across polygon surface to illuminate.

Fig: 3D object represented by Polygons

8. **Plot the ellipse centered at (0, 0) with radios $r_x = 8$ and $r_y = 6$, using midpoint ellipse drawing algorithm.** (5)

Ans: Input ellipse parameters $r_x = 8$ and $r_y = 6$ the midpoint ellipse algorithm by determining raster position along the ellipse path is the first quadrant. Initial values and increments for the decision parameter calculations are:

$2r_{y^2} x = 0$ (with increment $2r_y^2 = 72$)

$2r_x^2 y = 2r_x^2 r_y$ (with increment $- 2r_x^2 = -128$)

For region 1 the initial point for the ellipse centered on the origin is $(x_0, y_0) = (0, 6)$ and the initial decision parameter value is:

$p_{10} = r_y^2 - r_x^2 r_y^2 + 1/4 r_x^2 = -332$

Successive midpoint decision parameter values and the pixel positions along the ellipse are listed in the following table.

k	pl_k	x_{k+1}, y_{k+1}	$2r_y^2 x_{k+1}$	$2r_x^2 y_{k+1}$
0	– 332	(1, 6)	72	768
1	– 224	(2, 6)	144	768
2	– 44	(3, 6)	216	768
3	208	(4, 5)	288	640
4	– 108	(5, 5)	360	640
5	288	(6, 4)	432	512
6	244	(7, 3)	504	384

Move out of region 1, $2r_y^2 x > 2r_x^2 y$

For a region 2 the initial point is $(x_0, y_0) = (7, 3)$ and the initial decision parameter is,

$p_{20} = f_{ellipse}(7+1/2, 2) = -151$

The remaining positions along the ellipse path in the first quadrant are then calculated as,

k	$p2_k$	x_{k+1}, y_{k+1}	$2r_y^2 x_{k+1}$	$2r_x^2 y_{k+1}$
0	– 151	(8, 2)	576	256
1	233	(8, 1)	576	128
2	745	(8, 0)	-	-

9. **Define clipping. Discuss about cubic spline interpolation.** (2+3)

Ans: Clipping

The process of discarding (cutting off) those parts of a picture which are outside of a specified region (windows) is called clipping. Any procedure that identifies those parts of a picture that are either inside or outside of the specified region is called a clipping algorithm. The region against which the clipping operation is performed is called a clip window.

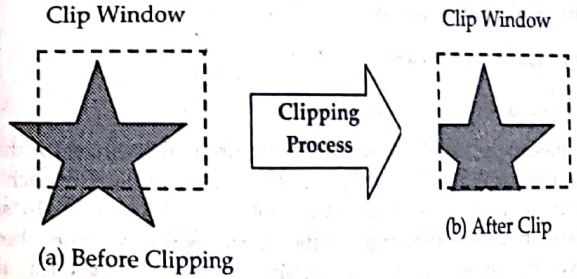


Figure: Clipping process (a) before clipping (b) after clipping

Cubic spline interpolation

The cubic spline was originally introduced by James Ferguson. Spline was a term for elastic rulers that were bent to pass through a number of predefined points ("knots"). These were used to make technical drawings for shipbuilding and construction by hand. Cubic spline interpolation is a special case for Spline interpolation that is used very often to avoid the problem of Runge's phenomenon. This method gives an interpolating polynomial that is smoother and has smaller error than some other interpolating polynomials such as Lagrange polynomial and Newton polynomial. Cubic polynomials offer a reasonable compromise between flexibility and speed of computation. Cubic spline requires less calculations compares to higher order polynomials and less memory. Compared to lower polynomials cubic spline are more flexible for modeling arbitrary curve shape.

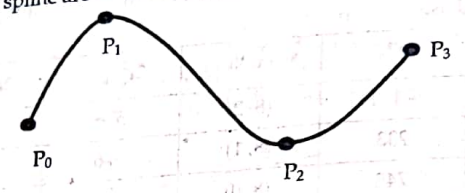


Figure: Interpolation with cubic splines between 4 control points

Given a set of control points, cubic interpolation splines are obtained by fitting the input points with a piecewise cubic polynomial curve that passes through every control points.

Suppose we have n+1 control points specified with co-ordinates

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

The parametric cubic polynomial that is to be filled between each pair of control points with the following set of equations.

$$x(u) = a_x u^3 + b_x u^2 + c_x u + dx$$

$$y(u) = a_y u^3 + b_y u^2 + c_y u + dy \quad (0 \le u \le 1)$$

$$z_u(u) = a_z u^3 + b_z u^2 + c_z u + dz$$

For these three equations, we need to determine the values for the four coefficients a, b, c, and d in the polynomial representation for each of the n curve sections between the n+1 control points. We do this by setting enough boundary conditions at the control-point positions between curve sections so that we can obtain numerical values for all the coefficients.

10. **How can we detect shadows in computer graphics? List the challenged in computing light and manipulating interfaces in virtual reality? Justify.** (2+3)

Ans: Shadows appear in many scenes. Human can easily distinguish shadows from objects, but it is one of the challenges for shadow detection intelligent automated systems. Accurate shadow detection can be difficult due to the illumination variations of the background and similarity between appearance of the objects and the background. Color and edge information are two popular features that have been used to distinguish cast shadows from objects. However, this become a problem when the difference of color information between object, shadow and background is poor, the edge of the shadow area is not clear and the shadow detection method is supposed to use only color or edge information method.

Challenges of virtual reality are listed below:

VR as a New Medium

We should say a word of warning here before continuing: adapting existing applications to VR is difficult if they weren't designed for this from the outset. VR is like radio or TV at their beginnings: radio was only used to broadcast opera, and TV was only used to broadcast theater plays. Slowly, people started to create content specifically tailored for these new media. Camera movement, zoom, and cuts created a new grammar for film, for instance.

Latency

The first enemy of VR is latency. If you move your head in the real world and the resulting image takes one second to appear, your brain will not accept that this image is related to the head movement. Moreover as a result, you will probably get sick.

Interactions

Interactions with a 3D/VR world are more complex than it seems. There are even several scientific conferences dedicated to this sole topic: 3D User Interfaces (3DUI), Spatial User Interfaces (SUI). The main problem is that you don't have access to a regular keyboard or mouse. The second problem is that interacting in 3D is a hard ergonomic problem!

There are multiple ways of interacting in VR:

- Navigation in an environment,
- Selecting and manipulating an object,
- Menus and graphical user interfaces (GUIs),
- Entering numbers and text,

All those tasks can be accomplished in a lot of different ways which depend on a particular application, hardware and even user! Think of the navigation: it can be done with a simple joystick, by pointing a particular destination in space with a hand-held device, by saying out loud the destination, by walking in place, by gestures, by looking at it, by picking it on a map.

Avatars

When you are using a head-mounted display (HMD), you are completely immersed in the virtual world, and you don't see your own body anymore. It is very important to display a virtual representation of yourself and others, called avatars. They can be realistic, look like yourself, or be completely different.

If your VR system offers full body tracking and if you want an avatar that has exactly the same dimensions as you, this can be a simple task. But if your VR system only has a few trackers and you want an avatar that is taller or smaller than you, it can be difficult to extrapolate the position of the limbs that are not tracked and to adapt your body posture to a different virtual body.

Collaboration

Once you start being in a virtual world, you very quickly want to share the experience and not be alone. VR is particularly suited for collaborative work with people physically in the same space or in completely different parts of the world! When creating a collaborative application, you need to make sure each VR application is connected to one server, that all data between the applications are synchronized securely, that you can see the avatar of others.

High-end VR systems

There are specific issues when dealing with high-end VR systems such as CAVEs(tm), domes, workbenches etc:

- computing the correct perspective depending on the position of the user,
- Displaying different types of stereoscopy (active, passive, auto stereoscopic, side-by-side etc.),
- managing multi-screens and/or multiple graphics cards,
- managing multiple computers and the synchronization of the application (Frame lock), display of new images (Swap lock) and display of stereoscopic images (Gen lock)
- managing the warping and blending of several projectors projecting on a non-planar surface,
- haptics force-feedback, etc.

Running on different VR hardware

When your application is finished, you might want to run it on a different hardware: You might have created your application for the Oculus Rift and want to try with another HMD, or maybe a stereoscopic wall or a Cave.

A Coherent World, Not Necessarily a Realistic One

We have seen that perceptive presence requires you to fool your senses in the most realistic way. Cognitive presence – fooling the mind, not the senses – results from a sense that your actions have effects on the virtual environment, and that these events are credible. This means that you must believe in the "rules" of the simulation. For this, you must

make sure that your world is coherent, not necessarily realistic. If a player can grab a particular glass, for example, but can't grab another one, it will break presence because the rules are not consistent. Once cognitive presence is broken, it's very difficult to "fix" it. The player is constantly reminded that the simulation is not real, and it will take some time to accept it again as reality.

11. List some applications of VR. What might be the possible navigation techniques and manipulating interfaces in virtual reality? Justify. (2+3)

Ans: Application area of virtual reality

Here is a list of the many applications of virtual reality:

- In the Military
- In Education
- in Healthcare
- in Entertainment
- Fashion
- Heritage
- in Business
- in Engineering
- in Sport
- in Media
- Scientific Visualizations
- in Telecommunications
- in Construction
- in Film
- Programming languages

The layout of the action-perception space is dependent on whether the user is left-handed or right-handed. The layout described below is meant for a right handed person. The layout consists of a linearly structured image database browser, virtual paper (7), a floating toolbar (6), EPP (9) and a 3to2D window (8) in the action perception space. The communication space is currently used to provide a surface rendering of the 3D model. This surface model is used to visualize the position of the RISP with respect to the 3D data.

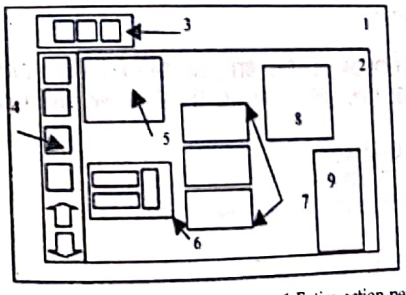
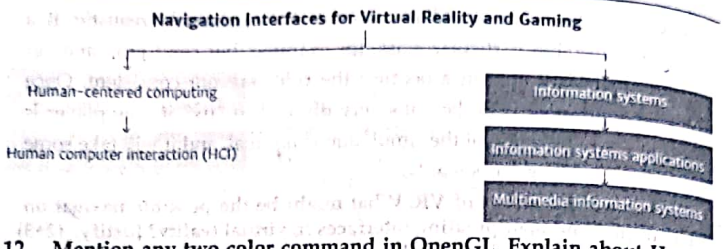


Figure: The action-perception space layout. 1-Entire action-perception space, 2-Sketchable area (UltraPad), 3, 4, 5-Image database browser, 6- Floating toolbar, 7-Virtual Papers, 8 – 3 to 2D window, 9 – Enhanced Paper Prop.



12. Mention any two color command in OpenGL. Explain about Hermite curve. (2+3)

Ans: Color command in OpenGL

OpenGL has two color modes: the RGBA mode and color index mode. In RGBA mode, a color is specified by three intensities (for the Red, Green, and Blue components of the color) and optionally a fourth value, Alpha, which controls transparency.

The function glColor4f (red, green, blue, alpha)

Maps available red, green, and blue intensities onto (0.0, 1.0) where 0.0 means that the color component is absent ((0.0, 0.0, 0.0) is black) and 1.0 is a saturated color ((1.0, 1.0, 1.0) is white.) The number of bits used to represent each color depends upon the graphics card. Current graphics cards have several Mbytes of memory, and use 24 or 32 bits for color (24-bit: 8 bits per color; 32-bit: 8 bits per color + 8 bits padding or transparency). The term bit plane refers to an image of single-bit values. Thus, a system with 24 bits of color has 24 bit planes.

For a system with 8 bit planes, 28 = 256 different colors could be displayed simultaneously. These 256 colors could be selected from a set of 2m colors. For m = 24 this gives 22 4 ≈ 16 million colors. The color table could be altered to give different palettes of colors. OpenGL supports this mode, called color index mode.

Using glColor3f

glColor3f() takes 3 arguments: the red, green and blue components of the color. After we use glColor3f, everything draw will be in that color. For example, consider this display function:

```
void display()
{
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
glColor3f(0.5f, 0.0f, 1.0f); // (0.5, 0, 1) is half red and full blue, giving purple.
glBegin(GL_QUADS);
glVertex2f(-0.75, 0.75);
glVertex2f(-0.75, -0.75);
glVertex2f(0.75, -0.75);
glVertex2f(0.75, 0.75);
glEnd();
glutSwapBuffers();
}
```


Hermite curve

Hermite curves are very easy to calculate but also very powerful. They are used to smoothly interpolate between key-points (like object movement in key frame animation or camera control). Understanding the mathematical background of hermite curves will help you to understand the entire family of splines. Maybe you have some experience with 3D programming and have already used them without knowing that (the so called kb-splines, curves with control over tension, continuity and bias are just a special form of the hermite curves).

To keep it simple we first with 2-dimensional curves here. Hermite curves work in any number of dimensions. To calculate a hermite curve you need the following vectors:

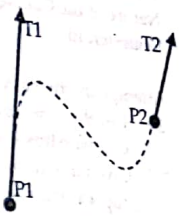


Figure: Hermite Curve

- P1: the start point of the curve
- T1: the tangent (e.g. direction and speed) to how the curve leaves the start point
- P2: the endpoint of the curve
- T2: the tangent (e.g. direction and speed) to how the curves meets the endpoint

Matrix form of Hermite Curve

Vector S: The interpolation-point and its powers up to 3:

Vector C: The parameters of our hermite curve:

Matrix h: The matrix form of the 4 hermite polynomials:

$$S = \begin{bmatrix} s^3 \\ s^2 \\ s^1 \\ 1 \end{bmatrix} \qquad C = \begin{bmatrix} P1 \\ P2 \\ T1 \\ T2 \end{bmatrix} \qquad h = \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

To calculate a point on the curve you build the Vector S, multiply it with the matrix h and then multiply with C.

$$P = S \times h \times C .$$

□□□

MODEL QUESTIONS SETS FOR PRACTICE

MODEL SET 1

TRIBHUVAN UNIVERSITY

Institution of Science and Technology

Course Title: Computer Architecture
Course No: CSC 209
Nature of the Course: Theory + Lab
Semester: III

Full Marks: 60
Pass Marks: 24
Time: 3 hrs

Section A

Attempt any two questions. (2 ×10 =20)

1. Develop the Bresenham's line drawing to draw lines of any scope. Compare this with the DDA Algorithm.
2. Given a window bordered by (1, 2) at the lower left and (16, 12) at the upper right, give the screen coordinates of a triangle with vertices (3, 2), (10, 7.5) and (5, 5) when mapped into a viewport with corners (100, 100) and (400, 200). Provide accurate illustrations of the window, viewport, and the untransformed and transformed triangles with your answer.
3. Explain the essential difference between the "Scan-Line" hidden surface removal algorithm and the depth buffer technique.

Section B

Short Answer Questions
Attempt any *EIGHT* questions [8 × 5 = 40]

4. The figure ABCD where A=(-2, 0), B=(0, -2), C=(-2, -4) and D=(-4, -2) can be transformed into A'B'C'D' where A'=(1, -1), B'=(3, 3), C'=(6, 3) and D'=(4, -1) by the composition of simple transforms T2*H1*S1*R1*T1. Give the matrix form of these five transformations. Then express the composite transform using only one scale, one rotation and one translation.
5. Explain Area Subdivision algorithm with suitable figure? List the advantages and disadvantages of Area Subdivision algorithm.
6. Discuss in detail about visible surface detection methods.
7. Given a 25cm x 20cm display operating in 1024 x 768 x 16 color mode which is refreshed 30 times per second, and for which 10% of the refresh cycle is spent in retrace, calculate
   (i) The pixel aspect ratio,
   (ii) The size of the frame buffer, and
   (iii) The required data transfer rate in kilobytes per second.
8. Trace the points for drawing a line from (0, 0) to (-6, -6) using simple DDA algorithm.

9. Why in practice does the running time for the z-buffer algorithm remain nearly constant as the number of polygons increases?
10. A problem that often arises in scan-conversion is writing a single pixel twice. For example in outlining polygons, pixels may be written twice if we simply scan convert each edge. Why might we want to avoid this situation?
11. Differentiate between parallel projections from perspective projection.
12. Explain the following term with practical applications.
    (a) 3D Rotation
    (b) 2D Shear

MODEL SET 2

Section A

Attempt any TWO questions. (2 ×10 =20)

1. What is a polygon mesh? Explain the application of polygon mesh with example.
2. Justify that hidden surface removal is required in computer graphics. Explain in detail about depth buffer method.
3. Consider 256 pixels X 512 scan lines image with 24-bit true color. If 20 minutes video is required to capture, calculate the total memory required? What is the color intensity model?

Section B

Short Answer Questions
Attempt any *EIGHT* questions [8 × 5 = 40]

4. What is a computer graphics? Explain in detail about the application of computer graphics.
5. Derive the window to viewport transformation coefficient matrix. Explain the application of this matrix.
6. What is a digital differential analyzer (DDA)? How can you draw the line using this algorithm?
7. How can you represent 3D object? How can you draw the line using this algorithm?
8. How curves be generated? Explain it with any suitable algorithm.
9. Explain in detail about plain equation method. Explain which algorithm is better for hidden surface removal.
10. Consider 256 pixel X 256 scan lines image with 24-bit true color. If 10 minutes video is required to capture, calculate the total memory required? Why intensity assignment is required?
11. Why shedding is required in the computer graphics? Explain in detail about constant intensity shading.
12. What do you mean by homogeneous coordinates? Explain it with equation and practical application.

MODEL QUESTIONS SETS FOR PRACTICE

MODEL SET 1

TRIBHUVAN UNIVERSITY

Institution of Science and Technology

Course Title: Computer Architecture
Course No: CSC 209
Nature of the Course: Theory + Lab
Semester: III

Full Marks: 60
Pass Marks: 24
Time: 3 hrs

Section A

Attempt any two questions. (2 ×10 =20)

1. Develop the Bresenham's line drawing to draw lines of any scope. Compare this with the DDA Algorithm.
2. Given a window bordered by (1, 2) at the lower left and (16, 12) at the upper right, give the screen coordinates of a triangle with vertices (3, 2), (10, 7.5) and (5, 5) when mapped into a viewport with corners (100, 100) and (400, 200). Provide accurate illustrations of the window, viewport, and the untransformed and transformed triangles with your answer.
3. Explain the essential difference between the "Scan-Line" hidden surface removal algorithm and the depth buffer technique.

Section B

Short Answer Questions

Attempt any *EIGHT* questions [8 × 5 = 40]

4. The figure ABCD where A=(-2, 0), B=(0, -2), C=(-2, -4) and D=(-4, -2) can be transformed into A'B'C'D' where A'=(1, -1), B'=(3, 3), C'=(6, 3) and D'=(4, -1) by the composition of simple transforms T2*H1*S1*R1*T1. Give the matrix form of these five transformations. Then express the composite transform using only one scale, one rotation and one translation.
5. Explain Area Subdivision algorithm with suitable figure? List the advantages and disadvantages of Area Subdivision algorithm.
6. Discuss in detail about visible surface detection methods.
7. Given a 25cm x 20cm display operating in 1024 x 768 x 16 color mode which is refreshed 30 times per second, and for which 10% of the refresh cycle is spent in retrace, calculate
   (i) The pixel aspect ratio,
   (ii) The size of the frame buffer, and
   (iii) The required data transfer rate in kilobytes per second.
8. Trace the points for drawing a line from (0, 0) to (-6, -6) using simple DDA algorithm.

9. Why in practice does the running time for the z-buffer algorithm remain nearly constant as the number of polygons increases?
10. A problem that often arises in scan-conversion is writing a single pixel twice. For example in outlining polygons, pixels may be written twice if we simply scan convert each edge. Why might we want to avoid this situation?
11. Differentiate between parallel projections from perspective projection.
12. Explain the following term with practical applications.
    (a) 3D Rotation
    (b) 2D Shear

MODEL SET 2

Section A

Attempt any TWO questions. (2 ×10 =20)

1. What is a polygon mesh? Explain the application of polygon mesh with example.
2. Justify that hidden surface removal is required in computer graphics. Explain in detail about depth buffer method.
3. Consider 256 pixels X 512 scan lines image with 24-bit true color. If 20 minutes video is required to capture, calculate the total memory required? What is the color intensity model?

Section B

Short Answer Questions

Attempt any *EIGHT* questions [8 × 5 = 40]

4. What is a computer graphics? Explain in detail about the application of computer graphics.
5. Derive the window to viewport transformation coefficient matrix. Explain the application of this matrix.
6. What is a digital differential analyzer (DDA)? How can you draw the line using this algorithm?
7. How can you represent 3D object? How can you draw the line using this algorithm?
8. How curves be generated? Explain it with any suitable algorithm.
9. Explain in detail about plain equation method. Explain which algorithm is better for hidden surface removal.
10. Consider 256 pixel X 256 scan lines image with 24-bit true color. If 10 minutes video is required to capture, calculate the total memory required? Why intensity assignment is required?
11. Why shedding is required in the computer graphics? Explain in detail about constant intensity shading.
12. What do you mean by homogeneous coordinates? Explain it with equation and practical application.

MODEL SET 3

Section A

Attempt any TWO questions. (2 ×10 =20)

1. What are the object space and image space method of hidden surface removal? Describe the back face detection method of hidden surface removal.
2. Perform the scaling transformation to the triangle with vertices A(6, 9), B(10, 5), C(4, 3) with scaling factors Sx = 3 and Sy = 2.
3. Explain about parametric cubic curve. Describe the properties of Bezier Curve.

Section B

Short Answer Questions

Attempt any *EIGHT* questions [8 × 5 = 40]

4. Explain the visual effect that occurs when during animation of a Gouraud shading polyhedron, the center of a highlight moves from one vertex to another along an edge.
5. Digitize the endpoint (10, 18), (15, 8) using Bresenham's algorithm.
6. Digitize an ellipse with center (20, 20) and x-radius=8 and y-radius=6.
7. Construct the polygon table for an object with six vertexes, eight edge and three surface.
8. Consider a raster scan system having 12 inch by 10 inch screen with resolution of 100 pixels per inch in each direction. If the display controller of this system refreshes the screen at the rate of 50 frames per second, how many pixels could be accessed per second and what is the access time per pixel of the system?
9. Explain the virtual reality and its applications in the computer graphics.
10. Discuss the problems of interpolated model. Find the CMY coordinate of a color at (0.2, 1, 0.5) in the RGB space
11. A system with 24 bits per pixel and resolution of 1024 by 1024, Calculate the size of frame buffer (in Megabytes).
12. Use Cohen Sutherland algorithm to clip the line for the following dimensions. Line endpoints (15, 45) and (25,15) Window: Lower left: (10, 20) and Upper right: (30, 40)

MODEL SET 4

Section A

Attempt any TWO questions. (2 ×10 =20)

1. Transform a square with corners A(2, 2), B(4, 2), C(2, 4), D(4, 4) into a rectangle whose breadth is half of the breadth of square and length is double of it.
2. If a monitor screen has 525 scan lines and an aspect ratio of 3:4 and if each pixel contains 8 bits worth of intensity information, how many bits per second are required to show 30 frames each second?
3. Differentiate between flood fill and boundary fill algorithms.

Section B

Short Answer Questions

Attempt any *EIGHT* questions [8 × 5 = 40]

4. What do you mean by refresh rate of a display device? Write short note about shadow mask method.
5. Consider a RGB raster system is to be designed using 8 inch by 10 inch screen with a resolution of 100 pixels per inch in each direction. If we want to store 8 bits per pixel in the frame buffer. How much storage (in bytes) do we need for the frame buffer?
6. Digitize the line with endpoints (1, 6), (6, 10) using DDA line drawing algorithm.
7. What do you mean by symmetry in octant of a circle? Digitize a circle $(x-2)^2 + (y-3)^2 = 25$
8. Rotate the triangle ABC by 45 degree in clock wise direction and scale it by (2, 3) about origin. The coordinates of ABC are (7, 15), (5, 8) and (10, 10) respectively.
9. Describe non-rigid body transformations with suitable examples.
10. Outline the Z-buffer method. List the advantages and disadvantages of the Z-buffer algorithm.
11. Differentiate between simple reality and mixed reality. Also write down applications of mixed reality with suitable example.
12. Discuss relative advantages and disadvantages of Gourand Shading and Phong Shading.

MODEL SET 5

Section A

Attempt any TWO questions. (2 ×10 =20)

1. Clip the polygon ABC against the window PQRS. Coordinates of polygon are A(80,80), B(120,60) and C(75,60) and Coordinates of window are P (50,50), Q(100,50), R(100,100) and S(50,100).
2. Compare storage type CRT against refresh type CRT display. List the important properties of phosphor being used in CRTs.
3. What do you mean by solid modeling? What are the differences between Boundary representation and spatial-partitioning representation?

Section B

Short Answer Questions

Attempt any *EIGHT* questions [8 × 5 = 40]

4. Suppose we have a quadrilateral ABCD with vertices A(3,1), B(-7,-4), C(5, - 8) and D(3,0). What is the coordinates of quadrilateral if it is to be doubled in size about origin. Then translate obtained coordinates with (-4, 3).

5. How much time is spent in scanning across each row of pixels during screen refresh on a raster system with a resolution of 1280 by 1024 and a refresh rate of 60 frames per second? Assume horizontal and vertical retrace times are negligible.
6. OpenGL is written in which language? Is it possible to implement (or use) same library in programming languages other than that?
7. Give the single-point perspective projection transformation matrix when projectors are placed on the z-axis.
8. Construct the B-spline curve of order 4 and with 4 polygon vertices A(1, 1), B(2, 3), C(4, 3) and D(6, 5).
9. Find the new co-ordinates of a unit cube 90 degree rotated about an axis defined by its end points A(2, 1, 0) and B(3, 3, 1).
10. How is color depth and resolution of an image related to the video memory requirement?
11. What is the main advantage of using fast Phong shading over shading? Write down advantages and disadvantages of Phong shading.
12. Explain the following any two algorithms for visible surface detection:
    a) Octree method
    b) Ray casting method
    c) Area sub division method
    d) Z-buffer method

MODEL SET 6

Section A

Attempt any TWO questions. **(2 ×10 =20)**

1. A polygon has four vertices located at A(20, 10) B(60, 10) C(60, 30) D(20, 30). Calculate vertices after applying a transformation matrix to double the size of polygon with point A located on the same place.
2. Describe briefly Bresenham's line drawing algorithm. Why we prefer incremental algorithm over DDA?
3. What is projection? List out the types of projection. Differentiate parallel and perspective projections.

Section B

Short Answer Questions

Attempt any *EIGHT* questions **[8 × 5 = 40]**

4. Rotate the triangle having co-ordinates (1, 2), (2, 3) (4, 5) by 60° about origin. Then translate with T(9, 0), and then scale about origin with scale factor S(2, 4).
5. Generate a homogeneous matrix representation for oblique projection of co-ordinate position (x, y, z) to position (x1, y1) on the view plane.
6. Construct the Bezier curve of order 3 and 4 polygon vertices A(1, 1) B(2, 3) C(4, 3) and D(6, 4).
7. Describe key components of virtual reality with suitable examples.

8. Consider three different raster systems with resolutions of 640 x 480, 1280 x 1024, and 2560 x 2048.
    a) What size is frame buffer (in bytes) for each of these systems to store 12 bits per pixel?
    b) How much storage (in bytes) is required for each system if 24 bits per pixel are to be stored?
9. How do we clear a window in OpenGL? Also write a code snippet for the same.
10. Describe RGB color model and HSV color mode.
11. What is Virtual Reality and how does it affects human life?
12. Derive the window to viewport transformation coefficient matrix. Write the applications of window to view port transformation.

MODEL SET 7

Section A

Attempt any TWO questions. **(2 ×10 =20)**

1. Clip a quadrilateral ABCD with coordinates (10, 18) (22, 18) (34, 27) and (10, 37) against a window QRST with coordinate (5, 15) (30, 15) (30, 25) and (5, 25) using Cohen Sutherland algorithm.
2. Explain the concept of Rendering. Explain Gourand shading with their advantages and disadvantages.
3. Given a 25cm x 20cm display operating in 1024 x 768 x 16 color mode which is refreshed 30 times per second, and for which 10% of the refresh cycle is spent in retrace, calculate
    a) the pixel aspect ratio,
    b) the size of the frame buffer, and
    c) The required data transfer rate in kilobytes per second.

Section B

Short Answer Questions

Attempt any *EIGHT* questions **[8 × 5 = 40]**

4. Rotate a triangle [(4, 6), (2, 2), (6, 2)] about the vertex (4, 6) by 180° CCW and find the new vertices.
5. Find the points on the Bezier curve which has starting and ending points $P_0(2, 3)$ and $P_3(4, -3)$ and is controlled by $P_1(5, 6)$ and $P_2(7, 1)$ for $u = 0.9$.
6. A tetrahedron of size 10 units is placed on the x-plane with one edge along the x-axis and the origin at the center of the object. Assuming the COP at (5, 0, 0), draw the projected image using perspective projection transformation.

7. What do you mean by transparency? Explain its properties.
8. Why use SDL and OpenGL instead of just OpenGL?
9. What is resolution? What is the impact of resolution in display devices?
10. Compare and contrast the operating characteristics of raster refresh systems, plasma panels and LCDs.
11. Digitize the line with endpoints (1, 6), (6, 10) using DDA algorithm.
12. Explain basic rendering and illumination models. Also describe their advantages and disadvantages.

MODEL SET 8

Section A

Attempt any TWO questions. (2 ×10 =20)

1. Given a clipping window P(0, 0), Q(30, 0), R(30, 20) and S(0,20) use the Cohen Sutherland algorithm to determine the visible portion of line joint A(10, 30) and B (40, 0).
2. Find the points on a circle one of its octants with the circle centered at (5, 5) and has a radius of 8 units.
3. Why we require hidden surface removal algorithm? Write and explain the scan line method for hidden surface removal. How the amount of computation can be reduced in the method?

Section B

Short Answer Questions [8 × 5 = 40]

Attempt any *EIGHT* questions

4. Rotate the ΔABC by 45° clock wise about origin and scale it by (2, 3) about origin, where A(7,15), B(5,8), and C (10,10).
5. Construct the Bezier curve of order 3 and with 4 polygon vertices A(1, 1), B(2, 3), C(4, 3) and D(6, 3).
6. Assume that one allows 256 depth value levels to be used. Approximately how much memory would a 512 x 512 pixel display require to stove the Z-buffer?
7. With 3 bits per pixel, we can accommodate 8 gray levels. If we use 8 bits per pixel then what is the value of gray levels? Describe RGB color model.
8. Explain Binary Space Partitioning Trees (BSP) with suitable example.
9. What is the difference between a local illumination model and a global illumination model?
10. OpenGL programming vs. Blender Software, which is better for custom video creation?
11. What is Mixed Reality? What is the most important technical challenge in Virtual Reality?
12. Define polygon. What are the different types of polygons? Explain with examples.

❑❑❑