Clipping and 3D object representation Concepts

Unit 2

The Viewing Pipeline

- A world-coordinate area selected for display is called a ***window***.
- An area on a display device to which a window is mapped is called a ***viewport***.
- The window defines ***what*** is to be viewed; the viewport defines ***where*** it is to be displayed.
- Often, windows and viewports are rectangles in standard position, with the rectangle edges parallel to the coordinate axes.

- In general, the mapping of a part of a world-coordinate scene to device coordinates is referred to as a ***viewing transformation.***
- Sometimes the 2D viewing transformation is simply referred to as the ***window-to-viewport transformation*** or the ***windowing transformation***.
- Following figure illustrates the mapping of a picture section that falls within a rectangular window onto a designated rectangular viewport.

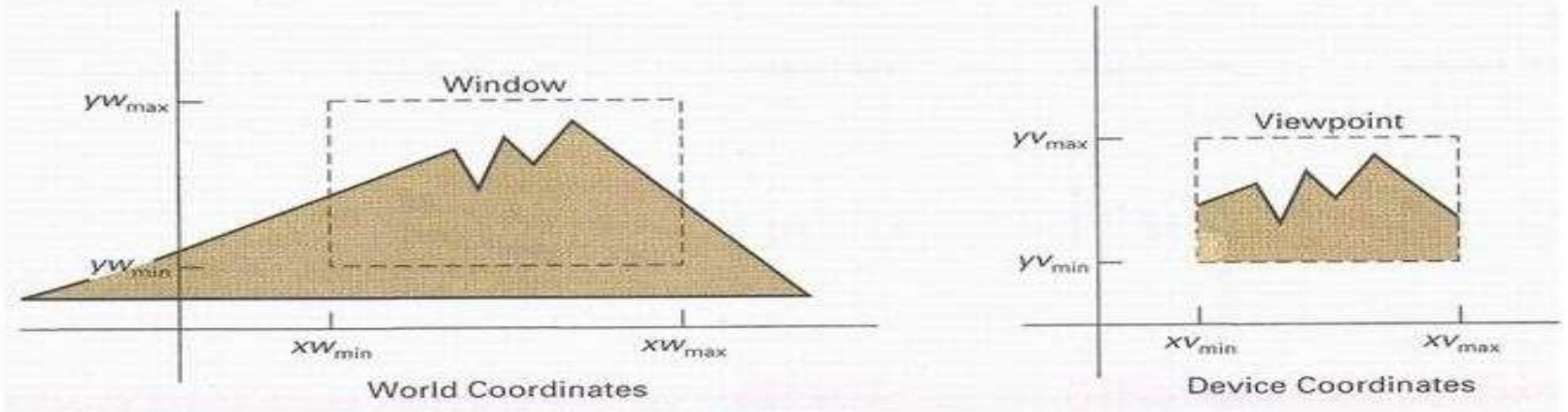


Figure 6-1
A viewing transformation using standard rectangles for the window and viewport.

- In computer graphics terminology, the term window originally referred to an area of a picture that is selected for viewing.
- Some graphics packages that provide window and viewport operations allow only standard rectangles, but a more general approach is to allow the rectangular window to have any orientation.
- Viewing transformation is carried out in several steps, which is called viewing-transformation pipeline.
- First, we construct a scene in world coordinates using the output primitives and attributes.
- To obtain a particular orientation for the window, we can setup a two-dimensional viewing-coordinate system in the world-coordinate plane, and define a window in the viewing-coordinate system.
- Once the viewing reference frame is established, we can transform descriptions in world coordinates to viewing coordinates.

- We then define a viewport in normalized coordinates in the range from 0 to 1 and map the viewing-coordinate description of the scene to normalized coordinates.
- At the final step, all parts of the picture that lie outside the viewport are clipped, and the contents of the viewport are transferred to device coordinates.

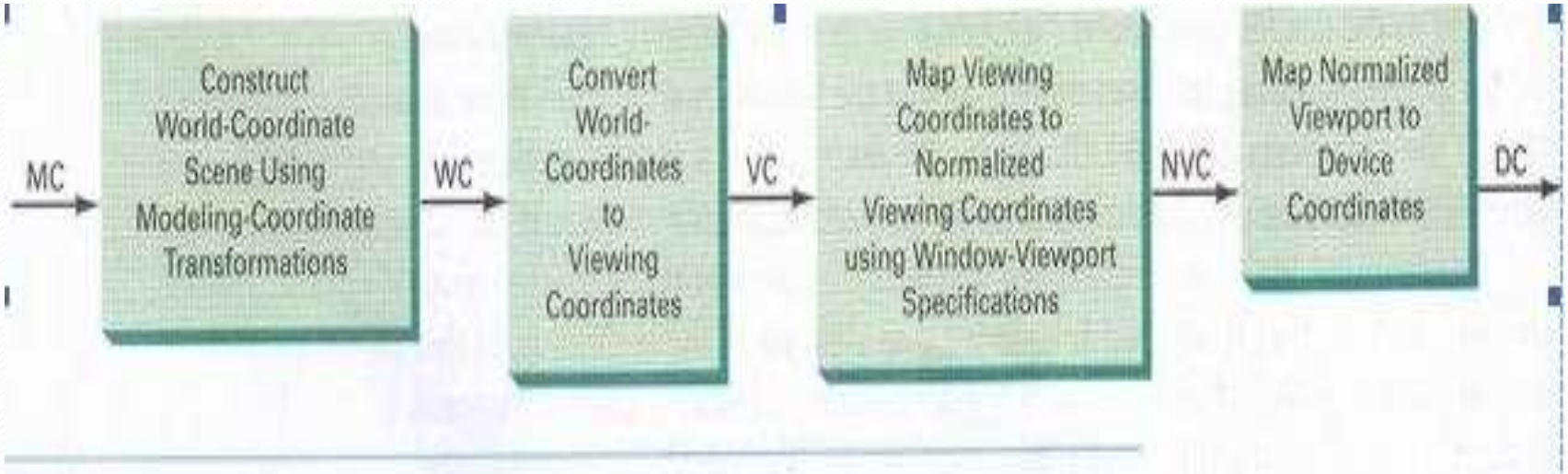


Figure 6-2
The two-dimensional viewing-transformation pipeline.

- By changing the position of the viewport, we can view objects at different positions on the display area of an output device.
- Also, by varying the size of viewports, we can change the size and proportions of displayed objects.
- 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.
- Once the scene has been transferred to normalized coordinates, the unit square is simply mapped to the display area for the particular output device in use at that time.

- When all coordinate transformations completed, viewport clipping can be performed in normalized coordinates or in device coordinates.
- 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.

Window-To-Viewport Coordinate Transformation

- Once object descriptions have been transferred to the viewing reference frame, we choose the window extents in viewing coordinates and select the viewport limits in normalized coordinates as shown in following figure.
- Object descriptions are then transferred to normalized device coordinates.
- We do this using a transformation that maintains the same relative placement of objects in normalized space as they had in viewing coordinates.

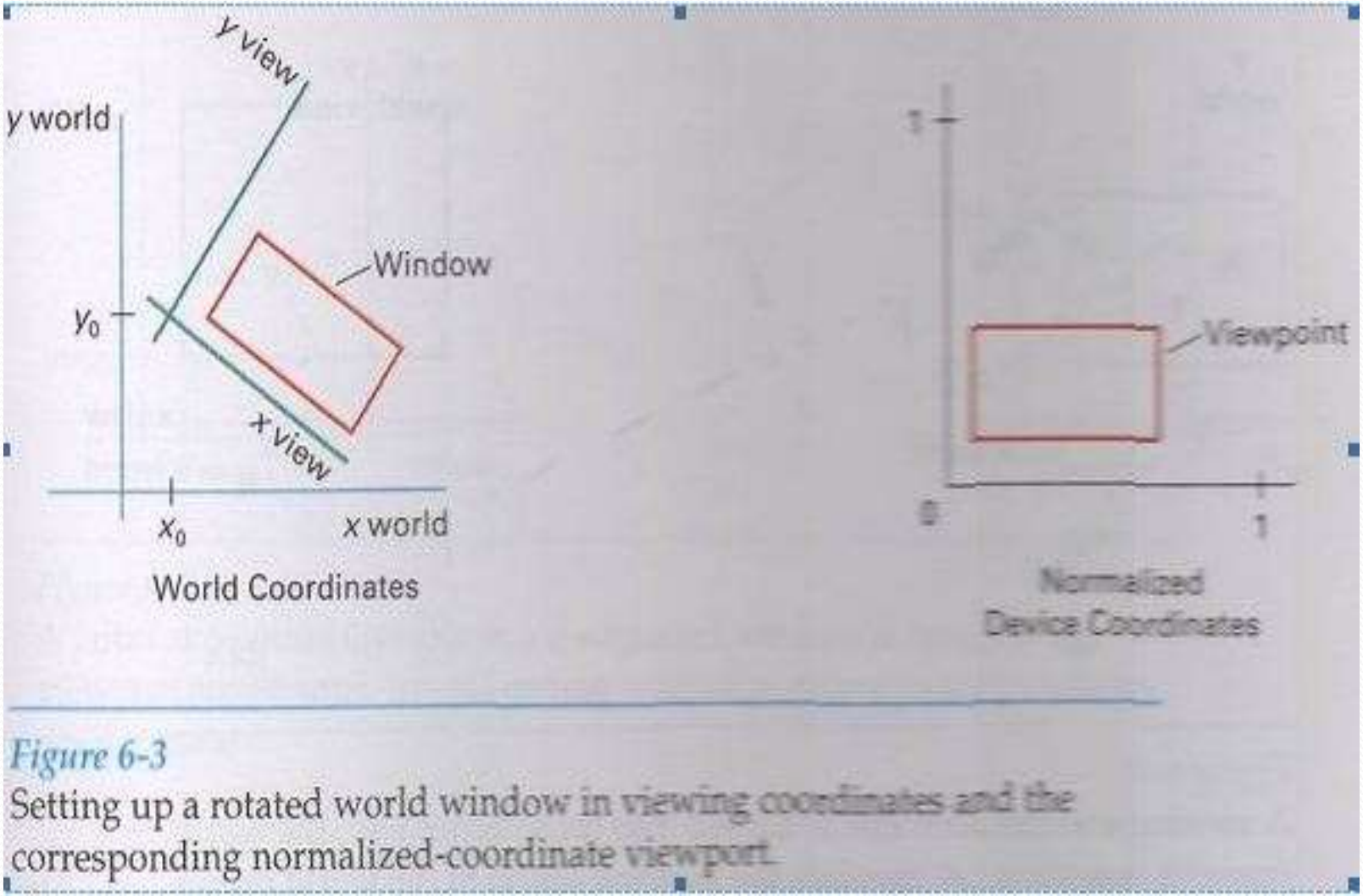
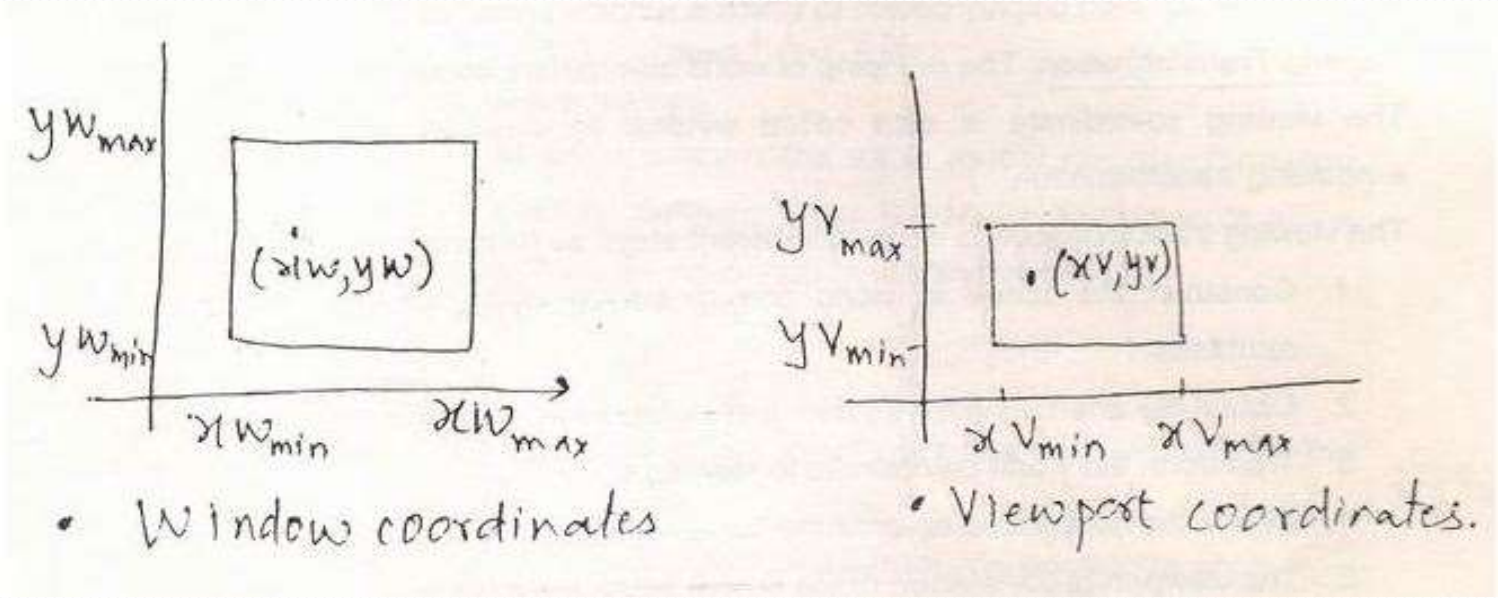


Figure 6-3
Setting up a rotated world window in viewing coordinates and the corresponding normalized-coordinate viewport.

- If a coordinate position is at the center of the viewing window, it will be displayed at the center of the viewport.
- Following figure illustrates the window-to-viewport mapping.



- To maintain the same relative placement in the viewport as in the window, we require that,

$$\frac{x_v - xv_{min}}{xv_{max} - xv_{min}} = \frac{x_w - xw_{min}}{xw_{max} - xw_{min}}$$

$$\frac{y_v - yv_{min}}{yv_{max} - yv_{min}} = \frac{y_w - yw_{min}}{yw_{max} - yw_{min}}$$

By solving these equations the coordinates of viewport (xv,yv) has the values

$$xv = xv_{min} + (xw - xw_{min})\, sx$$

$$yv = yv_{min} + (yw - yw_{min})\, sy$$

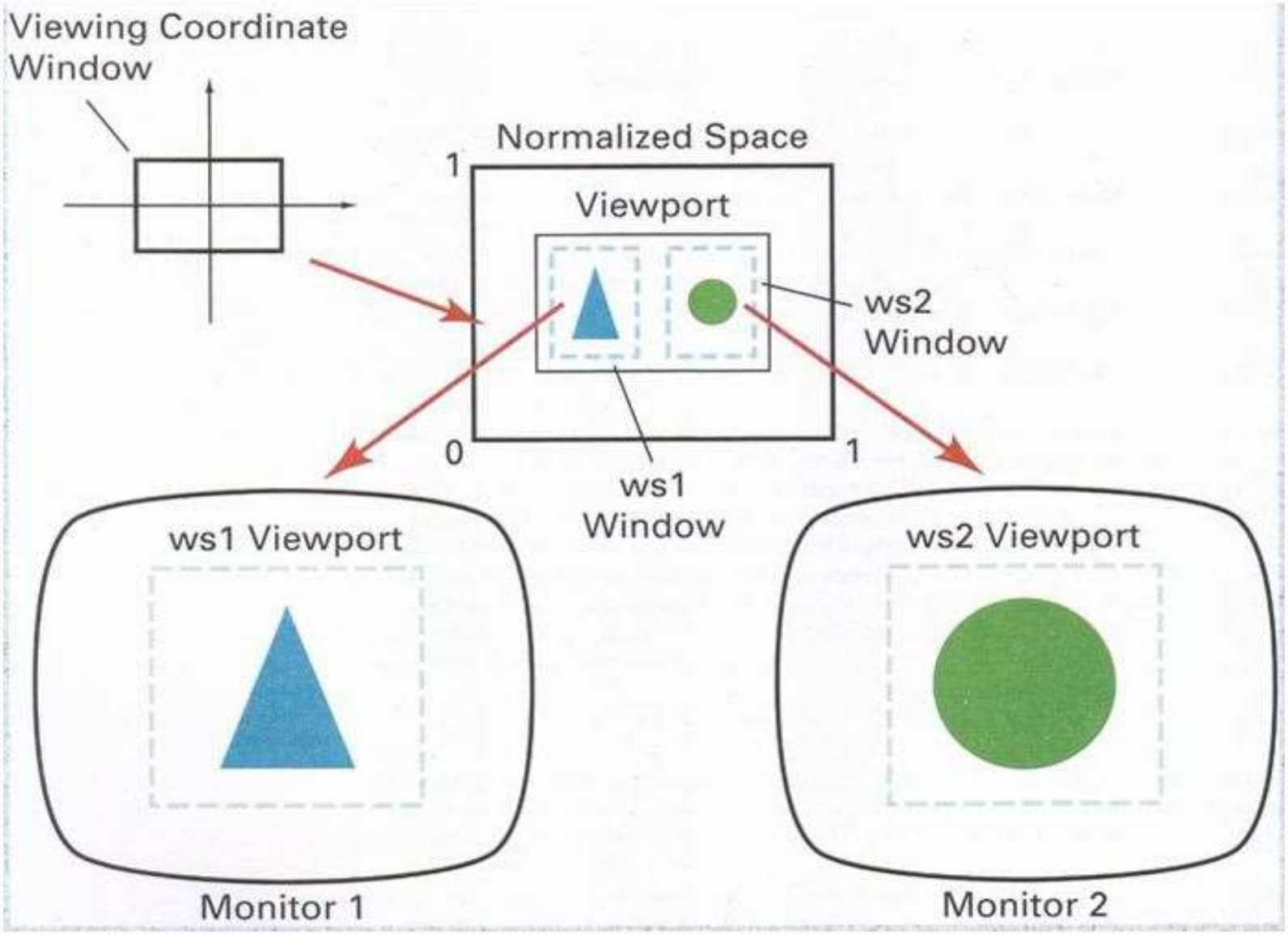
where, the values of ***sx and sy*** are

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

This transformation is performed in following two steps:

1. Perform scaling transformation of (xw_{min} , yw_{min}) to scale from window to viewport
2. Translate the window area into the position of viewport

- Relative proportions of objects are maintained if the scaling factors are the same (sx = sy).
- Otherwise, world objects will be stretched or contracted in either the x or y direction when displayed on the output device.
- From normalized coordinates, object descriptions are mapped to the various display devices.
- Any number of output devices can be open in a particular application, and another window-to-viewport transformation can be performed for each open output device. This mapping is called the workstation transformation.
- Workstation transformation is accomplished by selecting a window area in normalized space and a viewport area in the coordinates of the display device.


Viewing Coordinate Window
Normalized Space
1
Viewport
ws2 Window
0
1
ws1 Window
ws1 Viewport
ws2 Viewport
Monitor 1
Monitor 2

Clipping Operations

- Any procedure that identifies those portions of a picture that are either inside or outside of a specified region of space is referred to as a clipping algorithm, or simply clipping.
- The region against which an object is to clipped is called a clip window.
- The clipping operation can be performed on different objects. Thus it can be divided as :

1. Point Clipping
2. Line Clipping
3. Area (Polygon) Clipping
4. Text Clipping
5. Curve Clipping

Point Clipping

- Assuming that the clip window is a rectangle in standard position, we save a point P=(x,y) for display if the following inequalities are satisfied:

  XWmin <= X <= XWmax

  YWmin <= Y <= Ywmax

- where the edges of clip window are (**XWmin , XWmax , YWmin, Ywmax**)
- If any of the above equation is not satisfied then the point is clipped (not saved)

Line Clipping

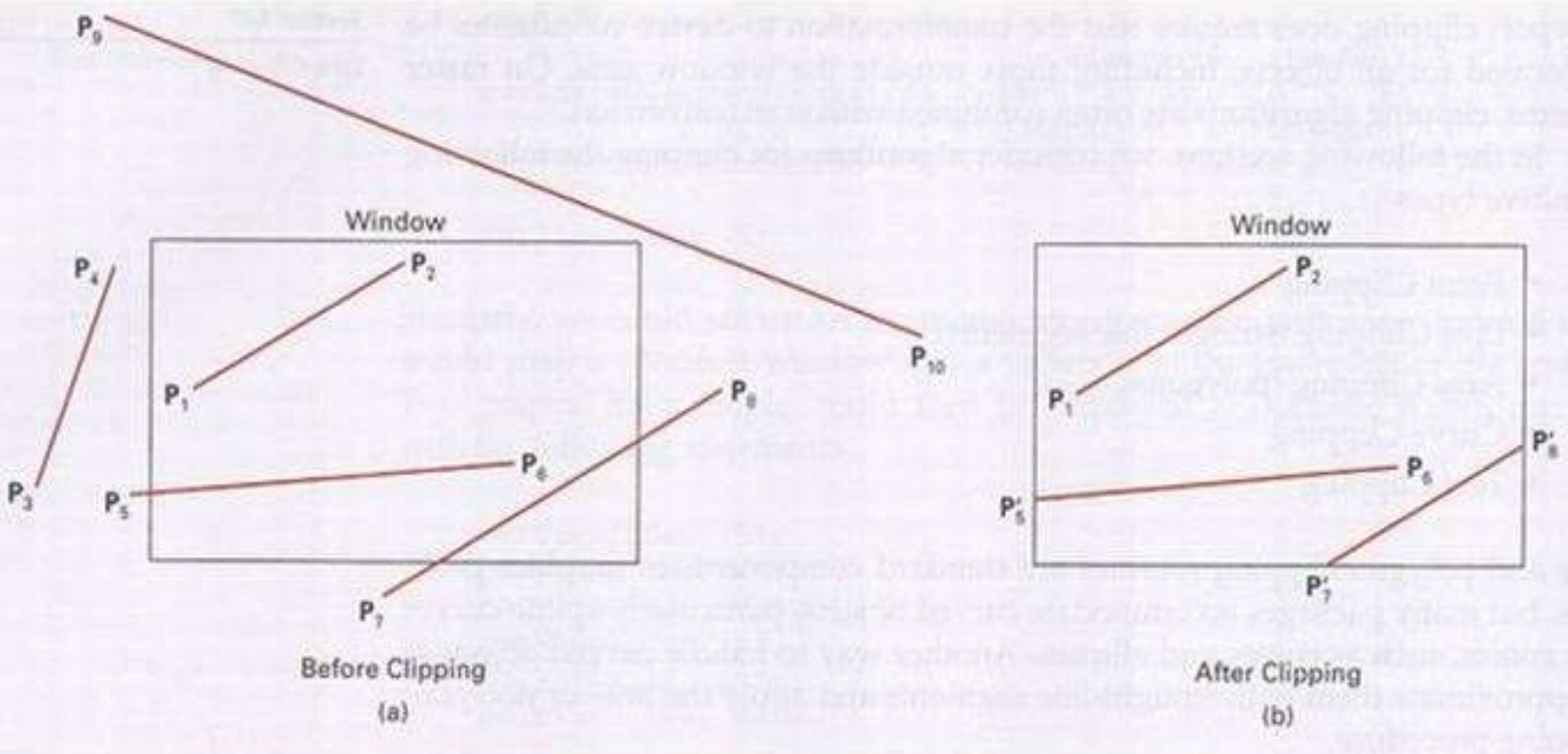


Figure 6-7
Line clipping against a rectangular clip window.

- Figure illustrates possible relationships between line positions and a standard rectangular clipping region.
- A line-clipping procedure involves several parts.
- First, we can test a given line segment to determine whether it lies completely inside the clipping window.
- If it does not, we try to determine whether it lies completely outside the window.
- Finally, if we cannot identify a line as completely inside or completely outside, we must perform intersection calculations with one or more clipping boundaries.
- A line with both endpoints inside all clipping boundaries, such as the line from P1 to P2 is saved.

- For the line with end points (**X1, Y1**) & (**X2 , Y2**) and one or both endpoints outside the clipping rectangle, the following parametric representation can be used;

$$X = X_1 + u\,(X_2 - X_1)$$

$$Y = Y_1 + u\,(Y_2 - Y_1)$$

$$0 <= u <= 1$$

- 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 of 0 to 1 then, it crosses the clipping area.

Cohen-Sutherland Line Clipping

- This is one of the oldest and most popular line-clipping procedures.
- Every line end-point in a picture is assigned a four-digit binary code, called a region code, that identifies the location of the point relative to the boundaries of the clipping rectangle.
- Regions are setup in reference to the boundaries as shown in following figure.
- 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.

Line Clipping

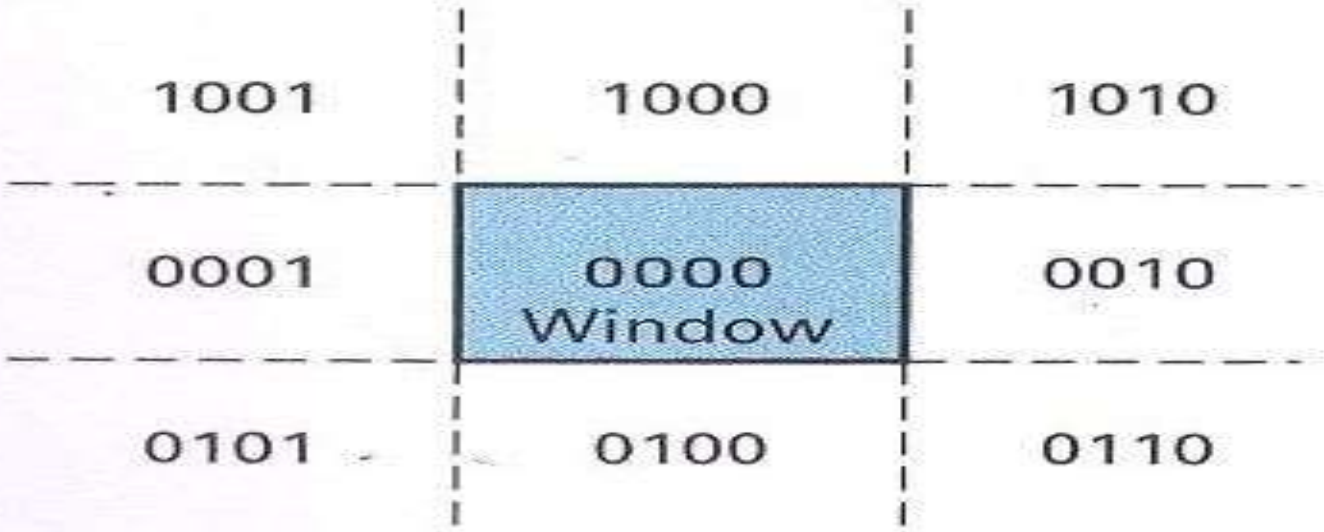


Figure 6-8
Binary region codes assigned to line endpoints according to relative position with respect to the clipping rectangle.

bit 1 represents : Left

bit 2 represents : Right

bit 3 represents : Below

bit 4 represents : Above

- 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.
- 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 $x < x_{wmin}$.
- The other three bit values can be determined using similar comparisons.

- By using such representation we can make following conclusions:

1. If the region code of both endpoints is 0000, then the line is completely inside.

2. If both the endpoints contain 1 in same bit position, then the line is completely outside. This can also be checked by logical AND operation. If the result is not 0000 then, line is completely outside.

3. If the line is not completely inside or outside, then we have to find out the intersection point of line with window.

- The intersection points can be calculated with the help of slope m of a line which has endpoints **(X1, Y1)** & **(X2 , Y2)**.

$$Y = Y_1 + m (X - X_1) \text{ and } X = X_1 + (Y - Y_1) / m$$

The equation of slope is given as, $m = (Y_2 - Y_1) / (X_2 – X_1)$

The values of X will be **XWmin** or **XWmax** and that for Y will by **YWmin** or **WYmax.**

Polygon Clipping

- To clip polygons, we need to modify the line-clipping procedures discussed in the previous section.
- A polygon boundary processed with a line clipper may be displayed as a series of unconnected line segments, depending on the orientation of the polygon to the clipping window.
- What we really want to display is bounded area after clipping, as following figure.

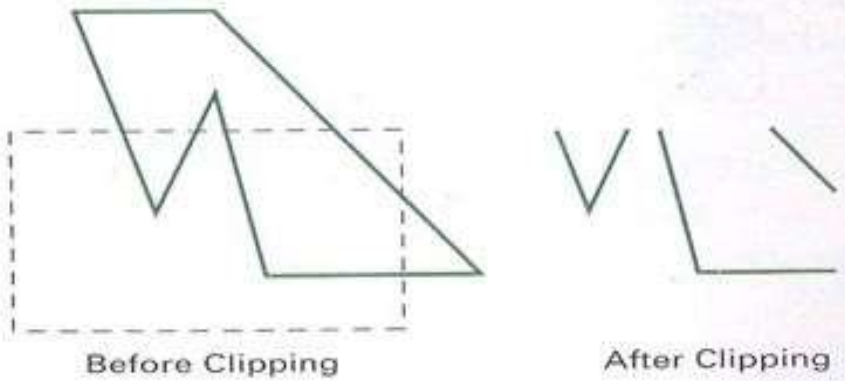


Figure 6-17
Display of a polygon processed by a line-clipping algorithm.

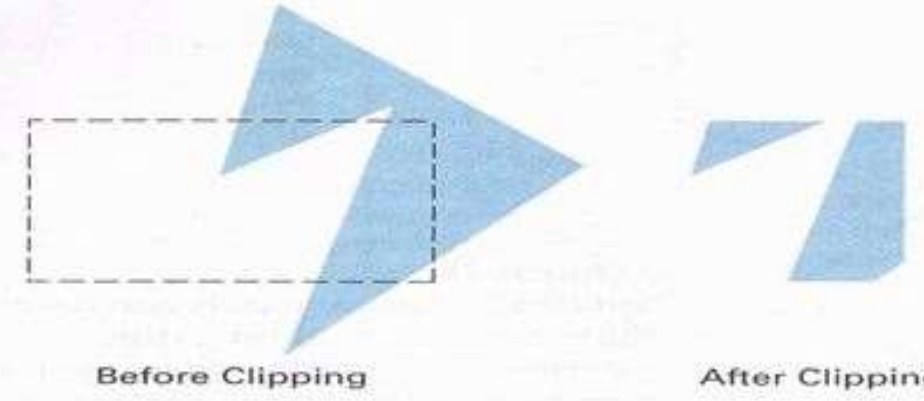


Figure 6-18
Display of a correctly clipped polygon.

Sutherland-Hodgeman Polygon Clipping

- We can correctly clip a polygon by processing the polygon boundary as a whole against each window edge.
- This could be accomplished by processing all polygon vertices against each clip rectangle boundary in turn.
- Beginning with the initial set of polygon vertices, we could first clip the polygon against the left rectangle boundary to produce a new sequence of vertices.
- The new set of vertices could then be successively passed to a right boundary clipper, a bottom boundary clipper, and a top boundary clipper, as shown in following figure.

Figure 6-19
Clipping a polygon against successive window boundaries.

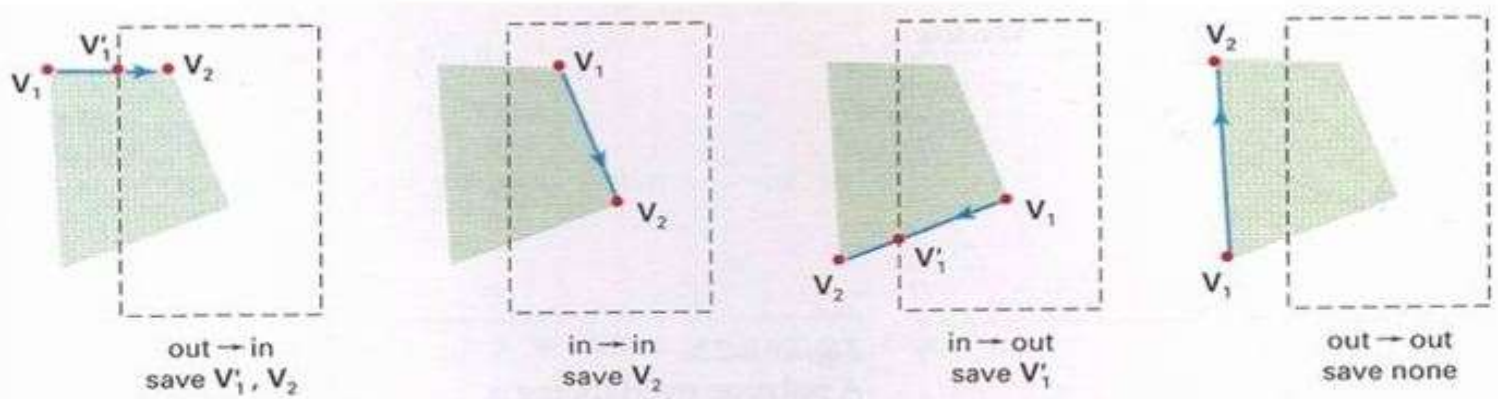
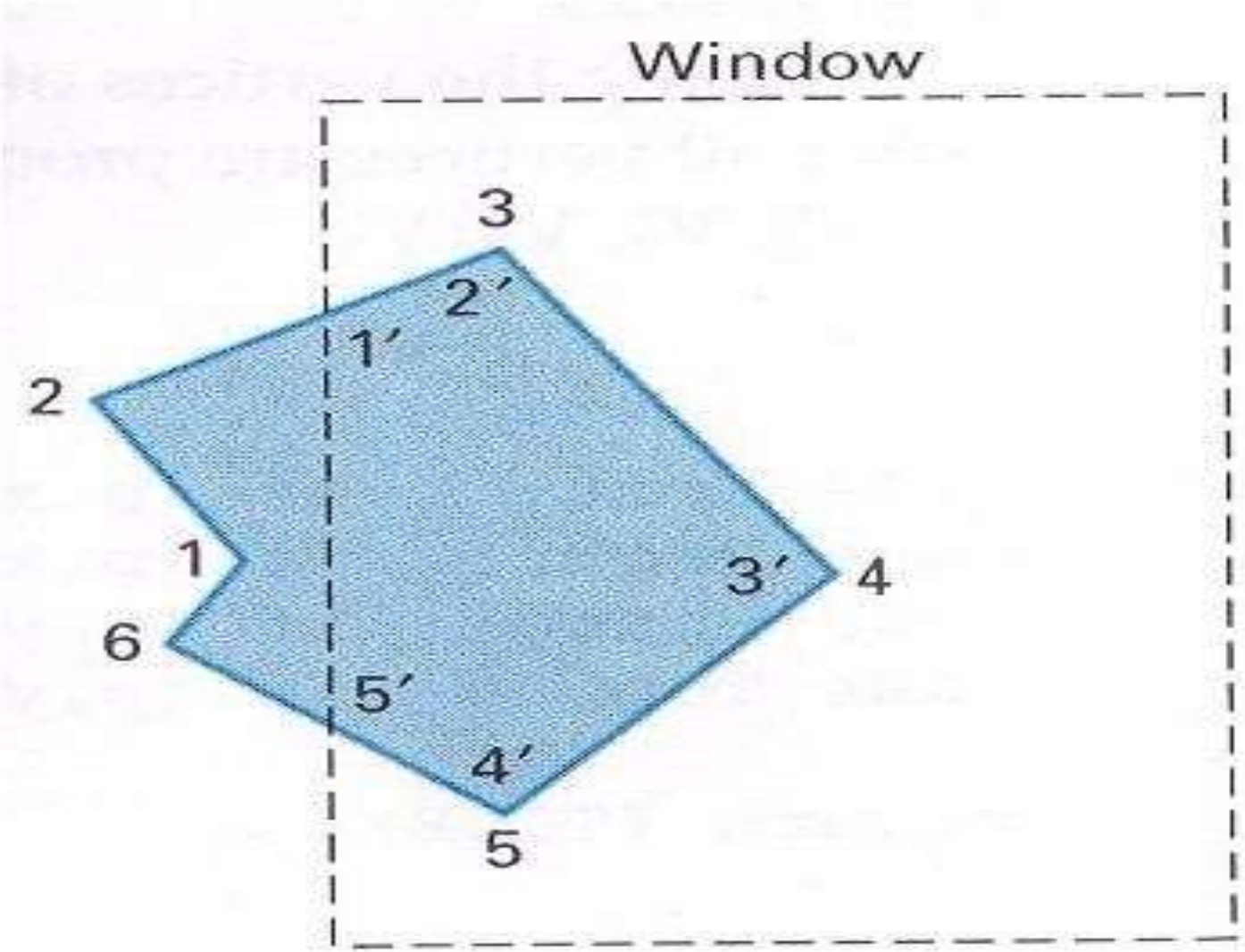


Figure 6-20
Successive processing of pairs of polygon vertices against the left window boundary.

- 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.

- We illustrate this method by processing the area in following figure against the left window boundary.
- Vertices 1 and 2 are found to be on the outside of the boundary.
- Moving along to vertex 3, which is inside, we calculate the intersection and save both the intersection point and vertex 3.
- Vertices 4 and 5 are determined to be inside, and they also are saved.
- The sixth and final vertex is outside, so we find and save the intersection point.
- Using the five saved point, we would repeat the process for the next window boundary.


Window
3
2′
1′
2
1
3′
4
6
5′
4′
5

Weiler-Atherton Polygon Clipping

- Here, the vertex-processing procedures for window boundaries are modified so that concave polygons are displayed correctly.
- The basic idea in this algorithm is that instead of always proceeding around the polygon edges as vertices are processed, we sometimes want to follow the window boundaries.
- Which path we follow depends on the polygon-processing direction (clockwise or counterclockwise) and whether the pair of polygon vertices currently being processed represents an outside-to-inside pair or an inside-to-outside pair.

- The basic idea for clock-wise processing of vertices is given as two rules:

1. For outside to inside pair of vertices, follow the polygon boundary.
2. For inside to outside pair of vertices, follow the window boundary.

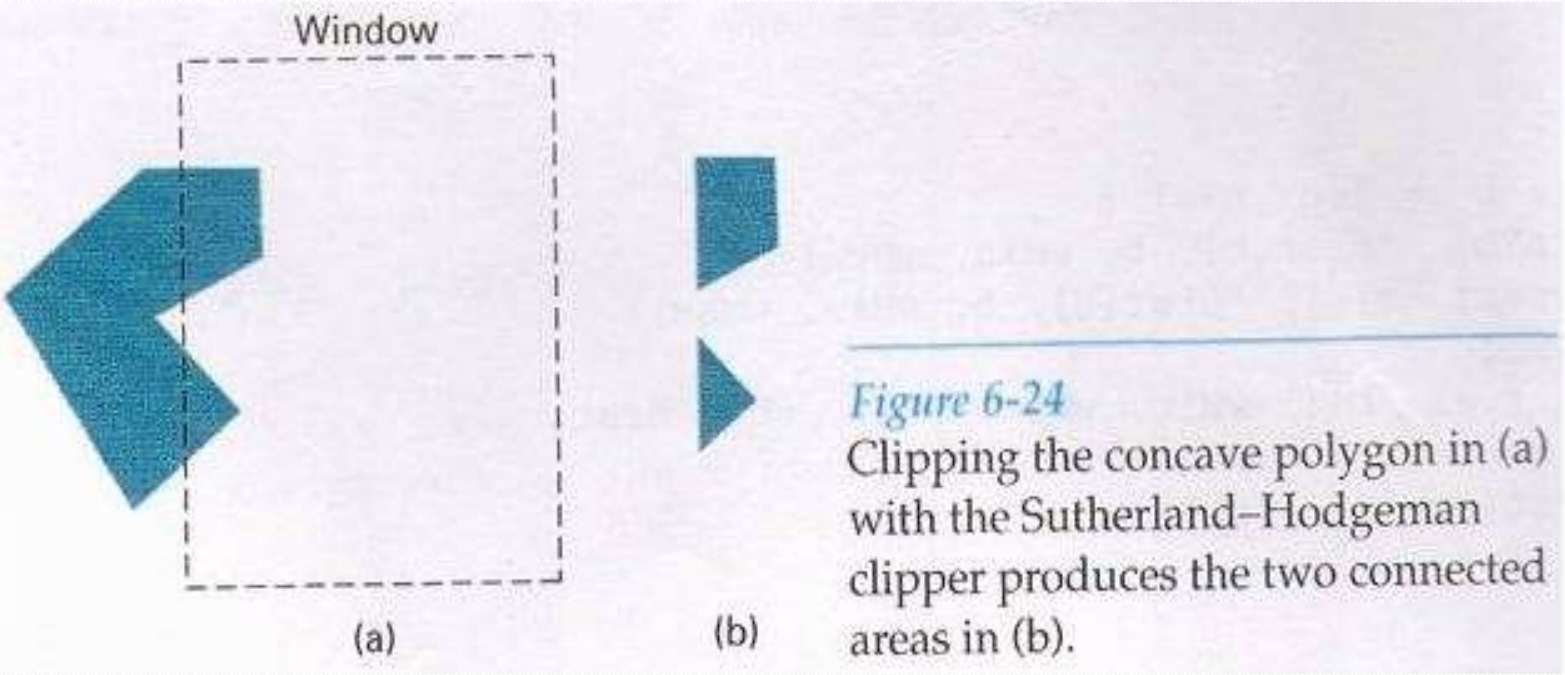


Figure 6-24
Clipping the concave polygon in (a) with the Sutherland–Hodgeman clipper produces the two connected areas in (b).

Curve Clipping

- Curve clipping procedures will involve nonlinear equations, however, and this requires more processing than for objects with linear boundaries.
- The bounding rectangle for a circle or other curved object can be used first to test for overlap with a rectangular clip window.
- If the bounding rectangle for the object is completely inside the window, we save the object.
- If the rectangle is determined to be completely outside the window, we discard the object.

- Else, for a circle, we can use the coordinate extents of individual quadrants and then octants for preliminary testing before calculating curve-window intersections.
- For an ellipse, we can test the coordinate extents of individual quadrants.

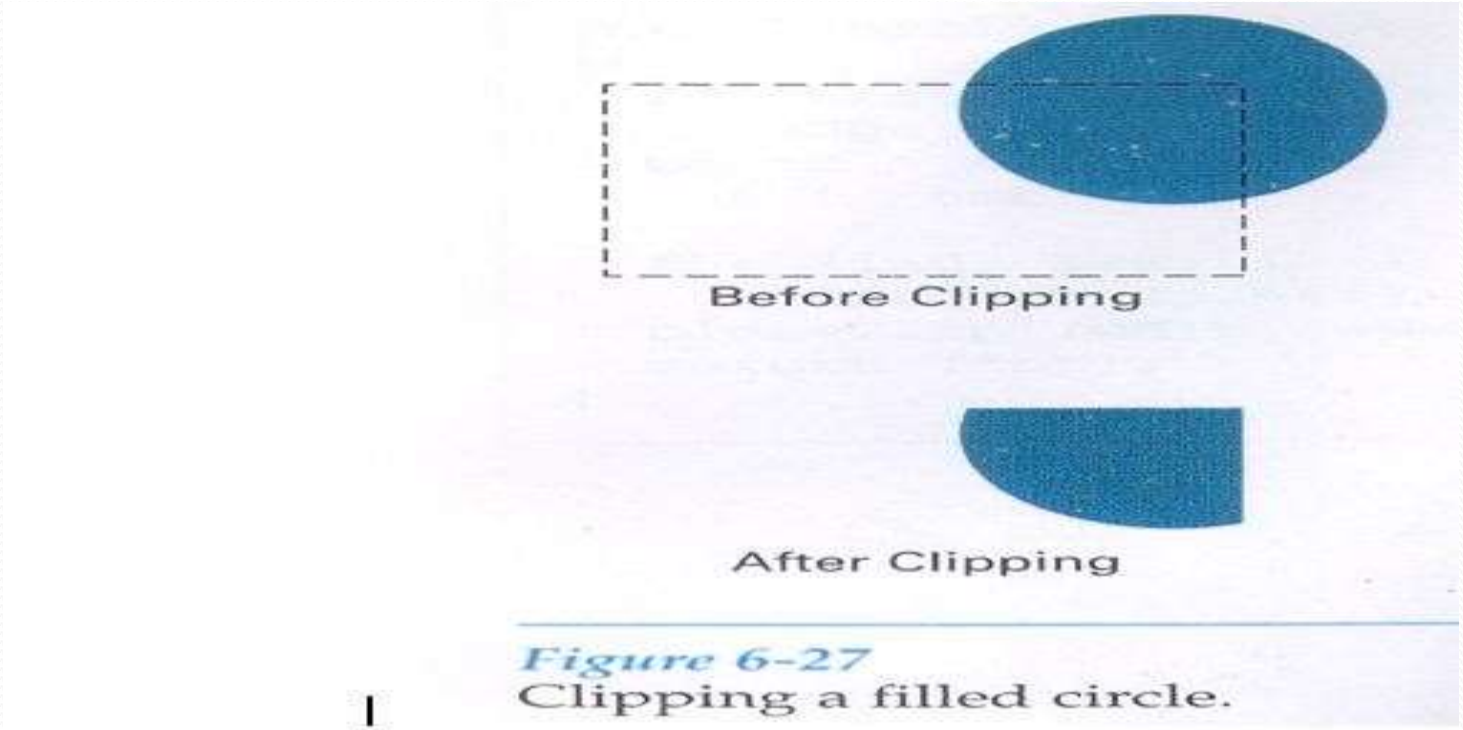
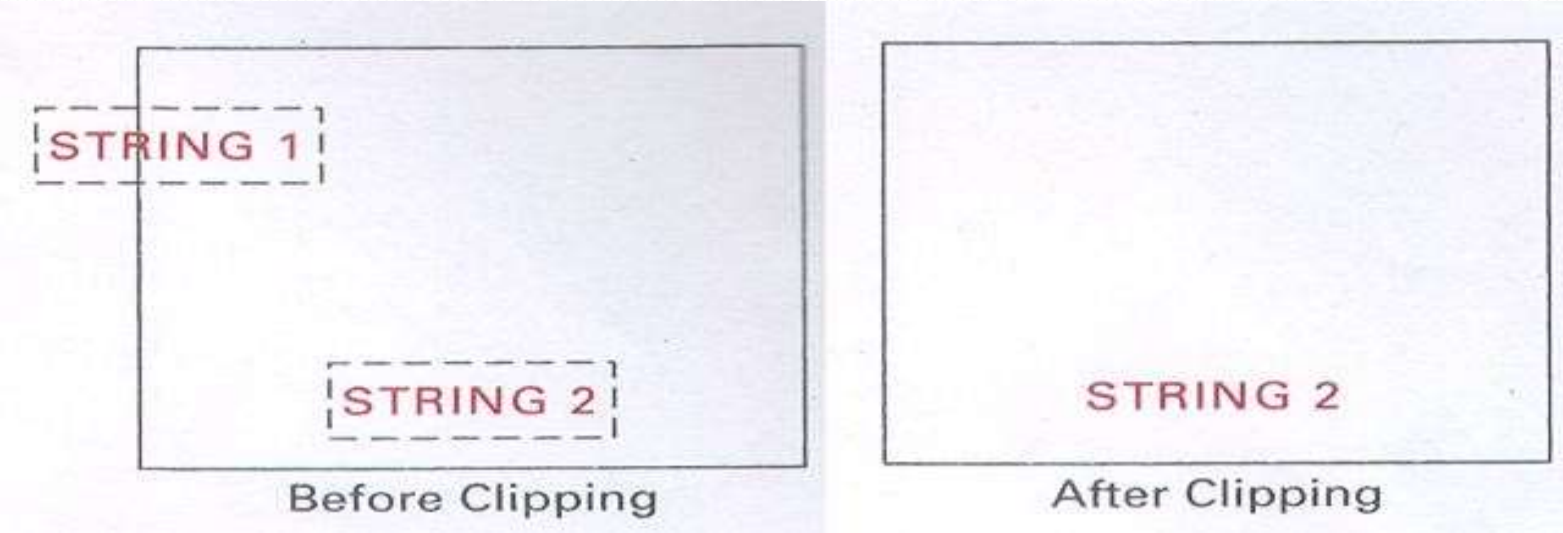


Figure 6-27
Clipping a filled circle.

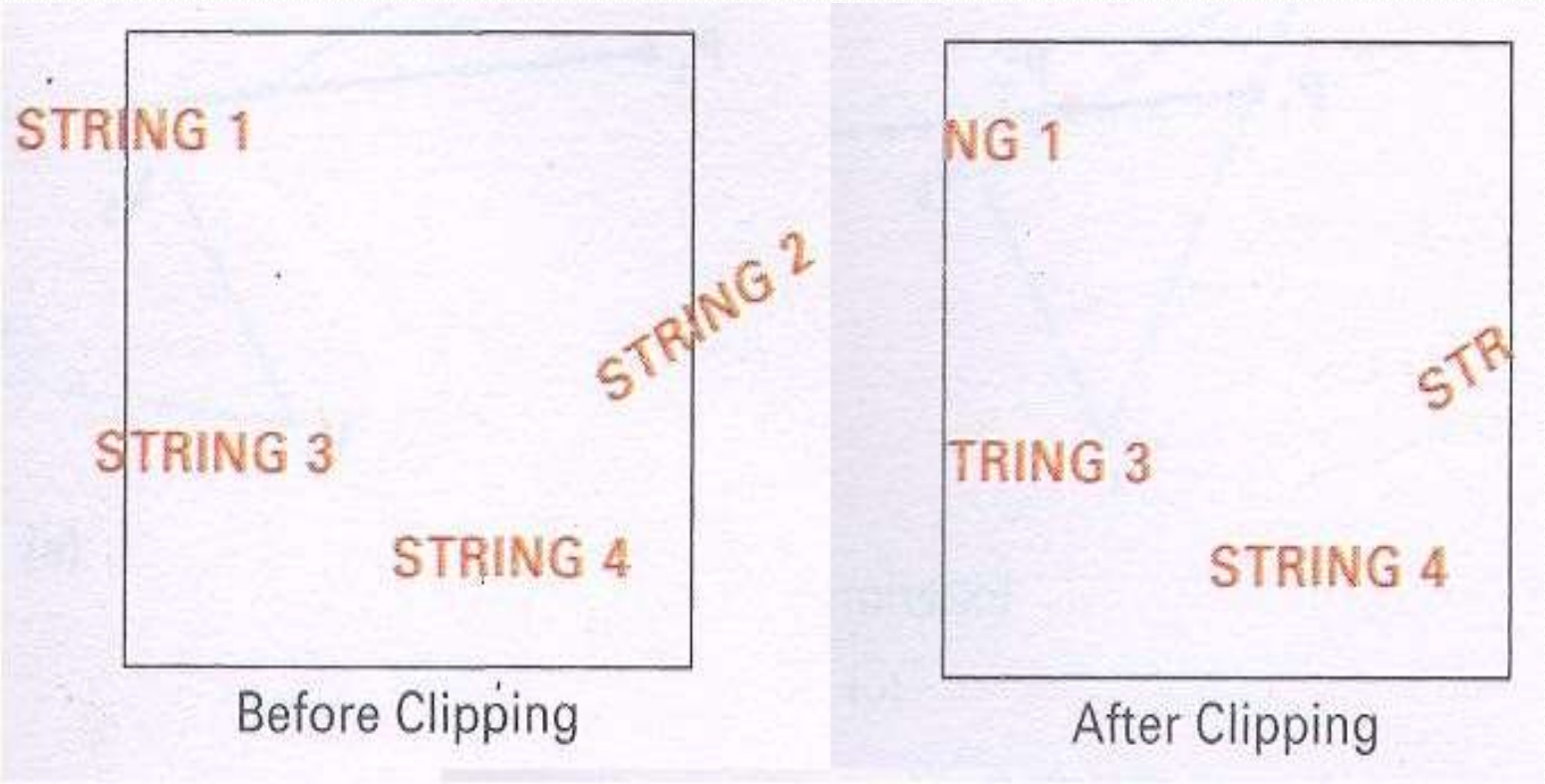
Text Clipping

- There are many methods for text clipping which can be used according to application.

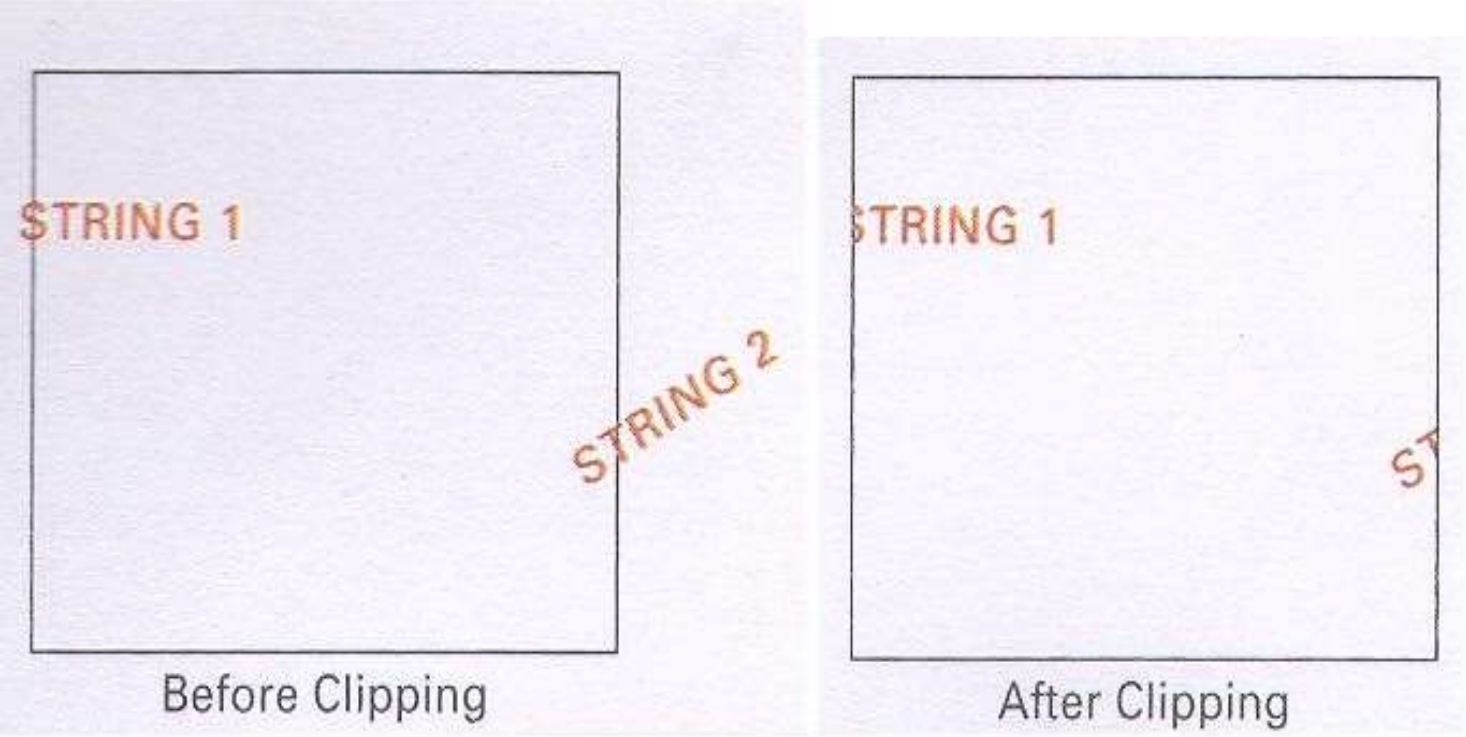
1. **All or none string clipping:** It is simplest method in which if any portion of text is outside clip window, it will be discarded. This procedure is implemented by considering a bounding rectangle around the text pattern.



2. All or none character clipping: If any portion of character is outside clip window, it will be discarded.



3. Component clipping: The parts of characters which are outside the clip window will be discarded. If an individual character overlaps a clip window boundary, we clip off the parts of the character that are outside the window.



Exterior Clipping

- If we want to clip a picture to the exterior of a specified region, exterior clipping is used.
- The picture parts to be saved are those that are outside the region. This is referred to as exterior clipping.
- A typical example of the application of exterior clipping is in multiple window systems. To correctly display the screen windows, we often need to apply both internal and external clipping.

Three Dimensional Concepts

- When we model and display a 3D scene, there are many more considerations we must take into account besides just including coordinate values for the third dimension.
- Object boundaries can be constructed with various combinations of plane and curved surfaces.
- There are some graphics packages that are use for 3D object representation.
- Viewing transformations in 3D are much more complicated because there may be so many parameters to select when specifying how 3D scene is to be mapped to display device.

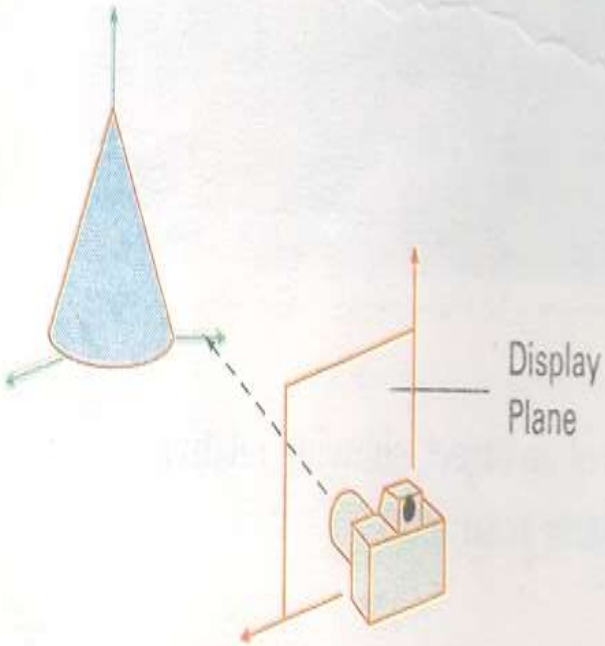


Figure 9-1
Coordinate reference for obtaining a particular view of a three-dimensional scene.

- To obtain a display of a 3D scene that has been modeled in world coordinates, we must first set up a coordinate reference for camera.
- This coordinate reference defines the position and orientation for the plane of the camera film, which is the plane we want to use to display a view of the object in the scene.
- Object descriptions are then transferred to the camera reference coordinates and projected onto the selected display plane.
- We can then display the objects in wireframe form.

Parallel Projection

- One method of generating a view of solid objects is to project points on the object surface **along parallel lines** on to 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 project into parallel lines on the two-dimensional display plane.
- This technique is used in engineering and architectural drawings to display an object with a set of views that maintain relative proportions of the object.

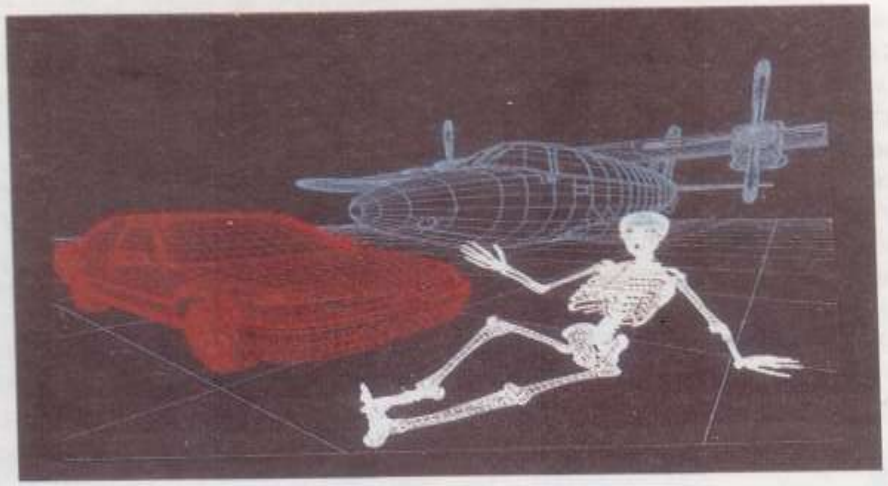


Figure 9-2
Wireframe display of three objects, with back lines removed, from a commercial database of object shapes. Each object in the database is defined as a grid of coordinate points, which can then be viewed in wireframe form or in a surface-rendered form. (*Courtesy of Viewpoint DataLabs.*)

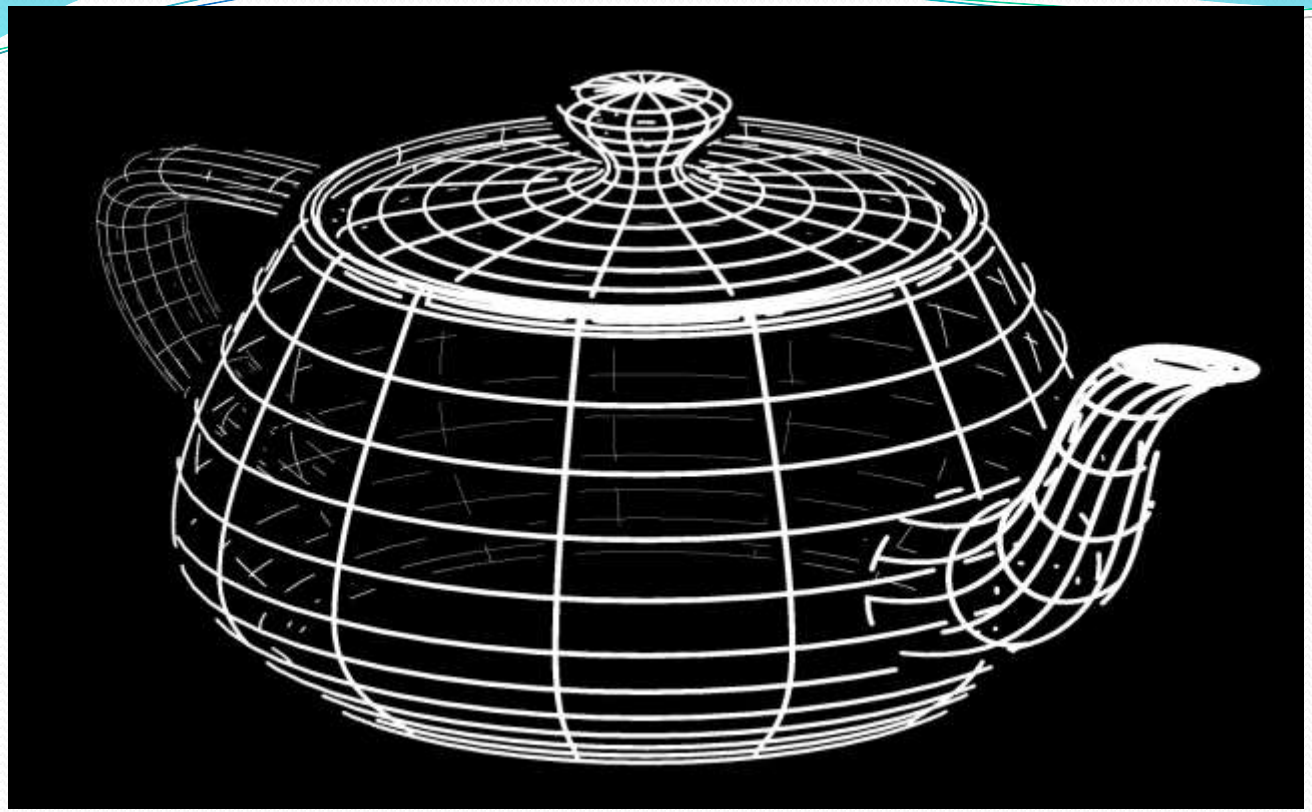
Figure 9-3
Three parallel-projection views of an object, showing relative proportions from different viewing positions.

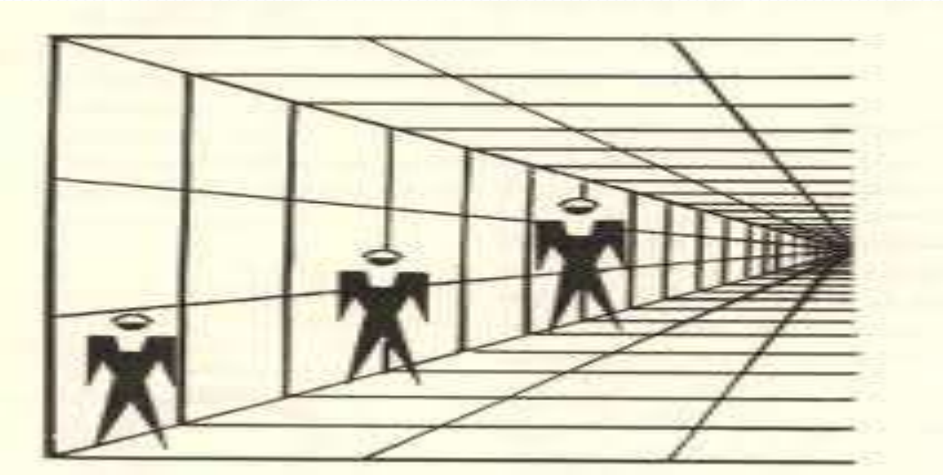
Perspective Projection

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

Depth Cueing

- With a few exceptions, depth information is more important, so that we can easily identify for a particular viewing direction, which is the front and which is the back of the displayed objects.
- There are several ways to include depth info. in two dimensional representation of the solid objects.
- A simple method for indicating depth with wireframe displays is to vary intensity of objects according to their distance from the viewing position. The lines closest to the viewing position are displayed with the highest intensities and lines farther away are displayed with the decreasing intensities.
- Depth cueing is applied by choosing maximum and minimum intensity (color values) and a range of distances over which the intensities are to vary.





Visible Line and Surface Identification

- 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.
- 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.

Surface Rendering

- Added realism is attained in displays by setting the surface intensity of objects according to the lighting conditions in the scene and according to assigned surface characteristics.
- Lighting specifications include the intensity and positions of light sources and the general background illumination required for a scene.
- Surface properties of objects include degree of transparency and how rough or smooth the surfaces are to be.

Three Dimensional Object Representations

- Representation schemes for solid objects are often divided into two broad categories, although not all representations fall neatly into any one of two categories:
  1. Boundary representations (B-reps).
  2. Space-partitioning representations

Boundary representations (B-reps) :

- This describes a three-dimensional objects as a set of surfaces that separate the object interior from the environment.

Space-partitioning representations

- They are used to describe interior properties, by partitioning the spatial region containing an object into a set of small, non-overlapping, contiguous solids (usually cubes).

Polygon Surfaces

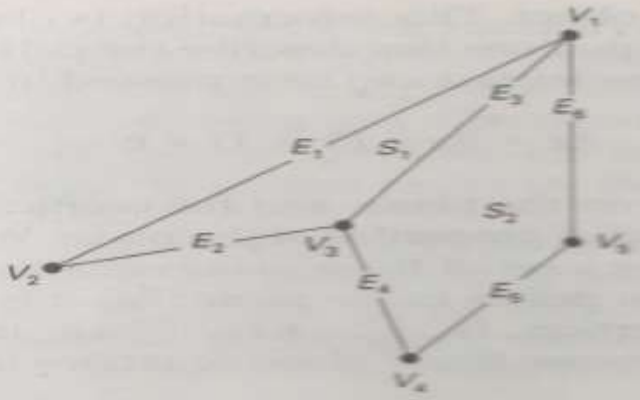
- It is example of boundary representation method.
- It simplify and speed up the surface rendering and display of objects, since all the surface are described with linear equations.

➢ **Polygon Tables**

- we specify the polygon surface with a set of vertex coordinate and associated attribute parameters.
- Polygon data table can be organized in two way:

1. Geometric table
2. Attribute table

- Geometric data tables contains vertex coordinate and parameter to identify the spatial orientation of the polygon surface.
- Attribute data table includes texture characteristics and its surface reflectivity.
- A convenient organization for storing geometric data is to create three list:
  1. A vertex table
  2. An edge table
  3. A polygon table
- Coordinate value for each vertex of an object is stored in vertex table.
- Edge table contains pointers back into vertex table to identify the vertices of each polygon edge.
- Polygon table contains pointer back into edge table to identify the edges for each polygon.



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

E_1:	V_1, V_2, S_1
E_2:	V_2, V_3, S_1
E_3:	V_3, V_1, S_1, S_2
E_4:	V_3, V_4, S_2
E_5:	V_4, V_5, S_2
E_6:	V_5, V_1, S_2

Curved Lines and Surfaces

- Displays of three-dimensional curved lines and surfaces can be generated from an input set of mathematical functions defining the objects or from a set of user specified data points.
- When functions are specified, a package can project the defining equations for a curve to the display plane and plot pixel positions along the path of the projected function.

Quadric Surfaces

- Frequently used classes of objects are the **quadric surfaces**, which are described with second-degree equations (quadratics). They include spheres, ellipsoids, paraboloids, and hyperboloids.
- Quadric surfaces, particularly spheres and ellipsoids, are common elements of graphics scenes, and they are often available in graphics packages as primitives from which more complex objects can be constructed.

Blobby Objects

- Some objects do not maintain a fixed shape, but change their surface characteristics in certain motions or when in proximity to other objects.
- Examples in this class of objects include molecular structures, water droplets, muscle shapes in the human body etc.
- These objects can be described as exhibiting “blobbiness” and are often simply referred to as blobby objects, since their shapes show a certain degree of fluidity.

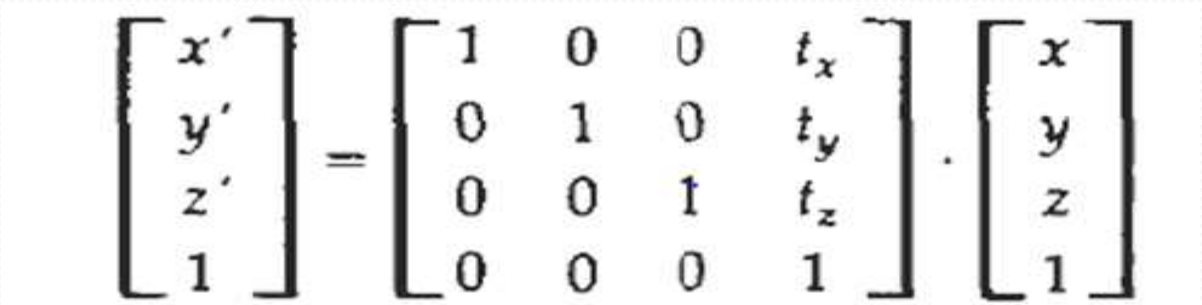
Spline Representations

- In drafting terminology, a Spline is a flexible strip used to produce a smooth curve through a designated set of points.
- Several small weights are distributed along the length of the strip to hold it in position on the drafting table as the curve is drawn.
- The term spline curve originally referred to a curve drawn in this manner.
- In computer graphics, the term spline curve now refers to any composite curve formed with polynomial sections satisfying specified continuity conditions at the boundary of the pieces.
- A spline surface can be described with two sets of orthogonal spline curves.

3-D Transformations

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



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

OR

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

Translation

- Parameters t_x t_y and t_z specifying translation distances for the coordinate directions ***x***, **y**, and **z**, are assigned any real values. The matrix representation in Eq. 11-1 is equivalent to the three equations

$$\mathbf{x}' = \mathbf{x} + t_x,$$

$$\mathbf{y}' = \mathbf{y} + t_y,$$

$$\mathbf{z}' = \mathbf{z} + t_z$$

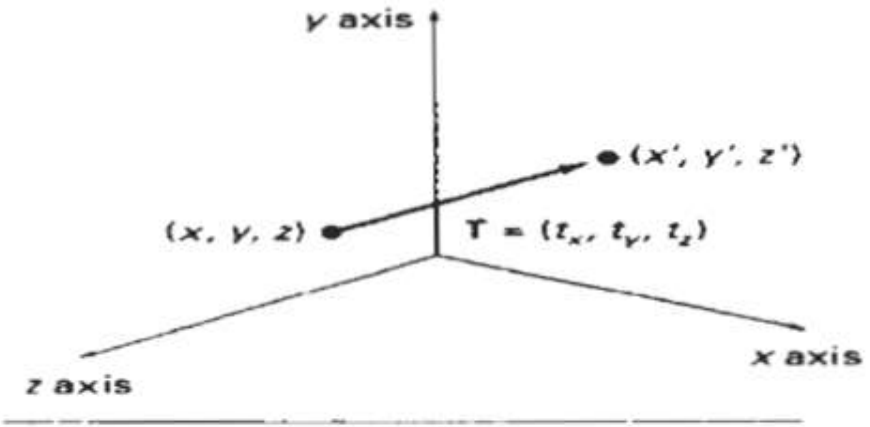
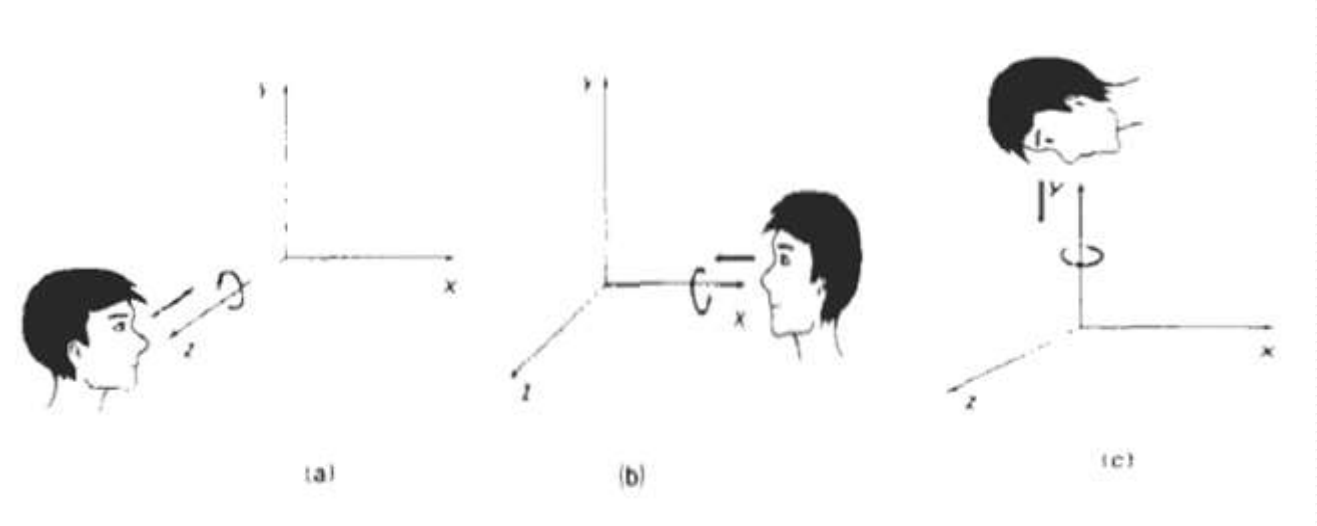


Figure 11-1
Translating a point with translation vector $\mathbf{T} = (t_x, t_y, t_z)$.

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**.



Rotation

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

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

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

$$z' = z$$

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

Rotation

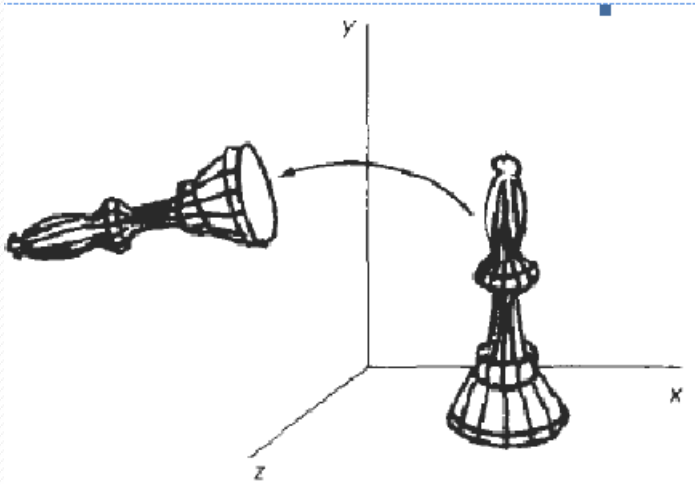


Figure 11-4
Rotation of an object about the z axis.

which we can write more compactly as

$$\mathbf{P}' = \mathbf{R}_z(\theta) \cdot \mathbf{P} \qquad (11\text{-}6)$$

Rotation

- The equations for an **x-axis rotation**:

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

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

$$x' = x$$

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

Rotation

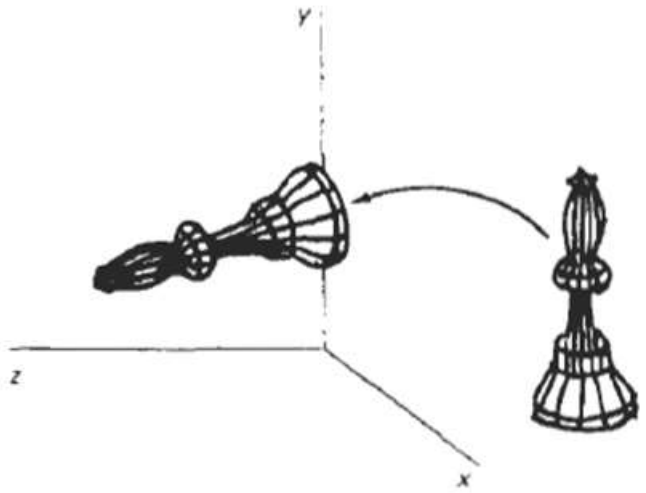


Figure 11-6
Rotation of an object about the x axis.

or

$$\mathbf{P}' = \mathbf{R}_x(\theta) \cdot \mathbf{P} \qquad (11\text{-}10)$$

Rotation

- the transformation equations for a **y-axis rotation:**

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

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

$$y' = y$$

The matrix representation for y-axis rotation is

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \qquad (11\text{-}12)$$

or

$$\mathbf{P}' = \mathbf{R}_y(\theta) \cdot \mathbf{P} \qquad (11\text{-}13)$$

Rotation

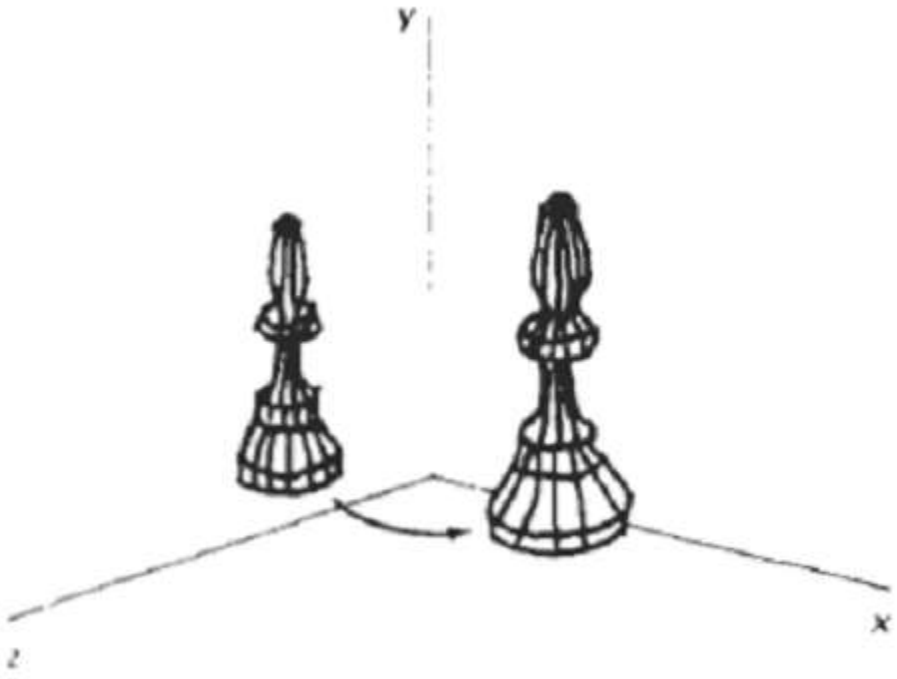


Figure 11-7
Rotation of an object about the y axis.

Scaling

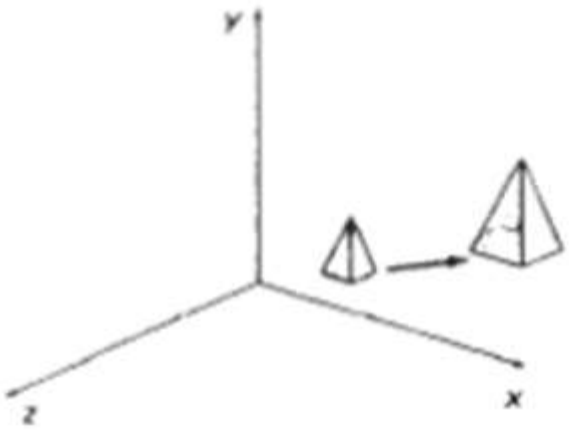
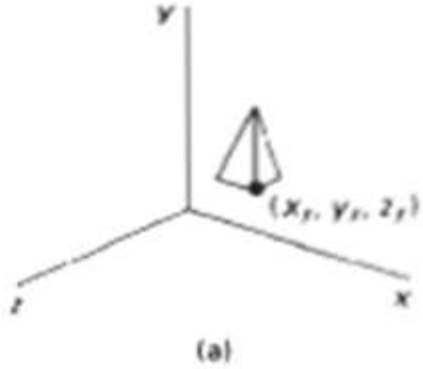


Figure 11-17
Doubling the size of an object with transformation 11-42 also moves the object farther from the origin.

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \qquad (11\text{-}42)$$



OR

$$\mathbf{P}' = \mathbf{S} \cdot \mathbf{P} \qquad (11.43)$$

Scaling

- Explicit expressions for the coordinate transformations for scaling relative to the origin are

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

Scaling with respect to a selected fixed position **(Xf, Yf, Zf,)** can be represented with the following transformation sequence:

1. Translate the fixed point to the origin.
2. Scale the object relative to the coordinate origin using Eq. 11-42.
3. Translate the fixed point back to its original position.

Scaling

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

$$\mathbf{T}(x_f, y_f, z_f) \cdot \mathbf{S}(s_x, s_y, s_z) \cdot \mathbf{T}(-x_f, -y_f, -z_f) = \begin{bmatrix} s_x & 0 & 0 & (1-s_x)x_f \\ 0 & s_y & 0 & (1-s_y)y_f \\ 0 & 0 & s_z & (1-s_z)z_f \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (11\text{-}45)$$

Visible Surface Detection

- A major consideration in the generation of realistic graphics displays is identifying those parts of a scene that are visible from a chosen viewing position.
- Numerous algorithms have been devised for efficient identification of visible objects for different types of applications.
- Some methods require more memory, some involve more processing time, and some apply only to special types of objects.

Classification of Visible Surface Detection Algorithms

- 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,
- Line display algorithms, on the other hand, generally use object-space methods to identify visible lines in wire frame displays, but many image-space visible-surface algorithms can be adapted easily to visible-line detection.

BACK-FACE DETECTION

- The equation for a plane surface can be expressed in the form:

$$Ax + By + Cz + D = 0$$

where (x, y, z) is any point on the plane, and the coefficients A, B, C, D are constants describing the spatial properties of the plane.

- A fast and simple object-space method for identifying the **back faces** of a polyhedron is based on the "**inside-outside**" tests.
- 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.

DEPTH-BUFFER/Z-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.
- For each pixel position (x, y) on the view plane, object depths can be compared by comparing z values.

DEPTH-BUFFER/Z-BUFFER METHOD

- Following figure shows three surfaces at varying distances along the orthographic projection line from position (x,y) in a view plane taken as the XvYv plane.
- Surface S1, is closest at this position, so its surface intensity value at (x, y) is saved.

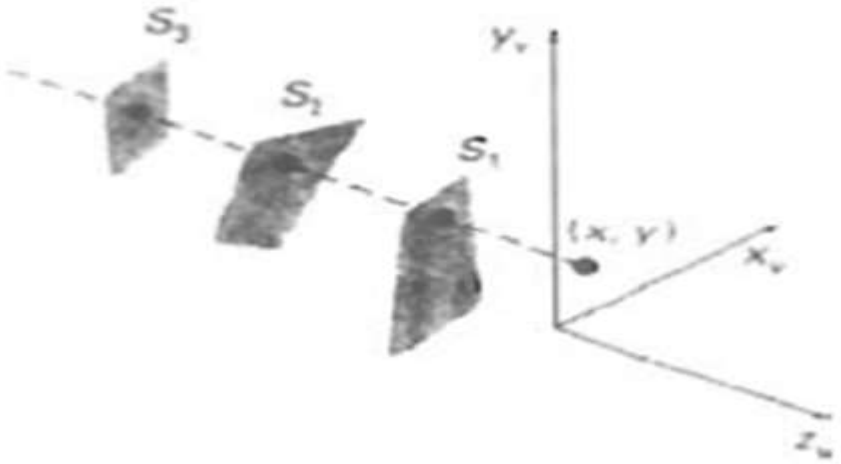


Figure 13-4
At view-plane position (x, y), surface S_1 has the smallest depth from the view plane and so is visible at that position.

DEPTH-BUFFER/Z-BUFFER METHOD

- As implied by the name of this method, two buffere areas are required:

1. A depth buffer is used to store depth values for each (x,y) position as surfaces are processed, and
2. The refresh buffer stores the intensity values for each position

- Initially, all positions in the depth buffer are set to o(minimum depth).
- The refresh buffer is initialized to the background intensity.

DEPTH-BUFFER/Z-BUFFER METHOD

- Each surface listed in the polygon tables is then processed, one scan line at a time, calculating the depth (z value) at each (x,y) pixel position.
- The calculated depth is compared to the value previously stored in the depth buffer is at that position.
- If the calculated depth is greater than the value stored in the depth buffer, the new depth value is stored, and the surface intensity at that position is determined and placed in the same x,y location in the refresh buffer.

Depth values for a surface position (x, y) are calculated from the plane equation for each surface:

$$z = \frac{-Ax - By - D}{C} \qquad (13\text{-}4)$$

Steps of Z-Buffer Algorithm

1. Initialize the depth buffer and refresh buffer so that for all buffer positions (x, y),

$$\text{depth}(x, y) = 0, \qquad \text{refresh}(x, y) = I_{\text{backgnd}}$$

2. For each position on each polygon surface, compare depth values to previously stored values in the depth buffer to determine visibility.

   - Calculate the depth z for each (x, y) position on the polygon.
   - If $z > \text{depth}(x, y)$, then set

$$\text{depth}(x, y) = z, \qquad \text{refresh}(x, y) = I_{\text{surf}}(x,y)$$

where I_{backgnd} is the value for the background intensity, and $I_{\text{surf}}(x,y)$ is the projected intensity value for the surface at pixel position (x,y). After all surfaces have been processed, the depth buffer contains depth values for the visible surfaces and the refresh buffer contains the corresponding intensity values for those surfaces.

3D Transformation Pipeline

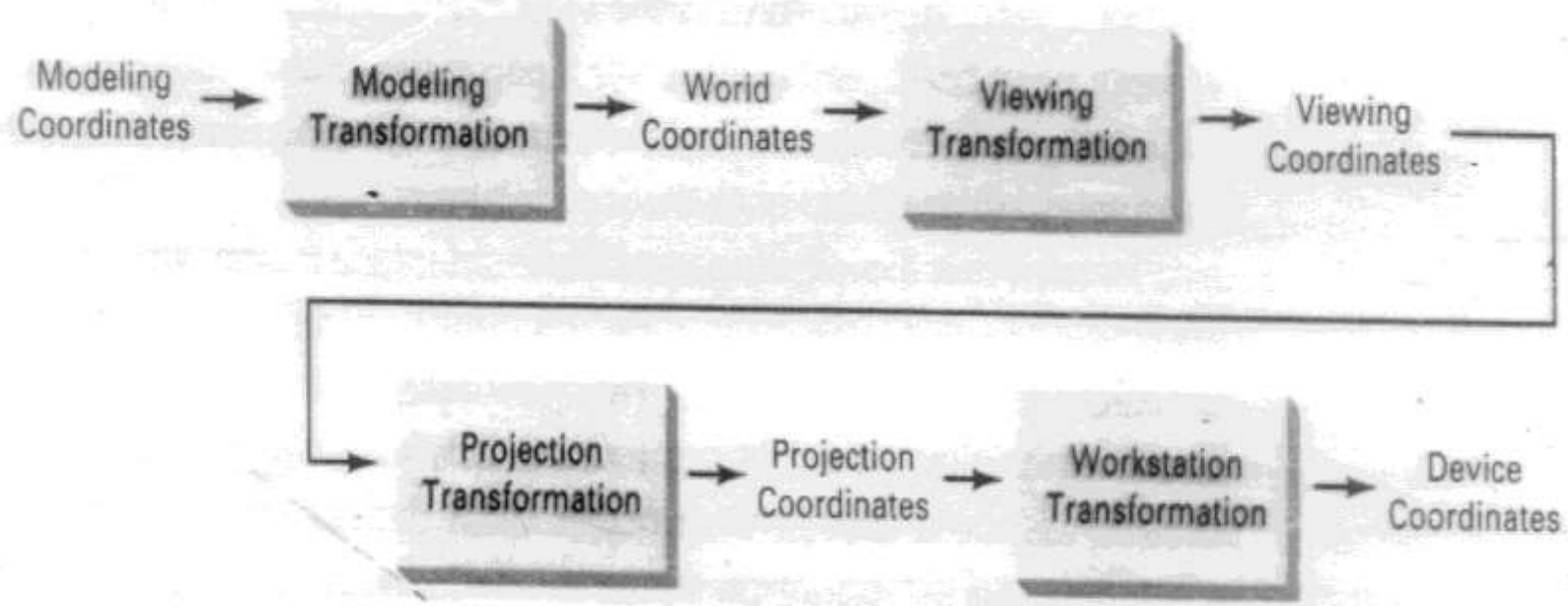


Figure 12-2
General three-dimensional transformation pipeline, from modeling coordinates to final device coordinates.