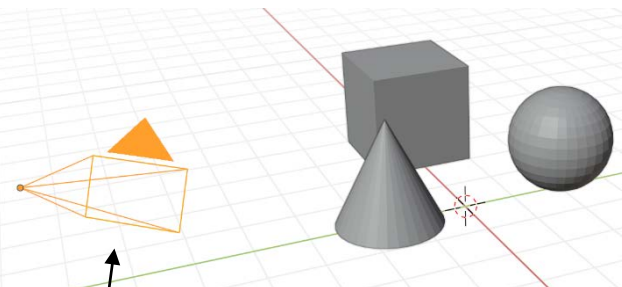


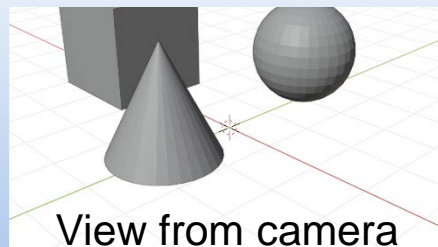
Viewing and Projection

The Computer Graphics Pipeline

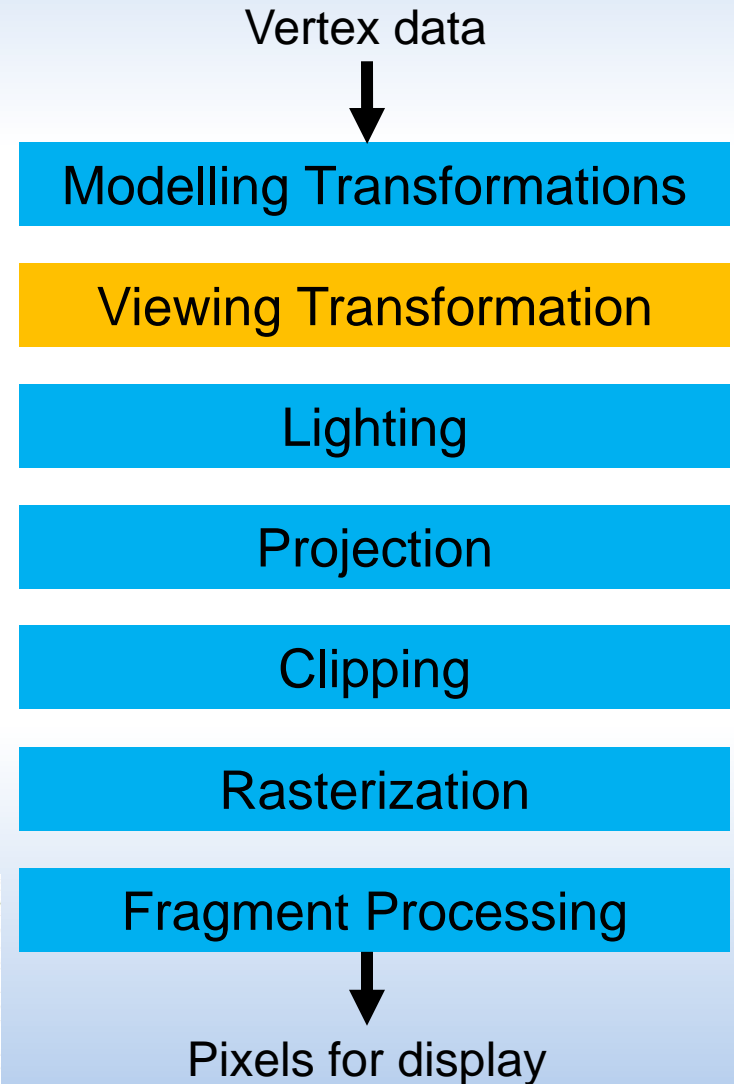
- Viewing transformations
 - World space to **view space** (or camera/eye space)
 - Viewing position transformed to origin and direction orientated along the z-axis
 - View seen from camera's viewpoint based on camera's orientation



Camera



View from camera

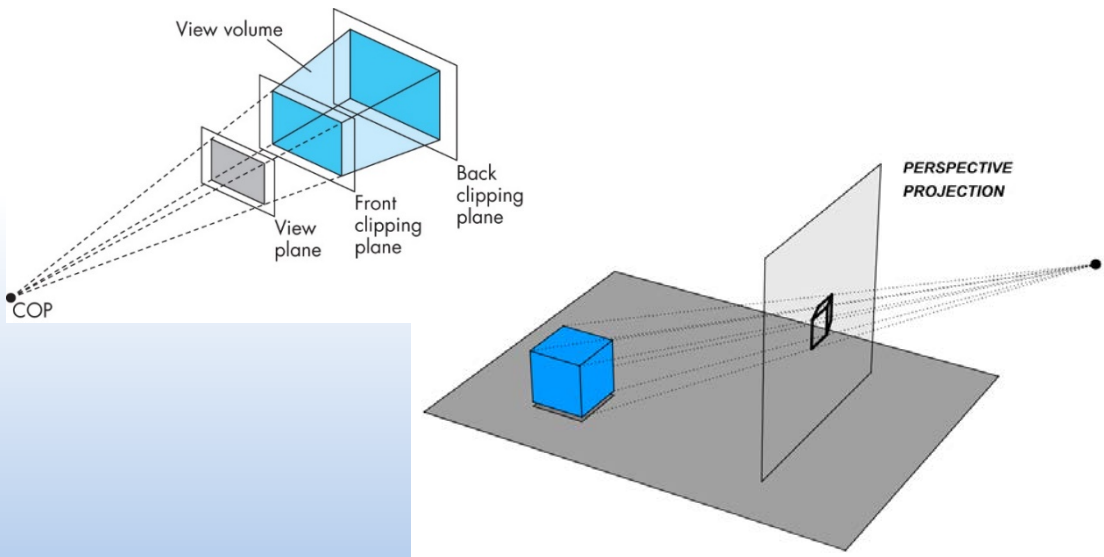


The Computer Graphics Pipeline

- Projection

- 3D scene projected to 2D surface

- Defines a **view volume**
 - Transformed into normalized device coordinates (NDC)



Vertex data



Modelling Transformations

Viewing Transformation

Lighting

Projection

Clipping

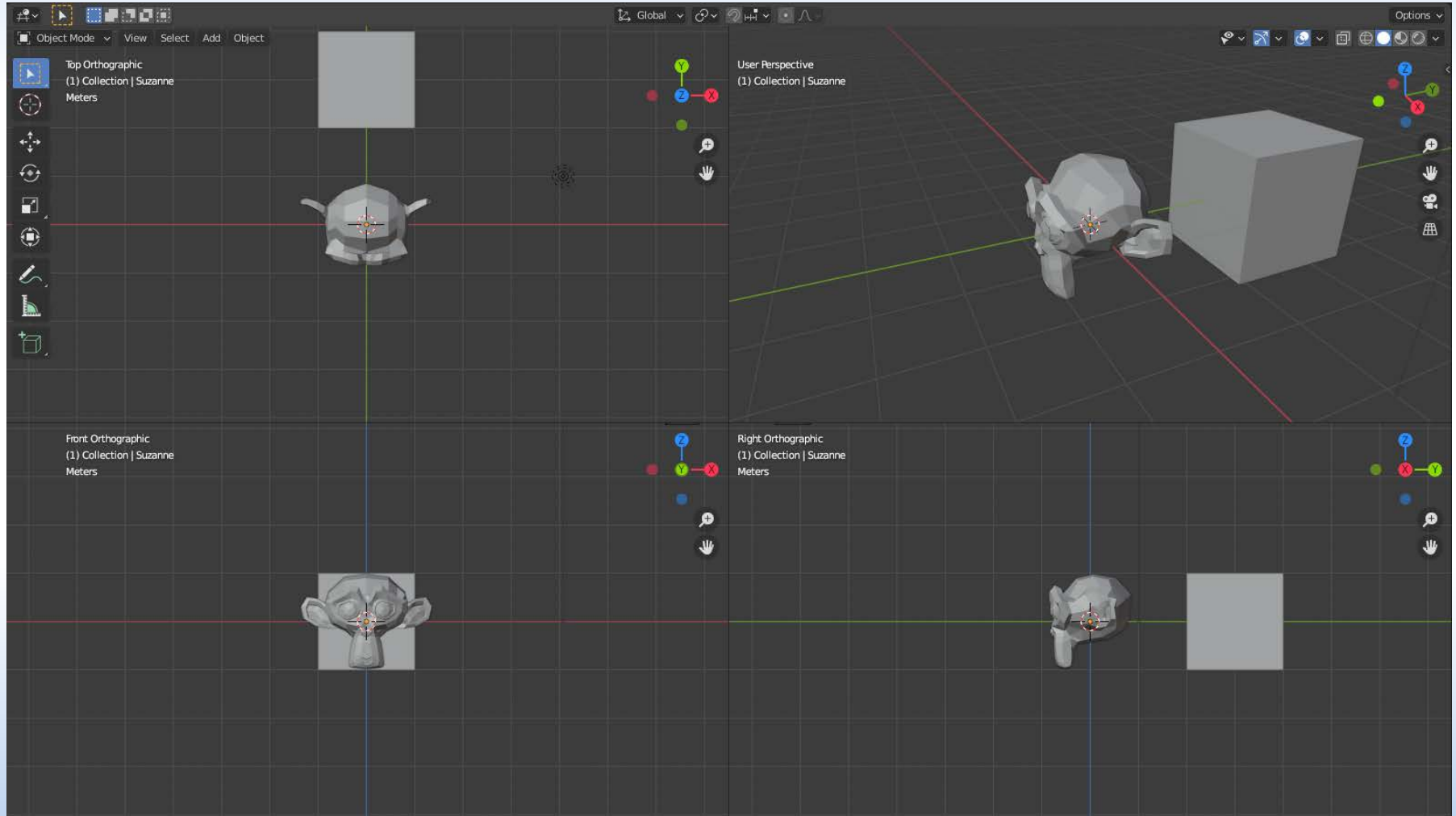
Rasterization

Fragment Processing



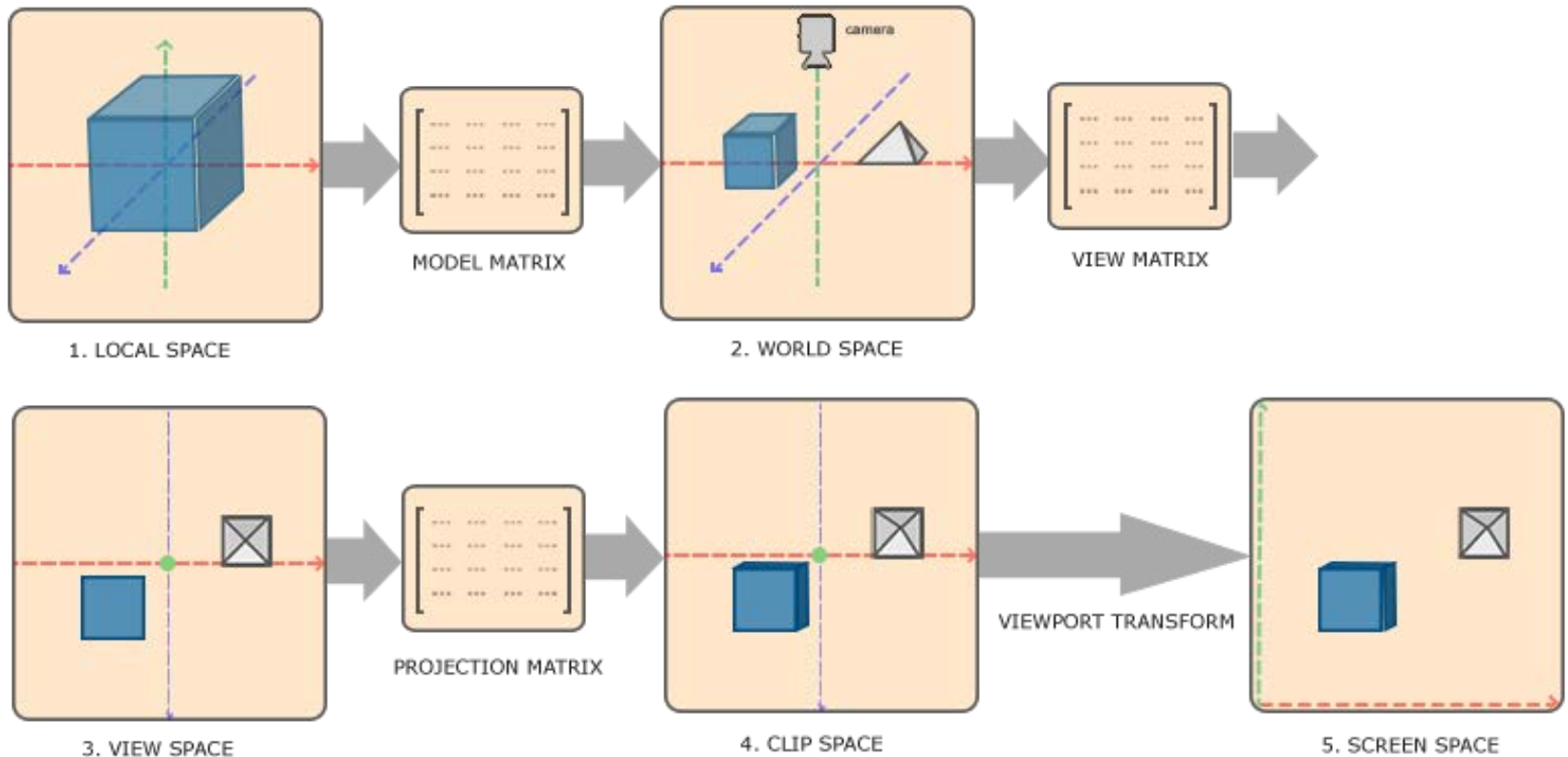
Pixels for display

Multiview Orthographic



The Computer Graphics Pipeline

- Coordinate systems
 - Transformation matrices



Camera Analogy

- Transformations

- Modelling transformations

- Moving the model

- Viewing transformation

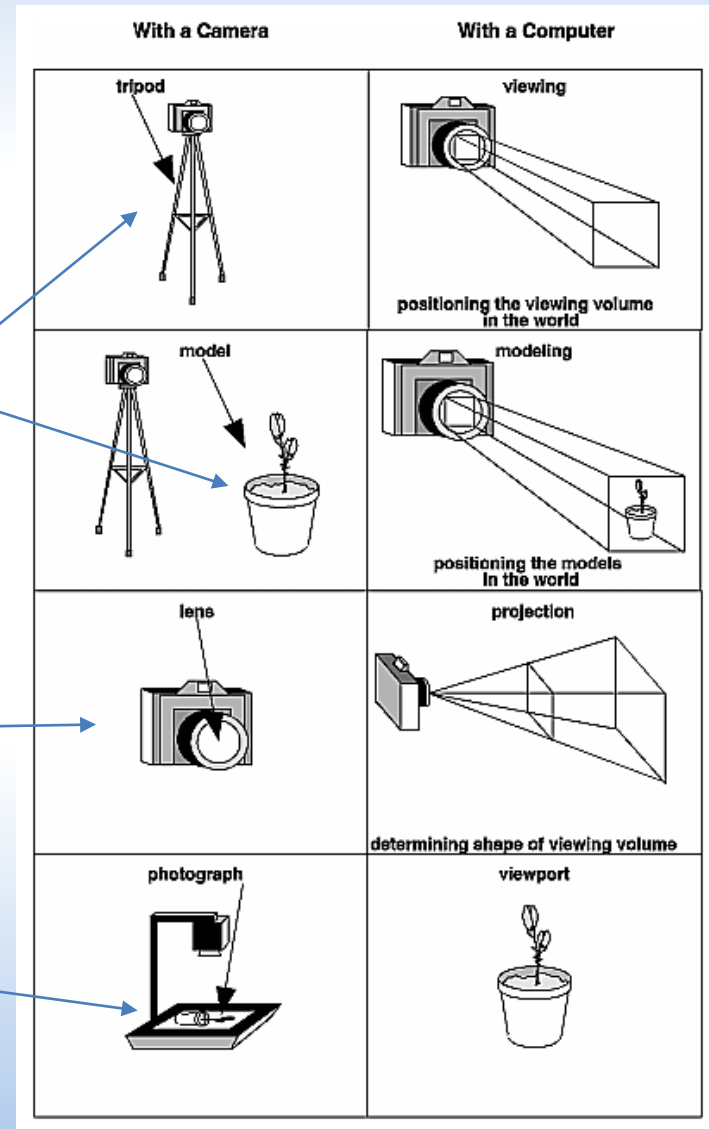
- Tripod-defined position and orientation of the viewing volume in the world

- Projection transformation

- Adjust the camera lens

- Viewport transformation

- Enlarge or reduce the physical photograph



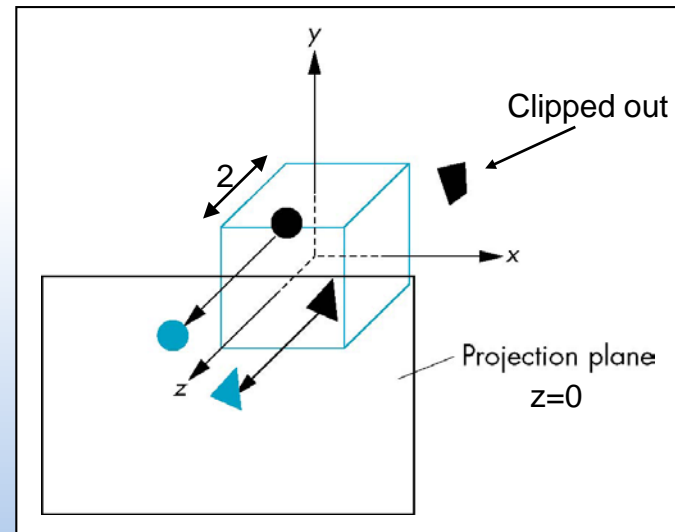
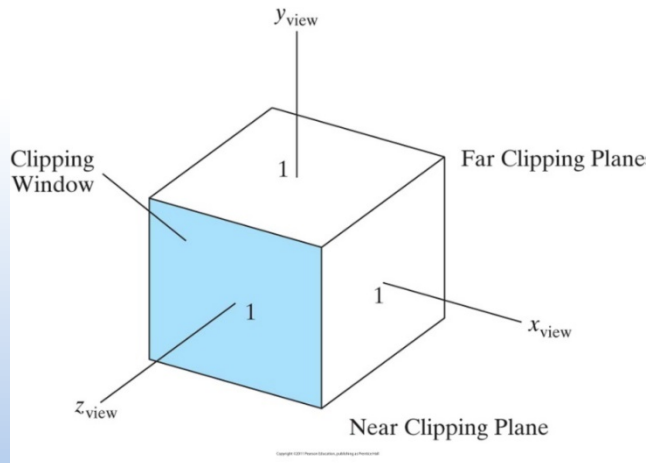
OpenGL

- OpenGL default camera

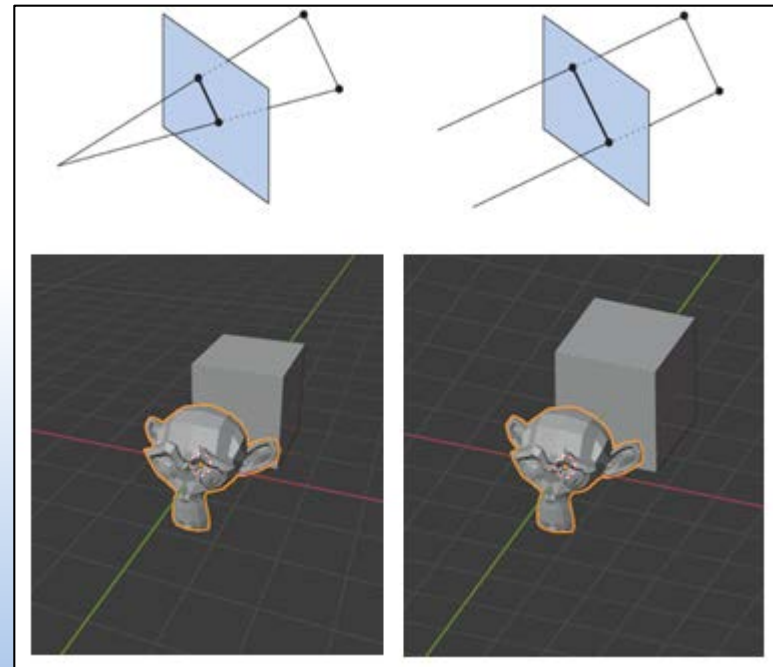
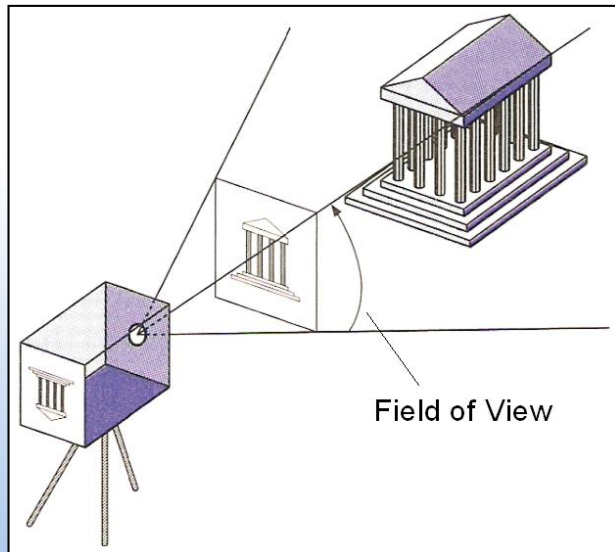
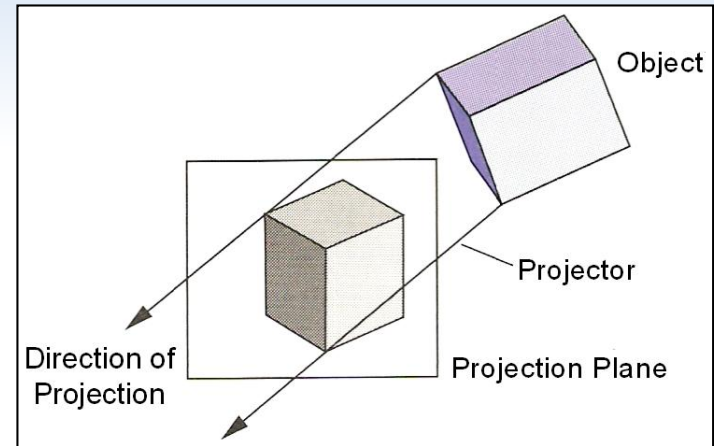
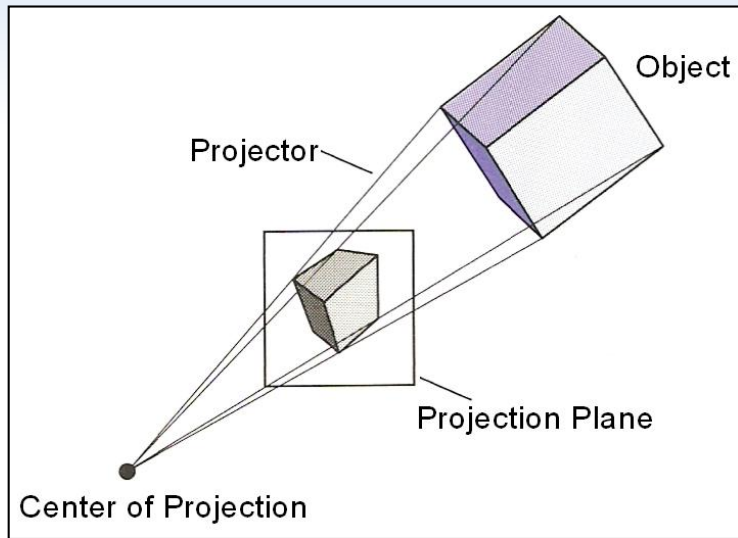
- View and projection matrices – identity matrix

- Orthographic view

- Camera at the origin in world space pointing in the negative z-direction
 - Default viewing volume is a box centered at the origin with axes between -1 and +1

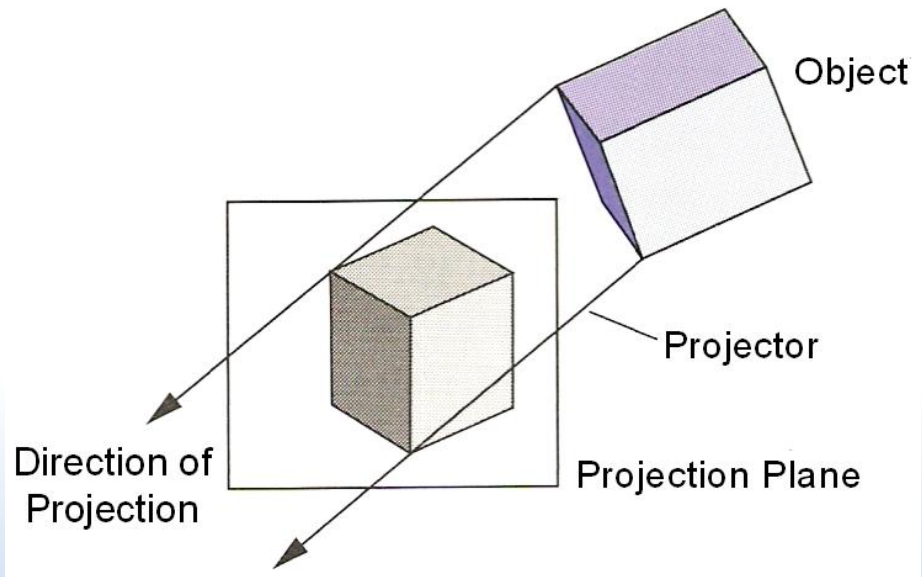
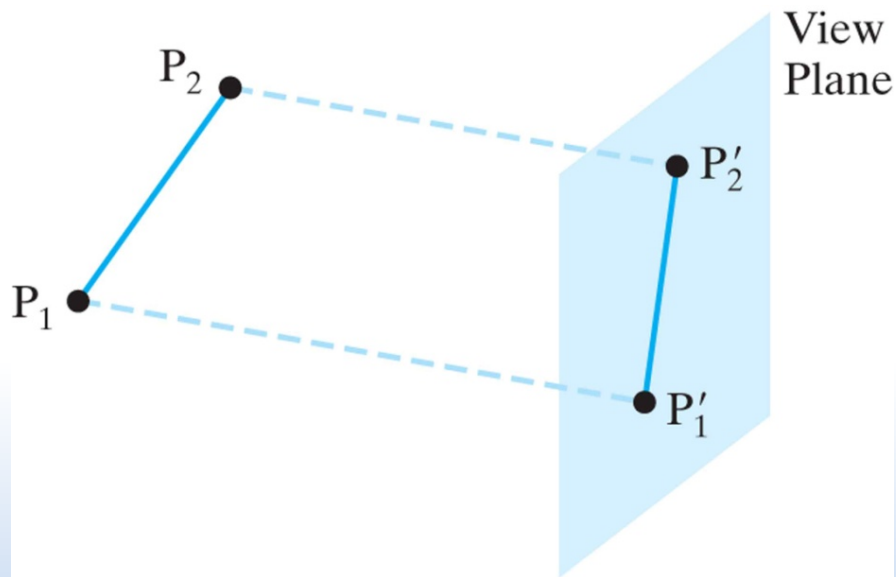


Perspective and Parallel Projection



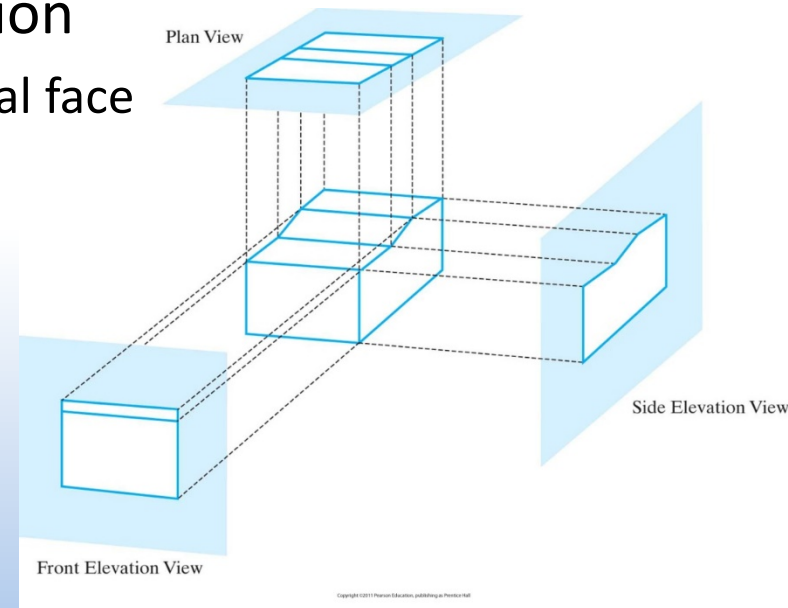
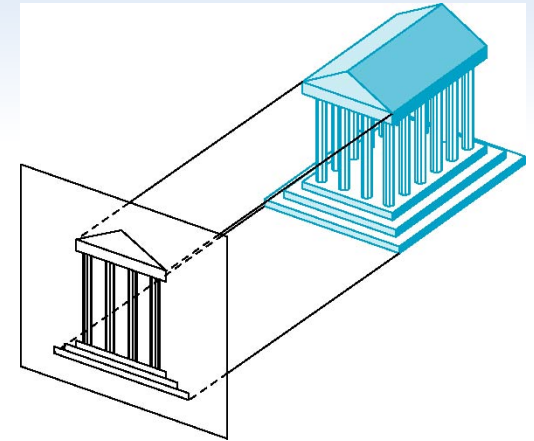
Parallel Projection

- Parallel projection
 - Coordinate positions transferred to the view plane along parallel lines



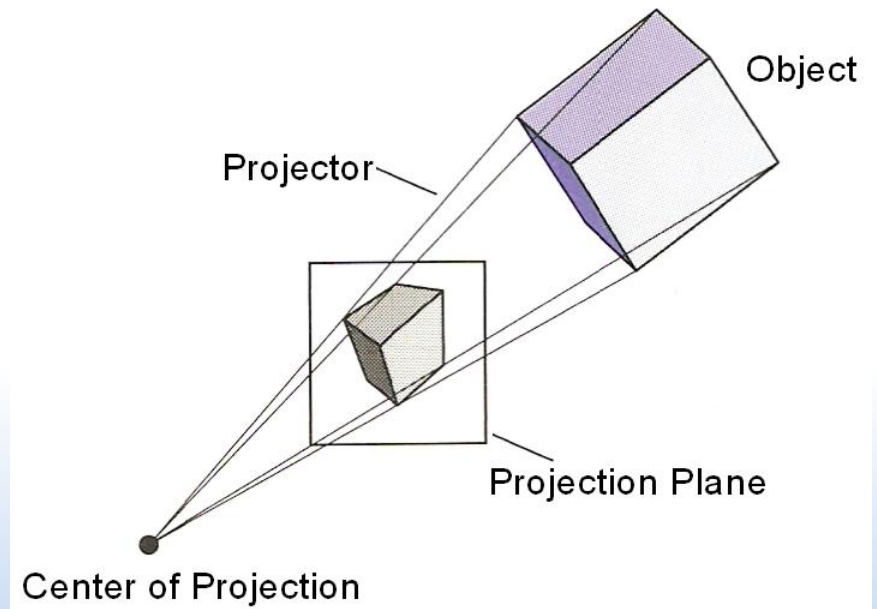
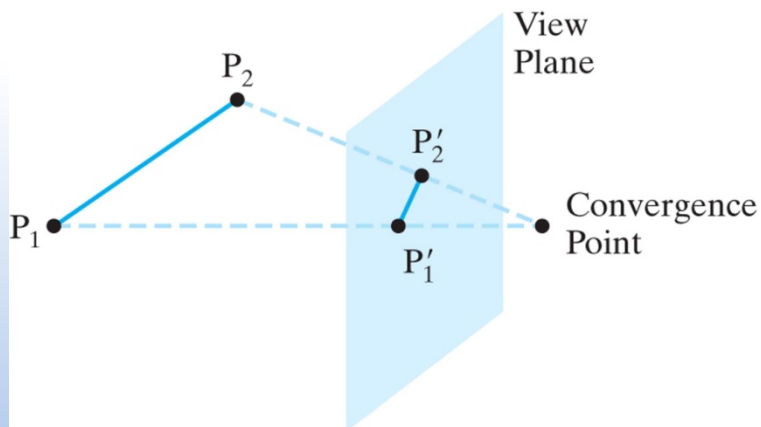
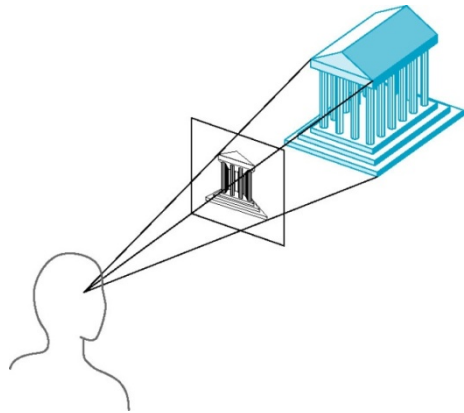
Parallel Projection

- Orthographic projection
 - Projectors orthogonal to projection surface
 - Projection lines perpendicular to view plane
 - Multi-view orthographic projection
 - Projection plane parallel to principal face
 - Usually form front, side and top orthogonal projections
 - Called views or elevations
 - Top orthogonal projection
 - Also known as plan view



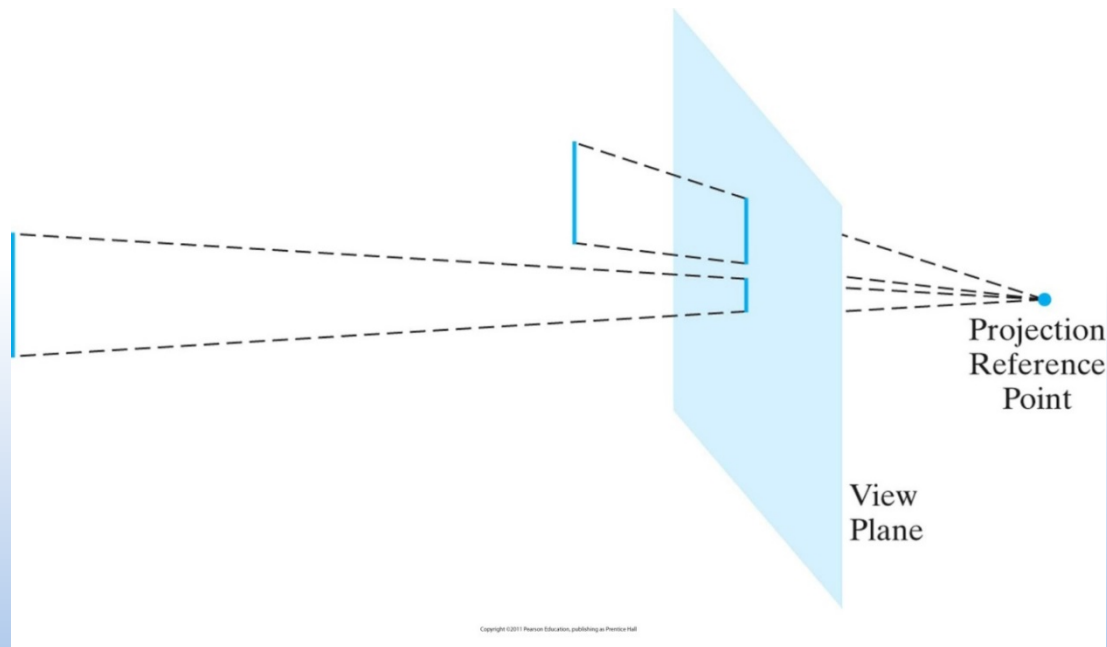
Perspective Projection

- Perspective projection
 - Projectors converge at centre of projection



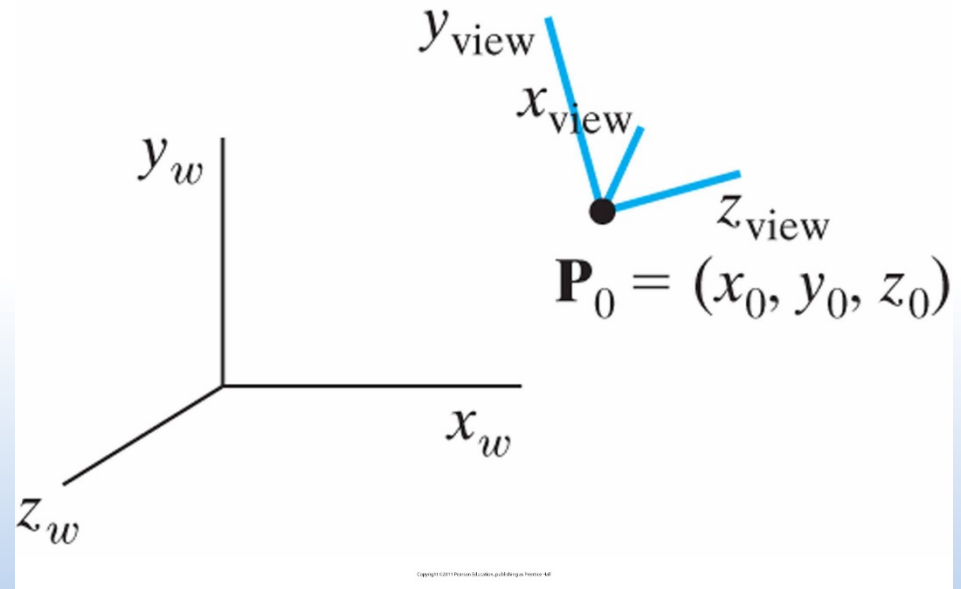
Perspective Projection

- Perspective projection
 - A perspective projection of two equal-length line segments at different distances from the view plane
 - Lines further from the view plane transferred to view plane as shorter lines



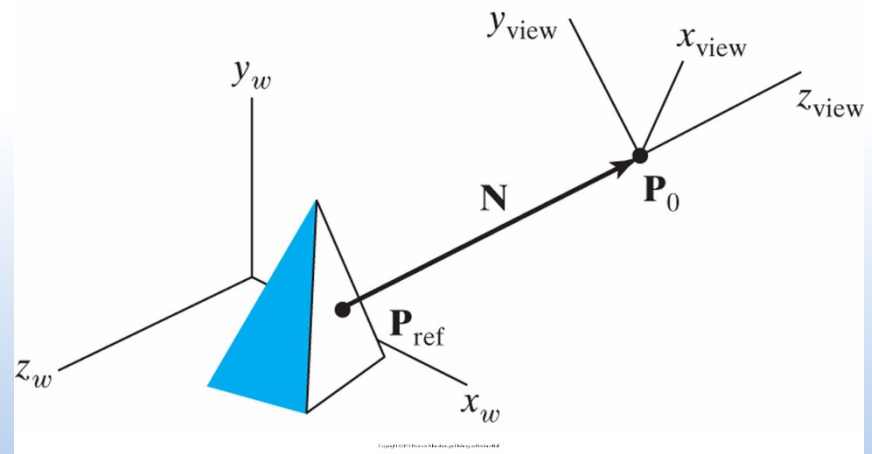
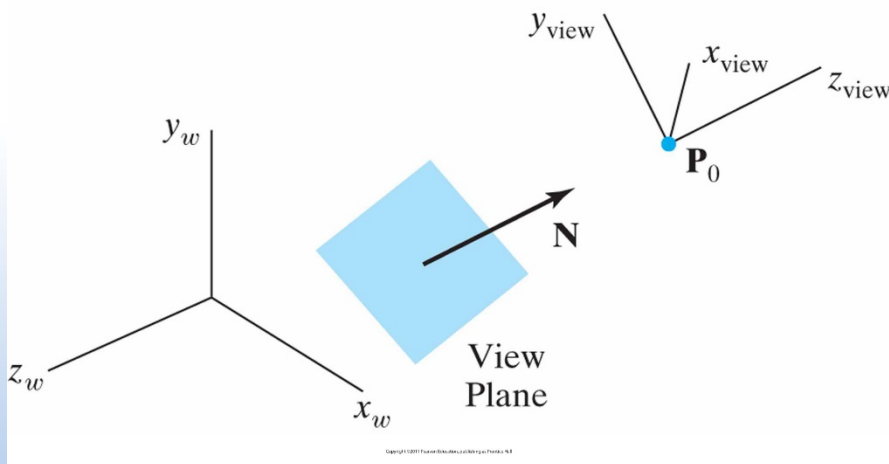
Viewing

- 3D viewing
 - A right-handed viewing coordinate system, with axes x_{view} , y_{view} , and z_{view} , relative to a right-handed world coordinate system
 - View point, \mathbf{P}_0
 - View up vector, \mathbf{V}
 - Defines the y_{view} direction



Viewing

- 3D viewing
 - Orientation of the view-plane and view-plane normal vector \mathbf{N} , in the z_{view} direction
 - \mathbf{N} can be obtained in the direction from reference point, \mathbf{P}_{ref} , to the viewing origin, \mathbf{P}_0
 - The reference point often referred to as the **look-at** point



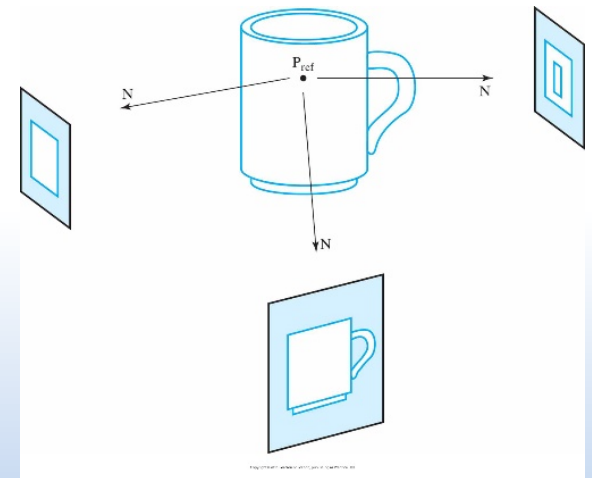
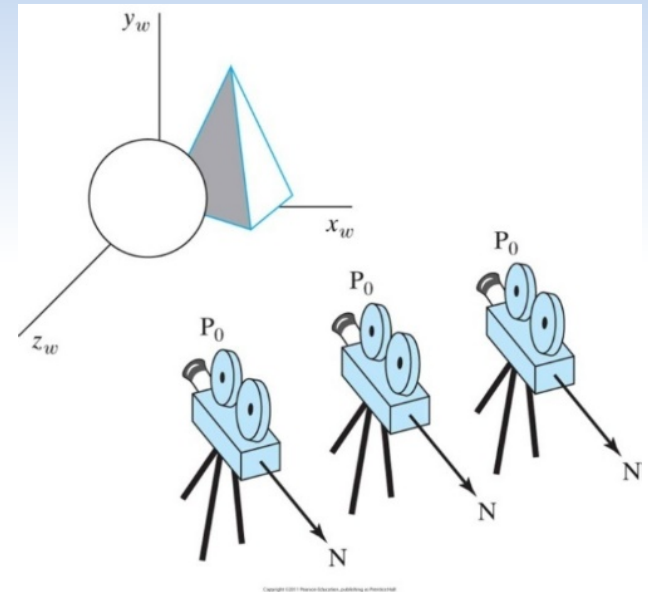
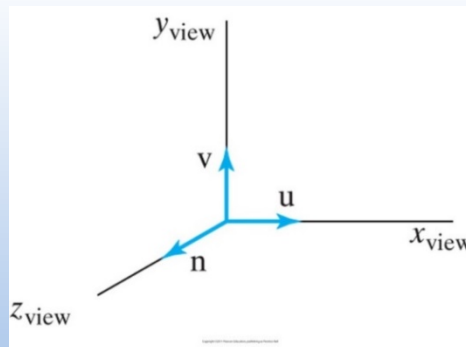
Viewing

- 3D viewing

- By varying viewing parameters can obtain different view of objects in a scene, e.g.

- Moving viewpoint P_0 , with fixed direction for N
- Viewing an object from different directions using a fixed reference point

- Viewing system defined with unit vectors u , v and n



Viewing Transformation

- Viewing transformation

- From world to view coordinates

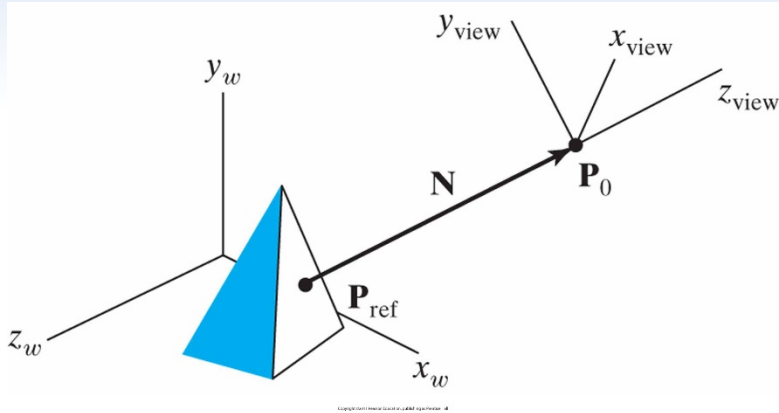
- Translate viewing coordinate origin to world coordinate origin
 - Viewing coordinate origin at world position $P_0 = (x_0, y_0, z_0)$, i.e. viewpoint in world space
- Apply rotations to align view axes with world axes, use unit vectors \mathbf{u} , \mathbf{v} , and \mathbf{n} to form composite rotation matrix
- Combine R and T for viewing transformation matrix

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

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

$$\mathbf{M}_{\text{wc,vc}} = \mathbf{RT} = \begin{bmatrix} u_x & u_y & u_z & -\vec{u} \bullet P_0 \\ v_x & v_y & v_z & -\vec{v} \bullet P_0 \\ n_x & n_y & n_z & -\vec{n} \bullet P_0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Viewing Transformation



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

$$\hat{\vec{u}} = \frac{\vec{V} \times \hat{\vec{n}}}{|\vec{V} \times \hat{\vec{n}}|} = (u_x, u_y, u_z)$$

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

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

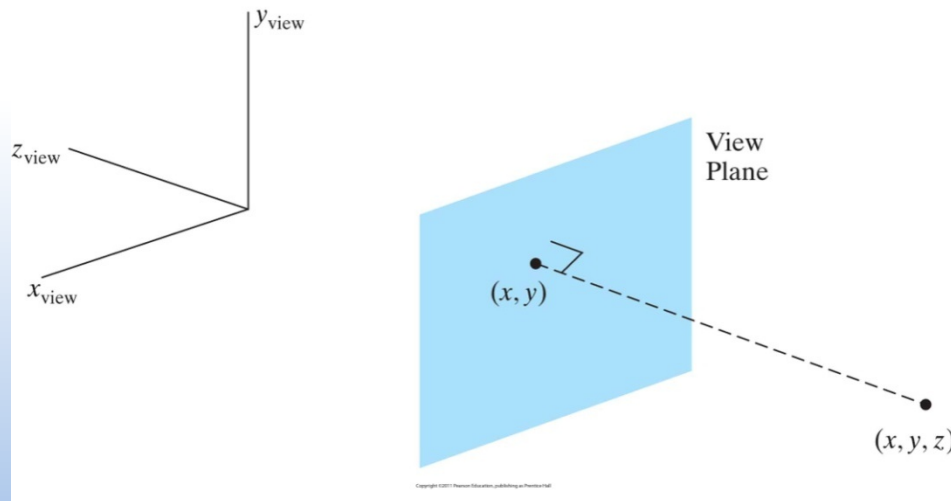
- GLM

```
glm::lookAt( P0, Pref, V )
```

```
glm::mat4 viewMatrix = glm::lookAt(  
    glm::vec3(0.0f, 0.0f, 5.0f),  
    glm::vec3(0.0f, 0.0f, 0.0f),  
    glm::vec3(0.0f, 1.0f, 0.0f));
```

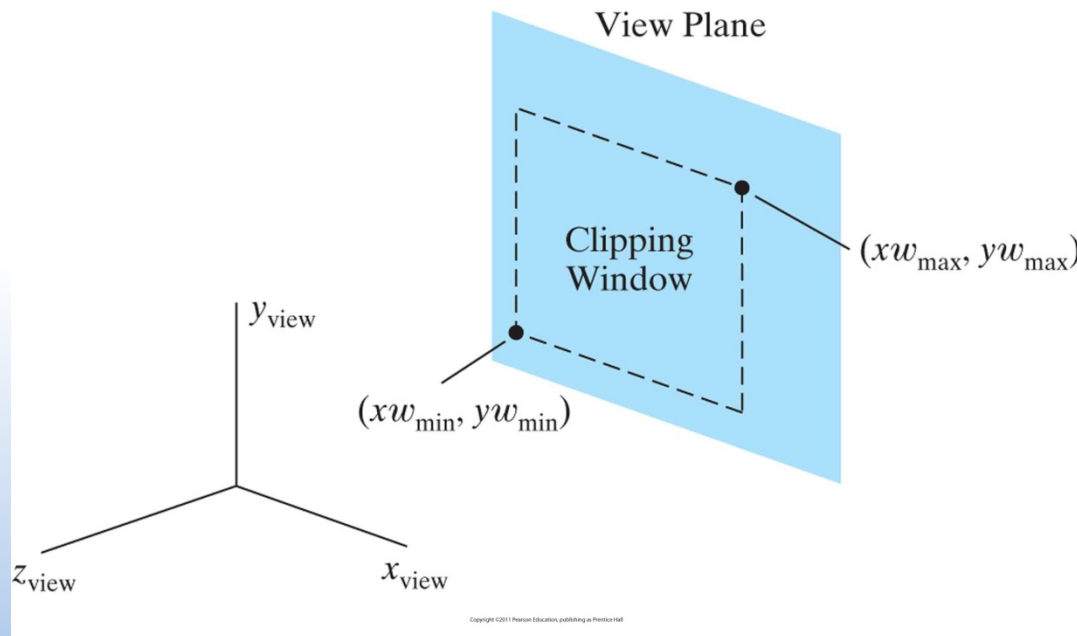
Orthogonal Projection View Volume

- Orthogonal projection
 - An orthogonal projection of a spatial position onto a view plane
 - Projection direction parallel to the z-axis, projection is trivial
 - Any position (x, y, z) in viewing coordinates are projected to (x, y)
 - z coordinate preserved for use in visibility determination



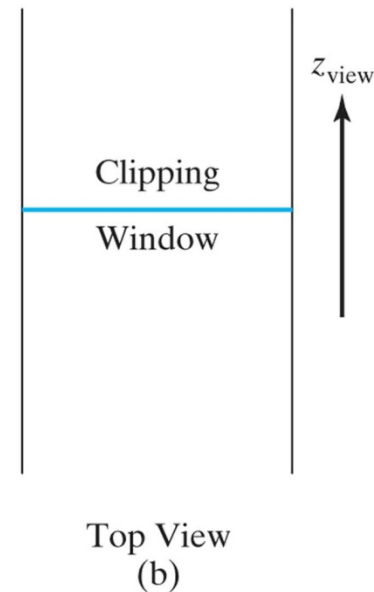
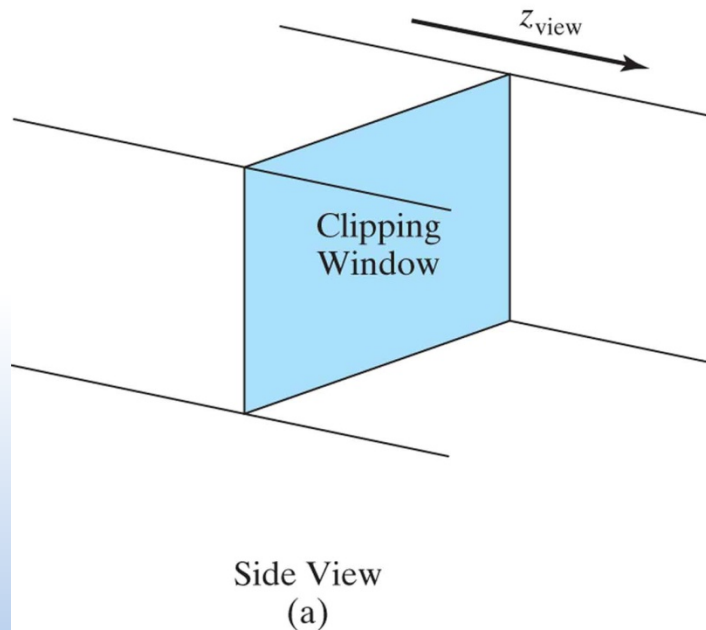
Orthogonal Projection View Volume

- Clipping window
 - A clipping window on the view plane
 - Similar to 2D viewing, typically clipping rectangle with minimum and maximum coordinates given in the viewing reference system



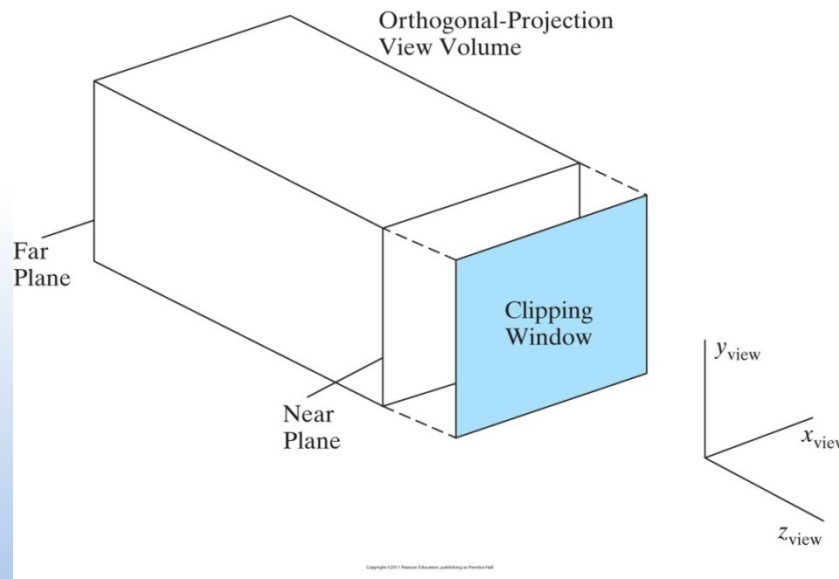
Orthogonal Projection View Volume

- View volume
 - How much of the scene to transfer to the view plane
 - An infinite orthogonal-projection view volume only clips object descriptions to four boundary planes



Orthogonal Projection View Volume

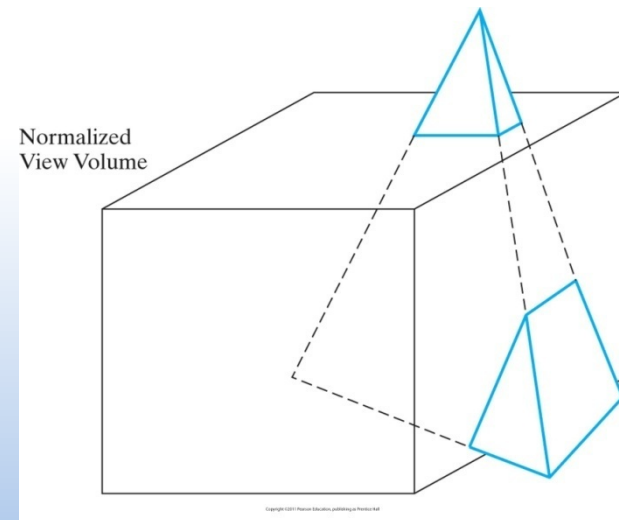
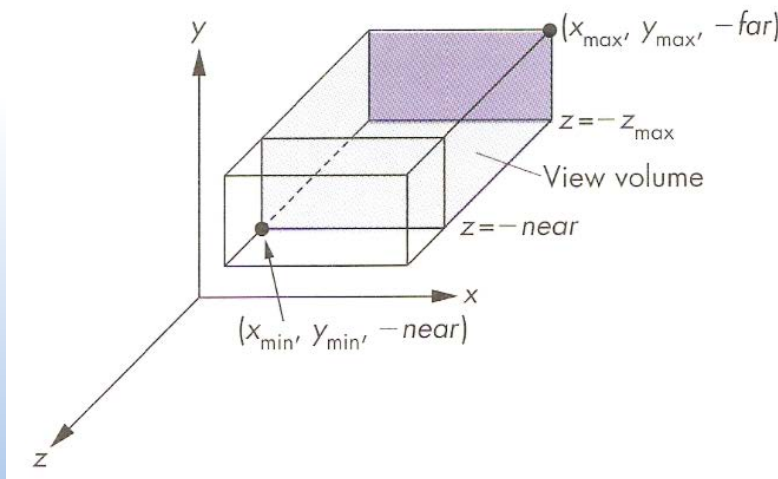
- View volume
 - Limit the extent in the z-direction by selecting two additional, near and far, boundary planes
 - A finite orthogonal-projection view volume with the view plane “in front” of the near plane
 - Forms a rectangular parallelepiped



Orthogonal Projection View Volume

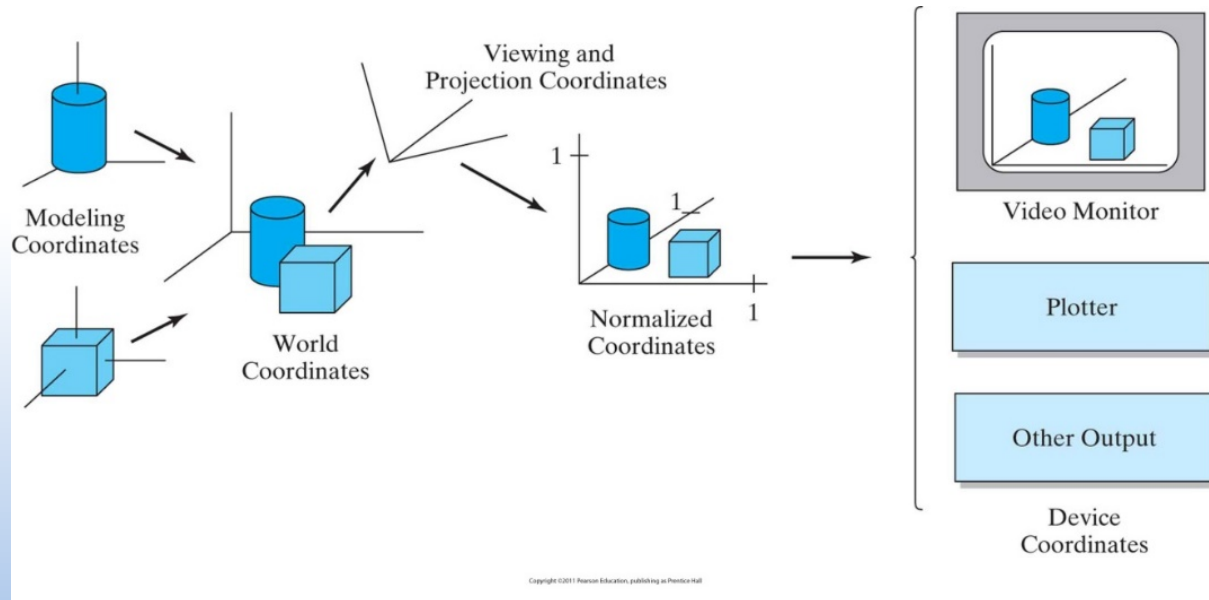
- Clipping

- View of the scene will only contain objects within the view volume
 - All parts of the scene outside eliminated by a 3D clipping algorithm
- Some graphics packages allow for additional clipping planes



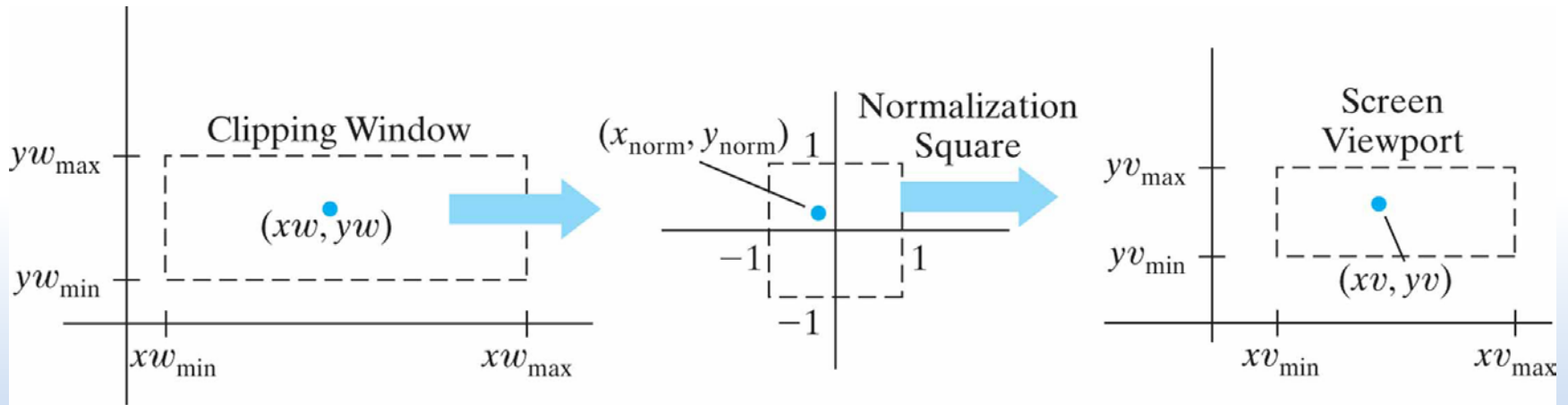
Normalization Transformation

- Normalized device coordinates (NDC)
 - Why NDC?
 - Representation where a graphics package is independent of the coordinate range of any specific output device
 - Intermediate coordinate system, maps easily to different device coordinates



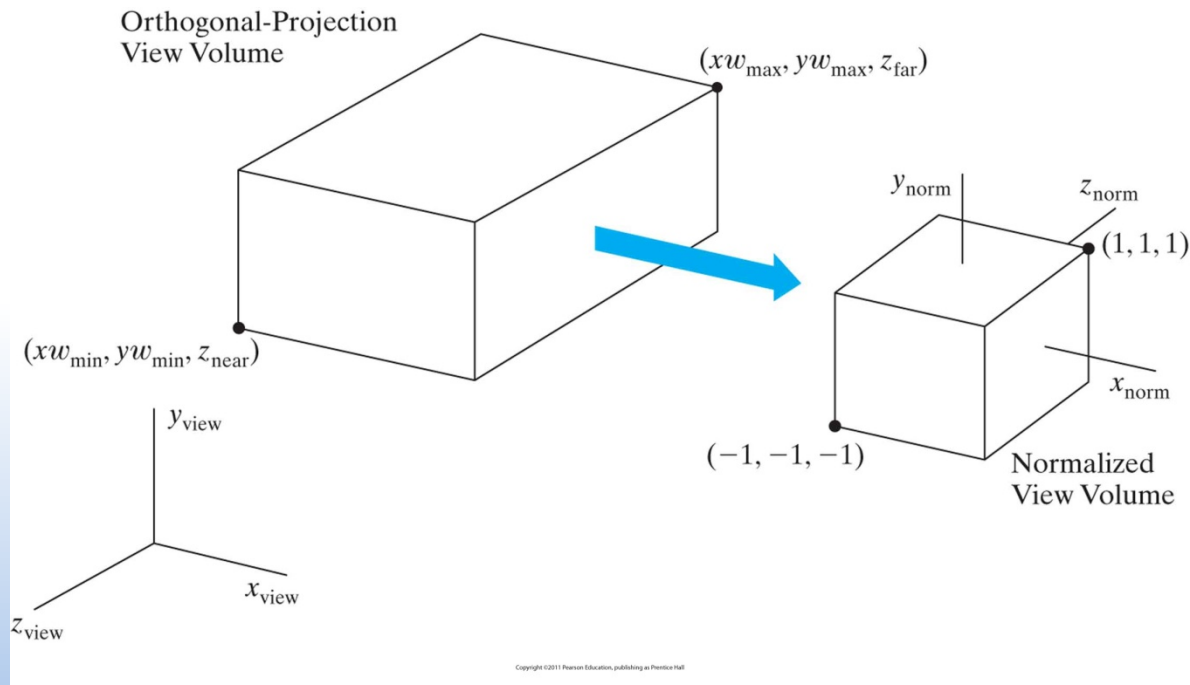
Normalization Transformation

- Normalized device coordinates (NDC)
 - Usually either unit cube or symmetric cube with coordinates between -1 to 1
 - Intermediate coordinate system
 - Depends on neither the original application units nor the particulars of the display device

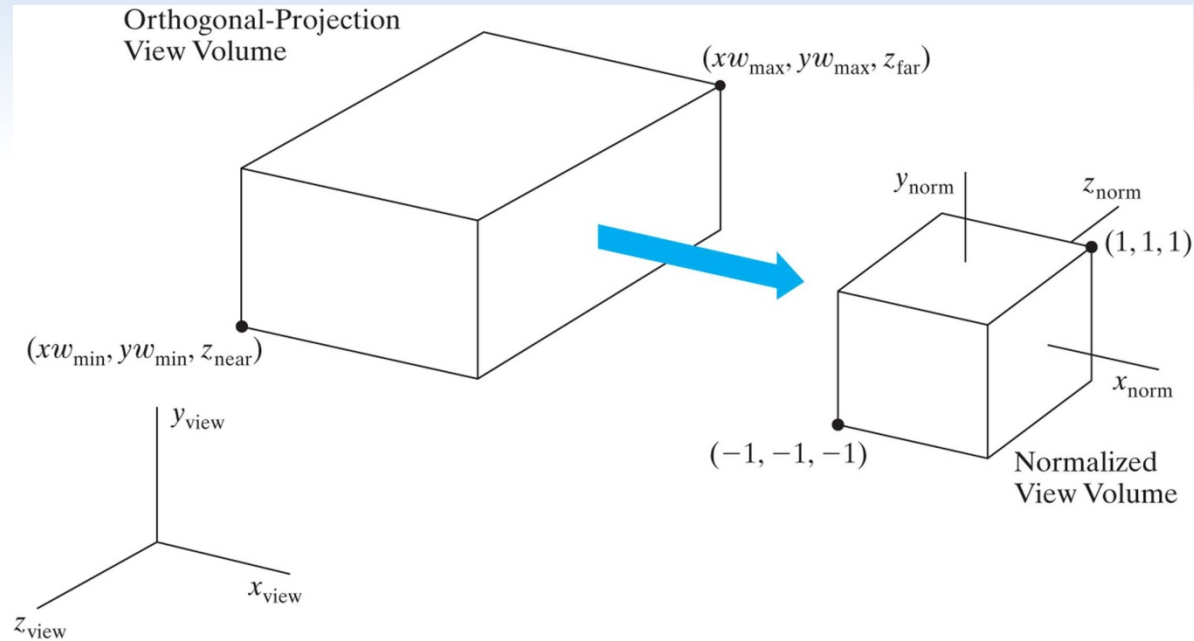


Normalization Transformation

- Normalization transformation for orthographic projection
 - From an orthogonal-projection view volume to the symmetric normalization cube



Normalization Transformation



Copyright ©2011 Pearson Education, publishing as Prentice Hall

$$\mathbf{T} = \mathbf{T} \left(-\frac{(xw_{max} + xw_{min})}{2}, -\frac{(yw_{max} + yw_{min})}{2}, \frac{(z_{near} + z_{far})}{2} \right)$$

$$\mathbf{S} = \mathbf{S} \left(\frac{2}{(xw_{max} - xw_{min})}, \frac{2}{(yw_{max} - yw_{min})}, \frac{2}{(z_{near} - z_{far})} \right)$$

Normalization Transformation

$$\mathbf{M}_{\text{ortho,norm}} = \mathbf{S}\mathbf{T}$$

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

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

Orthographic Projection

- GLM

```
glm::ortho( left, right, bottom, top, zNear, zFar )
```

- Default

- Identity matrix

```
glm::ortho(-1.0f, 1.0f, -1.0f, 1.0f, -1.0f, 1.0f);
```

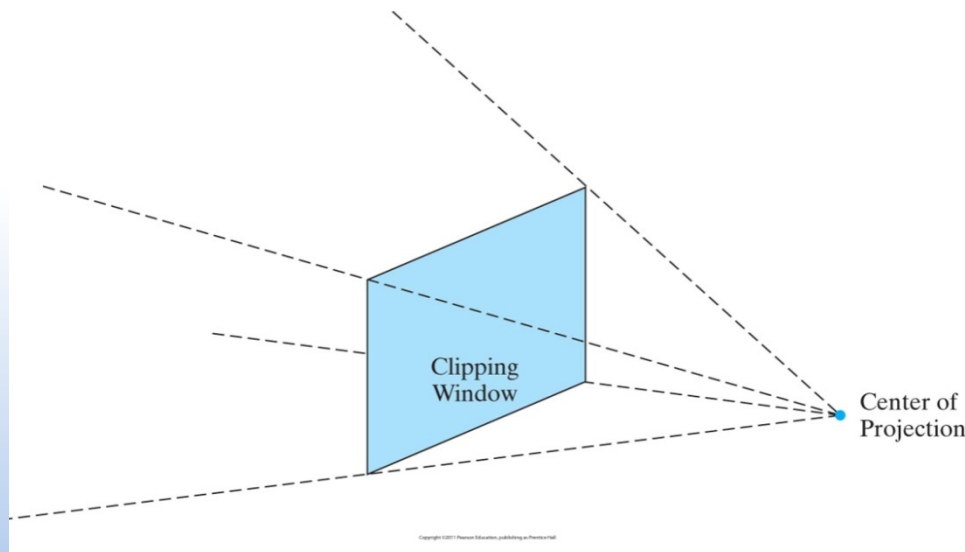
- Example: to match screen space coordinates

- If display resolution is 600 x 800
- For orthographics projection coordinates to match screen coordinates

```
glm::ortho(0.0f, 800.0f, 0.0f, 600.0f, 0.1f, 10.0f);
```

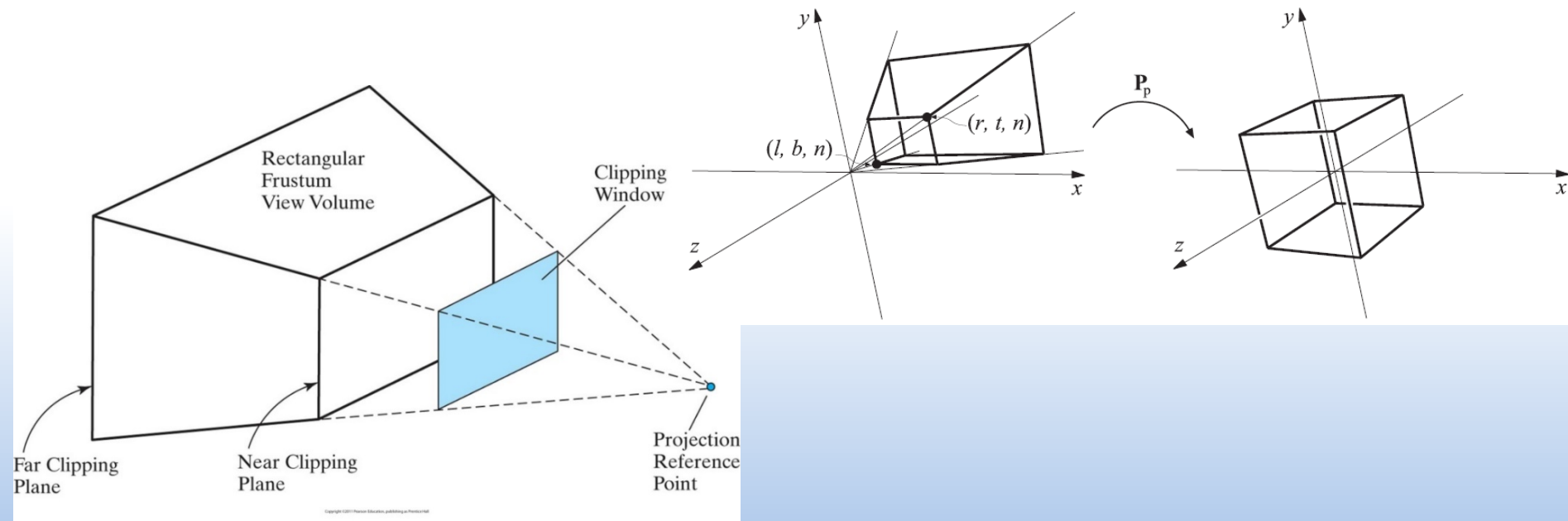
Perspective Projection View Volume

- Perspective projection
 - Viewing volume created by specifying rectangular clipping window on view plane
 - An infinite pyramid view volume for a perspective projection
 - Projection lines not parallel, so bounding planes of view volume not parallel



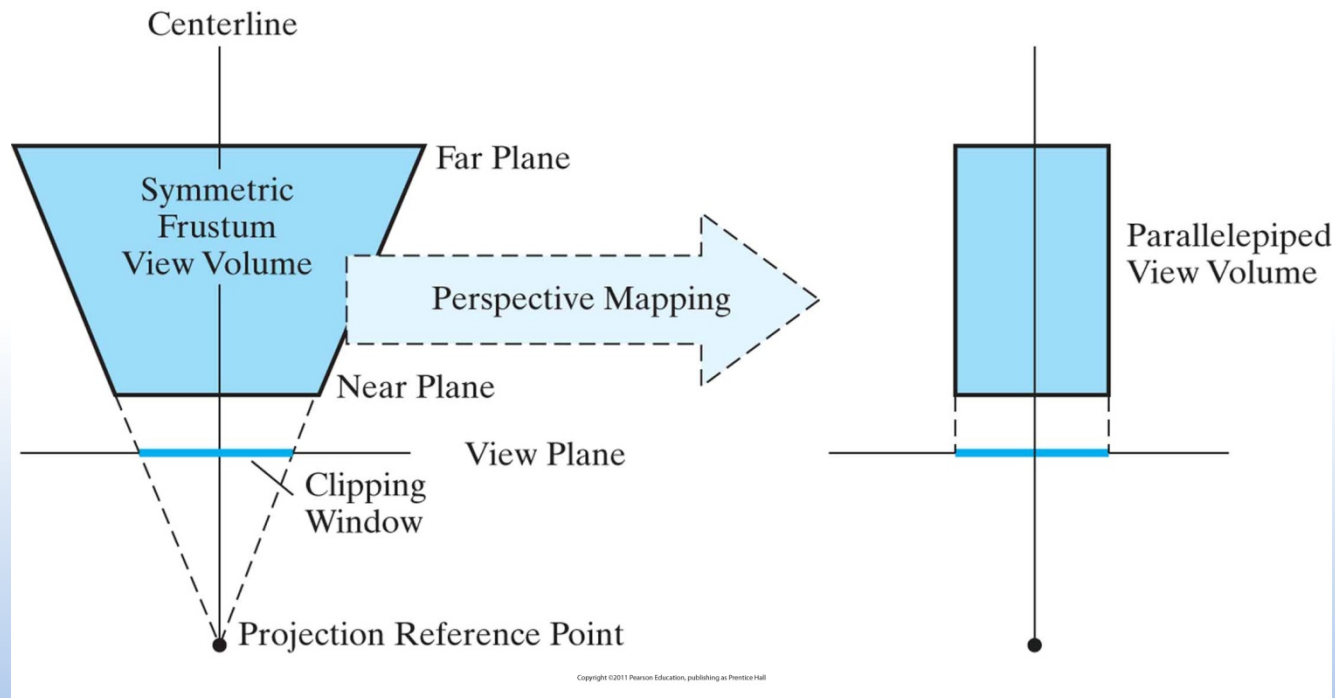
Perspective Projection View Volume

- View frustum
 - Finite view volume by specifying near and far clipping planes
 - A perspective-projection frustum view volume with the view plane 'in front' of the near clipping plane



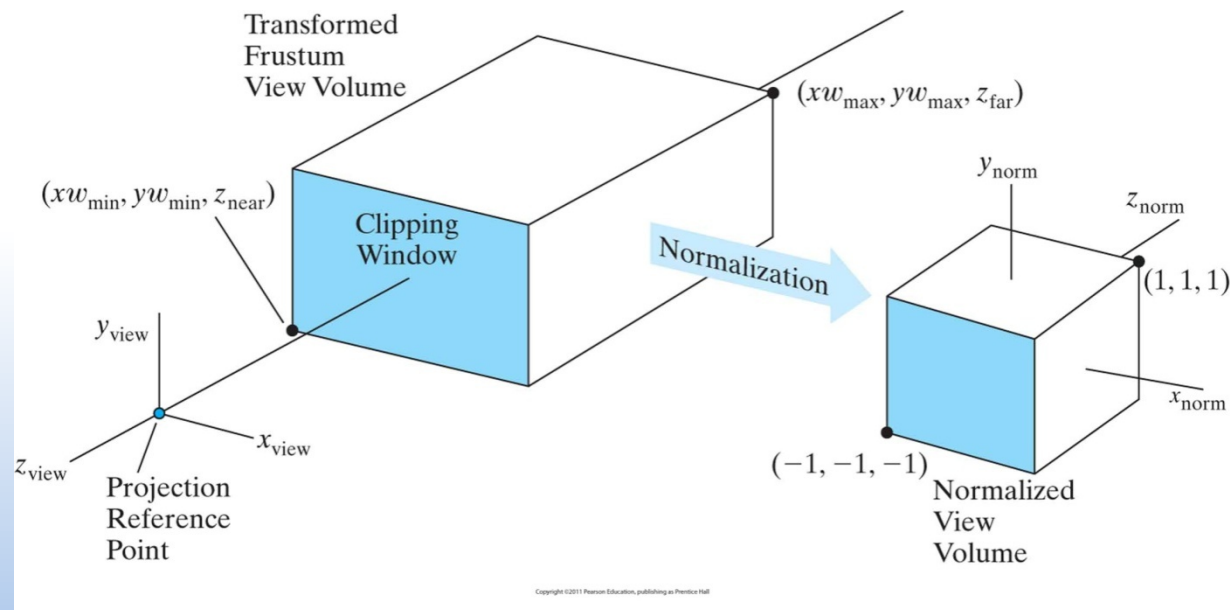
Normalized Transformation

- Perspective projection transformation
 - A symmetric frustum view volume mapped to an orthogonal parallelepiped by a perspective projection transformation

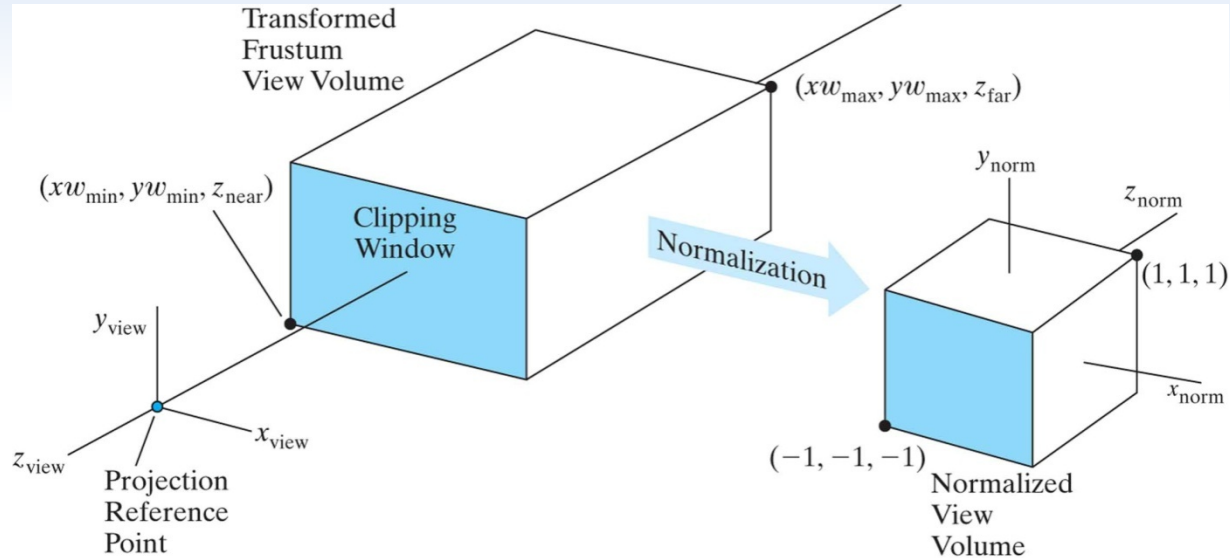


Normalized Transformation

- Normalized transformation for perspective projection
 - From a transformed perspective projection view volume (rectangular parallelepiped) to the symmetric normalization cube



Normalized Transformation

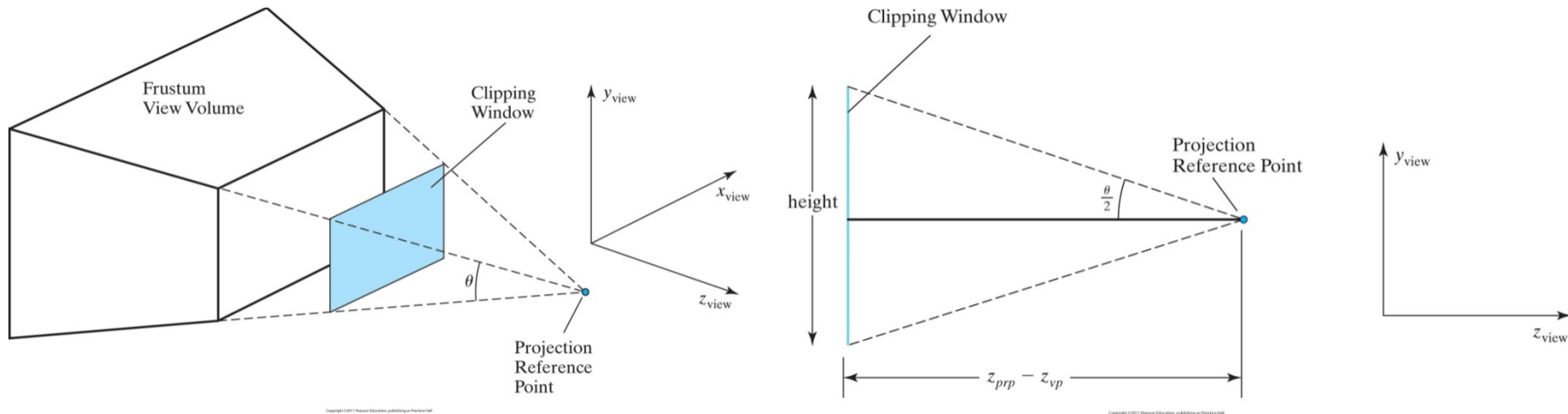


Copyright ©2011 Pearson Education, publishing as Prentice Hall

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

Normalized Transformation

➤ For a symmetric view frustum



$$\mathbf{M}_{\text{symmper,norm}} = \begin{bmatrix} \frac{\cot\left(\frac{\theta}{2}\right)}{\text{aspect}} & 0 & 0 & 0 \\ 0 & \cot\left(\frac{\theta}{2}\right) & 0 & 0 \\ 0 & 0 & \frac{z_{near} + z_{far}}{z_{near} - z_{far}} & -\frac{2z_{near}z_{far}}{z_{near} - z_{far}} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

Projection Transformation

- GLM

```
glm::perspective( FoV, aspectRatio, zNear, zFar )
```

```
int width, height;
```

```
glfwGetFramebufferSize(window, &width, &height);
```

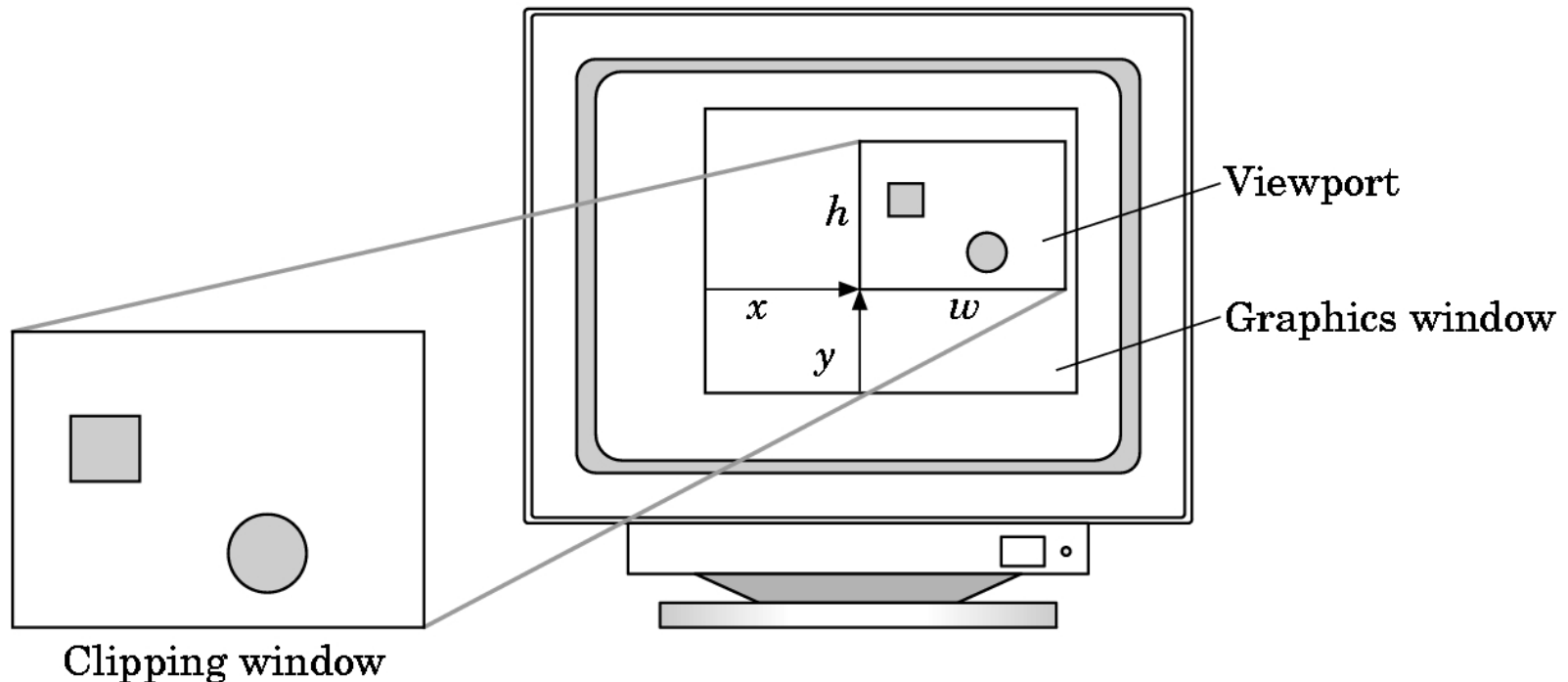
```
float aspectRatio = static_cast<float>(width) / height;
```

```
glm::mat4 projectionMatrix =
```

```
    glm::perspective(glm::radians(45.0f), aspectRatio,  
    0.1f, 100.0f);
```

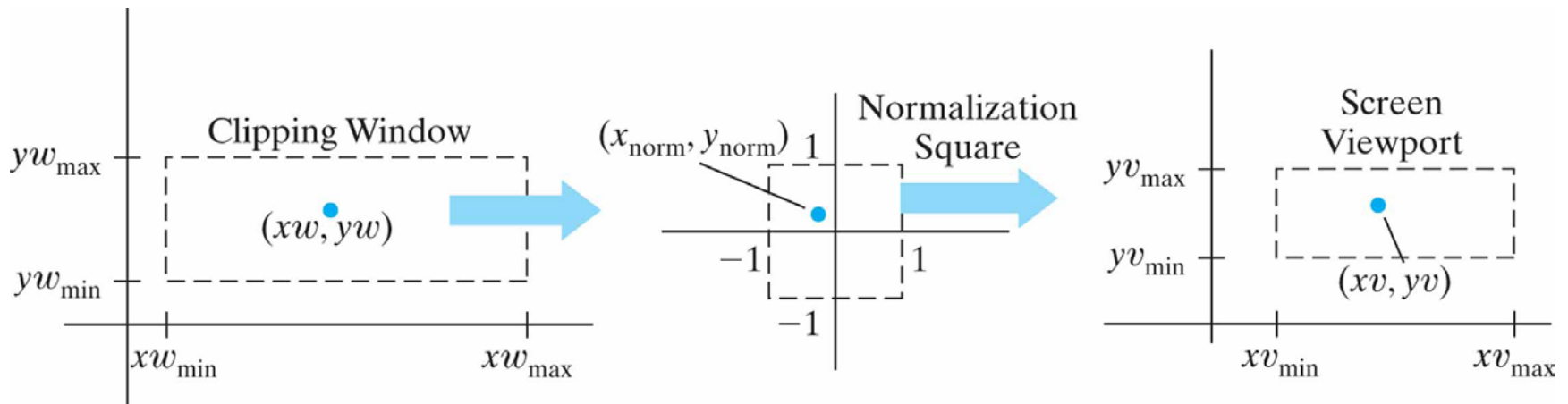
Viewport Transformation

- Viewport transformation
 - Do not have use the entire window for the image
`glViewport(x , y , w , h);`
 - Values in pixels (screen coordinates)



Viewport Transformation

- Viewport transformation

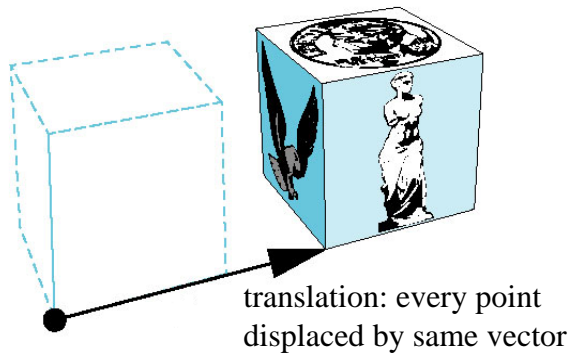


Copyright © 2011 Pearson Education, publishing as Prentice Hall

$$\mathbf{M}_{\text{normsquare,viewport}} = \begin{bmatrix} \frac{x_{v_{\max}} - x_{v_{\min}}}{2} & 0 & 0 & \frac{x_{v_{\max}} + x_{v_{\min}}}{2} \\ 0 & \frac{y_{v_{\max}} - y_{v_{\min}}}{2} & 0 & \frac{y_{v_{\max}} + y_{v_{\min}}}{2} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

3D Transformations

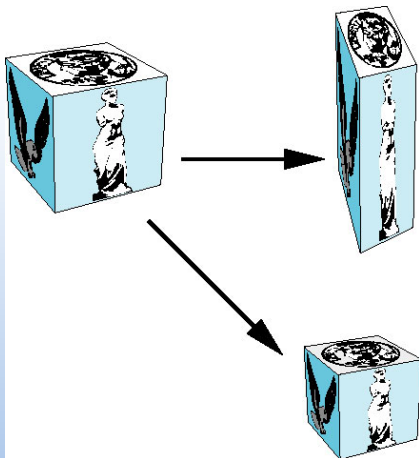
- 3D translation



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

$$P' = T P$$

- 3D scaling

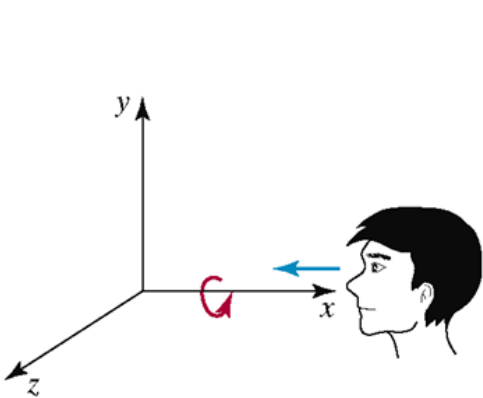


$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

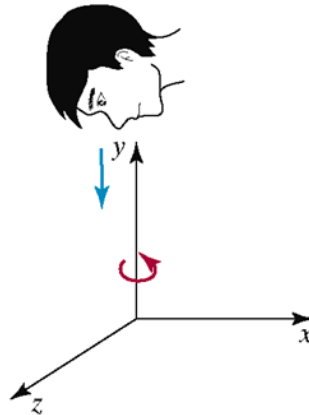
$$P' = S P$$

3D Transformations

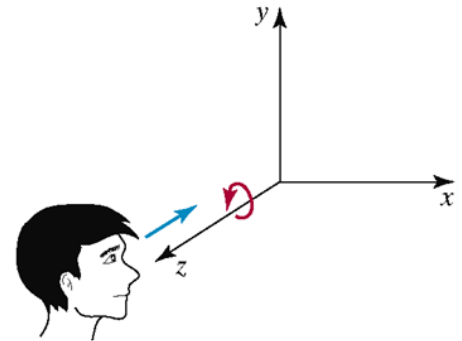
- 3D rotation
 - Rotation axes parallel to coordinate axes



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



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



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

$$P' = R_{axis}(\theta) P$$

3D Transformations

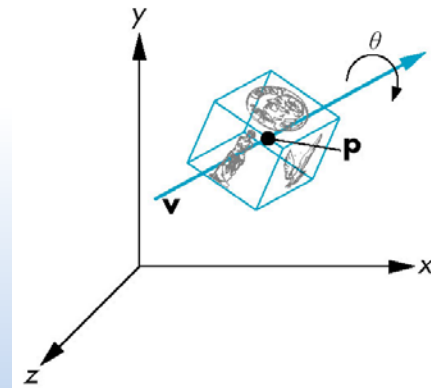
- Rotation by θ radians about an arbitrary normalised axis (r_x, r_y, r_z)

$$R = \begin{bmatrix} \cos \theta + (1 - \cos \theta)r_x^2 & (1 - \cos \theta)r_x r_y - r_z \sin \theta & (1 - \cos \theta)r_x r_z + r_y \sin \theta & 0 \\ (1 - \cos \theta)r_x r_y + r_z \sin \theta & \cos \theta + (1 - \cos \theta)r_y^2 & (1 - \cos \theta)r_y r_z - r_x \sin \theta & 0 \\ (1 - \cos \theta)r_x r_z - r_y \sin \theta & (1 - \cos \theta)r_y r_z + r_x \sin \theta & \cos \theta + (1 - \cos \theta)r_z^2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Direction of positive rotation depends on the “handedness” of the system



For a left-handed coordinate system



For a right-handed coordinate system

Transformations

- GLM

```
#include <glm/glm.hpp>
#include <glm/gtx/transform.hpp>
glm::mat4 translationMatrix =
    glm::translate(glm::vec3(1.0f, 1.0f, 1.0f));
glm::mat4 scalingMatrix =
    glm::scale(glm::vec3(2.0f, 2.0f, 2.0f));
glm::mat4 rotationMatrix =
    glm::rotate(angleInRadians,
    glm::vec3(0.0f, 1.0f, 0.0f));
```

Viewing and Projection

- Camera class

- Contains public member functions to

- Update position and orientation

```
void update(float moveForward, float moveRight,  
            float moveUp = 0.0f);  
void updateRotation(float deltaYaw, float deltaPitch);
```

- Set view and projection matrices

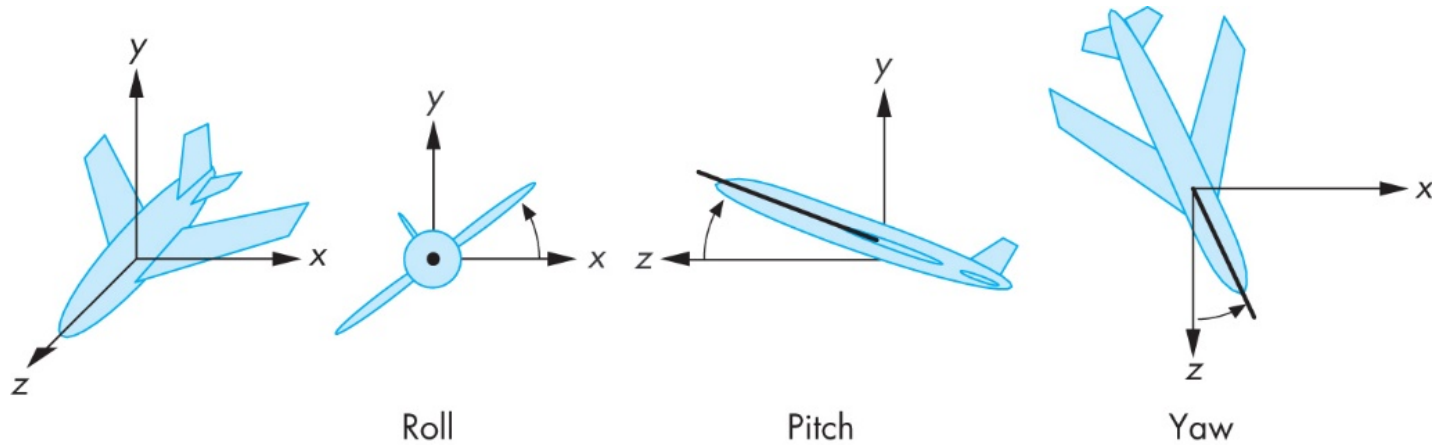
```
void setViewMatrix(glm::vec3 position, glm::vec3 lookAt);  
void setProjMatrix(glm::mat4 projMatrix);
```

- Get camera information

```
glm::mat4 getViewMatrix();  
glm::mat4 getProjMatrix();  
glm::vec3 getPosition();  
glm::vec3 getDirection();
```

Viewing and Projection

- Camera class
 - Yaw and pitch



Viewing and Projection

- Camera class

```
setViewMatrix()
```

- Calculate up vector

- Direction of camera

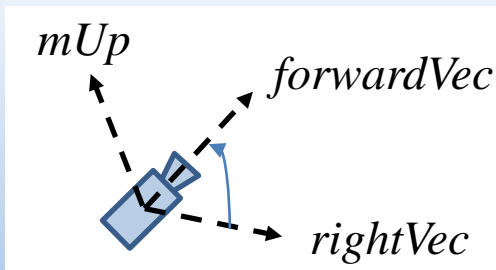
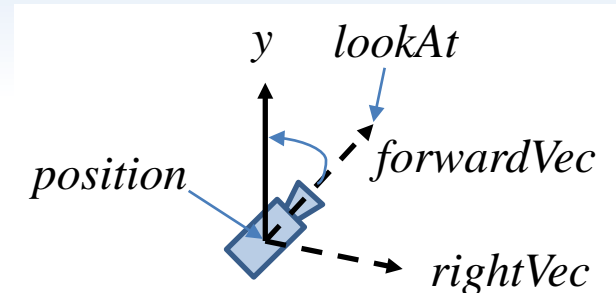
```
glm::vec3 forwardVec = glm::normalize(lookAt - position);
```

- Vector to the right of camera direction

```
glm::vec3 rightVec = glm::cross(forwardVec,  
    glm::vec3(0.0f, 1.0f, 0.0f));
```

- Up vector

```
mUp = glm::normalize(glm::cross(rightVec, forwardVec));
```



- Vector cross product

- Results in a vector perpendicular to both input vectors

Viewing and Projection

➤ Calculate initial yaw and pitch

- Project forward vector onto the xz-plane

```
glm::vec3 hLookAtVec = glm::vec3(forwardVec.x,  
    0.0f, forwardVec.z);
```

```
glm::vec3 vLookAtVec = glm::vec3(0.0f,  
    forwardVec.y, forwardVec.z);
```

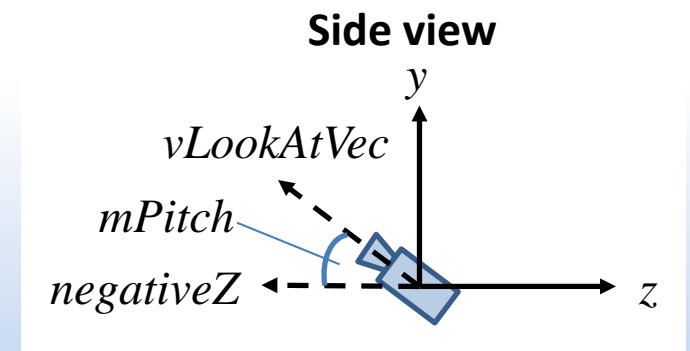
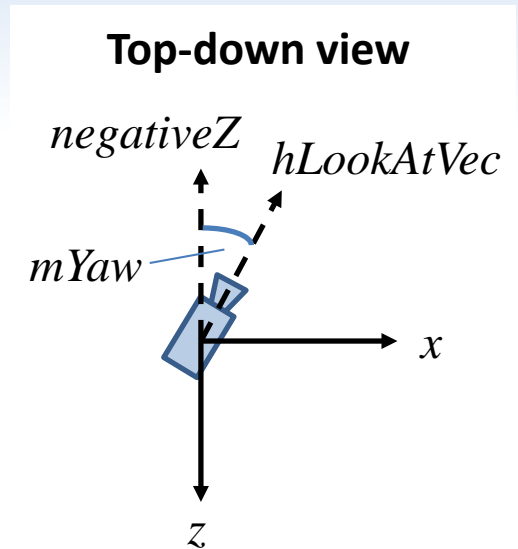
```
glm::vec3 negativeZ(0.0f, 0.0f, -1.0f);
```

- Use vector dot product to find angle

```
mYaw = acos(glm::dot(negativeZ,  
    glm::normalize(hLookAtVec)));
```

```
mPitch = acos(glm::dot(negativeZ,  
    glm::normalize(vLookAtVec)));
```

- Determine whether +ve or -ve



Viewing and Projection

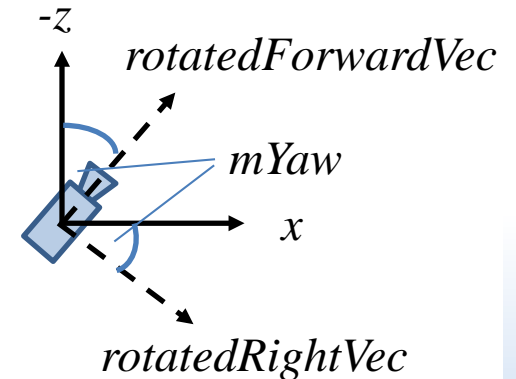
- Camera class

update()

➤ Rotate forward and right vectors by the yaw

```
glm::vec3 rotatedForwardVec = glm::vec3(
    glm::rotate(mYaw,
        glm::vec3(0.0f, 1.0f, 0.0f))
    * glm::vec4(0.0f, 0.0f, -1.0f, 0.0f));
glm::vec3 rotatedRightVec = glm::vec3(
    glm::rotate(mYaw,
        glm::vec3(0.0f, 1.0f, 0.0f))
    * glm::vec4(1.0f, 0.0f, 0.0f, 0.0f));
```

Top-down view



Viewing and Projection

➤ Rotate forward by the pitch

```
rotatedForwardVec = glm::vec3(  
    glm::rotate(mPitch, rotatedRightVec)  
    * glm::vec4(rotatedForwardVec, 0.0f));
```

➤ Move camera position

- Scale forward and right vectors by movement amount, then perform vector addition

```
mPosition += rotatedForwardVec * moveForward  
+ rotatedRightVec * moveRight;
```

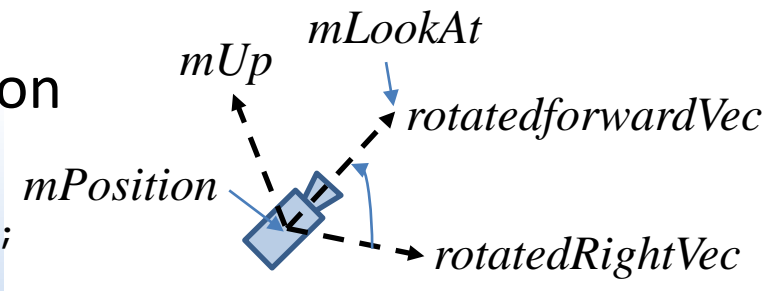
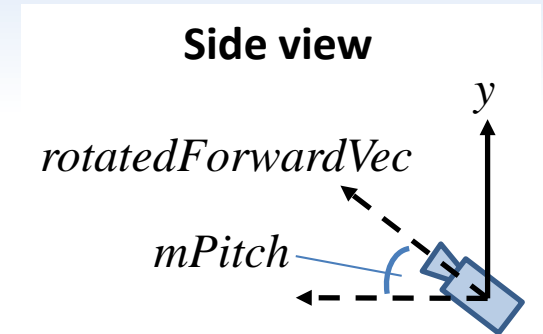
➤ Set look-at to be in front of position

- Move a point by a vector

```
mLookAt = mPosition + rotatedForwardVec;
```

➤ Find up vector

```
mUp = glm::cross(rotatedRightVec, rotatedForwardVec);
```



References

- Among others, material sourced from
 - Hearn, Baker & Carithers, “Computer Graphics with OpenGL”, Pearson/Prentice-Hall
 - Angel & Shreiner, “Interactive Computer Graphics: A Top-Down Approach with OpenGL”, Pearson/Addison Wesley
 - Akenine-Moller, Haines & Hoffman, “Real Time Rendering”, A.K. Peters
 - Joey de Vries, “Learn OpenGL,” <https://learnopengl.com/>
 - <http://en.wikipedia.org/wiki/>