Computer Graphics

Chapter 10

Three-Dimensional Viewing

Chapter 10
Three-Dimensional Viewing

Part I.

Overview of 3D Viewing Concept

3D Viewing Pipeline vs. OpenGL Pipeline

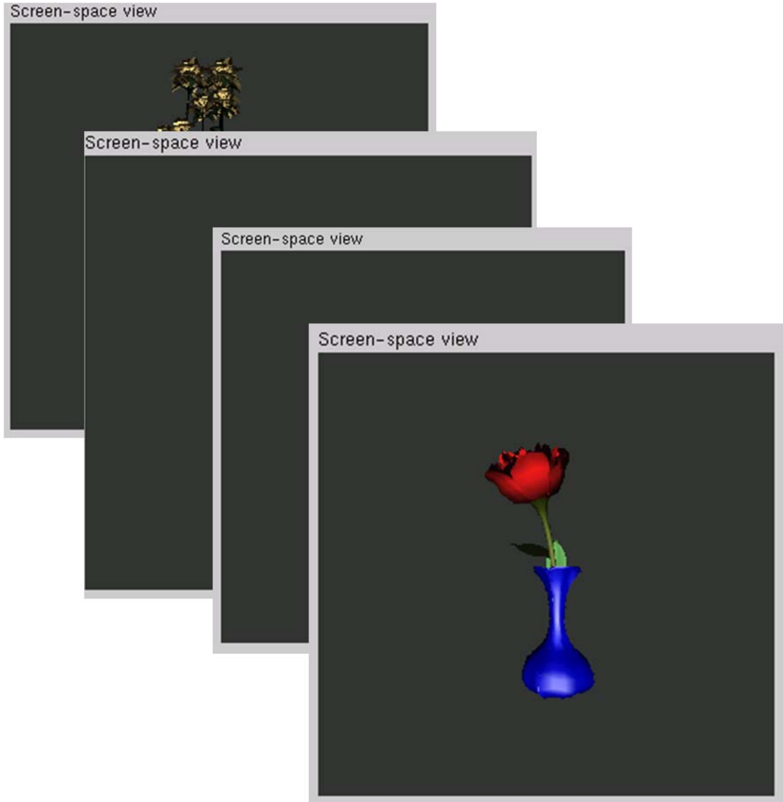
3D Viewing-Coordinate Parameters

Projection Transformations

Viewport Transformation and 3D Screen Coordinates

Overview of 3D Viewing Concept

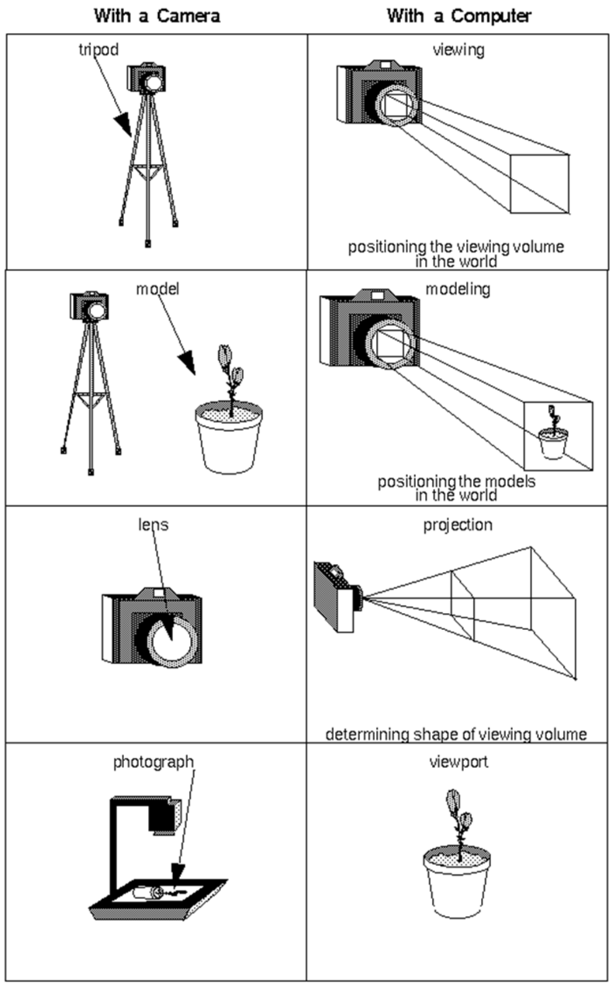
- How to construct 3D scenes in computers?



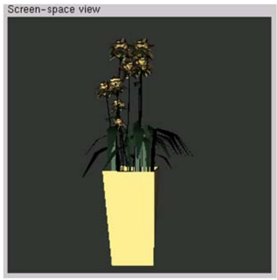
- How to take a picture by camera?

Overview of 3D Viewing Concept

- Camera analog

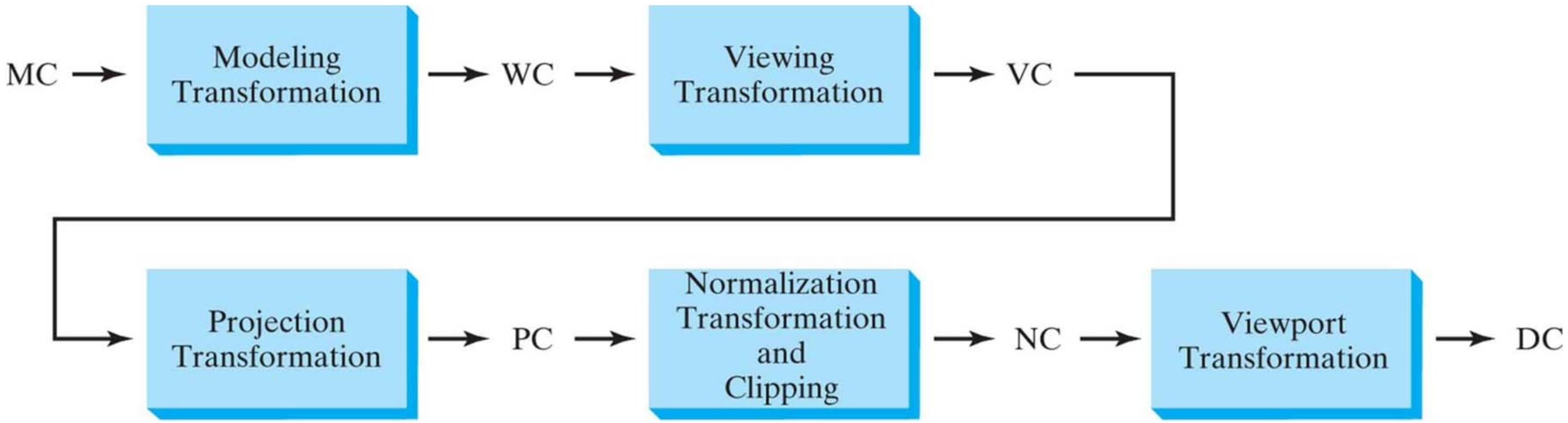


- Choose the position of the camera and pointing it at the scene (viewing transformation).
- Arranging the scene to be photographed into the desired composition (modeling transformation).
- Choosing a camera lens or adjusting the zoom (projection transformation).
- Determining how large you want the final photograph to be (viewport transformation).



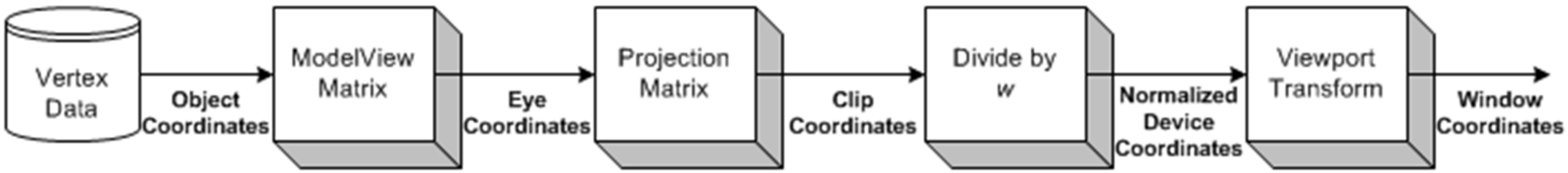
(From: the red book)

3D Viewing Pipeline vs. OpenGL Pipeline

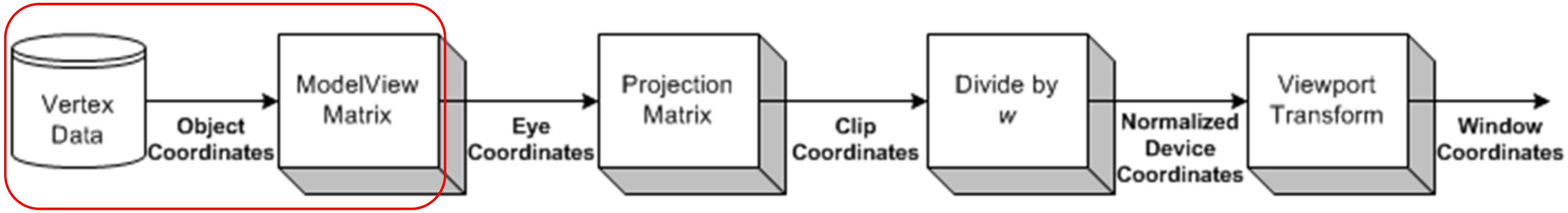


In OpenGL pipeline, geometric data such as vertex positions and normal vectors are transformed via **Vertex Operation** and **Primitive Assembly Operation** before rasterization process.

OpenGL vertex transformation:

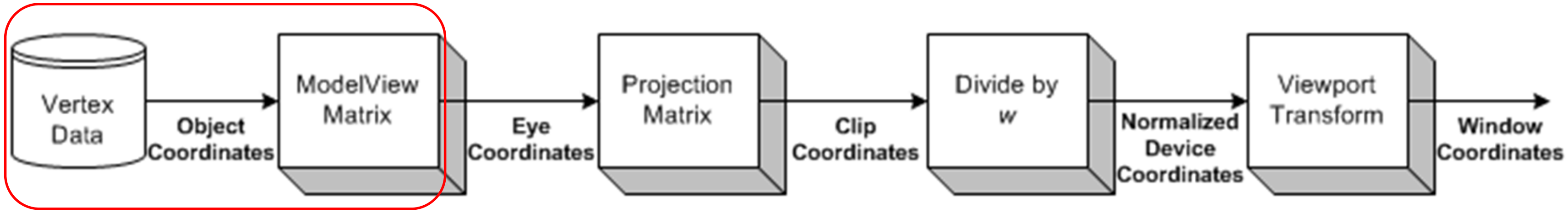


OpenGL Vertex Transformation

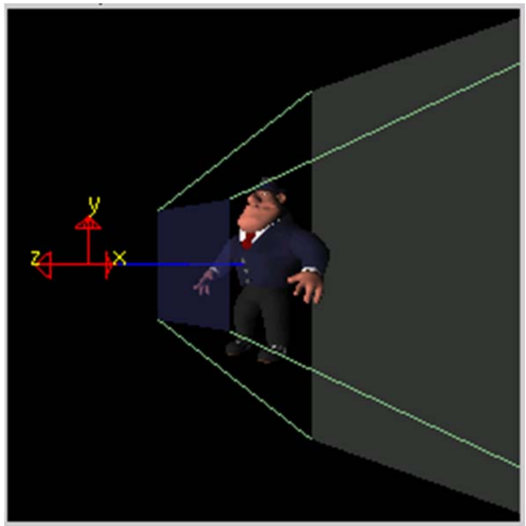


- Object coordinates
  - The local coordinate system of objects and represent the initial position and orientation of objects before any transform is applied.
  - Specify them with glVertex*() or glVertexPointer().
  - To transform objects, use glRotatef(), glTranslatef(), glScalef().

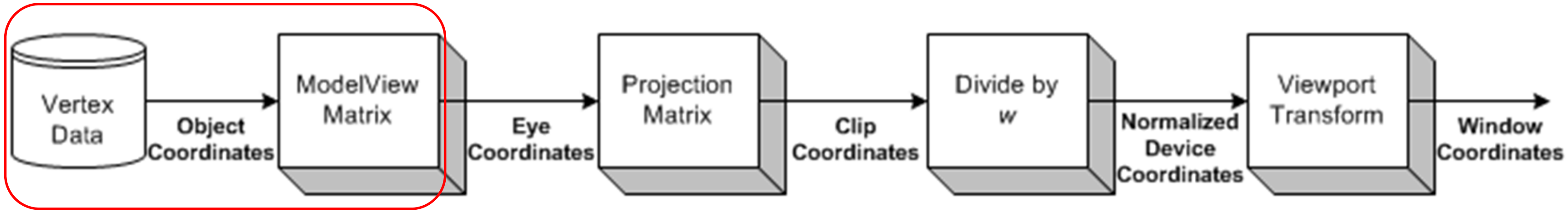
OpenGL Vertex Transformation



- Eye coordinates
  - Using GL_MODELVIEW matrix to transform objects from "object space" to "eye space". (multiply GL_MODELVIEW matrix and object coordinates)
  - GL_MODELVIEW matrix is a combination of Model and View matrices ($M_{view} \cdot M_{model}$).
    - M_{model} is to construct objects from "object space/local space" to "world space".
    - M_{view} is to convert objects from "world space" to "eye space" (camera).



OpenGL Vertex Transformation

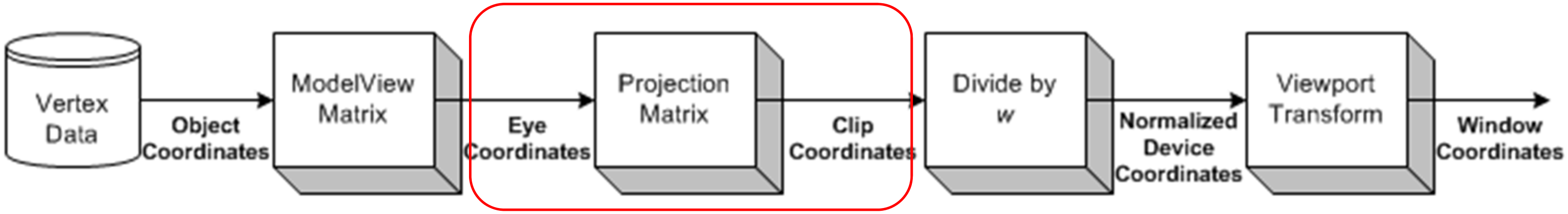


- Eye coordinates (cont.)

$$\begin{pmatrix} x_{eye} \\ y_{eye} \\ z_{eye} \\ w_{eye} \end{pmatrix} = M_{modelView} \cdot \begin{pmatrix} x_{obj} \\ y_{obj} \\ z_{obj} \\ w_{obj} \end{pmatrix} = M_{view} \cdot M_{model} \cdot \begin{pmatrix} x_{obj} \\ y_{obj} \\ z_{obj} \\ w_{obj} \end{pmatrix}$$

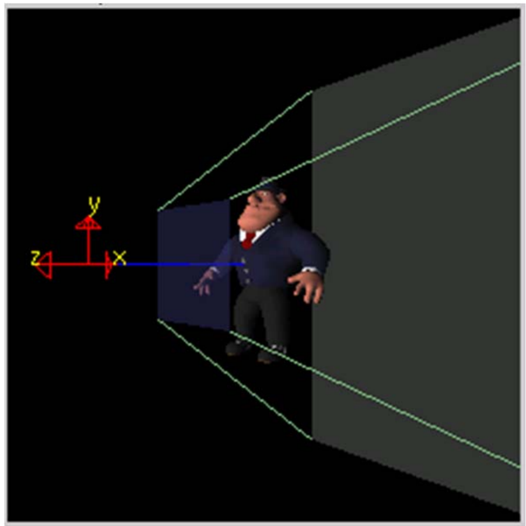
  - Note: by default, OpenGL defines that the camera is always located at (0, 0, 0) and facing to -Z axis in the **eye space coordinates**.

OpenGL Vertex Transformation



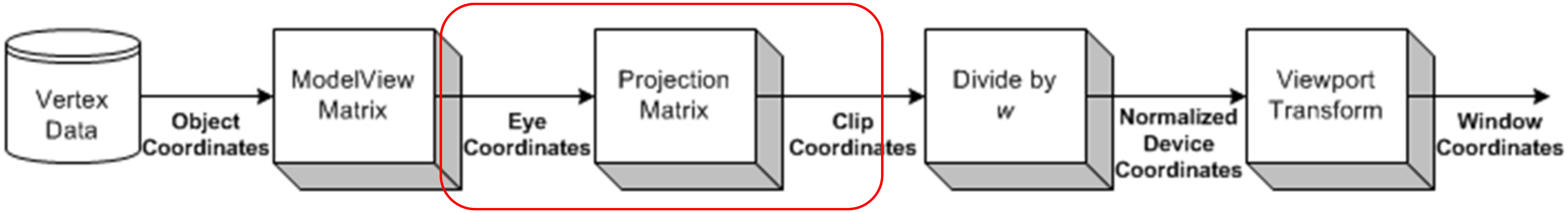
- Clip coordinates
  - Apply the **Projection matrix** to transform objects from Eye Coordinates to Clip Coordinates.

$$\begin{pmatrix} x_{clip} \\ y_{clip} \\ z_{clip} \\ w_{clip} \end{pmatrix} = M_{projection} \cdot \begin{pmatrix} x_{eye} \\ y_{eye} \\ z_{eye} \\ w_{eye} \end{pmatrix}$$

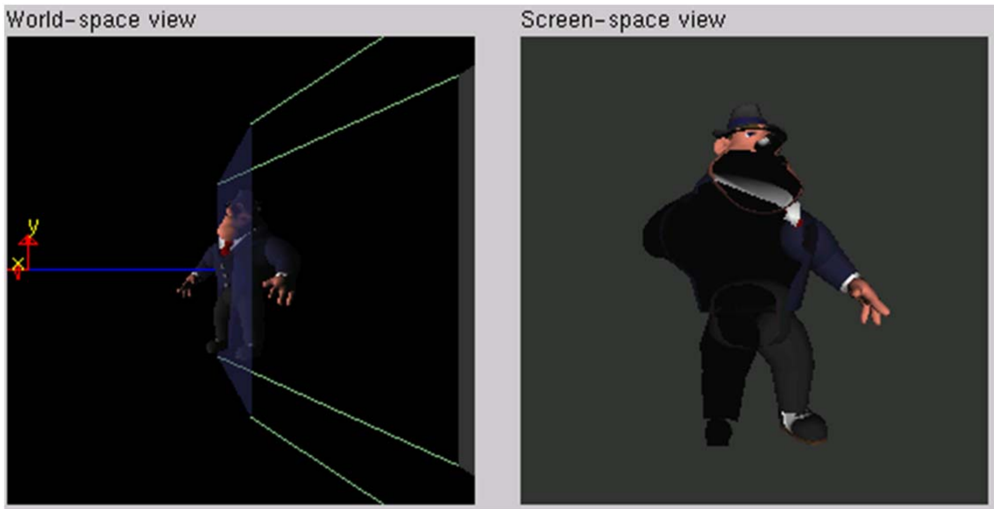


  - Viewing volume
    - How objects are projected onto screen (perspective or parallel(orthogonal));
    - Which objects or portions of objects are clipped out of the final image.

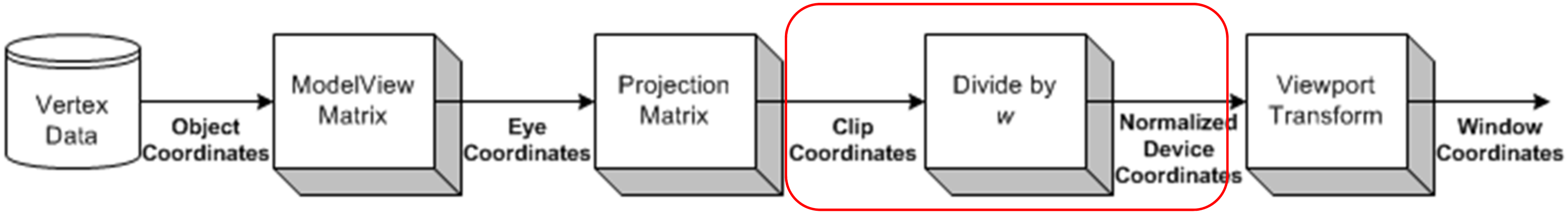
OpenGL Vertex Transformation



- Clip coordinates (cont.)
  - Objects are clipped out from the **viewing volume**



OpenGL Vertex Transformation

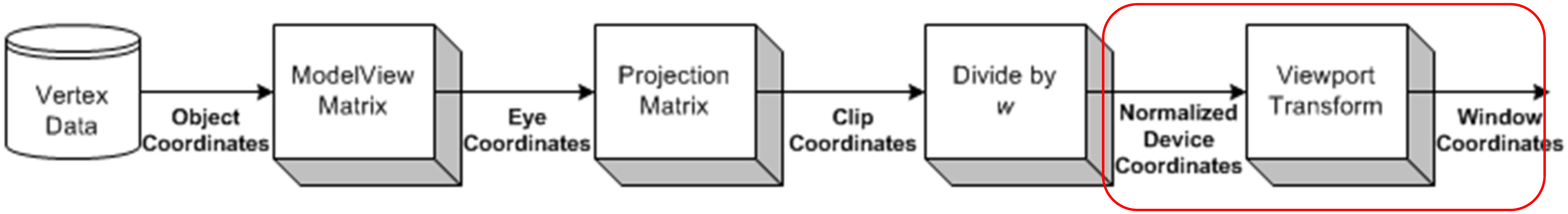


- Normalized Device Coordinates (NDC)
  - Transform the values into the range of [-1, 1] in all three axes.
  - In OpenGL, it is implemented by the **Perspective Division** on the Clip Coordinates.

$$\begin{pmatrix} x_{ndc} \\ y_{ndc} \\ z_{ndc} \end{pmatrix} = \begin{pmatrix} x_{clip}/w_{clip} \\ y_{clip}/w_{clip} \\ z_{clip}/w_{clip} \end{pmatrix}$$

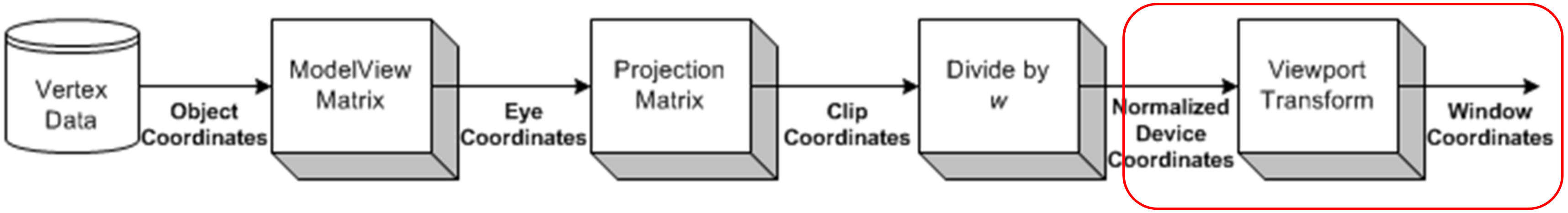
(That divides the Clip Coordinates by w_{clip}.)

OpenGL Vertex Transformation



- Window Coordinates
  - Result from scaling and translating Normalized Device Coordinates by the **viewport** transformation.
  - They are controlled by the parameters of the viewport you defined
    - **glViewport()**: to define the rectangle of the rendering area where the final image is mapped.
    - **glDepthRange()**: to determine the *z* value of the window coordinates.

OpenGL Vertex Transformation



- Window Coordinates (cont.)
  - **glViewport(x, y, w, h);** **glDepthRange(n, f);** **n**: near, **f**: far
- NDC->WC (Viewport)

[-1,1; -1,1; -1,1] => [x, x+w; y, y+h; n,f]

$$\begin{pmatrix} x_w \\ y_w \\ z_w \end{pmatrix} = \begin{pmatrix} \frac{w}{2} x_{ndc} + (x + \frac{w}{2}) \\ \frac{h}{2} y_{ndc} + (y + \frac{h}{2}) \\ \frac{f-n}{2} z_{ndc} + \frac{f+n}{2} \end{pmatrix}$$

Chapter 10 Three-Dimensional Viewing (OpenGL functions)

Part I.

Overview of 3D Viewing Concept

3D Viewing Pipeline vs. OpenGL Pipeline

3D Viewing-Coordinate Parameters

Projection Transformations

Viewport Transformation and 3D Screen Coordinates

Coordinate reference for “camera”

- To set up the viewing coordinate reference (or camera)
  - Position and orientation of a view plane (or projection plane)
  - Objects are transferred to the viewing reference coordinates and projected onto the view plane

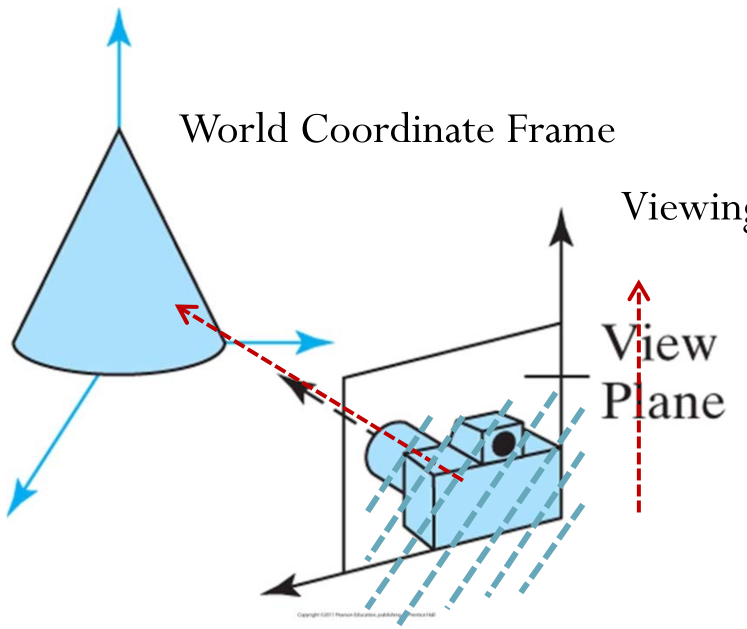
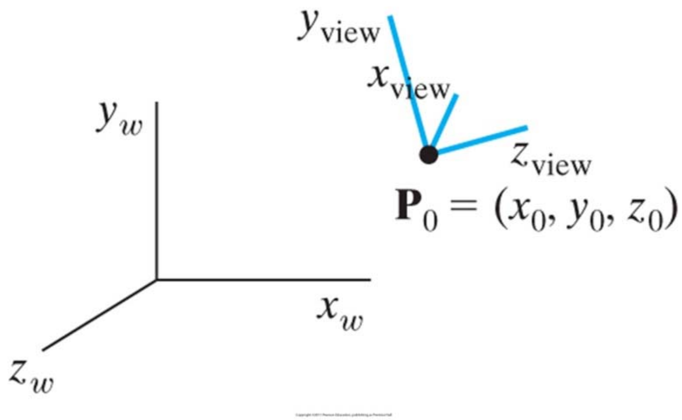


FIGURE 10-1 Coordinate reference for obtaining a selected view of a three-dimensional scene.

3D Viewing-Coordinate Parameters

- Establish a 3D viewing reference frame
  - Right-handed



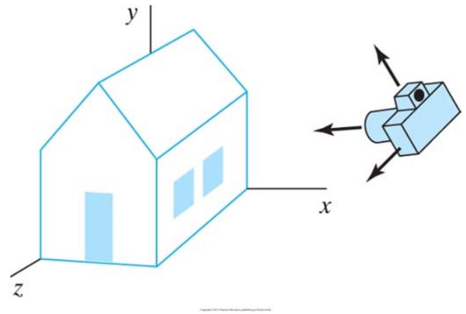


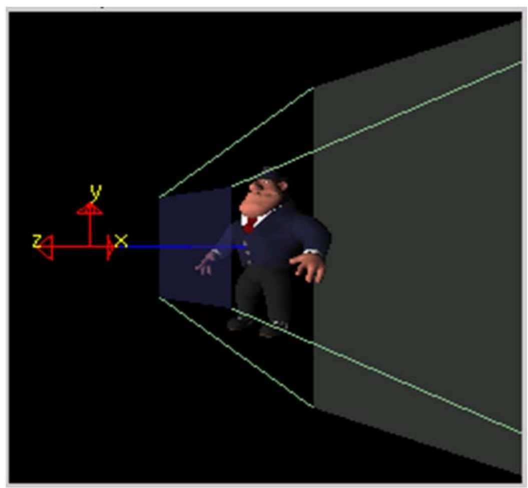
FIGURE 10-7 A right-handed viewing-coordinate system, with axes x_{view}, y_{view}, z_{view}, relative to a right-handed world-coordinate frame

a. The **viewing origin**

- Define the view point or viewing position (sometimes is referred to as the eye position or the camera position)

b. $\mathbf{y_{view}}$ -- view-up vector $\mathbf{V}$

- Defines y_{view} direction

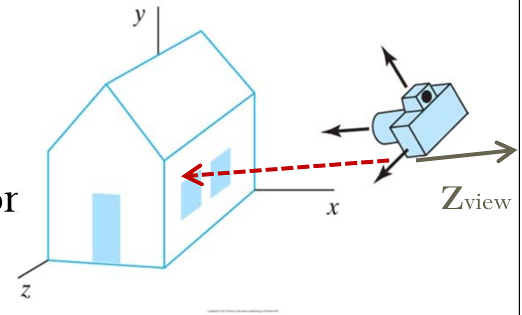


3D Viewing-Coordinate Parameters

- Viewing direction and view plane

  c. $\mathbf{z}_{\mathbf{view}}$: viewing direction

  - Along the z_{view} axis, often in the negative z_{view} direction



  d. The view plane (also called projection plane)

  - Perpendicular to $\mathbf{z}_{\mathbf{view}}$ axis
  - The orientation of the view plane can be defined by a view-plane **normal vector N**
  - The different position of the view-plane along the $\mathbf{z}_{\mathbf{view}}$ axis

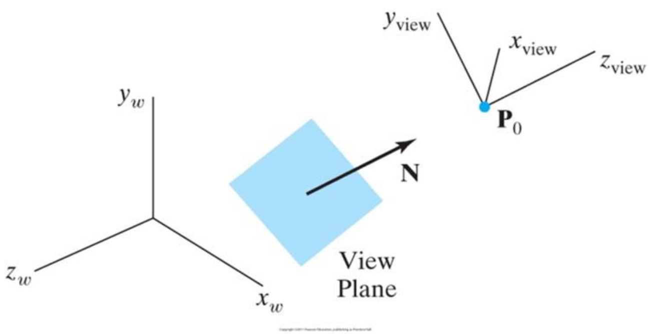


FIGURE 10-8 Orientation of the view plane and view-plane normal vector **N**

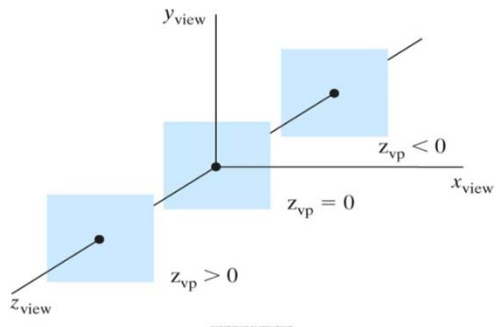
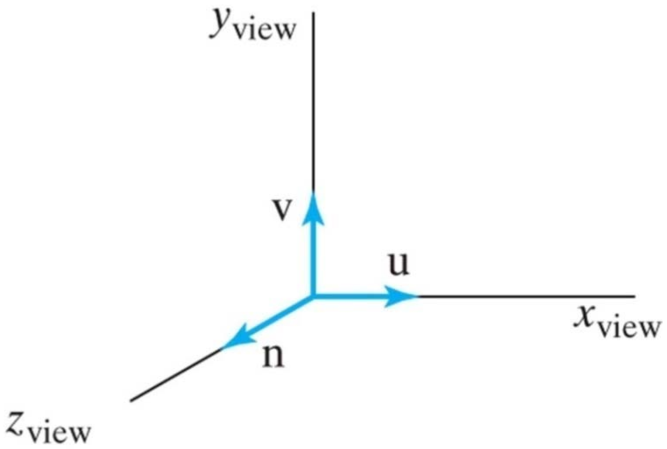


FIGURE 10-9 Three possible positions for the **view plane** along the $\mathbf{z}_{\mathbf{view}}$ axis

3D Viewing-Coordinate Parameters

- The **uvn** *Viewing-Coordinate Reference Frame* (*Viewing Coordinate System*)
  - Direction of $\mathbf{z}_{\mathbf{view}}$ axis: the view-plane normal vector **N**;
  - Direction of $\mathbf{y}_{\mathbf{view}}$ axis: the view-up vector **V**;
  - Direction of $\mathbf{x}_{\mathbf{view}}$ axis: taking the vector cross product of **V** and **N** to get **U**.

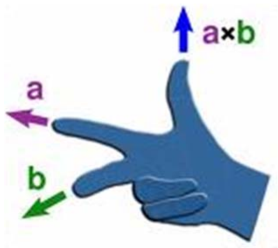


$$\mathbf{n} = \frac{\mathbf{N}}{|\mathbf{N}|} = (n_x, n_y, n_z)$$

$$\mathbf{u} = \frac{\mathbf{V} \times \mathbf{n}}{|\mathbf{V} \times \mathbf{n}|} = (u_x, u_y, u_z) \qquad (10\text{-}1)$$

$$\mathbf{v} = \mathbf{n} \times \mathbf{u} = (v_x, v_y, v_z)$$

FIGURE 10-12 A right-handed viewing system defined with unit vectors u, v, and n.



Chapter 10
Three-Dimensional Viewing

Part I.

Overview of 3D Viewing Concept

3D Viewing Pipeline vs. OpenGL Pipeline

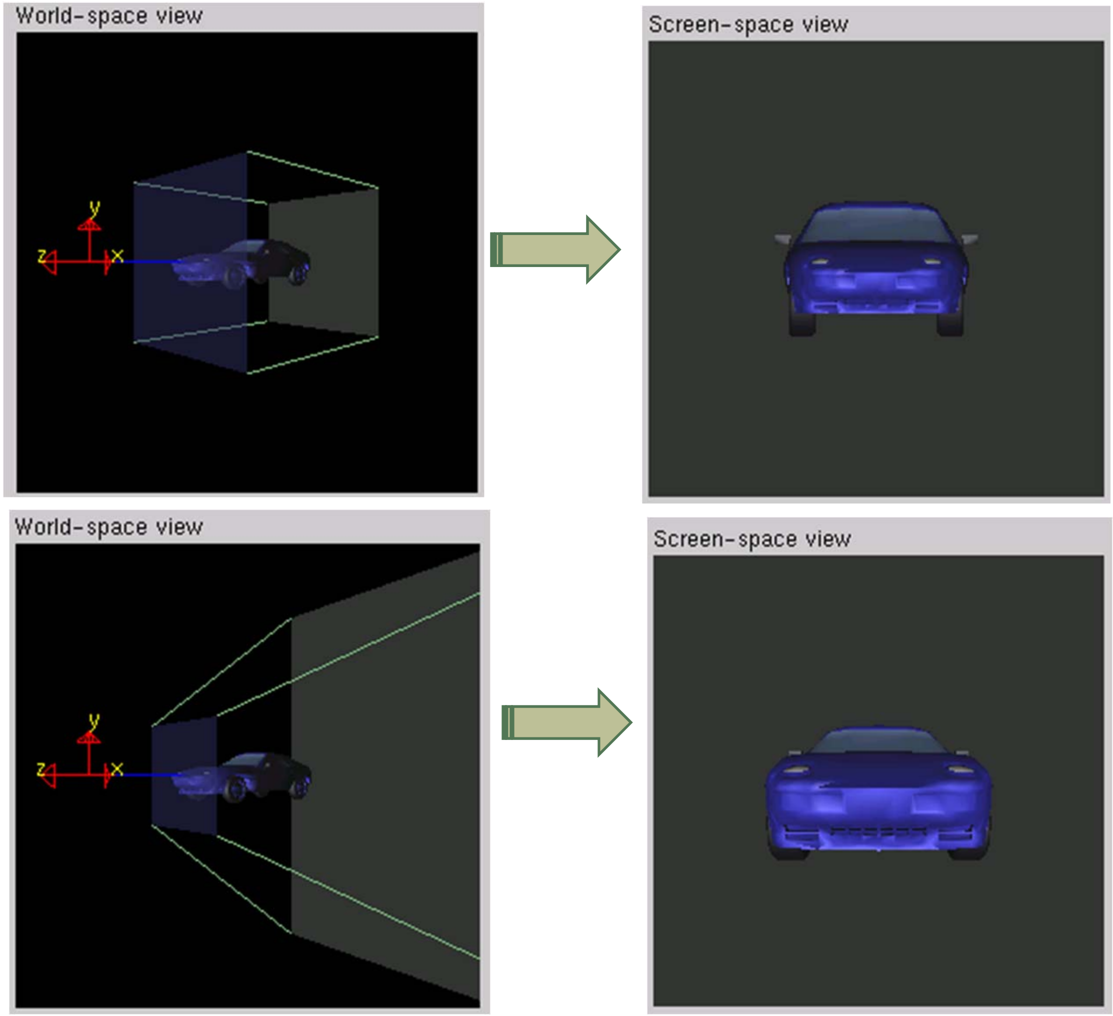
3D Viewing-Coordinate Parameters

Projection Transformations

Viewport Transformation and 3D Screen Coordinates

Projection Transformations

- Objects are projected to the view plane.



Orthogonal Projection (a special case of Parallel Proj.)

- Coordinates positions are transferred to the view plane along parallel lines

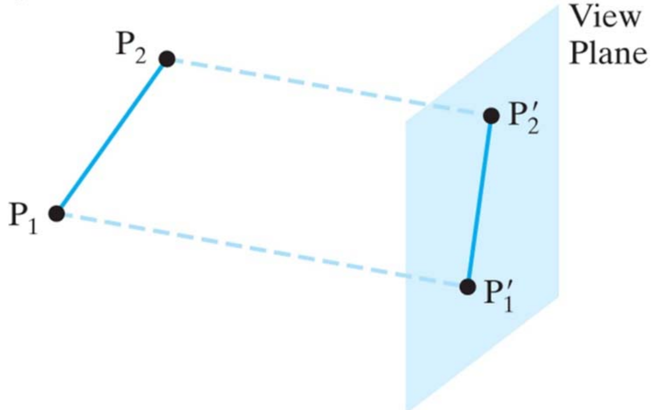
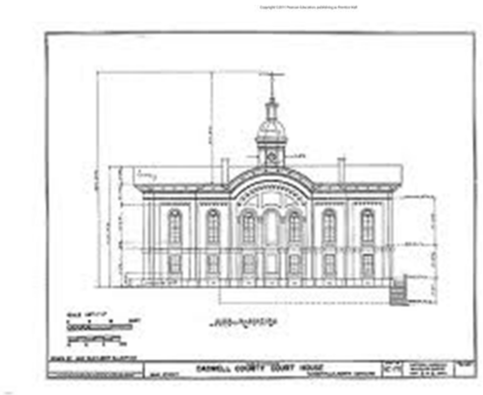
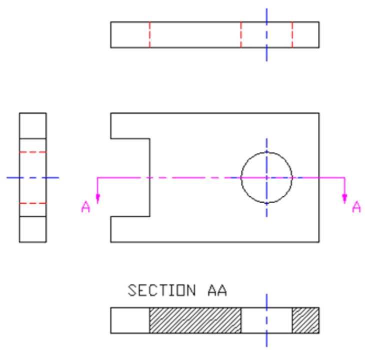
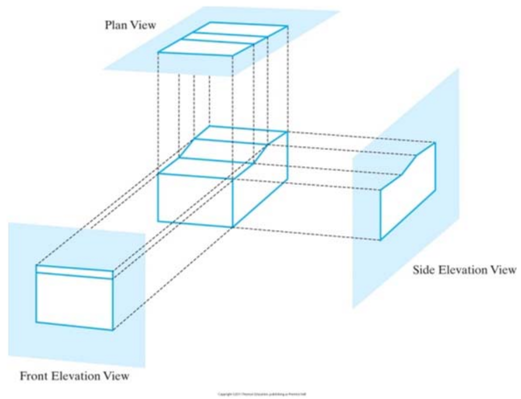


FIGURE 10-15 Parallel projection of a line segment onto a view plane.



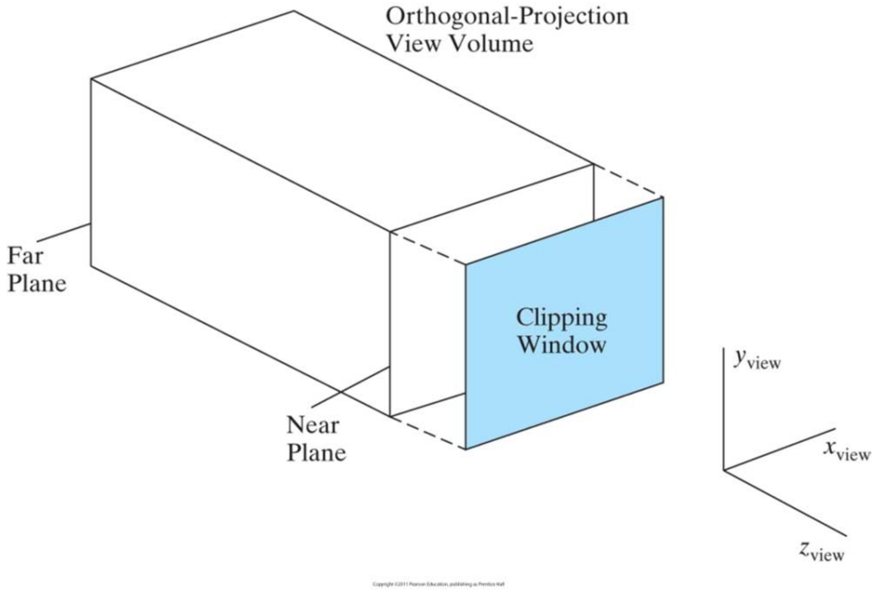




Engineering and architecture drawings commonly employ it. Length and angles are accurately depicted.

Orthogonal Projection
(Projector is orthogonal to the view plane)

- Clipping window and orthogonal-projection view volume
  - The clipping window: the **x** and **y** limits of the scene you want to display
  - These form the orthogonal-projection view volume
  - The depth is limited by near and far clipping planes in $\mathbf{z_{view}}$



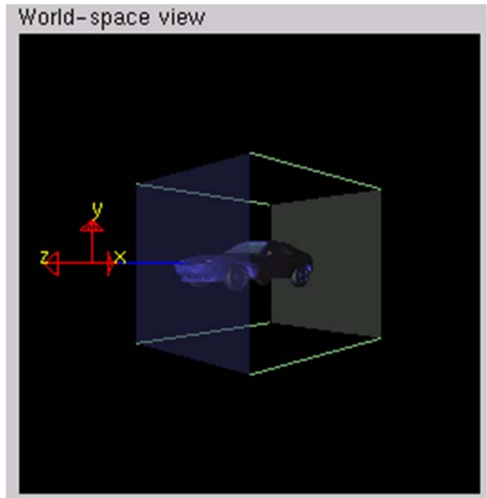


FIGURE 10-22 A finite orthogonal view volume with the **view plane** "in front" of the near plane

Normalization Transformation for Orthogonal Projections

- Map the view volume into a normalized view volume

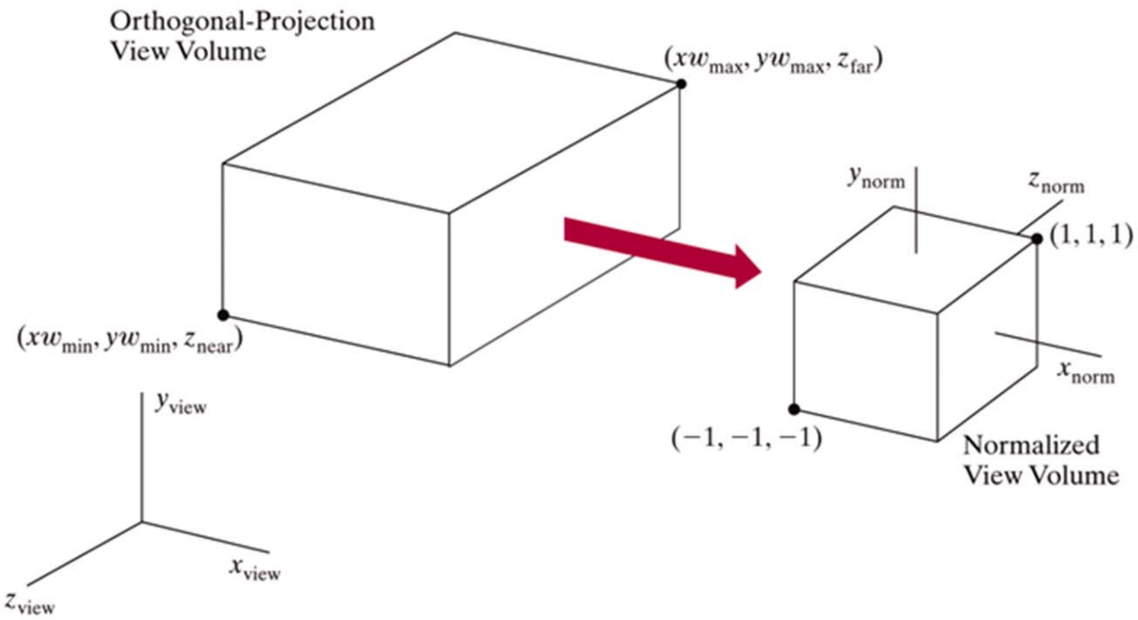


FIGURE 10-24 Normalization transformation from an orthogonal-projection view volume to the symmetric normalization cube within a left-handed reference frame.

Perspective Projections

- Positions are transferred along lines that converge to a point behind the view plane.

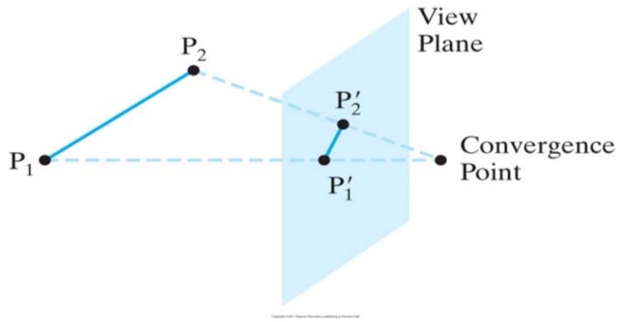


FIGURE 10-16 Perspective projection of a line segment onto a view plane.

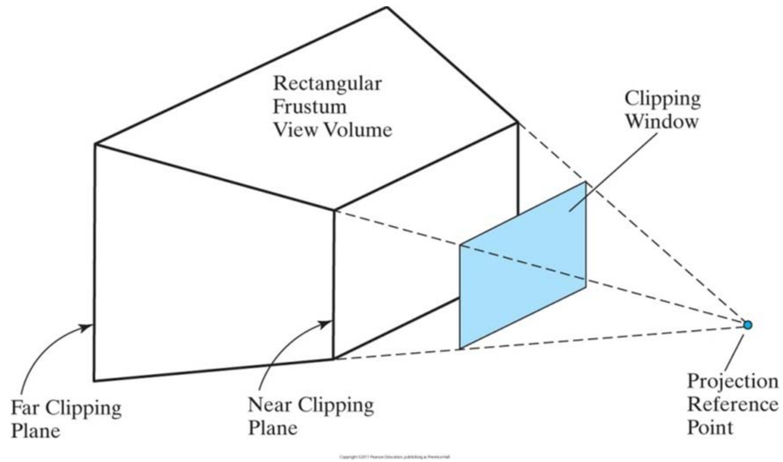
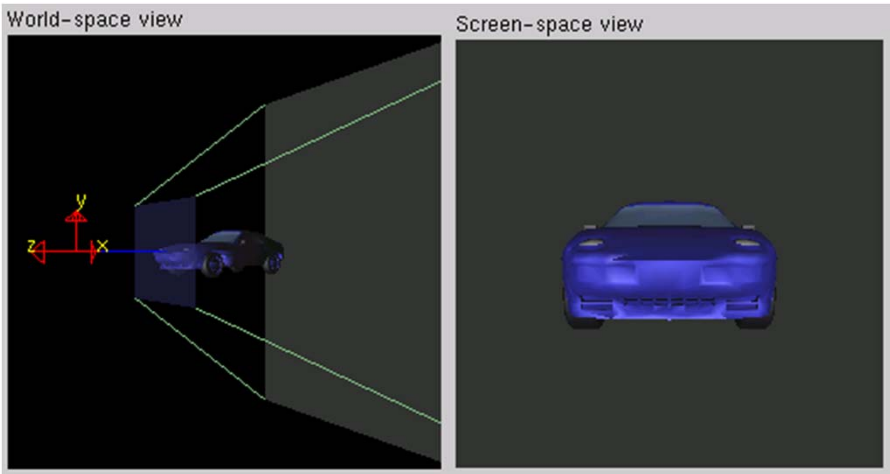
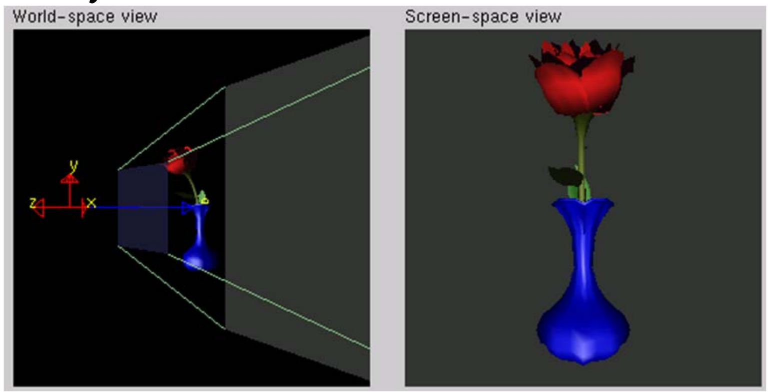


FIGURE 10-39
A perspective-projection frustum view volume with the **view plane** "in front" of the near clipping plane

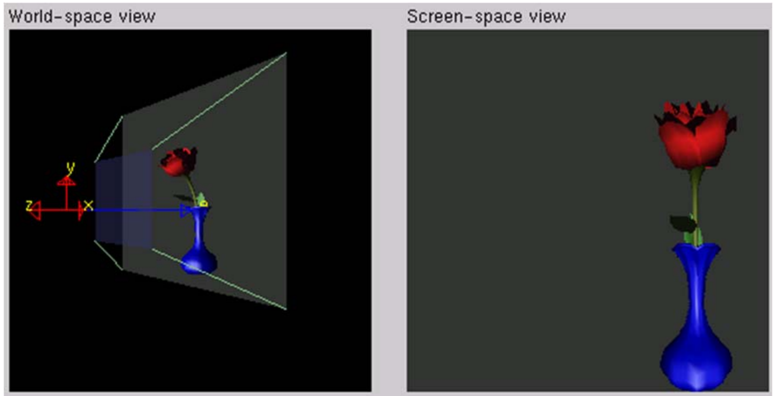


Perspective Projections

- Perspective projection view volume
  - Symmetric



  - Asymmetric



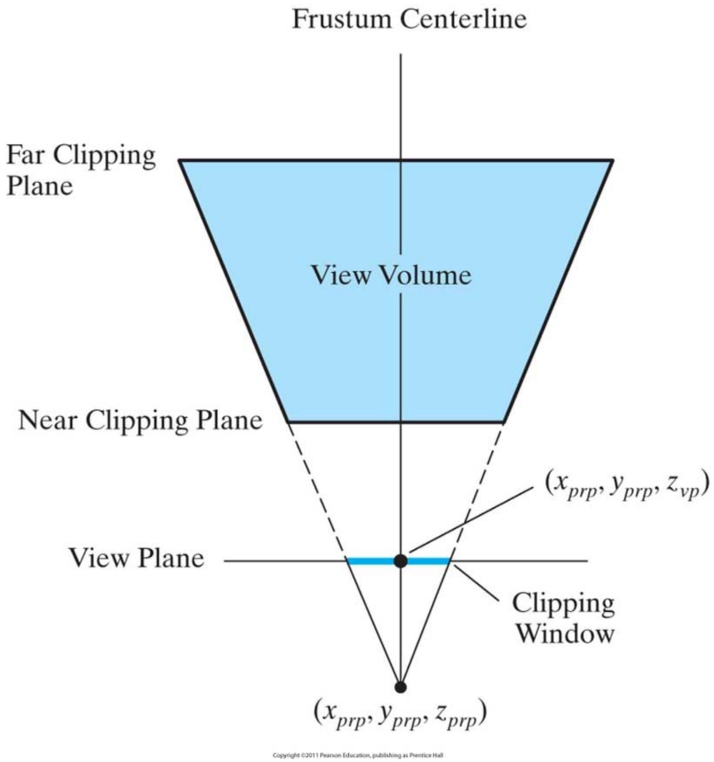


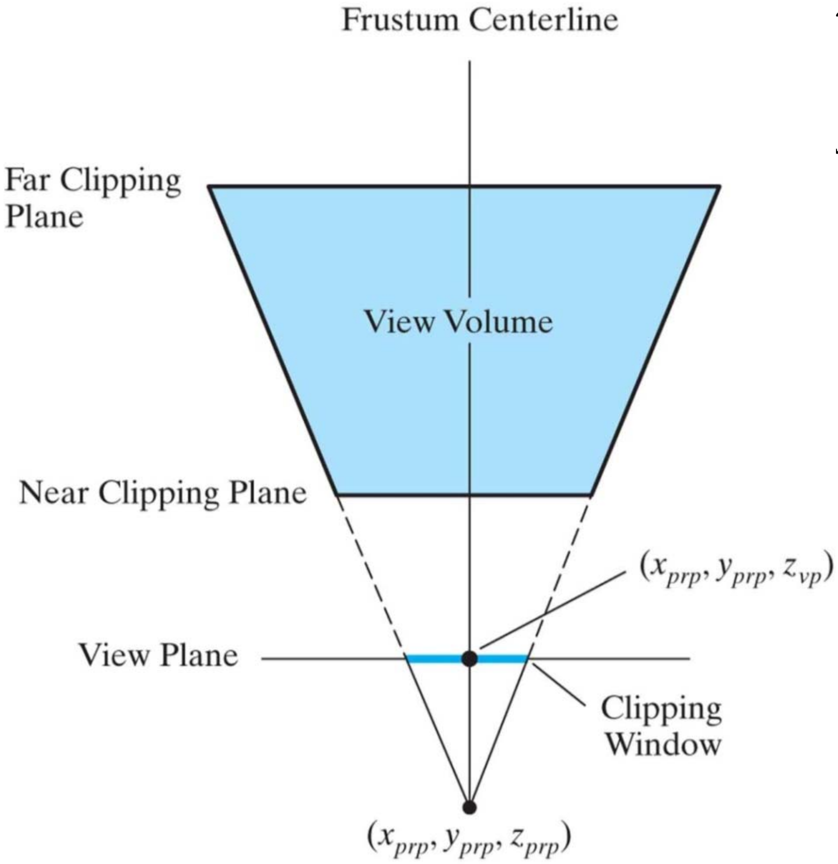
FIGURE 10-40 A symmetric perspective-projection frustum view volume.

Symmetric Perspective Projections Frustum

- The corner positions for the clipping window in terms of the window's width and height:

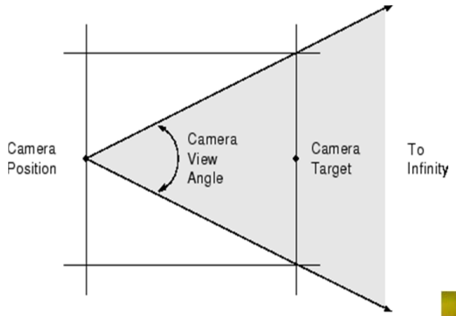
$$xw_{\min} = x_{prp} - \frac{width}{2}, \quad xw_{\max} = x_{prp} + \frac{width}{2}$$

$$yw_{\min} = y_{prp} - \frac{height}{2}, \quad yw_{\min} = y_{prp} + \frac{height}{2}$$



Symmetric Perspective Projections Frustum: field-of-view angle/ angle of view

- Another way to specify the symmetric-perspective projection volume
  - Approximate the properties of a camera lens: the **field-of-view angle / angle of view**
    - E.g.: a wide-angle lens corresponds to a larger angle of view.



Focal Length

http://www.dyxum.com/columns/photoworld/fundamentals/Field_of_view_Angle_of_view.asp

Symmetric Perspective Projections Frustum: field-of-view angle/ angle of view

- Another way to specify the symmetric-perspective projection volume
  - In CG, the angle is between the top clipping plane and the bottom clipping plane

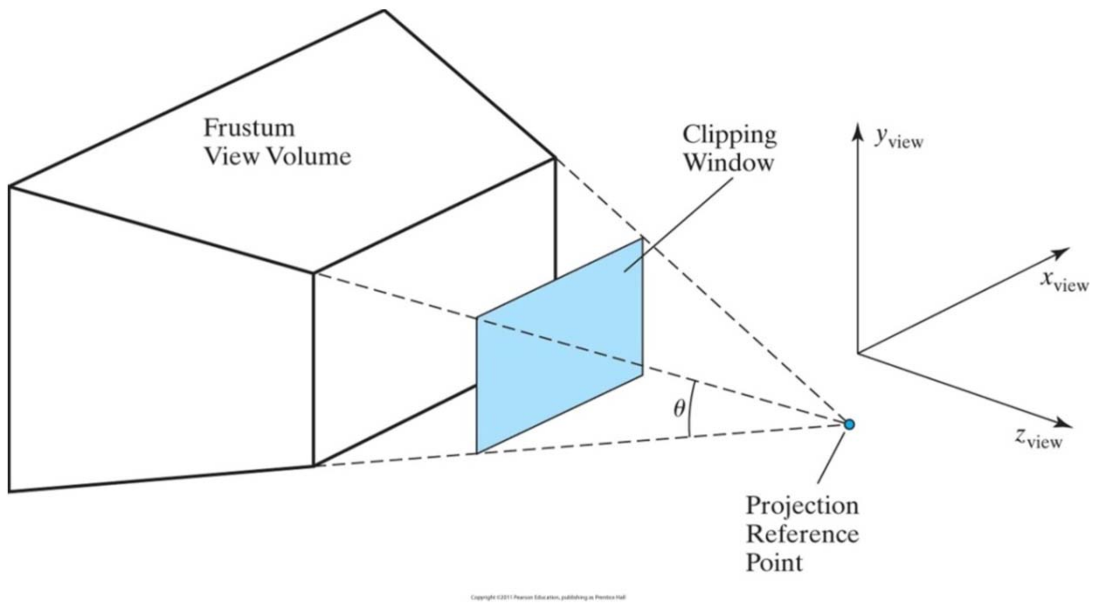


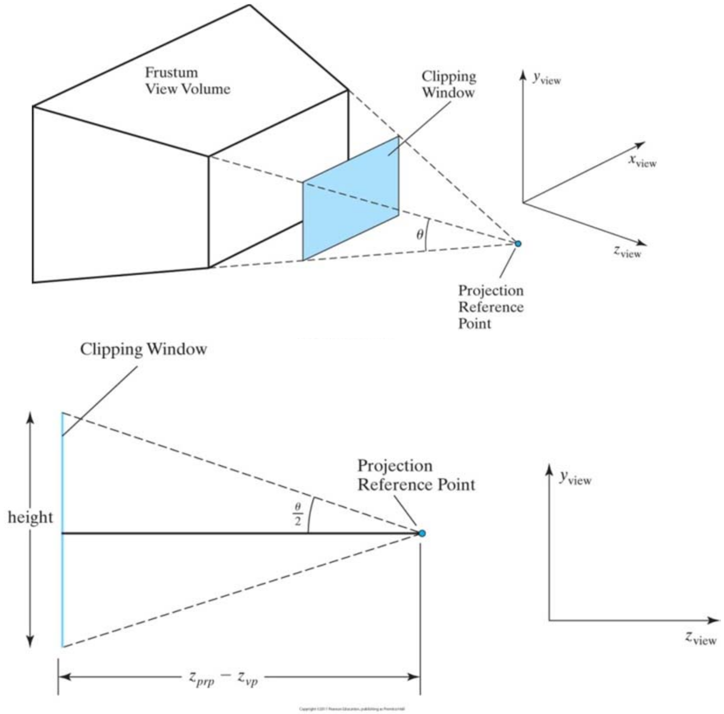
FIGURE 10-41 **Field-of-view angle** θ for a **symmetric** perspective-projection view volume.

Symmetric Perspective Projections Frustum: field-of-view angle

- For a given projection reference point and view plane position
  - The height of the clipping window is:

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

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

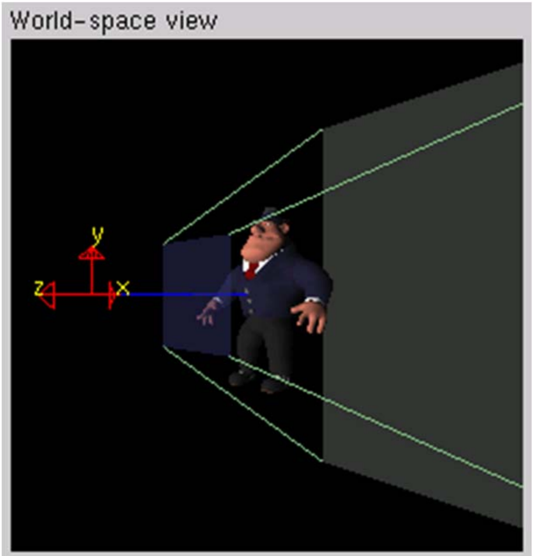


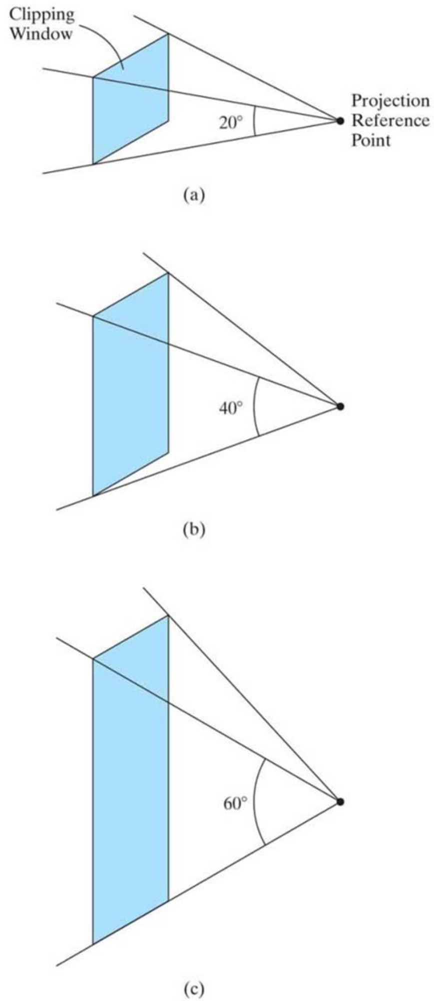
<u>How about the width?</u>

Another parameter: the aspect ratio.

Symmetric Perspective Projections Frustum: field-of-view angle

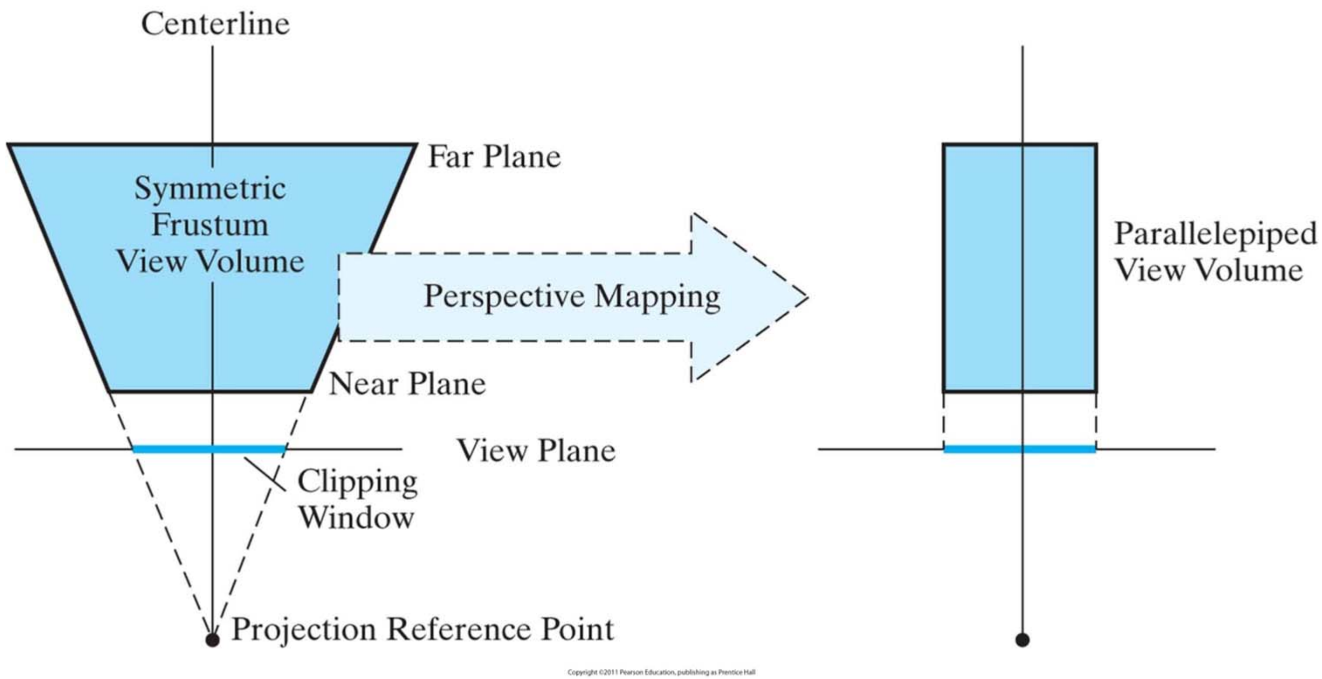
- Changing field-of-view angle





Normalization Transformation of Perspective Projections

- Mapped to a rectangular parallelepiped (平行六面体)
  - The centerline of the parallelepiped is the frustum centerline.
  - All points along a projection line within the frustum map to the same point on the view plane -> each projection line is converted by the perspective transformation to a line that is perpendicular to the view plane, and parallel to the frustum centerline.



Normalization Transformation of Perspective Projections

- The rectangular parallelepiped is mapped to a symmetric normalized cube within a left-handed frame.

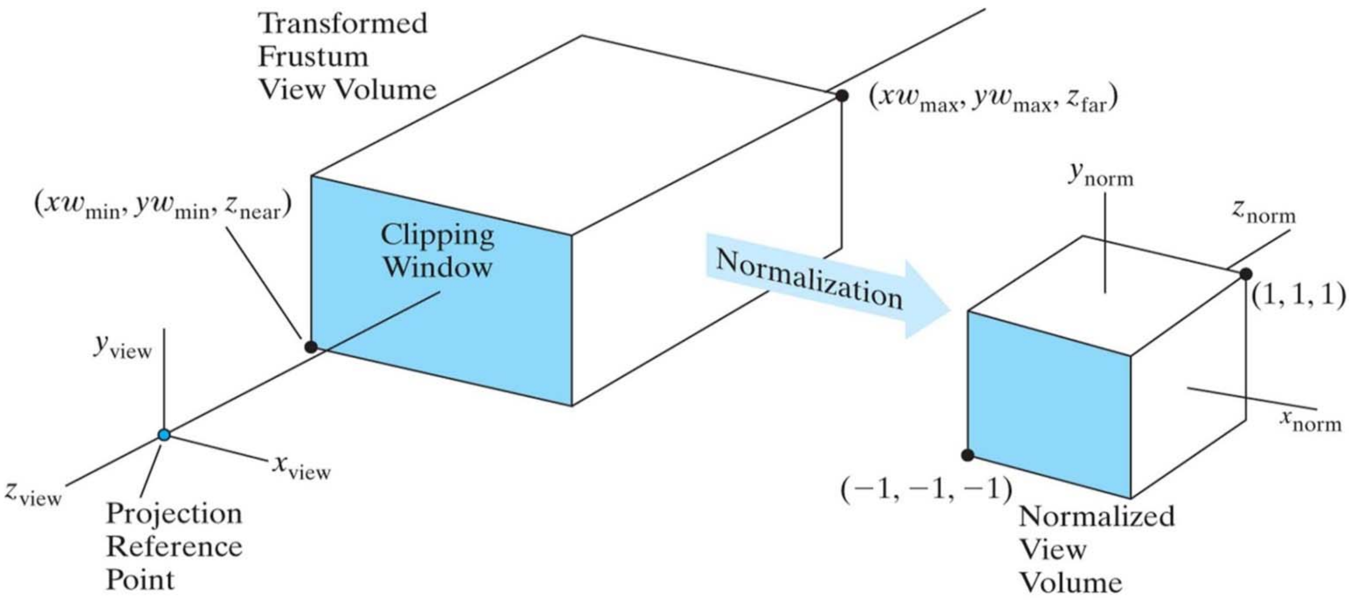


FIGURE 10-46 **Normalization transformation** from a transformed **perspective projection view volume** (rectangular parallelepiped) to the symmetric **normalization cube** within a left-handed reference frame, with the near clipping plane as the view plane and the **projection reference point** at the viewing-coordinate origin.

Chapter 10
Three-Dimensional Viewing

Part I.

Overview of 3D Viewing Concept

3D Viewing Pipeline vs. OpenGL Pipeline

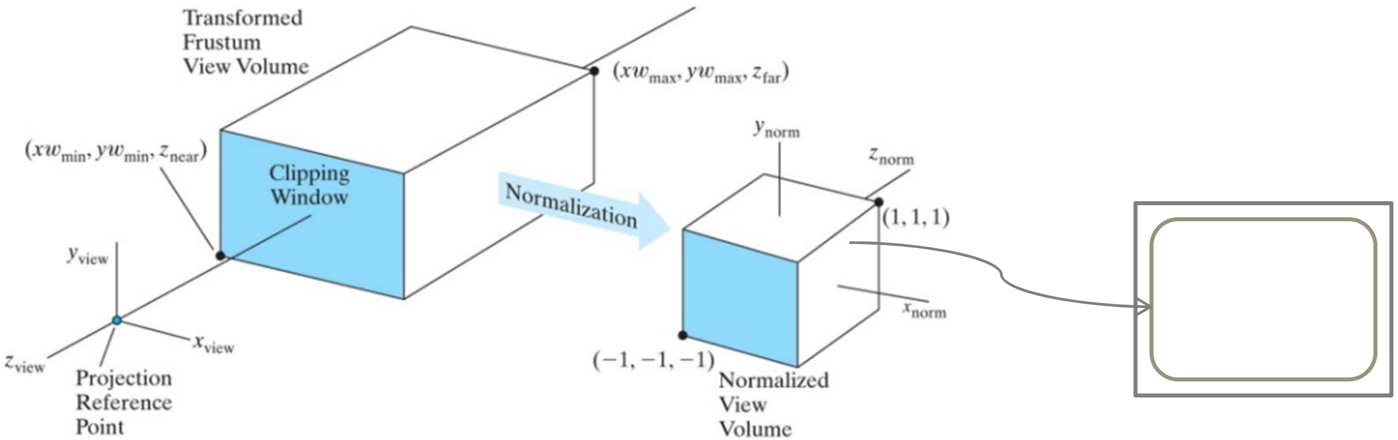
3D Viewing-Coordinate Parameters

Projection Transformations

Viewport Transformation and 3D Screen Coordinates

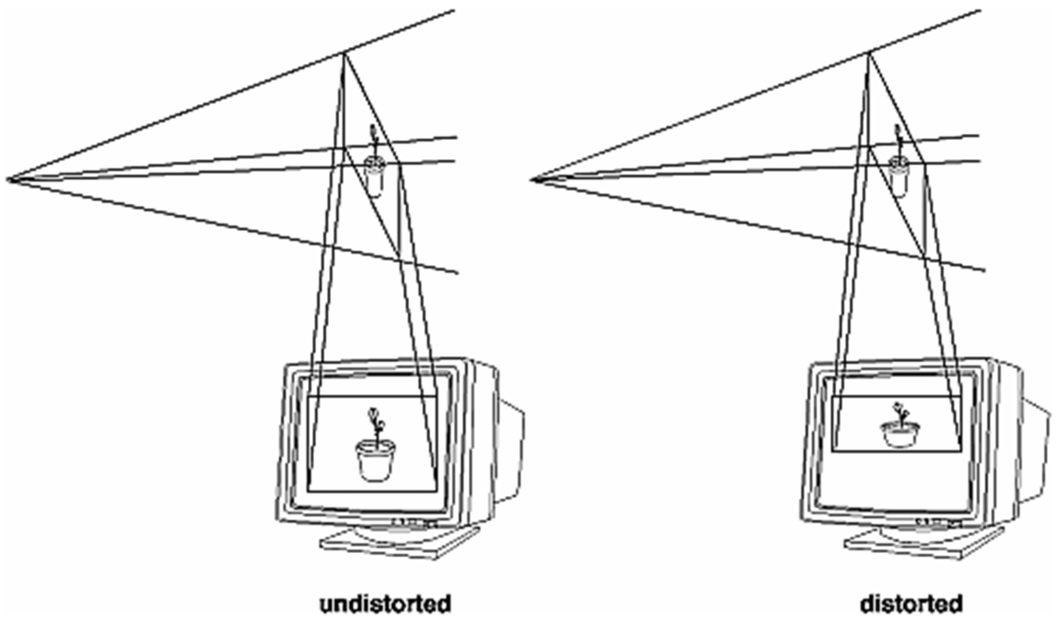
Viewport Transformation and 3D Screen Coordinates

- Transform the normalized projection coordinates to screen coordinates (3D screen coordinates)
  - For **x** and **y** in the normalized clipping window, the transformation is the same as 2D viewport transformation
  - For **z** (depth)
    - for the visibility testing and surface rendering algorithms
- In normalized coordinates, the Z_{norm} = -1 face of symmetric cube corresponds to the clipping-window area
  - This face is mapped to the 2D screen viewport, that is Zscreen = 0.
- The z (depth) value for each screen point
  - Depth buffer or z-buffer



Viewport Mapping

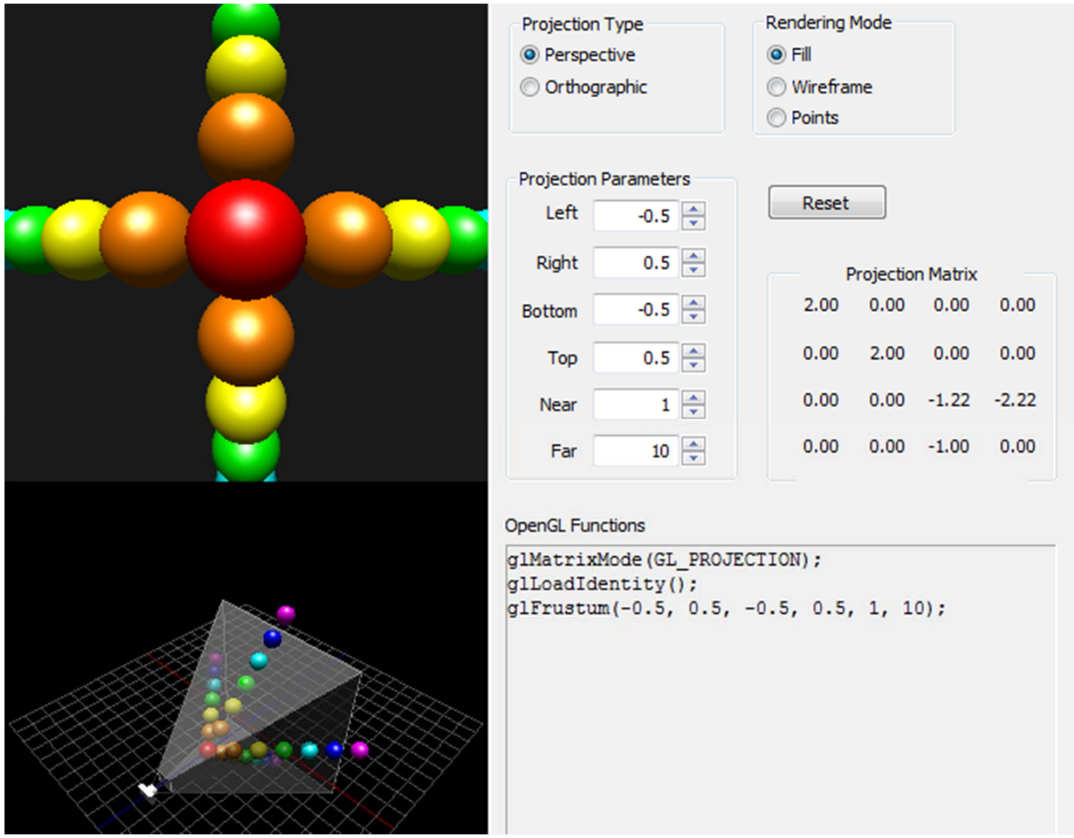
- Mapping the viewing volume to the viewport



(From OpenGL Super Bible)

The aspect ratio of a **viewport** should generally equal the aspect ratio of the **viewing volume**. If the two ratios are different, the projected image will be distorted when mapped to the viewport.

Projection Demo



Chapter 10
Three-Dimensional Viewing

Part II.

OpenGL 3D Viewing Functions

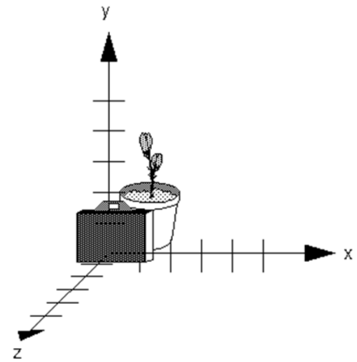
OpenGL 3D Projection Functions

- Orthogonal-Projection Function
- Perspective-Projection Functions

OpenGL 3D Viewing Program Example

OpenGL 3D Viewing Functions

- A **viewing transformation** changes the position and orientation of the viewpoint.
  - Recall the camera analogy, it positions the camera tripod, pointing the camera toward the model.
  - Composed of **translations** and **rotations**.
  - The same effects can be implemented either
    - move the camera or
    - move the objects in the opposite direction.

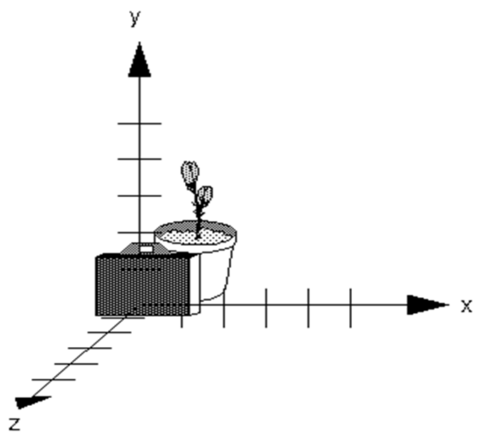


Note: The **viewing transformation** commands should be called **before** any **modeling transformations** are performed, so that the modeling transformations take effect on the objects first.

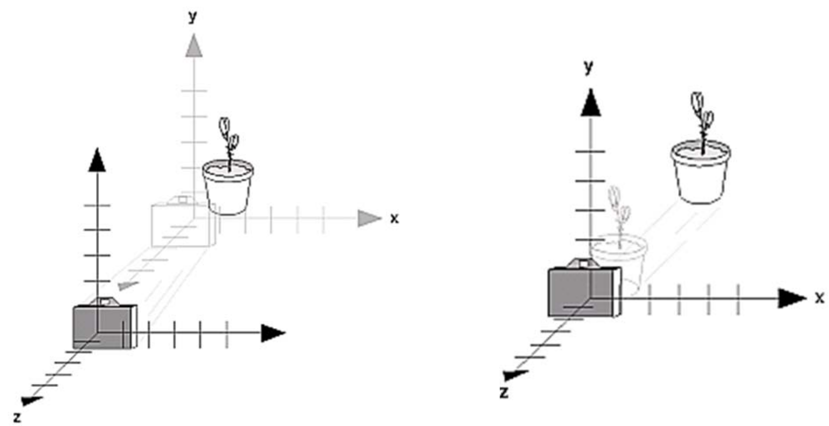
OpenGL 3D Viewing Functions

Method a. using **glTranslate*()** and **glRotate*()**

- To emulate the **viewpoint** movement in a desired way by translating the **objects:**



Object and Viewpoint: at the Origin.



Move the object away from the viewpoint by translating the object along "-z " direction:

glTranslatef (0.0, 0.0, -5.0);

(From: the red book)

OpenGL 3D Viewing Functions

Method b. using the **gluLookAt**(eyex, eyey, eyez, atx, aty, atz, upx, upy, upz**)** utility routine

- 3 sets of arguments
  - The **location of the viewpoint**,
  - A **reference point** where you look at
    - Some position in the center of a scene
  - **View-Up** direction

$$n = \frac{N}{|N|} = (n_x, n_y, n_z) \qquad (z+)$$

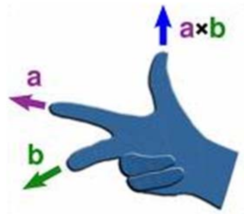
$$u = \frac{V \times n}{|V|} = (u_x, u_y, u_z) \qquad (x+) \qquad (10\text{-}1)$$

$$v = n \times u = (v_x, v_y, v_z) \qquad (y+)$$

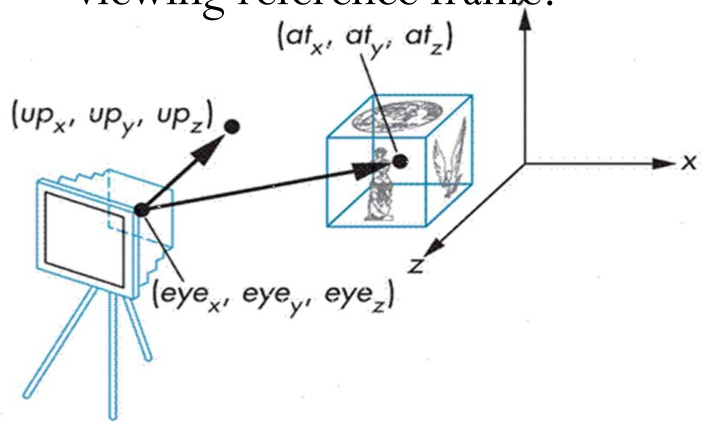
- The **default OpenGL** viewing parameters are:

$P_0 = (0, 0, 0)$, $P_{ref} = (0, 0, -1)$, $V = (0, 1, 0)$

- The viewing reference frame defined by the viewing parameters
  - Zview +: **N** = **at-eye**;
  - Yview +: **V** = **up**;
  - Xview +: **U** = **V x N** .

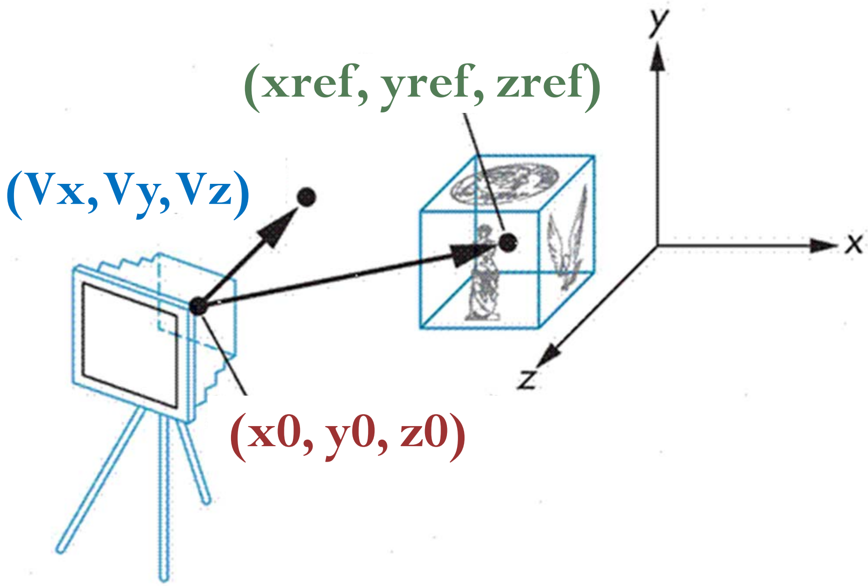


The unit axis vectors (*uvn*) for the viewing reference frame:

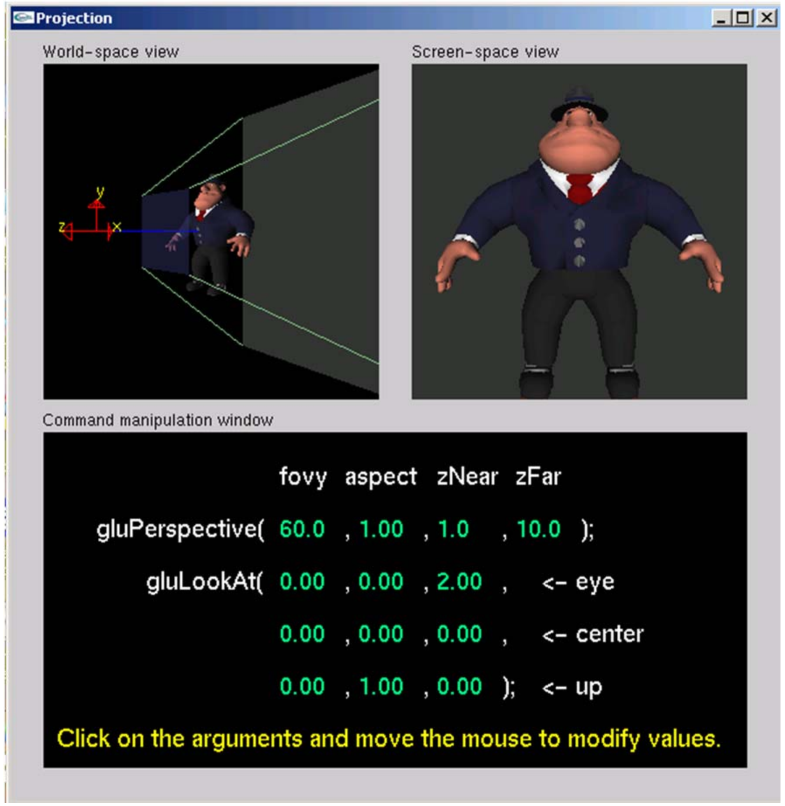


OpenGL 3D Viewing Functions

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




```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
gluLookAt(x0, y0, z0, xref, yref, zref,
Vx,Vy,Vz);
//gluLookAt( 0.0, 0.0, 5.0,  0.0, 0.0,
//0.0,  0.0, 1.0, 0.0);
```



The default OpenGL viewing parameters:

$P_0 = (0, 0, 0)$, $P_{ref} = (0, 0, -1)$, $V = (0, 1, 0)$

Viewing Transformation: HowTo

Method a. Use one or more modeling transformation commands (that is, **glTranslate*()** and **glRotate*()**).

Method b. Use the Utility Library routine **gluLookAt()** to define a line of sight. This routine encapsulates a series of rotation and translation commands.

Method c. Create your own utility routine that encapsulates rotations and translations.

OpenGL 3D Projection Functions

- The purpose of the **projection transformation** is to define a *viewing volume,* which is used in **two** ways
  - How an object is projected onto the screen;
  - Which objects or portions of objects are clipped out of the final image.
- Before issuing any of projection commands, you should call

  glMatrixMode(GL_PROJECTION);

  glLoadIdentity();

  so that the commands affect the projection matrix rather than the modelview matrix.

OpenGL 3D Projection Functions

- OpenGL provides two functions
  - **glOrtho():** to produce a orthographic (parallel) projection
  - **glFrustum():** to produce a perspective projection (general)

  Both functions require **6** parameters to specify six clipping planes: *left*, *right*, *bottom*, *top*, *near* and *far* planes.

- GLU library for symmetric perspective-projection
  - **gluPerspective():** with only **4** parameters

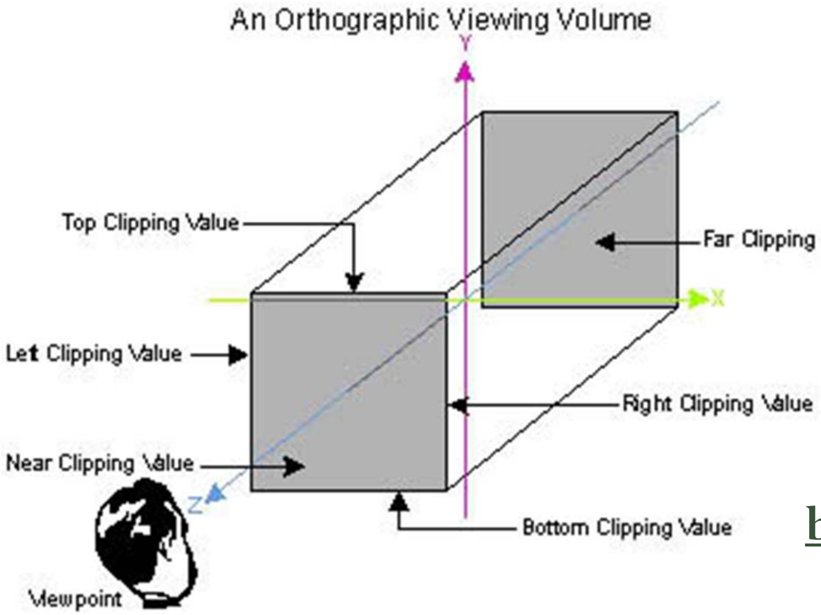
OpenGL Orthogonal-Projection Function

glOrtho (left, right, bottom, up, near, far);

up

left

near



far

right

bottom

$$\begin{pmatrix} \frac{2}{right - left} & 0 & 0 & t_x \\ 0 & \frac{2}{top - bottom} & 0 & t_y \\ 0 & 0 & \frac{-2}{farVal - nearVal} & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

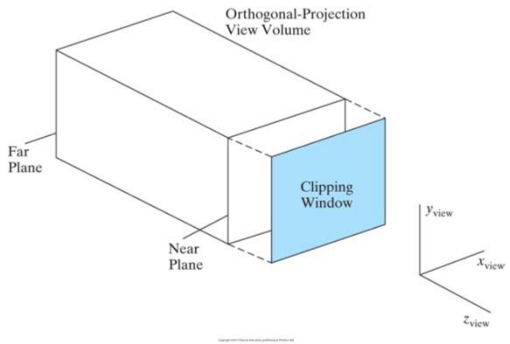
where

$$t_x = -\frac{right + left}{right - left} \quad t_y = -\frac{top + bottom}{top - bottom}$$

$$t_z = -\frac{farVal + nearVal}{farVal - nearVal}$$

In **OpenGL** there is no option for the placement of the view plane:

The near clipping plane = the **view plane**;



OpenGL Orthogonal-Projection Function

glOrtho (left, right, bottom, up, near, far);

- the default one:

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

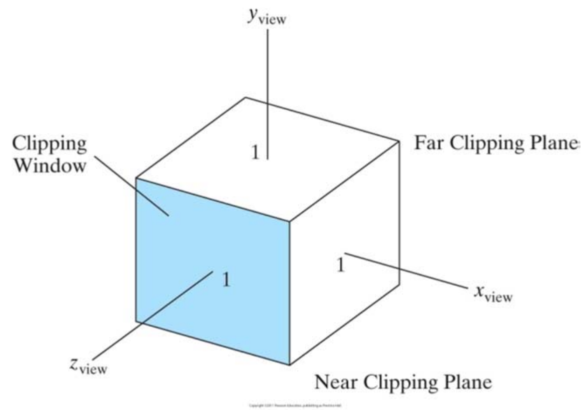
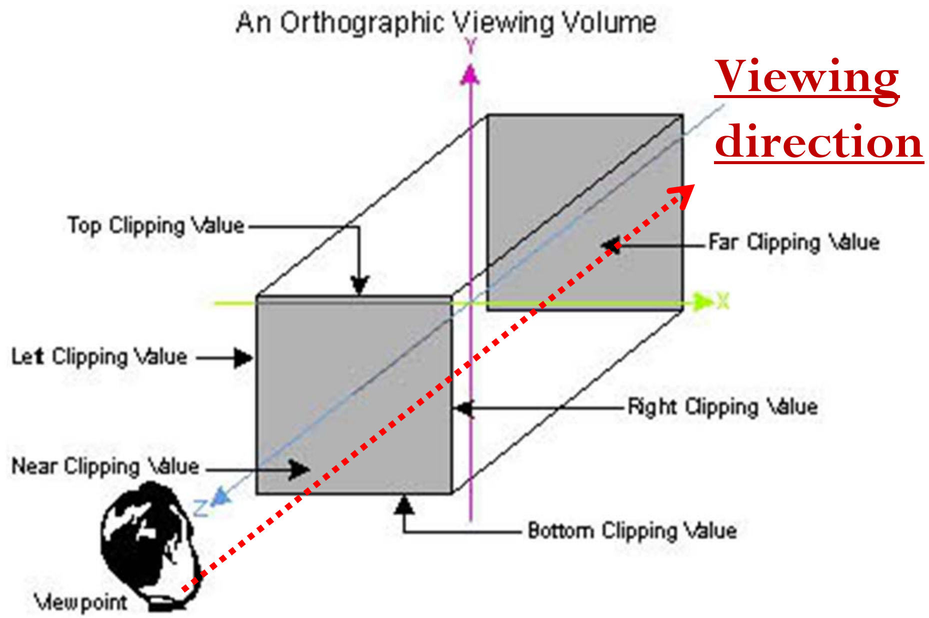


FIGURE 10-47 Default orthogonal-projection view volume.

2D: gluOrtho2D(*left, right, bottom, up*);
<=> a call to glOrtho() with near = -1.0 and far = 1.0.

glOrtho (*left, right, bottom, up, 0, 5.0*);
the far clipping plane: z_{far} = -5.0.
If near or far are **negative**, the plane is at the positive z_{view} axis (***behind*** the viewing origin)



OpenGL Perspective-Projection Functions

- General perspective-projection function

glFrustum (left, right, bottom, up, near, far);

left, right, bottom, up: set the size of the clipping window on the near plane.

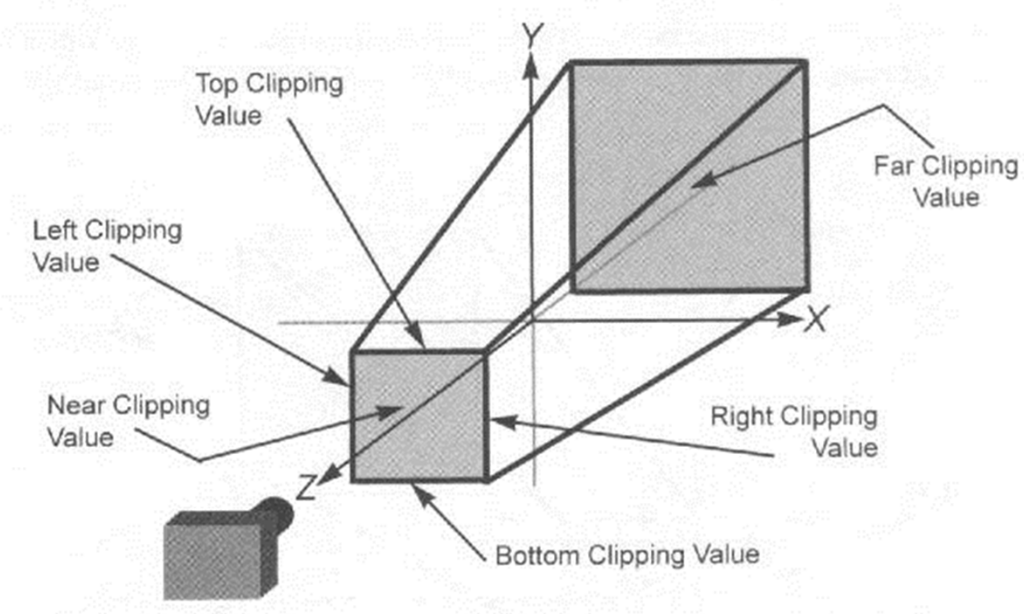
near, far: the **distances** from the origin to the near and far clipping planes along the $-\mathbf{z}_{\mathbf{view}}$ axis. They **must** be **positive**. ($\mathbf{z}_{\mathbf{near}}$ = -near and $\mathbf{z}_{\mathbf{far}}$ = -far)

$$\begin{bmatrix} \frac{2\ nearVal}{right - left} & 0 & A & 0 \\ 0 & \frac{2\ nearVal}{top - bottom} & B & 0 \\ 0 & 0 & C & D \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

$$A = \frac{right + left}{right - left} \qquad B = \frac{top + bottom}{top - bottom}$$

$$C = -\frac{farVal + nearVal}{farVal - nearVal}$$

$$D = -\frac{2\ farVal\ nearVal}{farVal - nearVal}$$



OpenGL Perspective-Projection Functions

- Symmetric perspective-projection function

gluPerspective(theta, aspect, near, far);

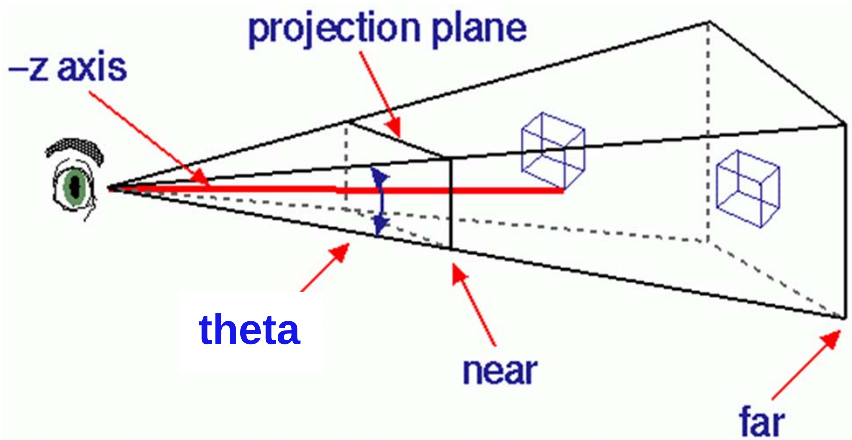
theta: the **field-of-view angle** between the top and bottom clipping planes in the range [0° , 180°].

aspect: the **aspect ratio (width/height)** of the clipping window.

near, far: specify the distances from the view point (coordinate origin) to the near and far clipping planes.

Both **near** and **far** must be **positive** values.

z_{near} = -near and z_{far} = -far refer to the positions of the near and far planes.



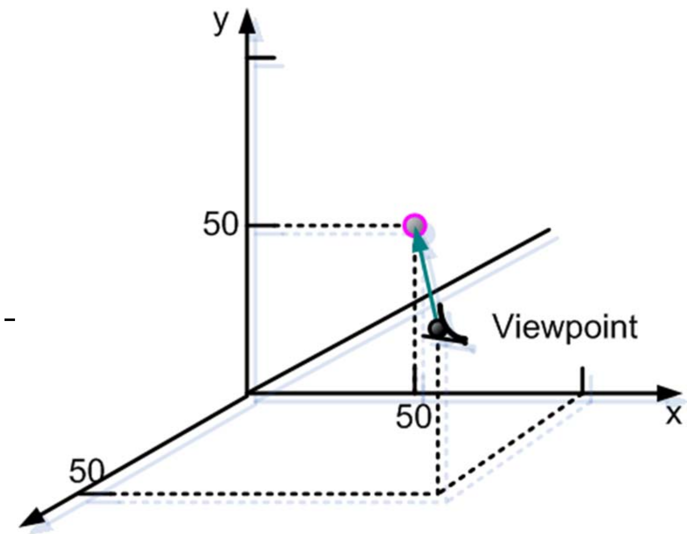
OpenGL 3D Viewing Program Example

```
#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 P0.
GLfloat xref=50.0, yref=50.0, zref=0.0; // Look-at point Pref;
GLfloat Vx=0.0, Vy=1.0, Vz=0.0;                    // View-up vector
/*positive zview axis N = P0 - Pref = (50.0, 0.0, 50.0)

/* Set coordinate limits for the clipping window: */
GLfloat xwMin = -40.0, xwMax= 40.0, ywMin = -60.0, ywMax= 60.0;

/* Set positions for near and far clipping planes: */
GLfloat dnear=25.0, dfar=125.0;
```

FIGURE 10-48 Output display generated by the three-dimensional viewing example program.



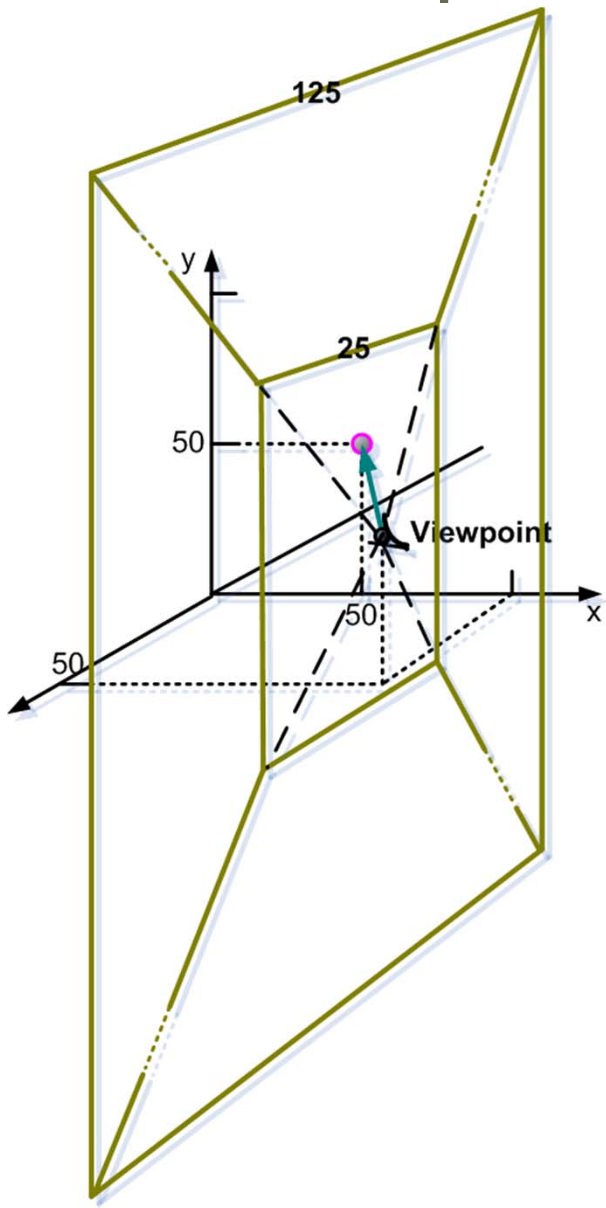
OpenGL 3D Viewing Program Example

```
void init()
{
    glClearColor(1.0, 1.0, 1.0, 0.0);

    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glFrustum(xwMin, xwMax, ywMin, ywMax, dnear, dfar);

    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    gluLookAt(x0, y0, z0, xref, yref, zref, Vx, Vy, Vz);
}

void reshapeFcn(GLsizei newWidth, GLsizei newHeight)
{
    glViewport(0, 0, newWidth, newHeight);
    winWidth=newWidth;
    winHeight=newHeight;
}
```



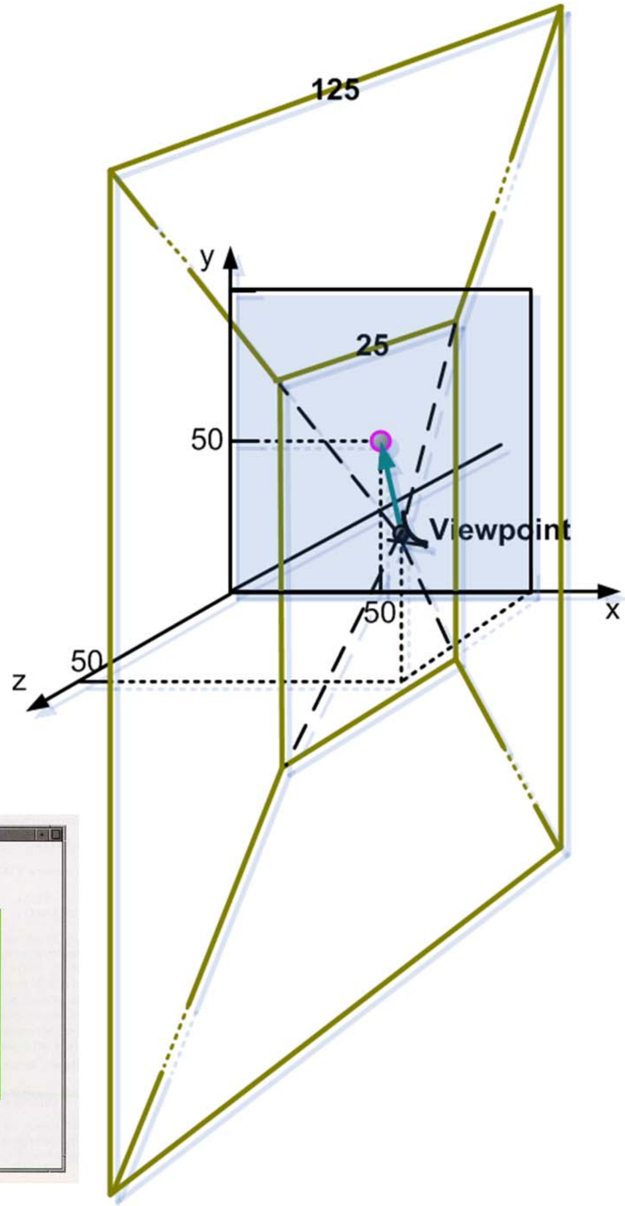
OpenGL 3D Viewing Program Example

```
void displayFcn(void)
{
   glClear(GL_COLOR_BUFFER_BIT |
             GL_DEPTH_BUFFER_BIT);

   /* Set parameters for a square fill area. */
   // Set fill color to green.
   glColor3f(0.0, 1.0, 0.0);
   glPolygonMode(GL_FRONT, GL_FILL);
   // Wire-frame back face.
   glPolygonMode(GL_BACK, GL_LINE);
   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();
}
```

Foreshortening effect
Square -> trapezoid



Summary

- 3D viewing pipeline and camera analogy
- 3D viewing transformation and projected transformation
  - viewing coordinates (eye)
  - Orthogonal projection
  - Perspective projection
- OpenGL and utility functions
  - glMatrixMode();
    - GL_MODELVIEW/GL_PROJECTION
  - gluLookAt();
  - glOrtho();
  - glFrustum();
  - gluPerspective();