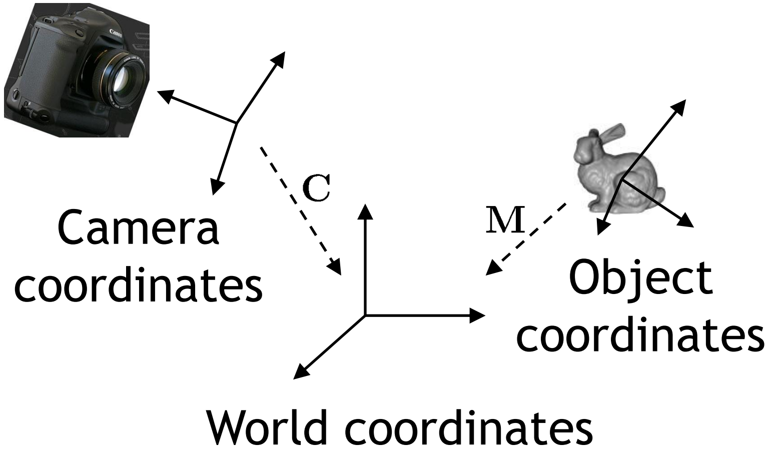
CMSC427
Transformations II:
Projection

Credit: some slides from Dr. Zwicker

Viewing transformations: the virtual camera

Need to know

- Where is the camera?
  - CAMERA TRANSFORM
- What lens does it have?
  - PROJECTIVE TRANSFORM

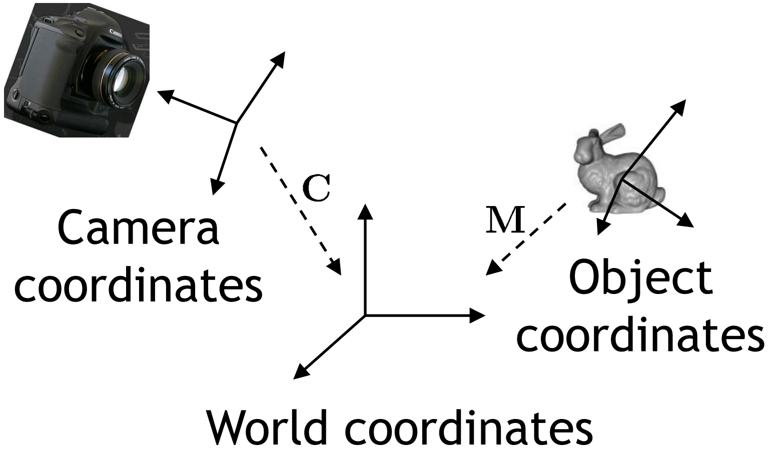


Virtual camera routines in Processing

- Camera (where)
- beginCamera()
- camera()
- endCamera()

- Projective (length of lens)
- frustum()
- ortho()
- perspective()

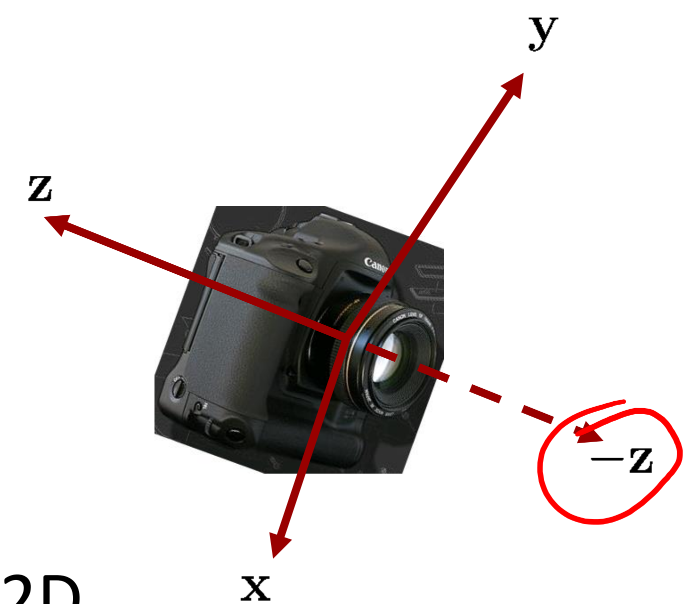
- Tracing
- printCamera()
- printProjection()



3D → 2D

Objects in camera coordinates

- We have things lined up the way we like them on screen
  - x to the right
  - y up
  - -z going into the screen
  - Objects to look at are in front of us, i.e. have negative z values
- But objects are still in 3D
- Today: how to project them into 2D



Camera coordinates

Projections

- Given 3D points (vertices) in camera coordinates, determine corresponding 2D image coordinates

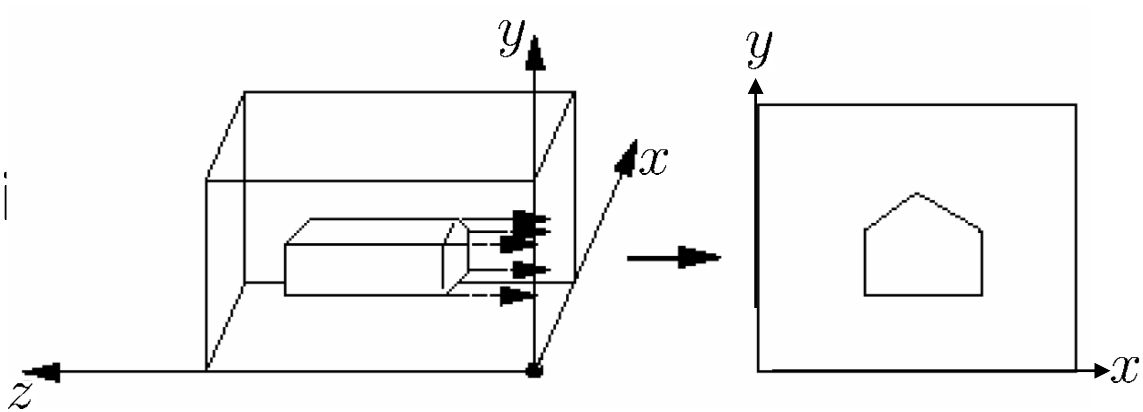
Orthographic projection

- Simply ignore z-coordinate
- Use camera space xy coordinates as image coordinates

- What we want, or not?

Orthographic projection

- Project points to *x*-*y* plane along parallel lines



- Graphi

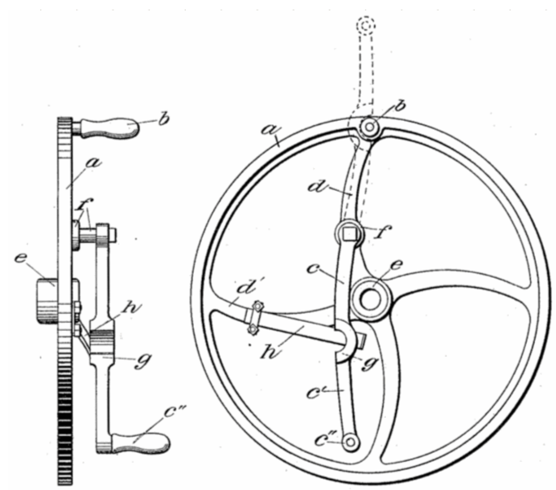
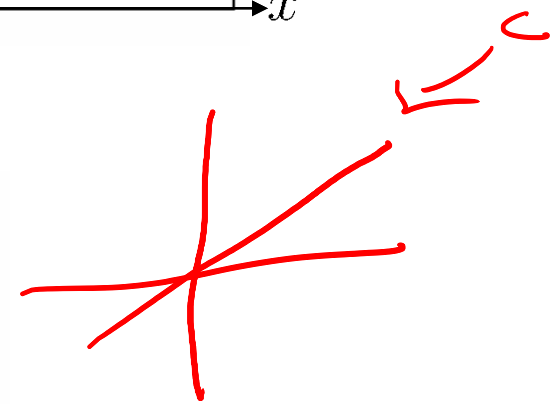


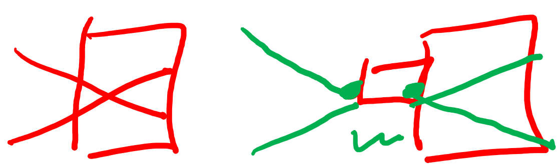
Fig. 2. *Fig. 1.*

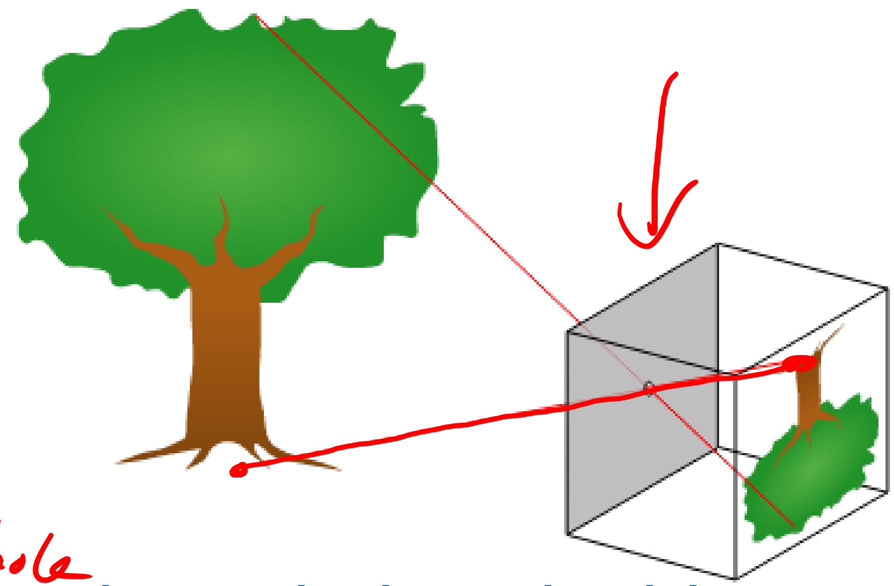


Perspective projection

- Most common for computer graphics
- Simplified model of human eye, or camera lens (pinhole camera)
- Things farther away seem smaller
- Discovery/description attributed to Filippo Brunelleschi, early 1400's

lens-modified pinhole model



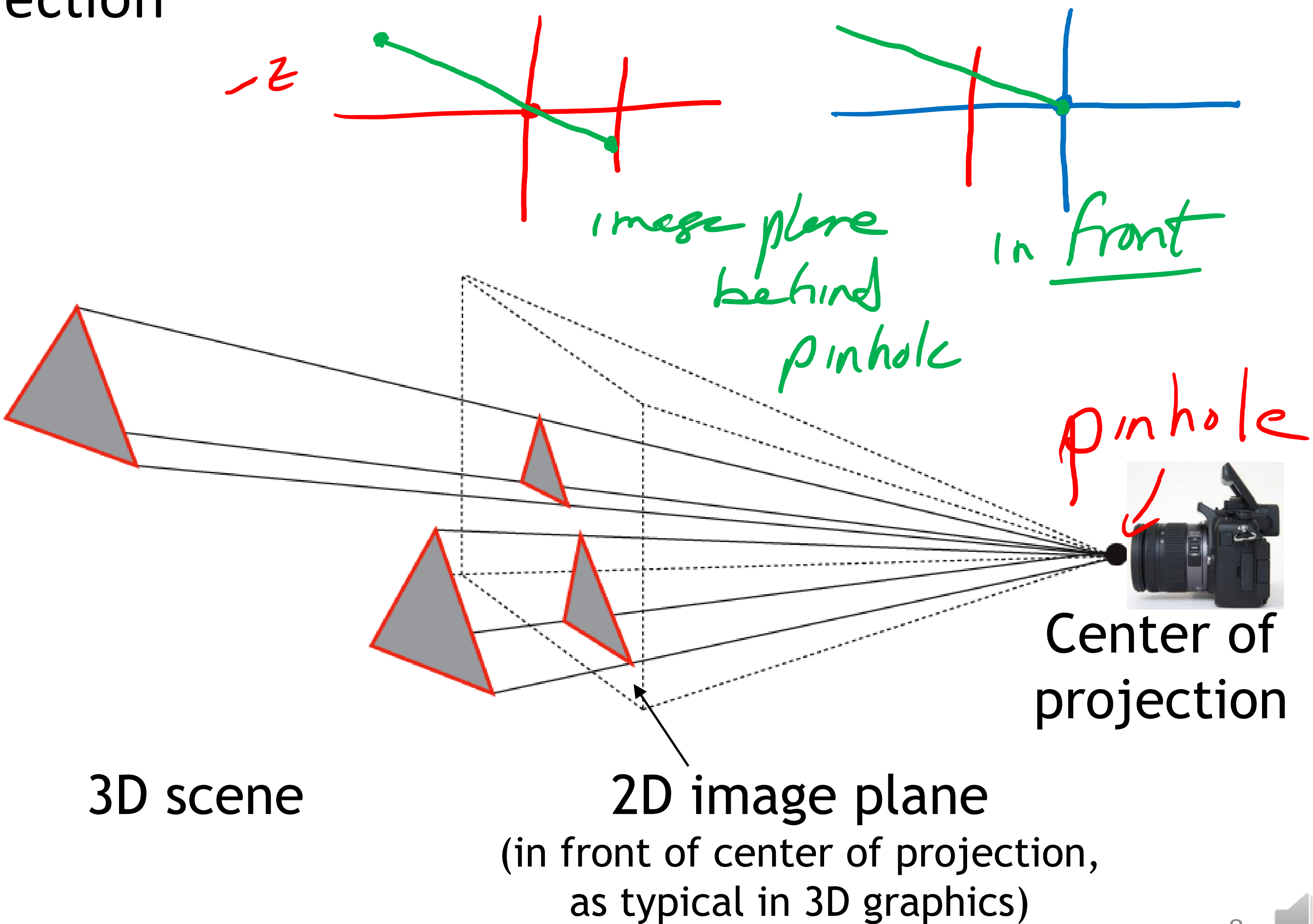


http://en.wikipedia.org/wiki/Pinhole_camera

Projection plane behind center of projection, flipped image

Perspective projection

- Project along rays that converge in center of projection



Perspective projection

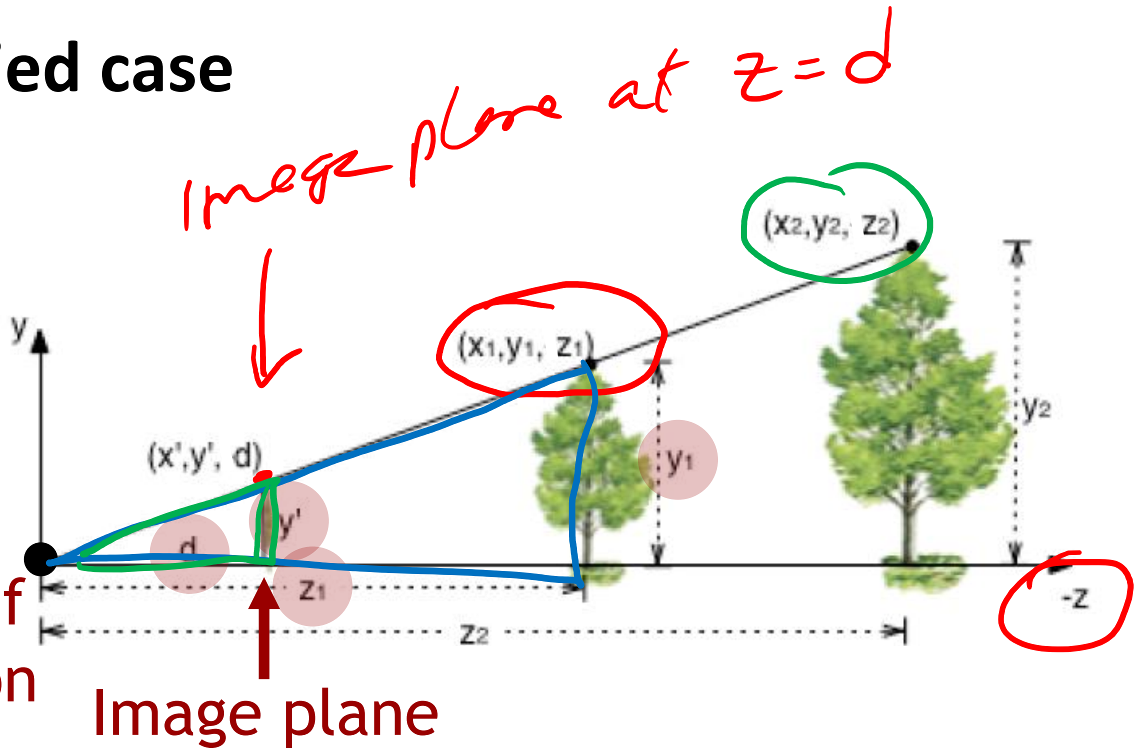
Parallel lines
no longer parallel,
converge at one point

Earliest example
La Trinitá (1427) by Masaccio
http://en.wikipedia.org/wiki/Holy_Trinity_(Masaccio)

The math: simplified case

$$y' = \frac{y_1 d}{z_1}$$

$$z' = d$$



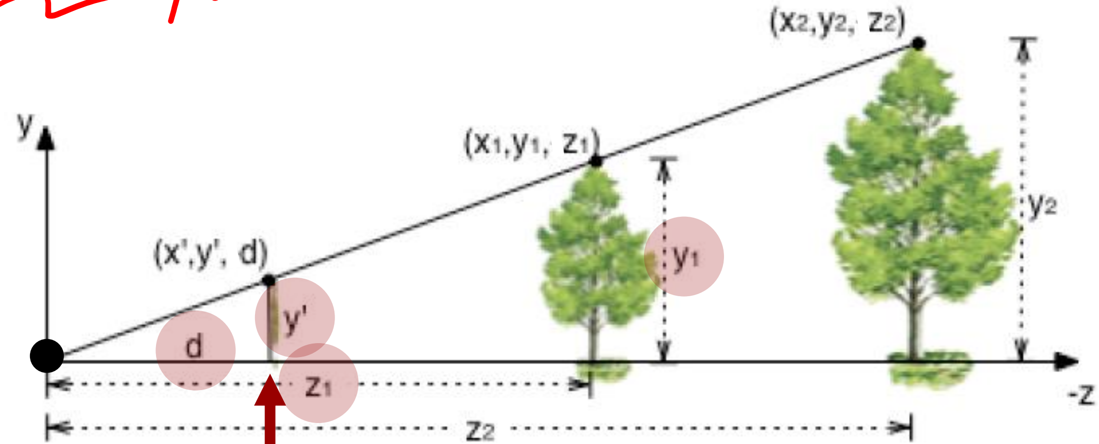
$$\frac{y'}{d} = \frac{y_1}{z_1} = \frac{y_2}{z_2}$$

The math: simplified case

focal length

$$y' = \frac{y_1 d}{z_1}$$

$$z' = d$$



Center of projection

Image plane

- Can express this using homogeneous coordinates, 4x4 matrices

wide angle ⇒ smaller d

⇒ smaller y'

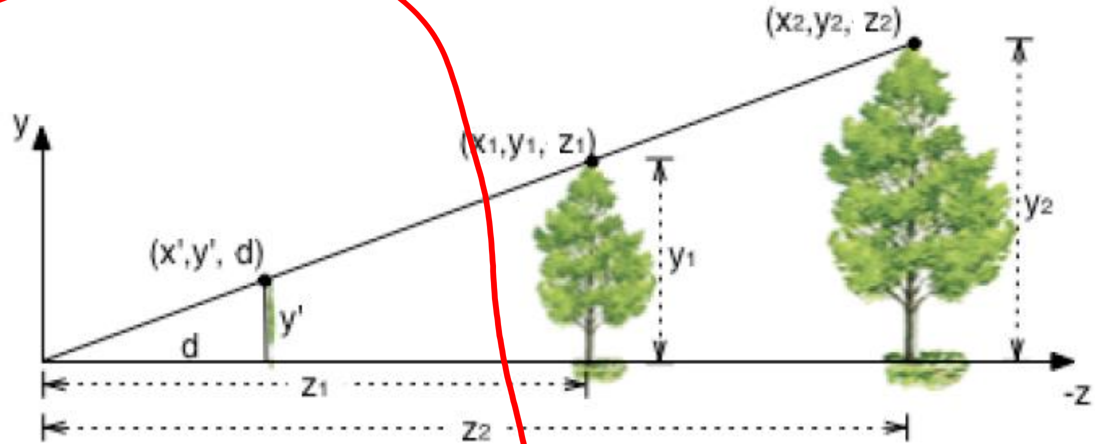
long lens ⇒ larger d

⇒ larger y'

The math: simplified case

$$y' = \frac{y_1 d}{z_1} \qquad \frac{f}{z_1}$$

$$z' = d$$



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

$$z/z/d = d$$

Projection matrix

Homogeneous coord. != 1!
Homogeneous division

Perspective projection

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \\ z/d \end{bmatrix} \rightarrow \begin{bmatrix} xd/z \\ yd/z \\ d \\ 1 \end{bmatrix}$$

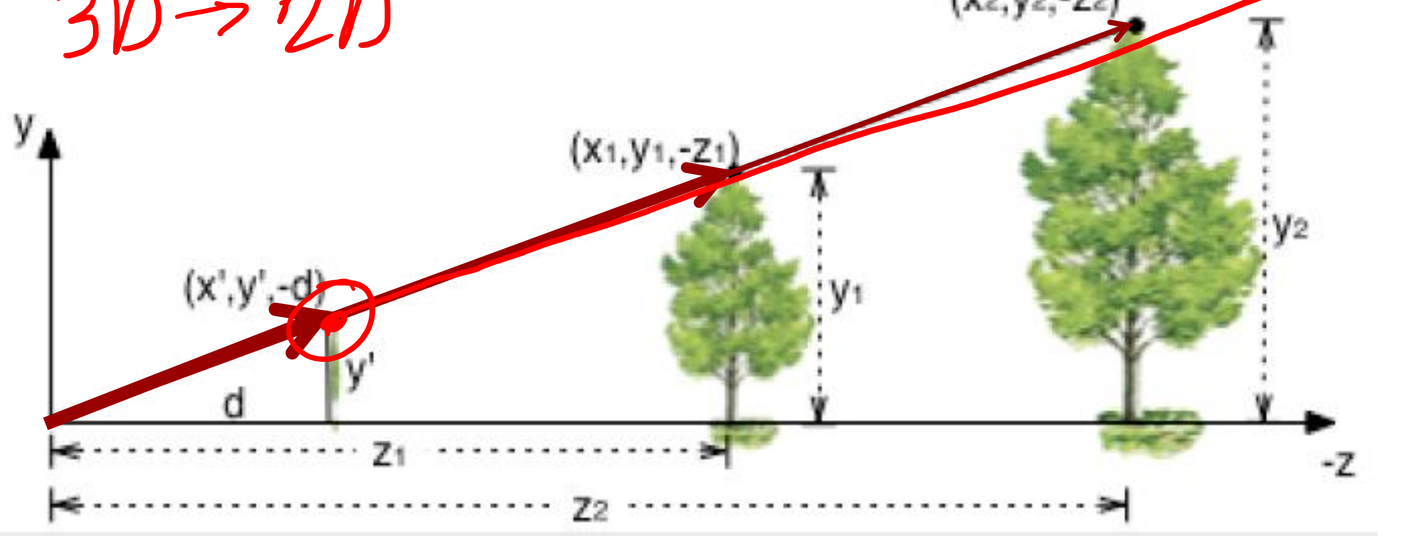
Projection matrix **Homogeneous division**

- Using projection matrix and homogeneous division seems more complicated than just multiplying all coordinates by *d/z*, so why do it?
- Will allow us to
  - handle different types of projections in a unified way
  - define arbitrary view volumes

Intuitive example

- All points that lie on one projection line (i.e., a "line-of-sight", intersecting with center of projection of camera) are projected onto same image point
- All 3D points on one projection line are equivalent
- Projection lines form 2D projective space, or 2D projective plane

3D → 2D



3D Projective space

- Projective space $\mathbf{P}^3$ represented using $\mathbf{R}^4$ and homogeneous coordinates
  - Each point along 4D ray is equivalent to same 3D point at w=1

$$\begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} \sim \begin{bmatrix} \lambda x \\ \lambda y \\ \lambda z \\ \lambda w \end{bmatrix} \sim \begin{bmatrix} x/w \\ y/w \\ z/w \\ 1 \end{bmatrix}$$

„equivalent“

w=1

embedded space

1D vector subspace, arbitrary scalar value λ

Equivalent element, for any λ

3D Projective space

- Projective mapping (transformation): any non-singular linear mapping on homogeneous coordinates, for example,

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \\ z/d \end{bmatrix} \sim \begin{bmatrix} xd/z \\ yd/z \\ \cancel{d} \\ 1 \end{bmatrix}$$

use it for depth mapping

- Generalization of affine mappings
  - 4th row of matrix is arbitrary (not restricted to [0 0 0 1])
- Projective mappings are collineations
  http://en.wikipedia.org/wiki/Projective_linear_transformation
  http://en.wikipedia.org/wiki/Collineation
  - Preserve straight lines, but not parallel lines
- Much more theory
  http://www.math.toronto.edu/mathnet/questionCorner/projective.html
  http://en.wikipedia.org/wiki/Projective_space

Projective space

Projective space

http://en.wikipedia.org/wiki/Projective_space

- [xyzw] homogeneous coordinates
- includes points at infinity (w=0)
- projective mappings (perspective projection)

Vector space

- [xyz] coordinates
- represents vectors
- linear mappings
  (rotation around origin,
  scaling, shear)

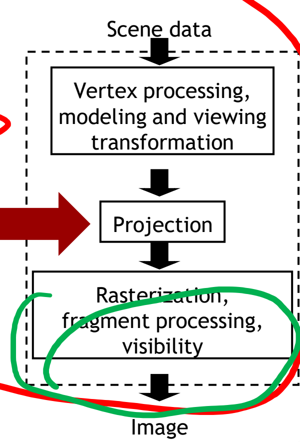
Affine space

- [xyz1], [xyz0]
  homogeneous coords.
- distinguishes points
  and vectors
- affine mappings
  (translation)

In practice

- Use 4x4 homogeneous matrices like other 4x4 matrices
- Modeling & viewing transformations are affine mappings
  - points keep $w=1$
  - no need to divide by w when doing modeling operations or transforming into camera space
- 3D-to-2D projection is a projective transform
  - Resulting w coordinate not always 1
- Divide by w (perspective division, homogeneous division) after multiplying with projection matrix
  - OpenGL rendering pipeline (graphics hardware) does this automatically

GPU
Matrix multiply step
Division step $\div w$



Today

- Rendering pipeline
- Projections
- View volumes, clipping
- Viewport transformation