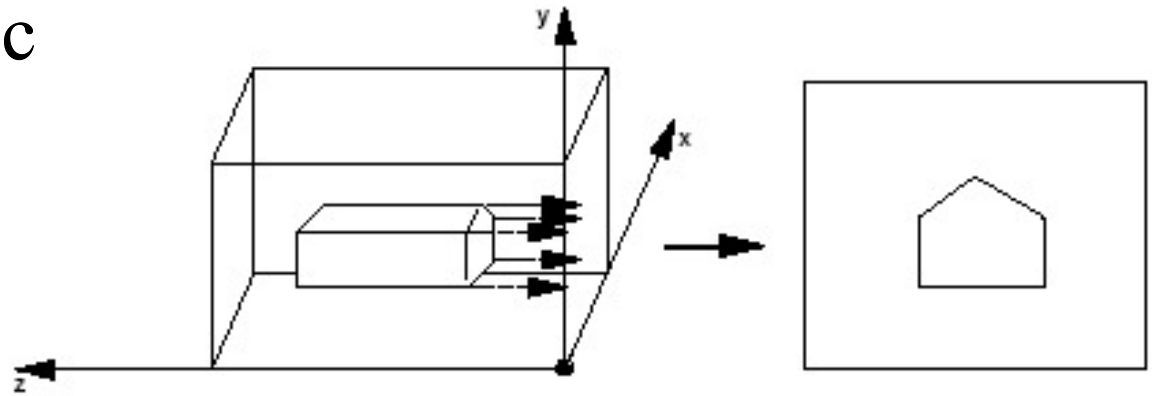
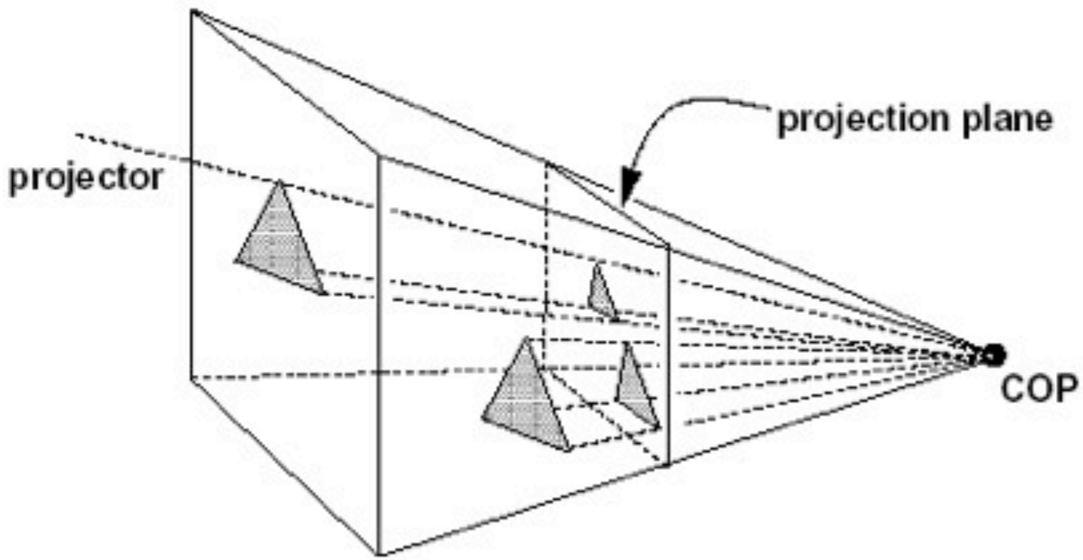
Orthographic vs. Perspective

- Orthographic

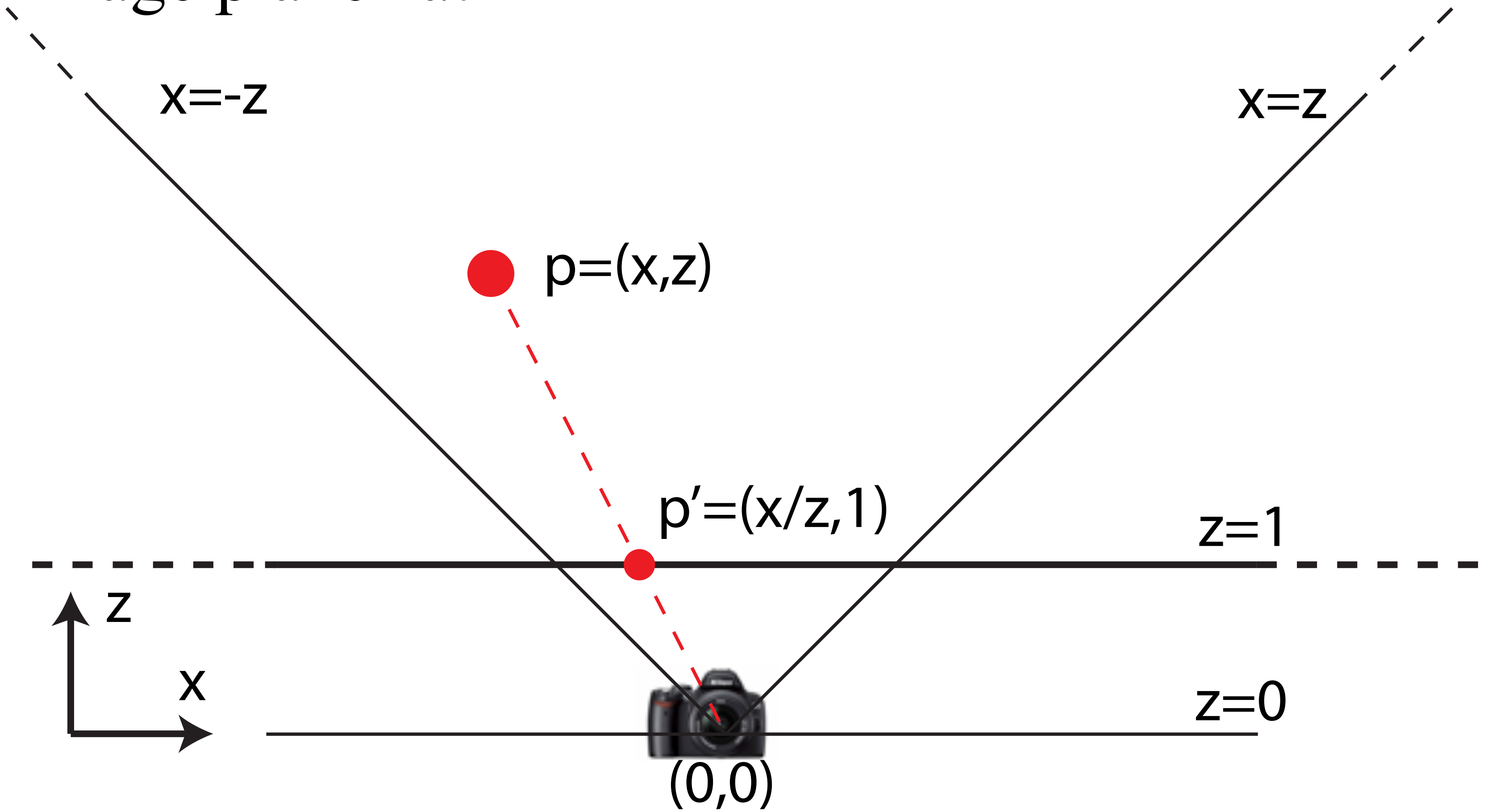


- Perspective



Perspective in 2D

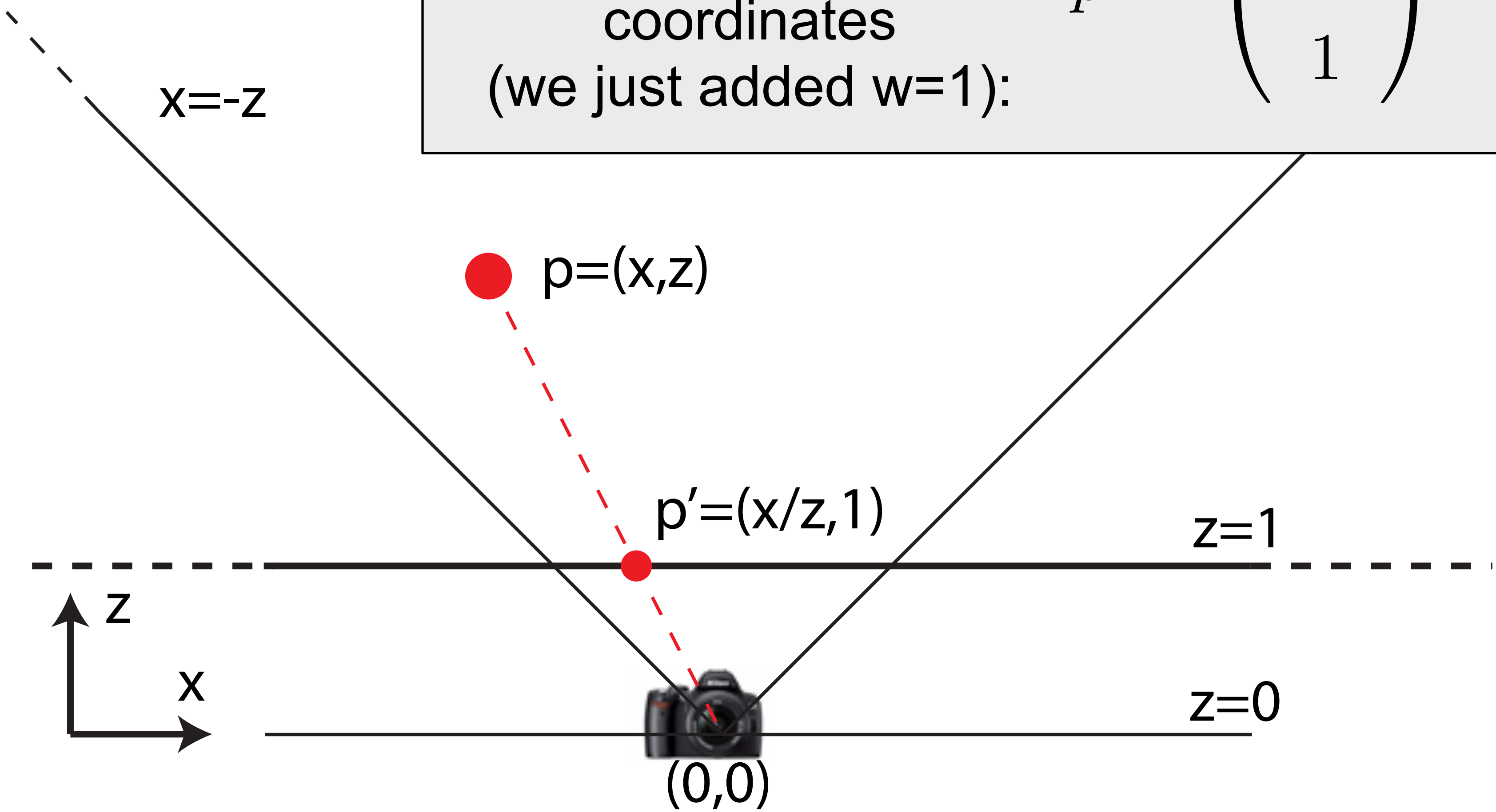
- Camera at origin, looking along z, 90 degree f.o.v., "image plane" at z=1



Perspective in 2D

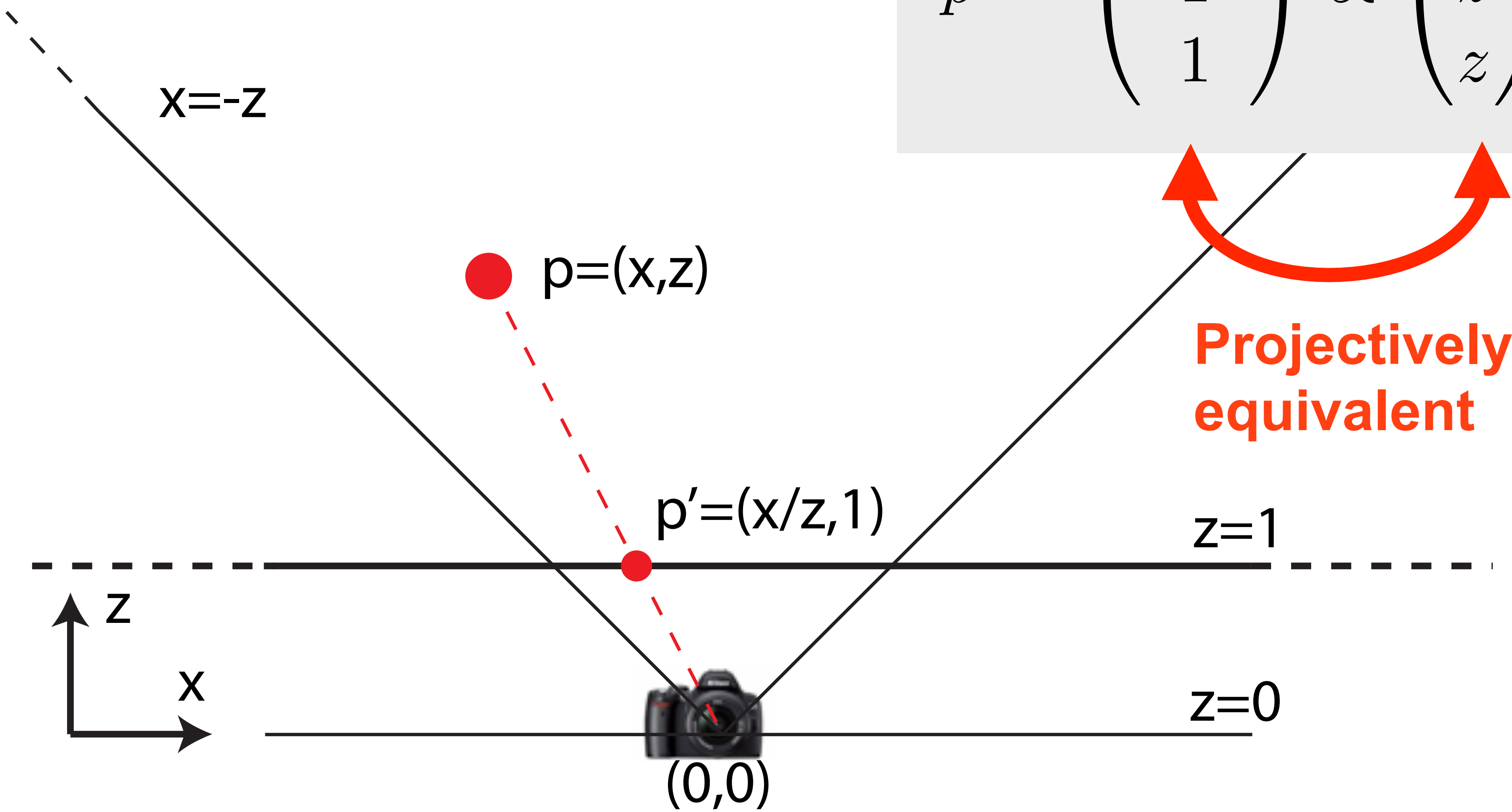
The projected point in homogeneous coordinates (we just added w=1):

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



Perspective in 2D

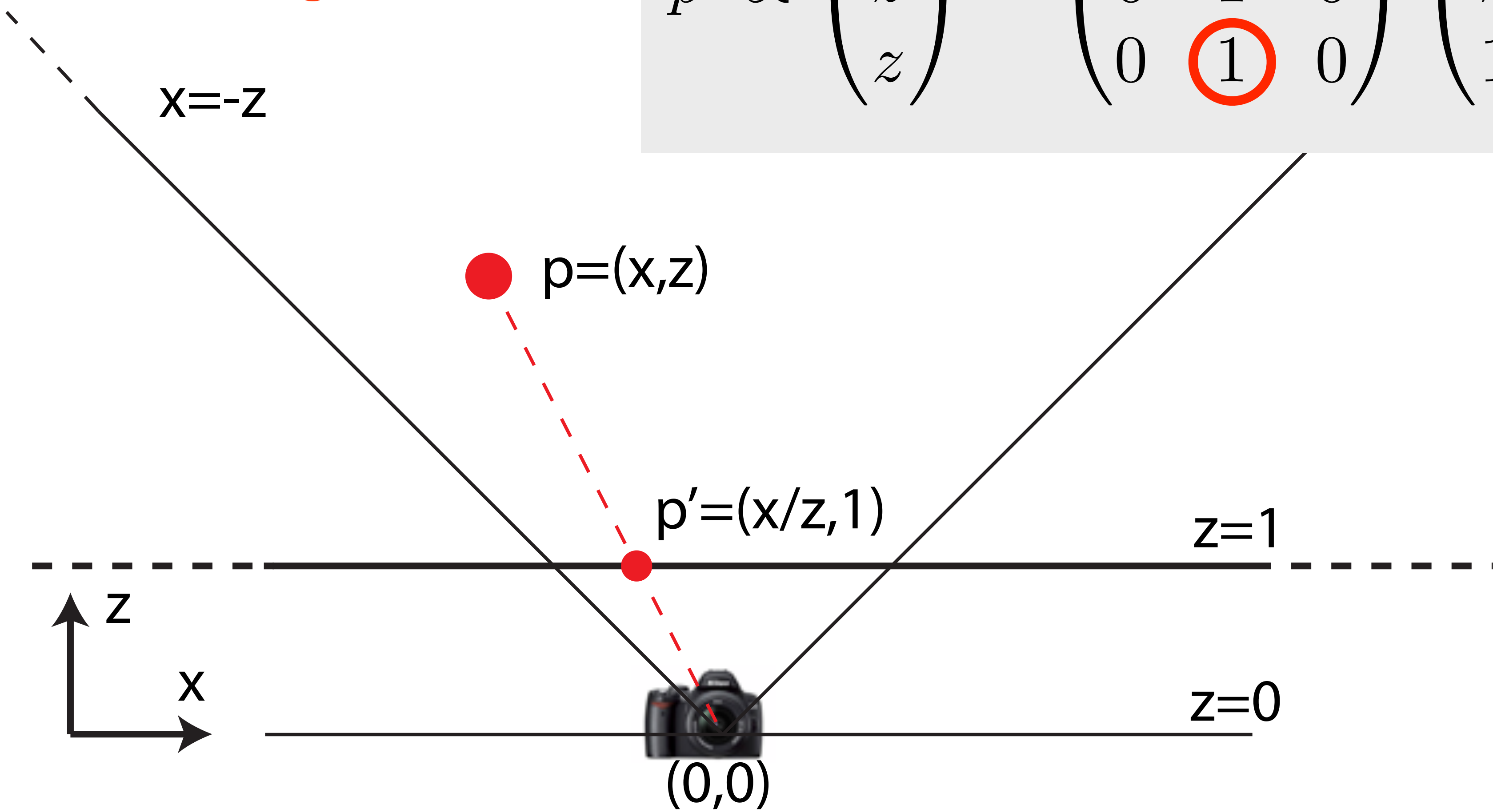
$$p' = \begin{pmatrix} x/z \\ 1 \\ 1 \end{pmatrix} \propto \begin{pmatrix} x \\ z \\ z \end{pmatrix}$$



Perspective in 2D

We'll just copy z to w, and get the projected point after homogenization!

$$p' \propto \begin{pmatrix} x \\ z \\ z \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} x \\ z \\ 1 \end{pmatrix}$$



Extension to 3D

- Trivial:
  Just add another dimension *y* and treat it like *x*
  - *z* is the special one, it turns into *w*'
- Different fields of view and non-square image aspect ratios can be accomplished by simple scaling of the *x* and *y* axes.

$$\begin{pmatrix} x' \\ y' \\ z' \\ w' \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

Caveat

- These projections matrices work perfectly in the sense that you get the proper 2D projections of 3D points.
- However, since we are flattening the scene onto the z=1 plane, we've lost all information about the distance to camera.
  - We need the distance for Z buffering, i.e., figuring out what is in front of what!

Basic Idea: store 1/z

$$\begin{pmatrix} x' \\ y' \\ z' \\ w' \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

Basic Idea: store 1/z

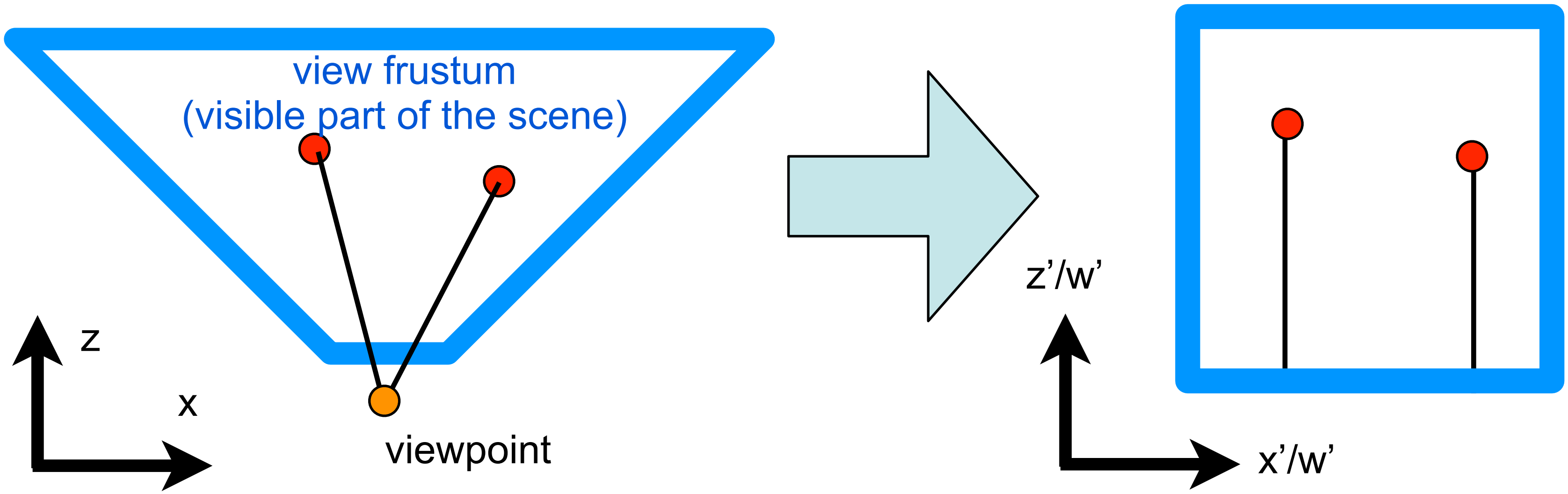
$$\begin{pmatrix} x' \\ y' \\ z' \\ w' \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

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

- z' = 1 before homogenization
- z'=1/z after homogenization

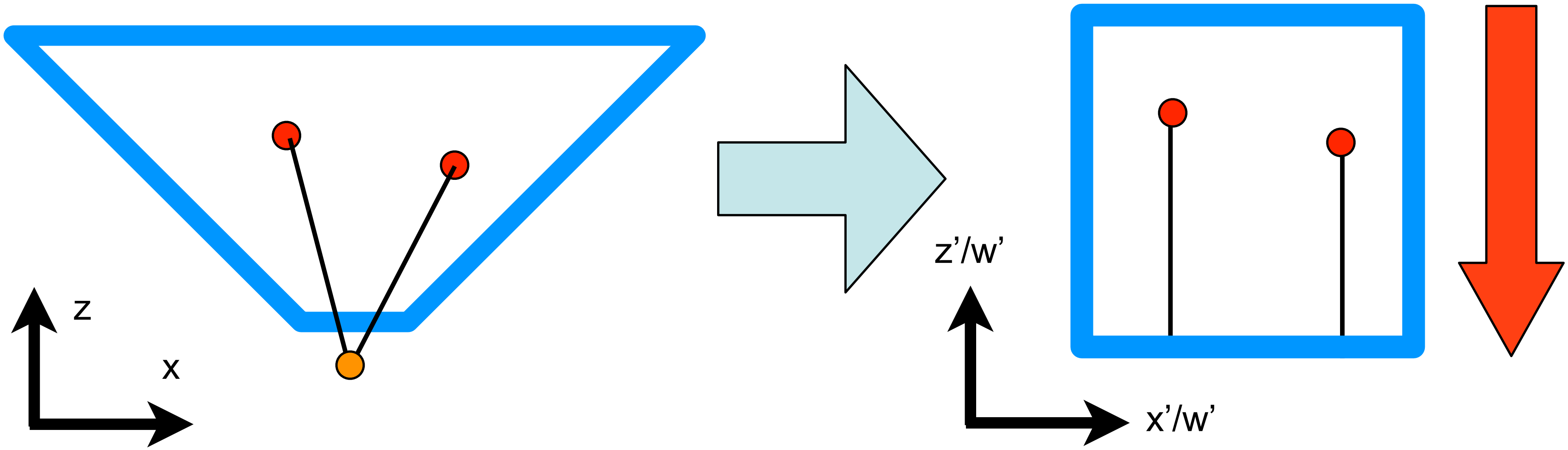
Full Idea: Remap the View Frustum

- We can transform the frustum by a modified projection in a way that makes it a square (cube in 3D) after division by *w*'.



The View Frustum in 2D

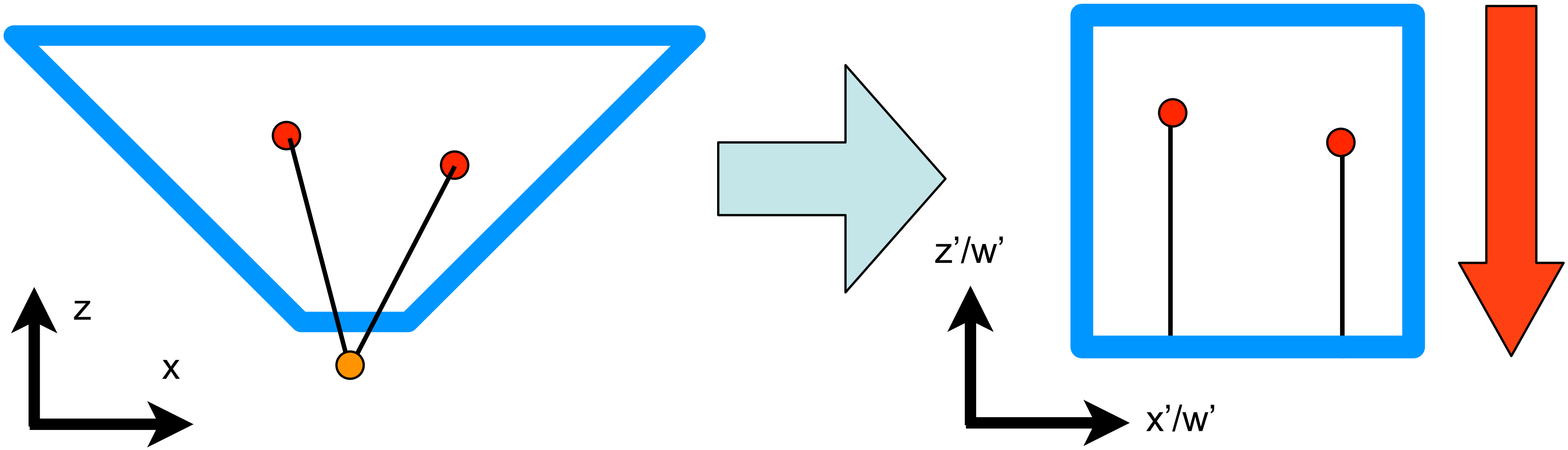
- We can transform the frustum by a modified projection in a way that makes it a square (cube in 3D) after division by *w*'.



The final image is obtained by merely dropping the z coordinate after projection (orthogonal projection)

The View Frustum in 2D

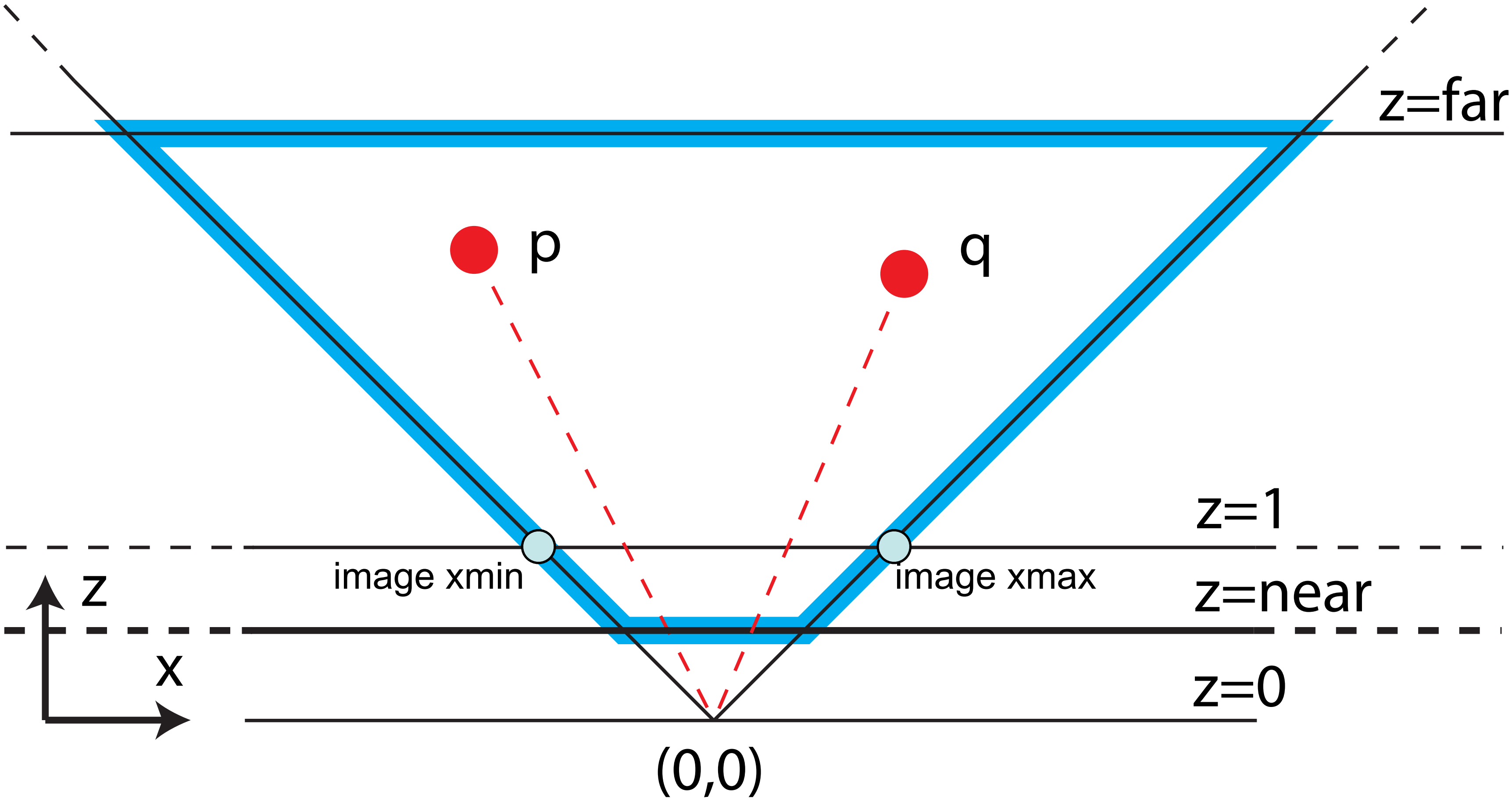
- We can transform the frustum by a modified projection in a way that makes it a square (cube in 3D) after division by *w*'.



- The x' coordinate does not change w.r.t. the usual flattening projection, i.e., x'/w' stays the same

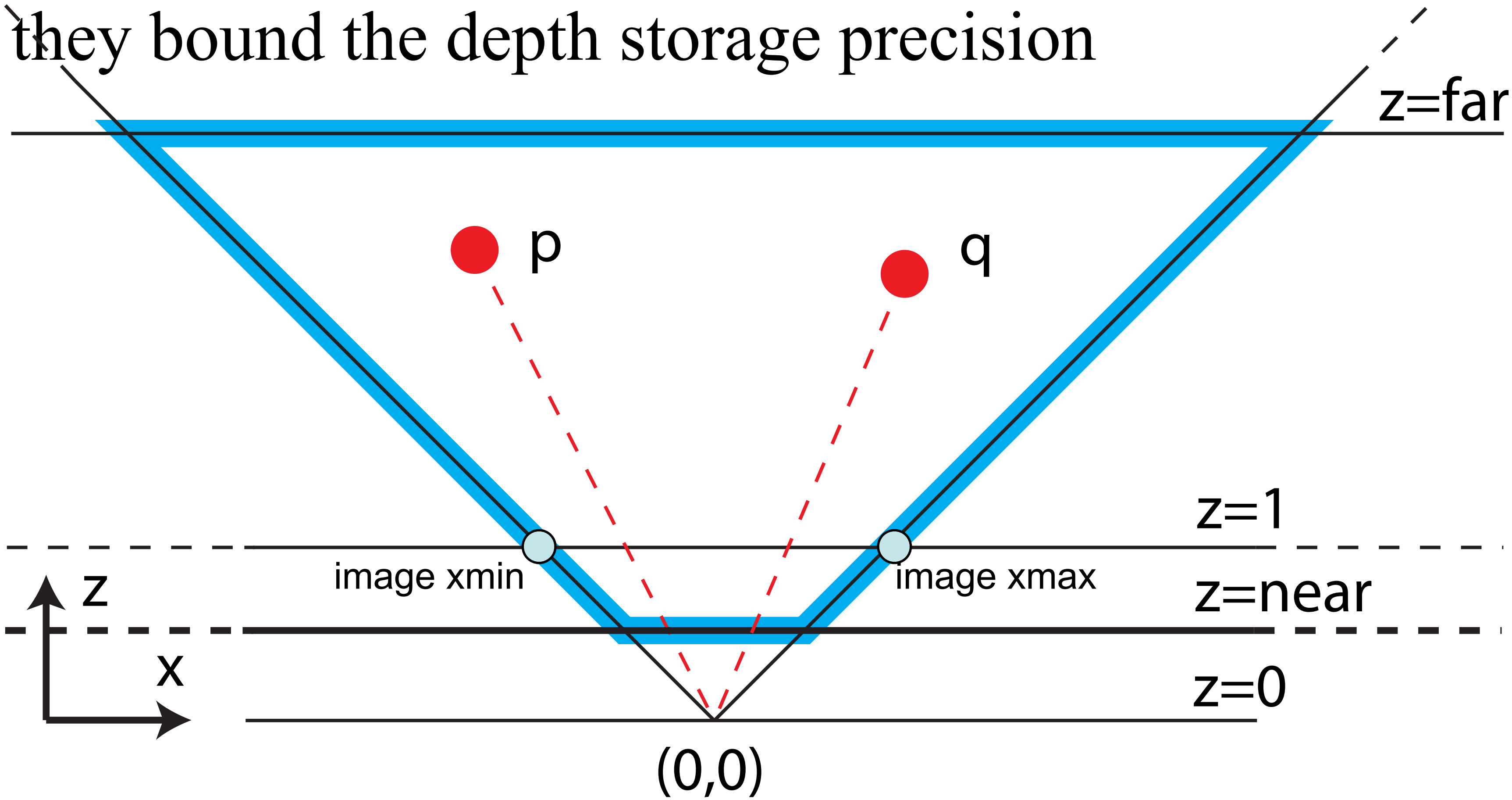
The View Frustum in 2D

- (In 3D this is a truncated pyramid.)

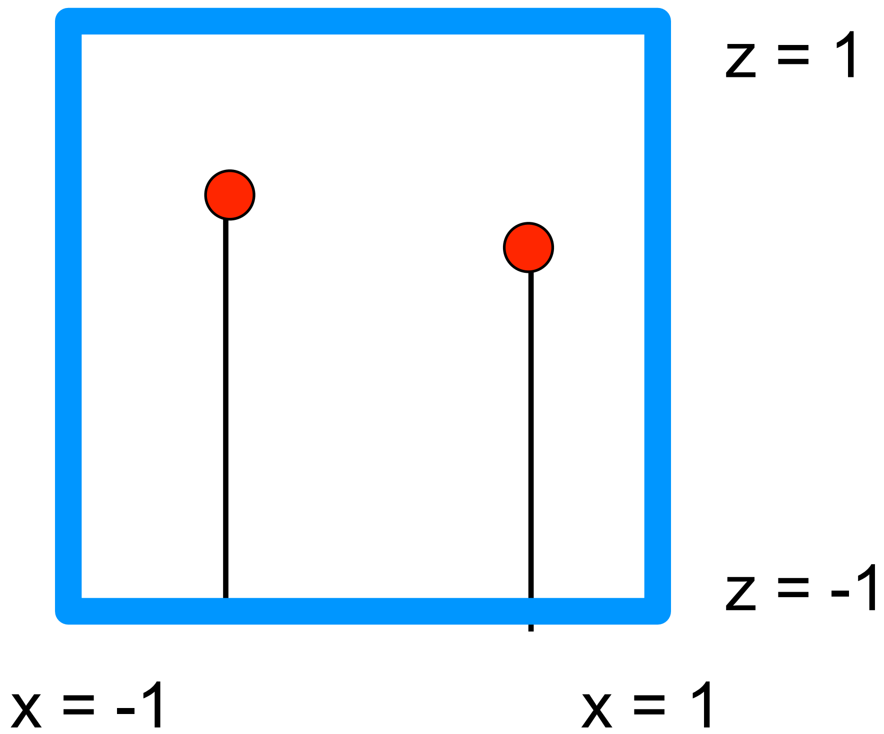


The View Frustum in 2D

- far and near are kind of arbitrary
- they bound the depth storage precision



The Canonical View Volume



- Point of the exercise: This gives screen coordinates and depth values for Z-buffering with unified math
  - Caveat: OpenGL and DirectX define Z differently [0,1] vs. [-1,1]

OpenGL Form of the Projection

$$\begin{pmatrix} x' \\ y' \\ z' \\ w' \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{\text{far}+\text{near}}{\text{far}-\text{near}} & -\frac{2*\text{far}*\text{near}}{\text{far}-\text{near}} \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

Homogeneous coordinates within canonical view volume

Input point in view coordinates

OpenGL Form of the Projection

$$\begin{pmatrix} x' \\ y' \\ z' \\ w' \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{\text{far}+\text{near}}{\text{far}-\text{near}} & -\frac{2*\text{far}*\text{near}}{\text{far}-\text{near}} \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

- z'=(az+b)/z =a+b/z
  - where a & b depend on near & far
- Similar enough to our basic idea:
  - z'=1/z

$$\begin{pmatrix} x' \\ y' \\ z' \\ w' \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

OpenGL Form of the Projection

$$\begin{pmatrix} x' \\ y' \\ z' \\ w' \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{\text{far}+\text{near}}{\text{far}-\text{near}} & -\frac{2*\text{far}*\text{near}}{\text{far}-\text{near}} \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

- Details/more intuition in handout in Stellar
  - "Understanding Projections and Homogenous Coordinates"

Recap: Projection

- Perform rotation/translation/other transforms to put viewpoint at origin and view direction along z axis
  - This is the OpenGL "modelview" matrix
- Combine with projection matrix (perspective or orthographic)
  - Homogenization achieves foreshortening
  - This is the OpenGL "projection" matrix
- **Corollary**: The entire transform from object space to canonical view volume $[-1,1]^3$ is a single matrix