RV UNIVERSITY
Go, change the world
an initiative of RV EDUCATIONAL INSTITUTIONS

Module 3

3D Geometric Transformations,3D Viewing

Geometric transformations in three-dimensional space

RV UNIVERSITY
Go, change the world
an initiative of RV EDUCATIONAL INSTITUTIONS

- Extend from two-dimensional transformation by including considerations for the z coordinates
- Translation and scaling are similar to two-dimension, include the three Cartesian coordinates
- Rotation method is less straight forward
- Representation
  - Four-element column vector for homogenous coordinates
  - Geometric transformation described 4by4 matrix

Geometric transformations in three-dimensional space (2)

Three-dimensional translation

A point P (x,y,z) in three-dimensional space is translated to new location with the translation distance T (tx, ty, tz)

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

In matrix format

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \qquad P' = T \cdot P$$

Figure 9-1 Moving a coordinate position with translation vector $T = (t_x, t_y, t_z)$.

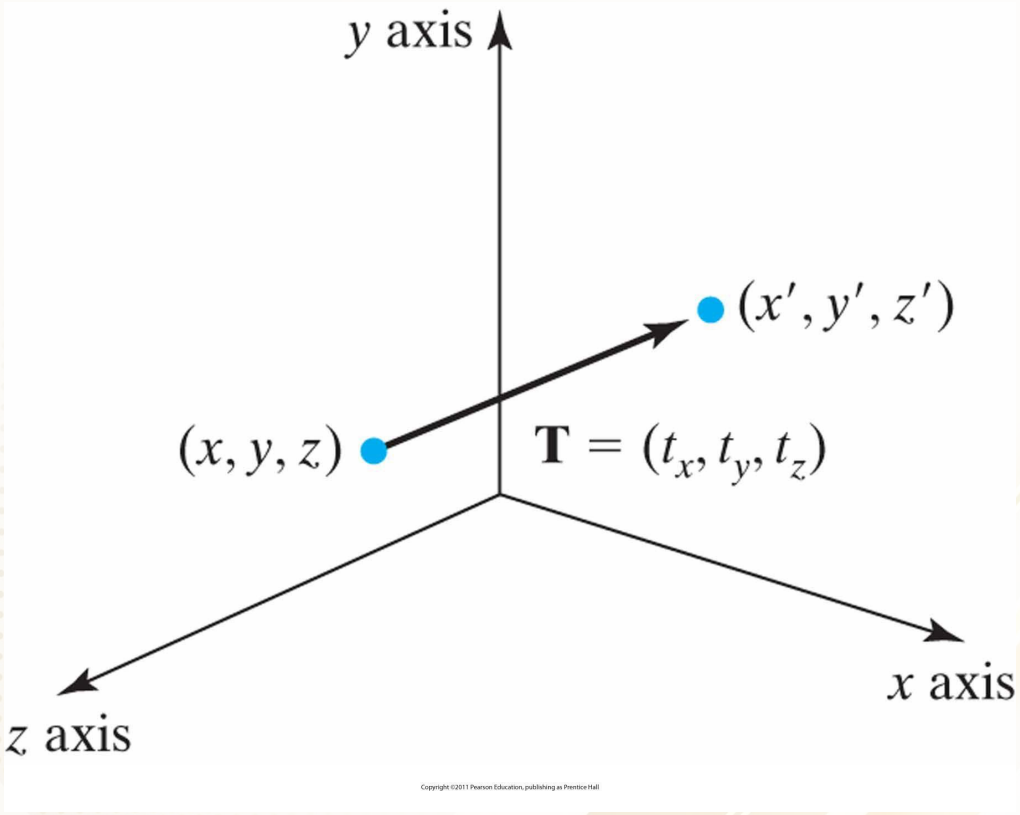
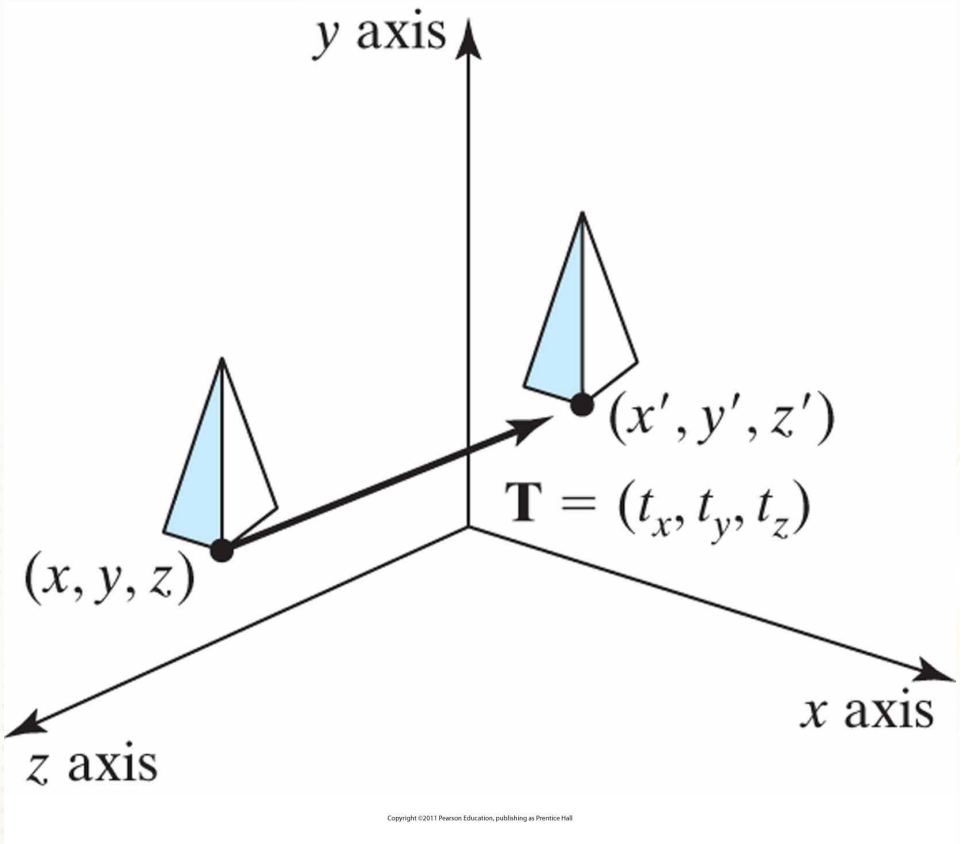


Figure 9-2 Shifting the position of a three-dimensional object using translation vector **T**.



Geometric transformations in three-dimensional space (3)

Three-dimensional scaling

Relative to the coordinate origin, just include the parameter for z coordinate scaling in the transformation matrix

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

Relative to a fixed point $(x_f, y_f z_f)$

Perform a translate-scaling-translate composite transformation

$$t(x_f, y_f, z_f) \cdot S(s_x, s_y, s_z) \cdot T(-x_f, -y_f, -z_f)$$

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

Figure 9-17 Doubling the size of an object with transformation 9-41 also moves the object farther from the origin.

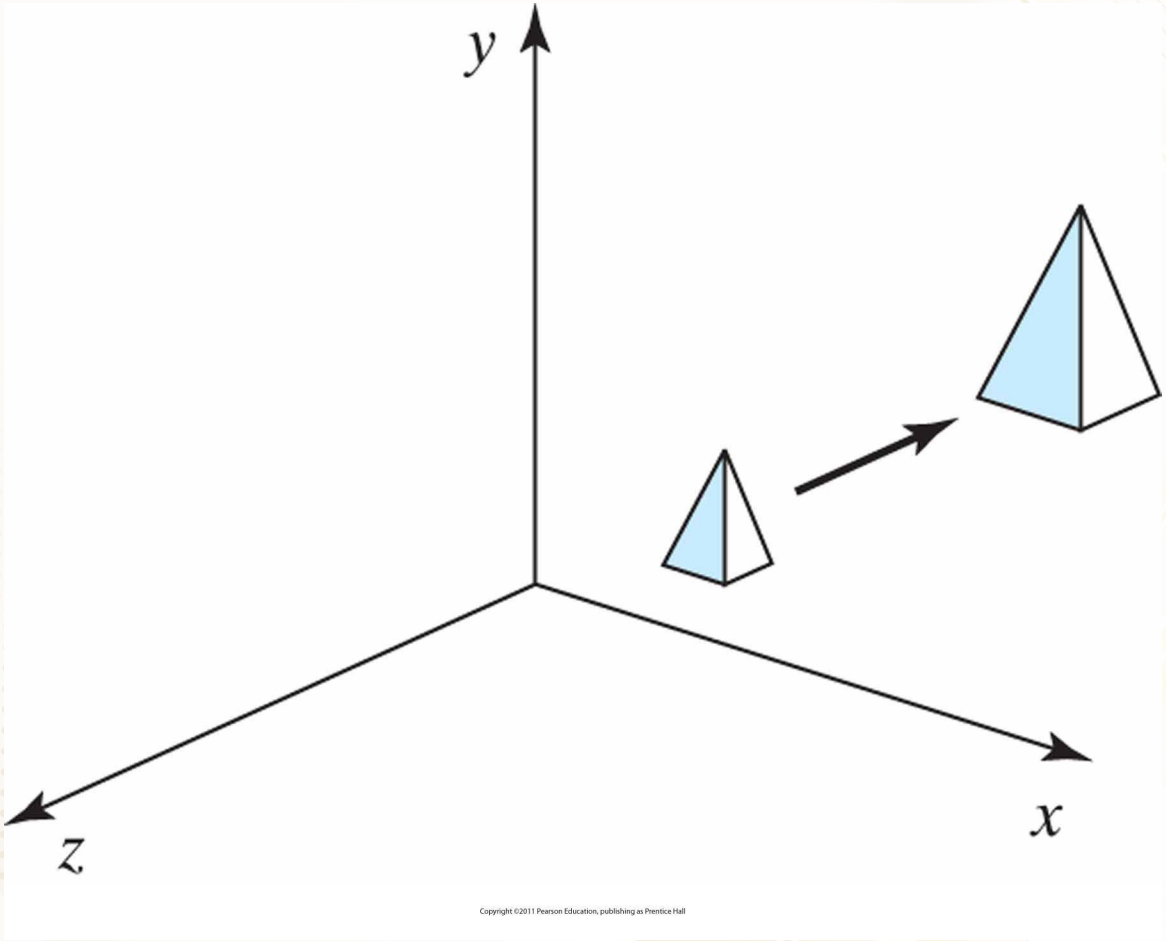
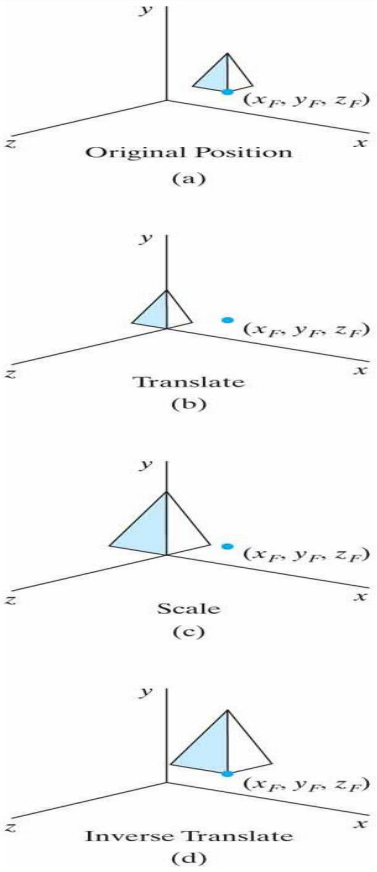


Figure 9-18 A sequence of transformations for scaling an object relative to a selected fixed point, using Equation 9-41.



Geometric transformations in three-dimensional space (4)

- Three-dimensional rotation definition
  - Assume looking in the negative direction along the axis
  - Positive angle rotation produce counterclockwise rotations about a coordinate axis

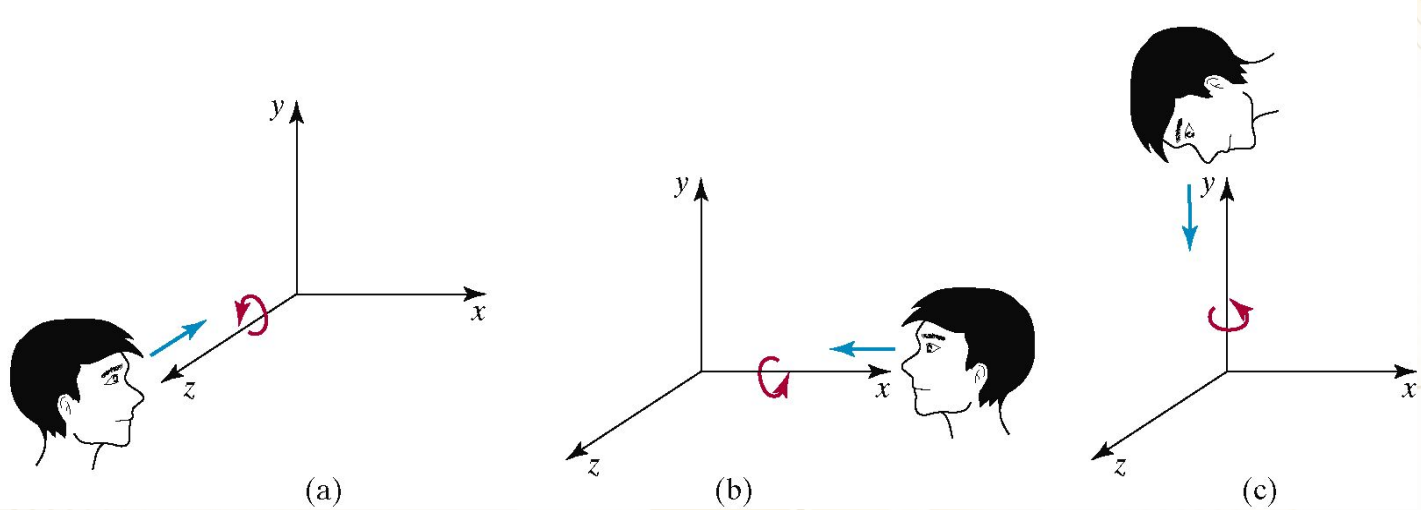
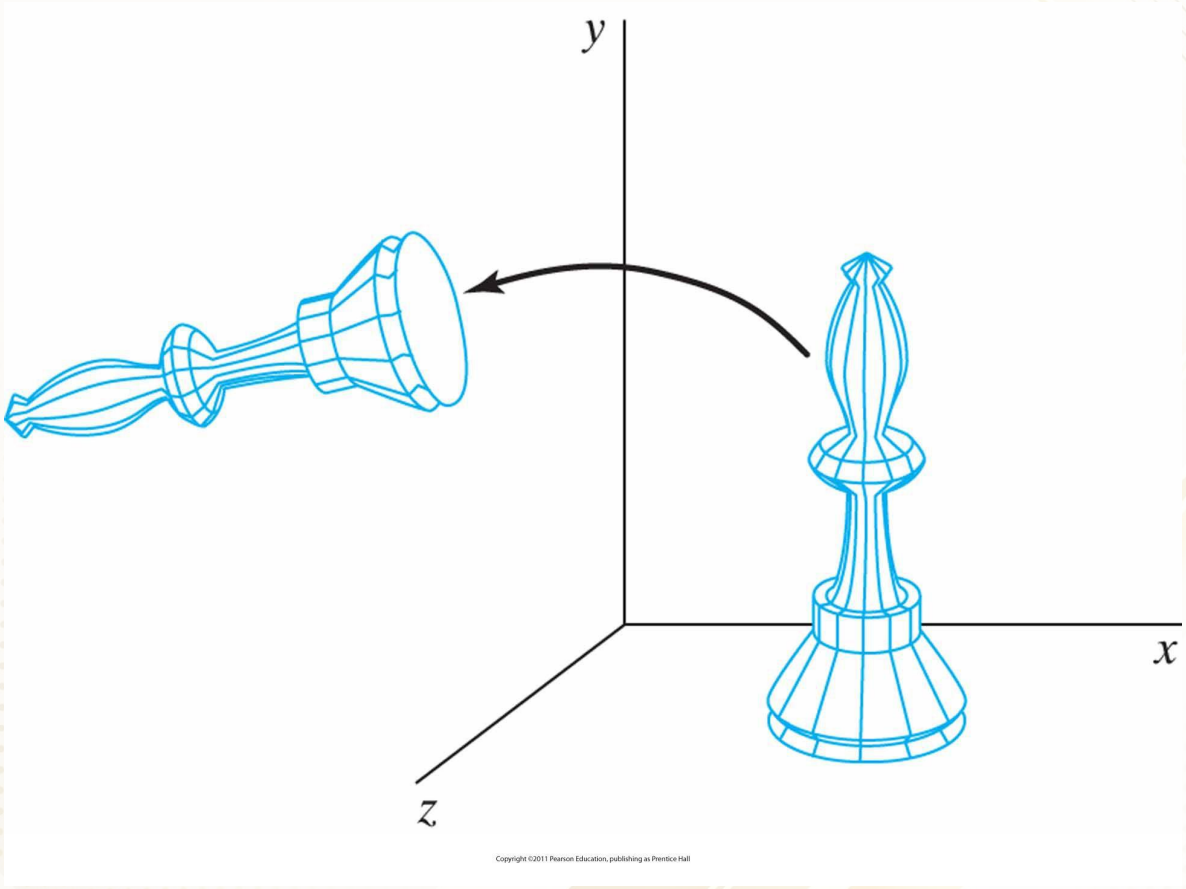


Figure 9-4 Rotation of an object about the *z* axis.



Geometric transformations in three-dimensional space (5)

Three-dimensional coordinate-axis rotation

Z-axis rotation equations

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

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

R_z

Transformation equations for rotation about the other two coordinate axes can be obtained by a cyclic permutation

x □ y □ z □ x

X-axis rotation equations

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

$$R_x = R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Figure 9-5 Cyclic permutation of the Cartesian-coordinate axes to produce the three sets of coordinate-axis rotation equations.

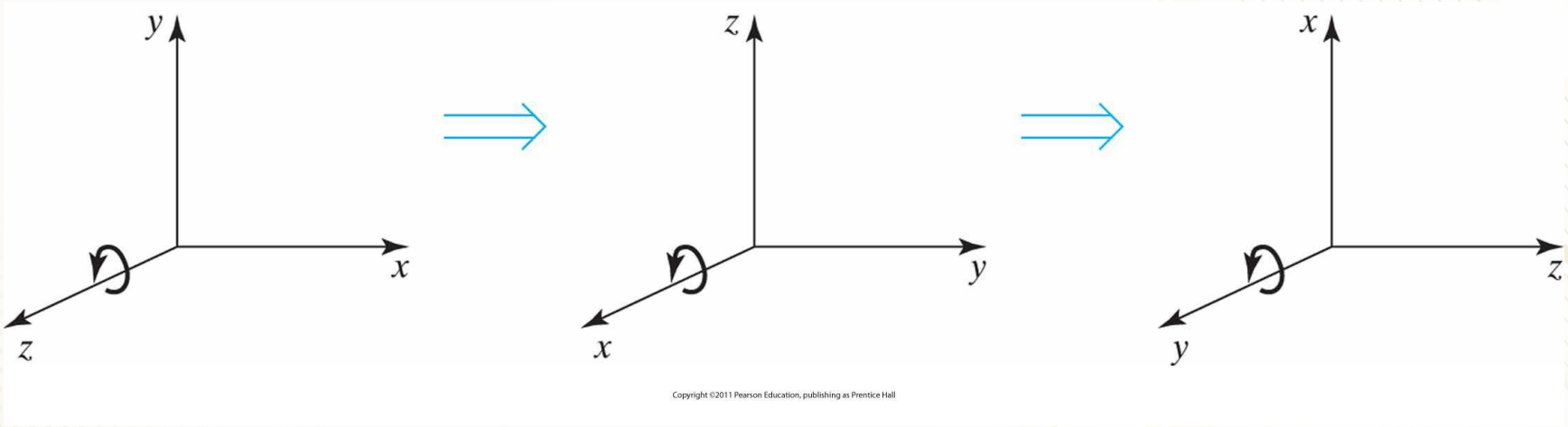


Figure 9-6 Rotation of an object about the *x* axis.

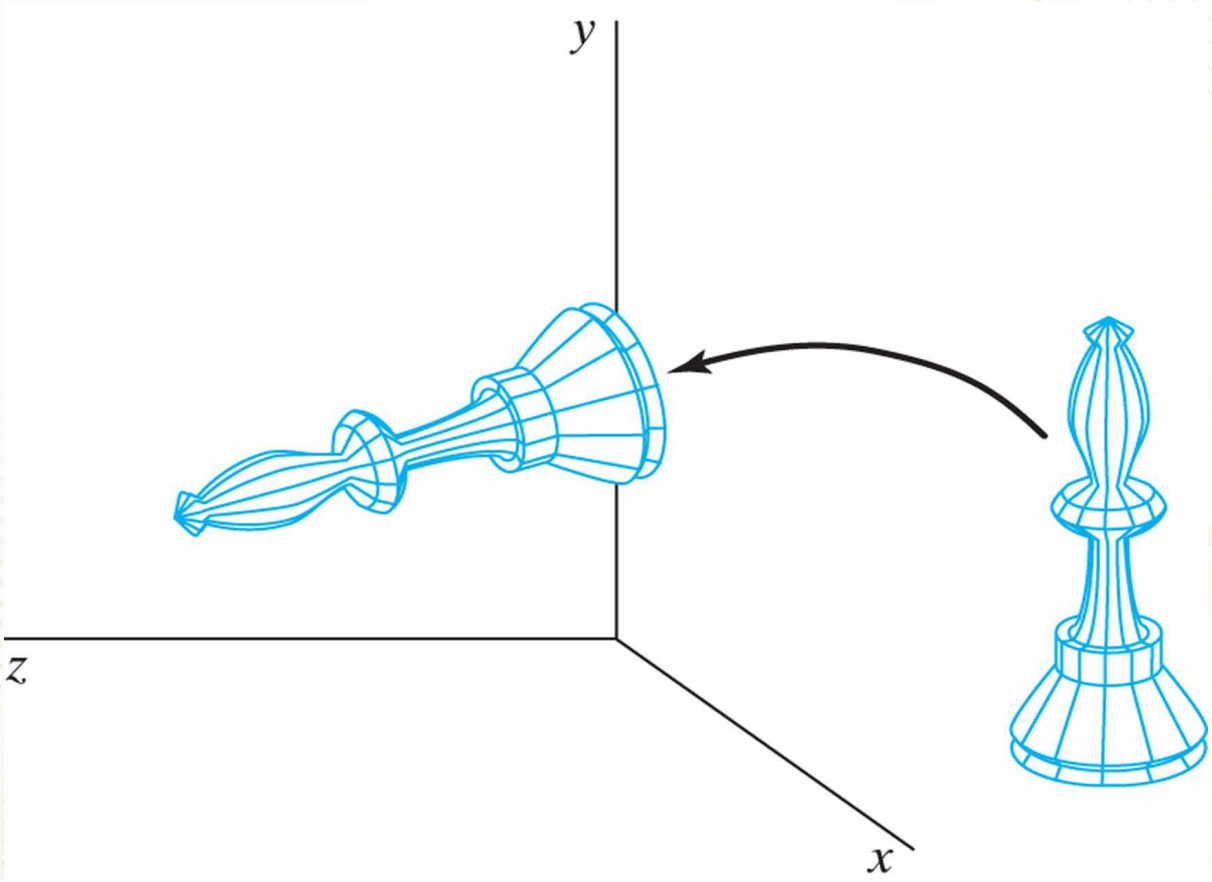
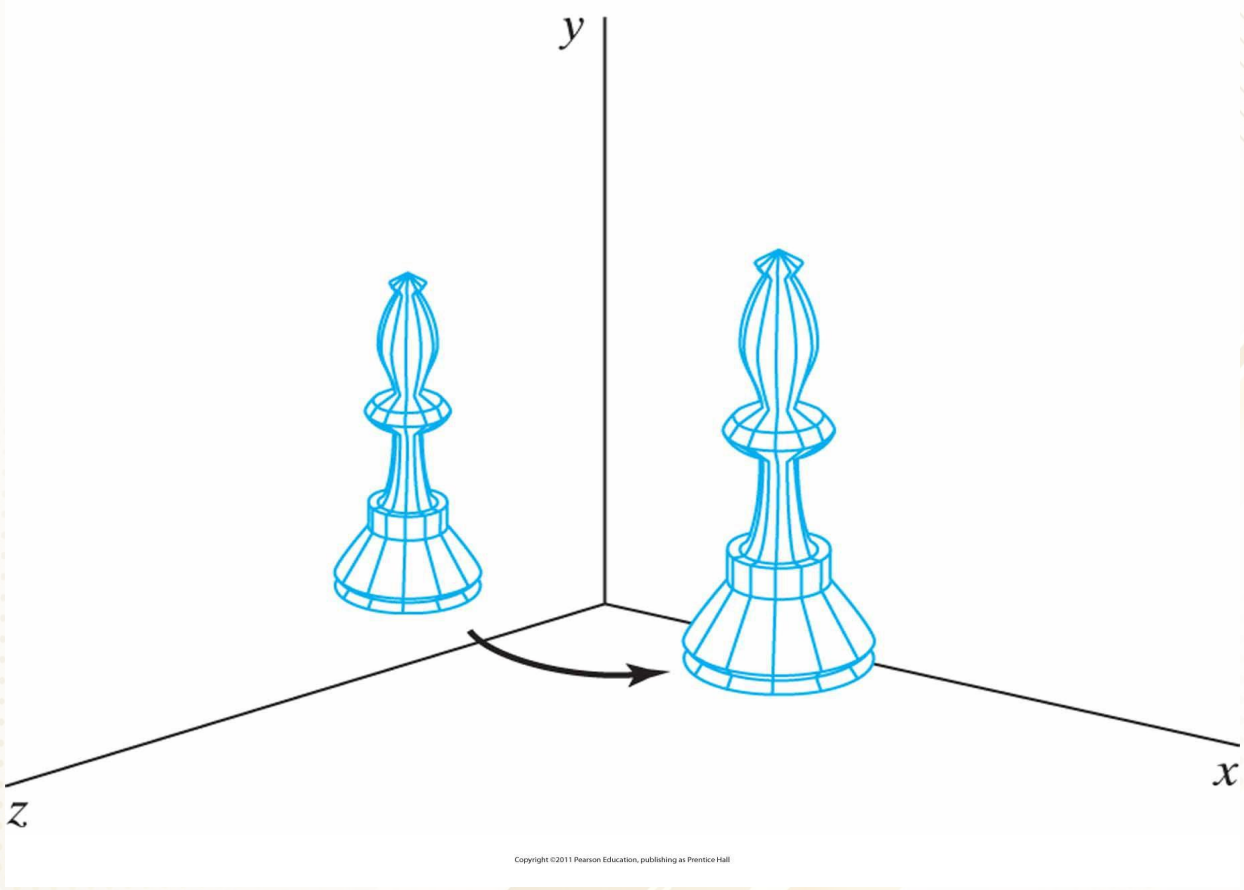


Figure 9-7 Rotation of an object about the *y* axis.



Geometric transformations in three-dimensional space (6)

Three-dimensional coordinate-axis rotation

Y-axis rotation equations

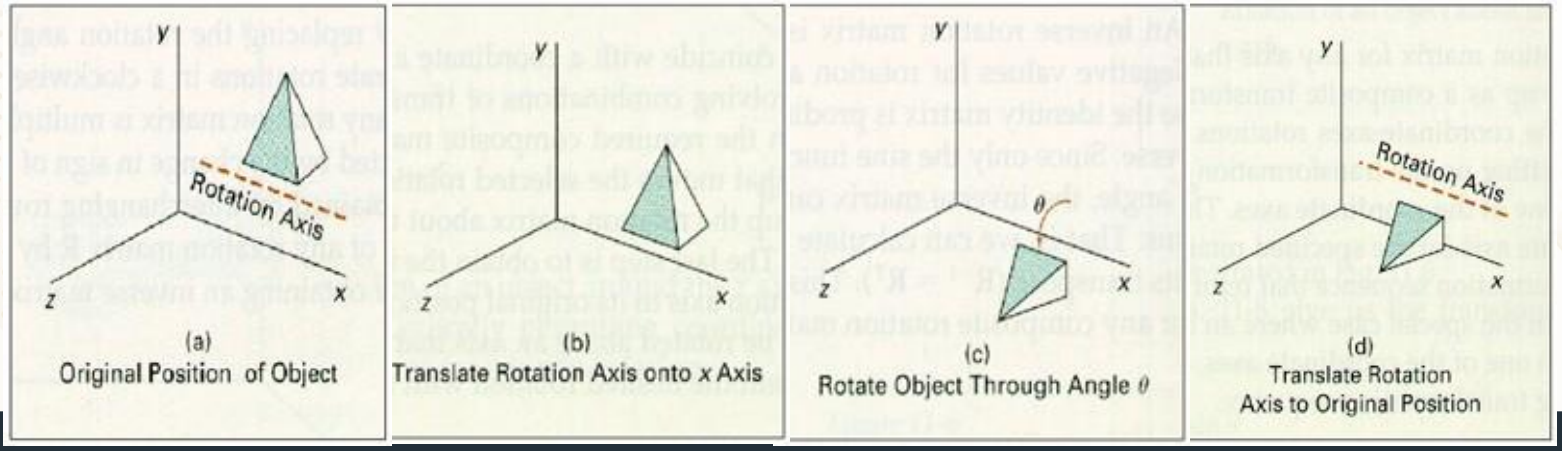
$$z' = z\cos\theta - x\sin\theta$$
$$x' = z\sin\theta + x\cos\theta$$
$$y' = y$$

$$R_y = R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Rotation about an Axis that is Parallel to One of the Coordinate Axes
  - **Translate** the object so that the rotation axis coincides with the parallel coordinate axis
  - Perform the specified **rotation** about that axis
  - **Translate** the object so that the rotation axis is moved back to its original position

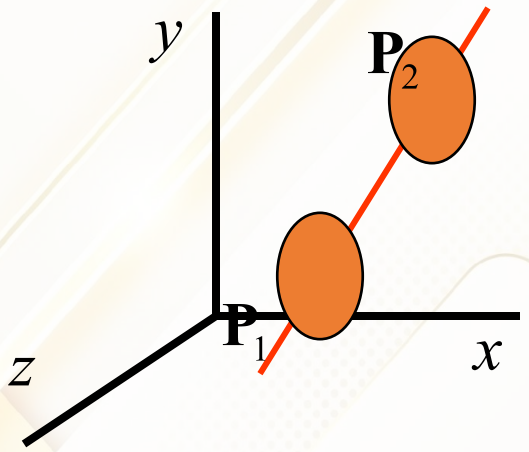
$$P' = T^{-1} \cdot R_x(\theta) \cdot T \cdot P$$

$$R(\theta) = T^{-1} \cdot R_x(\theta) \cdot T$$



3D Rotation around arbitrary axis

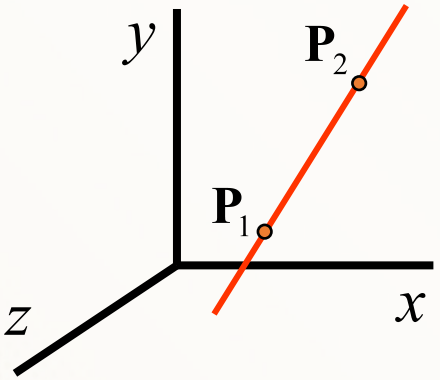
Rotation around axis through two points $\mathbf{P}_1$ and $\mathbf{P}_2$.



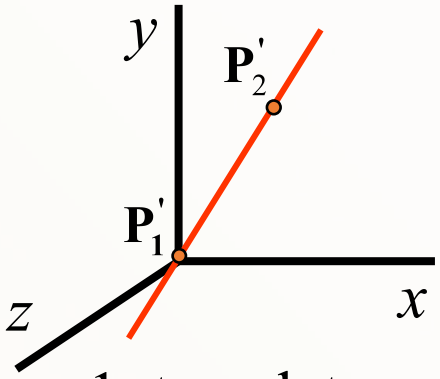
- More complex:

1. Translate such that axis goes through origin;
2. Rotate the object so that the axis of rotation coincides with one of the coordinate axes.
3. Perform the specified rotation about the selected coordinate axis.
4. Apply inverse rotations to bring the rotation axis back to its original orientation.
5. Apply the inverse translation to bring the rotation axis back to its original spatial position

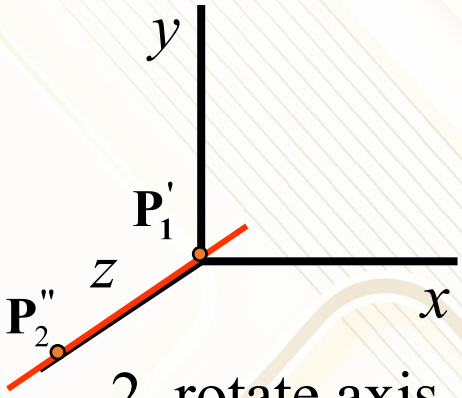
3D Rotation around arbitrary axis



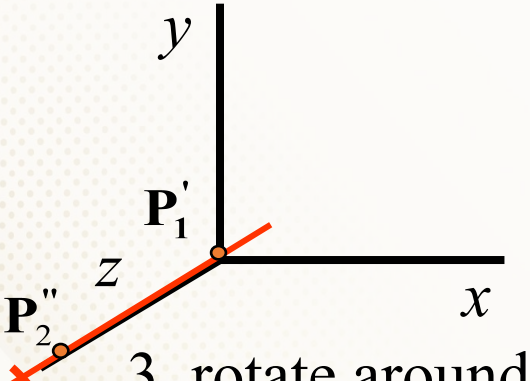
Initial



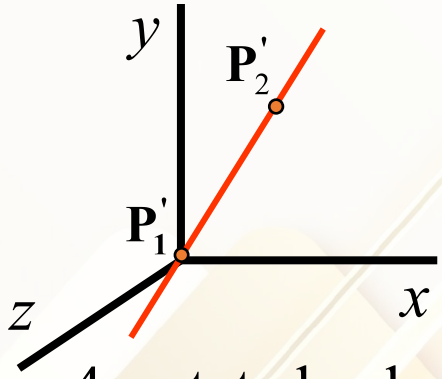
1. translate axis



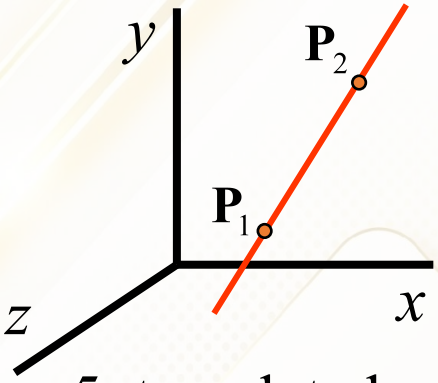
2. rotate axis



3. rotate around z-axis

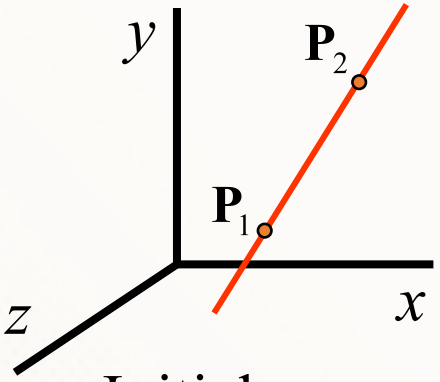


4. rotate back

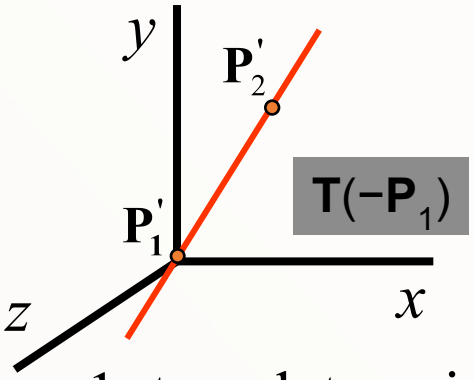


5. translate back

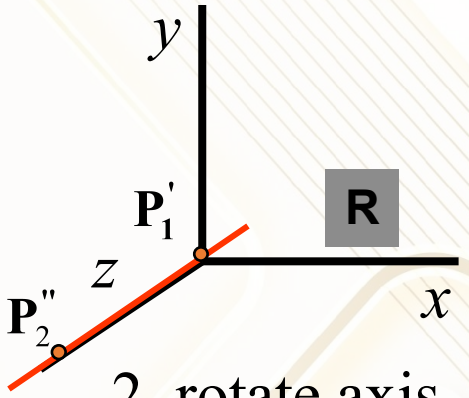
3D Rotation around arbitrary axis



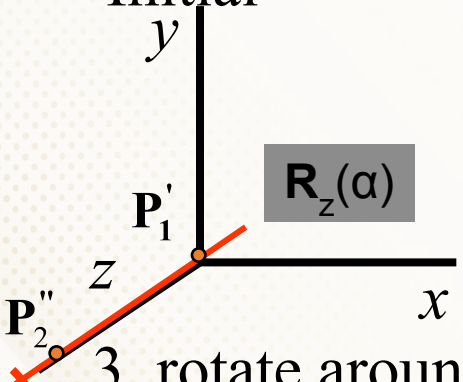
Initial



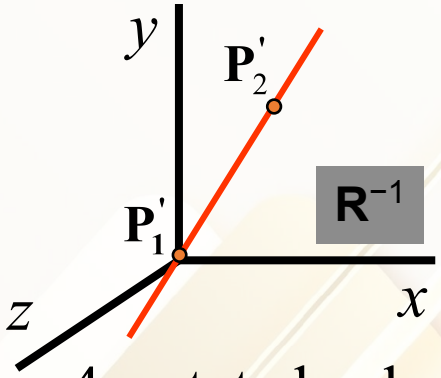
1. translate axis



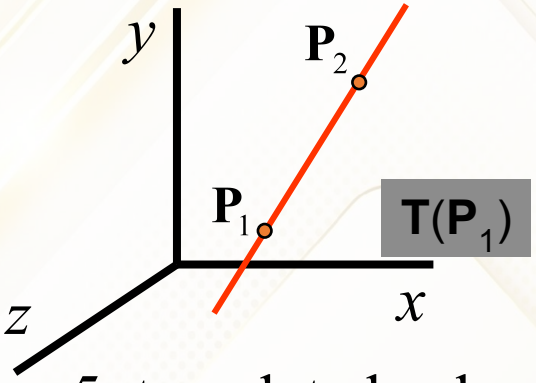
2. rotate axis



3. rotate around z-axis



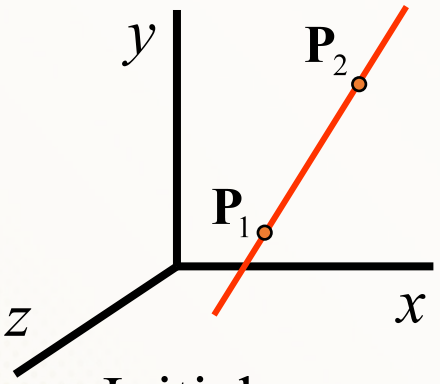
4. rotate back



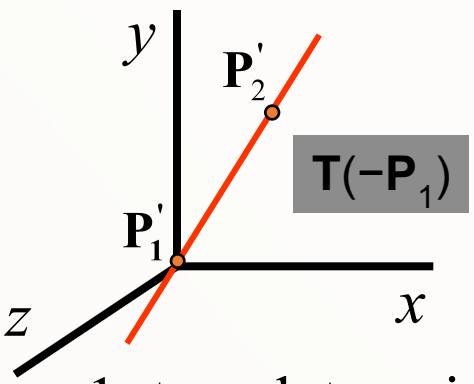
5. translate back

3D Rotation around arbitrary axis

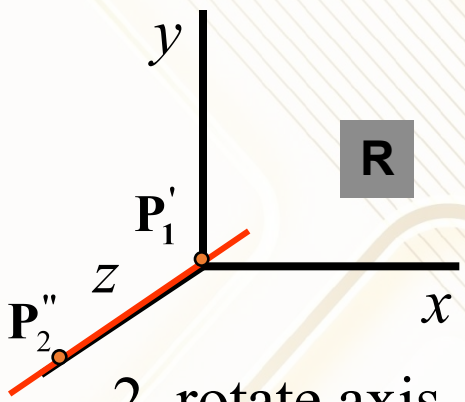
$$M = T(P_1)\, R^{-1} R_z(\alpha)\, RT(-P_1)$$



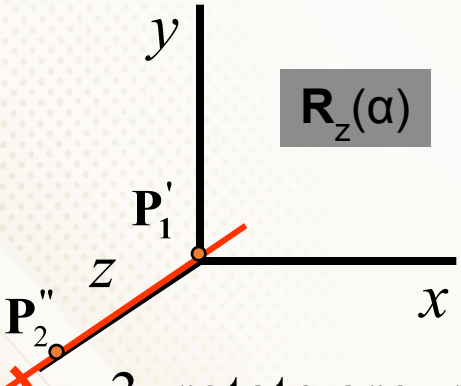
Initial



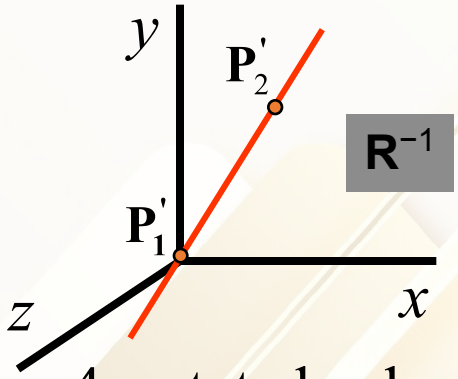
1. translate axis



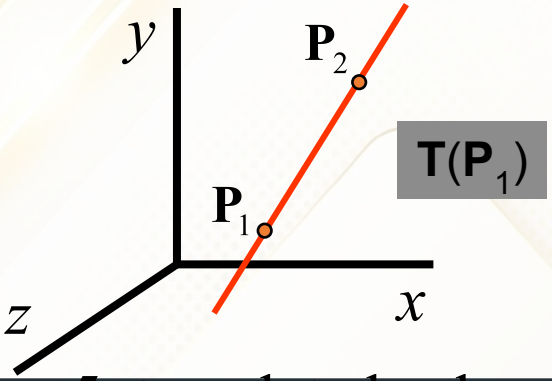
2. rotate axis



3. rotate around *z*-axis



4. rotate back



5. translate back

Final Transformation

$$R(\theta) = T^{-1} R_x(-\alpha) R_y(-\beta) R_z(\theta) R_y(\beta) R_x(\alpha).T$$

Overview of Three dimensional viewing Concepts

- Each object in the scene is defined with a set of surfaces that form a closed boundary around the object interior.
- Many processes in 3-D viewing, such as the clipping routines, are similar to those in the 2-D viewing pipeline, but 3-D viewing involves some tasks that are not present in 2-D viewing
  - Visible parts of a scene must be identified.
  - Lighting effects and surface characteristics.

Overview of Three dimensional viewing Concepts

RV UNIVERSITY
Go, change the world
an initiative of RV EDUCATIONAL INSTITUTIONS

- Viewing a Three-dimensional Scene
- Projections
- Depth Cueing
- Identifying Visible lines and surfaces
- Surface Rendering
- Exploded and cutaway views
- Three dimensional stereoscopic viewing

Viewing a Three-dimensional Scene

- First set up a coordinate reference for the viewing, or "camera", parameters
  - Defines the position and orientation for a view plane (or projection plane) that corresponds to a camera film plane
- Object descriptions are then transferred to the viewing reference coordinates and projected onto the view plane

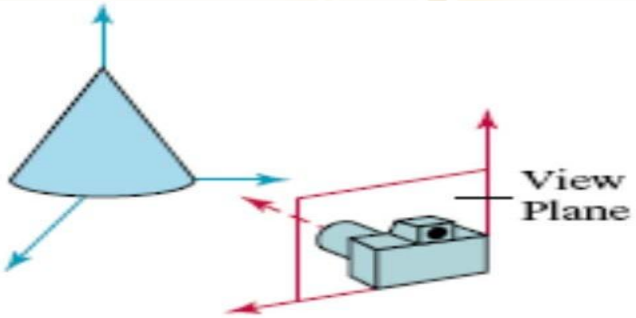
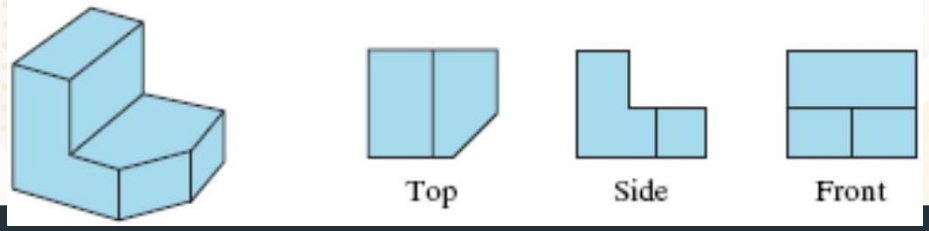


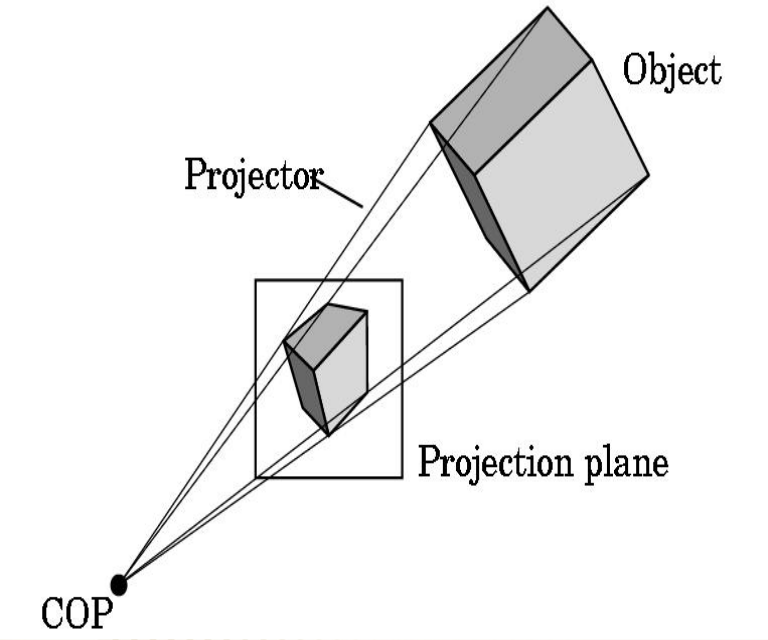
FIGURE 7–1 Coordinate reference for obtaining a selected view of a three-dimensional scene.

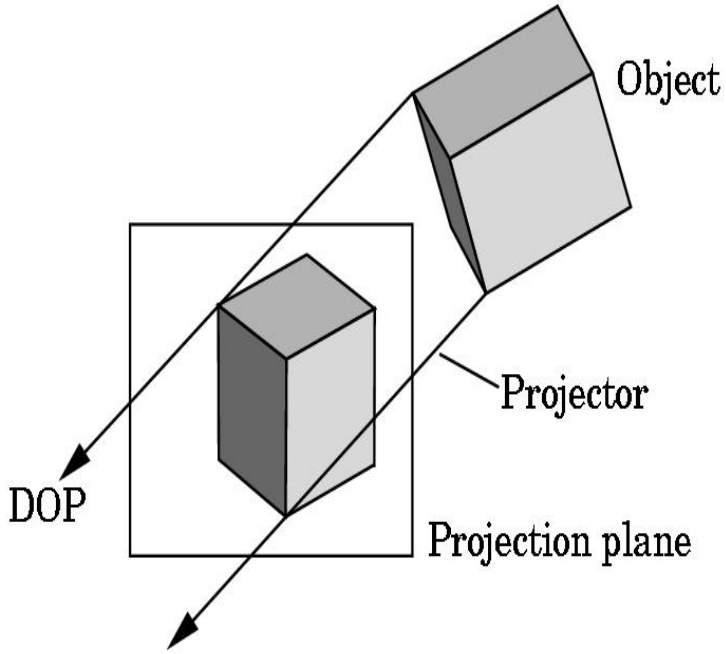
Projections

- Parallel projection
  - Getting the description of a solid object onto a view plane.
  - Project points on the object surface along parallel lines
- Perspective projection- more realistic
  - Generating a view of a three-dimensional scene
  - Project points to the view plane along converging paths



Projection

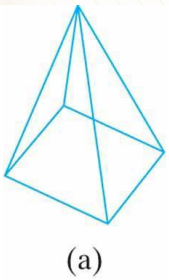

Object
Projector
Projection plane
COP


Object
Projector
DOP
Projection plane

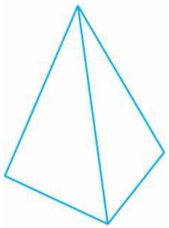
Depth Cueing

Can easily identify, for a particular viewing direction, which is the front and which is the back of each displayed object.

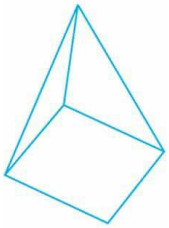
Fig illustrates ambiguity when wire-frame object is displayed without depth information



(a)



(b)



(c)

1.One 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.

Lines closer to the viewing position- highest intensity

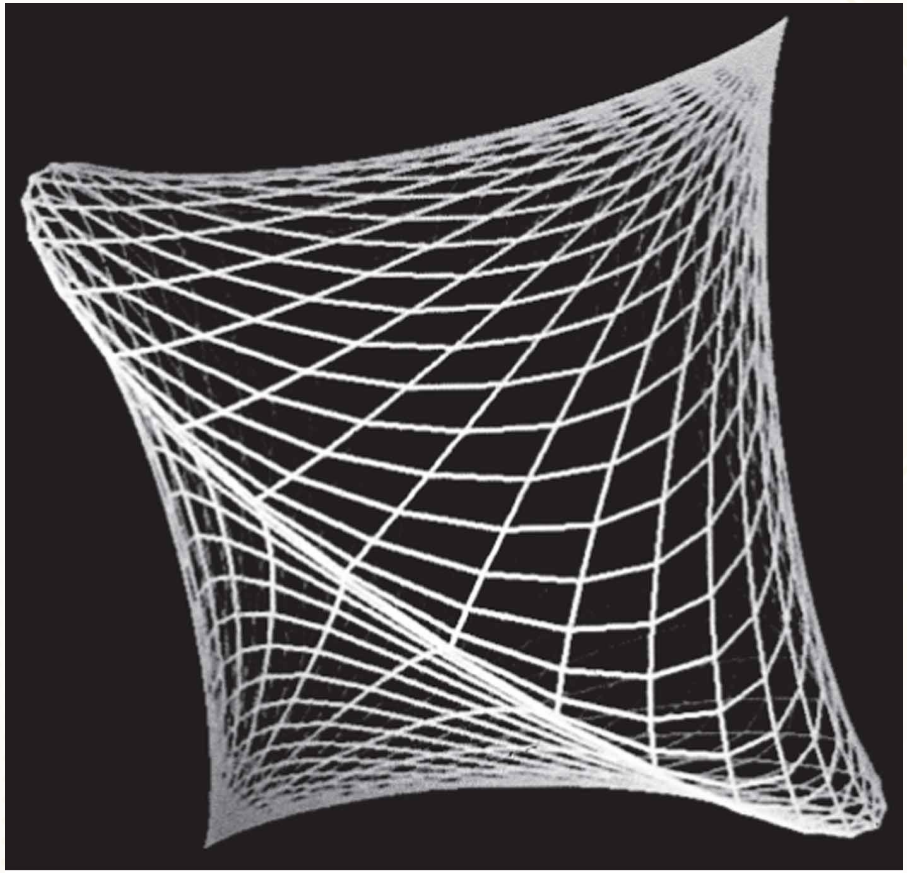
Lines farther- decreasing Intensities

Application

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

Figure 10-4 A wire-frame object displayed with depth cueing, so that the brightness of lines decreases from the front of the object to the back.



Identifying Visible Lines and Surfaces

- Clarify depth relationships in a wire-frame display using techniques other than depth cueing
  - Simply to highlight the visible lines or to display them in a different color
  - Common engineering technique is to display the nonvisible lines as dashed lines
  - Remove the nonvisible lines - also removes information about the shape of the back surfaces of an object
  - Wire-frame representations – indicate overall appearance, front and back.
  - For a realistic view, back parts of the objects are completely eliminated.

Surface Rendering

RV UNIVERSITY
Go, change the world
an initiative of RV EDUCATIONAL INSTITUTIONS

- Realism is attained By rendering object surfaces using the lighting conditions in the scene and the assigned surface characteristics.
- Also set background illumination effects
- Surface properties of objects - surface is transparent or opaque and whether the surface is smooth or rough.
- Surfaces such as glass, plastics, wood

RV UNIVERSITY
Go, change the world
an initiative of RV EDUCATIONAL INSTITUTIONS

Overview

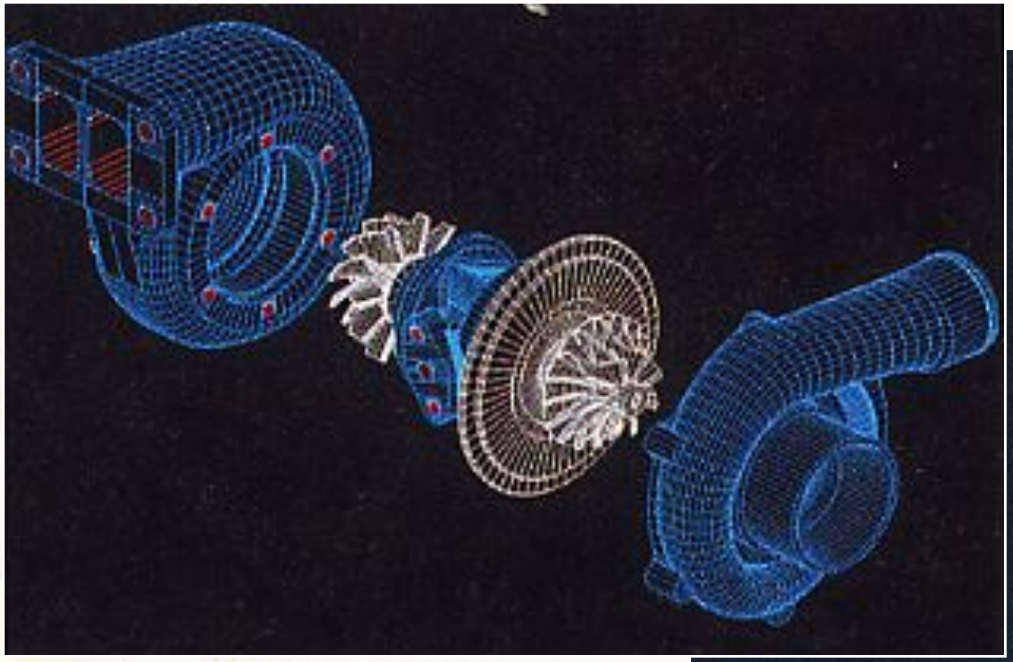
Surface rendering

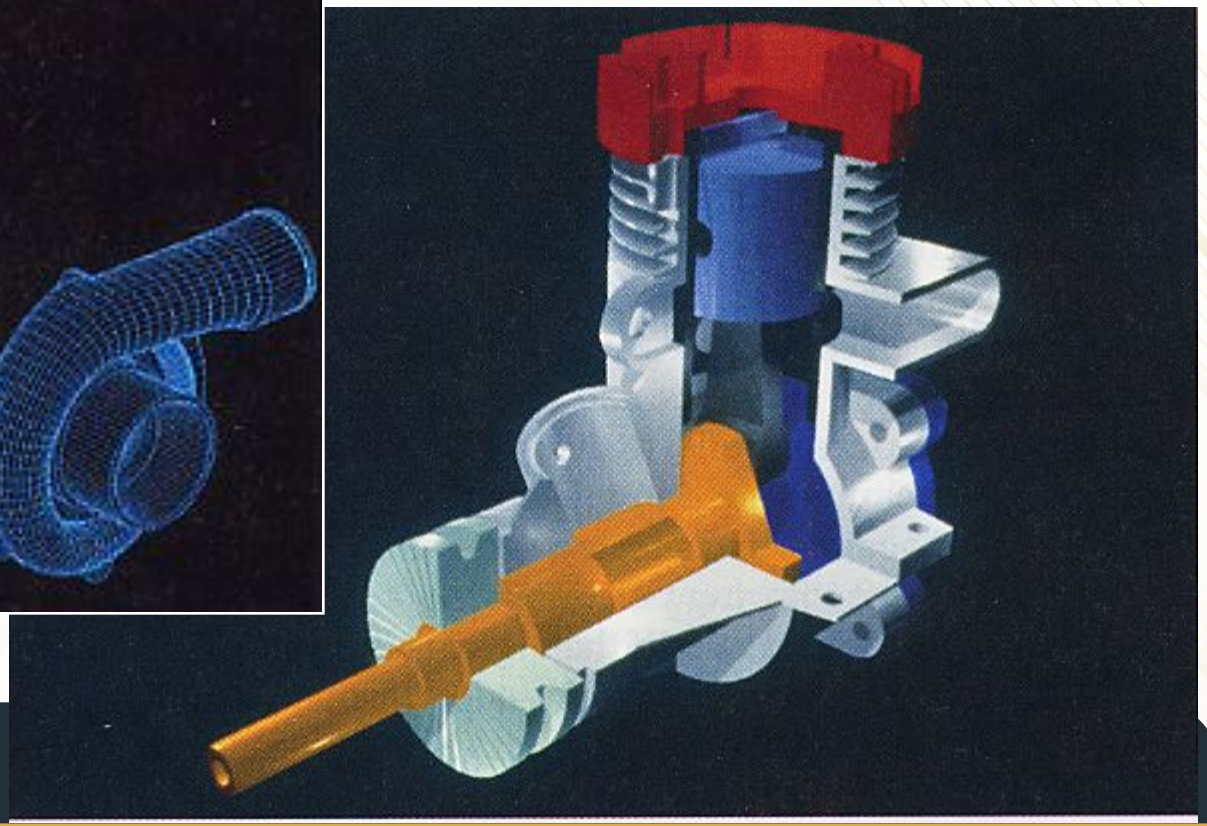
RV UNIVERSITY
Go, change the world
an initiative of RV EDUCATIONAL INSTITUTIONS

Exploded and cutaway views

Show internal structure and relationship of the object parts.

Remove visible surface to show internal structure.





Three dimensional and stereoscopic Viewing

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.

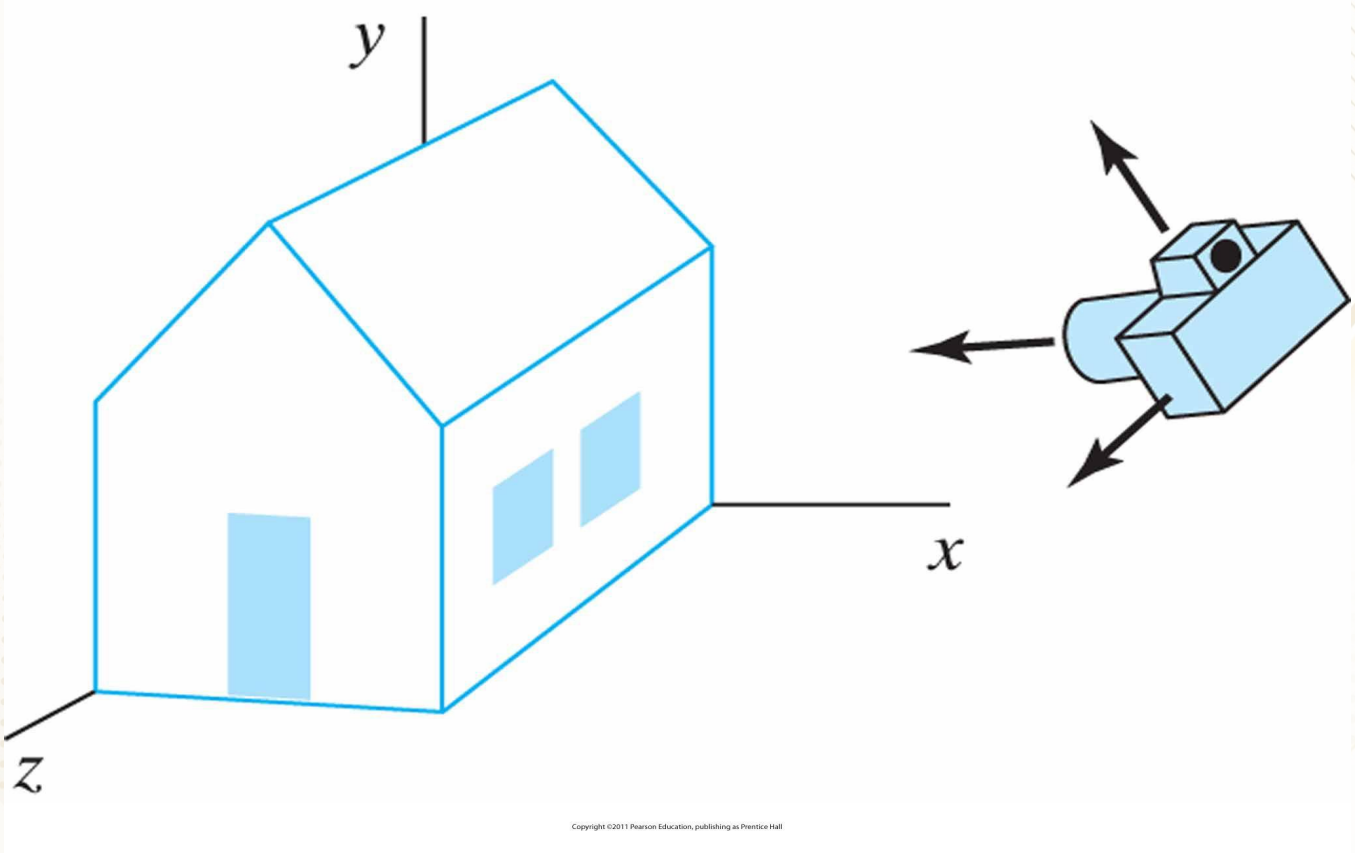
These two views are typically displayed on alternate refresh cycles of a raster monitor.

When we view the monitor through special glasses that alternately **darken first one lens** and then the other, in synchronization with the monitor refresh cycles, we see the scene displayed with a three-dimensional effect.

Three dimensional viewing pipeline

- Analogous to the processes involved in taking a photograph.
- 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.
- Then we must decide on the camera orientation.

Figure 10-5 Photographing a scene involves selection of the camera position and orientation.



More flexibility and many more options.

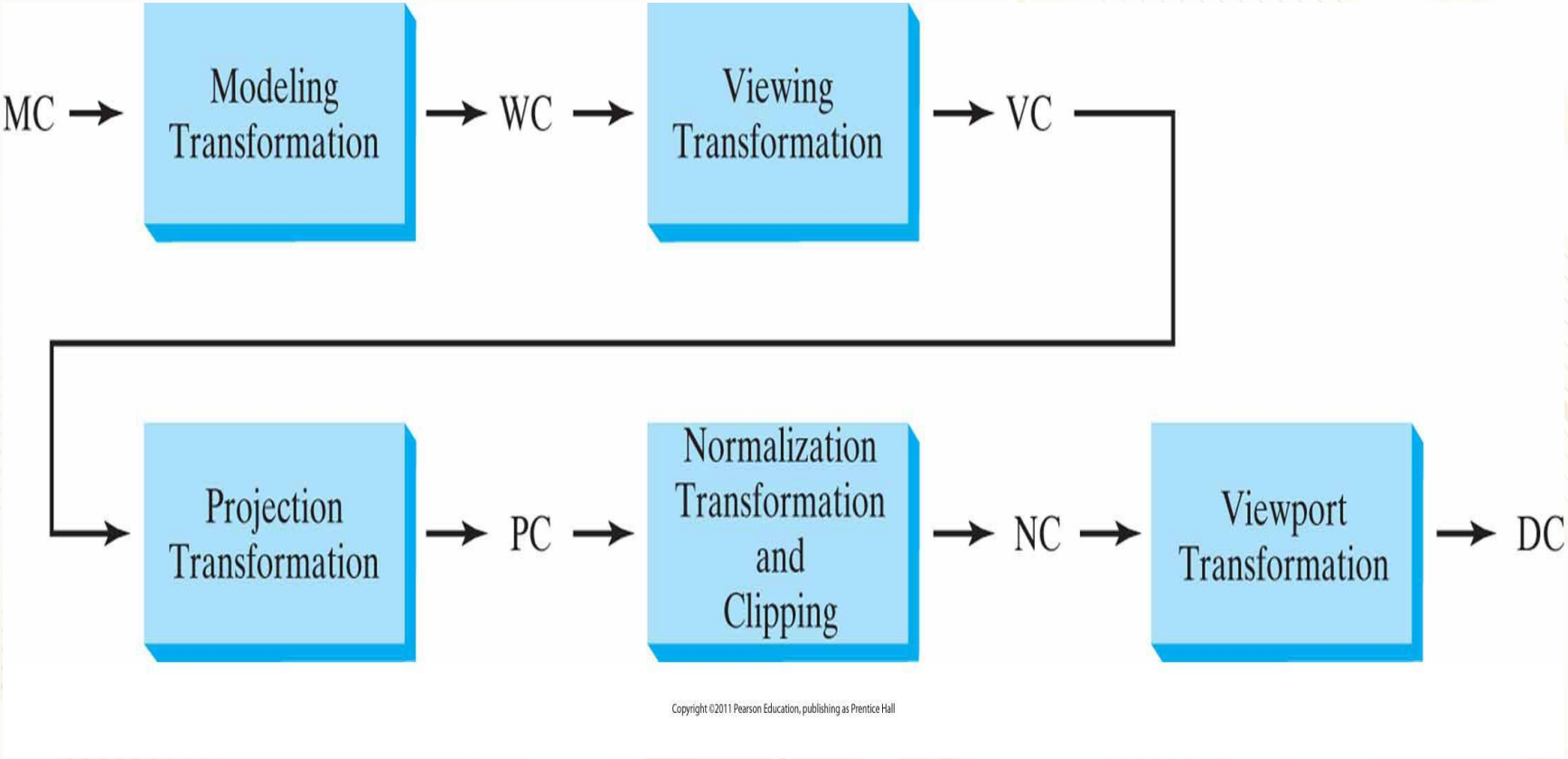
We can choose to use either a parallel projection or a perspective projection.

Move the projection plane away from the “camera” position.

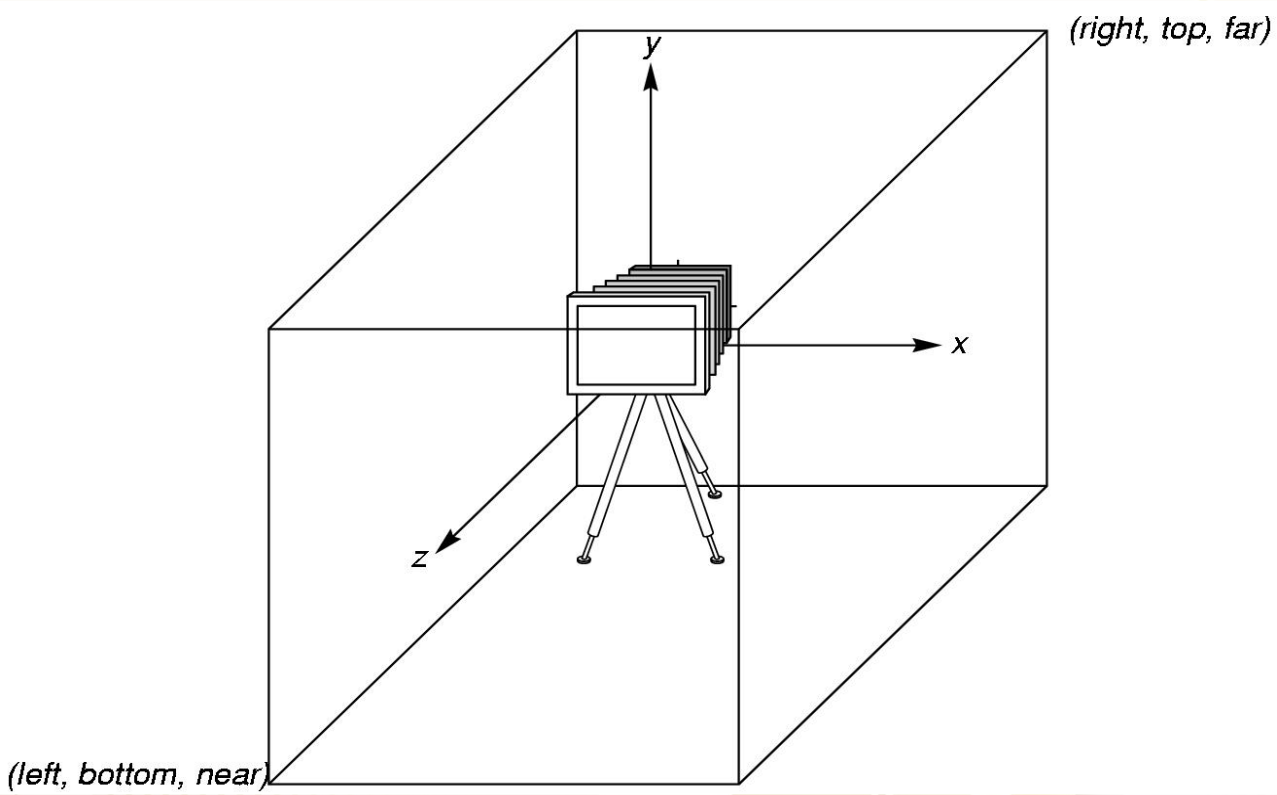
Even get a picture of objects in back of our synthetic camera.

Clipping windows, viewports, and display windows are usually specified as rectangles with their edges parallel to the coordinate axes.

Three dimensional viewing pipeline



The default camera and an orthographic view volume



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.

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.

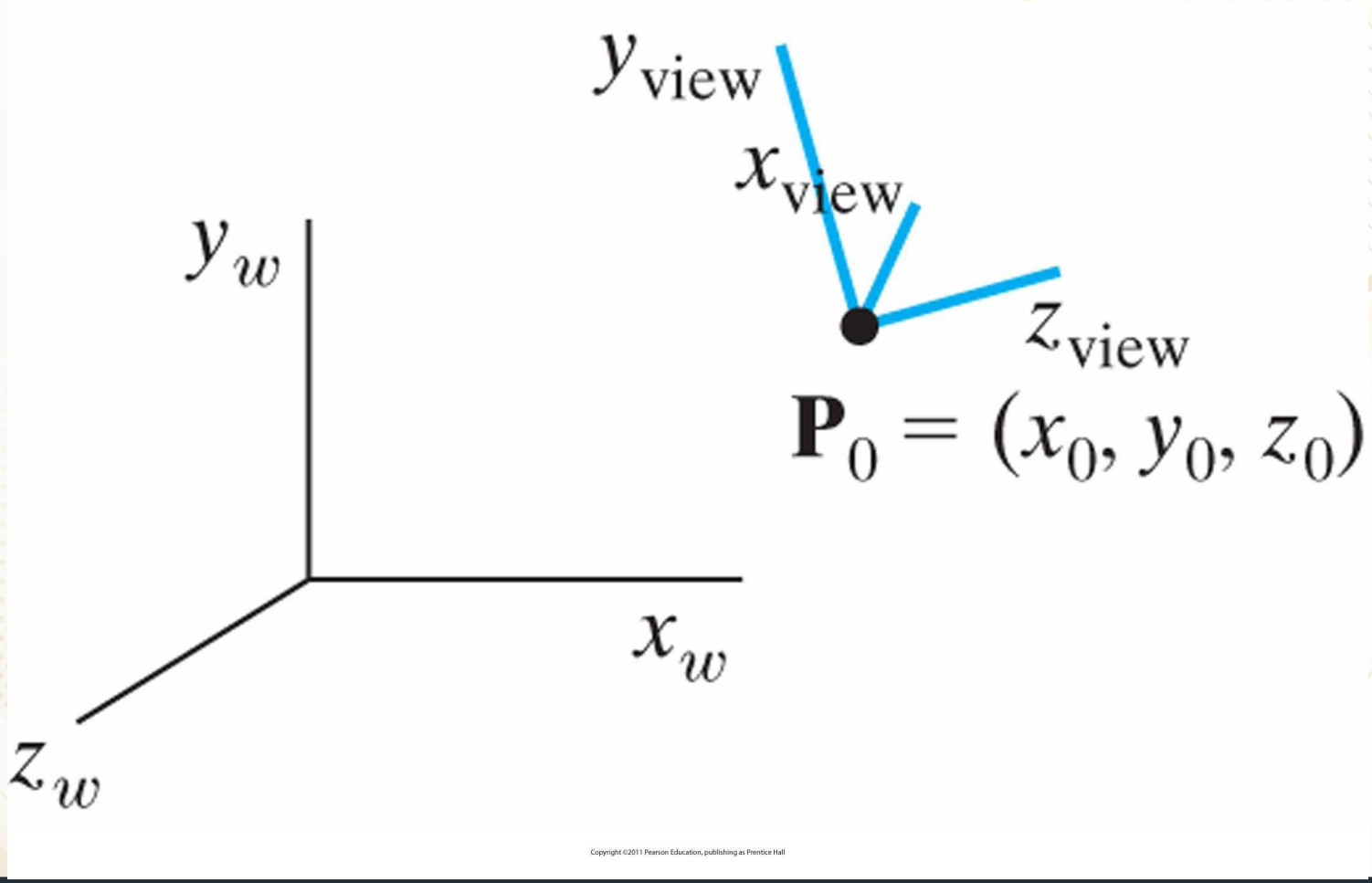
Normalized coordinates are transferred to viewport coordinates

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 or eye point or camera position.**

specify a view-up vector V, which defines the y_{view} direction.

Figure 10-7 A right-handed viewing-coordinate system, with axes x_{view}, y_{view}, and z_{view}, relative to a right-handed world-coordinate frame.



View Plane Normal Vector

Figure 10-8 Orientation of the view plane and view-plane normal vector **N**.

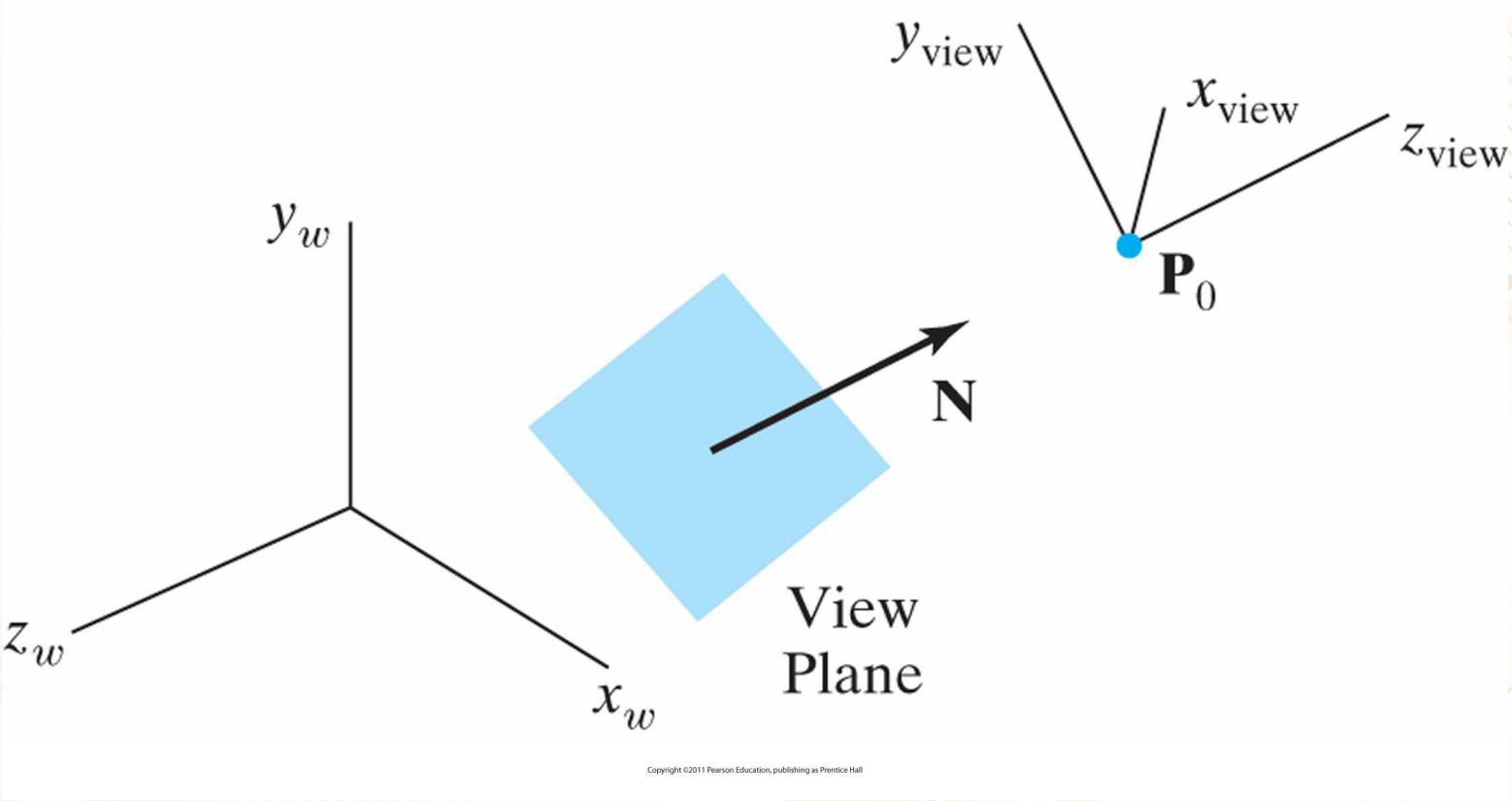
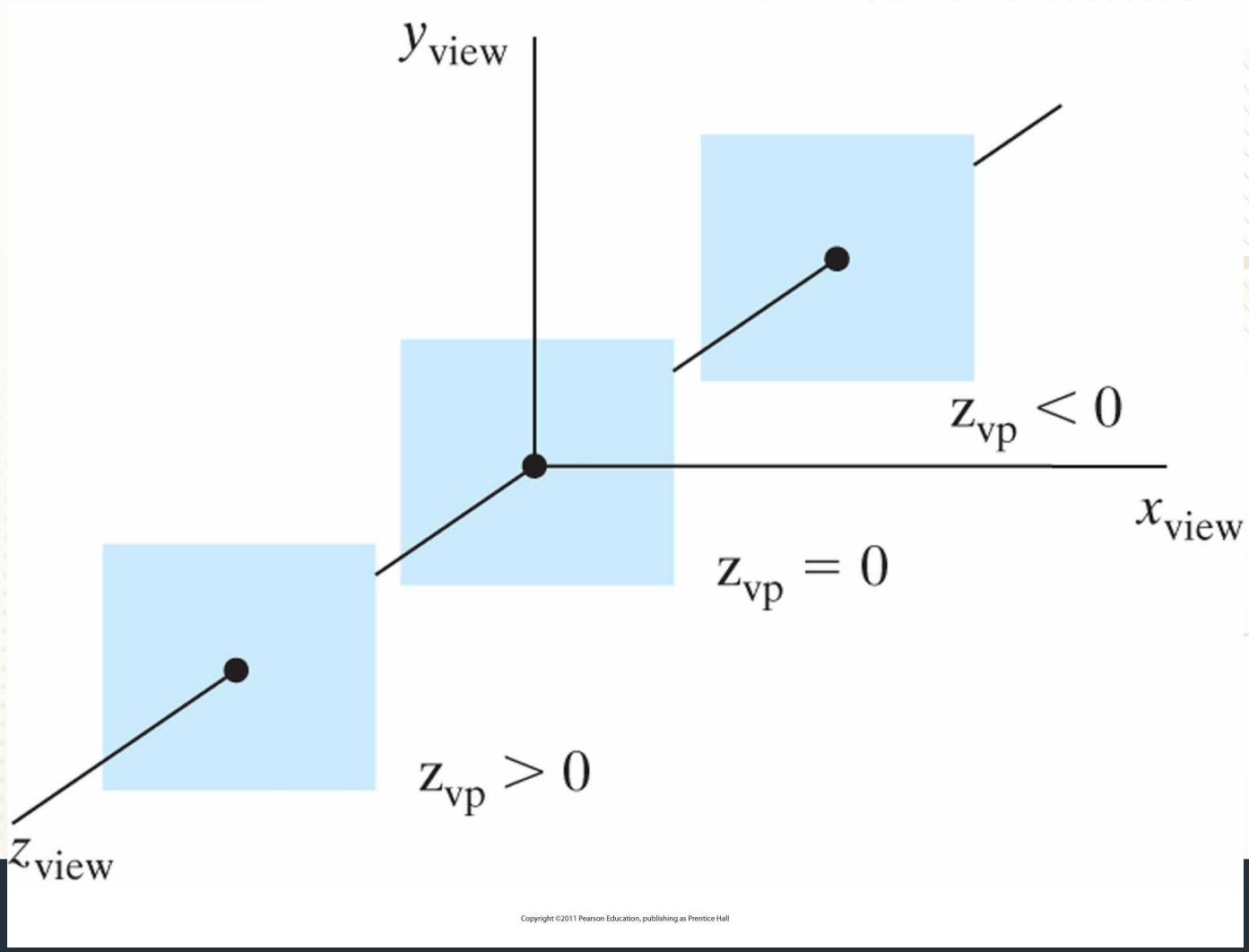


Figure 10-9 Three possible positions for the view plane along the z_{view} axis.



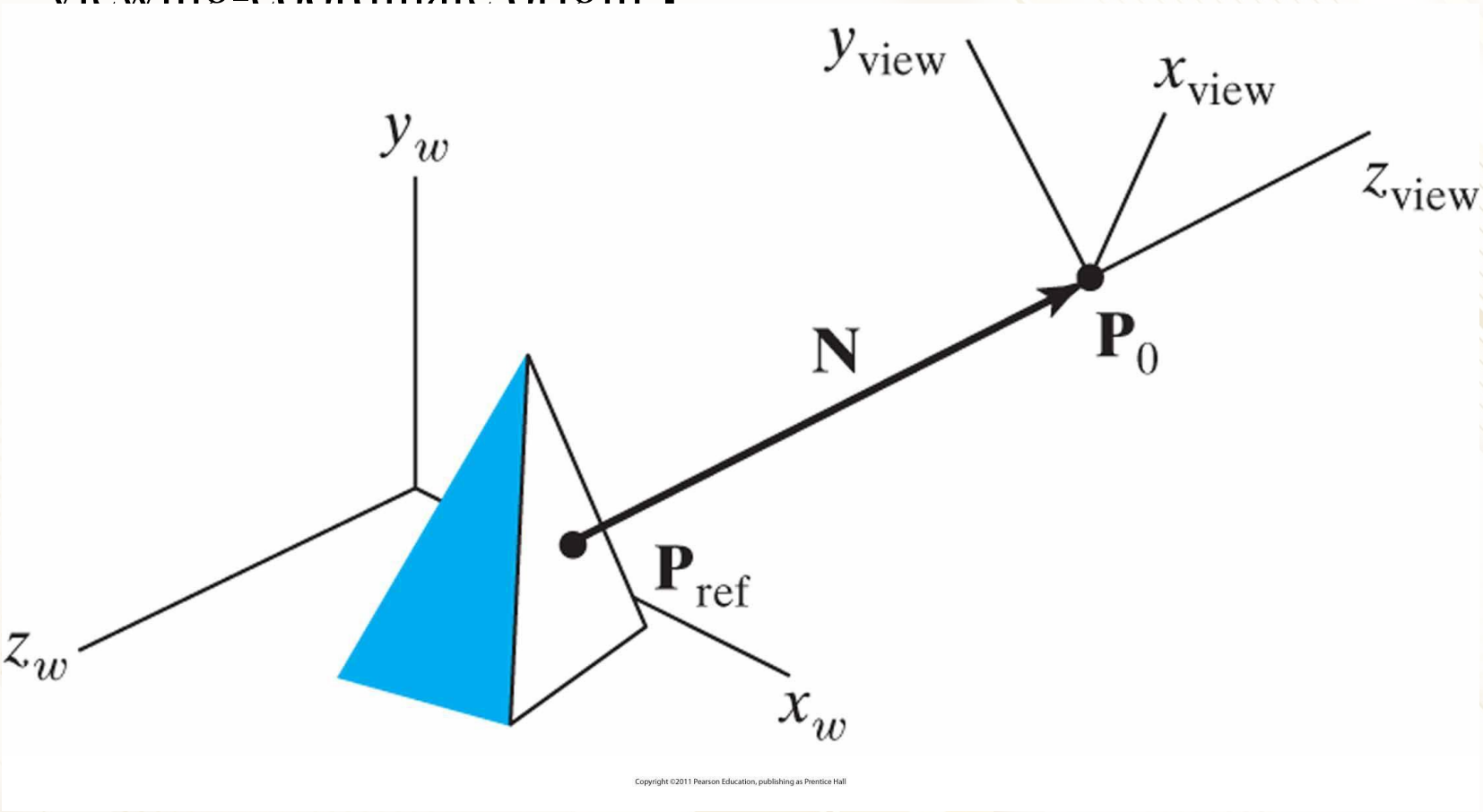
The view plane Normal vector

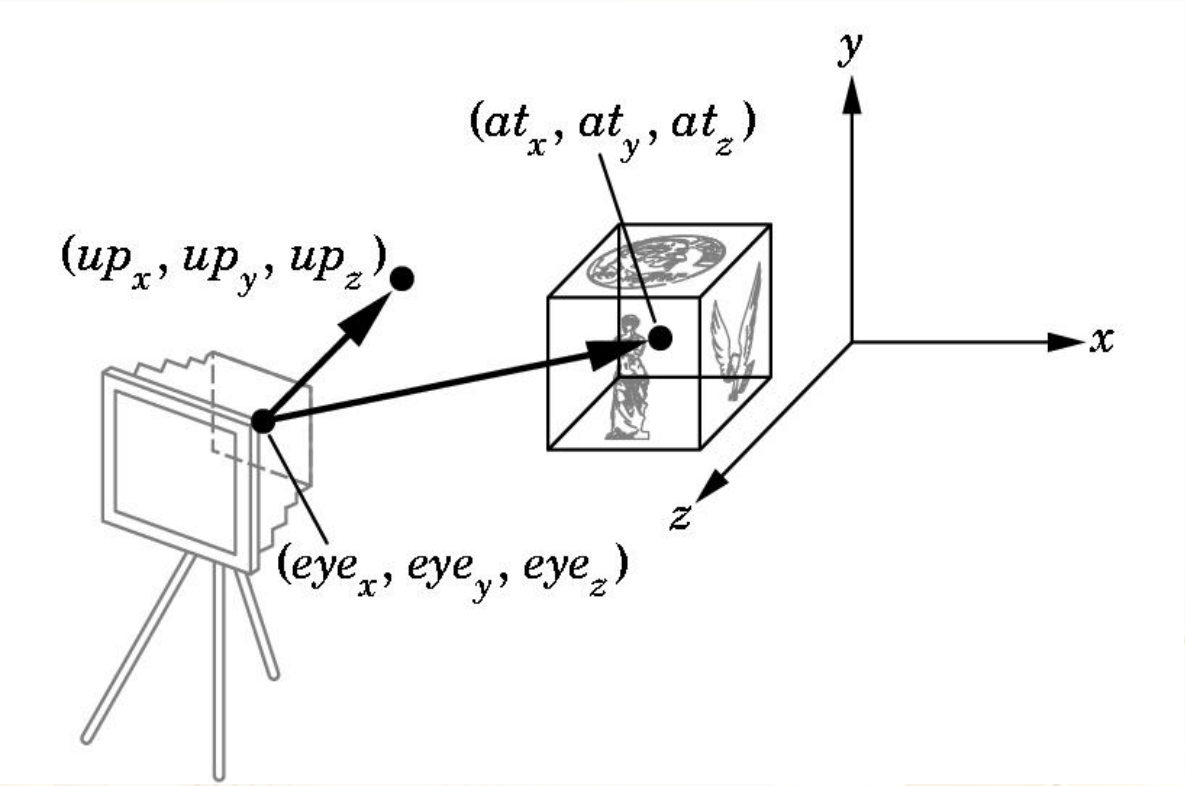
Viewing direction is usually along the z_{view} axis, the **view plane, also called the projection plane**, is normally assumed to be perpendicular to this axis.

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.**

view plane is always parallel to the $x_{view}y_{view}$ plane

Figure 10-10 Specifying the view-plane normal vector $\mathbf{N}$ as the direction from a selected reference point $\mathbf{P}_{ref}$ to the viewing-coordinate origin $\mathbf{P}_0$




(at_x, at_y, at_z)
(up_x, up_y, up_z)
(eye_x, eye_y, eye_z)
y
x
z

N to be in the direction from a reference point P_{ref} to the viewing origin P_0

Reference point is often referred to as a look-at point within the scene, with the viewing direction opposite to the direction of N.

View up vector

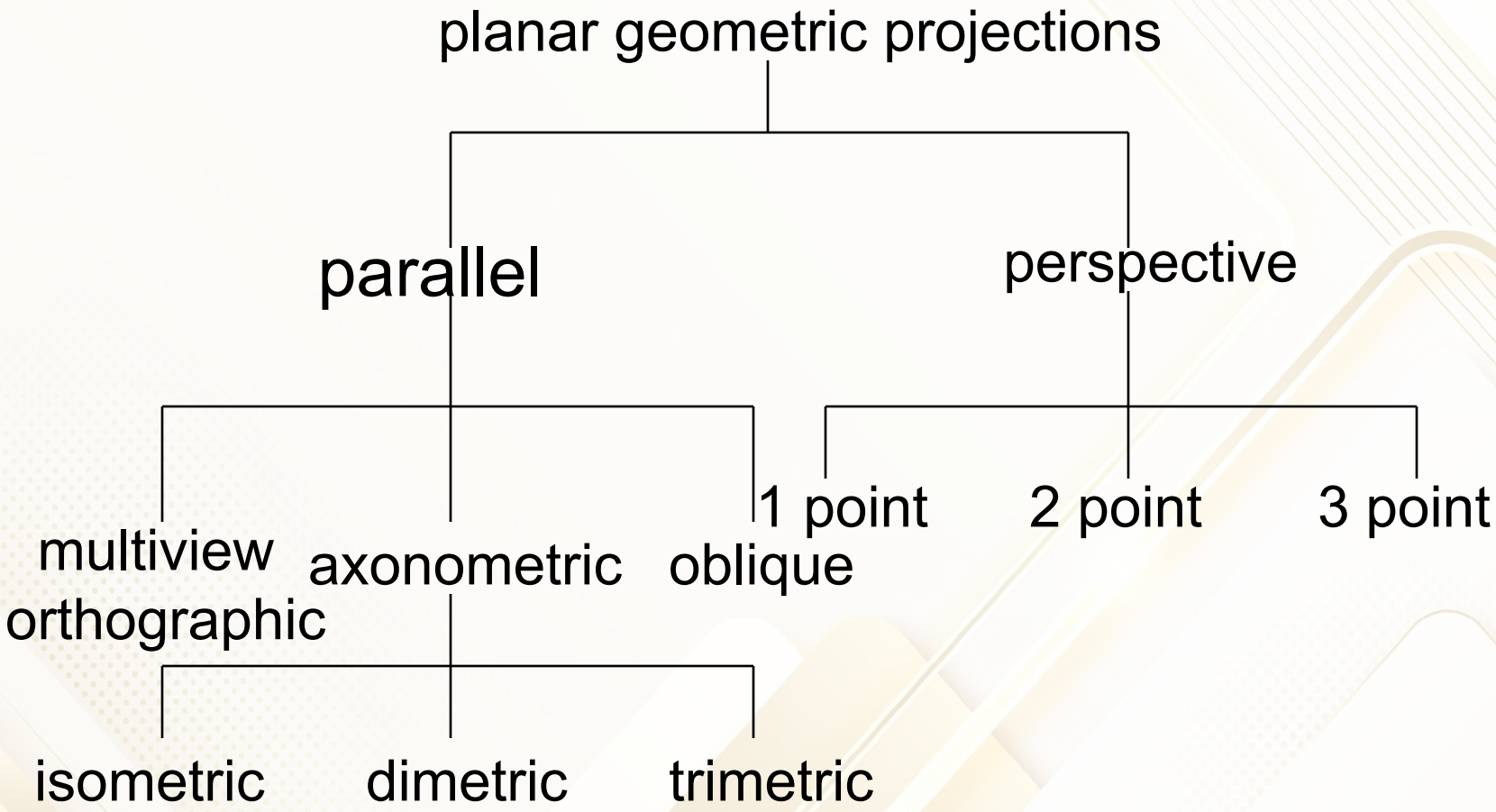
This vector is used to establish the positive direction for the y_{view} axis.

Because the view-plane normal vector N defines the direction for the z_{view} axis, vector V should be perpendicular to N.

We can choose any direction for the view-up vector V, so long as it is not parallel to N.

A convenient choice is often in a direction parallel to the world y_w axis; that is, we could set V= (0, 1, 0).

Taxonomy of Planar Geometric Projections

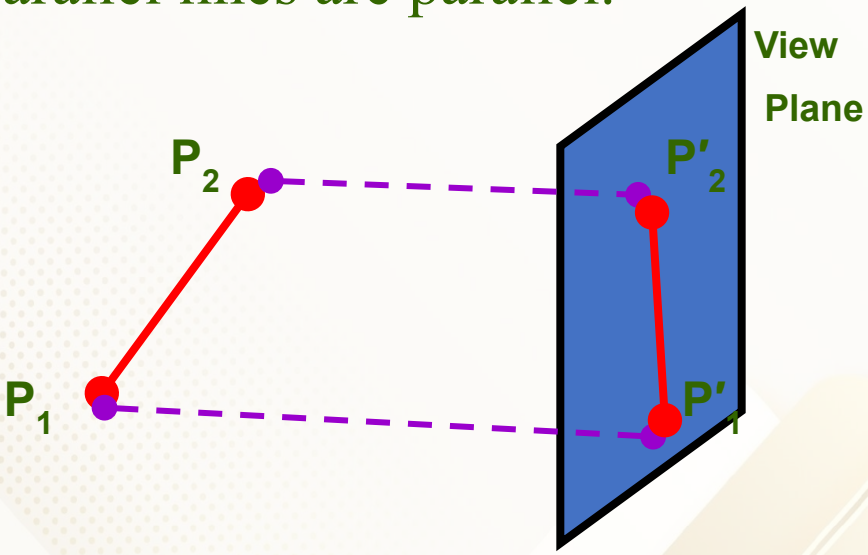


Projection Transformation
Parallel Projections

Coordinate Positions are transformed to the view plane along parallel lines.

Preserves relative proportions of objects.

Parallel lines are parallel.



Parallel Projections

- Orthographic parallel projection

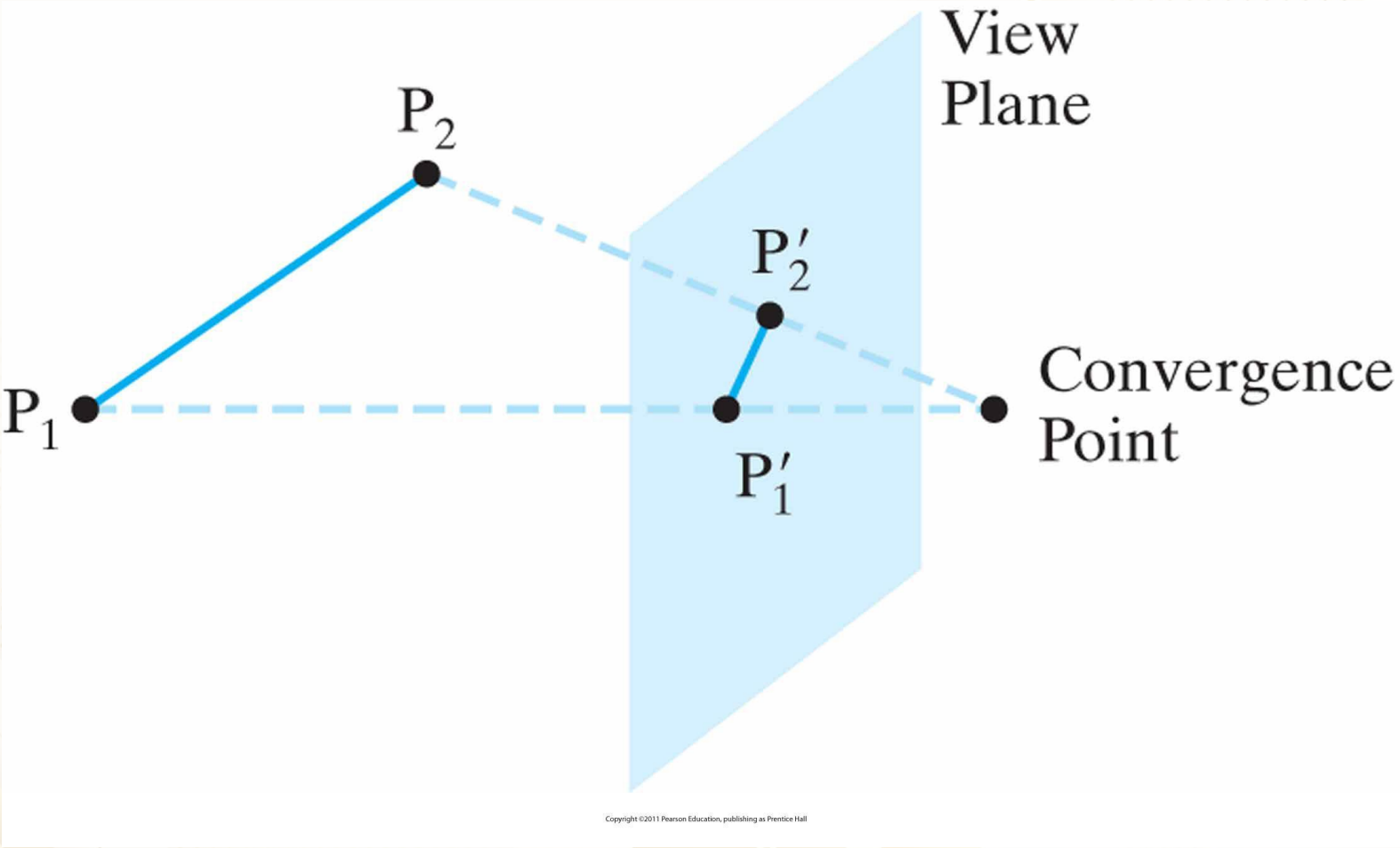
The projection is perpendicular to the view plane.

- Oblique parallel projecion

The parallel projection is not perpendicular to the view plane.

Project at an oblique angle to the view plane

Figure 10-16 Perspective projection of a line segment onto a view plane.



Perspective projection

object positions are transformed to projection coordinates along lines that converge to a point behind the viewplane.

Does not preserve relative proportions of objects.

More realistic.

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.

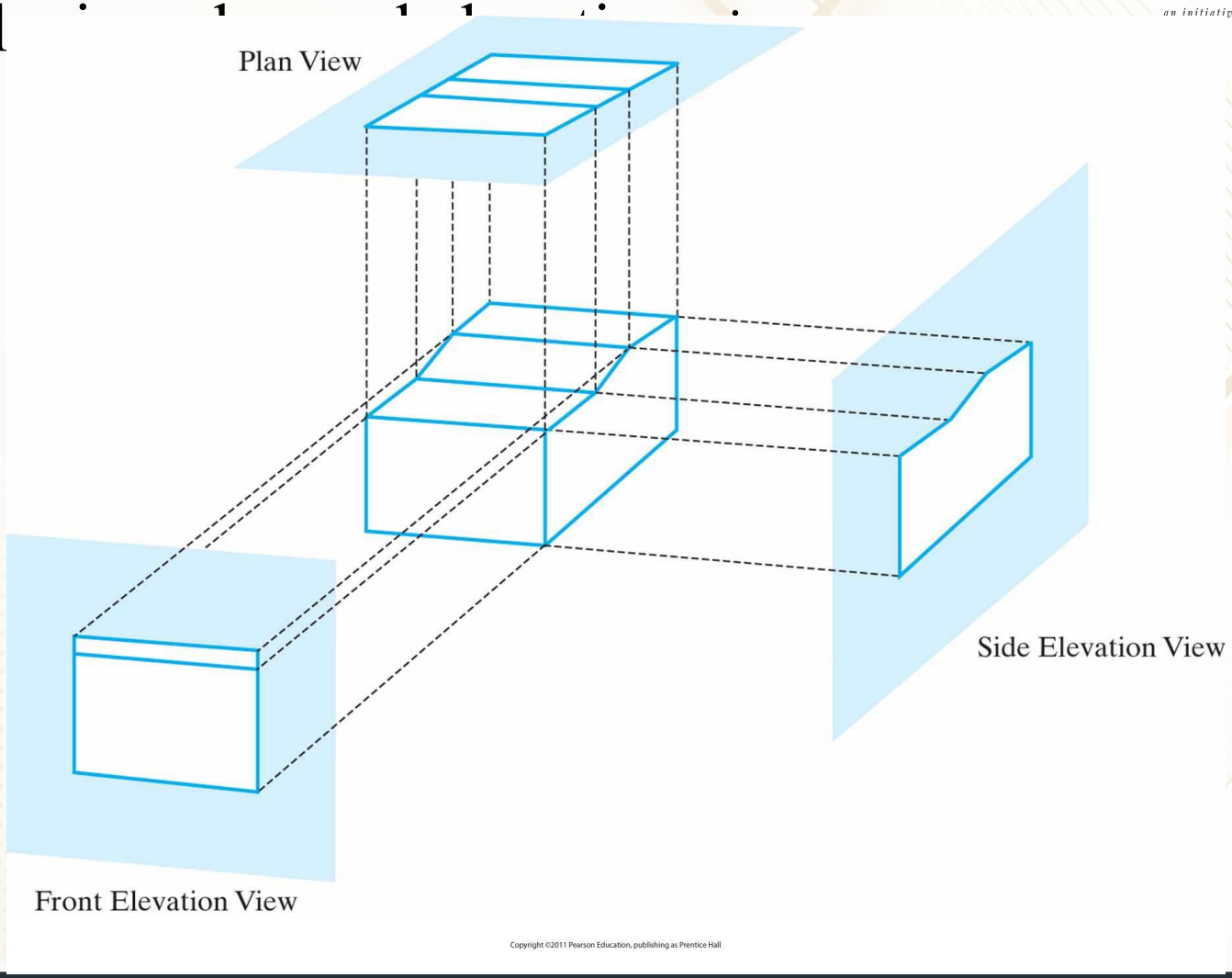
Projection lines are perpendicular to the view plane.

Front, side, and rear projections of an object are called **elevations.**

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.

Figure 10-17 Orthogonal projections of an object, displ



Axonometric Projections

If we want to see more principal faces of object in a single view ,we can use axonometric projections.

Projection plane is placed symmetrically
With respect to the three principal faces
Then the view is **isometric.**

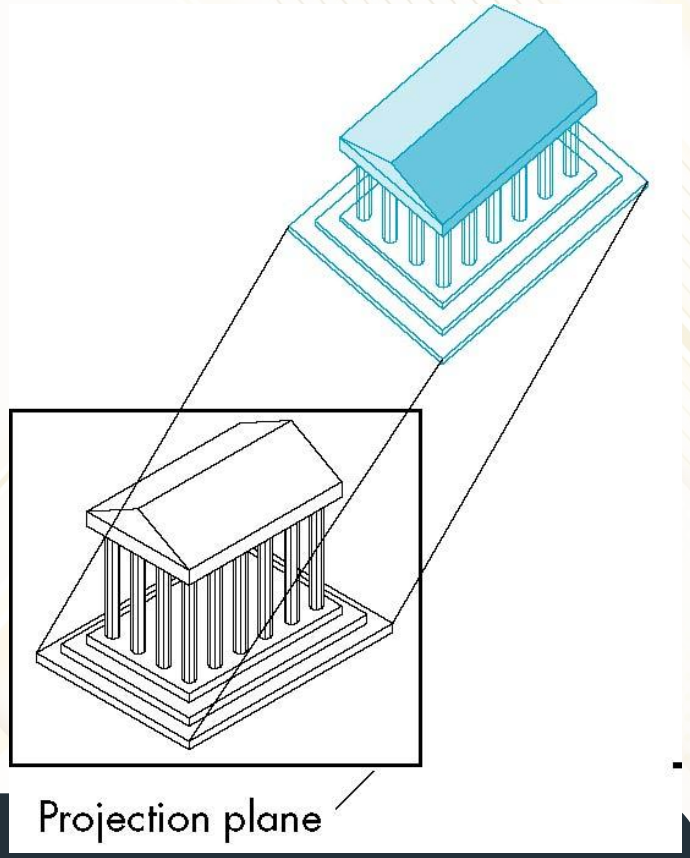
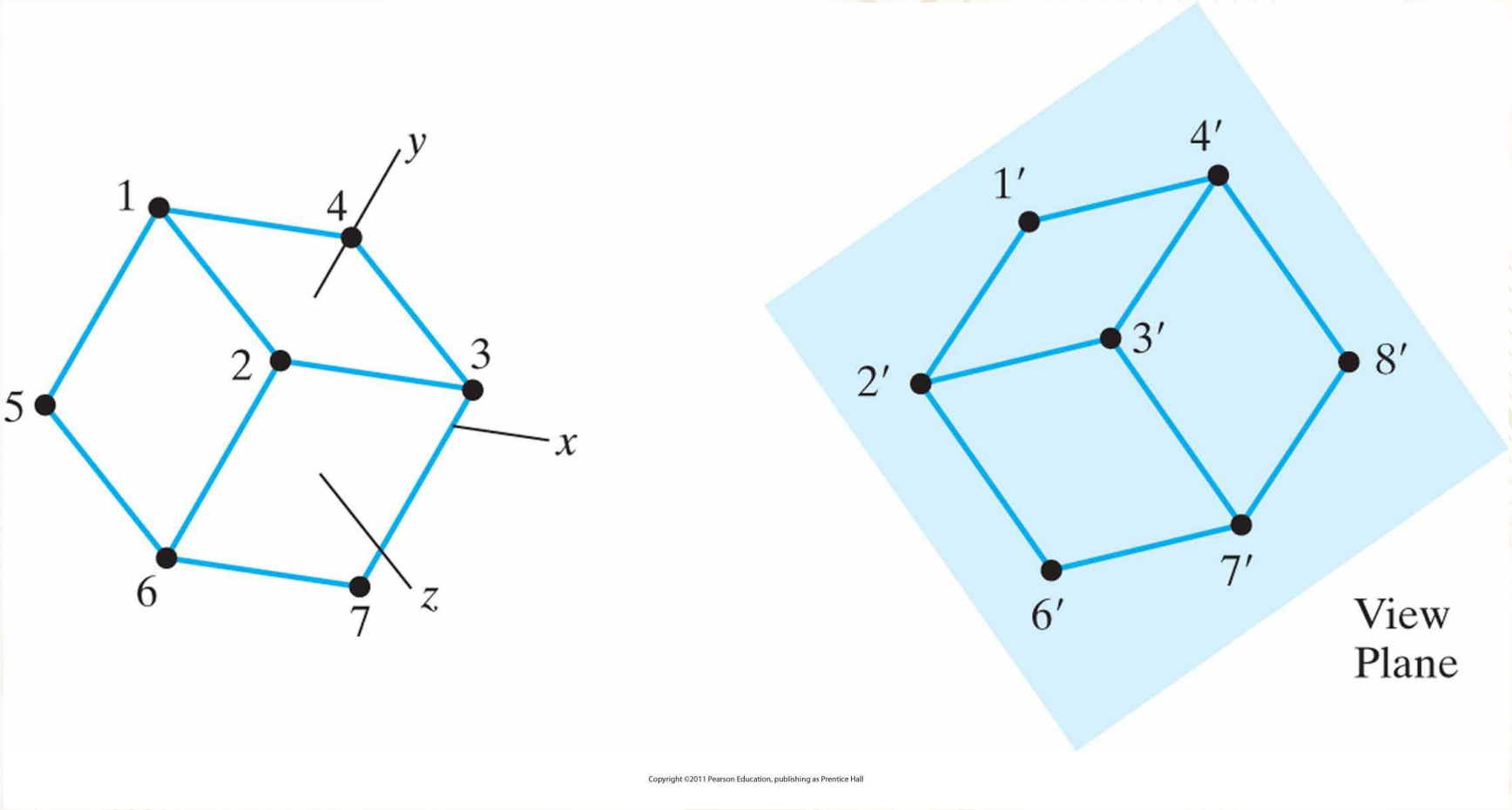


Figure 10-18 An isometric projection of a cube.



Isometric

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.

Clipping window and orthogonal –projection view volume

Type of lens determines how much of the scene is transferred to the film plane

Clipping window corresponds to type of lens.

Set up a clipping window for 3D viewing, by choosing lower left and upper right corners.

Orthogonal projection view volume

Finite orthogonal view volume- right rectangular parallelepiped

Figure 10-20 A clipping window on the view plane, with minimum and maximum coordinates given in the viewing reference system.

Clipping window is positioned on view plane with edges parallel to x_{view} and y_{view} axes.

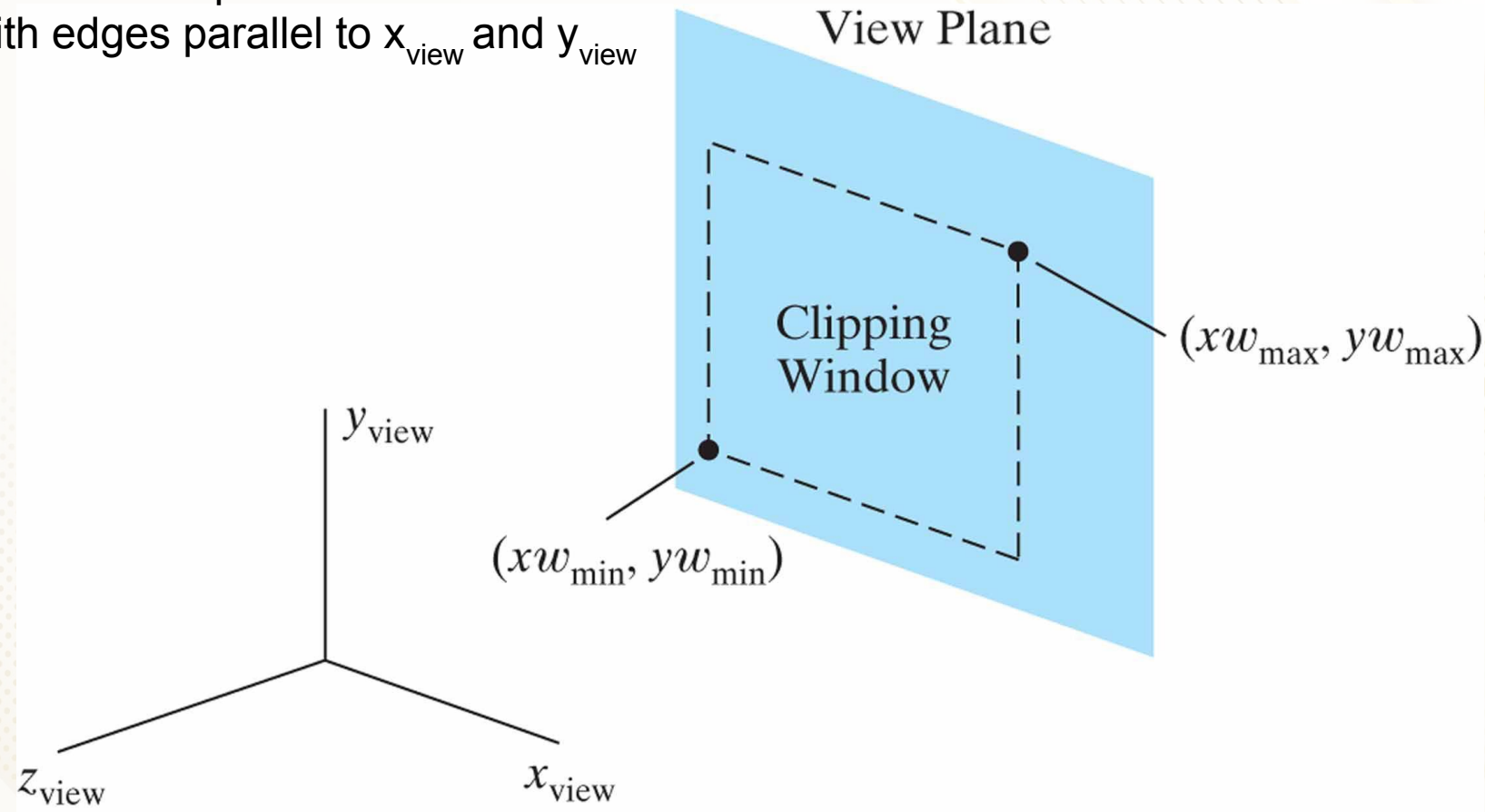


Figure 10-21 Infinite orthogonal-projection view volume.

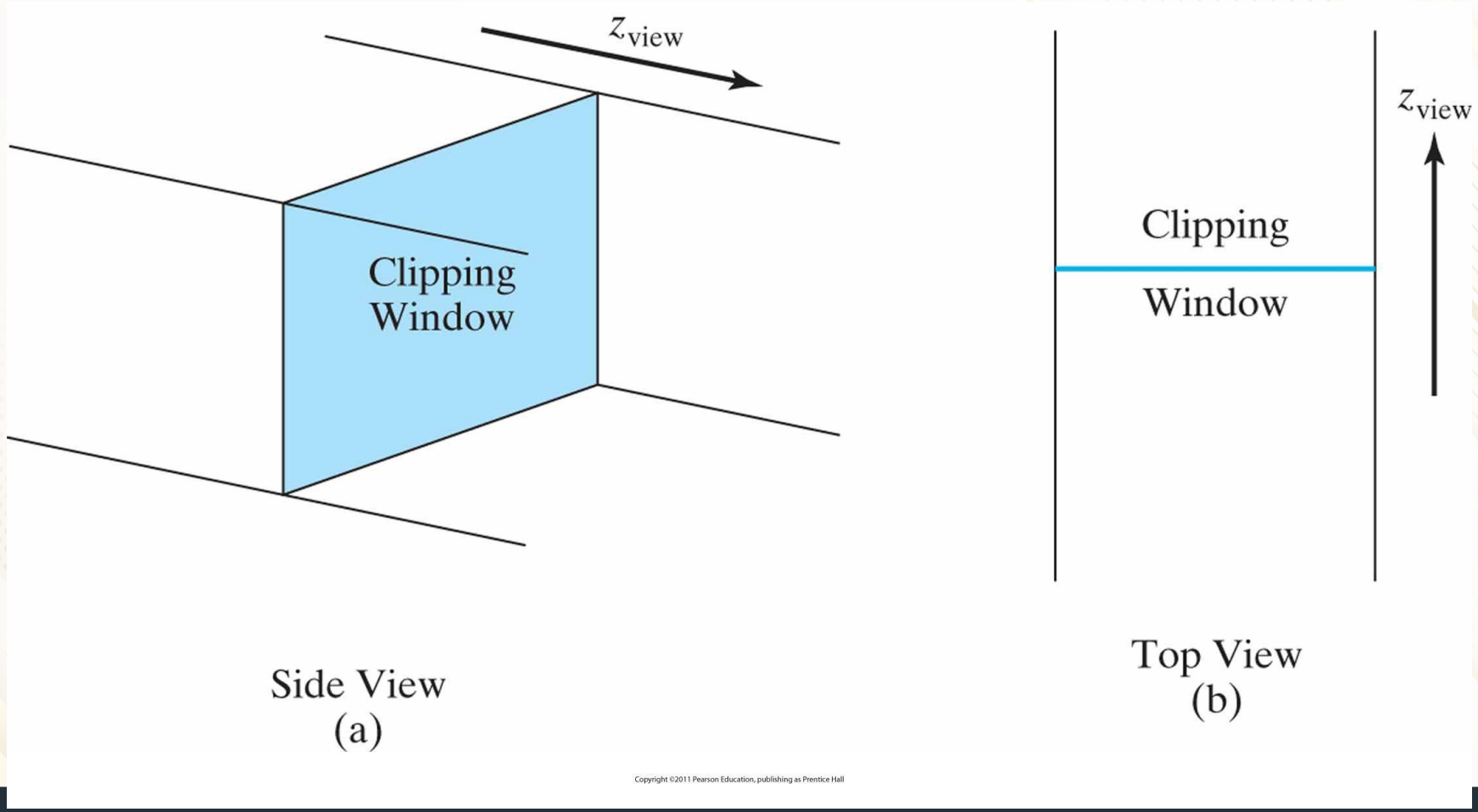
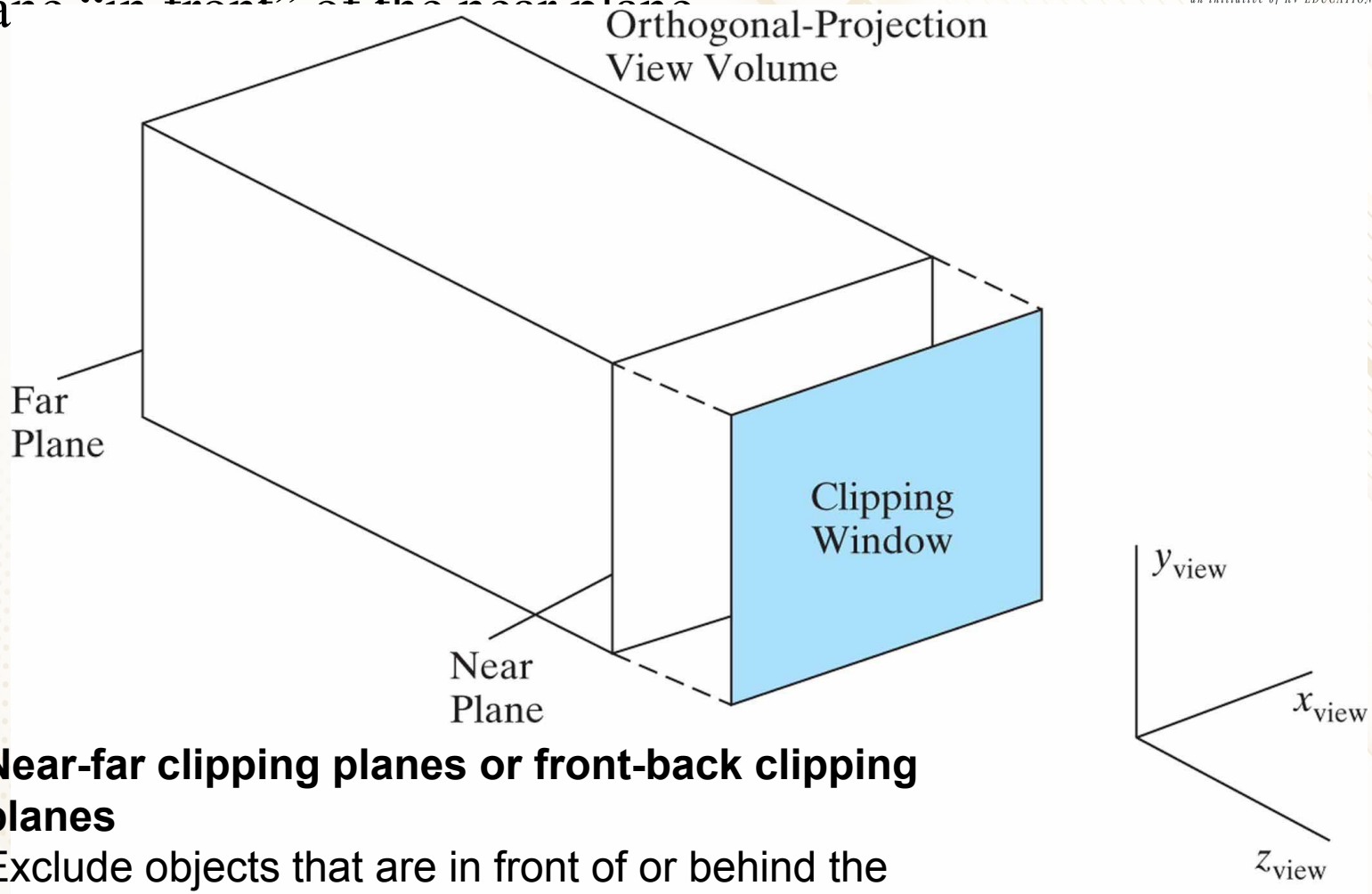


Figure 10-22 A finite orthogonal view volume with the view plane "in front" of the near plane



Near-far clipping planes or front-back clipping planes

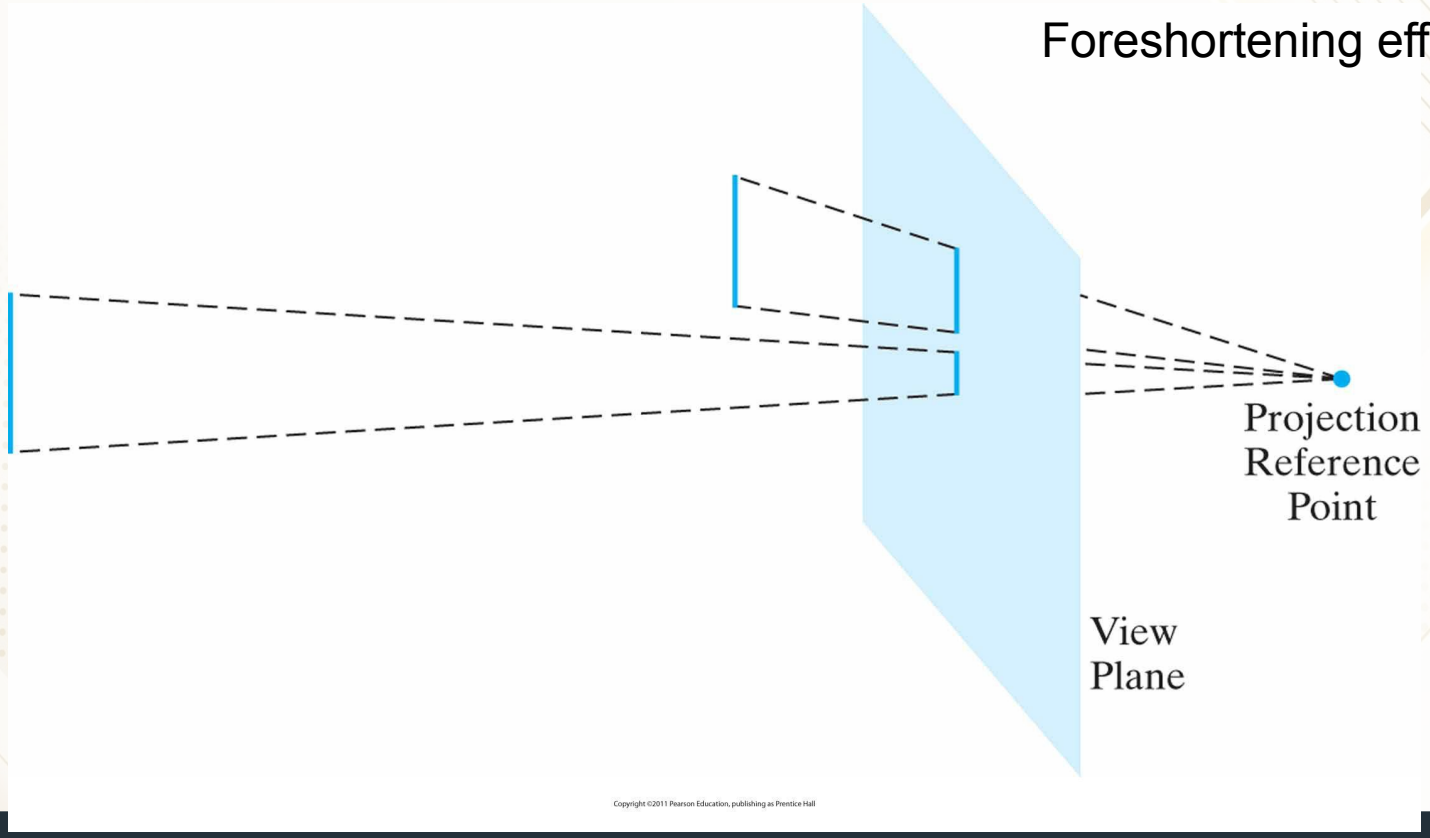
Exclude objects that are in front of or behind the part of the scene that we want to display.

Viewing direction along Negative Z axis - $z_{far} < z_{near}$

Perspective Projections

Figure 10-33 A perspective projection of two equal-length line segments at different distances from the view plane.

- projecting objects to the view plane along converging paths to a position called the projection reference point.



we cannot have the projection reference point on the view plane.

In that case, entire scene would project to a single point.

The view plane is usually placed between the projection reference point and the scene.

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.

Perspective-Projection View Volume

The bottom, top, and sides of the view volume are planes through the window edges that all intersect at the projection reference point.

This forms a view volume that is an infinite rectangular pyramid with its apex at the centre of projection.

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.

Figure 10-38 An infinite, pyramid view volume for a perspective projection.

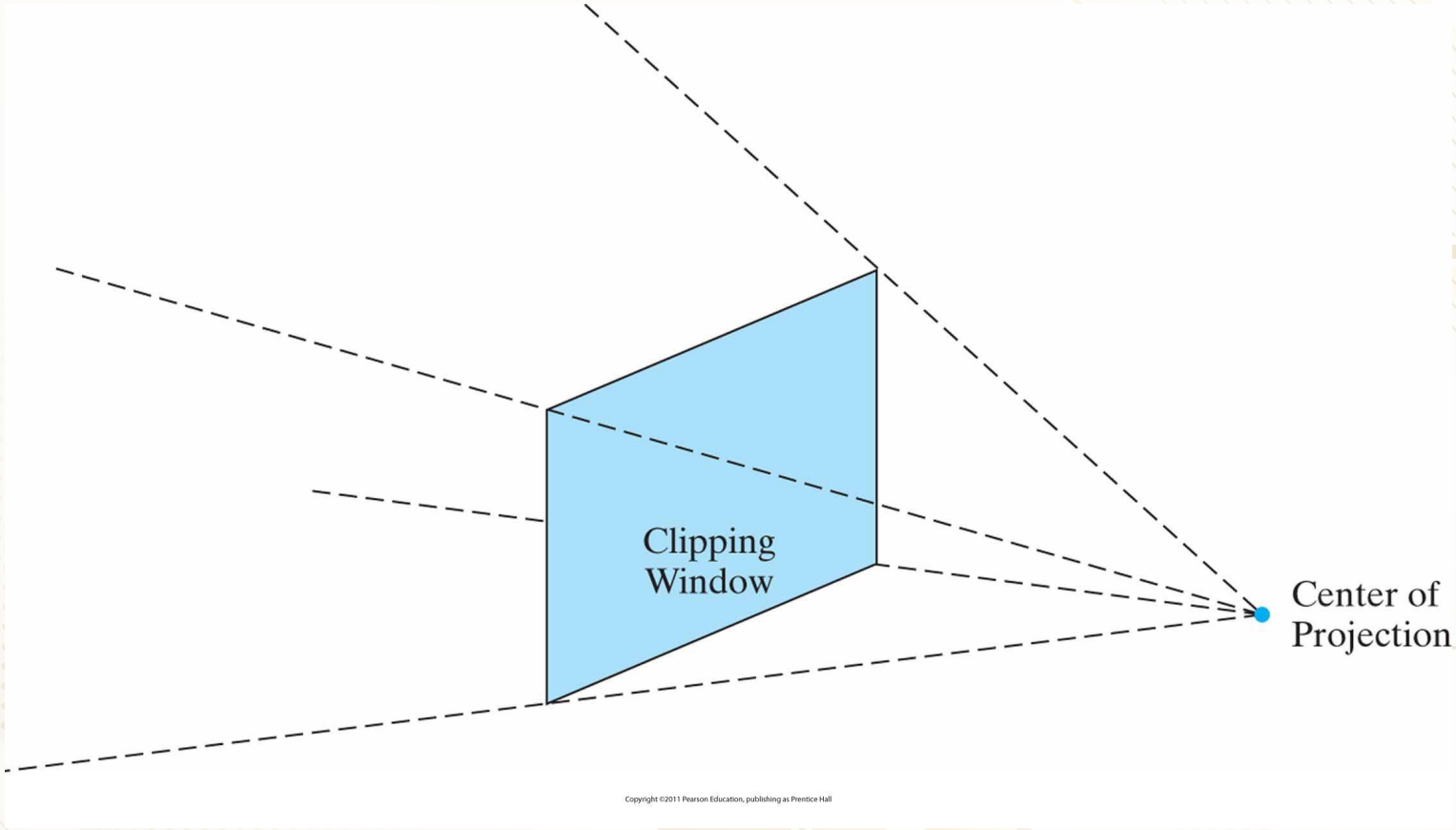
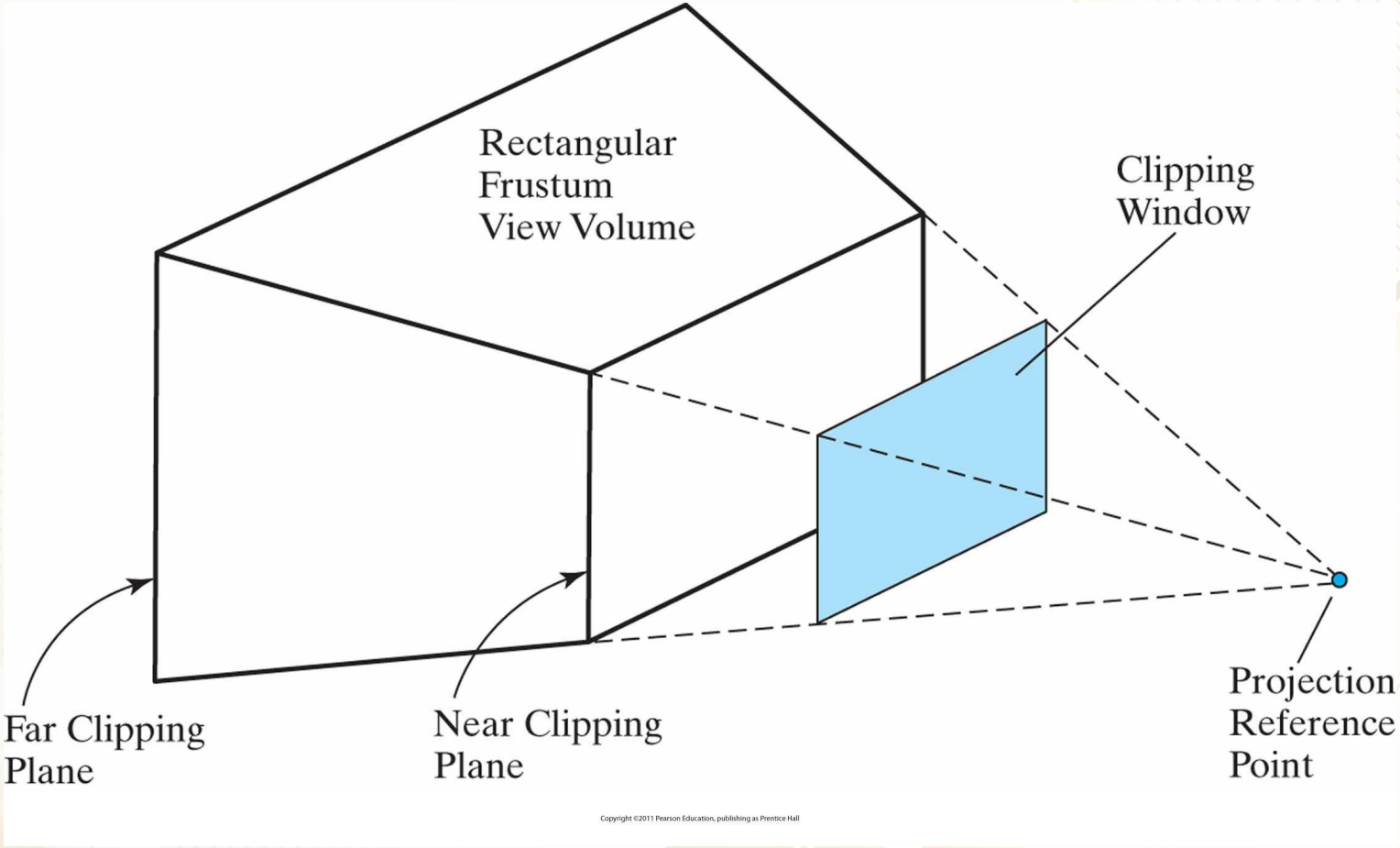


Figure 10-39 A perspective-projection frustum view volume with the view plane "in front" of the near clipping plane.



By adding near and far clipping planes that are perpendicular to the z_{view} axis(and parallel to the viewplane), we chop off parts of the infinite, perspective projection view volume to form a **truncated pyramid**, or frustum, view volume.

illustrates the shape of a finite, perspective-projection view volume with a viewplane that is placed between the near clipping plane and the projection reference point.

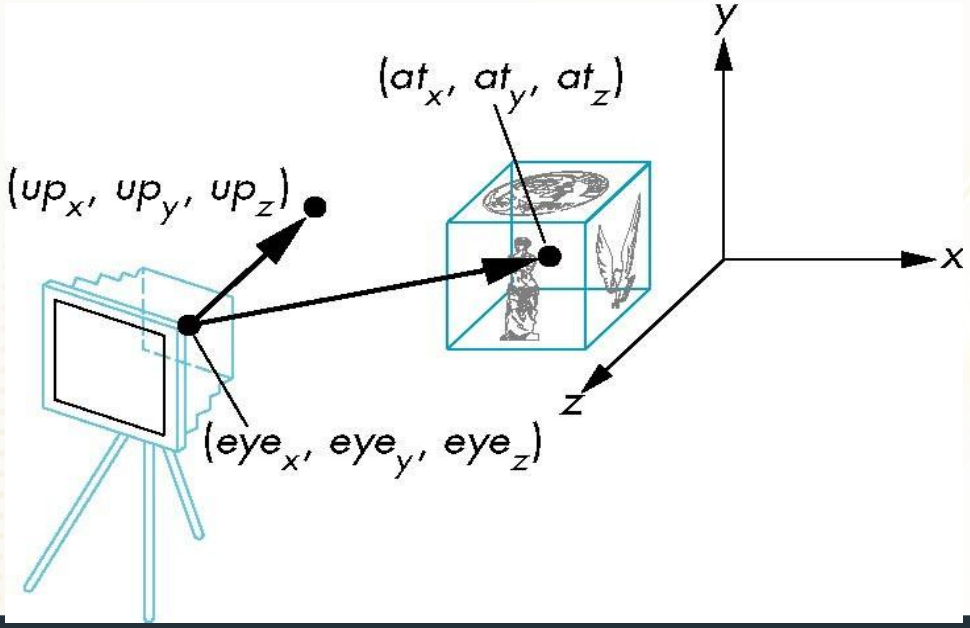
OpenGL Three-Dimensional Viewing Functions

OpenGL Viewing-Transformation Function

- Set the modelview mode
  - glMatrixMode (GL_MODELVIEW);
- Viewing parameters are specified
  - gluLookAt (x0, y0, z0, xref, yref, zref, Vx, Vy, Vz);
    - P_0 = (x0, y0, z0), P_{ref}=(xref, yref, zref), V = (Vx, Vy, Vz)
    - Since viewing direction is along the $-z_{view}$ axis, P_{ref}=look at point
    - $N = P_0 - P_{ref}$
    - V is perpendicular to N
  - Default values: P_0 = (0, 0, 0), P_{ref}=(0, 0, -1), V = (0, 1, 0)

gluLookAt

```
glLookAt(eyex,eyey,eyez,
                atx,aty,atz,
                    upx,upy,upz)
```



OpenGL – Orthogonal projection Function

Projection matrices are stored in OpenGL projection mode

- Set up a projection-transformation matrix
  - glMatrixMode (GL_PROJECTION);
- **Orthogonal projection**
  - glOrtho (xwmin, xwmax, ywmin, ywmax, dnear, dfar);
  - All parameters are double precision, floating point number.

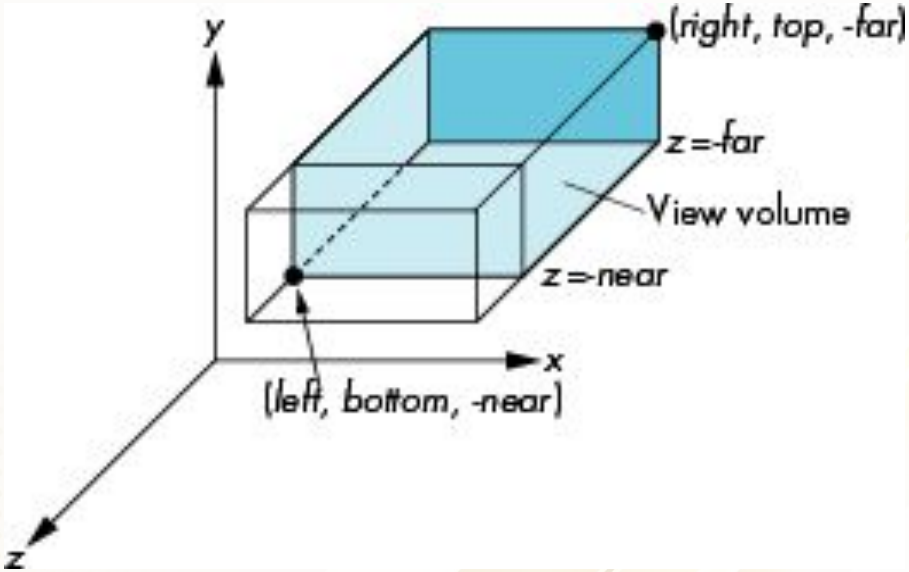
Parameters d_{near} and d_{far} denote distances in the negative z_{view} direction from the viewing-coordinate origin.

if $d_{far} = 55.0$, then the far clipping plane is at the coordinate position $z_{far} = -55.0$. A negative value for either parameter denotes a distance "behind" the viewing origin, along the positive z_{view} axis.

We can assign any values (positive, negative, or zero) to these parameters, so long as $d_{near} < d_{far}$.

OpenGL Orthogonal Viewing

```
glOrtho(left,right,bottom,top,near,far)
```



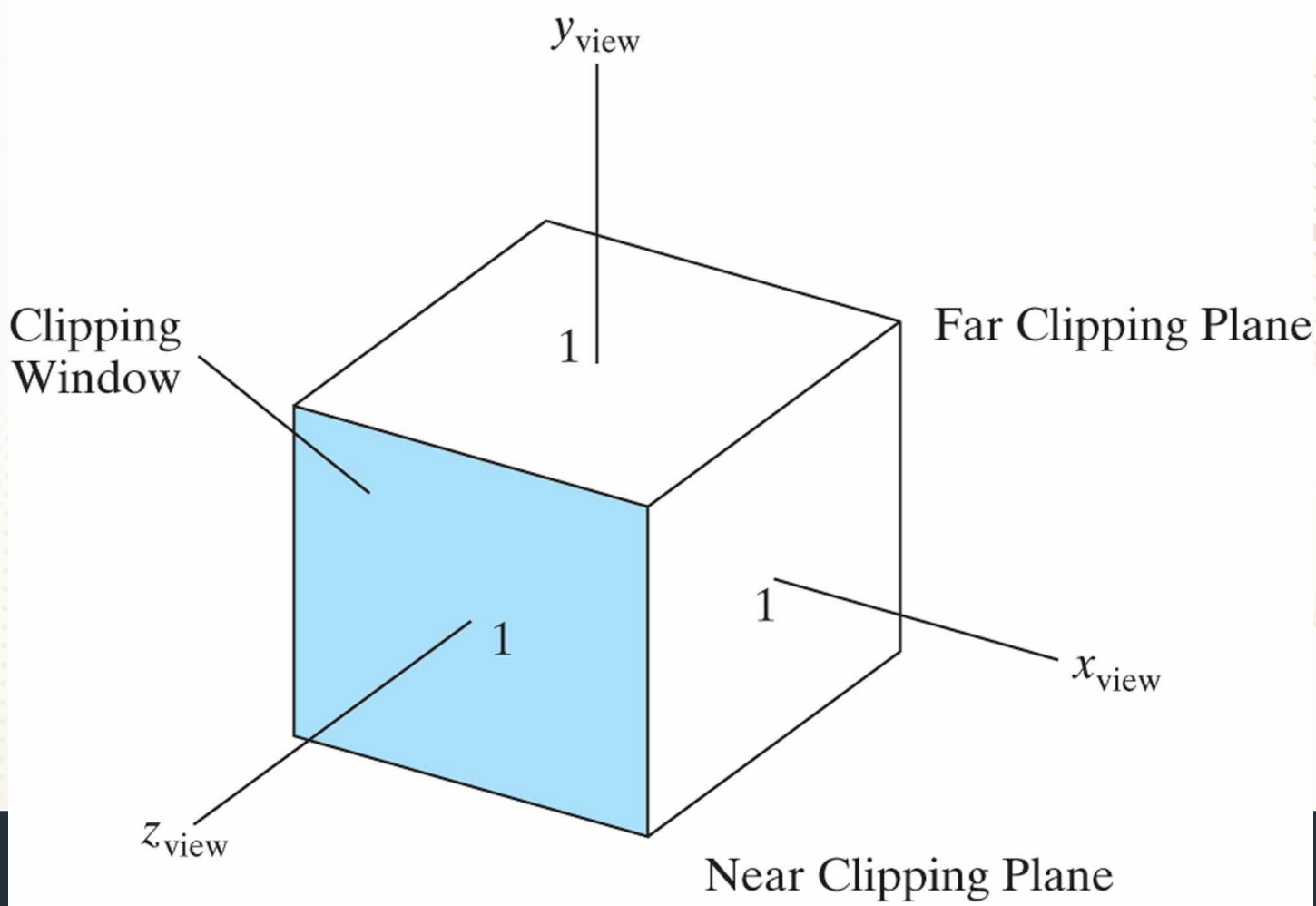
near and far measured from camera

Default

glOrtho (-1.0, 1.0, -1.0, 1.0, -1.0, 1.0);

Default view volume is a symmetric normalized cube with $z_{near} = 1.0$ and $z_{far} = -1.0$

Figure 10-47 Default orthogonal-projection view volume. Coordinate extents for this symmetric cube are from −1 to +1 in each direction. The near clipping plane is at $z_{near} = 1$, and the far clipping plane is at $z_{far} = -1$.



OpenGL Symmetric perspective-projection Function

Two functions are available

- One used to generate a symmetric frustum view volume
- Other function can be used for either a symmetric-perspective projection or an oblique-perspective projection.

For both functions, prp is viewing coordinate origin and near clipping plane is view plane.

gluPerspective (theta, aspect, dnear, dfar);

Feld-of-view angle- angle between the top and bottom clipping planes.

This angle -0◦ to 180◦.

Parameter aspect is assigned a value for the aspect ratio (width/height) of the clipping window.

Near and far clipping planes must always be somewhere along the negative z_{view} axis. Neither can be "behind" the viewing position.

Both d_{near} and d_{far} must be assigned positive numerical values, and the positions of the near and far planes are calculated as $z_{near} = -d_{near}$ and $z_{far} = -d_{far}$.

OpenGL General Perspective-Projection Function

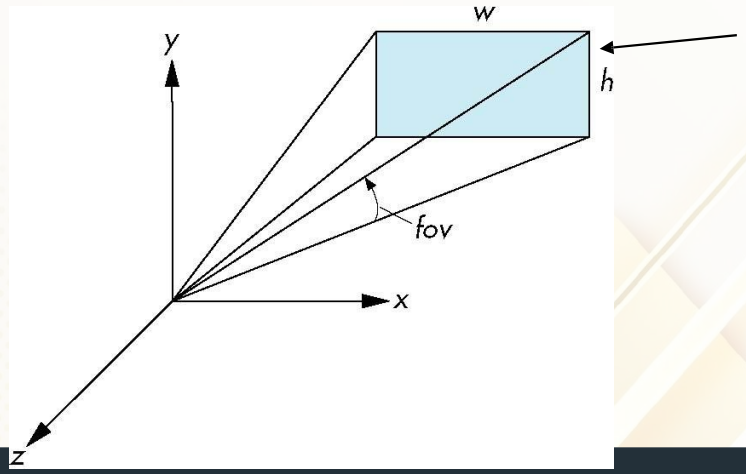
- Symmetric frustum view volume or an oblique frustum view volume

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

Using Field of View

- With `glFrustum` it is often difficult to get the desired view
- `gluPerpective(fovy, aspect, near, far)`

often provides a better interface

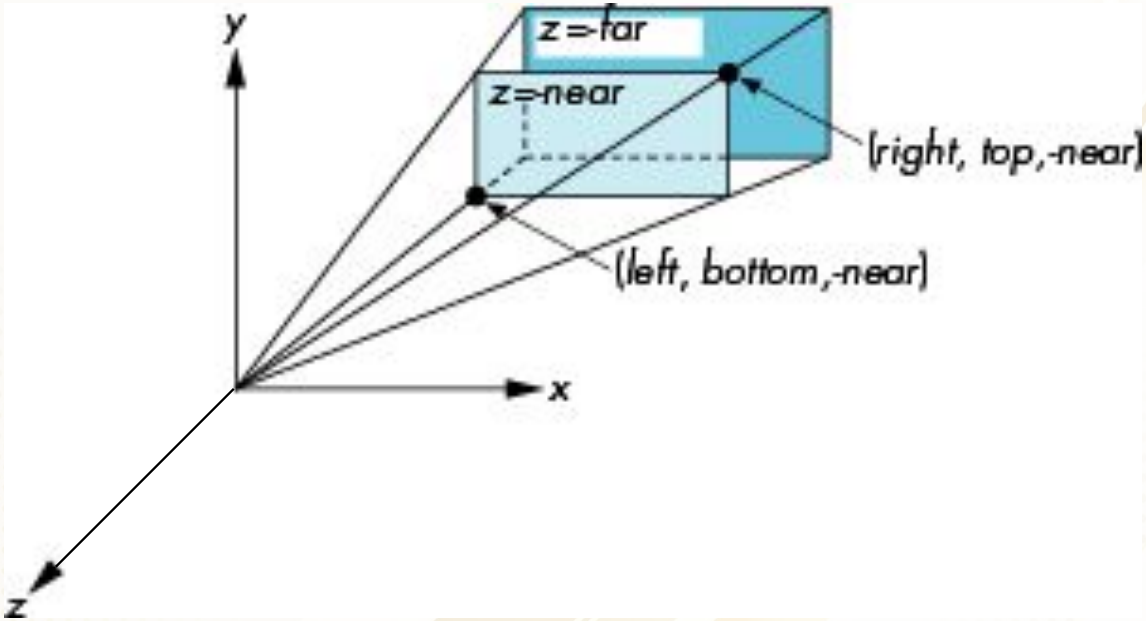


front plane

`aspect = w/h`

OpenGL Perspective

```
glFrustum(left,right,bottom,top,near,far)
```



OpenGL Viewports and Display Windows

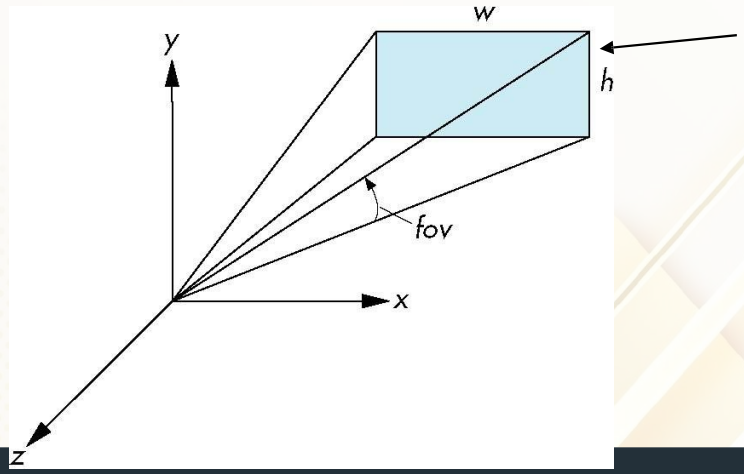
A rectangular viewport is defined with the following OpenGL function

- glViewport (xvmin, yvmin, vpWidth, vpHeight);

Using Field of View

- With `glFrustum` it is often difficult to get the desired view
- `gluPerpective(fovy, aspect, near, far)`

often provides a better interface



front plane

`aspect = w/h`

Visible surface detection (or) Hidden surface elimination

Classification of visible surface detection algorithms

Object-space methods vs. ***Image-space methods***

Object definition directly vs. their projected images

Most visible-surface algorithms use image-space methods

Object-space can be used effectively in some cases

Ex) Line-display algorithms

Object-space methods

Compares objects and parts of objects to each other

Image-space methods

Point by point at each pixel position on the projection plane

Back-face detection

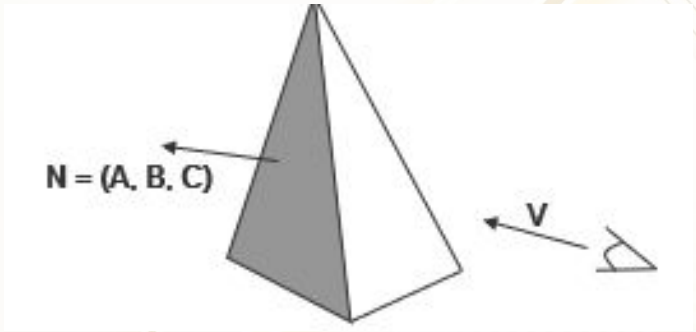
Fast and simple object-space method

A point (x, y, z) is "behind" a polygon surface with plane parameters A, B, C, and D if

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

The polygon is a back face if

$$V \cdot N > 0$$



V is a vector in the viewing direction from the eye(camera)

N is the normal vector to a polygon surface

Depth Buffer Method

Commonly used image-space approach

Compares depths of each pixel on the projection plane

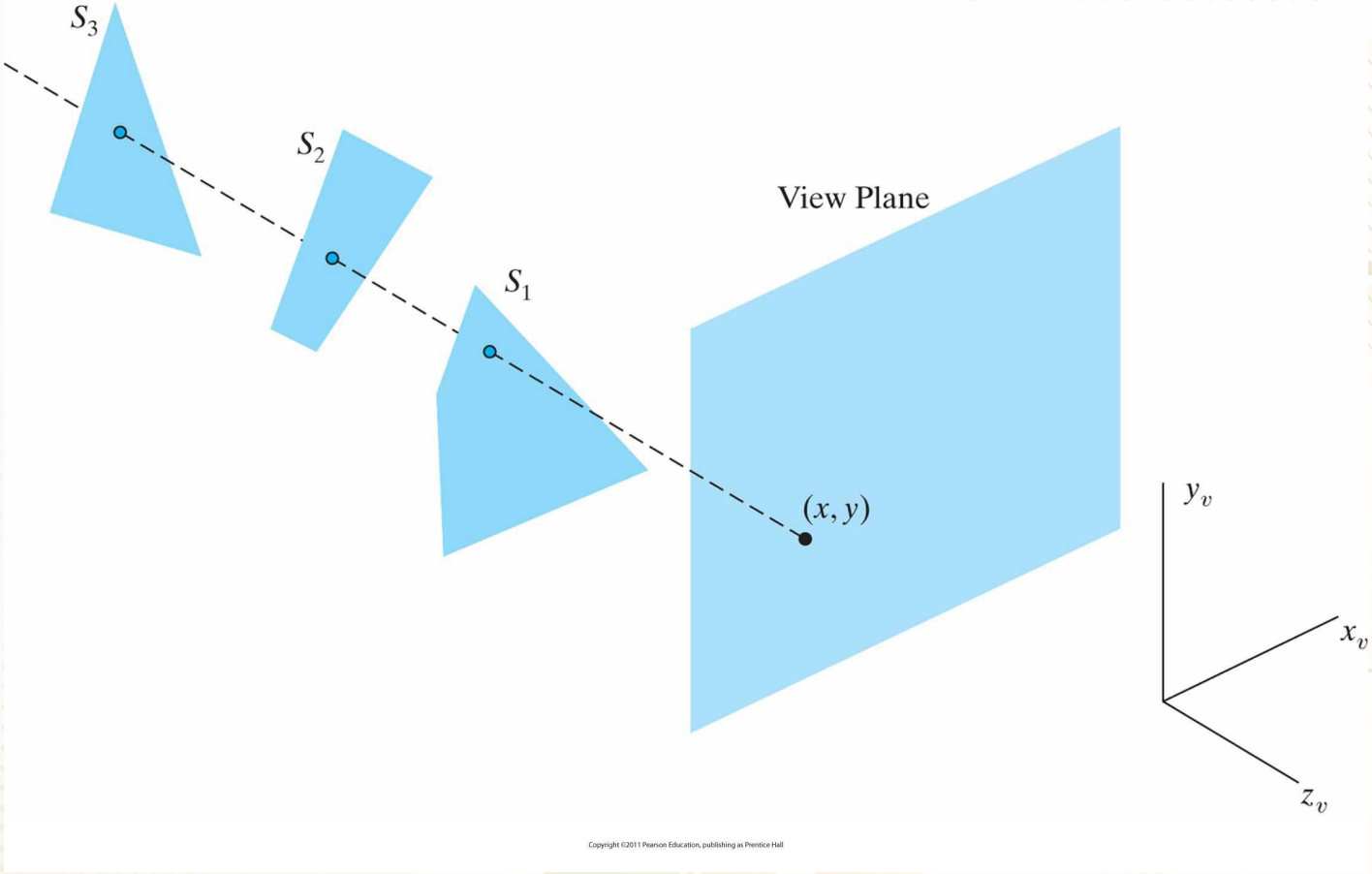
- Referred to as the ***z-buffer*** method

Each surface is processed separately

Usually applied to scenes of polygonal surfaces

- Depth values can be computed very quickly
- Easy to implement.

Figure 16-4 Three surfaces overlapping pixel position (x, y) on the view plane. The visible surface, S_1, has the smallest depth value.



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.

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

- Depth buffer
  - Store depth values for each (x, y) position
  - All positions are initialized to maximum depth
    - Usually 1 – most distant depth from the viewplane
- frame buffer
  - Stores the color values for each position
  - All positions are initialized to the background intensity.

Each surface is 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.

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

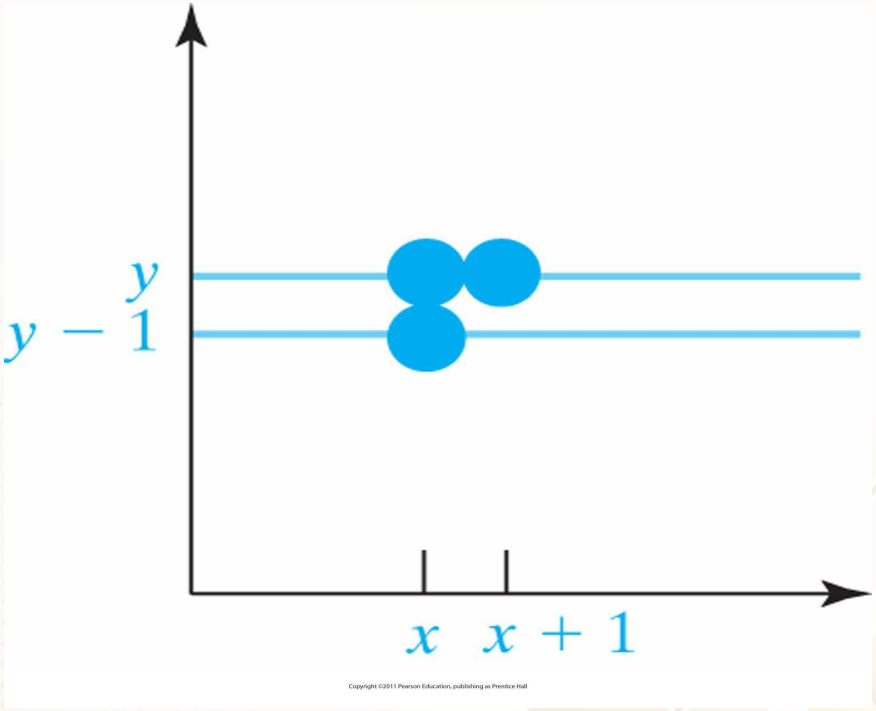
2. Process each polygon in a scene, one at a time.

- For each projected (x,y) pixel position of a polygon, calculate the depth z
- If z < depthBuff(x, y), compute the surface color at that position and set

  depthBuff(x, y) = z, frameBuff(x, y) = surfColor(x,y)

  Depth buffer contains depth values for the visible surfaces and the frame buffer contains the corresponding color values for those surfaces.

Figure 16-5 From position (x, y) on a scan line, the next position across the line has coordinates $(x + 1, y)$, and the position immediately below on the next line has coordinates $(x, y - 1)$.



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 d
z' of the next position (x+1, y) along the scan line is obtained from Eq

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

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

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.

Figure 16-6 Scan lines intersecting a polygon surface.

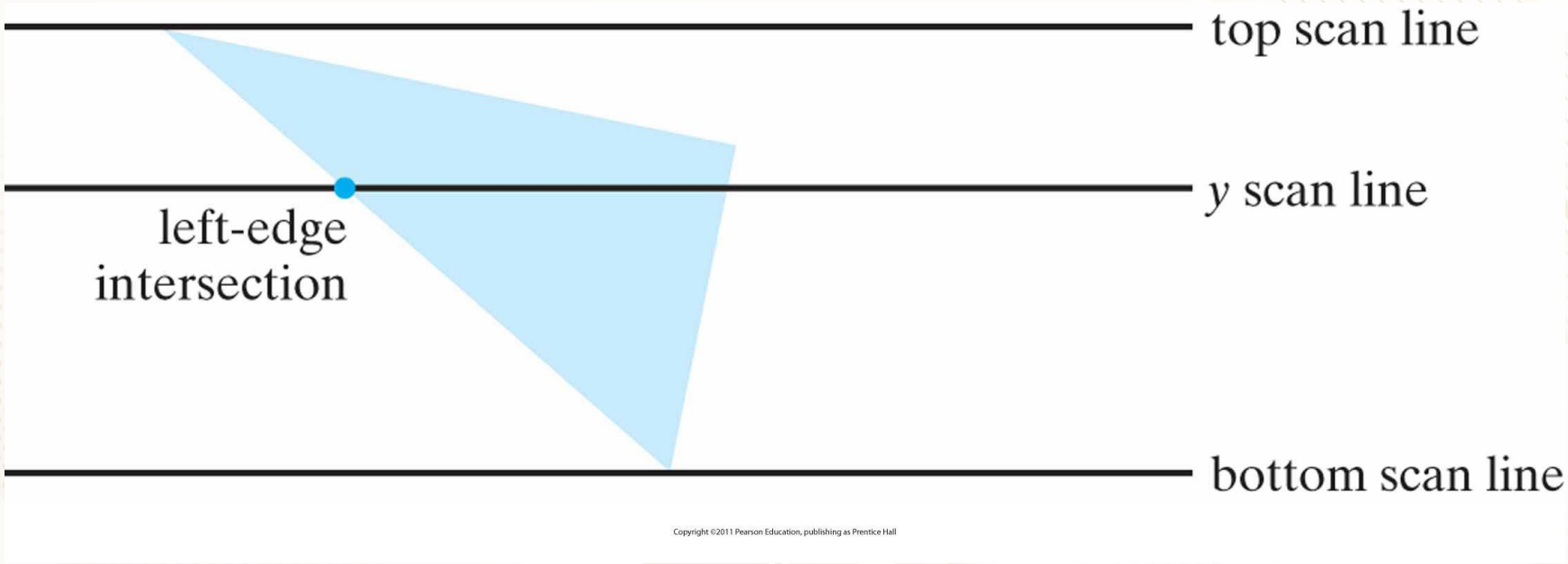
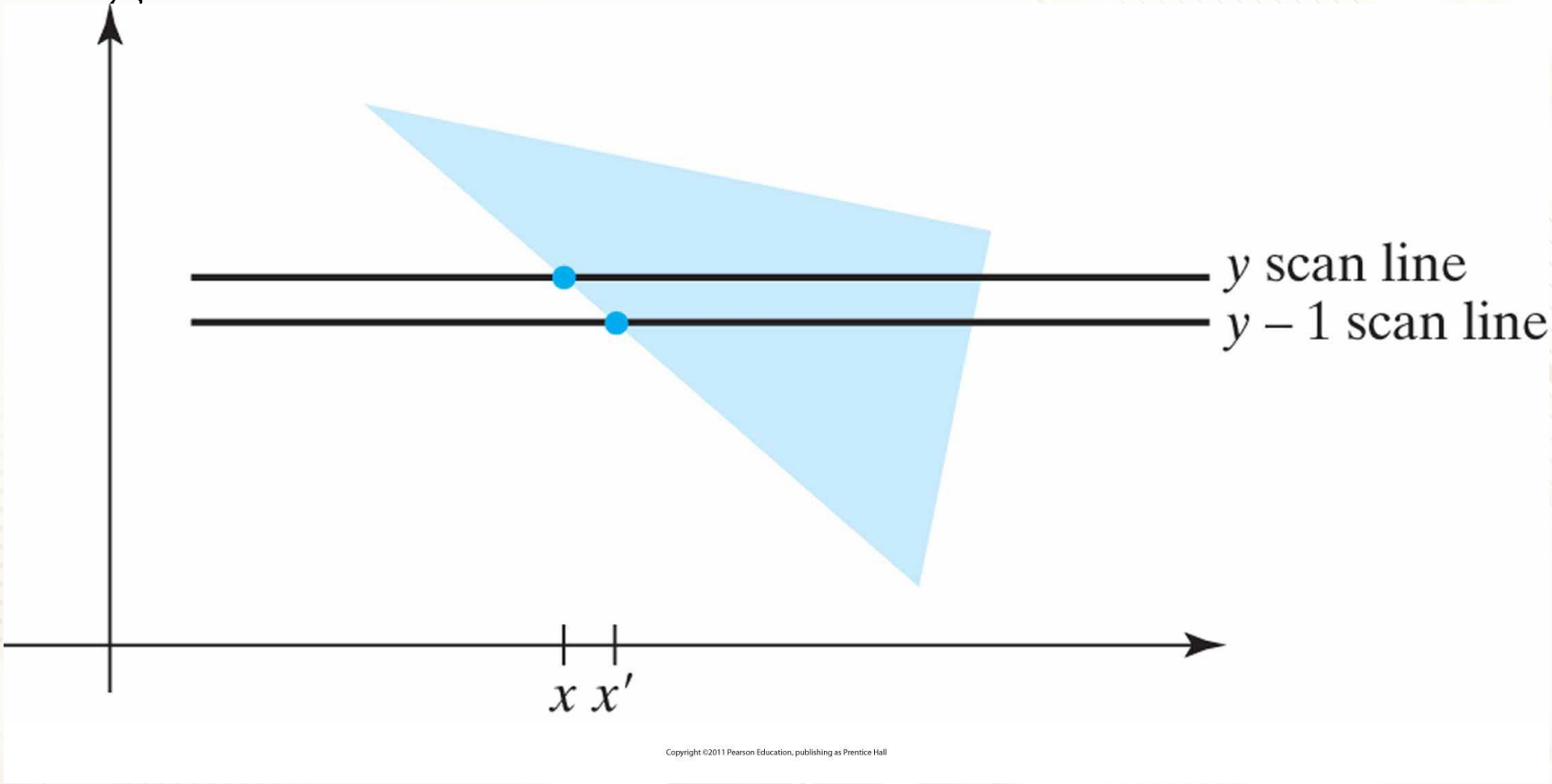


Figure 16-7 Intersection positions on successive scan lines along a left polygon edge.



Starting at a top vertex of the polygon.

Recursively calculate the x-coordinate values down a left edge of the polygon.

The x value for the beginning position on each scanline 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. Depth values down this edge are obtained recursively as

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

For a vertical edge, the slope is infinite and the recursive calculations reduce to

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

OpenGL Depth Buffer Functions

- Set display Mode
  glutDisplayMode(GLUT_DOUBLE | GLUT_RGB | *GLUT_DEPTH*);
- Clear screen and depth buffer every time in the display function
  glClear(GL_COLOR_BUFFER_BIT | *GL_DEPTH_BUFFER_BIT*);

Sets all depth-buffer values to the max value 1.0 by default.

The openGL Depth buffer visibility detection routines are activated with **glEnable(*GL_DEPTH_TEST*);**
glDisable(*GL_DEPTH_TEST*); -deactivate