Acknowledgements to

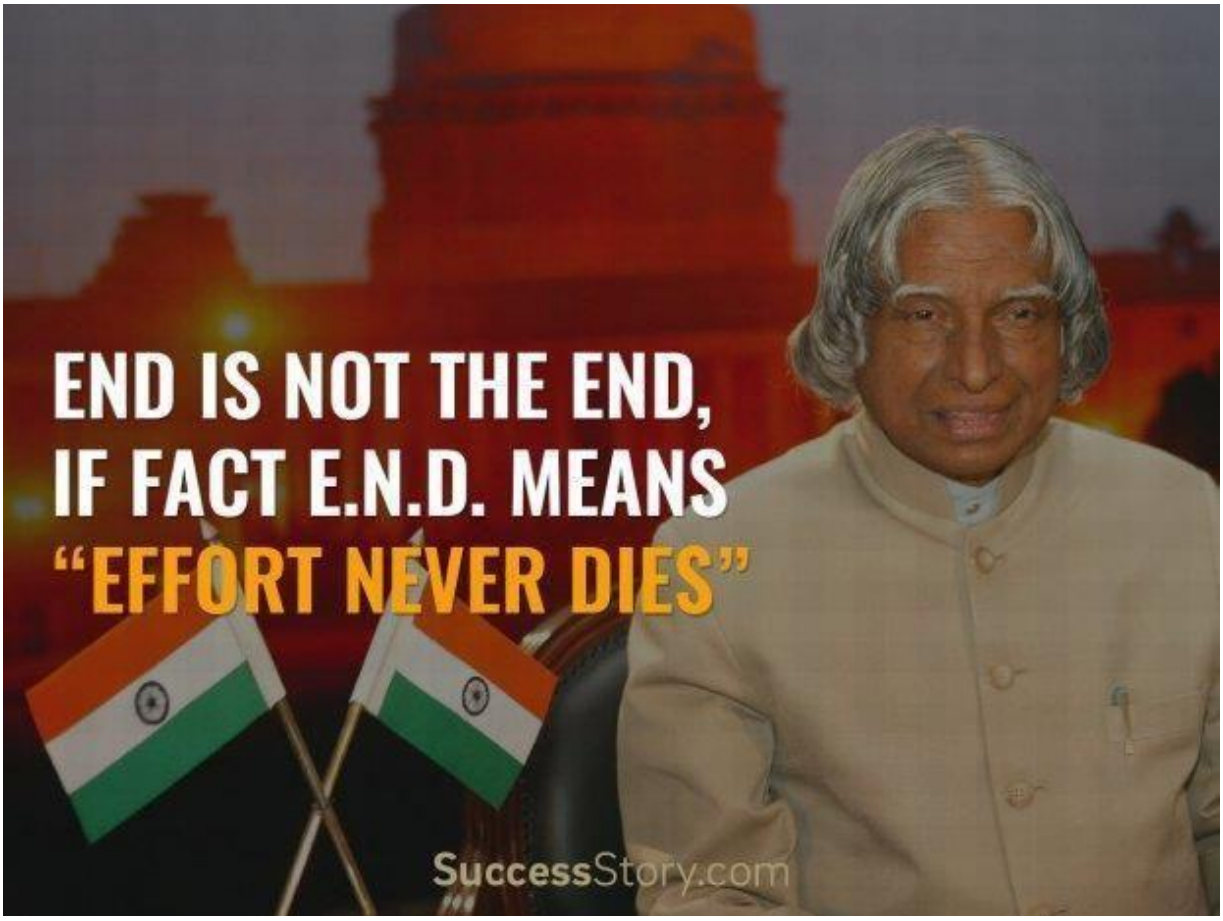
Donald Hearn & Pauline Baker: Computer Graphics with OpenGL

Version,3rd / 4th Edition, Pearson Education,2011

Edward Angel: Interactive Computer Graphics- A Top Down approach

with OpenGL, 5th edition. Pearson Education, 2008

M M Raiker, Computer Graphics using OpenGL, Filip learning/Elsevier



4 3D Viewing and Visible Surface Detection

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 viewingfunctions.
Visible Surface Detection Methods:
Classification of visible surface Detection algorithms,
4.11Back face detection,
Depth buffer method and
OpenGL visibility detection functions.

Three-Dimensional Viewing

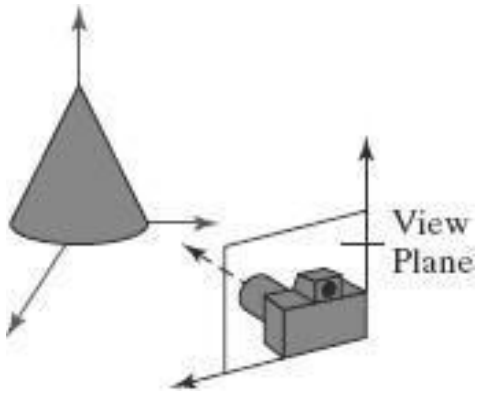
Overview of Three-Dimensional Viewing Concepts

- When we model a three-dimensional scene, each object in the scene is typically defined with a set of surfaces that form a closed boundary around the object interior.
- In addition to procedures that generate views of the surface features of an object, graphics packages sometimes provide routines for displaying internal components or cross-sectional views of a solid object.
- Many processes in three-dimensional viewing, such as the clipping routines, are similar to those in the two-dimensional viewing pipeline.
- But three-dimensional viewing involves some tasks that are not present in twodimensional Viewing

Viewing a Three-Dimensional Scene

- 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 as shown in below figure.

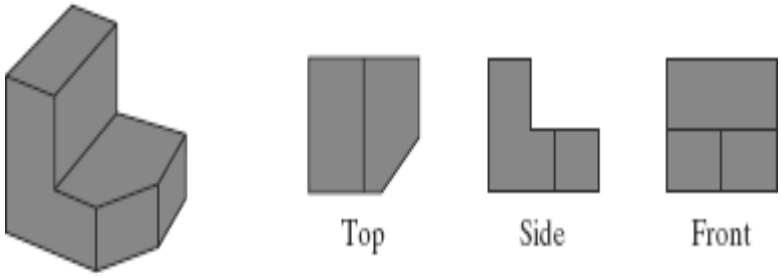


Coordinate reference for obtaining a selected view of a three-dimensional scene.

- We can generate a view of an object on the output device in wireframe (outline) form, or we can apply lighting and surface-rendering techniques to obtain a realistic shading of the visible surfaces Projections

Two methods:

1. 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*



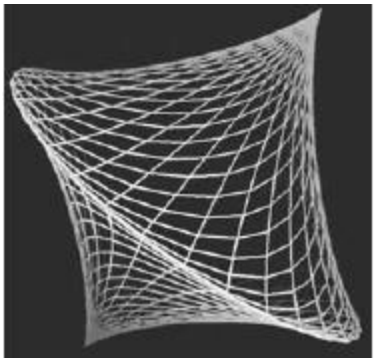
Three parallel-projection views of an object, showing relative proportions from different viewing positions

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

Depth Cueing

- ✓ 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.
- ✓ There are several ways in which we can include depth information in the two-dimensional representation of solid objects.
- ✓ A simple method for indicating depth with wire-frame displays is to vary the brightness of line segments according to their distances from the viewing position which is termed as depth cueing.



A wire-frame object displayed with depth cueing, so that the brightness of lines decreases from the front of the object to the back

- ✓ The lines closest to the viewing position are displayed with the highest intensity, and lines farther away are displayed with decreasing intensities.
- ✓ 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

Identifying Visible Lines and Surfaces

- ➢ One approach is simply to highlight the visible lines or to display them in a different color. Another technique, commonly used for engineering drawings, is to display the

nonvisible lines as dashed lines. Or we could remove the nonvisible lines from the display

Surface Rendering

- ➔ We set the lighting conditions by specifying the color and location of the light sources, and we 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.
- ➔ We 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

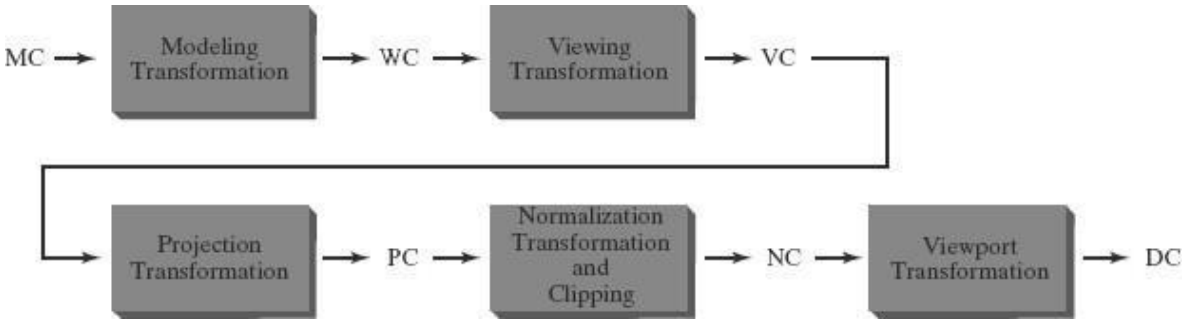
- ➢ Exploded and cutaway views of such objects can then be used to show the internal structure and relationship of the object parts.
- ➢ An alternative to exploding an object into its component parts is a cutaway view, which removes part of the visible surfaces to show internal structure

Three-Dimensional and Stereoscopic Viewing

- Three-dimensional views can be obtained by reflecting a raster image from a vibrating, flexible mirror.
- The vibrations of the mirror are synchronized with the display of the scene on the cathode ray tube (CRT).
- As the mirror vibrates, the focal length varies so that each point in the scene is reflected to a spatial position corresponding to its depth.
- Stereoscopic devices present two views of a scene: one for the left eye and the other for the right eye.
- The viewing positions correspond to the eye positions of the viewer. These two views are typically displayed on alternate refresh cycles of a raster monitor

The Three-Dimensional Viewing Pipeline

- ✓ First of all, we need to choose a viewing position corresponding to where we would place a camera.
- ✓ We choose the viewing position according to whether we want to display a front, back, side, top, or bottom view of the scene.
- ✓ We could also pick a position in the middle of a group of objects or even inside a single object, such as a building or a molecule.
- ✓ Then we must decide on the camera orientation.
- ✓ Finally, when we snap the shutter, the scene is cropped to the size of a selected clipping window, which corresponds to the aperture or lens type of a camera, and light from the visible surfaces is projected onto the camera film.
- ✓ Some of the viewing operations for a three-dimensional scene are the same as, or similar to, those used in the two-dimensional viewing pipeline.
- ✓ A two-dimensional viewport is used to position a projected view of the three dimensional scene on the output device, and a two-dimensional clipping window is used to select a view that is to be mapped to the viewport.
- ✓ Clipping windows, viewports, and display windows are usually specified as rectangles with their edges parallel to the coordinate axes.
- ✓ The viewing position, view plane, clipping window, and clipping planes are all specified within the viewing-coordinate reference frame.

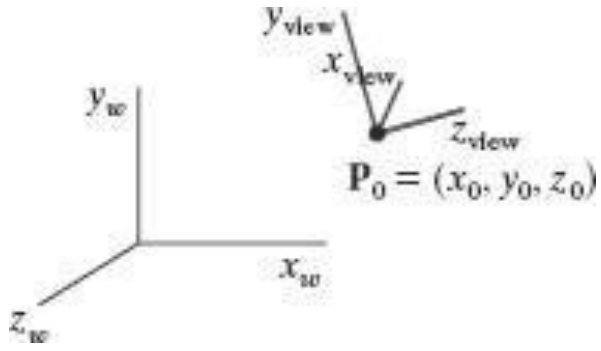


- ✓ Figure above shows the general processing steps for creating and transforming a three-dimensional scene to device coordinates.
- ✓ Once the scene has been modeled in world coordinates, a viewing-coordinate system is selected and the description of the scene is converted to viewing coordinates

- ✓ A two-dimensional clipping window, corresponding to a selected camera lens, is defined on the projection plane, and a three-dimensional clipping region is established.
- ✓ This clipping region is called the view volume.
- ✓ 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.
- ✓ We will assume that the viewport is to be specified in device coordinates and that normalized coordinates are transferred to viewport coordinates, following the clipping operations.
- ✓ The final step is to map viewport coordinates to device coordinates within a selected display window

Three-Dimensional Viewing-Coordinate Parameters

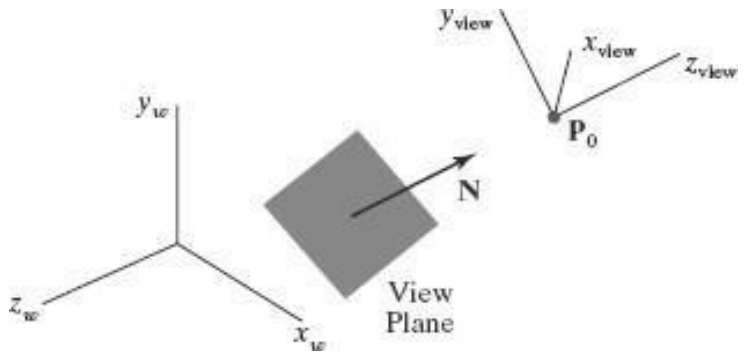
- ➢ Select a world-coordinate position $P_0 = (x_0, y_0, z_0)$ for the viewing origin, which is called the view point or viewing position and we specify a view-up vector V, which defines the y_{view} direction.
- ➢ Figure below illustrates the positioning of a three-dimensional viewing-coordinate frame within a world system.



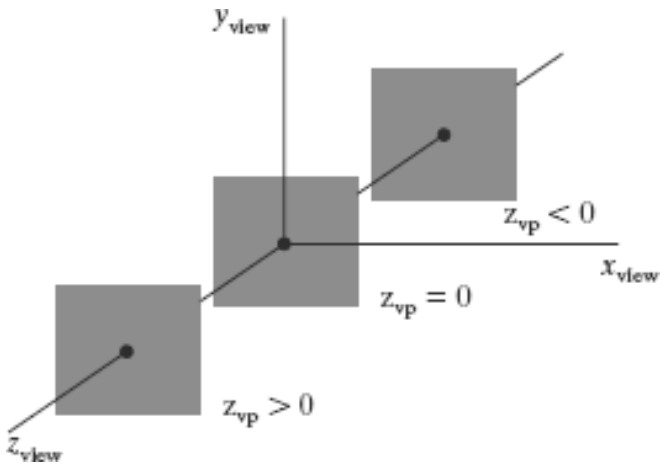
A right-handed viewing-coordinate system, with axes x $_{view}$, y $_{view}$, and z $_{view}$, relative to a right-handed world-coordinate frame.

The View-Plane Normal Vector

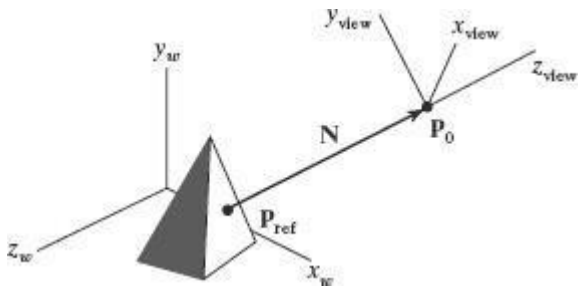
- ✓ Because the viewing direction is usually along the zview axis, the view plane, also called the projection plane, is normally assumed to be perpendicular to this axis.
- ✓ Thus, the orientation of the view plane, as well as the direction for the positive zview axis, can be definedwith a view-plane normal vector N,



- ✓ An additional scalar parameter is used to set the position of the view plane at some coordinate value z_{vp} along the z_{view} axis,



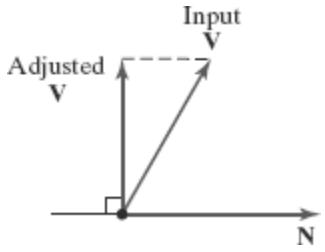
- ✓ This parameter value is usually specified as a distance from the viewing origin along the direction of viewing, which is often taken to be in the negative z_{view} direction.
- ✓ Vector N can be specified in various ways. In some graphics systems, the direction for N is defined to be along the line from the world-coordinate origin to a selected point position.
- ✓ Other systems take N to be in the direction from a reference point P_{ref} to the viewing origin P_0,



Specifying the view-plane normal vector N as the direction from a selected reference point P_{ref} to the viewing-coordinate origin P_0.

The View-Up Vector

- Once we have chosen a view-plane normal vector N, we can set the direction for the view-up vector V.
- This vector is used to establish the positive direction for the y_{view} axis.
- Usually, V is defined by selecting a position relative to the world-coordinate origin, so that the direction for the view-up vector is from the world origin to this selected position



- Because the view-plane normal vector N defines the direction for the z_{view} axis, vector V should be perpendicular to N.
- But, in general, it can be difficult to determine a direction for V that is precisely perpendicular to N.
- Therefore, viewing routines typically adjust the user-defined orientation of vector V,

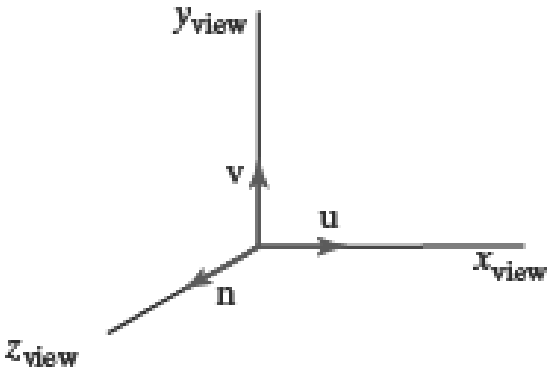
The uvn Viewing-Coordinate Reference Frame

- ✓ Left-handed viewing coordinates are sometimes used in graphics packages, with the viewing direction in the positive z_{view} direction.

- ✓ With a left-handed system, increasing z_{view} values are interpreted as being farther from the viewing position along the line of sight.
- ✓ But right-handed viewing systems are more common, because they have the same orientation as the world-reference frame.
- ✓ Because the view-plane normal N defines the direction for the z_{view} axis and the view-up vector V is used to obtain the direction for the y_{view} axis, we need only determine the direction for the x_{view} axis.
- ✓ Using the input values for N and V,we can compute a third vector, U, that is perpendicular to both N and V.
- ✓ Vector U then defines the direction for the positive x_{view} axis.
- ✓ We determine the correct direction for U by taking the vector cross product of V and N so as to form a right-handed viewing frame.
- ✓ The vector cross product of N and U also produces the adjusted value for V, perpendicular to both N and U, along the positive y_{view} axis.
- ✓ Following these procedures, we obtain the following set of unit axis vectors for a right-handed viewing coordinate system.

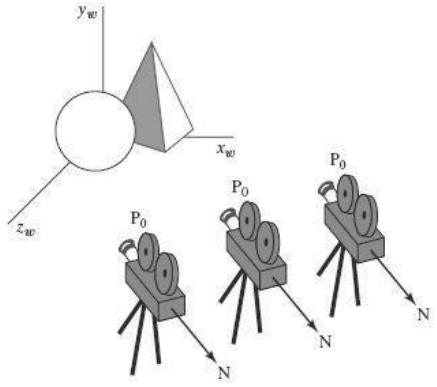
$$\mathbf{n} = \frac{\mathbf{N}}{|\mathbf{N}|} = (n_x, n_y, n_z)$$
$$\mathbf{u} = \frac{\mathbf{V} \times \mathbf{n}}{|\mathbf{V} \times \mathbf{n}|} = (u_x, u_y, u_z)$$
$$\mathbf{v} = \mathbf{n} \times \mathbf{u} = (v_x, v_y, v_z)$$

- ✓ The coordinate system formed with these unit vectors is often described as a uvn viewing-coordinate reference frame

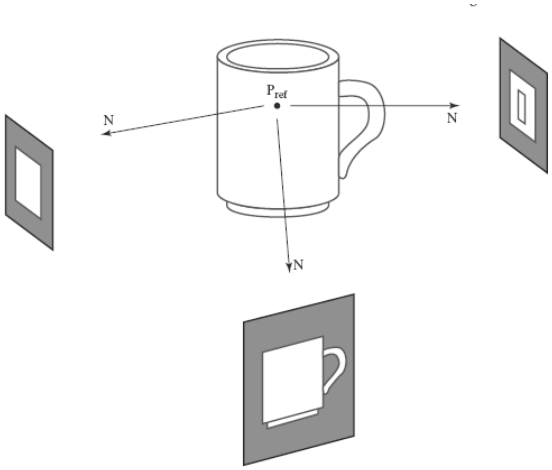


Generating Three-Dimensional Viewing Effects

- ✓ By varying the viewing parameters, we can obtain different views of objects in a scene.
- ✓ we could change the direction of N to display objects at positions around the viewing-coordinate origin.
- ✓ We could also vary N to create a composite display consisting of multiple views from a fixed camera position.
- ✓ In interactive applications, the normal vectorNis the viewing parameter that is most often changed. Of course,when we change the direction for N,we also have to change the other axis vectors to maintain a right-handed viewing-coordinate system.
- ✓ If we want to simulate an animation panning effect, as when a camera moves through a scene or follows an object that is moving through a scene, we can keep the direction for N fixed as we move the view point,



- ✓ Alternatively, different views of an object or group of objects can be generated using geometric transformations without changing the viewing parameters



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
    1. Translate the viewing-coordinate origin to the origin of the worldcoordinate system.
    2. Apply rotations to align the *x*view, *y*view, and *z*view axes with the world x_w, y_w, and z_w axes, respectively.
- ✓ The viewing-coordinate origin is at world position $P_0 = (x_0, y_0, z_0)$. Therefore, the matrix for translating the viewing origin to the world origin is

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

- ✓ For the rotation transformation, we can use the unit vectors u, v, and n to form the composite rotation matrix that superimposes the viewing axes onto the world frame. This transformation matrix is

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

  where the elements of matrix R are the components of the uvn axis vectors.
- ✓ The coordinate transformation matrix is then obtained as the product of the preceding translation and rotation matrices:

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

- ✓ Translation factors in this matrix are calculated as the vector dot product of each of the u, v, and n unit vectors with P_0, which represents a vector from the world origin to the viewing origin.

✓ These matrix elements are evaluated as

$$-\mathbf{u} \cdot \mathrm{P}_0 = -x_0 u_x - y_0 u_y - z_0 u_z$$
$$-\mathbf{v} \cdot \mathrm{P}_0 = -x_0 v_x - y_0 v_y - z_0 v_z$$
$$-\mathbf{n} \cdot \mathrm{P}_0 = -x_0 n_x - y_0 n_y - z_0 n_z$$

Projection Transformations

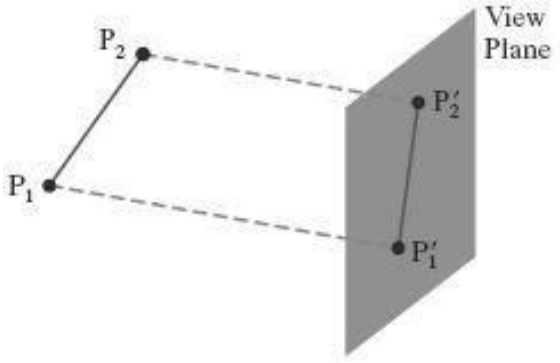
➔ Graphics packages generally support both parallel and perspective projections.

➔ In a parallel projection, coordinate positions are transferred to the view plane along parallel lines.

➔ A parallel projection preserves relative proportions of objects, and this is the method used in computeraided drafting and design to produce scale drawings of three-dimensional objects.

➔ All parallel lines in a scene are displayed as parallel when viewed with a parallel projection.

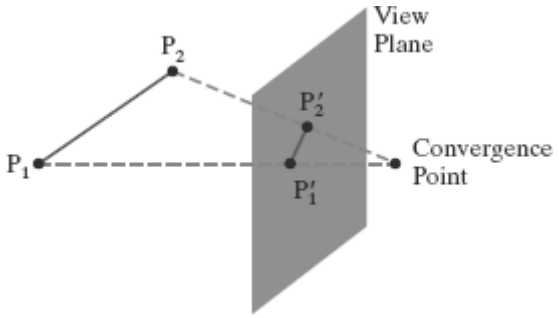
➔ There are two general methods for obtaining a parallel-projection view of an object: We can project along lines that are perpendicular to the view plane, or we can project at an oblique angle to the view plane



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

➔ 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.

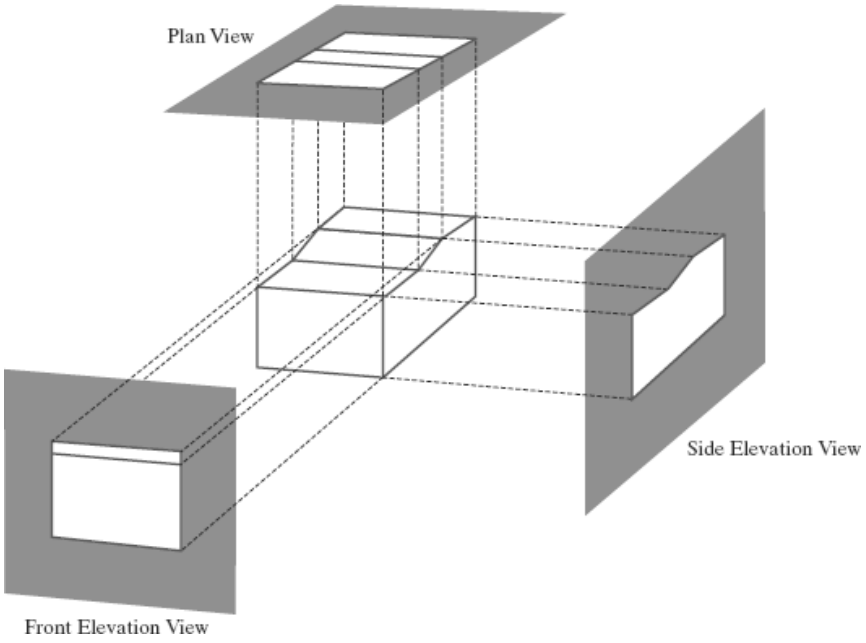


Orthogonal Projections

➔ 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 also termed as orthographic projection.

➔ This produces a parallel-projection transformation in which the projection lines are perpendicular to the view plane.

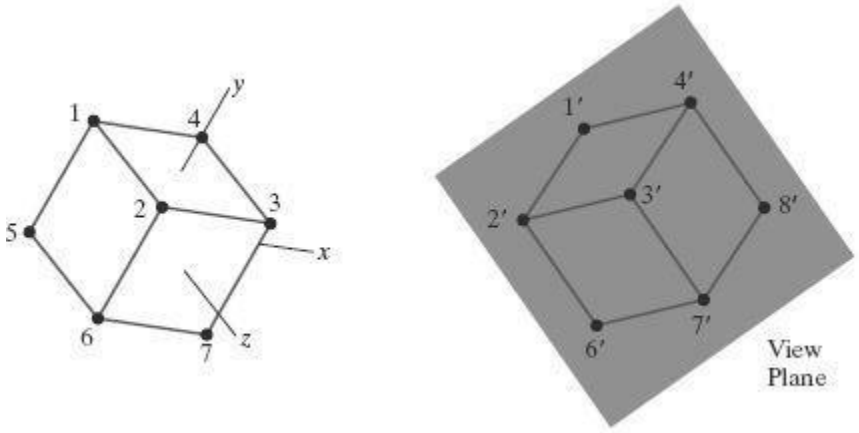
➔ Orthogonal projections are most often used to produce the front, side, and top views of an object



➔ Front, side, and rear orthogonal projections of an object are called *elevations;* and a top orthogonal projection is called a *plan view*

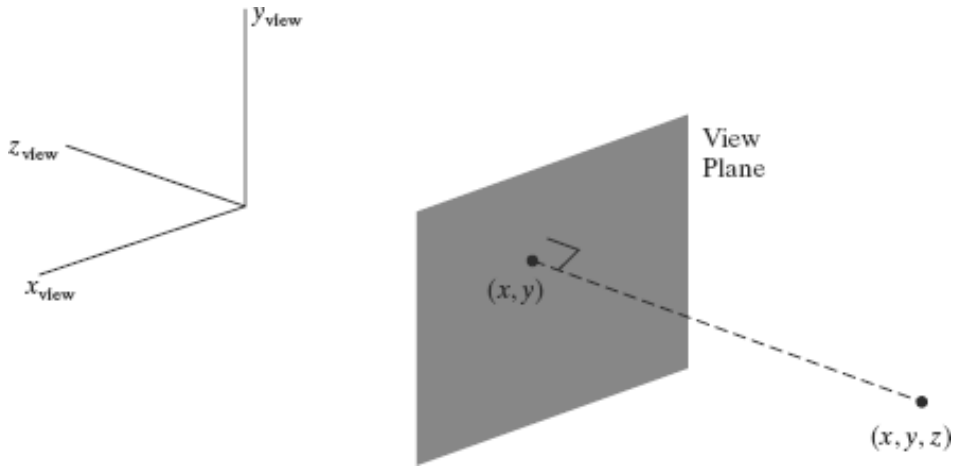
Axonometric and Isometric Orthogonal Projections

- 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,* at the same distance from the origin



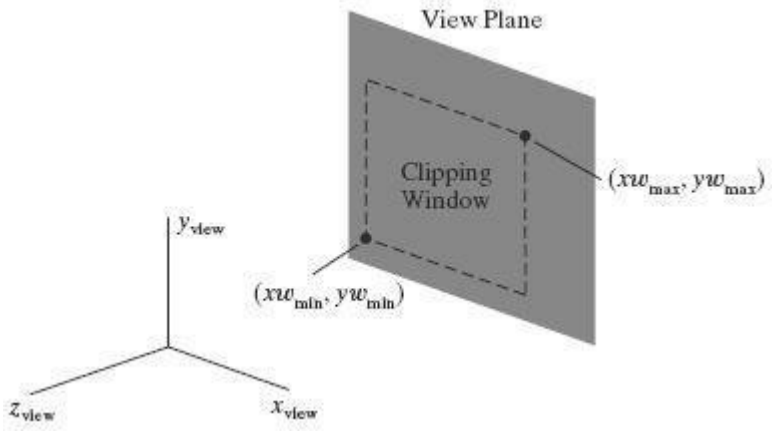
Orthogonal Projection Coordinates

- With the projection direction parallel to the z_{view} axis, the transformation equations for an orthogonal projection are trivial. For any position *(x, y, z)* in viewing coordinates, as in Figure below, the projection coordinates are $x_p = x$, $y_p = y$

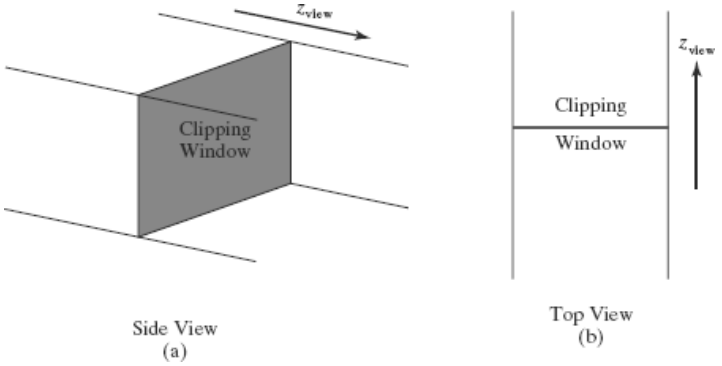


Clipping Window and Orthogonal-Projection View Volume

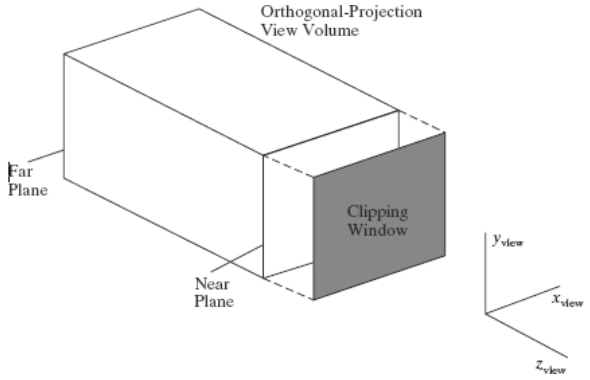
- In OpenGL, we set up a clipping window for three-dimensional viewing just as we did for two-dimensional viewing, by choosing two-dimensional coordinate positions for its lower-left and upper-right corners.
- For three-dimensional viewing, the clipping window is positioned on the view plane with its edges parallel to the x_{view} and y_{view} axes, as shown in Figure below . If we want to use some other shape or orientation for the clipping window, we must develop our own viewing procedures



- 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.
- Because projection lines are perpendicular to the view plane, these four boundaries are planes that are also perpendicular to the view plane and that pass through the edges of the clipping window to form an infinite clipping region, as in Figure below.

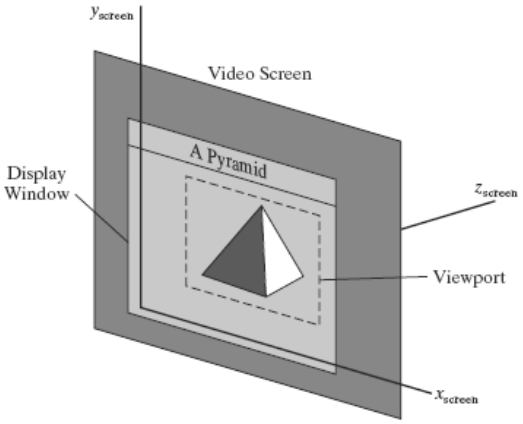


- These two planes are called the near-far clipping planes, or the front-back clipping planes.
- The near and far planes allow us to exclude objects that are in front of or behind the part of the scene that we want to display.
- When the near and far planes are specified, we obtain a finite orthogonal view volume that is a *rectangular parallelepiped*, as shown in Figure below along with one possible placement for the view plane

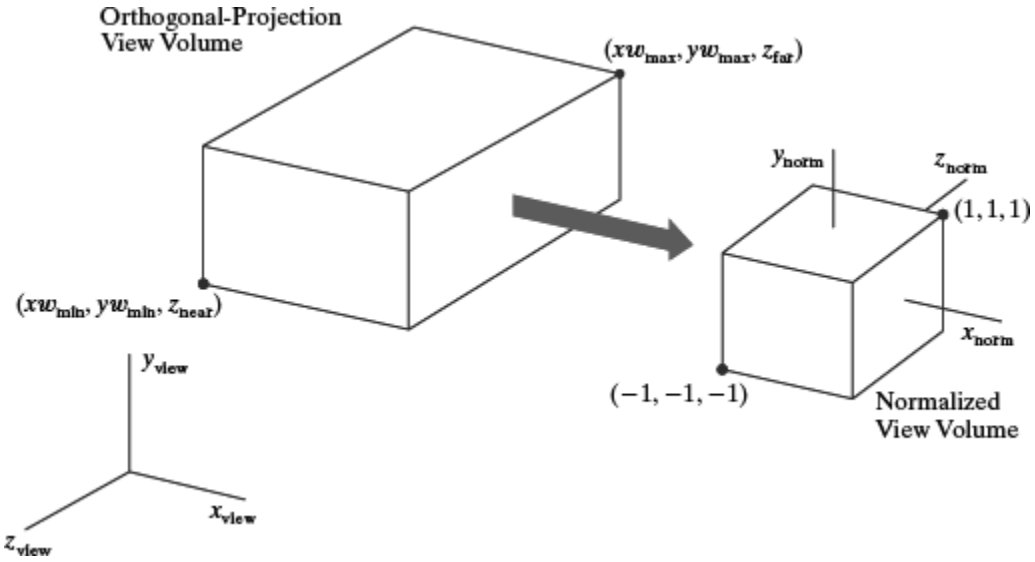


Normalization Transformation for an Orthogonal Projection

- Once we have established the limits for the view volume, coordinate descriptions inside this rectangular parallelepiped are the projection coordinates, and they can be mapped into a normalized view volume without any further projection processing.
- Some graphics packages use a unit cube for this normalized view volume, with each of the *x*, *y*, and *z* coordinates normalized in the range from 0 to 1.
- Another normalization-transformation approach is to use a symmetric cube, with coordinates in the range from −1 to 1



- We can convert projection coordinates into positions within a left-handed normalized-coordinate reference frame, and these coordinate positions will then be transferred to lefthanded screen coordinates by the viewport transformation.
- To illustrate the normalization transformation, we assume that the orthogonal-projection view volume is to be mapped into the symmetric normalization cube within a left-handed reference frame.
- Also, *z*-coordinate positions for the near and far planes are denoted as *z*near and *z*far, respectively. Figure below illustrates this normalization transformation



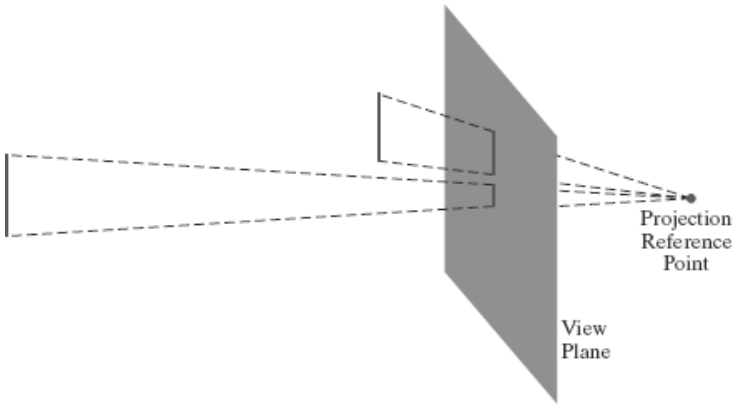
- The normalization transformation for the orthogonal view volume is

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

Perspective Projections

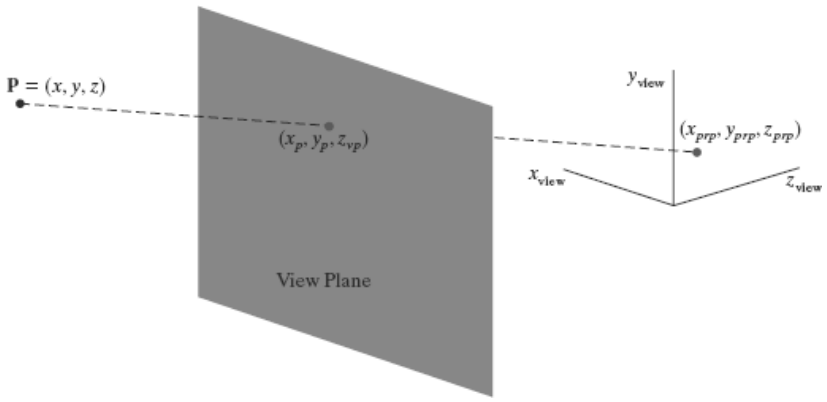
✓ We can approximate this geometric-optics effect by projecting objects to the view plane along converging paths to a position called the projection reference point (or center of projection).

- ✓ 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



<u>Perspective-Projection Transformation Coordinates</u>

- ✓ Figure below shows the projection path of a spatial position (x, y, z) to a general projection reference point at $(x_{prp}, y_{prp}, z_{prp})$.



- ✓ The projection line intersects the view plane at the coordinate position (x_p, y_p, z_{vp}), where z_{vp} is some selected position for the view plane on the z_{view} axis.
- ✓ We can write equations describing coordinate positions along this perspective-projection line in parametric form as

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

- ✓ On the view plane, $z' = z_{vp}$ and we can solve the z' equation for parameter u at this position along the 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

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

<u>Perspective-Projection Equations: Special Cases</u>

<u>Case 1:</u>

➔ To simplify the perspective calculations, the projection reference point could be limited to positions along the zview axis, then

$x_{prp} = y_{prp} = 0$:

$$x_p = x\left(\frac{z_{prp} - z_{vp}}{z_{prp} - z}\right), \qquad y_p = y\left(\frac{z_{prp} - z_{vp}}{z_{prp} - z}\right)$$

<u>Case 2:</u>

➔ Sometimes the projection reference point is fixed at the coordinate origin, and

$(x_{prp}, y_{prp}, z_{prp}) = (0, 0, 0)$:

$$x_p = x\left(\frac{z_{vp}}{z}\right), \qquad y_p = y\left(\frac{z_{vp}}{z}\right)$$

<u>Case 3:</u>

➔ If the view plane is the *uv* plane and there are no restrictions on the placement of the projection reference point, then we have

$z_{vp} = 0$:

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

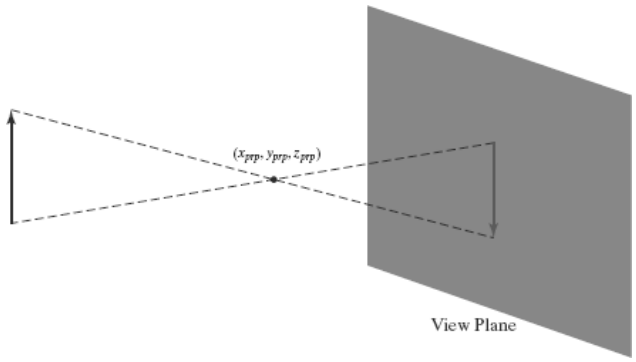
Case 4:

➔ With the *uv* plane as the view plane and the projection reference point on the z_{view} axis, the perspective equations are

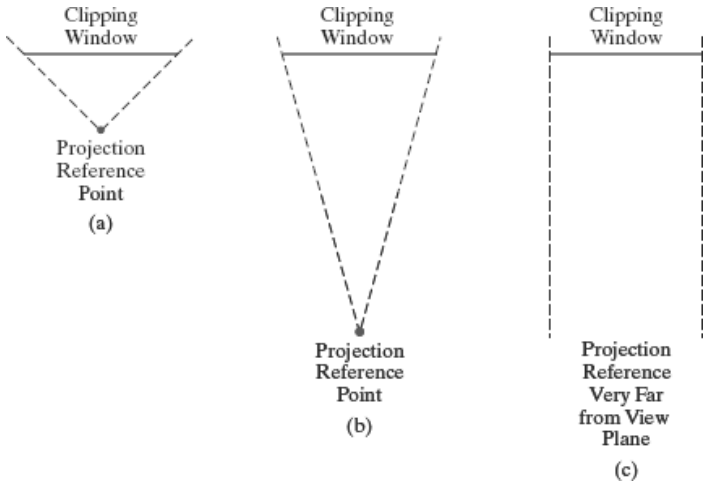
$x_{prp} = y_{prp} = z_{vp} = 0$:

$$x_p = x\left(\frac{z_{prp}}{z_{prp} - z}\right), \qquad y_p = y\left(\frac{z_{prp}}{z_{prp} - z}\right)$$

- ✓ 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 (refer below figure)



- ✓ Perspective effects also depend on the distance between the projection reference point and the view plane, as illustrated in Figure below.

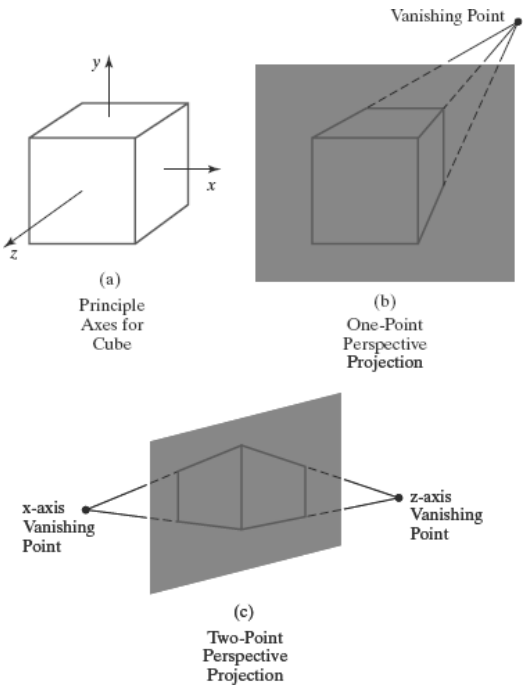


- ✓ If the projection reference point is close to the view plane, perspective effects are emphasized; that is, closer objects will appearmuchlarger thanmore distant objects of the same size.
- ✓ Similarly, as the projection reference point moves farther from the view plane, the difference in the size of near and far objects decreases

Vanishing Points for Perspective Projections

- ➢ 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

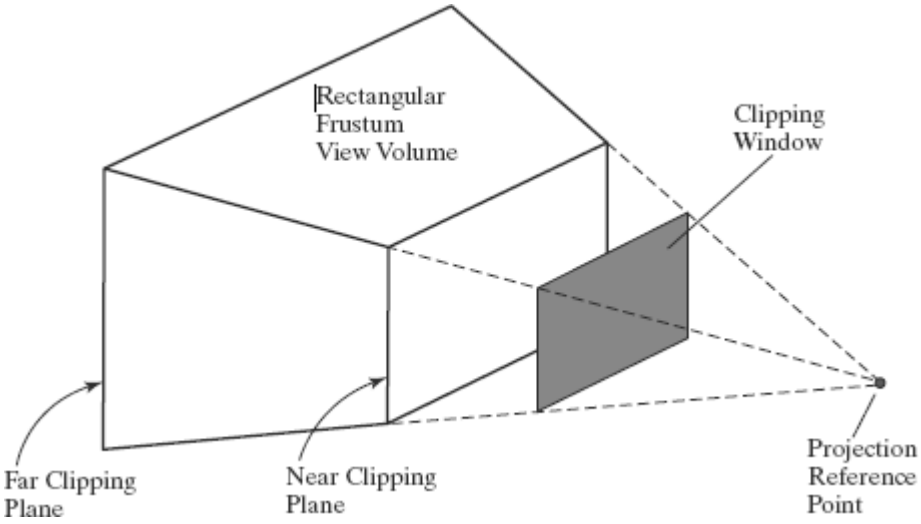
Principal vanishing points for perspective-projection views of a cube. When the cube in (a) is projected to a view plane that intersects only the z axis, a single vanishing point in the z direction (b) is generated. When the cube is projected to a view plane that intersects both the z and x axes, two vanishing points (c) are produced.



Perspective-Projection View Volume

- ➢ A perspective-projection view volume is often referred to as a pyramid of vision because it approximates the *cone of vision* of our eyes or a camera.

- The displayed view of a scene includes only those objects within the pyramid, just as we cannot see objects beyond our peripheral vision, which are outside the cone of vision.
- By adding near and far clipping planes that are perpendicular to the zview axis (and parallel to the view plane), we chop off parts of the infinite, perspectiveprojection view volume to form a truncated pyramid, or frustum, view volume



- But with a perspective projection, we could also use the near clipping plane to take out large objects close to the view plane that could project into unrecognizable shapes within the clipping window.
- Similarly, the far clipping plane could be used to cut out objects far from the projection reference point that might project to small blots on the view plane.

Perspective-Projection Transformation Matrix

- ✓ We can use a three-dimensional, homogeneous-coordinate representation to express the perspective-projection equations in the form

$$x_p = \frac{x_h}{h}, \qquad y_p = \frac{y_h}{h}$$

where the homogeneous parameter has the value

$$h = z_{prp} - z$$

$$x_h = x(z_{prp} - z_{vp}) + x_{prp}(z_{vp} - z)$$

$$y_h = y(z_{prp} - z_{vp}) + y_{prp}(z_{vp} - z)$$

- ✓ The perspective-projection transformation of a viewing-coordinate position is then accomplished in two steps.
- ✓ First, we calculate the homogeneous coordinates using the perspective-transformation matrix:

$$\mathbf{P}_h = \mathbf{M}_{\text{pers}} \cdot \mathbf{P}$$

Where,

P_h is the column-matrix representation of the homogeneous point (x_h, y_h, z_h, h) and

P is the column-matrix representation of the coordinate position $(x, y, z, 1)$.

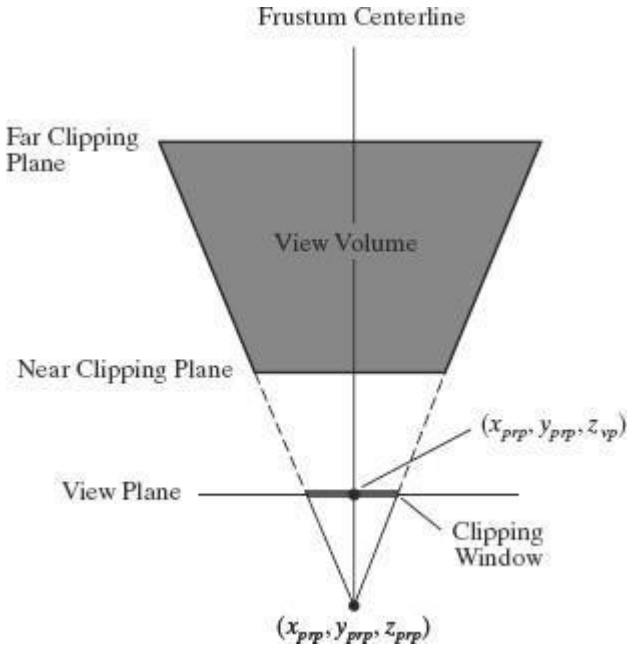
- ✓ Second, after other processes have been applied, such as the normalization transformation and clipping routines, homogeneous coordinates are divided by parameter *h* to obtain the true transformation-coordinate positions.
- ✓ The following matrix gives one possible way to formulate a perspective-projection matrix.

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

- ✓ Parameters s_z and t_z are the scaling and translation factors for normalizing the projected values of *z*-coordinates.
- ✓ Specific values for s_z and t_z depend on the normalization range we select.

Symmetric Perspective-Projection Frustum

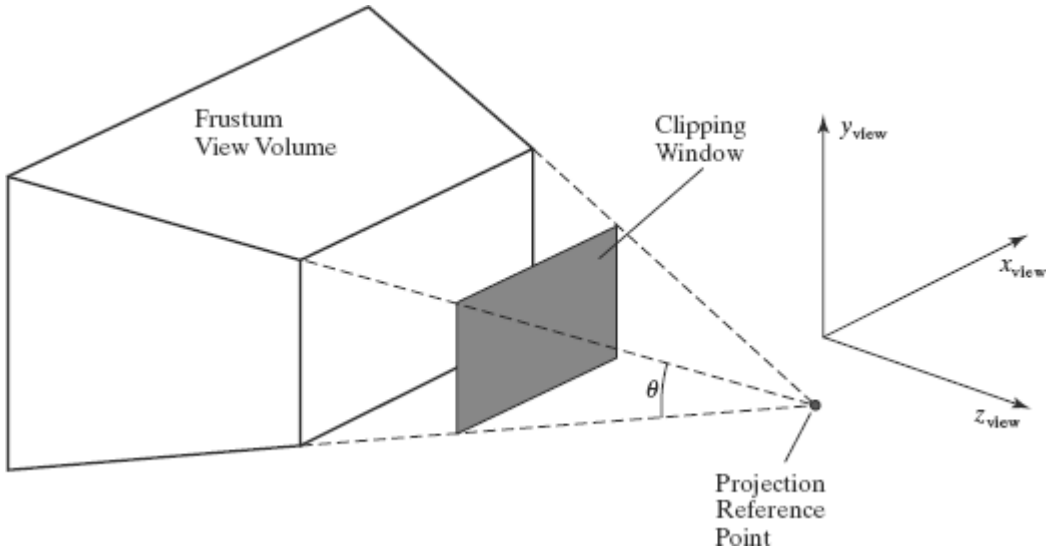
- ✓ The line from the projection reference point through the center of the clipping window and on through the view volume is the centerline for a perspectiveprojection frustum.
- ✓ If this centerline is perpendicular to the view plane, we have a symmetric frustum (with respect to its centerline)



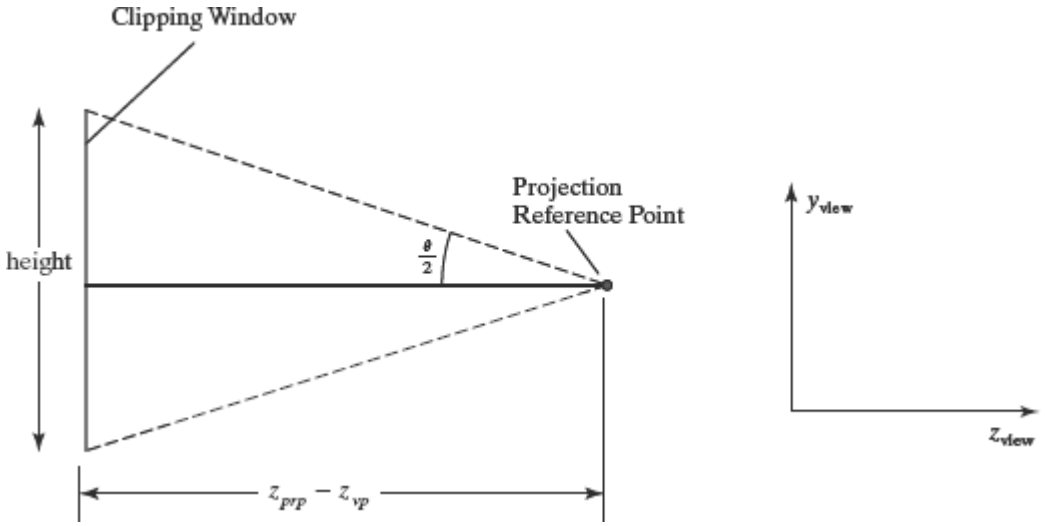
- ✓ Because the frustum centerline intersects the view plane at the coordinate location $(x_{prp}, y_{prp}, z_{vp})$, we can express the corner positions for the clipping window in terms of the window dimensions:

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

- ✓ Another way to specify a symmetric perspective projection is to use parameters that approximate the properties of a camera lens.
- ✓ A photograph is produced with a symmetric perspective projection of a scene onto the film plane.
- ✓ Reflected light rays from the objects in a scene are collected on the film plane from within the "cone of vision" of the camera.
- ✓ This cone of vision 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.



- ✓ For a given projection reference point and view-plane position, the field-of view angle determines the height of the clipping window from the right triangles in the diagram of Figure below, we see that



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

- ✓ so that the clipping-window height can be calculated as

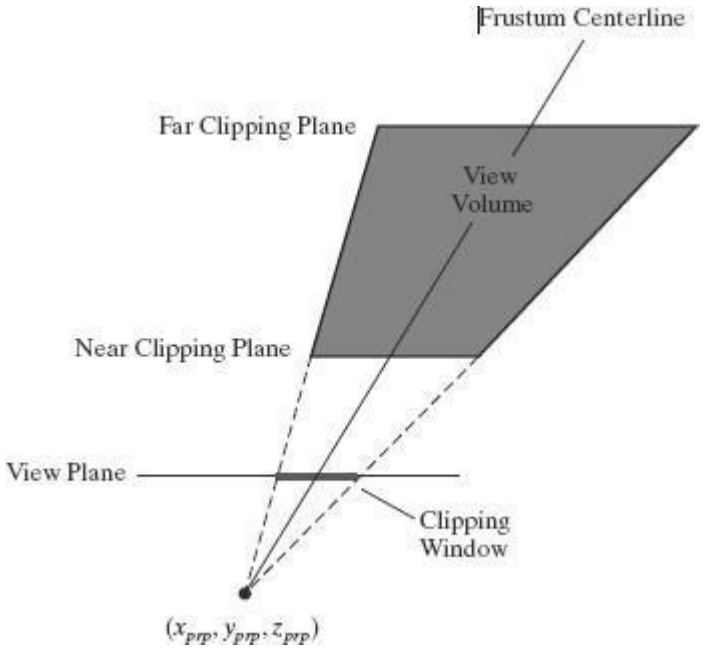
$$\text{height} = 2(z_{prp} - z_{vp}) \tan\left(\frac{\theta}{2}\right)$$

- ✓ Therefore, the diagonal elements with the value $z_{prp} - z_{vp}$ could be replaced by either of the following two expressions

$$z_{prp} - z_{vp} = \frac{\text{height}}{2} \cot\left(\frac{\theta}{2}\right)$$
$$= \frac{\text{width} \cdot \cot(\theta/2)}{2 \cdot \text{aspect}}$$

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



- ✓ In this case, we can first transform the view volume to a symmetric frustum and then to a normalized view volume.
- ✓ An oblique perspective-projection view volume can be converted to a sym metric 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.
- ✓ The computations for the shearing transformation, as well as for the perspective and normalization transformations, are greatly reduced if we take the projection reference point to be the viewing-coordinate origin.

- ✓ Taking the projection reference point as $(x_{prp}, y_{prp}, z_{prp}) = (0, 0, 0)$, we obtain the elements of the required shearing matrix as

$$\mathbf{M}_{z\,\text{shear}} = \begin{bmatrix} 1 & 0 & \text{sh}_{zx} & 0 \\ 0 & 1 & \text{sh}_{zy} & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- ✓ We need to choose values for the shearing parameters such that

$$\begin{bmatrix} 0 \\ 0 \\ z_{\text{near}} \\ 1 \end{bmatrix} = \mathbf{M}_{z\,\text{shear}} \cdot \begin{bmatrix} \frac{xw_{\min} + xw_{\max}}{2} \\ \frac{yw_{\min} + yw_{\max}}{2} \\ z_{\text{near}} \\ 1 \end{bmatrix}$$

- ✓ Therefore, the parameters for this shearing transformation are

$$\text{sh}_{zx} = -\frac{xw_{\min} + xw_{\max}}{2\, z_{\text{near}}}$$
$$\text{sh}_{zy} = -\frac{yw_{\min} + yw_{\max}}{2\, z_{\text{near}}}$$

- ✓ Similarly, with the projection reference point at the viewing-coordinate origin and with the near clipping plane as the view plane, the perspective-projection matrix is simplified to

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

- ✓ Concatenating the simplified perspective-projection matrix with the shear matrix we have

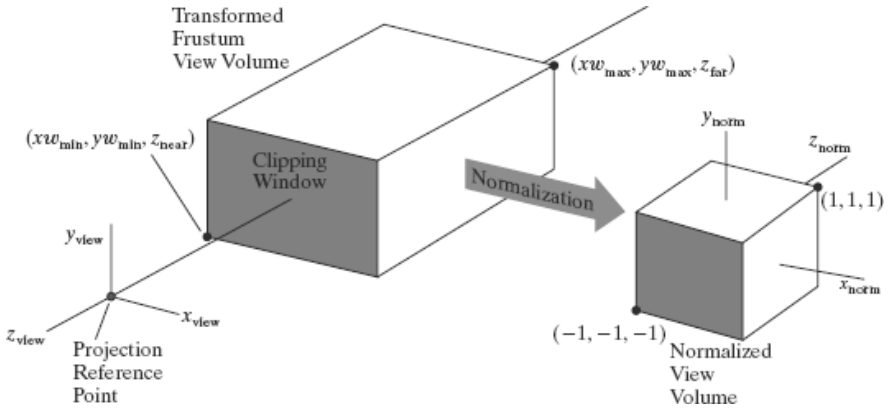
$$\mathbf{M}_{\text{obliquepers}} = \mathbf{M}_{\text{pers}} \cdot \mathbf{M}_{z\,\text{shear}}$$
$$= \begin{bmatrix} -z_{\text{near}} & 0 & \frac{xw_{\min} + xw_{\max}}{2} & 0 \\ 0 & -z_{\text{near}} & \frac{yw_{\min} + yw_{\max}}{2} & 0 \\ 0 & 0 & s_z & t_z \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

Normalized Perspective-Projection Transformation Coordinates

➔ When we divide the homogeneous coordinates by the homogeneous parameter *h*, we obtain the actual projection coordinates, which are orthogonal-projection coordinates

➔ The final step in the perspective transformation process is to map this parallelepiped to a *normalized view volume*.

➔ The transformed frustum view volume, which is a rectangular parallelepiped, is mapped to a symmetric normalized cube within a left-handed reference frame



➔ Because the centerline of the rectangular parallelepiped view volume is now the z_{view} axis, no translation is needed in the *x* and *y* normalization transformations: We require only the *x* and *y* scaling parameters relative to the coordinate origin.

➔ The scaling matrix for accomplishing the *xy* normalization is

$$\mathbf{M}_{xy\,\text{scale}} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

➔ Concatenating the *xy*-scalingmatrix produces the following normalization matrix for a perspective-projection transformation.

$$\mathbf{M}_{\text{normpers}} = \mathbf{M}_{xy\,\text{scale}} \cdot \mathbf{M}_{\text{obliquepers}}$$

$$= \begin{bmatrix} -z_{\text{near}} s_x & 0 & s_x \dfrac{xw_{\min} + xw_{\max}}{2} & 0 \\ 0 & -z_{\text{near}} s_y & s_y \dfrac{yw_{\min} + yw_{\max}}{2} & 0 \\ 0 & 0 & s_z & t_z \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

➔ From this transformation, we obtain the homogeneous coordinates:

$$\begin{bmatrix} x_h \\ y_h \\ z_h \\ h \end{bmatrix} = \mathbf{M}_{\text{normpers}} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

And the projection coordinates are

$$x_p = \frac{x_h}{h} = \frac{-z_{\text{near}} s_x x + s_x (xw_{\min} + xw_{\max})/2}{-z}$$
$$y_p = \frac{y_h}{h} = \frac{-z_{\text{near}} s_y y + s_y (yw_{\min} + yw_{\max})/2}{-z}$$
$$z_p = \frac{z_h}{h} = \frac{s_z z + t_z}{-z}$$

➔ To normalize this perspective transformation, we want the projection coordinates to be $(x_p, y_p, z_p) = (-1, -1, -1)$ when the input coordinates are $(x, y, z) = (x_{w\min}, y_{w\min}, z_{\text{near}})$, and we want the projection coordinates to be $(x_p, y_p, z_p) = (1, 1, 1)$ when the input coordinates are $(x, y, z) = (x_{w\max}, y_{w\max}, z_{\text{far}})$.

$$s_x = \frac{2}{xw_{\max} - xw_{\min}}, \qquad s_y = \frac{2}{yw_{\max} - yw_{\min}}$$
$$s_z = \frac{z_{\text{near}} + z_{\text{far}}}{z_{\text{near}} - z_{\text{far}}}, \qquad t_z = \frac{2\, z_{\text{near}}\, z_{\text{far}}}{z_{\text{near}} - z_{\text{far}}}$$

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

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

The Viewport Transformation and Three-Dimensional Screen Coordinates

- ✓ Once we have completed the transformation to normalized projection coordinates, clipping can be applied efficiently to the symmetric cube then the contents of the normalized view volume can be transferred to screen coordinates.
- ✓ Positions throughout the three-dimensional view volume also have a depth (*z* coordinate), and we need to retain this depth information for the visibility testing and surface-rendering algorithms
- ✓ If we include this *z* renormalization, the transformation from the normalized view volume to three dimensional screen coordinates is

$$\mathbf{M}_{\text{normviewvol,3D screen}} = \begin{bmatrix} \frac{xv_{\max} - xv_{\min}}{2} & 0 & 0 & \frac{xv_{\max} + xv_{\min}}{2} \\ 0 & \frac{yv_{\max} - yv_{\min}}{2} & 0 & \frac{yv_{\max} + yv_{\min}}{2} \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- ✓ In normalized coordinates, the $z_{\text{norm}} = -1$ face of the symmetric cube corresponds to the clipping-window area. And this face of the normalized cube is mapped to the rectangular viewport, which is now referenced at zscreen = 0.
- ✓ Thus, the lower-left corner of the viewport screen area is at position $(xv_{\min}, yv_{\min}, 0)$ and the upper-right corner is at position $(xv_{\max}, yv_{\max}, 0)$.

OpenGL Three-Dimensional Viewing Functions

OpenGL Viewing-Transformation Function

glMatrixMode (GL_MODELVIEW);

➔ a matrix is formed and concatenated with the current modelview matrix, We set the modelview mode with the statement above

gluLookAt (x0, y0, z0, xref, yref, zref, Vx, Vy, Vz);

➔ Viewing parameters are specified with the above GLU function.

- ➔ This function designates the origin of the viewing reference frame as the world-coordinate position $P_0 = (x_0, y_0, z_0)$, the reference position as $P_{ref} = (x_{ref}, y_{ref}, z_{ref})$, and the view-up vector as $V = (V_x, V_y, V_z)$.
- ➔ If we do not invoke the gluLookAt function, the default OpenGL viewing parameters are

$P_0 = (0, 0, 0)$

$P_{ref} = (0, 0, -1)$

$V = (0, 1, 0)$

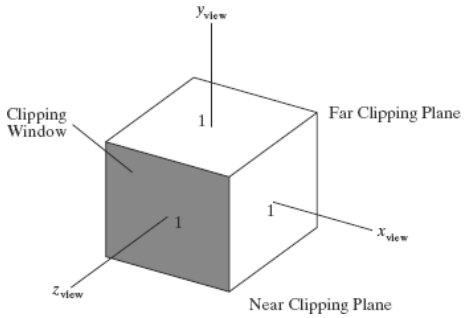
OpenGL Orthogonal-Projection Function

glMatrixMode (GL_PROJECTION);

- ➔ set up a projection-transformation matrix.
- ➔ Then, when we issue any transformation command, the resulting matrix will be concatenated with the current projection matrix.

glOrtho (xwmin, xwmax, ywmin, ywmax, dnear, dfar);

- ➔ Orthogonal-projection parameters are chosen with the function
- ➔ All parameter values in this function are to be assigned double-precision, floating point Numbers
- ➔ Function glOrtho generates a parallel projection that is perpendicular to the view plane
- ➔ Parameters d_{near} and d_{far} denote distances in the negative z_{view} direction from the viewing-coordinate origin
- ➔ We can assign any values (positive, negative, or zero) to these parameters, so long as $d_{near} < d_{far}$.
- ➔ Exa: glOrtho (-1.0, 1.0, -1.0, 1.0, -1.0, 1.0);



OpenGL General Perspective-Projection Function

glFrustum (xwmin, xwmax, ywmin, ywmax, dnear, dfar);

- ➔ specify a perspective projection that has either a symmetric frustum view volume or an oblique frustum view volume
- ➔ All parameters in this function are assigned double-precision, floating-point numbers.
- ➔ 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 z_{view} axis.

OpenGL Viewports and Display Windows

glViewport (xvmin, yvmin, vpWidth, vpHeight);

- ➔ A rectangular viewport is defined.
- ➔ 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.
- ➔ Display windows are created and managed with GLUT routines. The default viewport in OpenGL is the size and position of the current display window

OpenGL Three-Dimensional Viewing Program Example

```
#include <GL/glut.h>
GLint winWidth = 600, winHeight = 600; // Initial display-window size.
GLfloat x0 = 100.0, y0 = 50.0, z0 = 50.0; // Viewing-coordinate origin.
GLfloat xref = 50.0, yref = 50.0, zref = 0.0; // Look-at point.
GLfloat Vx = 0.0, Vy = 1.0, Vz = 0.0; // View-up vector.
/* Set coordinate limits for the clipping window: */
GLfloat xwMin = -40.0, ywMin = -60.0, xwMax = 40.0, ywMax = 60.0;
/* Set positions for near and far clipping planes: */
GLfloat dnear = 25.0, dfar = 125.0;
```

```
void init (void)
{
        glClearColor (1.0, 1.0, 1.0, 0.0);
        glMatrixMode (GL_MODELVIEW);
        gluLookAt (x0, y0, z0, xref, yref, zref, Vx, Vy, Vz);
        glMatrixMode (GL_PROJECTION);
        glFrustum (xwMin, xwMax, ywMin, ywMax, dnear, dfar);
}
void displayFcn (void)
{
        glClear (GL_COLOR_BUFFER_BIT);
        glColor3f (0.0, 1.0, 0.0); // Set fill color to green.
        glPolygonMode (GL_FRONT, GL_FILL);
        glPolygonMode (GL_BACK, GL_LINE); // Wire-frame back face.
        glBegin (GL_QUADS);
                glVertex3f (0.0, 0.0, 0.0);
                glVertex3f (100.0, 0.0, 0.0);
                glVertex3f (100.0, 100.0, 0.0);
                glVertex3f (0.0, 100.0, 0.0);
        glEnd ( );
        glFlush ( );
}
void reshapeFcn (GLint newWidth, GLint newHeight)
{
        glViewport (0, 0, newWidth, newHeight);
        winWidth = newWidth;
        winHeight = newHeight;
}
void main (int argc, char** argv)
{
        glutInit (&argc, argv);
```

```
    glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
    glutInitWindowPosition (50, 50);
    glutInitWindowSize (winWidth, winHeight);
    glutCreateWindow ("Perspective View of A Square");
    init ( );
    glutDisplayFunc (displayFcn);
    glutReshapeFunc (reshapeFcn);
    glutMainLoop ( );
}
```

Visible-Surface Detection Methods

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.
- **Object-space methods:** 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**.**
- **Image-space methods:** visibility is decided point by point at each pixel position on the projection plane**.**
- Although there are major differences in the basic approaches 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.
- Coherence methods are used to take advantage of regularities in a scene.

Back-Face Detection

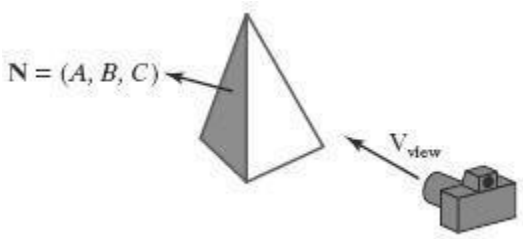
- ✓ A fast and simple object-space method for locating the back faces of a polyhedron is based on front-back tests. A point (x, y, z) is behind a polygon surface if

$$A_x + B_y + C_z + D < 0$$

where A, B,C, and $D$are the plane parameters for the polygon

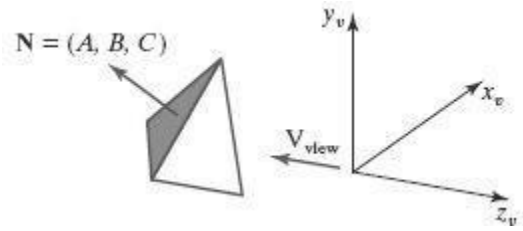
- ✓ We can simplify the back-face test by considering the direction of the normal vector N for a polygon surface. If V_{view} is a vector in the viewing direction from our camera position, as shown in Figure below, then a polygon is a back face if

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

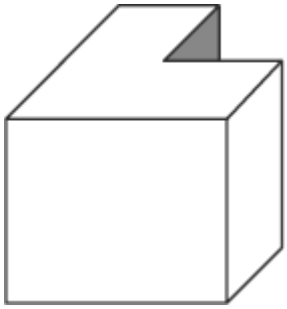


- ✓ In a right-handed viewing system with the viewing direction along the negative *zv* axis (Figure below), a polygon is a back face if the *z* component, *C*, of its normal vector N satisfies $C < 0$.
- ✓ Also, we cannot see any face whose normal has *z* component $C = 0$, because our viewing direction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

  $$\boldsymbol{C <= 0}$$



- ✓ Similar methods can be used in packages that employ a left-handed viewing system. In these packages, plane parameters *A*, *B*, *C*, and *D* can be calculated from polygon vertex coordinates specified in a clockwise direction.
- ✓ Inequality 1 then remains a valid test for points behind the polygon.
- ✓ By examining parameter *C* for the different plane surfaces describing an object, we can immediately identify all the back faces.
- ✓ For other objects, such as the concave polyhedron in Figure below, more tests must be carried out to determine whether there are additional faces that are totally or partially obscured by other faces



- ✓ In general, back-face removal can be expected to eliminate about half of the polygon surfaces in a scene from further visibility tests.

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.
- ❖ 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-detectionapproach is also frequently alluded to as the *z-buffer method,* because object depth is usually measured along the *z* axis of a viewing system

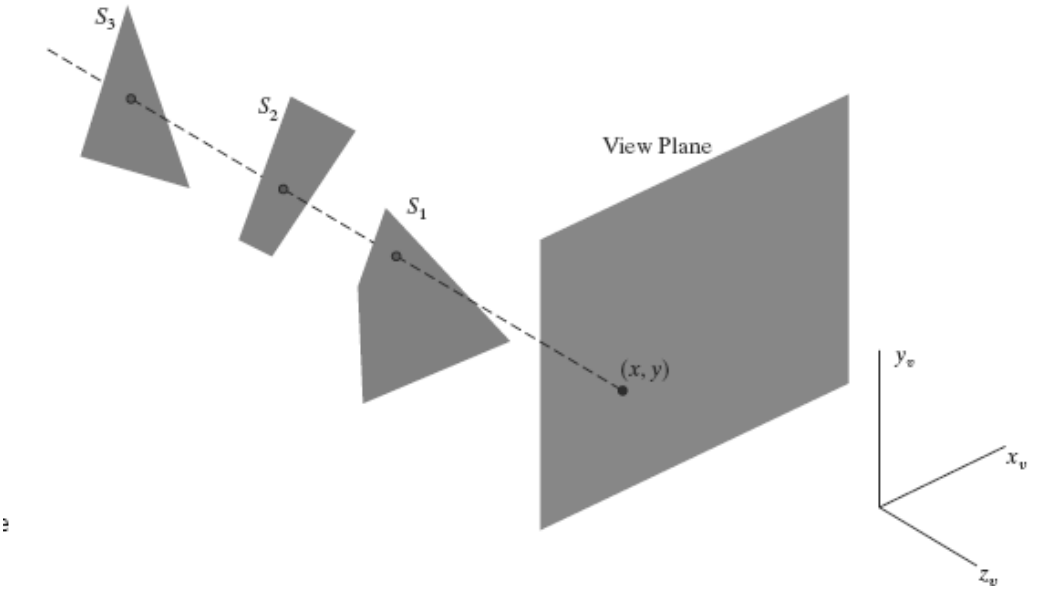


Figure above shows three surfaces at varying distances along the orthographic projection line from position *(x, y)* on a view plane.

- These surfaces can be processed in any order.
- 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
- As implied by the name of this method, 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.

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.

- Given the depth values for the vertex positions of any polygon in a scene, we can calculate the depth at any other point on the plane containing the polygon.
- At surface position *(x, y)*, the depth is calculated from the plane equation as

$$z = \frac{-Ax - By - D}{C}$$

- If the depth of position *(x, y)* has been determined to be *z*, then the depth *z*' of the next position *(x* + 1, *y)* along the scan line is obtained as

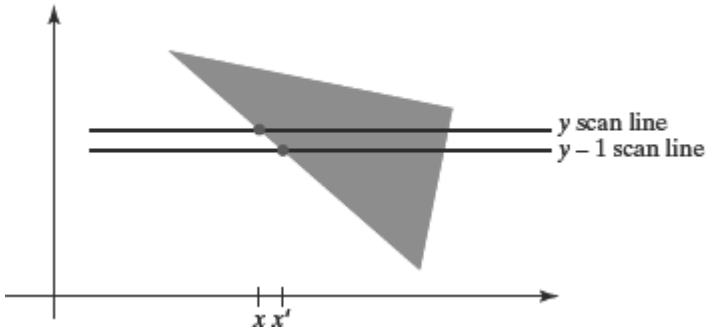
$$z' = \frac{-A(x+1) - By - D}{C}$$

$$z' = z - \frac{A}{C}$$

- The ratio $-A/C$ is constant for each surface, so succeeding depth values across a scan line are obtained from preceding values with a single addition.
- We can implement the depth-buffer algorithm by starting at a top vertex of the polygon.
- Then, we could recursively calculate the *x*-coordinate values down a left edge of the polygon.
- The *x* value for the beginning position on each scan line can be calculated from the beginning (edge) *x* value of the previous scan line as

$$x' = x - \frac{1}{m}$$

where *m* is the slope of the edge (Figure below).



- Depth values down this edge are obtained recursively as

$$z' = z + \frac{A/m + B}{C}$$

- If we are processing down a vertical edge, the slope is infinite and the recursive calculations reduce to

$$z' = z + \frac{B}{C}$$

- One slight complication with this approach is that while pixel positions are at integer *(x, y)* coordinates, the actual point of intersection of a scan line with the edge of a polygon may not be.
- As a result, it may be necessary to adjust the intersection point by rounding its fractional part up or down, as is done in scan-line polygon fill algorithms.
- An alternative approach is to use a midpoint method or Bresenham-type algorithm for determining the starting *x* values along edges for each scan line.

- The method can be applied to curved surfaces by determining depth and color values at each surface projection point.
- In addition, the basic depth-buffer algorithm often performs needless calculations.
- Objects are processed in an arbitrary order, so that a color can be computed for a surface point that is later replaced by a closer surface.

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, GL_FRONT, GL_FRONT_AND_BACK

  ➔ By default, parameter mode in the glCullFace function has the value GL_BACK

  ➔ The culling routine is turned off with

  glDisable (GL_CULL_FACE);

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

  glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);

- Depth buffer values can then be initialized with

  glClear (GL_DEPTH_BUFFER_BIT);

  - the preceding initialization sets all depth-buffer values to the maximum value 1.0 by default

- 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.

- As an option, we can adjust these normalization values with

  glDepthRange (nearNormDepth, farNormDepth);

  - By default, nearNormDepth = 0.0 and farNormDepth = 1.0.
  - But with the glDepthRange function, we can set these two parameters to any values within the range from 0.0 to 1.0, including nearNormDepth > farNormDepth

- Another option available in OpenGL is 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.
  - The default value for parameter testCondition 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

```
glEnable (GL_DEPTH_TEST);
glPolygonMode (GL_FRONT_AND_BACK, GL_LINE);
glColor3f (1.0, 1.0, 1.0);
/* Invoke the object-description routine. */
glPolygonMode (GL_FRONT_AND_BACK, GL_FILL);
glEnable (GL_POLYGON_OFFSET_FILL);
glPolygonOffset (1.0, 1.0);
glColor3f (0.0, 0.0, 0.0);
/* Invoke the object-description routine again. */
glDisable (GL_POLYGON_OFFSET_FILL);
```

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 to object colors using $d_{min} = 0.0$ and $d_{max} = 1.0$. But we can set different values for *d*min and *d*max with the following function calls:

glFogf (GL_FOG_START, minDepth);

glFogf (GL_FOG_END, maxDepth);

➔ In these two functions, parameters minDepth and maxDepth are assigned floating-point values, although integer values can be used if we change the function suffix to i.

➔ We can use the glFog function to set an atmosphere color that is to be combined with the color of an object after applying the linear depthcueing function

Difference Between perspective projection and parallel projection

Perspective projection	Parallel projection
The center of projection is at a finite distance from the viewing plane	Center of projection at infinity results with a parallel projection
Explicitly specify: center of projection	Direction of projection is specified
Size of the object is inversely proportional to the distance of the object from the center of projection	No change in the size of object
Produces realistic views but does not preserve relative proportion of objects	A parallel projection oreserves relative proportion of objects, but does not give us a reaistic representation of the appearance of object.
Not useful for recording exact shape and measurements of the object	Used for exact measurement
Parallel lines do not in general project as parallel	Parallel lines do remain parallel