B.Tech.

ECS-504

FIFTH SEMESTER EXAMINATION, 2010-11

COMPUTER GRAPHICS

Time: 3 *Hours* *Total Marks:* 100

Note: *Attempt all questions.*

1. Answer any two parts: **(5×4=20)**

Q.1. (*a*) What are the criteria that should be satisfied by a good line drawing algorithm? Explain.

Ans. The criteria that should be satisfied by a good line drawing algorithm is accomplished by calculating intermediate positions along the line path between two specified endpoint positions. An output device is then directed to fill in these positions between the endpoints. For analog devices, such as a vector pen plotter or a random-scan display, a straight line can be drawn smoothly from one endpoint to the other. Linearly varying horizontal and vertical deflection voltages are generated that are proportional to the required changes in the x and y directions to produce the smooth line.

Digital devices display a straight line segment by plotting discrete points between the two endpoints. Discrete coordinate positions along the line path are calculated from the equation of the line. For a raster video display, the line color (intensity) is then loaded into the frame buffer at the corresponding pixel coordinates. Reading from the frame buffer, the video controller then "plots" the screen pixels. Screen locations are referenced with integer values, so plotted positions may only approximate actual Line positions between two specified endpoints. A computed line position of (10.48, 20.51), for example, would be converted to pixel position (10,211). Thus rounding of coordinate values to integer's causes lines to be displayed with a stair step appearance ("the jaggiest"), the characteristic stair step shape of raster lines is particularly noticeable on systems with low resolution, and we can improve their appearance somewhat by displaying them on high-resolution systems. More effective techniques for smoothing raster lines are based on adjusting pixel intensities along the line paths. For the raster-graphics device-level algorithms, object point positions are specified directly in integer device coordinates. For the time being, we will assume that pixel positions are referenced according to scan-line number and column number (pixel position across a scan line). Scan lines are numbered consecutively from 0, starting at the bottom of the screen; and pixel columns are numbered from 0, left to right across each scan line. We also consider alternative pixel addressing schemes. To load a specified color into the frame buffer at a position corresponding to column x along scan line y, we will assume we have available a low-level procedure of the form Stair step effect (jaggiest) produced when a line is generated as a series of pixel positions.

Q.1. (*b*) Explain the mid point circle generating algorithm.

Ans. Mid point circle algorithm based on the spatial relationship between on artibilary point (x, y) and a circle of radius r centered at the origin.

$$f(x, y) = x^2 + y^2 - r^2 = \begin{cases} < 0 \text{ for } (x, y) \text{ inside the circle} \\ 0 \text{ for } (x, y) \text{ on the circle} \\ > 0 \text{ for } (x, y) \text{ outside the circle} \end{cases}$$

Now consider the coordinates of the point halfway between pixel T and pixel S in Fig.: $\left(x_i + 1, y_i - \frac{1}{2}\right)$. This is called the midpoint and we use it to define a decision parameter:

$$P_i = f\left(x_1 + 1, y_i - \frac{1}{2}\right)$$

$$= (x_i + 1)^2 + \left(y_i - \frac{1}{2}\right)^2 - r^2$$

If P_i is negative, the midpoint is inside the circle, and we choose pixel T. On the other hand, if p_i is positive (or equal to zero), the midpoint is outside them circle (or on the circle), and we choose pixel S. Similarly, the decision parameter for the next step is

$$p_{i+1} = (x_{i+1} + 1)^2 + \left(y_{i+1} - \frac{1}{2}\right) - r^2$$

Since $x_{i+1} = x_i + 1$, we have

$p_{i+1} - p_i =$

$$[(x_i + 1) + 1]^2 - (x_i + 1)^2 + \left(y_{i+1} - \frac{1}{2}\right)^2 - \left(y_i - \frac{1}{2}\right)^2$$

Hence

$$p_{i+1} = p_i + 2(x_i + 1) + 1 + (y^2_{i+1} - y^2_i) - (y_{i+1} - y_i)$$

If pixel T is chosen (meaning $p_i < 0$), we have y_{i+1} - y_i. On the other hand, if pixel S is chosen (meaning $p_i > 0$), we have $y_{j+1} = y_i - 1$. Thus

$$p_{i+1} = \begin{cases} p_i + 2(x_i + 1) + 1 & \text{if } p_i < 0 \\ p_i + 2(x_i + 1) + 1 - 2(y_i - 1) & \text{if } p_i \geq 0 \end{cases}$$

We can continue to simplify this in terms of (x_i, y_i) and get

$$p_{i+1} = \begin{cases} p_i + 2x_i + 3 & \text{if } p_i < 0 \\ p_i + 2(x_i + y_i) + 5 & \text{if } p_i \geq 0 \end{cases}$$

Or we can write it in terms of (x_{i+1}, y_{i+1}) and have

$$p_{i+1} = \begin{cases} p_i + 2x_{i+1} + 1 & \text{if } p_i < 0 \\ p_i + 2(x_{i+1} - y_{i+1}) + 1 & \text{if } p_i \geq 0 \end{cases}$$

Finally, we compute the initial value for the decision parameter using the original definition of p_i and $(0, r)$:

$$p_i = (0 + 1)^2 + \left(r - \frac{1}{2}\right)^2 - r^2 = \frac{5}{4} - r$$

One can see that this is not really integer computation. However, when r is an integer we can simply set $p_1 = 1 - r$. The error of being, less than the precise value does not prevent p_1 from getting the appropriate sign. It does not affect the rest of the scan-conversion process either, because the decision variable is only updated with integer increments in subsequent steps.

The following is a description of this midpoint circle algorithm that generates the pixel coordinates in the 90° to 45° octant:

```
int x = 0, y = r, p = 1 – r;
while (x < = y) {
set Pixel (x, y);
if(p<0)
p = p + 2x + 3; else
{
p = p + 2(x – y) + 5;
y– –;
}
x ++ ;
}
```

Q.1. (c) Write a short note on the following:
(i) Random scan and Raster scan display
(ii) Frame buffer and video controller.

Ans. Raster scan display:

(1) The Raster scan system is scanning technique in which the electrons sweep from top to bottom and from left to right. The intensity is turned on or off to light and unlighted the pixel.
(2) Raster displays have less resolution.
(3) The lines produced are ziz-zag as the plotted values are discrete.
(4) High degree realism is achieved in picture with the aid of advanced shading and hidden surface technique.
(5) Decreasing memory costs have made raster systems popular.
(6) In this case, the electron beam is swept across the screen, one row at a time from top to bottom.
(7) Picture definition is stored in a memory area called the refresh buffer/frame buffer.
(8) Refreshing on raster scan displays is carried out at the rate of 60 to 80 frames/second.

Random scan display:

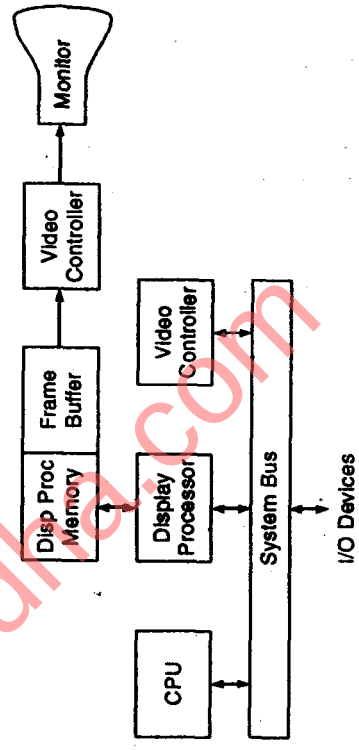
(1) Random scan is a method in which the display is made by electronic beam, which is directed, only to the points or parts of the screen where the picture is to be drawn.
(2) Random displays have high resolutions since the picture definition is stored as a set of line drawing commands and not as a set of intensity values.

(3) Smooth lines are produced as the electron beam directly follows the line path.
(4) Realism is difficult to achieve.
(5) Random-scan systems are generally costlier.
(6) Here CRT has the electron beam directly only to the parts of the screen where a picture is to be drawn.
(7) Picture definition is stored as a set of line drawing commands in an area of memory referred to as refresh display file.
(8) Random scan systems are designed to draw all the component lines of a picture 30 to 60 times each second.

***(ii)* Frame buffer:** 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 a 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.

Intensity range for pixel positions depends on the capability of the raster system. In a simple black-and-white system, each screen point is either on or off, so only one bit per pixel is needed to control the intensity of screen positions. For a bi-level system, a bit value of 1 indicates that the electron beam is to be turn on at that position, and a value of 0 indicates that the beam intensity is to be off. Additional bits are needed when color and intensity variations can be displayed. Up to 24 bits per pixel are included in high-quality systems, which can require several megabytes of storage for the frame buffer, depending on the resolution of the system.

Video Controller: Interactive raster graphics systems typically employ several processing units. In addition to the central processing unit, or CPU, a special-purpose processor, called the video controller or display controller, is used to control the operation of the display device. A fixed area of the system memory is reserved for the frame buffer, and the video controller is given direct access to the frame-buffer memory. The co-ordinates of the graphics monitor starts at the lower left screen corner. The below diagram shows the refresh operation of video controller.



2. Answer any two parts: **(6×2=12)**

Q.2. (*a*) Describe the Cohen Sutherland line clipping algorithm with suitable example.

Ans. The Cohen-Sutherland Algorithm: In this algorithm we divide the line clipping process into two phases: (1) identify those lines which intersect the clipping window and so need to be clipped and (2) perform the clipping. All lines fall into one of the following *clipping categories:*

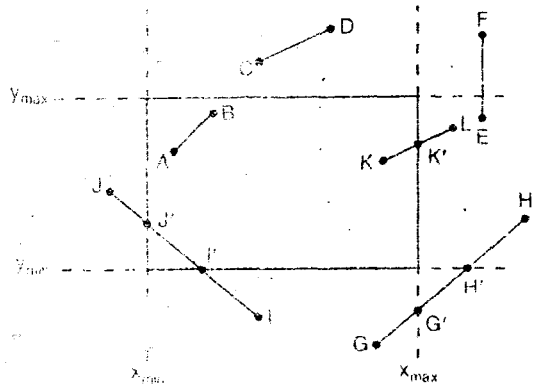
1. *Visible*—both endpoints of the line lie within the window.
2. *Not visible*—the line definitely lies outside the window. This will occur if the line from (x_1, y_1) to (xc_2, y_2) satisfies any one of the following four inequalities:

   $x_1, x_2 > x_{max}$ $y_1, y_2 > y_{max}$

   $x_1, x_2 < x_{min}$ $y_1, y_2 < y_{min}$
3. *Clipping candidate*—the line is in neither category 1 nor 2.

In Fig. line *AB* is in category 1 (visible); lines

CD and EF are in category 2 (not visible); and lines GH, IJ and KL are in category 3 (clipping candidate).



The algorithm employs an efficient procedure for finding the category of a line. It proceeds in two steps:

1. Assign a 4-bit region code to each endpoint of the line. The code is determined according to which of the following nine regions of the plane the endpoint lies in

	1001	1000	1010
Y_{max}	0001	0000	0010
	0101	0100	0110

x_{min} x_{max}

Starting from the left most bit, each bit of the code is set to true (1) or false (0) according to the scheme

Bit 1 = endpoint is above the window = sign $(y - y_{max})$

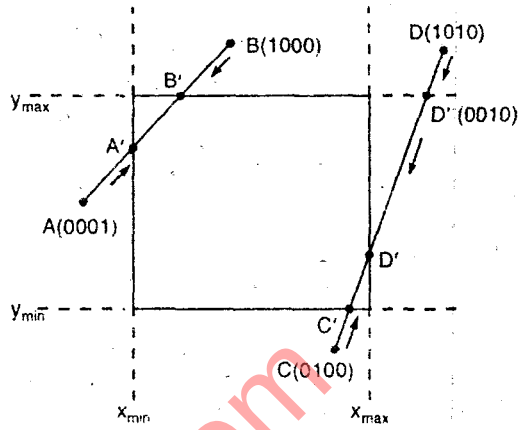
Bit 2 = endpoint is below the window = sign $(y_{min} - y)$

Bit 3 = endpoint is to the right of the window = sign $(x - x_{max})$

Bit 4 = endpoint is to the left of the window = sign $(x_{min} - x)$

We use the convention that sign(a) = 1 if a is positive, 0 otherwise. Of course, a point with code 0000 is inside the window.

2. The line is visible if both region codes are 0000, not visible if the bit wise logical AND of the codes is not 0000, and a candidate for clipping if the bitwise logical AND of the region codes is 0000.



For a line in category 3 we proceed to find the intersection point of the line with one of the boundaries of the clipping window, or to be exact, with the infinite extension of one of the boundaries (see Fig.). We choose an endpoint of the line, say (x_1, y_1), that is outside the window, *i.e.*, whose region, code is not 0000. We then select an extended boundary line by observing that those boundary lines that are candidates for intersection are the ones for which the chosen endpoint must be "pushed across" so as to change a "1" in its code to a "0" (see Fig.). This means:

If bit 1 is 1, intersect with line $y = y_{max}$. If bit 2 is 1, intersect with line $y = y_{min}$. If bit 3 is 1, intersect with line $x = x_{max}$. If bit 4 is 1, intersect with line $x = x_{min}$.

Consider line CD in Fig. If endpoint C is chosen, then the bottom boundary line $y = y_{min}$ is selected for computing intersection. On the other hand, if endpoint D is chosen, then either the top boundary line $y = y_{max}$ or the right boundary line $x = x_{max}$ is used. The coordinates of the intersection point are

$$\begin{cases} x_i = x_{min} \text{ or } x_{max} \\ y_i = y_1 + m(x_i - x_1) \end{cases}$$ if the boundary line is vertical

or

$$\begin{cases} x_i = x_1 + (y_i - y_1)/m \\ y_i = y_{min} \text{ or } y_{max} \end{cases}$$ if the boundary line is horizontal

where $m = (y_2 - y)/(x_2 - x_1)$ is the slope of the line.

Now we replace endpoint (x_1, y_1) with the intersection point (x_i, y_i), effectively eliminating the

portion of the original line that is on the outside of the selected window boundary. The new endpoint is then assigned an updated region code and the clipped line re-categorized and handled in the same way. This interactive process terminals when finally reach a clipped line that belongs to either category 1(visible) or category 2 (not visible).

Example: A clipping window PQRS has left corner at (3, 4) and upper corner at (10, 9). Find the section of the clipped line AB using Cohen-Sutherland line clipping algorithm. Also find the region codes on which the end points of line CD and EF rest.

The co-ordinates are A(2, 1) B(9, 2) C(1, 4) D(4, 6) E(7, 11), F(11, 7)

The region code for any point (x, y) will be set according to the scheme based on Cohen Sutherland algorithm

Bit 1 = sign$(y - yw_{max})$ = sign$(y - 9)$

Bit 2 = sign $(yw_{min} - y)$ = sign $(4 - y)$

Bit 3 = sign$(x - xw_{max})$ = sign$(x - 10)$

Bit 4 = sign $(xw_{min} - x)$ = sign $(3 - x)$

Hence the region code for $A(2, 11)$ -> 1001 and $B(9,2)$ -> 0100

$C(1, 4)$ -> 0001 and D(4, 6)-> 0000 $E(7, 11)$-> 1000 **and** $F(11,7)$-> 0010

Considering clipping of line AB.

Slope of the line AB is m = $(y2 - y1)/(x2 - x1)$ = (2–11) / (9-2) = –9/2. As the line has the slope less than 1, and the line has the code 1001, it intersects xw_{min} for which y can be calculated as $y = yl + m(x - xl) = 11 + m\ (xw_{min} - 2)$ which gives $y = 9.71$, that lies outside the window and above yw_{max} Thus, x is to calculated as $x = x_1 + (y - y_1)\ m = 2 + (yw_{max} - 11)/ -9/7 = 3.55$. And, the intersecting point id (3.55, 9). Similarly, considering the lower end point of the line AB, the region code for the end point B is 0100, hence it intersects the lower line $yw_{min} = 4$, and the x coordinate for the clipping point can be calculated as $x = 2 + (yw_{min} - 11)/-9/7 = 7.44$.

Hence the intersecting point is (7.44, 4). Thus, the clipped line is **A'B' = [(3.55, 9), (7.44, 4)].**

Q.2. (*b*) Discuss the following transformations with a relevant example:

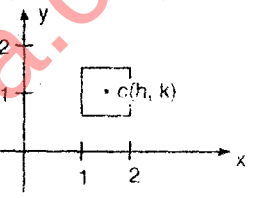
(*i*) Composite transformation

(*ii*) Reflection and shearing.

Ans. (*i*) Composite Transformation: In matrix representations, we can set up a matrix for any sequence of transformations as a composite transformation matrix by calculating the matrix product of the individual transformations. Forming products of transformation matrices is often referred to as a concatenation, or composition, of matrices. For column-matrix representation of coordinate positions, we form composite transformations by multiplying matrices in order from right to left. That is, each successive matrix pre-multiplies the product of the preceding transformation matrix.

Example : Express as a composite transformation matrix i.e. CTM the transformation which magnifies an object about its centre $c(h, k)$ in fig. The required transformation $S_{s,c}$ can be written as

$$S_{s,c} = T_{v} \cdot S_{s,s} \cdot T_{-v}$$



(*ii*) Reflection and Shearing

Reflection: A reflection is a transformation that produces a mirror image of an object. The mirror image for a two-dimensional reflection is generated relative to an axis of reflection by rotating the object 180° about the reflection axis. We can choose an axis of reflection in the xy plane or perpendicular to the xy plane. When the reflection axis is a line in the xy plane, the rotation path about this axis is in a plane perpendicular to the xy plane. For reflection axes that are perpendicular to the xy plane, the rotation path is in the xy plane. Following are examples of some common reflections.

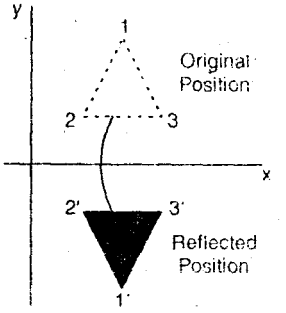


Fig.1 Reflection of an object about the x axis.

Reflection about the line $y = 0$, the x axis, is accomplished with the transformation matrix

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

The transformation keeps x values the same, but "flips" the y values of coordinate positions.

Shearing: A transformation that distorts the shape of an object such that the transformed shape appears as if the object were composed of internal layers that had been caused to slide over each other is called a shear. Two common shearing transformations are those that shift coordinate w values and those that shift y values. An x-direction shear relative to the x axis is produced with the transformation matrix

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

which transforms coordinate positions as

$$x' = x\ 4 + Sh_x\ (y - y_{ref}),\ y = y'$$

Any real number can be assigned to the shear parameter sh_x. A coordinate position (x, y) is then shifted horizontally by an amount proportional to its distance (y value) from the x axis ($y = 0$). Setting sh_x, to 2, for example, changes the square in Fig. 2 into a parallelogram. Negative values for sh_x, shift coordinate positions to the left.

We can generate x-direction shears relative to other reference lines with

$$\begin{bmatrix} 1 & sh_x & -sh_x \cdot y_{ref} \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

with coordinate positions transformed as

$$x' = x + sh_x\ (y - y_{ref}),\ y' = y$$

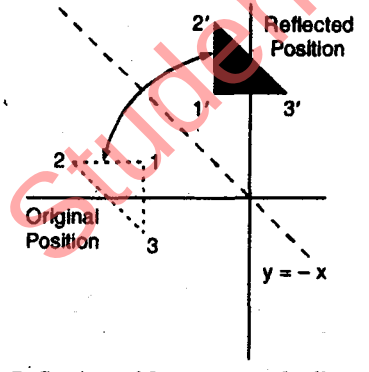


Fig.2 Reflection with respect to the line y = – x

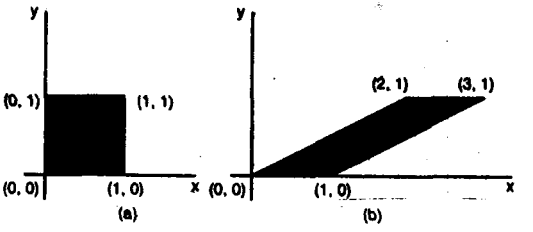


Fig.3 A unit square (a) is converted to a parallelogram (b) using the x-direction shear matrix 5-53 with sh_x 2.

Q.2. (c) Write an algorithm for polygon clipping.

Ans. Polygon Clipping (Sutherland Hodgeman): We can correctly clip a polygon by processing the polygon boundary as a whole against each window edge. This could be accomplished by processing all polygon vertices against each clip rectangle boundary in turn. Beginning with the initial set of polygon vertices, we

could first clip the polygon against the left rectangle boundary to produce a new sequence of vertices. The new set of vertices could then k successively passed to a right boundary clipper, a bottom boundary clipper, and a top boundary clipper, as in Fig. 1.

At each step, a new sequence of output vertices is generated and passed to the next window boundary clipper.

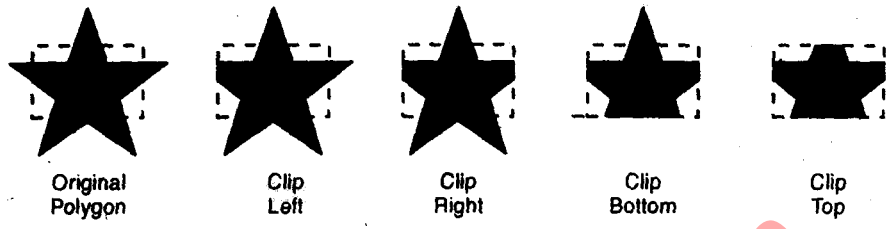


Fig: 1 Clipping a polygon against successive window boundaries.

There are four possible cases when processing vertices in sequence around the perimeter of a polygon. As each pair of adjacent polygon vertices is passed to a window boundary clipper, we make the following tests:

(1) If the first vertex is outside the window boundary and the second vertex is inside, both the intersection point of the polygon edge with the window boundary and the second vertex are added to the output vertex list.

(2) If both input vertices are inside the window boundary, only the second vertex is added to the output vertex list.

(3) If the first vertex is inside the window boundary and the second vertex is outside, only the edge intersection with the window boundary is added to the output vertex list.

(4) If both input vertices are outside the window boundary, nothing is added to the output list. These four cases are illustrated in Fig.2 for successive pairs of polygon vertices. Once all vertices have been processed for one clip window boundary, the output list of vertices is clipped against the next window boundary.

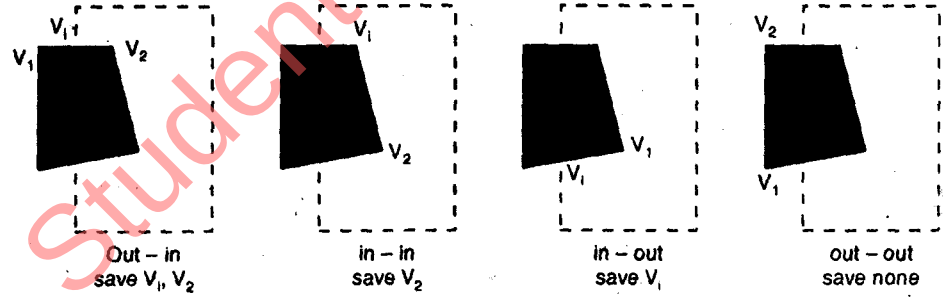


Fig: 2 Successive processing of pairs of polygon vertices against the left window boundary

Convex polygons are correctly clipped by the Sutherland-Hodgeman algorithm, but concave polygons may be displayed with extraneous lines. This occurs when the clipped polygon should have two or more separate sections. But since there is only one output vertex list, the last vertex in the list is always joined to the first vertex. There are several things we could do to correctly display concave polygons. For one, we could split the concave polygon into two or more convex polygons and process each convex polygon separately. Another possibility is to modify the Sutherland-Hodgeman approach to check the final vertex list for multiple vertex points along any clip window boundary and correctly join pairs of vertices.

3. Write short notes on any two of the following: (6×2=12)

Q.3. (*a*) 3-D transformation

Ans. (*a*) 3-D transformation system defined with 3-D coordinate system, an object obj is considered as a set 7 points

$$\text{obj} = \{p(x, y\ z)\}$$

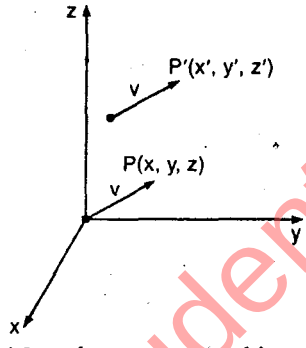
If object is moved to a new position *i.e.*, obj. all of the coordinate point $p'(x', y', z')$ can be obtained from original points $p(x, y, z)$ the transformation only.

Translation: An object is displaced at a given distance and direction from its original position. The direction and displacement of the translation is prescribed by a vector

$V = a\mathbf{I} + b\mathbf{J} + c\mathbf{K}$

The new coordinates of a translated point can be calculated by using the transformation

$$T_v : \begin{cases} x' = x + a \\ y' = y + b \\ z' = z + c \end{cases}$$



(see fig.) In order to represent this transformation as a matrix transformation, we need to use homogeneous coordinates. The required homogeneous matrix transformation can then be expressed as

$$(x'\ y'\ z'\ 1) = (x\ y\ z\ 1)\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ a & b & c & 1 \end{pmatrix}$$

Scaling: The process of scaling changes the dimensions of an object. The factor *s* determines whether the scaling is a magnification, $s > 1$, or a reduction, $s < 1$.

Scaling with respect to the origin, where the origin remains fixed, is effected by the transformation

$$S_{s_x, s_y, s_z} = \begin{cases} x' = s_x \cdot x \\ y' = s_y \cdot y \\ z' = s_z \cdot z \end{cases}$$

In matrix from this is

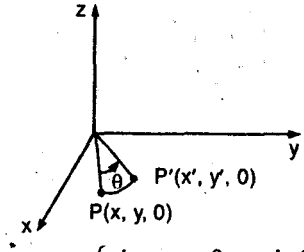
$$S_{s_x, s_y, s_z} = \begin{pmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & s_z \end{pmatrix}$$

Rotation : Rotation in three dimensions is considerably more complex than rotation in two dimensions. In two dimensions, a rotation is prescribed by an angle of rotation θ and a center of rotation *P*. Three dimensional rotations require the prescription of an angle of rotation and an axis of rotation. The conconical rotations are defined when one of the positive *x*, *y* or *z* coordinate axis is chosen as the axis of rotation. Then the construction of the rotation transformation proceeds just like that of a rotation in two dimensions about the origin (see fig.)

Rotation about the z axis: We know that

$$R_{\theta.K} : \begin{cases} x' = x\cos\theta - y\sin\theta \\ y' = x\sin\theta + y\cos\theta \\ z' = z \end{cases}$$

Rotation about the y axis : An analogous derivation leads to



$$R_{\theta.J} : \begin{cases} x' = x\cos\theta + z\sin\theta \\ y' = y \\ z' = -x\sin\theta + z\cos\theta \end{cases}$$

Rotation about the x axis: Similarly:

$$R_{\theta.I} : \begin{cases} x' = x \\ y' = y\cos\theta - z\sin\theta \\ z' = y\sin\theta + z\cos\theta \end{cases}$$

Note that the direction of a positive angle of rotation is chosen in accordance to the right-hand rule with respect to the axis of rotation.

The corresponding matrix transformations are

$$R_{\theta,K} = \begin{pmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

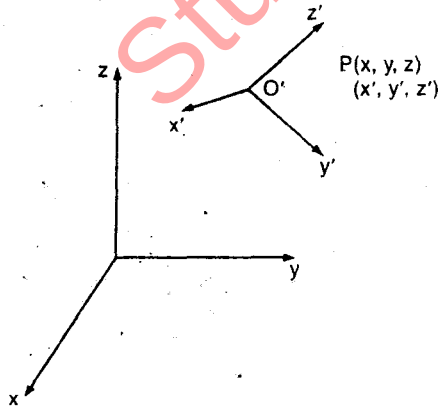
$$R_{\theta,J} = \begin{pmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{pmatrix}$$

$$R_{\theta,I} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & \sin\theta \\ 0 & -\sin\theta & \cos\theta \end{pmatrix}$$

The general use of rotation about an axis L can be built up from these canonical rotations using matrix multiplication.

Coordinate Transformations: We can also achieve the effects of translation, scaling, and rotation by moving the observer who views the object and by keeping the object stationary. This type of transformation is called a *coordinate transformation*.

We first attach a coordinate system to the observer and then move the observer and the attached coordinate system. Next, we recalculate the coordinates of the observed object with respect to this new observer coordinate system. The new coordinate values will be exactly the same as if the observer had remained stationary and the object had moved, corresponding to a geometric transformation.



If the displacement of the observer coordinate system to a new position is prescribed by a vector $\mathbf{V} = a\mathbf{I} + b\mathbf{J} + c\mathbf{K}$, a point $P(x, y, z)$ in the original coordinate system, and

$$\overline{T}_v : \begin{cases} x' = x - a \\ y' = y - b \\ z' = a - c \end{cases}$$

The derivations hold for coordinate scaling and coordinate rotation transformations.

Composite Transformations : More complex geometric and coordinate transformations are formed through the process of *composition of functions*. For matrix functions, however, the process of composition is equivalent to matrix multiplication or concatenation.

1. $A_{\mathbf{V},\mathbf{N}}$ = aligning a vector **V** with a vector **N**.
2. $R_{\theta,L}$ = rotation about an axis L. This axis is prescribed by giving a direction vector **V** and a point P through which the axis passes.
3. $S_{s_x,s_y,s_z,P}$ = scaling with respect to an arbitrary point P.

In order to build these more complex transformations through matrix concatenation, we must be able to multiply translation matrices with rotation and scaling matrices. This necessitates the use of homogeneous coordinates and 4 × 4 matrices. The standard 3 × 3 matrices of rotation and scaling can be represented 2s 4 × 4 homogeneous matrices by adjoining an extra row and column as follows:

$$\begin{pmatrix} a & b & c & 0 \\ d & e & f & 0 \\ g & h & i & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

These transformations are then applied to points $P(x, y, z)$ having the homogeneous form :

$(x\ y\ z\ 1)$

Shearing Transformations : Similar to the two-dimensional transformations, we have three-dimensional shearing transformations also. These transformations cause the image to slant in a given direction of x, y or z. The x-shear maintains the y and z coordinates but changes the values of x coordinates

causing it to tilt left or right depending on the x-shear value so that the image get slanted towards x direction. Similarly y-shear and z-shear transformations yield slanting towards the y and z directions. The general form of the three-dimensional shearing transformation is given by

$$\begin{pmatrix} 1 & a & b & c \\ c & 1 & d & 0 \\ e & f & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Instance Transformations : If an object is created and described in coordinates with respect to its own object coordinate space, we can place an instance or copy of it within a larger scene that is described in an independent coordinate space by the use of three-dimensional coordinate transformations. In this case, the transformations are referred to as *instance transformations*.

Q.3. (*b*) 3-D projection

Ans. Projection an be defined as a mapping of point $P(x, y, z)$ onto its image $P'(x', y', z')$ in the *projection plane* or *view plane*, which constitutes the display surface (fig. 1). The mapping is determined by a projection line called the *projector* that passes through P and intersects the view plane. The intersection point is P'.

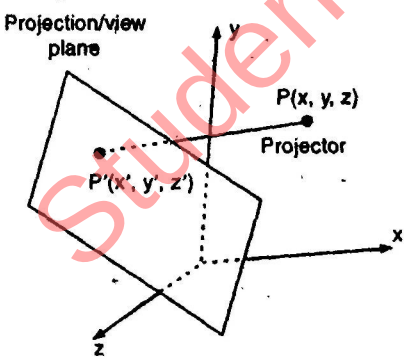


Fig. 1. The problem of projection

The projection preserves lines. That is the line joining the projected images of the end points of the original line is the some as the projection of that line. The two basic projection method.

- Perspective
- Parallel

A perspective transformation is determined by prescribing a center of projection and a view plane. The view plane is determined by its *view reference point* R_0 and *view plane normal* N. The *object point* P is located in world coordinates at (x, y, z). The problem is to determine the image point coordinates $P'(x', y', z')$ (see Fig. 2).

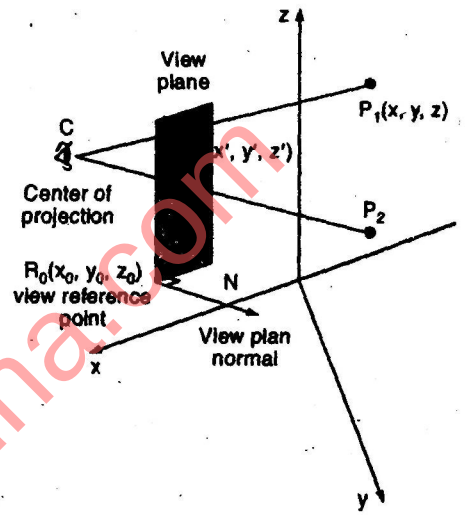


Fig. 2.

Parallel Projection: A *parallel projective transformation* is determined by prescribing a direction of projection vector V and a view plane. The view plane is specified by its view plane reference point R_0, and view plane normal N. The object point P is located at (x, y, z) in world coordinates. The problem is to determine the image point coordinates $P'(x', y', z')$. See fig. 3.

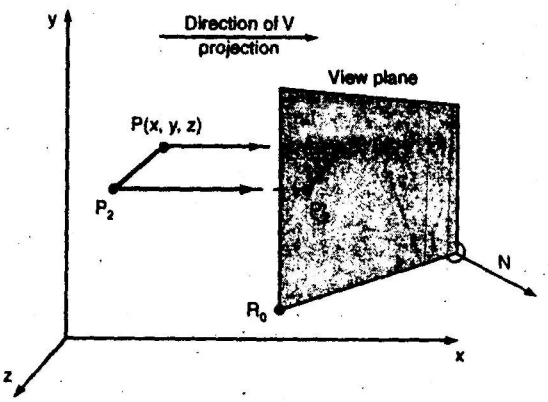


Fig. 3.

If the projection vector V has the direction of the view plane normal N, the projection is said to be *orthographic*. Otherwise it is called *oblique* (see fig. 4)

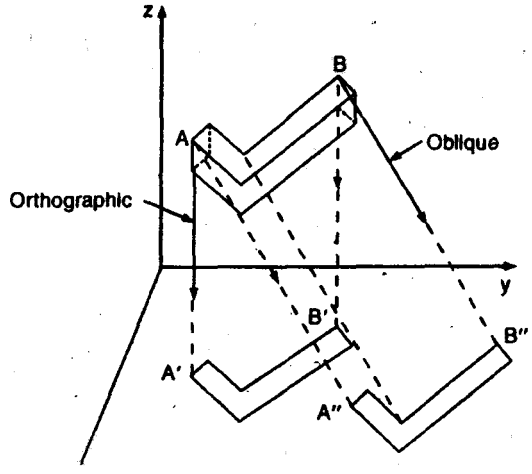


Fig. 4.

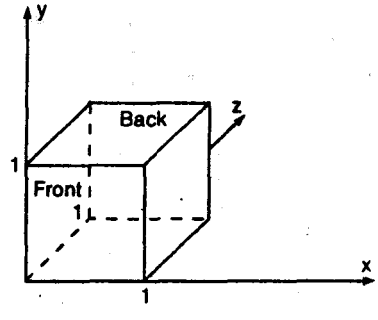
Q.3. (c) 3-D clipping.

Ans. 3-D clipping is used to perform two basic strategies :

1. Direct clipping : In this method, as the suggests, clipping is done directly against the view volume.

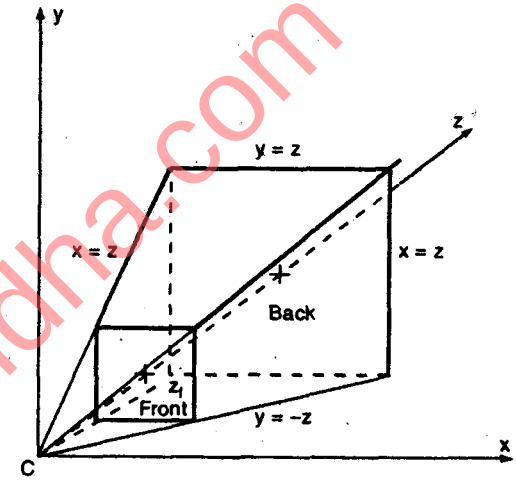
2. Canonical clipping : In this method, normalizing transformations are applied which transform the original view volume into a so-called canonical view volume. Clipping is then performed against the canonical view volume.

The canonical view volume for parallel projections is the unit cube whose faces are defined by the pales $x = 0$, $x = 1$, $y = 0$, $y = 1$, $z = 0$, and $z = 1$. The corresponding normalization transformation N_{par} is constructed in fig.



The canonical view volume for perspective projections is the truncated pyramid whose races are defined by the planes $x = z$, $x = -x$, $y = z$, $y = -z$, $z = z_f$, and $z = 1$ (where z_f is to be calculated).

The basis of the canonical clipping strategy is the fact that the computations involved such operations as finding the intersections of a line segment with the planes forming the faces of the canonical view volume are minimal. This is balanced by the overhead involved in transforming points, many of which will be subsequently clipped.



Clipping Algorithms: Three-dimensional clipping algorithms are often direct adaptations of their two-dimensional counterparts. The modifications necessary arise from the fact that we are now clipping against the six faces of the view volume, which are planes, as opposed to the four edges of the two-dimensional window, which are lines.

The technical differences involve.

1. Finding the intersection of a line and a plane.
2. Assigning region codes to the endpoints of line segments for the Cohen-Sutherland algorithm.
3. Deciding when a point is to the right (also said to be *outside*) or to the left (*inside*) of a plane for the Sutherland-Hodgman algorithm.
4. Determining the inequalities for points inside the view volume.

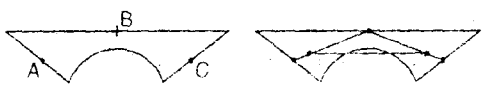
4. Answer any two parts: **(7×2=14)**

Q.4. (*a*) Write an algorithm to draw Bezier curves.

Ans. The de Casteljau algorithm is useful for splitting a Bezier curve into two segments at a given value for the t curve parameter.

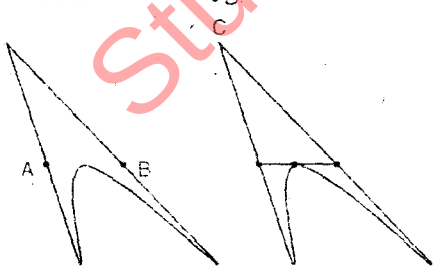
The applet below shows the procedure for the case of **cubic** Bezier curves:

1. At the first step, the A division point is the first control point for the first Bezier segment, while the C division point is the second control point for the second Bezier segment.
2. At the second step, the M point is the second control point for the first Bezier segment, while the N point is the first control point for the second Bezier segment.
3. The point resulted of the third step is the end anchor point for first Bezier segment **and** the start anchor point for the second Bezier segment (the splitting point).



A similar approach can be followed to subdivide a **quadratic** Bezier curve:

1. At the first step, the A division point is the control point for the first Bezier segment, while the B division point is the control point for the second Bezier segment.
2. At the second step, the *P* point is the end of the first Bezier segment and the start of the second Bezier segment.



Q.4. (*b*) What are the various back face detection algorithms? Explain any one of them.

Ans. Back-Face Detection: In a solid object, there are surfaces which are facing the viewer (front faces) and there are surfaces which are opposite to the viewer (back faces).

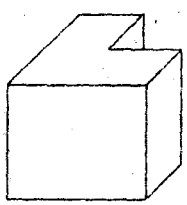
These back faces contribute to approximately half of the total number of surfaces. Since we cannot see these surfaces anyway, to save processing time, we can remove them before the clipping process with a simple test.

Each surface has a normal vector. If this vector is pointing in the direction of the center of projection, it is a front face and can be seen by the viewer. If it is pointing away from the center of projection, it is a back face and cannot be seen by the viewer.

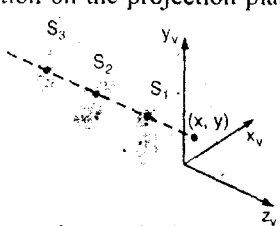
The test is very simple, if the z component of the normal vector is positive, then, it is a back face. If the z component of the vector is negative, it is a front face.

Note that this technique only caters well for nonoverlapping convex polyhedra.

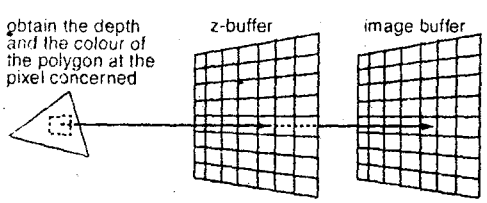
For other cases where there are concave polyhedra or overlapping objects, we still need to apply other methods to further determine where the obscured faces are partially or completely hidden by other objects (eg. Using Depth-Buffer Method or Depth-sort Method).



Depth-Buffer Method (Z-Buffer Method): This approach compare surface depths at each pixel position on the projection plane.



At view-plane position (x, y), surface S1 has the smallest depth from the view plane and so is visible at that position.



Object depth is usually measured from the view plane along the z axis of a viewing system.

This method requires 2 buffers: one is the image buffer and the other is called the z-buffer (or the depth buffer). Each of these buffers has the same resolution as the image to be captured.

As surfaces are processed, the image buffer is used to store the color values of each pixel position and the z-buffer is used to store the depth values for each (x, y) position.

Algorithm:

1. Initially each pixel of the z-buffer is set to the maximum depth value (the depth of the back clipping plane).
2. The image buffer is set to the background color.
3. Surfaces are rendered one at a time.
4. For the first surface, the depth value of each pixel is calculated.
5. If this depth value is smaller than the corresponding depth value in the z-buffer (i.e. it is closer to the view point), both the depth value in the z-buffer and the color value in the image buffer are replaced by the depth value and the color value of this surface calculated at the pixel position.
6. Repeat step 4 and 5 for the remaining surfaces.
7. After all the surfaces have been processed, each pixel of the image buffer represents the color of a visible surface at that pixel.

Q.4. (*c*) Explain any one of them.

(*i*) Illumination models

(*ii*) Text clipping

Ans. (*i*) Humiliation models : The Illumination models give good result and have following components

Ambient reflection

Diffuse reflection

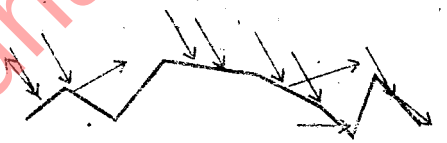
Specular reflection

Ambient reflection is used as a crude approximation to global effects of light being scattered in all directions from all surfaces which gives the effect of an almost even spread of brightness. To do this in computer graphics, we simply assign a constant amount of light to every point in the scene.

$$I_{out}\ k_{ambient}\cdot I_{ambient}$$

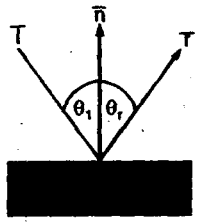
Where I is light intensity (power per unit area) or illumination, I_{out} is the light coming out from a single point on an object, $k_{ambient}$ is the ambient reflectance factor (which the user specifies when rendering graphics) which is the proportion of incoming light reflected ambiently, and $I_{ambient}$ is the intensity of light directly emitted from the light source.

Diffuse reflection models light interactions with different material properties of surfaces. It takes into account the direction and angle that the light source is relevant to the surface point, and also the roughness or smoothness of the surface. All surfaces have some roughness at the microscopic level. Here is a close up model of a surface.



The surface is really broken up into small pieces, all facing different directions. If we assume that light reflects off of each piece like a perfect mirror, then some of the reflected light will be obstructed by other pieces causing shadows. Also, since the pieces of the surface face different directions, rays from a single light source in one direction will reflect in different directions. In an ideal situation, we can think of light coming from a single source being scattered or reflected equally in all directions from a surface.

Specular reflection tries to model the reflections of shiny surfaces more accurately, taking into account not only the light source, but the viewer direction from the surface as well. On an ideal mirror, the light from some source will be perfectly reflected at an angle equal to the incident angle about the normal. However, as we said earlier, surfaces are not smooth at the microscopic level, so not all light will be reflected perfectly even for smooth surfaces. Empirical observations of this suggest that off of smooth surfaces *most* light will be reflected in the direction of a perfectly reflected ray, but with slight variations around the perfect ray.



(*ii*) Text clipping Solution: There are several techniques that can be used to provide text clipping in a graphics package. The clipping technique used will depend on the methods used to generate characters and the requirements of a particular application.

The simplest method for processing character strings relative to a window boundary is to use the **all-or-none string-clipping** strategy. If all of the string is inside a clip window, we k eep it. Otherwise, the string is discarded.

This procedure is implemented by considering a bounding rectangle around the text pattern. The boundary positions of the rectangle are then compared to the window boundaries, and the string is rejected if there is any overlap.

This method produces the fastest text clipping.

An alternative to rejecting an entire character string that overlaps a window boundary is to use the all-or-none **character-clipping** strategy. Here we discard only those characters that are not completely inside the window. In this case, the boundary limits of individual characters are compared to the window.

Any character that either overlaps or is outside a window boundary is clipped. A final method for handling text clipping is to clip the components of individual characters. We now treat characters in much the same way that we treated lines. If an individual character overlaps a clip window boundary, we clip off the parts of the character that are outside the window. Outline character fonts formed with line segments can be processed in this way using a line clipping algorithm. Characters defined with bit maps would be clipped by comparing the relative position of the individual pixels in the character grid patterns to the clipping boundaries.

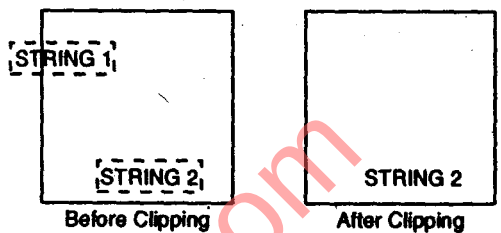


Fig: Text clipping using Bounding Rectangle about the entire text

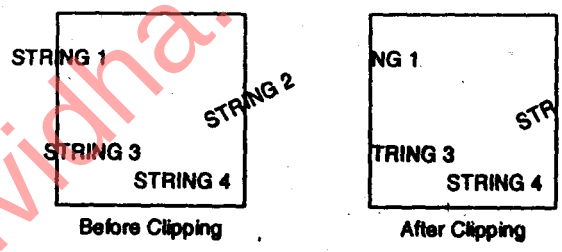


Fig. Text clipping using bounding rectangle about individual characters

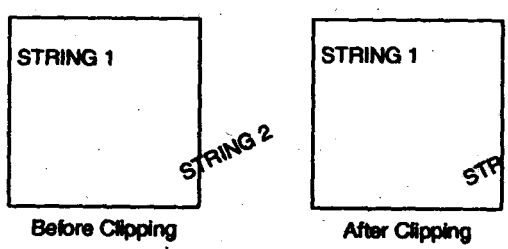


Fig. Text clipping performed on individual components of individual characters.