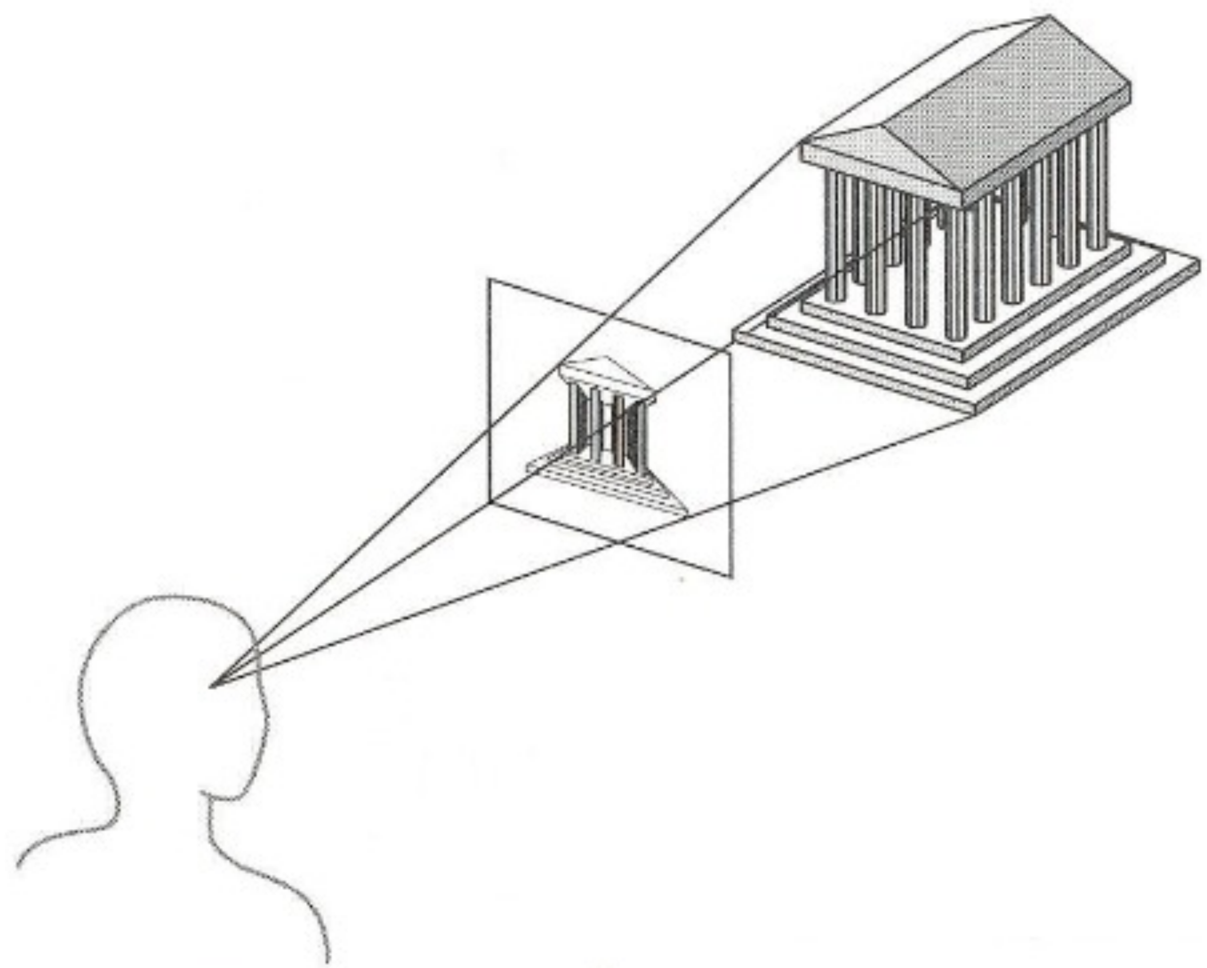
Geometric Transformations

Prof. Vladlen Koltun
Computer Science Department
Stanford University

Synthetic camera



Basic vertex processing

↓ vertices in model coordinates

Model-view transform

↓ camera coordinates

Projection transform

↓ clip coordinates

Conversion from homogeneous to Cartesian coordinates

↓ Normalized Device Coordinates

Window transform

↓ window coordinates

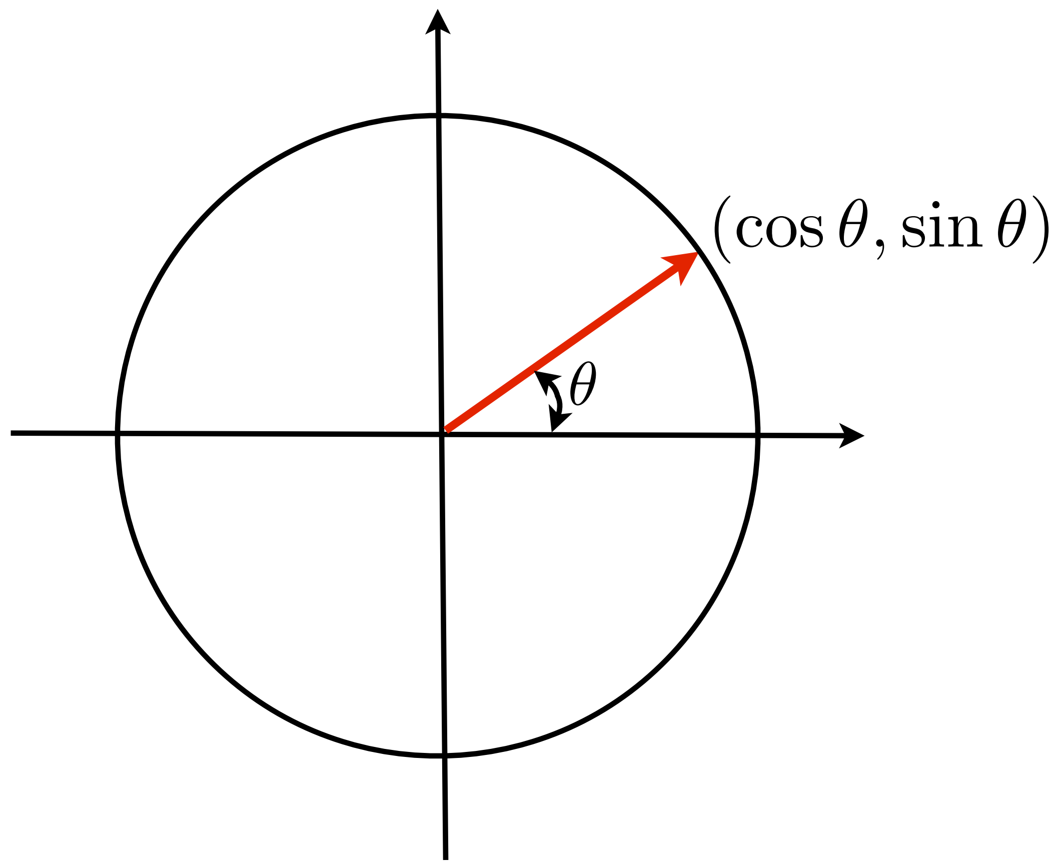
Geometric transformations as matrices

$$T_{t_x,t_y,t_z}(\mathbf{v}) = \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} v_x \\ v_y \\ v_z \\ 1 \end{pmatrix}$$

$$\mathbf{v} = \begin{pmatrix} v_x \\ v_y \\ v_z \\ 1 \end{pmatrix}$$

$$S_{s_x,s_y,s_z}(\mathbf{v}) = \begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} v_x \\ v_y \\ v_z \\ 1 \end{pmatrix}$$

Orientation in the plane



$$R_\theta \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

Orientation in 3D

$$R_\alpha^x = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) & 0 \\ 0 & \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$R_\beta^y = \begin{pmatrix} \cos(\beta) & 0 & \sin(\beta) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\beta) & 0 & \cos(\beta) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$R_\gamma^z = \begin{pmatrix} \cos(\gamma) & -\sin(\gamma) & 0 & 0 \\ \sin(\gamma) & \cos(\gamma) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

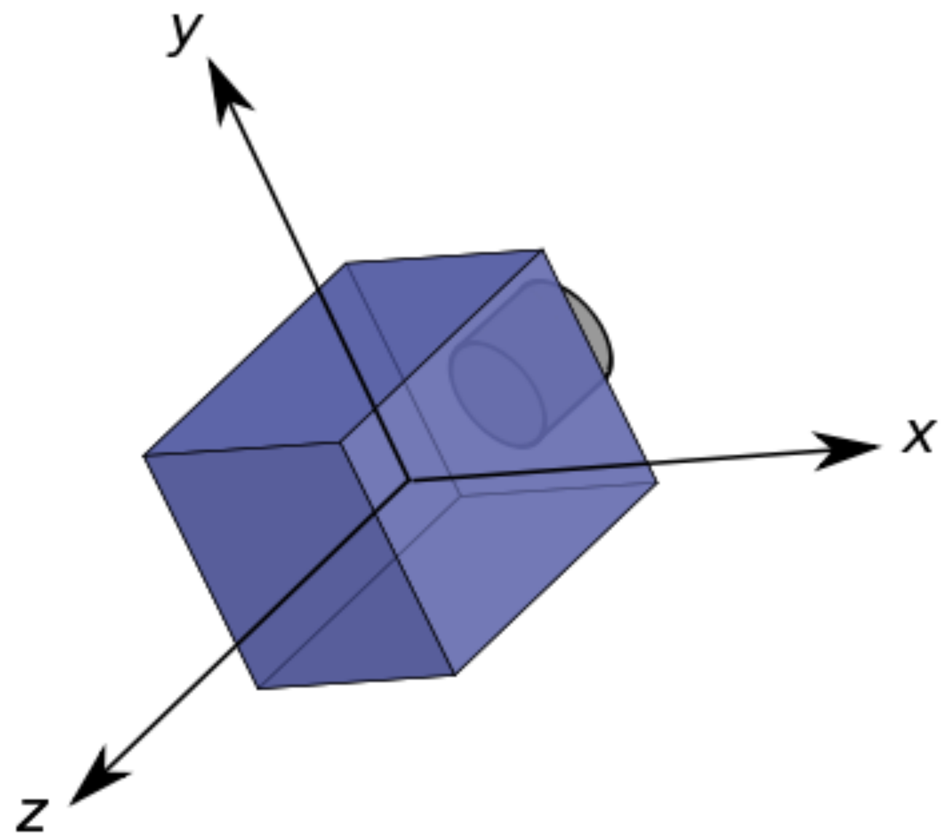
Concatenating transformations

$$\begin{aligned}
\mathbf{v_1} &= M_1 \mathbf{v_0} \\
\mathbf{v_2} &= M_2 \mathbf{v_1} \\
\mathbf{v_3} &= M_3 \mathbf{v_2}
\end{aligned}$$

$$\begin{aligned}
M_{\mathrm{c}} &= M_3 M_2 M_1 \\
\mathbf{v_3} &= M_{\mathrm{c}} \mathbf{v_0}
\end{aligned}$$

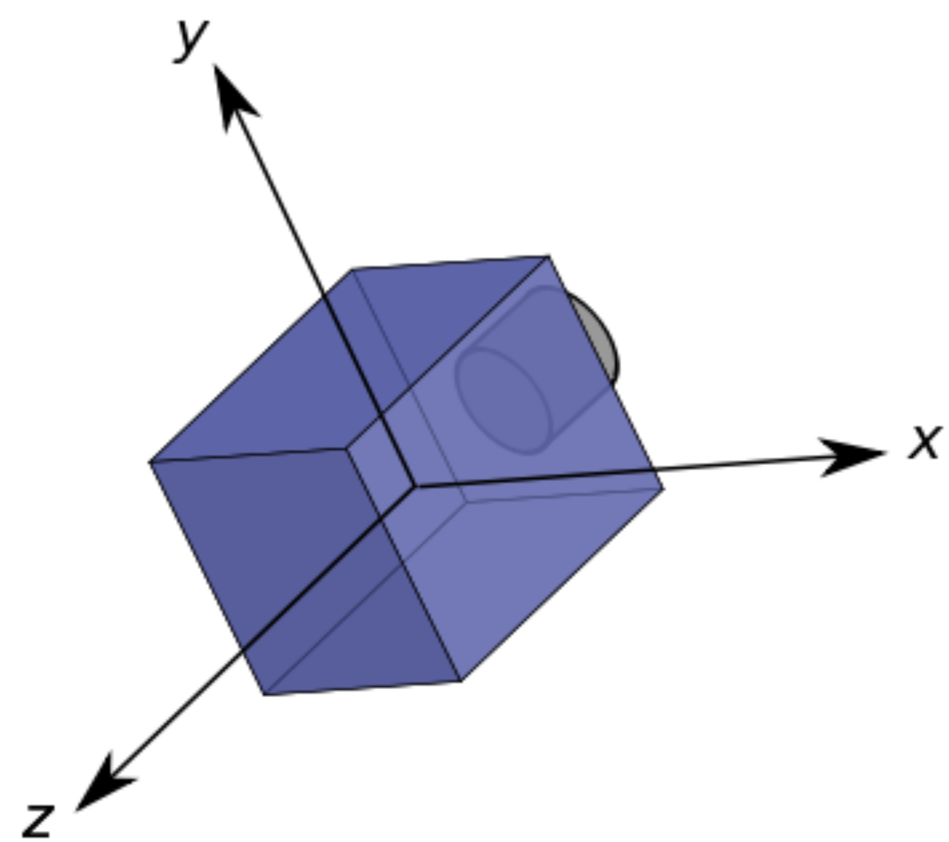
- Can represent any linear transformation as a single matrix.
  - Obtain 3D rotation by concatenating 2D rotations
  - Rotation about an arbitrary point: translate to origin, rotate, then translate back

Model-view transformation



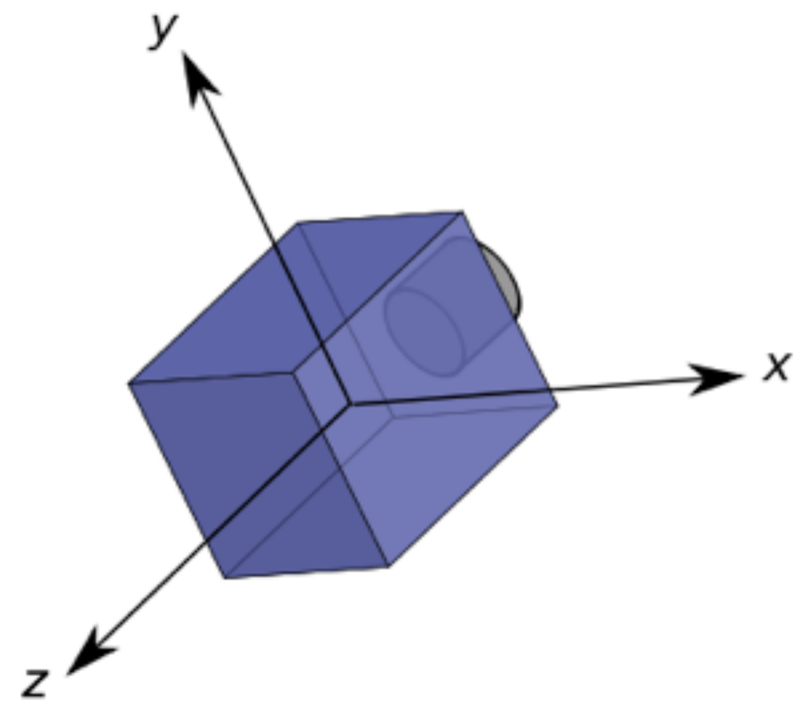
- The camera frame as fixed.
- The model-view transformation positions the object in front of the camera (not the other way around).
- In practice, often maintain two transforms: model-to-world and world-to-view

Model-view transformation



- The camera is at the origin and looks along the negative z direction
- The up direction is the y-axis
- x is oriented so that the directions form a right-handed coordinate system

Model-view transformation

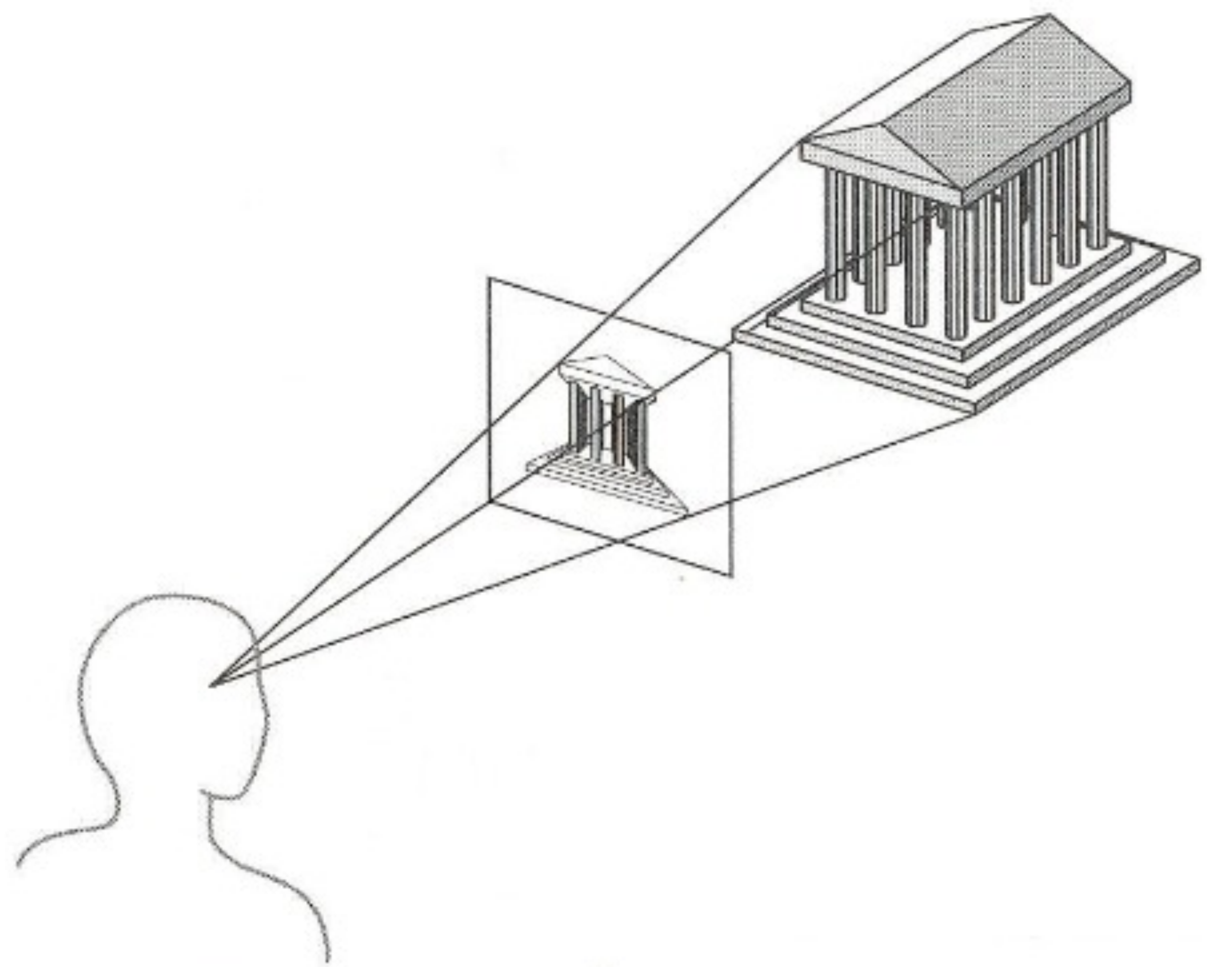


Example: The model-view matrix

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -d \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

positions the origin of the object d units in front of the camera.

Projection transformation



- The default projection in OpenGL is orthographic
- We will often use perspective projection

Orthographic projection

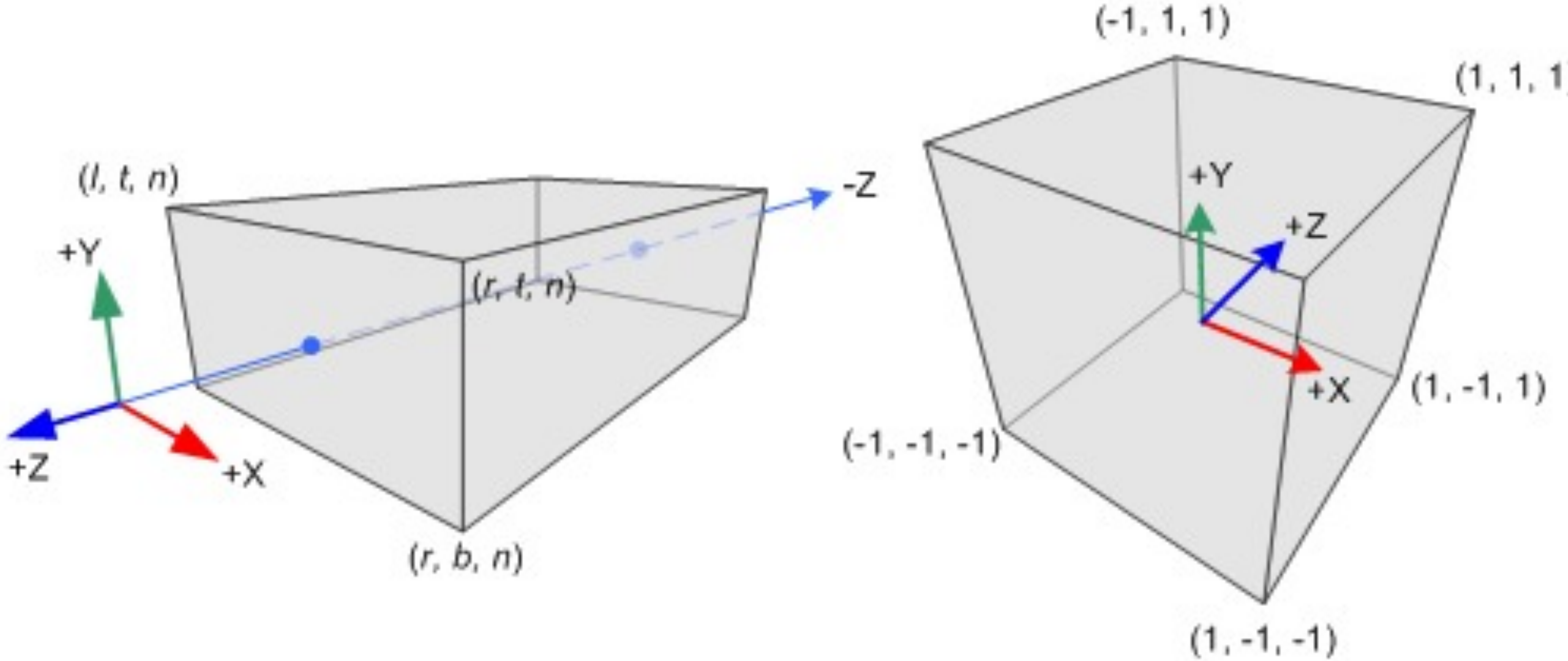


image from http://www.songho.ca/opengl/gl_projectionmatrix.html

Orthographic projection

To construct the projection matrix, we combine translation by

$$\left(\frac{l+r}{2}, \frac{t+b}{2}, \frac{f+n}{2}\right)$$

with scaling by

$$\left(\frac{2}{r-l}, \frac{2}{t-b}, \frac{2}{f-n}\right)$$

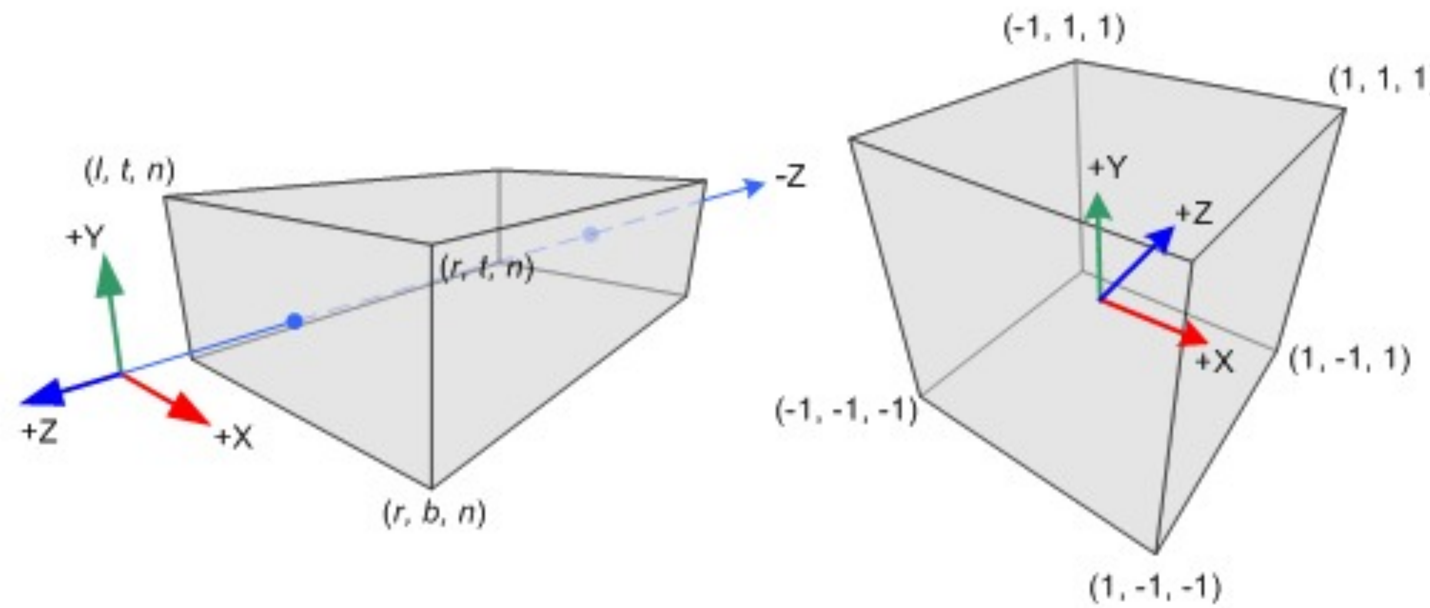


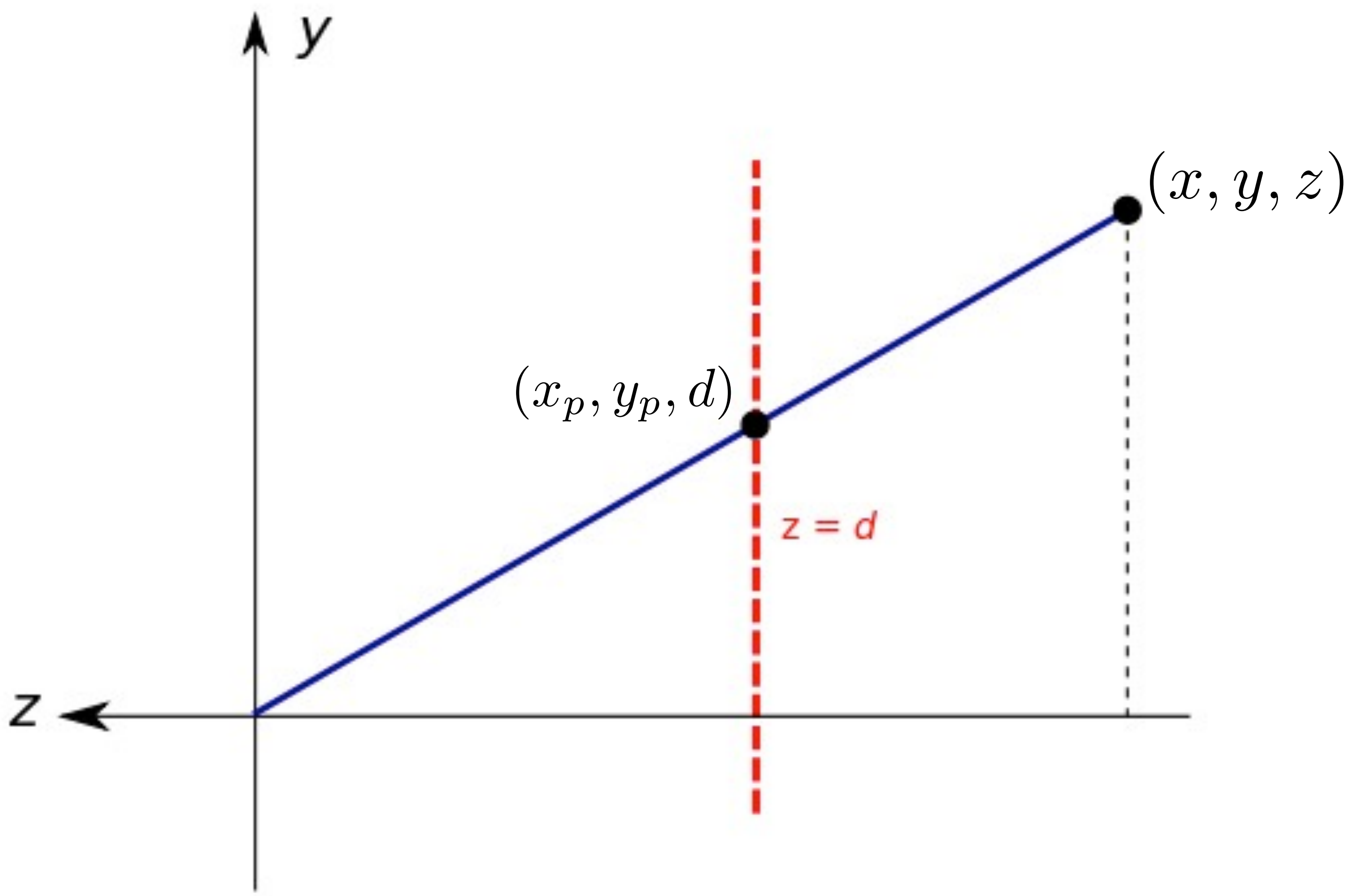
image from http://www.songho.ca/opengl/gl_projectionmatrix.html

Orthographic projection

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

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

Perspective projection



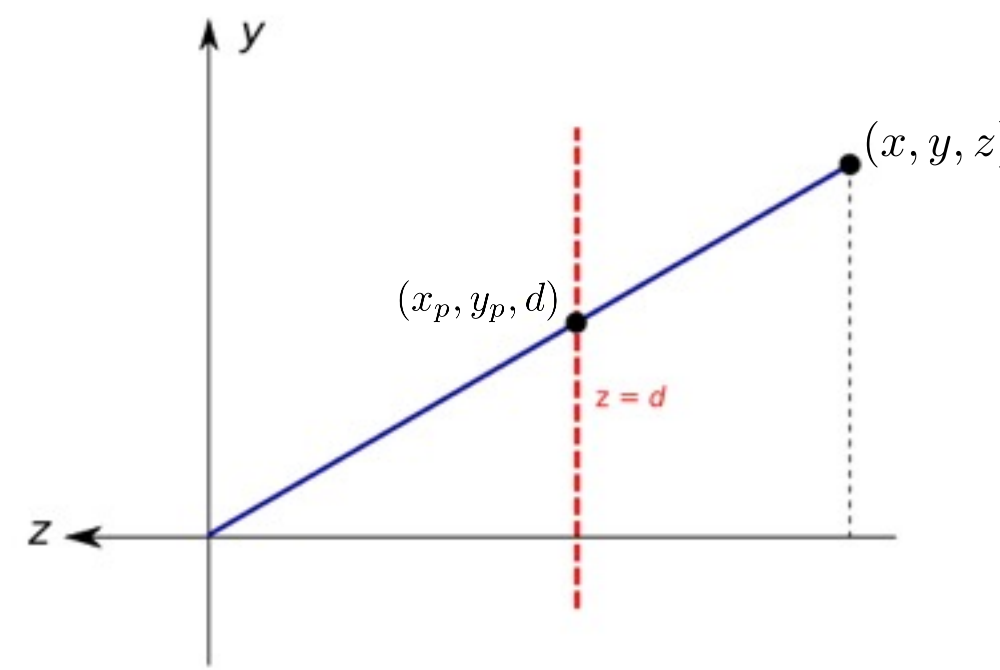
Similar triangles:

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

Perspective projection

In matrix form:

$$M = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{pmatrix}$$



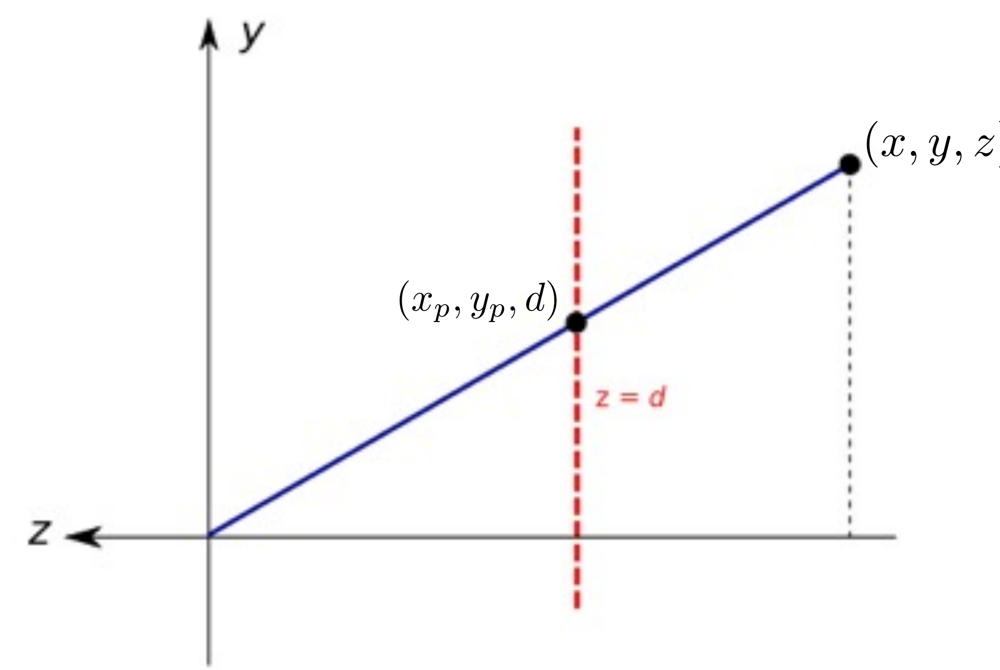
$$p = \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

$$Mp = \begin{pmatrix} x \\ y \\ z \\ z/d \end{pmatrix} = \begin{pmatrix} \frac{x}{z/d} \\ \frac{y}{z/d} \\ d \\ 1 \end{pmatrix}$$

Perspective projection

For the case d=-1:

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



$$p = \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

$$Mp = \begin{pmatrix} wx \\ wy \\ wz \\ -wz \end{pmatrix} = \begin{pmatrix} -\frac{x}{z} \\ -\frac{y}{z} \\ -1 \\ 1 \end{pmatrix}$$

View frustum

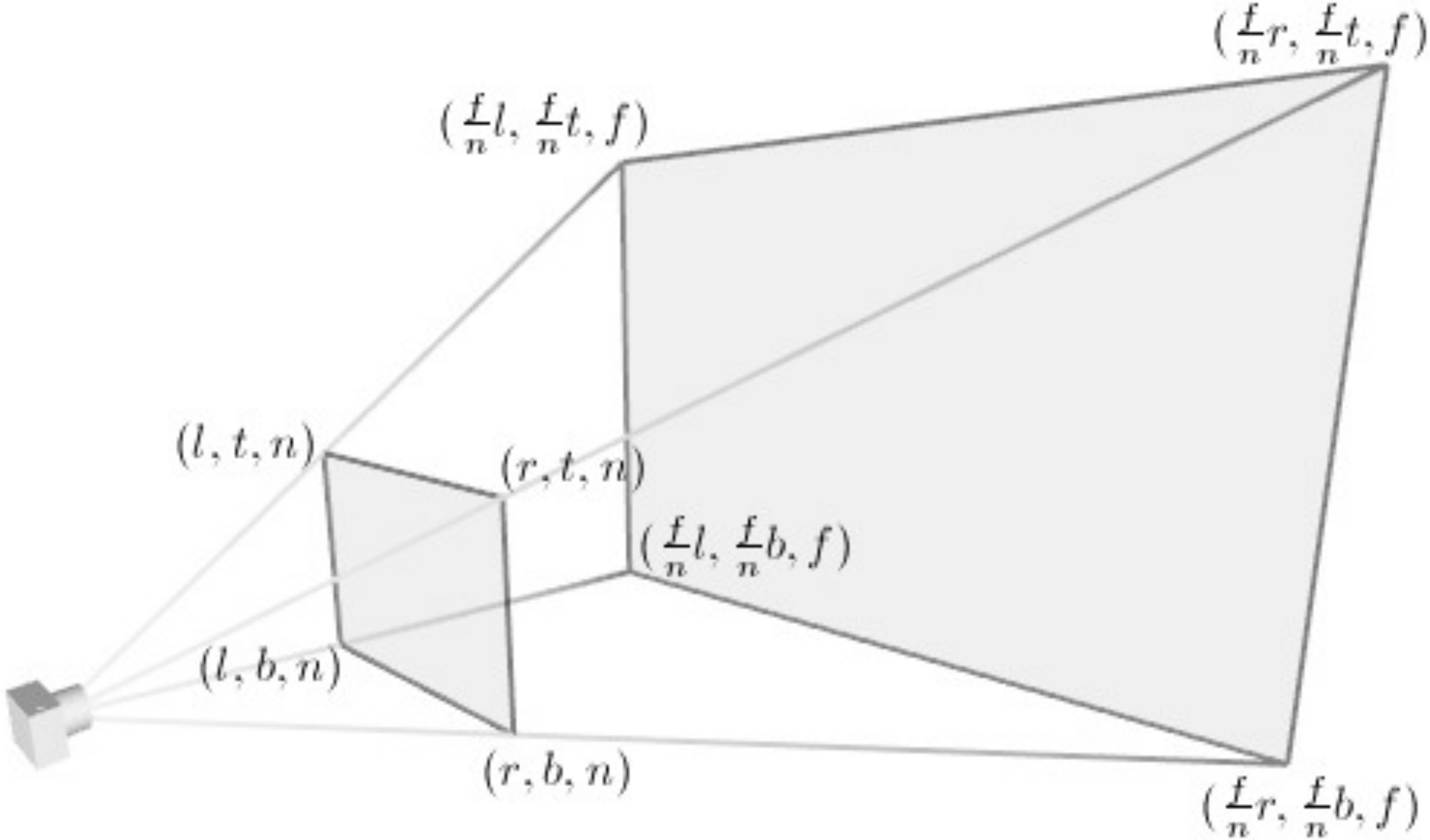


image from http://www.songho.ca/opengl/gl_projectionmatrix.html

Perspective projection

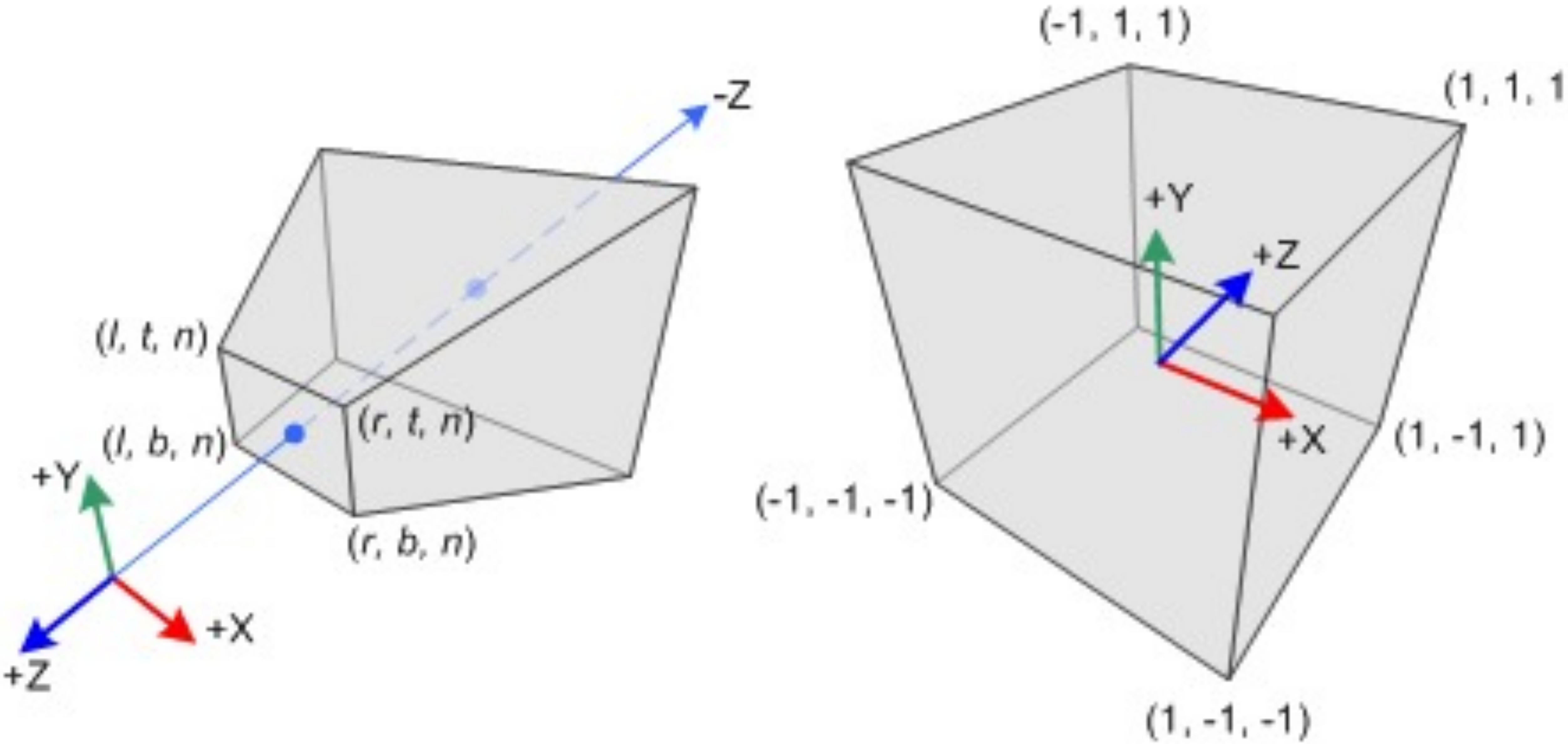
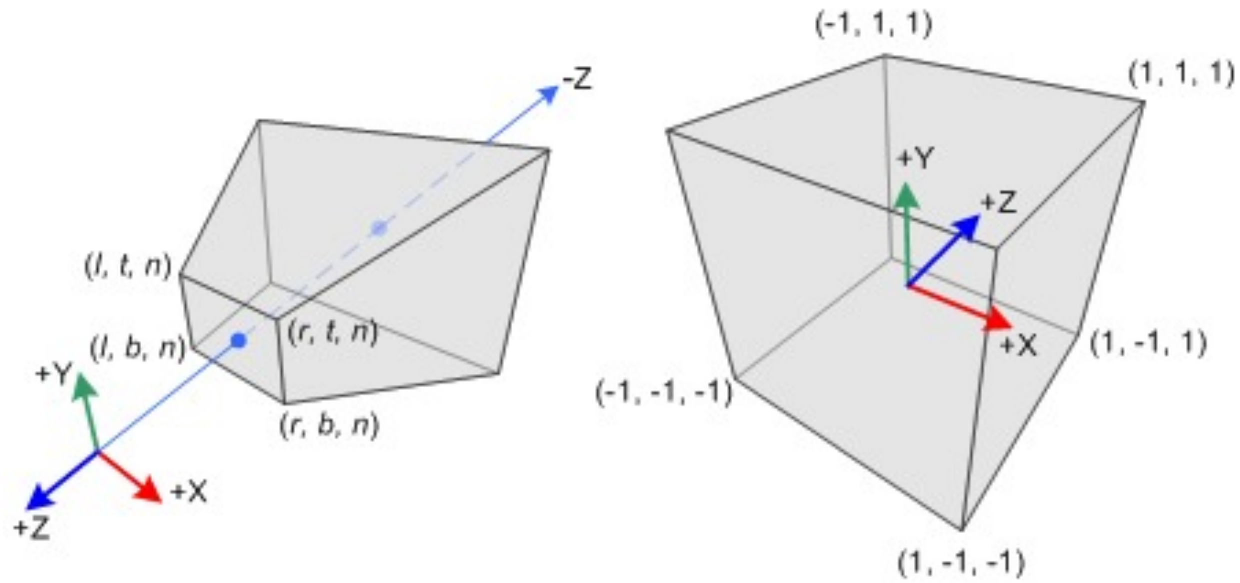


image from http://www.songho.ca/opengl/gl_projectionmatrix.html

Perspective projection



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

In practice

- OpenGL provides high-level commands for setting transformation matrices.
  - gluLookAt() helps set the model-view matrix
  - glFrustum() and glOrtho() help set the projection matrix