Q.2 a. Name some digital input devices and briefly explain them with respect to their functioning.

Answer:

Various devices are available for data input on graphics workstations. Most systems have a keyboard and one or more additional devices specially designed for interactive input. These include a mouse, trackball, spaceball, joystick, digitizers, are data gloves, touch panels, image scanners, and voice systems.

Keyboards

An alphanumeric keyboard on a graphics system is used primarily as a device for entering text strings. The keyboard is an efficient device for inputting such nongraphic data as picture labels associated with a graphics display. Keyboards can also be provided with features to facilitate entry of screen coordinates, menu selections, or graphics functions.

Cursor-control keys and function keys are common features on general purpose keyboards. Function keys allow users to enter frequently used operations in a single keystroke, and cursor-control keys can be used to select displayed objects or coordinate positions by positioning the screen cursor. Other types of cursor-positioning devices, such as a trackball or joystick, are included on some keyboards. Additionally, a numeric keypad is, often included on the keyboard for fast entry of numeric data.

For specialized applications, input to a graphics application may come from a set of buttons, dials, or switches that select data values or customized graphics operations. Buttons and switches are often used to input predefined functions, and dials are common devices for entering scalar values. Real numbers within some defined range are selected for input with dial rotations. Potentiometers are used to measure dial rotations, which are then converted to deflection voltages for cursor movement.

Mouse

A mouse is small hand-held box used to position the screen cursor. Wheels or rollers on the bottom of the mouse can be used to record the amount and direction of movement. Another method for detecting mouse motion is with an optical sensor. For these systems, the mouse is moved over a special mouse pad that has a grid of horizontal and vertical lines. The optical sensor detects movement across the lines in the grid.

Since a mouse can be picked up and put down at another position without change in cursor movement, it is used for making relative changes in the position of the screen cursor. One, two, or three buttons are usually included on the top of the mouse for signaling the execution of some operation, such as recording cursor position or invoking a function. Most general-purpose graphics systems now include a mouse and a keyboard as the major input devices.

Additional devices can be included in the basic mouse design to increase the number of allowable input parameters. The Z mouse in Figure below includes three buttons, a thumbwheel on the side, a trackball on the top, and a standard mouse ball underneath. This design provides six degrees of freedom to select Input Devices spatial positions, rotations, and other parameters. With the Z mouse, we can pick up an object, rotate it, and move it in any direction, or we can navigate our viewing position and orientation through a three

dimensional scene. Applications of
the Z mouse include virtual reality, CAD, and animation.

The Z mouse

Trackball and Spaceball

As the name implies, a trackball is a ball that can be rotated with the fingers or palm of the hand, as in Figure below, to produce screen-cursor movement. Potentiometers, attached to the ball, measure the amount and direction of rotation. Trackballs are often mounted on keyboards or other devices such as the Z mouse.

While a trackball is a two-dimensional positioning device, a spaceball provides six degrees of freedom. Unlike the trackball, a spaceball does not actually move. Strain gauges measure the amount of pressure applied to the spaceball to provide input for spatial positioning and orientation as the ball is pushed or pulled in various directions. Spaceballs are used for three dimensional positioning and selection operations in virtual-reality systems, modeling, animation, CAD, and other applications.

Trackball

Joysticks

A joystick consists of a small, vertical lever (called the stick) mounted on a base that is used to steer the screen cursor around. Most joysticks select screen positions with actual stick movement; others respond to pressure on the stick. Some joysticks are mounted on a keyboard; others function as stand-alone units.

The distance that the stick is moved in any direction from its center position corresponds to screen-cursor movement in that direction. Potentiometers mounted at the base of the joystick measure the amount of movement, and springs return the stick to the center position when it is released. One or more buttons can be programmed to act as input switches to signal certain actions once a screen position has been selected.

Data Glove

Data glove can be used to grasp a "virtual" object. The glove is constructed with a series of

sensors that detect hand and finger motions. Electromagnetic coupling between transmitting antennas and receiving antennas is used to provide information about the position and orientation of the hand. The transmitting and receiving antennas can each be structured as a set of three mutually perpendicular coils, forming a three-dimensional Cartesian coordinate system. Input from the glove can be used to position or manipulate objects in a virtual scene. A two-dimensional projection of the scene can be viewed on a video monitor, or a three-dimensional projection can be viewed with a headset.

Digitizers

A common device for drawing, painting, or interactively selecting coordinate positions on an object is a digitizer. These devices can be used to input coordinate values in either a two-dimensional or a three-dimensional space. Typically, a digitizer is used to scan over a drawing or object and to input a set of discrete coordinate positions, which can be joined with straight line segments to approximate the curve or surface shapes.

One type of digitizer *is* the graphics tablet (also referred to as a data tablet), which is used to input two-dimensional coordinates by activating a hand cursor or stylus at selected positions on a flat surface. A hand cursor contains cross hairs for sighting positions, while a stylus is a pencil-shaped device that is pointed at positions on the tablet.

Touch Panels

As the name implies, touch panels allow displayed objects or screen positions to be selected with the touch of a finger. A typical application of touch panels is for the selection of processing options that are represented with graphical icons. Some systems, such as the plasma panels are designed with touch screens. Other systems can be adapted for touch input by fitting a transparent device with a touch sensing mechanism over the video monitor screen. Touch input can be recorded using optical, electrical, or acoustical methods.

Optical touch panels employ a line of infrared light-emitting diodes (LEDs) along one vertical edge and along one horizontal edge of the frame. The opposite vertical and horizontal edges contain light detectors. These detectors are used to record which beams are interrupted when the panel is touched. The two crossing beams that are interrupted identify the horizontal and vertical coordinates of the screen position selected. Positions tin be selected with an accuracy of about ¼ inch With closely spaced LEDs, it is possible to break two horizontal or two vertical beams simultaneously. In this case, an average position between the two interrupted beams is recorded. The LEDs operate at infrared frequencies, so that the light is not visible to a user.

An electrical touch panel is constructed with two transparent plates separated by a small distance. One of the plates is coated with a conducting material, and the other plate is coated with a resistive material. When the outer plate is touched, it is forced into contact with the inner plate. This contact creates a voltage drop across the resistive plate that is converted to the coordinate values of the selected *screen* position.

Light Pens

Figure below shows the design of one type of light pen. Such pencil-shaped devices are used to select screen positions by detecting the light coming from points on the CRT screen. They are sensitive to *the* short burst of light emitted from the phosphor coating at

the instant the electron beam strikes a particular point. Other Light sources, such as the background light in the room, are usually not detected by a light pen. An activated light pen, pointed at a spot on the screen as the electron beam lights up that spot, generates an electrical pulse that causes the coordinate position of the electron beam to be recorded. As with cursor-positioning devices, recorded Light-pen coordinates can be used to position an object or to select a processing option.

Although Light pens are still with us, they are not as popular as they once were since they have several disadvantages compared to other input devices that have been developed. For one, when a light pen is pointed at the screen, part of the screen image is obscured by the hand and pen. And prolonged use of the light pen can cause arm fatigue. Also, light pens require special implementations for some applications because they cannot detect positions within black areas. To be able b select positions in any screen area with a light pen, we must have some nonzero intensity assigned to each screen pixel. In addition, light pens. Sometimes give false readings due to background lighting in a room.

Light Pen

b. Compute the following:
(i) Size of 800 x 600 image at 240 pixels per inch.
(ii) Resolution of 2 x 2 inch image that has 512 x 512 pixels.
(iii) Height of the resized image 1024 x 768 to one that is 640 pixels wide with the same aspect ratio.
(iv) Width of an image having height of 5 inches and an aspect ratio 1.5.

Answer:

i) 240 pixels corresponds to 1 inch
=> 800 pixels will corresponds to 800 / 240 inch = 31 / 3 inch
Similarly 600 pixels => 600 / 240 = 21 / 2 inch
Hence, the size of the image is 31 / 3 inch x 21 / 2 inch

ii) 512 / 2 = 256 pixels per inch.

iii) Aspect ratio of the 1024 x 768 image is 768 / 1024 = ¾.
Hence, width of the image having height of 640 pixels having aspect ratio ¾ is 640 x ¾ = 480.

iv) Width of the image of aspect ratio 1.5 = 5 x 1.5 = 7.5.

Q.3 a. State and explain DDA algorithm for line drawing along with its drawbacks.

Answer:

The *digital* ***differential*** analyzer **(DDA)** is a scan-conversion line algorithm based on calculating either Δy or $\Delta \boldsymbol{x}$, where $\Delta y = m.\Delta x$ We sample the line at unit intervals in one coordinate and determine corresponding integer values nearest the line path for the other coordinate.

Consider first a line with positive slope**.** If the slope is less than or equal to **1,** we sample at unit ***x*** intervals ***(***$\Delta \boldsymbol{x} = 1$) and compute each successive **y** value as

$$y_{k+1} = y_k + m \text{ ---------- equ 1.}$$

Subscript ***k*** takes integer values starting from 1, for the first point, and increases by 1 until the final endpoint is reached. Since m can **be** any real number between 0 and 1, the calculated y values must be rounded to the nearest integer.

For lines with a positive slope greater than 1, we reverse the roles of ***x*** and y. That is, we sample at unit y intervals ***(***$\Delta \boldsymbol{y} = 1$) and calculate each succeeding ***x*** value as

$$x_{k+1} = x_k + 1/m \text{ ---------- equ 2.}$$

Equations 1 and 2 are based on the assumption that lines are to be processed from the left endpoint to the right endpoint. If this processing is reversed, so that the starting endpoint is at the right, then either we have

$$\Delta \boldsymbol{x} = \mathbf{-1} \text{ and } y_{k+1} = y_k - m \text{ ----------equ 3}$$

or (when the slope is greater than I) we have

$$\Delta \boldsymbol{y} = -1 \text{ with } x_{k+1} = x_k - 1/m \text{ ----------equ 4}$$

Equations 1 through 4 can also be used to calculate pixel positions along a line with negative slope. If the absolute value of the slope is less than I and the start endpoint is at the left, we set

$$\Delta \boldsymbol{x} = 1$$

and calculate y values with Eq. 1 When the start endpoint is at the right (for the same slope), we set $\Delta \boldsymbol{x} = \mathbf{-1}$ and obtain y positions **from** Eq. 3. Similarly, when the absolute value of a negative slope is greater than **1,** we use $\Delta \mathbf{y} = -1$ and Eq. 4 or we use $\Delta y = 1$ and **Eq.2.**

Drawbacks: The accumulation of round-off error in successive additions of the floating-point increment, however, can cause the calculated pixel positions to drift away from the true line path for long line segments. Furthermore, the rounding operations and floating-point arithmetic are still time-consuming.

b. Discuss the two primary ways to describe the shape of a curved line.

Answer:

There are two principal ways to describe the shape of a curved line: *implicitly* and *parametrically*.

The **implicit form** describes a curve by a function F(x, y) that provides a relationship between the x- and y- coordinates: the point (x, y) lies on the curve if and only if it satisfies:

$F(x, y) = 0$ condition for (x, y) to lie on the curve

For example, the straight line through points A and B has implicit form:

$F(x, y) = (y - A_y)(B_x - A_x) - (x - A_x)(B_y - A_y)$

and the circle with radius R centered at the origin has implicit form:

$F(x, y) = x^2 + y^2 - R^2$ ----------(1)

A benefit of using the implicit is that you can easily test whether a given point lies on the curve. for certain classes of curves it is meaningful to speak of an inside and an outside of the curve, in which cases F(x, y) is also called the inside—outside function as given in the following equations:

$F(x, y) = 0$ for all (x, y) on the curve
$F(x, y) > 0$ for all (x, y) on the curve
$F(x, y) < 0$ for all (x, y) on the curve

Some curves are single valued in x. for instance, if g(x) is single valued, there is only one value of the function for each value of x. in fact only single-valued functions are "legitimate" functions. For such curves the implicit form may be written $F(x, y) = y - g(x)$. Other curves are single valued in y, so there is a function h(y) such that points on the curve satisfy $x = h(y)$. And some curves are not single valued at all: $F(x, y) = 0$ cannot be rearranged into either of the forms $y = g(x)$ or $x = h(y)$. The circle, for instance, can be expressed as

$y = \pm\sqrt{R^2 - x^2}$

but here there are two functions. One function uses the plus sign and the other uses the minus sign.

A **parametric form** for a curve produces different points on the curve based on the value a parameter. Parametric forms can be developed for a wide variety of curves, and they have much to recommend them, particularly when one wants to draw or analyze the curve. A parametric form suggests the movement of a point over time, which we can translate into the motion of a pen as it sweeps out the curve. The path of the particle traveling along the curve is fixed by two functions, [x() and y()], (three functions for a 3D curve, [x(), y(), z()]) that determine the position of the particle at t. The parameter t is frequently referred to as time. The curve itself is the totality of points visited by the particle as t varies over some internal. For any curve, therefore, if we can dream up suitable functions x() and y() or x(), y(), z(), they will represent the curve concisely and precisely.

The straight line equation (1) passes through points A and B. We choose a parametric form that visits A at t = 0 and B at t = 1, obtaining:

$x(t) = A_x + (B_x - A_x)t$
$y(t) = A_y + (B_y - A_y)t$

Thus the point P(t) = (x(t), y(t)) sweeps through all of the points on the line between A and B as t varies from 0 to 1.

Another example is the **ellipse**, a slight generalization of the circle. It is described parametrically by

$x(t) = W \cos(t)$

$y(t) = W \sin(t)$, for $0 \leq t \leq 2\Pi$

Here W is the "half width" and H is the "half height" of the ellipse. When W and H are equal, the ellipse is a circle of radius W.

Q.4 a. Explain the following functions related to Area-Fill attributes:
(i) setInteriorColourIndex(fc)
(ii) setInteriorStyleIndex(pi)
(iii)setPatternSize(dx, dy)
(iv)setPixel(x, y, cp(y mod ny + 1, x mod nx + 1)

Answer:

Refer page no. 158, 159, 161 of Computer Graphics C version 2nd edition, by Donald Hearn & M. Pauline Baker

b. Describe the different approaches to antialiasing techniques.

Answer:

Antialiasing techniques involve one form or another of blurring to smooth the image. In the case of black rectangle against a white background, the sharp transition from black to white is softened by using a mixture of gray pixels near the rectangle's border. When the picture is looked at from afar, the eye blends together the gracefully varying shades of gray and sees a smoother edge.
Three approaches to antialiasing are commonly used: **prefiltering, supersampling, and postfiltering.**

Prefiltering: Prefiltering techniques compute pixel colors based on an objects's coverage: the fraction of the pixel area that is covered by the object. Considered scan-converting a white polygon in a black background, as in figure below. Suppose the intensity values are 0 for black and 1 for white. The polygon is situated in a square grid, where the center of each square corresponds to the center of a pixel on the display. A pixel that is half-covered by the pixel should be given the intensity 1/2 ; one that is one-third covered should be given the intensity 1/3; and so on. If the frame buffer has 4 bits per pixel, so that black is represented by 0 and white by 15, a pixel that is one-quarter covered by the polygon sh0ould be given the value of 901/4)15, which rounds up to 4. The figure also shows the pixel values that result when the coverage of each pixel is calculated.

Figure 9.47, Page 489, Computer Graphics using OpenGL, 3rd Edition, by F.S. Hill Jr. & S.M. Kelly

The geometric computations required to find the coverage for each pixel can, of course, be rather time consuming. A number of efficient approaches have been developed, such as those by Pitteway and Watkinson and more recently by Xiaolin Wu. These algorithms calculate the coverage of each pixel in an incremental fashion, using only integer

arithmetic.
Thus, prefiltering operates on the detailed geometric shape 0of the object(s) being scan converted and computes an average intensity for each pixel based on the objects found lying within each pixel's area. For shapes other than polygons, it can be an expensive technique computationally, and so we shall seek alternative approaches to antialiasing.

Supersampling: Since aliasing arises from sampling an object at too few points, we can try to reduce its effects by sampling more densely than one sample per pixel. This is called **supersampling:** taking more intensity samples of the scene than are displayed. Each display pixel value is formed as the average of several samples.
Following figure shows an example of double sampling: the object, a tilted bar, is sampled twice more densely in both x and y than it is displayed. The squares indicate display pixels, and the x's denote spots at which the scene is sampled. Each final display pixel is formed as the average of the nine neighbor samples: the center one and the eight surrounding ones. Some samples are reused in several pixel calculations. The display pixel centered at A "sees" six is based on six samples within the bar and three samples of background. Its color is set to the sum of two-thirds the bar's color and one-third the background's color. The pixel at B is based on all nine samples within the bar. Its color is set to that of the bar.
Figure 9.48, Page 490, Computer Graphics using OpenGL, 3rd Edition, by F.S. Hill Jr. & S.M. Kelly

In general, supersampling computes N_s scene samples in both x and y for each display pixel, averaging some number of neighbor samples to form each display pixel value. Supersampling with $N_s = 4$, for example, averages 16 samples for each display pixel.

Postfiltering: In the double-sampling method, nine neighboring samples are averaged to compute each display pixel's intensity, giving each neighbor equal importance. This form of blurring or filtering might be improved by giving the center sample more weight and the eight neighbors less weight. Or it may help to include more neighbors in the averaging computation.
Postfiltering computes each display pixel as a **weighted average** of an appropriate set of neighboring samples of the secene. Following figure shows the situation for double sampling. Each value represents the intensity of a scene sample, the o0nes in gray indicating the centers of the various display pixels. The square **mask** or **window function** of weights is laid over each gray square in turn. Then each window weight is multiplied by its corresponding sample, and the nine products are summed to form the display pixel intensity. For example, when the mask shown is laid over the sample of intensity 30, the weighted average is found to be

$$(30)/2 + (28 + 16 + 4 + 42 + 17 + 53 + 60 + 62)/16 = 32.625$$

which rounds to intensity 33. This mask gives eight times as much weight to the center as to the other eight neighbours. The weight always sum to 1.

Figure 9.51, Page 491, Computer Graphics using OpenGL, 3rd Edition, by F.S. Hill Jr. & S.M. Kelly

Q.5 a. Explain the effects of 3D geometric transformations.

Answer:

With respect to some three-dimensional coordinate system, an object Obj is considered as a set of points.

Obj = {P(x,y,z)}

If the object moved to a new position, we can regard it as a new object Obj', all of whose coordinate points P'(x', y', z') can be obtained from the original coordinate points P(x,y,z) of Obj through the application of a geometric transformation.

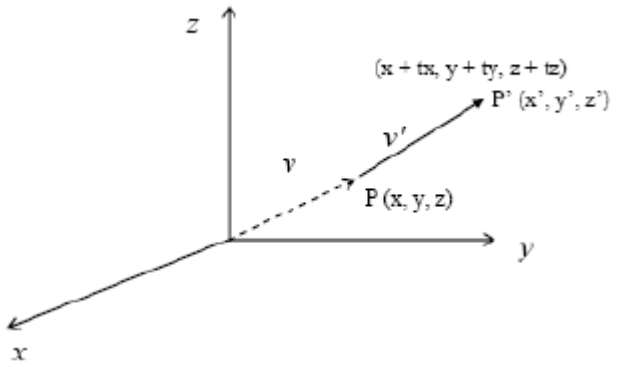
Translation

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

V = aI + bJ + cK

The new coordinates of a translated point can be calculated by using the transformation as shown in figure below:

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



In order to represent this transformation as a matrix transformation, we need to use homogeneous coordinate. The required homogeneous matrix transformation can then be expressed as

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

Scaling

The process of scaling changes the dimensions of an object. The scale 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 offered 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 form 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 then rotation in two dimensions. In 2-D, a rotation is prescribed by an angle of rotation θ and a centre of rotation, say P. Three dimensional rotation is prescribed by an angle of rotation θ and an axis of rotation. The canonical rotations are defined when one of the positive x, y or z coordinate axes 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 as shown in figure:

Rotation about the z axis

We know

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

Rotation about the y Axis

An analogous derivation leads to

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

Rotation about the x Axis

Similarly:

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

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

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

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

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

b. Perform a 45° rotation of triangle A (0, 0), B (1, 1), C (5, 2) (2×4)
(i) about the origin (ii) about P(-1, -1)

Answer:

We represent the triangle by a matrix formed from the homogeneous coordinates of the vertices:

$$\begin{bmatrix} 0 & 1 & 5 \\ 0 & 1 & 2 \\ 1 & 1 & 1 \end{bmatrix}$$

i. The matrix of rotation is

$$R_{45^\circ} = \begin{bmatrix} cos45^\circ & -sin45^\circ & 0 \\ sin45^\circ & cos45^\circ & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \frac{\sqrt{2}}{2} & \frac{-\sqrt{2}}{2} & 0 \\ \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

So the coordinates A′ B′ C′ of the rotated triangle ABC can be found as

$$[A'B'C'] = R_{45^\circ}[ABC] = \begin{bmatrix} \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & 0 \\ \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 & 5 \\ 0 & 1 & 2 \\ 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & \frac{3\sqrt{2}}{2} \\ 0 & \sqrt{2} & \frac{7\sqrt{2}}{2} \\ 1 & 1 & 1 \end{bmatrix}$$

Thus A′ = (0,0), B′=(0,$\sqrt{2}$), and C′ = $(\frac{3\sqrt{2}}{2}, \frac{7\sqrt{2}}{2})$

ii. The rotation matrix is given by

$$R_{45^\circ,P} = \begin{bmatrix} 1 & 0 & -1 \\ 0 & 1 & -1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \frac{\sqrt{2}}{2} & \frac{-\sqrt{2}}{2} & -1 \\ \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & (\sqrt{2}-1) \\ 0 & 0 & 1 \end{bmatrix}$$

Now

$$[A'B'C'] = R_{45^\circ,P}[ABC] = \begin{bmatrix} \frac{\sqrt{2}}{2} & \frac{-\sqrt{2}}{2} & -1 \\ \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & (\sqrt{2}-1) \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 & 5 \\ 0 & 1 & 2 \\ 1 & 1 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} -1 & -1 & (\frac{3\sqrt{2}}{2} - 1) \\ (\sqrt{2} - 1) & (2\sqrt{2} - 1) & (\frac{9\sqrt{2}}{2} - 1) \\ 1 & 1 & 1 \end{bmatrix}$$

So A′= (-1, $\sqrt{2} - 1$) , B′ = (-1, $2\sqrt{2} - 1$), C′ = ($\frac{3\sqrt{2}}{2} - 1, \frac{9\sqrt{2}}{2} - 1$)

Q.6 a. Explain Sutherland-Hodgeman algorithm for polygon clipping. For what type of clipping regions the algorithm is not suitable. Give reason.

Answer:

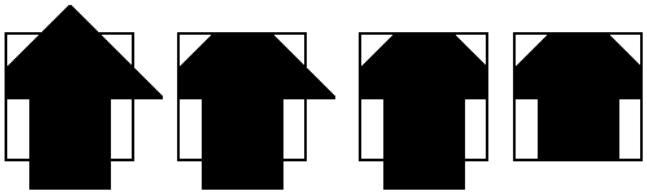
Sutherland - Hodgman Polygon Clipping

The Sutherland - Hodgman algorithm performs a clipping of a polygon against each window edge in turn. It accepts an ordered sequence of verices v1, v2, v3, ..., vn and puts out a set of vertices defining the clipped polygon. Beginning with the initialset 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 be successively passed to a right boundary clipper, a bottom boundary clipper, and a top boundary clipper, as in Fig. below. At each step, a new sequence of output vertices is generated and passed to the next window boundary clipper.

Before clipping

This figure represents a polygon (the large, solid, upward pointing arrow) before clipping has occurred.

The following figures show how this algorithm works at each edge, clipping the polygon.



[a][[b] [c] [d]

a. Clipping against the left side of the clip window.

b. Clipping against the top side of the clip window.

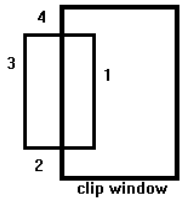
c. Clipping against the right side of the clip window.

d. Clipping against the bottom side of the clip window.

Four Types of Edges

As the algorithm goes around the edges of the window, clipping the polygon, it encounters four types of edges. All four edge types are illustrated by the polygon in the following figure. For each edge type, zero, one, or two vertices are added to the output list of vertices

that define the clipped polygon.

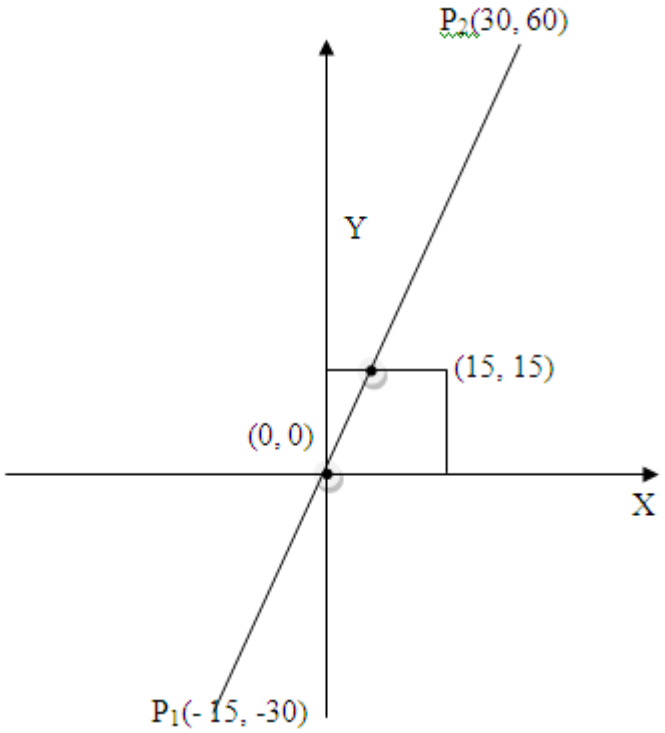


The four types of edges are:
1. Edges that are totally inside the clip window. - add the second inside vertex point
2. Edges that are leaving the clip window. - add the intersection point as a vertex
3. Edges that are entirely outside the clip window. - add nothing to the vertex output list
4. Edges that are entering the clip window. - save the intersection and inside points as vertices

Some Problems with this Algorithm
1. This algorithm does not work if the clip window is not convex.
2. If the polygon is not also convex, there may be some dangling edges.

b. Use the Liang-Barsky line clipping algorithm to clip the line P_1(-15, -30) – P_2(30, 60) against the window having diagonally opposite corners as (0, 0) and (15, 15).

Answer:



Line coordinates P_1(-15, -30), P_2(30,60)
Window coordinates x_{min} = 0, x_{max} = 0, y_{min} = 15, y_{max} = 15.

$dx = 30 – (-15) = 45$ $dy = 60 – (-30) = 90$

$d_1 = -d_x = -45$, $q_1 = x_1 – x_{min} = -15 – 0 = -15$, $u_1 = q_1/d_1 = 1/3$
$d_2 = d_x = 45$, $q_2 = x_{max} – x_1 = 15 – (-15) = 30$, $u_2 = q_2/d_2 = 2/3$
$d_3 = -d_y = - 90$, $q_3 = y_1 – y_{min} = -30 – 0 = -30$, $u_3 = q_3/d_3 = 1/3$
$d_4 = d_y = 90$, $q_4 = y_{max} – y_1 = 15 – (- 30) = 45$, $u_4 = q_4/d_4 = ½$

for $(d_i < 0)$ u_1 = MAXIMUM of (1/3, 1/3, 0) = 1/3
for $(d_i > 0)$ u_2 = MINIMUM of (2/3, ½, 1) = ½

Since $u_1 < u_2$ there is a visible section

Computing new end points

$x'_1 = x_1 + dx \times u_1 = - 15 + (45 \times 1/3) = 0$
$y'_1 = y_1 + dy \times u_1 = - 30 + (90 \times 1/3) = 0$
$x'_2 = x_1 + dx \times u_2 = - 15 + (45 \times 1/2) = 7\ ½$
$y'_2 = y_2 + dy \times u_2 = - 30 + (90 \times 1/2) = 15$

Q.7 a. Explain, why there is a need for visible surface detection? Differentiate between object precision and image precision methods for detecting visible surface.

Answer:

The surfaces that are blocked or hidden from view must be "removed" in order to construct a realistic view of the 3D scene. The identification and removal of these surfaces is called the visible surface detection or hidden – surface removal problem. The solution involves the determination of the closest visible surface along each projection line.
There are many different visible –surface detection algorithms. Each can be characterized as either an image-space method or an object-space method. An object-space method compares objects and parts of objects to each other within the scene definition to determine which surfaces, as a whole as visible. Surface visibility is determined using continuous models in the object space (or its transformation) without involving pixel based operations.
In an image-space algorithm, visibility is decided point by point at each pixel position on the projection plane. Most visible-surface algorithms use image-space methods in which the pixel grid is used to guide the computational activities that determine visibility at the pixel level.

b. Describe Depth Buffer hidden surface elimination method.

Answer: Refer Article 9.3 of Text Book-I

Q.8 a. What are the various input mode functions that specify how the program and input devices interact?

Answer:

Graphical input functions can be set up to allow users to specify the following options:

- Which physical devices are to provide input within a particular logical classification (for example, a tablet used as a stroke device).
- How the graphics program and devices are to interact (input mode). Either the program or the devices can initiate data entry, or both can operate simultaneously.
- When the data are to be input and which device is to be used at that time to deliver a particular input type to the specified data variables.

Input Modes

Functions to provide input can be structured to operate in various input modes, which specify how the program and input devices interact. Input could be initiated by the program, or the program and input devices both could be operating simultaneously, or data input could be initiated by the devices. These three input modes are referred to as request mode, sample mode, and event mode.

In **request mode**, the application program initiates data entry. Input values are requested and processing is suspended until the required values are received. This input mode corresponds to typical input operation in a general programming language. The program and the input devices operate alternately. Devices are put into a wait state until an input request is made; then the program waits until the data are delivered.

In **sample mode**, the application program and input devices operate independently. Input devices may be operating at the same time that the program is processing other data. New input values from the input devices are stored, replacing previously input data values. When the program requires new data, it samples the current values from the input devices.

In **event mode**, the input devices initiate data input to the application program. The program and the input devices again operate concurrently, but now the input devices deliver data to an input queue. All input data are saved. When the program requires new data, it goes to the data queue.

Any number of devices can be operating at the same time in sample and event modes. Some can be operating in sample mode, while others are operating in event mode. But only one device at a time can be providing input in request mode.

An input mode within a logical class for a particular physical device operating on a specified workstation is declared with one of six input-class functions of the form

set . . . Mode (ws, deviceCode, inputMode, echoFlag)

where deviceCode is a positive integer; inputMode is assigned one of the values: *request, sample,* or *event;* and parameter echoFlag is assigned either the value *echo* or the value *noecho*.

b. Describe the various techniques that can be incorporated into graphics packages to aid the interactive construction of pictures.

Answer:

There are several techniques that are incorporated into graphics packages to aid the interactive construction of pictures. Various input options can be provided, so that coordinate information entered with locator and stroke devices can be adjusted or interpreted according to a selected option. For example, we can restrict all lines to be either horizontal or vertical. Input coordinates can establish the position or boundaries for objects to be drawn, or they can be used to rearrange previously displayed objects.

Basic Positioning Methods

Coordinate values supplied by locator input are often used with positioning methods to specify a location for displaying an object or a character string. We interactively select coordinate positions with a pointing device, usually by positioning the screen cursor. Just how the object or text-string positioning is performed depends on the selected options. With a text string, for example, the screen point could be taken as the center string position, or the start or end position of the string, or any of the other string-positioning options. For lines, straight line segments can be displayed between two selected screen positions.

As an aid in positioning objects, numeric values for selected positions can be echoed on the screen. Using the echoed coordinate values as a guide, we can make adjustments in the selected location to obtain accurate positioning.

Constraints

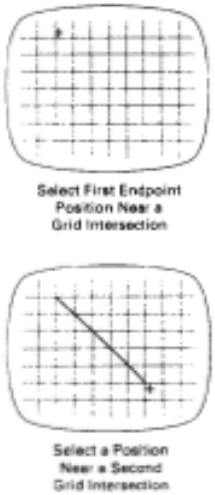
With some applications, certain types of prescribed orientations or object alignments are useful. **A** constraint is a rule for altering input-coordinate values to produce a specified orientation or alignment of the displayed coordinates. There are many kinds of constraint functions that can **be** specified, but the most common constraint is a horizontal or vertical alignment **of** straight lines. This type of constraint is useful in forming network layouts. With this constraint, we can create horizontal and vertical lines without worrying about precise specification of endpoint coordinates.

A horizontal or vertical constraint is implemented by determining whether any two input coordinate endpoints are more nearly horizontal or more nearly vertical. If the difference in the y values of the two endpoints is smaller than the difference in ***x*** values, a horizontal line is displayed. Otherwise, a vertical line is drawn. Other kinds of constraints can be applied to input coordinates to produce a variety of alignments. Lines could be constrained to have a particular slant, such as $\mathbf{45^o}$**,** and input coordinates could be constrained to lie along predefined paths, such as circular arcs.

Grids

Another kind of constraint is a grid of rectangular lines displayed in some part of the screen area. When a grid is used, any input coordinate position is rounded to the nearest intersection of two **grid** lines. Figure below illustrates line drawing with a grid. Each of the two cursor positions is shifted to the nearest grid intersection point, and the line is drawn between these grid points. Grids facilitate object constructions, because a new line can be joined easily to a previously drawn line by selecting any position near the endpoint grid intersection of one end of the displayed line.

Spacing between grid lines is often an option that can he set by the user. Similarly, grids can be turned on and off, and it is sometimes possible to use partial grids and grids of different sizes in different screen areas.



Line drawing using a grid

Gravity Field

In the construction of figures, we sometimes need to connect lines at position between endpoints. Since exact positioning of the screen cursor at the connecting point can be difficult, graphics packages can be designed to convert any input position near a line to a position on the line.

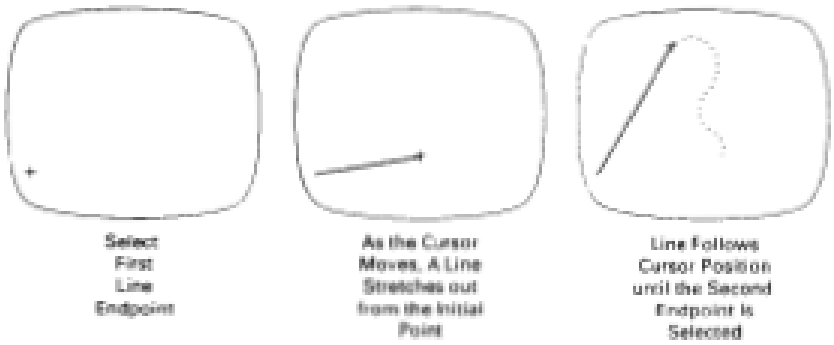
This conversion of input position is accomplished by creating a ***gravity field*** area around the line. Any selected position within the gravity field of a line is moved ("gravitated") to the nearest position on the line. **A** gravity field area around a line is illustrated with the shaded boundary shown in Figure below. Areas shaded area to a around the endpoints are enlarged to make it easier for us to connect lines at their endpoints. Selected positions in one of the circular areas of the gravity field are attracted to the endpoint in that area. The size of gravity fields is chosen large enough to aid positioning, but small enough to reduce chances of overlap with other lines. If many lines are displayed, gravity areas can overlap, and it may be difficult to specify points correctly. Normally, the boundary for the gravity field is not displayed.

Gravity field around a line

Rubber-Band Method.

Straight lines can be constructed and positioned using ***rubber-band*** methods, which stretch out a line from a starting position as the screen cursor is moved. Figure below demonstrates the rubber-band method. We first select a screen position for one endpoint of

the line. Then, as the cursor moves around, the line is displayed from the start position to the current position of the cursor. When we finally select a second screen position, the other line endpoint is set.
Rubber-band methods are used to construct and position other objects besides straight lines.



Rubber-band method for drawing and positioning a straight line segment

Dragging
A technique that is often used in interactive picture construction is to move objects into position by dragging them with the screen cursor. We first select an object, and then move the cursor in the direction we want the object to move, and the selected object follows the cursor path. Dragging objects to various positions in a scene is useful in applications where we might want to explore different possibilities before selecting a final location.

Painting and Drawing
Options for sketching, drawing, and painting come in a variety of forms. Straight lines, polygons, and circles can be generated with methods discussed in the previous sections. Curve drawing options can be provided using standard curve shapes, such as circular arcs and splines, or with freehand sketching procedures. Splines are interactively constructed by specifying a set of discrete screen points that give the general shape of the curve. Then the system fits the set of points with a polynomial curve. In freehand drawing, curves are generated by following the path of a stylus on a graphics tablet or the path of the screen cursor on a video monitor. Once a curve is displayed, the designer can alter the curve shape by adjusting the positions of selected points along the curve path.

Q.9 a. What do you mean by computer-assisted animation? Differentiate it with computer-generated animation.

Answer:

There are 2 main categories of computer animation

- Computer-assisted animation
- Computer-generated animation

Computer-assisted animation usually refers to 2D and 21/2Dsystems that computerizes the traditional animation process. Interpolation between key shapes is typically the only algorithmic use of computer in the production of this type of animation. Computers

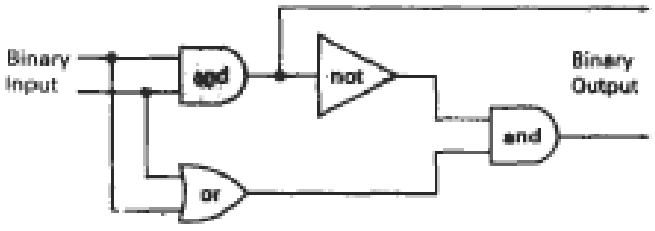
drastically reduce the time taken to draw intermediate frames and simplify artwork and skill requirements. Knowing the key frames, the intermediate frames can be automatically calculated by mathematical programming in computer. This is done by specifying the number of in-between frames interpolated frames required and the end frames. This method of interpolation can be linear or curvilinear to depict natural movement or for simulating accelerations or any simulating movements.

Computer-generated animation is basically concerned with motion control of objects. One of the major time consuming works of an artist of early animators is painting the sequence of frames drawn. This work can be automatically done with computer-generated frames. The method is to use seed fill algorithms and texturing & shading methods. Digital audio & video effects can be amalgamated with animation. Duplication of videos, mass reproduction, distribution, broadcasting, transfer and conversion between different video formats and data exchange have become processes that are completed in a matter of days with computer generated animation.

b. Describe the methods for specifying the information needed to construct and manipulate a model.

Answer:

There are two methods for specifying the information needed to construct and manipulate a model. One method is to store the information in a data structure**,** such as a table or linked list. The other method is to specify the information in procedures. In general, a model specification will contain both data structures and procedures, although some models are defined completely with data structures and others use only procedural specifications. An application to perform solid modeling of objects might use mostly information taken from some data structure to define coordinate positions, with very few procedures. A weather model, on the other hand, may need mostly procedures to calculate plots of temperature and pressure variations.
As an example of how combinations of data structures and procedures can be used, we consider some alternative model specifications for the logic circuit of Figure below.



Model of a logic circuit

One method is to define the logic components in a data table with processing procedures used to specify how the network connections are to be made and how the circuit operates. Geometric data in this table include coordinates and parameters necessary for drawing and

positioning the gates. These symbols could all be drawn as polygon shapes, or they could be formed as combinations of straight-line segments and elliptical arcs. Labels for each of the component parts also have been included in the table, although the labels could be omitted if the symbols are displayed as commonly recognized shapes. Procedures would then be used to display the gates and construct the connecting lines, based on the coordinate positions of the gates and a specified order for connecting them. An additional procedure is used to produce the circuit output (binary values) for any given input. This procedure could be set up to display only the final output, or it could be designed to display intermediate output values to illustrate the internal functioning of the circuit.

A data table defining the structure and position of each gate in the circuit of figure

Symbol Code	Geometric Description	Identifying Label
Gate 1	(Coordinates and other parameters)	and
Gate 2	:	or
Gate 3	:	not
Gate 4	:	and

Alternatively, we might specify graphical information for the circuit model in data structures. The connecting lines, as well as the sates, could then be defined in a data table that explicitly lists endpoints for each of the lines in the circuit. A single procedure might then display the circuit and calculate the output. At the other extreme, we could completely define the model in procedures, using no external data structures.

TEXT BOOK

I. Computer Graphics with OpenGL, 3rd Edition, Donad Hearn, M. Pauline Baker, Pearson Education, 2009