COMPUTER GARPHICS

(PGDCA08)

(PG - DIPLOMA)

ACHARYA NAGARJUNA UNIVERSITY

CENTRE FOR DISTANCE EDUCATION

NAGARJUNA NAGAR,

GUNTUR

ANDHRA PRADESH

UNIT – IV

THREE DIMENSIONAL GEOMETRIC AND MODELLING TRANSFORMATIONS

Objectives

After going through this unit, you should be able to:

- Describe the basic transformations for 3-D translation, rotation, scaling and shearing;
- Describe the other transformations;
- Discuss the composite transformations;
- Describe Modeling and coordinate transformations;
- Define parallel projection, perspective projection.
- Describe Visible surface detection methods

4.1 Introduction

Two-dimensional methods for geometric transformations and object modeling are extended by including the considerations for the ***z*** coordinate to obtain three dimensional methods. Translation, rotation, and scaling are done by including the third dimension.

In two-dimensional rotations, the *xy* plane, we needed to consider only rotations about axes that were perpendicular to the *xy* plane.

In three-dimensional space, we can now select any spatial orientation for the rotation axis. Most graphics packages handle three-dimensional rotation as a composite of three rotations, one for each of the three Cartesian axes. Alternatively, a user can easily set up a general rotation matrix, given the orientation of the axis and the required rotation angle.

4.2 3D Geometric and Modeling Transformations

4.2.1 Translation:

In a three-dimensional homogeneous coordinate representation, a point is translated from position P = ***(x, y, z)*** to position P' = ***(x', y',*** z') with the matrix operation

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Eq 4-1

or

$$\mathbf{P}' = \mathbf{T} \cdot \mathbf{P}$$

Eq 4-2

Parameters t_x t_y, and t_z, specifying translation distances for the coordinate directions ***x, y,*** and z, are asslgned any real values.

The matrix representation in Eq.4-1 is equivalent to the three equations

$$x' = x + t_x, \qquad y' = y + t_y, \qquad z' = z + t_z$$

Eq 4-3

Old coordinates: (x, y, z)

New coordinates: *(x ',* y', z') = (x + t_x, y + t_y, z+t_z)

Where t_x is distance in x direction

t_y is distance in y direction

t_z is distance in z direction

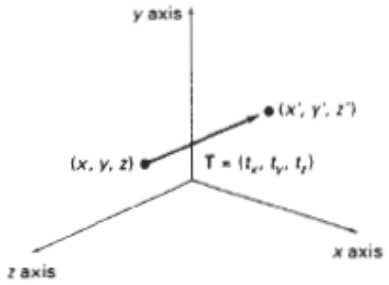


Figure 4.1

Translating a point with translation vector **T.**

4.2.2 Rotation:

To generate a rotation transformation for an object, we must designate an axis of rotation (about which the object is to be rotated) and the amount of angular rotation. Unlike two-dimensional applications, where all transformations are carried out in the **xy** plane, a three-dimensional rotation can be specified around any line in space.

The easiest rotation axes to handle are those that are parallel to the coordinate axes. Also, we can use combinations of coordinate axis rotations (along with appropriate translations) to specify any general rotation. By convention, positive rotation angles produce counterclockwise rotations about a coordinate axis, if we are looking along the positive half of the axis toward the coordinate origin (Fig. 4-2).

Coordinate-Axes Rotations:

The two-dimensional z-axis rotation equations are easily extended to three dimensions:

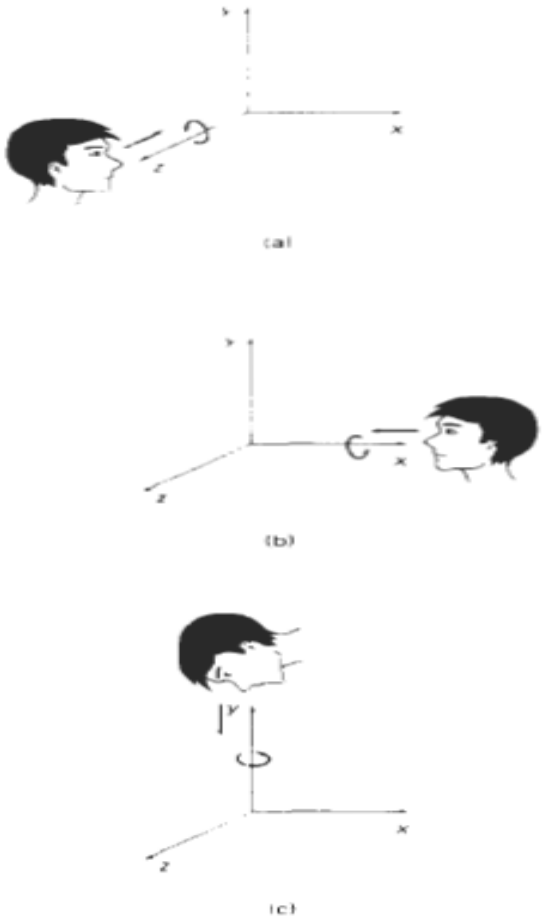


Figure 4-2 Positive rotation directions about coordinate axes are counter clock wise when looking toward the origin from a positive coordinate position on each axis.

Old coordinates: (x, y, z)

New coordinates: *(x ',* y', *z')*

Rotation angle: Θ

Rotation is of three types:

- Rotation over z-axis
- Rotation over x-axis
- Rotation over y-axis

Rotation over z-axis:

$$x' = x\cos\theta - y\sin\theta$$
$$y' = x\sin\theta + y\cos\theta$$
$$z' = z$$

Eq: 4-4

In homogeneous coordinate form, the three-dimensional z-axis rotation equations are expressed as

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

Eq: 4-5

Which can be written as

$$\mathbf{P}' = \mathbf{R}_z(\theta) \cdot \mathbf{P}$$

Eq: 4-6

Rotation over x,y-axes:

Transformation equations for rotations about the other two coordinate axes can be obtained with a cyclic permutation of the coordinate parameters x, y and z in Eq 4-4. ie; we use replacements as:

$$x \rightarrow y \rightarrow z \rightarrow x$$

Eq: 4-7

Rotation over x-axis:

$$y' = y\cos\theta - z\sin\theta$$

$$z' = y\sin\theta + z\cos\theta$$

$$x' = x$$

Eq: 4-8

In homogeneous coordinate form, the three-dimensional x-axis rotation equations are expressed as

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

Eq: 4-9

Which can be written as

$$\mathbf{P}' = \mathbf{R}_x(\theta) \cdot \mathbf{P}$$

Eq: 4-10

Rotation over y-axis:

$$z' = z\cos\theta - x\sin\theta$$

$$x' = z\sin\theta + x\cos\theta$$

$$y' = y$$

Eq: 4-11

In homogeneous coordinate form, the three-dimensional z-axis rotation equations are expressed as

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

Eq: 4-12

Which can be written as

$$\mathbf{P}' = \mathbf{R}_y(\theta) \cdot \mathbf{P}$$

Eq: 4-13

4.2.3 Scaling:

The matrix expression tor the scaling transformation of a position P(x, y, *z)* relative to the coordinate origin can be written as:

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

Eq: 4-14

$$\mathbf{P}' = \mathbf{S} \cdot \mathbf{P}$$

Eq: 4-15

where scaling parameters s_x, s_y, and s_z, are assigned any positive values. Explicit expressions for the coordinate transformations for scaling relative to the origin are:

$$x' = x \cdot s_x, \qquad y' = y \cdot s_y, \qquad z' = z \cdot s_z$$

Eq:4-16

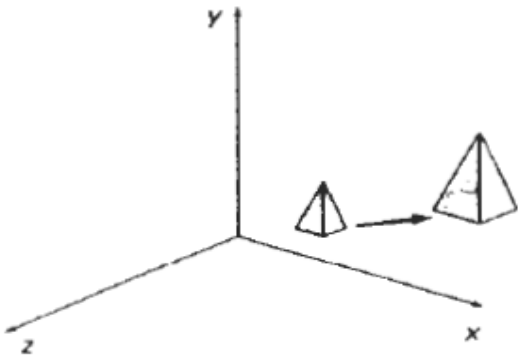


Figure: 4-3

Doubling the size of an object with transformation 11-42 also moves the object farther from the origin.

Scaling with respect to a selected fixed position(x, y, z) can be represented the following transformation sequence:

1. Translate the fixed point to the origin.

2. Scale the object relative to the coordinate origin (Eq. 4-14)

3. Translate the fixed point back to its original position.

The matrix representation for an arbitrary fixed-point scaling can then be expressed as the concatenation of these translate-scale-translate transformations as:

$$T(x_f, y_f, z_f) \cdot S(s_x, s_y, s_z) \cdot T(-x_f, -y_f, -z_f) = \begin{bmatrix} s_x & 0 & 0 & (1-s_x)x_f \\ 0 & s_y & 0 & (1-s_y)y_f \\ 0 & 0 & s_z & (1-s_z)z_f \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Eq: 4-17

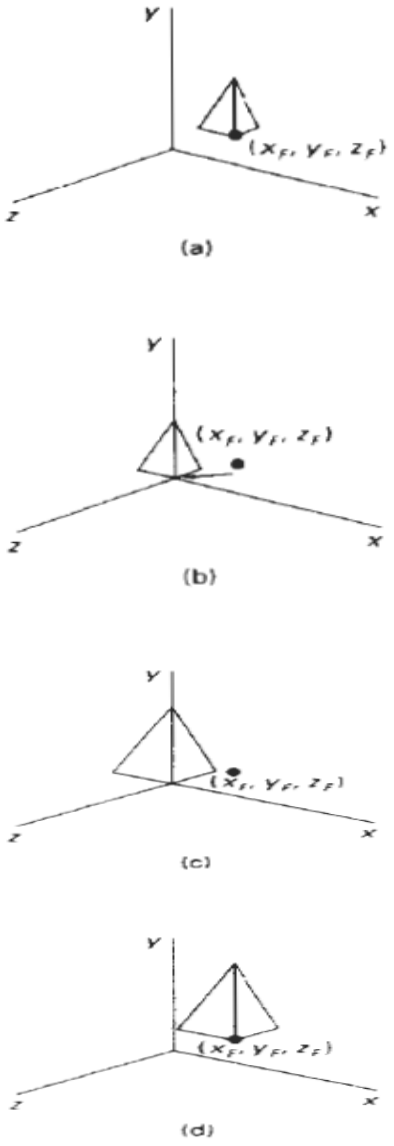


Figure: 4-4: Scaling an object

4.3 Other Transformations

In addition to translation, rotation, and scaling, there are various additional transformations that are often useful in three-dimensional graphics applications. Two of these are reflection and shear.

4.3.1 Reflection:

A three-dimensional reflection can be performed relative to a selected reflection axis or with respect to a selected reflection plane.

In general, three-dimensional reflection matrices are set up similarly to those for two dimensions. Reflections relative to a given axis are equivalent to 180 degree rotations about that axis.

Reflections with respect to a plane are equivalent to 160 degree rotations in four-dimensional space. When the reflection plane is a coordinate plane (xy, xz, or yz), we can think of the transformation as a conversion between Left-handed and right-handed systems.

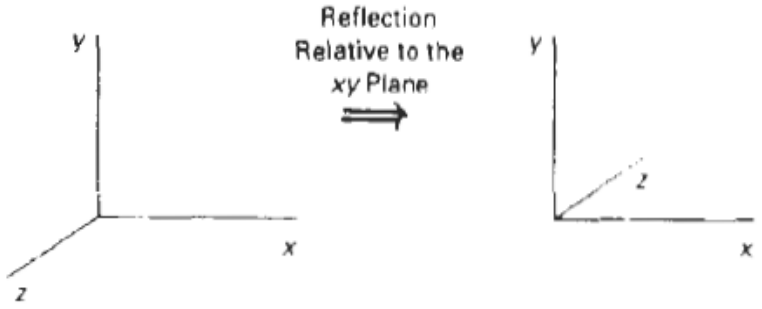


Figure: 4-5: Conversion of coordinate specifications from a right handed to a left-handed system by reflection

An example of a reflection that converts coordinate specifications from a right-handed system to a left-handed system (or vice versa) is shown in Fig.4.5. This transformation changes the sign of the z coordinates, Leaving the x and y-coordinate values unchanged. The matrix representation for this reflection of points relative to the xy plane is:

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

Eq: 4-18

4.3.2 Shears:

Shearing transformations can he used to modify object shapes. They are also useful in three-dimensional viewing for obtaining general projection transformations.

In two dimensions, we discussed transformation respective to the x or y axes to produce distortions in the shapes of objects. In three dimensions, we can also generate shears relative to the *z* axis.

As an example of three-dimensional shearing the following transformation produces a z-axis shear:

$$SH_z = \begin{bmatrix} 1 & 0 & a & 0 \\ 0 & 1 & b & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Eq:4-19

Parameters a and *b* can be assigned any real values. The effect of this transformation matrix is to alter x- and y-coordinate values by an amount that is proportional to the z value, while leaving the z coordinate unchanged. Boundaries of

planes that are perpendicular to the z axis are thus shifted by an amount proportional to z.

An example of the effect of this shearing matrix on a unit cube is shown in Fig. 4-6, for shearing values a = *b* = 1. Shearing matrices for the r axis and y axis are defined similarly.

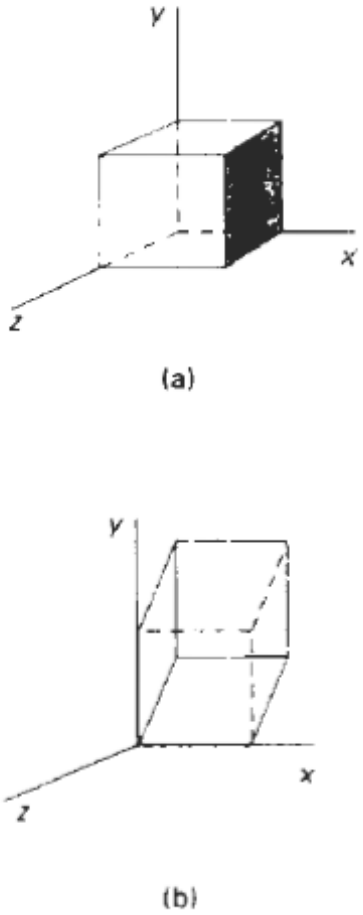


Figure: 4-6

A unit cube (a) sheared (b) by transformation

4.4 Composite Transformations

As with two-dimensional transformations, we form a composite three-dimensional transformation by multiplying the matrix representations for the individual operations in the transformation sequence.

This concatenation is carried out from right to left, where the rightmost matrix is the first transformation to be applied to an object and the leftmost matrix is the last transformation. A sequence of basic, three-dimensional geometric transformations are combined to produce a single composite transformation, which is then applied to the coordinate definition of an object

4.5 Modeling and coordinate transformations:

General three-dimensional viewing procedures, for example involve an initial transformation of world-coordinate descriptions to a viewing-coordinate system. Then viewing coordinates are transformed to device coordinates. And in modeling, objects are often described in a local (modeling) coordinate reference frame, and then the objects are repositioned into a world-coordinate scene.

Modeling transformation functions can be applied to create hierarchical representation for three-dimensional objects. We can define three-dimensional object shapes in local (modeling) coordinates, and then we construct a scene or a hierarchical representation with instances of the individual objects. That is, we transform object descriptions from modeling coordinates to world coordinates or to another system in the hierarchy.

Coordinate descriptions of objects are transferred from one system to another

With the same procedures used to obtain two-dimensional coordinate transformations. We need to set up the transformation matrix that brings the two coordinate systems into alignment. First, we set up a translation that brings the

new coordinate origin to the position of the other coordinate origin. This is followed by a sequence of rotations that corresponding coordinates axes. If different scales are used in the two coordinate systems, a scaling transformation may also be necessary to compensate or the differences in coordinate intervals.

If a second coordinate system is defined with origin (x_0, y_0, z_0) and unit axis vectors as shown in Fig. 4-7, relative to an existing Cartesian reference frame, we first construct the translation matrix T(x_0, y_0, z_0). Next, we can use the unit axis vectors to form the coordinate rotation matrix which transforms unit vectors u_x', u_y' and u_z' onto the x, y, and z axes, respectively.

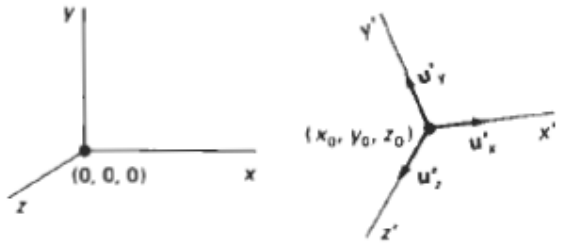


Figure:4-7

Transformation of object description from one system to other system

The complete coordinate-transformation sequence is then given by the composite matrix R . T. This matrix correctly transforms coordinate descriptions from one Cartesian system to another even if one system is left-handed and the other is right-handed.

$$\mathbf{R} = \begin{bmatrix} u'_{x1} & u'_{x2} & u'_{x3} & 0 \\ u'_{y1} & u'_{y2} & u'_{y3} & 0 \\ u'_{z1} & u'_{z2} & u'_{z3} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Eq: 4-20

4.6 Projections:

Once world-coordinate descriptions of the objects in a scene are converted to viewing coordinates, we can project the three-dimensional objects onto the tri-dimensional view plane.

There are two basic projection methods:

- Parallel projection
- Perspective projection

In a parallel projection, coordinate positions are transformed to the vied plane along parallel lines, as shown in the example of Fig. 4-8

For a perspective projection (Fig.4-9), object positions are transformed to the view plane along lines that converge to a point called the projection reference point (or center of projection).

The projected view of an object is determined calculating the intersection of the projection lines with the view plane.

A parallel projection preserves relative proportions of objects, and this is the method used in drafting to produce scale drawings of three-dimensional objects. Accurate views of the various sides of an object are obtained with a parallel projection, but this does not give us a realistic representation of the appearance of a three-dimensional object.

A perspective projection, on the other hand, produces realistic views but does not preserve relative proportions.

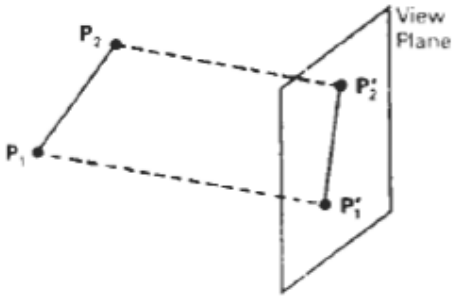


Figure: 4-8 parallel projection

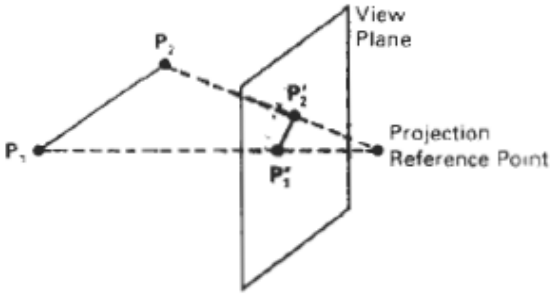


Figure: 4-9 perspective projection

4.7 Visible surface detection methods:

Visible-surface detection algorithms are broadly classified according to whether they deal with object definitions directly or with their projected images.

These two approaches are called object-space methods and image-space methods, respectively.

An object-space method compares objects and parts of objects to each other within the scene definition to determine which surfaces, as a whole, we should label as visible.

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, although object space methods can be used effectively to locate visible surfaces in some cases.

4.8 Classification of visible-surface detection Methods:

Line display algorithms, on the other hand, generally use object-space methods to identify visible lines in wireframe displays, but many image-space visible-surface algorithms can be adapted easily to visible-line detection. Although there are major differences in the basic approach taken by the various visible-surface detection algorithms, most use sorting and coherence methods to improve performance.

Sorting is used to facilitate depth comparisons by ordering the individual surfaces in a scene according to their distance from the view plane.

An individual scan line can be expected to contain intervals (runs) of con- Back-Face Detection constant pixel intensities, and scan-line patterns often change little from one line to the next. Animation frames contain changes only in the vicinity of moving objects. And constant relationships often can be established between objects and surfaces in a scene.

Back-face detection:

A fast and simple object-space method for identifying the back faces of a polyhedron is based on the "inside-outside" tests discussed in Chapter 10. A point (x,y,z) is "inside" a polygon surface with plane parameters A, B, C, and D if

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

When an inside point is along the line of sight to the surface, the polygon must be a back face. 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. 4-10 then this polygon is a back face if

$$\mathbf{V} \cdot \mathbf{N} > 0$$

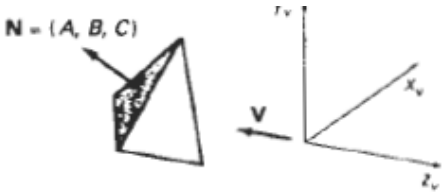
Vector **V** in the viewing direction and a back-face normal vector **N** of a polyhedron

Figure: 4-10

Furthermore, if object descriptions have been converted to projection coordinates and our viewing direction is parallel to the viewing z_p. axis, then V = $(0,0,V_z)$ and

$$\mathbf{V} \cdot \mathbf{N} = V_z C$$

so that we only need to consider the sign of C, the ; component of the normal vector N



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.

Figure: 4-11

In a right-handed viewing system with viewing direction along the negative z_p, axis (Fig. 4-11), 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:

$$C \leq 0$$

4.8.1 Depth-buffer method:

A commonly used image-space approach to detecting visible surfaces is the depth-buffer method, which compares surface depths at each pixel position on the projection plane. This procedure is also referred to as the z-buffer method, since object depth is usually measured from the view plane along the z axis of a viewing system. Each surface of a scene is processed separately, one point at a time across the surface.

The method is usually applied to scenes containing only polygon surfaces, because depth values can be computed very quickly and the method is easy to implement. But the method can be applied to non-planar surfaces. With object descriptions converted to projection coordinates, each (x, y, z) position on a polygon surface corresponds to the orthographic projection point *(x,* y) on the view plane.

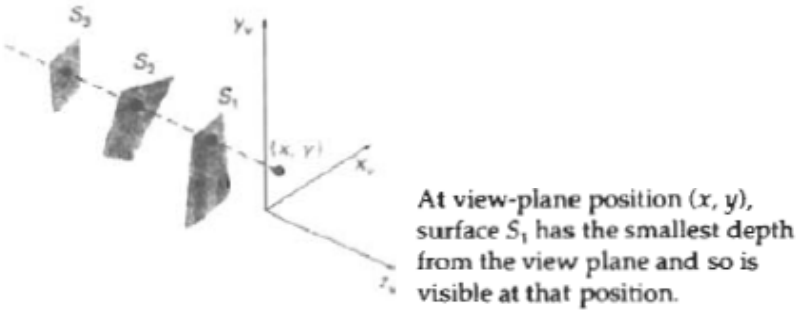


Figure 4-12

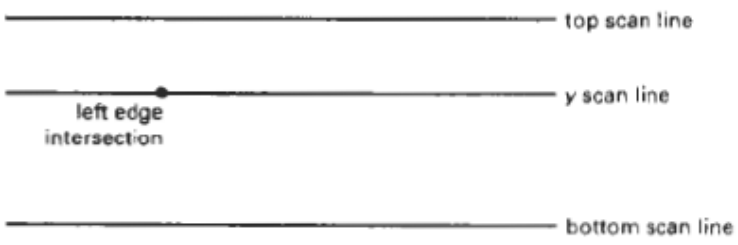
Therefore, for each pixel position (x, y) on the view plane, object depths can be compared by comparing z values. Figure 4-12 shows three surfaces at varying distances along the orthographic projection line from position (1,y) in a view plane taken as the x-plane. Surface 5, is closest at this position, so its surface intensity value at (x, y) is saved.

4.8.2 Scan-line method:

This image space method for removing hidden surface 5 is an extension of the scan-line algorithm for tilling polygon interiors. Instead of filling just one surface, we now deal with multiple surfaces. As each scan line is processed, all polygon surfaces intersecting that line are examined to determine which are visible. Across each scan line, depth calculations are made for each overlapping surface to determine which is nearest to the view plane. When the visible surface has been determined, the intensity value for that position is entered into the refresh buffer.

The polygon table contains coefficients of the plane equation for each surface, intensity information for the surfaces, and possibly pointers into the edge table. To facilitate the search for surfaces crossing an even scan line, we can set up an active list of edges from information in the edge table. This active list will contain only edges that cross the current

scan line, sorted in order of increasing ***x.*** In addition, we define a flag for each surface that is set on or off to indicate whether a position along a scan line is inside or outside of the surface. Scan lines are processed from left to right (fig: 4-13). At the leftmost boundary of a surface, the surface flag is turned on; and at the rightmost boundary, it is turned off.



Scan lines intersecting a polygon surface.

Figure : 4-13

4.8.3 Depth-sorting method:

Using image spaces and object-space operations, the depth-sorting method performs the following basic functions:

1. Surfaces are sorted in order of decreasing depth.

2. Surfaces are scan converted in order, starting with the surface of greatest depth.

Sorting operations are carried out in both image and object space, and the scan conversion of the polygon surfaces is performed in image space.

This method for solving the hidden-surface problem is often referred to as the painter's algorithm. In creating an oil painting, an artist first paints the background colors.

Next, the most distant objects are added, then the nearer objects, and so forth. At the final step, the foreground objects are painted on the canvas over the background and other objects that have been painted on the canvas. Each layer of paint covers up the previous layer. Using a similar technique, first sort surfaces according to their distance from the view plane. The intensity Depth-Sorting Method values for the farthest surface are then entered into the refresh buffer. Taking each succeeding surface in horn (in decreasing depth order), we "paint" the surface intensities onto the frame buffer over the intensities of the previously processed surfaces.

Painting polygon surfaces onto the frame buffer according to depth is carried out in several steps. Assuming we are viewing along the-z direction, surfaces are ordered on the first pass according to the smallest ***z*** value on each surface. Surface 5 with the greatest depth is then compared to the other surfaces in the list to determine whether there are any overlaps in depth.

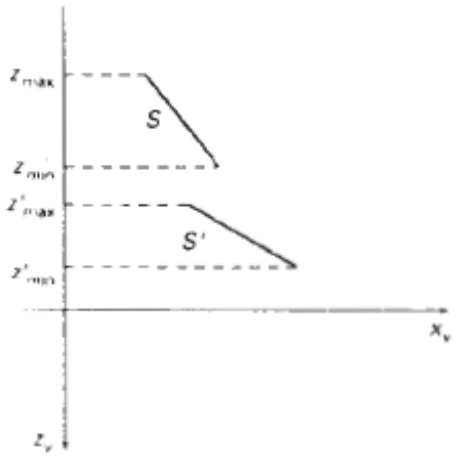


Figure: 4-14

Two surfaces with no depth overlap

If no depth overlaps occur, 5 is scan converted. Figure 4-14 shows two surfaces that overlap in the xy plane but have no depth overlap. This process is then repeated for the next surface in the list. As long as no overlaps occur, each surface is processed in depth order until all have been scan converted. If a depth overlap is detected at any point in the list, we need to make some additional comparisons to determine whether any of the surfaces should be reordered.

We make the following tests for each surface that overlaps with 5. If any one of these tests is true, no reordering is necessary for that surface. The tests are listed in order of increasing difficulty.

1. The bounding rectangles in the xy plane for the two surfaces do not overlap

2. Surface 5 is completely behind the overlapping surface relative to the viewing position.

3. The overlapping surface is completely in front of 5 relative to the viewing position.

4. The projections of the two surfaces onto the view plane do not overlap.

We perform these tests in the order listed and proceed to the next overlapping surface as soon as we find one of the tests is true. If all the overlapping surfaces pass at least one of these tests, none of them is behind 5. No reordering is then necessary and *S* is scan converted.

Test1 is performed in two parts. We first check for overlap in the x direction, and then we check for overlap in the y direction. If either of these directions shows no overlap, the two planes cannot obscure one other.

4.8.4 Area-subdivision method:

This technique for hidden-surface removal is essentially an image-space method, but object-space operations can be used to accomplish depth ordering of surfaces. The area-subdivision method takes advantage of area coherence in a scene by locating those view areas that represent part of a single surface. We apply this method by successively dividing the total viewing area into smaller and smaller rectangles until each small area is the projection of part of n single visible surface or no surface at all.

To implement this method, we need to establish tests that can quickly identify the area as part of a single surface or tell us that the area is too complex to analyze easily. Starting with the total view, we apply the tests to determine whether we should subdivide the total area into smaller rectangles. If the tests indicate that the view is sufficiently complex, we subdivide it.

Next, we apply the tests to each of the smaller areas, subdividing these if the tests indicate that visibility of a single surface is still uncertain. We continue this process until the subdivisions are easily analyzed as belonging to a single surface or until they are reduced to the size of a single pixel. An easy way to do this is to successively divide the area into four equal parts at each step, as shown in Fig. 4-15.

This approach is similar to that used in constructing a quadtree. A viewing area with a resolution of 1024 by 1024 could be subdivided ten times in this way before a subarea is reduced to a pint.

Tests to determine the visibility of a single surface within a specified area are made by comparing surfaces to the boundary of the area. There are four possible relationships that a surface can have with a specified area boundary.

We can describe these relative surface characteristics in the following:

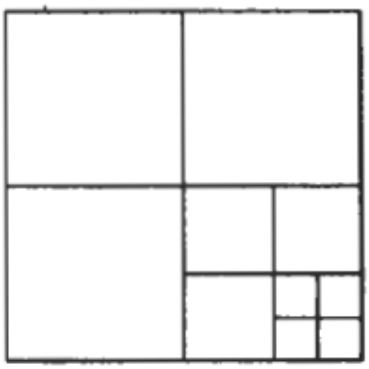
- Surrounding surface-One that completely encloses the area.
- Overlapping surface-One that is partly inside and partly outside the area.
- Inside surface-One that is completely inside the area.
- Outside surface-One that is completely outside the area.

The tests for determining surface visibility within an area can be stated in terms of these four classifications. No further subdivisions of a specified area are needed if one of the following conditions is true:

1. All surfaces are outside surfaces with respect to the area.

2. Only one inside, overlapping, or surrounding surface is in the area.

3. A surrounding surface obscures all other surfaces within the area boundaries.



Dividing a square area into equal-sized quadrants at each step.

Figure 4-15

Text Book:

Donald Hearn & Pauline M.Baker – Computer Graphics 2nd Edition –PHI.

Reference Book:

1.Foley, Van Dam, Feiner and Hughes – Computer Graphics Principles and Practice 3rd Edition.

2.N.Krishnamurthy - Introduction to Computer Graphics.

CONTENTS

UNIT – I

COMPUTER GRAPHICS FUNDAMENTALS

Objectives

After completing this unit, you should be able to:

- Describe computer graphics, its features and characteristics;
- Discuss applications of computer graphics in various fields,
- Describe various types of hardware, required to work with graphic systems.
- Describe the Line Generation Algorithm;
- Apply Line Generation Algorithm to practical problems;
- Describe the different types of Line Generation Algorithm;
- Describe Circle Generation Algorithm;
- Apply Circle Generation algorithm to practical problems;
- Describe the different types of Circle Generation Algorithms;
- Describe Polygon fill algorithm, and
- Describe Scan Line Polygon fill algorithm.

1.1. Introduction

There are two types of computer graphics: raster graphics, where each pixel is separately defined (as in a digital photograph), and vector graphics, where mathematical formulas are used to draw lines and shapes, which are then interpreted at the viewer's end to produce the graphic. Using vectors results in infinitely sharp graphics and often smaller files, but, when complex, vectors take time to render and may have larger file sizes than a raster equivalent.

A screenshot from the 2007 video game *Crisis* displaying extremely photo-realistic real-time computer graphics

In 1950, the first computer-driven display was attached to MIT's Whirlwind I computer to generate simple pictures. This was followed by MIT's TX-0 and TX-2, interactive computing which increased interest in computer graphics during the late 1950s. In 1962, Ivan Sutherland invented Sketchpad, an innovative program that influenced alternative forms of interaction with computers.

In the mid-1960s, large computer graphics research projects were begun at MIT, General Motors, Bell Labs, and Lockheed Corporation. Douglas T. Ross of MIT developed an advanced compiler language for graphics programming. S.A.Coons, also at MIT, and J. C. Ferguson at Boeing, began work in sculptured surfaces. GM developed their DAC-1 system, and other companies, such as Douglas, Lockheed, and

McDonnell, also made significant developments. In 1968, ray tracing was invented by Apple

During the late 1970s, personal computers became more powerful, capable of drawing both basic and complex shapes and designs. In the 1980s, artists and graphic designers began to see the personal computer, particularly the Commodore Amiga and Macintosh, as a serious design tool, one that could save time and draw more accurately than other methods. 3D computer graphics became possible in the late 1980s with the powerful SGI computers, which were later used to create some of the first fully computer-generated short films at Pixar. The Macintosh remains one of the most popular tools for computer graphics in graphic design studios and businesses.

Modern computer systems, dating from the 1980s and onwards, often use a graphical user interface (GUI) to present data and information with symbols, icons and pictures, rather than text. Graphics are one of the five key elements of multimedia technology.

3D graphics became more popular in the 1990s in gaming, multimedia and animation. In 1996, Quake, one of the first fully 3D games, was released. In 1995, Toy Story, the first full-length computer-generated animation film, was released in cinemas Since then, computer graphics have become more accurate and detailed, due to more advanced computers and better 3D modeling software applications, such as Maya (software), 3D Studio Max, and Cinema 4D.

Another use of computer graphics is screensavers, originally intended to preventing the layout of much-used GUIs from 'burning into' the computer screen. They have since evolved into true pieces of art, their practical purpose obsolete; modern screens are not susceptible to such burn in artifacts.

Web graphics

In the 1990s, Internet speeds increased, and Internet browsers capable of viewing images were released, the first being Mosaic. Websites began to use the GIF format to display small graphics, such as banners, advertisements and navigation buttons, on web pages. Modern web browsers can now display JPEG, PNG and increasingly, SVG images in addition to GIFs on web pages. SVG, and to some extent VML, support in some modern web browsers have made it possible to display vector graphics that are clear at any size. Plugins expand the web browser functions to display animated, interactive and 3-D graphics contained within file formats such as SWF and X3D.

Signature art used on web forums

Modern web graphics can be made with software such as Adobe Photoshop, the GIMP, or Corel Paint Shop Pro. Users of Microsoft Windows have MS Paint, which many find to be lacking in features. This is because MS Paint is a drawing package and not a graphics package.

Numerous platforms and websites have been created to cater to web graphics artists and to host their communities. A growing number of people use create internet forum signatures—generally appearing after a user's post—and other digital artwork, such as photo manipulations and large graphics.

1.1.1. Features/Applications of Computer Graphics

Graphics are visual elements often used to point readers and viewers to particular information. They are also used to supplement text in an effort to aid readers in their understanding of a particular concept or make the concept more clear or interesting. Popular magazines, such as *TIME*, *Wired* and *Newsweek*, usually contain graphic material in abundance to attract readers, unlike the majority of scholarly journals. In computing, they are used to create a graphical interface for the user; and graphics are one of the five key elements of multimedia technology. Graphics are among the primary ways of advertising the sale of goods or services.

Business

Graphics are commonly used in business and economics to create financial charts and tables. The term *Business Graphics* came into use in the late 1970s, when personal computers became capable of drawing graphs and charts instead of using a tabular format. Business Graphics can be used to highlight changes over a period of time.

Advertising

Advertising is one of the most profitable uses of graphics; artists often do advertising work or take advertising potential into account when creating art, to increase the chances of selling the artwork.

Political

The use of graphics for overtly political purposes—cartoons, graffiti, poster art, flag design, etc—is a centuries old practice which thrives today in every part of the world. The Northern Irish murals are one such example.

Education

Graphics are heavily used in textbooks, especially those concerning subjects such as geography, science, and mathematics, in order to illustrate theories and concepts, such as the human anatomy. Diagrams are also used to label photographs and pictures.

Educational animation is an important emerging field of graphics. Animated graphics have obvious advantages over static graphics when explaining subject matter that changes over time.

The *Oxford Illustrated Dictionary* uses graphics and technical illustrations to make reading material more interesting and easier to understand. In an encyclopedia, graphics are used to illustrate concepts and show examples of the particular topic being discussed.

In order for a graphic to function effectively as an educational aid, the learner must be able to interpret it successfully. This interpretative capacity is one aspect of graphicacy.

Film and animation

Computer graphics are often used in the majority of new feature films, especially those with a large budget. Films that heavily use computer graphics include *The Lord of the Rings* film trilogy, the Harry Potter films, Spider-Man and War of the Worlds.

Graphics education

The majority of schools, colleges and universities around the world educate students on the subject of graphics and art.

The subject is taught in a broad variety of ways, each course teaching its own distinctive balance of craft skills and intellectual response to the client's needs.

Some graphics courses prioritize traditional craft skills—drawing, printmaking and typography—over modern craft skills. Other courses may place an emphasis on teaching digital craft skills. Still other courses may downplay the crafts entirely, concentrating on training students to generate novel intellectual responses that engage with the brief. Despite these apparent differences in training and curriculum, the staff and students on any of these courses will generally consider themselves to be graphic designers.

The typical pedagogy of a graphic design (or graphic communication, visual communication, graphic arts or any number of synonymous course titles) will be broadly based on the teaching models developed in the Bauhaus school in Germany or Vkhutemas in Russia. The teaching model will tend to expose students to a variety of craft skills (currently everything from drawing to motion capture), combined with an effort to engage the student with the world of visual culture.

Famous graphic designers

Aldus Manutius designed the first Italic type style which is often used in desktop publishing and graphic design. April Greiman is known for her influential poster design. Paul Rand is well known as a design pioneer for designing many popular corporate logos, including the logo for IBM, NeXT and UPS. William Caslon, during the mid-18th century, designed many typefaces, including *ITC Founder's Caslon*, *ITC Founder's Caslon Ornaments*, *Caslon Graphique*, *ITC Caslon No. 224*, *Caslon Old Face* and *Big Caslon*.

Examples of graphics

Photograph

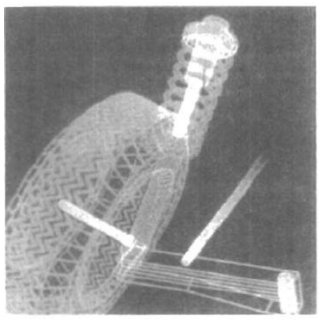
Drawing

Drawing

1.1.2 Computer Aided Design:

A major use of computer graphics is in design processes, particularly for engineering and architectural systems, but almost all products are now computer designed. Generally referred to as CAD, computer-aided design methods are now routinely used in the design of buildings, automobiles, aircraft, watercraft, spacecraft, computers, textiles, and many, many other products. For some design applications; objects are first displayed in a wireframe outline

form that shows the overall shape and internal features of objects. Wireframe displays also allow designers to quickly see the effects of interactive adjustments to design shapes. Following figure give example of wireframe displays in design applications.

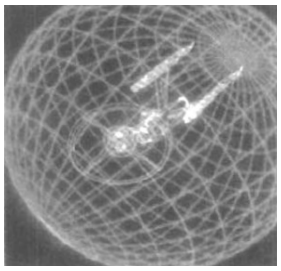


Color-coded wireframe display for an automobile wheel assembly.

1.1.3 Entertainment

Computer graphics methods am now commonly used in making motion pictures, music videos, and television shows. Sometimes the graphics scenes *are* displayed by themselves, and sometimes graphics objects *are* combined with the actors and live scenes.

A graphics scene generated for the movie *Star Trek- The Wrath* of *Khan* is shown in the following figure. The planet and spaceship are drawn in wirefame form and will be shaded with rendering methods to produce solid surfaces.

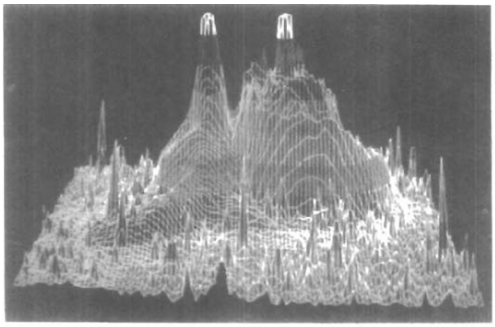


Graphics developed for the Paramount Pictures movie *Star Trek-The* Wrath of *Khan.*

1.1.4 Visualization

Scientists, engineers, medical personnel, business analysts, and others often need to analyze large amounts of information or to study the behavior of certain processes. Numerical simulations carried out on supercomputers frequently produce data files containing thousands and even millions of data values. Similarly, satellite cameras and other sources are amassing large data files faster than they can be interpreted. Scanning these large sets of number to determine trends and relationships is a tedious and ineffective process. But if the data are converted to a visual form, the trends and patterns are often immediately apparent. The following figure shows an example of a large data set that has been converted to

a color-coded display of relative heights above a ground plane. Once we have plotted the density values in this way, we can see easily the overall pattern of the data. Producing graphical representations for scientific, engineering, and medical data sets and processes is generally referred to as *scientific visualization.* And the then busi*ness visualization is used* in connection with data sets related to commerce, industry, and other nonscientific areas.



***A* color-coded plot with 16 million density points of relative brightness observed for the Whirlpool Nebula reveals two distinct galaxies.**

1.1.5 Image Processing:

Although methods used in computer graphics and Image processing overlap, the two areas are concerned with fundamentally different operations. In computer graphics, a computer is used to create a picture. Image processing, on the other hand. applies techniques to modify or interpret existing pictures, such as the photographs and TV scans. Two principal applications of image processing are (1) improving picture quality and (2) machine perception of visual information, as used in robotics. To apply image processing methods, we first digitize a photograph or other picture into an image file. Then digital methods can be applied to rearrange picture parts, to enhance color separations, or to improve the quality of shading. An example of the application of image processing methods to enhance the quality of a picture is shown in the following

figure. These techniques are used extensively in commercials art applications that involve the retouching and rearranging of sections of photographs and other artwork. Similar methods are used to analyze satellite photos of the earth and photos of galaxies.

***A* blurred photograph of a license plate becomes legible after the application of image processing techniques**

1.1.6 Graphical User Interface

It is common now for software packages to provide a graphical interface. A major component of a graphical interface is a window manager that allows a user to display multiple-window areas. Each window can contain a different process that can contain graphical or non graphical displays. To make a particular window active, we simply click in that window using an interactive pointing device. Interfaces also display menus and icons for fast selection of processing options or parameter values. An icon is a graphical symbol that is designed to look like the processing option it represents. The advantages of icons are that they take up less screen space than corresponding textual descriptions and they can be understood more quickly if well designed. Menus contain lists of textual descriptions and icons. The following figure illustrates a typical graphical interface, containing a window manager, menu displays, and icons. In this example, the menus allow selection of processing options, color values, and graphics parameters. The icons

represent options for painting, drawing, zooming, typing text strings, and other operations connected with picture construction.

A graphical user interface, showing multiple window areas, menus, and icons.

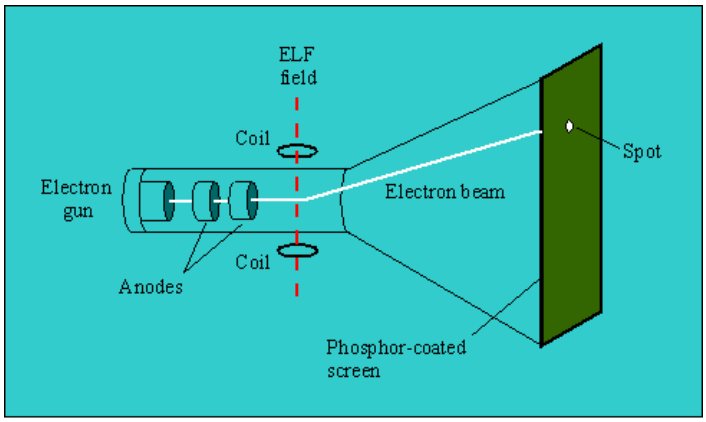
1.2. Overview of Graphics Systems

A display device is an output device for presentation of information for visual, tactile or additive reception, acquired, stored, or transmitted in various forms. When the input information is supplied as an electrical signal, the display is called *electronic display*. *Electronic displays* are available for presentation of visual, tactile and additive information. Tactile electronic displays (aka *refreshable Braille display*) are usually intended for the blind or visually impaired, they use electro-mechanical parts to dynamically update a tactile image (usually of text) so that the image may be felt by the fingers. Common applications for *electronic visual displays* are television sets or computer monitors.

VIDEO DISPLAY DEVICES:

1.2.1. Cathode Ray Tubes

A cathode ray tube (CRT) is a specialized vacuum tube in which images are produced when an electron beam strikes a phosphorescent surface. Most desktop computer displays make use of CRTs. The CRT in a computer display is similar to the "picture tube" in a television receiver. A cathode ray tube consists of several basic components, as illustrated below. The electron gun generates a narrow beam of electrons. The anodes accelerate the electrons. Deflecting coils produce an extremely low frequency electromagnetic field that allows for constant adjustment of the direction of the electron beam. There are two sets of deflecting coils: horizontal and vertical. (In the illustration, only one set of coils is shown for simplicity.) The intensity of the beam can be varied. The electron beam produces a tiny, bright visible spot when it strikes the phosphor-coated screen.



To produce an image on the screen, complex signals are applied to the deflecting coils, and also to the apparatus that controls the intensity of the electron beam. This causes the spot to race across the screen from right to left, and from top to bottom, in a sequence of horizontal lines called the raster. As viewed from the front of the CRT, the spot moves in a pattern

similar to the way your eyes move when you read a single-column page of text. But the scanning takes place at such a rapid rate that your eye sees a constant image over the entire screen.

The illustration shows only one electron gun. This is typical of a monochrome, or single-color, CRT. However, virtually all CRTs today render color images. These devices have three electron guns, one for the primary color red, one for the primary color green, and one for the primary color blue. The CRT thus produces three overlapping images: one in red (R), one in green (G), and one in blue (B). This is the so-called RGB color model.

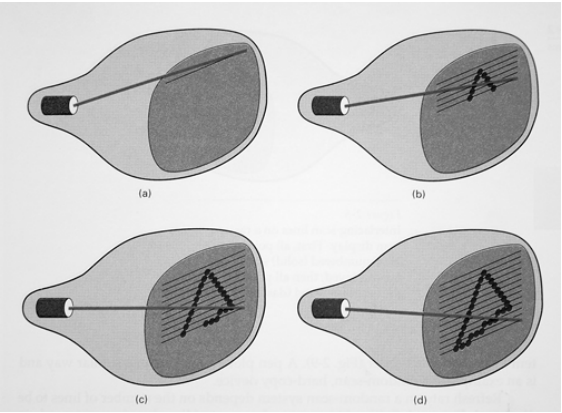
In computer systems, there are several display modes, or sets of specifications according to which the CRT operates. The most common specification for CRT displays is known as SVGA (Super Video Graphics Array). Notebook computers typically use liquid crystal display. The technology for these displays is much different than that for CRTs.

1.2.2 Raster Scan Displays:

In a raster- scan system, the electron beam is swept across the screen, one row at a time from top to bottom. As the electron beam moves across each row, the beam intensity is turned on and off to create a pattern of illuminated spots. Picture definition is stored in memory area called the refresh buffer or frame buffer. This memory area holds the set of intensity values for all the screen points. Stored intensity values are then retrieved from the refresh buffer and? painted? on the screen one row (scan line) at a time (fig.below). Each screen point is referred to as a pixel or pel (shortened forms of picture element).

Refreshing on raster-scan displays is carried out at the rate of 60 to 80 frames per second, although some systems are designed for higher refresh rates. Sometimes, refresh rates are described in units of cycles per second, or Hertz (Hz), where a cycle corresponds to one frame. At the end of each scan line, the electron beam returns to the left side of the screen to begin displaying the next scan line. The return to the left of the screen, after refreshing each scan line, is called the horizontal retrace of the electron beam. And at the end of each frame (displayed in 1/80th to 1/60th of a second), the electron beam returns (vertical retrace)to the top left corner of the screen to begin the next frame.

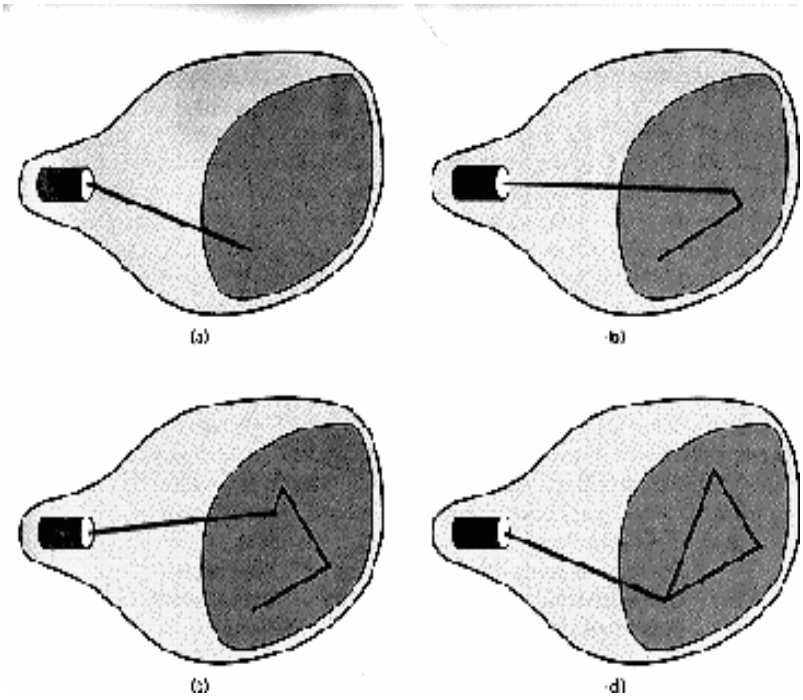
On some raster-scan systems (and in TV sets), each frame is displayed in two passes using an interlaced refresh procedure. In the first pass, the beam sweeps across every other scan line from top to bottom. Then after the vertical retrace, the beam sweeps out the remaining scan lines(fig.below). Interlacing of the scan lines in this way allows us to see the entire screen displayed in one-half the time it would have taken to sweep across all the lines at once from top to bottom.



1.2.3 Random-scan Display

In random scan display the electron beam is directed only to the part of the screen where the picture is to be drawn.

Random Scan monitor draw one picture at a time therefore they are also known as vector display. Refresh rate or random scan system depends on the number of times to be displayed. Picture definition is stored in an area of memory called refresh display file.



- Vector graphics (stroke-writing/calligraphic systems)
    - advantages/disadvantages
    - combination?
- Draws one line at a time
- Smooth lines
- Refresh display file/program/list
- Refresh rate

Difference between raster and random scan displays:

A raster scan, or raster scanning, is the rectangular pattern of image capture and reconstruction in television. By analogy, the term is used for raster graphics, the pattern of image storage and transmission used in most computer bitmap image systems.

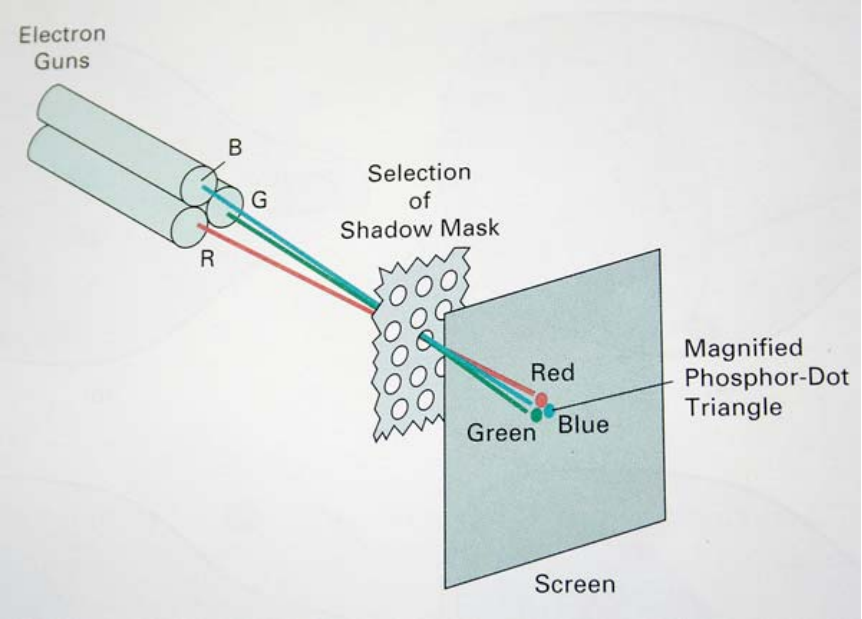
In random scan display the electron beam is directed only to the part of the screen where the picture is to be drawn. Random Scan monitor draw one picture at a time therefore they are also known as vector display. Refresh rate or random scan system depends on the number of times to be displayed. Picture definition is stored in an area of memory called refresh display file.

1.2.4 Color CRT Monitors

The beam penetration method for displaying color pictures has been used with random-scan monitors. Two layers of phosphor, usually red and green, are coated on to the inside of the CRT screen, and the displayed color depends on how far the electron beam penetrates into the phosphor layers.

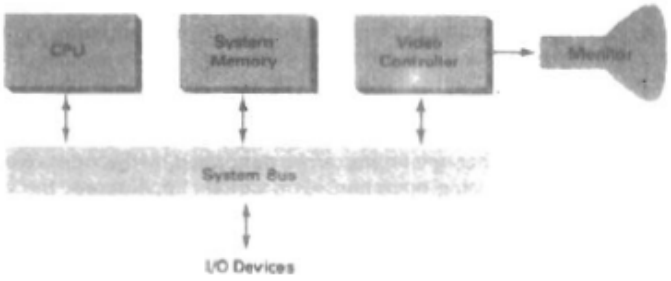
Shadow-mask methods are commonly used in raster-scan systems (including color TV) because they produce a much wider range of color than the beam penetration method. A shadow-mask CRT has three phosphor color dots at each pixel position. One phosphor dot emits a red light, another emits a green light, and the third emits a blue light. This type of CRT has three electron guns, one for each color dot, and a shadow-mask grid just behind the phosphor ?coated screen. Fig.below illustrates the delta-delta shadow-mask method, commonly used in color CRT systems. The three electron beam are deflected and focused as a group onto the shadow mask, which contains a series of holes aligned with the phosphor-dot patterns. When the three beams pass through a hole in the shadow mask, they activate a dot triangle, which appears as a

small color spot the screen the phosphor dots in the triangles are arranged so that each electron beam can activate only its corresponding color dot when it passes through the shadow mask.



1.2.5 Raster Scan Systems

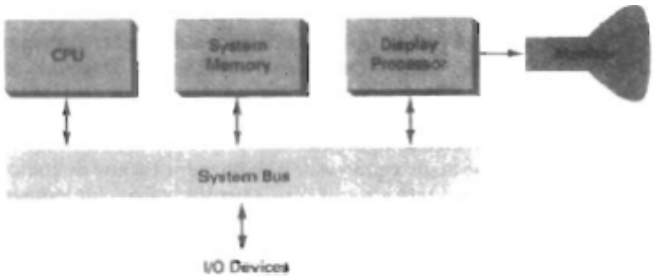
Interactive raster graphics systems typically employ several processing units. In addition to the central processing unit, or CPU, a special-purpose processor, called the video controller or display controller, is used to control the operation of the display device. Organization of a simple raster system is shown in the following figure. Here, the frame buffer can be anywhere in the system memory, and the video controller accesses the frame buffer to refresh the screen. In addition to the video controller, more sophisticated raster systems employ other processors as coprocessors and accelerators to implement various graphics operations.



Architecture of a simple raster graphics system

1.2.6 Random Scan Systems

The organization of a simple random-scan (vector) system is shown in the following figure. An application program is input and stored in the system memory along with a graphics package. Graphics commands in the application program are translated by the graphics package into a display file stored in the system memory. This display file is then accessed by the display processor to refresh the screen. The display processor cycles through each command in the display file program once during every refresh cycle. Sometimes the display processor in a random-scan system is referred to as a display processing unit or a graphics controller.



Architecture of a simple random scan system

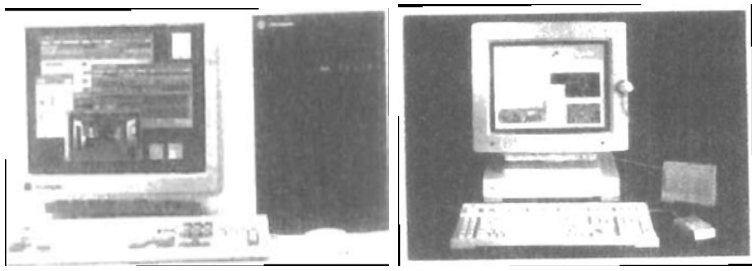
Graphics patterns are drawn on a random-scan system by directing the electron beam along the component lines of the picture. Lines are defined by the values for their coordinate endpoints, and these input coordinate values are converted to x and y deflection voltages. A scene is then drawn one line at a time by positioning the beam to fill in the line between specified endpoints.

1.2.6 Graphics Monitors and Workstations

Most graphics monitors today operate as raster scan displays, and here we survey a few of the many graphics hardware configurations available. Graphics systems range h m small general-purpose computer systems with graphics capabilities to sophisticated full color systems that are designed specifically for graphics applications. A typical screen resolution for personal computer systems, such as the Apple Quadra shown in the following figure is 640 by 480, although screen resolution and other system capabilities vary depending on the size and cost of the system. Diagonal screen dimensions for general-purpose personal computer systems can range from 12 to 21 inches, and allowable color selections range from 16 to over 32,000.

***A* desktop general-purpose computer system that can be used for graphics applications**

For workstations specifically designed for graphics applications, such as the systems shown in the following figure, typical *screen* resolution is 1280 by 1024, with a screen diagonal of 16 inches or more. Graphics workstations can be configured with from *8* to 24 bits per pixel (full-color systems), with higher screen resolutions, faster processors, and other options available in high-end systems.

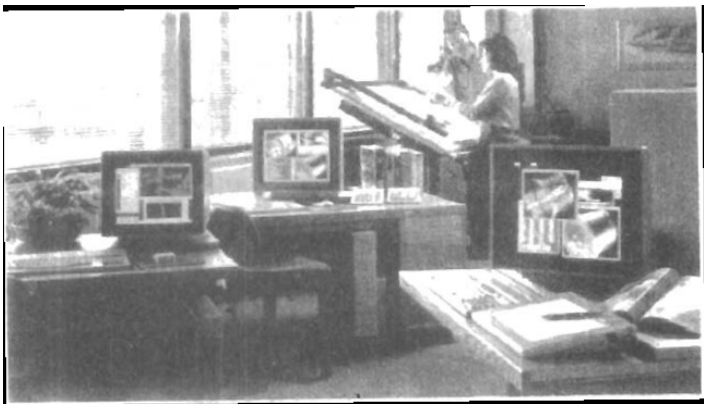


Computer graphics workstations with keyboard and mouse input devices. (a) The Iris Indigo. (b). SPARCstation *10*.

Many graphics workstations, such as some of those shown in the following figure are configured with two monitors. One monitor can be used to show all features of an object or scene, while the second monitor displays the detail in some part of the picture. Another use for dual-monitor systems is to view a picture on one monitor and display graphics options (menus) for manipulating the picture components on the other monitor.

Single- and dual-monitor graphics workstations

The following figure illustrate example of interactive graphics workstations containing multiple input and other devices. (Ex. A typical setup for CAD applications). Various keyboards, button boxes, tablets, and mice are attached to the video monitors for use in the design process.



Multiple workstations for a CAD group

1.3 Input Devices

1.3.1 Input Devices

What good is a computer, unless you have a way to get information into it? Therefore, input devices are necessary to provide ways to communicate with the computer. Information and commands are issued to the computer by way of these devices.

Examples of input devices include the keyboard, mouse, modem, joystick, digitizing pen and tablet, microphone, touch screens, scanner, camera, I/O Architecture

1.3.2 Key Board

What is it?

The keyboard is an input device. It has letter and number keys, and what are called function keys, computer specific task keys, that allow you, the user, to use an English-like language to issue instructions to an electronic environment. It is the primary input device. It uses a cursor to keep your place on the screen and to let you know where to begin typing. You are able to input commands, type data into documents, compose documents, draw pictures with use of certain keys, pull down menus, and respond to prompts issued by the computer. Almost all computers require you to use a keyboard unless, of course, it is adapted for individuals with disabilities or for a specified alternative input devices.

The keyboard contains special keys to manipulate the user interface. When a key is touched, an electrical impulse is

sent through the device which is picked up by the operating system software, and sent through the computer to be processed.

The keyboard operates as a typical typewriter and uses a standard "QWERTY" keyboard. QWERTY is the way the keyboard is set up for typing. If you look at the keyboard under the top number row, you will see that the alphabet top row begins with QWERTY.

1.3.3 Mouse

How Does It Work?

Mousing Around

The computer mouse moves by way of a roller and ball system. When you move the mouse across the desktop, the ball underneath rolls. This ball corresponds to the position of what is called a pointer on the screen. The pointer is usually shaped like an arrow, though some people like to change their pointer to look like objects. (One person in our group has changed his pointer to an ink pen icon.) When you move the pointer around it is called mousing. The speed of the mouse can be managed by your computer operating system software, or a commercial application program for your mouse. You can drag objects on the screen by clicking on the object, holding down on the mouse button, and rolling the mouse across the desktop until you get the object to a new location. When you reach the spot that you want, let go of the mouse button.

Most mice come with two buttons. You use the left button on the mouse to do most selecting of objects. The right button can be used for some menu actions. This is especially true

when using browser software to examine and manipulate pictures and graphics on the Internet. There is a three button mouse and the middle button can be programmed for specific application software, but usually the two button mouse is used the most. If you are left handed, you can change to a left-handed mouse option in your software so that you can use your mouse in your left hand.

1.3.4 Types Of Pointing Devices

Pen- The pen lets you draw on what is called a digitizing tablet that mirrors the surface area of the computer screen. The pen can be used as a standard mouse (without wires connected to it) or also as a free flowing drawing device. The pen is useful for drawing since drawing graphics with a mouse tends to be somewhat difficult.

Mouse - The mouse is a hand held device that lets you point to and selected items on your screen. In a PC mouse there are mostly 2-3 buttons and on a Mac there is one. A ball under the mouse senses movement. To ensure smooth motion your should remove the ball and clean it regularly.

Cordless Mouse - The cordless mouse is a lot better than a normal mouse thus by reducing the clutter of the work space needed to move the mouse around. This mouse runs on a battery. When you move the mouse it sends an infrared beam to a sensor which interprets it causing it to move.

Trackball - The trackball is an upside-down mouse that remains stationary on your desk. It is the same principle as the mouse except that the rollers are reversed and the ball is on top. This ball does not need as much attention as the normal mouse because the only thing that touches it is your hand as the normal mouse touches a surface.

Touchpad - The touchpad has sensors that sense your touch. When they sense your touch they send a signal to the computer to move the mouse pointer to that location on the screen.

Joystick- The joystick allows the user to move quickly in computer games.

Light pen- The light pen system allows the user to touch the computer screen with a lighted pen to activate commands and make selections.

Touch Screen- The touch screen lets the user touch the area to be activated by using the finger or hand.

Head Mounted Virtual Reality Displays, Wands, Special Trackballs, Data Gloves, and Special 3-D Flying Mice that can go in six different directions- These devices are currently the newest pointing devices.

1.3.5 Image Scanners

Drawings, graphs, color and black-and-white photos, or text can be stored for computer processing with an image scanner by passing an optical scanning mechanism over the information to be stored. The gradations of gray scale or color are then recorded and stored in an array. Once we have the internal representation of a picture, we can apply transformations to rotate, scale, or crop the picture to a particular screen area. We can also apply various image-processing methods to modify the array representation of the picture. For scanned text input, various editing operations can be performed on the stored documents. Some scanners are able to scan either graphical representations or text, and they come in a variety of sizes and capabilities. A small hand-model scanner is shown in the following figure.

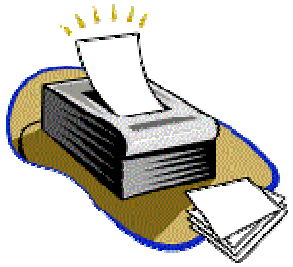
***A* hand-held scanner that can be used to input either text or graphics images**

1.4 Hard Copy Devices

Ink Jet Printer - This printer is set at the standard for home use. The print head sprays ink onto paper forming an image or text. With imagery on the ink jet it is lost by premature smearing if touched prematurely or if the wrong paper was used. The paper best used for imagery is a high clay content paper and for the ink a smudge resistance or oil based ink. For business letters a ink jest printer is the most common one used for its text plus imagery to add to the paper. The common speed on a ink jet printer is about 3 ppm and 300 dpi (dots per inch).

Laser Printer - This printer is used for high speed text printing and rough draft image quality, this printer is ideal for rough draft or proofing work. This is the most complicated of all printers because of its printing ability, Thus making this printer more expensive than the others on the market. This printer has

its own interpretation language called PCL (printer control language). The laser printer works similarly to a photocopier by sending a beam or a laser of positive electrons to the paper, when passed to the ink or toner it becomes attracted (remember opposites attract) causing it to stick to the paper then passed through to a hot presser which causes the ink to be permanently inscribed on the paper.

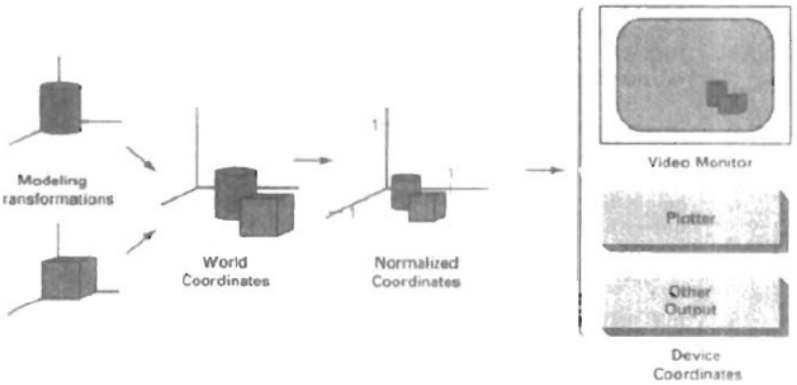


1.5 Graphics Software

There are two general classifications for graphics software: general programming packages and special-purpose applications packages. A general graphics programming package provides an extensive set of graphics functions that can be used in a high-level programming language, such as C or FORTRAN. An example of a general graphics programming package is the GL (Graphics Library) system on Silicon Graphics equipment. Basic functions in a general package include those for generating picture components (straight lines, polygons, circles, and other figures), setting color and intensity values, selecting views, and applying transformations. By contrast, application graphics packages are designed for nonprogrammers, so that users can generate displays without worrying about how graphics operations work. The interface to the graphics routines in such packages allows users to communicate with the programs in their own terms. Examples of such applications packages are the artist's painting programs and various business, medical, and CAD systems.

Coordinate Representations

With few exceptions, general graphics packages are designed to be used with Cartesian coordinate specifications. If coordinate values for a picture are specified in some other reference frame (spherical, hyperbolic, etc.), they must be converted to Cartesian coordinates before they can be input to the graphics package. Generally, a graphics system first converts world-coordinate positions to normalized device coordinates, in the range from 0 to 1, before final conversion to specific device coordinates. This makes the system independent of the various devices that might be used at a particular workstation. The following figure illustrates the sequence of coordinate transformations from modeling coordinates to device coordinates for a two-dimensional application. An initial modeling-coordinate position (x_{mc},y_{mc}) in this illustration is transferred to a device coordinate position (x_{dc}, y_{dc}) with the sequence: $(x_{mc}, y_{mc}) \rightarrow (x_{wc}, y_{wc}) \rightarrow (x_{nc}, y_{nc}) \rightarrow (x_{dc}, y_{dc})$ .The modeling and world-coordinate pitions'in this transformation can be any floating-point values; normalized coordinates satisfy the inequalities: $0<=x_{mc}<=1$, $0<=y_{mc}<=1$; and the device coordinates x_{dc} and y_{dc} are integers within the range (0,0)to (x_{max}, y_{max}) for a particular output device. To accommodate differences in scales and aspect ratios, normalized coordinates are mapped into a square area of the output device so that proper proportions are maintained.



The transformation sequence from modeling coordinates to device coordinates for a two dimensional scene. Object shapes a= defined in local modeling-coordinate systems, then positioned within the overall world-coordinate scene. World-coordinate specifications are then transformed into normalized coordinates. At the final step, individual device drivers transfer the normalized coordinate representation of the scene to the output devices for display.

1.6 Output Primitives and Attributes:

1.6.1POINTS AND LINES

Points and lines are two of the most fundamental concepts in Geometry, but they are also the most difficult to define. We can describe intuitively their characteristics, but there is no set definition for them: they, along with the plane, are the undefined terms of geometry. All other geometric definitions and concepts are built on the undefined ideas of the point, line and plane. Nevertheless, we shall try to define them.

Point

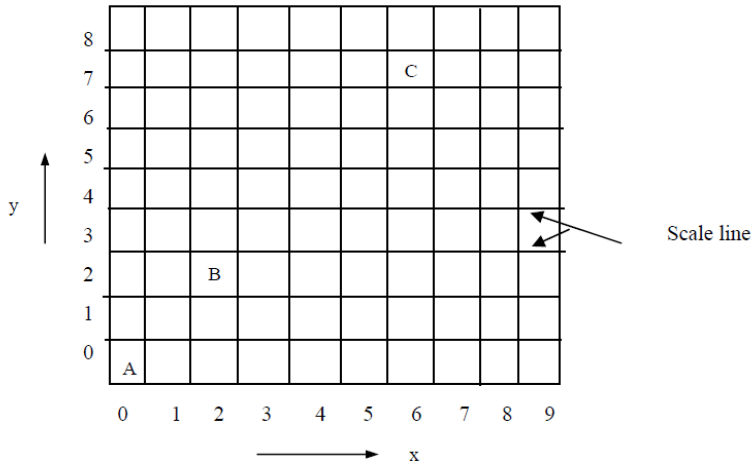
A point is a location in space. Points are dimensionless. That is, a point has no width, length, or height. We locate points relative to some arbitrary standard point, often called the "origin". Many physical objects suggest the idea of a point. Examples include the tip of a pencil, the corner of a cube, or a dot on a sheet of paper.

Line

As for a line segment, we specify a line with two points. Starting with the corresponding line segment, we find other line segments that share at least two points with the original line segment. In this way we extend the original line segment indefinitely. The set of all possible line segments findable in this way constitutes a line. A line extends indefinitely in a single

dimension. Its length, having no limit, is infinite. Like the line segments that constitute it, it has no width or height. You may specify a line by specifying any two points within the line. For any two points, only one line passes through both points. On the other hand, an unlimited number of lines pass through any single point.

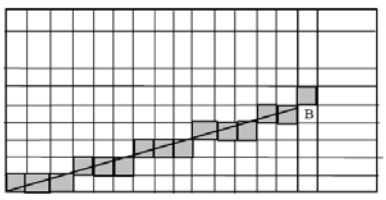
We have seen that in order to draw primitive objects, one has to first scan convert the objects. This refers to the operation of finding out the location of pixels to be intensified and then setting the values of corresponding bits, to the desired intensity level. Each pixel on the display surface has a finite size depending on the screen resolution and hence, a pixel cannot represent a single mathematical point. However, we consider each pixel as a unit square area identified by the coordinate of its lower left corner, the origin of the reference coordinate system being located at the lower left corner of the display surface. Thus, each pixel is accessed by a non-negative integer coordinate pair (x, y). The x values start at the origin and increase from left to right along a scan line and the y values (i.e., the scan line numbers) start at bottom and increase upwards.



Scan Lines

The above figure shows the Array of square pixels on the display surface. Coordinate of pixel A: 0, 0; B: 2, 2; C: 6, 7. A coordinate position (6.26, 7.25) is represented by C, whereas (2.3, 2.5) is represented by B. Because, in order to plot a pixel on the screen, we need to round off the coordinates to a nearest integer. Further, we need to say that, it is this rounding off, which leads to distortion of any graphic image.

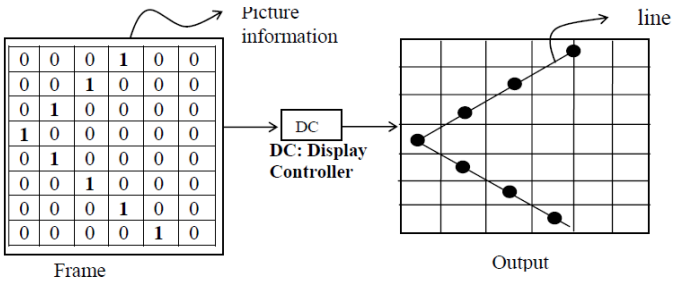
Line drawing is accomplished by calculating the intermediate point coordinates along the line path between two given end points. Since, screen pixels are referred with integer values, plotted positions may only approximate the calculated coordinates – i.e., pixels which are intensified are those which lie very close to the line path if not exactly on the line path which is in the case of perfectly horizontal, vertical or 45° lines only. Standard algorithms are available to determine which pixels provide the best approximation to the desired line, we will discuss such algorithms in our next section. Screen resolution however, is a big factor towards improving the approximation. In a high resolution system the adjacent pixels are so closely spaced that the approximated line-pixels lie very close to the actual line path and hence, the plotted lines appear to be much smoother — almost like straight lines drawn on paper. In a low resolution system, the same approximation technique causes lines to be displayed with a “stair step appearance” i.e., not smooth as shown in the following figure, the effect is known as the stair case effect. We will discuss this effect and the reason behind this defect in the next section of this unit.



A **Stair case effect**

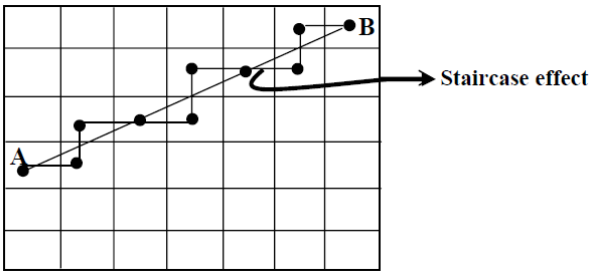
1.6.2 Line Drawing Algorithms

We have discussed the case of frame buffer where information about the image to be projected on the screen is stored in an m*n matrix, in the form of 0s and 1s; the 1s stored in an m* n matrix positions are brightened on the screen and 0's are not brightened on the screen and this section which may or may not be brightened is known as the Pixel (picture element). This information of 0s and 1s gives the required pattern on the output screen i.e., for display of information. In such a buffer, the screen is also in the form of m*n matrix, where each section or niche is a pixel (i.e., we have m*n pixels to constitute the output).



Basic Line Generation

Now, it is to be noted that the creation of a line is merely not restricted to the above pattern, because, sometimes the line may have a slope and intercept that its information is required to be stored in more than one section of the frame buffer, so in order to draw or to approximate such the line, two or more pixels are to be made ON. Thus, the outcome of the line information in the frame buffer is displayed as a stair; this effect of having two or more pixels ON to approximating a line between two points say A and B is known as the Staircase effect. The concept is shown below.



Staircase effect

So, from the Figure 4, I think, it is clear to you that when a line to be drawn is simply described by its end points, then it can be plotted by making close approximations of the pixels which best suit the line, and this approximation is responsible for the staircase effect, which miss projects the information of the geometry of line stored in the frame buffer as a stair. This defect known as Staircase effect is prominent in DDA Line generation algorithms, thus, to remove the defect Bresenham line generation Algorithm was introduced. We are going to discuss DDA (Digital Differential Analyser) Algorithm and Bresenham line generation Algorithm next.

Characteristics of Line Drawing Algorithms:

1. Lines should appear sharp and straight
2. Line should terminate accurately
3. Line should have constant density
4. Line density should be independent of line, length and angle.

Line drawing Algorithms:

Straight line segments are used greatly in computer generated pictures. They occur in block diagrams, bar charts

and graphics, civil and mechanical engineering drawings, logical schematics, and architectural plans are created and used in computer graphics.

There are two types of line drawing algorithms, Digital differential Algorithm (DDA) and Bresenham's Line drawing Algorithm.

Digital differential Algorithm (DDA)

In this algorithm, we sample the line at unit intervals in one coordinate and determine corresponding integer value nearest the line path for the other coordinates.

Algorithm:

Step1
Input 2 end point pixel positions (x1, y1) and (x2, y2)
Step2
Find horizontal and vertical difference between the end points dx = x2-x1 dy = y2-y1
Step3

The difference with the greater magnitude determines the value of parameter steps"
If abs (dx)>abs (dy) then
Steps = abs (dx)
Else
Steps = abs (dy)

Notes

Step4
Starting with pixel position (x1, y1) be determined offset needed at each step to generate next pixel along the line path. X increment = dx/steps Y increment = dy/steps
Step5
Assign the values of x1, y1 to x, y X: = x1; Y: = y1;
Step6
Plot the pixel at (x, y) position on screen set pixel (round(x), round(y), 1). Here '1' is the intensity of pixel i.e., intensity with which picture is illuminated.
step7
Calculate the values of x and y for the next pixel position. X = x + x Increment Y = y + y Increment
Step8
Plot the pixel at (x, y) position, set pixel (round(x), round(y), 1)
Step9
Repeat the steps 7 and 8 until the value of I i.e., start from 1 to steps.

EXAMPLE: Draw A line between the points (20,10) and (30,18)

Step 1: x1 = 20, x2 =30, y1 = 10, y2 = 18.
Step 2: dx = 30 -20 = 10, dy = 18 – 10 = 8.
Step 3: dx > dy => 10 >8, steps =10.
Step 4: x increment = 10/10 = dx/steps = 1.
Y increment = 8/10 = dy/steps = 0.8.
Step 5: x = 20, y = 10.
Step 6: pixel (20,10) .
Step 7: x = x + x increment.
Y = y + y increment.

I	x	y	(round(x),round(y))
1	20+1=21	10+0.8=10.8	(21,11)
2	21+1=22	10.8+0.8=11.6	(22,12)
3	22+1=23	11.6+0.8=12.4	(23, 12)
4	23+1=24	12.4+0.8=13.2	(24,13)
5	24+1=25	13.2+0.8=14	(25,14)
6	25+1=26	14+0.8=14.8	(26,15)
7	26+1=27	14.8+0.8=15.6	(27,16)
8	27+1=28	15.6+0.8=16.4	(28,16)
9	28+1=29	16.4+0.8=17.2	(29,17)
10	29+1=30	17.2+0.8=18	(30,18)

Advantage:

Does not calculate coordinates based on the complete equation (uses offset method)

Disadvantage:

- Round-off errors are accumulated, thus line diverges more and more from straight line
- Round-off operations take time
- Perform integer arithmetic by storing float as integers in numerator and denominator and performing integer arithmetic

Bresenham's Line Drawing Algorithm:

1. Input the two line endpoints and store the left endpoint in (x0, y0).
2. Set the color for the frame-buffer position (x0, y0) – i.e. plot the first point.
3. Calculate the constant $2\Delta y - \Delta x$, and set the starting value for the decision parameter as $p0 = 2\Delta y - \Delta x$.
4. At each xk along the line, from k=0, perform the following test: if pk<0, next point to plot is (xk + 1, yk) and $pk+1 = pk + 2\Delta y$ otherwise, next point to plot is (xk + 1, yk + 1) and $pk+1 = pk + 2\Delta y - 2\Delta x$
5. Repeat step 4 Δx - 1 times.

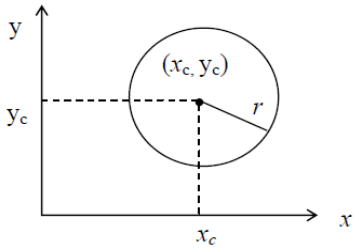
1.7 Circle Generation Algorithms

Circle is one of the basic graphic component, so in order to understand its generation, let us go through its properties first:

Properties of Circles

In spite of using the Bresenham circle generation (i.e., incremental approval on basis of decision parameters) we could use basic properties of circle for its generation.

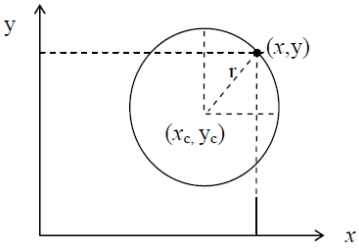
These ways are discussed below:



Property of circle

Generation on the basis of Cartesian Coordinates:

A circle is defined as the set of points or locus of all the points, which exist at a given distance r from center (x_c, y_c). The distance relationship could be expressed by using Pythagonous theorem in Cartesian coordinates as



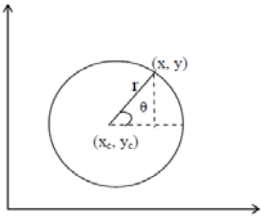
Circle generation (cartesian coordinate system)

$$(x - x_c)^2 + (y - y_c)^2 = r^2 \text{ ----------(1)}$$

Now, using (1) we can calculate points on the circumference by stepping along x-axis from x_c–r to x_c + r and calculating respective y values for each position.

$$y = y_c \pm \sqrt{r^2 - (x - x_c)^2} \quad \text{----------(2)}$$

Generation on basis of polar coordinates (r and θ)



Circle generation (polar coordinate system)

Expressing the circle equation in parametric polar form yields the following equations

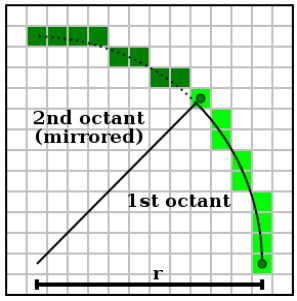
$$x = x_c + r \cos \theta$$

$$y = y_c + r \sin \theta \text{ ----------(3)}$$

Using a fixed angular step size we can plot a circle with equally spaced points along the circumference.

Midpoint circle algorithm

In computer graphics, the midpoint circle algorithm is an algorithm used to determine the points needed for drawing a circle. The algorithm is a variant of Bresenham's line algorithm, and is thus sometimes known as Bresenham's circle algorithm, although not actually invented by Bresenham.



Rasterisation of a circle by the Bresenham algorithm

The algorithm starts accordingly with the circle equation

$$x^2 + y^2 = r^2.$$

So, the center of the circle is located at (0,0). We consider first only the first octant and draw a curve which starts at point (r,0) and proceeds upwards and to the left, reaching the angle of 45°.

The "fast" direction here is the y direction. The algorithm always does a step in the positive y direction (upwards), and every now and then also has to do a step in the "slow" direction, the negative x direction.

The frequent computations of squares in the circle equation, trigonometric expressions or square roots can again be avoided by dissolving everything into single steps and recursive computation of the quadratic terms from the preceding ones.

From the circle equation we obtain the transformed equation $x^2 + y^2 - r^2 = 0$, where r^2 is computed only a single time during initialization:

$$\begin{aligned} y_{n+1}^2 &= (y_n + 1)^2 \\ &= y_n^2 + 2y_n + 1 \end{aligned}$$

and accordingly for the *x*-coordinate. Additionally we need to add the midpoint coordinates when setting a pixel. These frequent integer additions do not limit the performance much, as we can spare those square (root) computations in the inner loop in turn. Again the zero in the transformed circle equation is replaced by the error term.

The initialization of the error term is derived from an offset of ½ pixel at the start. Until the intersection with the perpendicular line, this leads to an accumulated value of *r* in the error term, so that this value is used for initialization.

A possible implementation of the Bresenham Algorithm for a full circle in C. Here another variable for recursive computation of the quadratic terms is used, which corresponds with the term $2n + 1$ above. It just has to be increased by 2 from one step to the next:

```
void rasterCircle(int x0, int y0, int radius)

{

  int f = 1 - radius;

  int ddF_x = 1;

  int ddF_y = -2 * radius;

  int x = 0;

  int y = radius;
```

```
 setPixel(x0, y0 + radius);
setPixel(x0, y0 - radius);
setPixel(x0 + radius, y0);
setPixel(x0 - radius, y0);
 while(x < y)
{
  // ddF_x == 2 * x + 1;
  // ddF_y == -2 * y;
  // f == x*x + y*y - radius*radius + 2*x - y + 1;
  if(f >= 0)
  {
    y--;
    ddF_y += 2;
    f += ddF_y;
  }
  x++;
  ddF_x += 2;
  f += ddF_x;
  setPixel(x0 + x, y0 + y);
  setPixel(x0 - x, y0 + y);
  setPixel(x0 + x, y0 - y);
```

```
    setPixel(x0 - x, y0 - y);
    setPixel(x0 + y, y0 + x);
    setPixel(x0 - y, y0 + x);
    setPixel(x0 + y, y0 - x);
    setPixel(x0 - y, y0 - x);
  }
}
```

Note: There is correlation between this algorithm and the sum of first *N* odd numbers, which this one basically does. That is,

$$1+3+5+7+9+\cdots=\sum_{n=0}^{N}2n+1=(N+1)^2.$$

So.

When we compare sum of N odd numbers to this algorithm we have.

ddF_y=-2*radius is connected to last member of sum of N odd numbers.

This member has index equal to value of radius (integral).

Since odd number is 2*n + 1 there is 1 handled elsewhere

or it should be -2*radius - 1

ddF_x = 0 should be 1. Because difference between two consecutive odd numbers is 2.

If so f += ddF_y + 1 is f+= ddF_y. Saving one operation.

f = - radius + 1 Initial error equal to half of "bigger" step.

In case of saving one addition it should be either-radius or-radius+2.

In any case there should be addition of 1 driven out of outer loop.

So.

f += ddF_y Adding odd numbers from Nth to 1st.

f += ddF_x Adding odd numbers from 1st to Nth. 1 is missing because it can be moved outside of loop.

Text Book:

Donald Hearn & Pauline M.Baker – Computer Graphics 2nd Edition –PHI.

Reference Book:

1.Foley, Van Dam, Feiner and Hughes – Computer Graphics Principles and Practice 3rd Edition.

2.N.Krishnamurthy - Introduction to Computer Graphics.

UNIT – II

TWO DIMENSIONAL GEOMETRIC TRANSFORMATIONS AND VIEWING

Objectives

After going through this unit, you should be able to:

- Describe the basic transformations for 2-D translation, rotation, scaling and shearing;
- Discuss the role of composite transformations;
- Describe composite transformations for Rotation about a point and reflection about a line;
- Define and explain the use of homogeneous coordinate systems for the transformations.
- Describe viewing pipeline, window to viewport coordinate transformation;
- Describe clipping operations-line clipping, cohen Sutherland line clipping;
- Describe polygon clipping- Sutherland hodgman polygon clipping;

2.1 Introduction

A wide range of pictures and graphs can be created with the procedures for displaying output primitives and their attributes. Many applications are used for altering or manipulating displays. Facility layouts and design applications

are created by arranging the orientations and sizes of the component parts of the scene. Changes in orientation, size and shape of a picture are accomplished with geometric transformations.

Geometric transformation is the process of altering the coordinate descriptions of objects. Transformations are of two types basic transformations which are sub classified into translation, rotation, and scaling and other transformations which are sub classified into reflection and shear.

We first discuss methods for performing geometric transformations and then consider how transformation functions can be incorporated into graphics packages.

2.2 Basic Transformations

The basic geometric transformations are:

- Translation
- Rotation
- Scaling

Let us discuss procedures for applying translation, rotation, and scaling parameters to reposition and resize two-dimensional objects. Then, in next section we consider transformation equations and expressing them in matrix formulation that allows efficient combination of object transformations.

2.2.1 Translation:

Translation is applied to an object by repositioning it along a straight-line path from one coordinate location to another.

We translate a two-dimensional point by adding translation distances t_x and t_y, to the original coordinate position (**x,** y) to move the point to a new position *(x '***,** y').

Old coordinates: (x, y)

New coordinates: *(x ',* y') = (x + t_x, y + t_y)

Where t_x is distance in x direction

t_y is distance in y direction

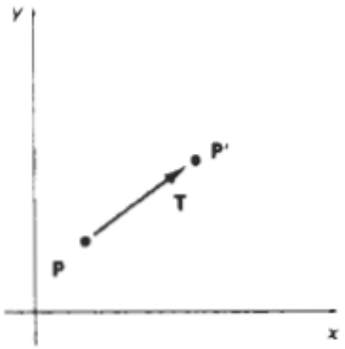


Figure 2. 1

Translating a point from position **P** to 'position **P'** with translation vector **T.**

$$x' = x + t_x, \qquad y' = y + t_y \tag{2.1}$$

The translation distance pair (t_x , t_y)is called a translation vector or shift vector.

The above equation 2.1 can be expressed as a single matrix equation by using column vectors to represent coordinate positions and the translation vector:

$$\mathbf{P} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, \qquad \mathbf{P}' = \begin{bmatrix} x_1' \\ x_2' \end{bmatrix}, \qquad \mathbf{T} = \begin{bmatrix} t_x \\ t_y \end{bmatrix} \tag{2.2}$$

This can be written as two-dimensional translation equations in the matrix form:

$$\mathbf{P}' = \mathbf{P} + \mathbf{T} \qquad 2.3$$

Matrix-transformation equations can be expressed in terms of coordinate row vectors instead of column vectors and is a generally followed convention.

$$P = [x, y]$$

$$T = [t_x, t_y]$$

Every point on the object is translated by the same amount. Translation is a *rigid-body transformation* that moves objects without deformation.

Polygons are translated by adding the translation vector to the coordinate position of each vertex and regenerating the polygon using the new set of vertex coordinates and the current attribute settings.

Figure 2-2 illustrates the application of a specified translation vector to move an object from one position to another.

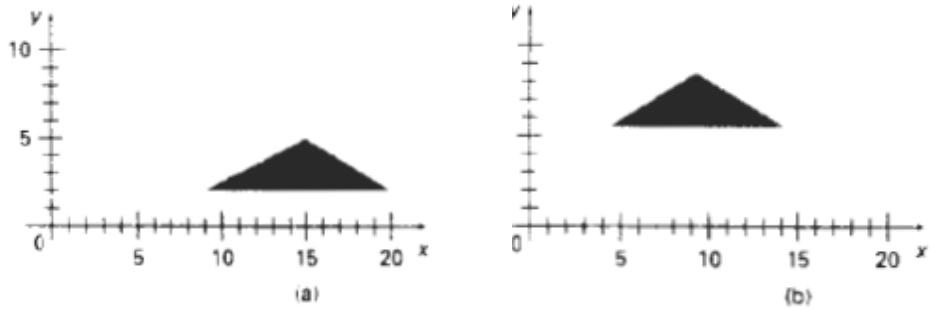


Figure 2.2

(a) Moving a polygon from position (a) 5 10 IS 20 to position (b) with the translation

(b) vector (-5.W, 3.75).

Similar methods are used to translate curved objects like circle or ellipse, splines by displacing the coordinate positions defining curves the objects, then we reconstruct the curve paths using the translated coordinate points.

2.2.3 Rotation:

A two-dimensional rotation is applied to an object by repositioning it along a circular path in the **xy** plane.

To generate a rotation, we specify a rotation angle Θ and the position (x_r, y_r) of the rotation point (or pivot point) about which the object is to be rotated.

Positive values for rotation angle define counterclockwise rotation about the pivot point (Fig. 2-3), and negative values for the rotation angle define clockwise direction rotation (Fig. 2-3).

This transformation can also be described as a rotation about a rotation axis that is perpendicular to the **xy** plane and passes through the pivot point.

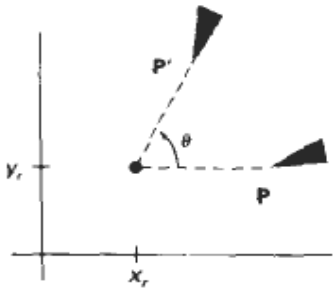


Figure 2-3

Rotation of an object through angle Θ about the pivot point (x_r, y_r).

Old coordinates: (x, y)

New coordinates: *(x ',* y')

Rotation angle: Θ

Rotation point or pivot point: (x_r, y_r)

Rotation is of two types:

- Rotation over origin
- Rotation over an arbitrary point

Rotation over origin:

First let us determine the transformation equations for rotation of a point position P when the pivot point is at the coordinate origin.

In the Fig. 2-4, the angular and coordinate relationships of the original and transformed point positions are shown. r is the constant distance of the point from the origin, angle Φ is the original angular position of the point from the horizontal, and Θ is the rotation angle.

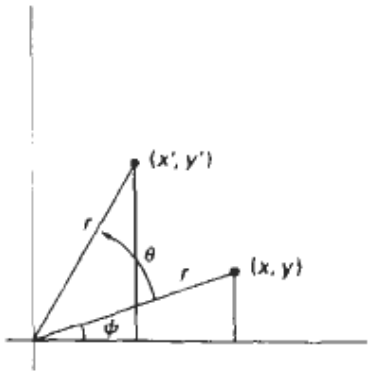


Figure 2-4

Rotation of a point from position (x, y) to position x', y ') through an angle Θ relative to the coordinate origin. The original angular displacement of the point from the x axis is Φ.

$$x' = r\cos(\phi + \theta) = r\cos\phi\cos\theta - r\sin\phi\sin\theta$$
$$y' = r\sin(\phi + \theta) = r\cos\phi\sin\theta + r\sin\phi\cos\theta \qquad 2.4$$

Using standard trigonometric identities, we can express the transformed coordinates in terms of angles Θ and Φ as

The original coordinates of the point in polar coordinates are

$$x = r\cos\phi, \qquad y = r\sin\phi \qquad 2.5$$

Substituting expressions 2.5 into 2.4, we obtain the transformation equations for rotating a point at position *(x,* y) through an angle Θ about the origin:

$$x' = x\cos\theta - y\sin\theta$$
$$y' = x\sin\theta + y\cos\theta \qquad 2.6$$

With the column-vector representations 2.2 for coordinate positions, we can write the rotation equations in the matrix form:

$$\mathbf{P}' = \mathbf{R} \cdot \mathbf{P} \qquad 2.7$$

Where the rotation matrix is

$$\mathbf{R} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \qquad 2.8$$

When coordinate positions are represented as row vectors instead of column vectors, the matrix product in rotation equation 5-7 is transposed so that the transformed row coordinate vector [x' y'] is calculated as

$$\mathbf{P}'^{T} = (\mathbf{R} \cdot \mathbf{P})^{T}$$
$$= \mathbf{P}^{T} \cdot \mathbf{R}^{T}$$

Where PT = [x y], and the transpose RT of matrix R is obtained by interchanging rows and columns. For a rotation matrix, the transpose is obtained by simply changing the sign of the sine terms.

Rotation over an arbitrary point:

Rotation of a point about an arbitrary pivot position is illustrated in Fig. 2-5.

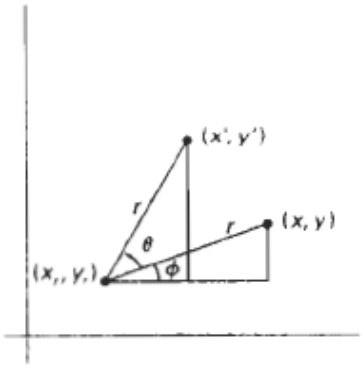


Figure 2.5

Rotating a point from position (x, y) to position (x ', y ') through an angle Θ about rotation point (x_1, y_1)

Using the trigonometric relationships in this figure, we can generalize Eqs. 2.6 to obtain the transformation equations for rotation of a point about any specified rotation position (x_1, y1):

$$x' = x_r + (x - x_r)\cos\theta - (y - y_r)\sin\theta$$
$$y' = y_r + (x - x_r)\sin\theta + (y - y_r)\cos\theta \qquad 2.9$$

2.2.4 Scaling:

Scaling transformation alters the size of an object. Scaling may be used to reduce or enlarge the size of an object.

Scaling can be carried out for polygons by multiplying the coordinate values (x, y) of each vertex by scaling factors s_x and s_y to produce the transformed coordinates (x', y'):

$$x' = x \cdot s_x, \qquad y' = y \cdot s_y \qquad 2.10$$

Scaling factor s_x scales objects in the **x** direction.

Scaling factor s_y scales objects in the y direction.

The transformation equations 2.10 can also be written in the matrix form:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} \qquad 2.11$$

Or

$$\mathbf{P}' = \mathbf{S} \cdot \mathbf{P} \qquad 2.12$$

where S is the 2 by 2 scaling matrix in Eq. 2.11.

Any positive numeric values can be assigned to the scaling factors s_x, and s_y:

- Values less than 1 reduce the size of objects;
- values greater than 1 produce an enlargement.
- Specifying a value of 1 for both s_x and s_y leaves the size of objects unchanged.
- When s_x and s_y are assigned the same value, a uniform scaling is produced that maintains relative object proportions.
- Unequal values for s_x and s_y result in a differential scaling that is often used in design applications, when pictures are constructed from a few basic shapes that can be adjusted by scaling and positioning transformations (Fig. 2.6).

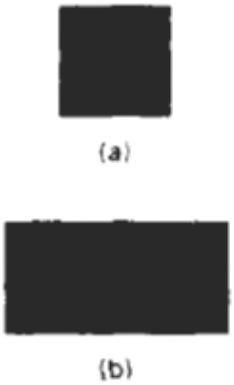


Figure 2.6

Turnmg a square (a) into a rectangle (b) wlth scaling factors s_x = 2 and s_y = 1.

Objects transformed with Eq. 2.11 are both scaled and repositioned. Scaling factors with values less than 1 move objects closer to the coordinate origin, while values greater than 1 move coordinate positions farther from the origin.

Figure 2.7 illustrates scaling a line by assigning the value *0.5* to both s_x and s_y in Eq. 2.11. Both the line length and the distance from the origin are reduced by a factor of 1 /2.

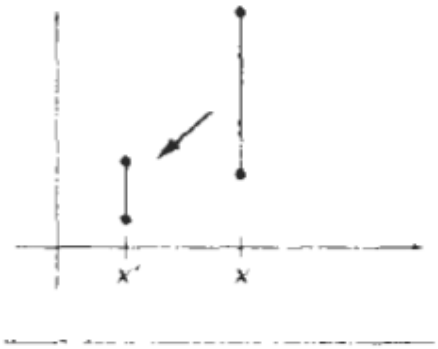


Figure 2.7

A line scaled with Eq 2.12 using $s_x - s_y = 0.5$ is reduced in size and moved closer to the coordinate origin.

We can control the location of a scaled object by choosing a position, called the fixed point that is to remain unchanged after the scaling transformation. Coordinates for the fixed point (x_f, y_f) can be chosen as one of the vertices, the object centroid, or any other position (Fig. 2.8).

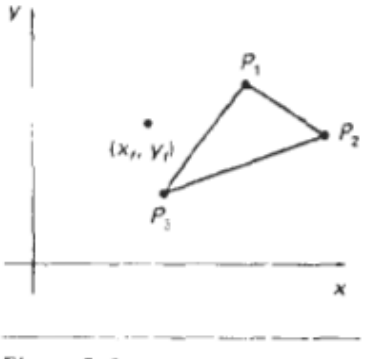


Figure 2.8

Scaling relative to a chosen fixed point (x_f,y_f).Distances from each polygon vertex to the fixed point are scaled by transformation equations 2.13.

A polygon is then scaled relative to the fixed point by scaling the distance from each vertex to the fixed point. For a vertex with coordinates (x, y) the scaled coordinates (x ', y ') are calculated as

$$x' = x_f + (x - x_f)s_x , \qquad y' = y_f + (y - y_f)s_y \qquad 2.13$$

Rewriting these scaling transformations to separate the multiplicative and additive terms:

$$x' = x \cdot s_x + x_f(1 - s_x)$$
$$y' = y \cdot s_y + y_f(1 - s_y) \qquad 2.14$$

where the additive terms $x_f(1 - s_x)$ and $y_f(1 – s_y)$ are constant for all points in the object.

2.3 Matrix Representations and Homogeneous Coordinates

Let P(x,y) be any point in 2-D Euclidean (Cartesian) system.

In Homogeneous Coordinate system, we add a third coordinate to a point. Instead of (x,y), each point is represented by a triple (x,y,H) such that H≠0; with the condition that (x1,y1,H1)=(x2,y2,H2) ↔ x1/H1 = x2/H2 ; y1/H1 = y2/H2.

(Here, if we take H=0, then we have point at infinity, i.e., generation of horizons).

Thus, (2,3,6) and (4,6,12) are the same points are represented by different coordinate triples, i.e., each point has many different Homogeneous Coordinate representation.

2-D Euclidian System	Homogeneous Coordinate System
Any point (x,y) ⟶	(x,y,1)
	If (x,y,H) be any point in HCS(such that H≠0); Then (x,y,H)=(x/H,y/H,1)
(x/H,y/H) ⟵	(x,y,H)

For translation transformation (x,y) ➔ (x+tx,y+ty) in Euclidian system, where tx and ty are the translation factor in x and y direction, respectively.

Unfortunately, this way of describing translation does not use a matrix, so it cannot be combined with other transformations by simple matrix multiplication. Such a combination would be desirable. In homogeneous coordinates we use 3x3 matrices instead of 2x2, introducing an additional dummy coordinate H. Instead of (x,y), each point is represented by a triple (x,y,H) such that H≠0; In two dimensions the value of H is usually kept at 1 for simplicity.

Thus, in HCS (x,y,1) → (x+tx,y+ty,1), now, we can express this in matrix form as:

$$(x',y',1)=(x,y,1)\begin{pmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ t_x & t_y & 1\end{pmatrix}$$

The matrix form of translation simplifies the operations on complex objects.

The basic transformations such as scaling, rotation, reflection, etc., can be expressed as 3x3 homogeneous coordinate matrices.

This can be accomplished by augmenting the 2x2 matrices with a third row (0,0,x) and a third column. That is

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

2.4 Composite Transformations

By calculating the matrix product of the individual transformations we can set up a matrix for any sequence of transformations as a composite transformation matrix. Forming products of transformation matrices is often referred to as a concatenation, or composition, of matrices.

For column-matrix representation of coordinate positions, we form composite transformations by multiplying matrices in order from right to left. That is, each successive transformation matrix pre-multiplies the product of the preceding transformation matrices.

2.4.1 Translations

If two successive translation vectors (t_{x1}, t_{y1}) and (t_{x2}, t_{y2}) are applied to a coordinate position P, the final transformed location P' is calculated as

$$\begin{aligned} \mathbf{P}' &= T(t_{x2}, t_{y2}) \cdot \{\mathbf{T}(t_{x1}, t_{y1}) \cdot \mathbf{P}\} \\ &= \{\mathbf{T}(t_{x2}, t_{y2}) \cdot \mathbf{T}(t_{x1}, t_{y1})\} \cdot \mathbf{P} \end{aligned} \qquad 2.15$$

where P and P are represented as homogeneous-coordinate column vectors.

2.4.2 Rotations

Two successive rotations applied to point p product. the transformed position

$$\begin{aligned} \mathbf{P}' &= \mathbf{R}(\theta_2) \cdot \{\mathbf{R}(\theta_1) \cdot \mathbf{P}\} \\ &= \{\mathbf{R}(\theta_2) \cdot \mathbf{R}(\theta_1)\} \cdot \mathbf{P} \end{aligned} \qquad 2.16$$

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

$$\mathbf{R}(\theta_2) \cdot \mathbf{R}(\theta_1) = \mathbf{R}(\theta_1 + \theta_2) \qquad 2.17$$

so that the final rotated coordinates can be calculated with the composite rotation matrix as

$$\mathbf{P}' = \mathbf{R}(\theta_1 + \theta_2) \cdot \mathbf{P} \qquad 2.18$$

2.4.3 Scaling

Concatenating transformation matrices for two successive scaling operations produces the following composite scaling matrix:

$$\begin{bmatrix} s_{x2} & 0 & 0 \\ 0 & s_{y2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} s_{x1} & 0 & 0 \\ 0 & s_{y1} & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} s_{x1} \cdot s_{x2} & 0 & 0 \\ 0 & s_{y1} \cdot s_{y2} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Or

$$\mathbf{S}(s_{x2}, s_{y2}) \cdot \mathbf{S}(s_{x1}, s_{y1}) = \mathbf{S}(s_{x1} \cdot s_{x2}, s_{y1} \cdot s_{y2}) \qquad 2.19$$

The resulting matrix in this case indicates that successive scaling operations are multiplicative. That is, if we were to triple the size of an object twice in succession, the final size would be nine times that of the original.

2.4.5 General Pivot-Point Rotation

With a graphics package that only provides a rotate function for revolving objects about the coordinate origin, we can generate rotations about any selected pivot point (x, y,) by performing the following sequence of translate-rotate translate-operations:

1. Translate the object so that the pivot-point position is moved to the coordinate origin.

2. Rotate the object about the coordinate origin.

3. Translate the object so that the pivot point is returned to its original position.

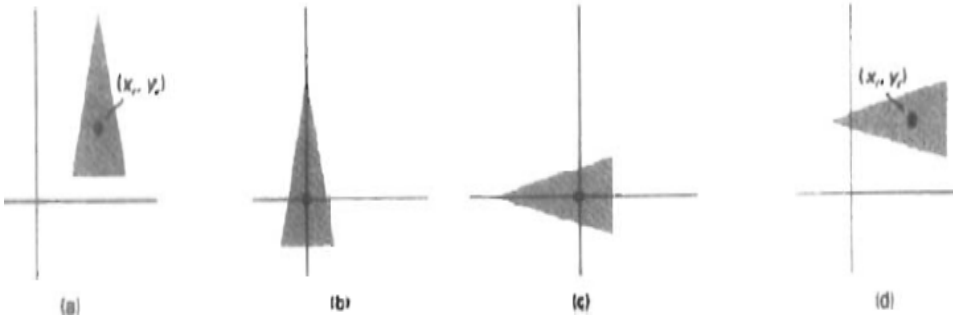


Figure 2.9

A transformation sequence for rotating an object about a specified pivot mint using the rotation matrix R(Θ)

a. Original position of object and pivot point

b. Translation of object so that pivot point is at origin

c. Rotation about object

d. Translation of object so that pivot point is returned to actual as in (a)

2.4.6 Concatenation Properties

Matrix multiplication is associative. For any three matrices, A, B, and C, the matrix product A . B . C can be performed by first multiplying A and B or by first multiplying B and C:

$$\mathbf{A} \cdot \mathbf{B} \cdot \mathbf{C} = (\mathbf{A} \cdot \mathbf{B}) \cdot \mathbf{C} = \mathbf{A} \cdot (\mathbf{B} \cdot \mathbf{C}) \qquad 2.20$$

Therefore, we can evaluate matrix products using either a left-to-right or a right to left associative grouping.

2.5 Other Transformations

Other transformations used are reflection and shear.

We can build complex transformations such as rotation about an arbitrary point, mirror reflection about a line, etc., by multiplying the basic matrix transformations called concatenation of matrices.

The resulting matrix is called the composite transformation matrix.

2.5.1 Rotation about a Point

Given a 2-D point P(x,y), which we want to rotate, with respect to an arbitrary point A(h,k). Let P'(x',y') be the result of anticlockwise rotation of point P by angle θ about A, which is shown in Figure 2,9.

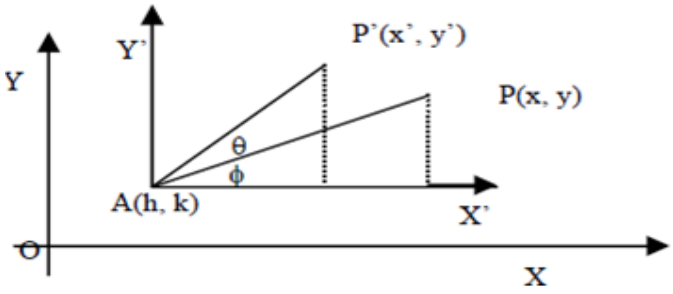


Figure 2.10

Rotation matrix R_θ is defined only with respect to the origin.

The basic transformation which constitutes the composite transformation to compute the rotation about a given arbitrary point A, denoted by $R_{\theta,A}$ can be determined in three steps:

1) Translate the point A(h,k) to the origin O, so that the center of rotation A is at the origin.

2) Perform the required rotation of θ degrees about the origin, and

3) Translate the origin back to the original position A(h,k).

Using v=hI+kJ as the translation vector, we have the following sequence of three transformations:

$$R_{\theta,A} = T_{-v}.\ R_\theta.\ T_v$$

$$= \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -h & -k & 1 \end{pmatrix} \begin{pmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ h & k & 1 \end{pmatrix}$$

$$= \begin{pmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ (1-\cos\theta).h+k.\sin\theta & (1-\cos\theta).k-h.\sin\theta & 1 \end{pmatrix} \quad 2.15$$

2.5.2 Reflection about a Line

Reflection is a transformation which generates the mirror image of an object. As discussed in the previous block, the mirror reflection helps in achieving 8-way symmetry for the circle to simplify the scan conversion process. For reflection we need to know the reference axis or reference plane depending on whether the object is 2-D or 3-D.

Let the line L be represented by y=mx+c, where 'm' is the slope with respect to the x axis, and 'c' is the intercept on y-axis, as shown in Figure 2.11. Let P'(x',y') be the mirror reflection about the line L of point P(x,y).

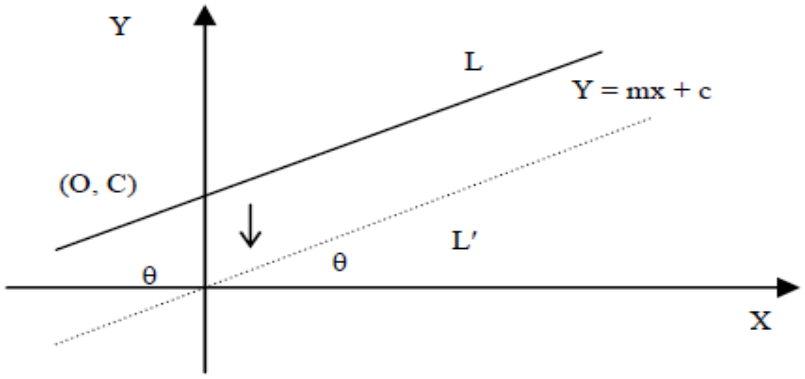


Figure 2.11

The transformation about mirror reflection about this line L consists of the following basic transformations:

1) Translate the intersection point A(0,c) to the origin, this shifts the line L to L'.

2) Rotate the shifted line L' by –θ degrees so that the line L' aligns with the x-axis.

3) Mirror reflection about x-axis.

4) Rotate the x-axis back by θ degrees

5) Translate the origin back to the intercept point (0,c).

In transformation notation, we have

$M_L = T_{-v}.R_{-\theta}.M_X.R_{\theta}.T_v$, where v=0I+cJ

$$M_L = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -c & 1 \end{pmatrix} \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & c & 1 \end{pmatrix}$$

$$= \begin{pmatrix} \cos^2\theta - \sin^2\theta & 2.\cos\theta.\sin\theta & 0 \\ 2.\sin\theta.\cos\theta & \sin^2\theta - \cos^2\theta & 0 \\ -2.c.\sin\theta.\cos\theta & -c.(\sin^2\theta - \cos^2\theta)+c & 1 \end{pmatrix}$$

2.16

Let $\tan\theta=m$, the standard trigonometry yields $\sin\theta=m/\sqrt{(m^2+1)}$ and $\cos\theta= 1/\sqrt{(m^2+1)}$. Substituting these values for $\sin\theta$ and $\cos\theta$ in the Eq 2.16 , we have:

$$M_L = \begin{pmatrix} (1-m^2)/(m^2+1) & 2m/(m^2+1) & 0 \\ 2m/(m^2+1) & (m^2-1)/(m^2+1) & 0 \\ -2cm/(m^2+1) & 2c/(m^2+1) & 1 \end{pmatrix}$$

2.17

2.5.3 Special cases

1) If we put c = 0 and $m=\tan\theta=0$ in the Eq 2.17 then we have the reflection about the line y = 0 i.e. about x-axis. In matrix form:

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

2.18

2) If c = 0 and $m=\tan\theta=\infty$ then we have the reflection about the line x=0 i.e. about y-axis. In matrix form:

$$My=\begin{pmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \qquad 2.19$$

3) To get the mirror reflection about the line y = x, we have to put m=1 and c=0. In matrix form:

$$M_{y=x}=\begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \qquad 2.20$$

4) Similarly, to get the mirror reflection about the line y = – x, we have to put m = –1 and c = 0. In matrix form:

$$M_{y=-x}=\begin{pmatrix} 0 & -1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \qquad 2.21$$

5) The mirror reflection about the Origin (i.e., an axis perpendicular to the xy plane and passing through the origin).

$$M_{org}=\begin{matrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{matrix} \qquad 2.22$$

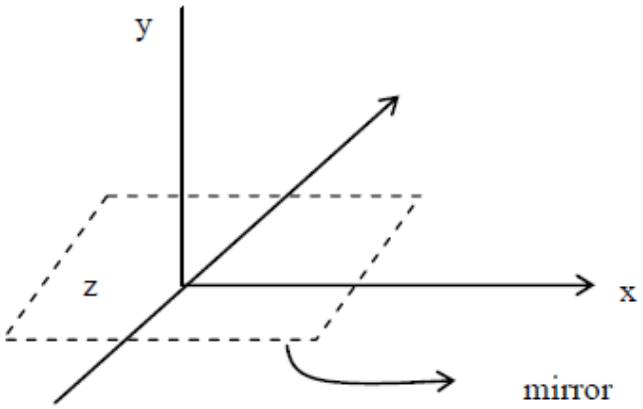


Figure 2.12

2.6 Viewing Pipline

A view is selected by specifying a subarea (single area for display, or several areas of the total picture area). The picture parts within the selected areas are then mapped onto specified areas of the device coordinates. When multiple view areas are selected, these areas can be placed in separate display locations, or some areas could be inserted into other, larger display areas. Transformations from world to device coordinates involve translation, rotation, and scaling operations, as well as procedures for deleting those parts of the picture that are outside the limits of a selected display area.

A world-coordinate area selected for display is called a **window**. The window defines *what* is to be viewed;

An area on a display device to which a window is mapped is called a **viewport**. The viewport defines *where* it is to be displayed. Viewports are typically defined within the unit square (normalized coordinates).

In general, the mapping of a part of a world-coordinate scene to device coordinates is referred to as a viewing transformation. Figure 2.13 illustrates the mapping of a picture

section that falls within a rectangular window onto a designated &angular viewport.

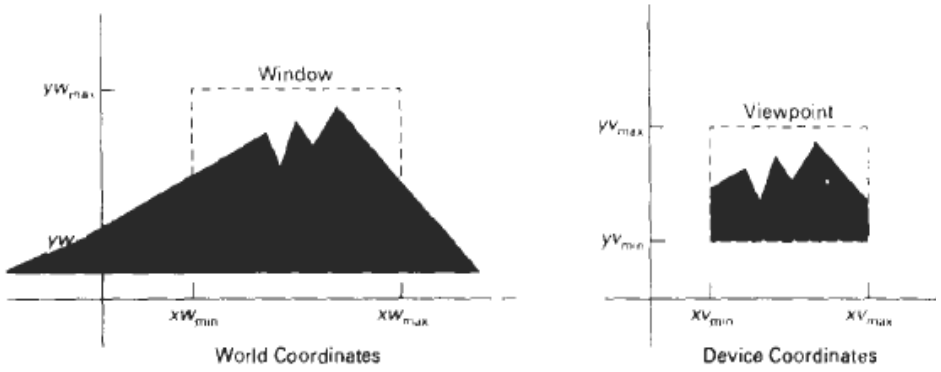


Figure 2.13

A viewing transformation using standard rectangles the window and viewport.

A1I parts of the picture that he outside the viewport are clipped, and the contents of the viewport are transferred to device coordinates.

We achieve zooming effects by successively mapping different-sized windows on a fixed-size viewport. As the windows are made smaller, we zoom in on some part of a scene to view details that are not shown with larger windows.

More overview is obtained by zooming out from a section of a scene with successively larger windows.

Panning effects are produced by moving a fixed-size window across the various objects in a scene.

Clipping procedures are of fundamental importance in computer graphics. They are used not only in viewing transformations, but also in window-manager systems, in painting and drawing packages to eliminate parts of a picture

inside or outside of a designated screen area, and in many other applications.

2.7 Window to Viewpoint coordinate Transformation

Once object descriptions have been transferred to the viewing reference frame, choose the window extents in viewing coordinates and select the viewport limits in normalized coordinates**.** Object descriptions are then transferred to normalized device coordinates.

Figure 2.14 illustrates the window-to-viewport mapping.

A point at position *(xw,yw)* in the window 1s mapped into position *(xv, yv)* in the associated viewport.

$$\frac{xv - xv_{min}}{xv_{max} - xv_{min}} = \frac{xw - xw_{min}}{xw_{max} - xw_{min}}$$

$$\frac{yv - yv_{min}}{yv_{max} - yv_{min}} = \frac{yw - yw_{min}}{yw_{max} - yw_{min}} \qquad 2.23$$

Figure 2.14

A point at position *(xw,yw)* in a designated window is mapped to view port coordinates *(xv,* yv) so that relative positions in the two areas are the same.

Solving these expressions for the viewport position *(xv, yv),* we have

$$xv = xv_{min} + (xw - xw_{min})sx$$
$$yv = yv_{min} + (yw - yw_{min})sy \qquad 2.24$$

where the scaling factors are

$$sx = \frac{xv_{max} - xv_{min}}{xw_{max} - xw_{min}}$$
$$sy = \frac{yv_{max} - yv_{min}}{yw_{max} - yw_{min}} \qquad 2.25$$

Equations 2.24 can also be derived with a set of transformations that converts the window area into the viewport area in the following steps:

1. Perform a scaling transformation using a fixed-point position of (xw_{min}, yw_{min}) that scales the window area to the size of the viewpoint.

2. Translate the scaled window area to the position of the viewport.

2.8 Clipping Operations

The operation that identifies those portions of a picture that are either inside or outside of a specified region of space is referred to as a clipping algorithm, or simply clipping.

The region against which an object is to clipped is called a clip window.

Depending on the application, the clip window can be a general polygon or it can even have curved boundaries.

Applications of clipping:

- extracting part of a defined scene for viewing;

- identifying visible surfaces in three-dimensiona1 views; antialiasing line *seg*ments or object boundaries;
- creating objects using solid-modeling procedures;
- displaying a multiwindow environment;
- drawing and painting operations that allow parts of a picture to be selected for copying, moving, erasing, or duplicating.

Clipping methods using rectangular clip regions:

For the viewing transformation, to display only those picture parts that are within the window area, everything outside the window is discarded. Clipping algorithms can be applied in world coordinates, so that only the contents of the window interior are mapped to device coordinates.

On other hand, the complete world-coordinate picture can be mapped first to device coordinates, or normalized device coordinates, then clipped against the viewport boundaries. It clipping removes those primitives outside the window from further consideration, thus eliminating the processing necessary to transform those primitives to device space.

Viewport clipping, reduce calculations by allowing concatenation of viewing and geometric transformation matrices.

Clipping algorithms for the following primitive types are discussed in this section:

- Point Clipping
- Line Clipping (straight-line segments)
- Area Clipping (polygons)

- Curve Clipping
- Text Clipping

Line and polygon clipping routines are standard components of graphics packages, but many packages accommodate curved objects, particularly spline curves and conics, such as circles and ellipses. Another way to handle curved objects is to approximate them with straight-line segments and apply the line or polygonclipping procedure.

Figure 2.16 illustrates possible relationships between line positions and a standard rectangular clipping region.

2.8.1 Line Clipping

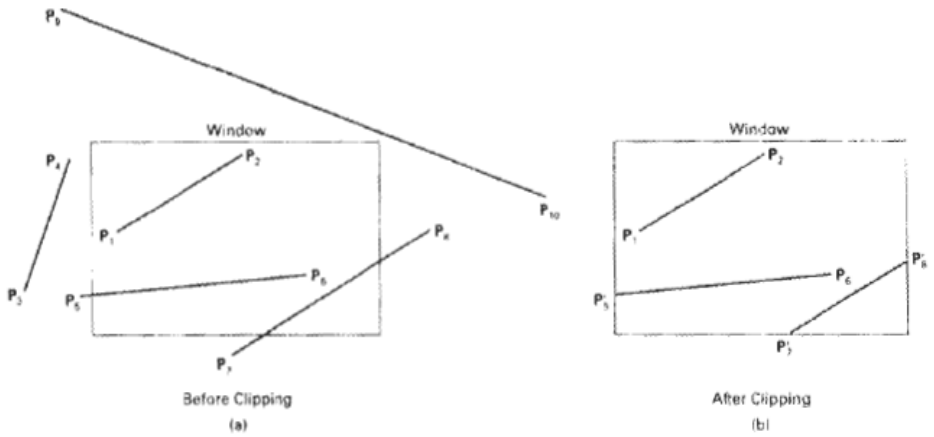


Figure 2.16

Line clipping against a rectangularship window.

A lineclipping procedure involves several parts:

- Test a given line segment to determine whether it lies completely inside the clipping window.

- If it does not, determine whether it lies completely outside the window.
- If a line cannot be identified as completely inside or completely outside, perform intersection calculations with one or more clipping boundaries.

Process lines through the "inside-outside'' tests by checking the line endpoints. A line with both endpoints inside all clipping boundaries, such as the line from P_1 to P_2, is saved. A line with both endpoints outside any one of the clip boundaries is outside the window. All other lines cross one or more clipping boundaries, and may require calculation of multiple intersection points.

For a line segment with endpoints (x_1, y_1) and (x_2, y_2) and one or both endpoints outside the clipping rectangle, the parametric representation

$$x = x_1 + u(x_2 - x_1)$$
$$y = y_1 + u(y_2 - y_1), \qquad 0 \le u \le 1$$

could be used to determine values of parameter 11 for intersections with the clipping boundary coordinates. If the value of *u* for an intersection with a rectangle boundary edge is outside the range *0* to 1, the line does not enter the interior of the window at that boundary. If the value of *u* is within the range from 0 to 1, the line segment does indeed cross into the clipping area. This method can be applied to each clipping boundary edge in turn to determine whether any part of the line segment is to be displayed. Line segments that are parallel to window edges can be handled as specia1 cases.

2.8.2 Cohesion-Sutherland Line Clipping

Every line end point in a picture is assigned a four-digit binary code, called a region code, identifies the location of the point relative to the boundaries of the clipping rectangle. This method speeds up the processing of line segments by performing initial

tests that reduce the number of intersections that must be calculated. Regions are set up in reference to the boundaries as shown in Fig. 2.17.

Each bit position in the region code is used to indicate one of the four relative coordinate positions of the point with respect to the clip window: to the left, right, top, or bottom. By numbering the bit positions in the region code as 1 through 4 from right to left, the coordinate regions can be correlated with the bit positions as

bit 1: left

bit 2: right

bit 3: below

bit 4: above

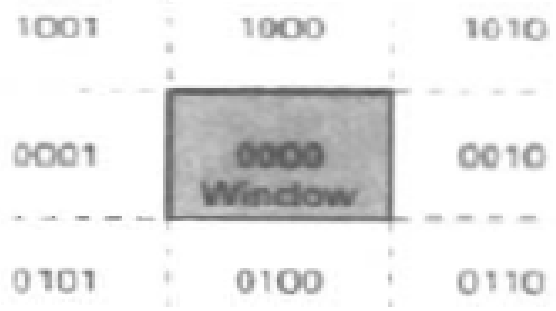


Figure 2.17

Binary region codes assigned to line endpoints according to relative position with respect to the clipping rectangle.

A value of 1 in any bit position indicates that the point is in that relative position; otherwise, the bit position is set to 0. If a point is within the clipping rectangle, the region code is 0000. A point that is below and to the left of the rectangle has a region code of 0101.

Bit values in the region code are determined by comparing endpoint coordinate values (x, y) to the clip boundaries. Bit 1 is set to 1 if $x < xw_{min}$.

The other three bit values can be determined using similar comparisons. For languages in which bit manipulation is possible, region-code bit values can be determined with the following two steps:

(1) Calculate differences between endpoint coordinates and clipping boundaries.

(2) Use the resultant sign bit of each difference calculation to set the corresponding value in the region code.

Bit 1 is the sign bit of x - bit 2 is the sign bit of $xw_{max} - x$;

bit 3 is the sign bit of $y - yw_{min}$;

bit 4 is the sign bit of $yw_{max} - y$.

Once region codes are established for all line endpoints, determine which lines are completely inside the clip window and which are clearly outside.

- Any lines that are completely contained within the window boundaries have a region code of 0000 for both endpoints, and trivially accept these lines.
- Any lines that have a 1 in the same bit position in the region codes for each endpoint are completely outside the clipping rectangle, and trivially reject these lines.
- Discard the line that has a region code of 1001 for one endpoint and a code of 0101 for the other endpoint.

Both endpoints of this line are left of the clipping rectangle, as indicated by the 1 in the first bit position of each region code.

A method that can be used to test lines for total clipping is to perform the logical and operation with both region codes. If the

result is not 0000, the line is completely outside the clipping region.

Lines that cannot be identified as completely inside or completely outside a clip window by these tests are checked for intersection with the window boundaries. As shown in Fig. 6-9, such lines may or may not cross into the window interior. Clip a line by comparing an outside endpoint to a clipping boundary to determine how much of the line can be discarded.

Then the remaining part of the Line is checked against the other boundaries, and we continue until either the line is totally discarded or a section is found inside the window. Set the algorithm to check line endpoints against clipping boundaries in the order left, right, bottom, top.

To illustrate the specific steps in clipping lines against rectangular boundaries using the Cohen-Sutherland algorithm, Fig. 2.18 could be processed.

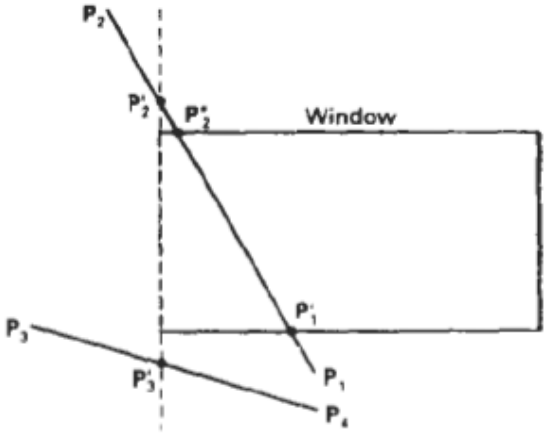


Figure 2.18

Lines extending from one coordinate region to another may pass through the clip window, or they may intersect clipping boundaries without entering the window.

- Starting with the bottom endpoint of the line from P_1 to P_2.
- Check P, against the left, right, and bottom boundaries in turn and find that this point is below the clipping rectangle.
- Find the intersection point Pi with the bottom boundary and discard the line section from P_1 to P_1'.
- Reduce the line to the section from P_1' to P_2.
- Since P_2 is outside the clip window, check this endpoint against the boundaries and find that it is to the left of the window.
- Intersection point P_2' is calculated, but this point is above the window.
- The final intersection calculation yields P_2", and the line from P_1' to P_2' is saved.
- Point P_3 in the next line is to the left of the clipping rectangle, so we determine the intersection P_3' and eliminate the line section from P_3 to P_3'. By checking region codes for the line section from P_3' to P_4, Find that the remainder of the line is below the clip window and can be discarded also.

Intersection points with a clipping boundary can be calculated using the slope-intercept form of the line equation. For a line with the endpoint coordinates *(x1* ,y1,) and *(x2, y2),* they

coordinate of the intersection point with a vertical boundary can be obtained with the calculation

$$y = y_1 + m(x - x_1) \qquad 2.26$$

where the ***x*** value is set either to xw_{min} or to xw_{max} and the slope of the line is calculated as m = *(y2 - y 1) / (x2 - x1).* Similarly, if we are looking for the intersection with a horizontal boundary, the ***x*** coordinate can be calculated as

$$x = x_1 + \frac{y - y_1}{m} \qquad 2.27$$

with y set either to yw_{min} or to $yw_{max.}$

2.9 Polygon Clipping

A polygon boundary processed with a line clipper may be displayed as a series of unconnected line segments (Fig. 2.18), depending on the orientation of the polygon to the clipping window. What we really want to display is a bounded area after clipping, as in Fig. 2.19. For polygon clipping, we require an algorithm that will generate one or more closed areas that are then scan converted for the appropriate area fill. The output of a polygon clipper should be a sequence of vertices that defines the clipped polygon boundaries.

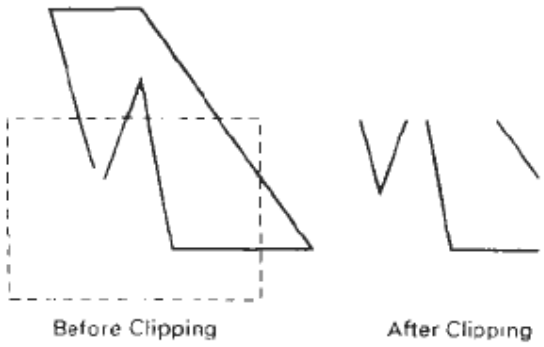


Figure 2-19 a

Display of a polygon processed by a line-dipping algorithm

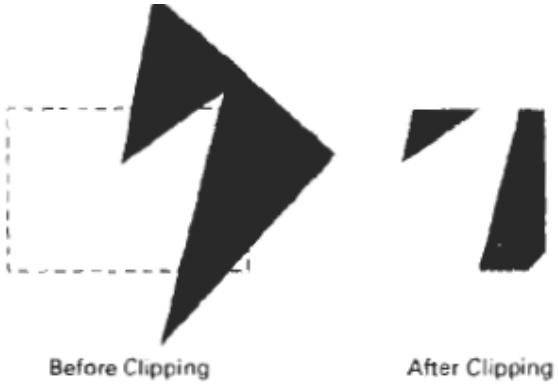


Figure 2.19 b

Display of a correctly clipped polygon

2.9.1 Sutherland-Hodgeman polygon Clipping

Clipping a polygon can be correctly done by processing the polygon boundary as a whole against each window edge by processing all polygon vertices against each clip rectangle boundary in turn. Beginning with the initial set of polygon vertices, clip the polygon against the left rectangle boundary to produce a new sequence of vertices.

The new set of vertices could then k successively passed to a right boundary clipper, a bottom boundary clipper, and a top boundary clipper, as in Fig. 2.20.

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

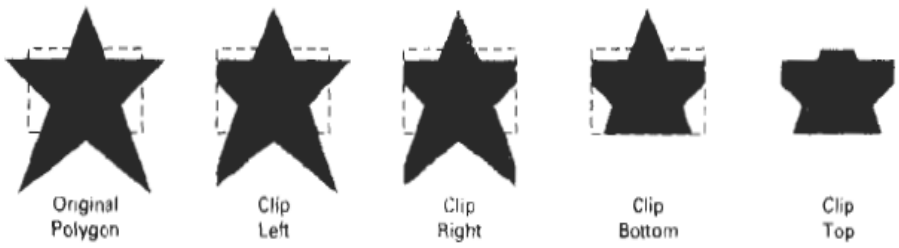


Figure. 2.20

Clipping polygon against successive window boundaries.

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

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

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

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

(4) If both input vertices are outside the window boundary, nothing is added to the output list.

These four cases are illustrated in Fig. 2.21 for successive pairs of polygon vertices. Once all vertices have been processed for one clip window boundary, the output 11st of vertices is clipped against the next window boundary.

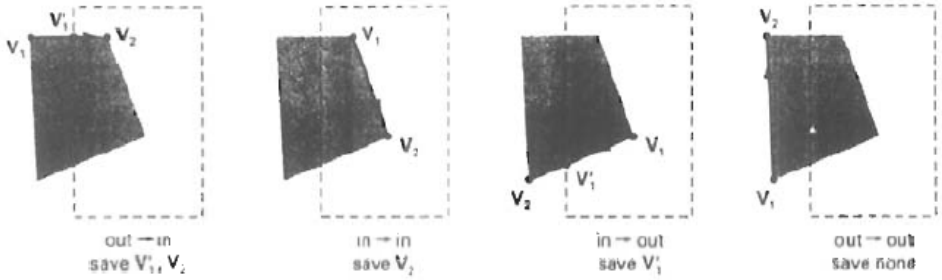


Figure 2.21

Successive processing of pairs of polygon vertices against the left window boundary.

Text Book:

Donald Hearn & Pauline M.Baker – Computer Graphics 2nd Edition –PHI.

Reference Book:

1.Foley, Van Dam, Feiner and Hughes – Computer Graphics Principles and Practice 3rd Edition.

2.N.Krishnamurthy - Introduction to Computer Graphics.

UNIT – III

GRAPHICAL USER INTERFACES & INTERACTIVE INPUT METHODS

Objectives

After completing this unit, you should be able to:

- Describe logical classification of input devices;
- Discuss input functions, input modes;
- Describe Interactive picture construction techniques;
- Discuss various 3D Concepts and Object Representations;
- Discuss various projections;

3.1 Input of Graphical Data

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.

3.2 Logical classification of Input devices

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

- LOCATOR-a device for specifying 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 value:
- CHOICE-a device for selecting menu options
- PICK-a device selecting picture components

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 tor entering coordinate data than a keyboard, for example.

In the following sections we discuss how the various physical are used to provide input with each of the logical classification.

3.2.1 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, thumbwheels, 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 selected cursor key. Alternatively, a joystick, joydisk, 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. .4s 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.

3.2.2 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.

3.2.3 String Devices:

The primary physical device used for string input is the keyboard. lnput 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.

3.2.4 Valuator Devices

This logical class of devices is employed in graphic systems to input scalar values. Valuators are used for setting various graphics parameters, such as rotation angle and sdle factors, afd for setting physical parameters associated with a particular application (temperature 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 with in 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 toa scalar range. For one direction of movement, say, let3 to right, increasing scalar values can be input. Movement in the opposite direction decreases the scalar input value.

Another technique for providing valuator input is to display sliders, buttons, rotating scales, and menus on the video monitor. Figure 3.1 illustrates some possibilities for scale representations. Locator input from a mouse, joystick, spaceball, or other device is used to select a coordinate position on the display, and the screen coordinate position is then converted to a numeric input value. As a feedback mechanism for the w r , the selected position on a scale can be marked with some symbol. Numeric values may also be echoed somewhere onthe screen to confirm the selections.

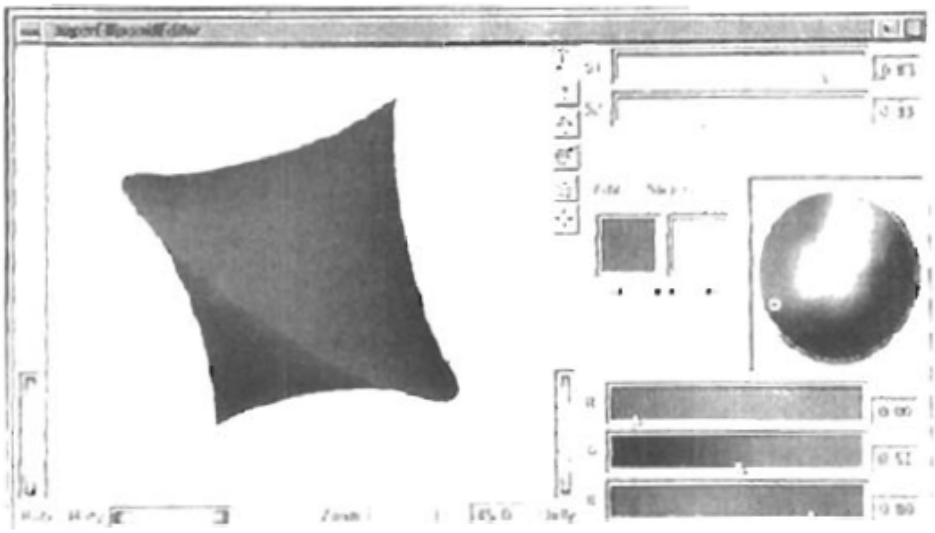


Figure 3.1

Scales displayed on a video monitor for interactive selection of parameter values. In this display, sliders are provided for selecting scalar values for super ellipse parameters, sl and s2 and for individual R, G, and B color values. In addition, a small circle can be positioned on the color wheel for selection of a combined RGB color, and buttons can be activated to make small changes in color values.

3.2.5 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} and y_{min}, y_{max} is selected if the input coordinates (x, y) satisfy the inequalities.

$$x_{min} \leq x \leq x_{max}, \qquad y_{min} \leq y \leq y_{max} \qquad 3.1$$

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).

3.2.6 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 object (if any) that is to be elected.

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.

One way to find the closest line to the cursor position is to calculate the distance squared from the cursor coordinates (x, y) to each line segment whose bounding rectangle contains the cursor position (Fig. 3.2. For a line with endpoints (x_1, y_1) and (x_2, y_2) distance squared from *(x,* y) to the line is calculated as

$$d^2 = \frac{[\Delta x(y - y_1) - \Delta y(x - x_1)]^2}{\Delta x^2 + \Delta y^2} \qquad 3.2$$

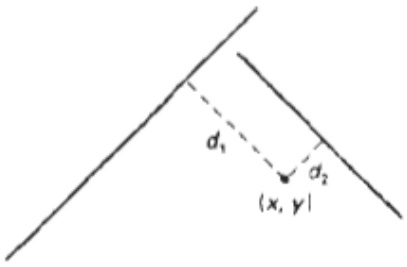


Figure 3.2

Distances to line segments from the pick position

where $\Delta x = x_2 - x_1$, and $\Delta y = y_2 - y_1$

Various approximations can be used to speed up this distance calculation, or other identification schemes can be used.

Another method for finding the closest line to the cursor position is to specify the size of a pick window. The cursor coordinates are centered on this window and the candidate lines are clipped to the window, as shown in Fig. 3.3. By making the pick window small enough, we can ensure that a single line will cross the window.

A method for avoiding the calculation of pick distances or window clipping intersections is to highlight the candidate structures and allow the user to resolve he pick ambiguity. One way to do this is to highlight [he structures that overlap the cursor position one by one. The user then signals when the desired structure is highlighted.

An alternative to cursor positioning is to use button input to highlight successive structures. A second button is used to stop the process when the desired structure is highlighted. I t very many structures are to he searched in this way, the process can be speeded up and an additional button is used to help identify the structure. The first button can initiate a rapid successive highlighting of structures. A second button call again be used to stop the process, and a third button can be used to back up more slowly if the desired structure passed before the operator pressed the stop button.

Finally, we could use a keyboard to type in structure names. This is a straightforward, but less interactive, pick-selection method. Descriptive names can be used to help the user in the pick process, but the method has several drawbacks. It is generally slower than interactive picking on the screen, and a user will probably need prompts to remember the various structure names. In addition, picking structure

subparts from the keyboard can be more difficult than picking the subparts on the screen.

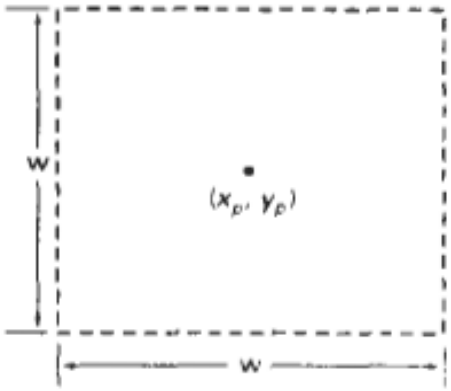


Figure 3.3

A pick window centered on pick coordinates (x_p, y_p) used to resolve pick object overlaps

3.3 Input Functions

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.

3.4 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 progtam 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 (us, deviceCode, inputMode. echoFlag)

where device code is a pos~tivei nteger; inputMode is assigned one of the values: request, sample, or event; and parameter echoFlag is assigned either the value echo or the value no echo.

Device code assignment is installation-dependent.

3.4.1 Request Mode

Input commands used in this mode correspond to standard input functions in a high-level programming language. When we ask for an input in request mode, other processing is suspended until the input is received. After a device has been assigned to request mode. as discussed in the preceding section, input requests can be made to that device using one of the six logical-class functions represented by the following:

request . . . (ws, devicecode, status . . . . 1)

Values input with this function are the workstation code and the device code. Returned values are assigned to parameter status and to the data parameters corresponding to the requested logical class.

A value of ok or none is returned in parameter status, according to the validity of the input data. A value of none indicates that the input device was activated so as to produce invalid data. For locator input, this could mean that the coordinates were out of range. For pick input, the device could have been activated while not pointing at a structure. Or a

"break" button on the input device could have been pressed. A returned value of *none* can be used as an end-of-data signal to terminate a programming sequence.

3.4.2 Sample Mode

Once sample mode has been set for one or more physical devices, data input begins without waiting for program direction. If a joystick has been designated as a locator device in sample mode, coordinate values for the current position of the activated joystick are immediately stored. As the activated stick position changes, the stored values are continually replaced with the coordinates of the current stick position.

Sampling of the current values from a physical device in this mode begins when a sample command is encountered in the application program. A locator device is sampled with one of the six logical-class functions represented by the following:

sample . . . (ws, devicecode, .)

Some device classes have a status parameter in sample mode, and some do not.

Other input parameters are the same as in request mode.

As an example of sample input, suppose we want to translate and rotate a selected object. A final translation position for the object can be obtained with a locator, and the rotation angle can be supplied by a valuator device, as demonstrated in the following statements.

samplelocacor (wsl, devl, viewIndex, p:)

samplevalt~ator (ws2. dev2, angle)

3.4.5 Event Mode

When an input device is placed in event mode, the program and device operate simultaneously. Data input from the device is accumulated in an event queue, or input queue.

All input devices active in event mode can enter data (referred to as "events") into this single-event queue, with each device entering data values as they are generated. At any one time, the event queue can contain a mixture of data types, in the order they were input. Data entered into the queue are identified according to logical class, workstation number, and physical-device code.

An application program can be directed to check the event queue for any input with the function

awaitEvent (time, ws, devic eClass, deviceCode)

Parameter time is used to set a maximum waiting time for the application program. It the queue happens to be empty, processing is suspended until either the number of seconds specified in time has elapsed or an input arrives. Should the waiting time run out before data values are input, the parameter *deviceClass* is assigned the value none. When t i m e is given the value 0, the program checks the queue and immediately returns to other processing if the queue is empty.

If processing is directed to the event queue with the *awaitEvent* function and the queue is not empty, the first event in the queue is transferred to a current event record. The particular logical device class, such as locator or stroke, that made this input, is stored in parameter *deviceClass*. Codes, identifying the particular workstation and physical device that made the input, are stored in parameters

ws and *deviceCode*, respectively.

To retrieve a data input from the current event record, an event-mode input function is used. The functions in event mode are similar to those in request and sample modes. However, no workstation and device-code parameters are necessary in the commands, since the values for these parameters are stored in the data record. A user retrieves data with

get ... (. . .)

3.5 Interactive Picture Construction Techniques

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.

3.5.1 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. 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.

3.5.2 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, shown in Figs. 3.4 and 3.5, 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 45", and input coordinates could be constrained to lie along predefined paths, such as circular arcs.

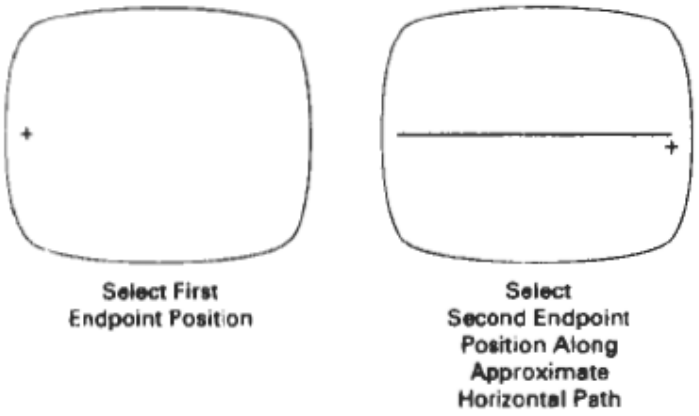


Figure 3.4

Horizontal line constraint

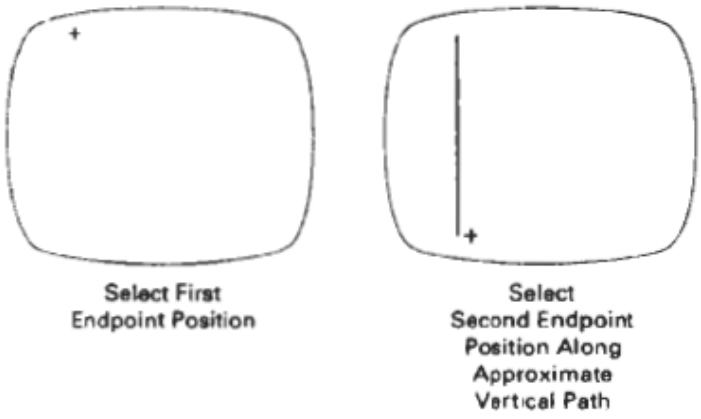


Figure 3.4

Vertical line constraint

3.5.3 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 3.5 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.

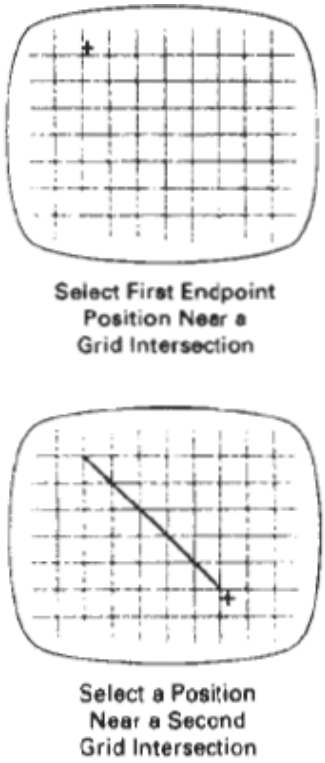


Figure 3.5

Line drawing using a grid.

3.5.4 Gravity Field

In the construction of figures, we sometimes need to connect lines at positions 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 Fig. 3.6. Areas 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

Figure 3.6

Gravity field around a line. Any selected point in tne shaded area is shifted to the line

3.5.5 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 3.7 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. Figure 3.8 demonstrates rubber-band construction of a rectangle, and Fig3.9 shows a rubber-band circle construction.

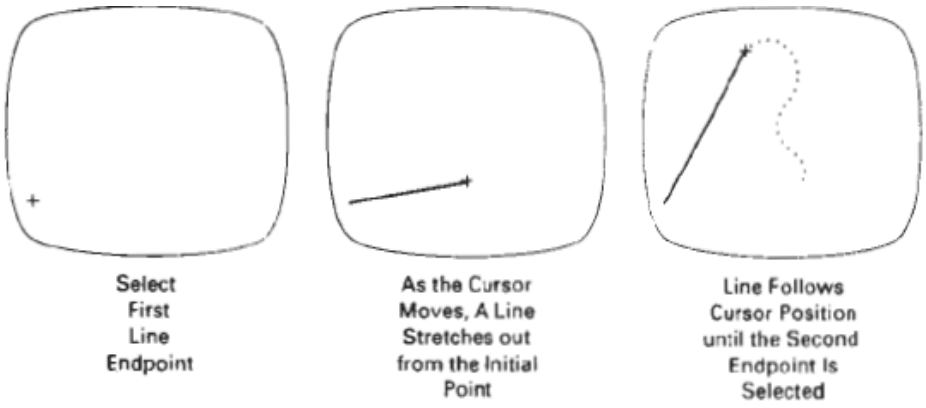


Figure 3.7

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

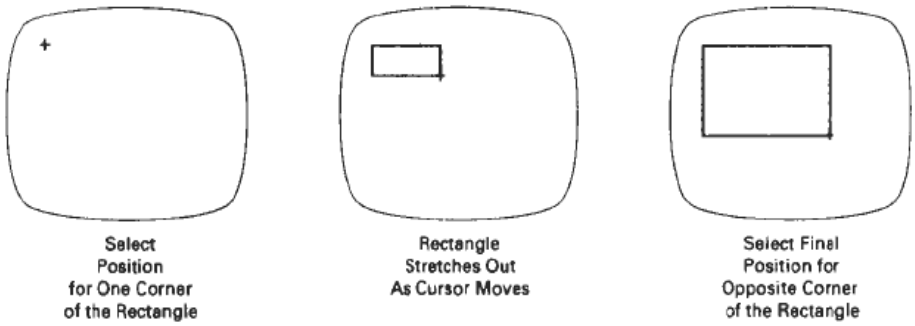


Figure 3.8

Rubber-band method for constructing a rectangle

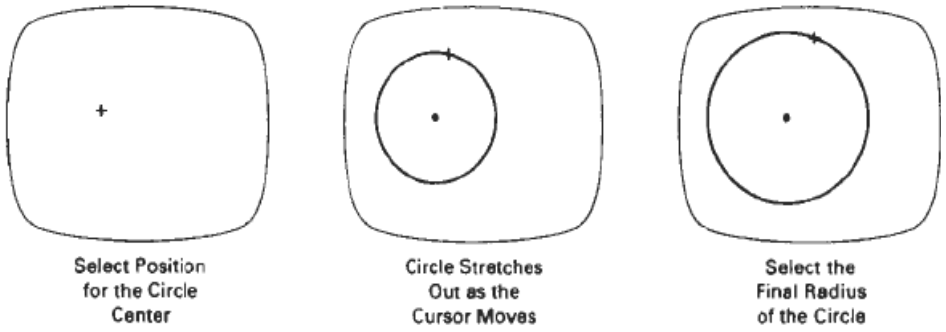


Figure 3.9

Constructing a circle using a rubber-band method.

3.5.6 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, 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.

3.5.7 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. Curvedrawing 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.

Line widths, line styles, and other attribute options are also commonly found in -painting and drawing packages various brush styles, brush patterns, color combinations, object shapes, and surface-texture patterns are also available on many systems, particularly those designed as artist's workstations. Some paint systems vary the line width and brush strokes according to the pressure of the artist's hand, on the stylus. Fig 3.10 shows a window and menu system used with a painting package that allows an artist to select variations of a specified object shape, different surface textures, and a variety of lighting conditions for a scene.

Figure 3.10

A screen layout showing one type of interface to an artist's painting package

3.6 Three-dimensional display methods:

To obtain a display of a three-dimensional scene that has been modeled in world coordinates. We must first set up a

coordinate reference for the "camera". This coordinate reference defines the position and orientation for the plane ot the camera film, which is the plane we want to display a view of the objects in the scene. Object descriptions are then translated to the camera reference coordinates and projected onto the selected display planar we can then display the objects in wireframe (outline) form or we can apply lighting and surface rendering techniques to shade the visible surfaces.

3.6.1 Parallel Projection:

One method for generating a view of a solid object is to project points on the object surface along parallel lines onto the display plane. By selecting different viewing positions, we can project visible points on the object onto the display plane to obtain different two-dimensional views of the object.

In a parallel projection**,** parallel lines in the world-coordinate scene projected into parallel lines on the two-dimensional display plane. This technique is used in engineering and architectural drawings to represent an object with a set of views that maintain relative proportions of the object. The appearance of the solid object can then be re-constructed from the maps views.

3.6.2 Perspective Projection:

Another method for generating a view of a three-dimension is to project Methods points to the display plane along converging paths. This causes objects farther from the viewing position to be displayed smaller than objects of the same size that are nearer to the viewing position. In a perspective projection, parallel lines in a scene that are not parallel to the display plane are projected into converging lines. Scenes displayed using perspective projections appear more realistic, since this is the way that our eyes and a camera lens form images.

3.6.3 Visible Line and Surface Identification

We can also clarify depth relationships in a wireframe display by identifying visible lines in some way. The simplest method is to highlight the visible lines or to display them in a different color. Another technique, commonly used for engineering drawings, is to display the non-visible lines as dashed lines.

Another approach is to simply remove the non-visible lines. But removing the hidden lines also removes information about the shape of the back surfaces of an object. These visible-line methods also identify the visible surface of objects.

When objects are to be displayed with color or shaded surfaces, we apply surface-rendering procedures to the visible surfaces so that the hidden surfaces are obscured. Some visible surface algorithms establish visibility pixel by pixel across the viewing plane; other algorithms determine visibility for object surfaces as a whole.

3.7 Polygon surfaces:

The most commonly used boundary =presentation for a three-dimensional graphics object is a set of surface polygons that enclose the object interior. Many graphics systems store all object descriptions as sets of surface polygons. This simplifies and speeds up the surface rendering and display of objects, since all surfaces are described with linear equations. For this reason, polygon descriptions are often referred to as "standard graphics objects."

In some cases, a polygonal representation is the only one available, but many packages allow objects to be described with other schemes, such as spline surfaces, that are then converted to polygonal representations for processing.

A polygon representation for a polyhedron precisely defines the surface features of the object. But for other objects, surfaces are tiled to produce

the polygon-mesh approximation. In Fig. 3-11, the surface of a cylinder is represented as a polygon mesh. Such representations are common in design and solid-modeling applications, since the wireframe outline can be displayed quickly to give a general indication of the surface structure. Realistic renderings are produced by interpolating shading patterns across the polygon surfaces to eliminate or reduce the presence of polygon edge boundaries. And the polygon-mesh approximation to a curved surface can be improved by dividing the surface into smaller polygon facets.

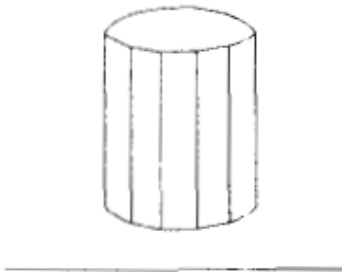


Figure 3.11

3.7.1 Polygon Tables:

We specify a polygon surface with a set of vertex coordinates and associated attribute parameters. As information for each polygon is input, the data are placed into tables that are to be used in the subsequent processing, display, and manipulation of the objects in a scene.

Polygon data tables can be organized into two groups:

- geometric tables
- attribute tables

Geometric data tables contain vertex coordinates and parameters to identify the spatial orientation of the polygon surfaces. Attribute information for an object includes parameters specifying the degree of transparency of the object and its surface reflectivity and texture characteristics. A

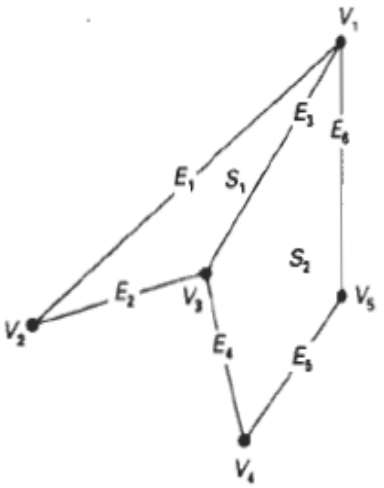
convenient organization for storing geometric data is to create three lists:

- a vertex table
- an edge table
- polygon table

Coordinate values for each vertex in the object are stored in the vertex table. The edge table contains pointers back into the vertex table to identify the vertices for each polygon edge. And the polygon table contains pointers back into the edge table to identify the edges for each polygon. This scheme is illustrated in Fig. 3-12 for two adjacent polygons on an object surface. In addition, individual objects and their component polygon faces can be assigned object and facet identifiers for ease) reference.

Listing the geometric data in three tables, as in Fig. 3-12, provides a convenient reference to the individual components (vertices, edges, and polygons) of each object. Also, the object can be displayed efficiently by using data from the edge table to draw the component lines. An alternative arrangement is to use just two tables: a vertex table and a polygon label. But this scheme is less convenient, and some edges could get drawn twice.

Another possibility is to use only a polygon table, but this duplicate coordinate information, since explicit coordinate values are listed for each vertex in each polygon. Also edge Information would have to be reconstructed from the vertex listings in the polygon table. We can add extra information to the data tables of Fig. 3-12 for faster information extraction.



VERTEX TABLE	
V_1:	x_1, y_1, z_1
V_2:	x_2, y_2, z_2
V_3:	x_3, y_3, z_3
V_4:	x_4, y_4, z_4
V_5:	x_5, y_5, z_5

EDGE TABLE	
E_1:	V_1, V_2
E_2:	V_2, V_3
E_3:	V_3, V_1
E_4:	V_3, V_4
E_5:	V_4, V_5
E_6:	V_5, V_1

POLYGON-SURFACE TABLE	
S_1:	E_1, E_2, E_3
S_2:	E_3, E_4, E_5, E_6

Figure: 3-12

3.7.2 Plane Equations:

To produce a display of a three-dimensional object, we must process the input data representation for the object through several procedures. These processing steps include transformation of the modeling and world-coordinate descriptions to viewing coordinates, then to device coordinates; identification of visible surfaces; and the application of surface-rendering procedures.

Text Book:

Donald Hearn & Pauline M.Baker – Computer Graphics 2nd Edition –PHI.

Reference Book:

1.Foley, Van Dam, Feiner and Hughes – Computer Graphics Principles and Practice 3rd Edition.

2.N.Krishnamurthy - Introduction to Computer Graphics.