MODEL - QUESTION PAPER.

SIXTH SEMESTER B.E DEGREE EXAMINATIONS

COMPUTER GRAPHICS and VISULIZATION

(18CS62)

Time: 3hrs MaxMarks: 100

Note: Answer any FIVE full questions, choosing ONE full question from Each Module.

Module 1.

1 a) Describe various applications of Computer Graphics with appropriate Examples. 08 Marks

b) Explain DDA line drawing algorithm with procedure. 06 Marks

c) With a neat diagram, explain the architecture of raster display system with integrated display processor. 06M

OR.

2 a) Explain the basic operation of CRT, with its primary components with neat diagram. 08 Marks

b) Digitize the line by using Bresenham's line drawing algorithm with end-points (20, 10) and (30, 18) having slope 0.8 06 Marks

c) Explain with diagram the different cartesian reference frames are used in the process of constructing and displaying a scene. 06 Marks.

Module - 2

3a) Explain with example any two algorithms used to identify the interior area of a polygon. 06 Marks

b) Explain translation, Rotation, and scaling in 2D Homogeneous co-ordinate System with matrix representations. 09 Marks

c) Obtain a matrix representation for rotation of a object about a specified pivot-point in 2-Dimension 05 Marks.

OR

4 a) Explain the scanline polygon filling algorithm and also Explain the use of Sorted edge table & active edge list. 10 Marks.

b) What are the entities required to perform a rotation? Show that two successive rotations are additive 06 Marks.

c) Explain OpenGL Functions for i) Translation ii) Rotation with the parameters. 04 Marks.

Module - 3.

5a) Define Clipping, Briefly explain Cohen Sutherland line clipping without code, Discuss four cases 10 Marks.

b) Explain Basic Illumination models 06 Marks.

c) Explain RGB and CMY Color Models. 04 Marks.

OR

a) Explain Sutherland - Hodgman polygon clipping algorithm. with neat Sketches. 08 Marks

b) Describe phong lighting model. 06 Marks

c) Explain the different types of light sources Supported by openGL 06 Marks

Module-4

7 a) Explain the perspective projections with reference point and vanishing point with neat diagrams 10 Marks

b) Explain with example the depth buffer algorithm used for visible surface detection. And also list the advantages and disadvantages of depth buffer algorithm. 10 Marks

OR.

8 a) Bring out the differences between perspective and parallel projections. 05 Marks

b) Explain orthogonal projections in detail. 10 Marks

c) Explain back-face detection method with Example. 05 Marks

Module - 5

9 a) Explain the logical classifications of input devices with Example. 06 Marks.

b) Discuss request mode, sample mode & event mode with figures. 06 Marks

c) What is Display list? write a openGL code segment that generate a blue colored square using Display list. 08 Marks.

OR.

10a) Explain Bezier Curves with equations. and demonstrate the appearance of Bezier Curves for various selection of control points. 08 Marks.

b) What is Double Buffering? How it is implemented in openGL 08 Marks

c) List the various features that a good interactive program should include. 04 Marks.

* * *

Module 1

1a) Applications of Computer Graphics.

1) Graphs & Charts.

* Early application of Computer Graphics is the displa of Simple data graphs usually printed on a charac printer. Data plotting is still one of most Common Graphics application.
* Graphs and charts are commonly used to summari functional, statistical, mathematical, engineering & economic data for research reports, managerial Summaries and other types of publications.
* Typical Ex: of data plots are line graphs, bar charts Pie charts, surface graphs, contour plots & other displays showing relationships between multiple parameters in 2D, 3D ~~of~~ or higher-dimensional spaces.

2) Computer Aided Design:

* A major use of Computer Graphics is in design-processes particularly for engineering and architectural systems.
* CAD, Computer Aided Design, CADD methods are now routinely used in the automobiles, aircraft Spacecraft, computers, home appliances.
* Circuits and networks for Communications, water supply or other utilities are constructed with repeated placement of a few geographical shapes.
* Animations are often used in CAD applications, Realtime, computer animations using wire-frame shapes are useful for quickly testing the performance of a Vehicle or system.

3) Virtual Reality environments:

* Animations in VR environments are often used to train heavy-equipment operators or to analyse effectiveness of various cabin configurations & control placements.

* with Virtual - Reality Systems, designers and others can move about and interact with objects in various ways. Architectural Designs can be examined by taking simulated walk thro' rooms or around the outsides of buildings to better appreciate the overall effect of a particular design.

4) Data Visualization:

* producing graphical representations for scientific, engineering and medical data sets and processes is another fairly new application of Computer Graphics which is generally refered to as scientific visualization.
* Business visualization is used in connection with data sets related to commerce, industry & other non-scientific areas.
* There are many different kinds of datasets and effective visualization schemes depend on the characteristics of the data. A collection of data can contain scalar values, vectors or higher-order tensors.

5) Education and Training:

* computer generated models of physical, financial, polytical, social, economic & other systems are often used as educational aids.
* Models of physical processes physiological functions, equipment, such as color coded diagram, can help trainees to understand the operation of system
* For some training applications, special hardware systems are designed. Examples are simulators for practice sessions, aircraft pilots, air-traffic control personnel.
* Some simulators have no vedio screens, For Ex flight simulator with only a control panel for instrument flying.

6) Computer Art

* The picture is usually painted electronically on a graphics tablet using a Stylus, which can simulate different brush strokes, brush widths, colors, etc
* Fine artists use a variety of other computer technologies to produce images, to create pictures Ex: 3D modeling packages, texture mapping, drawing programs and CAD s/w etc.
* Commercial art also uses the painting technique for generating logos and other designs, page layouts combining text & graphics.

7) Entertainment

* Television production, motion pictures, and music vedios routinely a computer Graphics methods.
* Graphics images are combined a live actors & scenes films are generated & animation techniques.
* Cartoon characters, animation techniques to combine computer generated figures of people, animals etc.

8) Image processing:

* Image processing methods are used to improve picture quality, analyse images, or recognize visual patterns for robotics applications.
* Image processing methods are often used in Computer Graphics, for Ex: Medical applications for enhancements in tomography, and in simulations and surgical operations.
* It is also used in computed X-ray Tomography (CT), Position emission Tomography (PET) etc.

9) Graphical User Interfaces:

* Applications software to provide GUI, is a window manager that allows a user to display multiple, rectangular screen areas called "Display windows"

1b) * Digital Differential Analyzer (DDA) is a scan-conversion line algorithm based on calculating either δy or δx.

* A line is sampled at unit intervals in one co-ordinate and the corresponding integer values nearest the line path are determined for the other co-ordinate.

* DDA Algorithm has three cases. from the eqn

$$Y = mx + c.$$

where $m = (y_{k+1} - y_k) / (x_{k+1}) - x_k$.

case 1: if $m < 1$, x increment in unit intervals (+ve slope)

$$x_{k+1} = x_k + 1 \quad (\delta x = 1)$$

compute successive y values

$$y_{k+1} = y_k + m. \quad \text{'k' - from 'o' to final endpt}$$

case 2: if $m > 1$ (+ve slope > 1), reverse the roles of x & Y.

we sample at unit 'y' intervals ($\delta Y = 1$)

& calculate consecutive 'x' values,

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

(rounded to nearest pixel position along with current 'y' scanline)

case 3: if $m = 1$,

$$x_{k+1} = x_k + 1, \quad Y_{k+1} = Y_k + 1$$

The above cases based on the assumption that lines are to be processed from Left to Right.

If this processing is reversed, so that the starting endpt is at the right, then we have

$$\delta x = -1, \quad Y_{k+1} = y_k - m.$$

or $\delta Y = -1, \quad x_{k+1} = x_k - \frac{1}{m}.$

/* procedure

```
void line_DDA ( int x0, int y0,  int xend , int yend)
{
    int dx= xend-x0;
    int dy = yend - y0;
    int steps, k;
    float  xincre , yincre,  x= x0, y=y0;
    if (fabs (dx) > fabs(dy))
        steps= fabs (dx);
    else  steps = fabs (dy);
    x incre  = float (dx) / float (steps);
    yincre = float (dy) / float (steps);
    setpixel ( round (x), round (y));
    for (k=0 ; k< steps ; k++)
     {   x += x incre;
         y + = yincre;
        setpixel (round (x), round (y));
     }
}.
```

1 c) Raster - scan Display processor

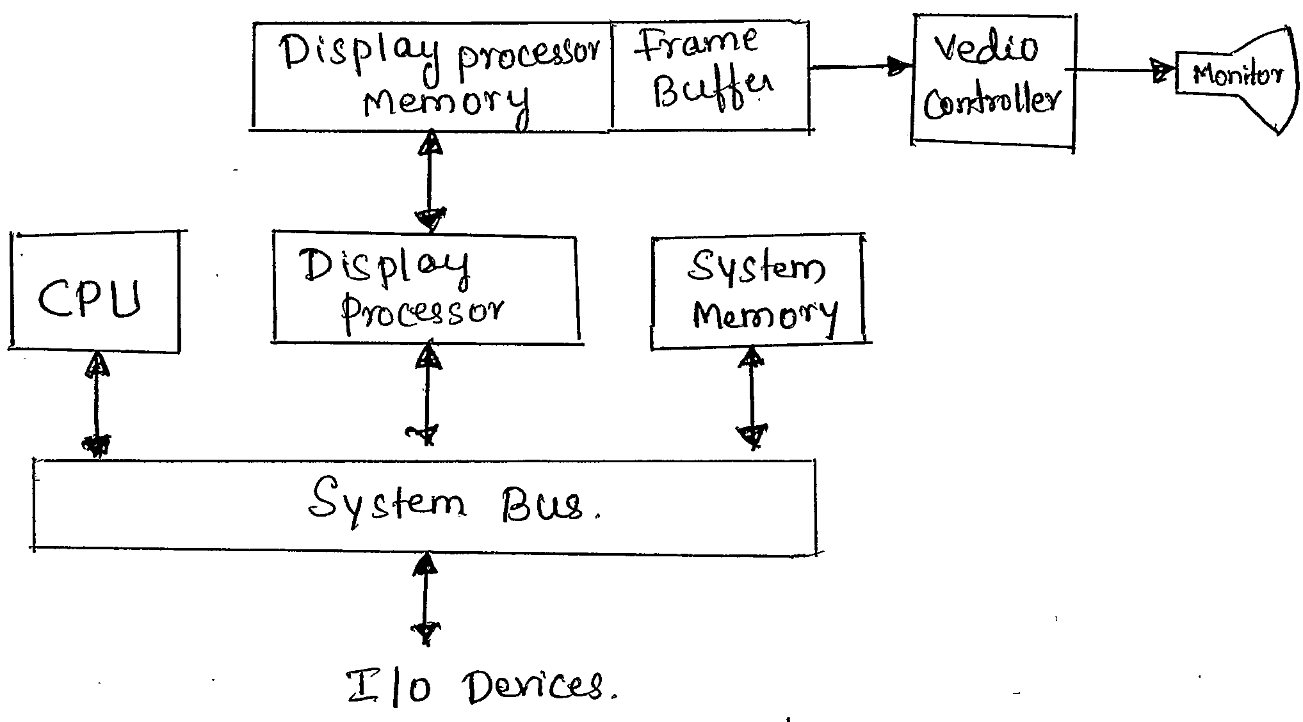


Fig. Raster System with Display processor.

* Figure shows one way to organise the components of a raster system that contains a seperate display processor, sometimes refered to as a "graphics controller" or "display co-processor"
* central processing unit (CPU), a special purpose processor called vedio controller is used to control the operation of the display device.
* A fixed area of the system memory is reserved for frame buffer, & the vedio controller is given direct access to the frame buffer memory.
* Frame buffer locations, and the corresponding screen positions are referenced in the cartesian co-ord.
* The purpose of the display processor is to free the CPU from the graphics chores.
* In addition to the system memory, a seperate display processor memory area can be provided.

2a) Basic design of magnetic deflection CRT.

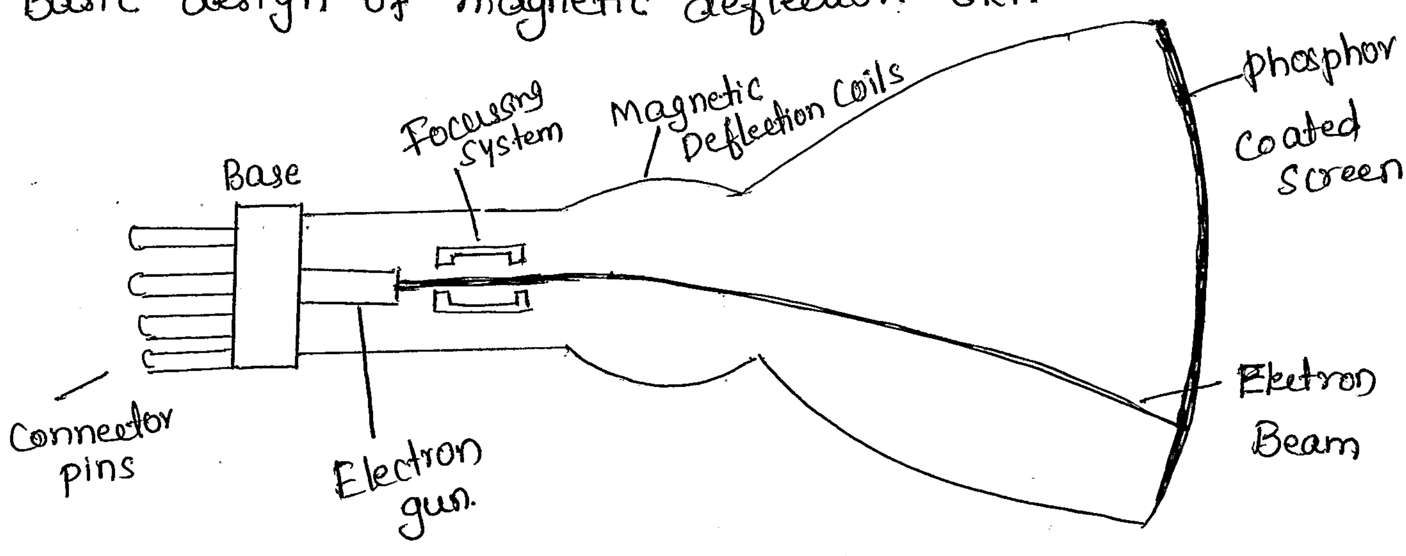


Fig. Basic design of magnetic-deflection CRT.

* Fig shows the basic design and primary components of CRT.
* The primary components of electron gun in a CRT are heated filament.

* A beam of electrons, emitted by electron gun, passes thro' focussing & deflection systems, that direct the beam toward specified positions on the phosphor coated screen.
* The phospor then emits a small-spot of light at each position contacted by the electron beam and the light emitted by the phosphor fades very rapidly.
* Heat is supplied to the cathode by directing a current through a coil of wire, called "filament" inside the cylendrical cathode structure.
* One way to maintain the screen picture is to store the picture information as a charge distribution within the CRT in order to keep the phosphors activated.
* The most common method now employed for maintaining phosphor glow is to redraw the picture repeatedly by quickly directing the electron beam back over the same screen points. This type of display is called refresh CRT.
* The frequency at which a picture is redrawn on the screen is referred to as the refresh rate.

2b) Digitize the line with end points (20,10) and (30,18)

This line has a slope $= 0.8$.

with $\Delta x = 10$, $\Delta y = 8$.

The initial decision parameter has the value.

$$P_0 = 2\Delta y - \Delta x$$
$$= 6.$$

& the increments for calculating successive decision parameters are

$$2\Delta y = 16, \quad 2\Delta y - 2\Delta x = -4.$$

We plot the initial point $(x_0, y_0) = (20, 10)$ & determine successive pixel positions along the line path.

from the decision parameter as:

K	P_k	(x_{k+1}, y_{k+1})
0	6	(21, 11)
1	2	(22, 12)
2	-2	(23, 12)
3	14	(24, 13)
4	10	(25, 14)
5	6	(26, 15)
6	2	(27, 16)
7	-2	(28, 16)
8	14	(29, 17)
9	10	(30, 18)

Pixel positions along the line path between end-pts (20,10) and (30,18), plotted with Bresenham's line algorithm.

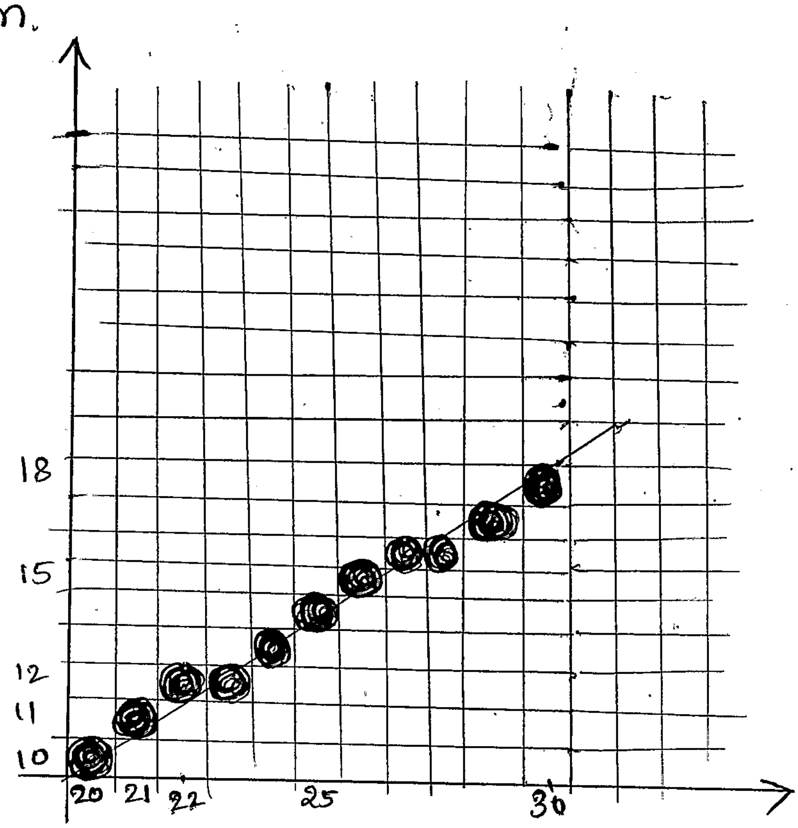


Fig. pixel positions along the line path between end-points (20,10) and (30,18).

2C) Cartesian reference Frame:

* Frame buffer locations and the corresponding screen positions, are referenced in cartesian co-ordinates.
* In an application (user) program, we use the commands within the graphics software package to set co-ordinate positions for displayed objects relative to the origin of the cartesian reference frame.
* The co-ordinate origin is referenced at the lower left corner of a screen display area by the software commands, although we can typically set the origin at any convenient location for a particular application.

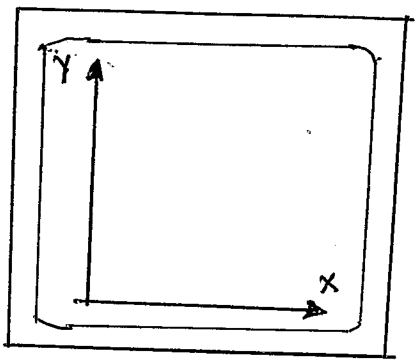


Fig. A cartesian reference frame with origin at the lower-left corner of a vedio monitor.

Figure shows a two-Dimensional cartesian reference frame with the origin at the lower-left screen corner.

* The screen surface is then represented as the first quadrant of a two-dimensional system with positive x and y-values increasing from left to right and bottom of the screen to the top resply

* pixel positions are then assigned integer x-values that range from 0 to **Xmax** across the screen, left to right, and integer y values that vary from 0 to **ymax** bottom to top.

* Some software systems, reference the pixel position from the top-left corner of the screen.

Module - 2

3a) Two commonly used algorithms for identifying interior areas of a plane are

1) odd-even rule.
2) Non-zero winding number.

i) odd-even rule / Inside-outside test.

* Also called the odd-parity rule or the even-odd rule
* By first drawing a line from any position p to a distant point outside the co-ordinate extents of the closed polyline.
* Then we count the number of line-segment crossings along this line.
* If the number of segments crossed by this line is odd, then p is considered to be an interior-point otherwise p is exterior point.

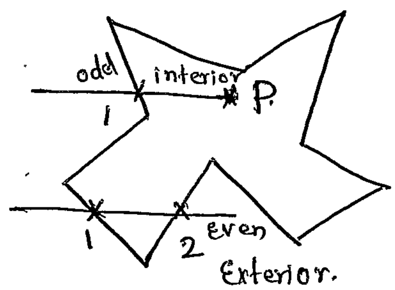


Fig shows the interior and exterior regions obtained using odd-even rule.

Fig. odd-even rule.

ii) Non-zero winding-Number rule:

* This counts the number of times that the boundary of an object "winds" around a particular point in the counter-clockwise direction termed as "winding number."

* Initialize the winding number to '0' and again imagining a line drawn from any position p to a distant point beyond the co-ordinate extents of the object

* The line we choose must not pass thro' any end point co-ordinates.

* As we move along the line from position 'p' to the distant point, we count the number of object line segments that cross the reference line in each direction.
* we add 1 to the winding number every time we intersect a segment that crosses the line in the direction from right to left and we subtrac 1 every time we intersect segment that crosses from left to right.
* The final value of the winding number, after all boundary crossings have been counted,
If the winding number is nonzero, P is considered to be an interior point otherwise 'p' is taken to be an exterior point.

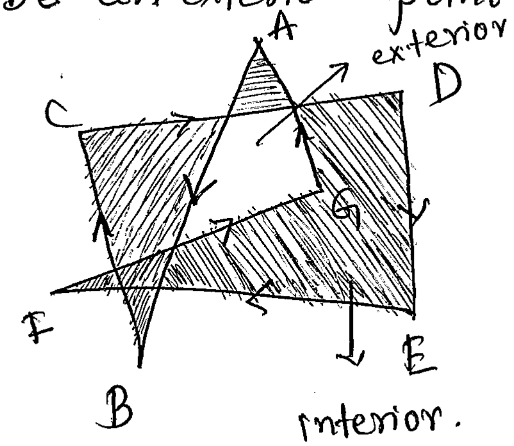


Fig. Shows the path in the direction for counting the winding number.

3b) i) Two - dimensional - Translation Matrix

* Using homogeneous Co-ordinate approach, we can represent the equations for 2D - Translation of a co-ordinate position using the following matrix multiplication

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & tx \\ 0 & 1 & ty \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

This translation operation can be written, as

$$P' = T(tx, ty) \cdot P$$

ii) Two-Dimensional Rotation Matrix:

2D - rotation matrix about the co-ordinate origin can be expressed in the matrix form

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

or

$$P' = R(\theta) \cdot P.$$

where $R(\theta)$ is rotation transformation operator is 3x3 matrix, with rotation angle θ

iii) Two-Dimensional Scaling Matrix:

Scaling Transformation relative to the co-ordinate origin can now be expressed as, matrix multiplication

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

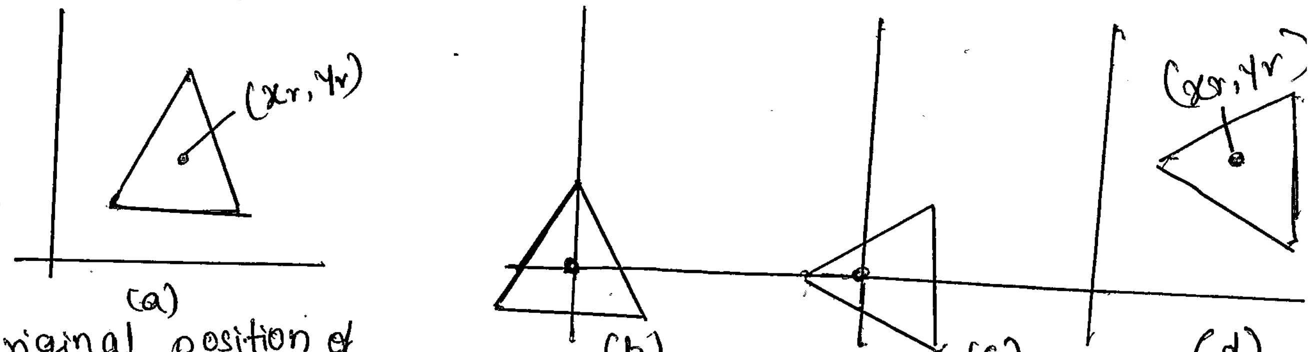
$$P' = S(s_x, s_y) \cdot P.$$

Scaling parameter $S(s_x, s_y)$ is the 3x3 matrix, with scaling parameters s_x, & s_y.

3c) Two-Dimensional Pivot Point Rotation:

* when a graphics package provides only a rotate function w.r.t to the co-ordinate origin.
* we can Generate a 2-D rotation about any other pivot point (x_r, y_r) by performing the following sequence of translate - rotate - translate operations.
  1. Translate the Object so that pivot-point position is moved to the co-ordinate origin.
  2. Rotate the object about the co-ordinate origin
  3. Translate the object so that the pivot point is returned to its original Position.

This transformation sequence is illustrated



(a) original position of object and pivot point

(b) Translation of object so that pivot pt (xr, yr) is at origin.

(c) Rotate about origin

(d) Translatio of obj. pivot pt(xr is returned position (xr

The composite transformation matrix for this sequence is obtained with the concatenation

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

$$= \begin{bmatrix} \cos\theta & -\sin\theta & x_r(1-\cos\theta) + y_r\sin\theta \\ \sin\theta & \cos\theta & y_r(1-\cos\theta) - x_r\sin\theta \\ 0 & 0 & 1 \end{bmatrix}$$

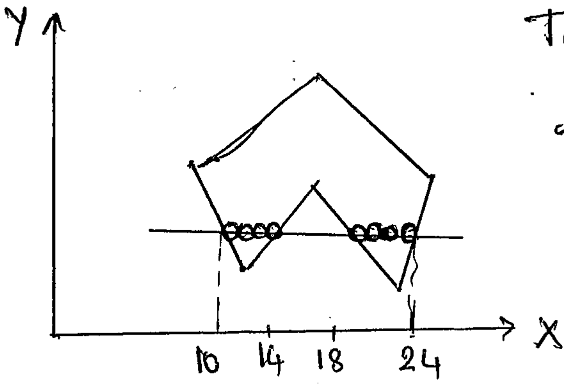
which can be expressed in the form,

$$T(x_r, y_r) \cdot R(\theta) \cdot T(-x_r, -y_r) = R(x_r, y_r, \theta)$$

where $T(-x_r, -y_r) = T^{-1}(x_r, y_r)$.

4a) General - Scan line - polygon - Filling Algorithm :

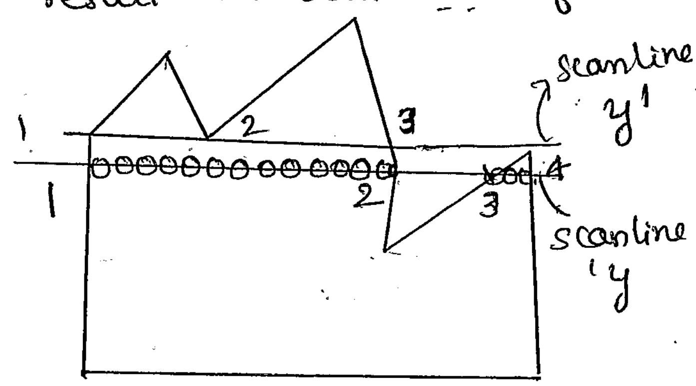
- A scanline fill of a region is performed by first determining the intersection position of the boundaries of the fill region with screen scan lines.
- Then fill the colors, to each section of a scan line that lies within the interior of the fill region.
- The simplest area to fill is a polygon because each scan-line intersection point with a polygon boundary is obtained by solving a pair of simultaneous linear equations, where the equation for the scanline is simply $y =$ constant.



This Figure illustrates the basic scan line procedure for solid-color fill of a polygon.

For Each scan line that crosses the polygon, the edge intersections are sorted from left to right. and then the pixel-positions between, and including, each intersection pair are set to the specified fill color. the fill color is applied to the five pixels from $x=10$ to 14 and seven pixels from $x=18$ to $x=24$.

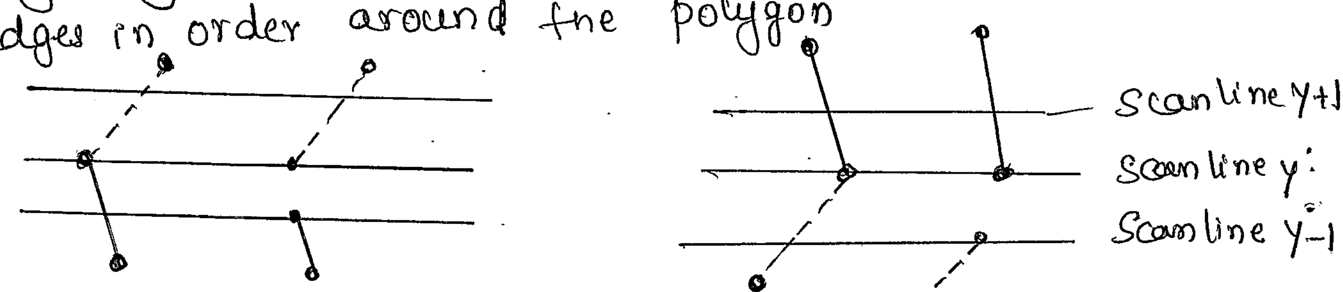
* whenever a scan line passes through a vertex, it intersects two polygon edges at that point. In some cases it result in odd no of boundary intersections for a scanline



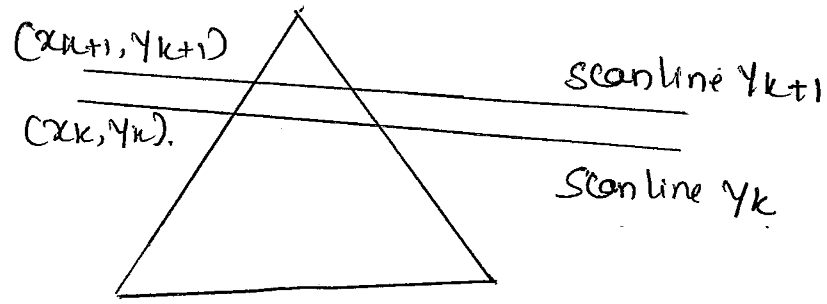
* scan line 'y'' intersects even no of edges. and the two pairs of intersection points along this scanline correctly identify the interior pixel spans
* scan line 'y' intersects 5 polygon edges.

* For scan line 'y', the two edges sharing an intersection vertex are on opposite sides of the scanline. But for the Scanline y', the two intersecting edges are both above the scanline.

- we can process non horrizontal edges around the polygon boundary in the order Clockwise / anticlockwise
- Adjusting endpoint 'y' values for a polygon, as we process edges in order around the polygon



(a) (b)



→ The slope of this edge can be expressed in terms of scan line intersection co-ord:

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

→ Because the change in 'y' co-ordinates between the two scan-lines is simply

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

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

→ The x-intersection value x_{k+1}, on the upper scan line can be determined from x-intersection value x_k on the preceding scan-line as

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

→ Each successive x intercept can thus be calculated by adding the inverse of the slope and rounding to nearest integer.

→ Along an edge with slope 'm', the intersection 'x_k' value for scanline 'k' above the initial scanline can be calculated as

$$x_k = x_0 + k/m$$

- where 'm' is ratio of two integers.

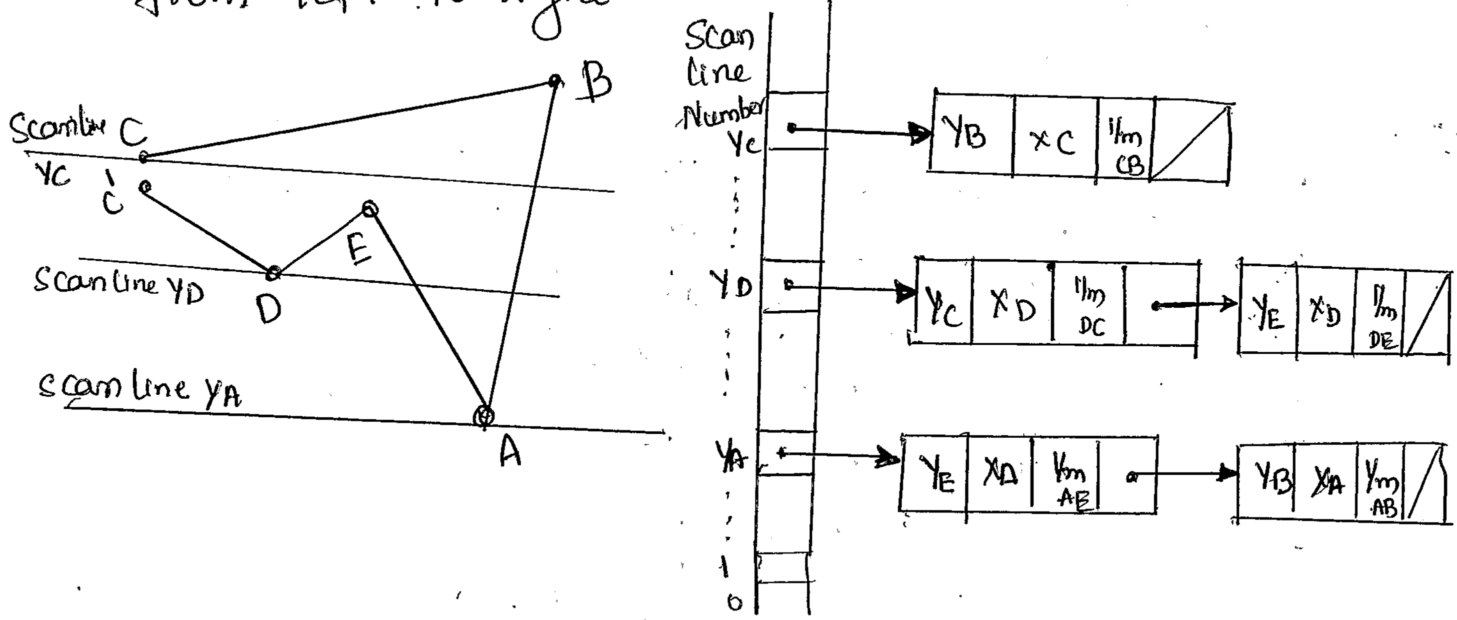
$$m = \frac{\Delta y}{\Delta x}.$$

where Δx and Δy are the differences between the edge end-point x and y-co-ordinate values.

Thus, incremental calculations of 'x' intercepts along an edge for successive scan-lines can be expressed as

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

- To perform a polygon fill efficiently, we can first store the polygon boundary in a sorted edge table that contains all the information necessary to process the scan-lines efficiently.
- Proceeding around the edges in either a clockwise or counter clockwise order, we can use a bucket sort to store the edges, sorted on the smallest 'y' value of each edge, in the correct scan-line positions.
- Only non horizontal edges are entered into the sorted edge table.
- Each entry in the table for a particular scan line contains the maximum 'y' value for that edge, the x-intercept value (at the lower vertex) for the edge, and the inverse slope of the edge. For each scan-line, the edges are in sorted order from left to right.



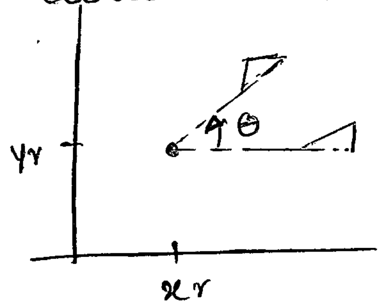
- We process the scanlines from the bottom of the polygon to its top, producing an active edge list (AEL) for each scanline crossing the polygon boundaries.
- The active edgelist for scanline contains all edges crossed by that scanline, with iterative co-herence calculations used to obtain edge x^ns
- Implementation of edge x^n calculations can be done by sorting Δx & Δy

4b) Rotation transformation of an object by specifying a rotation axis and rotation angle.

* All points of the object are transformed to new positions by rotating the points through the specified angle about the rotation axis.
* 2D Rotation of an object is obtained by repositioning the object along a circular path in the xy plane.

→ Rotating object about a rotation axis i.e perpendicular to the xy plane. (parallel to z-co-ord axis)

* Parameters for 2D Rotation are
  1) Rotation angle
  2) position (x_r, y_r) called rotation point (or pivot point) about which the object is to be rotated.



Shown in Fig. pivot point is the x^n-point / position of the rotation axis with the xy plane.

• +ve value for angle 'θ' defines counter clockwise rotation about the pivot-point.

• -ve values rotates objects in the clockwise direction

Two successive rotations are applied to a point 'p' to produce the transformation point

$$p' = R(\theta_2) \cdot \{ R(\theta_1) \cdot p \}$$
$$= R\{\theta_2) . R(\theta_1)\} . p$$

By multiplying the two rotation matrices, we can verify that two successive rotations are additive.

$$R(\theta_2) . R(\theta_1) = R(\theta_1 + \theta_2).$$
$$p' = R(\theta_1 + \theta_2) \cdot p.$$

Final rotation matrix co-ord point is additive of two rotations. Hence two successive rotations are additive

4C) Open GL Functions for

1) Translation : glTranslate* (tx, ty, tz).

* 4×4 translation matrix is constructed with the routine glTranslate*.

* Translational parameters tx, ty, and tz can be assigned any real-number values, and the single suffix code to be affixed to this function

* Suffix code is either f (float) or d (double)

* For 2D-applications we set tz=0.0.

Ex: we translate co-ord positions 25 units in x-direction & -10 units in y-directions.

glTranslatef (25.0, -10.0, 0.0).

2) Rotation:

4×4 rotation matrix is generated with,

glRotate* (theta, vx, vy, vz).

* Vector V=(vx, vy, vz) can have any floating-point values for its components.

* This vector defines the orientation for a rotation axis that passes through the co-ord origin.

* If 'V' is not specified as a unit vector, then it is automatically normalized, before the elements of the rotation matrix are computed.

* Suffix code can be either 'f or d' and parameter 'theta' is to be assigned a rotation angle in degrees which converts to radians for the trignometric calculations.

* This fun generates rotation matrix. Ex:

gl Rotatef (90.0, 0.0, 0.0, 1.0)

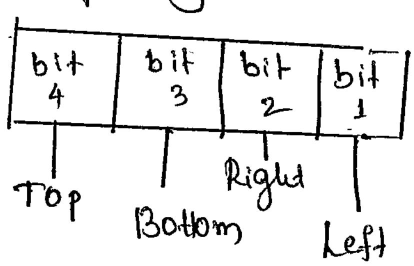
sets the rotation matrix of 90° about the z-axis

Module - 3

5a) Any procedure that eliminates those portions of a picture that are either inside or outside a specified region of space is referred to as a clipping algorithm or simply clipping.

<u>Co-hen Sutherland line Clipping:</u>

* Clipping straight line-segments using a standard rectangular clipping window.
* A line-clipping algo processes each line in a scene through a series of tests and intersections calculations to determine whether the entire line or any part of it is to be saved.
* major goal is to minimize the x^n calculations. & processing time is reduced in Co-hen-Sutherland method by performing more tests before proceeding to the x^n calculations.
* Initially every line end-point in a picture is assigned a four-digit binary value called "region code". & each bit position is used to indicate whether the point is inside or outside one of the clipping window boundaries.



* A possible ordering for the clipping window boundaries corresponding to the bit positions in the Co-hen-Sutherland end-pt region code.
* Thus, for this ordering, the rightmost position (bit 1) references to the left clipping-window boundary, & the left most position (bit 4) references the top window boundary.
* A value of '1' (or true) in any bit position indicates that the end-point is outside that window border.

Illy, a value of '0' (or false) in any - bit position indicates that the end-point is not outside (it is inside or on) the corresponding window edge.

* Region code is refered to as an "outcode" because a value of '1' in any bit position indicates that the spatial point is outside the corresponding clipping boundary.

The nine - binary region codes for identifying the position of a line end-point, relative to the clipping window boundaries.

1001	1000	1010
0001	0000 clipping window	0010
0101	0100	0110

* Bit values in a region code are determined by comparing the co-ord values (x, y) of an end-point to the clipping boundaries.

* Bit 1 is set to 1 if $x < x_{wmin}$, & the other three bits values are determined similarly.

* For a line with end-pt co-ord (x_0, y_0) and (x_{end}, y_{end}) the 'y' co-ord of the intersection point with a vertical clipping border line can be obtained with the calculation

$$y = y_0 + m(x - x_0)$$

where the 'x' value is set to either x_{wmin} or x_{wmax} & slope of line is calculated as

$$m = (y_{end} - y_0) / (x_{end} - x_0)$$

Illy if we are looking for the intersection with a horizontal border, the x - co-ord can be calculated as, $x = x_0 + y - y_0/m$.

* once we have established region codes for all line end-points.
we can quickly determine which lines are completely inside the clip window and which are clearly outside.

* Any lines that are completely contained within the window edges have a region code of 0000 for both end-points, and we save these line segments.

* Any line that has a region code value of 1 in the same bit position for each - end point is completely outside the clipping rectangle and we eliminate that line segment.

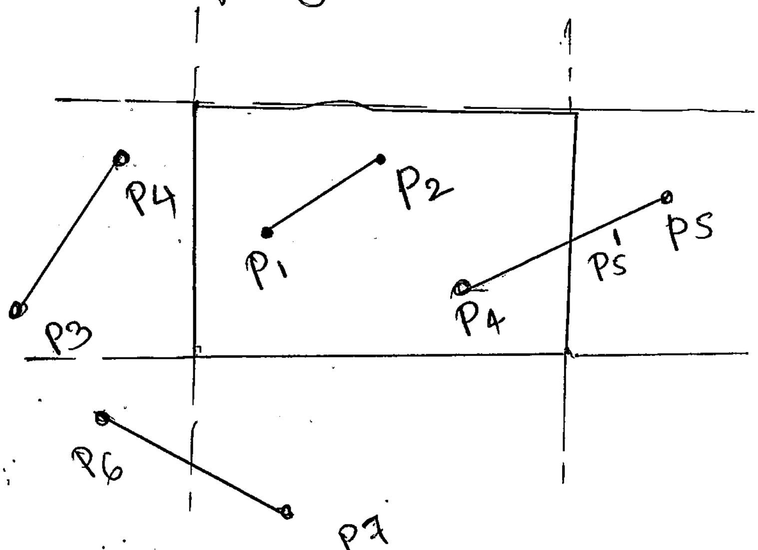
* We can perform the inside - outside tests for line segments ~~us~~ using logical operators.

Case 1: when the **or** operations between two end-points region-codes for a line segment is false (0000), the line is inside the clipping window. Therefore we save the line. $P_1 P_2$ in Diagram.

Case 2: when the **and** operation between the two end-point region codes for a line is true (not 0000), the line is completely outside the clipping window. & we can eliminate it from the scene description. $P_3 P_4$ in Diagram.

Case 3: Lines that cannot be identified as being completely inside or completely outside a clipping window by the region-code tests are next checked for intersection with the window border lines. $P_4 P_5'$

Case 4: Line segments can intersect clipping boundary lines without entering the interior of the window. x^n calculations, with clipping-window edge, a section of the line clipped, remaining part of the line is checked against the other window borders, we continue to eliminating the sections until either the line is totally clipped or the remaining part of the line is inside the clipping window. P6 P7 line in the Diagram



5b) Basic Illumination models:

* Light-emitting objects in a basic illumination model are generally limited to point sources. Many graphics packages provide additional functions for dealing with directional lighting (spotlights) & extended light sources.

1) Ambient light:

* This produces a uniform ambient lighting that is the same for all objects, and it approximates the global diffuse reflections from the various illuminated surfaces.
* Reflections produced by ambient light illumination are simply a form of diffuse reflection and they are independent of the viewing direction & the spatial orientation of a surface.
* However the amount of incident ambient light i.e reflected depends on surface optical properties, which determine how much of the incident energy is reflected and much is absorbed.

ii) Diffuse Reflection:

→ The incident light on the surface is scattered with equal intensity in all directions, independent of the viewing position.

→ Such surfaces are called ideal diffuse reflectors or Lambertian reflectors, because the reflected radiant light energy from many point on the surface is calculated with Lambert's cosine law.

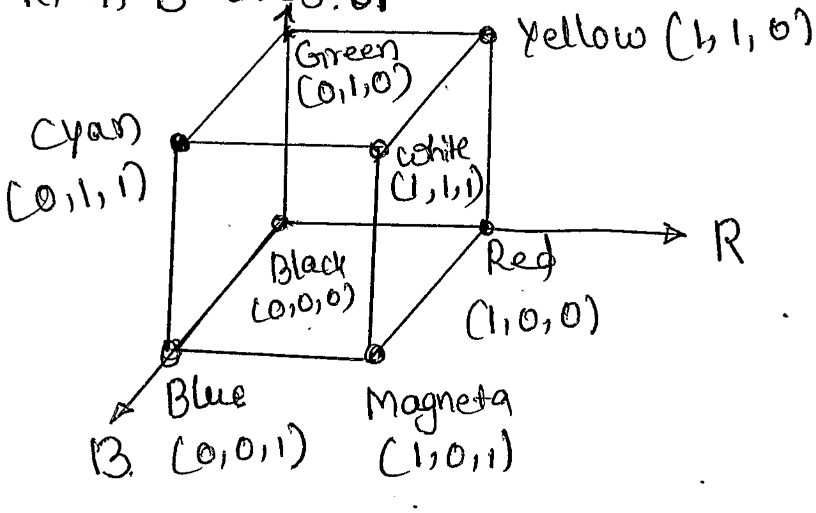
iii) Specular Reflection:

→ The bright spot or specular reflection, that we can see on a shiny surface is the result of total or near total, reflection of the incident light in a concentrated region around the specular-reflection angle.

5c) RGB Color Model:

→ The three primaries color red, green and blue is refered to as RGB model.

→ we can represent this model using unit cube defined on R, G, B axes.

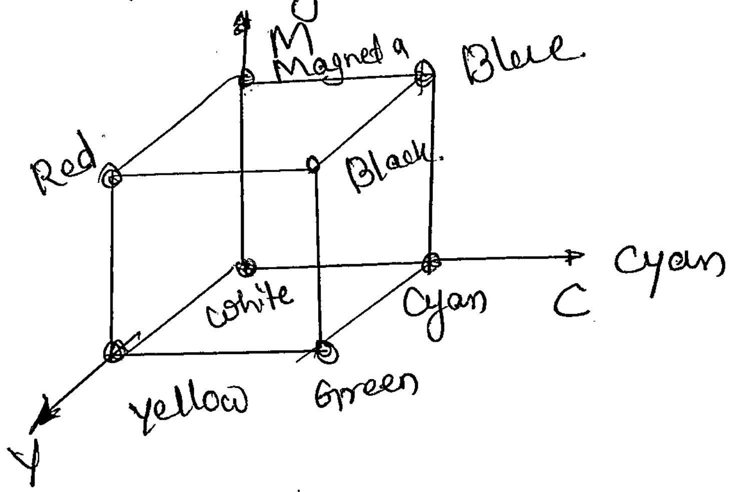


* The origin represents black and the diagonally opposite vertex, with co-ordinates (1,1,1) is white the RGB color scheme is an "additive model"

CMY Color Model:

→ The Subtractive color model can be formed with 3-primary colors, Cyan, Magneta, Yellow.

→ The unit cube representation for the CMY model is illustrated in Fig.



→ In CMY model, the spatial position (1, 1, 1) represents black, coz all the components of the incident light are Subtracted.

→ The Origin represents white light.

- Equal amounts of each primary colors produce shades of gray along the main diagonal of the cube.
- combination of cyan & magneta, produces blue light coz the red & green components of the incident light are absorbed.
- ~~Hly~~ Combination of cyan & yellow, produces green light & combination of magneta and yellow ink yields red light.

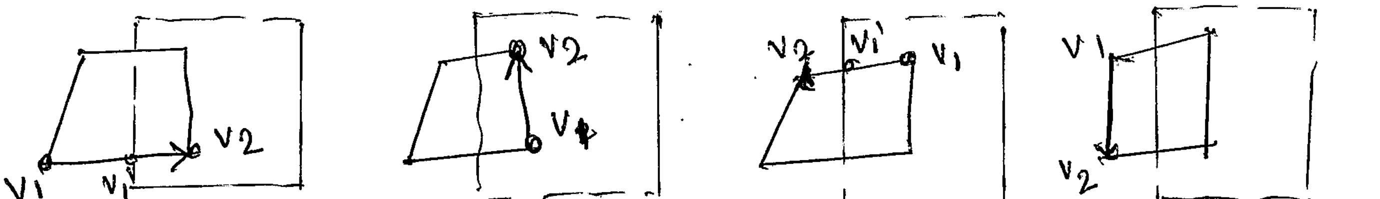
6 a) Sutherland - Hodgman polygon clipping algorithm:

* An efficient method for clipping convex polygon fill area developed by Sutherland & Hodgman, is to send the polygon vertices through each clipping stage, so that a single clipped vertex can be immediately passed to the next stage.

* The final output is a list of vertices, that describe the edges of the clipped polygon fill area.
* The general strategy in this algorithm is to send the pair of end-points for each successive polygon line segment through series of clippers (left, right, bottom & top)
* There are 4 cases that need to be considered when processing a polygon edge against one of the clipping boundaries.

1) One possibility is that the first edge point is outside the clipping boundary and the second-point is inside.
2) or both end-points could be inside the clipping boundary
3) Another possibility is that the first end-pt is inside the clipping boundary & the second end-point is outside
4) And finally, both endpoints could be outside the clipping boundary.

To facilitate the passing of vertices from one clipping stage to the next, the output from each clipper can be formulated as shown below.



(a) out → in
output: V_1', V_2

(b) in → in
out: V_2

(c) in → out
output: V_1'

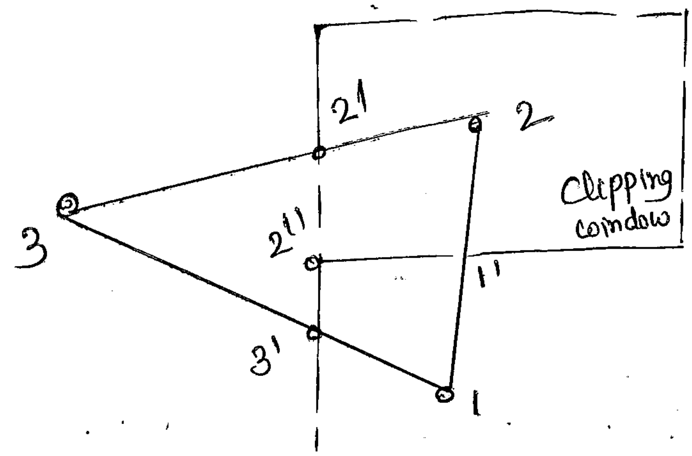
(d) out → out
out: none.

The selection of the vertex edge of intersection for each clipper is given as follows:

1) If the first input vertex is outside the clipping window border, & second vertex is inside, both the intersection point of the polygon edge with the window border & the second vertex are sent to the next clipper
2) If both input vertices are inside the clipping window border, only the second vertex is sent to the next clipper

3) If the first vertex is inside this clipping window border & the second vertex is outside, only the polygon edge-intersection position with the clipping window border is sent to the next clipper.

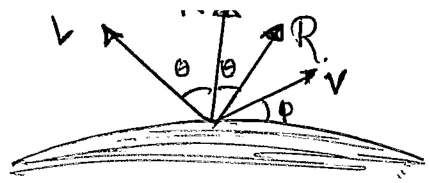
4) If both input vertices are outside this clipping window boundary, no vertices are sent to the next clipper.



Input edge	Left Clipper	Right Clipper	Bottom Clipper	Top Clipper
[1, 2] :	(in - in) → {2}			
[2, 3] :	(in - out) → {2'}	{2, 2'} : (in - in) → {2'}	{2', 3'} : in-out → {2''}	
[3, 1] :	(out - in) → {3', 1}	{2', 3'} : (in → in) → {3'}	{3', 1} : out-out {1}	
		{3', 1} : (in, in) → {1}	{1, 2} : (out - in → {1', 2})	{2'', 1'} : in-in → {1'
		{1, 2} : (in-in) → {2}	{2, 2} : in → in {2'}	{1', 2} : {in-in → {2
				{2, 2'} : (in-in → {2'
				{2, 2''} : in-in {2''

6b) Phong Lighting model:

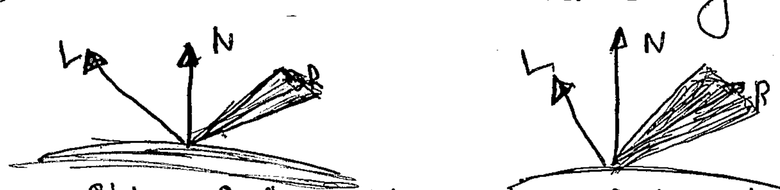
* The bright spot, or specular reflection, that we can see on a shiny surface is the result of total reflection of the ~~to~~ incident light in a concentrated region around the specular-reflection angle.
* The fig below shows specular-reflection direction for a position on an illuminated surface.



1. 'N' represents : unit normal surface vector, The specular reflection angle equals the angle of the incident light, with the two angles measured on opposite sides of the unit normal surface vector N
2. 'R' represents the unit vector in the direction of ideal specular reflection
3. 'L' is the unit vector directed toward the point light source
4. 'V' is unit vector pointing to the viewer from the selected surface position.

* Angle ϕ is the viewing angle relative to the specular reflection direction R.
* An empirical model for calculating the specular-reflection range, developed by Phong Bui Tuong, called the Phong-specular reflection model or simply Phong Model sets the intensity of specular reflection proportional to $\cos^{ns}\phi$.

- Angle 'ϕ' can be assigned values in the range $0°$ to $90°$ so that $\cos\phi$ varies from 0 to 1
- the value assigned to the specular-reflection co-eff ns is determined by the type of surface that we want to display.
- A very shiny surface is modelled with a large value for ns. (say 100 or more) & smaller values (down to 1) are used as duller surfaces.
- For a perfect reflector, ns is infinite, For a rough surface, such as chalk 'ns' is assigned a value near 1

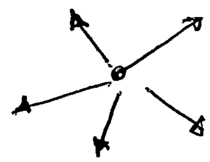


Shiny surface. (large ns)   Dull surface (small ns)

6c) Light Sources supported by open GL :

1. Point light sources :

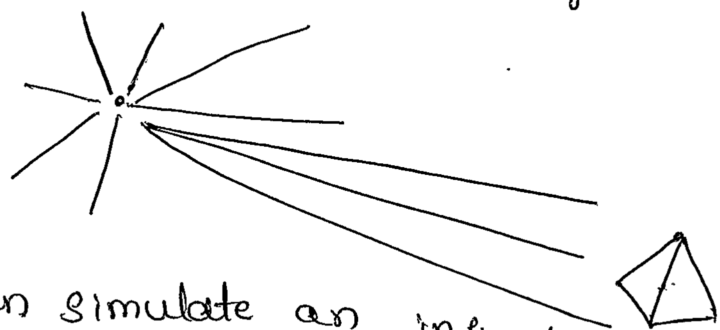
* The simplest model, for an object i.e emitting radiant energy is a point light source. with a single color. Specified with 3, RGB components.



* A point source for a scene by giving its position and the color of the emitted light. Light rays are generated along radially diverging paths from the single-color source position.
* This light-source model is a reasonable approximation for sources whose dimensions are small compared to the size of the objects in the scene.

2. Infinitely Distant light Sources:

* A large light source, such as the sun, that is very far from a scene can also be approximated as a point emitter, but there is a little variation in its directional effects.
* The light path from a distant light source to any position in the scene is nearly const.

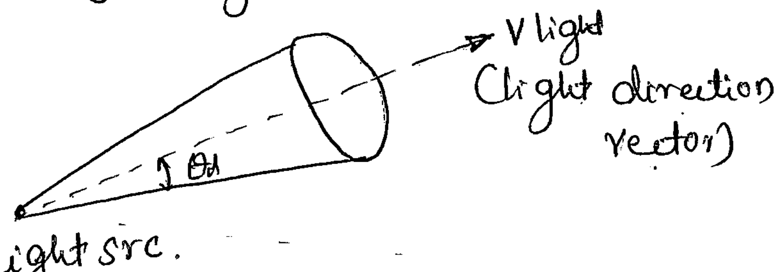


* We can simulate an infinitely distant light source by assigning it a color value and a fixed direction for the light rays emanating from the source.
* The vector for the emission direction and the light-source color are needed in the illumination calculations, but not the position of the source.

3) Spotlight | Directional Light Sources:

* A local light source can be modified easily to pro a directional, or spotlight, beam of light.
* If an object is outside the directional limits of the light source, we exclude it from illumination by tha source.
* One way to set up a directional light source is to assign it a vector direction & angular limit θ_l meas from the vector direction, in addition to its position an color.
* we can denote Vlight as the unit vector in the light source-direction and Vobj as the unit vector in the direction from the light position to an object position

Then $V_{obj} \cdot V_{light} = \cos\alpha$



* where an angle 'α' is angular distance of the object from the light direction vector.
* If we restrict the angular extent of any light cone so that $0^\circ < \theta_l \leq 90^\circ$, then the object is within the spotlight if $\cos\alpha \geq \cos\theta_l$.
* If $V_{obj} \cdot V_{light} < \cos\theta_l$, however the object is outside the cone.

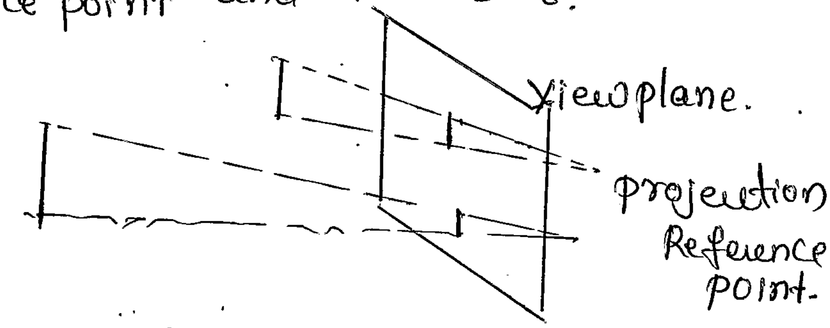
Module-4

7a) Perspective projections

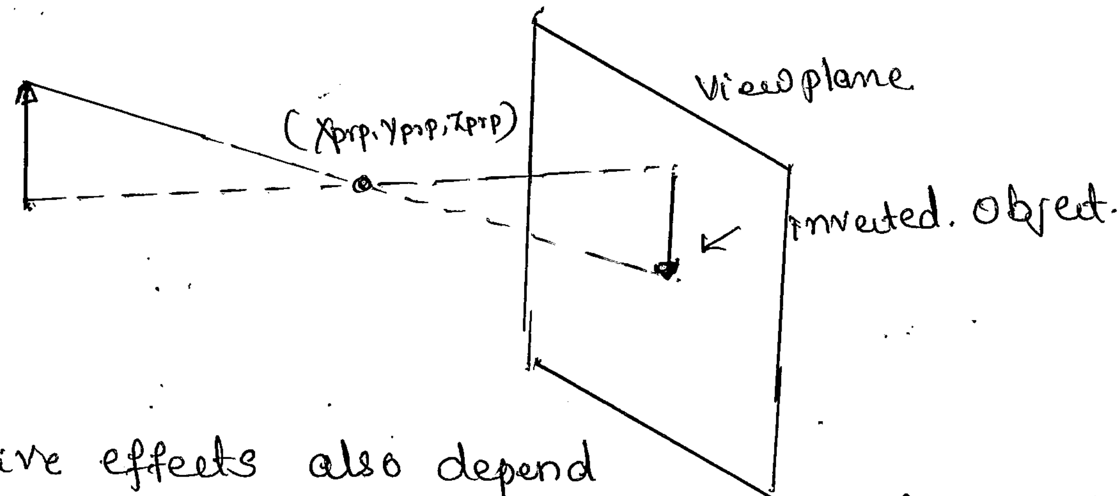
* we can approximate this geometric-optics effect by projecting objects to the view plane along converging paths to a position called "projection reference point"

* Objects are then displayed with foreshortening effects, and projections of distant objects are smaller than the projections of objects of the same size, that are closer to the view-plane.

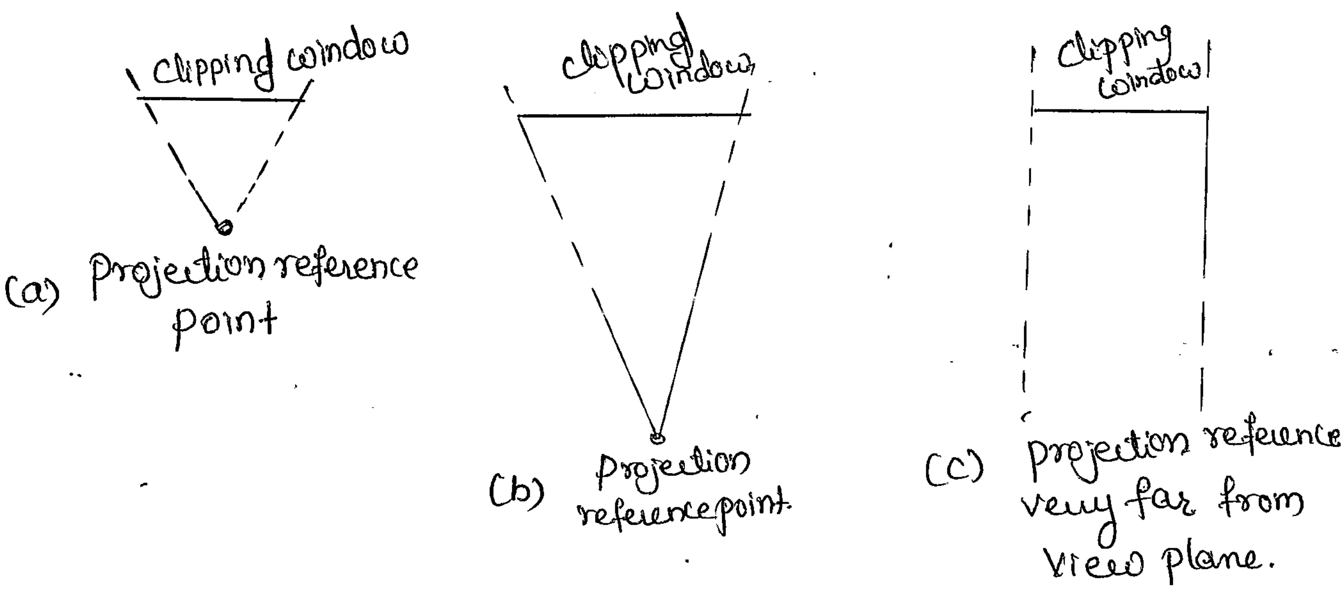
* The view-plane is usually placed between projection reference point and the scene.



* If the projection reference point is between the view plane and the scene, the objects are inverted on the view-plane.

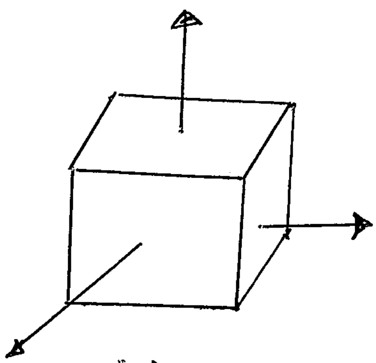


* perspective effects also depend upon the distance between the projection reference point and the view plane.

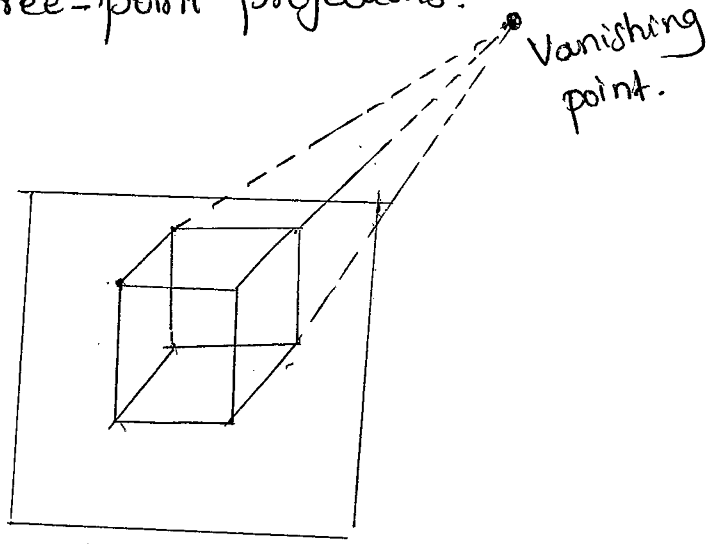


<u>Vanishing points for perspective projections.</u>

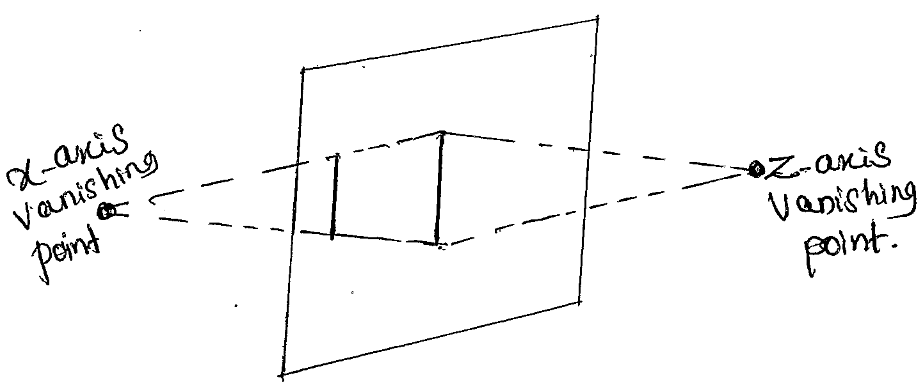
* The point at which a set of projected parallel line appears to converge is called a "vanishing point".
* Each set of projected parallel lines has a seperate vanishing point.
* For a set of lines that are parallel to one of the principa axes of an object, the vanishing point is referred to as a principal vanishing point.
* We control the no of principal vanishing points (one, two or three) with the orientation of the projection plane and perspective projections are accordingly classified as one-pt, two-pt or three-point projections.



(a) principal axes for Cube.



(b) One-point perspective projection



(c) two-point perspective projection.

principal vanishing points for perspective-projection views of a cube.

7b) Depth - Buffer algorithm:

* A commonly used image-space approach for detecting visible surfaces is the depth-buffer method, which compares surface depth values throught a scene for each pixel position on the projection plane.
* The algorithm is usually applied to scenes contain only polygon surfaces, because depth values can be computed very quickly and the method is easy to implement.
* This visibility-detection approach is also frequently alluded to as z-buffer method, because object depth is usually measured along the z-axes of a viewing system.

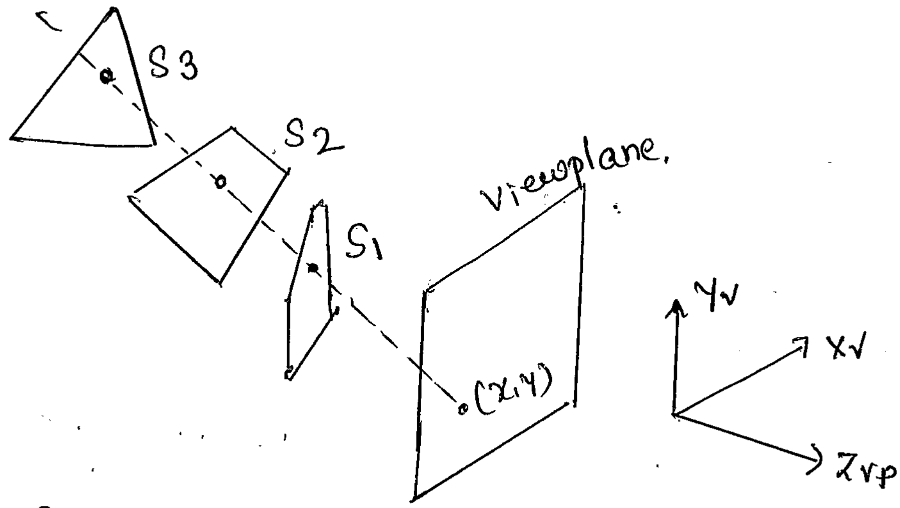


Fig. above shows three surfaces at varying distances along the orthographic projection line from position (x,y) on a view-plane.

* These surfaces can be processed in any order.
* If a surface is closer than any previously processed surfaces, its surface color is calculated & saved, along with its depth.
* The visible surfaces in a scene are represented by the set of surface colors that have been saved after all surface processing is completed.
* A depth buffer is used to store depth values for each (x,y) position as surfaces are processed, and the frame buffer.

Depth-Buffer Algorithm:

1. Initialize the Depth buffer and Frame buffer, so tha for all buffer positions (x, y),
   depthBuff $(x,y) = 1.0$, framebuf (x,y) = backgndco
2. process each polygon in a scene, one at a time as follows:
   - For each projected (x, y) pixel position of a polygon, Calculate the depth z.
   - If $z <$ depthBuffer (x, y), compute the Surface color at that position and set
     depthBuffer $(x,y) = z$, frameBuff (x,y) = surfcolor (x,y).

After all surfaces have been processed, the depth buffer contains depth values for the visible surfaces. and frame buffer contains the corresponding color values for those surfaces.

8a) Differences between perspective and parallel projection.

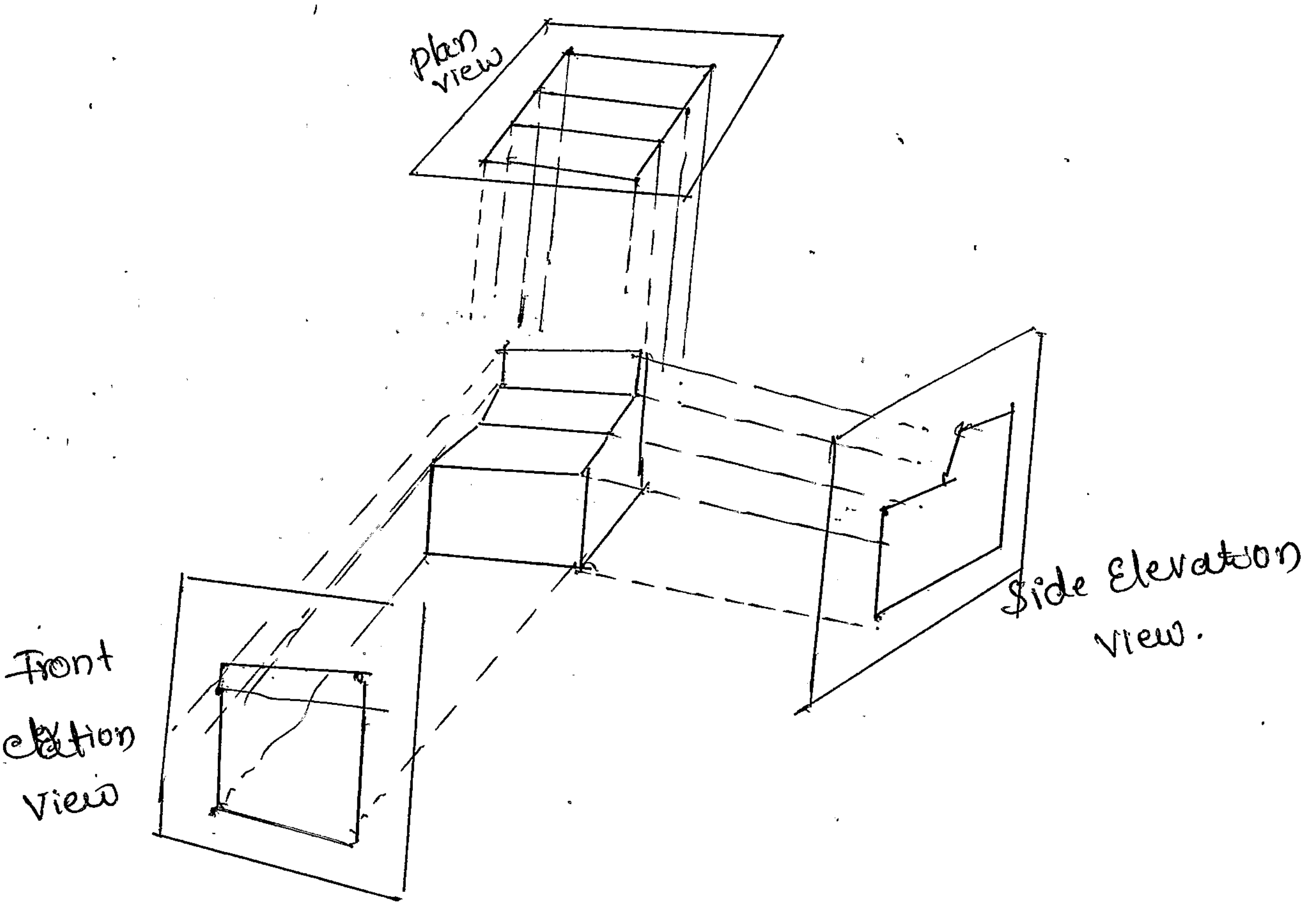
perspective projection	Parallel projection
1. The center of projection is at a finite distance from the viewing plane	1. Center of projection at infinity results with a parallel projection
2. Explicitly specify: center of projection	2. Direction of projection is specified
3. Size of the object is inversely proportional to the distance of the object from the center of projection.	3. No change in the size of object
4. produces realistic views, but doesn't preserve relative proportion of objects	4. Preserves relative proportions of objects, but doesn't give us a realistic representation of the appearance of object.
5. Not useful for recording exact shape and measurements of the object.	5. used for exact measurement.

8b) Orthogonal projections:

→ A transformation of object descriptions to a view-plane along lines that are all parallel to the view-plane normal vector 'N' is called an orthogonal projection also termed as "orthographic projection.

→ This produces a parallel projection transformation in which the projection lines are perpendicular to the view-plane.

→ Orthogonal projections are most often used to produc the front, side and top views of an object.



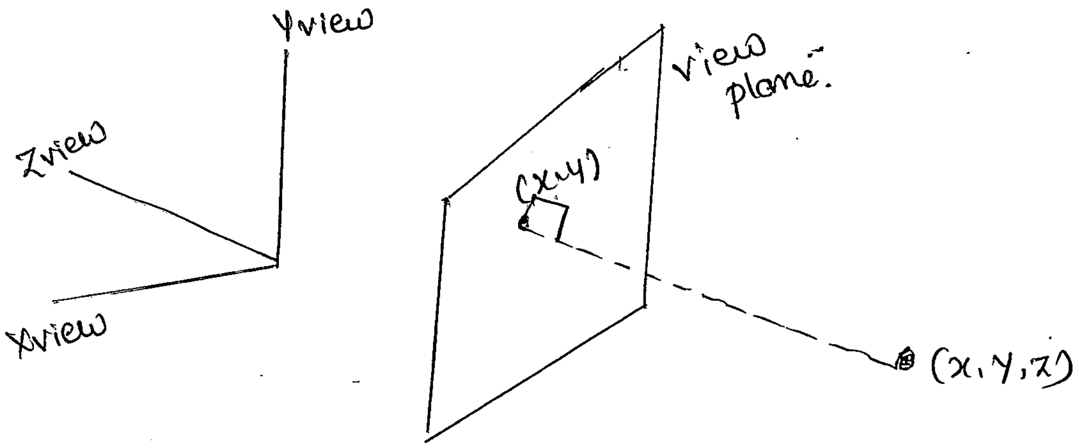
→ Front, side and rear orthogonal projections of an object are called "Elevations". and top orthogonal projection is called a "plan-view"

There are 2 types of orthogonal projections:

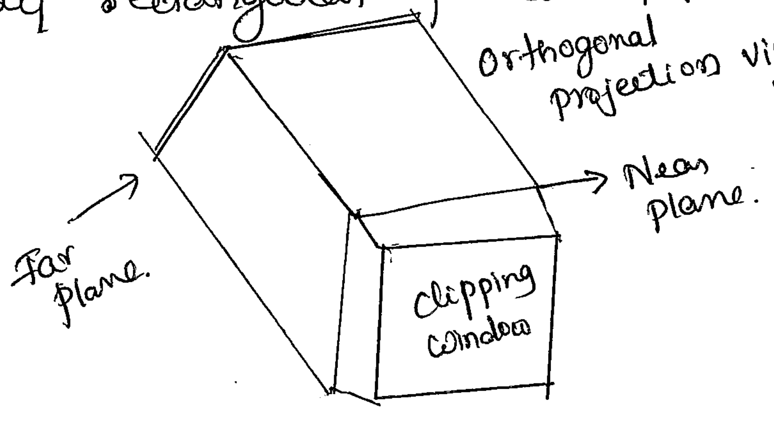
1) axonometric orthogonal projection
2) isometric projection.

* Orthogonal projections that display more than one face of an object, Such views are called axonometric orthogonal projections.
* Most commonly used axonometric projection is the isometric projection, which is generated by aligning the projection plane (or the object) so that plane intersects each-cordinate axis in which the object is defined called the principal axes, at the same distance from the origin.
* with the projection direction parallel to z-view axis, the transformation equations for an orthogonal projection are trivial.

For any position (x, y, z) in viewing co-ordinates, as in figure below, the projection co-ordinates are $x_p = x$, $y_p = y$.



* The two planes near and far planes or front back clipping planes are specified, in orthogonal view volume by "rectangular parallelepiped"



8c) Back-Face Detection :

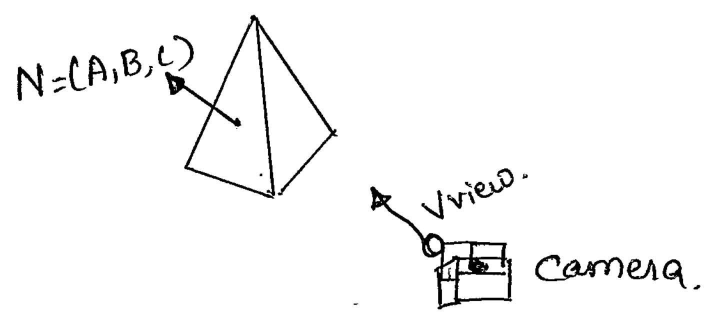
- A fast and simple object-space method for locating the back faces of a polyhedron is based on front back-tests.
- A point (x, y, z) is behind a polygon surface if

$$Ax + By + Cz + D < 0$$

where A, B, C and D are the plane parameters for the polygon.
- We can simply the back-face test by considering the direction of the normal vector N for a polygon surface.

If V view is a vector in the viewing direction from our camera position, as shown in fig below, then a polygon is a back face if,

$$V_{view} \cdot N > 0$$



* In a right handed co-ord viewing system with the viewing direction along the negative z_v axis a polygon is back face if the z component, C, of its normal vector N satisfies $C < 0$

* we cannot see any face, whose normal has z component $C = 0$, because our viewing direction is grazing that polygon. Thus in general, we can label any polygon as a back face if its normal vector has a z-component value that satisfies the inequality. $C \leq 0$

* By examining the C-parameter for different planes describing an object, we can immediately identify all the back faces.

Module 5

9a) Six -classes of logical -input devices:

1. **String**: A string device is a logical device that provides the ASCII values of input characters to the user program. This logical device is usually implemented by means of physical keyboard.

2. **Locator**: A locator device provides a position in world co-ordinates to the user program, It is usually implemented by means of pointing devices such as mouse or track-ball.

3. **Pick**: A pick device returns the identifier of an object on the display to the user program. It is usually implemented with the same physical device as the locator but has a seperate Software interface to the user-program. In open GL, we can use a process of selection to accomplish picking.

4. **Choice**: A choice device allows the user to select one of a discrete number of options. In openGL we can use various widgets provided by the window system. A widget is a graphical interactive component provided by the window system or a toolkit. The widgets include menus, scroll bars and graphical buttons. For Ex: a menu with 'n' selections acts as a "choice." device, allowing user to select one of 'n' alternatives.

5. **Valuators**: They provide analog input to the user program on some graphical systems. there are boxes or dials to provide value.

6. **Stroke**: A stroke device returns array of locations, Ex: pushing down a mouse button, starts the transfer of data into specified array and releasing of button ends this transfer.

9b) The application program can obtain the measure and trigger in 3 - distinct modes:

1. <u>Request Mode:</u>

→ In this mode, measure of the device is not return to the program until the device is triggered.

For Ex: consider a typical C-program which reads a character input using scanf(), when the program needs the input, it halts when it encounters the scanf() statement and waits while user type characters at the terminal. The data is placed in a keyboard buffer (measure) whose contents are returned to the program only after enter key (trigger) is pressed.

- Another Ex, consider a logical device such as locator we can move out pointing device to the desired location and then trigger, the device with its button, the trigger will cause the location to be returned to the application program.

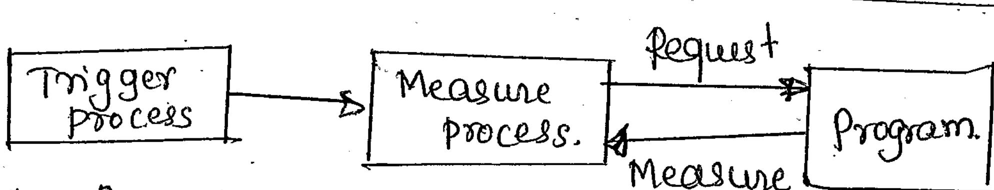


Fig. Request Mode.

2. <u>Sample Mode:</u>

* In this Mode input is immediate, as soon as the function call ~~is~~ in the user program is executed, the measure is returned. Hence no trigger is needed.

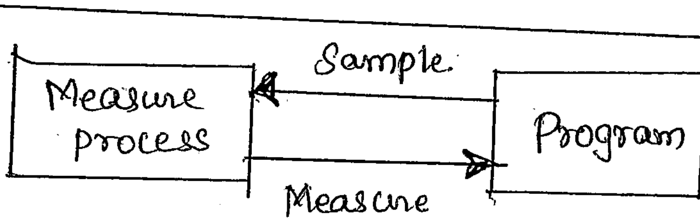


Fig. Sample Mode.

* Both Request mode and sample modes are useful

for the situation, if and only if there is a single input device from which the input is to be taken. However, in case of flight simulators or computer games, variety of input devices are used and these mode cannot be used. Thus event mode is used

3. Event Mode:

* This mode can handle the multiple interactions.
- Suppose that we are in an environment, with multiple input devices, each with its own trigger and each running a measure process.
- whenever a device is triggered, an event is generated. The device measure including the identifier for the device is placed in an event queue
- If the queue is empty, then the application program will wait until an event occurs. If there is an event in a queue, the program can look at the first event type and then decide what to do.

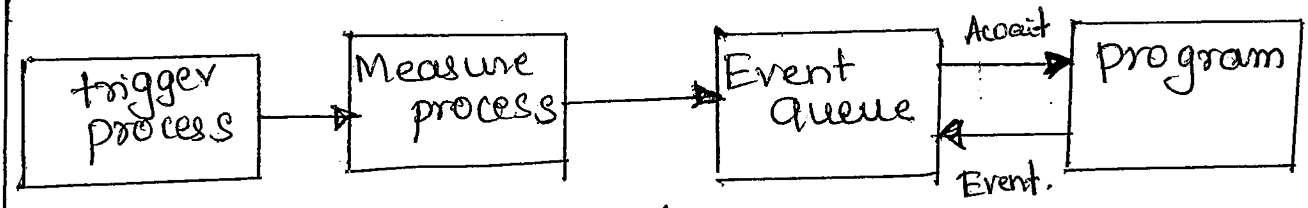


Fig. Event mode Model.

9c) Display List:

* The user program is processed by the host computer which results a compiled list of instruction that was then sent to the display processor, where the instruction are sorted in a display memory called as "display file" or "display list".
* Display processor exeutes its display list contents repeatedly at a sufficient high rate to produce flicker-free image.

* Display lists are defined similarly to the geometric primitives. i.e glNewList() at the beginning and glEndList() at the end is used to define a display list.

* Each display list must have a unique identifier - an integer i.e usually a macro defined in the C-program by means of # define directive to an appropriate name for the object in the list.

For Ex, the following code defines, blue colored box. (square).

```
#define Box 1
glNewList (BOX, GL-COMPILE)
  glBegin (GL-POLYGON)
    glcolor3f (0.0, 0.0, 1.0) ;      // bluecolor
      glvertex2f (-1.0, -1.0);
      glvertex2f (1.0, -1.0);
      glvertex2f (1.0, 1.0);
      glvertex2f (-1.0, 1.0);
  glEnd();
glEndList();

glMatrixMode (GL_PROJECTION);
~~for(l~~
```

* The Flag GL_COMPILE indicate the System to Send the list to the Server, but not to display its contents.

* If we want an immediate display of the contents while the list is being constructed then GL_COMPILE_AND_EXECUTE flag is set.

10 a) Bezier curves:

→ It was developed by French engineer, pierre Bezier, for use in the design of Renault automobile ~~mod~~ bodies

* Bezier curves have number of properties that make them highly useful and convinient for curve and surface design. & easy to implement.
* Bezier curves can be controlled by any number of control points, and some graphics packages limit the no of control points to four.

→ we first consider the general case of n+1 control-point positions, denoted as $P_k = (x_k, y_k, z_k)$ with 'k' varying from 0 to n.

→ These co-ordinate points are blended to produce the following position vector $P(u)$, which describes the path of an approximating Bezier polynomial function between P_0 and P_n.

$$P(u) = \sum_{k=0}^{n} P_k \cdot BEZ_{k,n}(u), \quad 0 \le u \le 1$$

→ Bezier blending functions $BEZ_{k,n}(u)$ are the Bernstein polynomials.

$$BEZ_{k,n}(u) = C(n,k)\, u^k (1-u)^{n-k}.$$

where parameters $C(n,k)$ are the binomial co-fficients.

$$C(n,k) = \frac{n!}{k!\,(n-k)!}$$

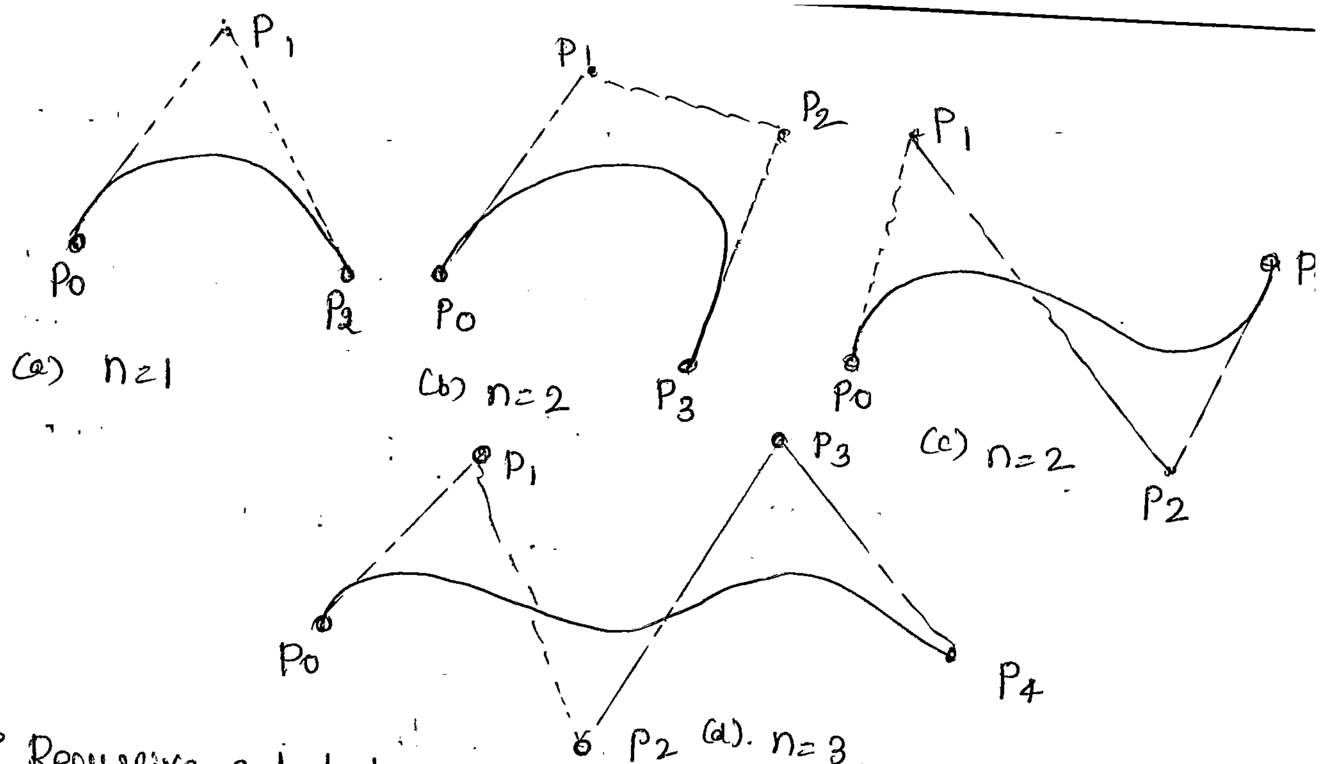
→ A set of three equations for the individual curve co-ord's can be represented as,

$$x(u) = \sum_{k=0}^{n} x_k \cdot BEZ_{k,n}(u)$$

$$y(u) = \sum_{k=0}^{n} y_k \cdot BEZ_{k,n}(u)$$

$$z(u) = \sum_{k=0}^{n} z_k \cdot BEZ_{k,n}(u).$$

→ Below Figure demonstrates the appearance of some Bezier curves for various selections of control points in the xy plane ($z=0$).



→ Recursive calculations can be used to obtain successive binomial-co-eff values as

$$C(n,k) = \frac{n-k+1}{k} C(n,k-1)$$

for $n \geq k$, Also, the Bezier blending functions satisfy the recursive relationship.

$$BEZ_{k,n}(u) = (1-u)\, BEZ_{k,n-1}(u) + u\, BEZ_{k-1,n-1}(u).$$

with $BEZ_{k,k} = u^k$, $BEZ_{0,k} = (1-u)^k$. $n > k \geq 1$

10b) Double Buffering

* one method for producing a real-time animation with a raster system is to employ two refresh buffers.
* We create a frame for the animation in one of the buffers. Then, while the screen is being refreshed from that buffer we construct the next frame in the other buffer.
* when that frame is complete, we switch the roles of the two buffers, so that the refresh routines use the second buffer during the process of creating the next frame in the first buffer.
* when a call is made to switch two refresh buffers, the interchange could be performed at various times.

* The most straight forward implementation is to switch the two buffers, at the end of the current refresh cycle, during the vertical retrace of the electron beam.
* If a program can complete the construction of a frame within the time of refresh cycle, say 1/60 of a second. each motion sequence is displayed in synchronization with the screen refresh rate.
* If the time to construct a frame is longer than the refresh time, the current frame is displayed for two or more refresh cycles while the next animation frame is being generated.
* Similarly, if the frame construction time is 1/25 of ~~of~~ a second, the animation frame rate is reduced to 20 frames per second because each frame is displayed three times.
* Irregular animation frame rates can occur with double buffering when the frame construction time is very nearly equal to an integer multiple of the screen refresh ~~b~~ time the animation frame rate can change abruptly and erratically.
* one way to compensate for this effect is to add a small delay to the program.
* Another possibility is to alter the motion or scene description to shorten the frame construction time.

Double buffering is activated in openGL using GLUT Function

glutInitDisplayMode (GLUT_DOUBLE);

This provides two buffers called the "front buffer" and the "back buffer", that we can use alternately to refresh the screen display.

10c) Features that a good interactive program:

1. A smooth display, showing neither flicker, nor any artifacts of the refresh process.
2. A variety of interactive devices on the display.
3. A variety of methods for entering and displaying information
4. An easy to use interface that doesn't require substantial effort to learn.
5. Feedback to the user.
6. Tolerance for user errors.
7. A design that incorporates consideration of both the visual and motor properties of the human.

(Any 4).

— x —