Viewing in 3D

Viewing in 3D

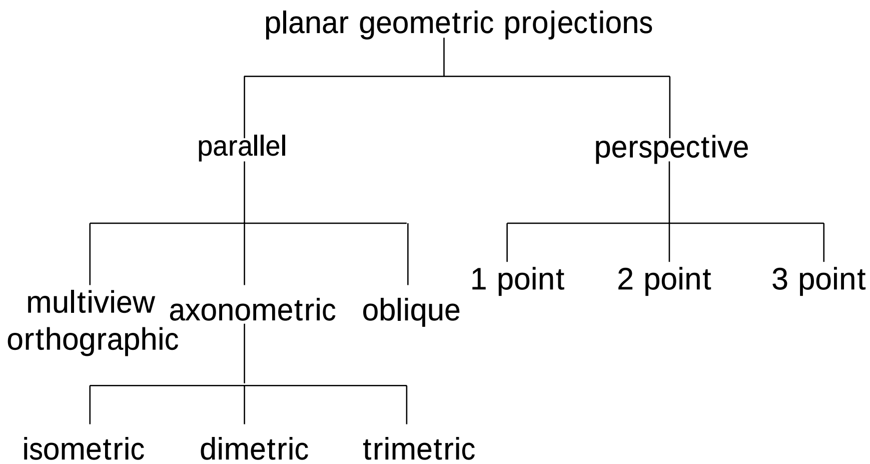
- How to specify which part of the 3D world is to be viewed?
  - 3D viewing volume
- How to transform 3D world coordinates to 2D display coordinate?
  - Projections
- Conceptual viewing pipeline:

Xform to eye coords ⇨ 3D clipping ⇨ Project ⇨ Xform to viewport

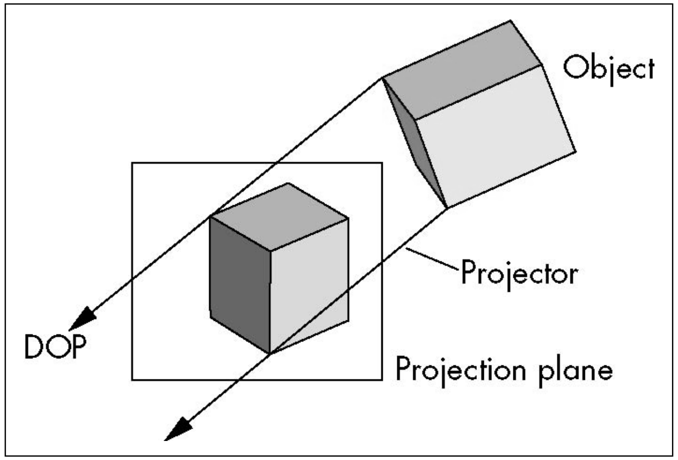
Planar Geometric Projections

- A projection is formed by the intersection of certain lines (*projectors*) with a plane (*the projection plane*)
- Projectors are lines from the *center of projection* through each point on object
- Center of projection at infinity results in a *parallel projection*
- A finite center of projection results in a *perspective projection*

Taxonomy of Projections

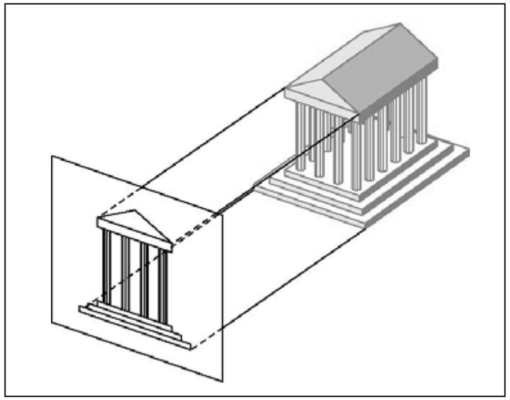


Parallel Projection



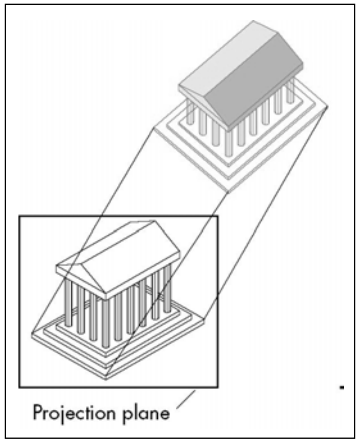
Orthographic Projection

- Projectors are orthogonal to projection surface, which is typically parallel to one of the coordinate planes:

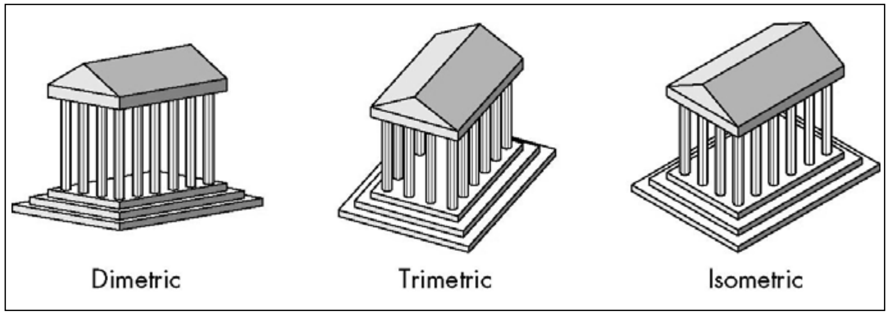


Axonometric Projections

- Allow projection plane to move relative to object.
- How many angles of a cube's corner are equal?
  - none: trimetric
  - two: dimetric
  - three: isometric

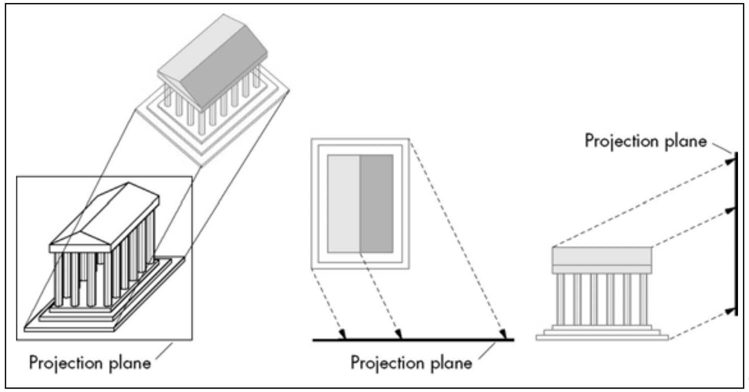


Axonometric Projections

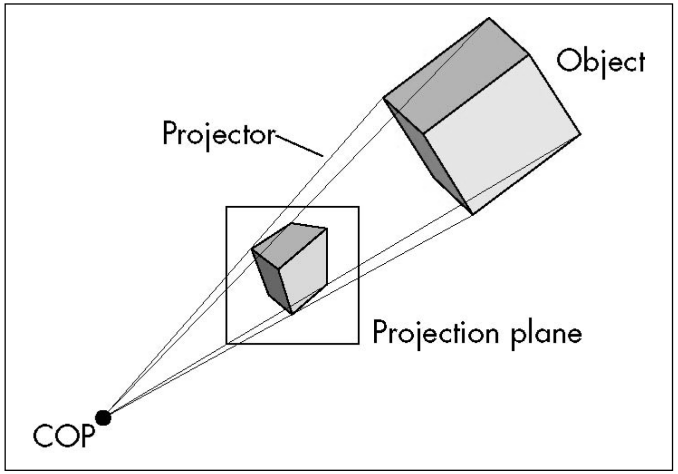


Oblique Projections

- Arbitrary relationship between projectors and projection plane.

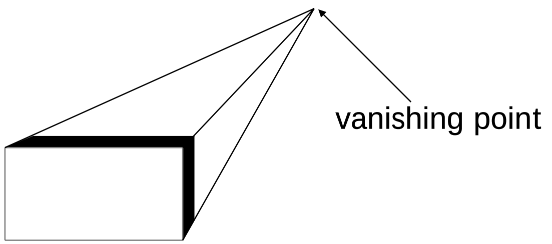


Perspective Projection

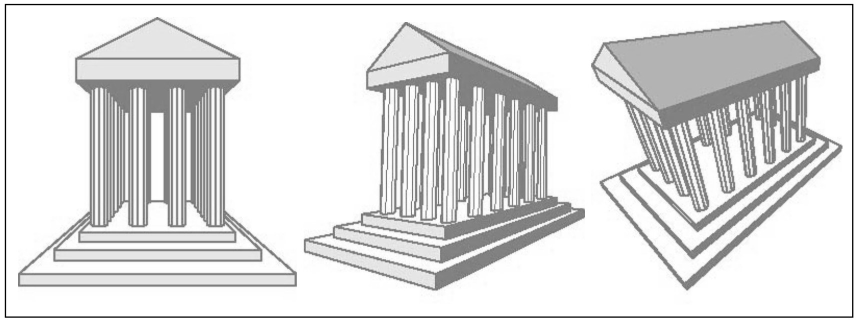


Vanishing Points

- Parallel lines (not parallel to the projection plane) on the object converge at a single point on the projection plane (the *vanishing point*):



N-point Perspective



one-point two-point three-point

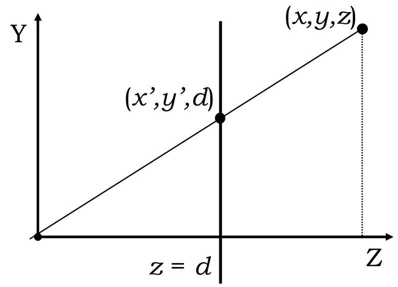
Orthographic Projections

- Direction of projection is normal to the projection plane.
- Typically, project onto one of the coordinate planes. For example:

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

- Typically, several of these projections (e.g., front, right, and top/plan views) are shown together

Perspective Projection



$$\frac{x'}{d} = \frac{x}{z} \quad \Rightarrow \quad x' = \frac{xd}{z}$$

$$\frac{y'}{d} = \frac{y}{z} \quad \Rightarrow \quad y' = \frac{yd}{z}$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \\ z/d \end{bmatrix} \quad \Rightarrow \quad \begin{bmatrix} xd/z = x' \\ yd/z = y' \\ zd/z = d \\ 1 \end{bmatrix}$$

Observations

- The rank of the matrix is 3 (= projection)
- Points on the projection plane are not changed by the perspective projection
- Let's see what happens to a point at infinity along the Z axis:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 1/d \end{bmatrix} \quad \Rightarrow \quad \begin{bmatrix} 0 \\ 0 \\ d \\ 1 \end{bmatrix}$$

- This is a vanishing point

Taking a Photograph

- Arrange objects
- Position and point the camera
- Choose a lens, set the zoom
- Take a picture
- Enlarge and crop to get a print

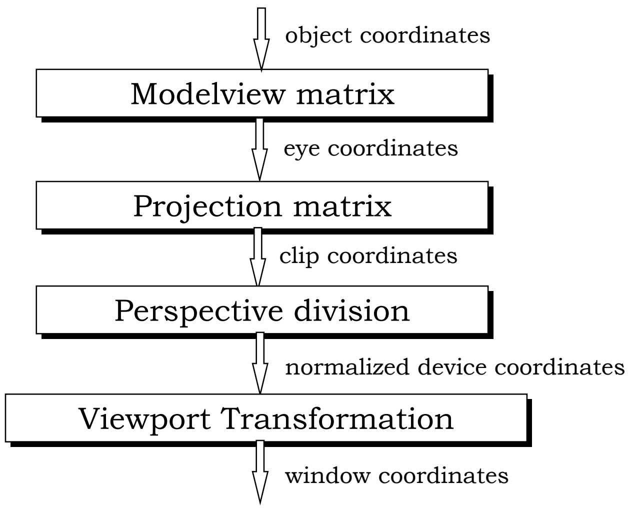
Taking a Virtual Photograph

- Arrange objects
  - Apply modeling transformations to objects: change from object coordinates to world coordinates
- Position and point the camera
  - Position, point, and orient the virtual camera: define a transformation from world to eye coordinates
- Choose a lens, set the zoom
  - Specify a view volume: define a perspective transformation that transforms eye coordinates to canonical normalized viewing space (clip coordinates)

Taking a Virtual Photograph

- Take a picture
  - Project objects by applying the perspective transformation followed by a perspective divide. The result is normalized device coordinates.
- Enlarge and crop to get a print
  - Apply viewport transformation to obtain actual window coordinates.

The OpenGL Viewing Pipeline



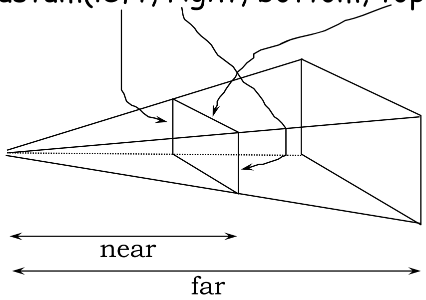
Modelview Matrix

- The initial OpenGL camera is at the origin, pointing down the negative *z*-axis.
- The modelview matrix is composited from simple 3D transformations:
  - glLoadIdentity
  - glTranslate, glRotate, glScale
  - glLoadMatrix, glMultMatrix
- Camera can also be positioned by the gluLookAt routine:
  gluLookAt(eye_x,eye_y,eye_z,ctr_x,ctr_y,ctr_z,up_x,up_y,up_z)

Projection Matrix

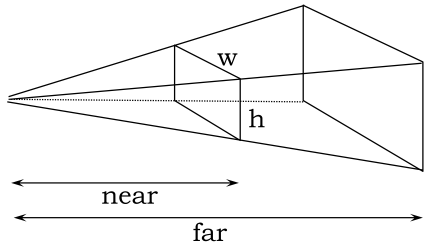
- Specified by defining a view volume (*view frustum*):

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



Projection Matrix

- Also can be specified by gluPerspective(fov, aspect, near, far)
  - aspect = w/h
  - fov = vertical field of view angle (degrees)



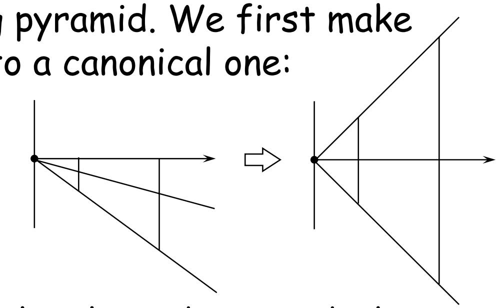
Projection Matrix

- glFrustum defines the following perspective transformation matrix:

$$\begin{bmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & -\frac{f+n}{f-n} & \frac{-2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

Derivation, part I

- glFrustum defines a general (possibly skewed) viewing pyramid. We first make this pyramid into a canonical one:



- We first shear the skewed pyramid, then scale.

Shearing Matrix

- Transforms the center of the viewing window (on the near plane) to (0,0,-n), making the view pyramid symmetric about the Z-axis:

$$\begin{bmatrix} 1 & 0 & \frac{r+l}{2n} & 0 \\ 0 & 1 & \frac{t+b}{2n} & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

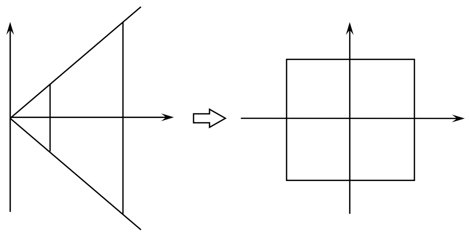
Scaling Matrix

- Scale the symmetric pyramid to create a 45 degree angle between each plane and the Z-axis:

$$\begin{bmatrix} \frac{2n}{r-l} & 0 & 0 & 0 \\ 0 & \frac{2n}{t-b} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Derivation, part II

- The canonical pyramid is then transformed into a cube, using a perspective transformation:



$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{-(f+n)}{f-n} & \frac{-2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

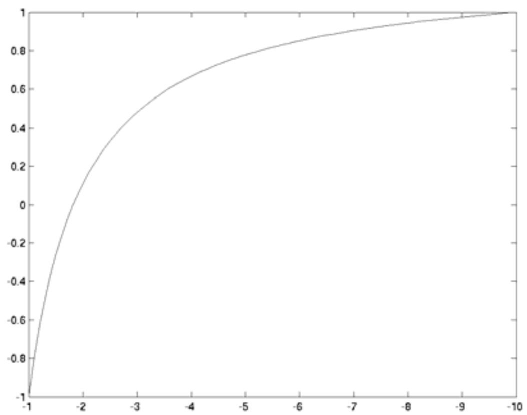
Finally...

- Multiplying the transformations gives us the desired matrix:

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

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

The effect on Z



View Frustum Clipping

- In homogeneous coordinates all points inside the view frustum satisfy all of the following inequalities:

$$w > 0 \quad and \quad \begin{matrix} x < w & x > -w \\ y < w & y > -w \\ z < w & z > -w \end{matrix}$$

- Lines must be clipped against the planes:

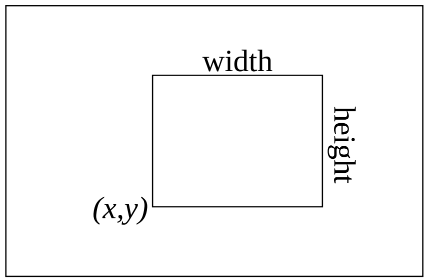
$$x = w \quad x = -w \quad y = w \quad y = -w \quad z = w \quad z = -w$$

Viewport Transformation

- Defines a pixel rectangle in the window into which the final image is mapped:

  glViewport(x, y, width, height)

- (x, y) specify the lower left corner of the viewport:



Viewport Transformation

- Transfroms normalized device (nd) coordinates to window (w) coordinates.
- nd coordinates range in [-1,1]
- w coordinates range in [x, x+width], [y,y+height]
- The resulting transformation is:

$$x_w = (x_{nd} + 1)\left(\frac{width}{2}\right) + x$$

$$y_w = (y_{nd} + 1)\left(\frac{height}{2}\right) + y$$