Q.2 a. List some of the application areas where computer graphics can be used? (4)

Answer:

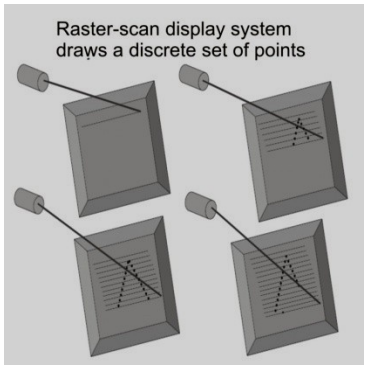
- Art, Entertainment, and Publishing
  - Computer games
  - Movie production, Animation
  - Slide, Book, and Magazine design
- Computer Graphics, Perception, and Image Processing
- Computer-Aided Design
- Displaying Simulations
- Monitoring a Process
- Scientific Analysis and Volume Visualization

(1x4=4)

b. Discuss the major differences between raster scan and random scan display. (4)

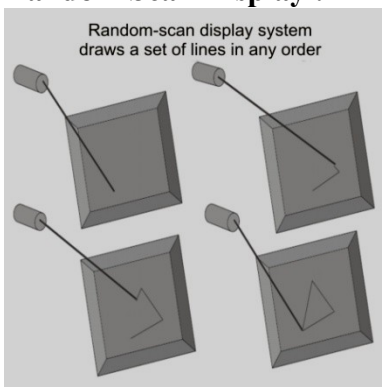
Answer:

Raster Scan display :



It is the most common type of monitor employing a CRT is raster scan display. In Raster scan display the electron beam is swept across the screen one row at a time from top to bottom. Beam intensity is turned on and off to create a pattern of illuminated spots as the electron moves across each row. In a raster scan system entire screen is a matrix of pixels. Frame buffer or refresh buffer holds the set of all intensity values for all screen points. Stored intensity values are then retrieved from the refresh buffer and painted on the screen one row at a time.

Random Scan Display :



The electron beam is directed only to the parts of the screen where a picture is to be drawn. Picture definition is stored as a set of line drawing commands in an area of memory referred to as refresh display file.

To display a specified picture, the system cycles through a set of commands in the display file, drawing each component line. After processing, all lines drawing commands the system cycle back to the first line command in the list.

(2+2 Marks)

c. What are the important components of GUI? Mention some of the general practices which should be maintained to develop an effective GUI. (8)

Answer: *(6 X ½ Marks=3 Marks)*

The following are the essential components of GUI

1. **Graphics Pointer:** It is a mouse cursor or a symbol that appears on the display screen and that you move to select objects, icons or any menu commands within the interface. Usually, the pointer is shaped like a small angled arrow. However, in text processing applications, an I-beam pointer is used, which is shaped like a capital I.
2. **Pointing device:** It is a device, such as a mouse or a trackball, which enables the user to select objects, an item from the menu, an icon on the graphics screen.
3. **Icons:** Small images that represents commands, files or windows. By moving the pointer to the icon and pressing a mouse button, one can execute a command or convert the icon to a window.
4. **Desktop:** It is the area on the display screen where icons are often grouped. It is known as desktop because the icons are intended to represent the real objects on a real workspace.
5. **Windows:** It divides the screen into different areas. In each window, one can run different programs or different components of a program or display a different file. One can move or resize any window around the graphics screen.
6. **Menus:** Most graphical user interfaces allow the execution of commands by making a choice from a menu.

General practices for developing good GUI are as follows: ***(5 Marks)***

- **Consistency at the widget level:** The user must be able to anticipate widget (mouse, tracker ball, or joystick) behaviour from its visual properties. It maintains the consistency with the standard operating system behaviour.
- **Consistency at the platform level:** The user must be able to anticipate the behaviour of the program using knowledge gained from other programs.
- **Good amount of error handling before the program ceases to run:** A good GUI should rarely need or use any warnings and error dialogs. Exceptions include hardware errors such as a disk failure, memory error, or any error related to the minimum system requirements. It is better to maintain error log, which is very helpful in case of program termination.
- **Adequate user feedback:** When any object is selected using the left mouse, it should display a message at the appropriate place that it has been selected.
- **The GUI should promote exploration:** It enables the user to try out any unknown command easily. For instance, the Undo/Redo capabilities eliminate the need for dialogs requesting permission to perform a seemingly erroneous function.
- A good GUI makes the application self-evident. A good package should rarely require any help manuals.
- Sound, colour, animation and multimedia clips should be used as and when required very cautiously.
- Users should be able to customize and save the settings of their preferred workspace environment.
- Avoid manual behaviours. A single software package may contain multiple ways to perform the same thing.
- Transparent interface design should be used. Interface transparency occurs when the user's attention is drawn away from the interface and naturally directed at the task itself.
- The desktop should be ergonomically designed. It should promote multiple skill levels. It should require minimum memorization to perform ant task.

Q.3 a. Define polygon filling. Explain boundary filling for a polygon with suitable example. (8)

Answer:

Filling a polygon means finding coordinates of interior points and assigning values calculated by using one of the incremental shading schemes. This requires construction of an edge list for each polygon. There are two approaches to solid area scan conversion of polygon- scan conversion and seed filling. *(2 Marks)*

Boundary-Fill Algorithm

- This algorithm starts at a point inside a region called seed and paint the interior outward towards the boundary. If the boundary is specified in a single color, the fill algorithm proceeds outward pixel by pixel until the boundary color is encountered. **This** method is particularly useful in interactive painting packages, where interior points ***are*** easily selected. There are two methods for proceeding to neighboring pixels from the current test position. In Fig. 3(a), four neighboring points are tested. These are the pixel positions that are right, left, above, and below the current pixel. Areas **filled** by this method are called 4-connected. The second method, shown in Fig. 3(b), is **used** to fill more complex figures. Here the set of neighboring positions to **be** tested includes the four diagonal pixels. Fill methods using this approach are called 8connected. *(6 Marks)*

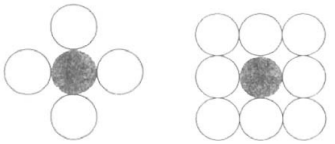
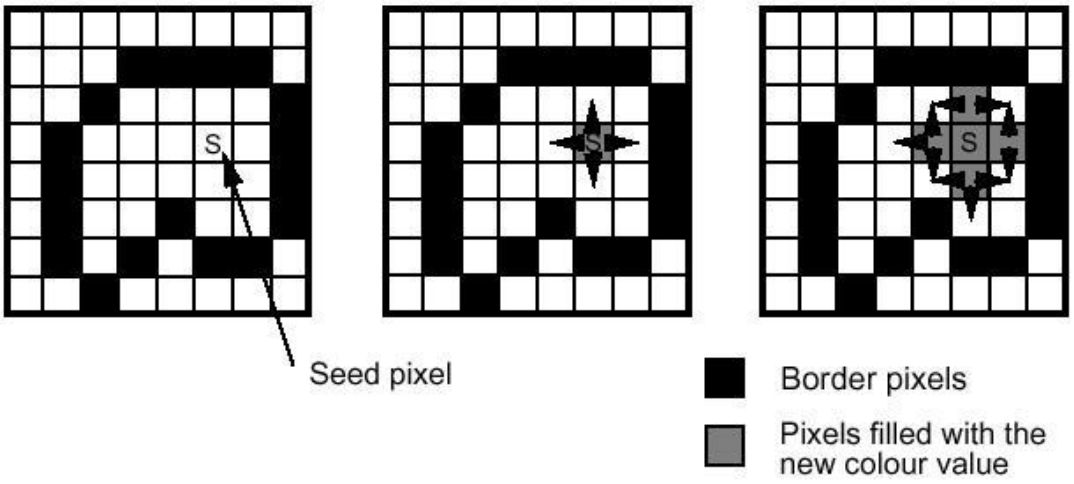


Fig 3(a) fig 3(b)



This is a simple method but not efficient: It is recursive method which may occupy a large stack size in the main memory.

```
void BoundaryFill(int x, int y, COLOR fill, COLOR boundary)
{          COLOR current; current=GetPixel(x,y);
          if (current<>boundary) and (current<>fill) then
          {
           SetPixel(x,y,fill);
```

```
        BoundaryFill(x+1,y,fill,boundary);
        BoundaryFill(x-1,y,fill,boundary);
        BoundaryFill(x,y+1,fill,boundary);
        BoundaryFill(x,y-1,fill,boundary);
}
}
```

b. Digitize a line from with endpoints (20, 10) and (30, 18) using Bresenham's straight line algorithm. Give first five pixel locations. (8)

Answer:

The line has a slope of *0.8,* with

$\Delta x = 10, \quad \Delta y = 8$ *(1 Mark)*

The initial decision parameter has the value

$p_0 = 2\Delta y - \Delta x$

$= 6$ *(1 Mark)*

and the increments for calculating successive decision parameters are

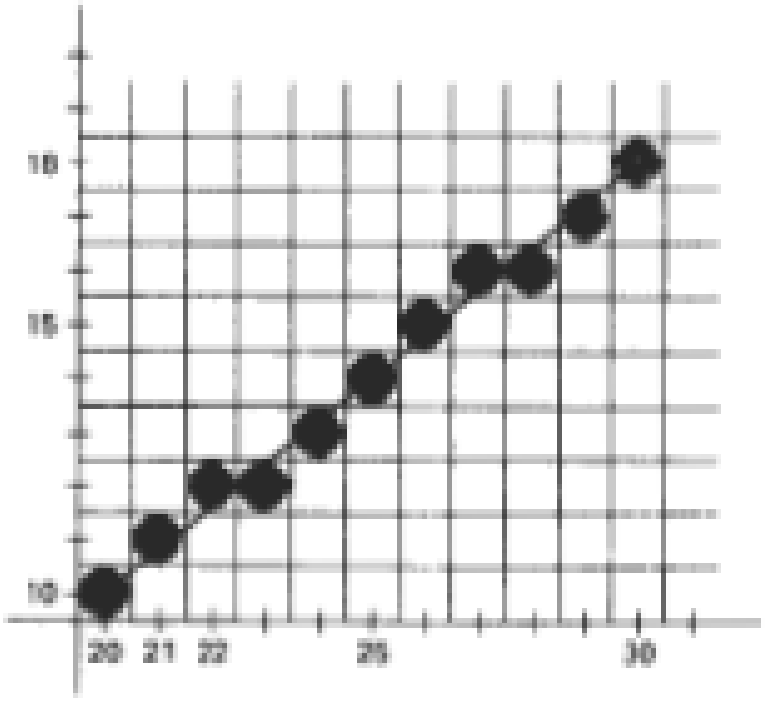
$2\Delta y = 16, \quad 2\Delta y - 2\Delta x = -4$ *(1 Mark)*

We plot the initial point *(xo, yo) = (20, 10),* and 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)

(5 Marks)

A plot of the pixels generated along this line path is shown in Figure below.



Pixel position along the line path between endpoints (20, 10) and (30, 18), plotted with Bresenham's line algorithm.

Q.4 a. What are the basic attributes of a straight line segment? Explain with the help of suitable functions. (8)

Answer: *(4 Marks)*

Basic attributes of a straight line segment are its type, its width, and its color. In some graphics packages, lines can also be displayed **using** selected pen or brush options.

Line Type

Possible selections for the line-type attribute include solid lines, dashed lines, and dotted lines. We modify a line drawing algorithm to generate such lines by setting the length and spacing of displayed solid sections along the line path. A dashed line could be displayed by generating an interdash spacing that is equal to the length of the solid sections. Both the length of the dashes and the interdash spacing are often specified as user options. **A** dotted line can be displayed by generating very short dashes with the spacing equal to or greater than the dash size. Similar methods are used to produce other line-type variations. To set line type attributes in a **PHIGS** application program, a user invokes the function

setLinetype **(lt)**

where parameter lt is assigned a positive integer value of **1,2,3,** or 4 to generate **lines** that are, respectively, solid, dashed, dotted, or dash-dotted. Other values for the **line-type** parameter lt could be used to display variations in the dot-dash patterns. Once the line-type parameter has been **set** in a **PHIGS** application program, all subsequent line-drawing commands produce lines with this Line type.

Raster line algorithms display line-type attributes by plotting pixel spans. For the various dashed, dotted, and dot-dashed pattern, the line-drawing procedure outputs sections of contiguous pixels along the line path, skipping over a number of intervening pixels between the solid spans. Pixel counts for the span length and interspan spacing can be specified in a pixel mask, which is a string containing the digits 1 and 0 to indicate which positions to plot along the line path. The mask 1111000, for instance, could be used to display a dashed line with a dash length of four pixels and an interdash spacing of three pixels. On a bilevel system, the mask gives the bit values that should be loaded into the frame buffer along the line path to display the selected line type.

Line Width

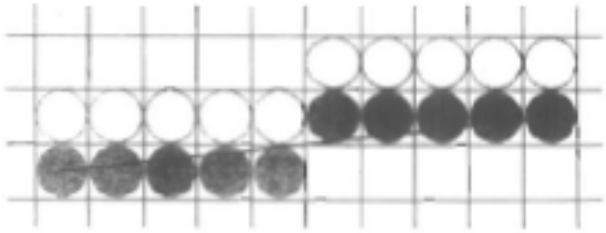
Implementation of line- width options depends on the capabilities of the output device. **A** heavy line on a video monitor could be displayed as adjacent parallel lines, while a pen plotter might require pen changes. **As** with other PHIGS attributes, a line-width command is used to set the current line-width value in the attribute list. This value is then used by line-drawing algorithms to control the thickness of lines generated with subsequent output primitive commands.

We set the line-width attribute with the command:

setLinewidthScaleFactor(lw)

Line-width parameter lw is assigned a positive number to indicate the relative width of the line to be displayed. A value of 1 specifies a standard-width line. On a pen plotter, for instance, a user could set **lw** to a value of 0.5 to plot a line whose width is half that of the standard line. Values greater than 1 produce lines thicker than the standard.

For raster implementation, a standard-width line is generated with single pixels at each sample position, as in the **Bresenham** algorithm. Other-width line are displayed as positive integer multiples of the standard line by plotting additional pixels along adjacent parallel line paths. For lines with slope magnitude less than 1, we can modify a line-drawing routine to display thick lines by plotting a vertical span of pixels at each ***x*** position along the line. The number of pixels in each span is set equal to the integer magnitude of parameter lw. In Figure below, we plot a double-width line by generating a parallel line above the original line path. At each ***x*** sampling position, we calculate the corresponding ***y*** coordinate and plot pixels with screen coordinates ***(x, y)*** and ***(x,* y+l).** We display lines with lw ≥ 3 by alternately plotting pixels above and below the single-width line path.

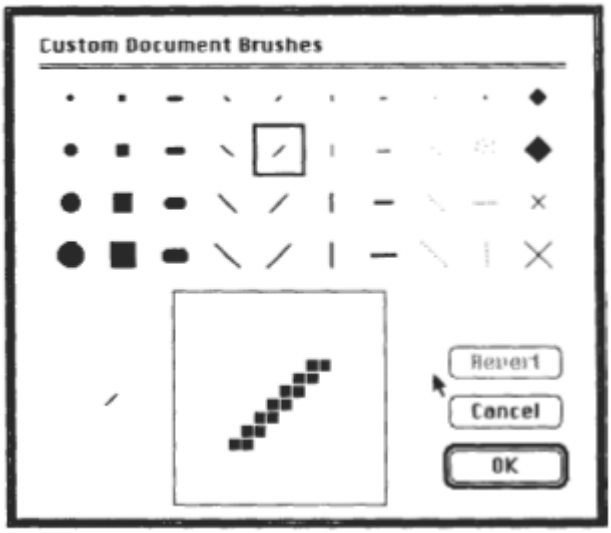


Double-wide raster line with slope |m| < 1 generated with vertical pixel spans

Pen and Brush Options

With some packages, lines can be displayed with pen or brush selections. ***Op***tions in this category include shape, size, and pattern. Some possible pen or brush shapes are given in Figure below. These shapes can be stored in a pixel mask that identifies the array of pixel positions that are to be set along the line path. To avoid setting pixels more than once in the frame buffer, we can simply

accumulate the horizontal spans generated at each position of the mask and keep track of the beginning and ending ***x*** positions for the spans across each scan line.
Lines generated with pen (or brush) shapes can be displayed in various widths by changing the size of the mask. For example, the rectangular pen line in. Also, lines can be displayed with selected patterns by superimposing the pattern values onto the pen or brush mask. An additional pattern option that can be provided in a paint package is the display of simulated brush strokes.



Pen and brush shapes for line display

Line Color
When a system provides color (or intensity) options, a parameter giving the current color index is included in the list of system-attribute values. A polyline routine displays a line in the current color by setting this color value in the frame buffer at pixel locations along the line path using the **setpixel** procedure. The number of color choices depends on the number of bits available per pixel in the frame buffer.
We set the line color value in **PHIGS** with the function

setPolylineColourIndex(lc)

Nonnegative integer values, corresponding to allowed color choices, are assigned to the line color parameter **lc.** A **line** drawn in the background color is invisible, and a user can erase a previously displayed line by respecifying it in the background color (assuming the line does not overlap more than one background color area).
An example of the use of the various line attribute commands in an application program is given by the following sequence of statements:

(4 Marks)

```
setlinetype(2);
setLinewidthScaleFactor ( 2 ) ;
setPolylineColourIndex (5);
polyline (n1, wcpoints1) ;
setPolylinecolourIndex(6);
polyline (n2, wcpoints2);
```

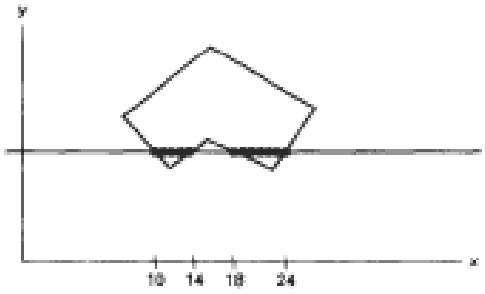
This program segment would display ***two*** figures, drawn with double-wide dashed lines. The first is displayed in a color corresponding to code 5, and the second in color 6.

b. Describe general Scan-Line Polygon-Fill Algorithm. (8)

Answer:

Scan-Line Polygon Fill Algorithm

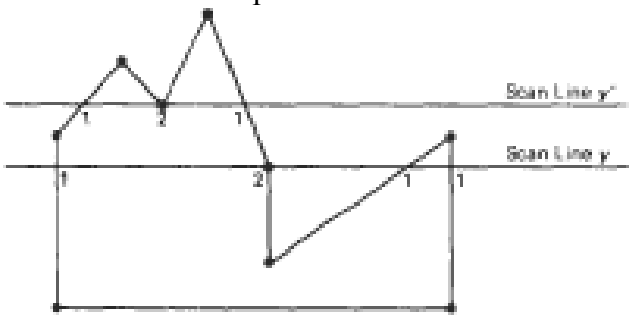
Figure (A) below illustrates the scan-line procedure for solid filling of polygon areas. For each scan line crossing a polygon, the area-fill algorithm locates the intersection points of the scan line with the polygon edges. These intersection points are then sorted from left to right, and the corresponding frame-buffer positions between each intersection pair are set to the specified fill color. In the example of below figure (A)**,** the four pixel intersection positions with the polygon boundaries define two stretches of interior pixels from $x = 10$ to $x = 14$ and from $x = 18$ to $x = 24$.



(4 Marks)

(A) Interior pixels along a scan line passing through a polygon area

Some scan-line intersections at polygon vertices require special handling. **A** scan line passing through a vertex intersects two edges at that position, adding two points to the list of intersections for the scan line. Figure (B) below shows two scan lines at positions y and y' that intersect edge endpoints. Scan line y intersects five polygon edges. Scan line y', however, intersects an even number of edges although it also passes through a vertex. Intersection points along scan line **y'** correctly identify the interior pixel spans. But with scan line y, we need to do some additional processing to determine the correct interior points.



(B)

The topological difference between scan line y and scan line y' in figure (B) is identified by noting the position of the intersecting edges relative to the scan line. For scan line y, the two intersecting edges sharing a vertex are on opposite sides of the scan line. But for scan line **y',** the two intersecting edges are both above the scan line. Thus, the vertices that require additional processing are those that have connecting edges on opposite sides of the scan line. We can identify these

vertices by tracing around the polygon boundary either in clockwise or counterclockwise order and observing the relative changes in vertex y coordinates as we move from one edge to the next. If the endpoint y values of two consecutive edges monotonically increase or decrease, we need to count the middle vertex as a single intersection point for any scan line passing through that vertex. Otherwise, the shared vertex represents a local extremum (minimum or maximum) on the **polv**gon boundary, and the two edge intersections with the scan line passing through that vertex can be added to the intersection list.
One way to resolve the question as to whether we should count a vertex as one intersection or two is to shorten some polygon edges to split those vertices that should be counted as one intersection. We can process nonhorizontal edges around the polygon boundary in the order specified, either clockwise or counterclockwise. As we process each edge, we can check to determine whether that edge and the next nonhorizontal edge have either monotonically increasing or decreasing endpoint y values. If so, the lower edge can be shortened to ensure that only one intersection point is generated for the scan line going through the common vertex joining the two edges.

(4 Marks)

Q.5 a. Prove that a midpoint of a straight line PQ[(0, 2), (3, 2)] after transformation $\begin{bmatrix}1 & 2\\ 3 & 1\end{bmatrix}$ will be the same midpoint of the transformed straight line P′Q′ drawn after the transformation. (8)

Answer:
Applying transformation to the straight line PQ

$$\begin{bmatrix}P\\ Q\end{bmatrix}[T] = \begin{bmatrix}0 & 2\\ 3 & 2\end{bmatrix}\begin{bmatrix}1 & 2\\ 3 & 1\end{bmatrix}$$

(2 Marks)

or

$$\begin{bmatrix}P'\\ Q'\end{bmatrix} = \begin{bmatrix}6 & 2\\ 9 & 8\end{bmatrix}$$

Midpoint of the transformed straight line is $[x'_m, y'_m] = [(6+9)/2, (2+8)/2] = (7.5, 5)$
Midpoint of the original straight line PQ = $[x'_m, y'_m] = [(0+3)/2, (2+2)/2] = (1.5, 2)$

(2 Marks)

Applying transformation to the original midpoint

$$[1.5 \quad 2]\begin{bmatrix}1 & 2\\ 3 & 1\end{bmatrix} = [7.5 \quad 5]$$

(4 Marks)

This is same as midpoint of the transformed straight line.

b. Show that 2D rotation followed by scaling operation is commutative if S_x=S_y. (5)

Answer:

The matrix representation for a two-dimensional transformation sequence consisting of scaling followed by rotation, relative to the coordinate origin, is

$$S_{sx,sy} * R_\theta = \begin{bmatrix} S_{sx} & 0 & 0 \\ 0 & S_{sy} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad \textit{(1+1 Mark)}$$

$$= \begin{bmatrix} S_{sx}\cos\theta & -S_{sy}\sin\theta & 0 \\ S_{sx}\sin\theta & S_{sy}\cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad \textit{(1 Mark)}$$

And the matrix representation for a rotation followed by scaling, relative to the coordinate origin, is

$$R_\theta * S_{sx,sy} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} S_{sx} & 0 & 0 \\ 0 & S_{sy} & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad \textit{(1 Mark)}$$

$$= \begin{bmatrix} S_{sx}\cos\theta & -S_{sx}\sin\theta & 0 \\ S_{sy}\sin\theta & S_{sy}\cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad \textit{(1 Mark)}$$

These two matrices are equivalent only when $S_x = S_y$.

c. Explain scaling in 2D with respect to a general fixed point. (3)

Answer:

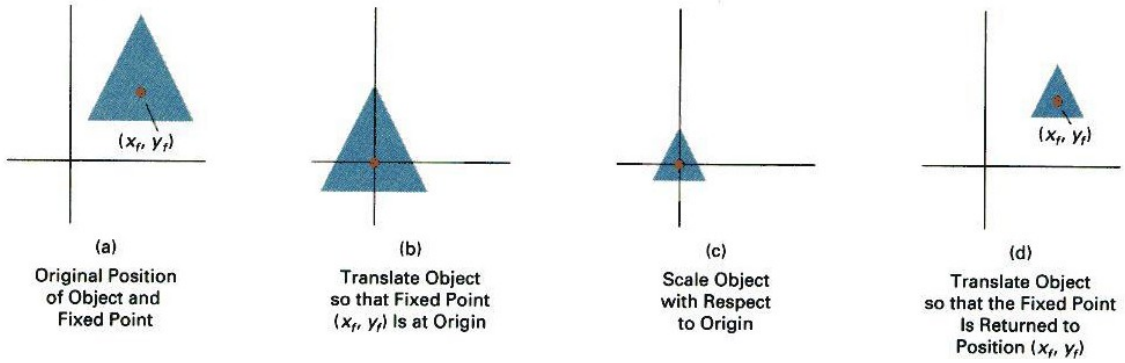
General Fixed-Point Scaling

Scaling with respect to an arbitrary fixed point is not as simple as scaling with respect to the origin. The procedure of scaling with respect to an arbitrary fixed point is:

1. Translate the object so that the fixed point coincides with the origin.

2. Scale the object with respect to the origin.

3. Use the inverse translation of step 1 to return the object to its original position.



(4 Marks)

Q.6 a. Describe Liang-Barsky line clipping algorithm? (8)

Answer:

Faster line clippers have been developed that are based on analysis of the parametric equation of a line segment, which can be in the form

$$x = x_1 + u\Delta x$$
$$y = y_1 + u\Delta y, \qquad 0 \le u \le 1$$

where $\Delta x = x_2 - x_1$ and $\Delta y = y_2 - y_1$**.** Using these parametric equations, Cyrus and Beck developed an algorithm that is generally more efficient than the Cohen-Sutherland algorithm. Later, Liang and Barsky independently devised an even faster parametric line-clipping algorithm. Following the Liang-Barsky approach, we first write the point-clipping conditions in the parametric form:

$$xw_{min} \le x_1 + u\Delta x \le xw_{max}$$
$$yw_{min} \le y_1 + u\Delta y \le yw_{max}$$

Each of these four inequalities can be expressed as

$$up_k \le q_k, \qquad k = 1,2,3,4$$

where parameters p and q are defined as

$$p_1 = -\Delta x, \qquad q_1 = x_1 - xw_{min}$$
$$p_2 = \Delta x, \qquad q_2 = xw_{max} - x_1$$
$$p_3 = -\Delta y, \qquad q_3 = y_1 - yw_{min}$$
$$p_4 = \Delta y, \qquad q_4 = yw_{max} - y_1$$

Any line that is parallel to one of the clipping boundaries has $p_k = 0$ for the value of ***k*** corresponding to that boundary ***(k*** = 1, ***2,*** 3, and 4 correspond to the left, right, bottom, and top boundaries, respectively). If, for that value of ***k,*** we also find $q_k < 0$, then the line is completely outside the boundary and can be eliminated from further consideration. If $q_k \ge 0$, the line is inside the parallel clipping boundary.

When $p_k < 0$, the infinite extension of the line proceeds from the outside to the inside of the infinite extension of this particular clipping boundary If $p_k > 0$, the line proceeds from the inside to the outside. For **a** nonzero value of p_k**,** we can calculate the value of **u** that corresponds to the point where the infinitely extended line intersects the extension of boundary k as

$$u = q_k / p_k$$

For each line, we can calculate values for parameters $\mathbf{u_1}$**,** and $\mathbf{u_2}$ that define that part of the line that lies within the clip rectangle. The value of u_1, is determined by looking at the rectangle edges for which the line proceeds from the outside to the inside ($p < 0$). For these edges, we calculate $r_k = q_k / p_k$ **.** The value of u_1 is taken as the largest of the set consisting of *0* and the various values of ***r.*** Conversely, the value of $\mathbf{u_2}$ is determined **by** examining the boundaries for which the line proceeds from inside to outside ($p > 0$). A value of r_k **,** is calculated for each of these boundaries, and the value u_2**,** is the minimum of the set consisting of 1 and the calculated ***r*** values. If $u_1 > u_2$**,** the line is completely outside the clip window and it can be rejected. Otherwise, the endpoints of the clipped line are calculated from the two values of parameter u.

In general, the Liang-Barsky algorithm is more efficient than the Cohen-Sutherland algorithm, since intersection calculations are reduced. Each update of parameters u_1 and u_2 requires only one division; and window intersections of the line are computed only once, when the final values of u_1 and u_2 have been computed. In contrast, the Cohen-Sutherland algorithm can repeatedly calculate intersections along a line path, even though the line may be completely outside the clip window. And, each intersection calculation requires both a division and a multiplication. Both the Cohen-Sutherland and the Liang-Barsky algorithms can be extended to three-dimensional clipping

b. Using Cohen-Sutherland line clipping, compute the visible portion of the line segment A(0.6,0.8), B(2.4,1.7) for window (x_{min},y_{min})=(0,0) and (x_{max},y_{max})=(2,2). (8)

Answer:

Starting from leftmost bit, each bit of region code for point(x,y) is set to True (1) or false (0) as follows:

Bit 1 ≡ end point above window = sign($y-y_{max}$)
Bit 2 ≡ end point below window = sign($y_{min}-y$)
Bit 3 ≡ end point to right of window = sign($x-x_{max}$)
Bit 4 ≡ end point to left of window = sign($x_{min}-x$) *(2 Marks)*

Here sign(a)= 1 if a is positive
0 otherwise

By Cohen Sutherland Algorithm

A(0.6,0.8) → (0000) and B(2.4,1.7) → (0010) *(2 Marks)*

Since logical AND of AB line segment is 0000, so this line is candidate of clipping.
Also, this line intersects line x=2
Calculate y= y1+m*(x-x1)= 0.6+0.5*(2-0.6)=1.3 where m=(y2-y1)/(x2-x1) *(2 Marks)*

So, intersection point I(2,1.3)
Region code for point I → (0000) *(1 Mark)*

Since both points A and I have region code (0000), therefore line is visible and having endpoints A(0.6,0.8) and I(2,1.3) *(1 Mark)*

Q.7 a. Define Diffuse reflection and Specular reflection. Give the intensity of light in both the cases. (8)

Answer:

The two different types of reflection of incident light are as follows:

- **Diffuse scattering:** occurs when some of the incident light slightly penetrates the surface and is re-radiated uniformly in all directions. Scattered light interacts strongly with the surface, and so its color is usually affected by the nature of the surface material.
- **Specular reflections:** are more mirrors like and are highly directional. Incident light does not penetrate the object but instead is reflected directly from its outer surface. This gives rise to highlights and makes the surface look shiny. In the simplest model for specular light the

reflected light has the same color as the incident light. This tends to make the material look like plastic. In a more complex model the color of the specular light varies over the highlight, providing a better approximation to the shininess of metal surfaces.

(2+2+2+2 Marks)

b. Describe Back face hidden surface removal algorithm along with its limitations.(8)

Answer:

Object surfaces that are oriented away from the viewer are called back-faces. The back-faces of an opaque polyhedron are completely blocked by the polyhedron itself and hidden from view. We can therefore identify and remove these back-faces based solely of their orientation without further processing (projection and scan-conversion) and without regard to other surfaces and objects in the scene.

A point ***(x,*** y,z) is "inside" a polygon surface with plane parameters A, B, C, and D if Ax+By+Cz<0. When an inside point is along the line of sight to the surface, the polygon must be a back face (we are inside that face and cannot see the front of it from our viewing position).

We can simplify this test by considering the normal vector N to a polygon surface, which has Cartesian components ***(A, B, C).*** In general, if V is a vector in the viewing direction from the eye (or "camera") position, as shown in Fig. 13-1,then this polygon is a back face if **V.M>0**. Furthermore, if object descriptions have been converted to projection coordinates and our viewing direction is parallel to the viewing z_v axis, then V = **(0,** 0, ***V_z)***and so that we only need to consider the sign of C, the z component of the normal vector N.

In a right-handed viewing system with viewing direction along the negative z_v, axis (Fig. 13-21, the polygon is a back face if C < 0. AIso, we cannot see any face whose normal has z component C = 0, since 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: C<=0

Figure 13-1
Vector **V** in the viewing direction and a back-face normal vector **N** of a polyhedron

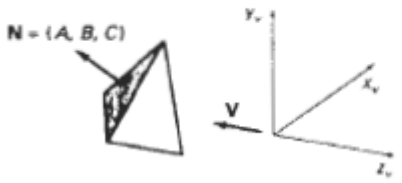


Figure 13-2
A polygon surface with plane parameter $C < 0$ in a right-handed viewing coordinate system is identified as a back face when the viewing direction is along the negative z_v axis.

Although this method identifies or removes back-faces quickly it does not handle polygons that face the viewer but are hidden (partially or completely) behind other surfaces. It can be used as a preprocessing step for other algorithms.

Q.8 a. Explain how logical input-device can be classified in terms of the kind of data to be input by the devices? (10)

Answer: *(1 Mark)*

Graphics programs use several kinds of input data. Picture specifications need values for coordinate positions, values for the character-string parameters, scalar values for the transformation parameters, values specifying menu options, and values for identific.hon of picture parts. Any of the input devices can be used to input the various graphical data types, but **some** devices are better suited for certain data types than others. To make graphics packages independent of the particular hardware devices used, input functions can he structured according to the data description to be handled by each Function. This approach provides a logical input-device classification in terms of the kind of data to be input by the device.

The various kinds of input data are summarized in the following six logical device classifications used by PHlGS and GKS:

- LOCATOR-a device for specifying a coordinate position (x, y)
- STROKE- a device for specifying a series of coordinate positions
- STRING- a device for specifying text input
- VALUATOR- a device for specifying scalar values
- CHOICE- a device for selecting menu options
- PICK-a device for selecting picture components

(6 X 3/2 Marks)

In some packages, a single logical device is used for both locator and stroke operations. Some other mechanism, such as a switch, can then be used to indicate whether one coordinate position or a "stream" of positions 1s to be input.

Each of the six logical input device classifications can be implemented with any of the hardware devices, but some hardware devices are more convenient for certain kinds **of** data than others. A device that can be pointed at a screen position is more convenient for entering coordinate data than a keyboard, for example.

Locator Devices

A standard method for interactive selection of a coordinate point is by positioning the screen cursor. We can do this with a mouse, joystick, trackball, spaceball, thurnbwheels, dials, a digitizer stylus or hand cursor, or some other cursor-positioning device. When the screen cursor is at the desired location, a button is activated to store the coordinates of that screen point.

Keyboards can be used for locator input in several ways. A general-purpose keyboard usually has four cursor-control keys that move the screen cursor up, down, left, and right. With an additional four keys, we can move the cursor diagonally as well. Rapid cursor movement is accomplished by holding down the **se**lected cursor key. Alternatively, a joystick, joystick, trackball, or thumbwheels can be mounted on the keyboard for relative cursor movement. As a last resort, we could actually type in coordinate values, but this is a slower process that also requires us to know exact coordinate values.

Light pens have also been used to input coordinate positions, but some special implementation considerations are necessary. Since light pens operate by detecting light emitted from the screen phosphors, some nonzero intensity level must be present at the coordinate position to be selected. With a raster system, we can paint a color background onto the screen. As long as no black areas are present, a light pen can be used to select any screen position. When it is not possible to eliminate all black areas in a display (such as on a vector system, for example), a light pen can be

used as a locator by creating a small Light pattern for the pen to detect. The pattern is moved around the screen until it finds the light pen.

Stroke Devices

This class of logical devices is used to input a sequence of coordinate positions. Stroke-device input is equivalent to multiple calls to a locator device. The set of input points is often used to display line sections.

Many of the physical devices used for generating locator input can be used as stroke devices. Continuous movement of a mouse, trackball, joystick, or tablet hand cursor is translated into a series of input coordinate values. The graphics tablet is one of the more common stroke devices. Button activation can be used to place the tablet into "continuous" mode. As the cursor is moved across the tablet surface, a stream of coordinate values is generated. This process is used in paintbrush systems that allow artists to draw scenes on the screen and in engineering systems where layouts can be traced and digitized for storage.

String Devices

The primary physical device used for string input is the keyboard. Input character strings are typically used for picture or graph labels.

Other physical devices can be used for generating character patterns in a "text-writing" mode. For this input, individual characters are drawn on the screen with a stroke or locator-type device. A pattern-recognition program then interprets the characters using a stored dictionary of predefined patterns.

Valuator Devices

This logical class of devices is employed in graphics systems to input scalar values. Valuators are used for setting various graphics parameters, such as rotation angle and scale factors, and for setting physical parameters associated with a particular application (temprature settings, voltage levels, shess factors, etc.).

A typical physical device used to provide valuator input is a set of control dials. Floating-point numbers within any predefined range are input by rotating the dials. Dial rotations in one direction increase the numeric input value, and opposite rotations decrease the numeric value. Rotary potentiometers convert dial rotation into a corresponding voltage. This voltage is then translated into a real number within a defined scalar range, such as -10.5 to 25.5. Instead of dials, slide potentiometers are sometimes used to convert linear movements into scalar values.

Any keyboard with a set of numeric keys can be used as a valuator device. A user simply types the numbers directly in floating-point format, although this is a slower method than using dials or slide potentiometers

Joystick, trackball, tablets, and other interactive devices can be adapted for valuator input by interpreting pressure or movement of the device relative to a scalar range. For one direction of movement, say, left to right, increasing scalar values can be input. Movement in the opposite direction decreases the scalar input value.

Choice Devices

Graphics packages use menus to select programming options, parameter values, and object shapes to be used in constructing a picture. **A** choice device IS defined as one that enters a selection from a

list (menu) of alternatives. Commonly used choice devices are a set of buttons; a cursor positioning device, such as a mouse, trackball, or keyboard cursor keys; and a touch panel.
A function keyboard, or "button box", designed as a stand-alone unit, is often used to enter menu selections. Usually, each button is programmable, so that its function can be altered to suit different applications. Single-purpose buttons have fixed, predefined functions. Programmable function keys and fixed function buttons are often included with other standard keys on a keyboard.
For screen selection of listed menu options, we can use cursor-control devices. When a coordinate position ***(x,* y)** is selected, it is compared to the coordinate extents of each listed menu item. **A** menu item with vertical and horizontal boundaries at the coordinate values x_{min}, x_{max}, y_{min} , and y_{max} is selected if the input coordinates **(x, y)** satisfy the inequalities

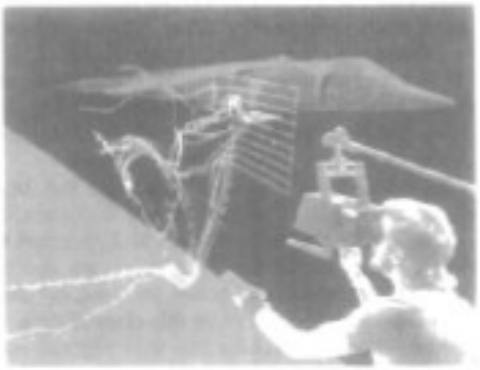
$$x_{min} \leq x \leq x_{max} , \qquad y_{min} \leq y \leq y_{max}$$

For larger menus with a few options displayed at a time, a touch panel is commonly used. As with a cursor-control device, such as a mouse, a selected screen position is compared to the area occupied by each menu choice.
Alternate methods for choice input include keyboard and voice entry. **A** standard keyboard can be used to type in commands or menu options. For this method of choice input, some abbreviated format is useful. Menu listings can **be** numbered or given short identifying names. Similar codings can be used with voice-input systems. Voice input is particularly useful when the number of options is small (20 or less).

Pick Devices
Graphical object selection is the function of this logical class of devices. Pick devices are used to select parts of a scene that are to be transformed or edited in some way.
Typical devices used for object selection are the same as those for menu ***se***lection: the cursor-positioning devices. With a mouse or joystick, we can position the cursor over the primitives in a displayed structure and press the selection button. The position of the cursor is then recorded, and several levels of search may be necessary to locate the particular o b p t (if any) that is to **be** selected. First, the cursor position is compared to the coordinate extents of the various structures in the scene. If the bounding rectangle of a structure contains the cursor coordinates, the picked structure has been identified. But if two or more structure areas contain the cursor coordinates, further checks are necessary. The coordinate extents of individual lines in each structure can **be** checked next. If the cursor coordinates are determined to be inside the coordinate extents of only one line, for example, we have identified the picked object. Otherwise, we need additional checks to determine the closest line to the cursor position.

b. Explain virtual reality environment. (6)

Answer:
A typical virtual-reality environment is illustrated in Figure below. Interactive input is accomplished in this environment with a data glove, which is capable of grasping and moving objects displayed in a virtual scene. The computer generated scene is displayed through a head-mounted viewing system **as** a stereoscopic projection. Tracking devices compute the position and orientation of the headset and data glove relative to the object positions in the scene. With this system, a user can move through the scene and rearrange object positions with the data glove.



Another method for generating virtual scenes is to display stereoscopic projections on a raster monitor, with the two stereoscopic views displayed on alternate refresh cycles. The scene is then viewed through stereoscopic glasses. Interactive object manipulations can again be accomplished with a data glove and a tracking device to monitor the glove position and orientation relative to the position of objects in the scene.

Q.9 a. What is animation? What are the different methods to produce real time animation? (2+6)

Answer:
Real Time Animation is the rapid display of a sequence of images of 2-D or 3-D artwork or model positions in order to create an illusion of movement. It is an optical illusion of motion due to the phenomenon of persistence of vision, and can be created and demonstrated in a number of ways. The most common method of presenting animation is as a motion picture or video program, although several other forms of presenting animation also exist.

Computer animation (or CGI animation) is the art of creating moving images with the use of computers. It is a subfield of computer graphics and animation. Increasingly it is created by means of 3D computer graphics, though 2D computer graphics are still widely used for stylistic, low bandwidth, and faster real-time rendering needs. Sometimes the target of the animation is the computer itself, but sometimes the target is another medium, such as film. It is also referred to as CGI (computer-generated imagery or computer-generated imaging), especially when used in films.
To create the illusion of movement, an image is displayed on the computer screen and repeatedly replaced by a new image that is similar to the previous image, but advanced slightly in the time domain (usually at a rate of 24 or 30 frames/second). This technique is identical to how the illusion of movement is achieved with television and motion pictures.

Computer animation is essentially a digital successor to the art of stop motion animation of 3D models and frame-by-frame animation of 2D illustrations. For 3D animations, objects (models) are built on the computer monitor (modeled) and 3D figures are rigged with a virtual skeleton. For 2D figure animations, separate objects (illustrations) and separate transparent layers are used, with or without a virtual skeleton. Then the limbs, eyes, mouth, clothes, etc. of the figure are moved by the animator on key frames. The differences in appearance between key frames are automatically calculated by the computer in a process known as tweening or morphing. Finally, the animation is rendered.

For 3D animations, all frames must be rendered after modeling is complete. For 2D vector animations, the rendering process is the key frame illustration process, while tweened frames are rendered as needed. For pre-recorded presentations, the rendered frames are transferred to a different format or medium such as film or digital video. The frames may also be rendered in real time as they are presented to the end-user audience. Low bandwidth animations transmitted via the internet (e.g. 2D Flash, X3D) often use software on the end-users computer to render in real time as an alternative to streaming or pre-loaded high bandwidth animations.

There are several methods for generating the Avar values to obtain realistic motion. Traditionally, animators manipulate the Avars directly. Rather than set Avars for every frame, they usually set Avars at strategic points (frames) in time and let the computer interpolate or 'tween' between them, a process called keyframing. Keyframing puts control in the hands of the animator, and has roots in hand-drawn traditional animation.

In contrast, a newer method called motion capture makes use of live action. When computer animation is driven by motion capture, a real performer acts out the scene as if they were the character to be animated. His or her motion is recorded to a computer using video cameras and markers, and that performance is then applied to the animated character.

b. Write short notes on: (2×4)

(i) Frame-by-frame animation

(ii) Keyframes

Answer:

(i) Frame-by-frame animation

This type of animation is required for creating sophisticated animation on graphics workstation with minute detail, shadows, smooth-shaded 3D objects, innumerous lightening, trxturing, ray tracingscenes with global reflection & refractions. Each frame of the scene is produced one by one and is captured and stored in any suitable video frame format. Hence the completesequence can be loaded onto frame buffer at 30 frames per second to generate an animation.

(ii) Keyframe

A keyframe is a detailed drawing of the scene at a **certain** time in the animation sequence. Within each key frame, each object is positioned according to the time for that frame. Some key frames are chosen at extreme positions in the action; others are spaced **so** that the time interval between key frames is not *too* great. More key frames are specified for intricate motions than for simple, slowly varing motions.

TEXT BOOK

Systems Programming and Operating Systems, D. M. Dhamdhere, Tata McGraw-Hill, Second Revised Edition, 2005