COMP3271 Computer Graphics

Viewing

2019-20

Objectives

Understand the viewing process

Derive the projection matrices used for standard OpenGL projections

Transformation

Three kinds of transformations are involved in the 3D graphics processing pipeline:

- model transformation *M*: It applies to objects in the 3D world coordinate system (the object space);
- view transformation *V*: It maps objects from the 3D world coordinate system to the 3D eye coordinate system, with the origin at the eye-point (viewpoint);
- view projection *P*: It maps objects from the 3D eye-coordinate system to the 2D view plane.

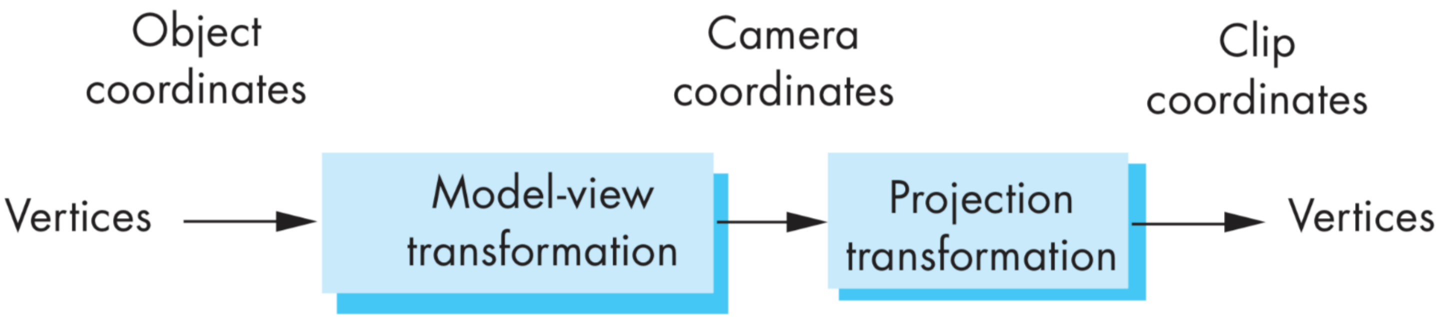
A vertex will be transformed by the concatenation of these transformations before appearing on screen

$$X_3' = P_{3\times 4} V_{4\times 4} M_{4\times 4} X_4.$$

Viewing

There are two main steps in the viewing process:

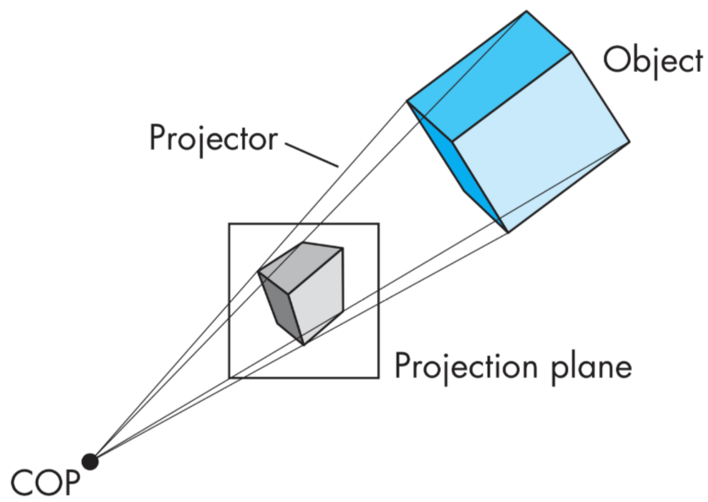
- Position and orient the camera
  - Setting the model-view matrix
  - Vertices in object coordinates will be transformed to eye or camera coordinates
- Selecting a lens
  - Setting the projection matrix
  - Normalize to a canonical view volume



Viewing

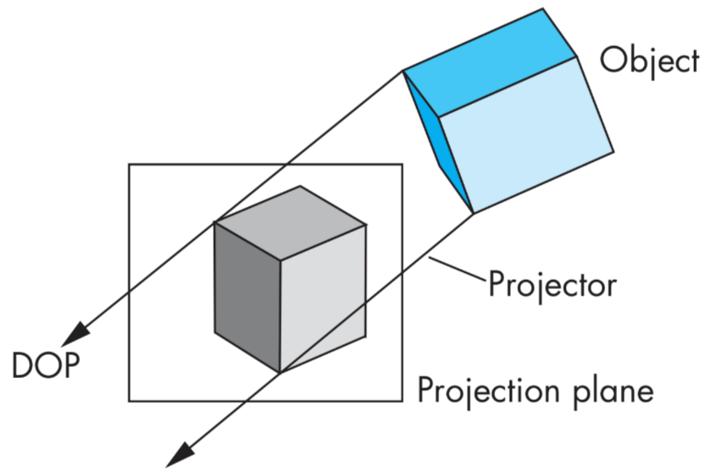
Projection determines how objects appear on screen

Perspective projection



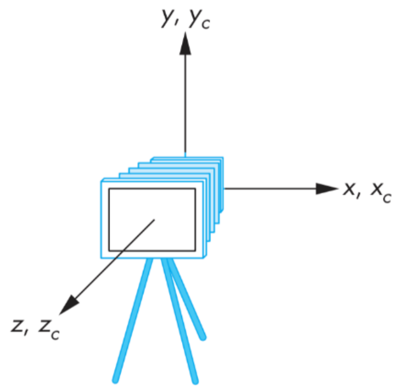
COP: Center of Projection
Original of the camera frame

Orthogonal projection



DOP: Direction of Projection
same as COP at infinity

The OpenGL Camera



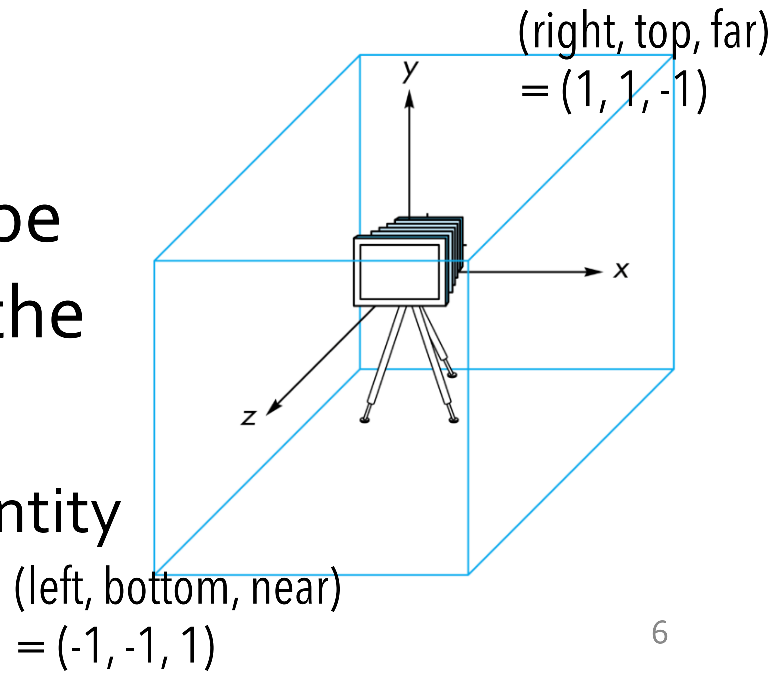
In OpenGL, initially the object and camera frames are the same

- 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 canonical view volume that is a cube with sides of length 2 centered at the origin

- Default projection matrix is an identity (i.e., orthogonal projection)



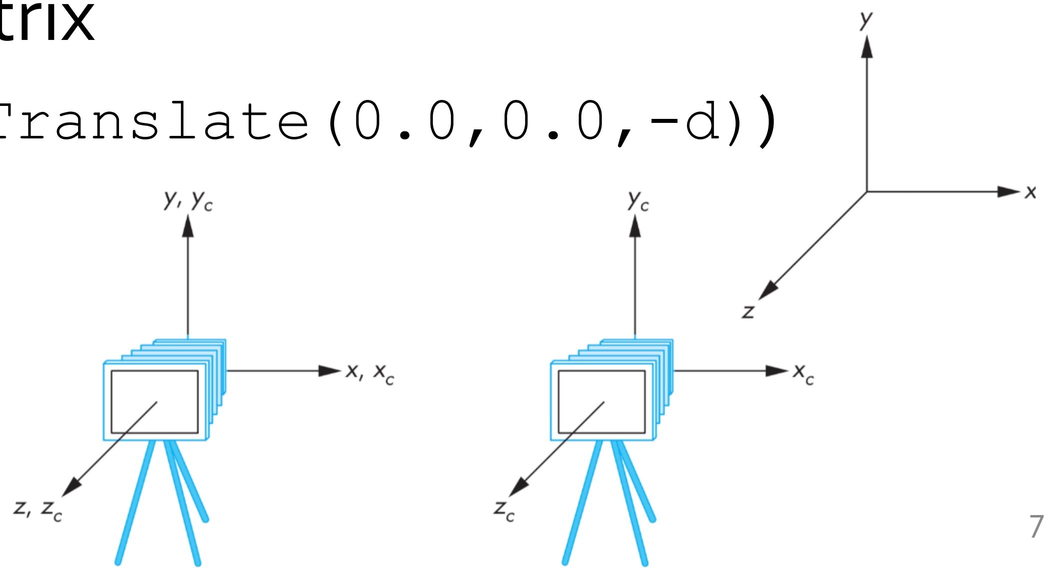
Moving the Camera Frame

Consider

- Moving the camera in the positive z direction
  - Translate the camera frame
- Moving 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 (`Translate(0.0,0.0,-d)`)
- `d > 0`

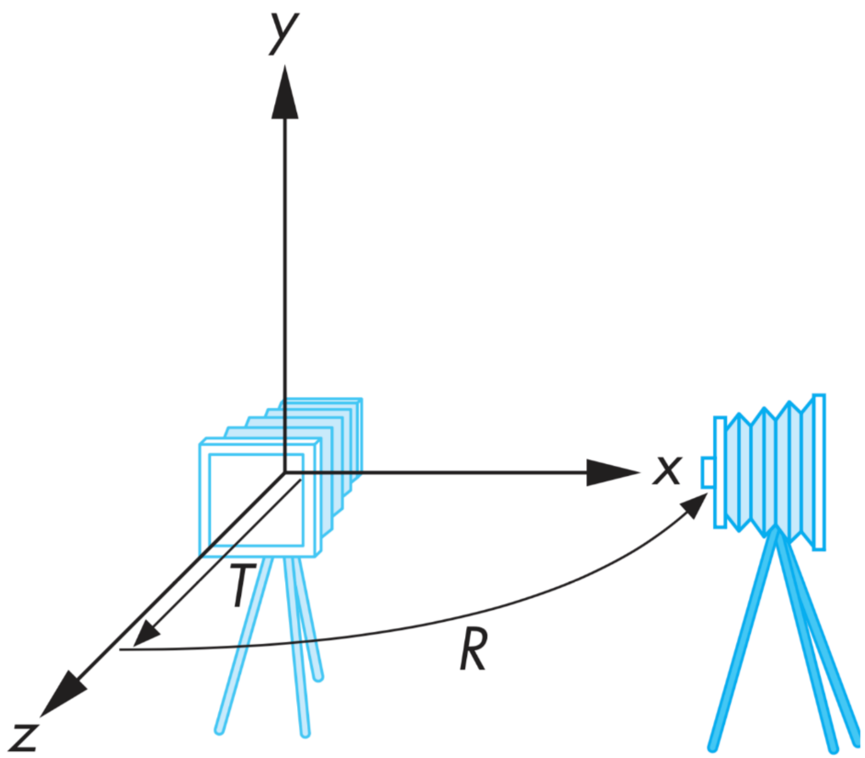


Moving the Camera Frame

We can move the camera to any desired position by a sequence of rotations and translations

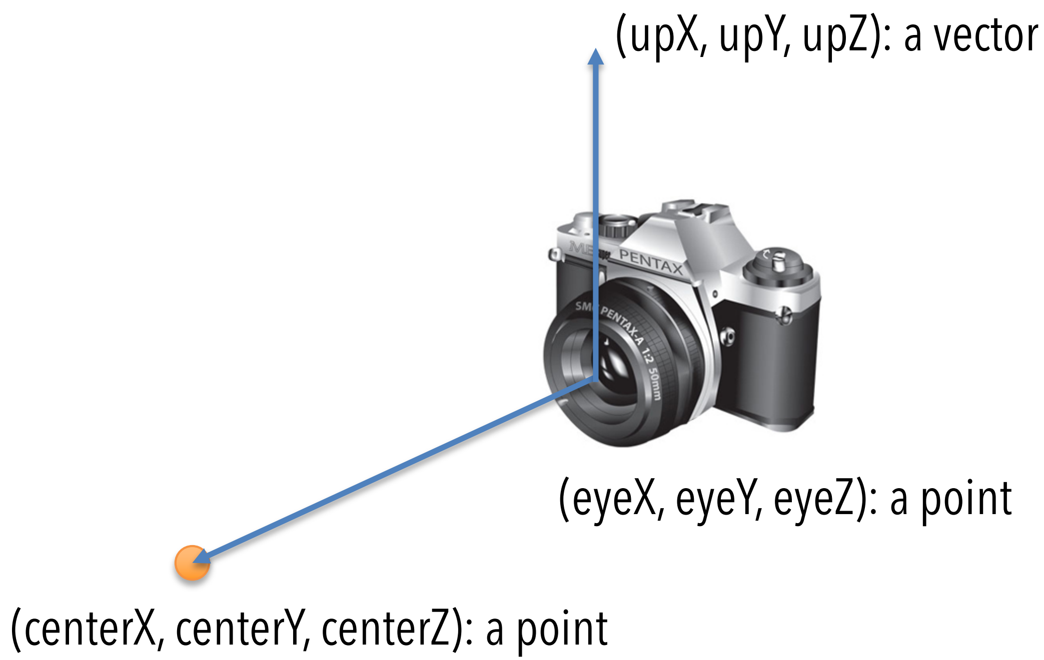
Example: side view

- Rotate the camera
- Move it away from origin
- Model-view matrix C = TR



OpenGL API

```
LookAt(eyeX,eyeY,eyeZ,
 centerX,centerY,centerZ,upX,upY,upZ);
```



Note that this is a transformation that applies to the ModelView matrix

Projections and Normalization

The default projection in the eye (camera) frame is orthogonal

For points within the default view volume

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

Projection plane at $z = 0$

Most graphics systems use view normalization

- All other views are converted to the canonical view by transformations that determine the projection matrix
- Allows use of the same pipeline for all views

Default orthographic projection

$$\mathbf{p}_\mathrm{p} = \mathbf{M}\mathbf{p}$$

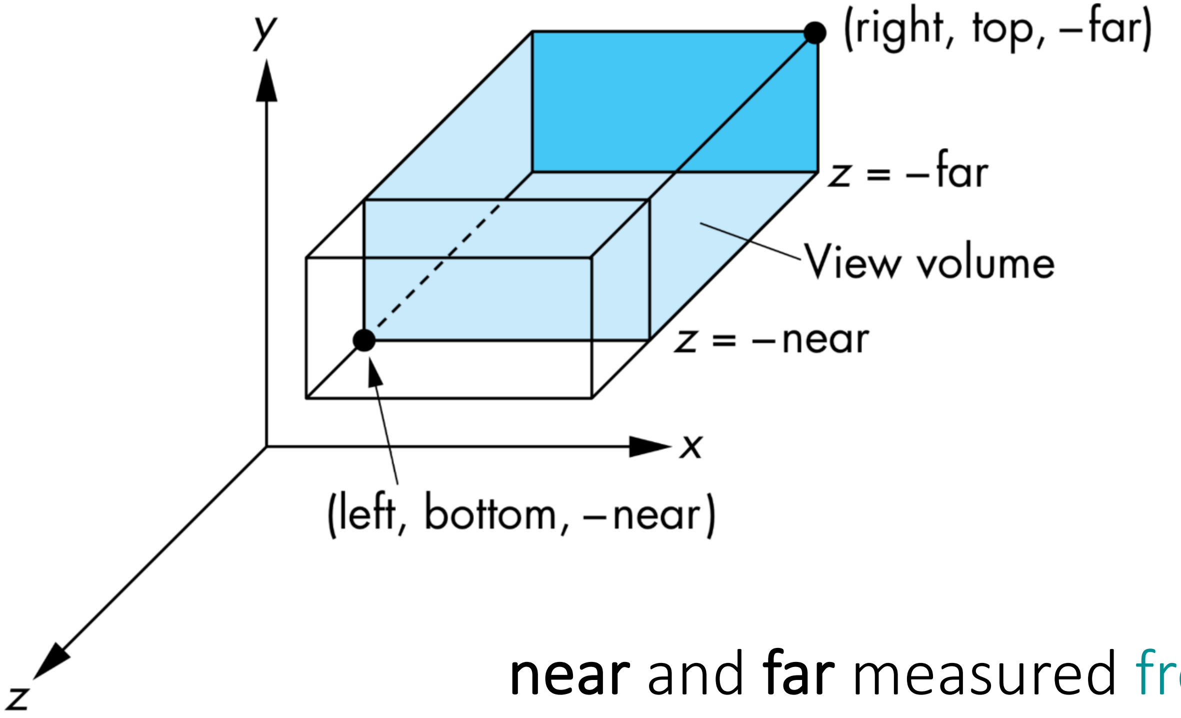
$$\mathbf{M}_{\mathbf{orth}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{p} = (x, y, z, 1)^\mathrm{T}$$

$$\mathbf{p}_\mathrm{p} = (x, y, 0, 1)^\mathrm{T}$$

OpenGL Orthogonal Viewing

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

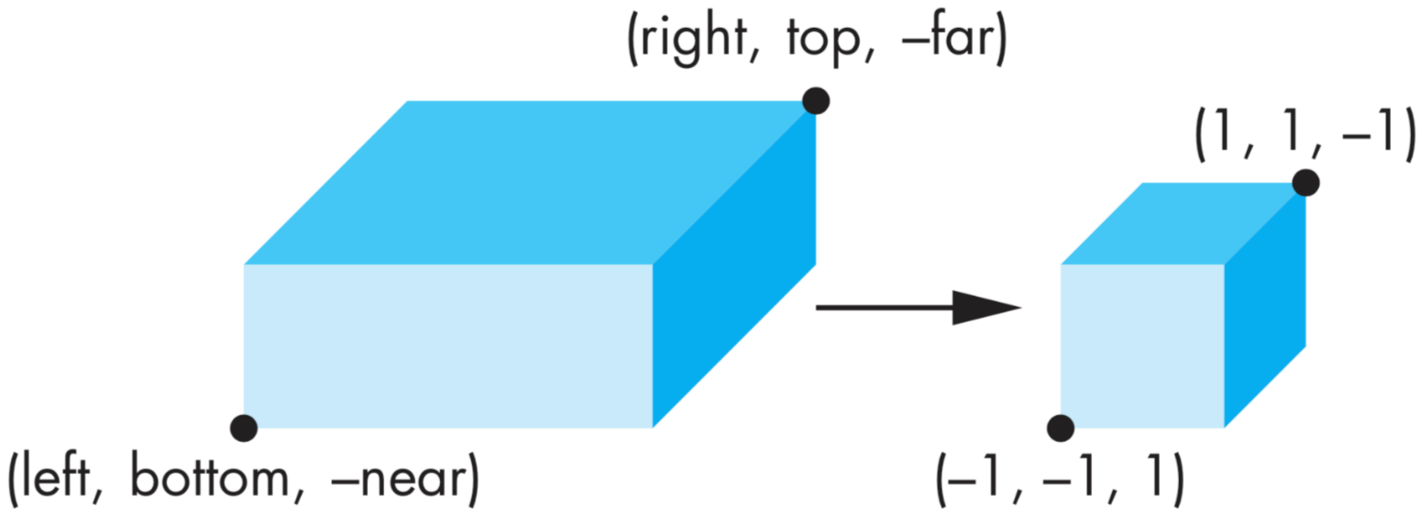


near and **far** measured from camera

How to normalize this into the canonical view?

Orthogonal Normalization

normalization $\Rightarrow$ find transformation to convert specified clipping volume to canonical volume



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

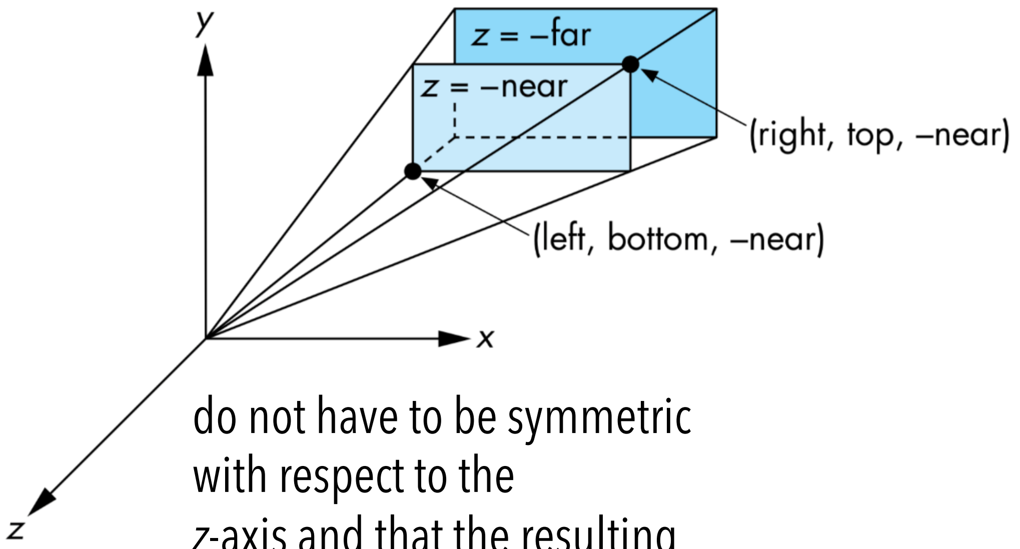
$$\mathbf{M}_{\text{orth}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Hence, general orthogonal projection in 4D is

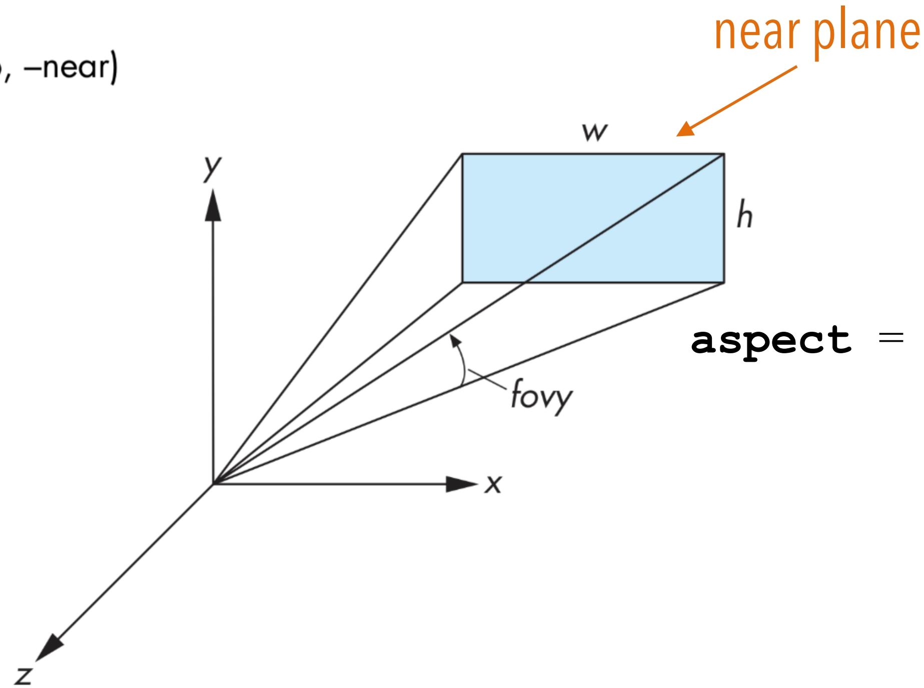
$$\mathbf{P} = \mathbf{M}_{\text{orth}}\mathbf{S}\mathbf{T}$$

OpenGL Perspective Viewing

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



do not have to be symmetric with respect to the *z*-axis and that the resulting frustum also does not have to be symmetric



`aspect` = **`w/h`**

`Perpective(fovy, aspect, near, far)`

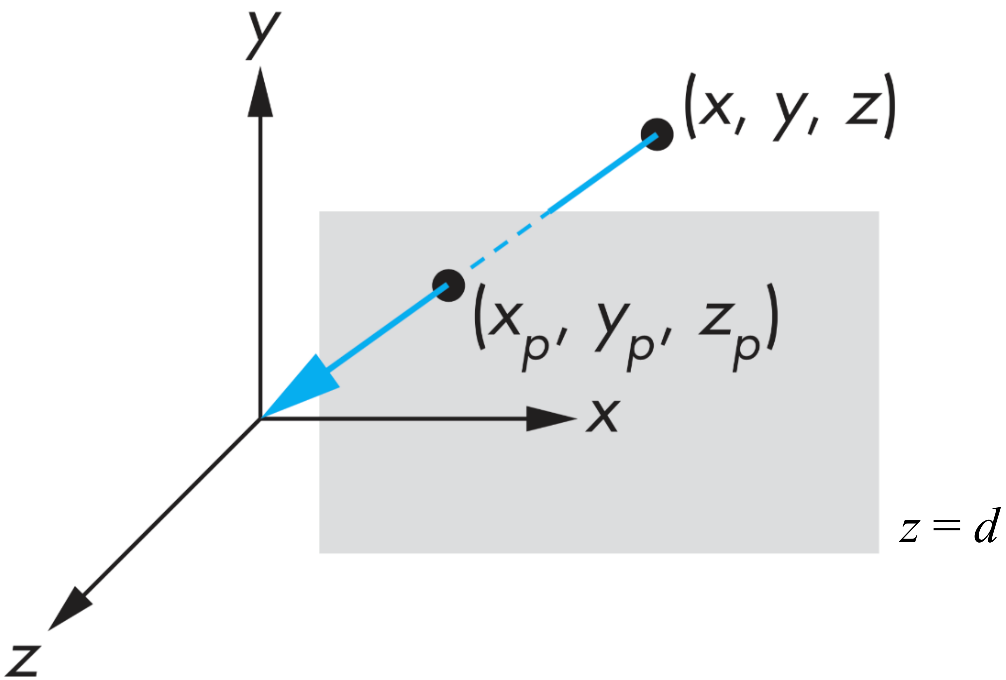
`fovy`: field of view in degrees in y direction

this often provides a better interface

Simple Perspective

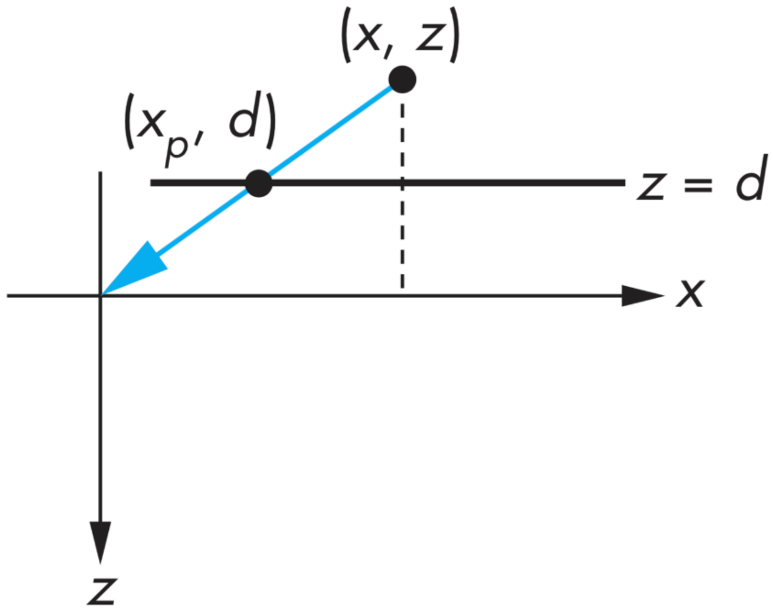
Center of projection at the origin

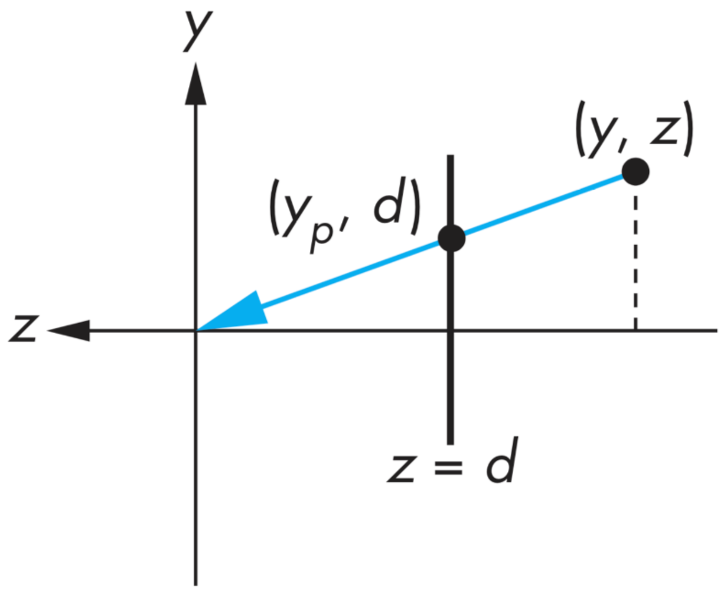
Projection plane $z = d$, $d < 0$



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 $\mathbf{p} = \mathbf{Mq}$ where

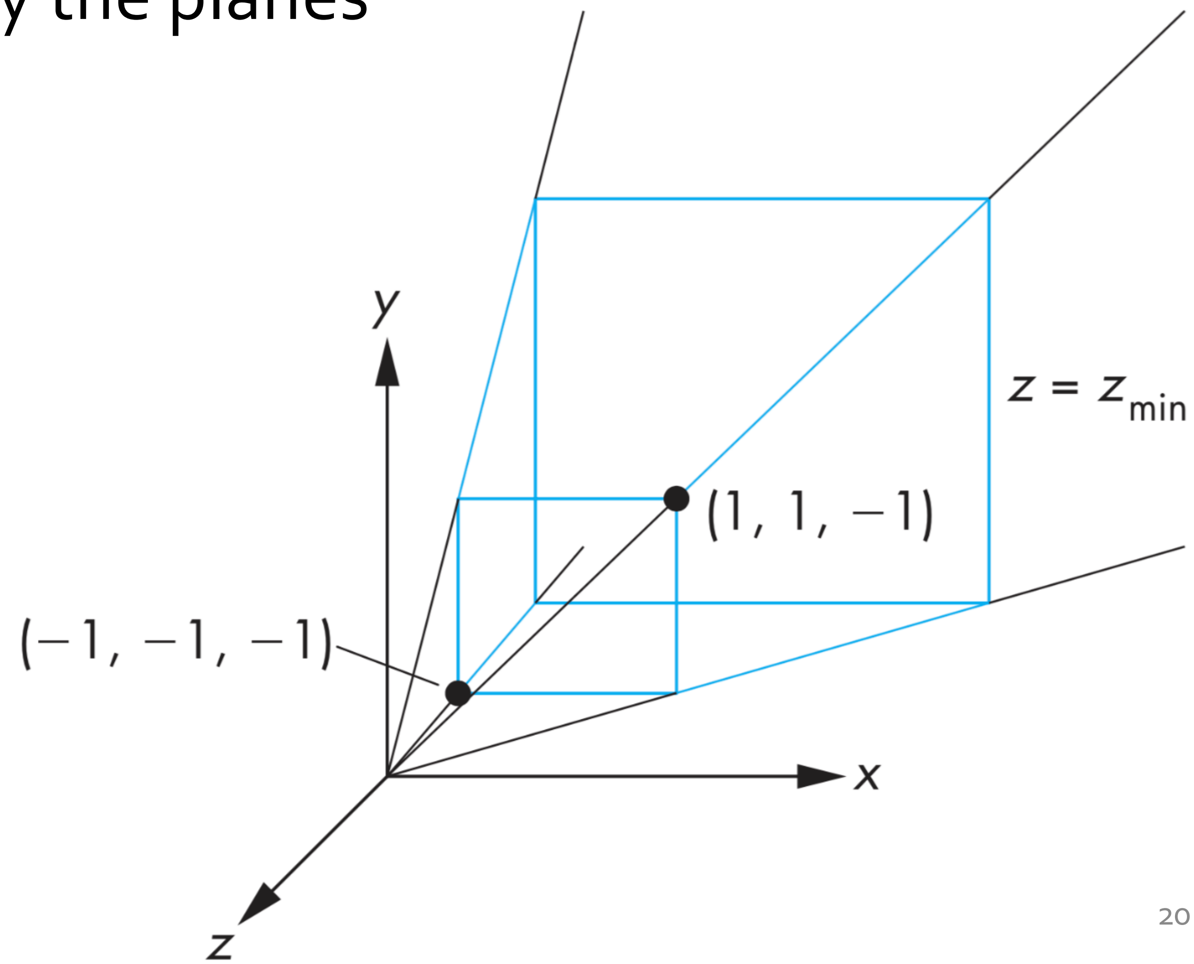
$$\mathbf{M} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{pmatrix}$$

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

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

Generalization

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

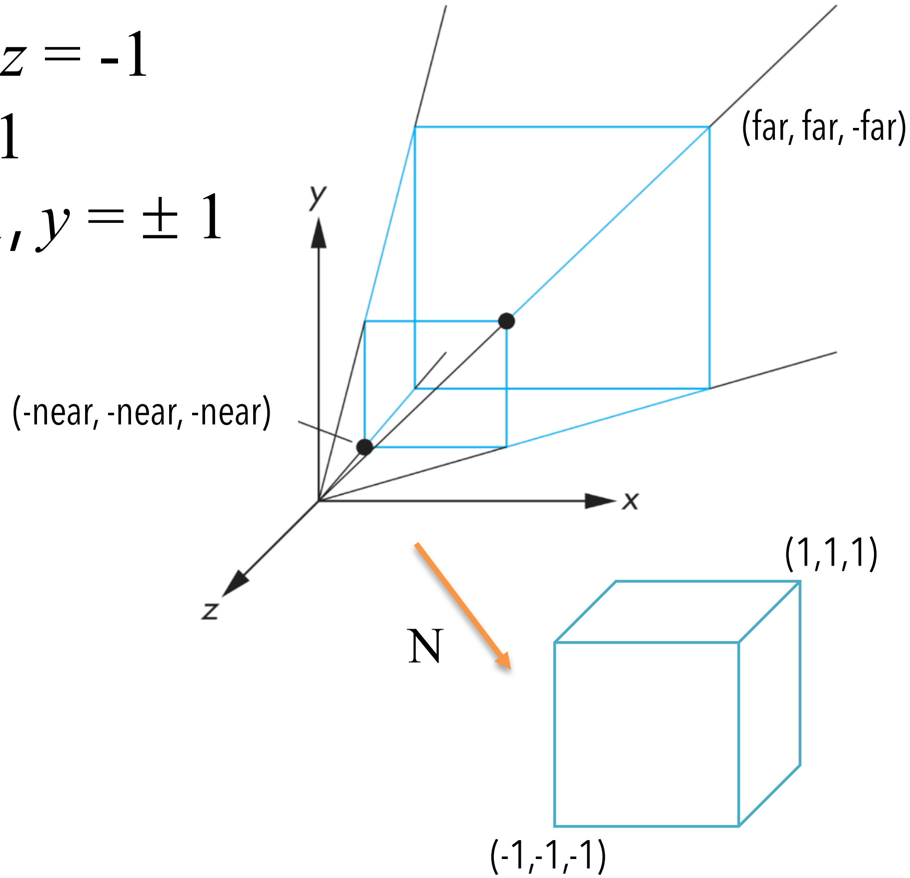
By this mapping, the point $(x, y, z, 1)$ goes to

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

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

Picking α and β

We want:
near plane z = -near be mapped to z = -1
far plane z = -far be mapped to z =1
and the sides be mapped to $x = \pm 1$, $y = \pm 1$



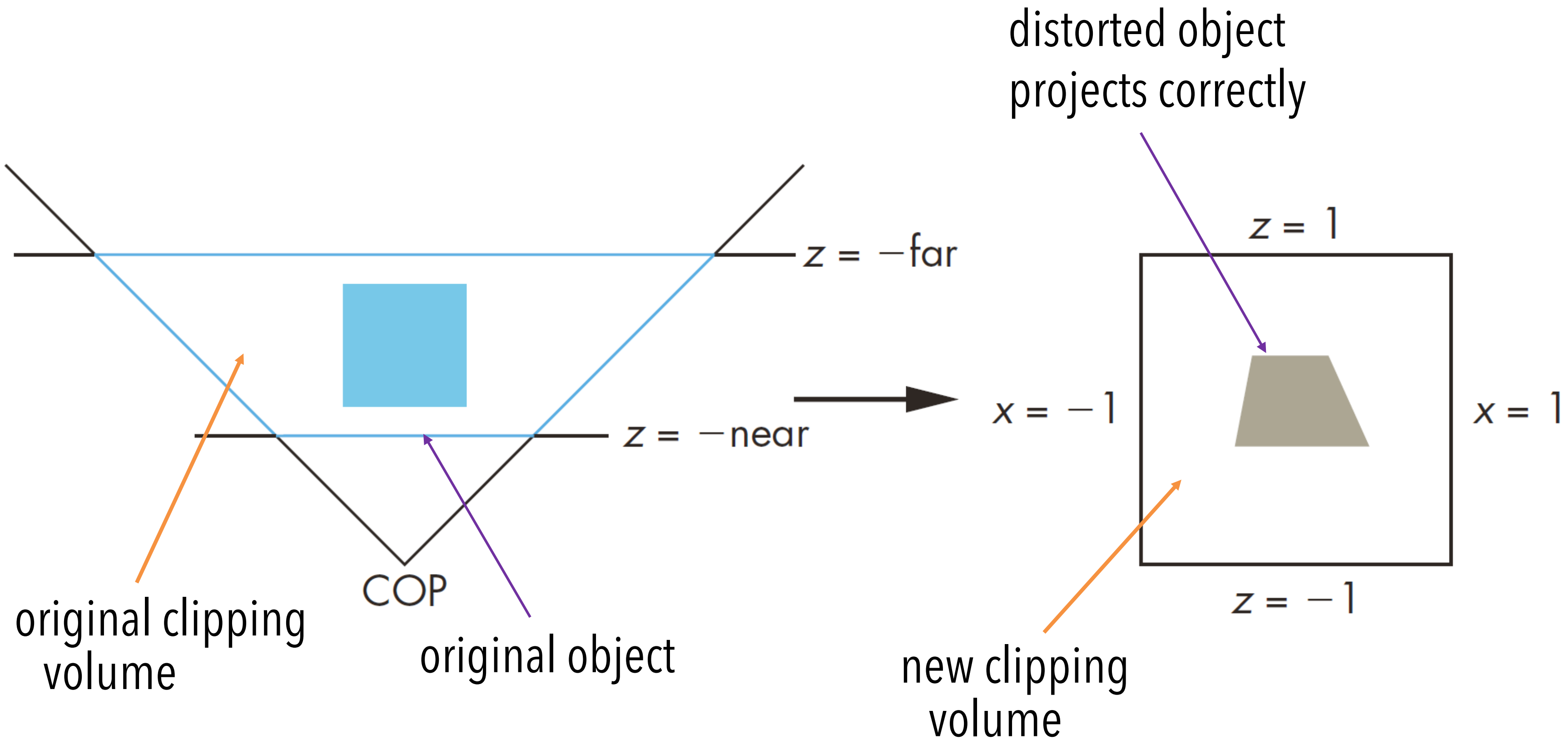
Solving two linear equations,
we have

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

$$\beta = \frac{2 \times \text{near} \times \text{far}}{\text{near} - \text{far}}$$

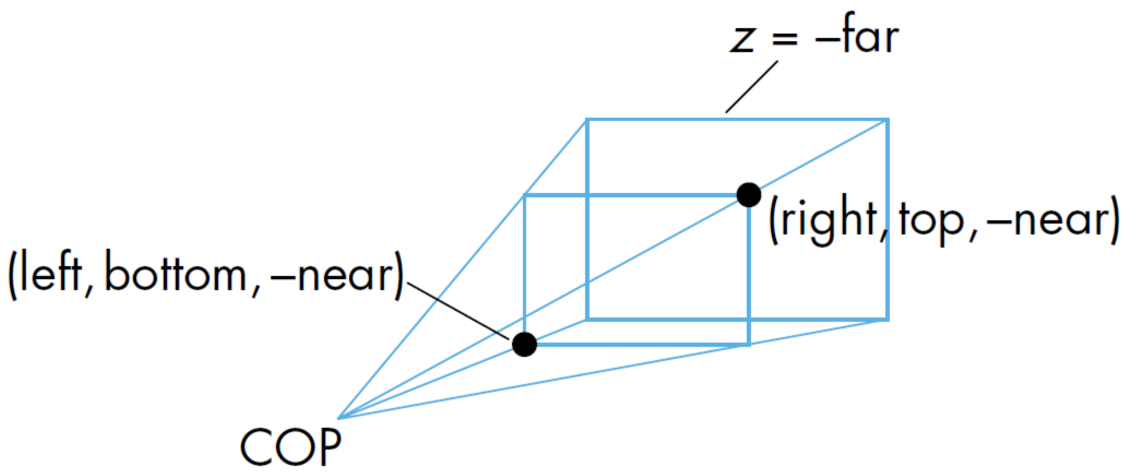
Then the new clipping volume is the canonical clipping volume

Normalization Transformation



OpenGL Perspective

glFrustum allows for an unsymmetric viewing frustum (although Perspective does not)



An unsymmetric viewing frustum can be normalized to the canonical view volume by first apply a shear and a scaling before applying N

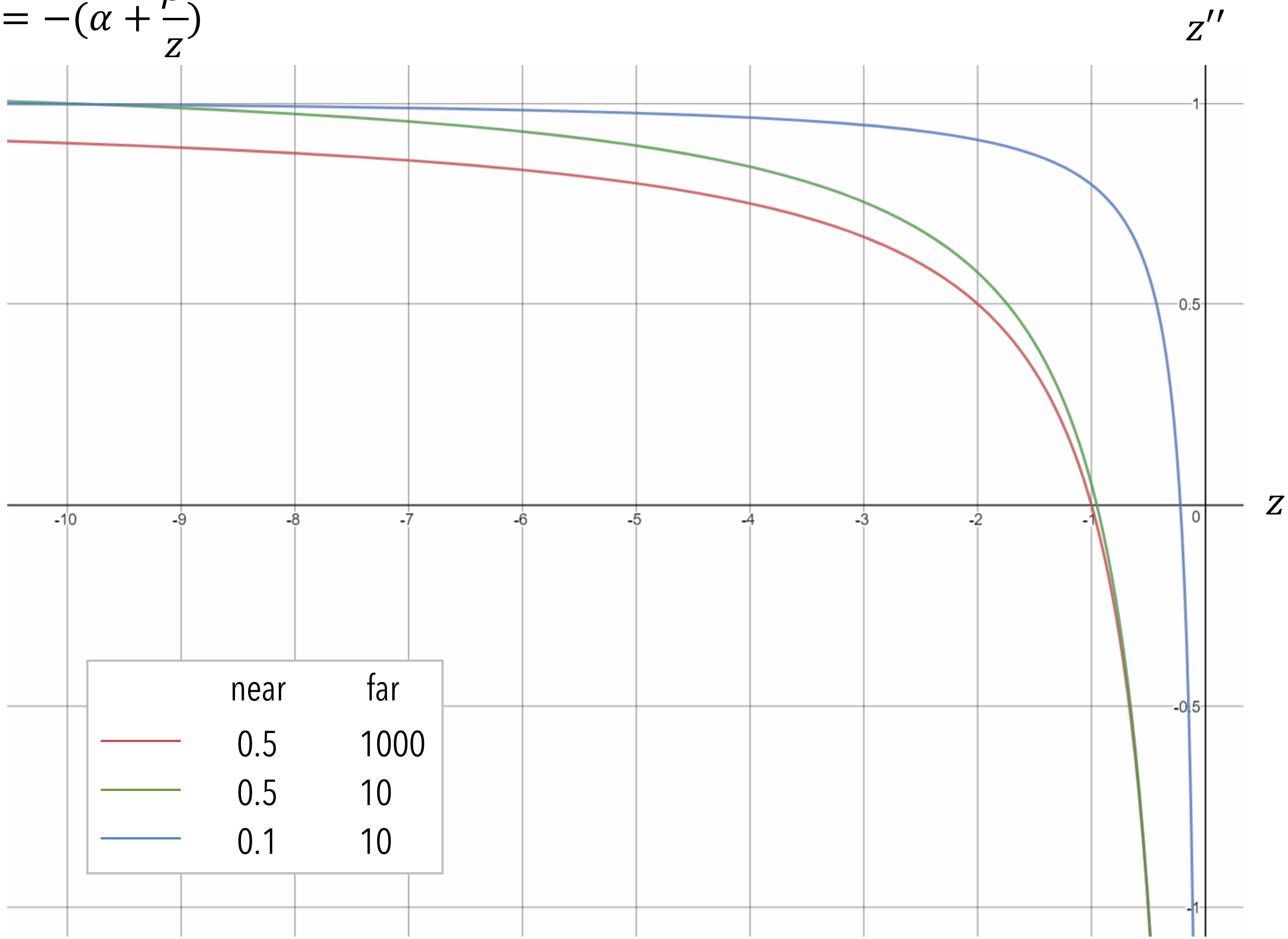
Normalization and Hidden-Surface Removal

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, note that the formula $z'' = -(\alpha+\beta/z)$ is nonlinear, which implies that the distances are distorted by the normalization which can cause numerical problems especially if the near distance is small

$$z'' = -(\alpha + \frac{\beta}{z})$$



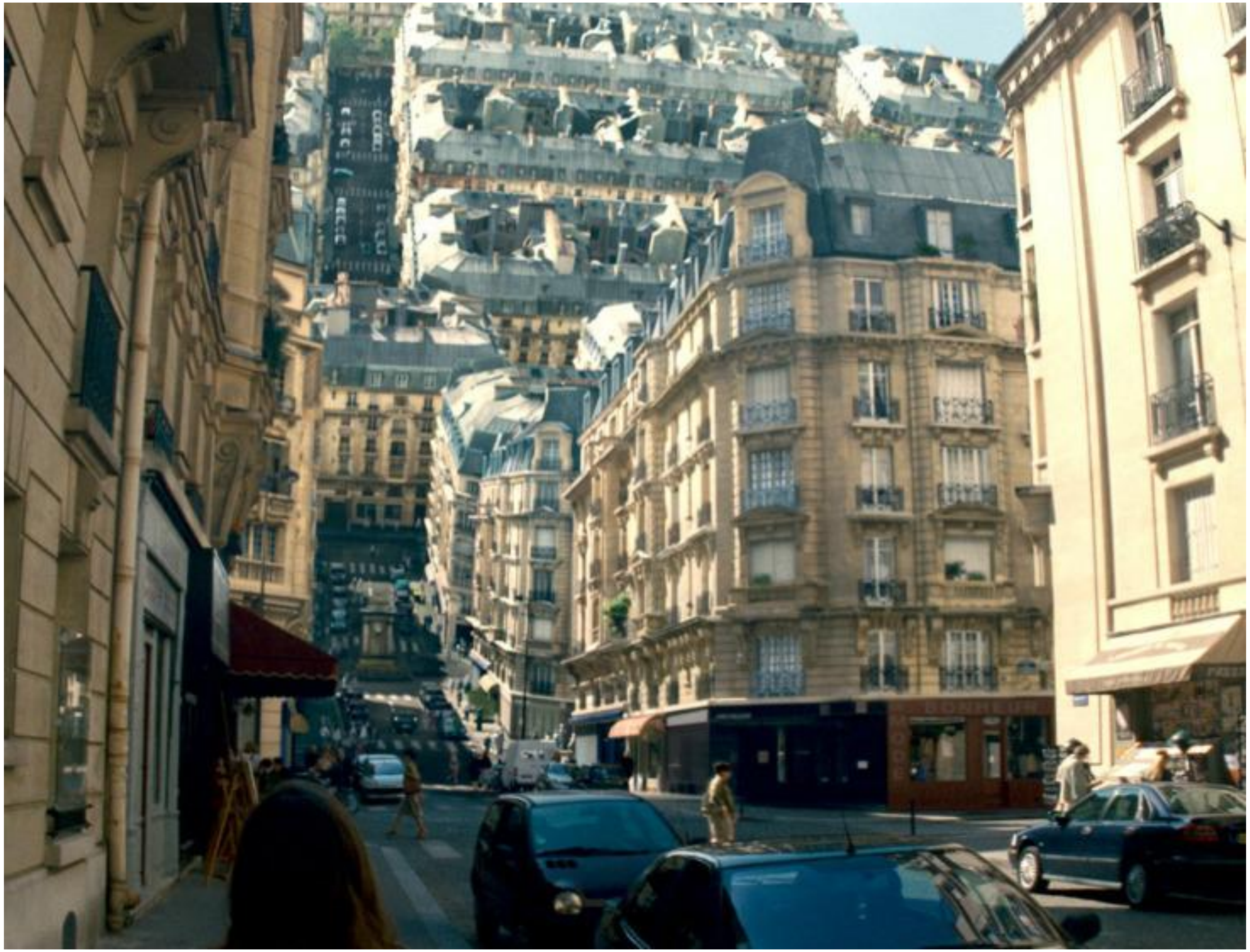
Why do we do it this way?

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

Special Projection Effects



The Movie "Inception"