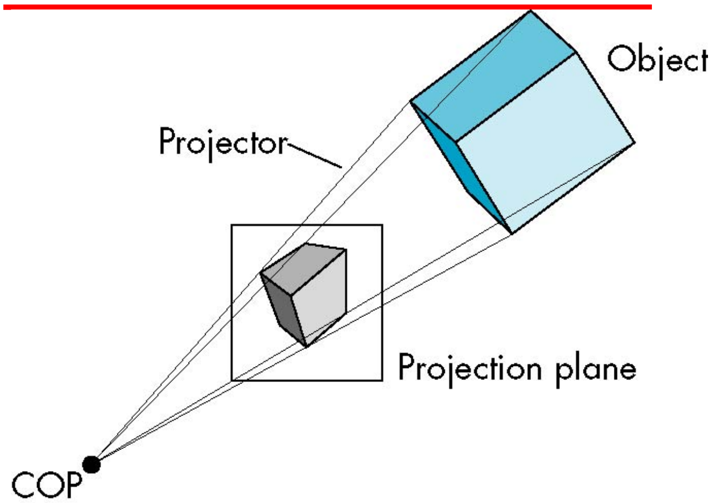
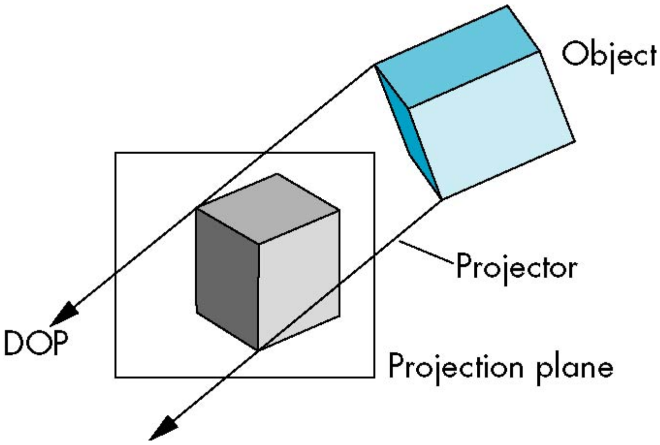
Computer Viewing

Ed Angel
Professor of Computer Science, Electrical and Computer Engineering, and Media Arts
University of New Mexico

Perspective Projection

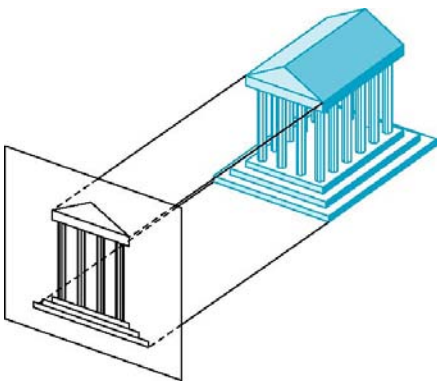


Parallel Projection



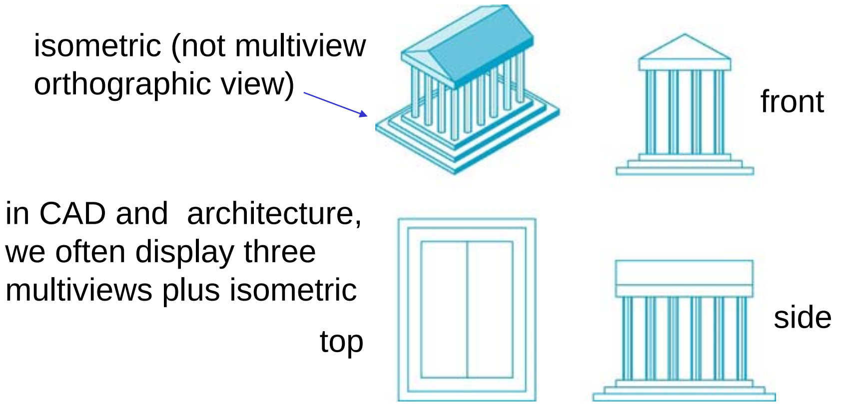
Orthographic Projection

Projectors are orthogonal to projection surface



Multiview Orthographic Projection

- Projection plane parallel to principal face
- Usually form front, top, side views

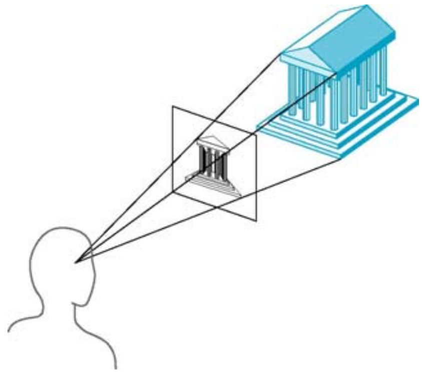


Advantages and Disadvantages

- Preserves both distances and angles
  - Shapes preserved
  - Can be used for measurements
    - Building plans
    - Manuals
- Cannot see what object really looks like because many surfaces hidden from view
  - Often we add the isometric

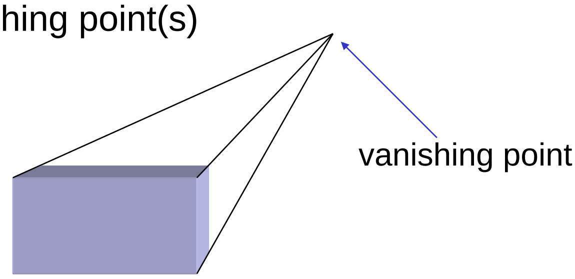
Perspective Projection

Projectors converge at center of projection



Vanishing Points

- Parallel lines (not parallel to the projection plan) on the object converge at a single point in the projection (the *vanishing point*)
- Drawing simple perspectives by hand uses these vanishing point(s)



One-Point Perspective

- One principal face parallel to projection plane
- One vanishing point for cube

Advantages and Disadvantages

- Objects further from viewer are projected smaller than the same sized objects closer to the viewer (*diminution*)
  - Looks realistic
- Equal distances along a line are not projected into equal distances (*nonuniform foreshortening*)
- Angles preserved only in planes parallel to the projection plane
- More difficult to construct by hand than parallel projections (but not more difficult by computer)

Computer Viewing

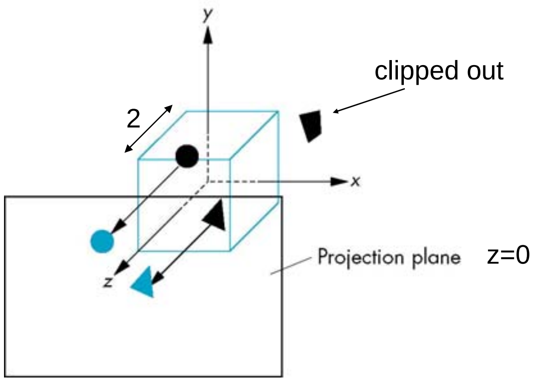
- There are three aspects of the viewing process, all of which are implemented in the pipeline,
  - Positioning the camera
    - Setting the model-view matrix
  - Selecting a lens
    - Setting the projection matrix
  - Clipping
    - Setting the view volume

The OpenGL Camera

- In OpenGL, initially the object and camera frames are the same
  - Default model-view matrix is an identity
- The camera is located at origin and points in the negative z direction
- OpenGL also specifies a default view volume that is a cube with sides of length 2 centered at the origin
  - Default projection matrix is an identity

Default Projection

Default projection is orthogonal



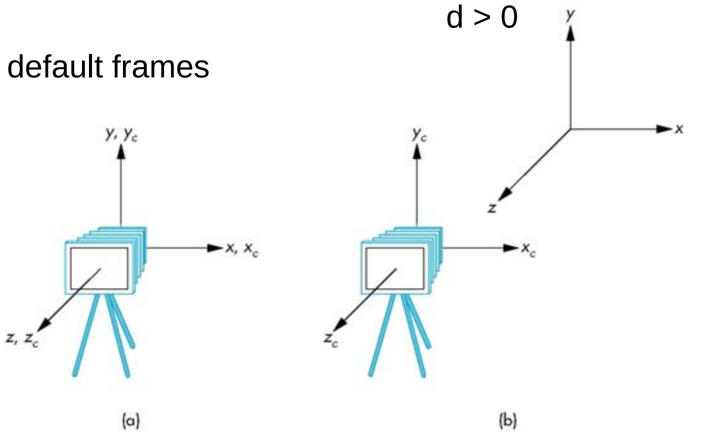
Moving the Camera Frame

- If we want to visualize object with both positive and negative z values we can either
  - Move the camera in the positive z direction
    - Translate the camera frame
  - Move the objects in the negative z direction
    - Translate the world frame
- Both of these views are equivalent and are determined by the model-view matrix
  - Want a translation (`glTranslatef(0.0,0.0,-d);`)
  - `-d > 0`

Moving Camera back from Origin

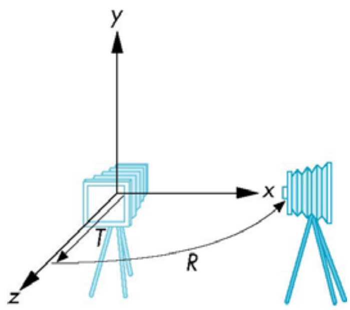
frames after translation by –d
d > 0

default frames



Moving the Camera

- We can move the camera to any desired position by a sequence of rotations and translations
- Example: side view
  - Rotate the camera
  - Move camera away from origin
  - Model-view matrix C = TR



OpenGL code

- Remember that last transformation specified is first to be applied

```
glMatrixMode(GL_MODELVIEW)
glLoadIdentity();
glTranslatef(0.0, 0.0, -d);
glRotatef(90.0, 0.0, 1.0, 0.0);
```


Projections and Normalization

- The default projection in the eye (camera) frame is orthogonal
- For points within the default view volume

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

- Most graphics systems use *view normalization*
  - All other views are converted to the default view by transformations that determine the projection matrix
  - Allows use of the same pipeline for all views

Homogeneous Coordinate Representation

default orthographic projection

$x_p = x$
$y_p = y$
$z_p = 0$
$w_p = 1$

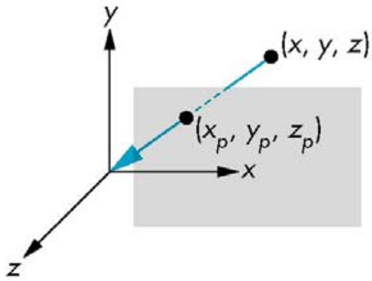
$$\mathbf{p}_p = \mathbf{M}\mathbf{p}$$

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

In practice, we can let $\mathbf{M} = \mathbf{I}$ and set the z term to zero later

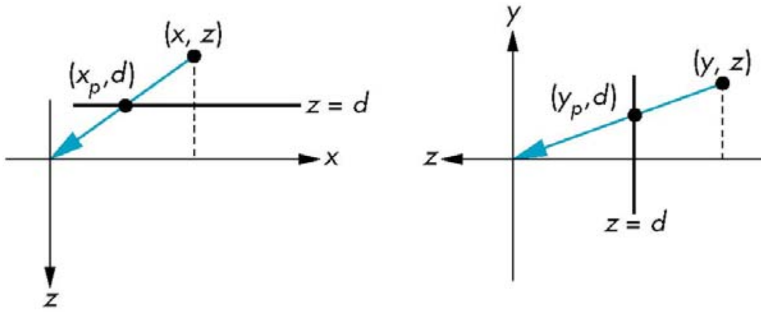
Simple Perspective

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



Perspective Equations

Consider top and side views



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

Homogeneous Coordinate Form

consider $\mathbf{q} = \mathbf{M}\mathbf{p}$ where $\mathbf{M} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{bmatrix}$

$$\mathbf{q} = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \Rightarrow \mathbf{p} = \begin{bmatrix} x \\ y \\ z \\ z/d \end{bmatrix}$$

Perspective Division

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

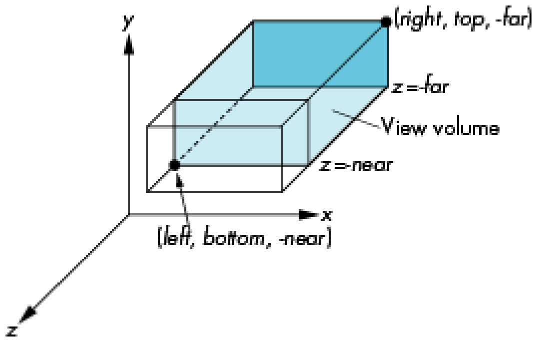
$$x_p = \frac{x}{z/d} \qquad y_p = \frac{y}{z/d} \qquad z_p = d$$

the desired perspective equations

- We will consider the corresponding clipping volume with the OpenGL functions

OpenGL Orthogonal Viewing

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

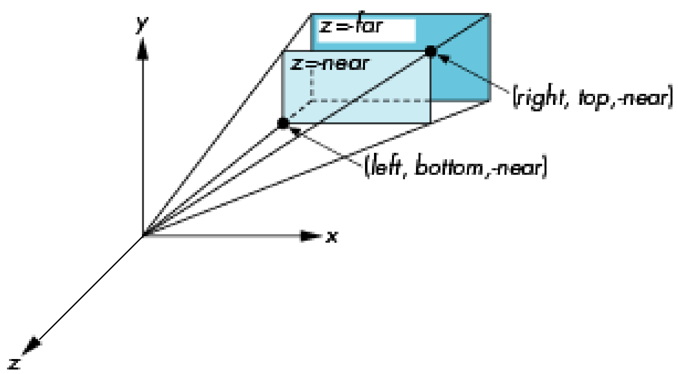


`near` and **`far`** measured <u>from</u> camera

The University of New Mexico

OpenGL Perspective

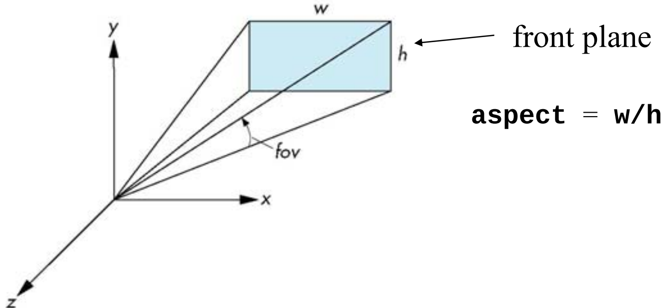
```
glFrustum(left,right,bottom,top,near,far)
```



The University of New Mexico

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



The University of New Mexico

Projection Matrices

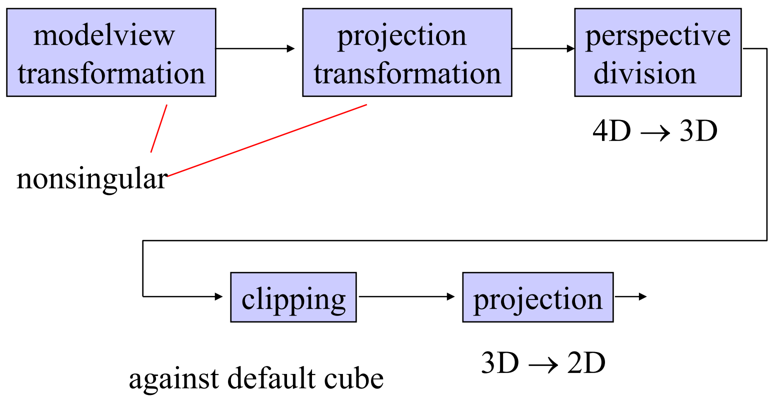
Ed Angel
Professor of Computer Science, Electrical and Computer Engineering, and Media Arts
University of New Mexico

The University of New Mexico

Normalization

- Rather than derive a different projection matrix for each type of projection, 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

Pipeline View



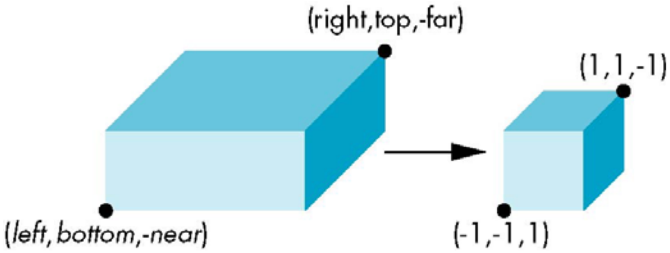
Notes

- We stay in four-dimensional homogeneous coordinates through both the modelview and projection transformations
  - Both these transformations are nonsingular
  - Default to identity matrices (orthogonal view)
- Normalization lets us clip against simple cube regardless of type of projection
- Delay final projection until end
  - Important for hidden-surface removal to retain depth information as long as possible

Orthogonal Normalization

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

normalization ⇒ find transformation to convert specified clipping volume to default



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

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

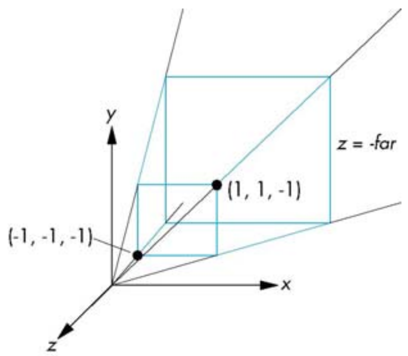
- Hence, general orthogonal projection in 4D is

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

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

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



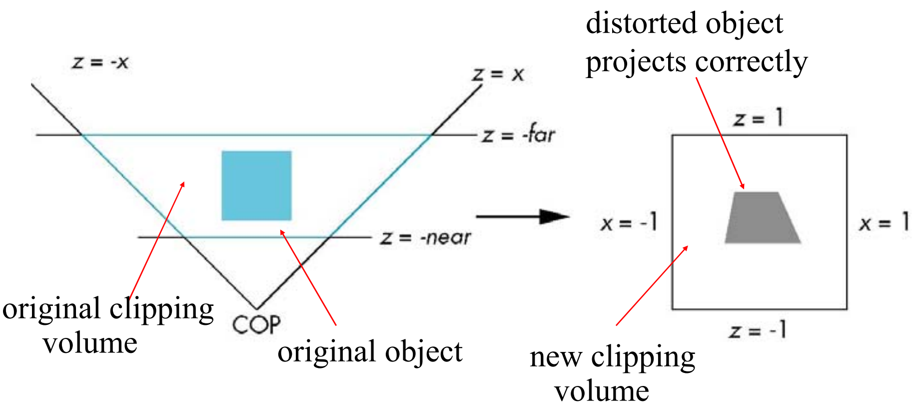
Perspective Matrices

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 this matrix is independent of the far clipping plane

Normalization Transformation



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 (both nonzero)

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 $x = \pm 1$, $y = \pm 1$

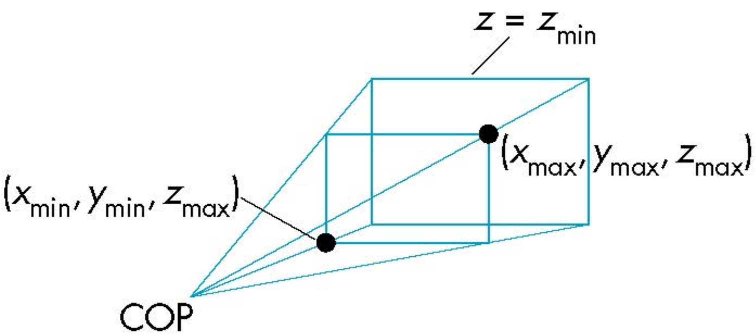
Hence the new clipping volume is the default clipping volume

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 same holds 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 (depth buffer usually 24 or 32 bits)

OpenGL Perspective

- `glFrustum` allows for an unsymmetric viewing frustum (although `gluPerspective` does not)



OpenGL Perspective Matrix

- The normalization in `glFrustum` 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