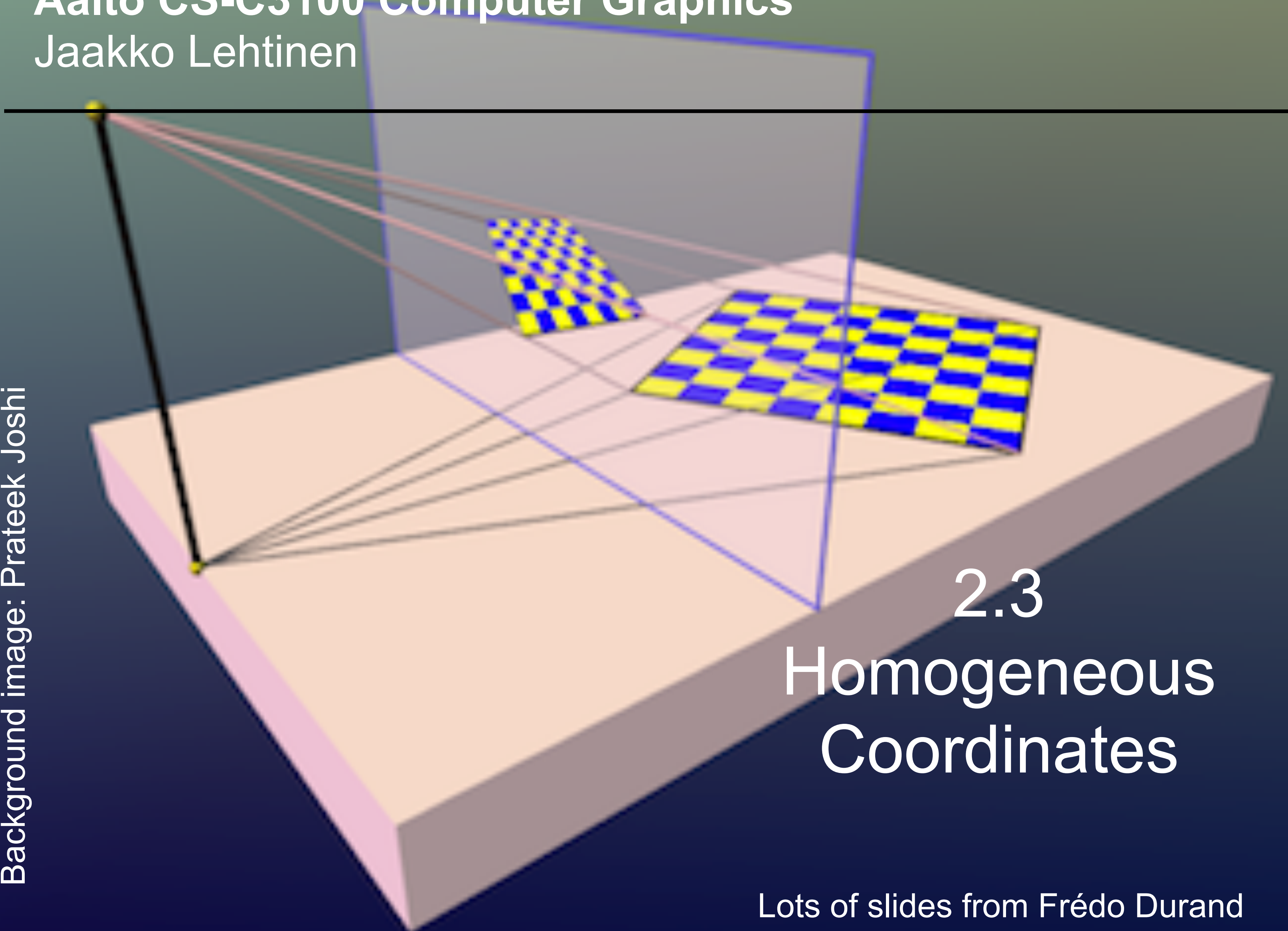
**Aalto CS-C3100 Computer Graphics**

Jaakko Lehtinen



Background image: Prateek Joshi

# 2.3 Homogeneous Coordinates

Lots of slides from Frédo Durand

# In This Video

- Properties of homogeneous coordinates
- Perspective transformations

# Recap: The Trick

This is what we want:

$$x' = ax + by + c$$
$$y' = dx + ey + f$$

**Affine formulation**

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b \\ d & e \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} c \\ f \end{bmatrix}$$

$$p' = Mp + t$$

**Homogeneous formulation**

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$p' = Mp$$

# Homogeneous Coordinates

- Add an extra dimension
  - in 2D, we use 3-vectors and 3 x 3 matrices
  - In 3D, we use 4-vectors and 4 x 4 matrices
  - Makes affine transformations linear in one higher dimension (can be represented by a matrix)
- Each point has an extra value, *w*
  - You can think of it as "scale"
  - For all transformations except perspective, you can just set *w*=1 and not worry about it

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

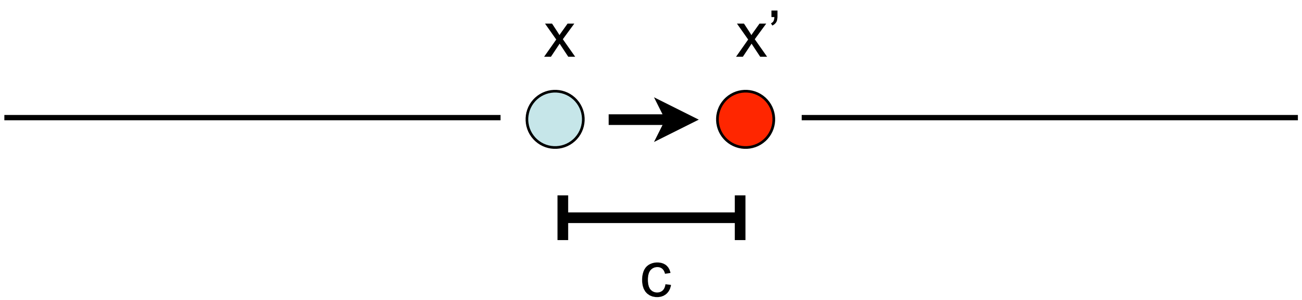
# Homogeneous Coordinates

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

- If we multiply a homogeneous point by an *affine matrix*, $w$ is unchanged
  - "Affine matrix" means the last row is (0 0 0 1)
  - This form encodes all possible affine transformations!
  - What happens when you multiply an affine matrix with another affine matrix?

# What’s Actually Happening?

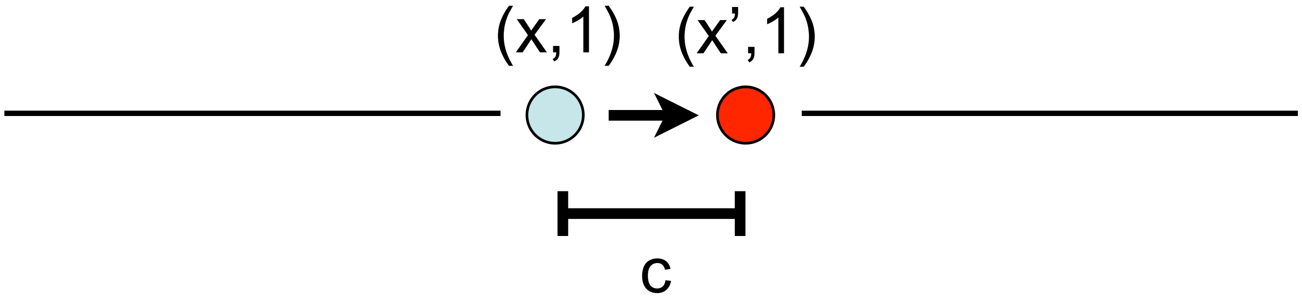
$$x' = x + c$$



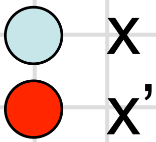
# What's Actually Happening?

$$x' = x + c$$

Let's first add the 2nd dimension w and set coordinates to 1. What does this mean..?

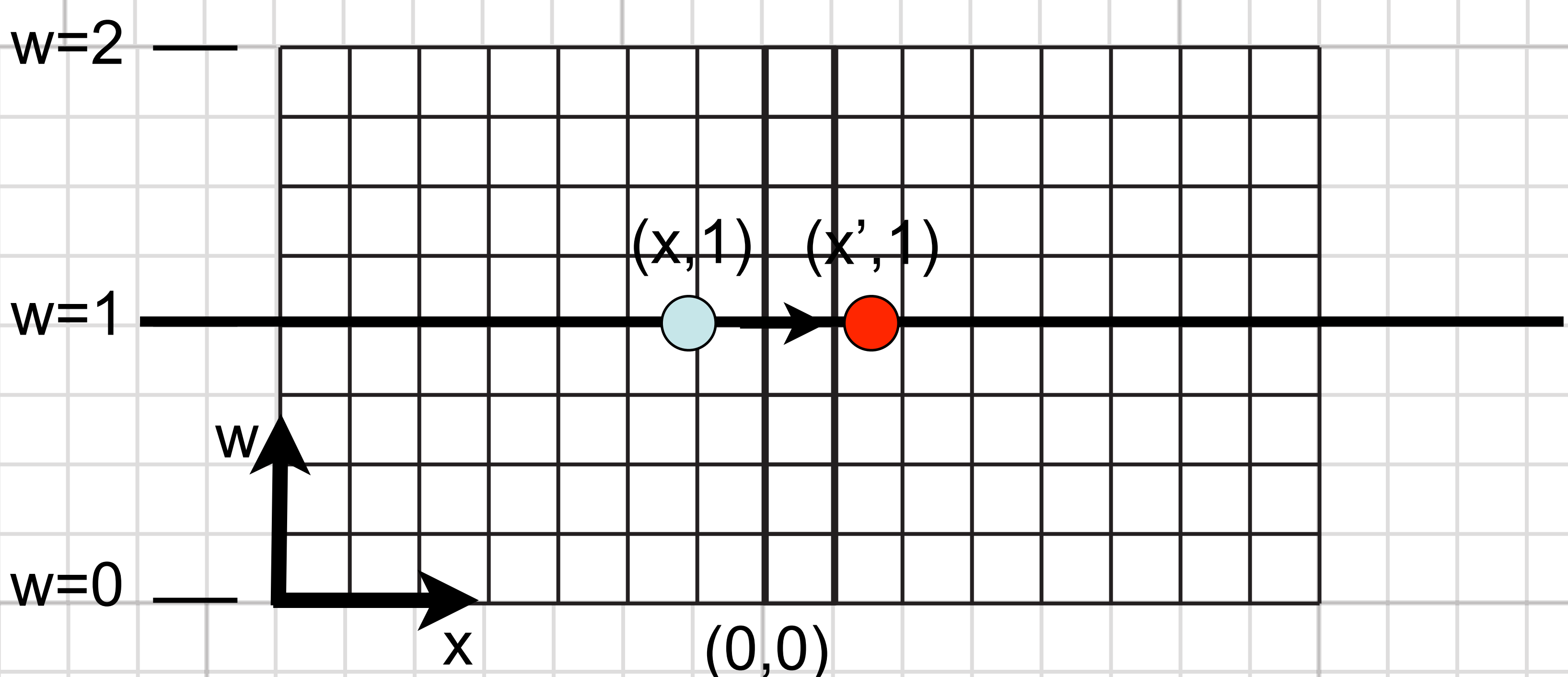


# What's Actually Happening?




$$x' = x + c$$

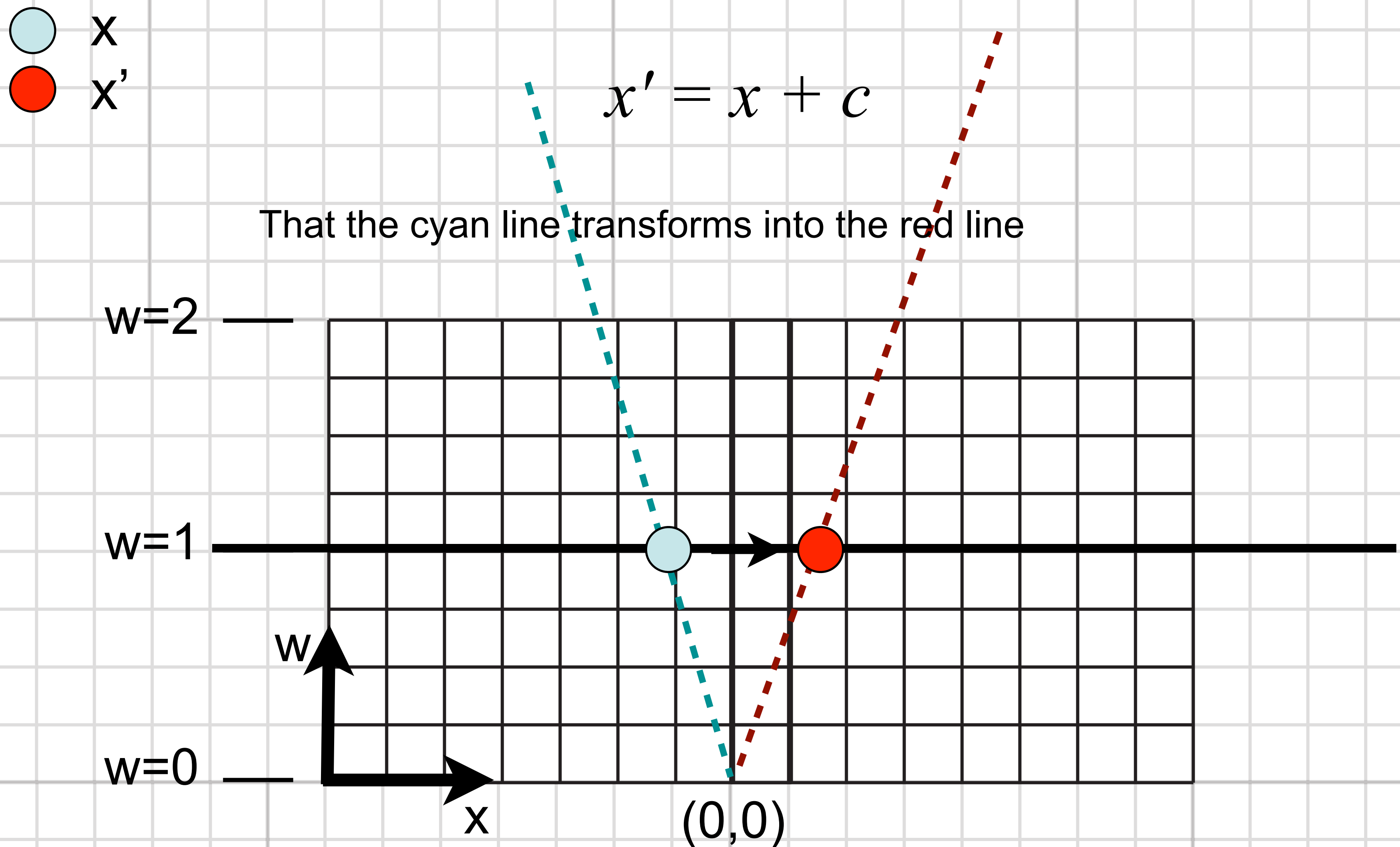
Translating *x* into *x*' means...



# What’s Actually Happening?

$$x' = x + c$$

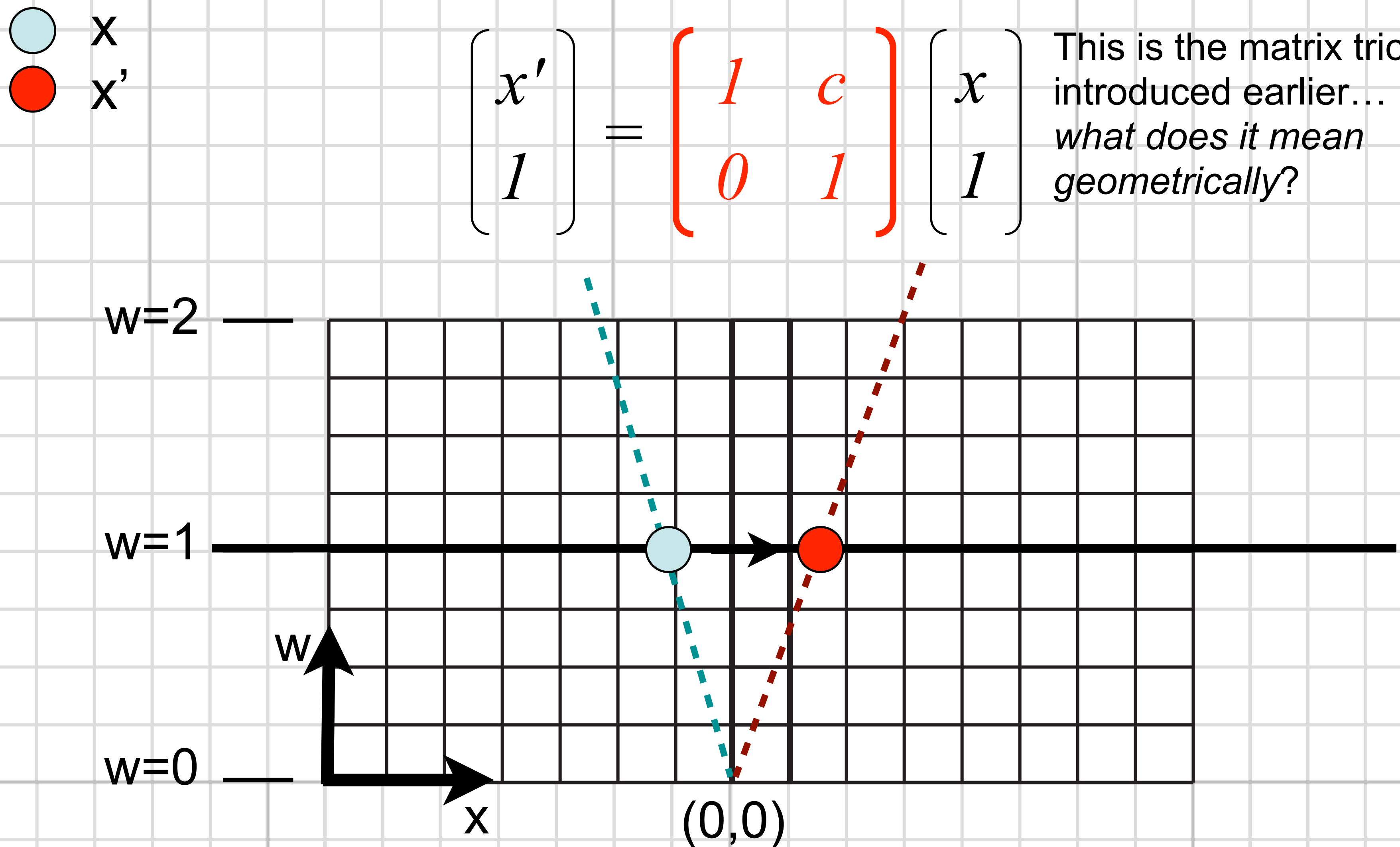
That the cyan line transforms into the red line



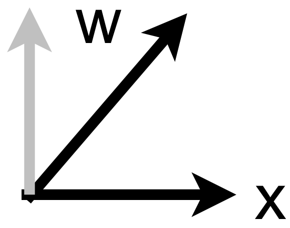
# What's Actually Happening?

$$\begin{pmatrix} x' \\ 1 \end{pmatrix} = \begin{bmatrix} 1 & c \\ 0 & 1 \end{bmatrix} \begin{pmatrix} x \\ 1 \end{pmatrix}$$

This is the matrix trick we introduced earlier… *what does it mean geometrically?*

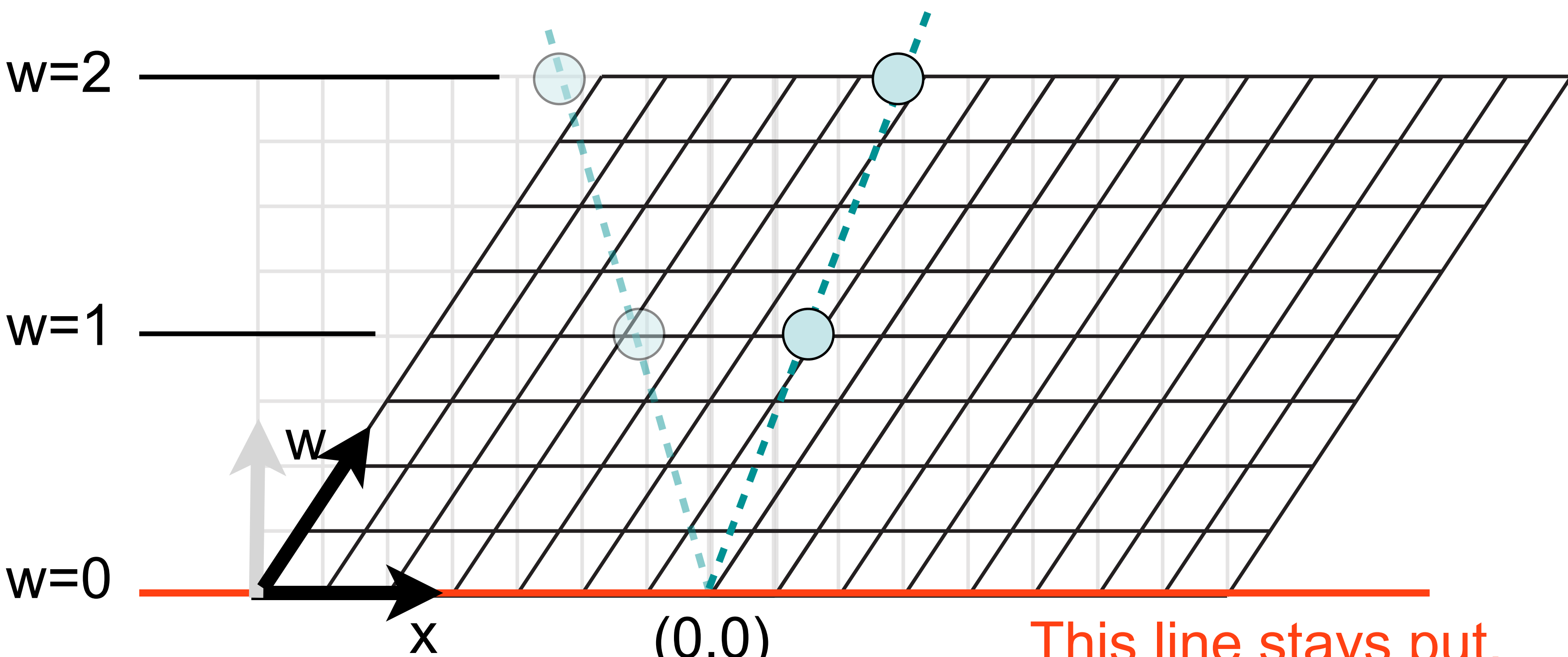


# What's Actually Happening?




$$\begin{bmatrix} x' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & c \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ 1 \end{bmatrix}$$

It's a 2D shear!




This line stays put, and origin stays still

# Summary So Far

Important!

- Affine n-D transformations are encoded by linear transformations in (n+1)D that move corresponding homogeneous points **in the *w*=1 (hyper)plane**
  - Technically, *w*=any constant works too, just have to scale translation distances as well.

# More on *w*

- You can think of it as “scale”.
- Let’s see:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$? = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2x \\ 2y \\ 2 \end{bmatrix}$$

# More on *w*

- You can think of it as “scale”.
- Let’s see:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

(DUH!)

$$\begin{bmatrix} 2x' \\ 2y' \\ 2 \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2x \\ 2y \\ 2 \end{bmatrix}$$

# More on *w*

- You can think of it as “scale”.
- Let’s see:

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

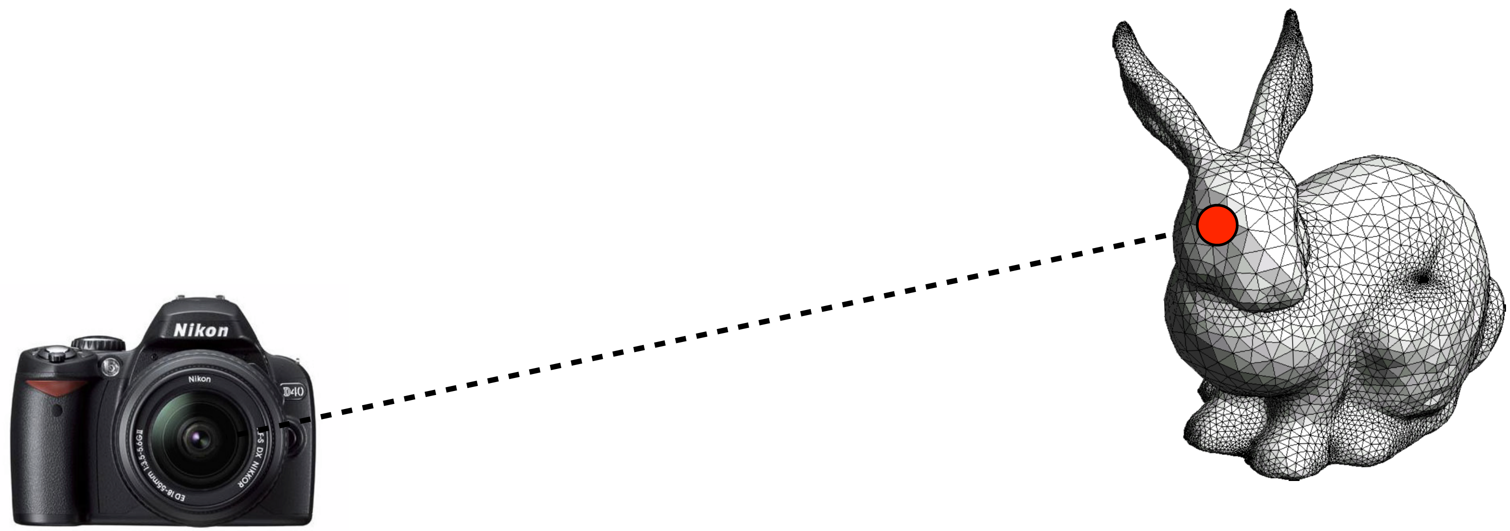
(DUH!)

$$\begin{pmatrix} 2x' \\ 2y' \\ 2 \end{pmatrix} = \begin{pmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 2x \\ 2y \\ 2 \end{pmatrix}$$

It’s tempting to divide by *w*, because this will get us back to the result above.

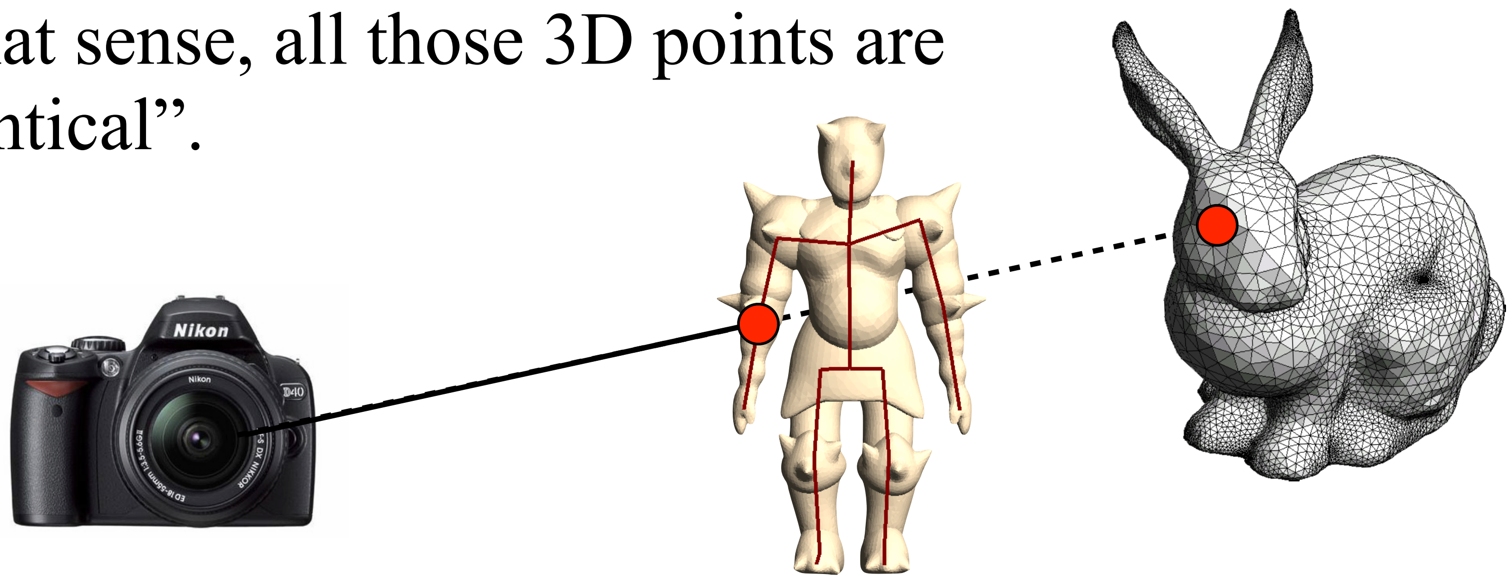
# “Projective Equivalence” – Why?

- For affine transformations, adding $w$=1 in the end proved to be convenient.
- The real showpiece is **perspective.**

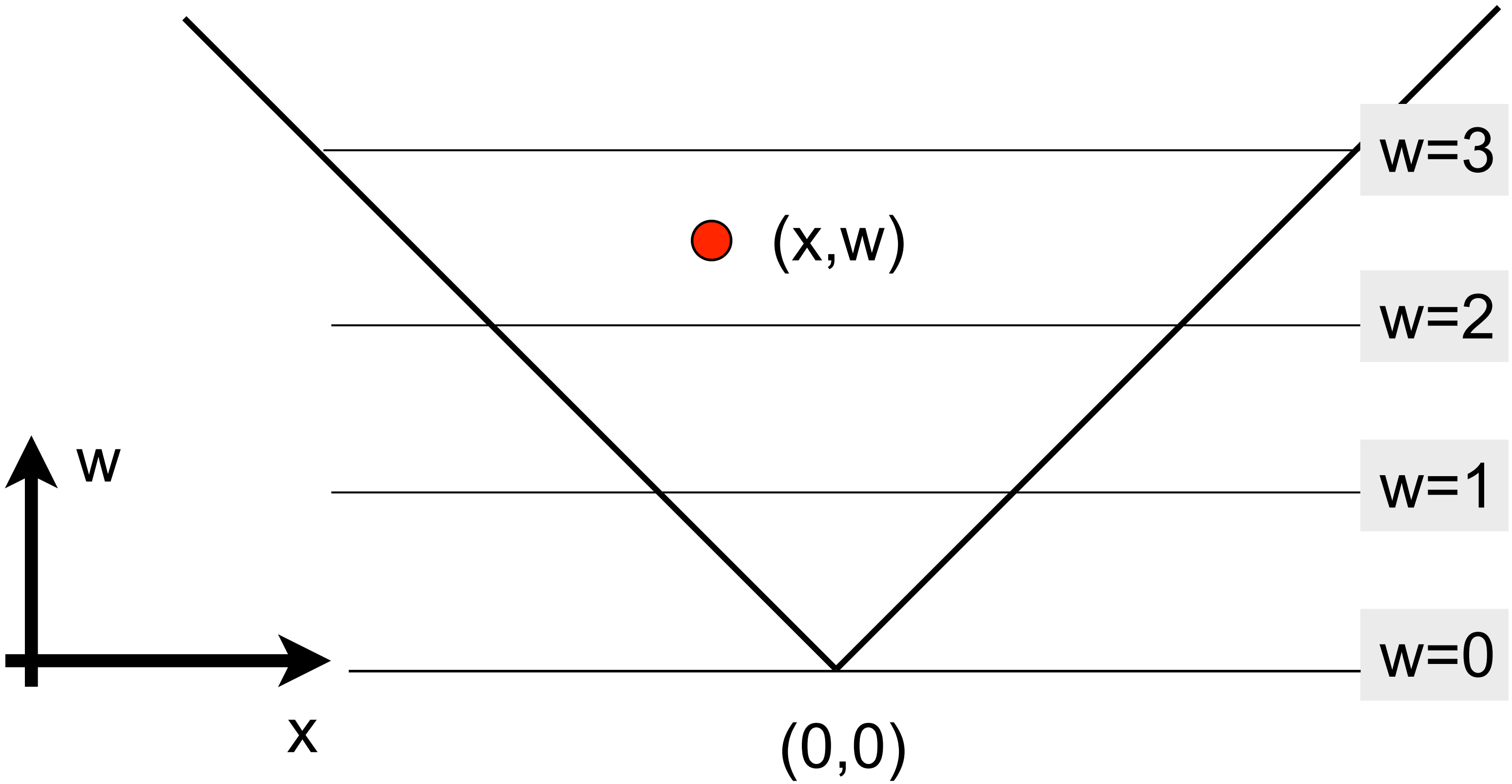


# "Projective Equivalence" – Why?

- For affine transformations, adding $w$=1 in the end proved to be convenient.
- The real showpiece is **perspective.**
  - From the camera's point of view, all 3D points on the line fall on the same 2D coordinate in the image.
  - In that sense, all those 3D points are "identical".

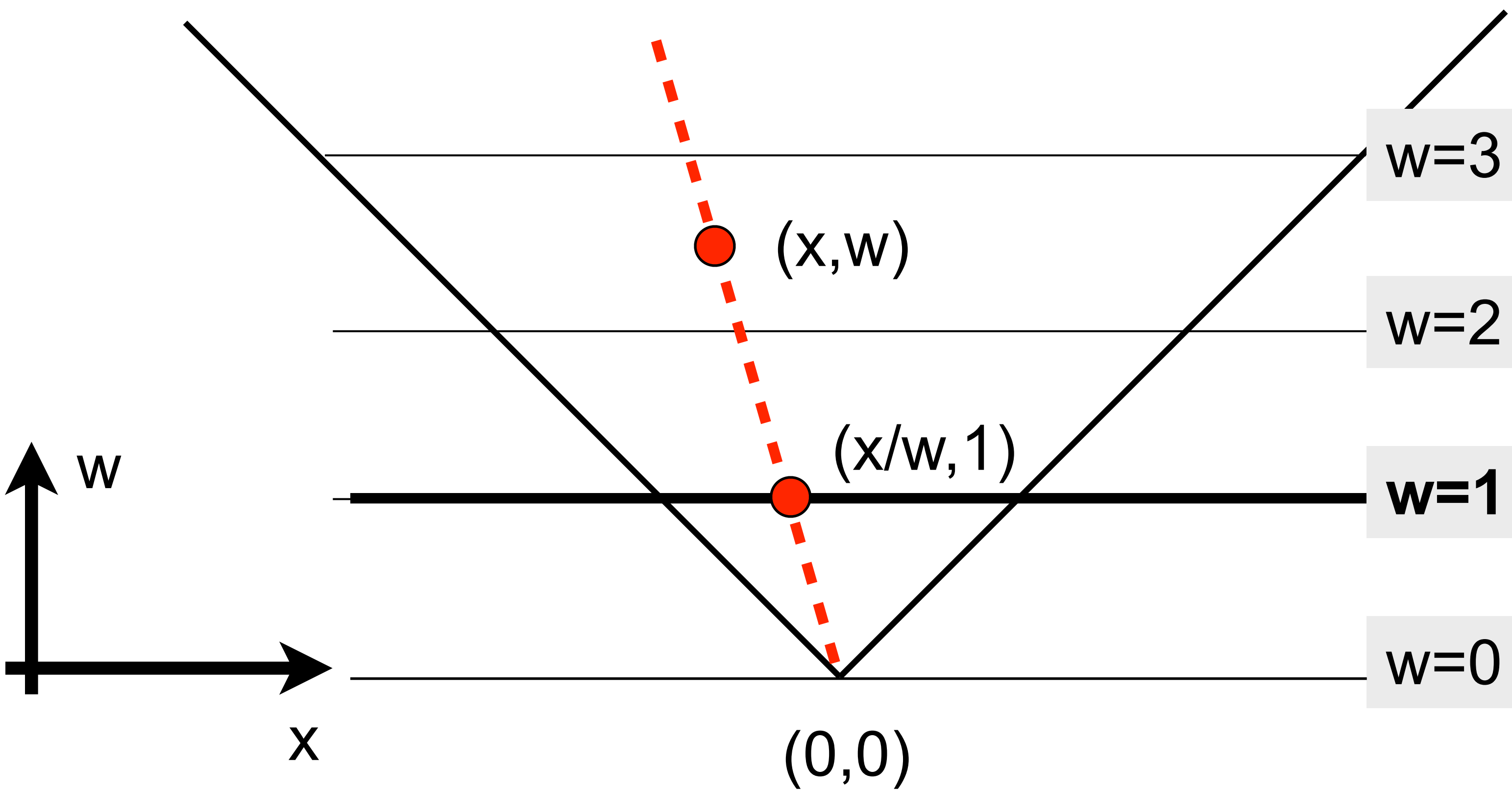


# “Projective Equivalence” in 1D



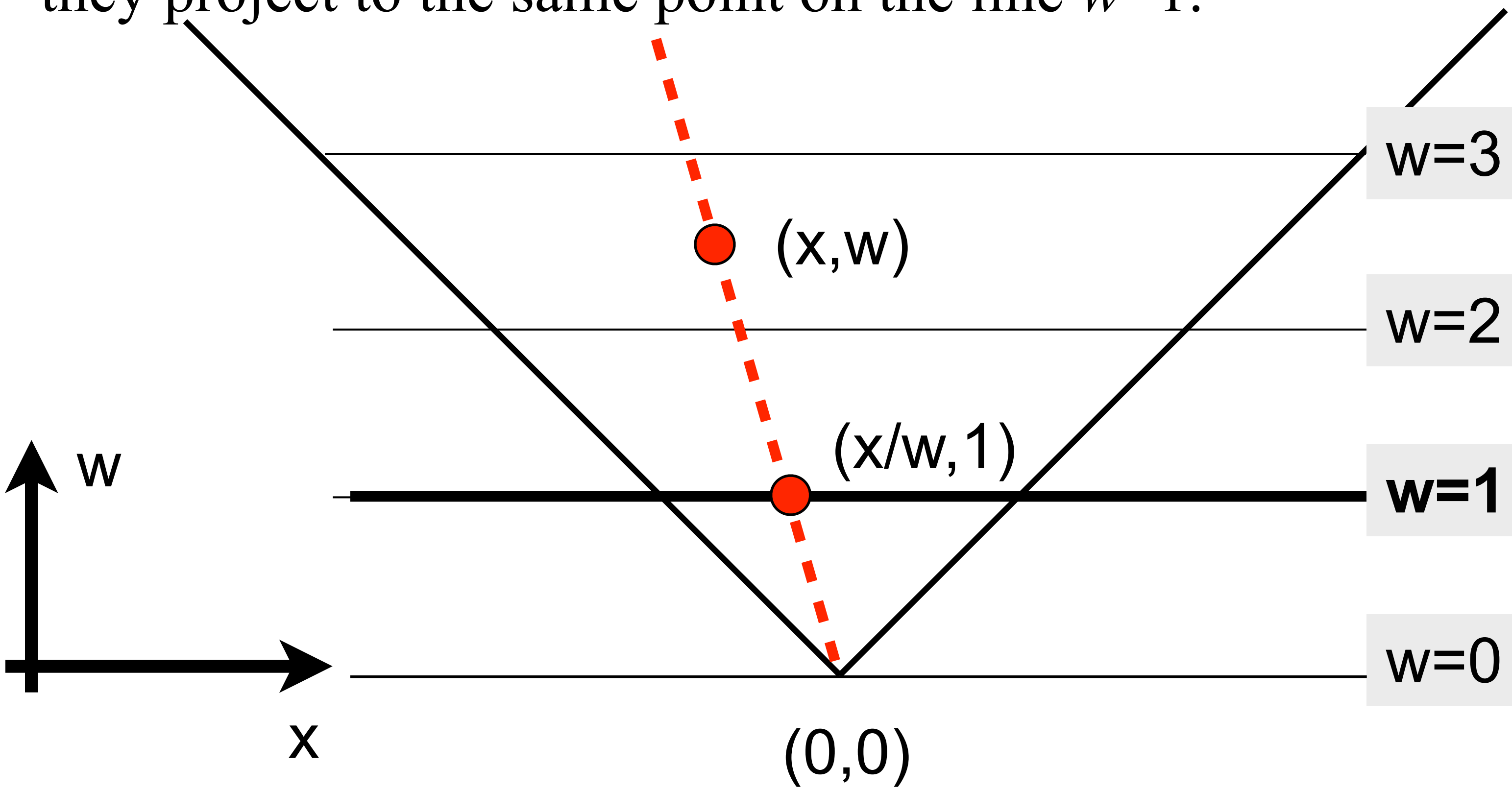
# Projective Equivalence in 1D

- We can get into a "canonical form" by dividing by $w$. This projects $(x, w)$ onto the line $w$=1 yielding $(x/w, 1)$.
- Similar triangles!



# Projective Equivalence in 1D

- We can get into a "canonical form" by dividing by *w*. This projects ($x$, $w$) onto the line $w$=1 yielding ($x/w$, 1).
- We say that all points on dashed line are **identical** because they project to the same point on the line $w$=1.



# Projective Equivalence

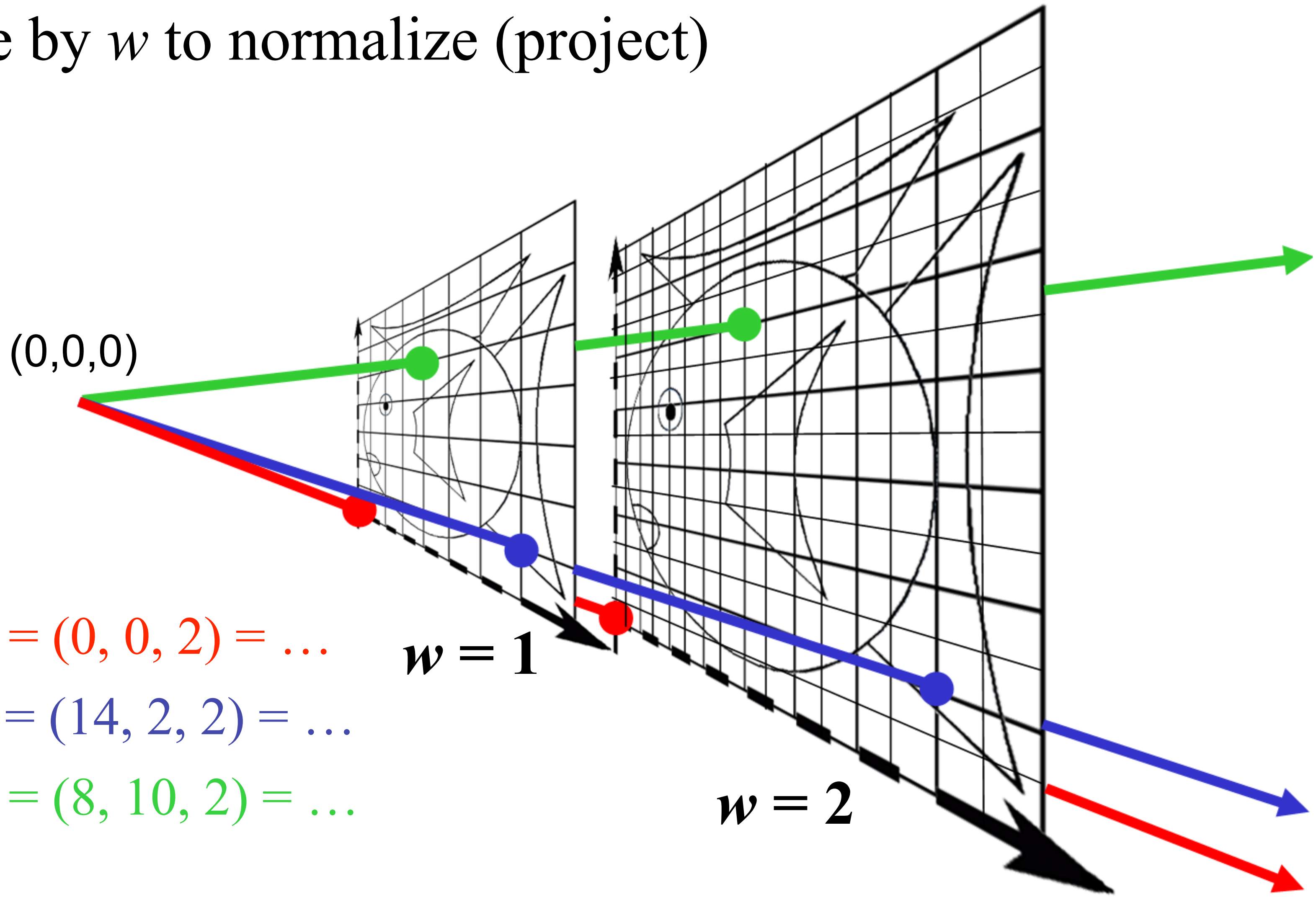
- More mathematically, all non-zero scalar multiples of a point are considered identical
  - So, in fact, we are saying that points in ND are identified with **lines through the origin** in (N+1)D

$$\begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = \begin{bmatrix} ax \\ ay \\ az \\ aw \end{bmatrix} \overset{w\,!=0}{=} \begin{bmatrix} x/w \\ y/w \\ z/w \\ 1 \end{bmatrix}$$

$$a \;\; != 0$$

# Homogeneous Visualization in 2D

- Divide by $w$ to normalize (project)



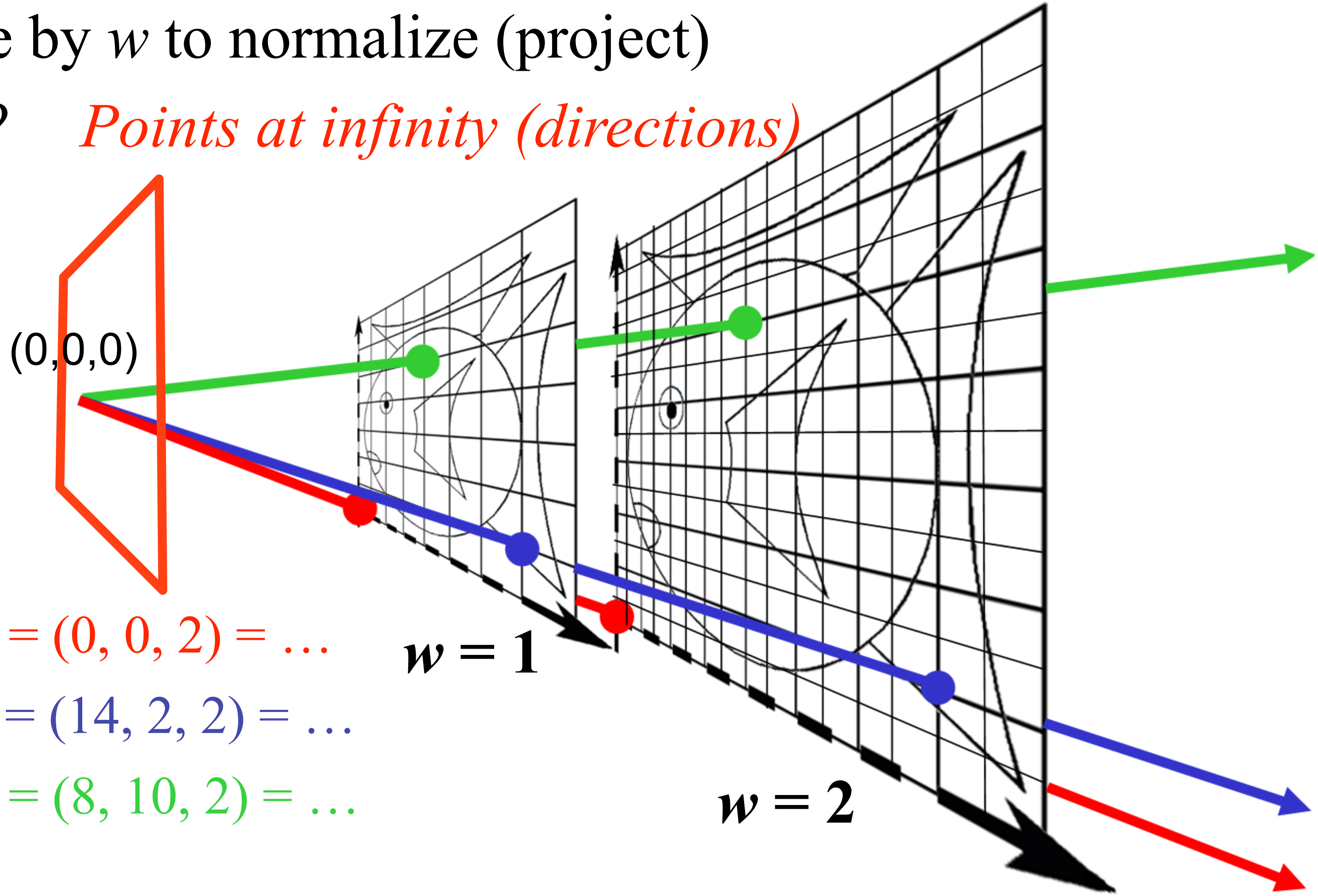

$(0, 0, 1) = (0, 0, 2) = \ldots$

$(7, 1, 1) = (14, 2, 2) = \ldots$

$(4, 5, 1) = (8, 10, 2) = \ldots$

# Homogeneous Visualization in 2D

- Divide by $w$ to normalize (project)
- $w = 0$? *Points at infinity (directions)*




$(0, 0, 1) = (0, 0, 2) = \ldots$

$(7, 1, 1) = (14, 2, 2) = \ldots$

$(4, 5, 1) = (8, 10, 2) = \ldots$

# A Word of Warning

- In "regular" 3D, adding a displacement $d$ to a point $p$ is simple

$$p' = p + d = \begin{pmatrix} p_x + d_x \\ p_y + d_y \\ p_z + d_z \end{pmatrix}$$

- What about homogeneous coordinates?

$$p' = \begin{pmatrix} p_x \\ p_y \\ p_z \\ 1 \end{pmatrix} + \begin{pmatrix} d_x \\ d_y \\ d_z \\ ? \end{pmatrix}$$

# A Word of Warning

- In "regular" 3D, adding a displacement $d$ to a point $p$ is simple

$$p' = p + d = \begin{pmatrix} p_x + d_x \\ p_y + d_y \\ p_z + d_z \end{pmatrix}$$

- What about homogeneous coordinates?

$$p' = \begin{pmatrix} p_x \\ p_y \\ p_z \\ 1 \end{pmatrix} + \begin{pmatrix} d_x \\ d_y \\ d_z \\ 0 \end{pmatrix} = \begin{pmatrix} p_x + d_x \\ p_y + d_y \\ p_z + d_z \\ 1 \end{pmatrix}$$

**You can't add homogeneous points to each other like in 3D.**

# A Word of Warning

$$\begin{pmatrix} p_x' \\ p_y' \\ p_z' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & d_x \\ 0 & 1 & 0 & d_y \\ 0 & 0 & 1 & d_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} p_x \\ p_y \\ p_z \\ 1 \end{pmatrix}$$

**w stays the same!**

- What about homogeneous coordinates?

$$p' = \begin{pmatrix} p_x \\ p_y \\ p_z \\ 1 \end{pmatrix} + \begin{pmatrix} d_x \\ d_y \\ d_z \\ 0 \end{pmatrix} = \begin{pmatrix} p_x + d_x \\ p_y + d_y \\ p_z + d_z \\ 1 \end{pmatrix}$$

**You can't add homogeneous points to each other like in 3D.**

# The Same for Scaling

$$a\,p = \begin{pmatrix} a\,p_x \\ a\,p_y \\ a\,p_z \\ a \end{pmatrix}$$

Remember projective equivalence

# The Same for Scaling

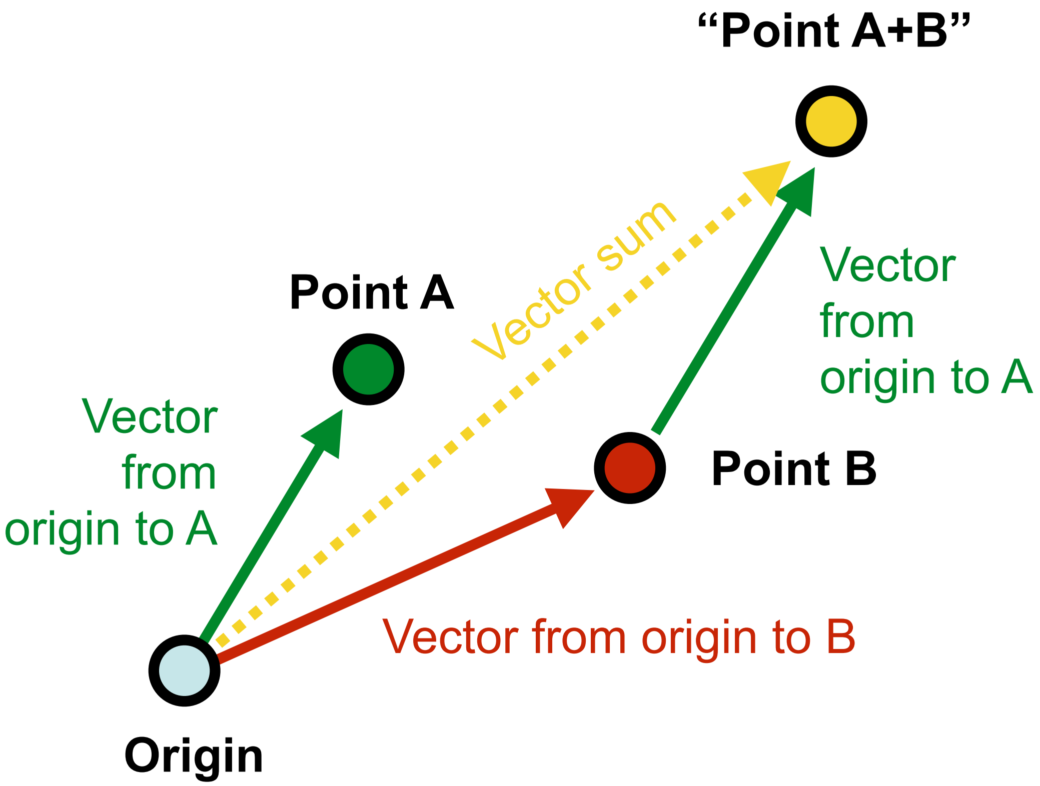
$$a\,p = \begin{pmatrix} a\,p_x \\ a\,p_y \\ a\,p_z \\ 1 \end{pmatrix}$$

- You also leave *w* fixed.

# Points vs. Vectors

- This all makes sense if you think as follows..
  - A *point* is a point (has nonzero *w*)
  - A *vector* is an arrow with direction and length, attached to a point (has zero *w*)
  - In Euclidean spaces, you the two are equivalent
    - “A point <=> a vector pointing to it from the origin”
    - *But projective space is not Euclidean*
  - The last point is needed to make sense of an expression “adding two points”. Really you’re adding two vectors together, or, equivalently, adding a vector to a point. *You don’t add two points*.

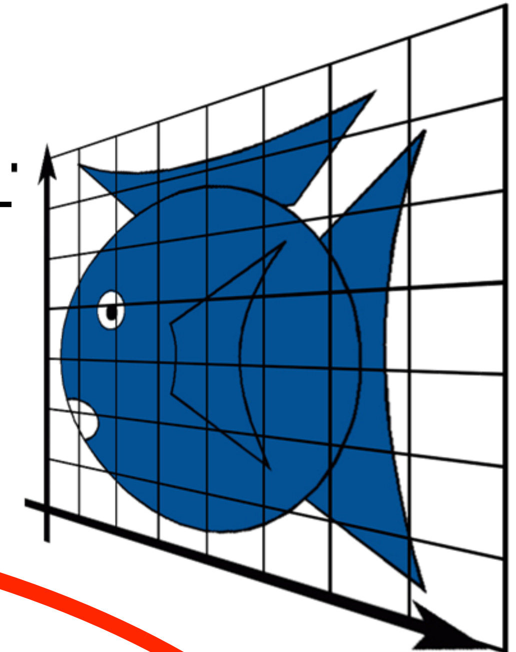
# (Visually…)

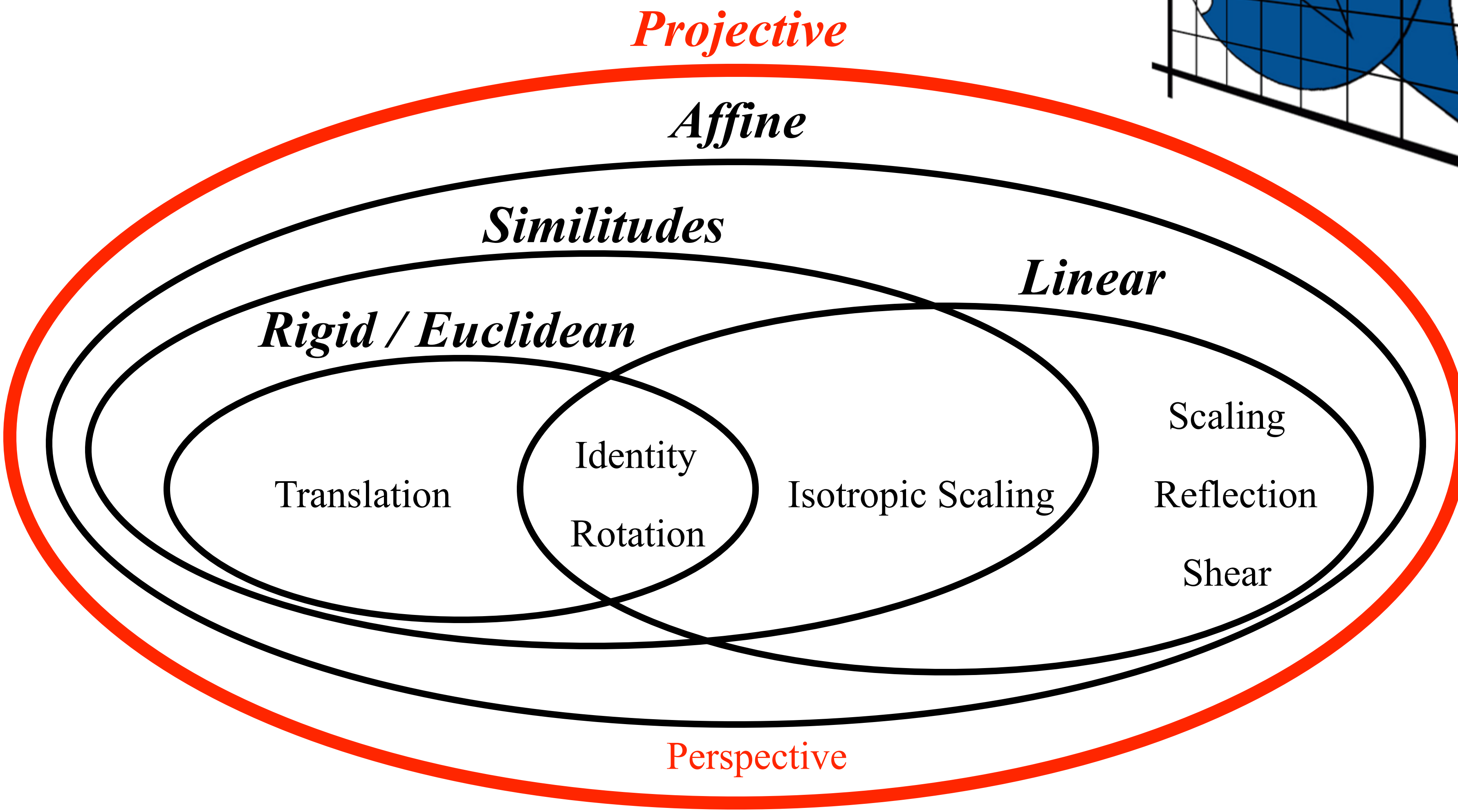


“Point A+B”
Vector sum
Point A
Vector
from
origin to A
Vector
from
origin to A
Point B
Vector from origin to B
Origin

# Phase 3: Profit

- **All these are linear in homogeneous coordinates!**

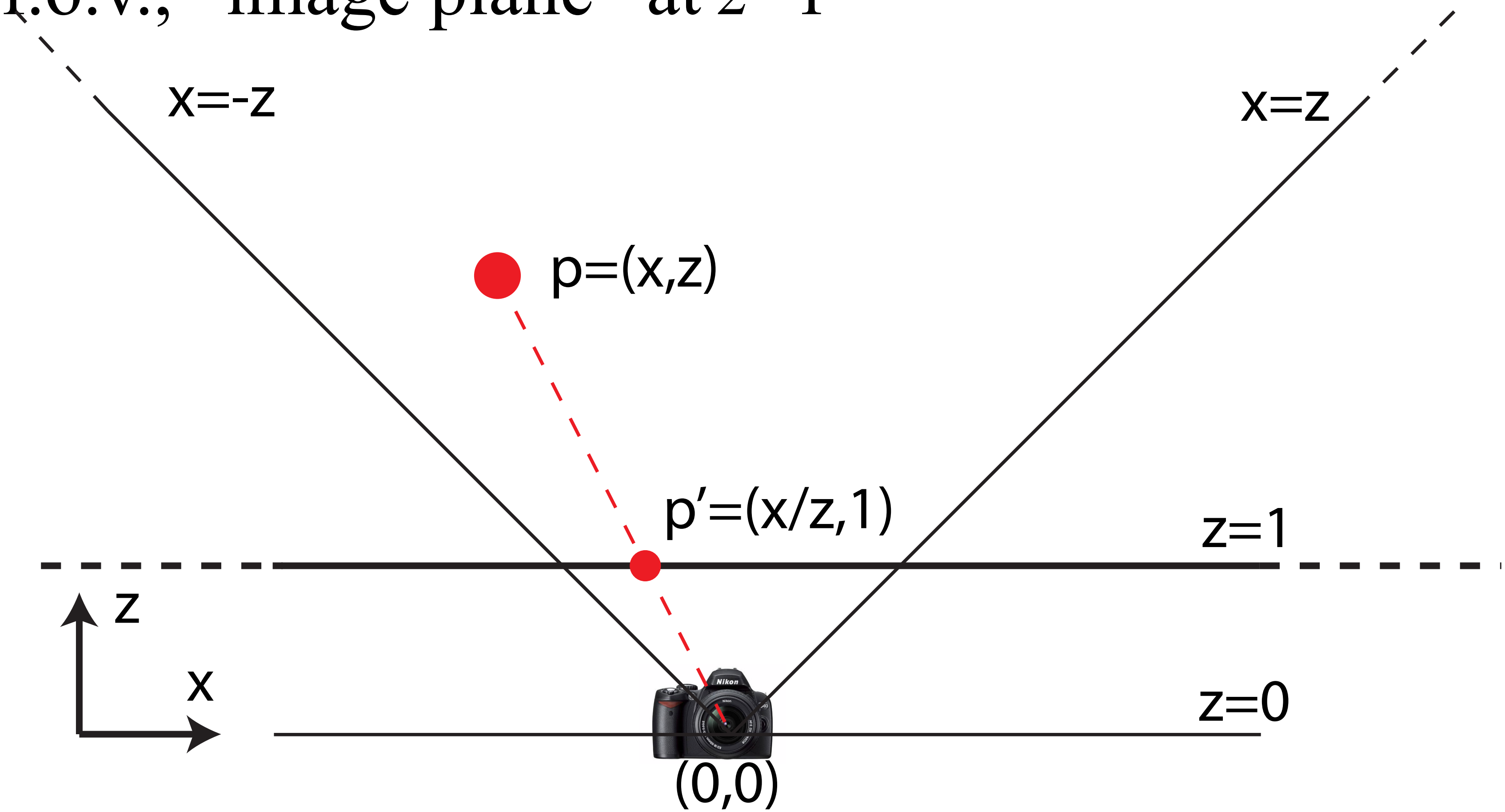




**See rest of slides & handout in MyCourses for more information on projections**

# Perspective in 2D

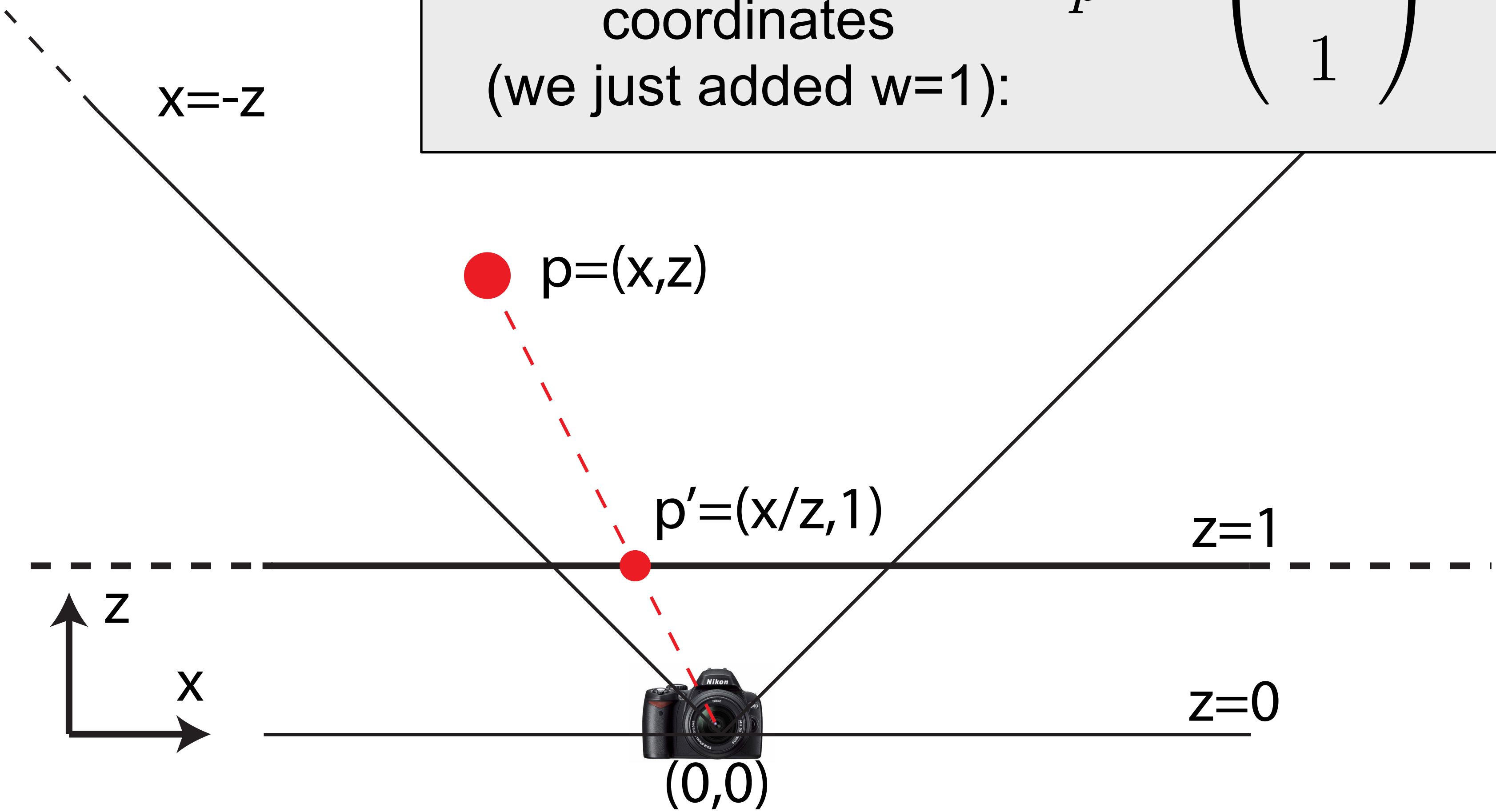
- Camera at origin, looking along *z*, 90 degree f.o.v., "image plane" at *z*=1



# Perspective in 2D

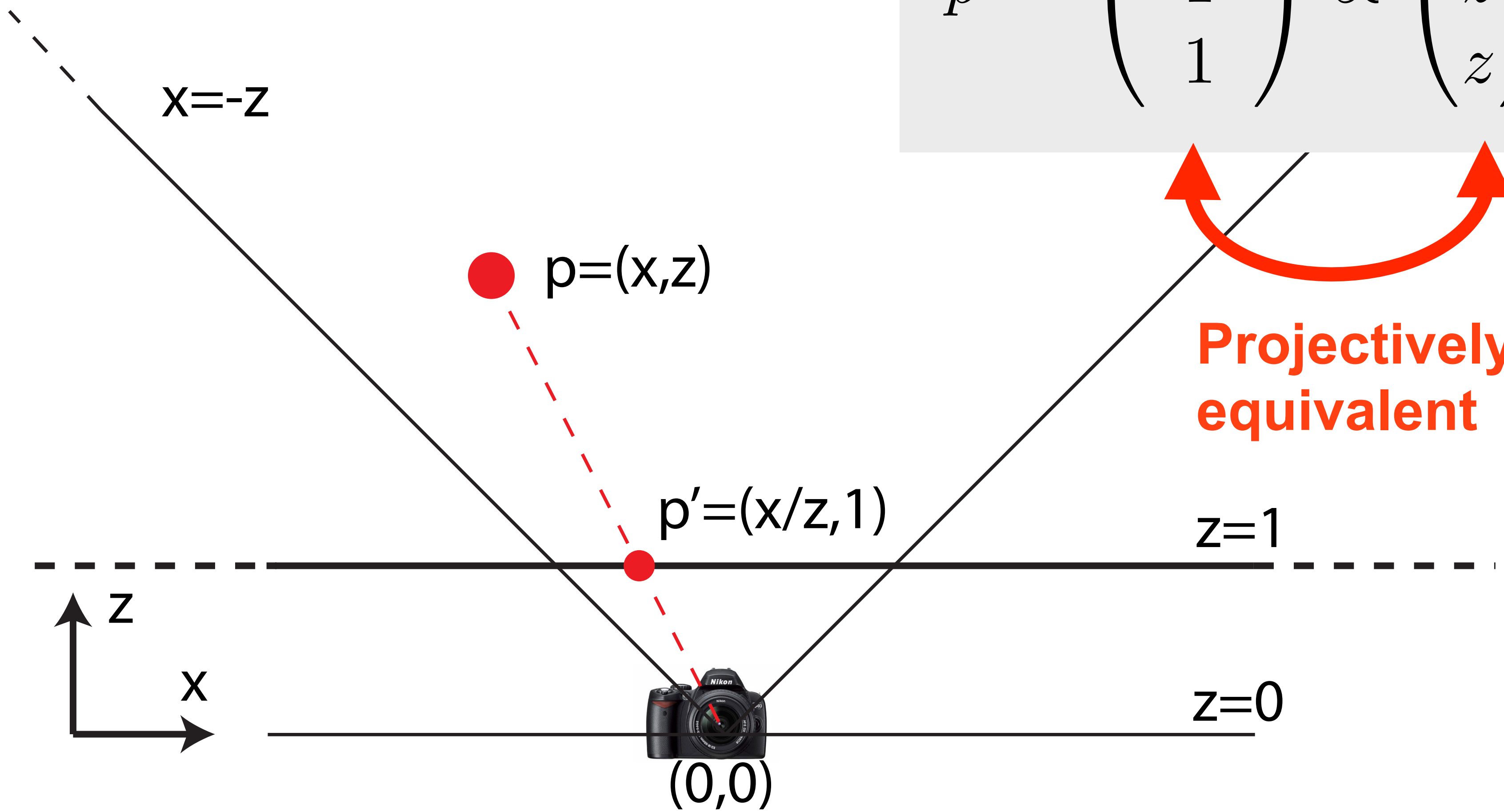
The projected point in homogeneous coordinates (we just added w=1):

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



# Perspective in 2D

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



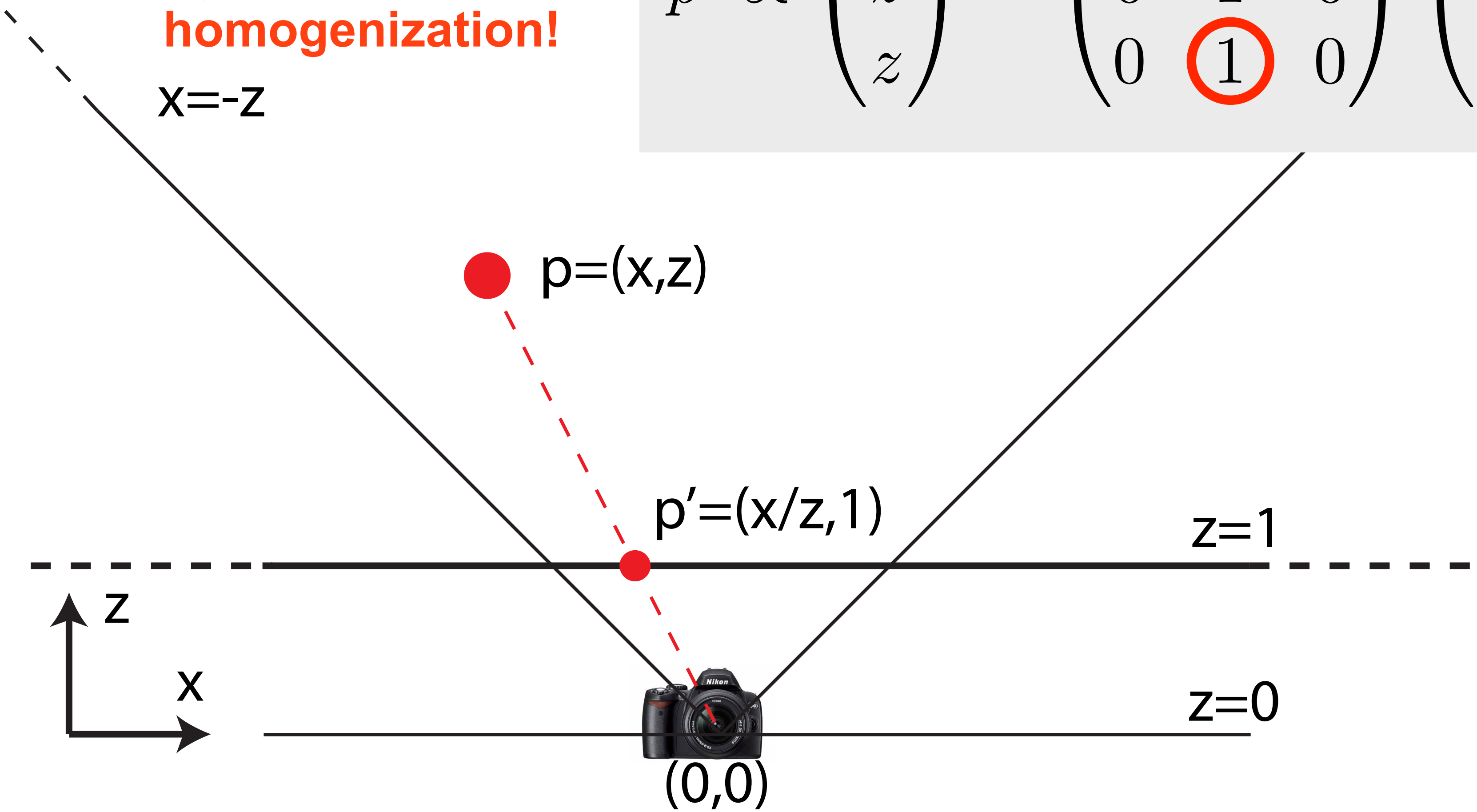
# How do you do this with a matrix?

$$\begin{pmatrix} x \\ z \\ z \end{pmatrix} = \begin{pmatrix} & & \\ & ? & \\ & & \end{pmatrix} \begin{pmatrix} x \\ z \\ 1 \end{pmatrix}$$

# 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 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.

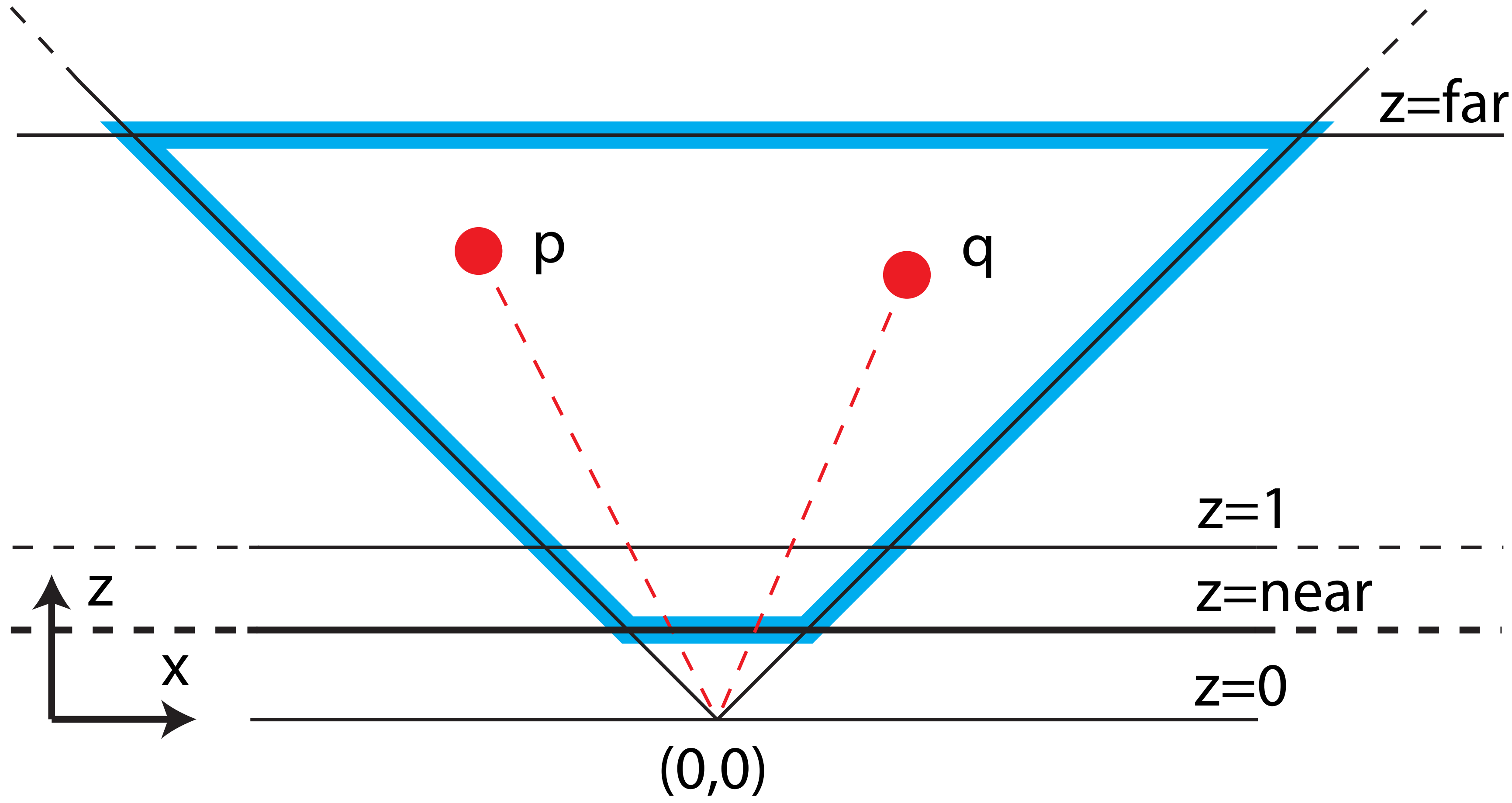
$$\begin{pmatrix} x' \\ y' \\ z' \\ w' \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

# 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.
  - Not a big deal for ray tracers, but GPUs need distances for "Z buffering", i.e., figuring out what is in front of what.

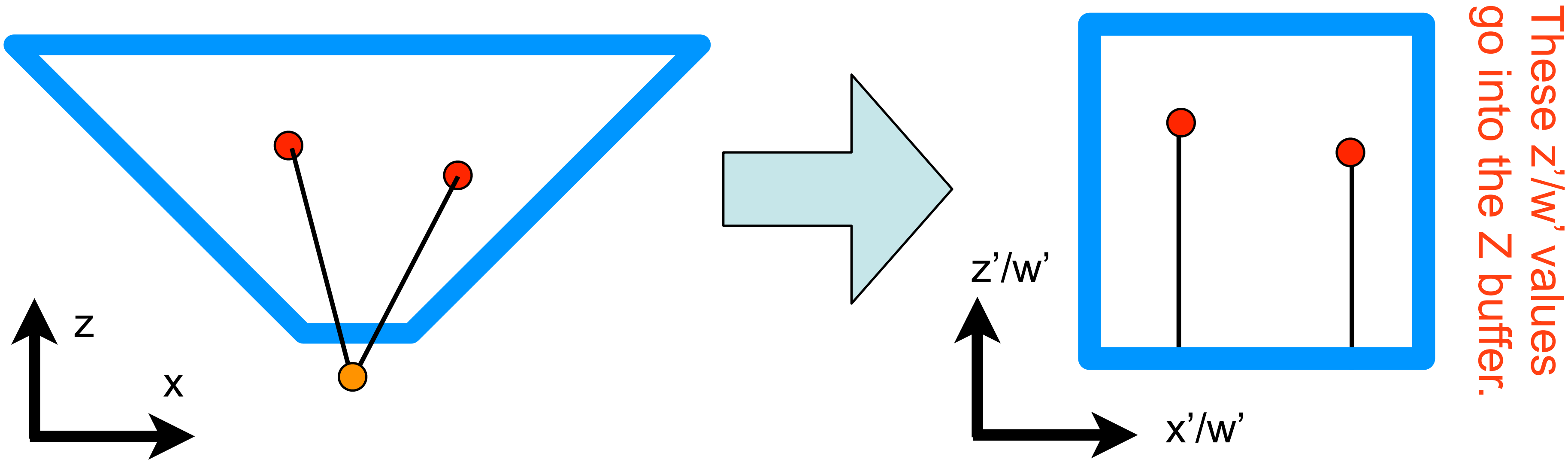
# The "View Frustum" in 2D

- (In 3D this would be a truncated pyramid.)



# The View Frustum in 2D

- We can transform the frustum by a modified projection in a way that makes it a square after projection and homogenization (division by $w$).

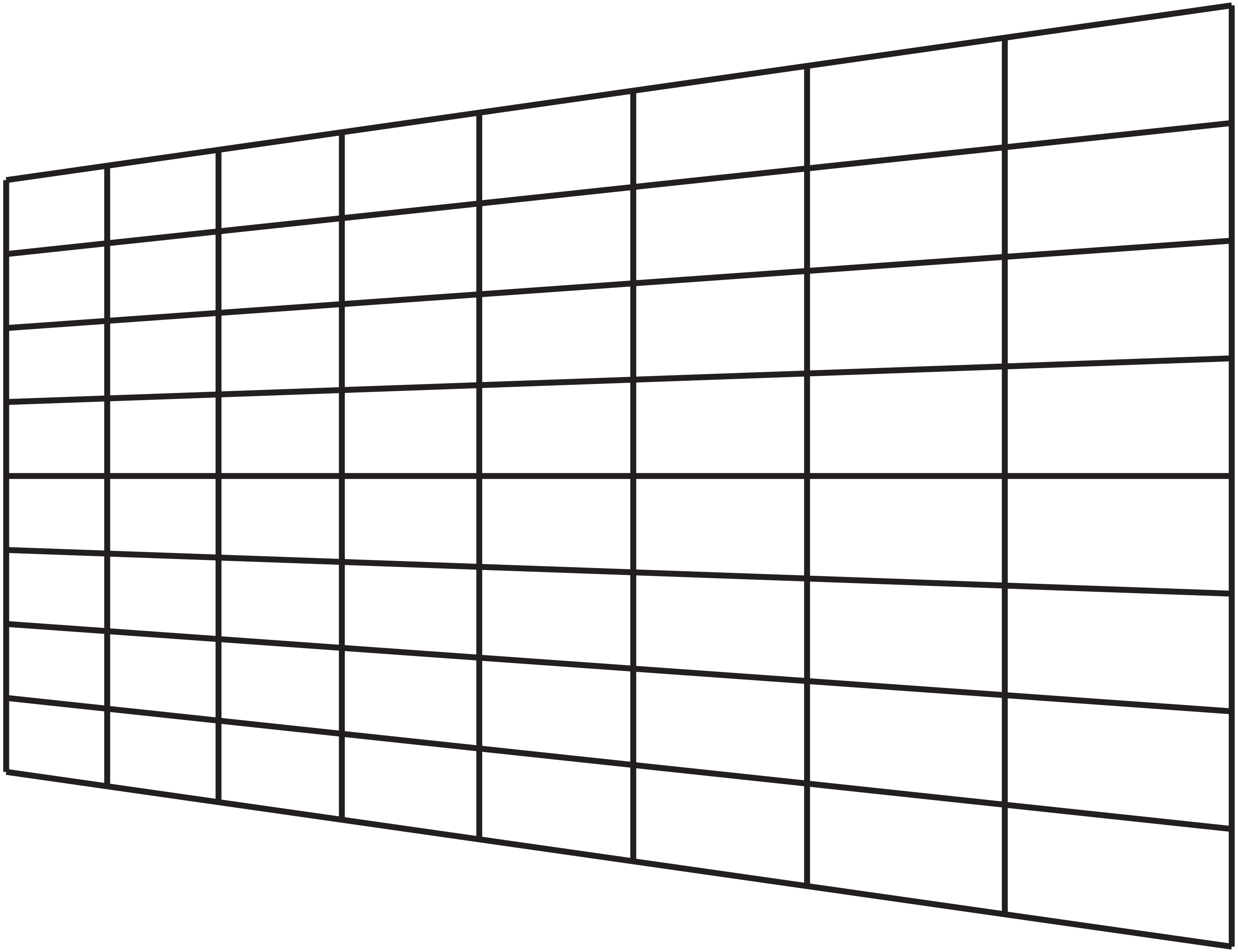



$$\begin{pmatrix} x' \\ z' \\ w' \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \frac{f+n}{f-n} & -\frac{2*f*n}{f-n} \\ 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} x \\ z \\ 1 \end{pmatrix},$$

# Details in Handout in MyCourses

- "Understanding Projections and Homogenous Coordinates"
  - How you get the matrix from previous slide

- Fun demonstration: Print the following page out. Holding the paper as flat as possible, try to see if you can hold it at such an angle in front of you so that the grid looks like a square with a regular grid in it.



# Homogeneous coordinates in vision

- "Structure from Motion" -algorithms
  - SfM if a branch of computer vision that tries to understand the 3D structure of the scene from pictures taken from different viewpoints.
  - It's all based on projective geometry and homogeneous coordinates.
  - Examples:
    - http://phototour.cs.washington.edu/
    - http://phototour.cs.washington.edu/findingpaths/
    - **http://visual.cs.ucl.ac.uk/pubs/instant3d/ (CS3100 alumni Peter Hedman's PhD work!)**