CS 248 Review Session #3

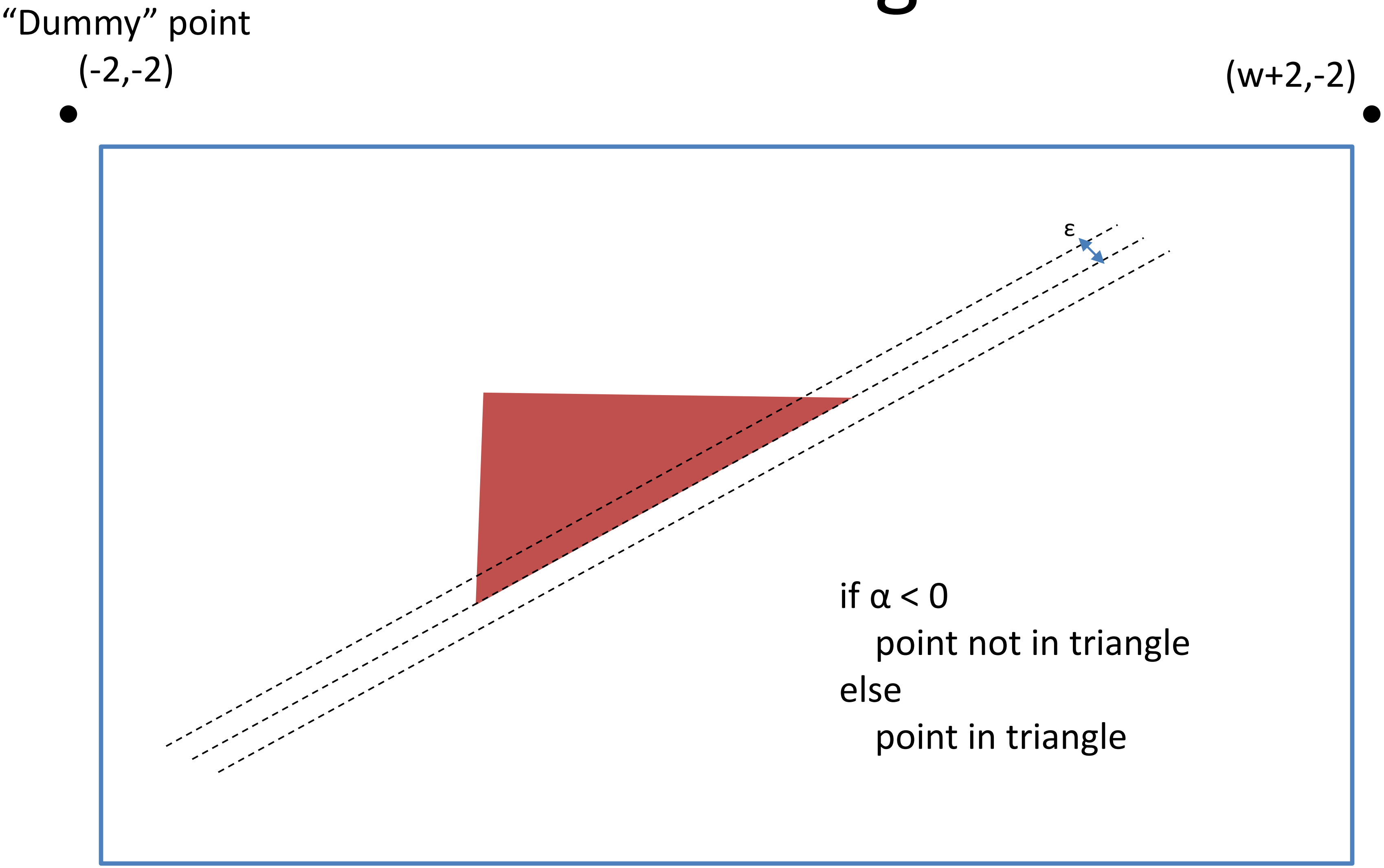
Handling Shadow Edges
Perspective Correct Transformation
Normal Transformation

Shadow Edges

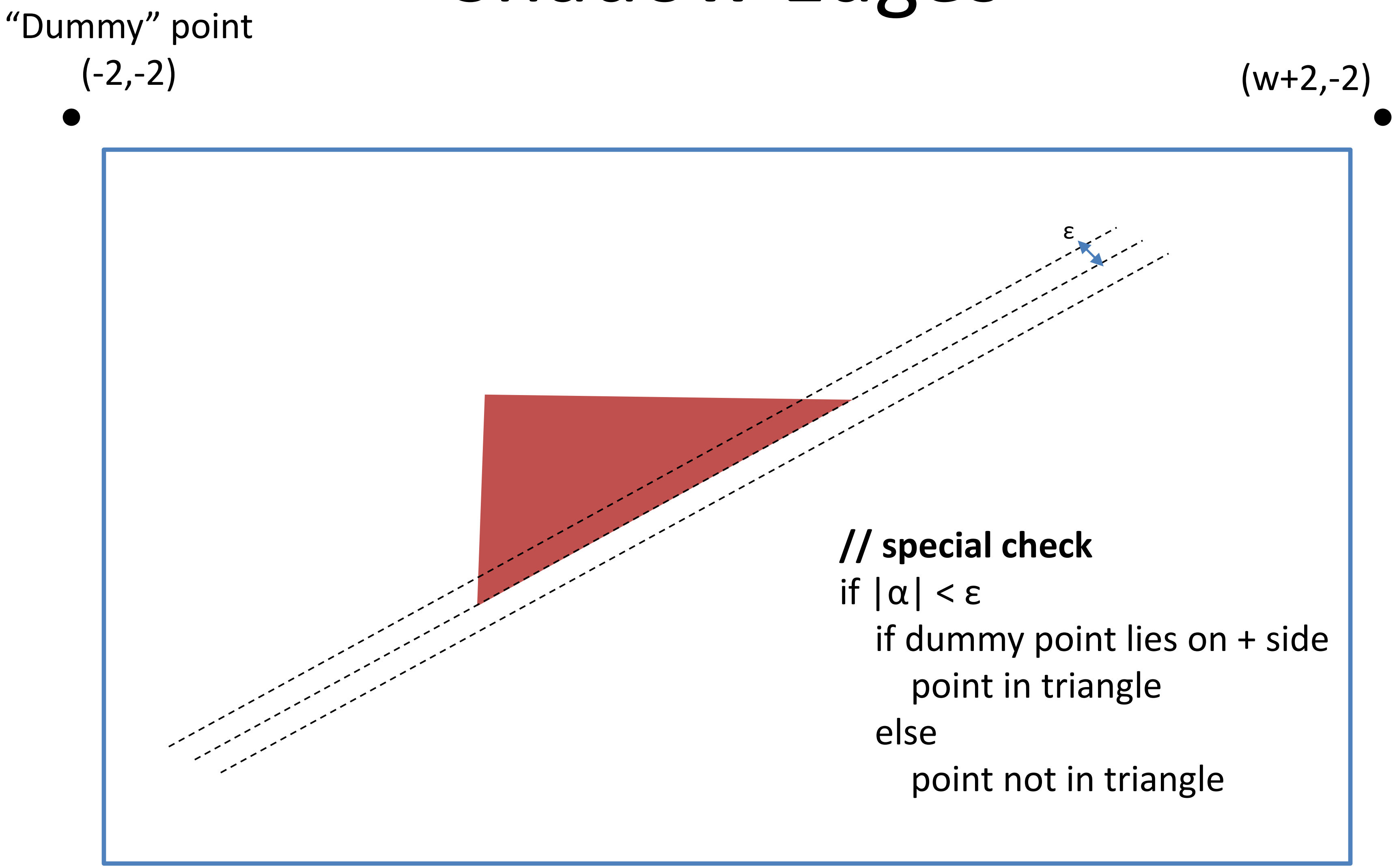
- Caused by floating point precision
  - Not really possible to check if a value is < **0**

- A proper solution may not be accurate, but should be consistent among pixel centers
  - Adjacent triangles should write to edge pixels **exactly** once

Shadow Edges



Shadow Edges



// special check

if |α| < ε

    if dummy point lies on + side

        point in triangle

    else

        point not in triangle

Perspective Correct Transformation

- **Affine**
  - Linearly interpolates texture coordinates
- **Perspective Correct**
  - Stretches near front, squeezes near back

Comparison

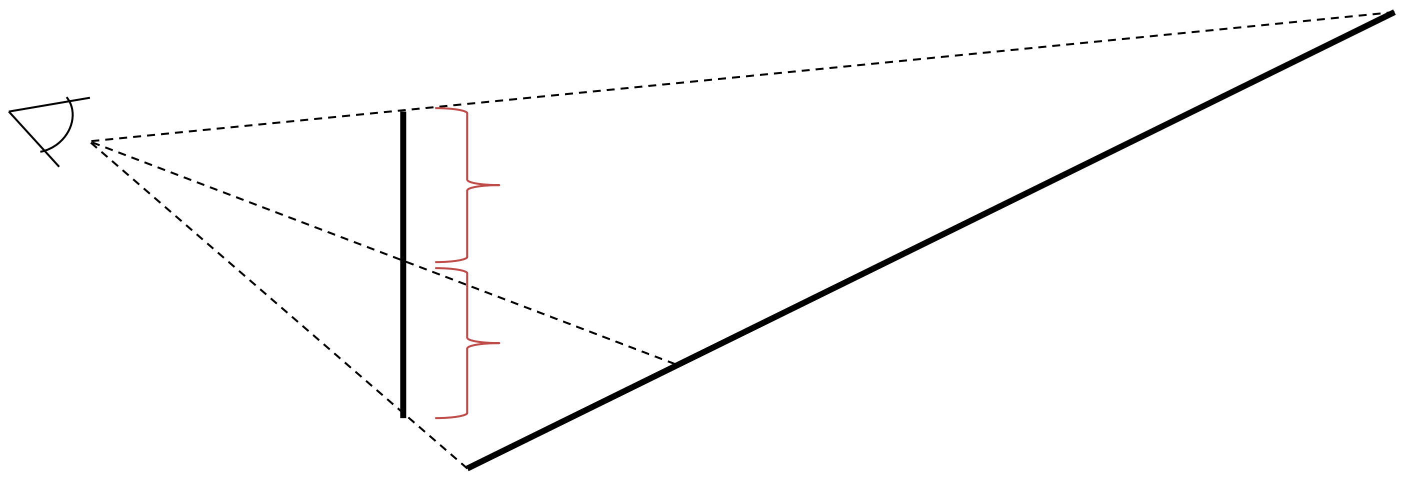
- **Affine**
  - Fast to compute
  - Useful for hacks such as drawing vertical/horizontal lines in *Doom*

- **Perspective Correct**
  - Accurate texture mapping accounting for depth
- **Hybrid *(Persp Correct every 16 pixels)***
  - Fast and more or less accurate

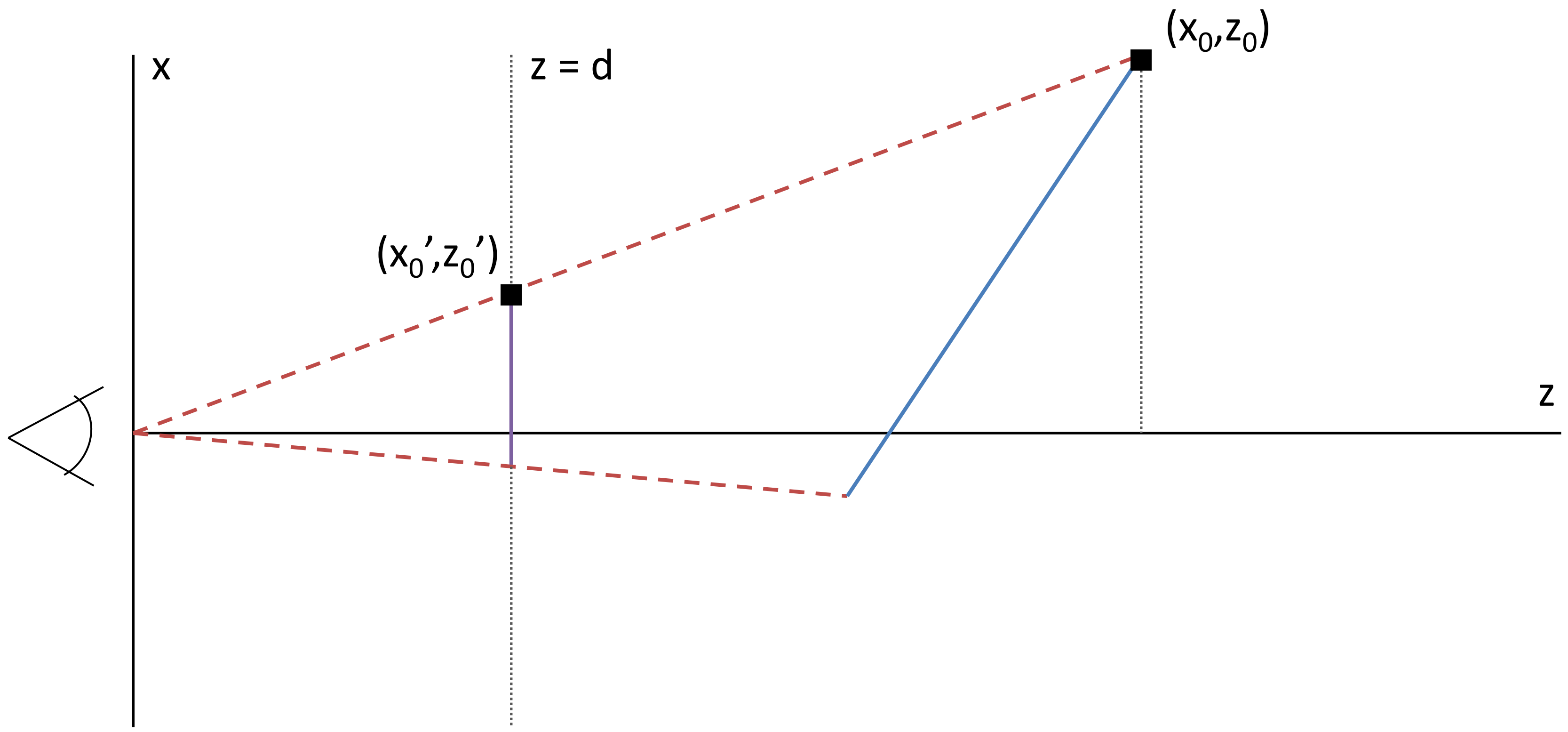
Image Source: Wikipedia

Why not linearly interpolate?

- U, V texture coordinates can't be linearly interpolated in screen space

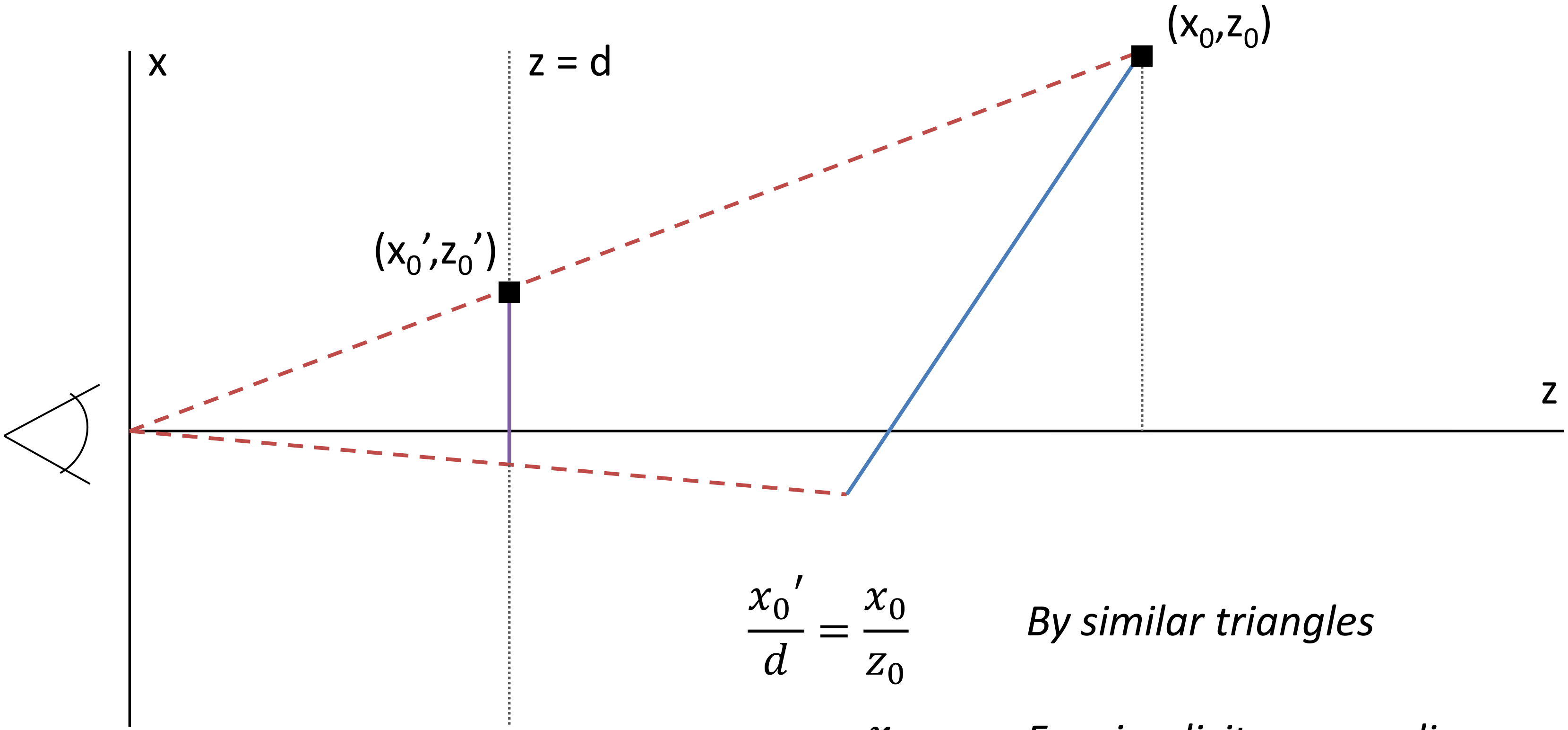


2D Derivation



Source: Game Developer (April 1995) http://chrishecker.com/images/4/41/Gdmtex1.pdf

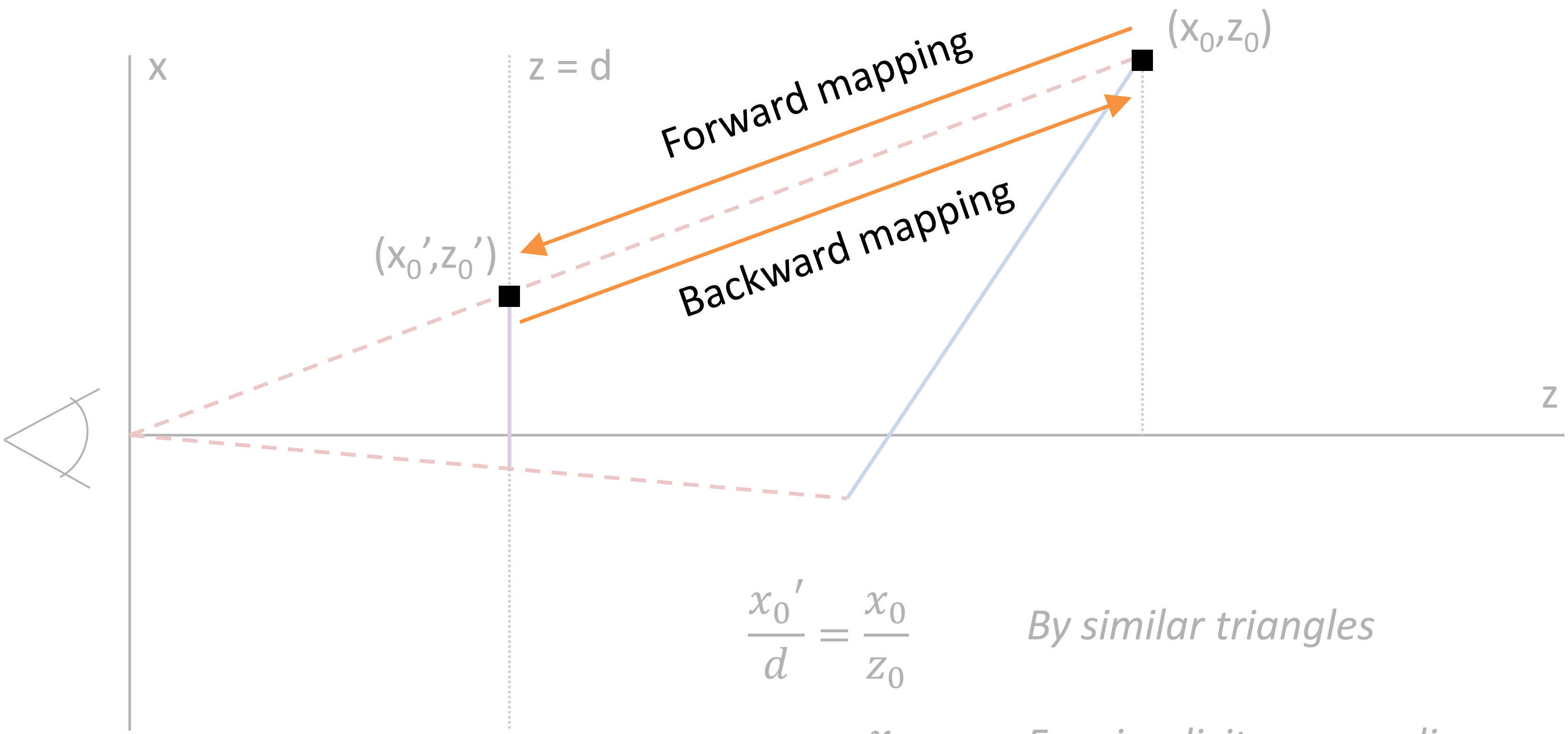
2D Derivation



$$\frac{x_0'}{d} = \frac{x_0}{z_0}$$ *By similar triangles*

$$x' = \frac{x}{z}$$ *For simplicity, generalize x, z and assume d = 1 for simplicity*

2D Derivation



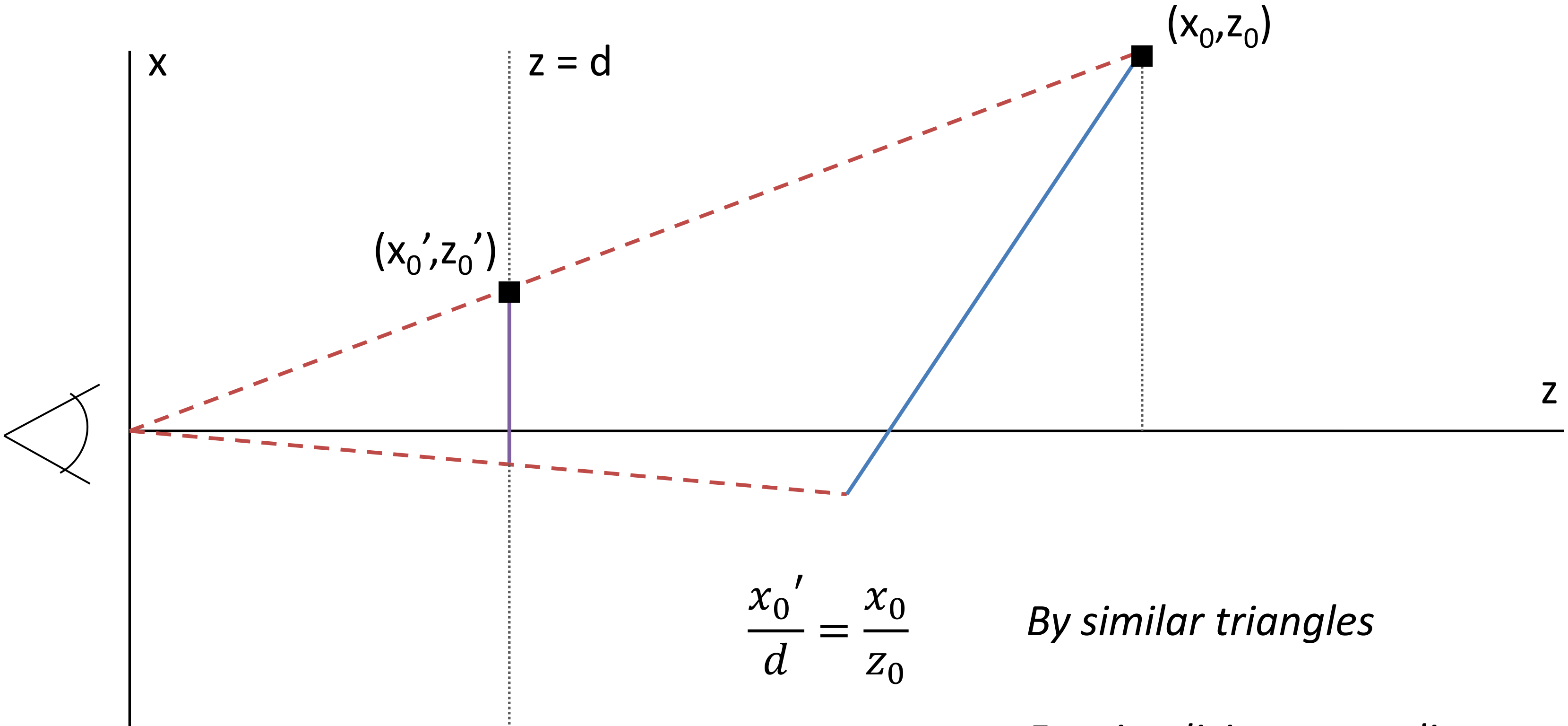
$$\frac{x_0'}{d} = \frac{x_0}{z_0}$$

By similar triangles

$$x' = \frac{x}{z}$$

For simplicity, generalize x, z and assume d = 1 for simplicity

2D Derivation



$$\frac{x_0{}'}{d} = \frac{x_0}{z_0}$$ *By similar triangles*

$$x' = \frac{x}{z}$$ *For simplicity, generalize x, z and assume d = 1 for simplicity*

$$x = x'z$$ *Using backward mapping*

Can it be linearly interpolated?

- Linear relationships can be written as:

$$y = Ax + B$$

- Find a linear relationship between x' and z *(why?)*

$$x = Az + B$$ *Start with the unprojected line in object space*

$$x'z = Az + B$$

$$z(x' - A) = \mathrm{B}$$

$$z = \frac{B}{x' - A}$$ *This is **not** a linear equation of z w.r.t. x'*

$$\frac{1}{z} = \frac{x' - A}{B}$$

$$\frac{1}{z} = \frac{1}{B}x' - \frac{A}{B}$$ *This is a linear equation of 1/z w.r.t. x'*

Can it be linearly interpolated?

- This means we can recover the original unprojected *x* value:

$$\frac{x'}{\frac{1}{z}} = \frac{\frac{x}{z}}{\frac{1}{z}} = x$$

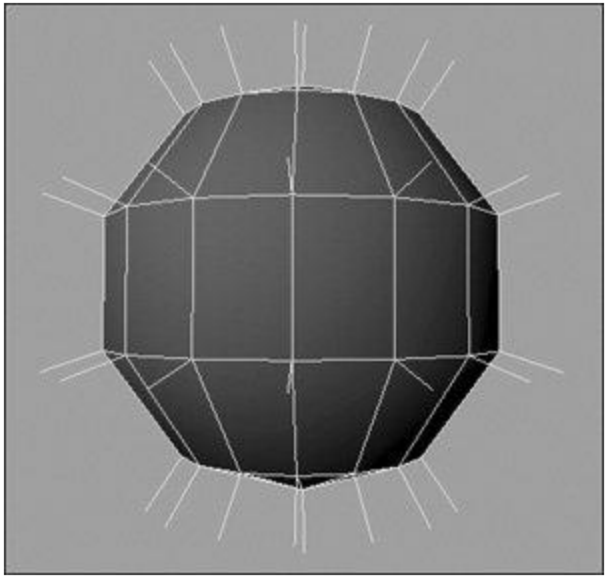
- *x/z* and *1/z* are linear in screen space
- *u, v* and *x* are linear w.r.t. each other
- Thus, *u/z* is linear in screen space

Overview

- Find $z' = 1/z$ at each vertex
- Linearly interpolate u' and z' in screen space
- At each pixel, calculate u by u'/z' to recover the u value at the unprojected line
- Generalize from 2D to 3D case

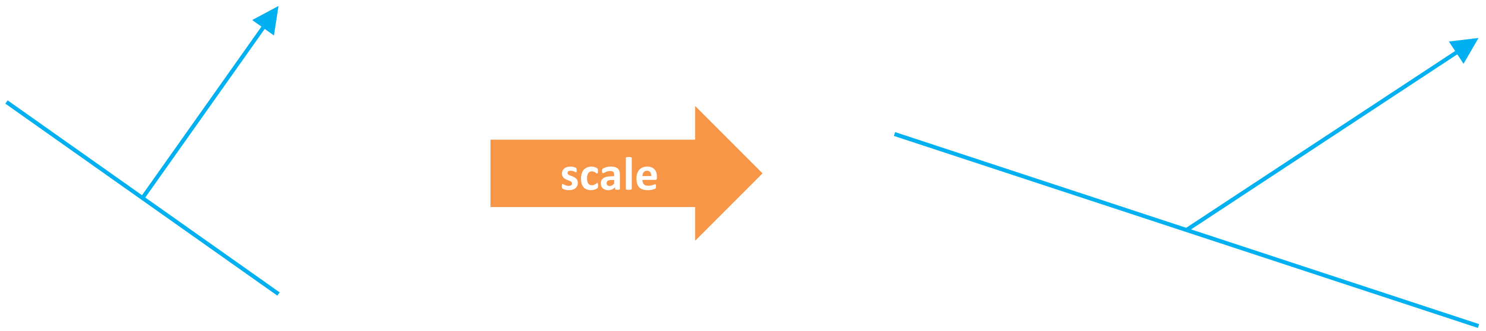
Normals

- Normals are specified for each vertex
- Uses include lighting effects, bump mapping
- Interpolate the normals with perspective correctness (like texture coordinates)



Normal Transformations

- Naïve approach: apply modelview transform on normal vector



Normal Transformations

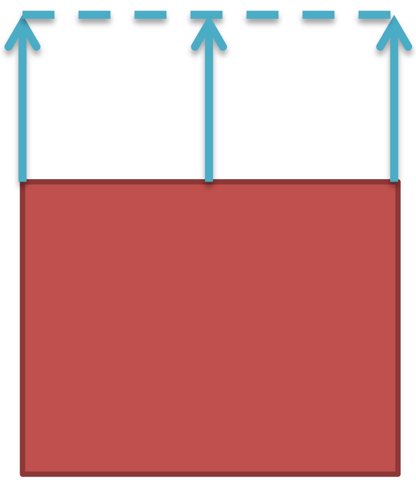
- Normals are directions
  - w = 0 (w = 1 only for points)
  - Translations have no effect
  - Ignore 4^{th} column of modelview matrix
- Normals are non-projective
  - Ignore 4^{th} row as well

Normal Transformations

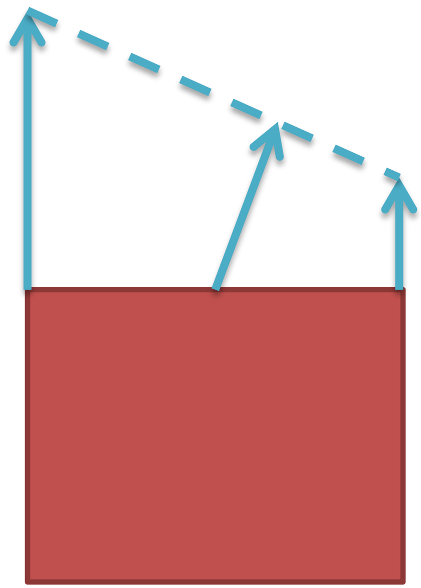
- Transform normals by multiplying by the inverse transpose of the top-left 3x3 part of the modelview matrix
- Why?
  - Translations don’t matter ($w = 0$)
  - Inverse transpose of a rotation matrix is the same
  - Inverse transpose of scaling is 1/scaling factor

Normal Transformations

- Remember to normalize!
  - After any transformation, normalize the vertex normal
  - Normalize the result at each pixel/fragment before computing lighting



vs



Questions?

- Assignment 2 involves
  - Perspective correct transformations
  - Normal transformations
  - Lighting calculations