B.TECH. TCS-501

FIFTH SEMESTER THEORY EXAMINATION 2009-10

COMPUTER GRAPHICS

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

Note: *(i) Attempt all questions.*
(ii) All questions carry equal marks.
(iii) The figures on right indicate marks.

1. Attempt any four of the following: (5 × 4 = 20)

(a) Write a general function for Rasterization.

Ans. General function for Rasterization are defined as follows:

- Screen is traversed 60 times/sec
- Between TV's are basically oscilloscope
- Smooth motion on dynamic screens
- High resolution on static screen
- Optimize bandwidth
- Can use interlaced display (even/odd lines on alternate scans)

(b) Explain the role of pixel and frame buffer in graphics devices.

Ans. In digital imaging, a pixel (or picture element) is a single point in a raster image. The pixel is the smallest addressable screen element, it is the smallest unit of picture which can be controlled, Each pinel has sits address. The address of a pixel corresponds to its coordinate. Pixels are normally arranged in a 2-dimensional grid, and are often represented using dots or squares. Each pixel is a sample of an original image, where more samples typically provide more-accurate representations of the original. The intensity of each pixel is variable. In color image systems, a color is typically represented by three or four component intensities such as red, green and blue, or cyan, magenta, yellow and black.

A **framebuffer** is a video output device that drives a video display from a memory buffer containing a complete *frame* of data. The information in the memory buffer typically consists of color values for every *pixel* (point that can be displayed) on the screen. Color values are commonly stored in 1-bit *monochrome*, 4-bit *palettized*, 8-bit *palettized*, 16-bit *highcolor* and 24-bit *truecolor* formats. An additional *alpha channel* is sometimes used to retain information, about pixel transparency. The total amount of the memory required to drive the framebuffer depends on the *resolution* of the output signal, and on the *color depth* and palette size.

(c) How much time is spent scanning across each row of pixels during screen refresh on a raster system with resolution of 1280 × 1024 and a refresh rate of 60 frames per second?

Ans. Resolution = 1280* 1024

Refresh rate = 60 frames/sec

Time taken by 60 frames to refresh = 1 sec

Time taken by 1 frames to refresh = 1/60 sec

1 frame = 1024 rows

1024 (rows) take time to refresh = 1/60 sec

1 row take time to refresh = 1/60* 1/1024

= 0.058 sec

= 58 m sec

(d) Consider two raster systems with resolutions of 640 × 480 and 1280 × 1024. How many pixels could be accessed per second in each of these systems by a display controller that refreshes the screen at a rate of 60 frames per second?

Ans. Refresh rate = 60 frames/sec

Resolution = 640*480

No. of pixels in one frame = 640*480

= 307200

Controller can access 60 frames/sec

Total no. of pixels accessed = 60*307200

= 18432000 pixels/sec

Resolution = 1280* 1024

Total no. of pixels accessed = 60 *1280* 1024

= 78643200 pixels/sec

(e) Explain how to display file structure and control test.

Ans. Display File Structure: Each display file command contains two parts, an *operation code (opcode),* which indicates what kind of command it is (*e.g.*, LINE or MOVE), and *operands,* which are the coordinates of a point (x, y). The display file is made up of a series of these instructions. One possible method storing these instructions is to use three separate arrays: one for the operation code (*DF-OP*), one for the x coordinate (*DF-X*), and one of the y coordinate (*DF-Y*). To piece together the seventh display file instruction, we would get the seventh element from each of these three arrays *DF OP*[7], *DF-X*[7] and *DF-Y*[7].The display file must be large enough to hold all the commands needed to create our image.

We need to consider only absolute MOVE and LINE commands, since relative commands can be converted to absolute commands before they are entered into the display file. Let us define an opcode of 1 to mean and MOVE command a opcode of 2 to mean a LINE command. A command to move to position $x = 0.3$ and $y = 0.7$ would look like 1, 0.3, 0.7. The statements

DF-OP[3] ← 1;

DF-*X*[3] ← 0.3;

DF-*Y*[3] ← 0.7;

would store this instruction in the third display file position. If DF-OP [4]had the value 2, DF-X[4] had the value 0.5, and DF-Y[4] had the value 0.8, then the display would show a line segment from (0.3, 0.7) to (0.5, 0.8) when display-file instruction 3 and 4 were interpreted.

(f) Compare the computation done in Digital Differential Analyzer (DDA) algorithm with Bresenham's line drawing algorithm.

Ans. DDA algorithm involves floating point calculation so, algorithm is slow. The line drawn by DDA sometimes lies wholely on one side of actual line. The line drawn by DDA algo sometimes doesn't even pass through the 2 given points.

Bresenham Algorithm involves integer point calculation that need to be done in pixel. The accumulation of round of error is successive addition of the floating point increments is used to find the pixel position but it take lot of time to compute the pixel position.

DDA uses float numbers and uses operators such as division and multiplication in its calculation. Bresenhams algorithm uses ints and only uses addition and subtraction. Due to the use of only addition subtraction and bit shifting (multiplication and division use more resources and processor power) bresenhams algorithm is faster than DDA in producing the line.

One note concerning efficiency: Fixed point DDA algorithms are generally superior to Bresenhams that algorithms on modern computers. The reason is Bresenhams algorithm uses a conditional branch in the loop, and this results in frequent branch mispredictions in the CPU. Fixed point DDA also has fewer instructions in the loop body (one bit shift, one increment and one addition to be exact. In addition to the loop instructions and the actual plotting). As CPU pipelines become deeper, mispredictions penalties will become more severe.

Since DDA uses rounding off of the pixel position obtained by multiplication or division, causes an accumulation of error in the proceeding pixels whereas in Bresenhams line algorithm the new pixel is calculated with a small unit change in one direction and checking of nearest pixel with the decision variable satisfying the line equation.

BUT this error can be calculated and will not cause problems for typical line drawing applications. Lines longer than what fits on a typical computer screen (a few thousand pixels) will be identical to Bresenham lines when using 32 bit integers. Fixed point DDA also has another advantage: Since it does not require conditional jumps, you can compute several lines in parallel with SIMD (Single Instruction Multiple Data) techniques.

2. Answer any four questions. **(5 × 4 = 20)**

(a) What is a segment and segment table? Write the utility of segment.

Ans. Segment: The display file divided into segments corresponding to a component of overall display. In other words to manage large pictures we use segment, because it is very difficult to handle a very large file. For efficient handling of large pictures, we use segment. Every segment has its own attributes *e.g.* visibility, its size, start position, image transformation etc.

Segment Table: It is basically used to keep the information of the image. Each row in the segment table represents information of one segment including name, position, size, attributes and the image information parameters. If we wish to make segment any of the segment visible then this is activated by setting the corresponding entry in the array to ON. The display file interpreter initially checks the start, size and visible attribute of the segments and it interprets only those segment which are to made visible.

Approaches to Construct the Segment Table:

1. Using Array
2. Using Link List
3. Using Paging Scheme

Using Array: To access any segment of the array we need to assign the name of the array. In this segment 0 is used for unnamed segment.

Link List: The link list uses the additional field called the link or pointer which gives the location of the segment in the segment table. In case of array, maximum number of segments that can be included in table are equal to the length of the array but in link list there is no limit for size.

Seg.start	Seg.size	Scale x	Scale y	Color	Visibility	Link

(*b*) Write a procedure for creating, delEting and renaming segments.

Ans. Creating a Segment: With the help of following procedure, we can create a segment.

Algorithm create segment (segment name)

1. Input all the arrays that make up the segment table.
2. Check (a) any segment is open
   (b) segment name is not valid
   (c) Existence of segment being created if so, return respective error.
3. Otherwise initialize all the arrays in the segment table under this segment name.
4. Mention this segment is open and return.

Deleting a Segment: The function used for deleting a segment n is Delete Segment (n). This function will close any currently open segment. If segment n does not exist then there is no further effect.

Old Seg 1	New Seg 1	Old Seg 2	New Seg 2	Old Seg 3

Repetitive deletions of different segments may result in memory management problems as it will create small blocks of free memory distributed at different locations. This may create fractured segments as shown in the above figure.

Algorithm Delete Segment (Segment name)

1. Input all the require variables.
2. Check
   (a) segment is open
   (b) segment name is not valid
3. If segment size is zero *i.e.*, have no any instruction then return otherwise delete and shift the display file instruction.
4. Recover the deleted storage.
5. Update the table and return.

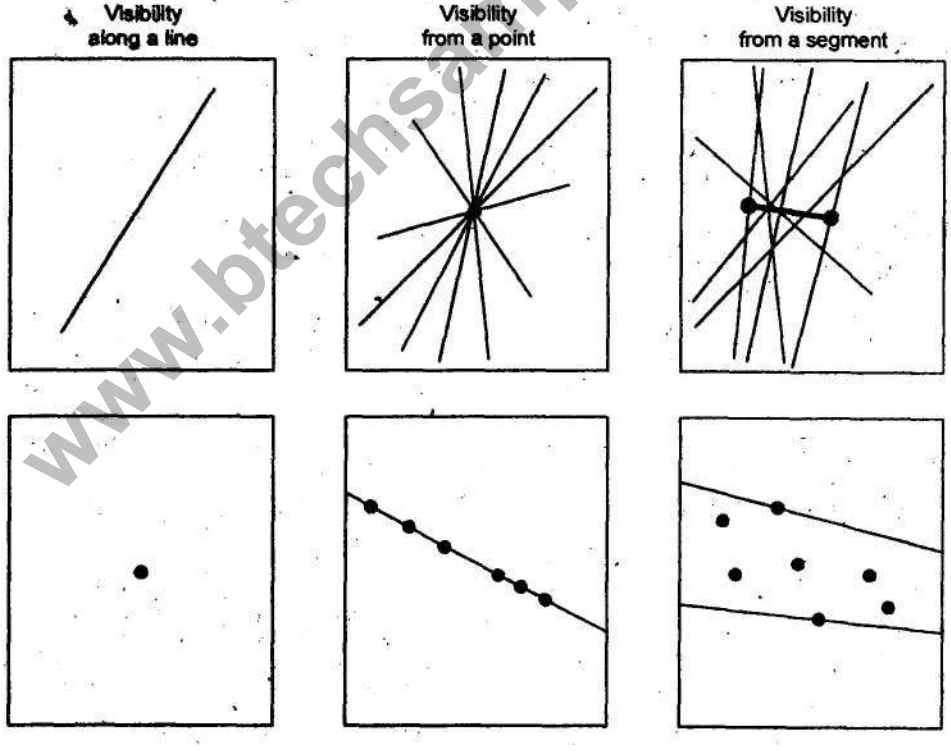
Renaming a Segment: With the help of following procedures we can rename the segment.

Algorithm Rename Segment (Segment name)

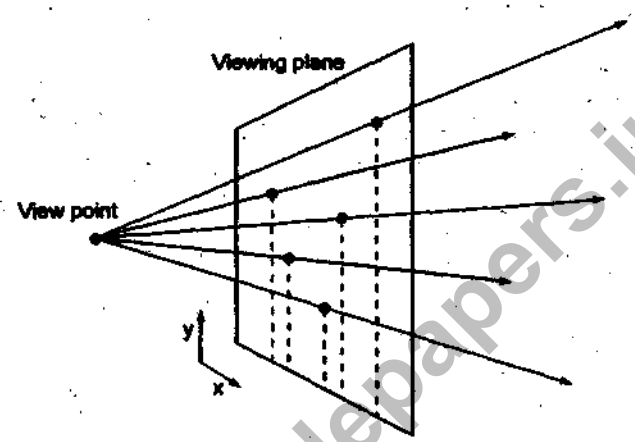
1. Input all the required variables *e.g.* old segment name, new segment name, the segment table arrays, the size of the segment table etc.
2. Check
   (a) Validity of both the segments
   (b) Segments not open
   (c) New segment doesn't already exist if these conditions are not met return respective error.
3. Copy the old segment entry into new segment position.
4. Delete old segment and return.

(c) Explain visibility in detail.

Ans. Visibility: The simplest visibility problems deal with visibility along a signal line. The problem relevant line set is zero-dimensional, i.e., it is fully specified by the given line. The visibility along a line problems can be solved by computing intersections of the given line with the scene objects.

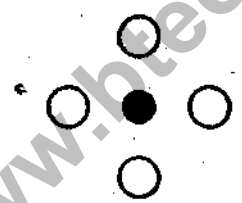


A scene object. A similar problem to ray shooting is point-to-point visibility. Point-to point visibility determines whether the line segment between two points is unclouded, *i.e.*, the line segment has no intersection with an opaque object in the scene. Another visibility along a line problem is determining the *maximal free line segments* on a given line. See Figure 2 for an illustration of typical visibility along a line problems.



(*d*) Write a boundary fill procedure to fill an 8-connected region.

Ans. Start at a point inside the figure and paint with a particular color. Filling continues until a boundary color is encountered.

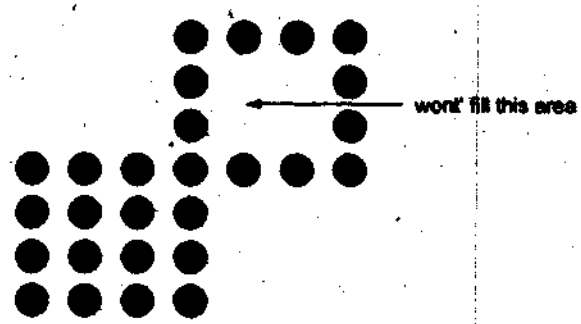


There are two ways to do this:

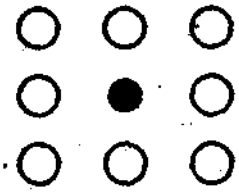
1. **Four-connected fill where we propagate: left right up down**

```
Procedure Four_underscore Fill (x, y, fill_col,
bound_col : integer);
var
curr_colour : integer;
begin
    curr_color : = inquire_color (x, y)
    if (curr_color < > bound color) and (curr_color
    < > fill _col) then
begin
    set_pixel (x, y, fill_col)
    Four_Fill (x + 1, y, fill_col, bound_col);
    Four_Fill (x – 1, y, fill_col, bound_col);
    Four_Fill (x, y + 1, fill_col, bound_col);
    Four_Fill (x, y – 1, fill_col, bound_col);
    end;
```

There is the following problem with four–fill:



This leads to the 2. Eight-connected fill algorithm where we test all eight adjacent pixels.



So we add the calls:

eight_fill ($x + 1, y - 1$, ...)

eight_fill ($x + 1, y + 1$, ...)

eight_fill ($x - 1, y - 1$, ...)

eight_fill ($x - 1, y + 1$, ...)

Note: Above 4-fill and 8-fill algorithms involve heavy duty recursion which may consume memory and time. Better algorithms are faster, but more complex. They make use of pixel runs (horizontal groups of pixels).

(e) Explain scan fill algorithm with the help of suitable example.

Ans. Scan Line Conversion for Polygon Filling:

1. Read n, the number of vertices of polygon.
2. Read x and y coordinates of all vertices in array $x[n]$ and $y[n]$.
3. Find y_{min} and y_{max}.
4. Store the initial x value (x_1), y values y_1 and y_2 for two end points and x increment Δx from scan line to scan line for each edge in the array edges $[n][4]$.
5. Sort the rows of array, edges $[n][4]$ in descending order of y_1, descending order of y_2 and ascending order of x_2.
6. Set $y = y_{max}$.
7. Find the active edges and update active edge list:

   if ($y > y_2$ and $y <= y_1$)

   {edge is active}

   else {edge is not active}
8. Compute the x intersects for all active edges for current y value [initially x-intersect is x_1 and x intersects for successive y values can be given as]

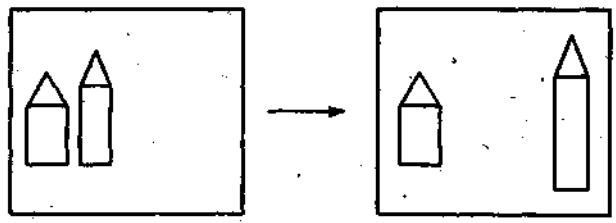
   $xi + 1 \leftarrow xi + \Delta x$

   where $\Delta x = -1/m$ and $m = y_2 - y_1/x_2 - x_1$ *i.e.*, slope of a line segment.
9. If x-intersect is vertex *i.e.*, x-intersect = x_1 and $y = y_1$ then apply vertex test to check whether to consider one intersect or two intersect. Store all x intersects in the x intersects array.
10. Sort x-intersect [] array in the ascending order.
11. Extract pairs of intersects from the sorted x-intersects [] array.
12. Pass pairs of x-value to the line drawing routine to draw corresponding line segments.
13. Set $y = y - 1$
14. Repeat steps 7 through 13 units $y >= y_{min}$.
15. Stop.

(f) Explain different procedures for image transformation.

Ans. Image transformations are the mechanism used to compose an image from modeling primitives. The modeling primitives are defined in their own modeling coordinate system and then placed in the final scene by using modeling transformations. We can change the Camera position or World Coordinate Window to scale or move an entire image, but we may want to only change a particular part of the image.

Example: We might want to translate and scale the tall house but not change the short house. We can't do this by modifying the window, so we want to apply transformations to the individual objects in the scene.

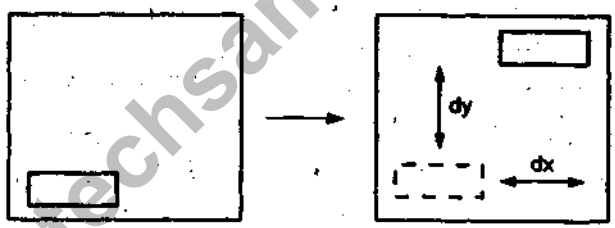


Possible modeling transformations include the following:

- change size of object : Scaling
- move object: Translation
- Rotate object: Rotation

2D Translation

Translation is a simple straight line movement of the object.



(old coordinates are (x, y) and the new coordinates are (x', y'))

$$x' = x + Tx$$

$$y' = y + Ty$$

For a complex object such as a polygon, add the translation distances (Tx, Ty) for each endpoint of the object. For a circle or ellipse:

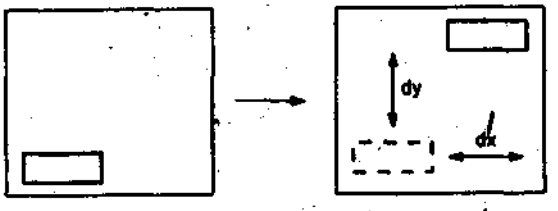
$$xc' = xc + Tx$$

$$yc' = yc + Ty$$

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

(a) Write a procedure for rotation and translation transformation. Derive reflection metrics for reflection about the X axis.

Ans. 2D Translation: Translation is a simple straight line movement of the object.



(old coordinates are (x, y) and the new coordinates are (x', y'))

$$x' = x + Tx$$

$$y' = y + Ty$$

For a complex object such as a polygon, add the translation distances (Tx, Ty) for each endpoint of the object. For a circle or ellipse:

$$xc' = xc + Tx$$

$$yc' = yc + Ty$$

2-D Rotation

$$x' = r\cos(A + B)$$

$$= r\cos A\cos B - r\sin A\sin B$$

$$y' = r\sin(A + B)$$

$$= r\cos A\sin B + r\sin A\cos B$$

$$x = r\cos A, y = r\sin A$$

So,

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

$$y' = x\sin B + y\cos B$$

$$P = R.P$$

$$\text{Rotation Matrix } R = \begin{pmatrix} \cos B & -\sin B \\ \sin B & \cos B \end{pmatrix}$$

Reflection Matrix: Reflection is a transformation that produces a mirror image of an object. The mirror image for a 2D reflections generated relative to an axis of reflection by rotating the object 180 about the reflection axis.

Reflection about x-axis is accomplished with the transformation matrix:

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

(b) Write and compare Cohen-Sutherland line clipping, Liang-Barsky line clipping and Nicholl-Lee-Nicholl line clipping.

Ans. Any procedure that identifies those portions of a picture that are either inside or outside of a specified region of space is referred to as a **clipping algorithm**, or simply clipping. The region against which an object is to clipped is called a **clip window**.

Applications of clipping include extracting part of a defined scene for viewing; identifying visible surfaces in three-dimensional views; antialiasing line segments or object boundaries; creating objects using solid-modeling procedures; displaying a multi window environment; and drawing and painting operations that allow parts of a picture to be selected for copying, moving, erasing or duplicating.

The numbers in the figure above are called outcodes. The bits in the outcode represents: Top, Bottom, Right, Left. Each bit in the code is set to either 1(true) or a 0(false).

```
// Cohen-Sutherland line clipping algorithm
# define LEFT-EDGE 0 × 1
# define RIGHT-EDGE 0 × 2
# define BOTTOM-EDGE 0 × 4
# define TOP-EDGE 0 × 8
/* Points encoded as 0000 are completely inside the clip rectangle; all others are outside at least one edge. If OR'ing two codes is FALSE (no bits are set in either code), the line can be accepted. If the. WD operation between two codes is TRUE, the line defuled by tioseendpoints is completely outside the clip region and can be rejected.
*/
# define INSIDE (a1 (!a)
# define REJECT (a, b) (ahb)
# define ACCEPT (a, b) (! (alb))
unsigned char encode fwcPti pt, dcPt winMln. dcPt winMax)
unsigned char code = O × CO :
if (pt. x win Mln. x)
code = code/LEFT-EDGE;
if (pt.x > winMax.x)
code: code I RIGHT-EKE;
if (pc. y > winMin. yl
code:
code = code I BOTOM-EDGE:
if (pt.y < winMin. y)
code = code 1 TOP-EDGE;
return(code);
)
votd swapPts (wcPt p1, wcPt2 p2)
(
cpt2: tmp;
void swapcodes (unsigned char c1, c2)
tmp = c1; *c1 = c2; *c2 = tmp;
>
void clipLine (wcPt winMin, wcPt winMax, wcpt2 p1, wcpt z1)
(
unsigned char code1, code2;
int done = FALSE, draw = FALSE:
float m;hile (!done) (
code I = encode (p1, widin, winMax);
code 2 = encode (p2, winin, winMax);
if (ACCEPT (codel, code 2) (
done = TRUE;
draw = TRUE
I
else
if (REJECT lcodel, code2))
done: TRUE;
else {
/* Ensure that pl is outside window */
if (INSIDE (codei)) [swapPts (hpl, and p2);
swapcodes (&code:, ccode2);
}
/* Use slope (m) to find line-clipEdge intersections */
if (p2 .x ! = p1.x)
m = (p2.y – p1.y : / (p2.x – p1.x);
if (codel LEFT-EDGE) (
p1.y + = (winl4in.x – p1x) m;
p1.x = v8inMin.x;
1
else
if (codel RIGHT-EDGE) (
p1.y 4: (winMax.x – p1.x) m;
```

```
P1.x = winG4ax.x;
)
else
if (codel BOTTOMKEDCE) L
/* Need to updace p1.x for non-vertical lines only */
else
if (codel h TOP-EDGE) {
if (p2.x !– p1.x)
p1.x + =; winMax. y – p1.y) 1 n,
p1.y = winf4ax .y;
1
)
1
If (draw)
line DDA(p1.x, p1y, p2.x, p2.y);
}
```

(c) **What is window-to-view point coordinate transformation? What are issues related to multiple windowing?**

Ans.

1. In this 2 things are done. We are changing the window size to become the size of the viewport and we are positioning it to desired location on the screen.
2. The positioning is just moving the lower left corner of the window to the viewport's lower-left corner location.
3. The overall transformation which performs this is viewing transformation. It creates a particular view of the object.

e.g., If window has left and right boundaries of 3 and 5 and lower and upper boundaries of 0 and 4, then the 1st translation matrix could be

$$T_{xy} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -3 & 0 & 1 \end{pmatrix}$$

Suppose, the viewport is upper-right quadrant of the screen with boundaries at 0.5 and 1.0 for both x and y directions.

The length of window is $5 - 3 = 2$ in the x-direction.

The length of viewport is $1.0 - 0.5 = 0.5$

So x-scale factor is $S_x = 0.5/2 = 0.25$

In y-direction $S_y = 0.5/4 = 0.125$

$$Ss_x, s_y = \text{Scaling matrix} = \begin{pmatrix} 0.25 & 0 & 0 \\ 0 & 0.25 & 0 \end{pmatrix}$$

Finally, to position the viewport requires a translation.

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0.5 & 0.5 & 1 \end{pmatrix}$$

Viewing transformation is

$$= \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -3 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0.25 & 0 & 0 \\ 0 & 0.125 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0.5 & 0.5 & 1 \end{pmatrix} = \begin{pmatrix} 0.25 & 0 & 0 \\ 0 & 0.125 & 0 \\ -0.25 & 0.5 & 1 \end{pmatrix}$$

Multiple windowing

1. A few image is created by one or more window transformations on the object.
2. Then, windows are applied to this first image to create a second image.
3. Further, windowing transformation may be done until the desired picture is created.
4. Every application of a window transformation allows the user to slice up a portion of the picture and reposition it on the screen
5. Thus the multiple windowing gives the user freedom to rearrange components of the picture.
6. The same effect may be achieved, however, by applying a number of single window.

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

(a) Explain parallel projection, perspective projection and depth curing projection for 3-D display methods.

Ans. The perspective projection requires greater definition. A conceptual aid to understanding the mechanics of this projection involves treating the 2D projection as being viewed through a camera view finder. The camera's position, orientation, and field of view control, the behaviour of the projection transformation. The following variables are defined to describe this transformation:

- ${}^{a}x$-y-z-the point in 3D space that is to be projected.
- ${}^{c}x$-y-z-the location of the camera.
- ${}^{0}x$-y-z-The rotation of the camera. When ${}^{c}x$-y-z = < 0, 0, 0 >, and ${}^{0}x$-y-z = < 0, 0, 0 >, the 3D vector <1, 2, 0 > is projected to the 2D vector <1, 2>.
- ${}^{e}x$-y-z the viewer's position relative to the display surface.

Which results in :

- ${}^{b}x, y$-the 2D projection of a.

First, we define a point $d_{x\text{-}y\text{-}z}$ as a translation of point a into a coordinate system defined by c.

This is achieved by subtracting c from a and then applying a vector rotation matrix using –0 to the result. This transformation is often called a camera transform (note that these calculations assume a left-handed system of axes):

$$\begin{bmatrix} d_x \\ d_y \\ d_z \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos-\theta_x & \sin-\theta_x \\ 0 & -\sin-\theta_x & \cos-\theta_x \end{bmatrix}$$

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

Or, for those less comfortable with matrix multiplication, Signs of angles are inconsistent with matrix form:

$$d_x = \cos\theta_y \cdot (\sin\theta_z \cdot (a_y - c_y) + \cos\theta_z \cdot (a_x - c_x)) - \sin\theta_y \cdot (a_z - c_z)$$

$$d_y = \sin\theta_x \cdot (\cos\theta_y \cdot (a_z - c_z) + \sin\theta_z \cdot (a_y - c_y) + \cos\theta_z \cdot (a_x - c_x)$$

$$d'_z = \cos\theta_x \cdot (\cos\theta_y \cdot (a_z - c_z) + \sin\theta_y \cdot (\sin\theta_2 \cdot (a_y - c_y) + \cos\theta_z \cdot (a_x - c_x)$$

This transformed point can then be projected onto the 2D plane using the formula (here, *x/y* is used as the projection plane, literature also may use *x/z*)

$$b_x = (d_x - e_x)(e_z/d_z)$$

$$b_y = (d_y - e_y)(e_z/d_z)$$

Or, in matrix form using homogeneous coordinates:

$$\begin{bmatrix} f_x \\ f_y \\ f_z \\ f_w \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & -e_x \\ 0 & 1 & 0 & -e_y \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/e_z & 0 \end{bmatrix} \begin{bmatrix} d_x \\ d_y \\ d_z \\ 1 \end{bmatrix}$$

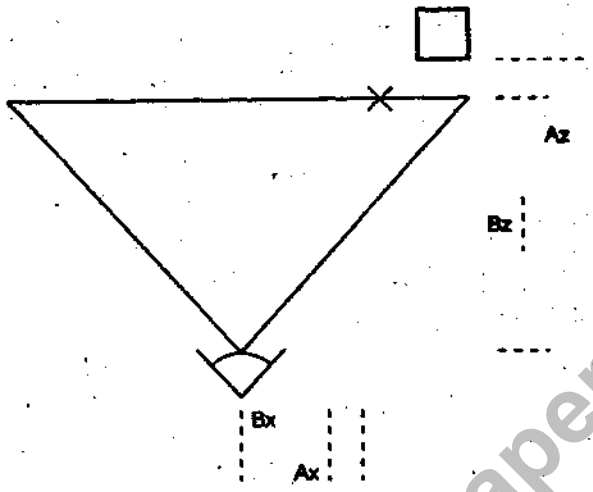
and

$$b_x = f_x/f_w$$

$$b_y = f_y/f_w$$

The distance of the viewer from the display surface, ex, directly relates to the field of view, where $a = 2 \cdot \tan^{-1}(1/e_z)$ is the viewed angle. (Note : This assumes that you map the points (–1, –1) and (1, 1) to the corners of your viewing surface)

Subsequent clipping and scaling operations may be necessary to map the 2D plane onto any particular display media.



(b) Explain event handling with the help of example.

Ans. Routine to enable and input device class:

Routine takes a class number, perform the necessary operation to turn the device classes logically ON(enable) and set the corresponding device flag to true.

ENABLE_GROUP (CLASS)

Arguments: Class

Global variables: Keyboard, locator, valuator, Pick, Button

Begin:

if class = 1 then

perform all the operations needed to turn ON Button devices.

Button< - true;

end

if class = 2 then Perform

All the operation needed to turn ON pick devices.

if class = 3 then perform all the operations needed to turn ON keyboard.

if class = 4

exit;

end;

Routine to disable an input device class:-

DISABLE_GROUP (CLASS)

arguments: class

Global variables:

Begin:

if class = 1 then Perform

all the operations needed to turn OFF button devices.

Button < -False.

(c) What is echoing and its use? Write different interactive techniques in computer graphics.

Ans. Interactive Techniques: The interactive devices are the devices which give the input to the graphics system and provide necessary feedback to the user about the accepted input. There are following interactive devices

1. **Keyboard:** The keyboard is a primary input device for any graphics system. It is used for entering text and numbers, *i.e.*, on graphics data associated with pictures such as labels *x-y* coordinates etc.
2. **Mouse:** A mouse is a palm-sized box used to position the screen cursor. It consists of boll on the bottom connected to wheels or rollers to provide the amount and direction of movement. One, two or three buttons are usually included on the top of the mouse for signalling and execution of same portion.
3. **Trackball and Space ball:** The trackball is a two dimensional positioning device whereas space ball provides six degree of freedom. It does not actually move.
4. **Joysticks:** A joystick has a small, vertical lever mounted on the base and used to steer the screen cursor around.
5. **Touch Panels:** It allows displayed objects or screen position to be selected with the touch of finger. The touch panel is transparent devices which are fitted on the screen. They consist of touch sensing mechanism.
6. **Light Pens:** This is an event driven device. After displaying each point, one can test the light pen flip-flop. The exact position of the spot to which the light pen is pointing can be detected.

5. Answer any two questions. **(10 × 2 = 2)**

(a) Make a comparison of Bezier, Hermite and B-spline algorithms for curve generation.

Ans. Bezier Curve:

1. Bezier curve always passes through the first and last control point its i.e., curve has same end points as the guiding polygon.
2. The degree of polynomial defining the curve segment is one less than the number of defining polygon points. Therefore for 4, control points the degree of polynomial is three.
3. The curve generally follows the shape of the defining polygon.
4. The curve lies entirely within the convex hull formed by control points.
5. The curve exhibits the variation diminishing property. This means that the curve does not oscillate about any straight-line move often than the defining polygon.
6. The curve is invariant under an affine transformation.

B-Spline :

1. The degree of B-Spline polynomial is independent of the number of vertices of the defining polygon.
2. The curve generally follows the shape of the defining polygon.
3. The curve lies within the convex hull of its defining polygon.
4. The curve exhibits the variation diminishing property. This means that the curve does not oscillate about any straight-line move often than the defining polygon.
5. The B-Spline allows local control over the curve surface because each vertex affects the shape of the curve only over a range of parameter values where its associated basic function is non-zero.

Hermite-Spline :

It is useful for some digitizing applications where it may not be too difficult to specify or approximate the curve slopes.

It is more useful to generate spline curves without requiring input values for curve slopes or other geometric information.

(b) List the advantages and disadvantages of Back-face detection, Dept-Buffer method and A Buffer method.

Ans. Back-face Detection:

Advantage:

1. Based on inside and outside test. So it is fast and simple.
2. It can eliminate about half of the polygon surface in a scene form surface in a scene from further visibility tests.

Disadvantage:

1. Not used for 3D image.
2. 2D algorithm are not implemented in this
3. Based only on solid objects modeled as polygon mesh
4. Works fine for convex polyhedra

Depth Buffer:

Advantage:

1. It requires no sorting

Disadvantage

1. It cannot be applied to transparent objects.

A-Buffer

Advantage

1. It can be applied for transparent objects.

(c) Explain in detail different illumination methods and different Randering methods.

Ans. Illumination Model

(a) Ambient light

(b) Diffuse Reflection

(c) Specular Reflection and Phong Model

(d) Warn Model

Rendering is converting the object picture into 2-D image adding details of surface properties

(a) Phong Shading

(b) Constant Intensity

(c) Gourand Shading

(d) Fast Phong Shading