Module 4
3D Viewing and Visible Surface Detection

- 3DViewing:3D viewing concepts
- 3D viewing pipeline
- 3D viewing coordinate parameters
- Transformation from world to viewing coordinates
- Projection transformation
- Orthogonal projections
- Perspective projections
- The viewport transformation and 3D screen coordinates
- OpenGL 3D viewing functions
- Visible Surface Detection Methods
- Classification of visible surface Detection algorithms
- Back face detection
- Depth buffer method
- OpenGL visibility detection functions.

Text-1: Chapter: 7-1 to 7-10(Excluding 7-7), 9-1 to 9-3, 9-14

Overview of Three-Dimensional Viewing Concepts

Three-dimensional viewing involves many tasks like projection routines are needed to transfer the scene to a view on a planar surface, visible parts of a scene must be identified, and, for a realistic display, lighting effects and surface characteristics must be taken into account.

Viewing a Three-Dimensional Scene

Briefly discuss the three-dimensional Viewing-Coordinate Parameters

- To obtain a display of a three-dimensional world-coordinate scene, we first set up a coordinate reference for the viewing, or "camera," parameters.
- This coordinate reference defines the position and orientation for a view plane (or projection plane) that corresponds to a camera film plane (Figure 1).
- Object descriptions are then transferred to viewing reference coordinates & projected onto view plane.

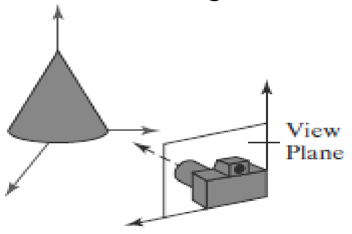
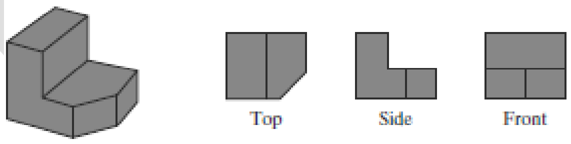


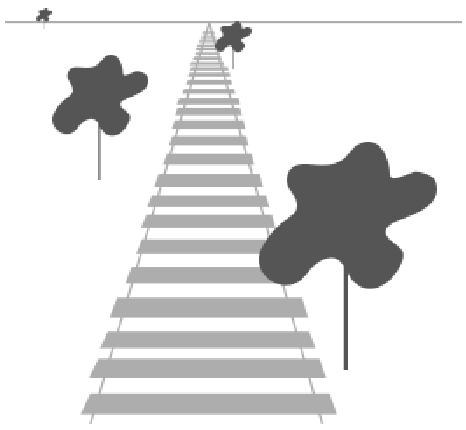
FIGURE 1
Coordinate reference for obtaining a selected view of a three-dimensional scene.

Projections

- Choose different methods for projecting a scene onto the view plane.
    1. Parallel projection 2. Perspective projection
- One method for getting the description of a solid object onto a view plane is to project points on the object surface along parallel lines.This technique, called parallel projection, is used in engineering and architectural drawings to represent an object with a set of views that show accurate dimensions
- of the object, as in Figure 2.

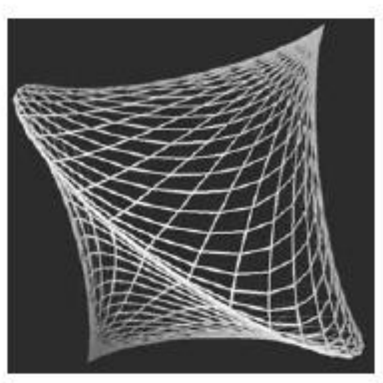


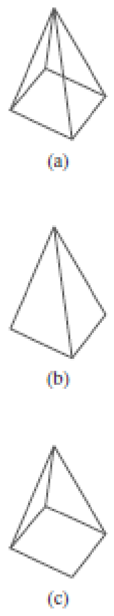
- Another method for generating a view of a three-dimensional scene is to project points to the view plane along converging paths. This process, called a perspective projection, causes objects farther from the viewing position to be displayed smaller than objects of the same size that are nearer to the viewing position. Generates more realistic images.



Depth Cueing

- With few exceptions, depth information is important in a three-dimensional scene so that we can easily identify, for a particular viewing direction, which is the front and which is the back of each displayed object.
- Figure 3 illustrates the ambiguity that can result when a wire-frame object is displayed without depth
- information.
- There are several ways to include depth information in the two-dimensional representation of solid objects.
- For example, with wire-frame displays is to vary the brightness of line segments according to their distances from the viewing position.
- Depth cueing is applied by choosing a maximum and a minimum intensity value and a range of
- distances over which the intensity is to vary.
- Another application of depth cuing is modeling the effect of the atmosphere on the perceived intensity of objects.
- More distant objects appear dimmer to us than nearer objects due to light scattering by dust particles, haze, and smoke.





Identifying Visible Lines and Surfaces

- One approach is simply to highlight the visible lines or to display them in a different color.
- Another technique is to display the non visible lines as dashed lines.
- Or could remove the non visible lines from the display.
- A wire-frame object displayed with depth cueing, so that the brightness of lines decreases from the front of the object to the back.
- When a realistic view of a scene is to be produced, back parts of the objects are completely eliminated so that only the visible surfaces are displayed. In this case, surface-rendering procedures are applied so that screen pixels contain only the color patterns for the front surfaces.

Surface Rendering

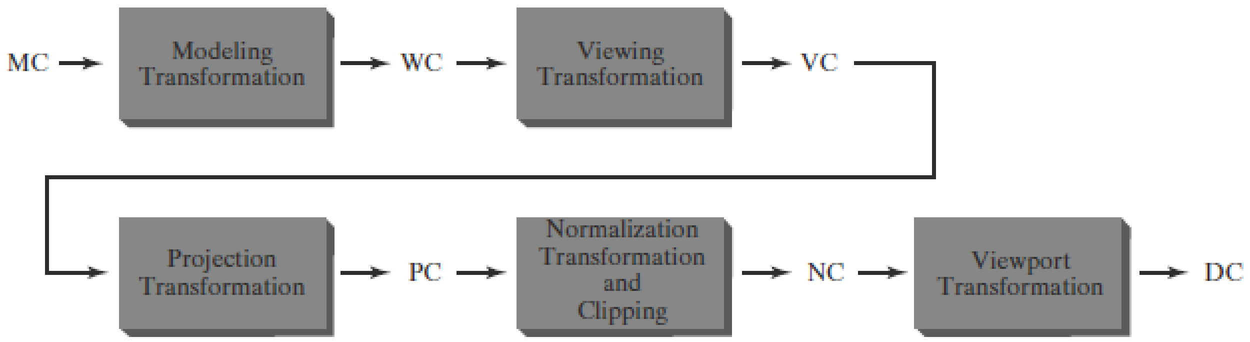
- Added realism is attained in displays by rendering object surfaces using the lighting conditions in the scene and the assigned surface characteristics.
- By specifying the color and location of the light sources, and can also set background illumination effects.

- Surface properties of objects include whether a surface is transparent or opaque and whether the
- surface is smooth or rough.
- Set values for parameters to model surfaces such as glass, plastic, wood-grain patterns, and the bumpy appearance of an orange.

Exploded and Cutaway Views

- Many graphics packages allow objects to be defined as hierarchical structures, so that internal details can be stored.
- Exploded and cutaway views of such objects can then be used to show the internal structure and relationship of the object parts.
- An alternative to exploding an object into its component parts is a cutaway view, which removes part of the visible surfaces to show internal structure.

The Three-Dimensional Viewing Pipeline



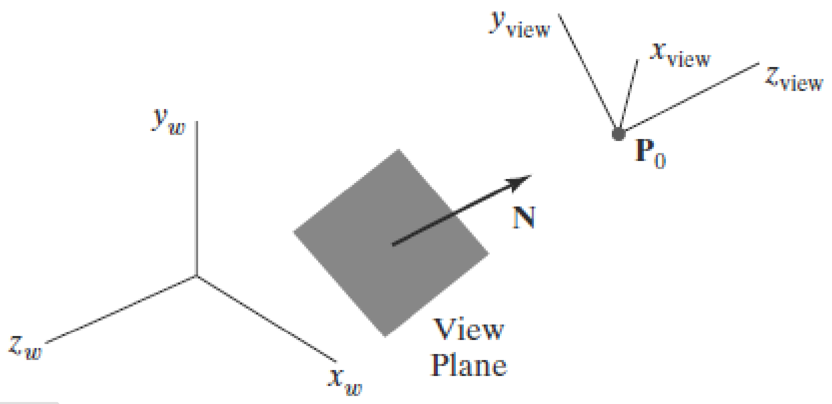
- Figure shows the general processing steps for creating and transforming a three-dimensional scene to device coordinates.
- Once the scene has been modeled in world coordinates, a viewing-coordinate system is selected and the description of the scene is converted to viewing coordinates.
- The viewing coordinate system defines the viewing parameters, including the position and orientation of the projection plane (view plane).
- A two-dimensional clipping window is defined on the projection plane, and a three-dimensional clipping

region is established.

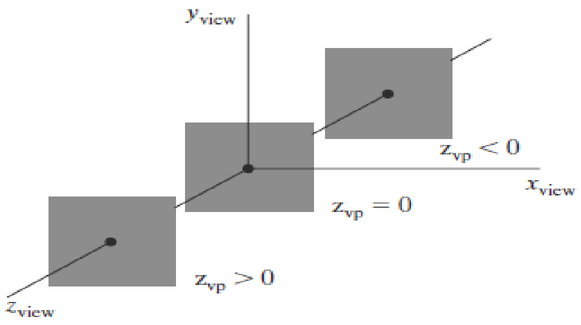
- This clipping region is called the view volume, and its shape and size depends on the dimensions of the clipping window, the type of projection we choose, and the selected limiting positions along the viewing direction.
- Projection operations are performed to convert the viewing-coordinate description of the scene to coordinate positions on the projection plane.
- Objects are mapped to normalized coordinates, and all parts of the scene outside the view volume are clipped off.
- The clipping operations can be applied after all device-independent coordinate transformations (from world coordinates to normalized coordinates) are completed.
- The viewport limits could be given in normalized coordinates or in device coordinates.
- There are also a few other tasks that must be performed, such as identifying visible surfaces and applying the surface-rendering procedures.
- The final step is to map viewport coordinates to device coordinates within a selected display window.

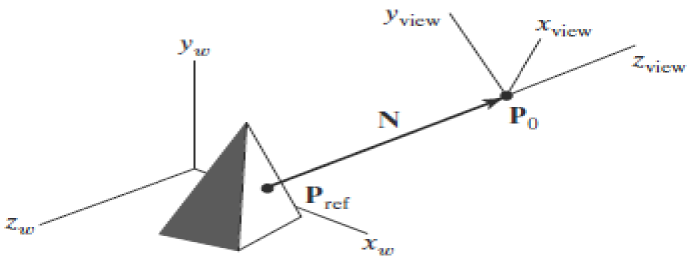
Three-Dimensional Viewing-Coordinate Parameters

- Establishing a three-dimensional viewing reference frame first select a world-coordinate position P0 =(x0, y0, z0) for the viewing origin, which is called **the view point or viewing position**.
- Specify a **view-up vector V**, which defines the **y view direction**.
- Specify **viewing direction**, along z view axis.
- Because the viewing direction is usually along the z view axis, **the view plane**, also called the projection plane, is normally assumed to be perpendicular to this axis.
- Thus, the orientation of the view plane, as well as the direction for the positive z view axis, can be defined with a **view-plane normal vector N**



- Coordinate value zvp along the z view axis is used to set the position of the view plane, as shown in below fig.





- Take **N** to be in the direction from a reference point Pref to the viewing origin P0, as in Figure above.
- In this case, the reference point is often referred to as a look-at point within the scene, with the viewing direction opposite to the direction of N.
- Set the direction for **the view-up vector V** used to establish the positive direction for the y view axis.

Transformation from World to Viewing Coordinates

- In the three-dimensional viewing pipeline, the first step after a scene has been constructed is to transfer object descriptions to the viewing-coordinate reference frame.
- This conversion of object descriptions is equivalent to a sequence of transformations that superimposes the viewing reference frame onto the world frame.
- We can accomplish this conversion using the methods for transforming between coordinate system :
  1. Translate the viewing-coordinate origin to the origin of the world coordinate system.

$$\mathbf{T} = \begin{bmatrix} 1 & 0 & 0 & -x_0 \\ 0 & 1 & 0 & -y_0 \\ 0 & 0 & 1 & -z_0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

  2. Apply rotations to align the *x*view, *y*view, and *z*view axes with the world *xw*, *yw*, and *zw* axes, respectively.

$$\mathbf{R} = \begin{bmatrix} u_x & u_y & u_z & 0 \\ v_x & v_y & v_z & 0 \\ n_x & n_y & n_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The coordinate transformation matrix is then obtained as the product of the preceding translation and rotation matrices:

$$\mathbf{M}_{WC,\,VC} = \mathbf{R} \cdot \mathbf{T} = \begin{bmatrix} u_x & u_y & u_z & -\mathbf{u} \cdot \mathbf{P}_0 \\ v_x & v_y & v_z & -\mathbf{v} \cdot \mathbf{P}_0 \\ n_x & n_y & n_z & -\mathbf{n} \cdot \mathbf{P}_0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Projection Transformations

In the next phase of the three-dimensional viewing pipeline, after the transformation to viewing coordinates, object descriptions are projected to the view plane. Graphics packages generally support both **parallel projection and perspective projections.**

Parallel Projection

- In a parallel projection, coordinate positions are transferred to the view plane along parallel lines. Fig below illustrates a parallel projection for a straight line segment defined with endpoint coordinates P1 and P2.
- A parallel projection preserves relative proportions of objects, and this is the method used in computer aided drafting and designs to produce scale drawings of three-dimensional objects.
- All parallel lines in a scene are displayed as parallel when viewed with a parallel projection.
- There are two general methods for obtaining a parallel-projection view of an object: We can project along lines that are perpendicular to the view plane, or we can project at an oblique angle to view plane.

Perspective Projection

- For a perspective projection, object positions are transformed to projection coordinates along lines that converge to a point behind the view plane.
- An example of a perspective projection for a straight-line segment, defined with endpoint coordinates **P**1 and **P**2, is given in Figure above.
- Unlike a parallel projection, a perspective projection does not preserve relative proportions of objects.
- But perspective views of a scene are more realistic because distant objects in the projected display are reduced in size.

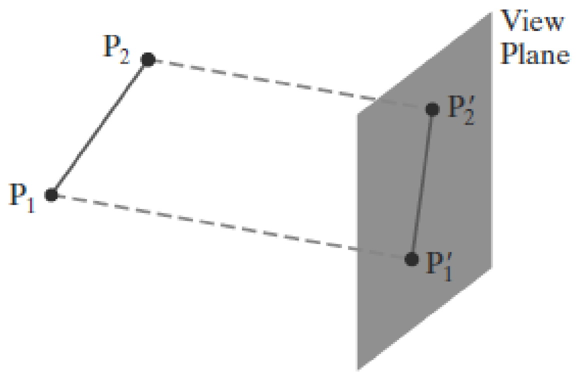


FIGURE 15
Parallel projection of a line segment onto a view plane.

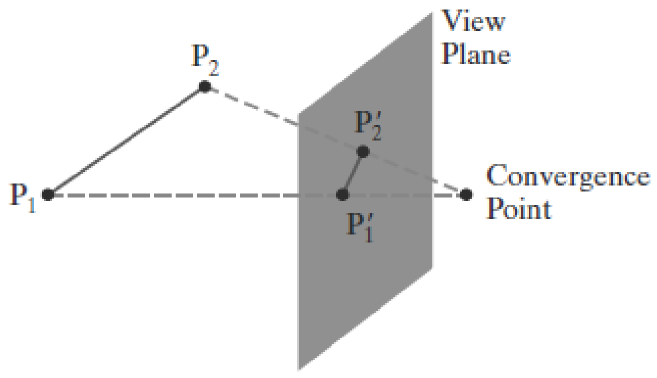
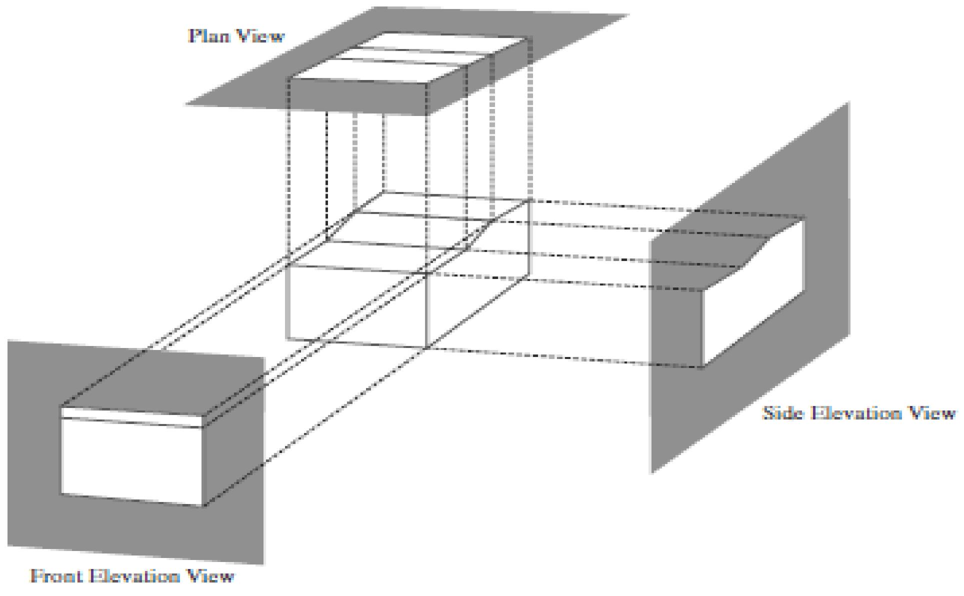


FIGURE 16
Perspective projection of a line segment onto a view plane.

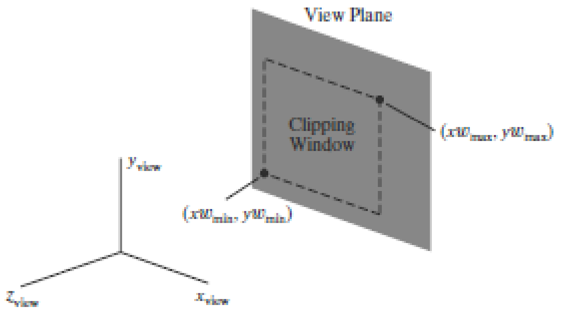
Orthogonal Projections

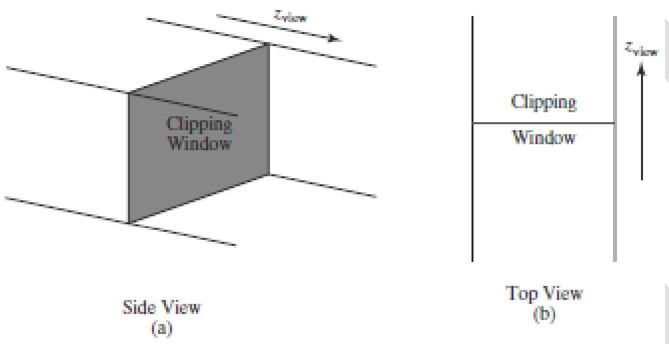
- A transformation of object descriptions to a view plane along lines that are all parallel to the view-plane normal vector N is called an **orthogonal projection** .This produces a **parallel-projection** transformation in which the projection lines are perpendicular to the view plane.
- Orthogonal projections are most often used to produce the front, side, and top projections of an object are called **elevations**; and a top orthogonal projection is called **a plan view**.
- Engineering and architectural drawings commonly employ these orthographic projections, because lengths and angles are accurately depicted and can be measured from the drawings.



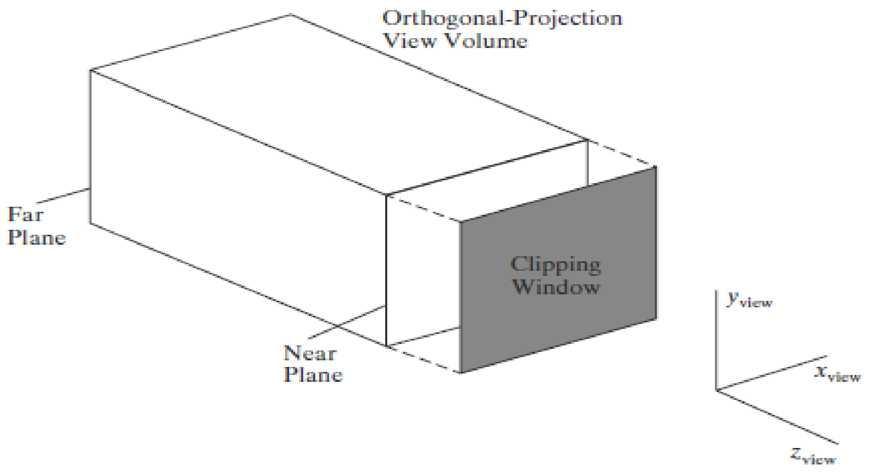
Clipping Window and Orthogonal-Projection View Volume

- For three-dimensional viewing, the clipping window is positioned on the view plane with its edges parallel shape or orientation for the clipping window.
- The edges of the clipping window specify the x and y limits for the part of the scene that we want to display. These limits are used to form the top, bottom, and two sides of a clipping region called the orthogonal-projection view volume.



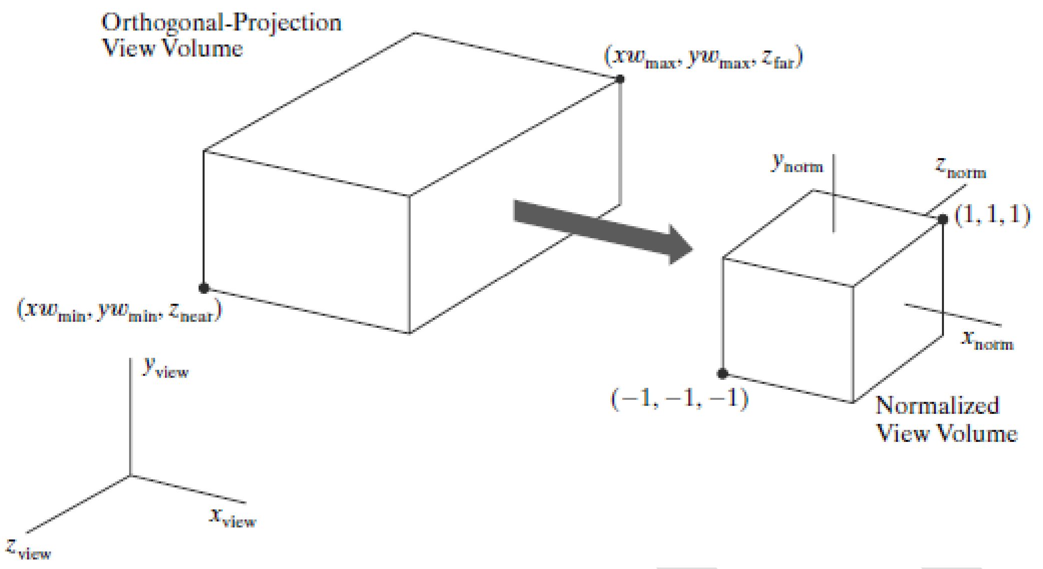


- We can limit the extent of the orthogonal view volume in the *z* view direction that are parallel to the view plane. These two planes are called the **near-far clipping planes,** or the **front-back clipping planes.**
- The near and far planes allow us to exclude objects that are in front of or behind the part of the scene that we want to display.
- With the viewing direction along the negative *z* view axis, we usually have *z*far < *z*near, so that the far plane is father out along the negative *z*view axis.
- When the near and far planes are specified, we obtain a finite orthogonal view volume that is a rectangular parallelepiped, as shown in Fig below along with one possible placement for the view plane.
- Our view of the scene will then contain only those objects within the view volume, with all parts of the scene outside the view volume eliminated by the clipping algorithms.



Normalization Transformation for an Orthogonal Projection

Transforming the rectangular-parallelepiped view volume to a normalized cube is similar to the methods for converting the clipping window into the normalized symmetric square.

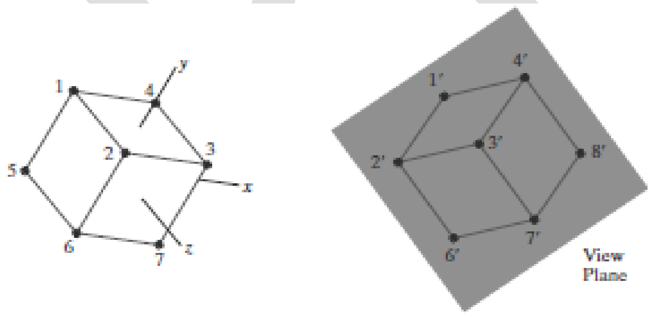


The normalization transformation for the orthogonal view volume is

$$\mathbf{M}_{ortho,norm} = \begin{bmatrix} \frac{2}{xw_{max} - xw_{min}} & 0 & 0 & -\frac{xw_{max} + xw_{min}}{xw_{max} - xw_{min}} \\ 0 & \frac{2}{yw_{max} - yw_{min}} & 0 & -\frac{yw_{max} + yw_{min}}{yw_{max} - yw_{min}} \\ 0 & 0 & \frac{-2}{z_{near} - z_{far}} & \frac{z_{near} + z_{far}}{z_{near} - z_{far}} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Axonometric and Isometric Orthogonal Projections

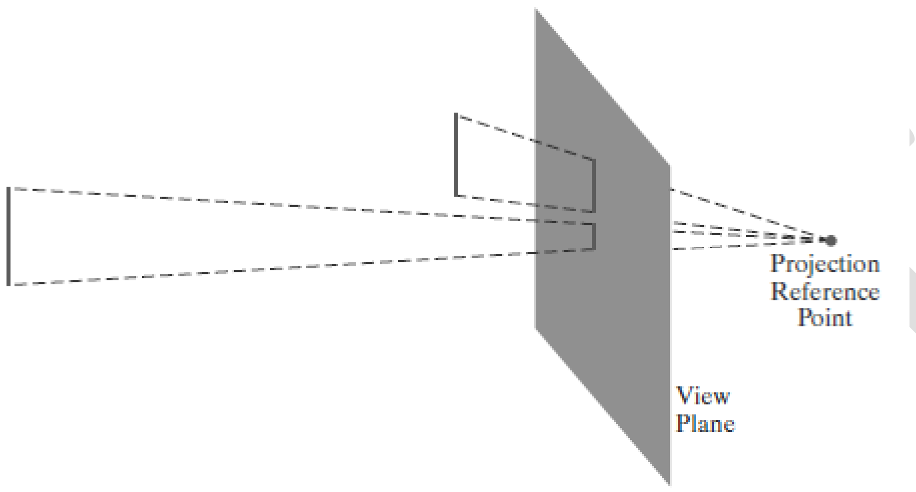
- We can also form orthogonal projections that display more than one face of an object. Such views are called **axonometric orthogonal projections**. The most commonly used axonometric projection is the **isometric projection**, which is generated by aligning the projection plane (or the object) so that the plane intersects each coordinate axis in which the object is defined, called the principal axes.



Perspective Projections

For a perspective projection, object positions are transformed to projection coordinates along lines that converge to a point behind the view plane.

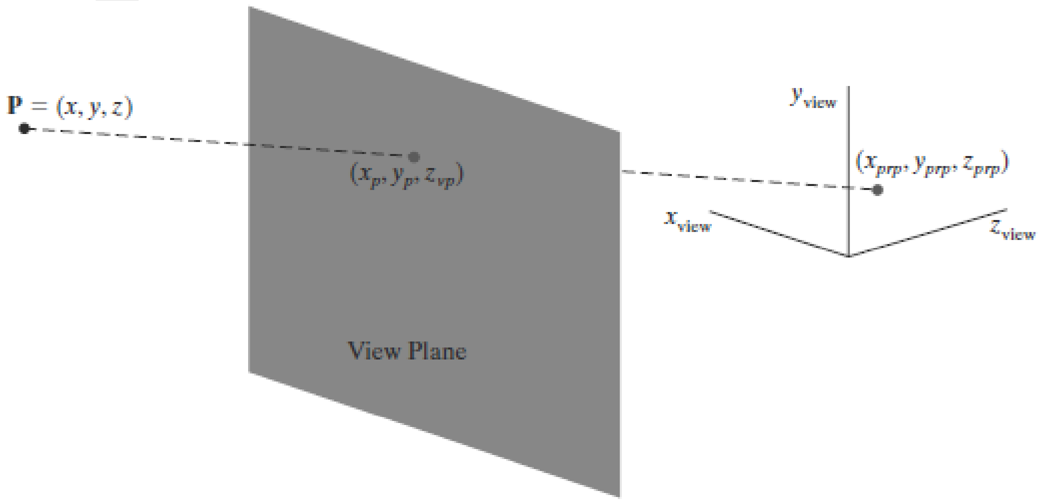
- We can approximate this geometric-optics effect by projecting objects to the view plane along converging paths to a position called the **projection reference point** (or center of projection).
- Objects are then displayed with foreshortening effects, and projections of distant objects are smaller than the projections of objects of the same size that are closer to the view plane as shown in below fig.



Perspective-Projection Transformation Coordinates

- The projection path of a spatial position (x, y, z) to a general projection reference point at (xprp, yprp, zprp). The projection line intersects the view plane at the coordinate position (xp, yp, zvp), where zvp is some selected position for the view plane on the zview axis.
- Equations describing coordinate positions along this perspective-projection line in parametric form as

$$x' = x - (x - x_{prp})u$$
$$y' = y - (y - y_{prp})u \qquad 0 \le u \le 1$$
$$z' = z - (z - z_{prp})u$$



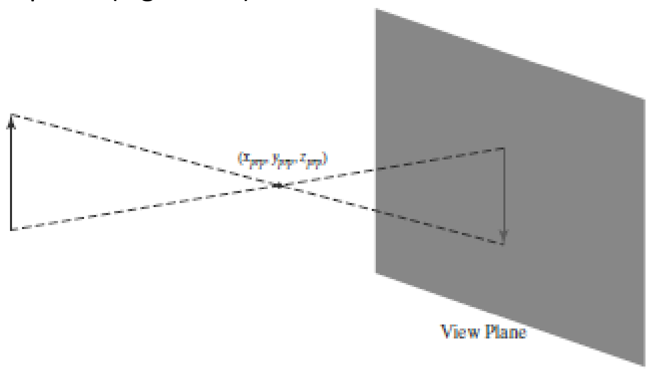
- Coordinate position (x', y', z') represents any point along the projection line.
- When u = 0, we are at position P = (x, y, z). At the other end of the line, u = 1 and we have the projection reference-point coordinates (xprp, yprp, zprp).
- On the view plane, z' = zvp & solve the z' equation for parameter u at this position along projection line:

$$u = \frac{z_{vp} - z}{z_{prp} - z}$$

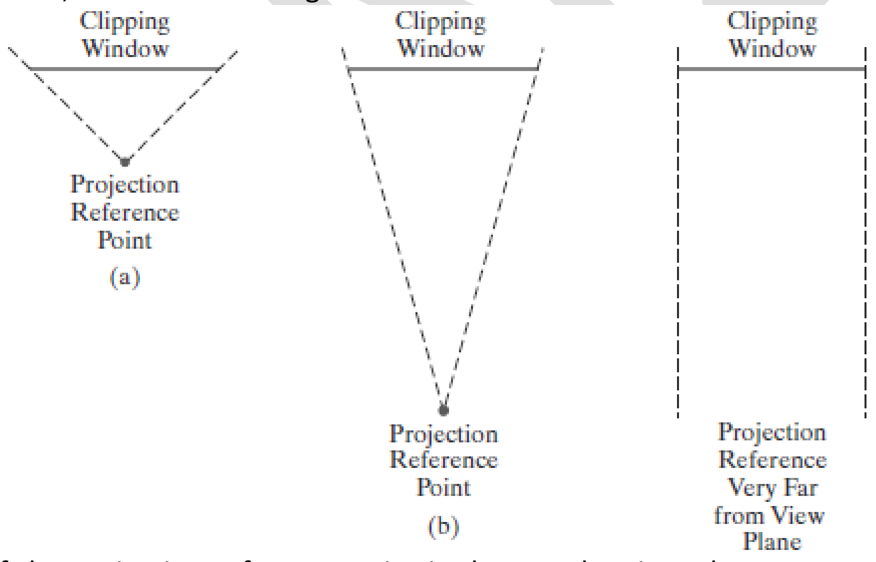
- Substituting this value of u into the equations for x' and y', we obtain the general perspective-transformation equations

$$x_p = x\left(\frac{z_{prp} - z_{vp}}{z_{prp} - z}\right) + x_{prp}\left(\frac{z_{vp} - z}{z_{prp} - z}\right)$$
$$y_p = y\left(\frac{z_{prp} - z_{vp}}{z_{prp} - z}\right) + y_{prp}\left(\frac{z_{vp} - z}{z_{prp} - z}\right)$$

- The projection reference point cannot be on the view plane. If in that case, the entire scene would project to a single point.
- The view plane is usually placed between the projection reference point and the scene, but, in general, the view plane could be placed anywhere except at the projection point.
- If the projection reference point is between the view plane and the scene, objects are inverted on the view plane (Fig below).



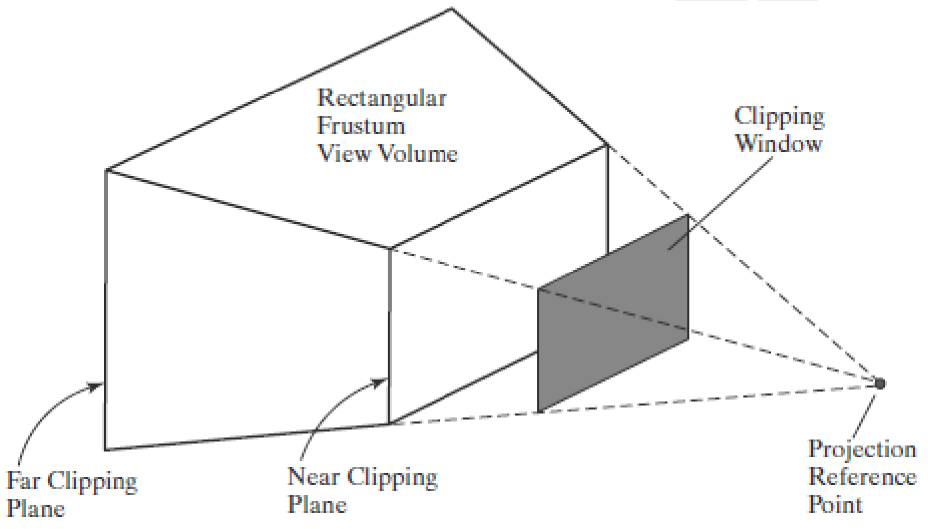
-
- With the scene between the view plane and the projection point, objects are simply enlarged as they are projected away from the viewing position onto the view plane.
- Perspective effects also depend on the distance between the projection reference point and the view plane, as illustrated in Fig below.


Clipping Window
Projection Reference Point
(a)
Clipping Window
Projection Reference Point
(b)
Clipping Window
Projection Reference Very Far from View Plane

- If the projection reference point is close to the view plane, perspective effects are emphasized; that is, closer objects will appear much larger than more distant objects of the same size.(fig a.)
- Similarly, as the projection reference point moves farther from the view plane, the difference in the size of near and far objects decreases. (fig b)
- When the projection reference point is very far from the view plane, a perspective projection approaches a parallel projection. (fig c)
-

Perspective-Projection View Volume

- View volume can be created by specifying the position of a rectangular clipping window on the view plane. But now the bounding planes for the view volume are not parallel. This forms a view volume that is an infinite rectangular pyramid with its apex at the center of projection (Figure below).
- All objects outside this pyramid are eliminated by the clipping routines.
- By adding near and far clipping planes that are perpendicular to the zview axis , chop off parts of the infinite, perspective projection view volume to form a truncated pyramid, or frustum, view volume.
- Figure illustrates the shape of a finite, perspective-projection view volume with a view plane that is placed between the near clipping plane and the projection reference point.
- Usually, both the near and far clipping planes are on the same side of the projection reference point, with the far plane farther from the projection point than the near plane along the viewing direction.



Perspective-Projection Transformation Matrix

$$\mathbf{M}_{\text{pers}} = \begin{bmatrix} z_{prp} - z_{vp} & 0 & -x_{prp} & x_{prp}z_{prp} \\ 0 & z_{prp} - z_{vp} & -y_{prp} & y_{prp}z_{prp} \\ 0 & 0 & s_z & t_z \\ 0 & 0 & -1 & z_{prp} \end{bmatrix}$$

Vanishing Points for Perspective Projections

- When a scene is projected onto a view plane using a perspective mapping, lines that are parallel to the view plane are projected as parallel lines. But any parallel lines in the scene that are not parallel to the view plane are projected into converging lines. The point at which a set of projected parallel lines appears to converge is called a **vanishing point.**
- Each set of projected parallel lines has a separate vanishing point.
- For a set of lines that are parallel to one of the principal axes of an object, the vanishing point is referred to as a **principal vanishing point.**

- We control the number of principal vanishing points (one, two, or three) with the orientation of the projection plane, and perspective projections are accordingly classified as **one-point, two-point, or three-point projections.**
- The number of principal vanishing points in a projection is equal to the number of principal axes that intersect the view plane.
- Figure 37 illustrates the appearance of one-point(fig 1) and two-point (fig 2)perspective projections for a cube.

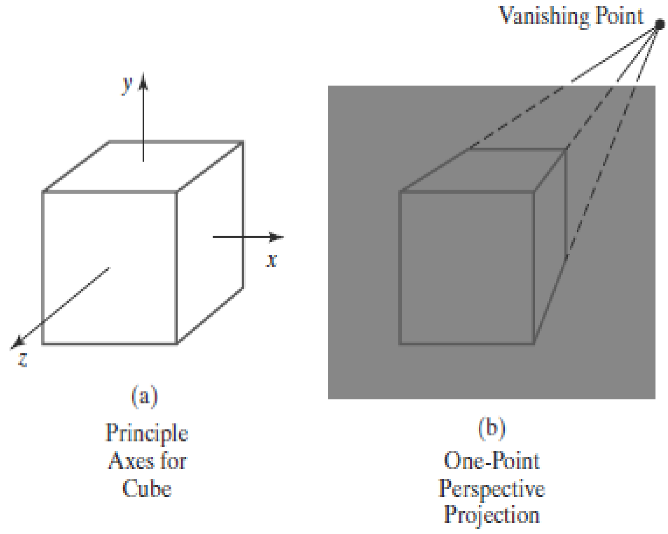


Fig 1

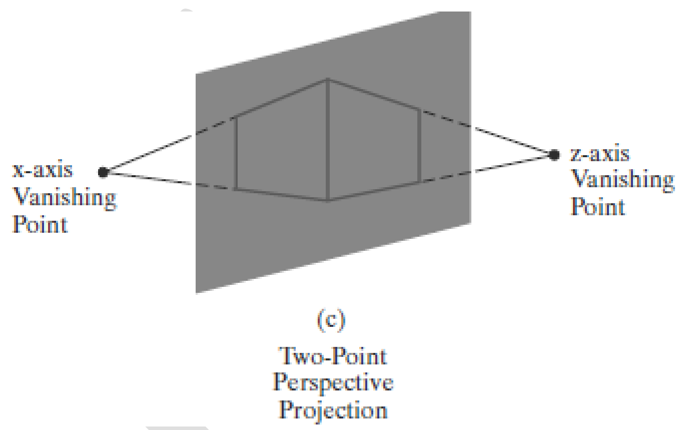
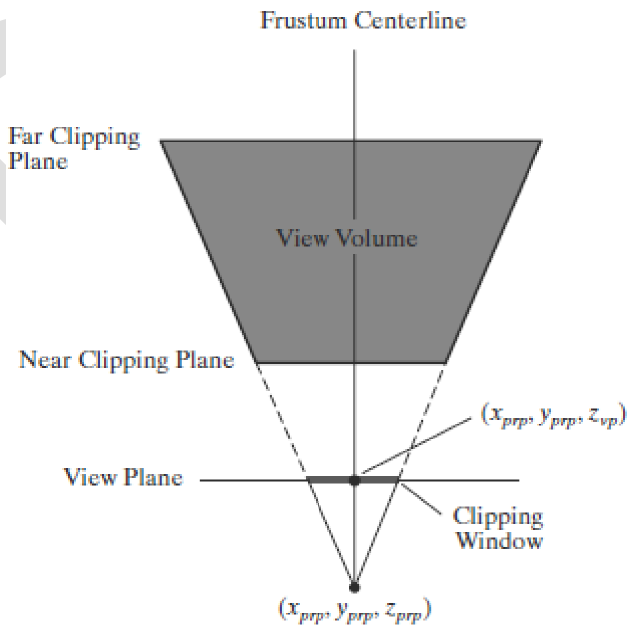


Fig 2

Symmetric Perspective-Projection Frustum

- The line from the projection reference point through the center of the clipping window and on through the view volume is the centerline for a perspective projection frustum. If this centerline is perpendicular to the view plane, we have a symmetric frustum (with respect to its centerline) as in Figure below

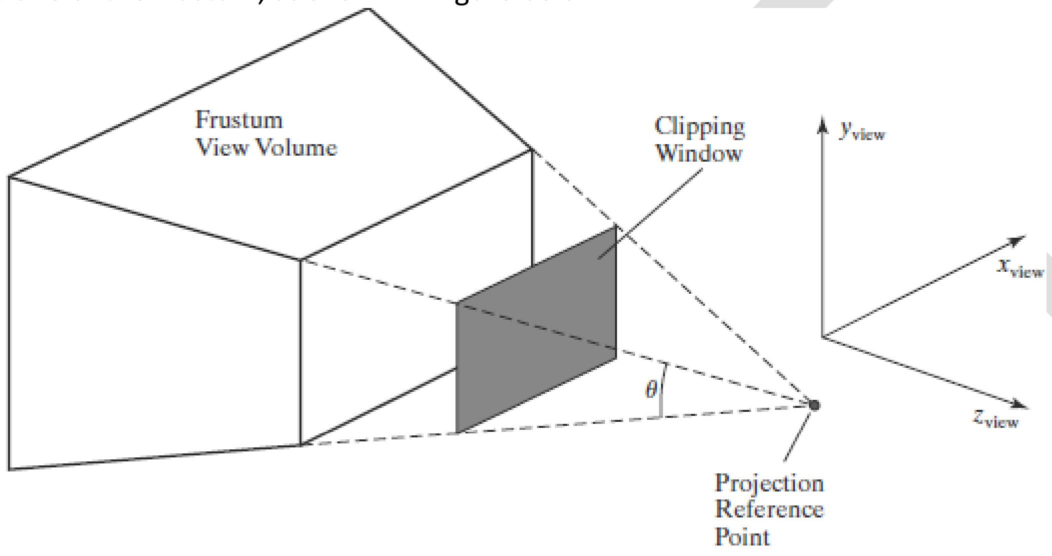


- Because the frustum centerline intersects the view plane at the coordinate location (xprp, yprp, zvp), we can express the corner positions for the clipping window in terms of the window dimensions:

$$xw_{\min} = x_{prp} - \frac{\text{width}}{2}, \qquad xw_{\max} = x_{prp} + \frac{\text{width}}{2}$$
$$yw_{\min} = y_{prp} - \frac{\text{height}}{2}, \qquad yw_{\max} = y_{prp} + \frac{\text{height}}{2}$$

- Therefore, we could specify a symmetric perspective-projection view of a scene using the width and height of the clipping window instead of the window coordinates.

- Another way to specify a symmetric perspective projection is to use the cone of vision which can be referenced with a **field-of-view angle**, which is a measure of the size of the camera lens.
- A large field-of-view angle, for example, corresponds to a wide-angle lens.
- In computer graphics, the cone of vision is approximated with a symmetric frustum, and we can use a field-of-view angle to specify an angular size for the frustum.
- Typically, the field-of-view angle is the angle between the top clipping plane and the bottom clipping plane of the frustum, as shown in Figure below.

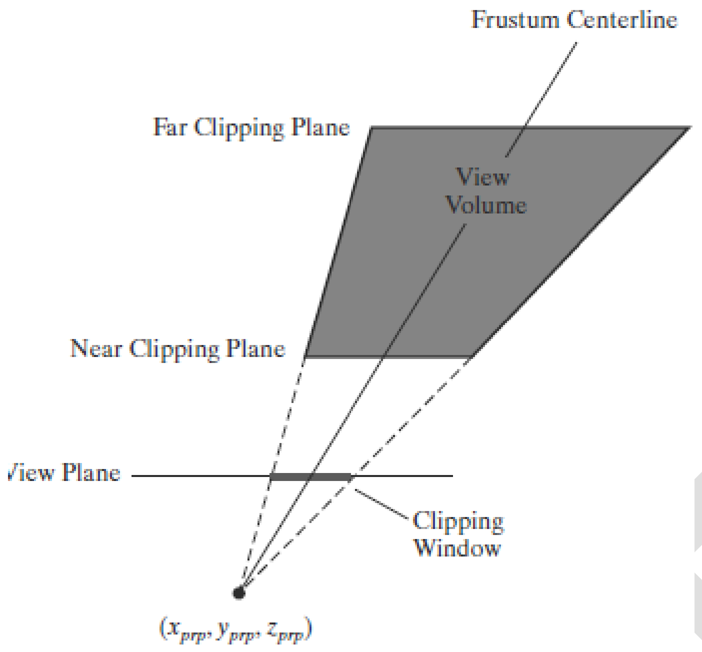


- For a given projection reference point and view-plane position, the field-of view angle determines the height of the clipping window(above fig), but not the width.
- We need an additional parameter to define completely the clipping window dimensions, and this second parameter could be either the **window width or the aspect ratio** (width/height) of the clipping window.
- From the right triangles in the diagram of Figure above , the clipping-window height can be calculated as

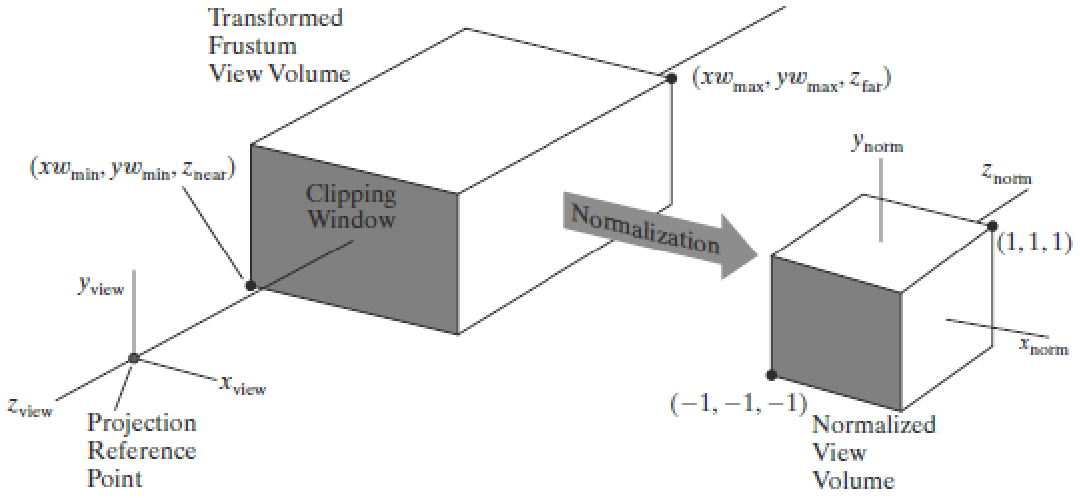
$$\text{height} = 2(z_{prp} - z_{vp}) \tan\left(\frac{\theta}{2}\right)$$

Oblique Perspective-Projection Frustum

- If the centerline of a perspective-projection view volume is not perpendicular to the view plane, we have an oblique frustum.
- An oblique perspective-projection view volume can be converted to a symmetric frustum by applying a z-axis shearing-transformation matrix. This transformation shifts all positions on any plane that is perpendicular to the z axis by an amount that is proportional to the distance of the plane from a specified z- axis reference position



Normalized Perspective-Projection Transformation Coordinates



The elements of the normalized transformation matrix for a general perspective-projection are

$$\mathbf{M}_{\text{normpers}} = \begin{bmatrix} \frac{-2z_{\text{near}}}{xw_{\max} - xw_{\min}} & 0 & \frac{xw_{\max} + xw_{\min}}{xw_{\max} - xw_{\min}} & 0 \\ 0 & \frac{-2z_{\text{near}}}{yw_{\max} - yw_{\min}} & \frac{yw_{\max} + yw_{\min}}{yw_{\max} - yw_{\min}} & 0 \\ 0 & 0 & \frac{z_{\text{near}} + z_{\text{far}}}{z_{\text{near}} - z_{\text{far}}} & -\frac{2z_{\text{near}}z_{\text{far}}}{z_{\text{near}} - z_{\text{far}}} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

OpenGL Three-Dimensional Viewing Functions

OpenGL Viewing-Transformation Function

- When we designate the viewing parameters in OpenGL, a matrix is formed and concatenated with the current modelview matrix. Consequently, this viewing matrix is combined with any geometric transformations we may have also specified. This composite matrix is then applied to transform object descriptions in world coordinates to viewing coordinates. We set the modelview mode with the statement:

  glMatrixMode (GL_MODELVIEW);

- **Viewing parameters** are specified with the following GLU function, which is in the OpenGL Utility library because it invokes the translation and rotation routines in the basic OpenGL library.

  gluLookAt (x0, y0, z0, xref, yref, zref, Vx, Vy, Vz);
- Values for all parameters are to be assigned double-precision, floating-point values.
- This function designates the origin of the viewing reference frame as the world-coordinate position P0 = (x0, y0, z0), the reference position as Pref =(xref, yref, zref), and the view-up vector as V = (Vx, Vy, Vz).
- If we do not invoke the gluLookAt function, the default OpenGL viewing parameters are

  P0 = *(*0, 0, 0*)* **P**ref = *(*0, 0, −1*)* **V** = *(*0, 1, 0*)*

OpenGL Orthogonal-Projection Function

- Projection matrices are stored in the OpenGL projection mode. So, to set up a projection-transformation matrix, we must first invoke that mode with the

  glMatrixMode (GL_PROJECTION);
- Then, when we issue any transformation command, the resulting matrix will be concatenated with the current projection matrix.
- Orthogonal-projection parameters are chosen with the function

  glOrtho (xwmin, xwmax, ywmin, ywmax, dnear, dfar);
- All parameter values are to be assigned double-precision, floating pointnumbers.
- We use glOrtho to select the clipping-window coordinates and the distances to the near and far clipping planes from the viewing origin.
- There is no option in OpenGL for the placement of the view plane. The near clipping plane is always also the view plane, and therefore the clipping window is always on the near plane of the view volume.
- Function **glOrtho** generates a parallel projection that is perpendicular to the view plane
- In OpenGL, the near and far clipping planes are not optional;
- This default is equivalent to issuing the statement glOrtho (-1.0, 1.0, -1.0, 1.0, -1.0, 1.0);
- The default clipping window is thus a symmetric normalized square, and the default view volume is a symmetric normalized cube with znear = 1.0 (behind the viewing position) and zfar = −1.0.

OpenGL Symmetric Perspective-Projection Function

- There are two functions available for producing a perspective-projection view of a scene.
  * One of these functions generates a symmetric frustum view volume about the viewing direction (the negative zview axis).
  * The other function can be used for either a symmetric-perspective projection or an oblique-perspective projection.
- For both functions, the projection reference point is the viewing-coordinate origin and the near clipping

plane is the view plane.

- A symmetric, perspective-projection, frustum view volume is set up with the GLU function
  gluPerspective (theta, aspect, dnear, dfar);
- with each of the four parameters assigned a double-precision, floating-point number.
- The first two parameters define the size and position of the clipping window on the near plane, and the second two parameters specify the distances from the view point (coordinate origin) to the near and far clipping planes.
- Parameter theta represents the field-of-view angle, which is the angle between the top and bottom clipping planes This angle can be assigned any value from 0◦ to 180◦.
- Parameter aspect is assigned a value for the aspect ratio (width/height) of the clipping window.
- The near and far clipping planes must always be somewhere along the negative zview axis; neither can be "behind" the viewing position. This restriction does not apply to an orthogonal projection, but it precludes the inverted perspective projection of an object when the view plane is behind the view point.
- Therefore, both dnear and dfar must be assigned positive numerical values, and the positions of the near and far planes are calculated as znear = −dnear and zfar = −dfar.

OpenGL General Perspective-Projection Function

- We can use the following function to specify a perspective projection that has either a symmetric frustum view volume or an oblique frustum view volume.
  glFrustum (xwmin, xwmax, ywmin, ywmax, dnear, dfar);
- All parameters are assigned double-precision, floating-point numbers.
- The near plane is the view plane and the projection reference point is at the viewing position (coordinate origin).
- This function has the same parameters as the orthogonal, parallel-projection function, but now the near and far clipping-plane distances must be positive.
- The first four parameters set the coordinates for the clipping window on the near plane, and the last two parameters specify the distances from the coordinate origin to the near and far clipping planes along the negative zview axis.
- Locations for the near and far planes are calculated as znear = −dnear and zfar = −dfar.

OpenGL Viewports and Display Windows

- A rectangular viewport is defined with the follow
  glViewport (xvmin, yvmin, vpWidth, vpHeight);
- The first two parameters in this function specify the integer screen position of the lower-left corner of the viewport relative to the lower-left corner of the display window.
- And the last two parameters give the integer width and height of the viewport.
- To maintain the proportions of objects in a scene, we set the aspect ratio of the viewport equal to the aspect ratio of the clipping window.

Visible-Surface Detection Algorithms

A major consideration in the generation of realistic graphics displays is determining what is visible within a scene from a chosen viewing position.

Classification of Visible-Surface Detection Algorithms

We can broadly classify visible-surface detection algorithms according to whether they deal with the object definitions 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.

Line-display algorithms, for instance, 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.
Visible-surface detection algorithms mostly 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.
Coherence methods are used to take advantage of regularities in a scene.

Back-Face Detection

A fast and simple object-space method for locating the back faces of a polyhedron

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

where A, B,C, and Dare the plane parameters for the polygon. When this position is along the line of sight to the surface, we must be looking at the back of the polygon.
Therefore, we could use the viewing position to test for back faces.

We can simplify the back-face test by considering the direction of the normal vector N for a polygon surface. If Vview is a vector in the viewing direction from our camera position, as shown in Figure 1, then a polygon is a back face if

$$V_{view} \cdot N > 0$$

In a right-handed viewing system with the viewing direction along the negative zv axis, a polygon is a back face if the z component, C, of its normal vector N satisfies C < 0.
Also, we cannot see any face whose normal has z component C = 0, because 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 that satisfies the inequality

$$C \leq 0$$

back faces have normal vectors that point away from the viewing position and are identified by C ≥ 0 when the viewing direction is along the positive zv axis.

Visible-Surface Detection Methods

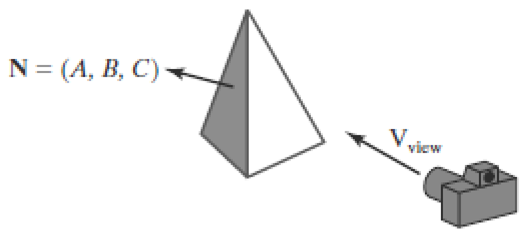


FIGURE 1
A surface normal vector **N** and the viewing-direction vector $\mathbf{V}_{view}$.

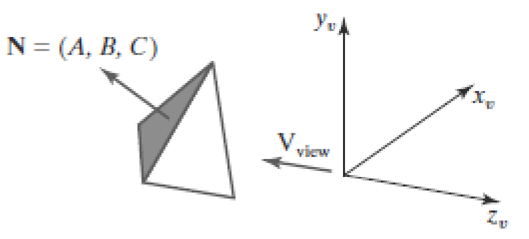


FIGURE 2
A polygon surface with plane parameter $C < 0$ in a right-handed viewing coordinate system is identified as a back face when the viewing direction is along the negative z_v axis.

Depth-Buffer Method

- A commonly used image-space approach for detecting visible surfaces is the depth-buffer method, which compares surface depth values throughout a scene for each pixel position on the projection plane.
- Each surface of a scene is processed separately, one pixel position at a time, across the surface.
- The algorithm is usually applied to scenes containing only polygon surfaces, because depth values can be computed very quickly and the method is easy to implement.
- This visibility-detection approach is also frequently alluded to as the **z-buffer method**, because object depth is usually measured along the z axis of a viewing system.
- Figure 4 shows three surfaces at varying distances along the orthographic projection line from position (x, y) on a view plane. These surfaces can be processed in any order.
- As each surface is processed, its depth from the view plane is compared to previously processed surfaces.
- If a surface is closer than any previously processed surfaces, its surface color is calculated and saved, along with its depth.
- The visible surfaces in a scene are represented by the set of surface colors that have been saved after all surface processing is completed.
- Implementation of the depth-buffer algorithm is typically carried out in normalized coordinates,
- so that depth values range from 0 at the near clipping plane (the view plane) to 1.0 at the far clipping plane.
- Two buffer areas are required.
- A **depth buffer** is used to store depth values for each *(x, y)* position as surfaces are processed, and the frame buffer stores the surface-color values for each pixel position.
- Initially, all positions in the depth buffer are set to 1.0 (maximum depth), and the frame buffer (refresh buffer) is initialized to the background color.
- Each surface listed in the polygon tables is then processed, one scan line at a time, by calculating the depth value at each *(x, y)* pixel position.
- This calculated depth is compared to the value previously stored in the depth buffer for that pixel position.
- If the calculated depth is less than the value stored in the depth buffer, the new depth value is

stored.

- Then the surface color at that position is computed and placed in the corresponding pixel location in the frame buffer.
- This algorithm assumes that depth values are normalized on the range from 0.0 to 1.0 with the view plane at depth= 0 and the farthest depth= 1.

Depth-Buffer Algorithm

```
1. Initialize the depth buffer and frame buffer so that for all buffer positions (x, y),
       depthBuff (x, y) = 1.0, frameBuff (x, y) = backgndColor

2. Process each polygon in a scene, one at a time, as follows:
       • For each projected (x, y) pixel position of a polygon, calculate the depth z (if not already
         known).
       • If z < depthBuff (x, y), compute the surface color at that position and set
             depthBuff (x, y) = z, frameBuff (x, y) = surfColor (x, y)
```

After all surfaces have been processed, the depth buffer contains depth values for the visible surfaces and the frame buffer contains the corresponding color values for those surfaces.

OpenGL Visibility-Detection Functions

OpenGL Polygon-Culling Functions
Back-face removal is accomplished with the functions

```
glEnable (GL_CULL_FACE);
glCullFace (mode);
```

where parameter mode is assigned the value GL BACK.
In fact, we could use this function to remove the front faces instead, or we could even remove both front and back faces.
By default, parameter mode in the glCullFace function has the value GL BACK.

OpenGL Depth-Buffer Functions
To use the OpenGL depth-buffer visibility-detection routines, we first need to modify the GL Utility Toolkit (GLUT) initialization function for the display mode to include a request for the depth buffer, as well as for the refresh buffer. We do this, for example, with the statement

```
glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);
```

Depth buffer values can then be initialized with

```
glClear (GL_DEPTH_BUFFER_BIT);
```

Normally, the depth buffer is initialized with the same statement that initializes the refresh buffer to the background color.

The OpenGL depth-buffer visibility-detection routines are activated with the following function:

glEnable (GL_DEPTH_TEST);

And we deactivate the depth-buffer routines with

glDisable (GL_DEPTH_TEST);

We can also apply depth-buffer visibility testing using some other initial value for themaximumdepth, and this initial value is chosen with theOpenGLfunction:

glClearDepth (maxDepth);

Parameter maxDepth can be set to any value between 0.0 and 1.0.

Projection coordinates in OpenGL are normalized to the range from −1.0 to 1.0, and the depth values between the near and far clipping planes are further normalized to the range from 0.0 to 1.0. The value 0.0 corresponds to the near clipping plane (the projection plane), and the value 1.0 corresponds to the far clipping plane. As an option, we can adjust these normalization values with

glDepthRange (nearNormDepth, farNormDepth);

By default, nearNormDepth = 0.0 and farNormDepth = 1.0.

the test condition that is to be used for the depth-buffer routines.We specify a test condition with the following function:

glDepthFunc (testCondition);

Parameter testCondition can be assigned any one of the following eight symbolic constants: **GL LESS, GL GREATER, GL EQUAL, GL NOTEQUAL, GL LEQUAL, GL GEQUAL, GL NEVER (no points are processed), and GL ALWAYS**(all points are processed).
The default value for parameter test Condition is GL LESS.

We can also set the status of the depth buffer so that it is in a read-only state or in a read-write state. This is accomplished with

glDepthMask (writeStatus);

When writeStatus = **GL_TRUE** (the default value), we can both read from and write to the depth buffer.With writeStatus **= GL_FALSE**, the write mode for the depth buffer is disabled and we can retrieve values only for comparison in depth testing.

OpenGL Wire-Frame Surface-Visibility Methods

A wire-frame display of a standard graphics object can be obtained in OpenGL by requesting that only its edges are to be generated. We do this by setting the polygon-mode function as, for example:

glPolygonMode (GL_FRONT_AND_BACK, GL_LINE);

But this displays both visible and hidden edges.

OpenGL Depth-Cueing Function

We can vary the brightness of an object as a function of its distance from the viewing position with

glEnable (GL_FOG);
glFogi (GL_FOG_MODE, GL_ LINEAR);

This applies the linear depth function in Eq. 9 to object colors using dmin = 0.0 and dmax = 1.0. But we can set different values for dmin and dmax with the following function calls:

glFogf (GL_FOG_START, minDepth);
glFogf (GL_FOG_END, maxDepth);

Question Bank

1. Illustrate 3D viewing pipeline with diagram.
2. Illustrate with diagram perspective projection frustrum view volume
3. Discuss viewing parameters.
4. Define the following: View plane and perspective reference point
5. Differentiate between parallel projection and perspective projection.
6. What is vanishing point and principal vanishing point?
7. Explain different types of perspective projections with respect to vanishing point.
8. Discuss the two classifications of visible-surface detection algorithm.
9. a). gluPrespective() b). glFrustrum()
10. c). glLookAt() d). GlViewport()
11. Define Orthogonal Projection.
12. Discuss Depth buffer algorithm.
13. Classification of visible surface Detection algorithms
14. Explain the following OpenGL visibility detection Functions
    i). glCullFace() ii). glClearDepth()
    iii). glDepthRange() iv). glFogi() v). glDepthFunc()

END