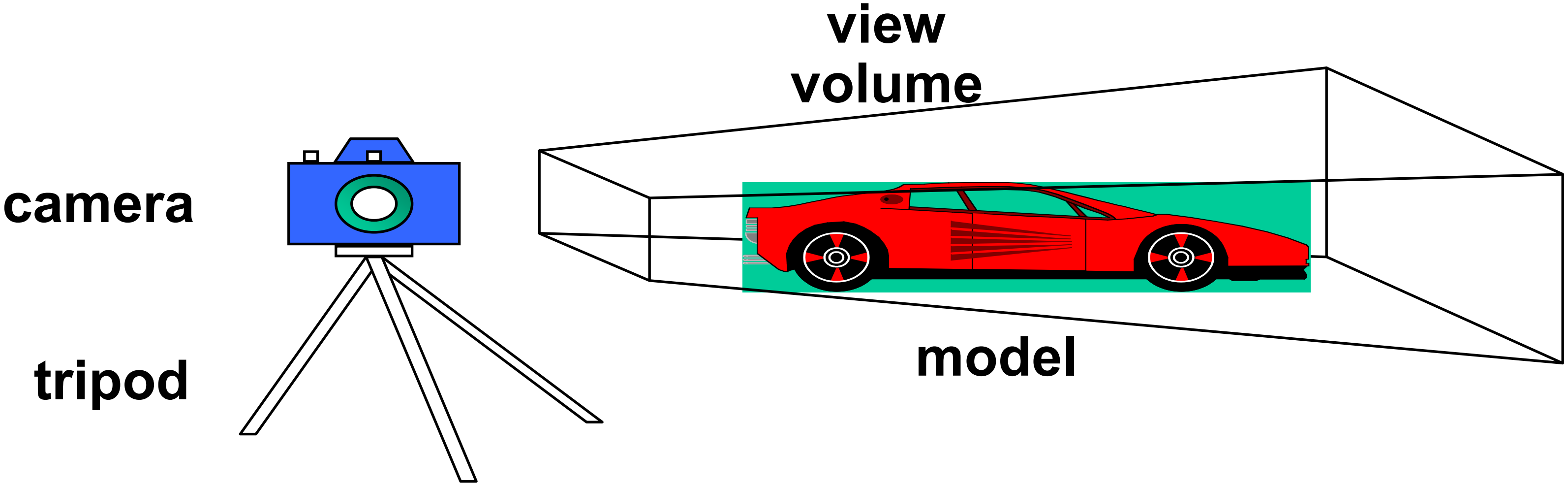
Computer Viewing

Prof. George Wolberg
Dept. of Computer Science
City College of New York

Objectives

- Introduce the mathematics of projection
- Introduce OpenGL viewing functions
- Look at alternate viewing APIs

Viewing Process

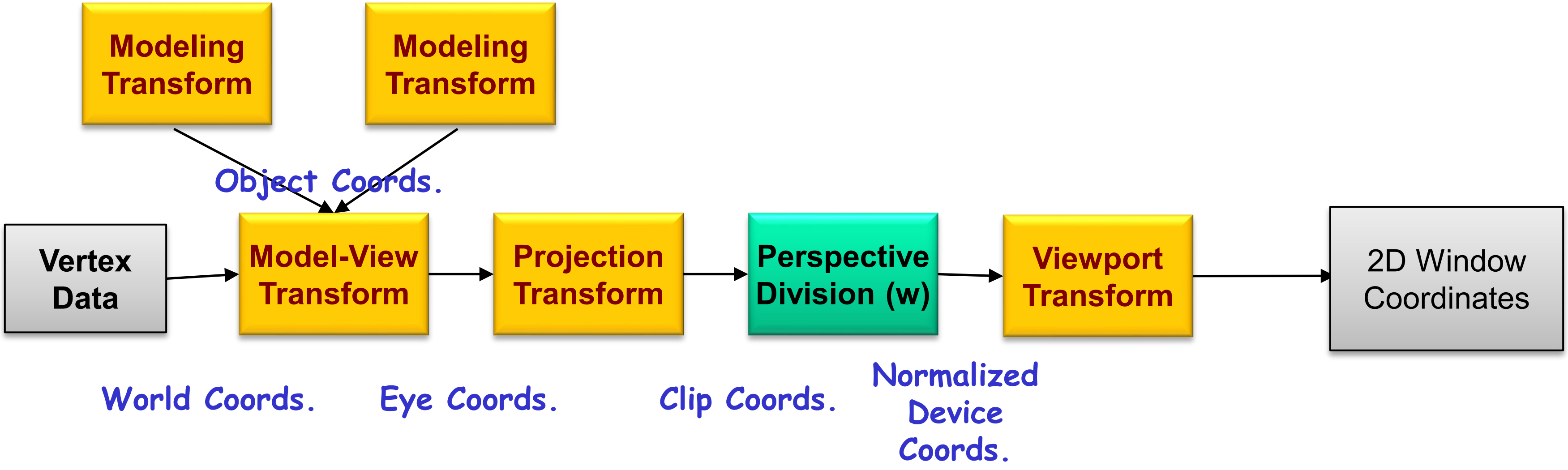


Computer Viewing

- There are three aspects of the viewing process, all of which are implemented in a pipeline,
  - Positioning the camera
    - Setting the model-view matrix
  - Selecting a lens
    - Setting the projection matrix
  - Clipping
    - Setting the view volume

Transformation Pipeline

- Transformations take us from one "space" to another
  - All of our transforms are 4×4 matrices



Transformations

- Modeling transformations
  - move models into world coordinate system
- Viewing transformations
  - define position and orientation of the camera
- Projection transformations
  - adjust the lens of the camera; define view volume
- Viewport transformations
  - enlarge or reduce the physical photograph

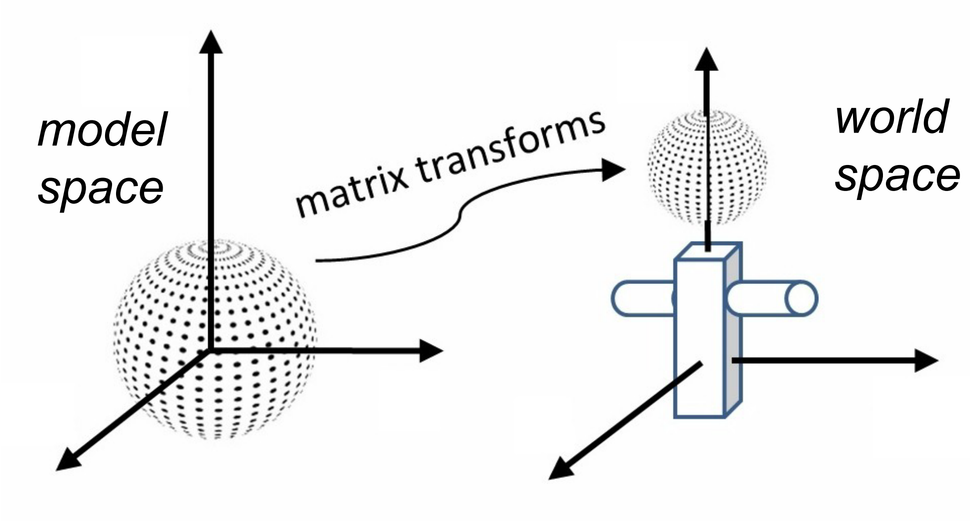
Modeling Transformations

Local, or “Model” space

- The space in which a model is defined
- Usually centered at the origin

“World” space

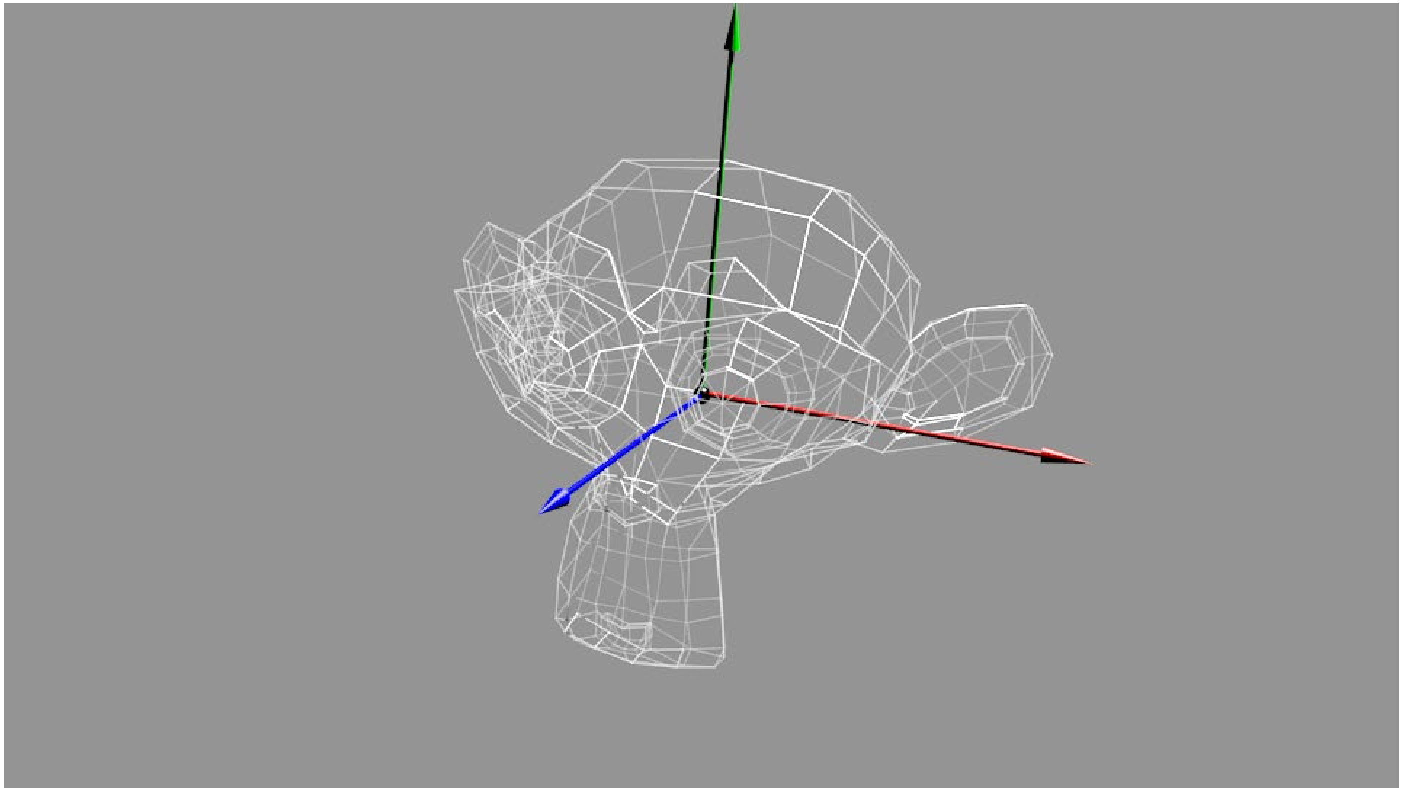
- The space in which the models are assembled/collected
- Dimensions and orientation conforms to simulated scene



*The matrix transforms (concatenated) that place an object in world space is called its Model matrix, or **M***

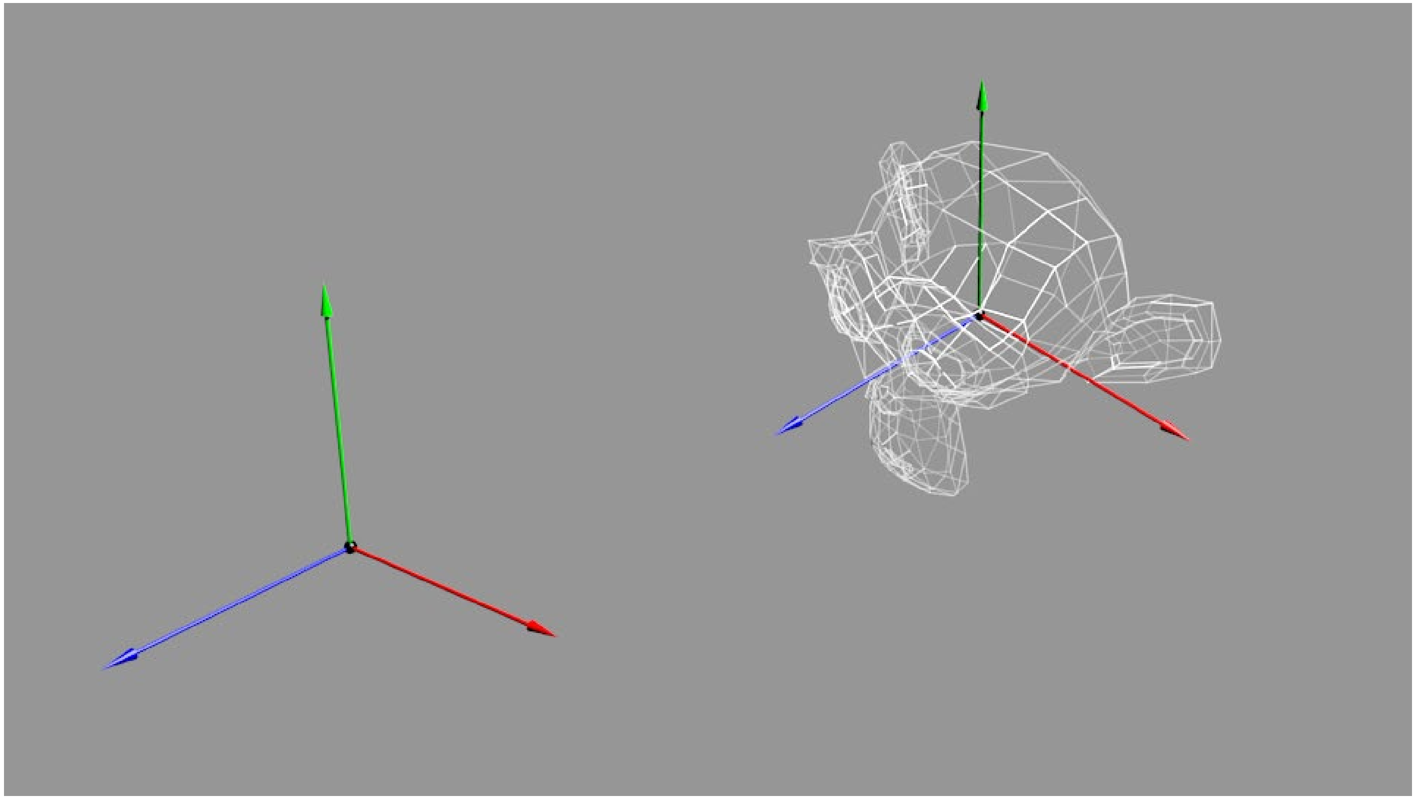
The Model Coordinate System

The X,Y,Z coordinates of the model's vertices are defined relative to the object's center, where (0,0,0) is the center of the object.



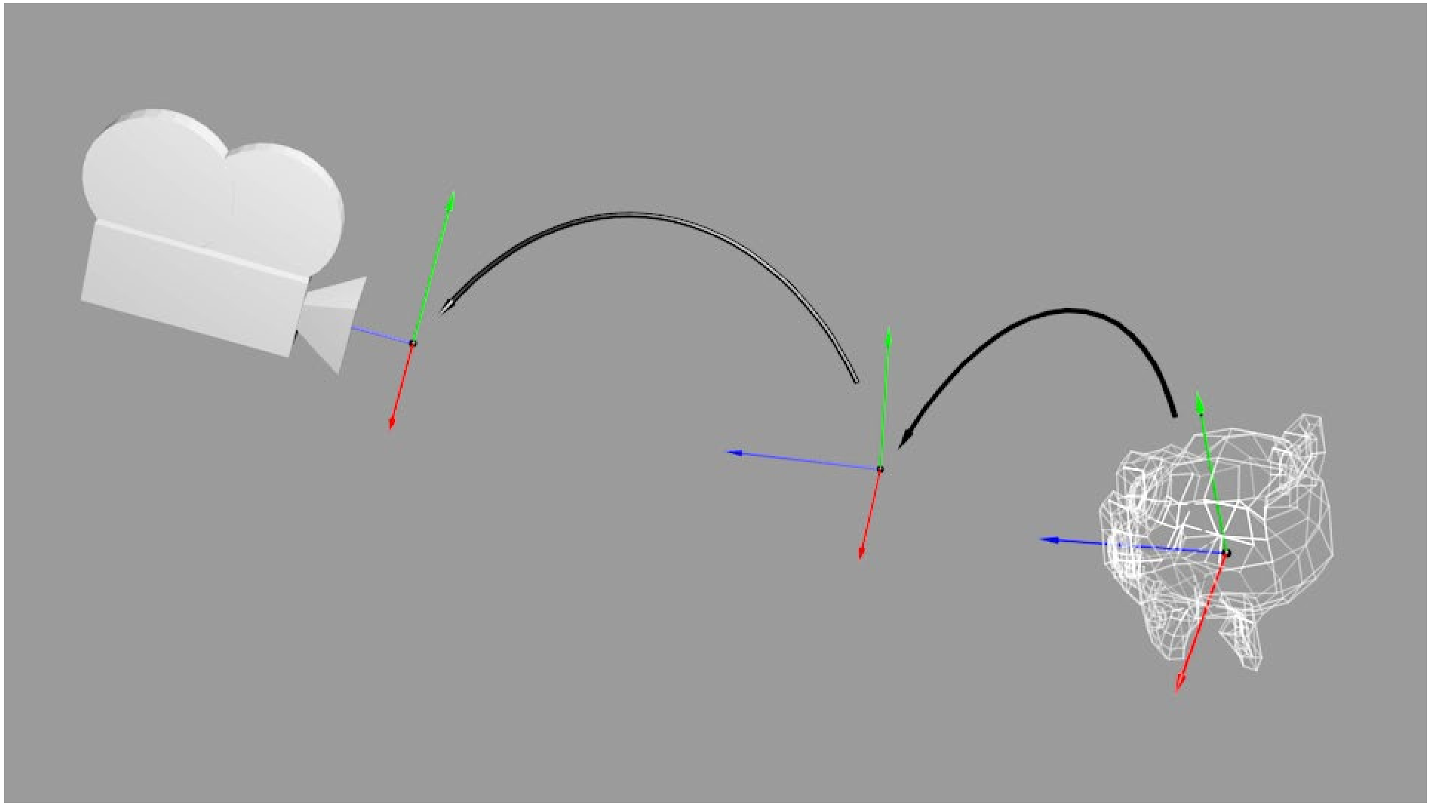
The World Coordinate System

The model is moved to a new position, and possibly included with other models, in the world coordinate system.



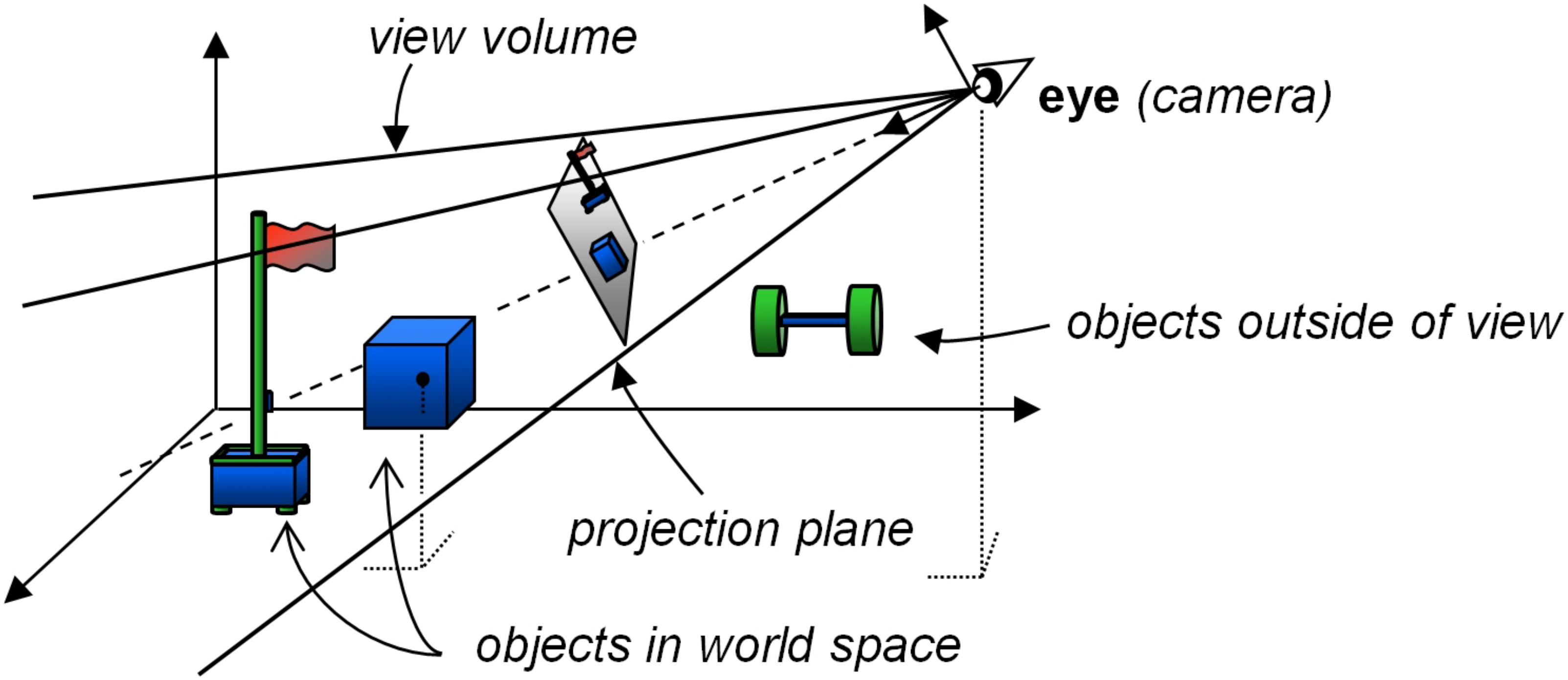
The View Coordinate System

The vertices expressed in the world coordinate system must be transformed into the view coordinate system since they are now relative to the camera.



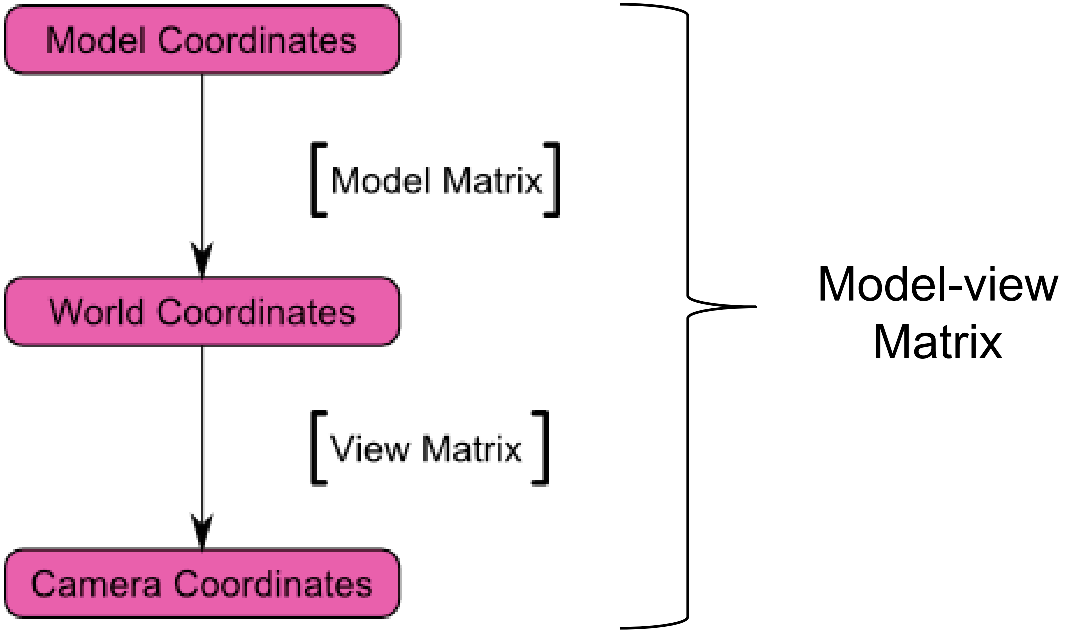
View Space

- World space as seen from a simulated camera or “eye”
- Also known as view, camera, or eye space.



Model-View Transformation

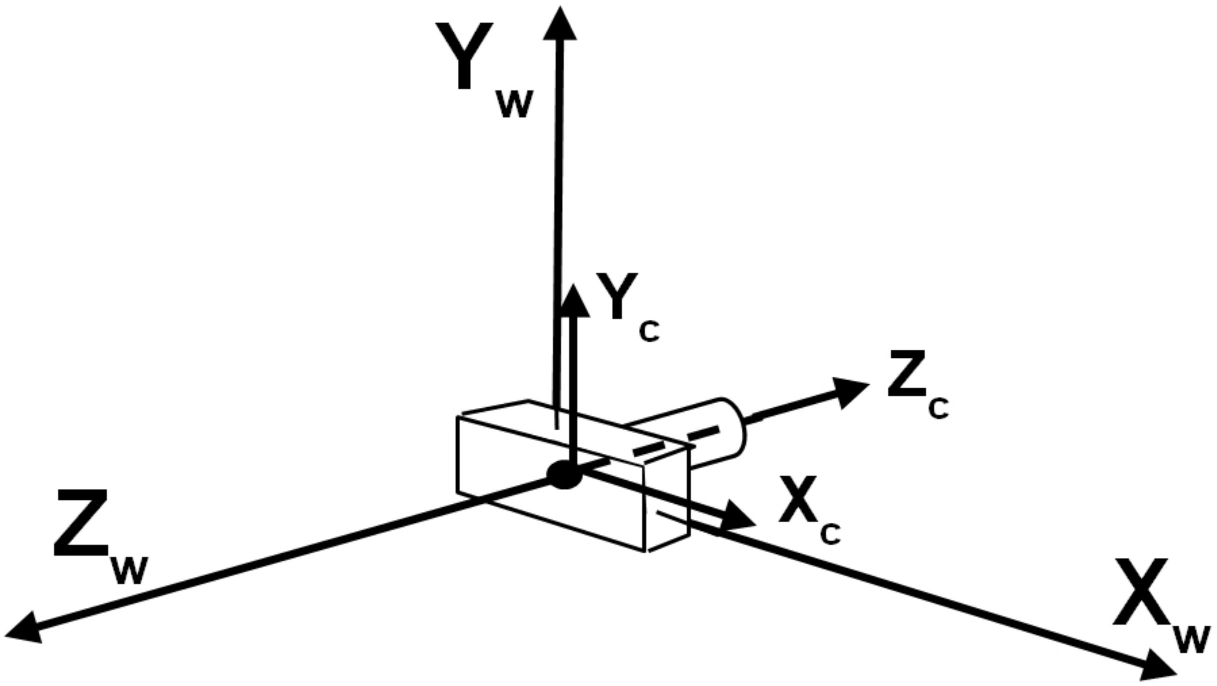
- A 4x4 matrix transforms vertices from the model to the world coordinate system.
- A second 4x4 matrix maps the world to the view coordinate system.
- The product of these two matrices is called the model-view matrix
- It maps the object from the original model coordinate system directly to the camera's (viewer's) coordinate system



The World and Camera Frames

- Changes in frame are defined by 4 x 4 matrices
- In OpenGL, we start with the world frame
- We move models from the world frame to the camera frame by using the model-view matrix $\mathbf{M}$
- Initially these frames are the same ($\mathbf{M}=\mathbf{I}$)
- If you want to move the camera three units to the right (+x), this is achieved by moving the objects three units to the left (-x).
- Camera always stays at the origin and points in the negative z direction

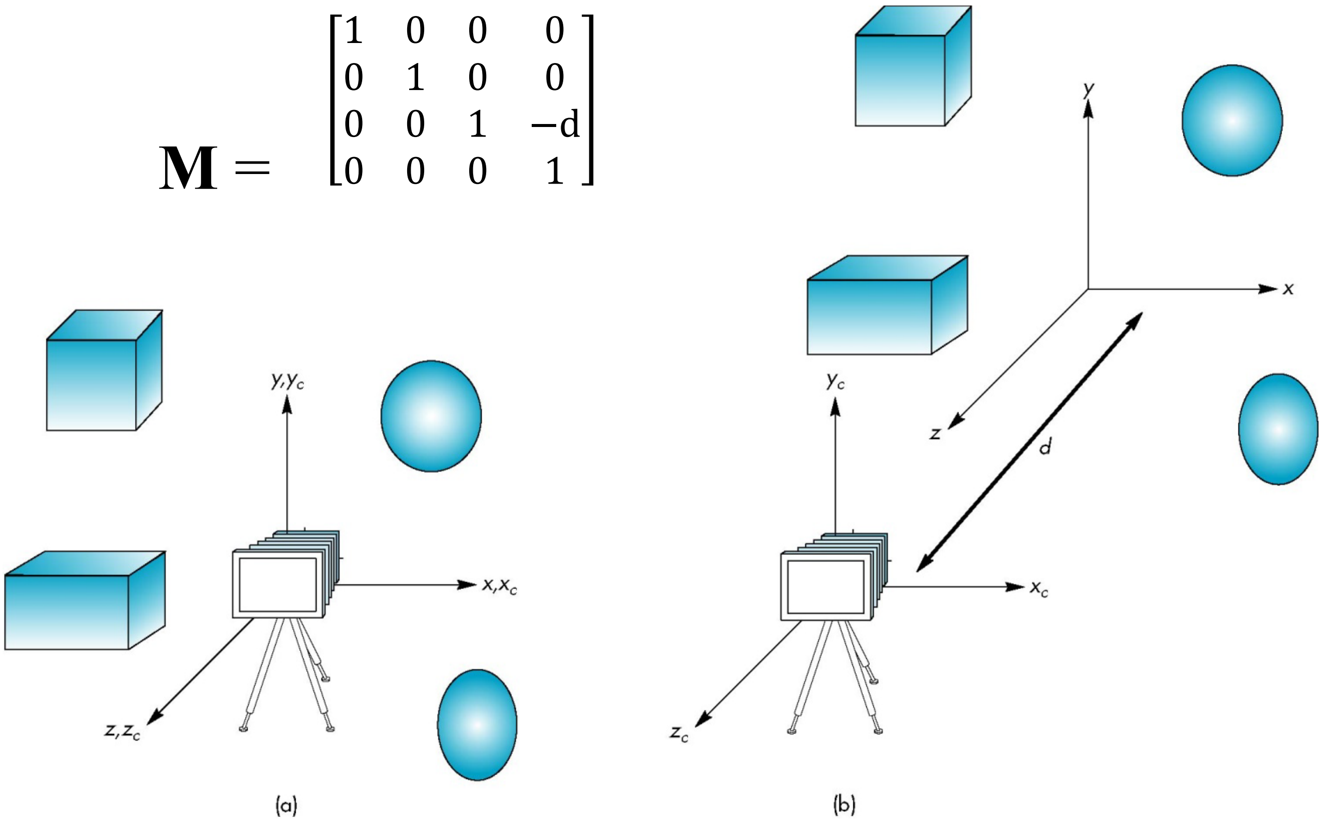
The OpenGL Fixed Camera



Moving the Objects

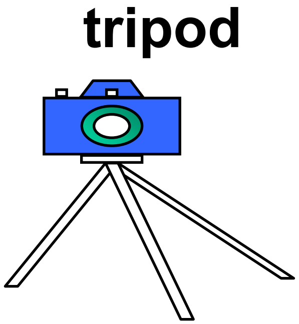
Move objects back (along –z direction) to view it in front of camera, which is at origin.

$$\mathbf{M} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -\mathrm{d} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



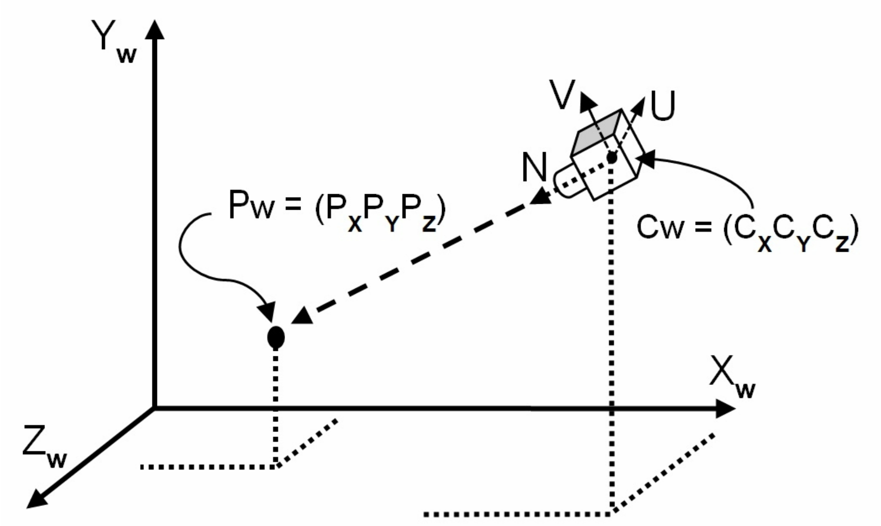
Viewing Transformations

- Position the camera/eye in the scene
  - place the tripod down; aim camera
- To “fly through” a scene
  - change viewing transformation and redraw scene



Building a View Matrix (V)

camera orientation:



$$\underset{\textit{point } P_C \textit{ in eye space}}{\begin{pmatrix}X_C\\Y_C\\Z_C\\1\end{pmatrix}} = \underbrace{\overset{\textit{negative of camera rotation angles}}{\underset{\textbf{R}\textit{ (rotation)}}{\begin{bmatrix}\hat{U}_X & \hat{U}_Y & \hat{U}_Z & 0\\ \hat{V}_X & \hat{V}_Y & \hat{V}_Z & 0\\ \hat{N}_X & \hat{N}_Y & \hat{N}_Z & 0\\ 0 & 0 & 0 & 1\end{bmatrix}}} * \overset{\textit{negative of camera location}}{\underset{\textbf{T}\textit{ (translation)}}{\begin{bmatrix}1 & 0 & 0 & -C_X\\ 0 & 1 & 0 & -C_Y\\ 0 & 0 & 1 & -C_Z\\ 0 & 0 & 0 & 1\end{bmatrix}}}}_{\textbf{V}\textit{ (viewing transform)}} * \underset{\textit{world point } P_W}{\begin{pmatrix}P_X\\P_Y\\P_Z\\1\end{pmatrix}}$$

The Model-View Matrix

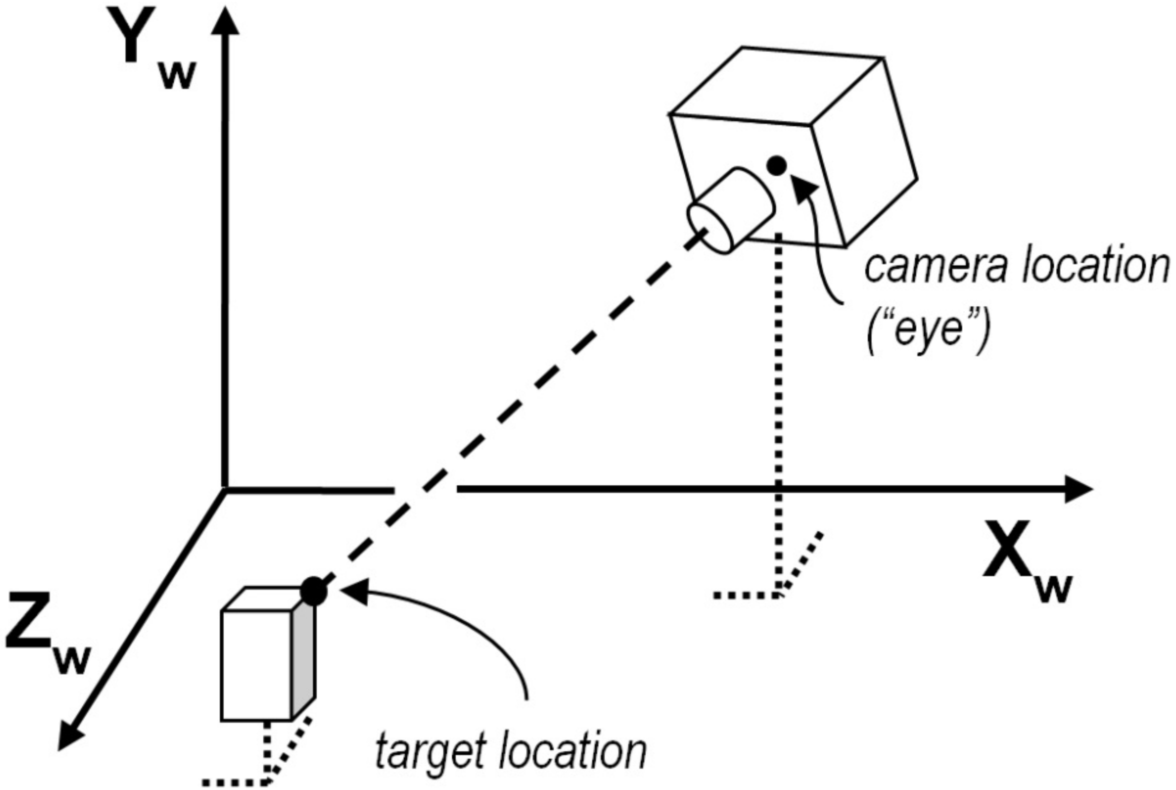
$$\mathbf{MV = V*M}$$

A point $\mathbf{P_M}$ in its own model space can then be transformed to camera space in one step, as follows:

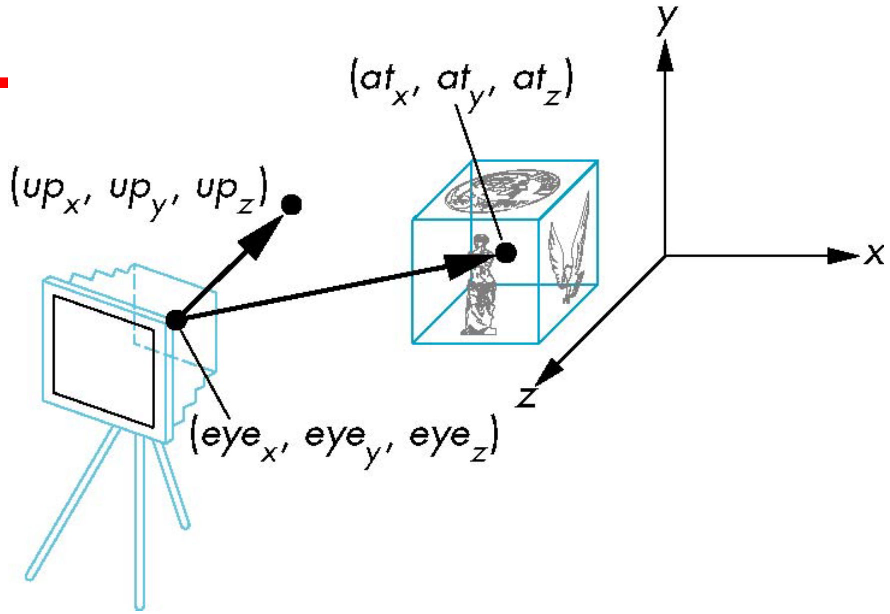
$$\mathbf{P_C = MV*P_M}$$

LookAt

- Simple viewing interface: `LookAt(eye, at, up)`
- up vector determines unique orientation



Creating the LookAt Matrix



$$\hat{n} = \frac{\overrightarrow{at} - \overrightarrow{eye}}{\left\| \overrightarrow{at} - \overrightarrow{eye} \right\|}$$

$$\hat{u} = \frac{\hat{n} \times \overrightarrow{up}}{\left\| \hat{n} \times \overrightarrow{up} \right\|} \Rightarrow \begin{pmatrix} u_x & u_y & u_z & -(eye \cdot \vec{u}) \\ v_x & v_y & v_z & -(eye \cdot \vec{v}) \\ -n_x & -n_y & -n_z & -(eye \cdot \vec{n}) \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\hat{v} = \hat{u} \times \hat{n}$$

Specifying What You Can See (1)

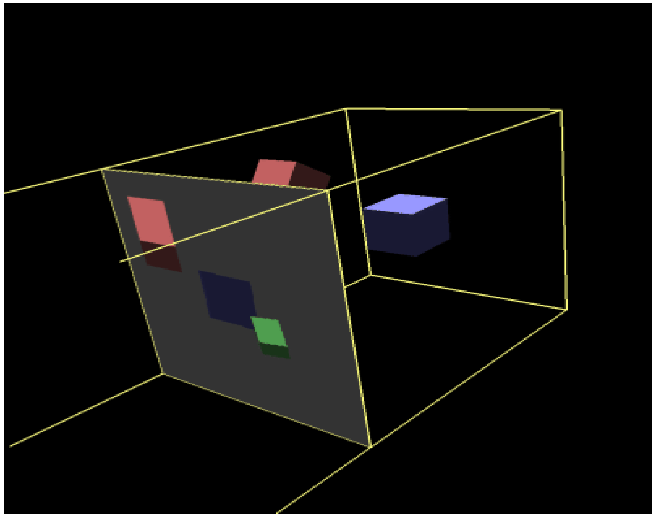
- Once camera is positioned in scene, we must set up a viewing frustum (view volume) to specify how much of the world we can see
- Done in two steps
  - specify the size of the frustum (projection transform)
  - specify its location in space (model-view transform)
- Anything outside of viewing frustum is clipped
  - primitive is either modified or discarded (if entirely outside frustum)

Specifying What You Can See (2)

- OpenGL projection model uses eye coordinates
  - the “eye” is located at the origin
  - looking down the -z axis
- Projection matrices use a six-plane model:
  - near (image) plane and far (infinite) plane
    - both are distances from the eye (positive values)
  - enclosing planes
    - top & bottom, left & right

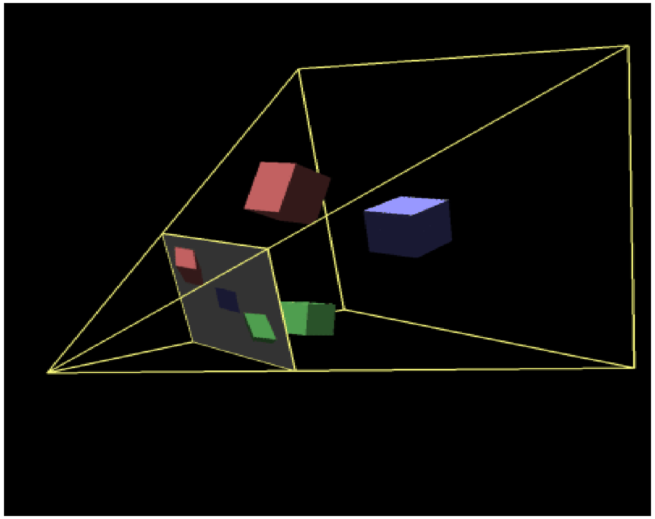
Specifying What You Can See (3)

Orthographic View



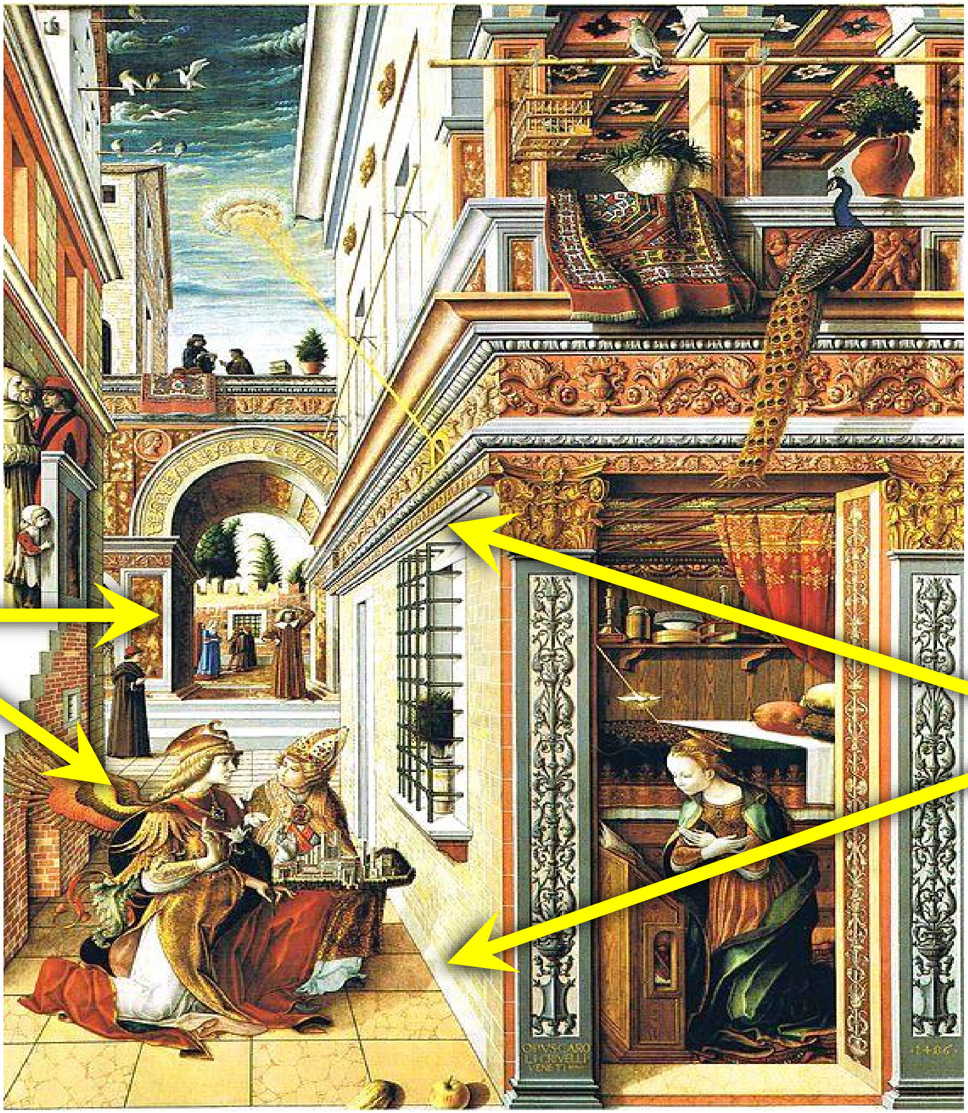
$$O = \begin{pmatrix} \frac{2}{r-l} & 0 & 0 & \frac{r+l}{r-l} \\ 0 & \frac{2}{t-b} & 0 & \frac{t+b}{t-b} \\ 0 & 0 & \frac{-2}{f-n} & \frac{f+n}{f-n} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Perspective View



$$P = \begin{pmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & \frac{-(f+n)}{f-n} & \frac{-2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

Perspective

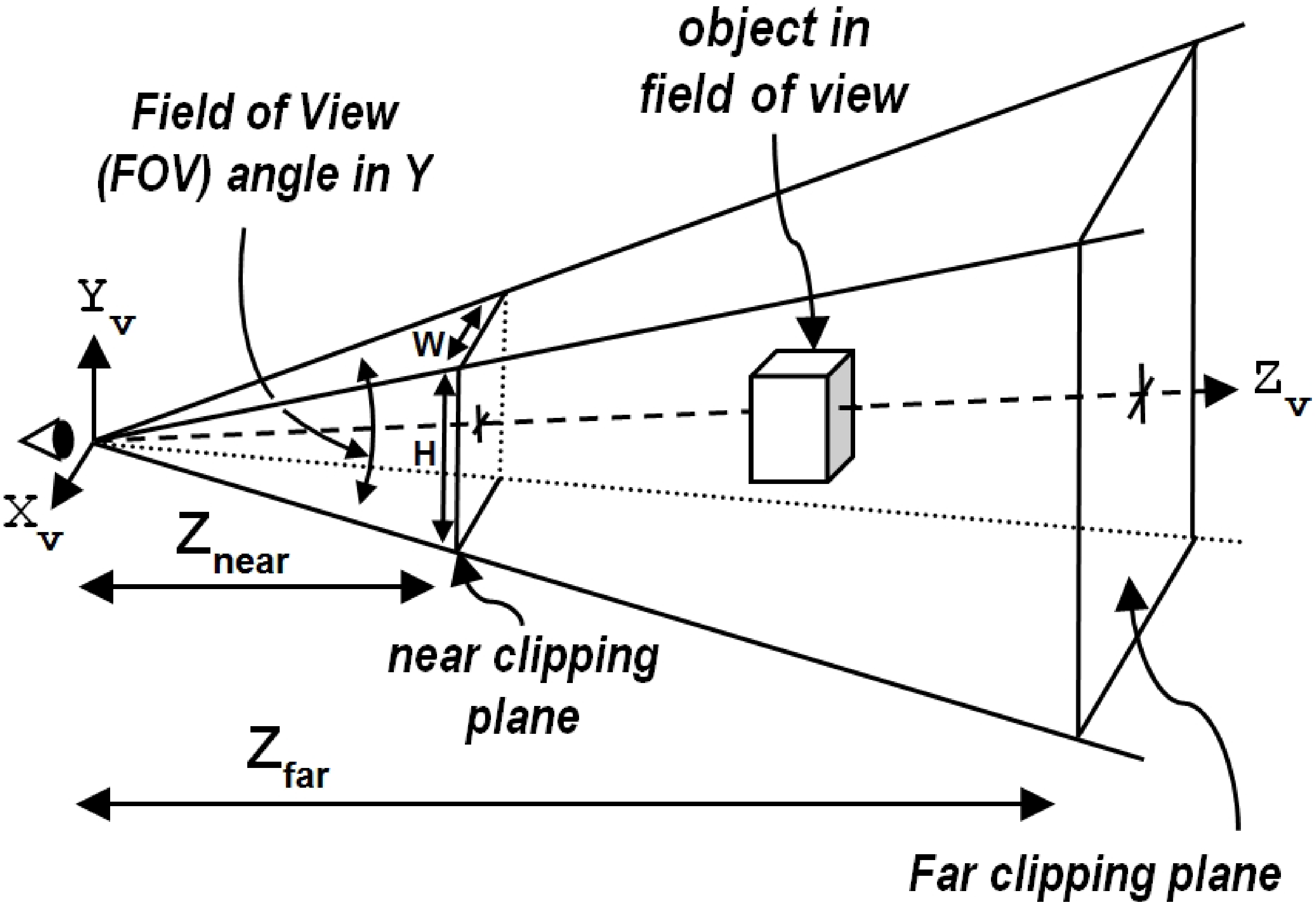


Scale by distance

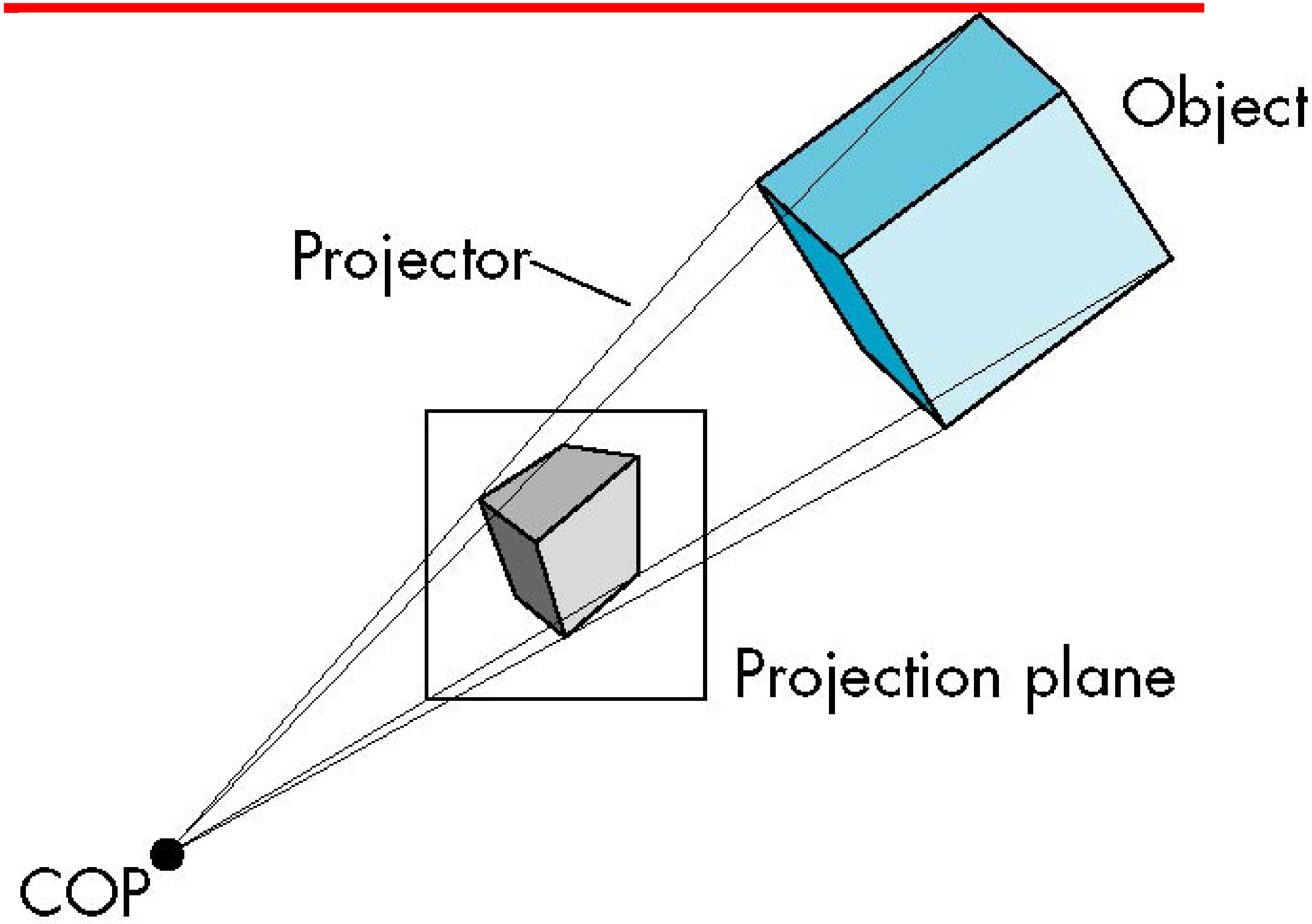
parallel lines

Annunciation with Saint Emidius (Crivelli - 1486)

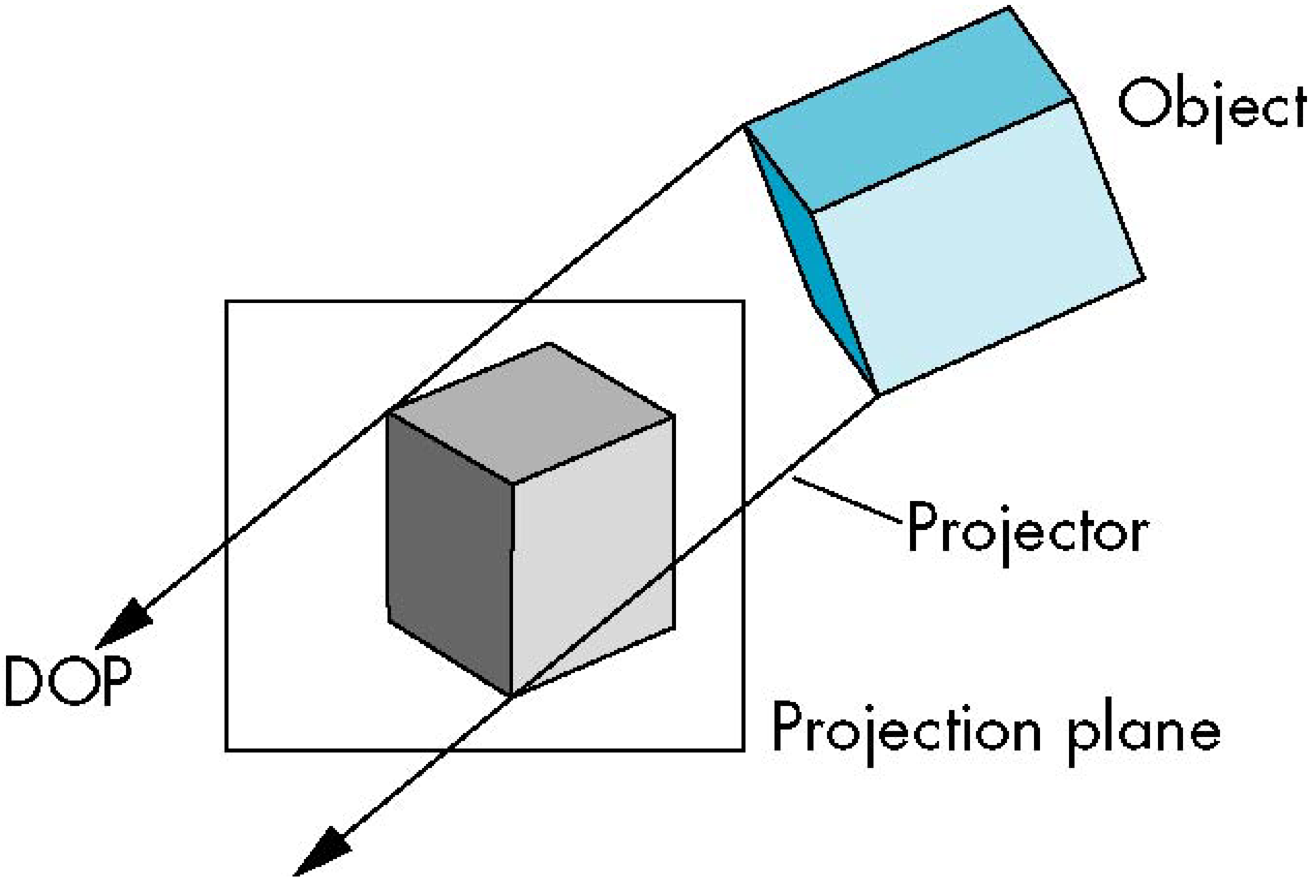
Perspective View Volume (Frustum)



Perspective Projection

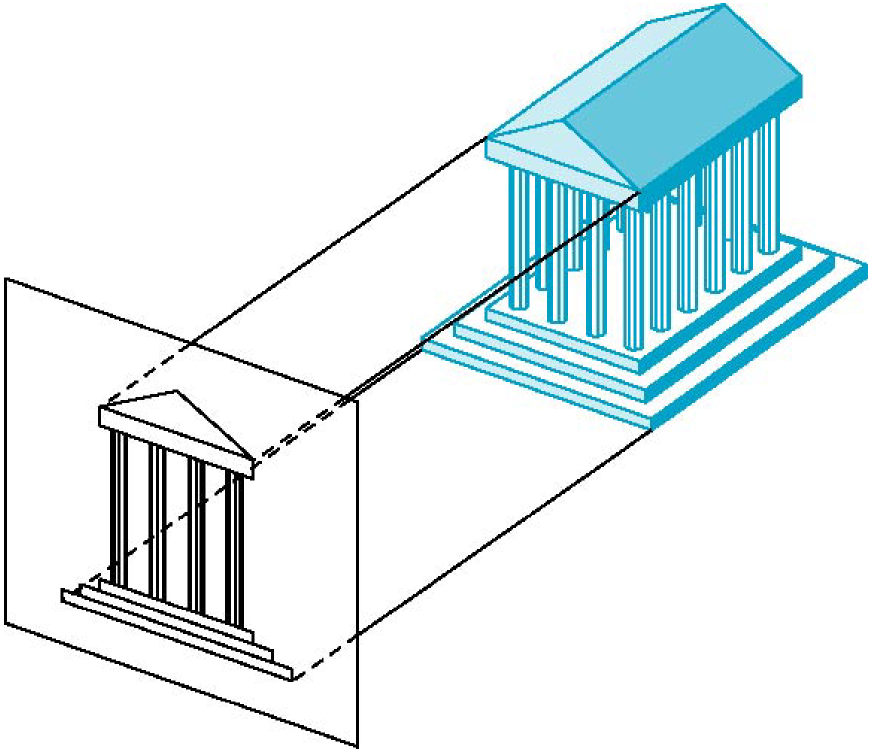


Parallel Projection



Orthographic Projection

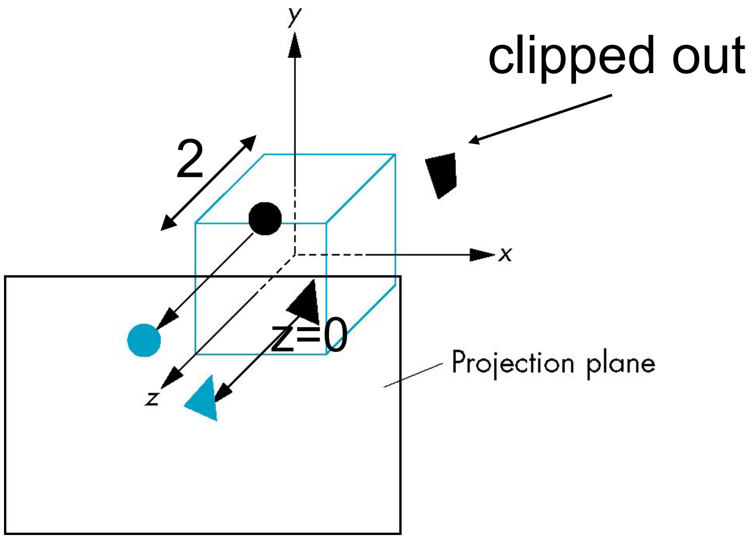
- Projectors are orthogonal to projection surface.
- Special (and most common) case of parallel projections



Default OpenGL Viewing

- Default view volume is a cube with sides of length 2 centered at the origin (from -1 to 1)
- Default projection is orthographic
- For points within the default view volume:

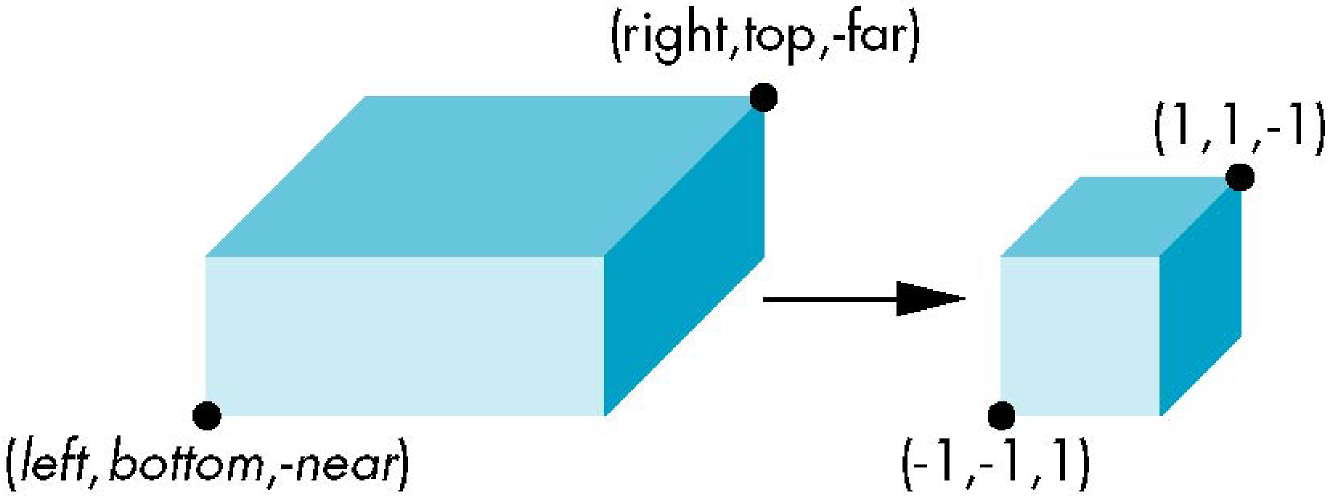
$$x_p = x$$
$$y_p = y$$
$$z_p = 0$$



Orthogonal Normalization

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

normalization ⇒ find transformation to convert specified clipping volume to default cube



near and **far** measured from camera

Orthogonal Matrix

- Two steps
  - Move center to origin

    T(-(left+right)/2, -(bottom+top)/2,(near+far)/2))
  - Scale to have sides of length 2

    S(2/(left-right),2/(top-bottom),2/(near-far))

$$\mathbf{P} = \mathbf{ST} = \begin{bmatrix} \frac{2}{right-left} & 0 & 0 & -\frac{right+left}{right-left} \\ 0 & \frac{2}{top-bottom} & 0 & -\frac{top+bottom}{top-bottom} \\ 0 & 0 & \frac{2}{near-far} & \frac{far+near}{far-near} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Final Projection

- Set $z = 0$
- Equivalent to the homogeneous coordinate transformation

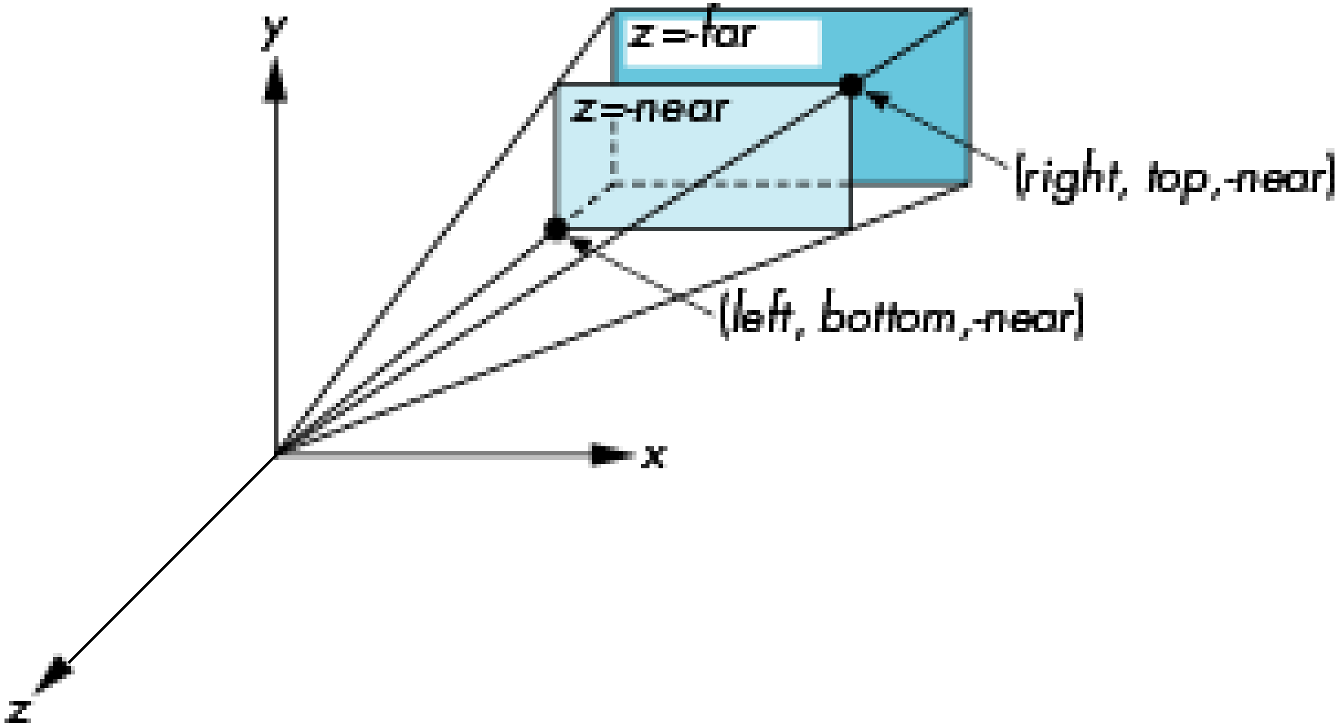
$$\mathbf{M}_{\text{orth}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Hence, general orthogonal projection in 4D is

$$\mathbf{P} = \mathbf{M}_{\text{orth}}\mathbf{ST}$$

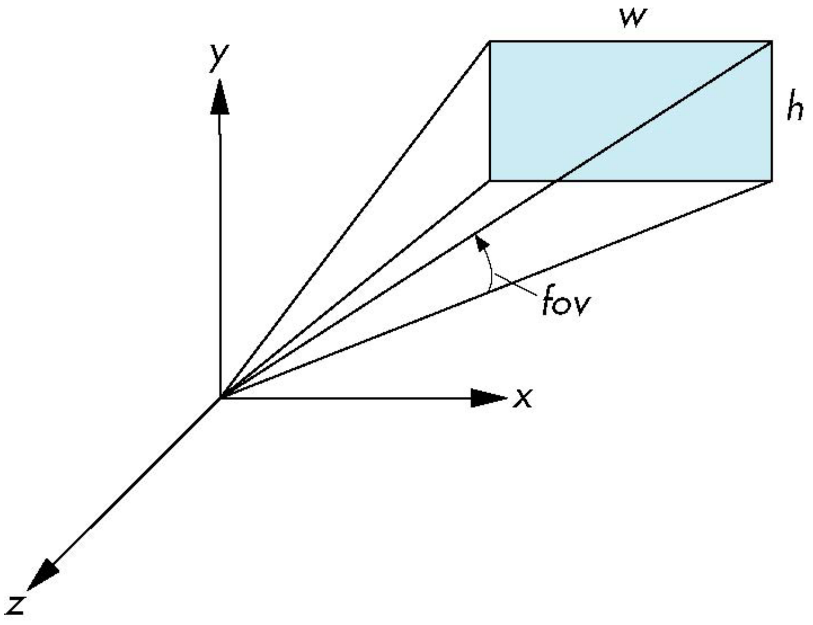
OpenGL Perspective

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



Using Field of View

- It is often difficult to get desired view with **`frustum()`**
- **`perspective(fovy, aspect, near, far)`** often provides a better interface

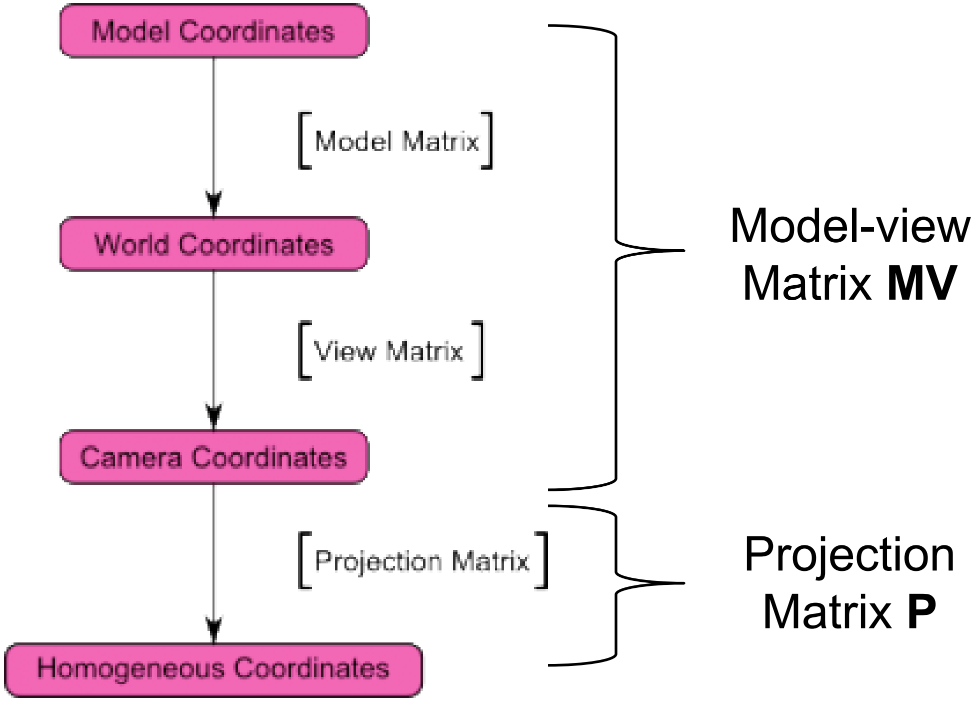


front plane

`aspect = w/h`

Projection Matrix

```
QMatrix4x4 Projection;
Projection.perspective(
                 45.0f,            // vertical field of view
                 4.0f/3.0f,        // aspect ratio
                 0.1f,             // near clipping plane
                 100.0f            // far clipping plane
)
```



Model-view and Projection Matrices

- In OpenGL the model-view matrix is used to
  - Position the camera
    - Easily done by using a LookAt function
  - Build models of objects
    - Positioning model elements together in world coordinates
- The projection matrix is used to define the view volume and to select a camera lens
- We create the model-view and projection matrices in our own applications and pass them to the vertex shader

Composite MVP Matrix

- We may sometimes build a single Model-View-Projection matrix (MVP):

$$\mathbf{MVP = P * V *M}$$

A point $\mathbf{P_M}$ in its own model space can then be transformed to its final perspective orientation in one step, as follows:

$$\mathbf{P_C = MVP * P_M}$$

Putting It All Together (1)

```
QMatrix4x4 Projection, View, Model, MVP;
Projection.perspective(
       45.0f,              // vertical field of view
       4.0f/3.0f,          // aspect ratio
       0.1f,               // near clipping plane
       100.0f              // far clipping plane
);

View.lookAt(
       vec3(4, 3, 3);      // camera in world space
       vec3(0, 0, 0);      // and looks at the origin
       vec3(0, 1, 0);      // up direction
);

Model.setToIdentity();     // model matrix is identity

MVP = Projection * View * Model;  // composite matrix
```

Putting It All Together (2)

```
// get a handle for our “u_MVP" uniform at initialization time
GLuint MatrixLoc = glGetUniformLocation(programID, “u_MVP");

// send our transformation to the currently bound shader
// in the “u_MVP" uniform
glUniformMatrix4fv(MatrixLoc, 1, GL_FALSE, &MVP[0][0]);
```

In vertex shader:

```
in  vec4 a_Position;      // vertex position
uniform mat4 u_MVP;       // Projection * Modelview
void main()
{
    gl_Position = u_MVP * a_Position;
}
```

Projection Matrices

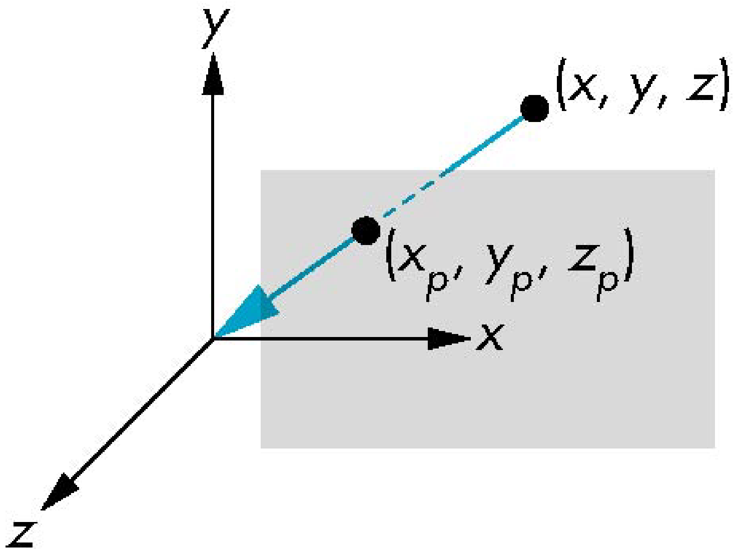
Prof. George Wolberg
Dept. of Computer Science
City College of New York

Objectives

- Derive the projection matrices used for standard OpenGL projections
- Introduce projection normalization

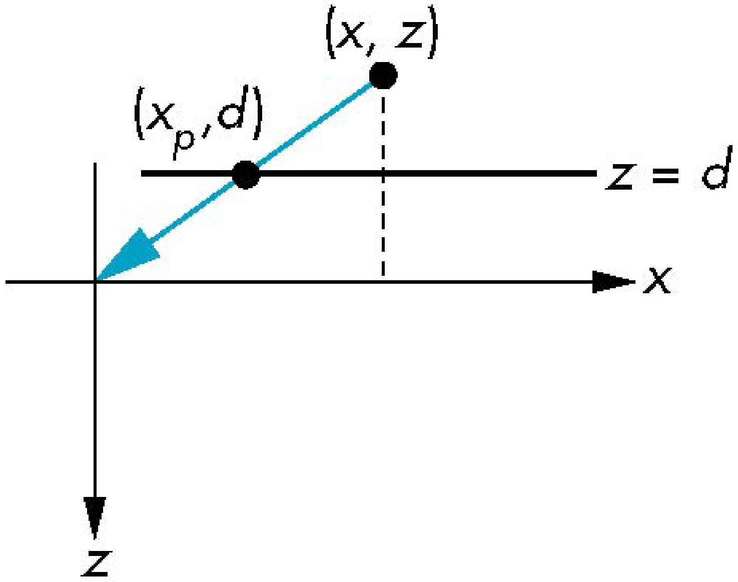
Simple Perspective

- Center of projection at the origin
- Projection plane $z = d, d < 0$

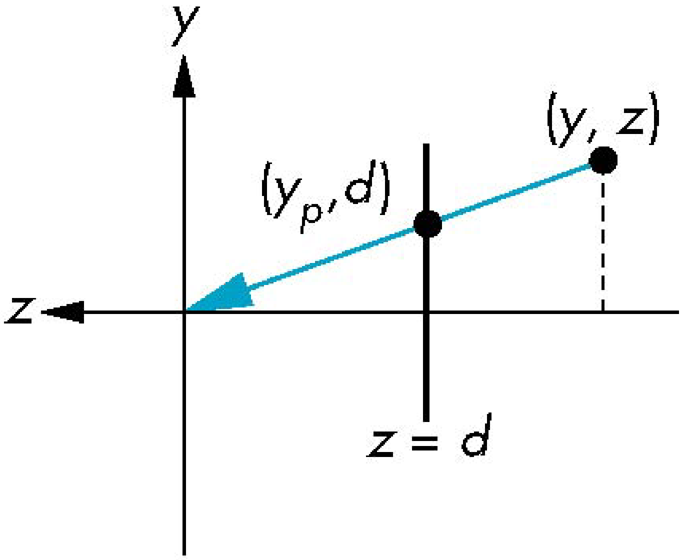


Perspective Equations

Consider top and side views



$\frac{x}{z/d}$



$x_p = \dfrac{x}{z/d}$ $\quad$ $y_p = \dfrac{y}{z/d}$ $\quad$ $z_p = d$

Homogeneous Coordinate Form

Consider **p** = **Mq** where:

$$\begin{bmatrix} x \\ y \\ z \\ z/d \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

p **M** **q**

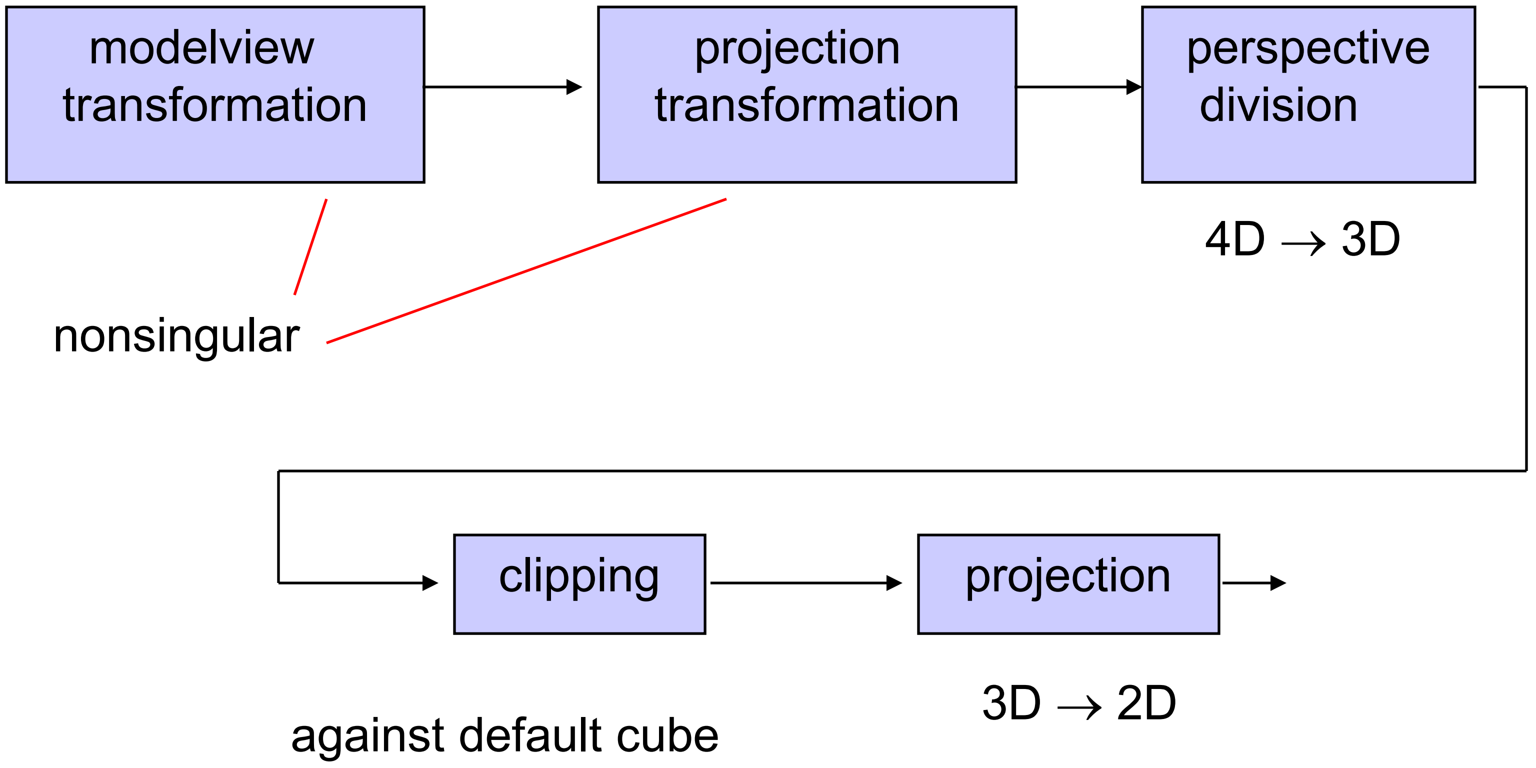
Perspective Division

- However $w \neq 1$, so we must divide by w to return from homogeneous coordinates
- This *perspective division* yields

$$x_p = \frac{x}{z/d} \qquad y_p = \frac{y}{z/d} \qquad z_p = d$$

the desired perspective equations

Pipeline View


modelview transformation

projection transformation

perspective division

4D → 3D

nonsingular

clipping

projection

3D → 2D

against default cube

Model-view and Projection Matrices

- In OpenGL the model-view matrix is used to
  - Position the camera
    - Easily done by using the LookAt function
  - Build models of objects
    - Positioning model elements together in world coordinates
- The projection matrix is used to define the view volume and to select a camera lens
  - `ortho(left,right,bottom,top,near,far)`
  - `perspective(fovy, aspect, near, far)`

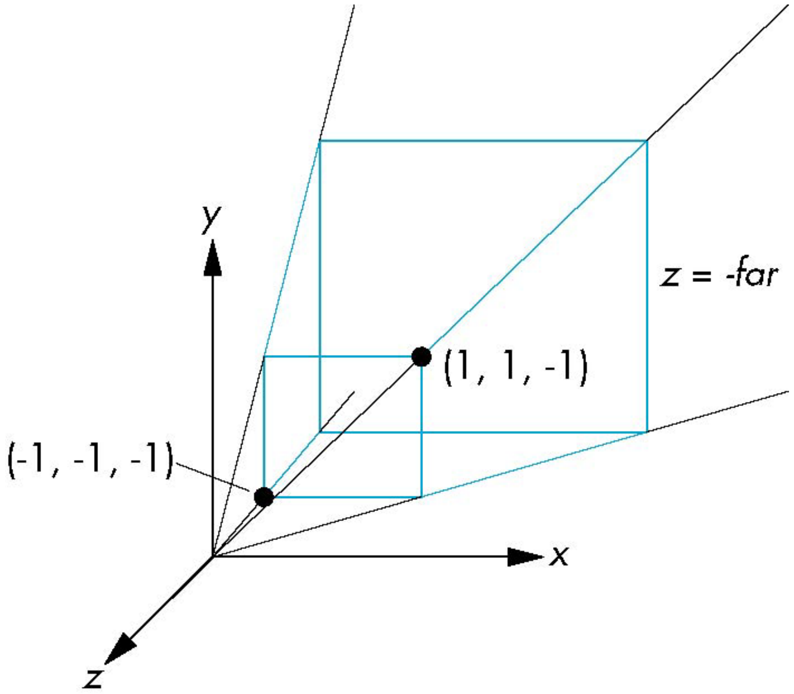
View Normalization

- Rather than derive a different projection matrix for orthographic and perspective projections, we can convert all projections to orthogonal projections with the default view volume
- This strategy allows us to use standard transformations in the pipeline and makes for efficient clipping

Simple Perspective

Consider a simple perspective with the COP at the origin, the near clipping plane at z = -1, and a 90 degree field of view determined by the planes

$x = \pm z, y = \pm z$



Perspective Matrices

Simple projection matrix in homogeneous coordinates

$$\mathbf{M} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

Note that -1 = 1/d where d = -1 and that

M is independent of the far clipping plane.

Generalization

$$\mathbf{N} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \alpha & \beta \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

After perspective division, the point (x, y, z, 1) goes to

$x'' = -x/z$
$y'' = -y/z$
$Z'' = -(\alpha + \beta/z)$

which projects orthogonally to the desired point regardless of α and β.

Picking α and β

If we pick

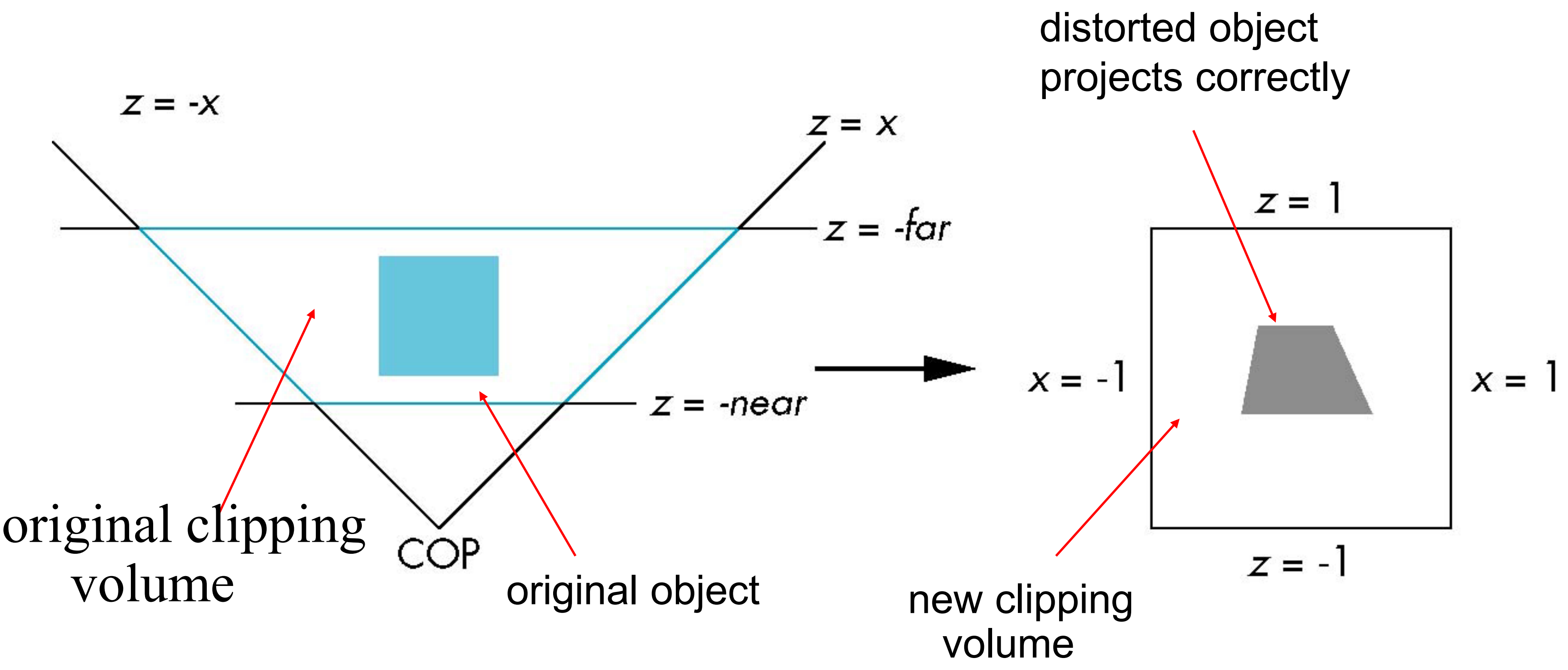
$$\alpha = \frac{\text{near} + \text{far}}{\text{far} - \text{near}}$$

$$\beta = \frac{2(\text{near} * \text{far})}{\text{near} - \text{far}}$$

the near plane is mapped to z = -1
the far plane is mapped to z =1
and the sides are mapped to x = ± 1, y = ± 1

Hence the new clipping volume is the default clipping volume

Normalization Transformation



distorted object projects correctly

original clipping volume

original object

new clipping volume

Normalization and Hidden-Surface Removal

- Although our selection of the form of the perspective matrices may appear somewhat arbitrary, it was chosen so that if $z_1 > z_2$ in the original clipping volume then the for the transformed points $z_1' > z_2'$
- Thus hidden surface removal works if we first apply the normalization transformation
- However, the formula $z'' = -(\alpha+\beta/z)$ implies that the distances are distorted by the normalization which can cause numerical problems especially if the near distance is small

General Case

$$\mathbf{N} = \begin{bmatrix} A & 0 & 0 & 0 \\ 0 & q & 0 & 0 \\ 0 & 0 & \alpha & \beta \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

where
q = 1 / tan(fovy/2)
A = q / aspectRatio = q * h/w

This takes into account the viewplane dimensions and the field of view in the y-direction.

OpenGL Perspective Matrix

- The normalization in `frustum()` requires an initial shear to form a right viewing pyramid, followed by a scaling to get the normalized perspective volume. Finally, the perspective matrix results in needing only a final orthogonal transformation

$$\mathbf{P} = \mathbf{NSH}$$

our previously defined perspective matrix

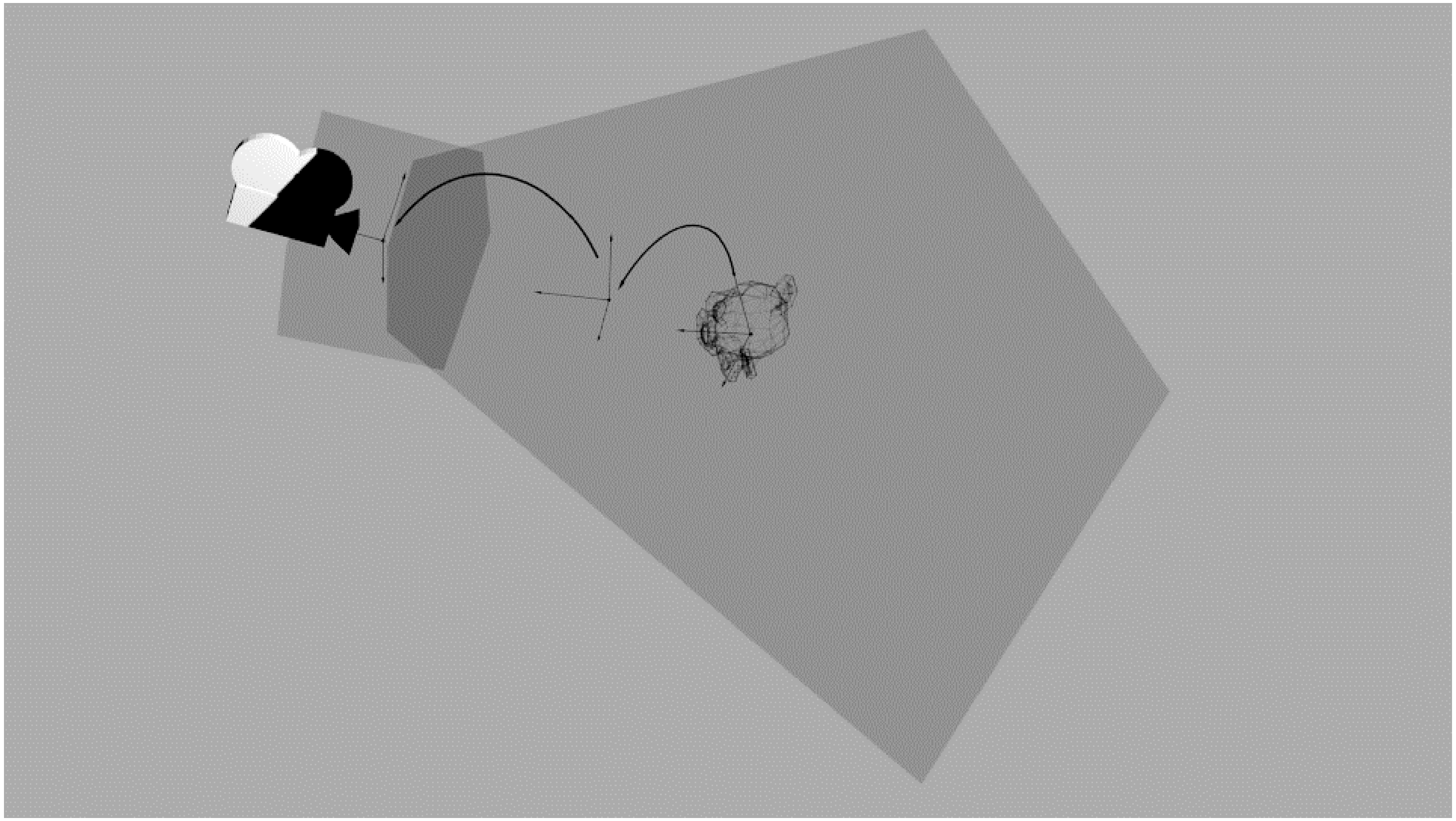
shear and scale

Why do we do it this way?

- Normalization allows for a single pipeline for both perspective and orthogonal viewing
- We stay in four dimensional homogeneous coordinates as long as possible to retain three-dimensional information needed for hidden-surface removal and shading
- We simplify clipping

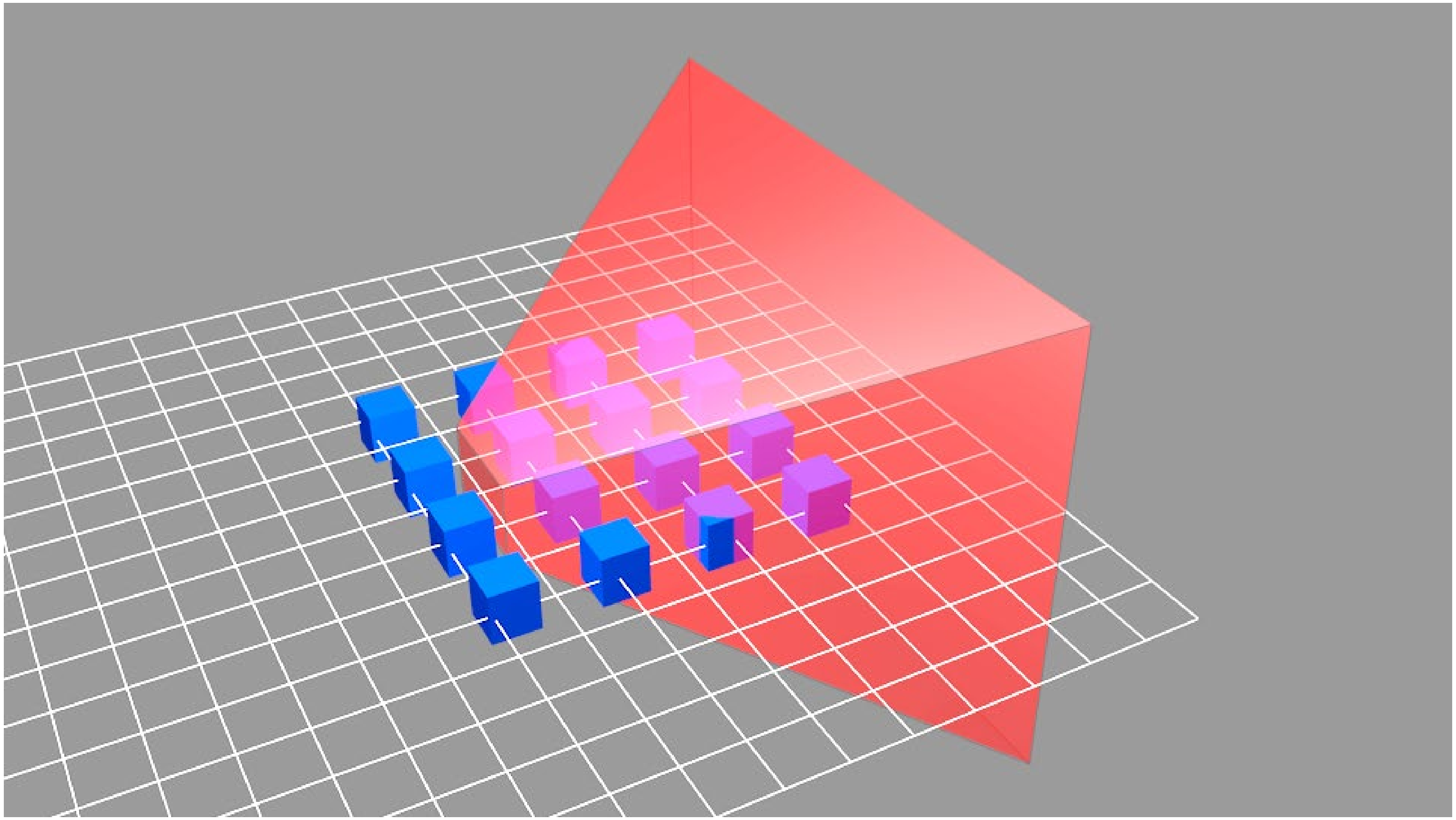
Perspective Projection

A perspective projection of the scene is generated as the rays that connect the vertices to the center of projection intersect the viewplane. The view volume consists of a frustum (truncated pyramid) extending from the camera.



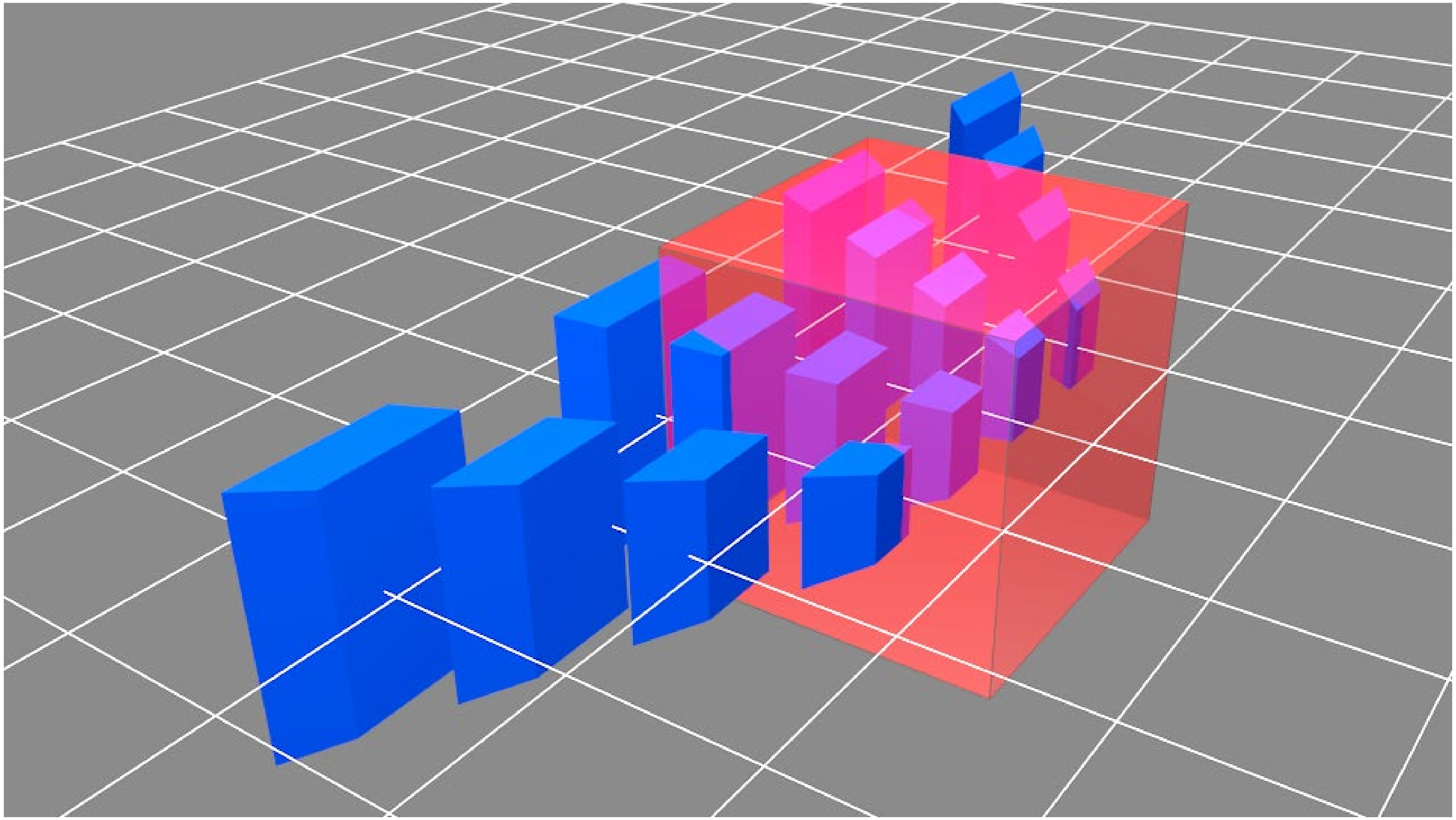
Before Projection

Before projection, we have the blue objects in camera space and the red camera frustum.

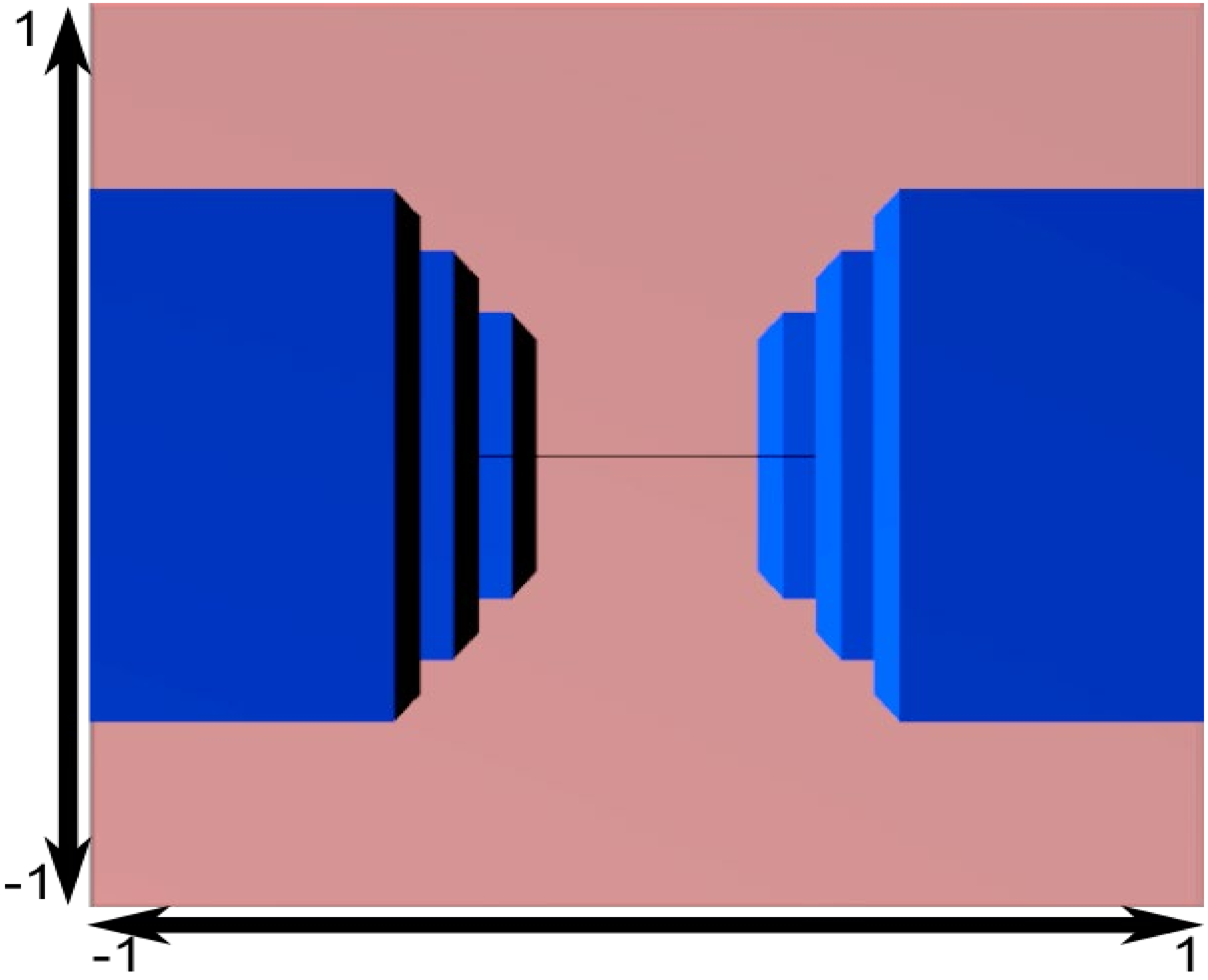


After Projection

Multiplying everything by the projection matrix has the following effect: the frustum is now a unit cube and the blue objects have been deformed.



View from Behind Frustum



Resized to Window