CS-184: Computer Graphics

Lecture #4: 2D Transformations

Prof. James O'Brien
University of California, Berkeley

V2008-F-04-1.0

Today

- 2D Transformations
  - “Primitive” Operations
    - Scale, Rotate, Shear, Flip, Translate
  - Homogenous Coordinates
  - SVD
  - Start thinking about rotations...

Introduction

- Transformation:

  An operation that changes one configuration into another

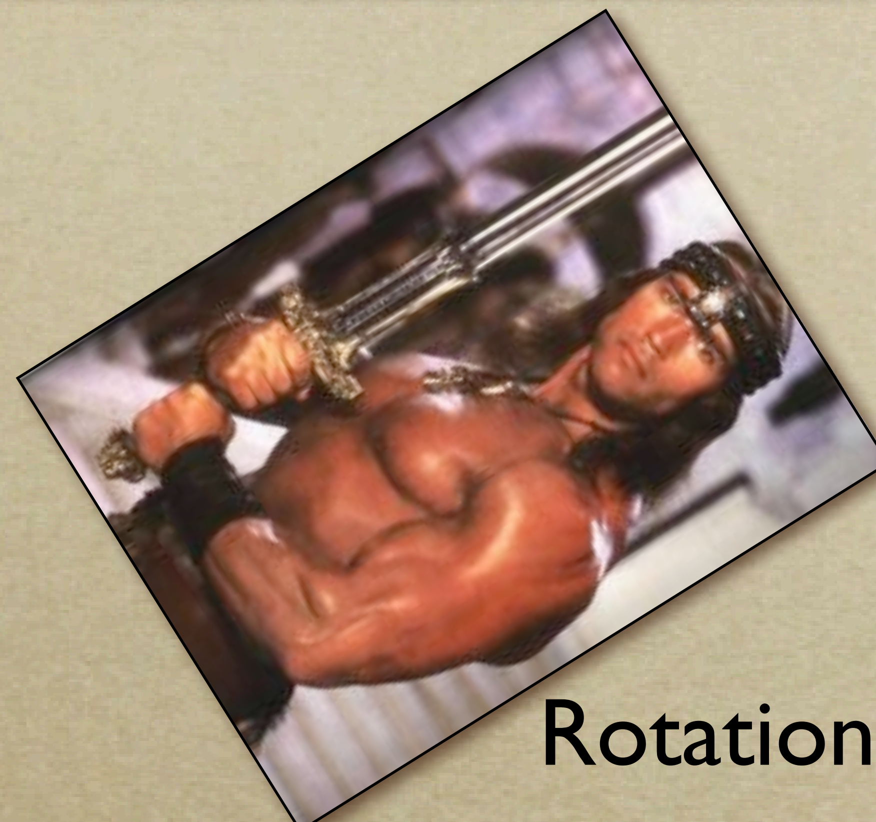
- For images, shapes, *etc*.

  A geometric transformation maps positions that define the object to other positions

  Linear transformation means the transformation is defined by a linear function... which is what matrices are good for.

Some Examples

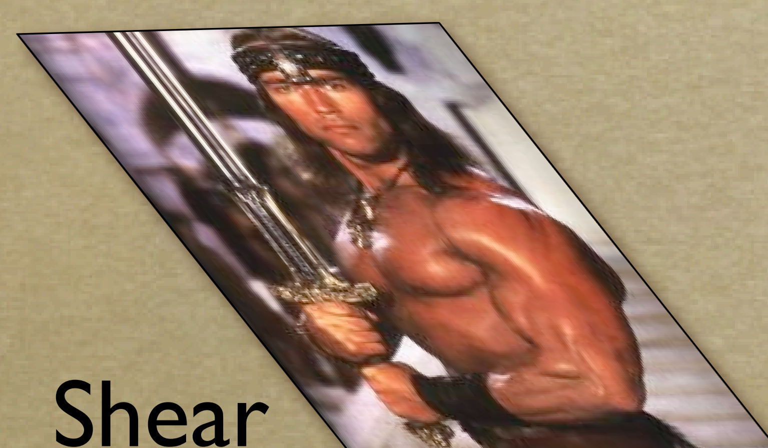
Original



Rotation

Uniform Scale

Nonuniform Scale

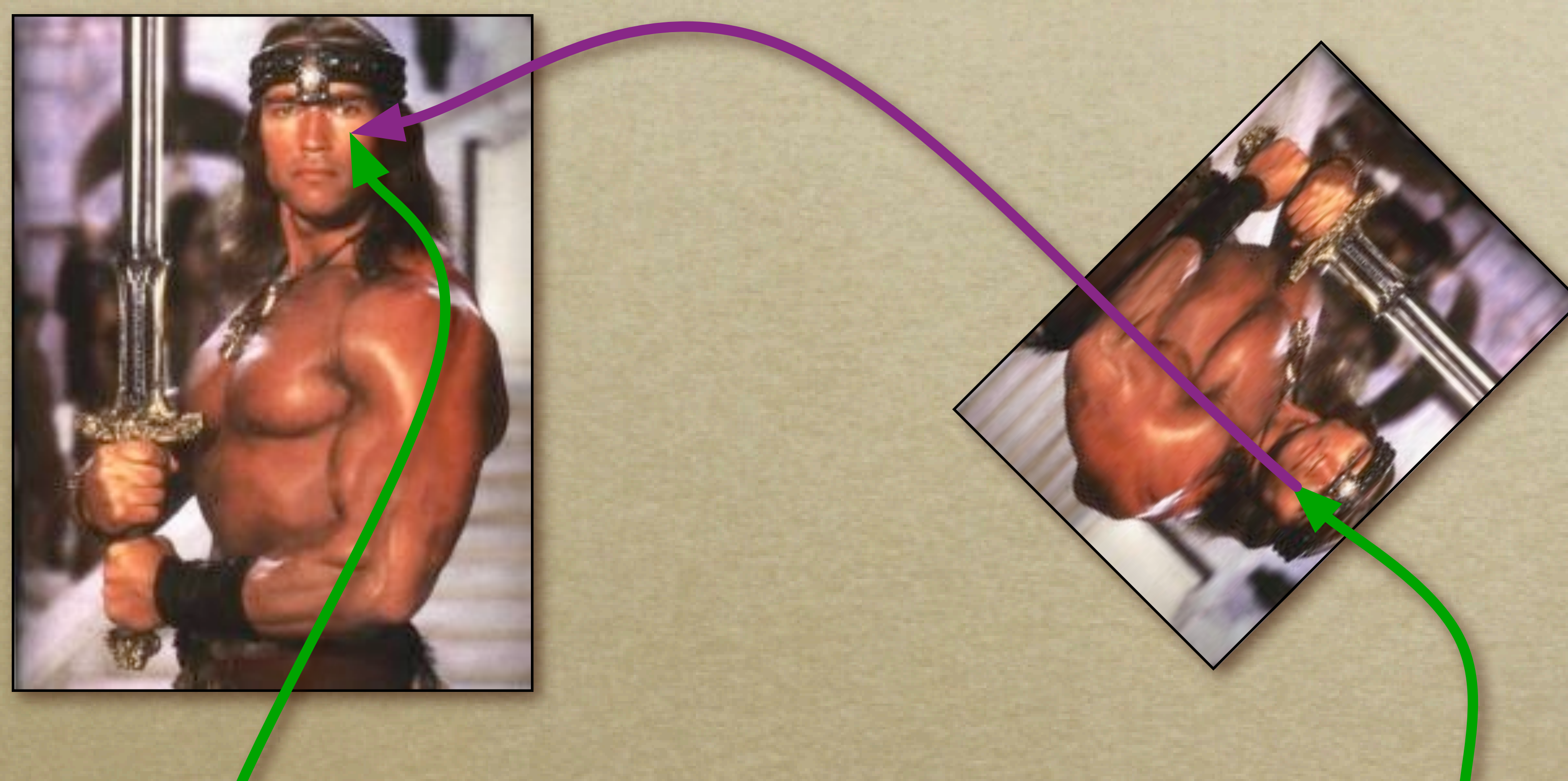


Shear

Images from *Conan The Destroyer*, 1984

Mapping Function

$$f(x) = x \text{ in old image}$$



$$c(x) = [195, 120, 58]$$

$$c'x = c(f(x))$$

Linear -vs- Nonlinear

Nonlinear (swirl)

Linear (shear)

Geometric -vs- Color Space

Color Space Transform
(edge finding)

Linear Geometric
(flip)

Instancing

RHW

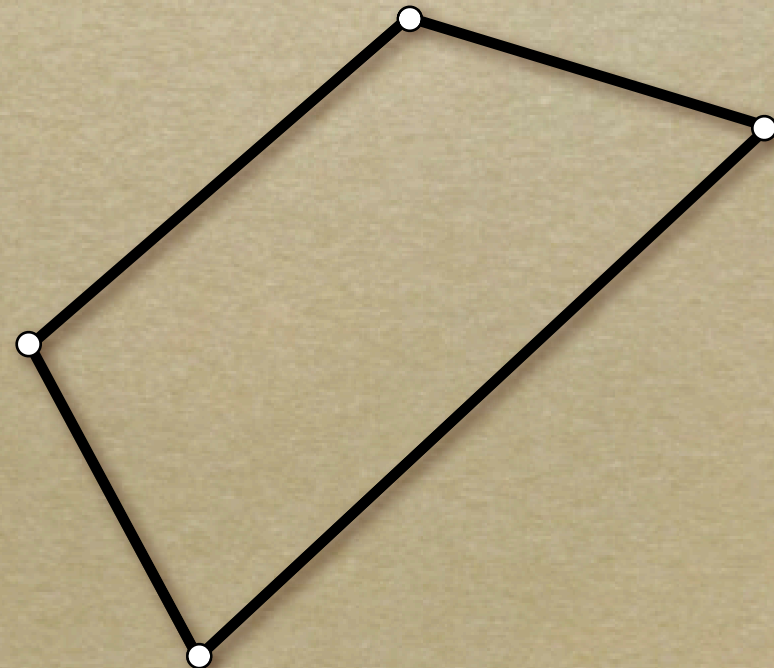
M.C. Escher, from Ghostscript 8.0 Distribution

Instancing

- Reuse geometric descriptions
- Saves memory

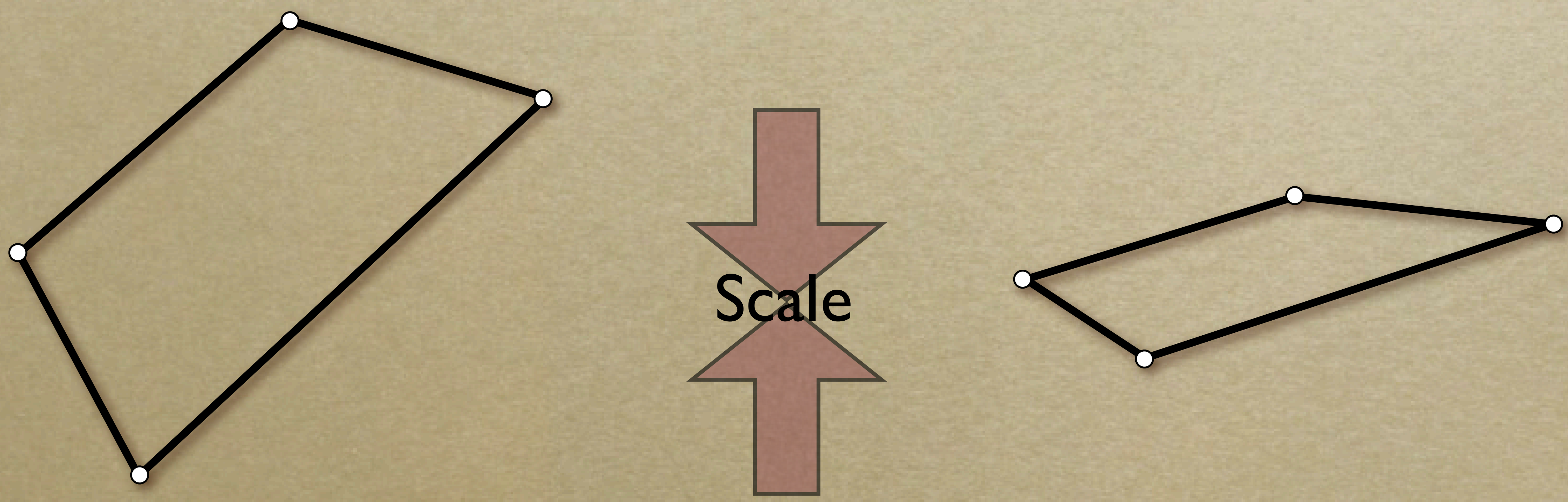
Linear is Linear

- Polygons defined by points
- Edges defined by interpolation between two points
- Interior defined by interpolation between all points
- *Linear* interpolation



Linear is Linear

- Composing two linear function is still linear
- Transform polygon by transforming vertices



Linear is Linear

- Composing two linear function is still linear
- Transform polygon by transforming vertices

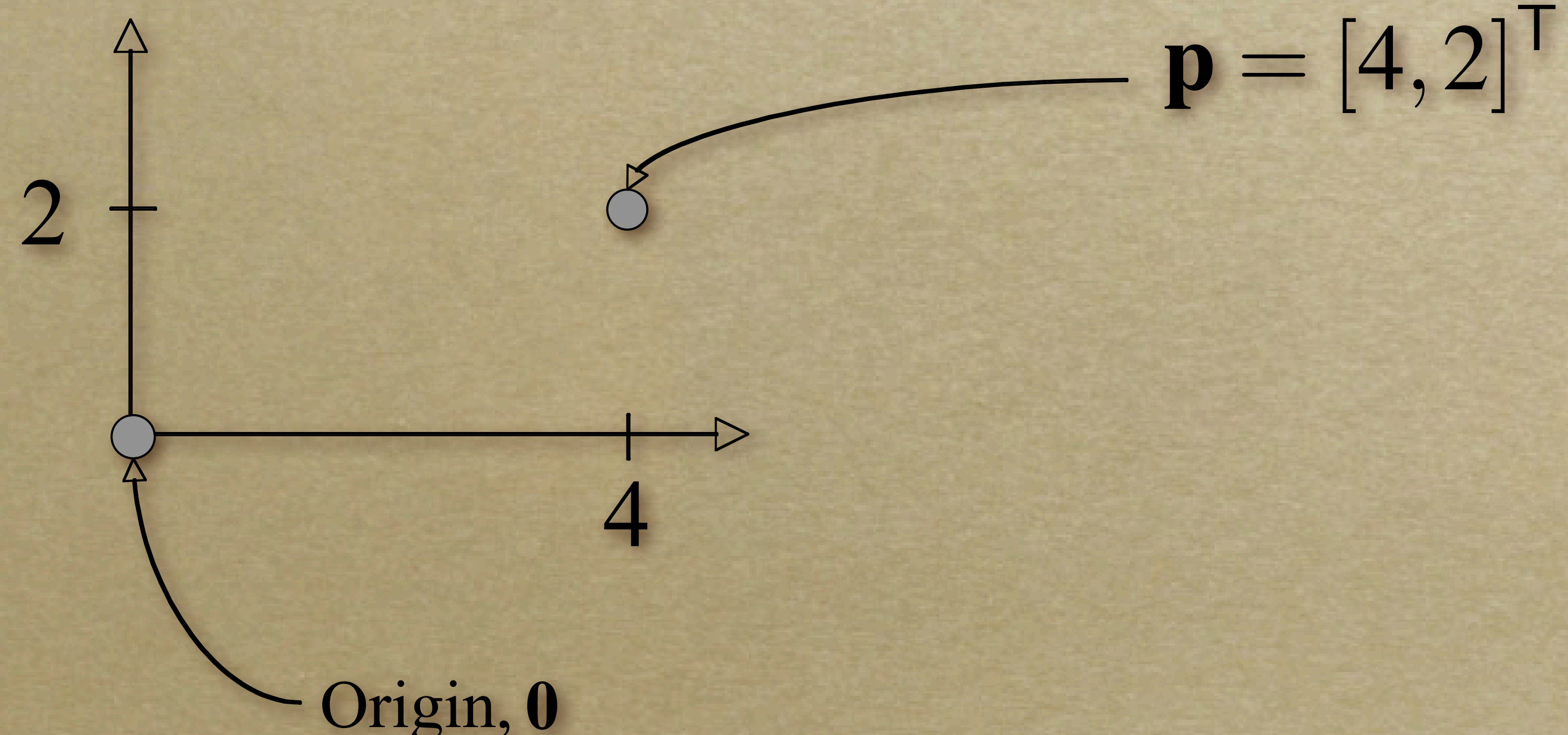
$$f(x) = a + bx \qquad g(f) = c + df$$

$$g(x) = c + df(x) = c + ad + bdx$$

$$g(x) = a' + b'x$$

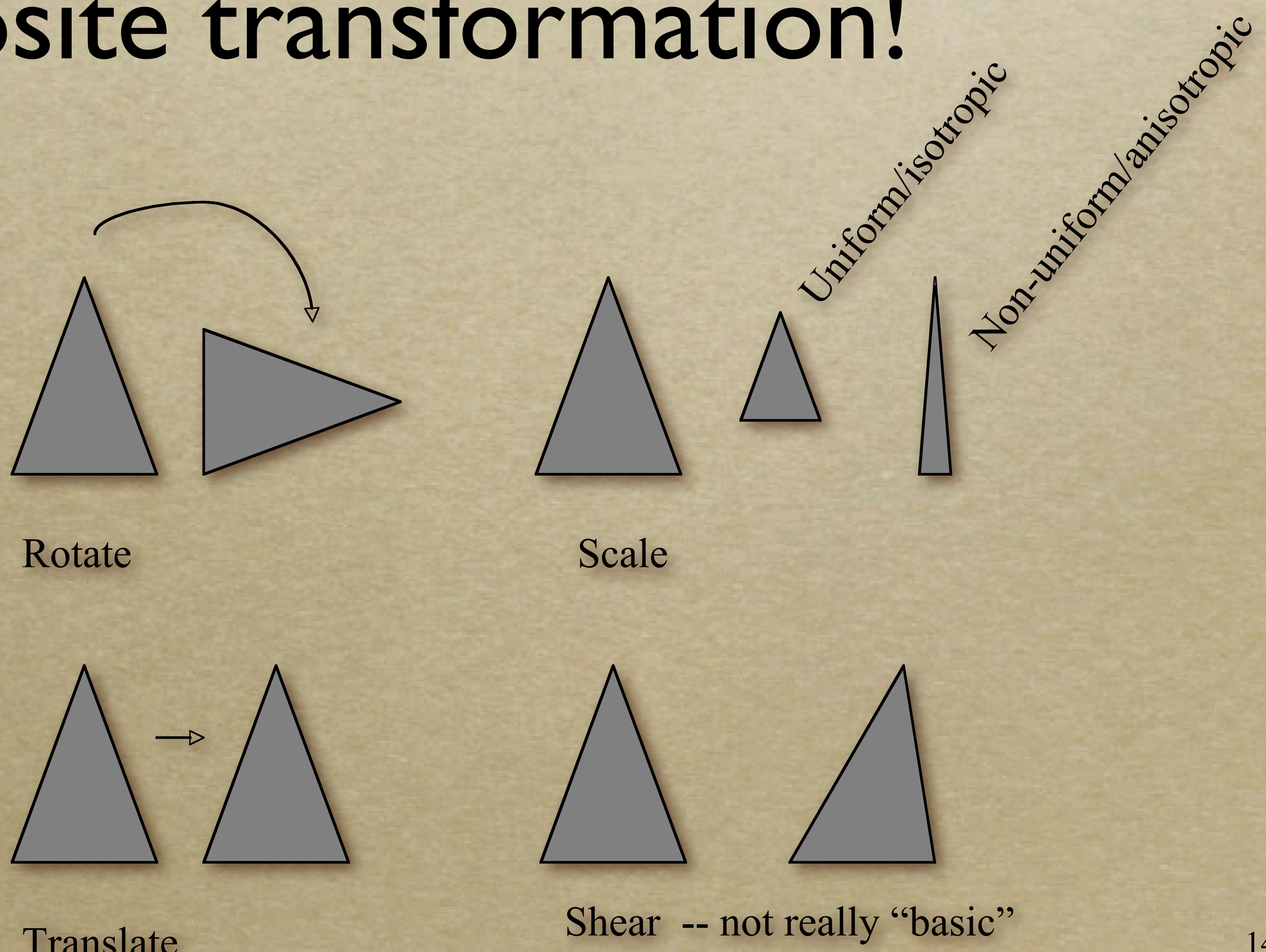
Points in Space

- Represent point in space by vector in R^n
  - Relative to some origin!
  - Relative to some coordinate axes!
- Later we'll add something extra...



Basic Transformations

- Basic transforms are: rotate, scale, and translate
- Shear is a composite transformation!



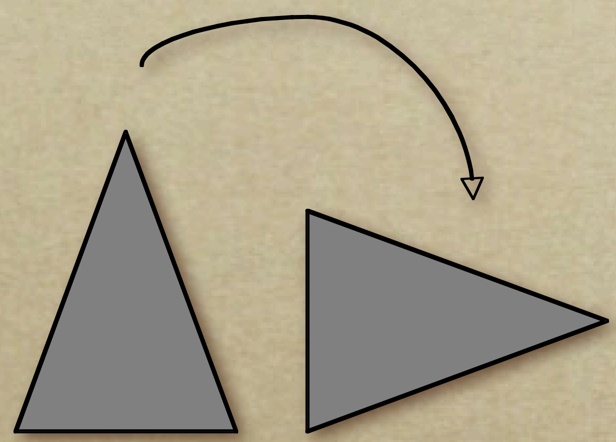
Linear Functions in 2D

$$x' = f(x, y) = c_1 + c_2 x + c_3 y$$
$$y' = f(x, y) = d_1 + d_2 x + d_3 y$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} t_x \\ t_y \end{bmatrix} + \begin{bmatrix} M_{xx} & M_{xy} \\ M_{yx} & M_{yy} \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix}$$

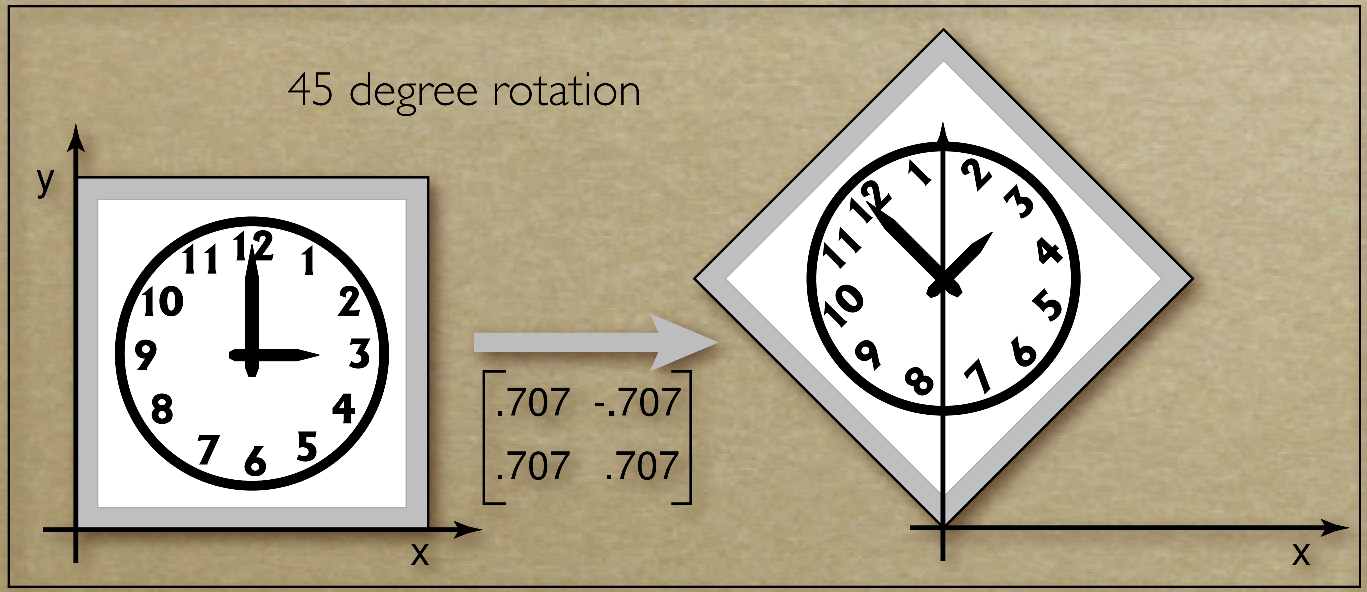
$$\mathbf{x}' = \mathbf{t} + \mathbf{M} \cdot \mathbf{x}$$

Rotations



$$\mathbf{p}' = \begin{bmatrix} Cos(\theta) & -Sin(\theta) \\ Sin(\theta) & Cos(\theta) \end{bmatrix} \mathbf{p}$$

Rotate



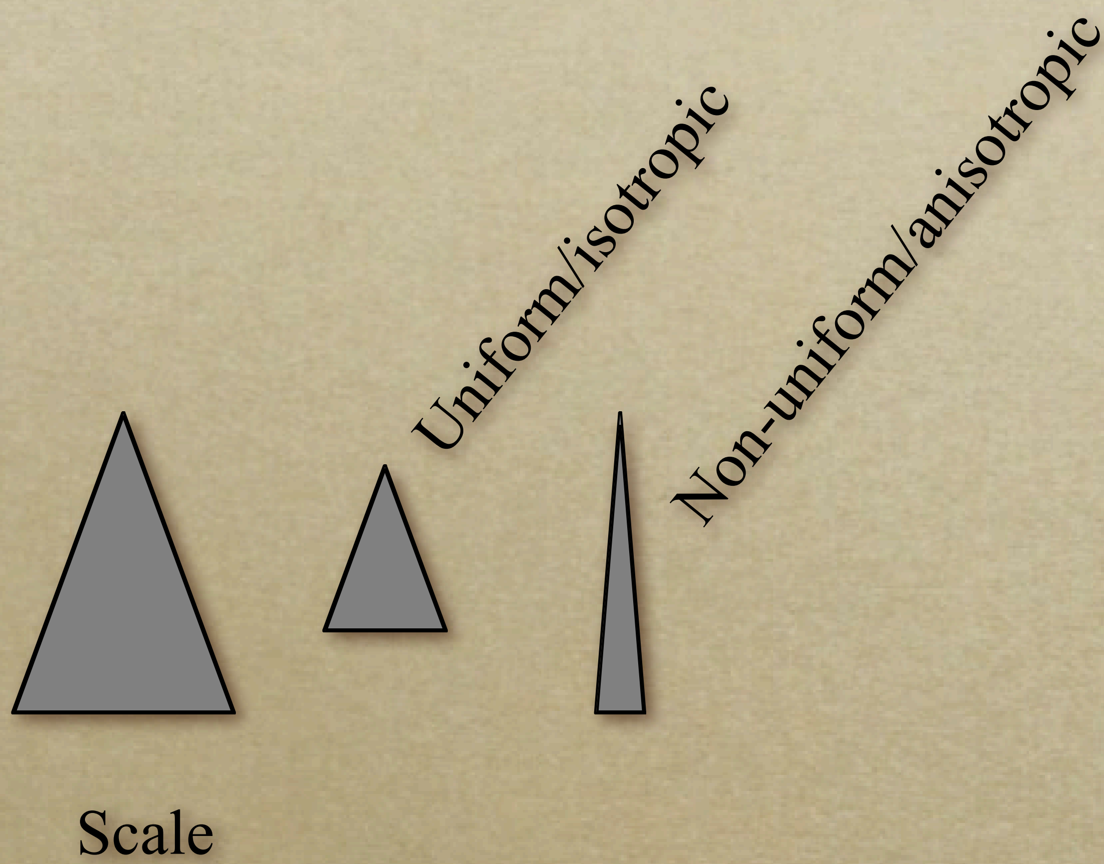
Rotations

- Rotations are positive counter-clockwise
- Consistent w/ right-hand rule
- Don’t be different...
- Note:
  - rotate by zero degrees give identity
  - rotations are modulo 360 (or 2π)

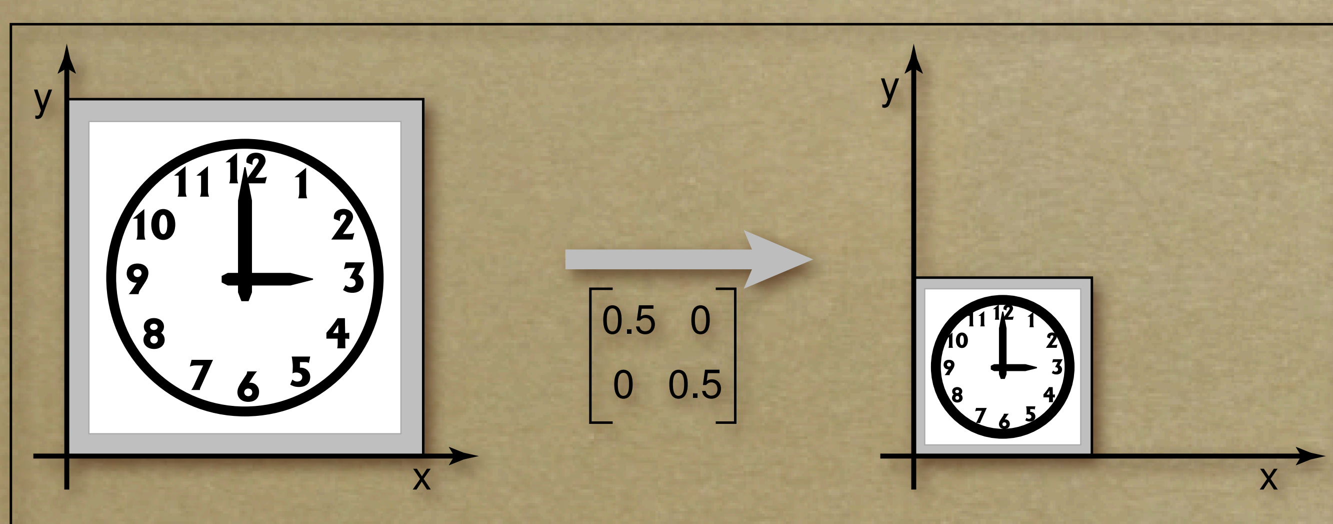
Rotations

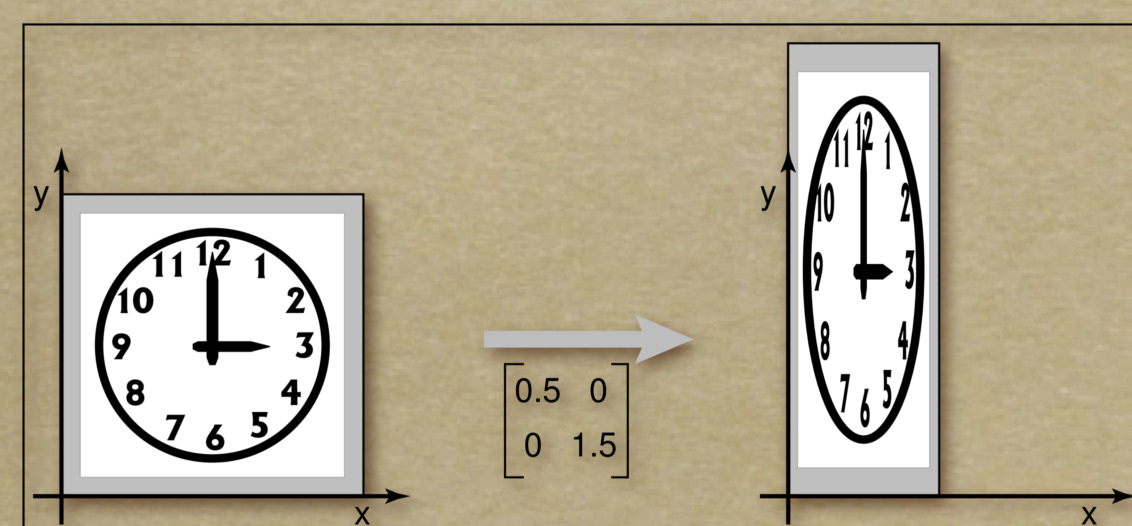
- Preserve lengths and distance to origin
- Rotation matrices are orthonormal
- $\mathrm{Det}(\mathbf{R}) = 1 \neq -1$
- In 2D rotations commute...
  - But in 3D they won't!

Scales



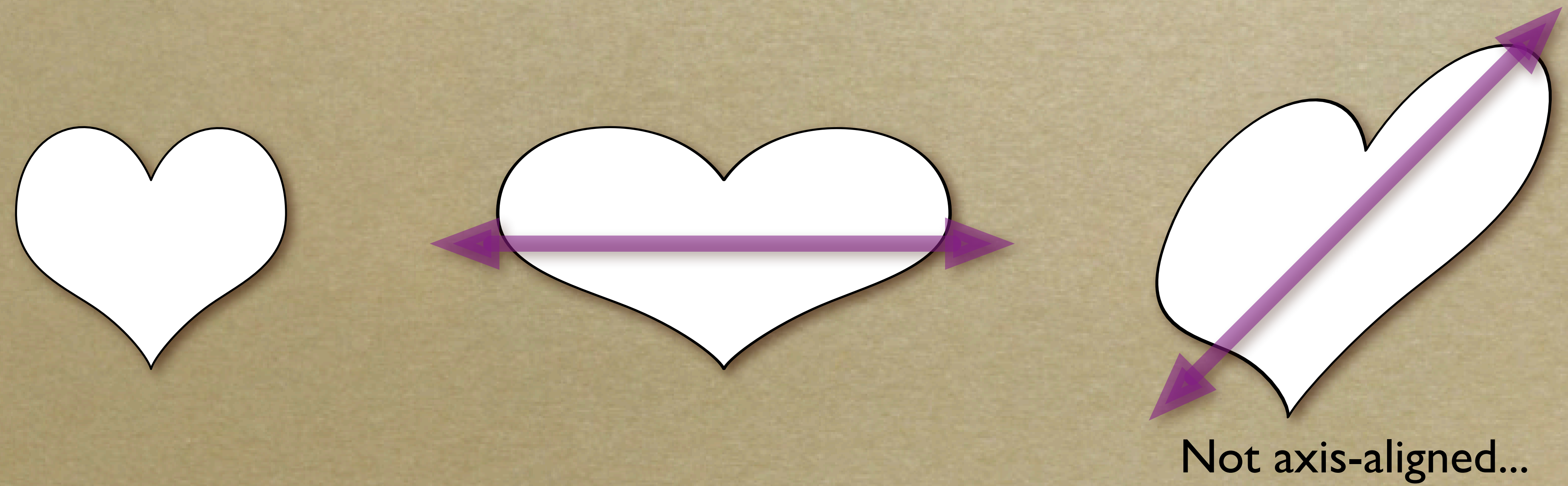
$$\mathbf{p}' = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \mathbf{p}$$



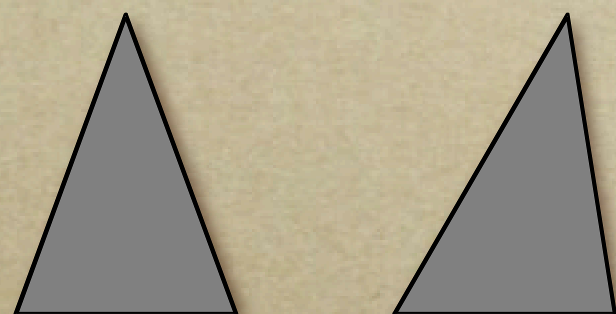


Scales

- Diagonal matrices
  - Diagonal parts are scale in X and scale in Y directions
  - Negative values flip
  - Two negatives make a positive (180 deg. rotation)
  - Really, axis-aligned scales

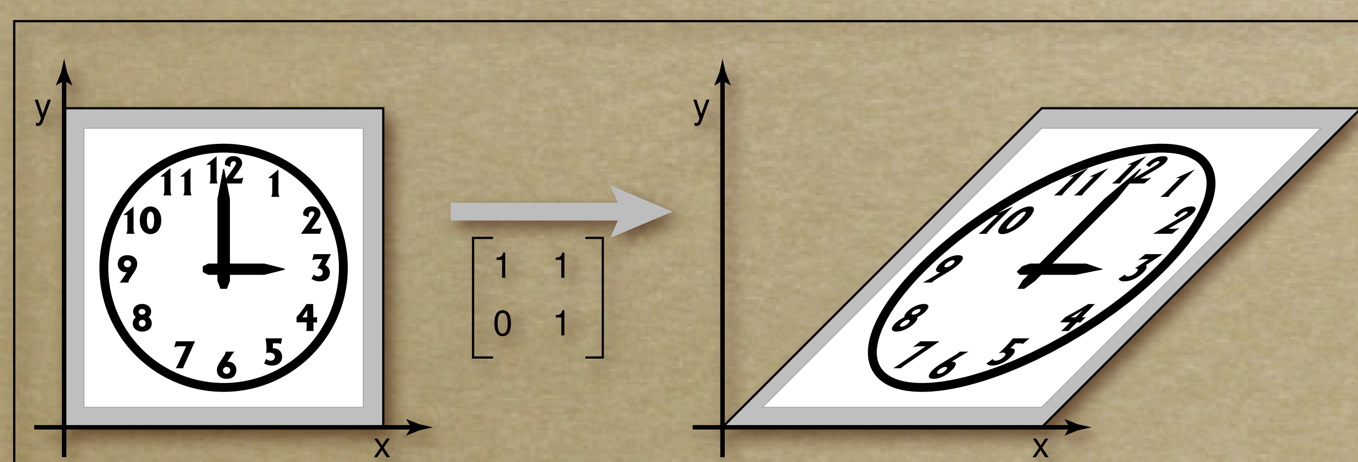


Shears



Shear

$$\mathbf{p}' = \begin{bmatrix} 1 & H_{yx} \\ H_{xy} & 1 \end{bmatrix} \mathbf{p}$$

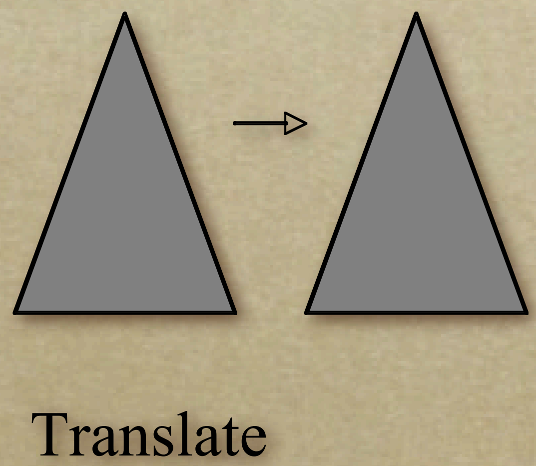


Shears

- Shears are not really primitive transforms
- Related to non-axis-aligned scales
- More shortly.....

Translation

- This is the not-so-useful way:



$$\mathbf{p}' = \mathbf{p} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

Note that its not like the others.

Arbitrary Matrices

- For everything but translations we have:

$$\mathbf{x}' = \mathbf{A} \cdot \mathbf{x}$$

- Soon, translations will be assimilated as well

- What does an arbitrary matrix mean?

Singular Value Decomposition

- For any matrix, $\mathbf{A}$, we can write SVD:

$$\mathbf{A} = \mathbf{QSR}^{T}$$

where $\mathbf{Q}$ and $\mathbf{R}$ are orthonormal and $\mathbf{S}$ is diagonal

- Can also write Polar Decomposition

$$\mathbf{A} = \mathbf{QRSR}^{T}$$

not the same $\mathbf{Q}$

where $\mathbf{Q}$ is still orthonormal

Decomposing Matrices

- We can force **Q** and **R** to have Det=1 so they are rotations
- Any matrix is now:
  - Rotation:Rotation:Scale:Rotation
  - See, shear is just a mix of rotations and scales

Composition

- Matrix multiplication composites matrices

$$\mathbf{p}' = \mathbf{B}\mathbf{A}\mathbf{p}$$

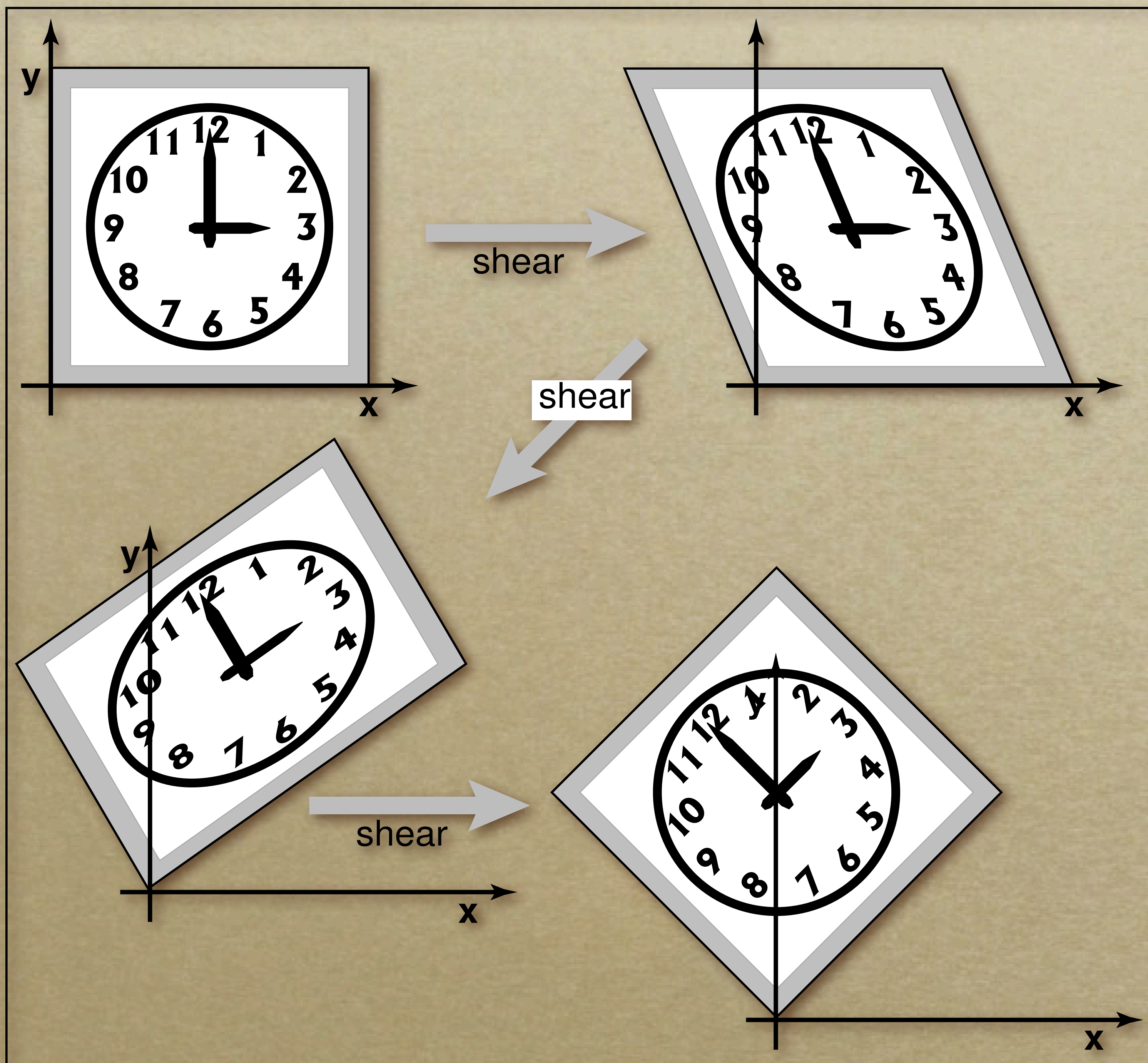
"Apply $\mathbf{A}$ to $\mathbf{p}$ and then apply $\mathbf{B}$ to that result."

$$\mathbf{p}' = \mathbf{B}(\mathbf{A}\mathbf{p}) = (\mathbf{B}\mathbf{A})\mathbf{p} = \mathbf{C}\mathbf{p}$$

- Several translations composted to one
- Translations still left out...

$$\mathbf{p}' = \mathbf{B}(\mathbf{A}\mathbf{p} + \mathbf{t}) = \cancel{\mathbf{B}\mathbf{A}}\mathbf{p} + \mathbf{B}\mathbf{t} = \mathbf{C}\mathbf{p} + \mathbf{u}$$

Composition



Transformations built up from others

SVD builds from scale and rotations

Also build other ways

i.e. 45 deg rotation built from shears

Homogeneous Coordiantes

- Move to one higher dimensional space
  - Append a 1 at the end of the vectors

$$\mathbf{p} = \begin{bmatrix} p_x \\ p_y \end{bmatrix} \qquad \tilde{\mathbf{p}} = \begin{bmatrix} p_x \\ p_y \\ 1 \end{bmatrix}$$

  - For *directions* the extra coordinate is a zero

Homogeneous Translation

$$\tilde{\mathbf{p}}' = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ 1 \end{bmatrix}$$

$$\tilde{\mathbf{p}}' = \tilde{\mathbf{A}}\tilde{\mathbf{p}}$$

The tildes are for clarity to distinguish homogenized from non-homogenized vectors.

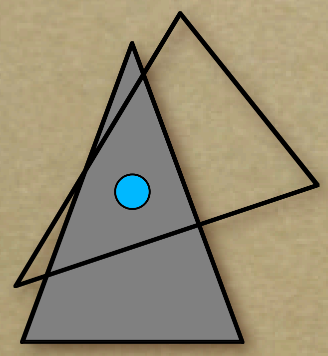
Homogeneous Others

$$\widetilde{\mathbf{A}} = \begin{bmatrix} & \mathbf{A} & \begin{matrix} 0 \\ 0 \end{matrix} \\ 0 & 0 & 1 \end{bmatrix}$$

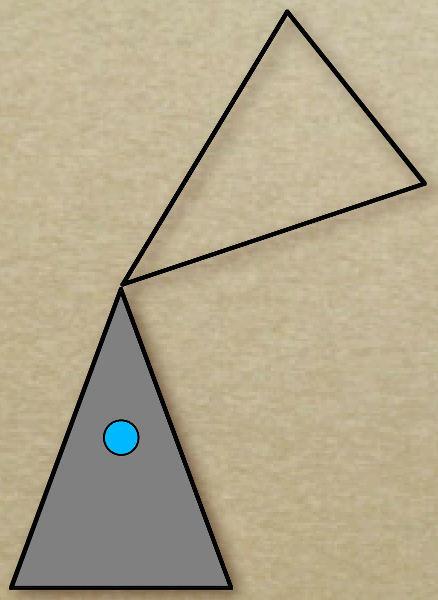
Now everything looks the same...
Hence the term "homogenized!"

Compositing Matrices

- Rotations and scales always about the origin
- How to rotate/scale about another point?

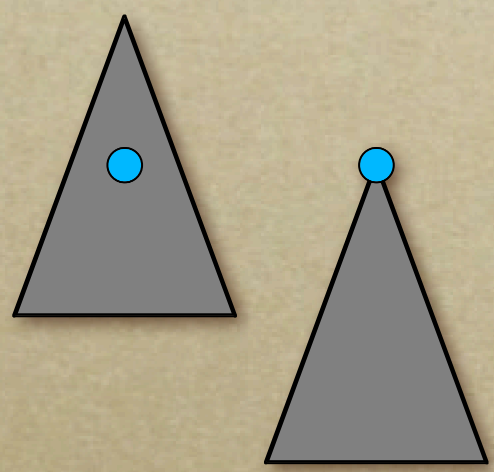


-vs-



Rotate About Arb. Point

- Step 1: Translate point to origin



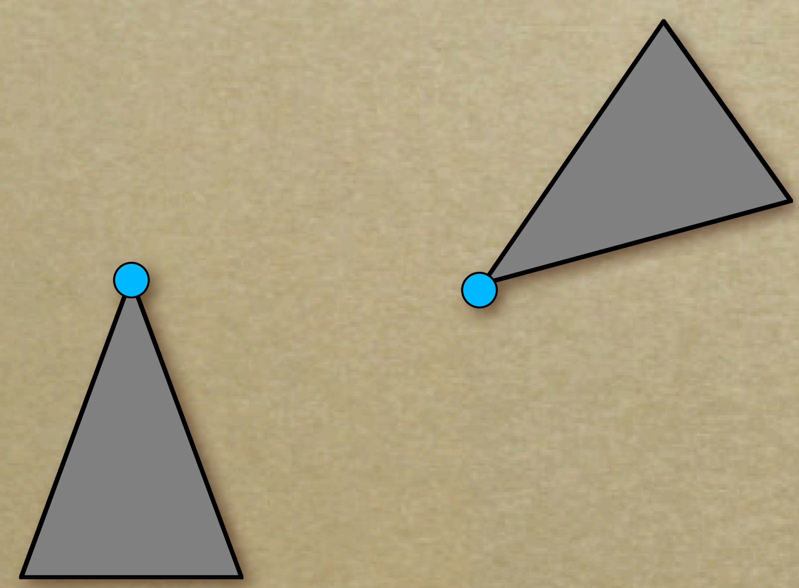
Translate (-**C**)

Rotate About Arb. Point

- Step 1: Translate point to origin
- Step 2: Rotate as desired

Translate (-**C**)

Rotate (θ)



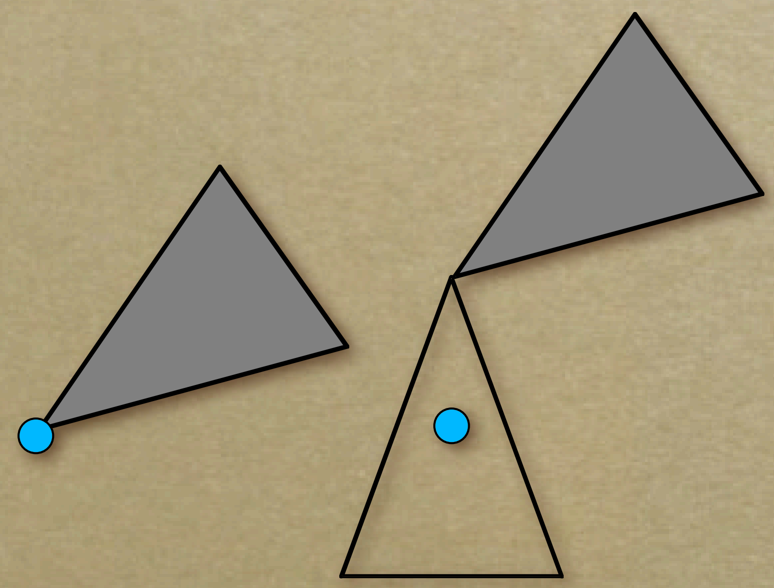
Rotate About Arb. Point

- Step 1: Translate point to origin
- Step 2: Rotate as desired
- Step 3: Put back where it was

Translate (-**C**)

Rotate (θ)

Translate (**C**)

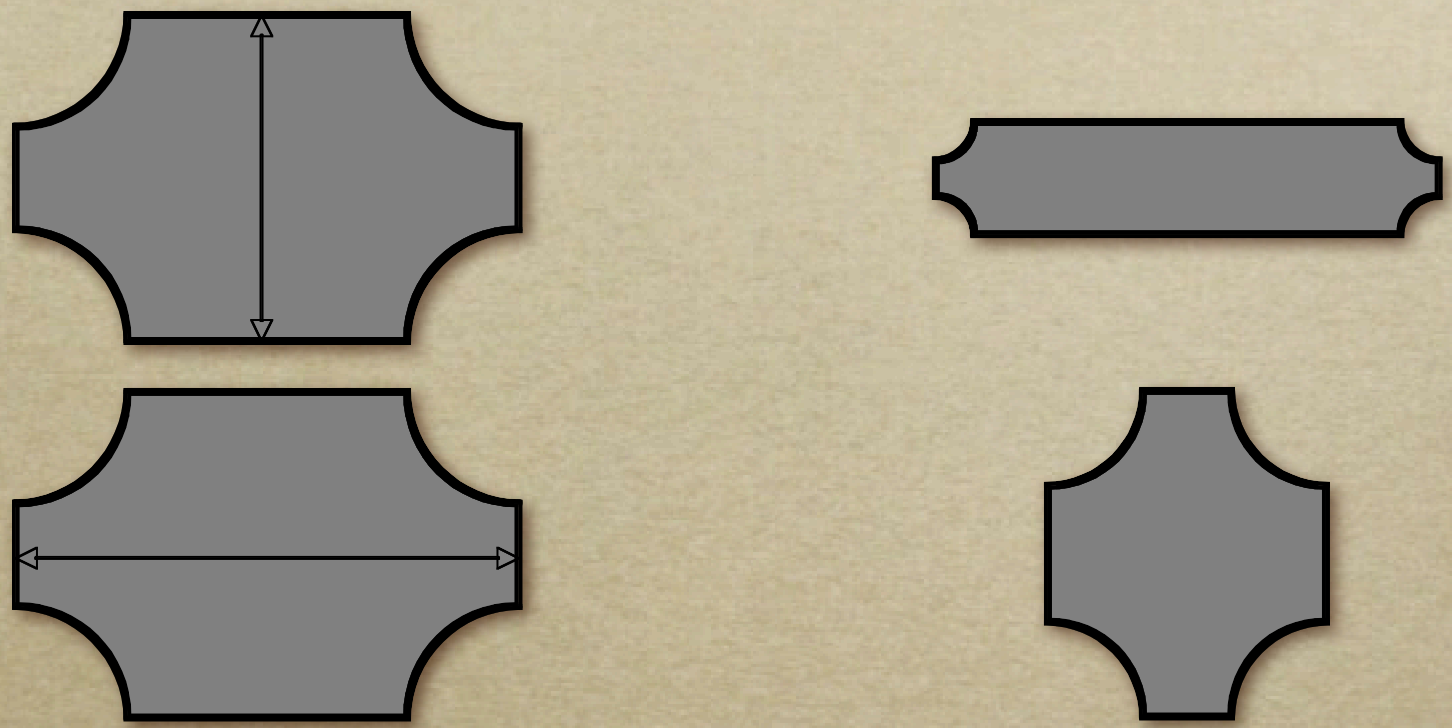


$$\tilde{\mathbf{p}}' = (-\mathbf{T})\mathbf{R}\mathbf{T}\tilde{\mathbf{p}} = \mathbf{A}\tilde{\mathbf{p}}$$

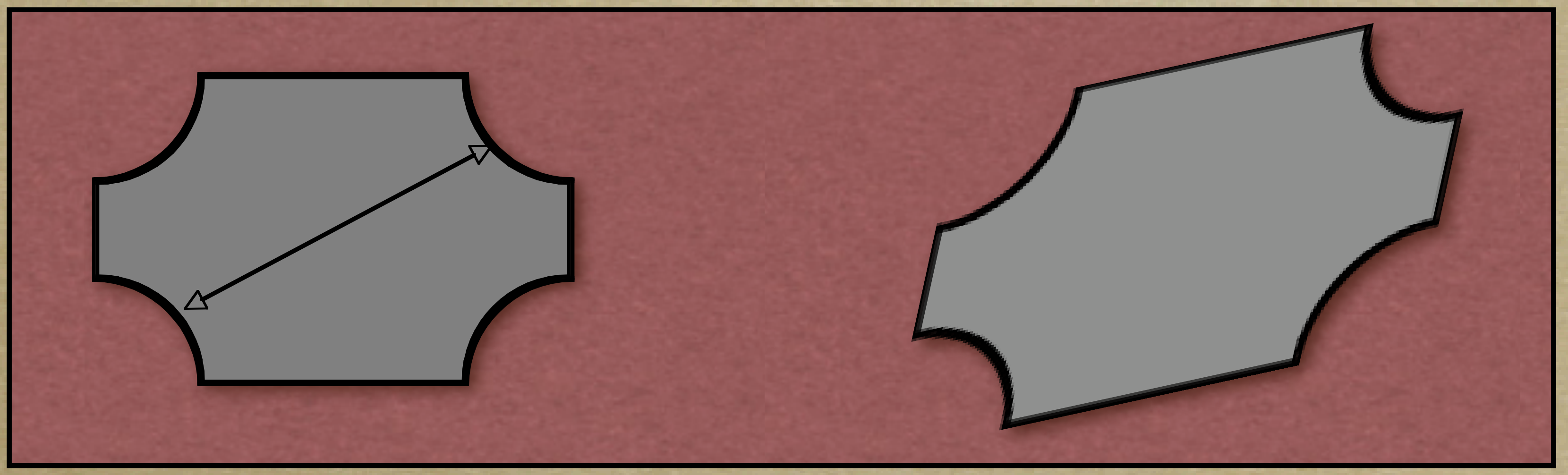
Don't negate the 1...

Scale About Arb. Axis

- Diagonal matrices scale about coordinate axes only:

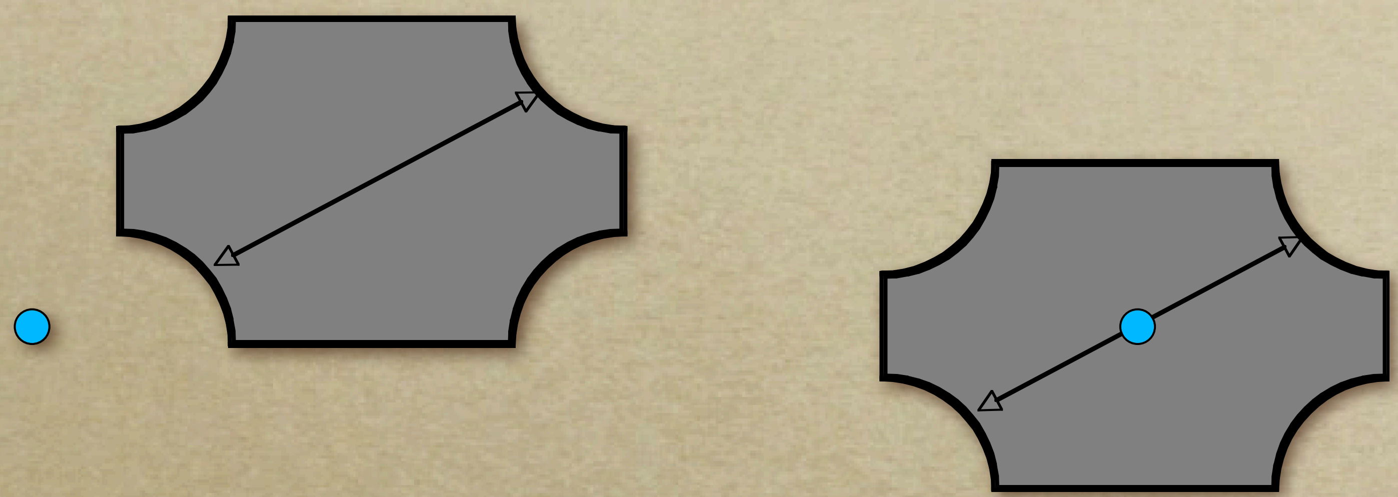


Not axis-aligned



Scale About Arb. Axis

- Step 1: Translate axis to origin

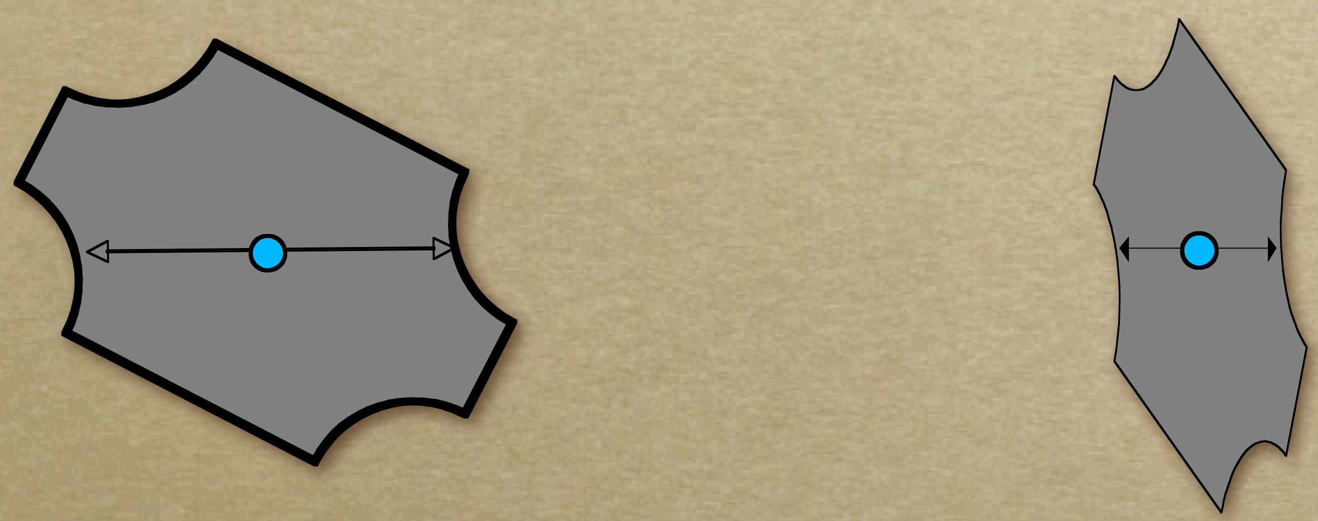


Scale About Arb. Axis

- Step 1: Translate axis to origin
- Step 2: Rotate axis to align with one of the coordinate axes

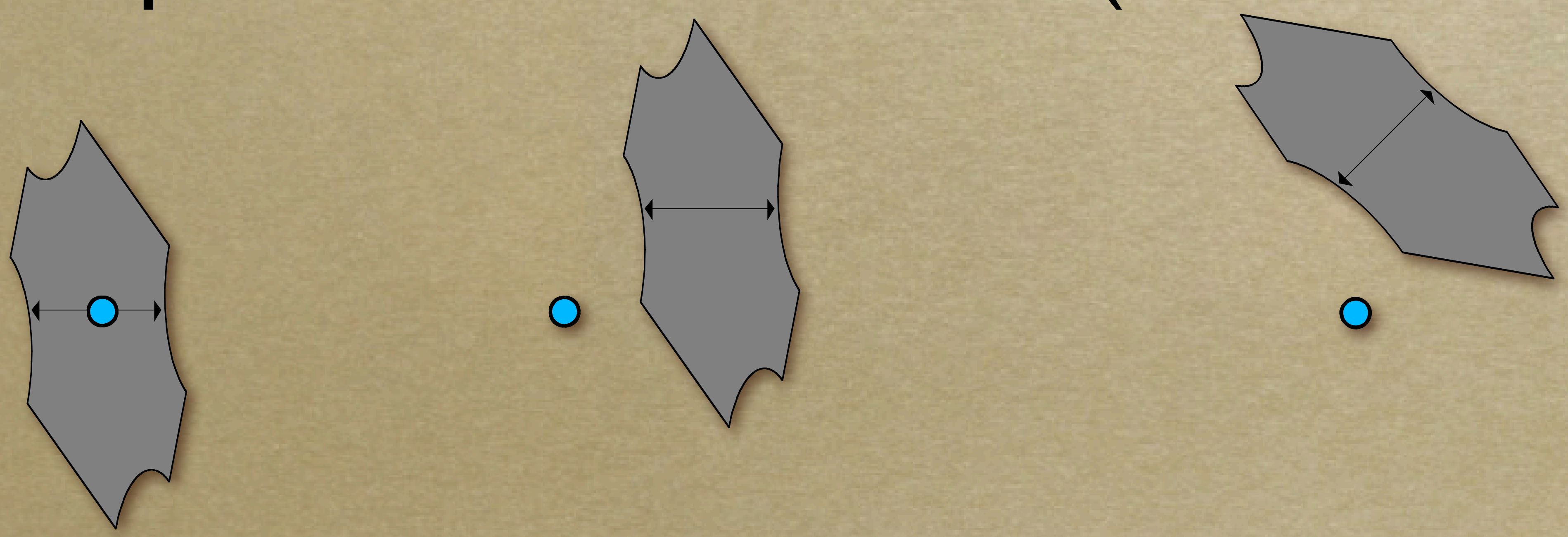
Scale About Arb. Axis

- Step 1: Translate axis to origin
- Step 2: Rotate axis to align with one of the coordinate axes
- Step 3: Scale as desired



Scale About Arb. Axis

- Step 1: Translate axis to origin
- Step 2: Rotate axis to align with one of the coordinate axes
- Step 3: Scale as desired
- Steps 4&5: Undo 2 and 1 (reverse order)



Order Matters!

- The order that matrices appear in matters

$$\mathbf{A} \cdot \mathbf{B} \neq \mathbf{BA}$$

- Some special cases work, but they are special
- But matrices are associative

$$(\mathbf{A} \cdot \mathbf{B}) \cdot \mathbf{C} = \mathbf{A} \cdot (\mathbf{B} \cdot \mathbf{C})$$

- Think about efficiency when you have many points to transform...

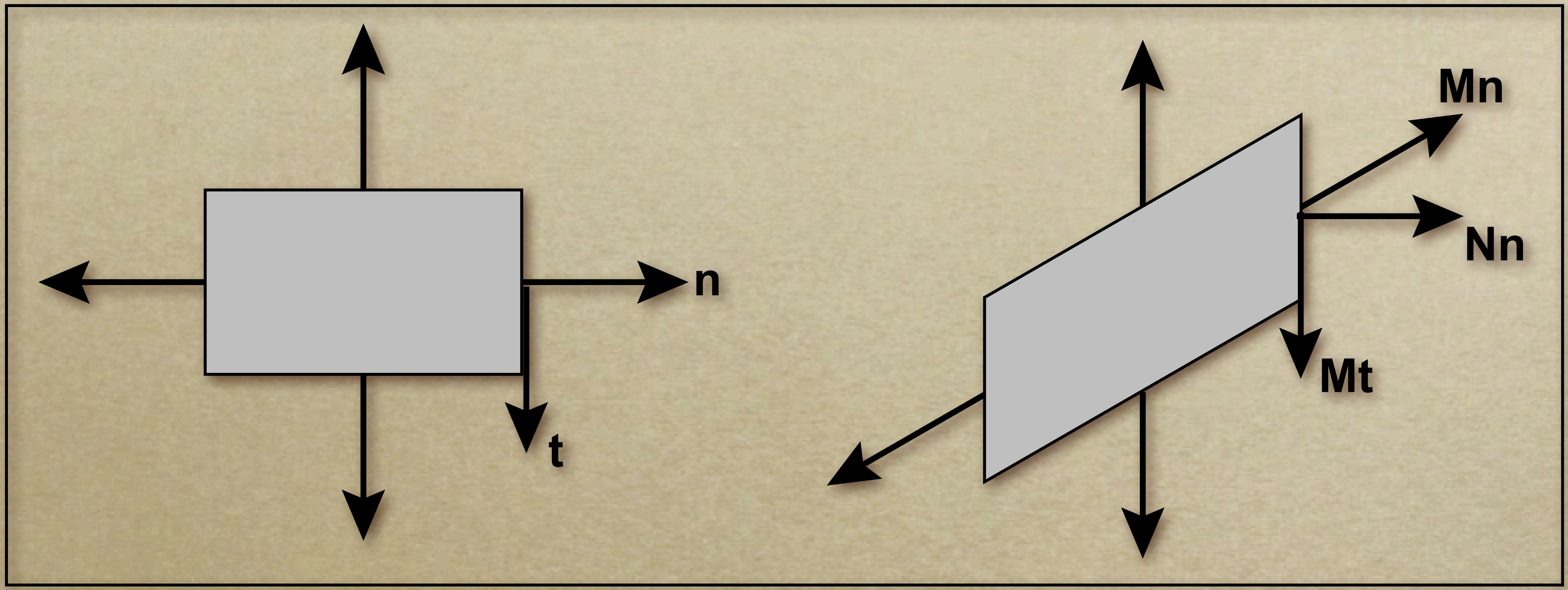
Matrix Inverses

- In general: $\mathbf{A}^{-1}$ undoes effect of $\mathbf{A}$
- Special cases:
  - Translation: negate t_x and t_y
  - Rotation: transpose
  - Scale: invert diagonal (axis-aligned scales)
- Others:
  - Invert matrix
  - Invert SVD matrices

Point Vectors / Direction Vectors

- Points in space have a 1 for the "w" coordinate
- What should we have for $\mathbf{a} - \mathbf{b}$?
  - $w = 0$
  - Directions not the same as positions
  - Difference of positions is a direction
  - Position + direction is a position
  - Direction + direction is a direction
  - Position + position is nonsense

Somethings Require Care



For example normals do not transform normally

$$\mathbf{M}(\mathbf{a} \times \mathbf{b}) \neq (\mathbf{M}\mathbf{a}) \times (\mathbf{M}\mathbf{b})$$

Use inverse transpose of the matrix for normals.
See text book.

Suggested Reading

- Fundamentals of Computer Graphics by Pete Shirley
  - Chapter 5
  - And re-read chapter 4 if your linear algebra is rusty!

CS-184: Computer Graphics

Lecture #5: 3D Transformations and Rotations

Prof. James O'Brien
University of California, Berkeley

V2008-F-05-1.0

Today

- Transformations in 3D
- Rotations
  - Matrices
  - Euler angles
  - Exponential maps
  - Quaternions

3D Transformations

- Generally, the extension from 2D to 3D is straightforward
  - Vectors get longer by one
  - Matrices get extra column and row
  - SVD still works the same way
  - Scale, Translation, and Shear all basically the same
- Rotations get interesting

Translations

$$\tilde{\mathbf{A}} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

For 2D

$$\tilde{\mathbf{A}} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

For 3D

Scales

$$\tilde{\mathbf{A}} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

For 2D

$$\tilde{\mathbf{A}} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

For 3D

(Axis-aligned!)

Shears

$$\tilde{\mathbf{A}} = \begin{bmatrix} 1 & h_{xy} & 0 \\ h_{yx} & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

For 2D

$$\tilde{\mathbf{A}} = \begin{bmatrix} 1 & h_{xy} & h_{xz} & 0 \\ h_{yx} & 1 & h_{yz} & 0 \\ h_{zx} & h_{zy} & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

For 3D

(Axis-aligned!)

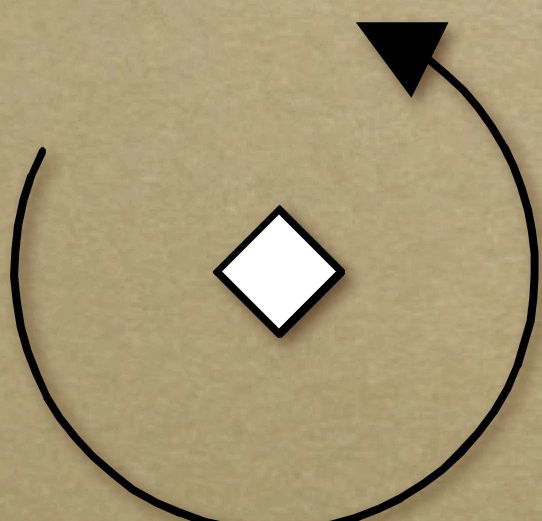
Shears

$$\tilde{\mathbf{A}} = \begin{bmatrix} 1 & h_{xy} & h_{xz} & 0 \\ h_{yx} & 1 & h_{yz} & 0 \\ h_{zx} & h_{zy} & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Shears y into x

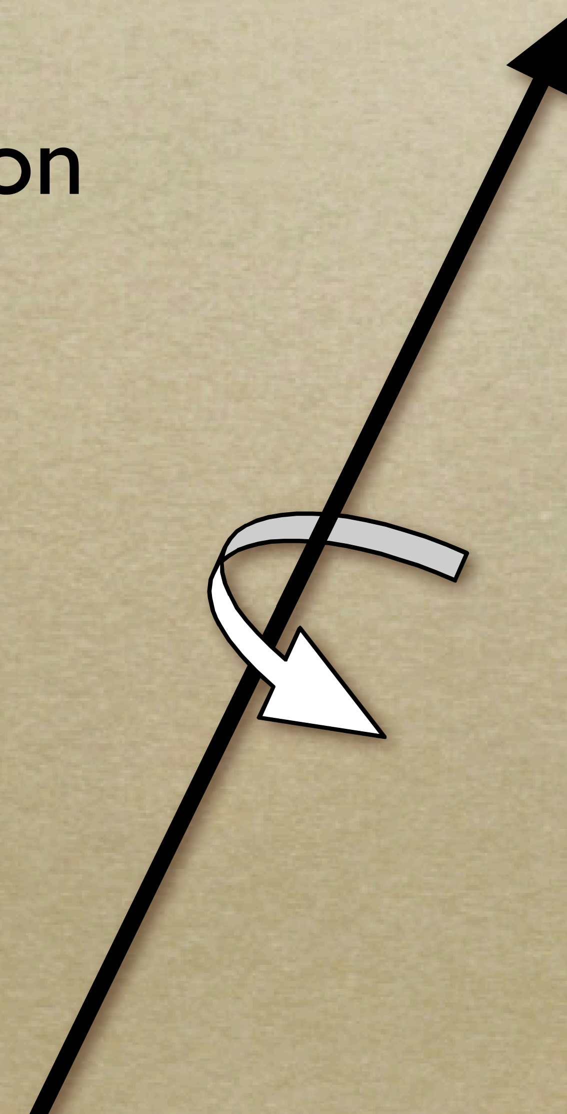
Rotations

- 3D Rotations fundamentally more complex than in 2D
  - 2D: amount of rotation
  - 3D: amount and axis of rotation



2D

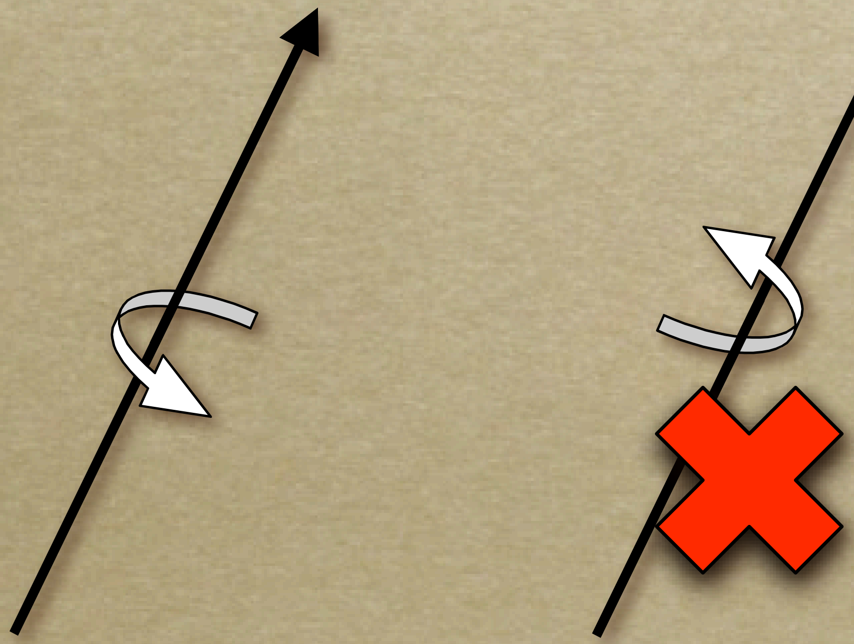
-vs-



3D

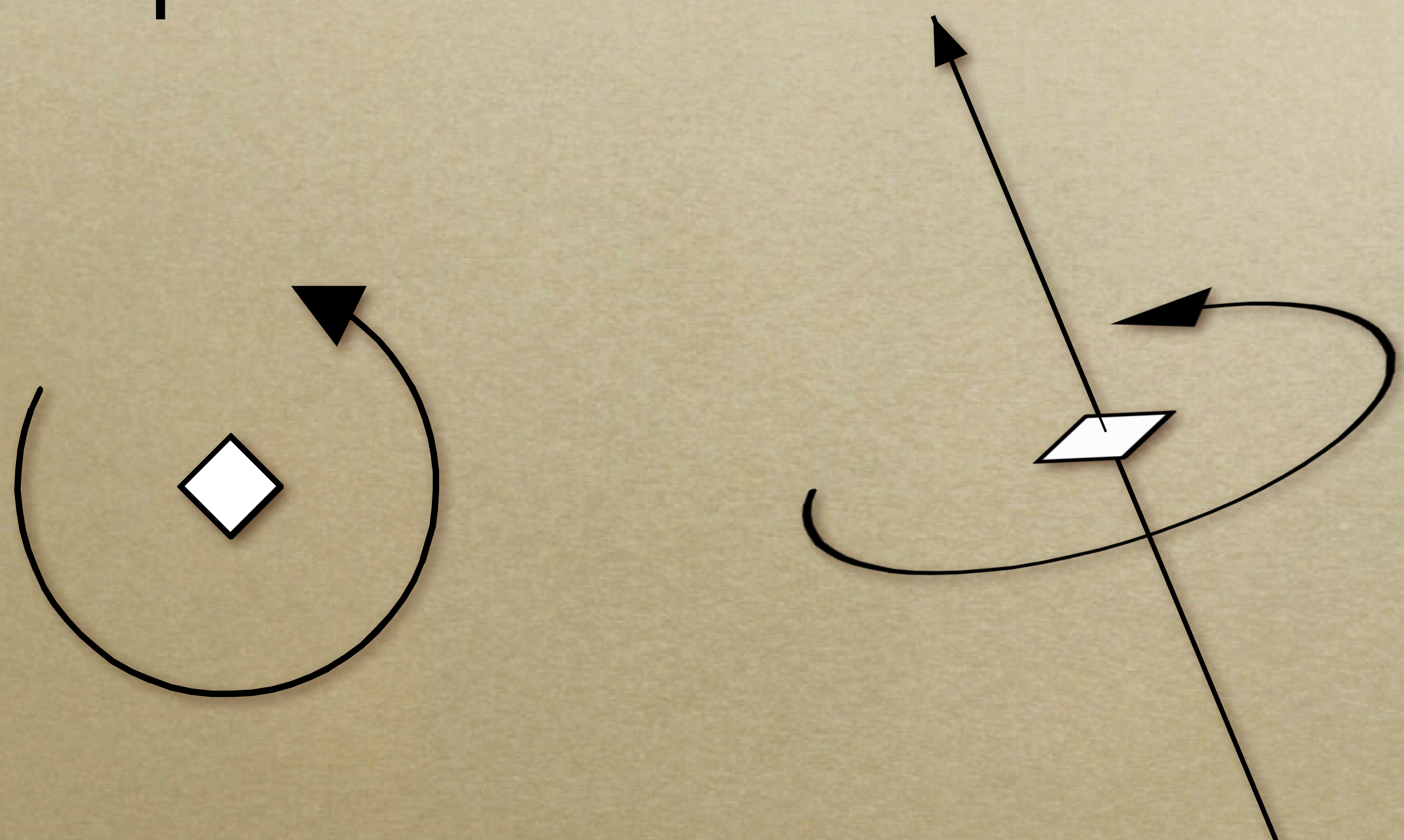
Rotations

- Rotations still orthonormal
- $\mathrm{Det}(\mathbf{R}) = 1 \neq -1$
- Preserve lengths and distance to origin
- 3D rotations **DO NOT COMMUTE!**
- Right-hand rule
- Unique matrices



Axis-aligned 3D Rotations

- 2D rotations implicitly rotate about a third out of plane axis

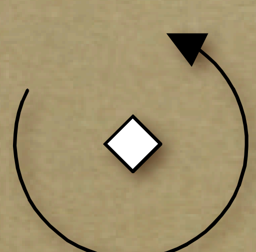


Axis-aligned 3D Rotations

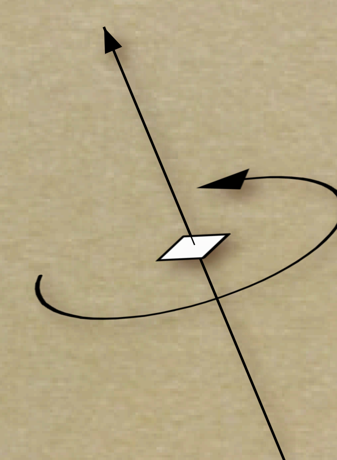
- 2D rotations implicitly rotate about a third out of plane axis

$$\mathbf{R} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}$$

$$\mathbf{R} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



Note: looks same as $\tilde{\mathbf{R}}$



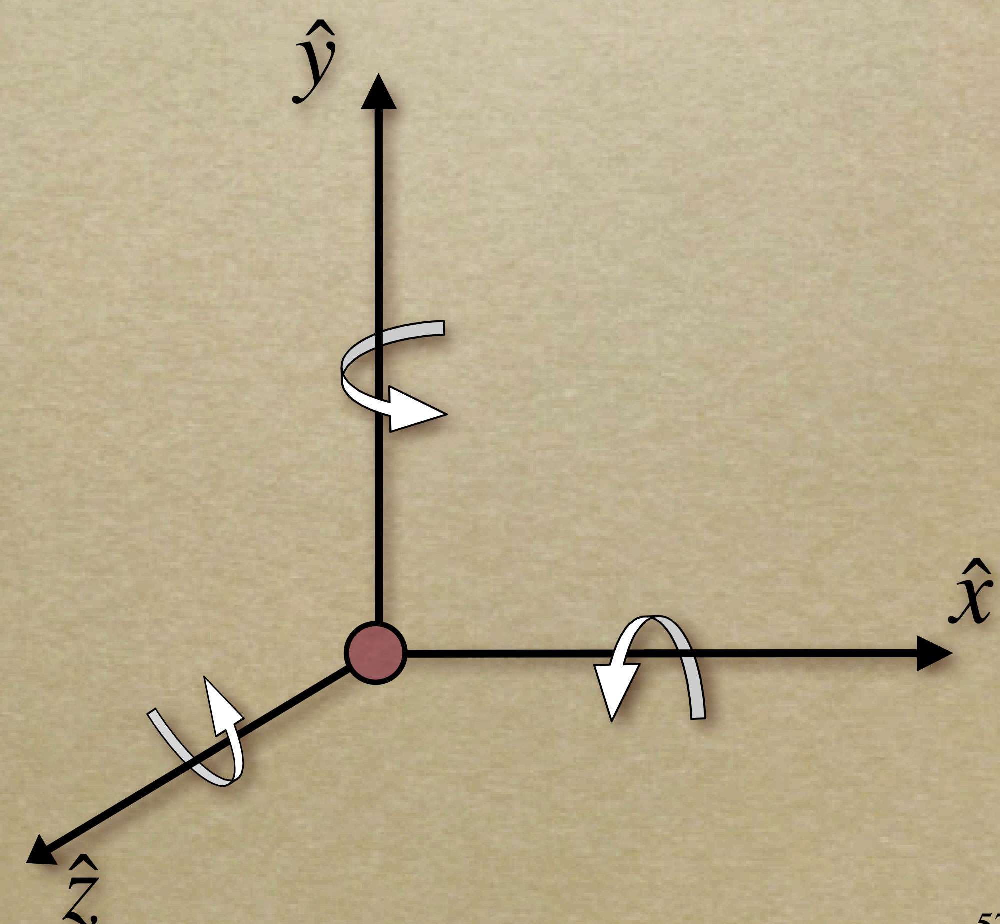
Axis-aligned 3D Rotations

$$\mathbf{R}_{\hat{x}} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) \\ 0 & \sin(\theta) & \cos(\theta) \end{bmatrix}$$

$$\mathbf{R}_{\hat{y}} = \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{bmatrix}$$

$$\mathbf{R}_{\hat{z}} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

"Z is in your face"



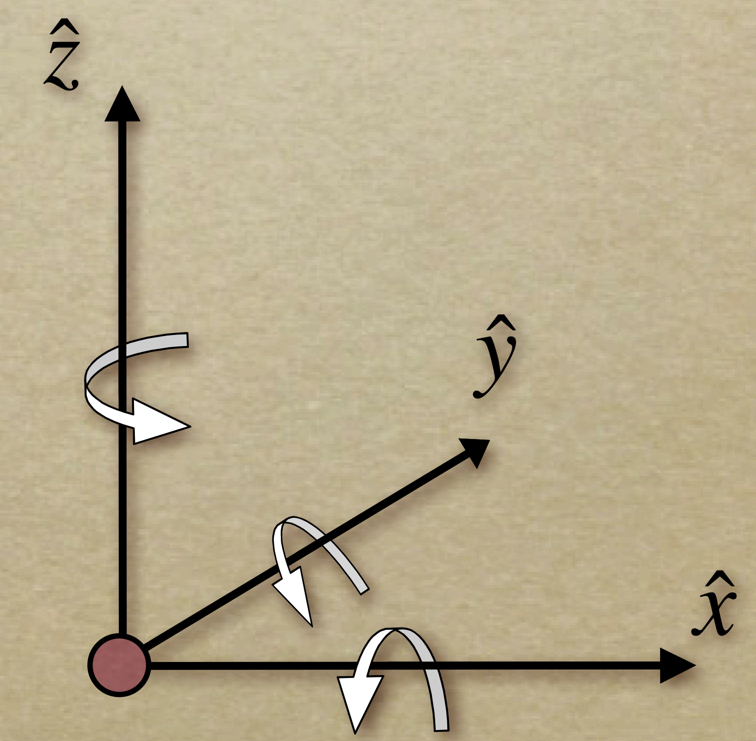
Axis-aligned 3D Rotations

$$\mathbf{R}_{\hat{x}} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) \\ 0 & \sin(\theta) & \cos(\theta) \end{bmatrix}$$

$$\mathbf{R}_{\hat{y}} = \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{bmatrix}$$

$$\mathbf{R}_{\hat{z}} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Also right handed "Zup"



Axis-aligned 3D Rotations

- Also known as "direction-cosine" matrices

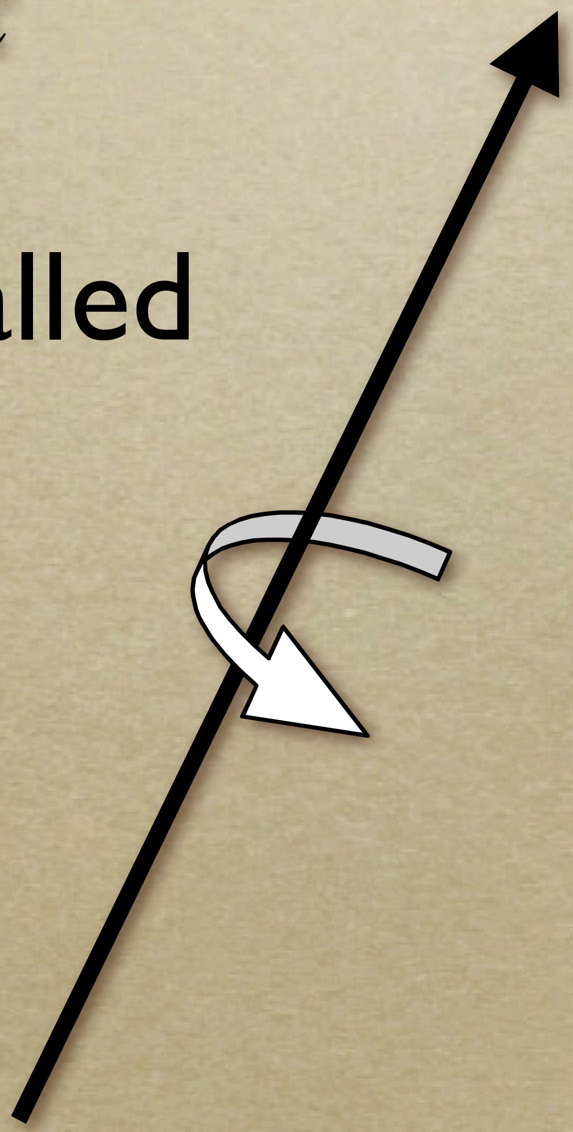
$$\mathbf{R}_{\hat{x}} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) \\ 0 & \sin(\theta) & \cos(\theta) \end{bmatrix} \qquad \mathbf{R}_{\hat{y}} = \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{bmatrix}$$

$$\mathbf{R}_{\hat{z}} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Arbitrary Rotations

- Can be built from axis-aligned matrices:

$$\mathbf{R} = \mathbf{R}_{\hat{z}} \cdot \mathbf{R}_{\hat{y}} \cdot \mathbf{R}_{\hat{x}}$$



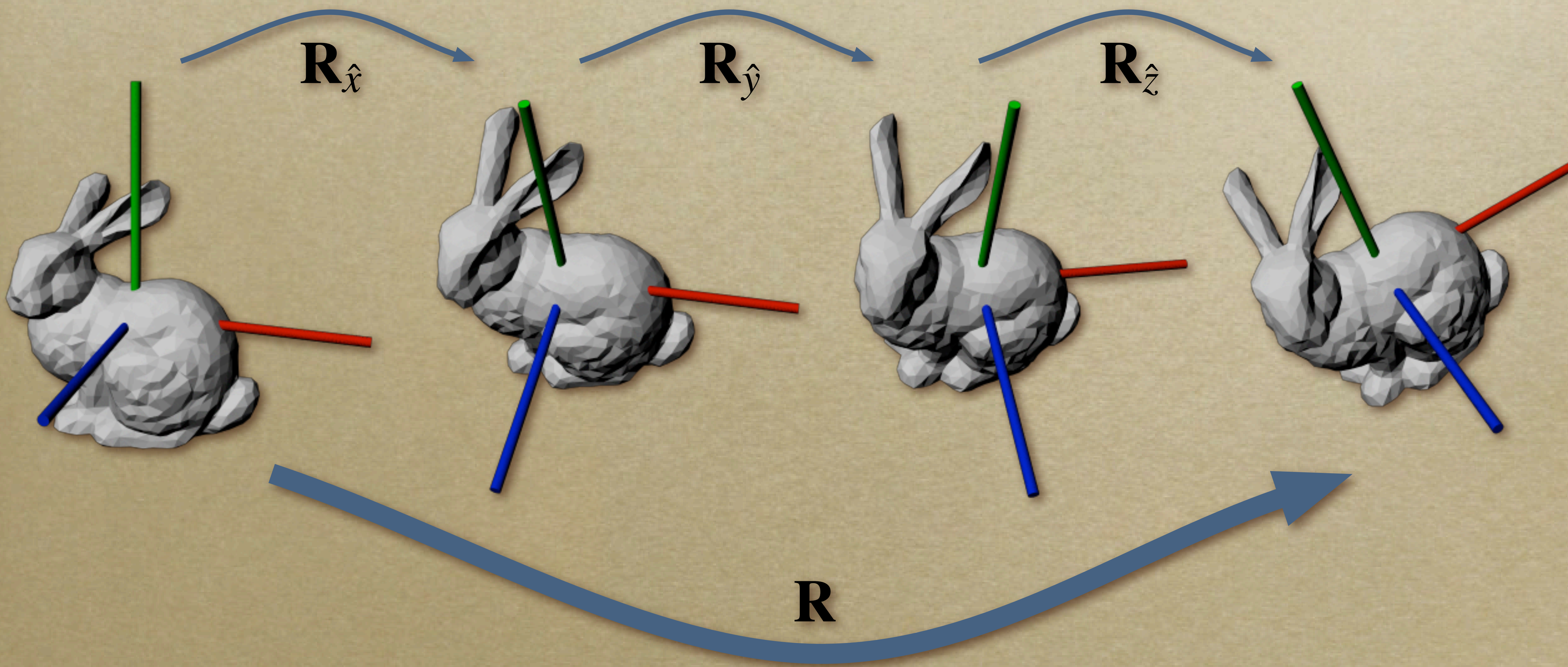
- Result due to Euler... hence called
  Euler Angles
- Easy to store in vector

$$\mathbf{R} = \mathrm{rot}(x, y, z)$$

- But NOT a vector.

Arbitrary Rotations

$$\mathbf{R} = \mathbf{R}_{\hat{z}} \cdot \mathbf{R}_{\hat{y}} \cdot \mathbf{R}_{\hat{x}}$$



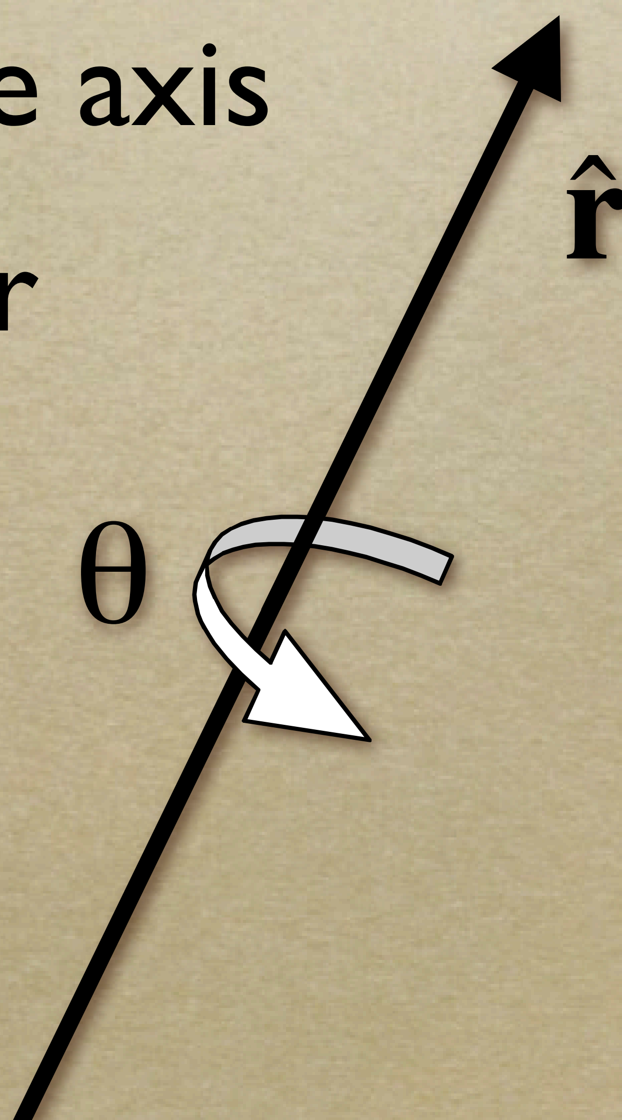
Arbitrary Rotations

- Allows tumbling
- Euler angles are non-unique
- Gimbal-lock
- Moving -vs- fixed axes
  - Reverse of each other

Exponential Maps

- Direct representation of arbitrary rotation
- AKA: axis-angle, angular displacement vector
- Rotate θ degrees about some axis
- Encode θ by length of vector

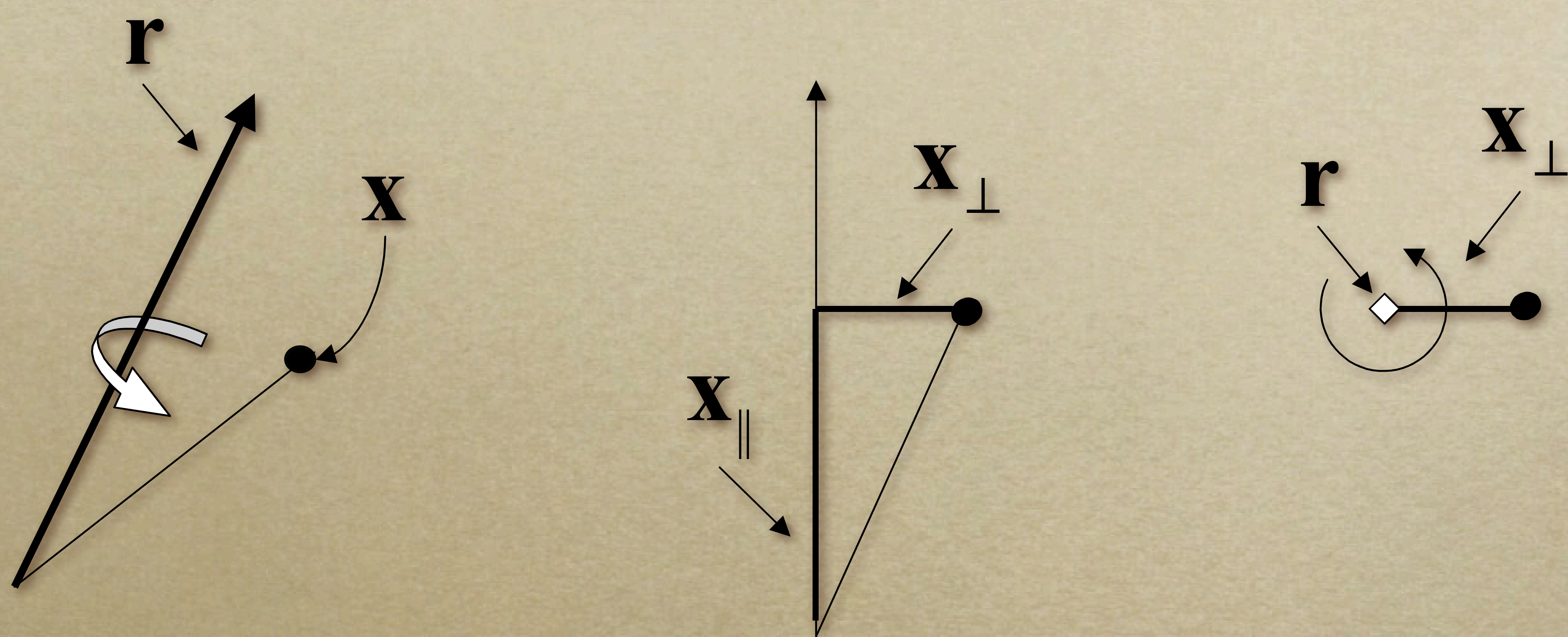
$$\theta = |\mathbf{r}|$$



Exponential Maps

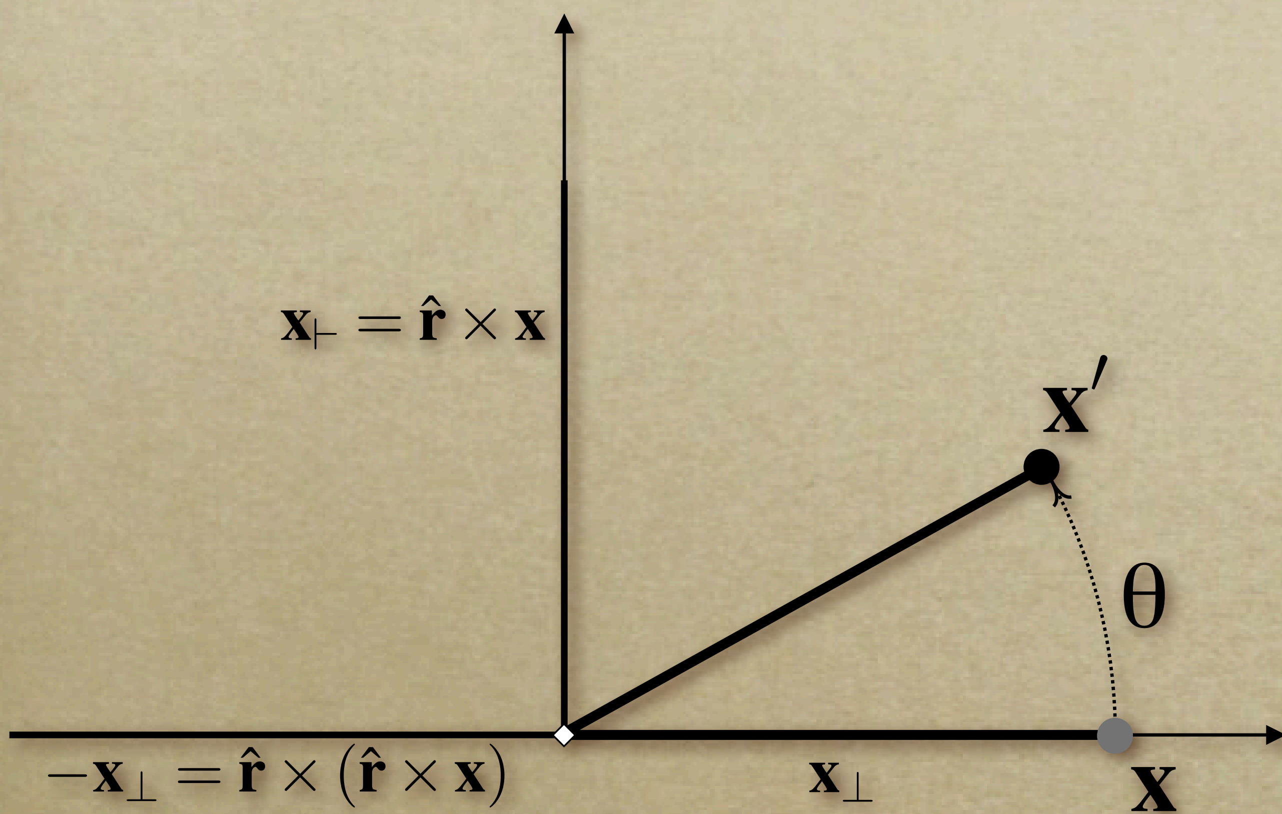
- Given vector $\mathbf{r}$, how to get matrix $\mathbf{R}$
- Method from text:
  1. rotate about x axis to put $\mathbf{r}$ into the x-y plane
  2. rotate about z axis align $\mathbf{r}$ with the x axis
  3. rotate θ degrees about x axis
  4. undo #2 and then #1
  5. composite together

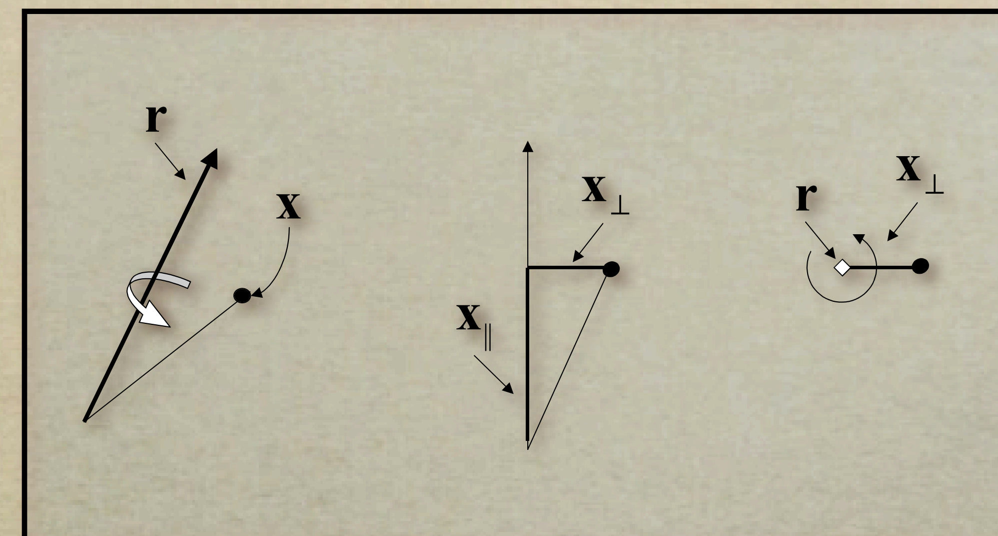
Exponential Maps



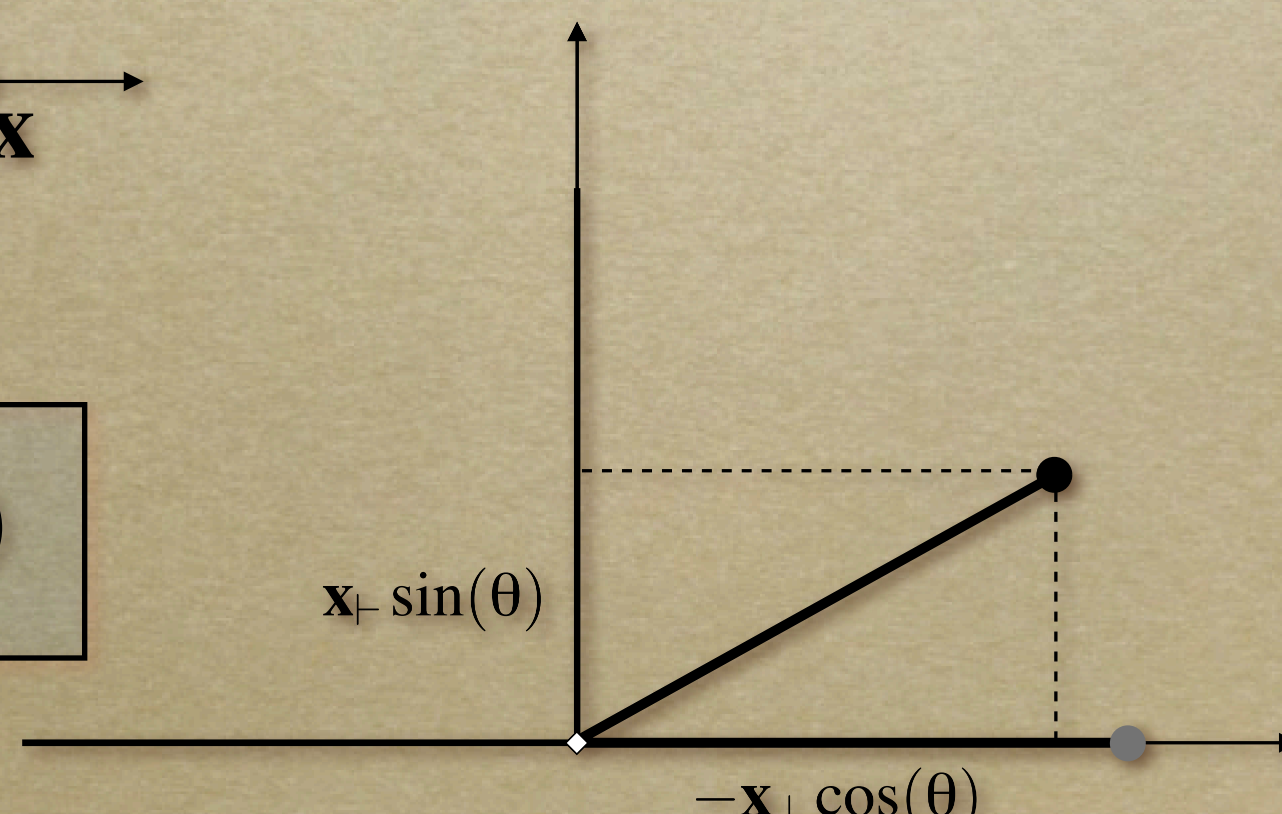
- Vector expressing a point has two parts
  - $\mathbf{x}_\parallel$ does not change
  - $\mathbf{x}_\perp$ rotates like a 2D point

Exponential Maps





$$\mathbf{x}' = \mathbf{x}_{||} + \mathbf{x}_{\vdash}\sin(\theta) + \mathbf{x}_{\perp}\cos(\theta)$$

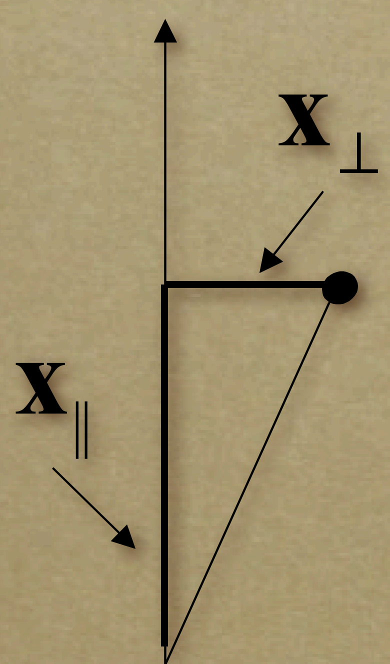


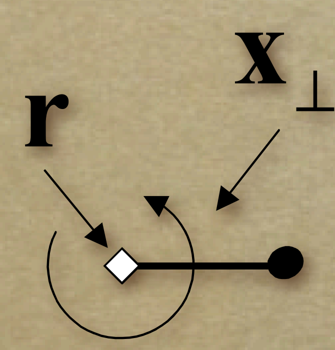
Exponential Maps

- Rodriguez Formula

$$
\begin{aligned}
\mathbf{x}' = \hat{\mathbf{r}}(\hat{\mathbf{r}} \cdot \mathbf{x}) \\
+\sin(\theta)(\hat{\mathbf{r}} \times \mathbf{x}) \\
-\cos(\theta)(\hat{\mathbf{r}} \times (\hat{\mathbf{r}} \times \mathbf{x}))
\end{aligned}
$$

Linear in $\mathbf{x}$





Actually a minor variation ...

Exponential Maps

- Building the matrix

$$\mathbf{x}' = ((\hat{\mathbf{r}}\hat{\mathbf{r}}^{\mathrm{t}}) + \sin(\theta)(\hat{\mathbf{r}}\times) - \cos(\theta)(\hat{\mathbf{r}}\times)(\hat{\mathbf{r}}\times))\,\mathbf{x}$$

$$(\hat{\mathbf{r}}\times) = \begin{bmatrix} 0 & -\hat{r}_z & \hat{r}_y \\ \hat{r}_z & 0 & -\hat{r}_x \\ -\hat{r}_y & \hat{r}_x & 0 \end{bmatrix}$$

Antisymmetric matrix

$(\mathbf{a}\times)\mathbf{b} = \mathbf{a} \times \mathbf{b}$

Easy to verify by expansion

Exponential Maps

- Allows tumbling
- No gimbal-lock!
- Orientations are space within π-radius ball
- Nearly unique representation
- Singularities on shells at 2π
- Nice for interpolation

Exponential Maps

- Why exponential?
- Recall series expansion of e^x

$$e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots$$

Exponential Maps

- Why exponential?
- Recall series expansion of e^x
- Euler: what happens if you put in $i\theta$ for x

$$e^{i\theta} = 1 + \frac{i\theta}{1!} + \frac{-\theta^2}{2!} + \frac{-i\theta^3}{3!} + \frac{\theta^4}{4!} + \cdots$$

$$= \left(1 + \frac{-\theta^2}{2!} + \frac{\theta^4}{4!} + \cdots\right) + i\left(\frac{\theta}{1!} + \frac{-\theta^3}{3!} + \cdots\right)$$

$$= \cos(\theta) + i\sin(\theta)$$

Exponential Maps

- Why exponential?

$$e^{(\hat{\mathbf{r}}\times)\theta} = \mathbf{I} + \frac{(\hat{\mathbf{r}}\times)\theta}{1!} + \frac{(\hat{\mathbf{r}}\times)^2\theta^2}{2!} + \frac{(\hat{\mathbf{r}}\times)^3\theta^3}{3!} + \frac{(\hat{\mathbf{r}}\times)^4\theta^4}{4!} + \cdots$$

But notice that: $(\hat{\mathbf{r}}\times)^3 = -(\hat{\mathbf{r}}\times)$

$$e^{(\hat{\mathbf{r}}\times)\theta} = \mathbf{I} + \frac{(\hat{\mathbf{r}}\times)\theta}{1!} + \frac{(\hat{\mathbf{r}}\times)^2\theta^2}{2!} + \frac{-(\hat{\mathbf{r}}\times)\theta^3}{3!} + \frac{-(\hat{\mathbf{r}}\times)^2\theta^4}{4!} + \cdots$$

Exponential Maps

$$e^{(\hat{\mathbf{r}}\times)\theta} = \mathbf{I} + \frac{(\hat{\mathbf{r}}\times)\theta}{1!} + \frac{(\hat{\mathbf{r}}\times)^2\theta^2}{2!} + \frac{-(\hat{\mathbf{r}}\times)\theta^3}{3!} + \frac{-(\hat{\mathbf{r}}\times)^2\theta^4}{4!} + \cdots$$

$$e^{(\hat{\mathbf{r}}\times)\theta} = (\hat{\mathbf{r}}\times)\left(\frac{\theta}{1!} - \frac{\theta^3}{3!} + \cdots\right) + \mathbf{I} + (\hat{\mathbf{r}}\times)^2\left(+\frac{\theta^2}{2!} - \frac{\theta^4}{4!} + \cdots\right)$$

$$e^{(\hat{\mathbf{r}}\times)\theta} = (\hat{\mathbf{r}}\times)\sin(\theta) + \mathbf{I} + (\hat{\mathbf{r}}\times)^2(1 - \cos(\theta))$$

Quaternions

- More popular than exponential maps
- Natural extension of $e^{i\theta} = \cos(\theta) + i\sin(\theta)$
- Due to Hamilton (1843)
  - Interesting history
  - Involves “hermaphroditic monsters”

Quaternions

- Uber-Complex Numbers

$$\mathrm{q} = (z_1, z_2, z_3, s) = (\mathbf{z}, s)$$

$$\mathrm{q} = iz_1 + jz_2 + kz_3 + s$$

$$i^2 = j^2 = k^2 = -1$$

$$\begin{array}{ll} ij = k & ji = -k \\ jk = i & kj = -i \\ ki = j & ik = -j \end{array}$$

Quaternions

- Multiplication natural consequence of defn.

$$\mathsf{q} \cdot \mathsf{p} = \left(\mathbf{z}_q s_p + \mathbf{z}_p s_q + \mathbf{z}_p \times \mathbf{z}_q \;\;,\;\; s_p s_q - \mathbf{z}_p \cdot \mathbf{z}_q\right)$$

- Conjugate

$$\mathsf{q}^* = (-\mathbf{z}, s)$$

- Magnitude

$$||\mathsf{q}||^2 = \mathbf{z} \cdot \mathbf{z} + s^2 = \mathsf{q} \cdot \mathsf{q}^*$$

Quaternions

- Vectors as quaternions

$$\mathsf{v} = (\mathbf{v}, 0)$$

- Rotations as quaternions

$$\mathsf{r} = (\hat{\mathbf{r}} \sin\frac{\theta}{2}, \cos\frac{\theta}{2})$$

- Rotating a vector

$$\mathsf{x}' = \mathsf{r} \cdot \mathsf{x} \cdot \mathsf{r}^*$$ Compare to Exp. Map

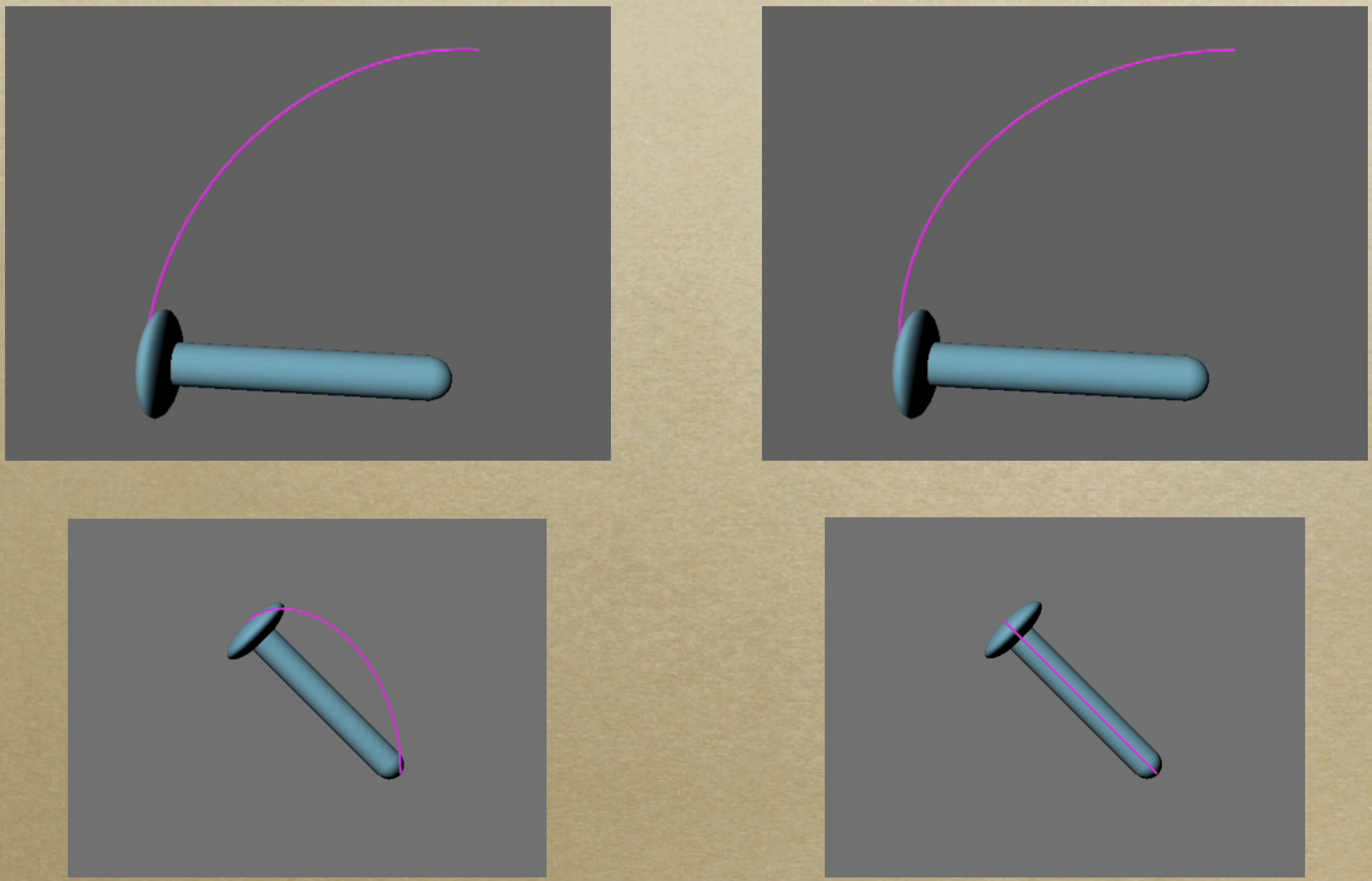
- Composing rotations

$$\mathsf{r} = \mathsf{r}_1 \cdot \mathsf{r}_2$$

Quaternions

- No tumbling
- No gimbal-lock
- Orientations are “double unique”
- Surface of a 3-sphere in 4D $||\mathrm{r}|| = 1$
- Nice for interpolation

Interpolation



Rotation Matrices

- Eigen system
  - One real eigenvalue
  - Real axis is axis of rotation
  - Imaginary values are 2D rotation as complex number
- Logarithmic formula

$$(\hat{\mathbf{r}}\times) = \ln(\mathbf{R}) = \frac{\theta}{2\sin\theta}(\mathbf{R} - \mathbf{R}^{\mathsf{T}})$$

$$\theta = \cos^{-1}\left(\frac{\mathrm{Tr}(\mathbf{R}) - 1}{2}\right)$$

Similar formulae as for exponential...

Rotation Matrices

- Consider:

$$\mathbf{RI} = \begin{bmatrix} r_{xx} & r_{xy} & r_{xz} \\ r_{yx} & r_{yy} & r_{yz} \\ r_{zx} & r_{zy} & r_{zz} \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

- Columns are coordinate axes after transformation (true for general matrices)
- Rows are original axes in original system (not true for general matrices)

Note:

- Rotation stuff in the book is a bit weak... luckily you have these nice slides!

CS-184: Computer Graphics

Lecture #8: Projection

Prof. James O'Brien
University of California, Berkeley

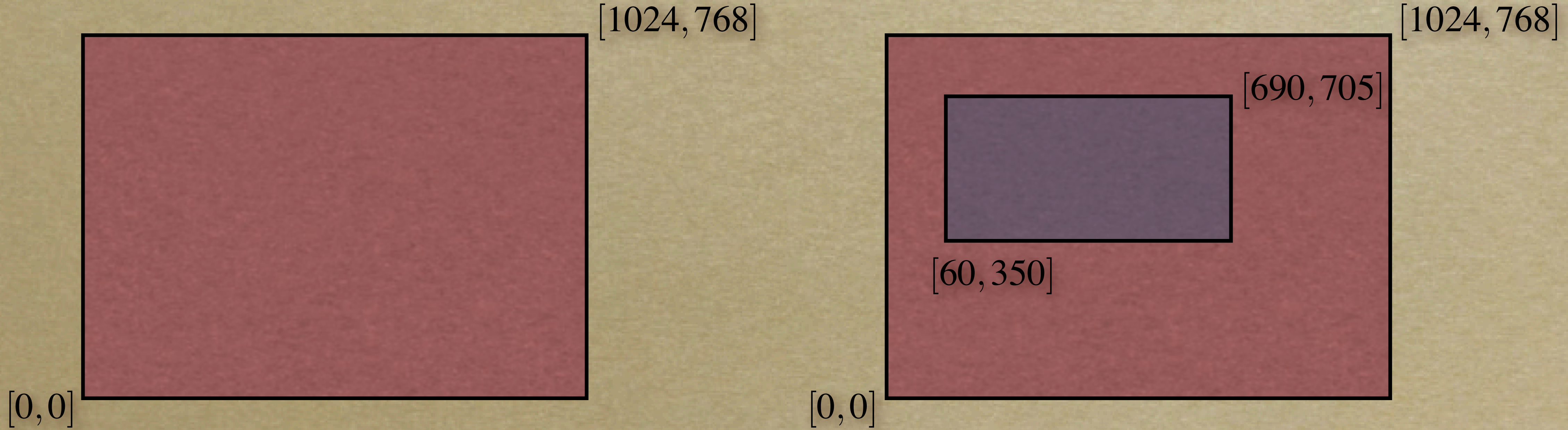
V2008-F-08-1.0

Today

- Windowing and Viewing Transformations
  - Windows and viewports
  - Orthographic projection
  - Perspective projection

Screen Space

- Monitor has some number of pixels
  - e.g. 1024 x 768
- Some sub-region used for given program
  - You call it a window
  - Let's call it a viewport instead



Screen Space

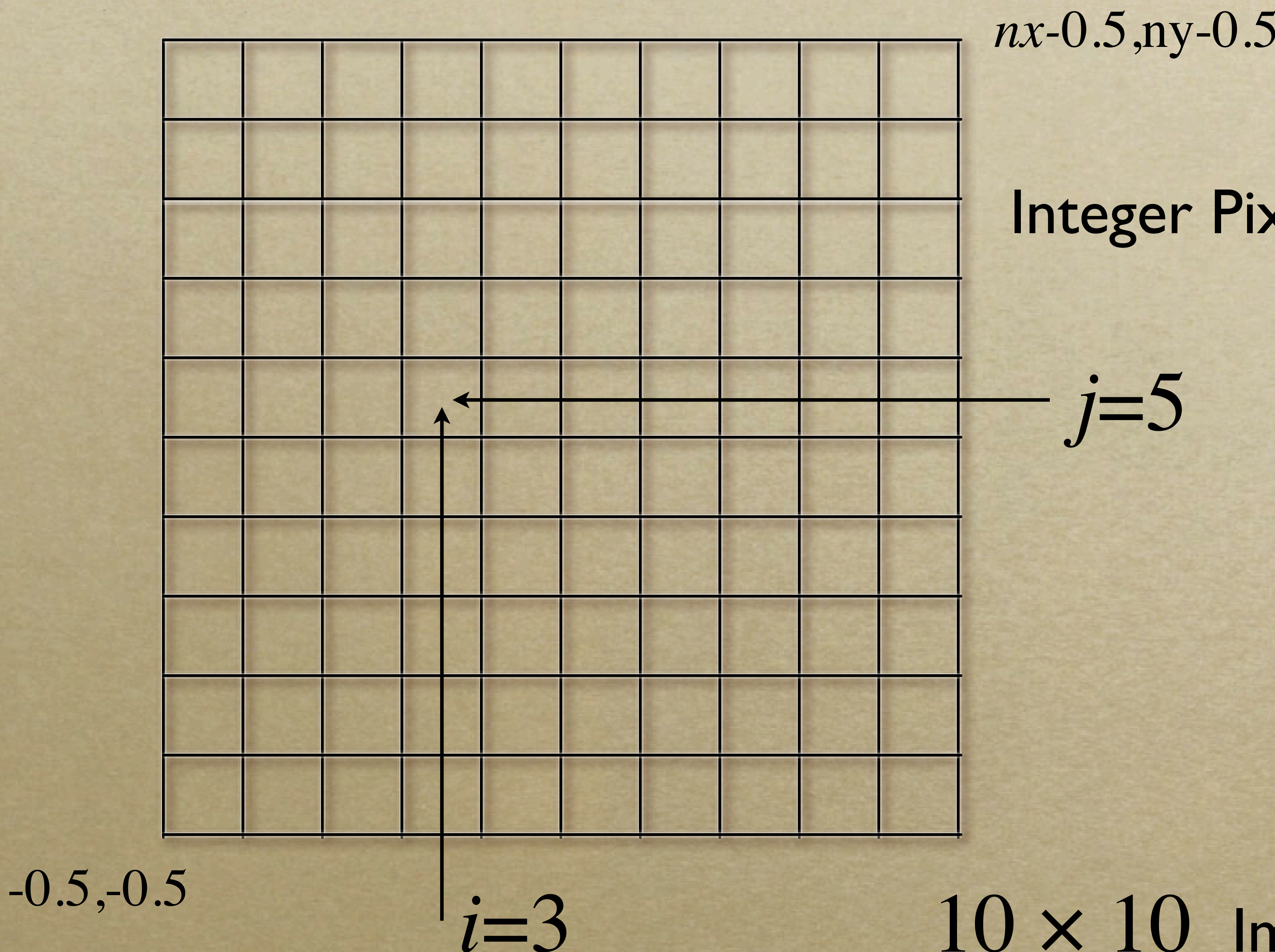
- May not really be a “screen”
  - Image file
  - Printer
  - Other
- Little pixel details
- Sometimes odd
  - Upside down
  - Hexagonal

From Shirley textbook.

Screen Space

- Viewport is somewhere on screen
  - You probably don't care where
  - Window System likely manages this detail
  - Sometimes you care exactly where
- Viewport has a size in pixels
  - Sometimes you care (images, text, *etc*.)
  - Sometimes you don't (using high-level library)

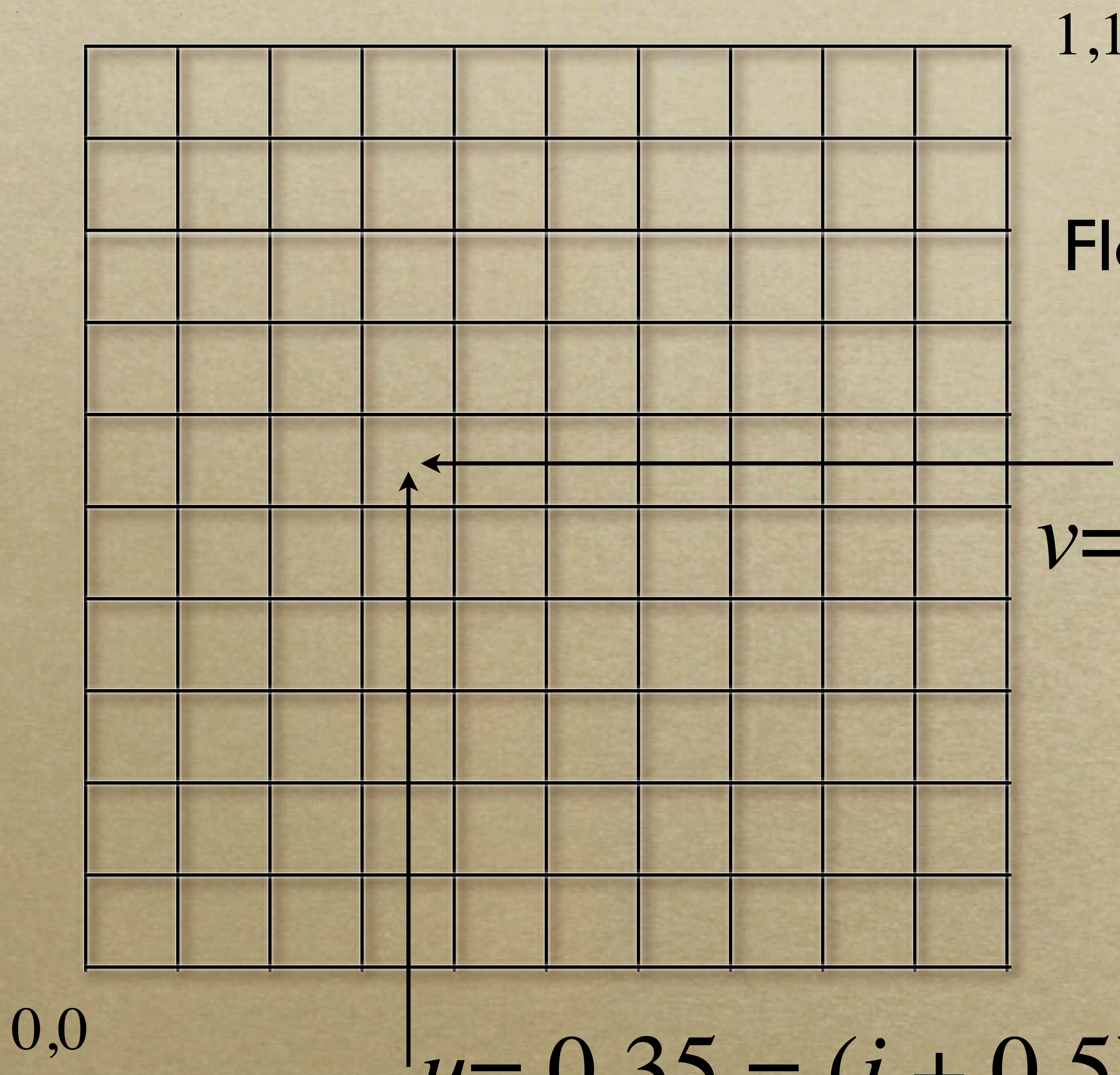
Screen Space



Integer Pixel Addresses

10×10 Image Resolution

Screen Space



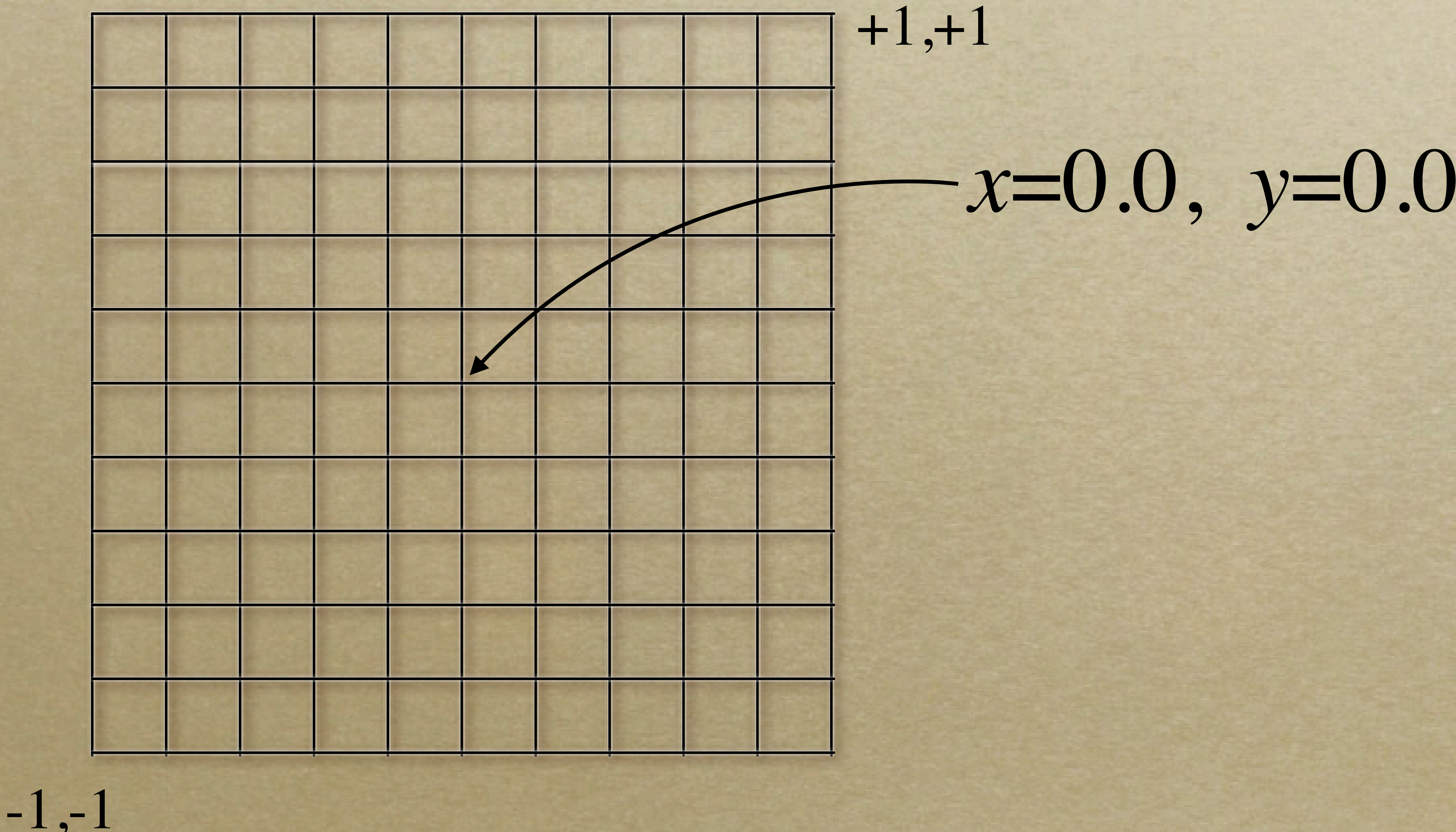
Float Pixel Coordinates

$v = 0.55 = (j + 0.5)/ny$

$u = 0.35 = (i + 0.5)/nx$

Canonical View Space

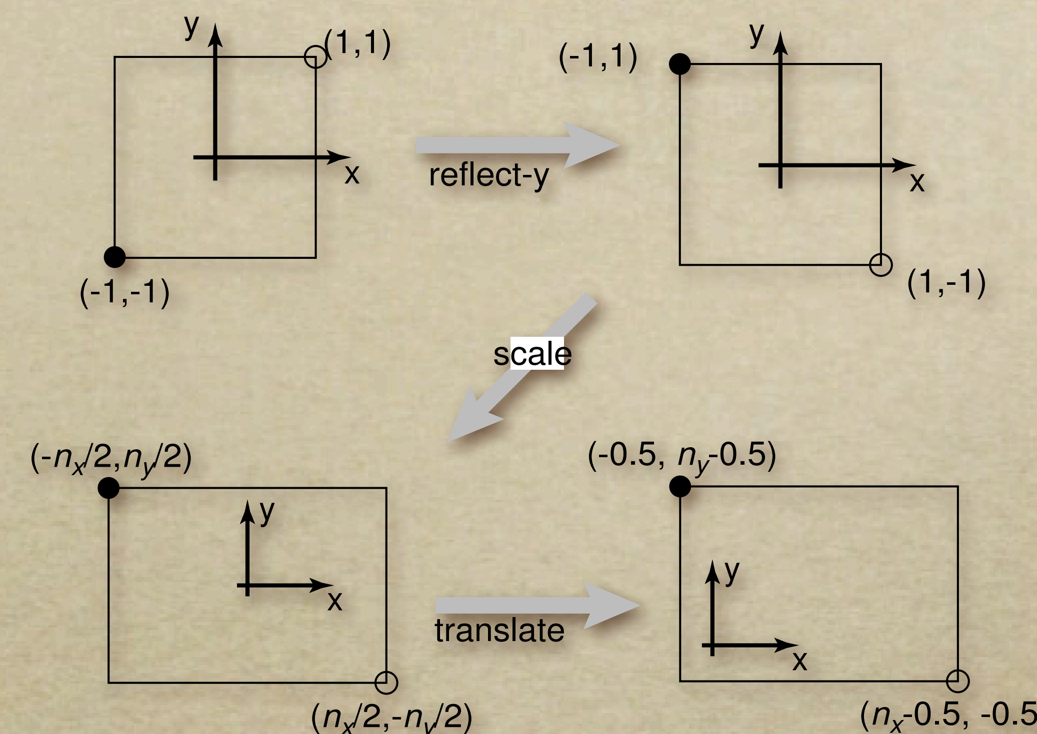
- Canonical view region
  - 2D: [-1,-1] to [+1,+1]



From Shirley textbook.

Canonical View Space

- Canonical view region
  - 2D: [-1,-1] to [+1,+1]



From Shirley textbook.
(Image coordinates are up-side-down.)

$$\begin{bmatrix} i \\ j \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{n_x}{2} & 0 & \frac{n_x - 1}{2} \\ 0 & -\frac{n_y}{2} & \frac{n_y - 1}{2} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

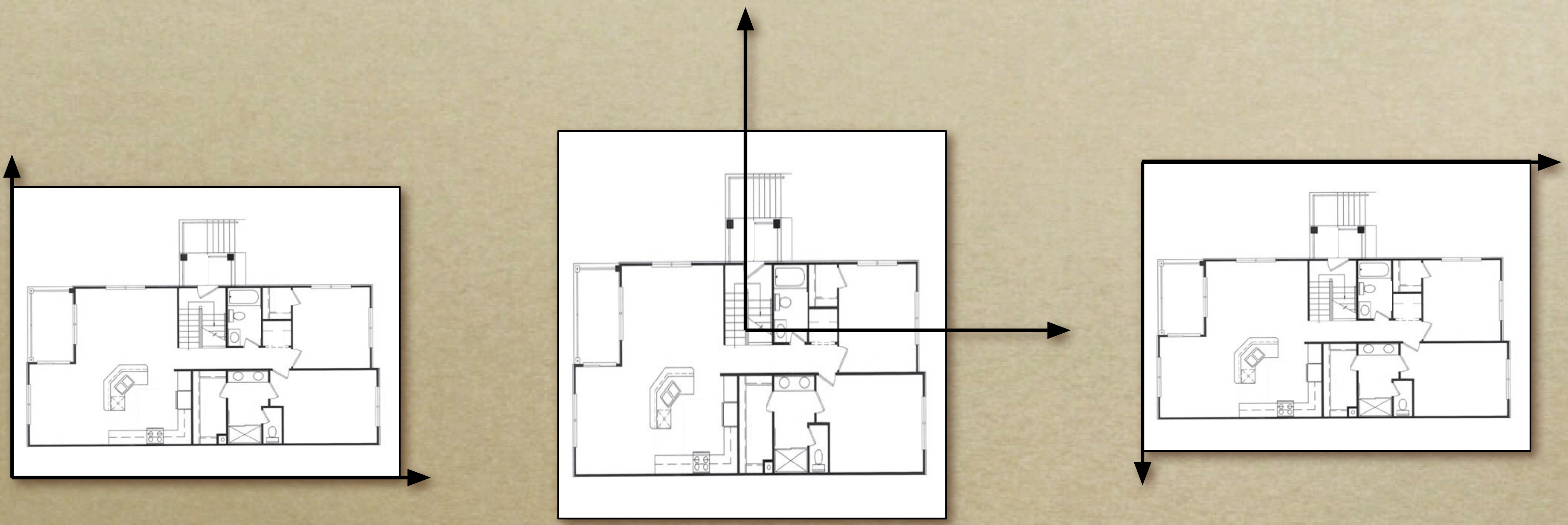
Remove minus for right-side-up

Canonical View Space

- Canonical view region
  - 2D: [-1,-1] to [+1,+1]
- Define arbitrary *window* and define objects
- Transform window to canonical region
- Do other things (we'll see clipping latter)
- Transform canonical to screen space
- Draw it.

From Shirley textbook.

Canonical View Space



World Coordinates (Meters) | Canonical | Screen Space (Pixels)

Note distortion issues...

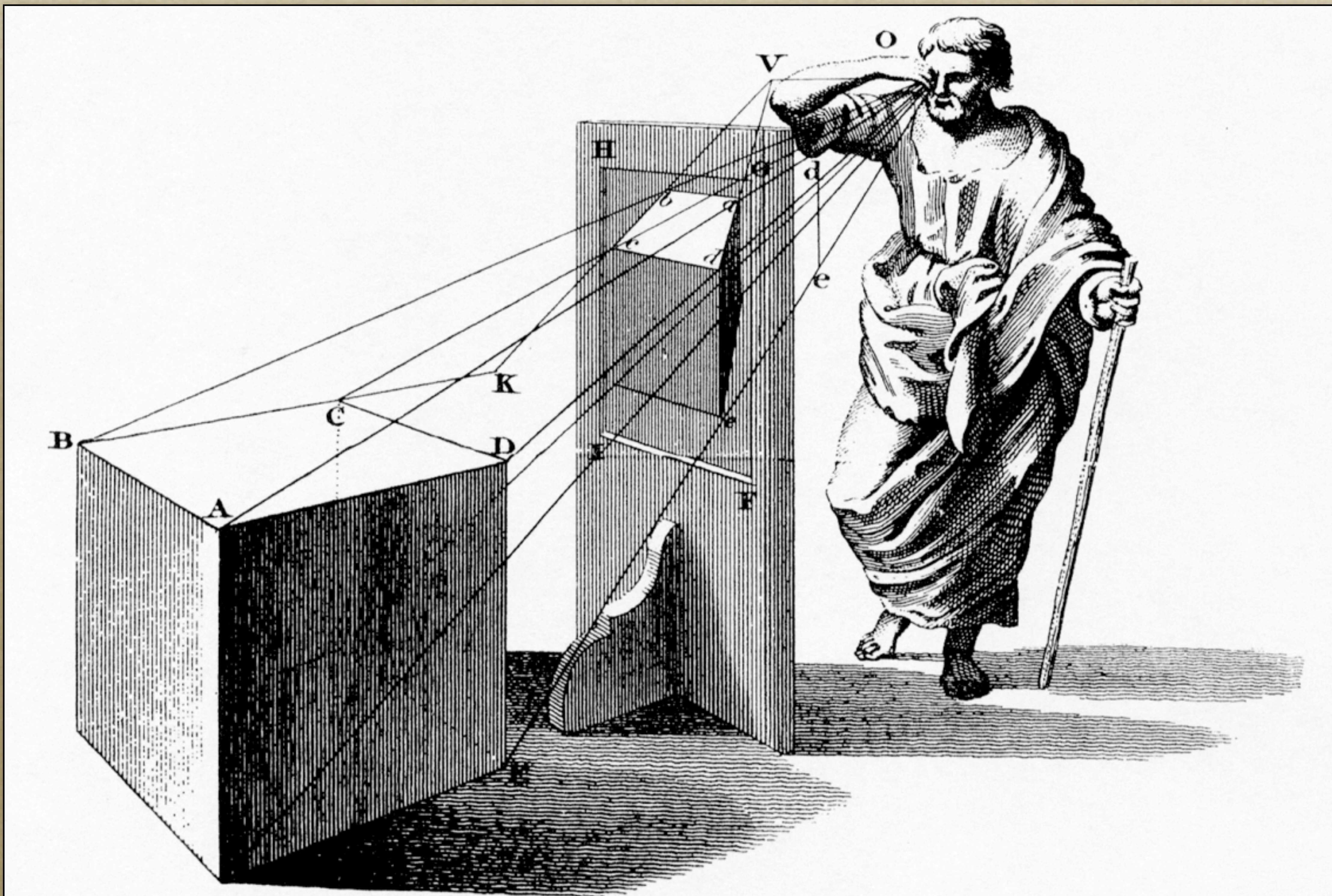
Projection

- Process of going from 3D to 2D
- Studies throughout history (e.g. painters)
- Different types of projection
  - Linear
    - Orthographic
    - Perspective
  - Nonlinear

Many special cases in books just one of these two...

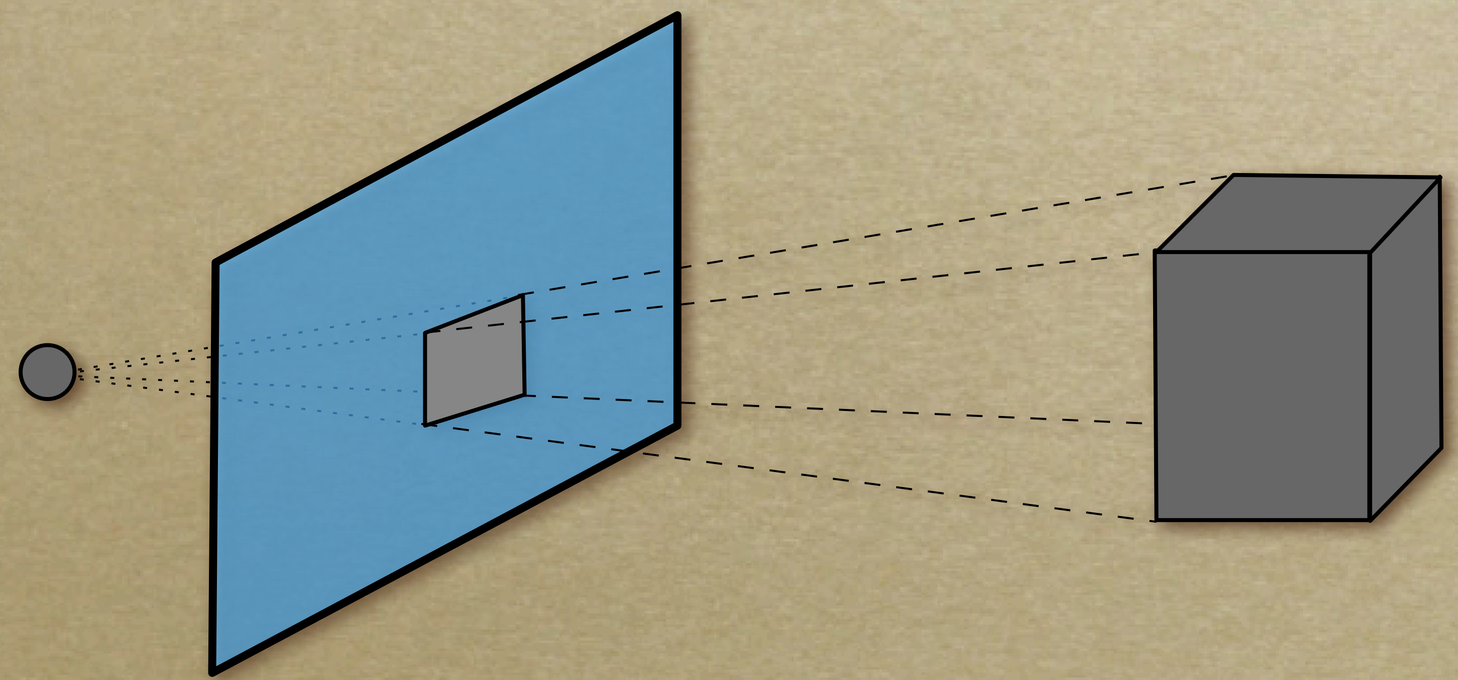
Orthographic is special case of perspective...

Perspective Projections



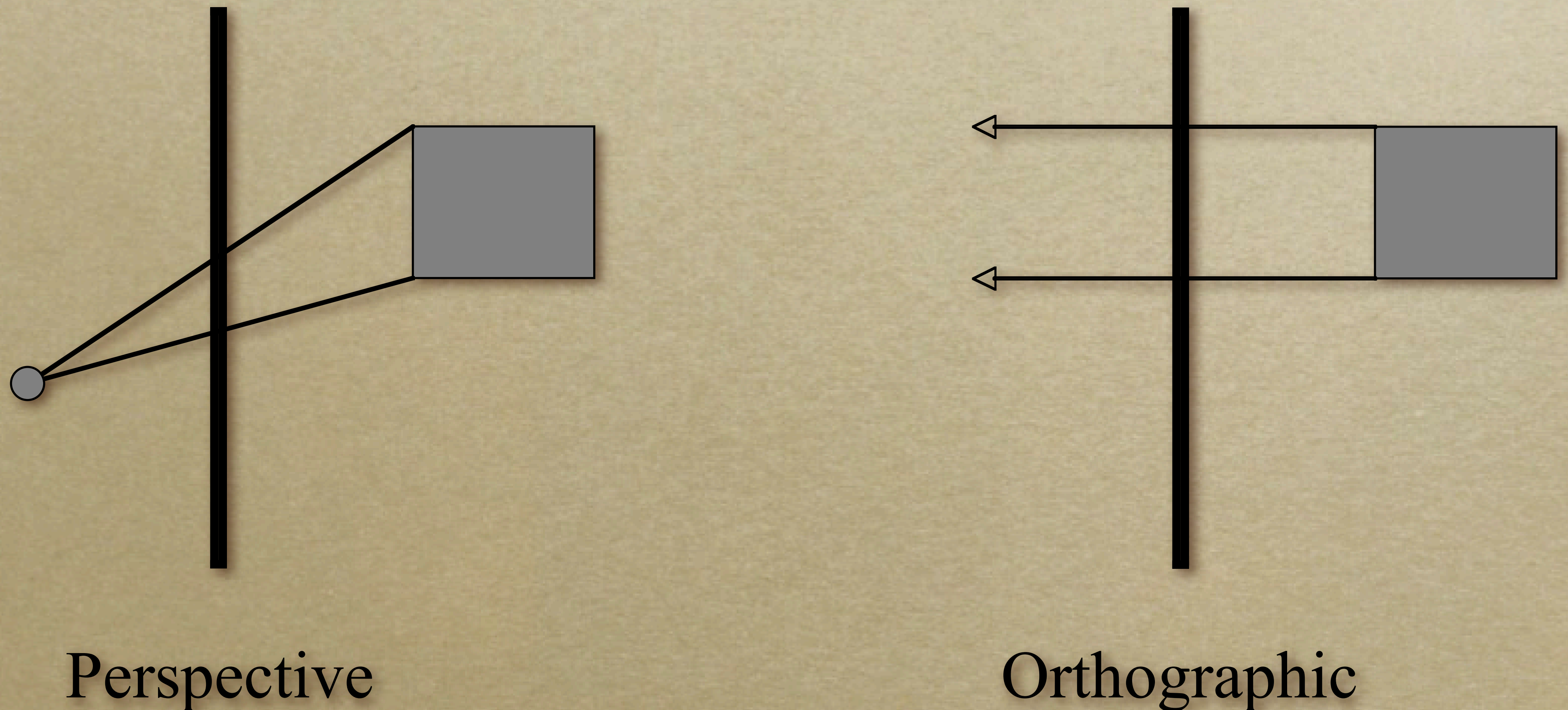
Linear Projection

- Projection onto a planar surface
- Projection directions either
  - Converge to a point
  - Are parallel (converge at infinity)

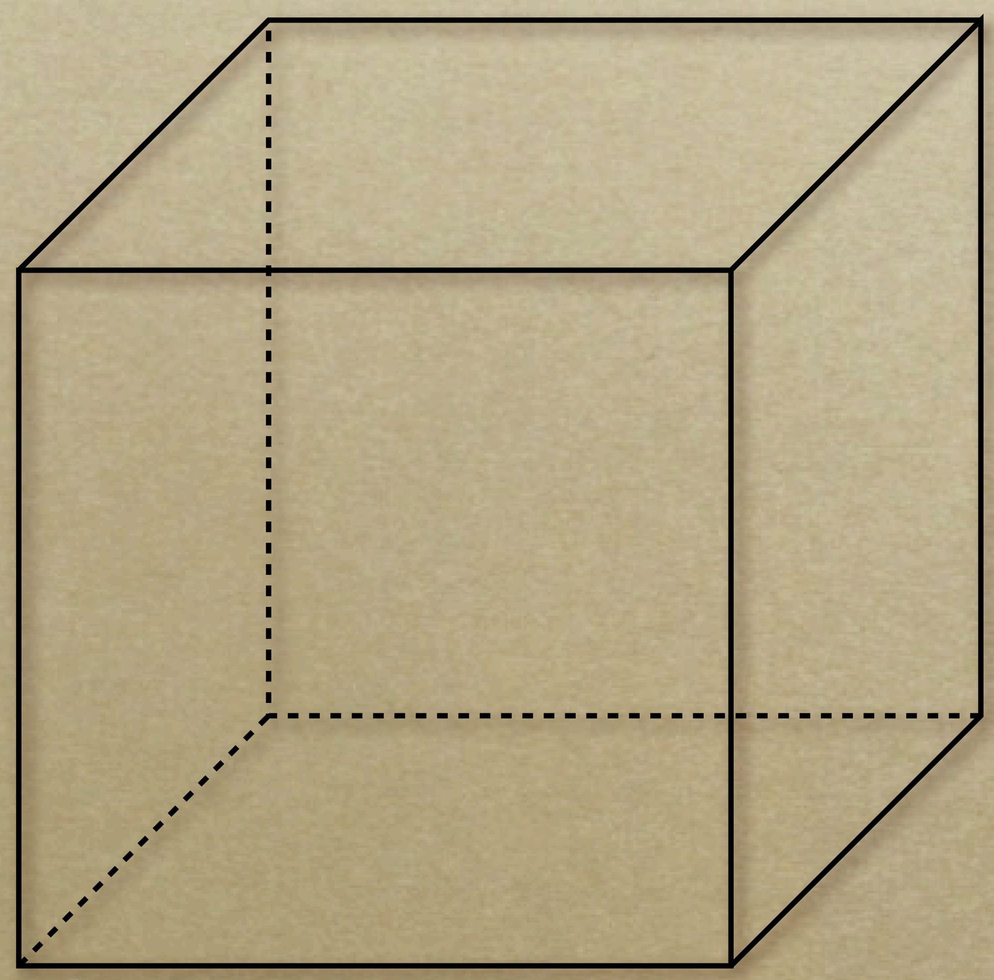


Linear Projection

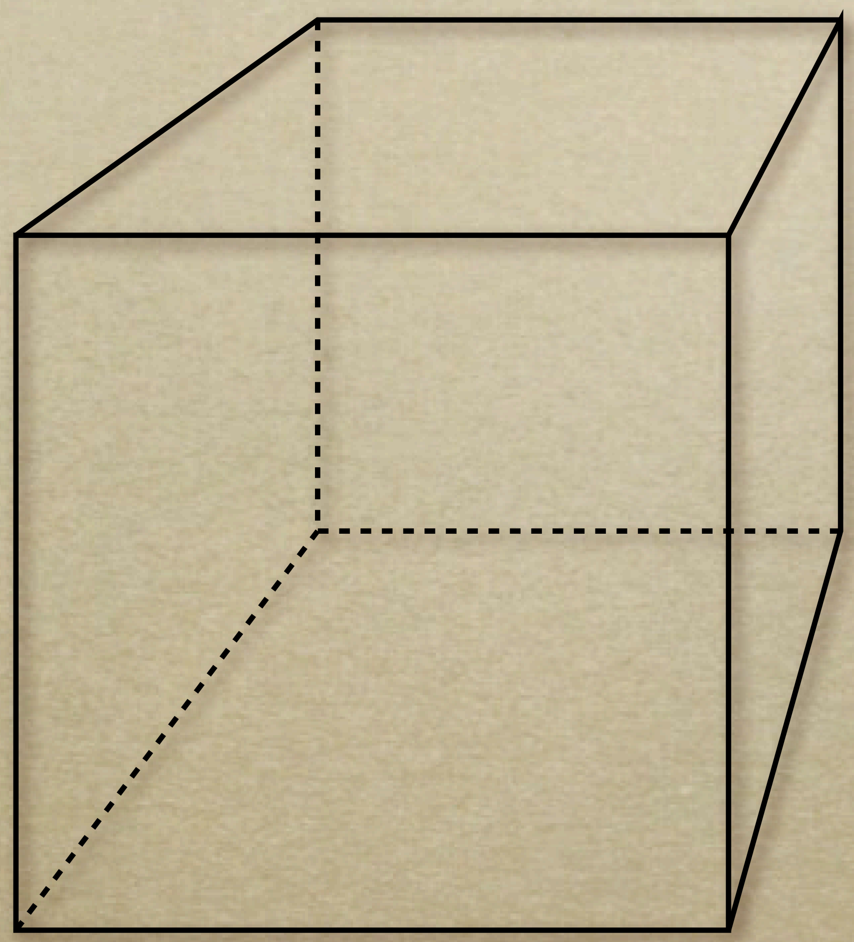
- A 2D view



Linear Projection

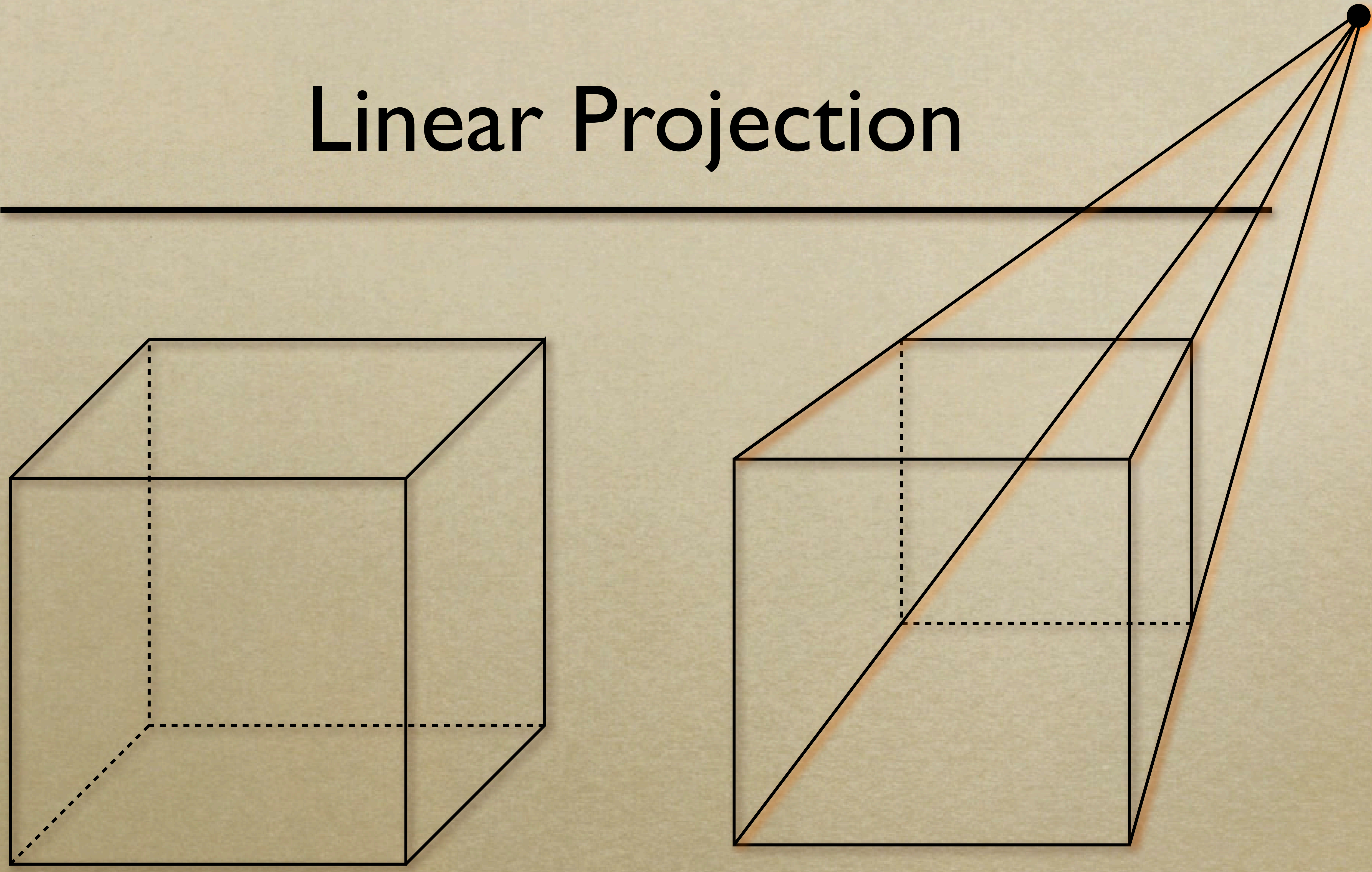


Orthographic



Perspective

Linear Projection



Orthographic

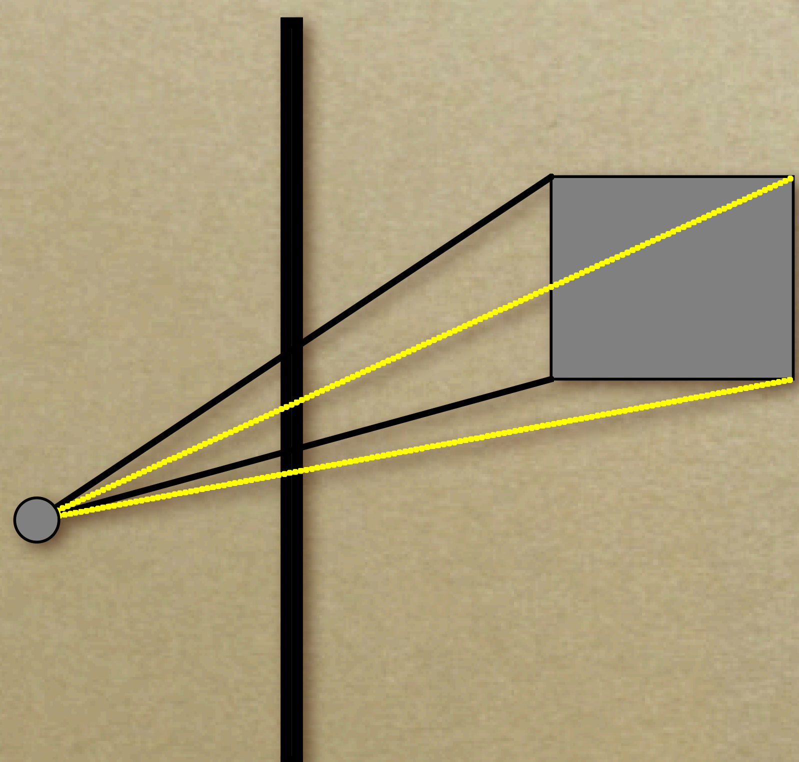
Perspective

Linear Projection

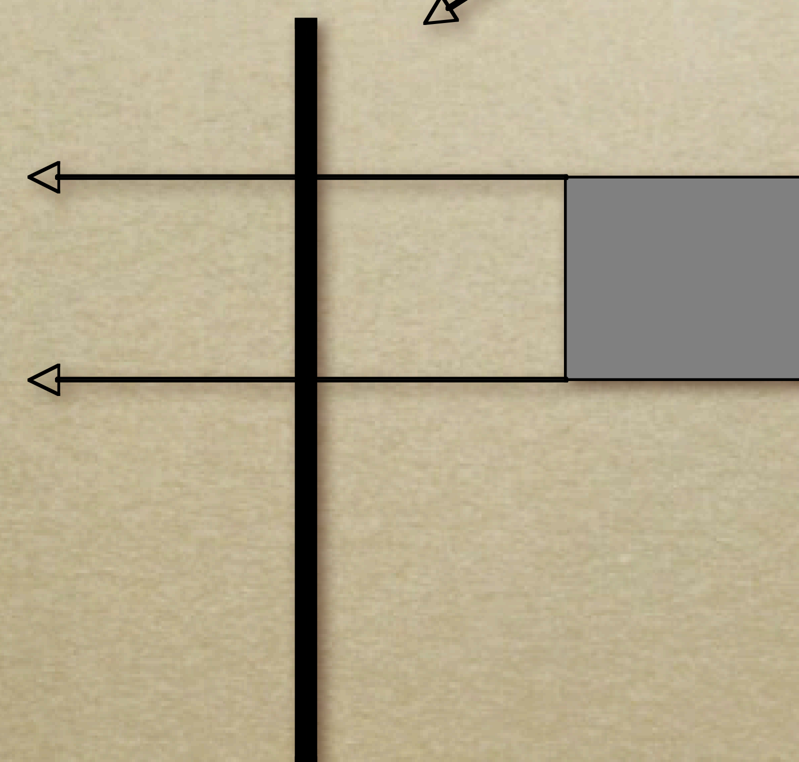
- A 2D view

Note how different things can be seen

Parallel lines “meet” at infinity



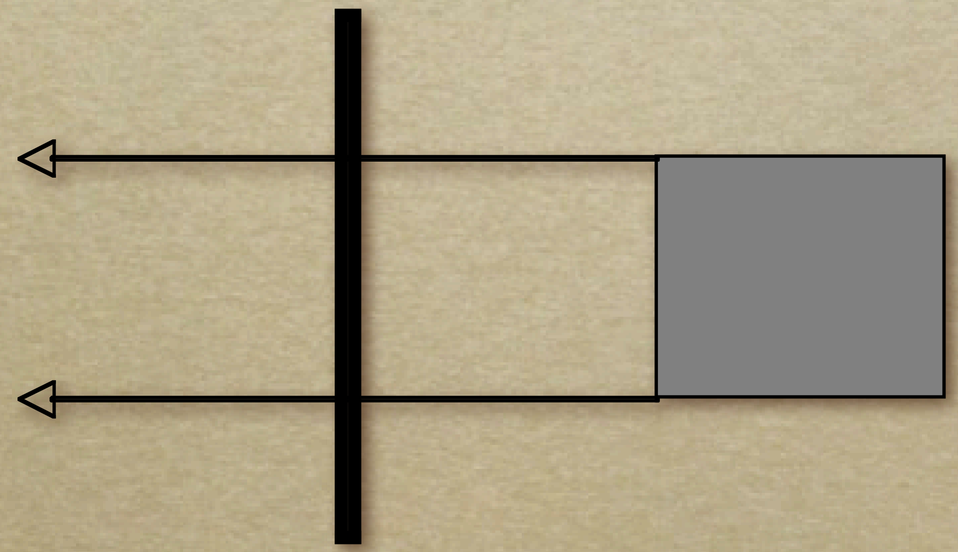
Perspective



Orthographic

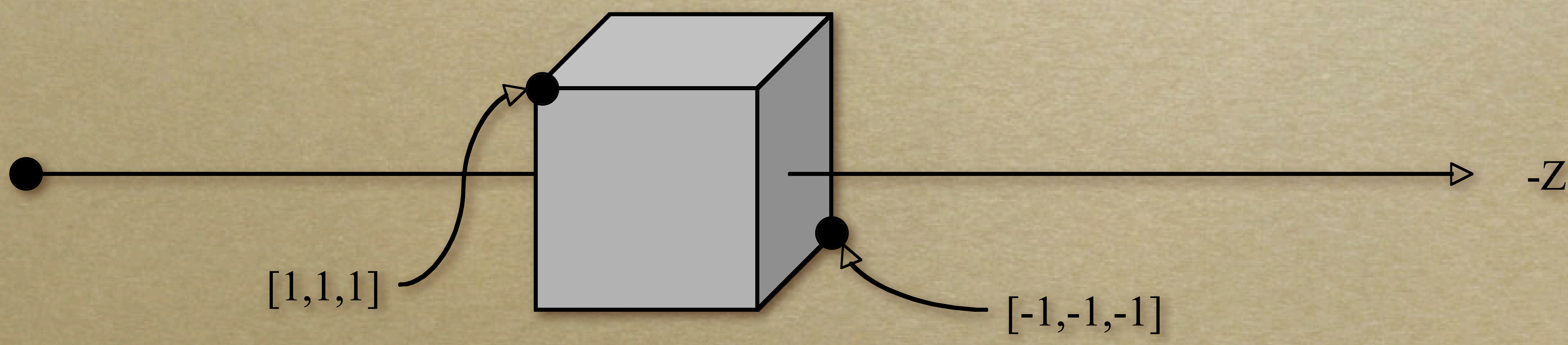
Orthographic Projection

- No foreshortening
- Parallel lines stay parallel
- Poor depth cues



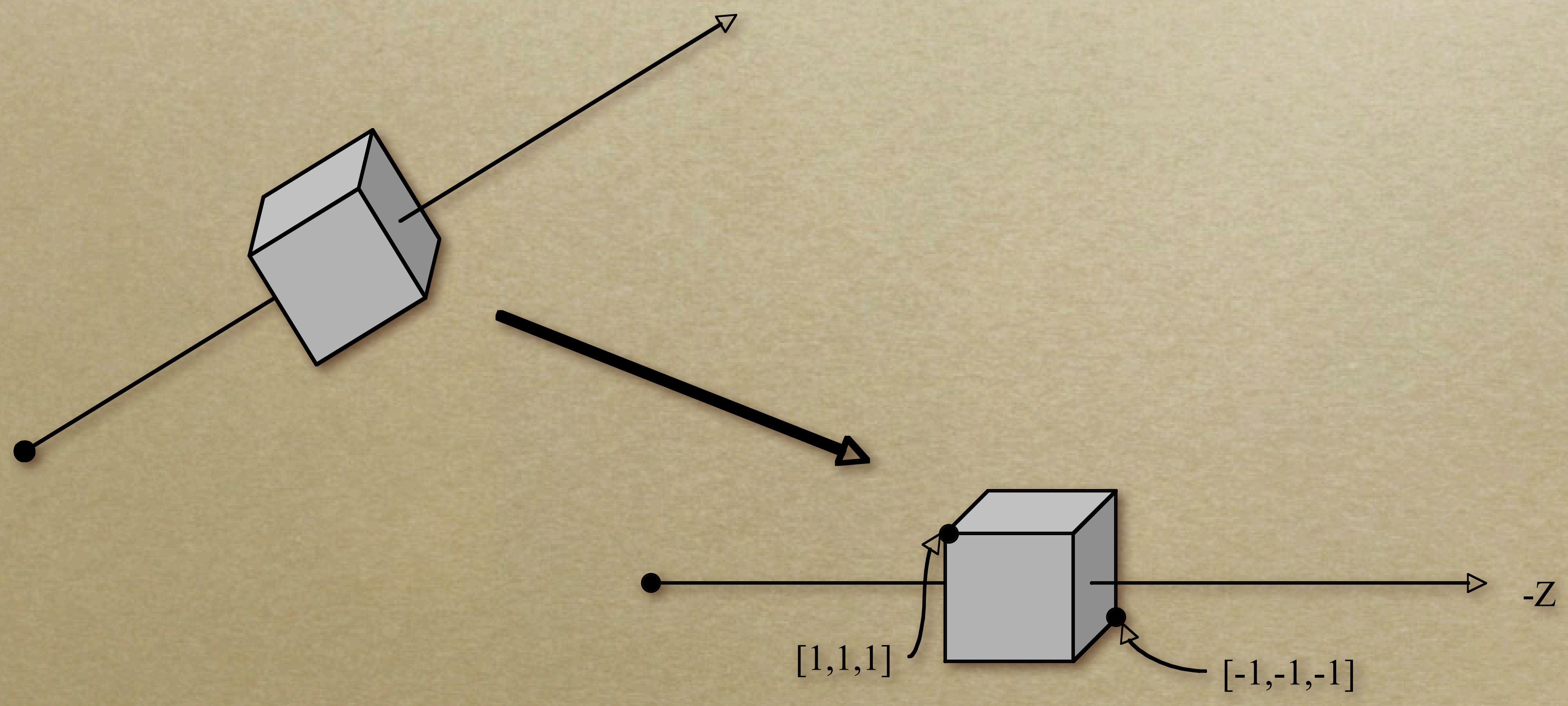
Canonical View Space

- Canonical view region
  - 3D: [-1,-1,-1] to [+1,+1,+1]
- Assume looking down *-Z* axis
  - Recall that "Z is in your face"

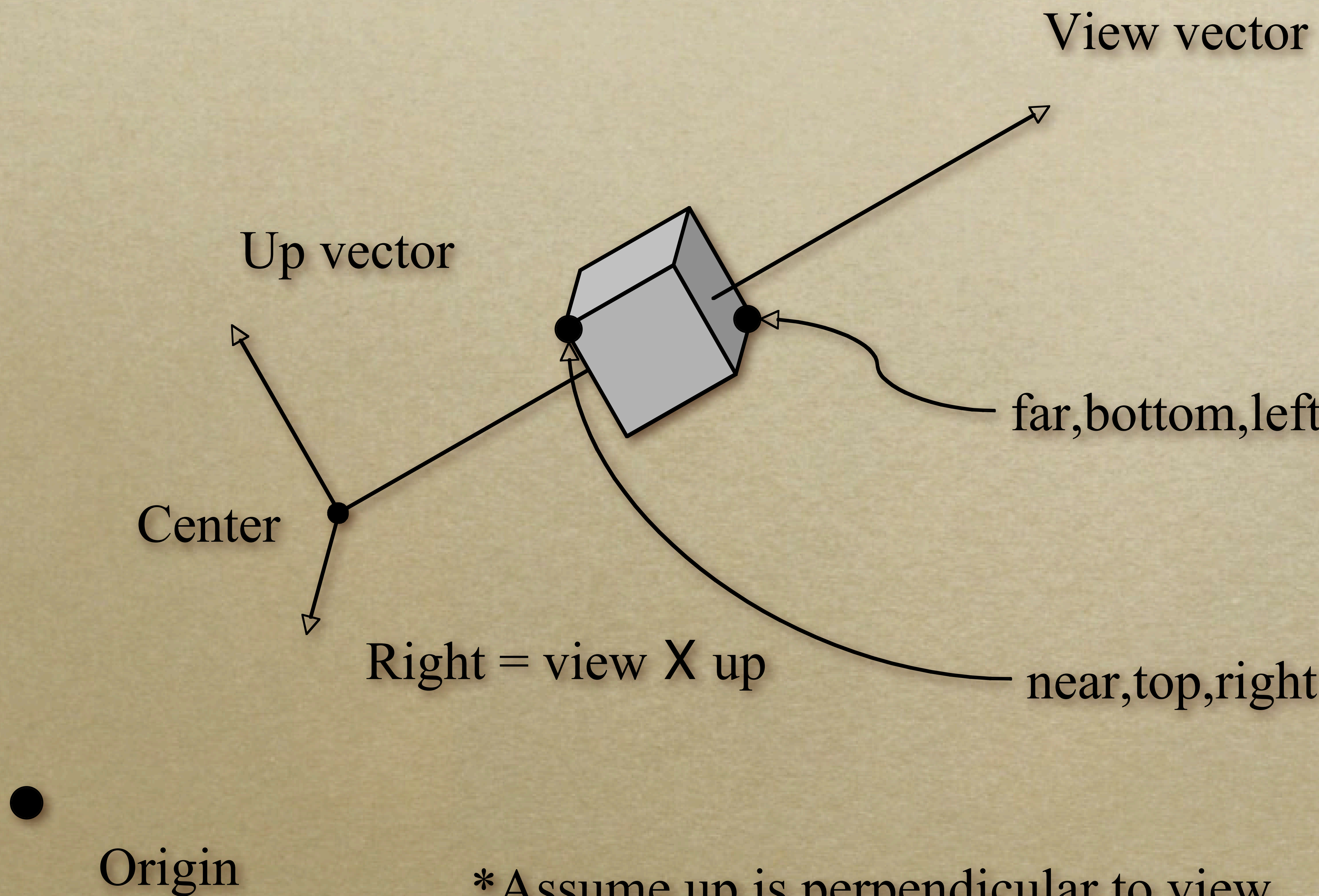


Orthographic Projection

- Convert arbitrary view volume to canonical



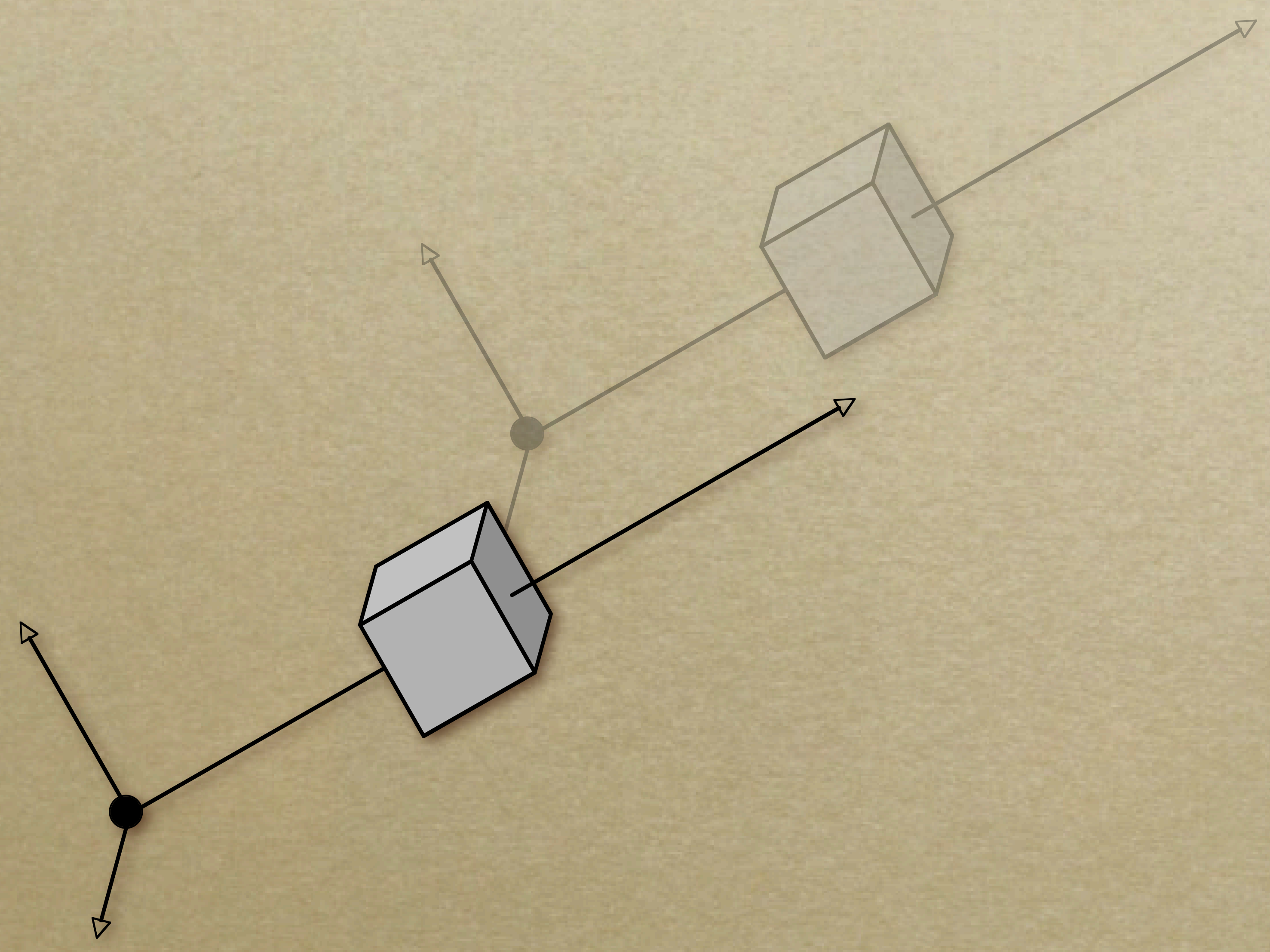
Orthographic Projection



*Assume up is perpendicular to view.

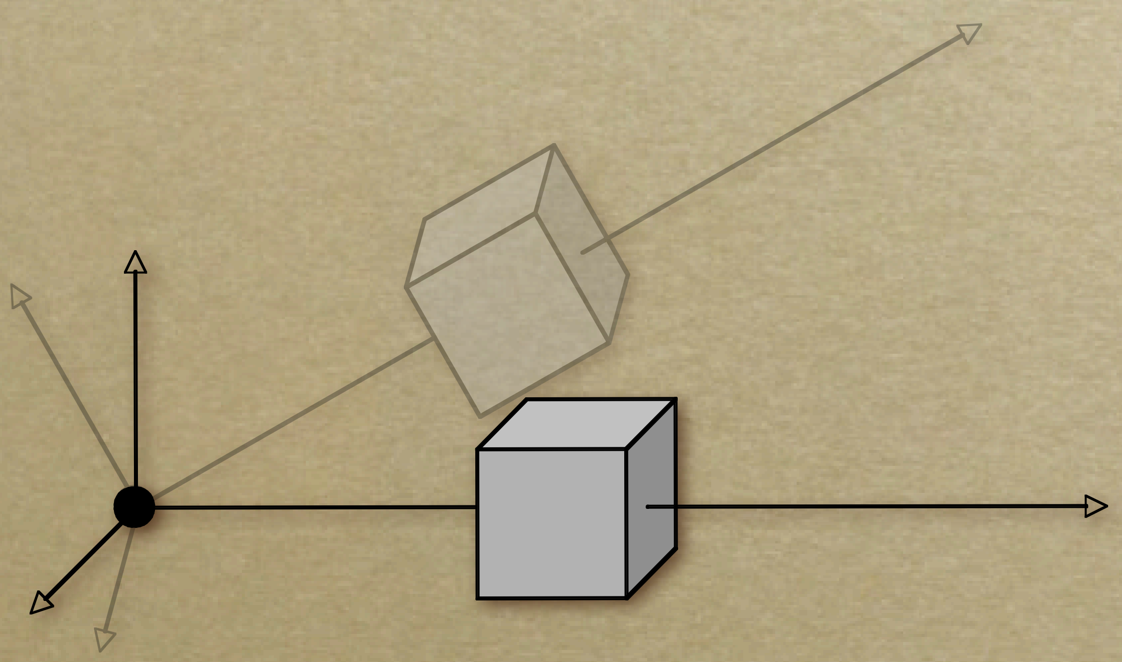
Orthographic Projection

- Step 1: translate center to origin



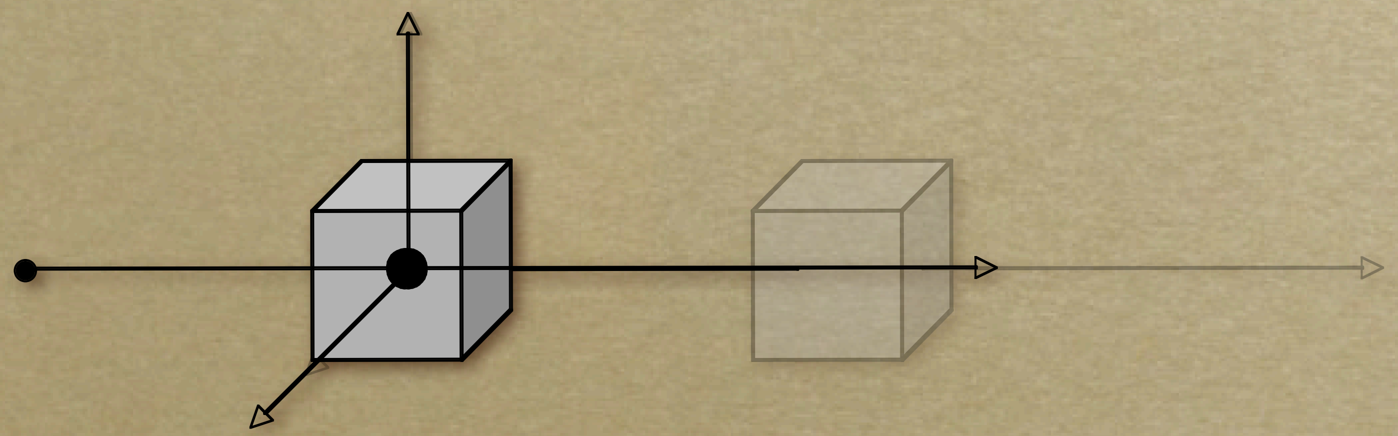
Orthographic Projection

- Step 1: translate center to origin
- Step 2: rotate *view* to **-Z** and *up* to **+Y**



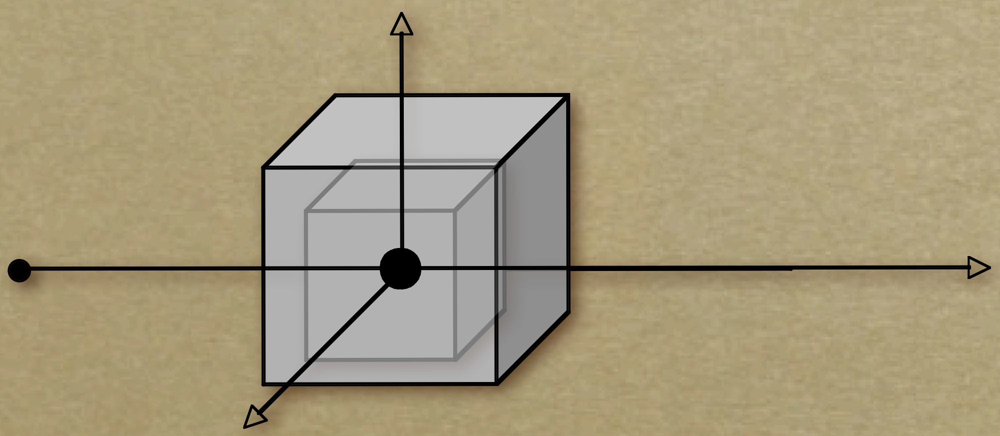
Orthographic Projection

- Step 1: translate center to origin
- Step 2: rotate *view* to **-Z** and *up* to **+Y**
- Step 3: center view volume



Orthographic Projection

- Step 1: translate center to origin
- Step 2: rotate *view* to **-Z** and *up* to **+Y**
- Step 3: center view volume
- Step 4: scale to canonical size

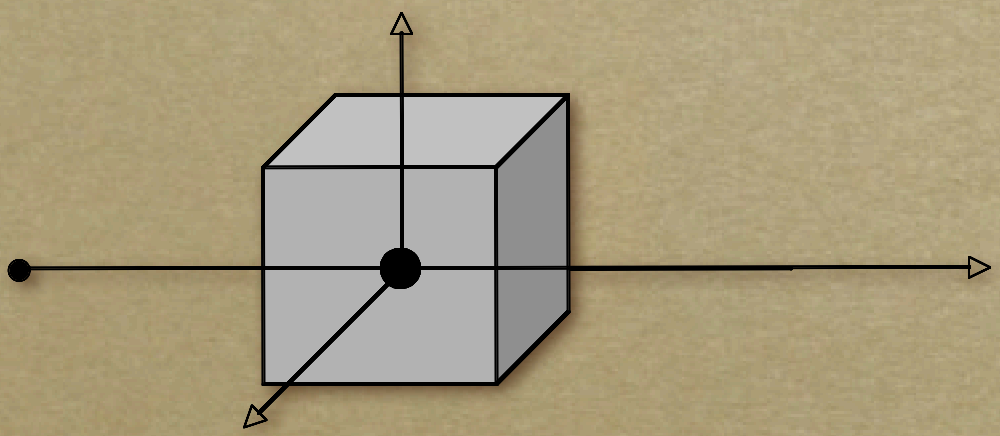


Orthographic Projection

- Step 1: translate center to origin
- Step 2: rotate *view* to **-Z** and *up* to **+Y**
- Step 3: center view volume
- Step 4: scale to canonical size

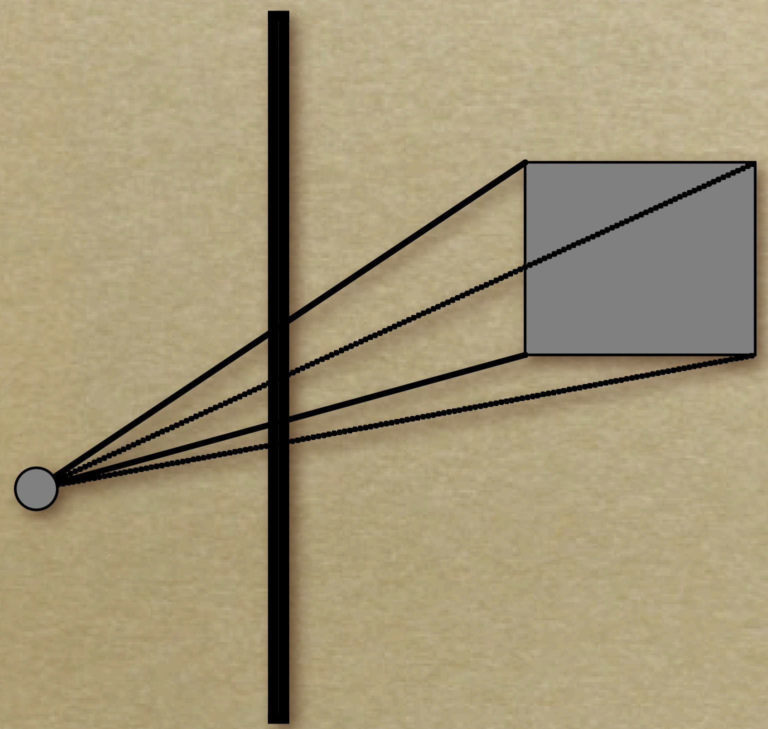
$$\mathbf{M} = \underbrace{\mathbf{S}\cdot\mathbf{T}_2}_{\mathbf{M}_o}\cdot\underbrace{\mathbf{R}\cdot\mathbf{T}_1}_{\mathbf{M}_v}$$

$$\mathbf{M} = \mathbf{M}_o\cdot\mathbf{M}_v$$



Perspective Projection

- Foreshortening: further objects appear smaller
- Some parallel line stay parallel, most don't
- Lines still look like lines
- $\mathbf{z}$ ordering preserved (where we care)



Perspective Projection

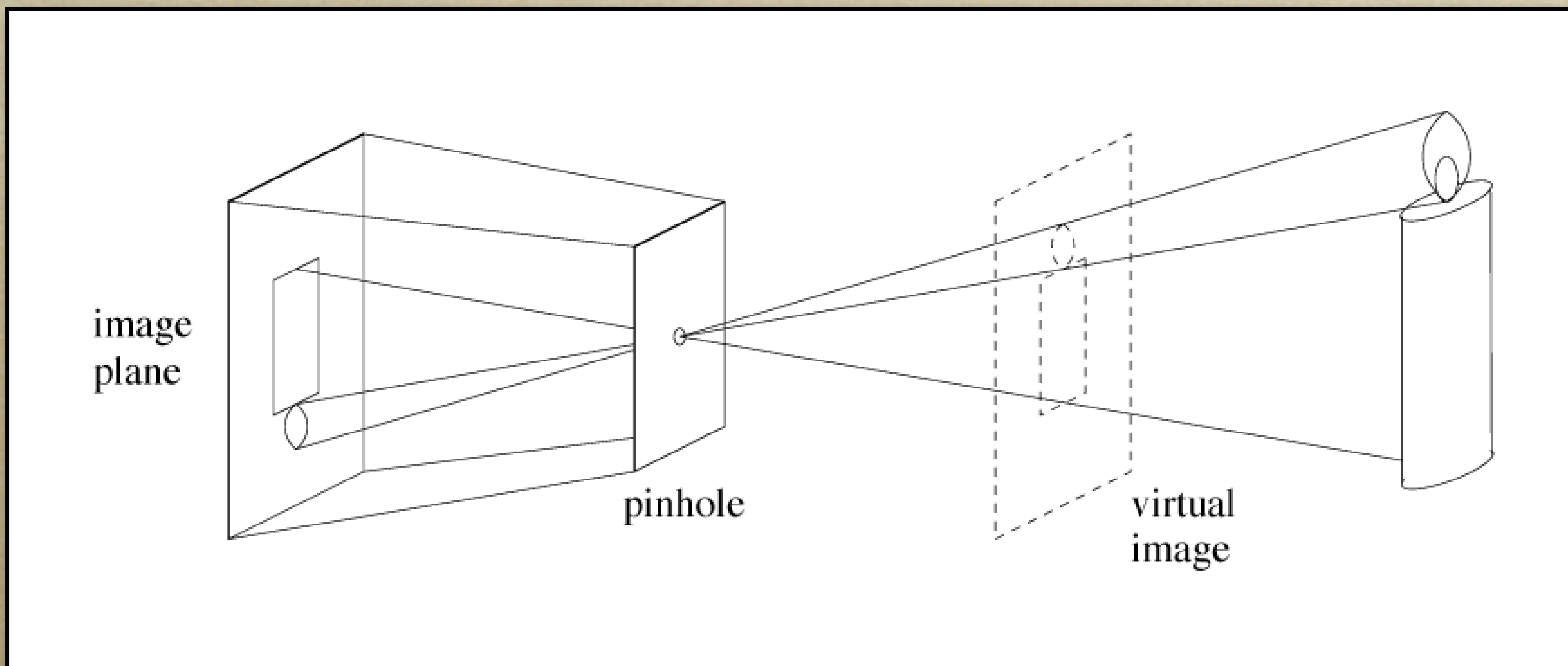


Image from D. Forsyth

Pinhole *a.k.a* center of projection

Perspective Projection

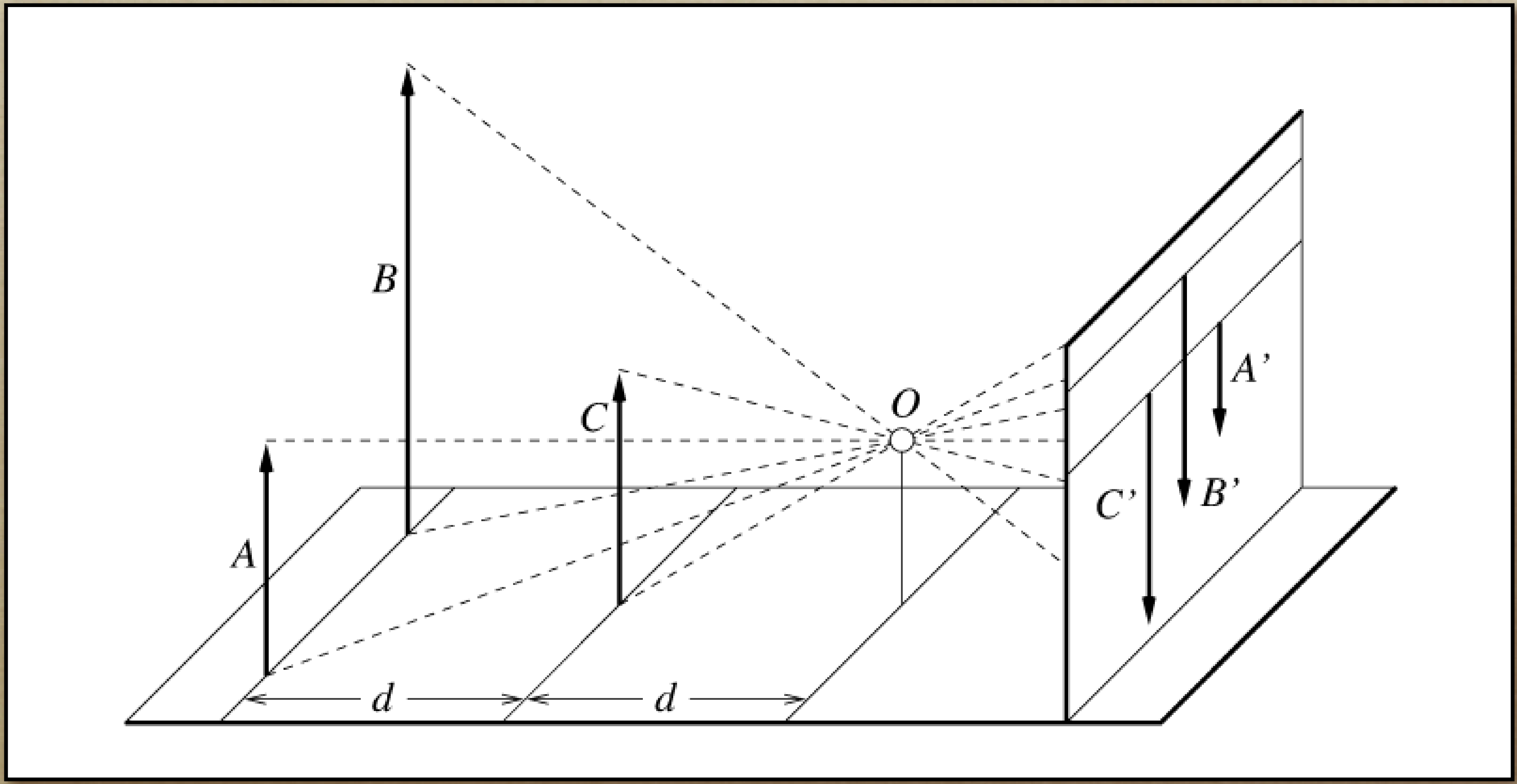
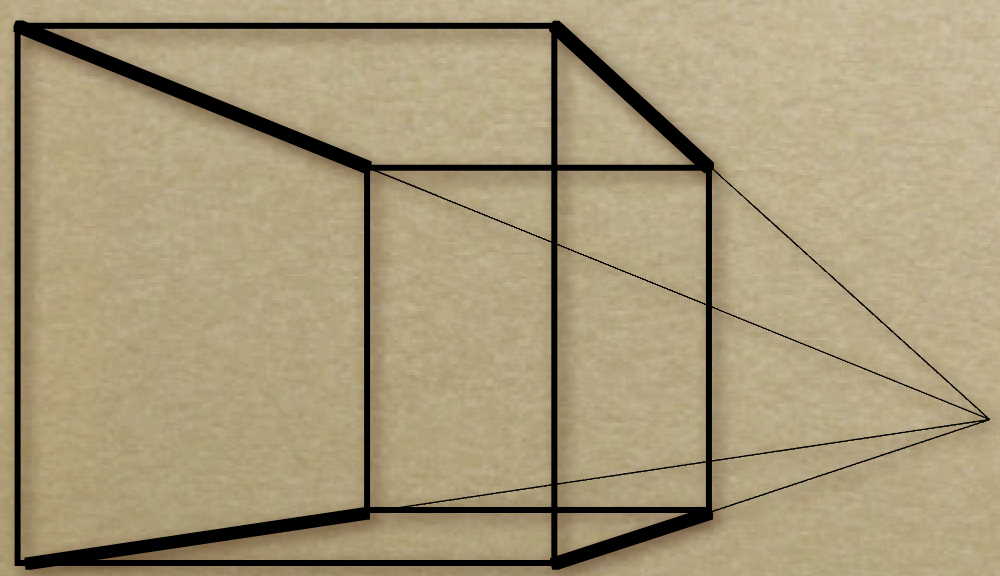


Image from D. Forsyth

Foreshortening: distant objects appear smaller

Perspective Projection

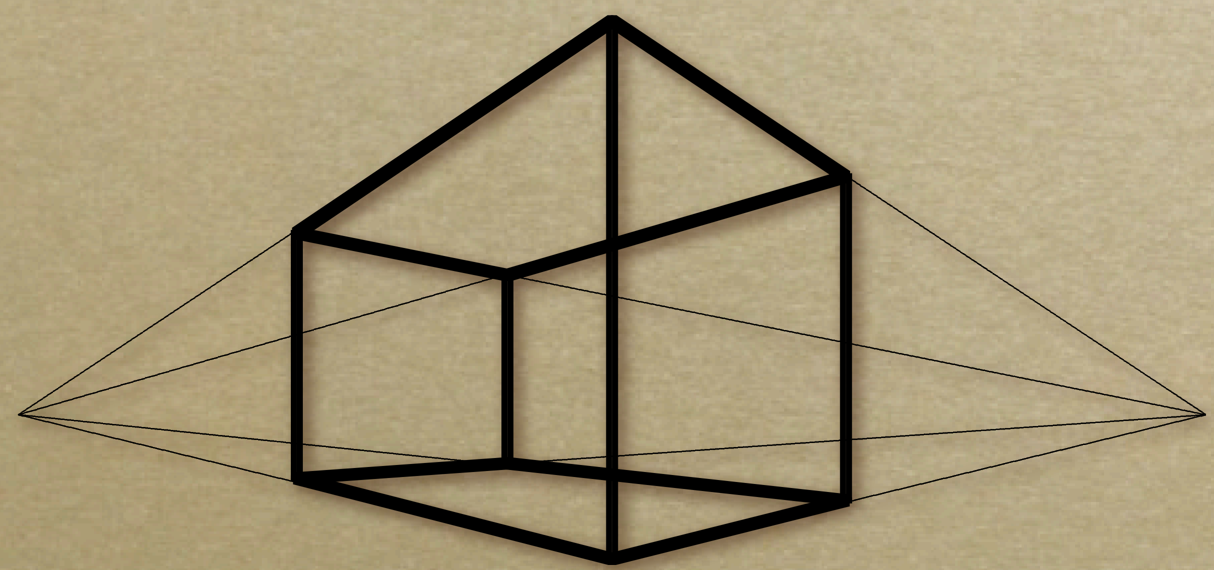
- Vanishing points
  - Depend on the scene
  - Not intrinsic to camera



"One point perspective"

Perspective Projection

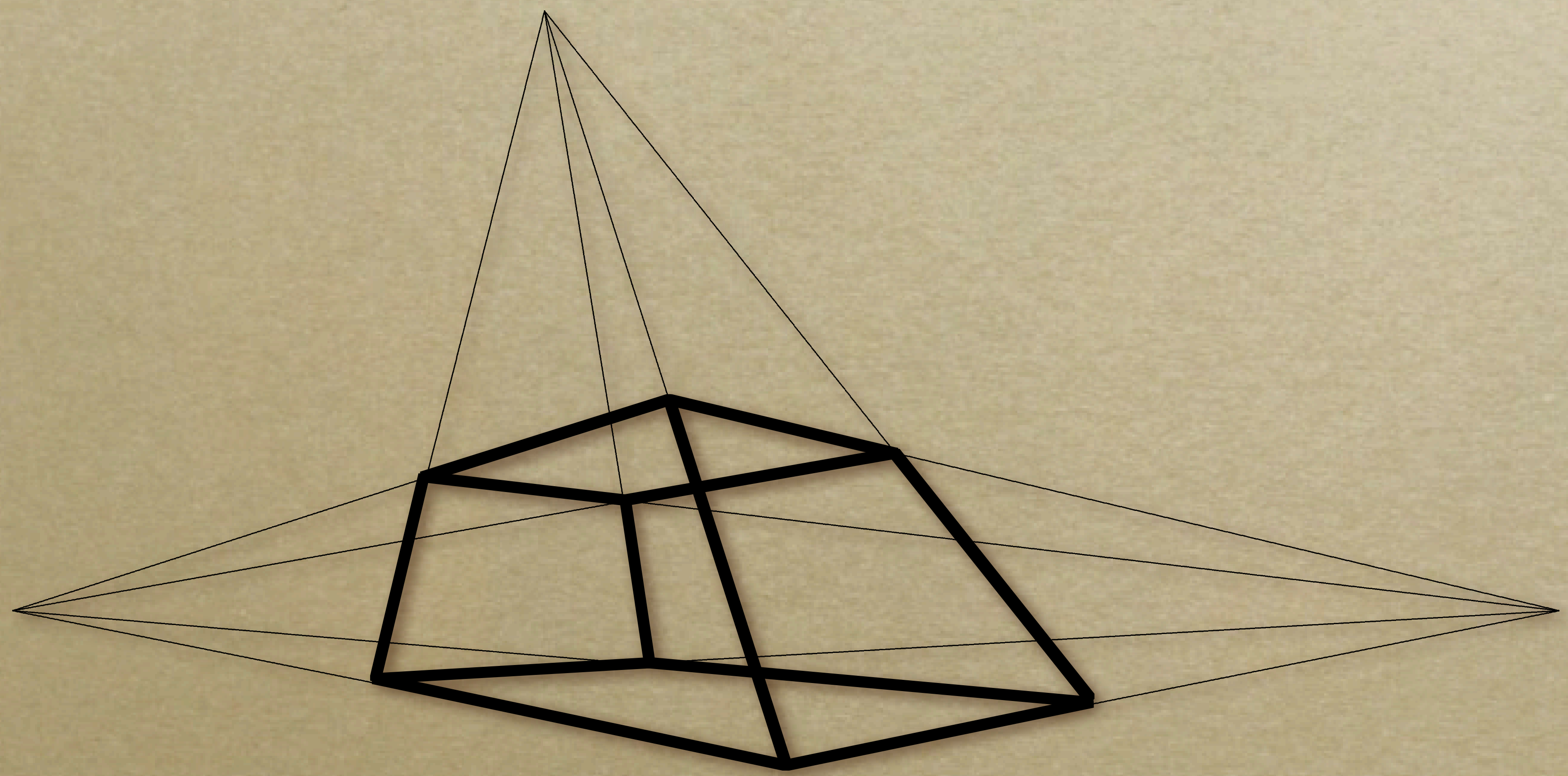
- Vanishing points
  - Depend on the scene
  - Nor intrinsic to camera



"Two point perspective"

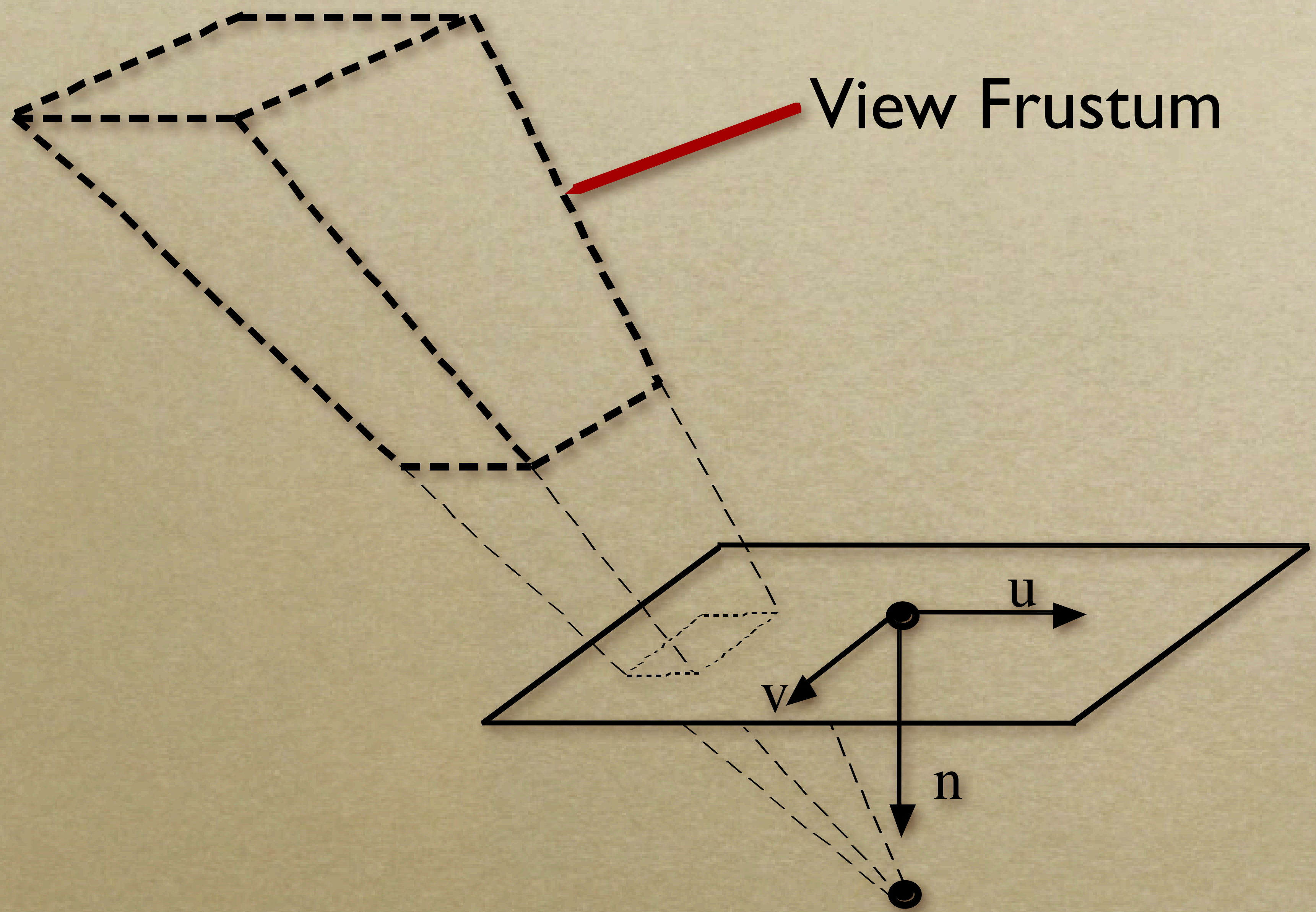
Perspective Projection

- Vanishing points
  - Depend on the scene
  - Not intrinsic to camera

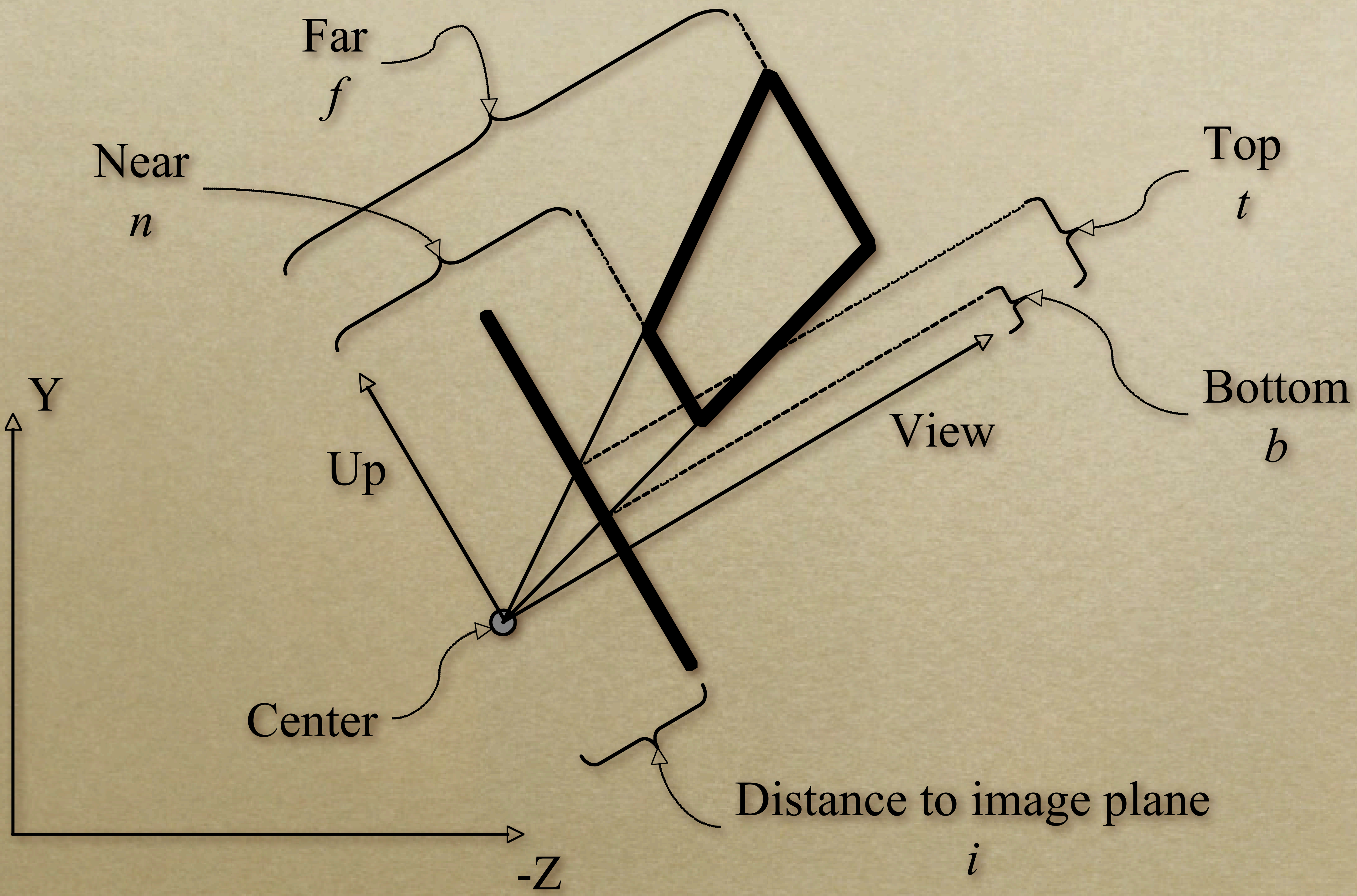


"Three point perspective"

Perspective Projection

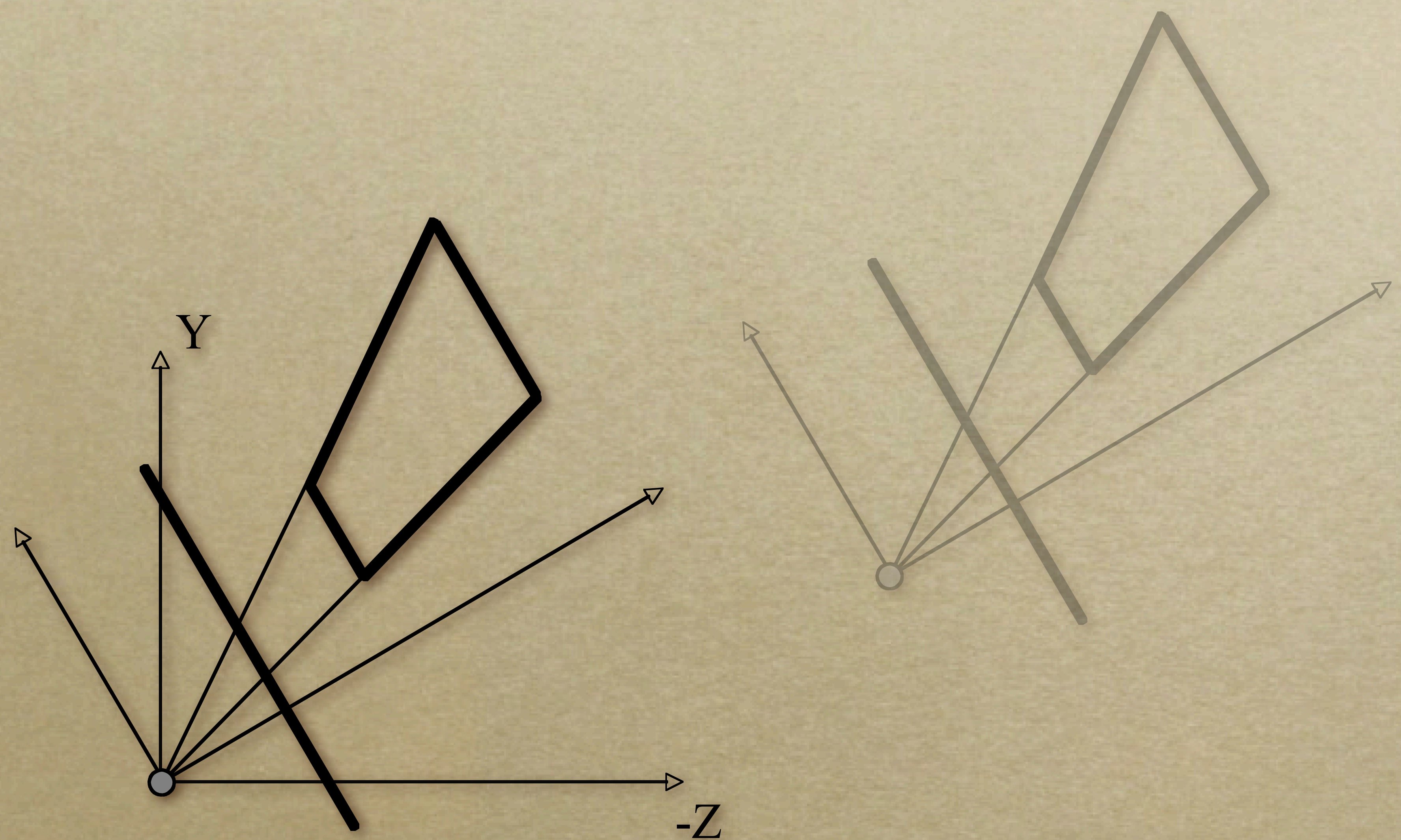


Perspective Projection



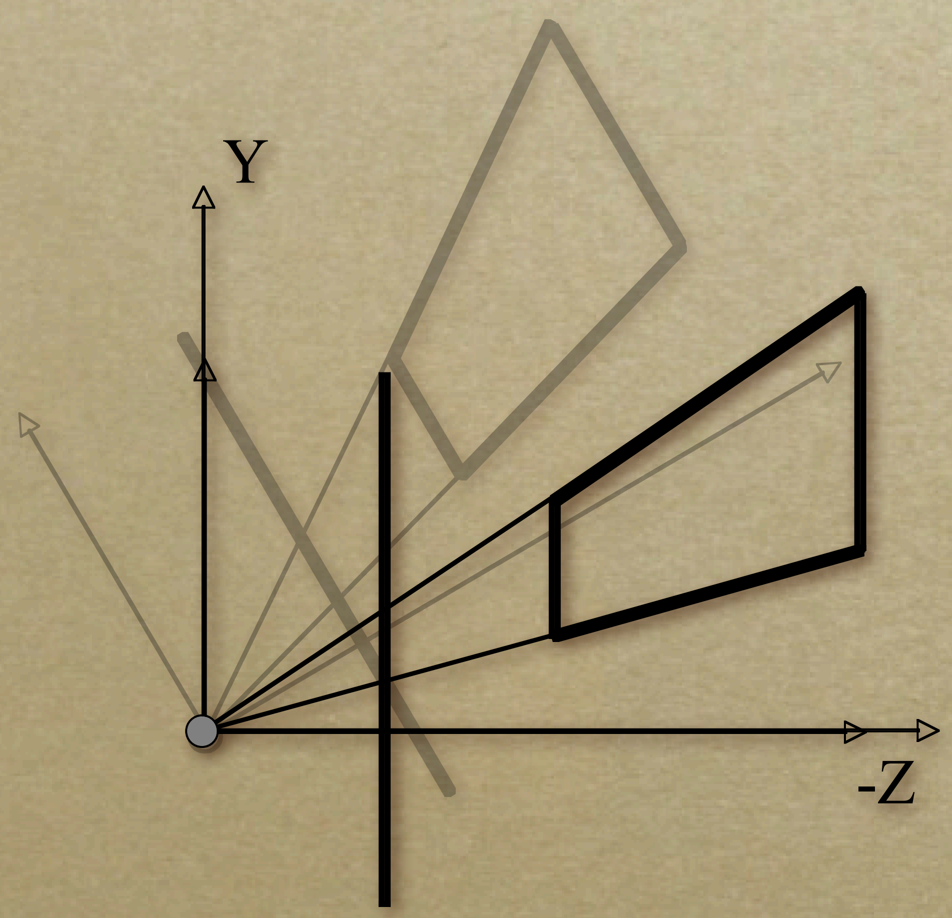
Perspective Projection

- Step 1: Translate *center* to origin



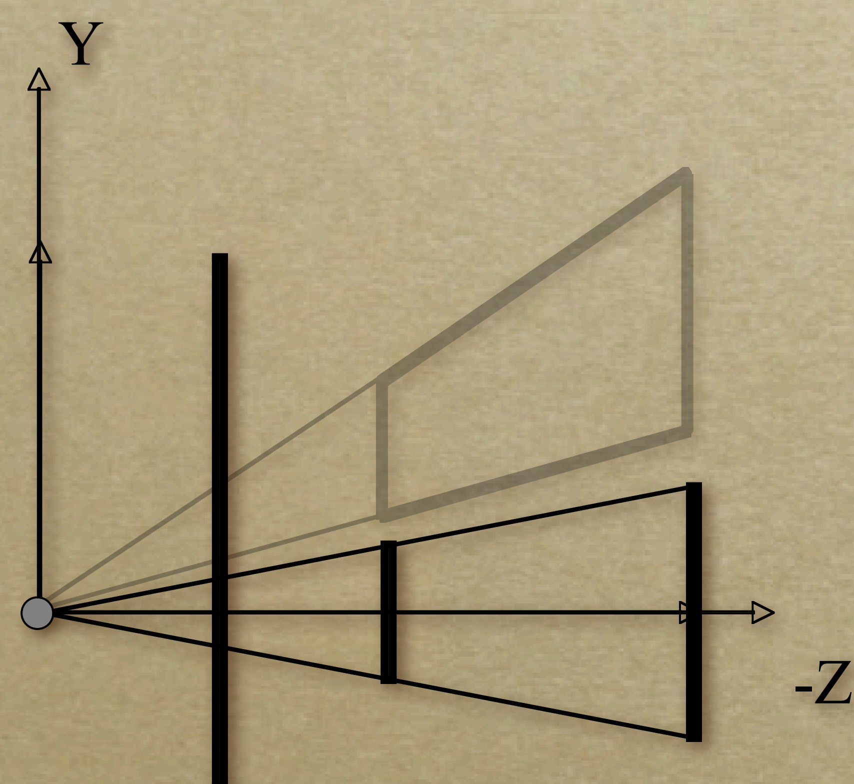
Perspective Projection

- Step 1: Translate *center* to origin
- Step 2: Rotate *view* to **-Z**, *up* to **+Y**



Perspective Projection

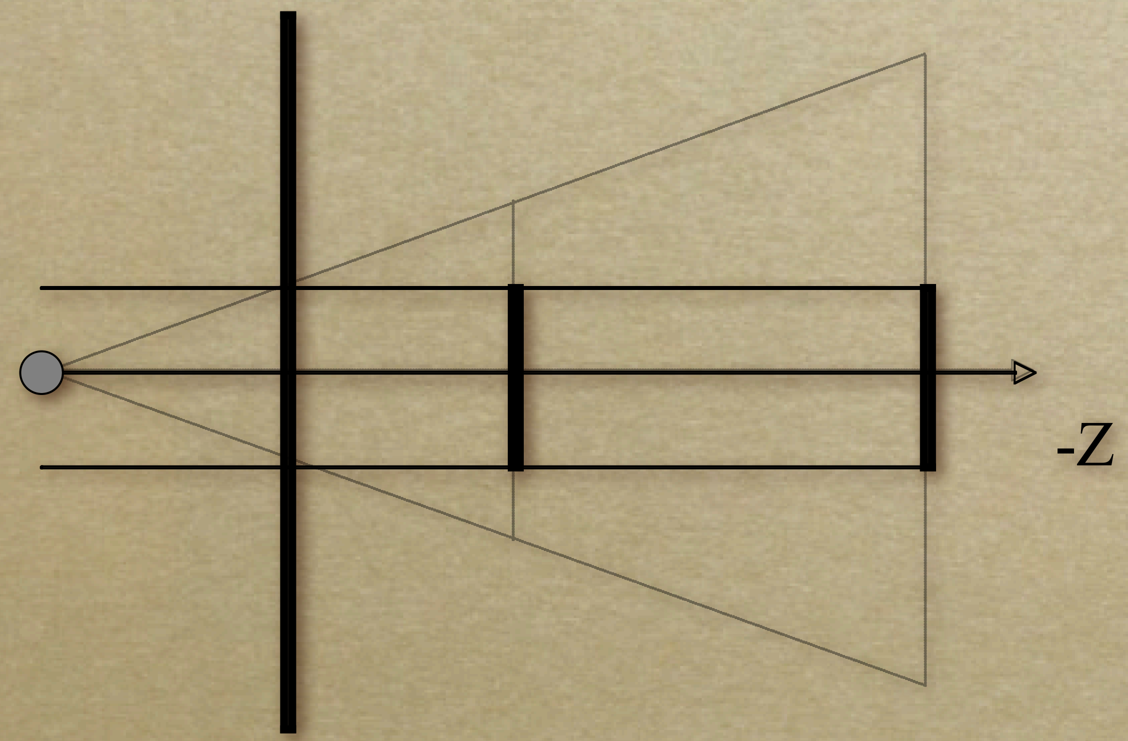
- Step 1: Translate *center* to origin
- Step 2: Rotate *view* to **-Z**, *up* to **+Y**
- Step 3: Shear center-line to **-Z** axis



Perspective Projection

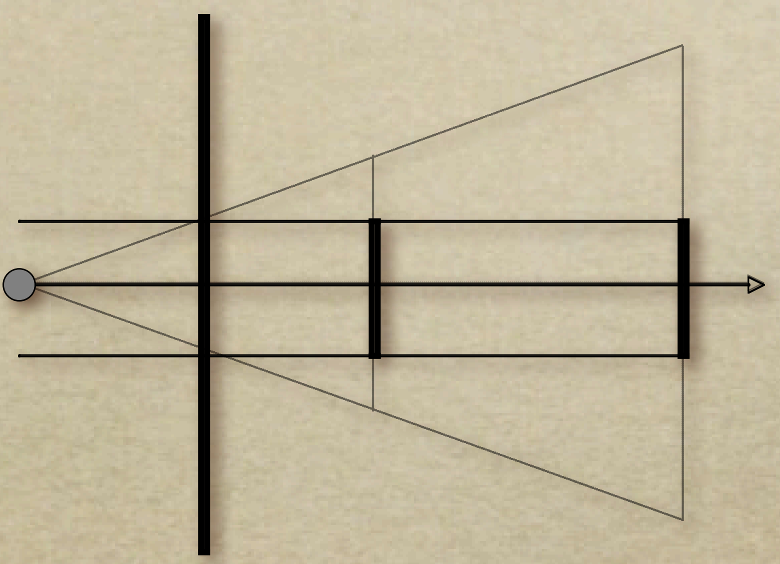
- Step 1: Translate *center* to origin
- Step 2: Rotate *view* to **-Z**, *up* to **+Y**
- Step 3: Shear center-line to **-Z** axis
- Step 4: Perspective

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{i+f}{i} & f \\ 0 & 0 & \frac{-1}{i} & 0 \end{bmatrix}$$



Perspective Projection

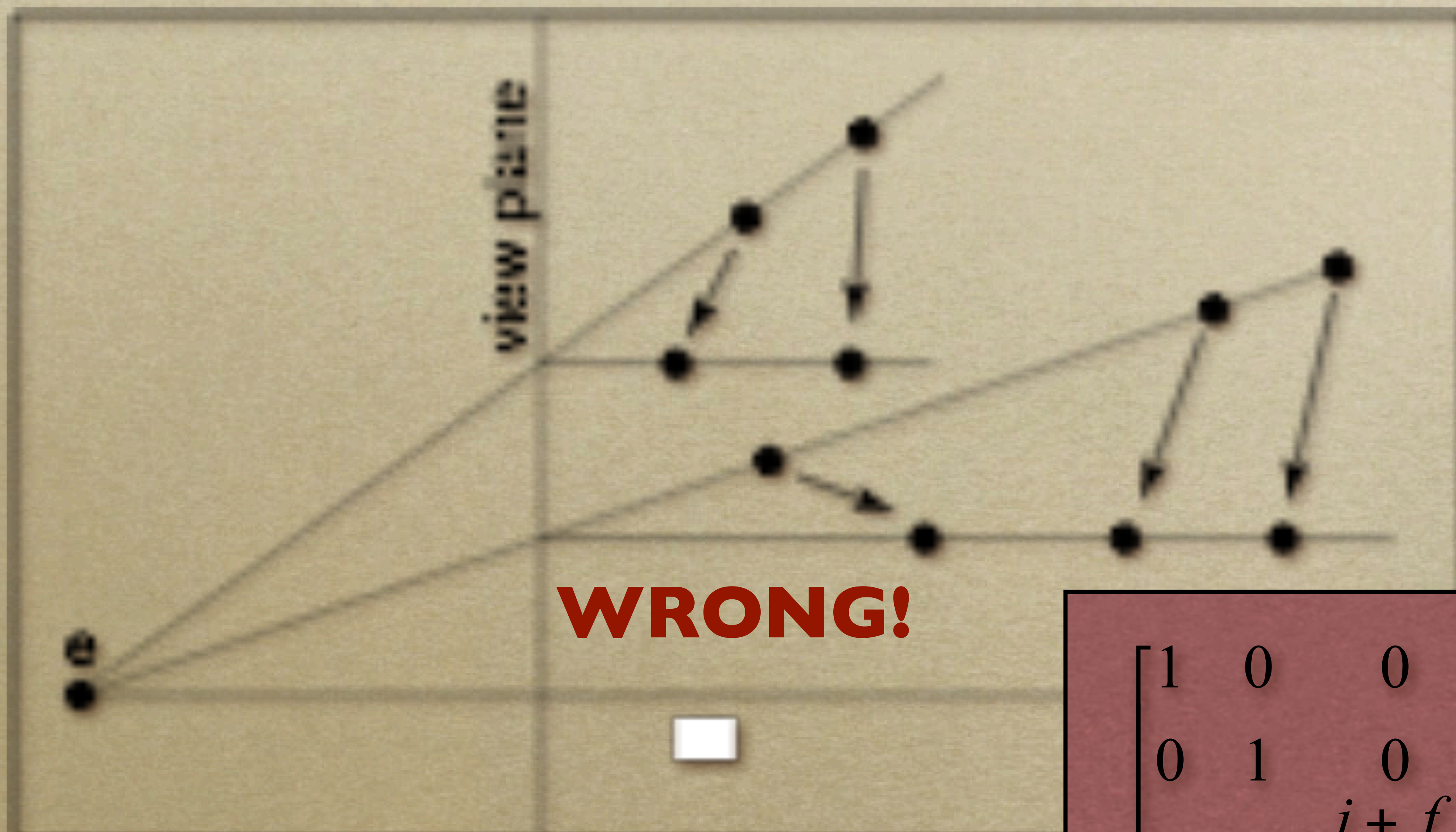
- Step 4: Perspective
  - Points at $z = -i$ stay at $z = -i$
  - Points at $z = -f$ stay at $z = -f$
  - Points at $z = 0$ goto $z = \pm\infty$
  - Points at $z = -\infty$ goto $z = -(i+f)$



  - x and y values divided by $-z/i$
  - Straight lines stay straight
  - Depth ordering preserved in $[-i, -f]$
  - Movement along lines distorted

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{i+f}{i} & f \\ 0 & 0 & \frac{-1}{i} & 0 \end{bmatrix}$$

Perspective Projection

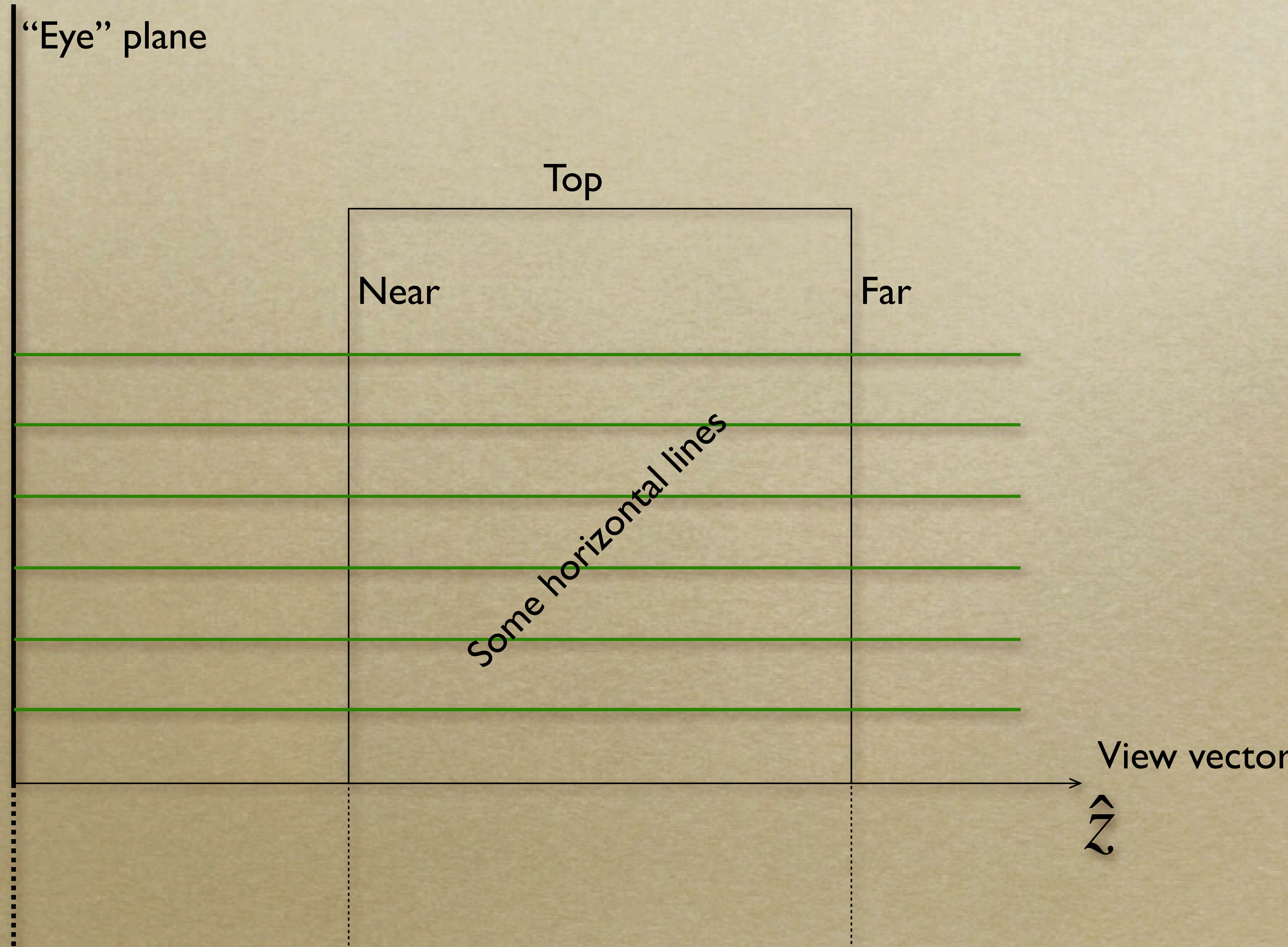


xtbook.

WRONG!

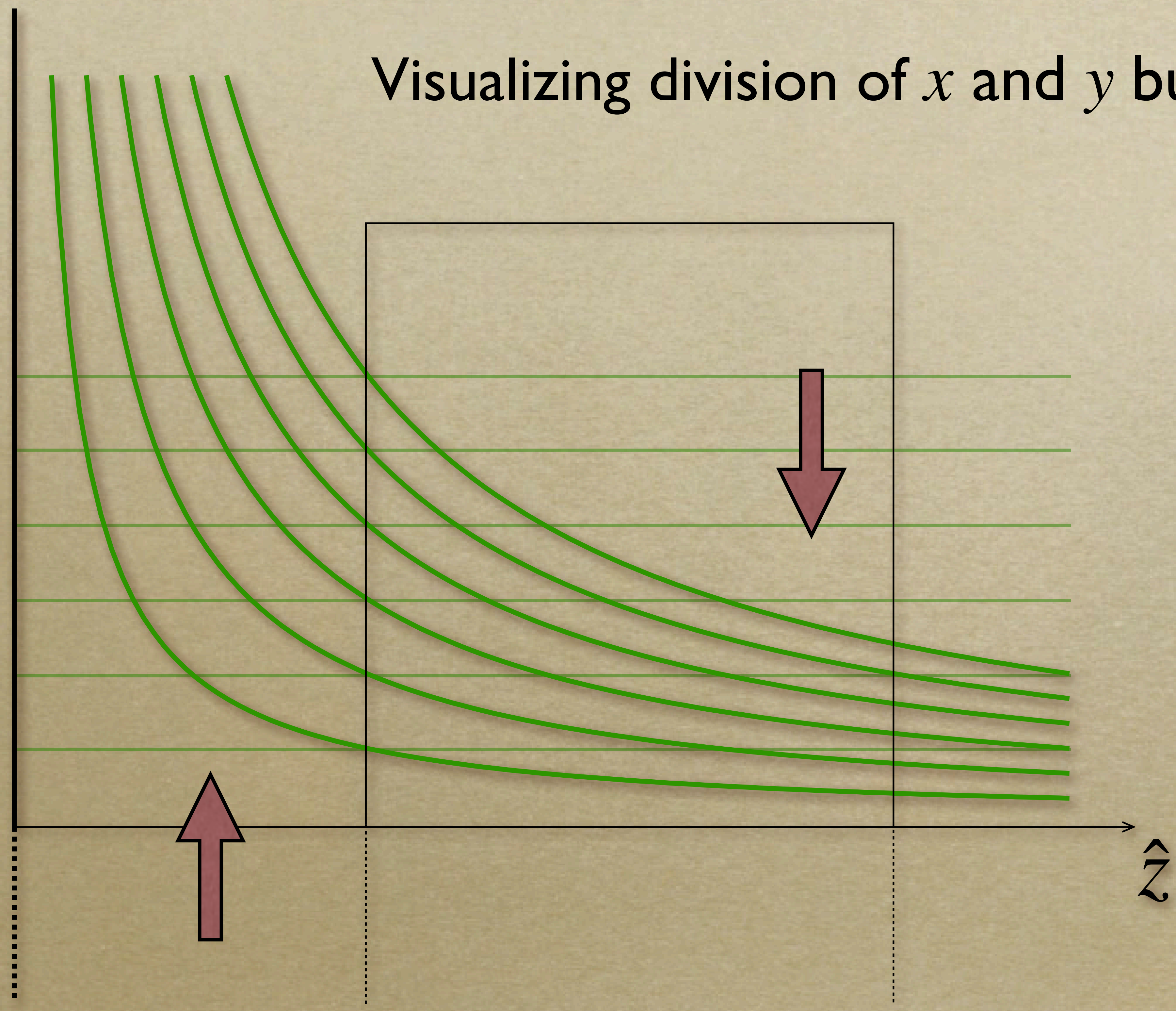
$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{i+f}{i} & f \\ 0 & 0 & \frac{-1}{i} & 0 \end{bmatrix}$$

Perspective Projection

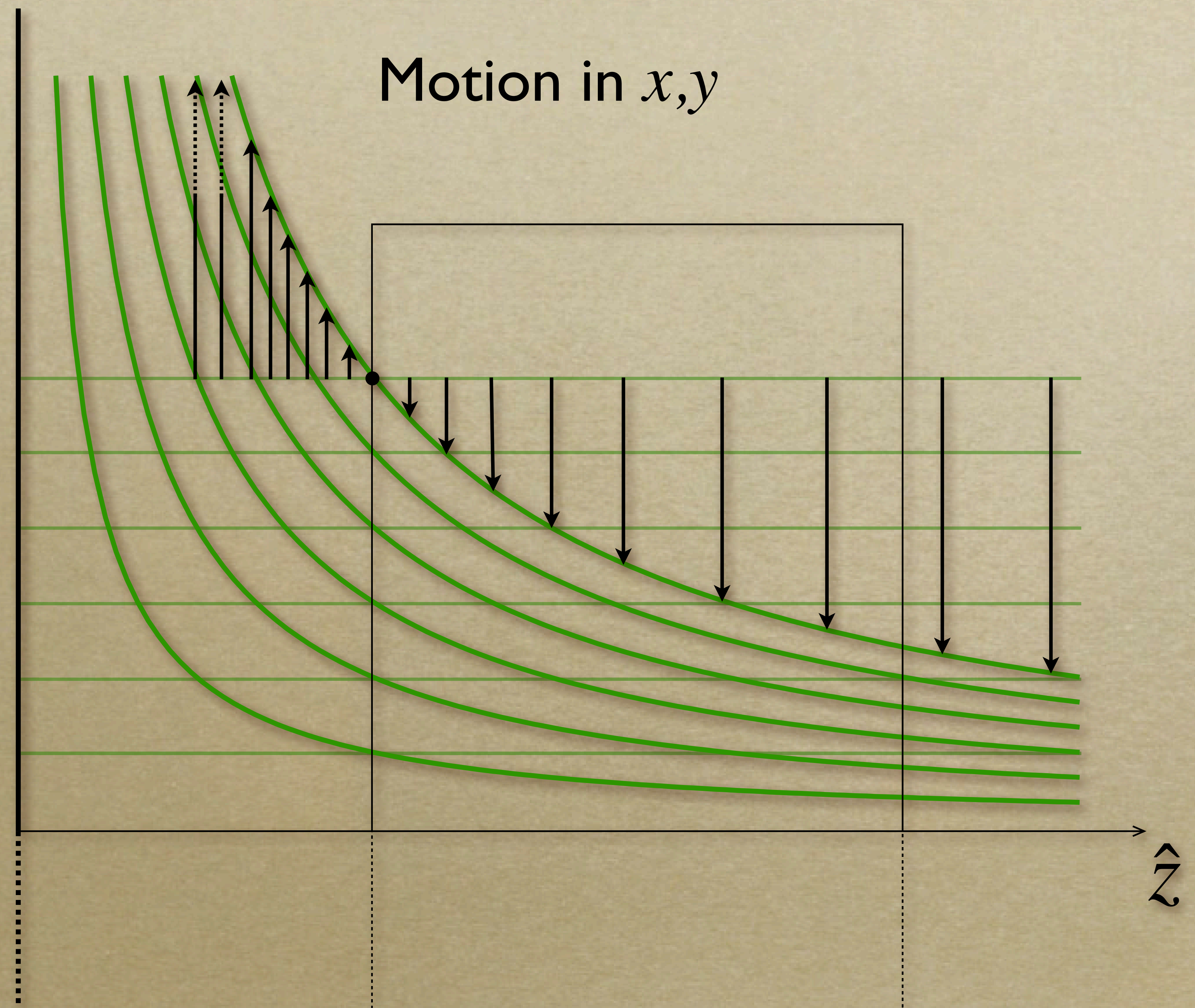


Perspective Projection

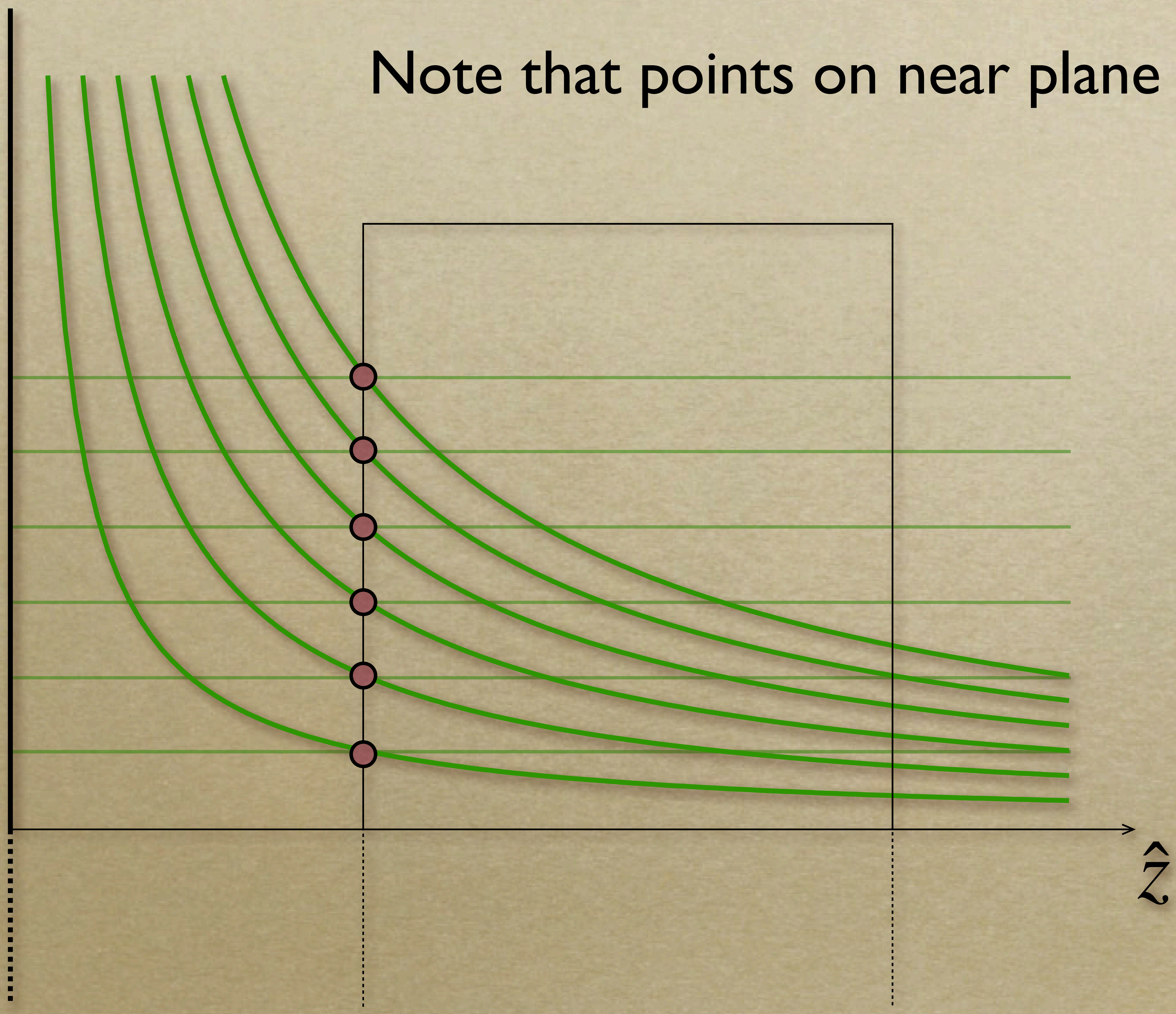
Visualizing division of x and y but not z



Perspective Projection

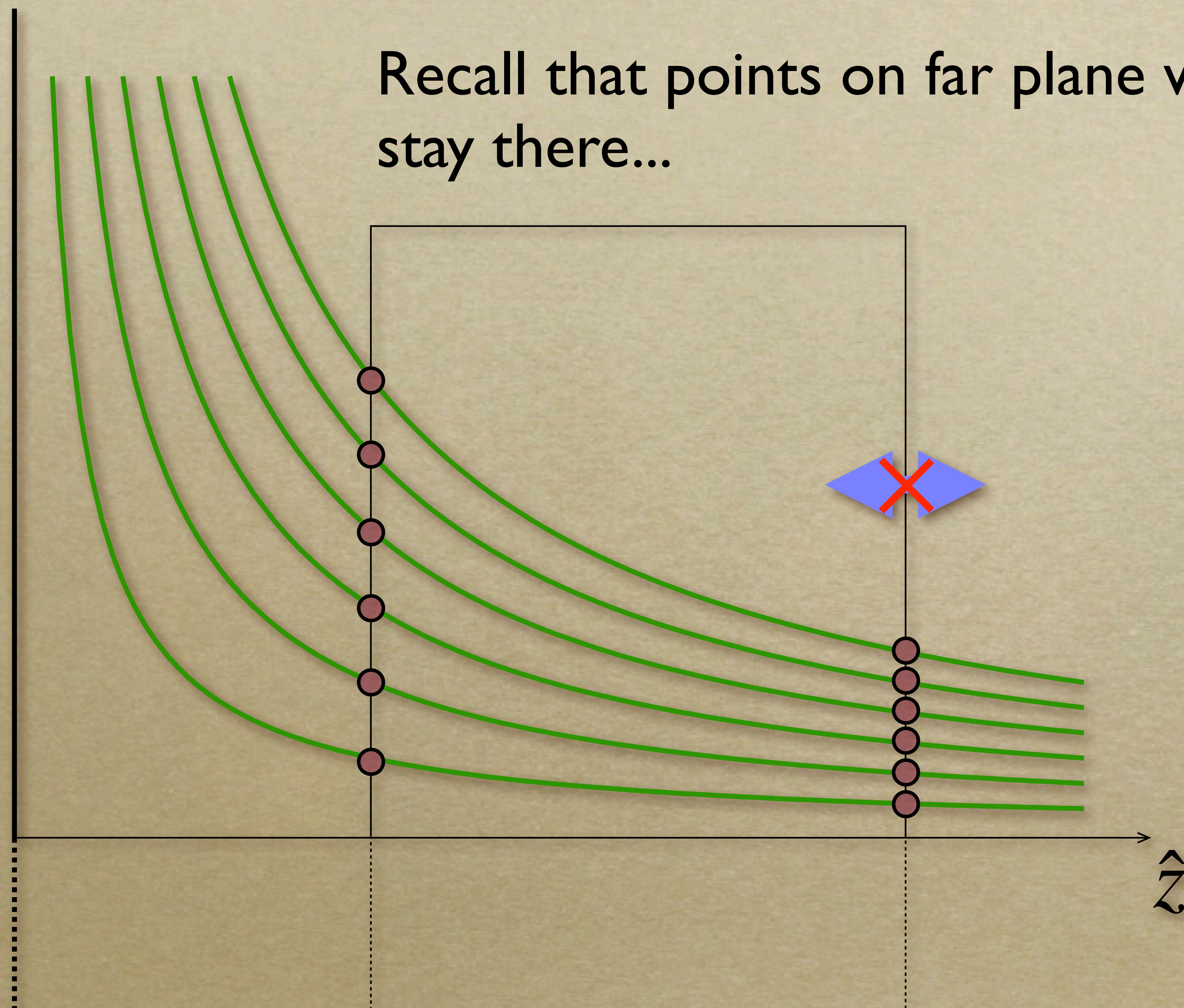


Perspective Projection



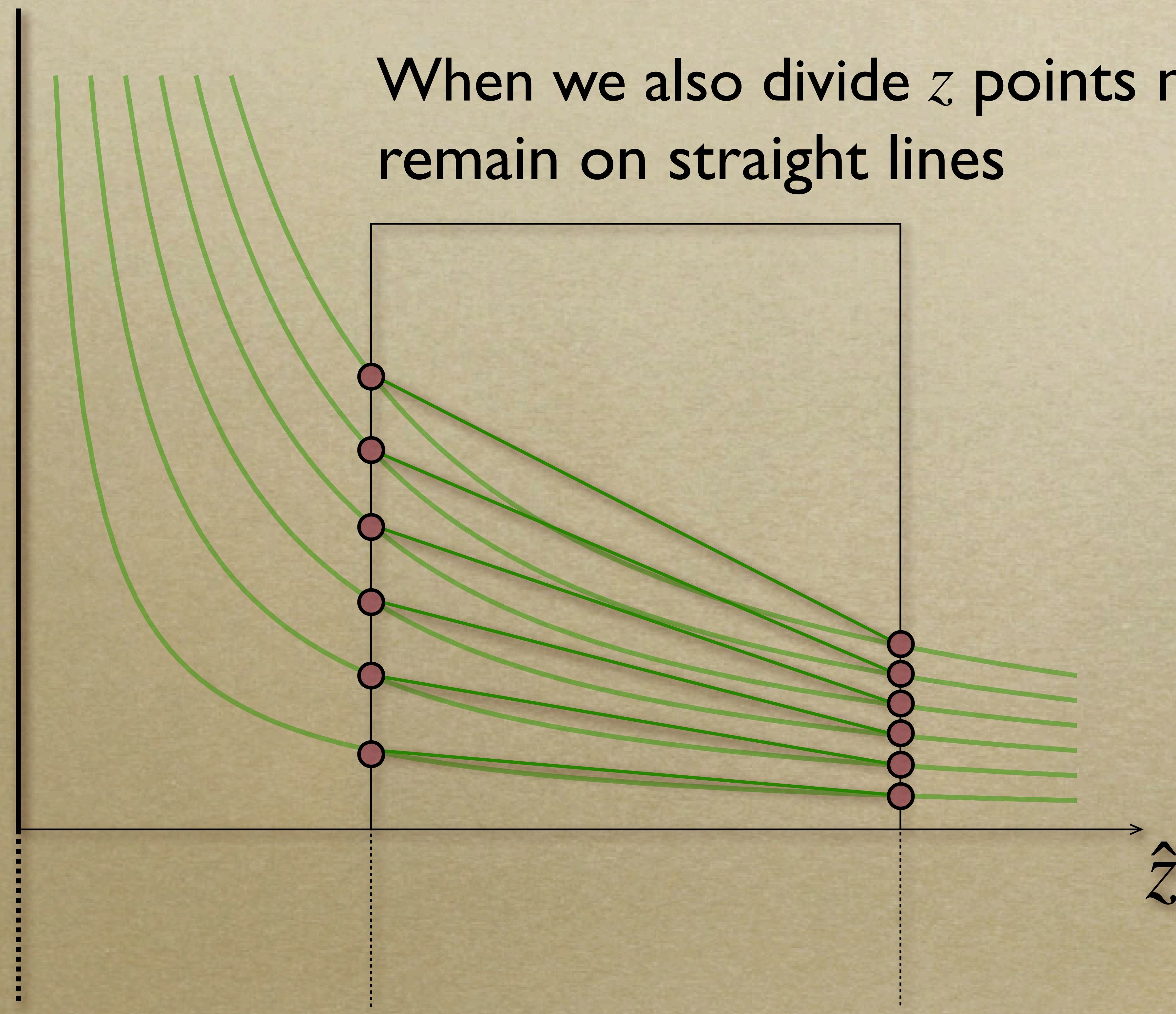
Perspective Projection

Recall that points on far plane will stay there...



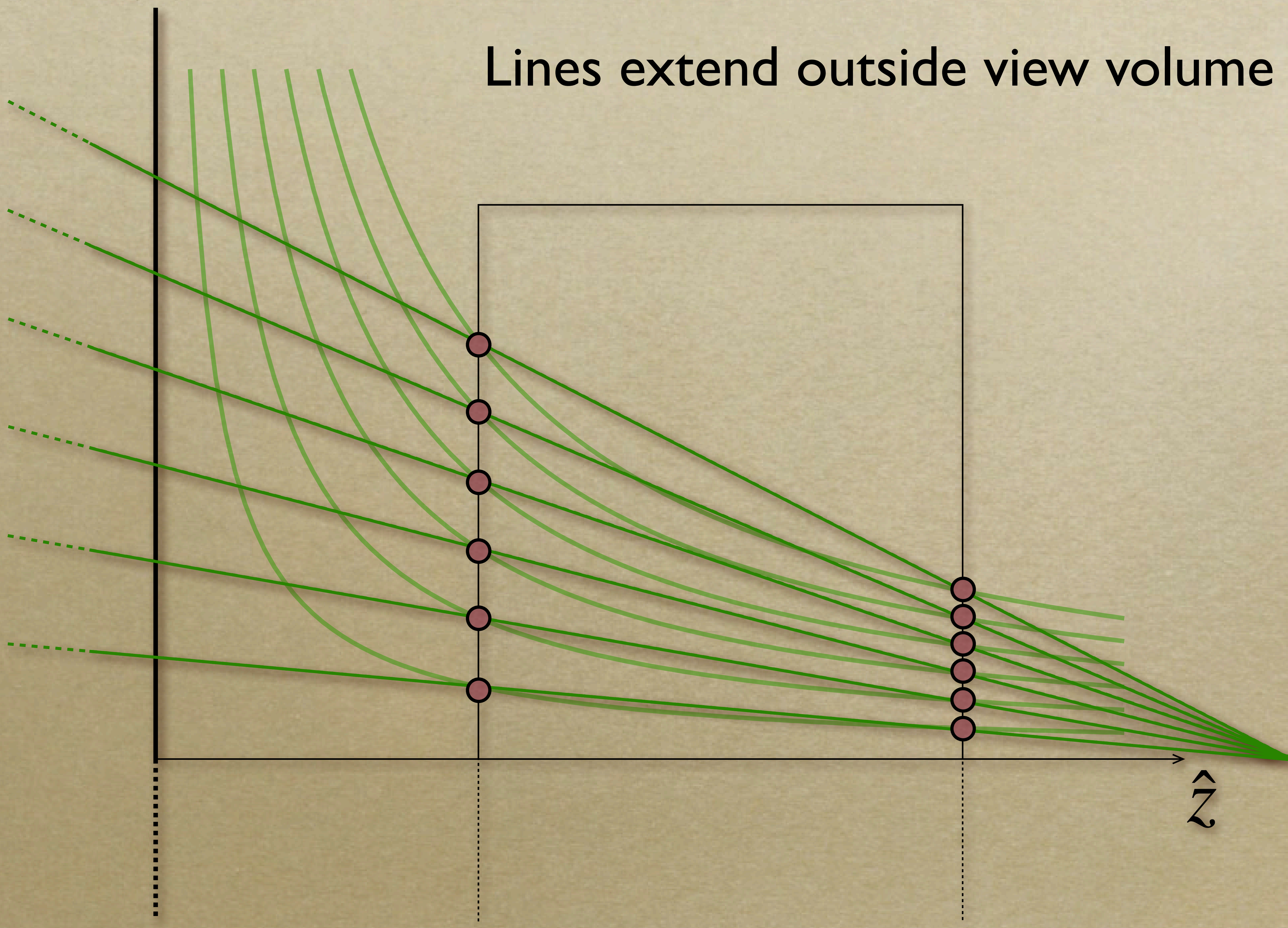
Perspective Projection

When we also divide z points must remain on straight lines

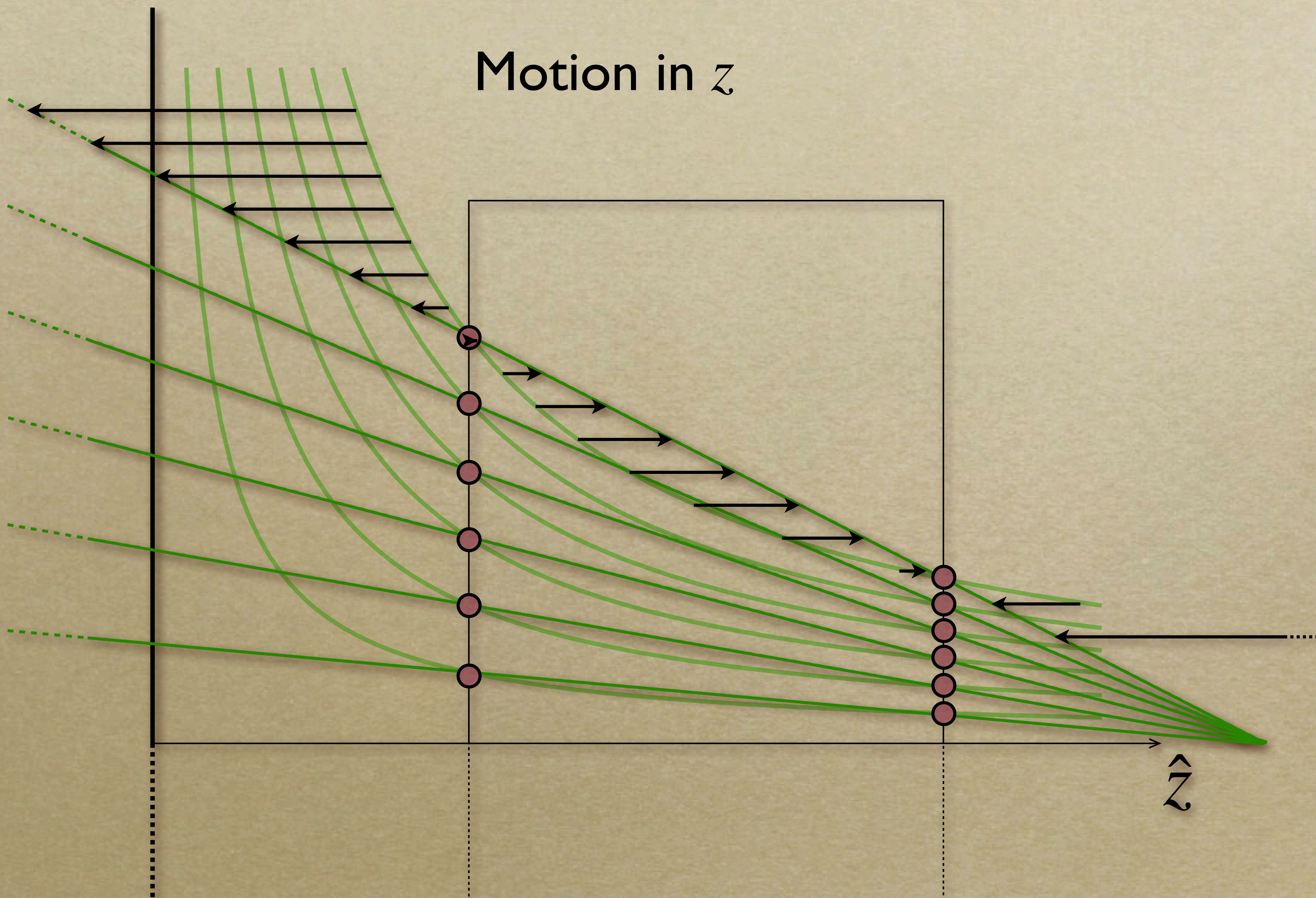


Perspective Projection

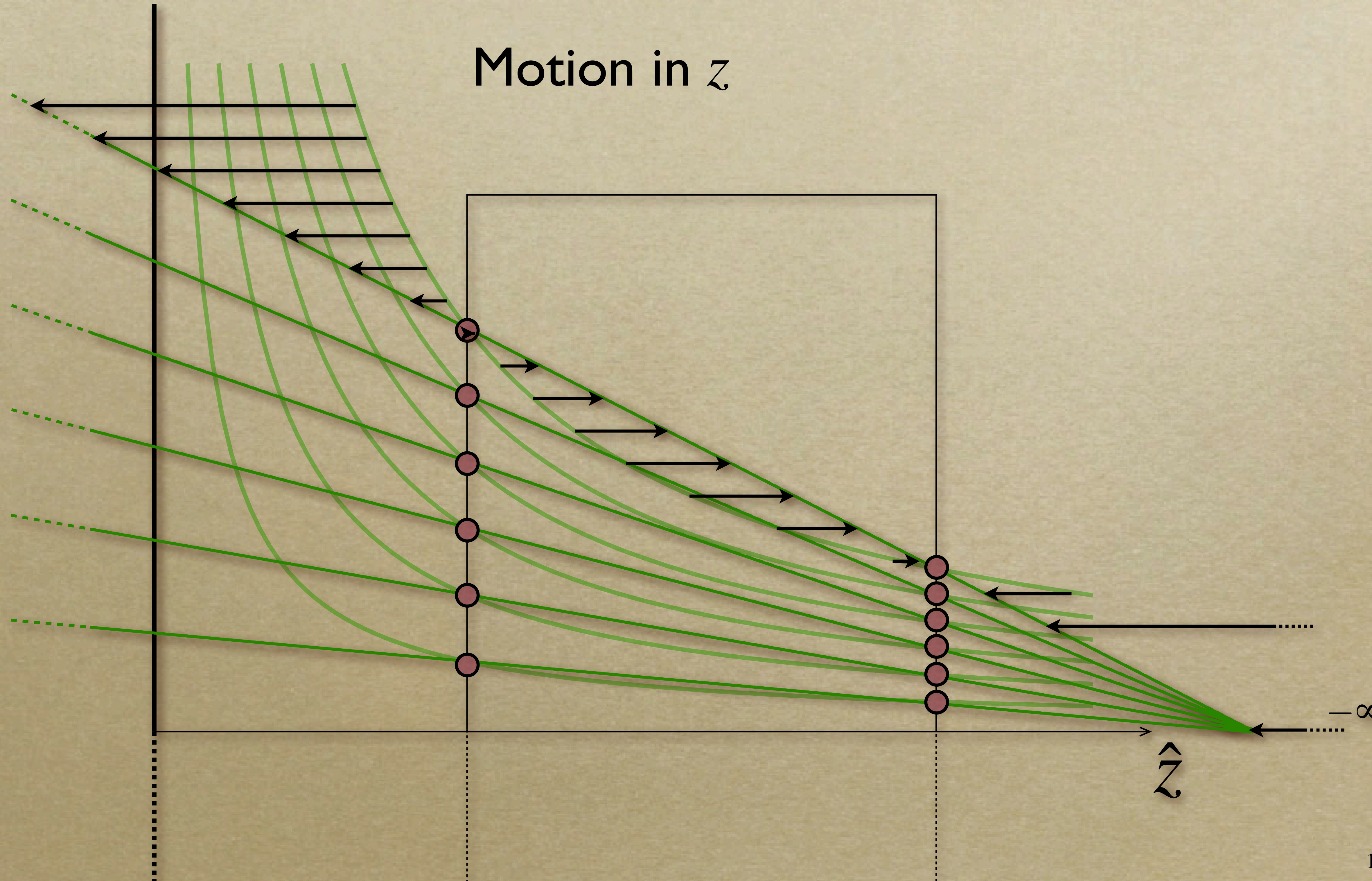
Lines extend outside view volume



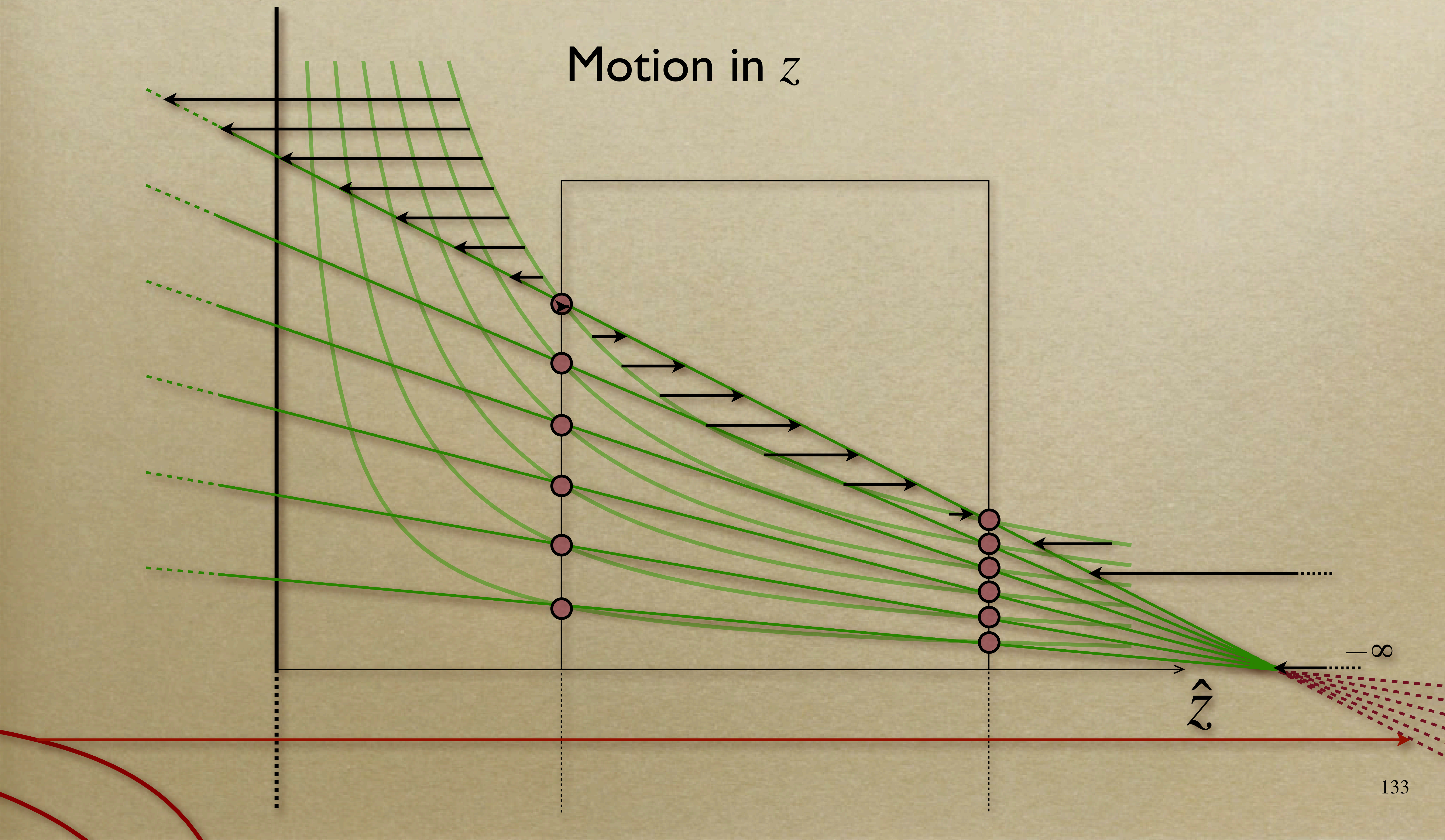
Perspective Projection



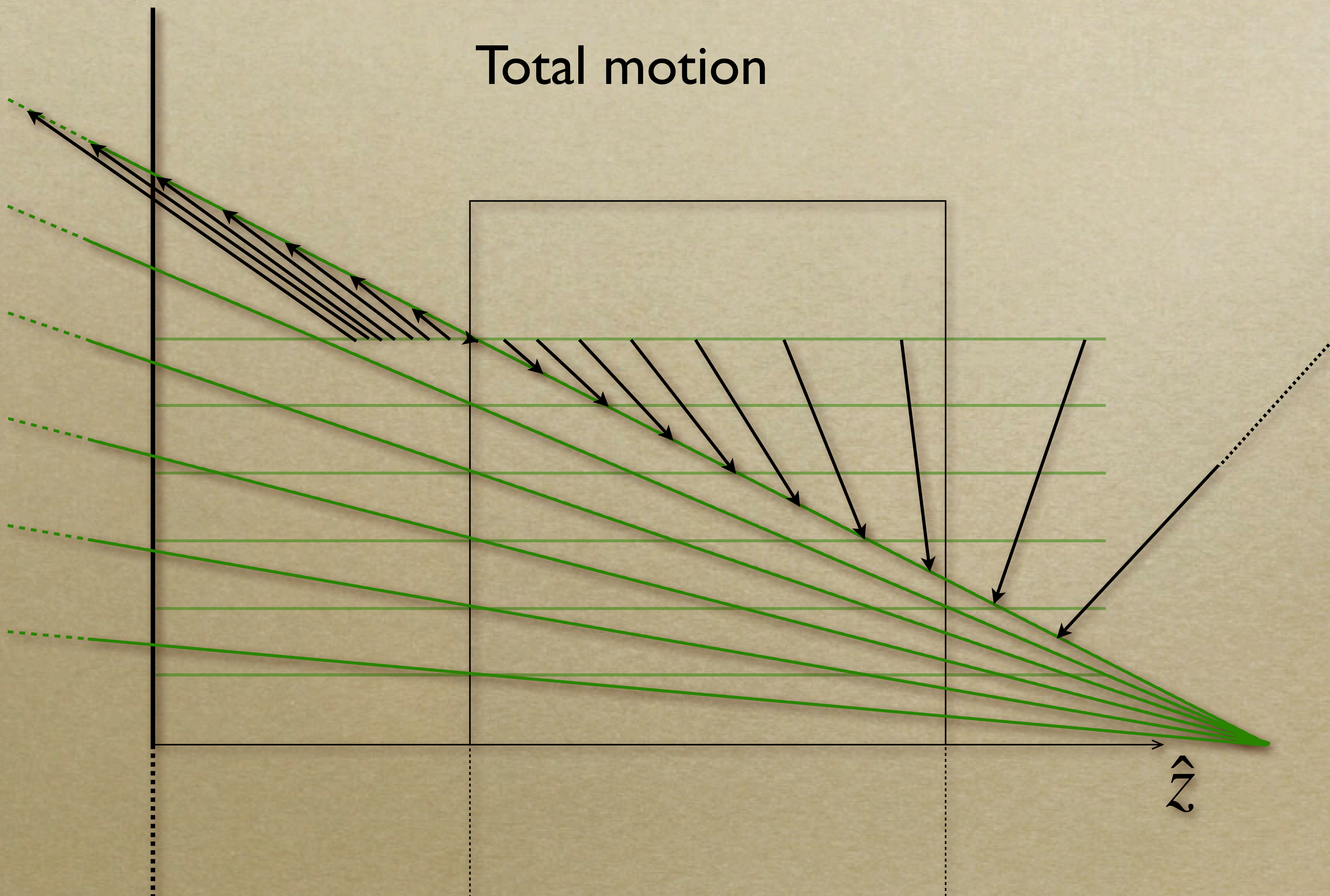
Perspective Projection



Perspective Projection

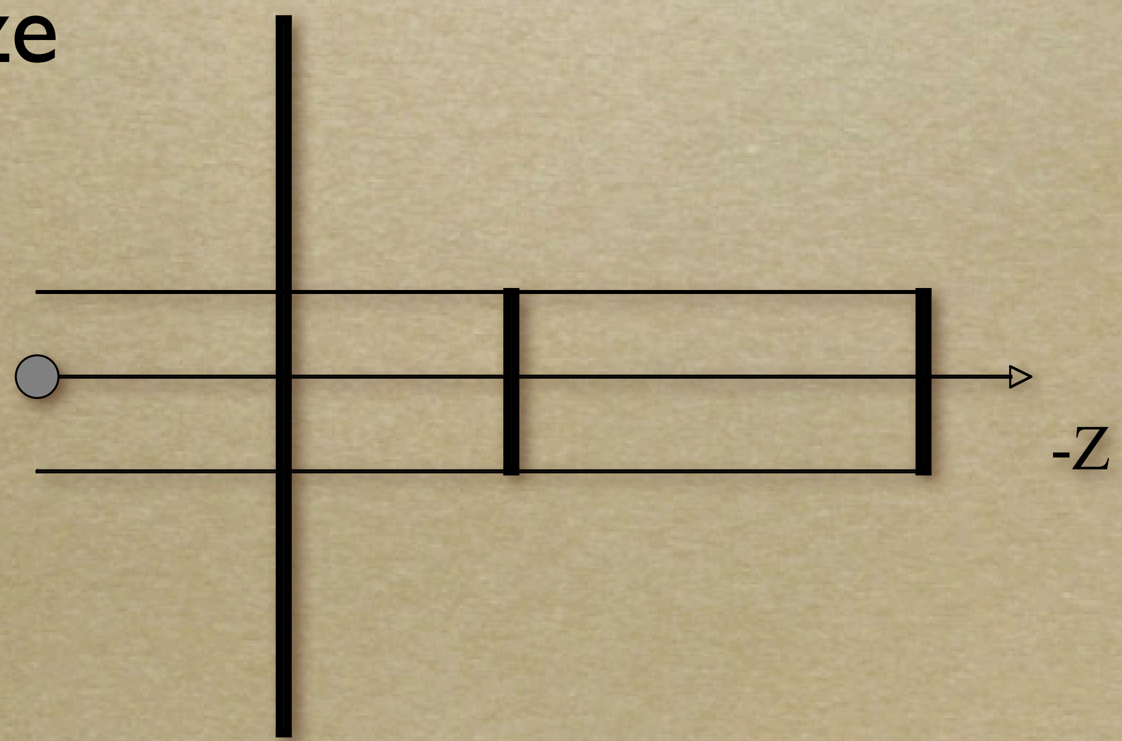


Perspective Projection



Perspective Projection

- Step 1: Translate *center* to orange
- Step 2: Rotate *view* to **-Z**, *up* to **+Y**
- Step 3: Shear center-line to **-Z** axis
- Step 4: Perspective
- Step 5: center view volume
- Step 6: scale to canonical size



Perspective Projection

- Step 1: Translate *center* to orange
- Step 2: Rotate *view* to **-Z**, *up* to **+Y**

$\}\ \mathbf{M}_v$

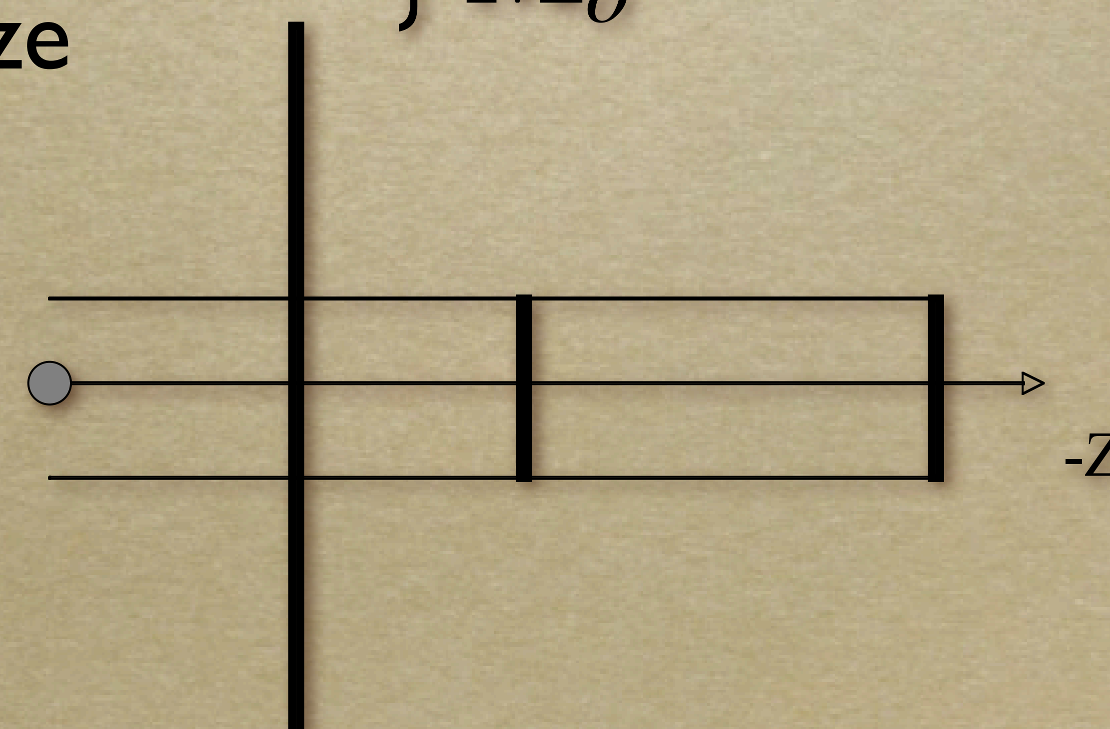
- Step 3: Shear center-line to **-Z** axis
- Step 4: Perspective

$\}\ \mathbf{M}_p$

- Step 5: center view volume
- Step 6: scale to canonical size

$\}\ \mathbf{M}_o$

$$\mathbf{M} = \mathbf{M}_o \cdot \mathbf{M}_p \cdot \mathbf{M}_v$$



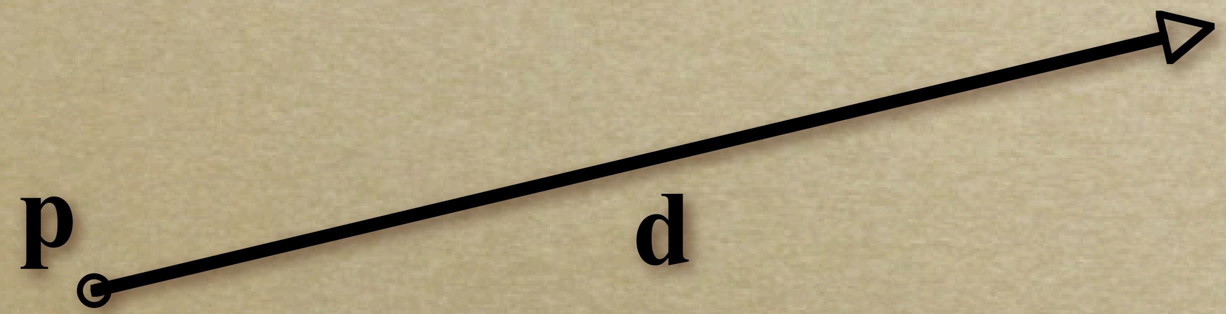
Perspective Projection

- There are other ways to set up the projection matrix
  - View plane at $z=0$ zero
  - Looking down another axis
  - *etc...*
- Functionally equivalent

Vanishing Points

- Consider a ray:

$$\mathbf{r}(t) = \mathbf{p} + t\,\mathbf{d}$$



Vanishing Points

- Ignore $\mathbf{Z}$ part of matrix
- $\mathbf{X}$ and $\mathbf{Y}$ will give location in image plane
- Assume image plane at z=-i

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ & \text{whatever} & & \\ 0 & 0 & -1 & 0 \end{bmatrix} \longrightarrow \begin{bmatrix} I_x \\ I_y \\ I_w \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

Vanishing Points

$$\begin{bmatrix} I_x \\ I_y \\ I_w \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} x \\ y \\ -z \end{bmatrix}$$

$$\begin{bmatrix} I_x / I_w \\ I_y / I_w \end{bmatrix} = \begin{bmatrix} -x/z \\ -y/z \end{bmatrix}$$

Vanishing Points

- Assume $d_z = -1$

$$\begin{bmatrix} I_x / I_w \\ I_y / I_w \end{bmatrix} = \begin{bmatrix} -x/z \\ -y/z \end{bmatrix} = \begin{bmatrix} \dfrac{p_x + td_x}{-p_z + t} \\ \dfrac{p_y + td_y}{-p_z + t} \end{bmatrix}$$

$$\lim_{t \to \pm\infty} = \begin{bmatrix} d_x \\ d_y \end{bmatrix}$$

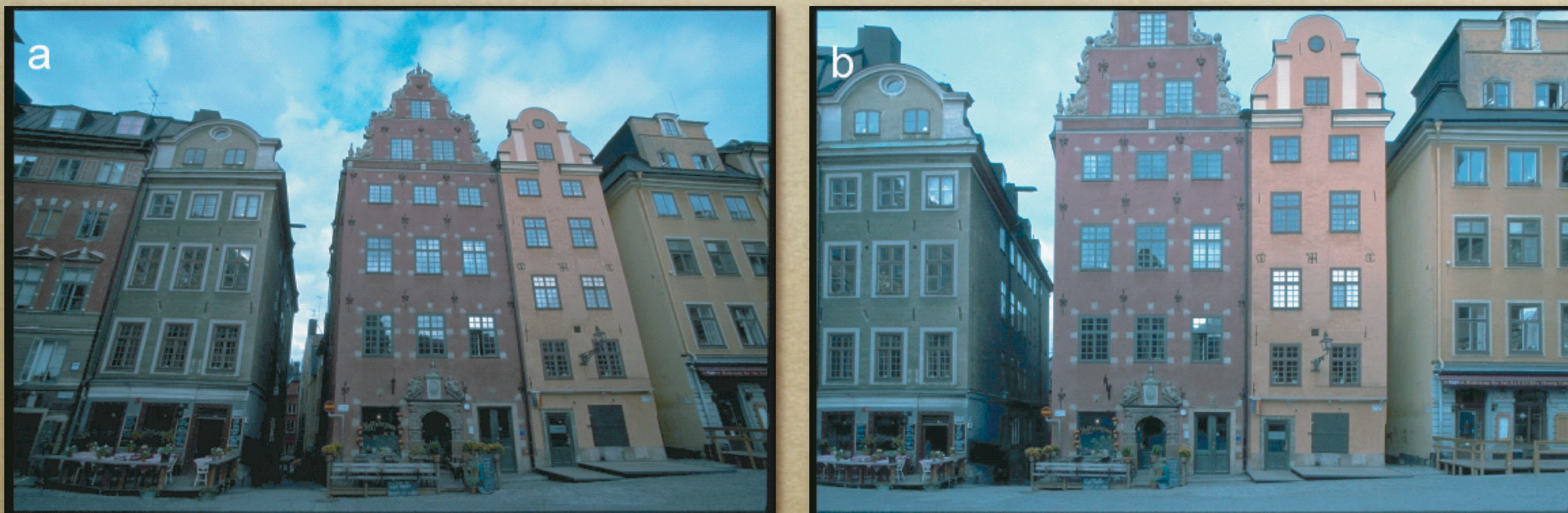
Vanishing Points

$$\lim_{t \to \pm\infty} = \begin{bmatrix} d_x \\ d_y \end{bmatrix}$$

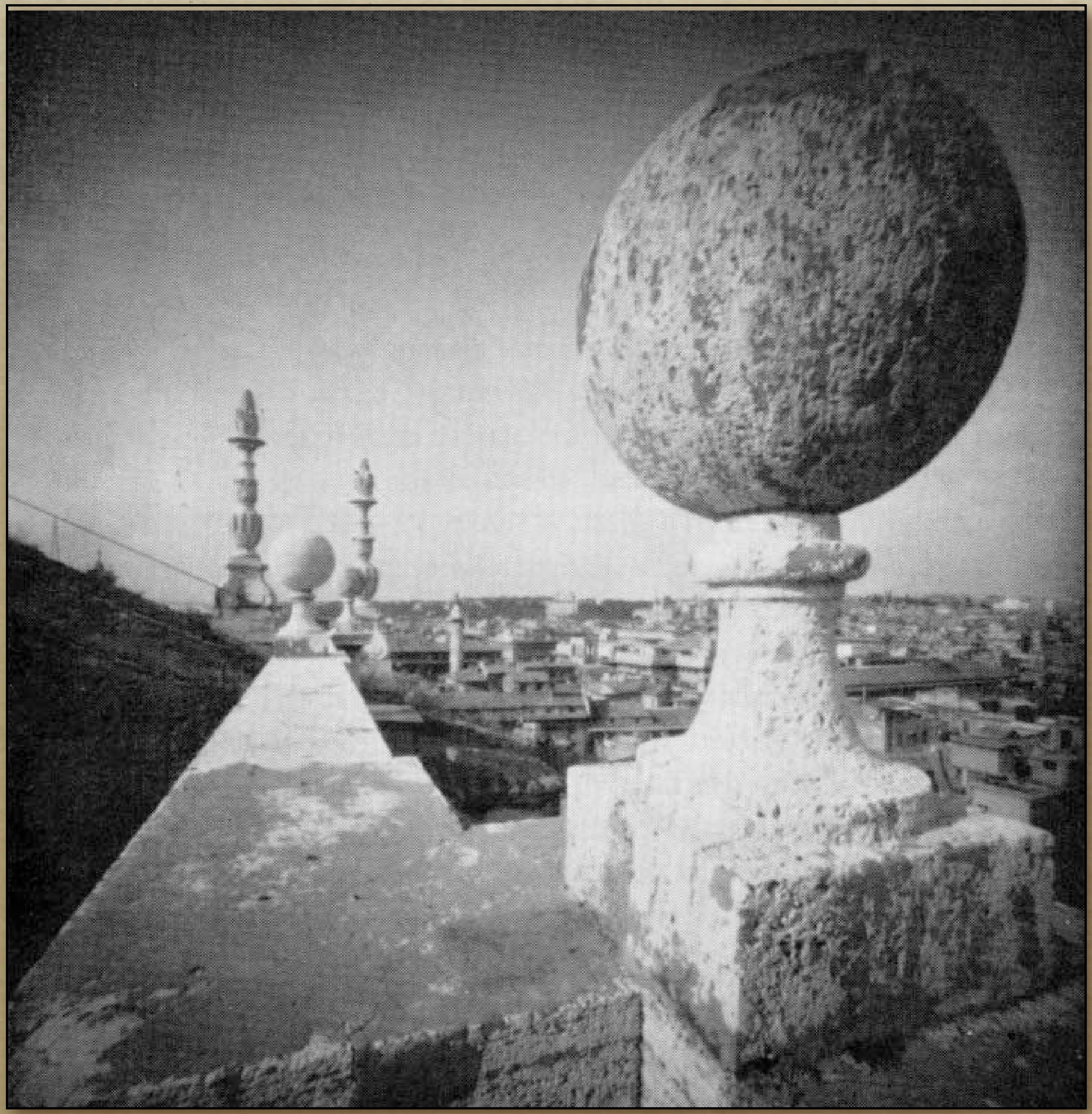
- All lines in direction $\mathbf{d}$ converge to same point in the image plane -- the vanishing point
- Every point in plane is a v.p. for some set of lines
- Lines parallel to image plane ($d_z = 0$) vanish at infinity

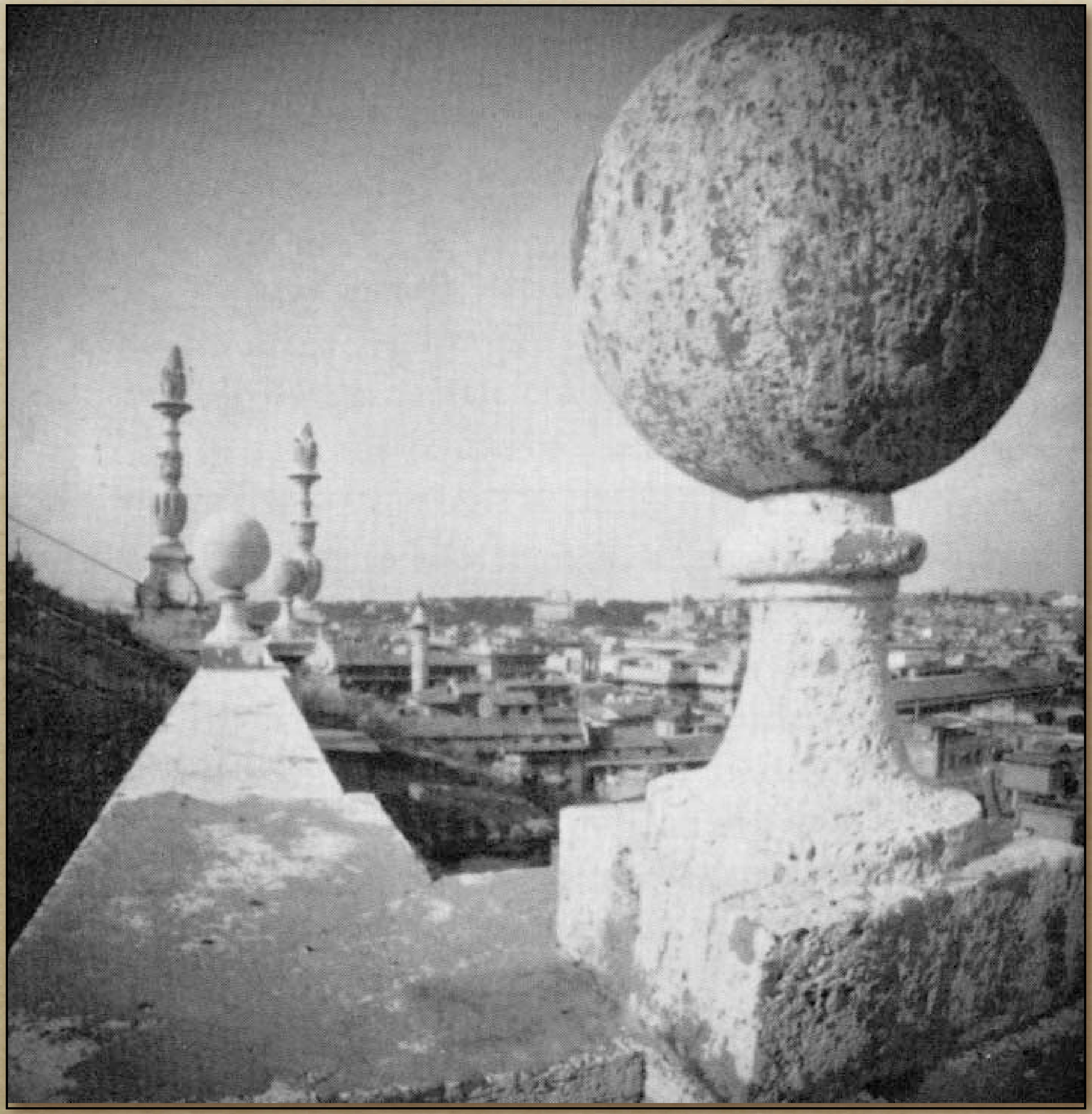
What's a horizon?

Perspective Tricks



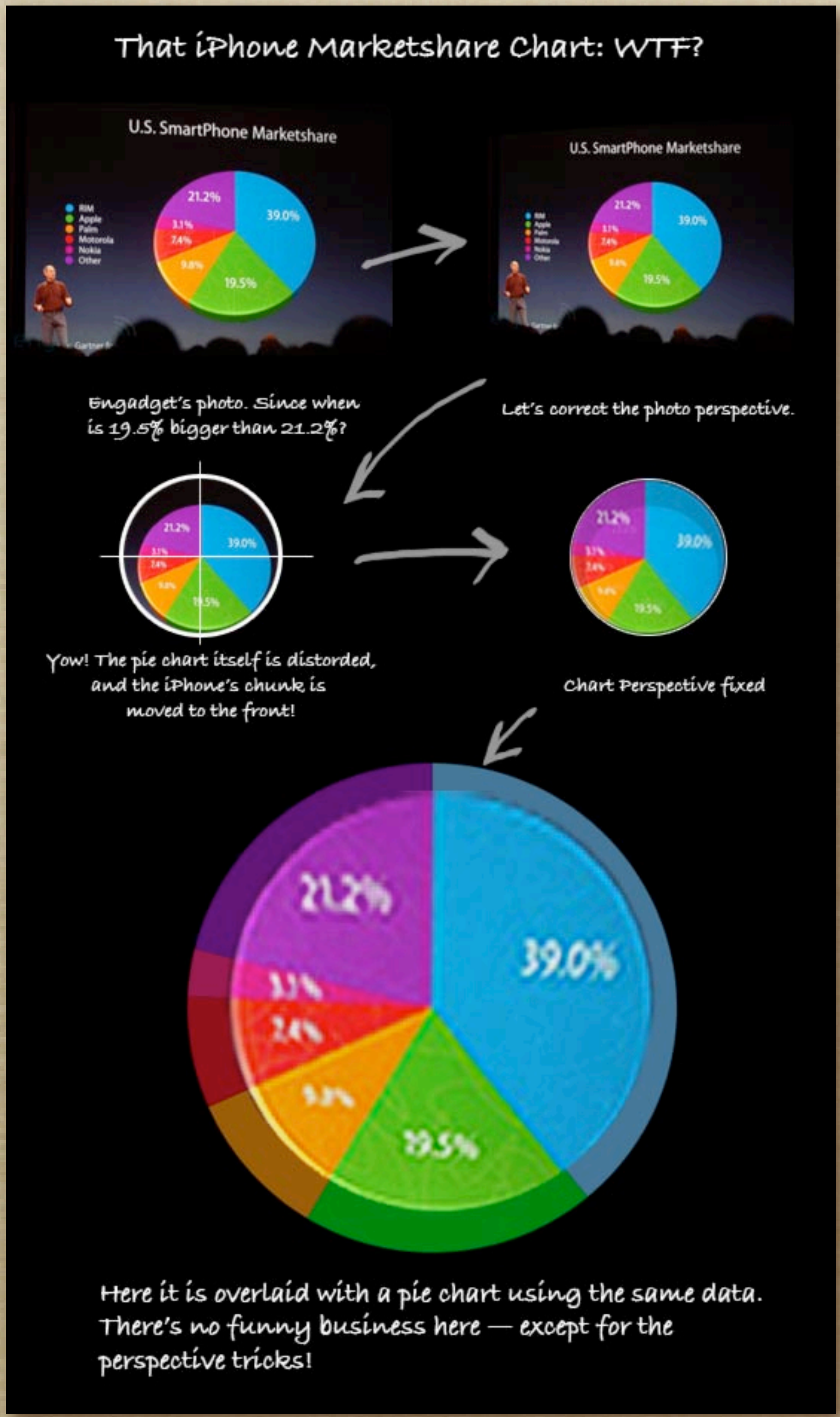
Right Looks Wrong (Sometimes)





From *Correction of Geometric Perceptual Distortions in Pictures*, Zorin and Barr SIGGRAPH 1995

Right Looks Wrong (Sometimes)



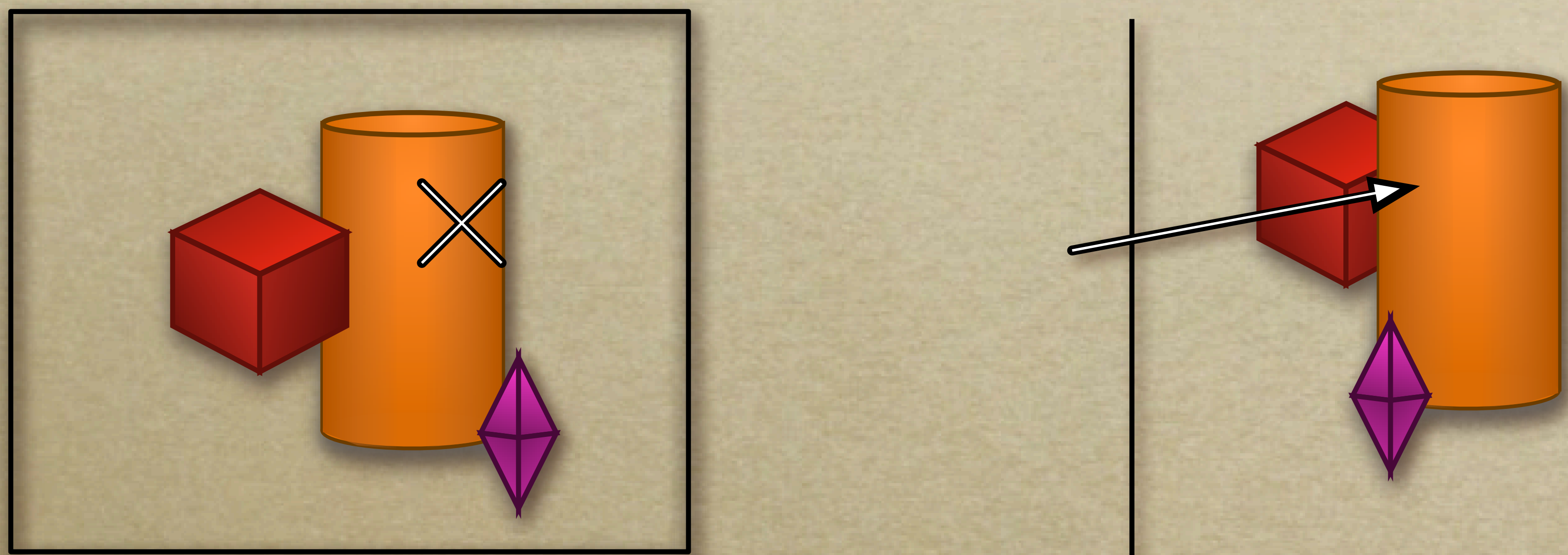
From WIRED Magazine

Strangeness

The Ambassadors
by Hans Holbein the Younger

Ray Picking

- Pick object by picking point on screen



- Compute ray from pixel coordinates.

Ray Picking

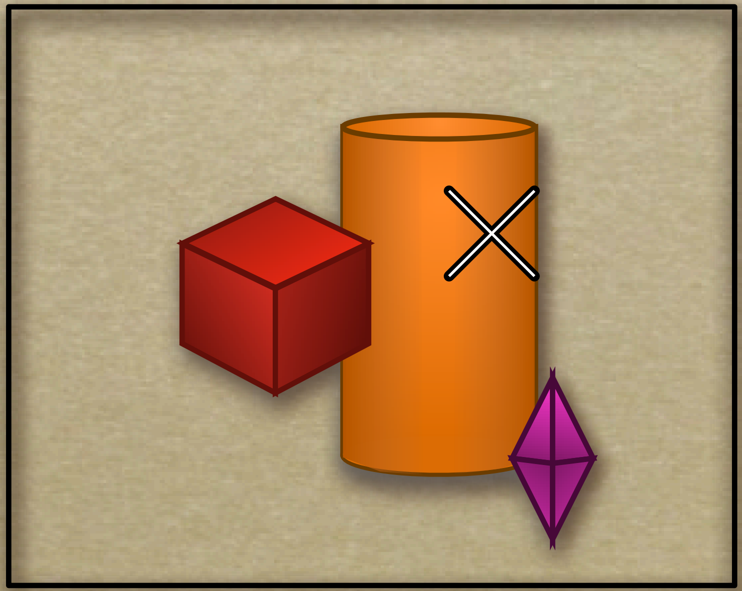
- Transform from World to Screen is:

$$\begin{bmatrix} I_x \\ I_y \\ I_z \\ I_w \end{bmatrix} = \mathbf{M} \begin{bmatrix} W_x \\ W_y \\ W_z \\ W_w \end{bmatrix}$$

- Inverse:

$$\begin{bmatrix} W_x \\ W_y \\ W_z \\ W_w \end{bmatrix} = \mathbf{M}^{-1} \begin{bmatrix} I_x \\ I_y \\ I_z \\ I_w \end{bmatrix}$$

- What $\mathbf{Z}$ value?



Ray Picking

- Recall that:
  - Points at z=-i stay at z=-i
  - Points at z=-f stay at z=-f

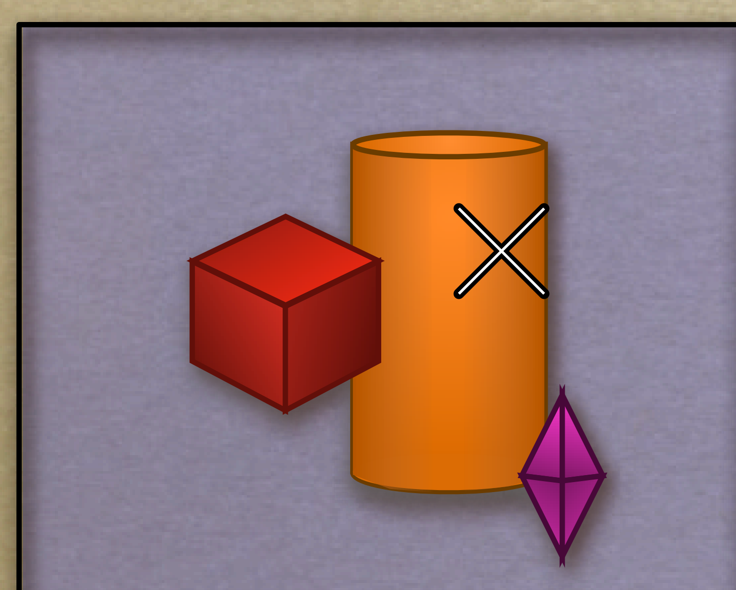
Depends on screen details, YMMV
General idea should translate...

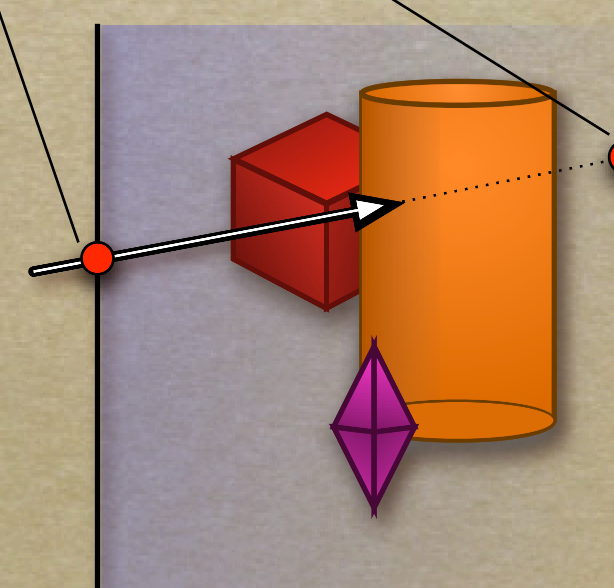
$$\mathbf{r}(t) = \mathbf{p} + t\,\mathbf{d}$$

$$\mathbf{r}(t) = \mathbf{a}_w + t(\mathbf{b}_w - \mathbf{a}_w)$$

$$\mathbf{a}_s = [s_x, s_y, -i]$$

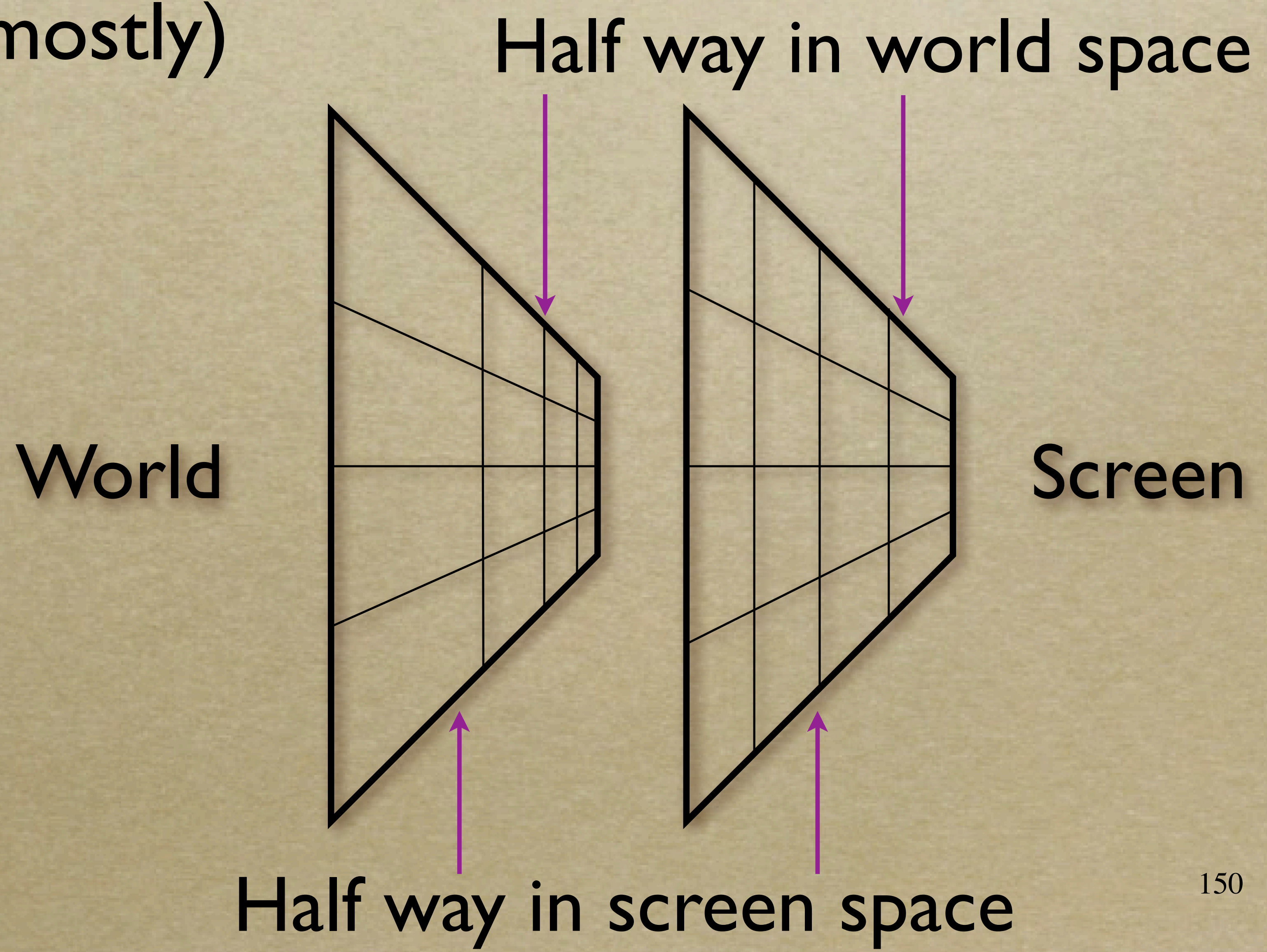
$$\mathbf{b}_s = [s_x, s_y, -f]$$



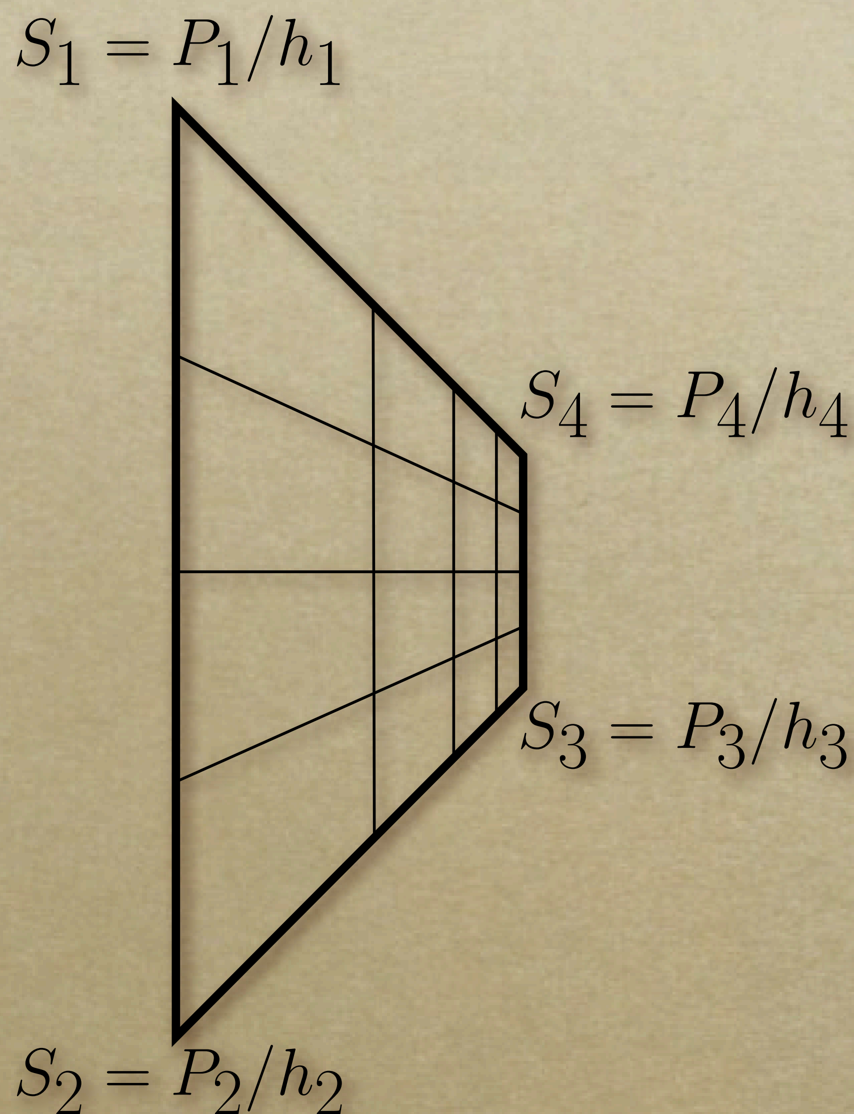


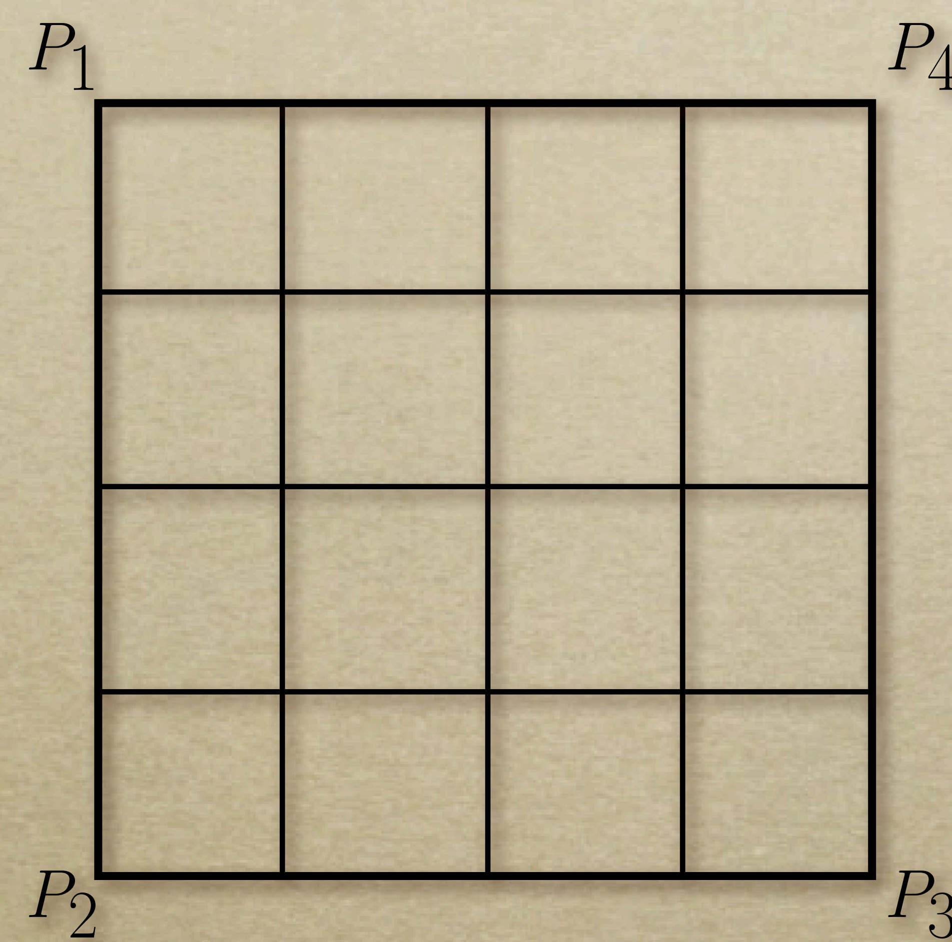
Depth Distortion

- Recall depth distortion from perspective
  - Interpolating in screen space different than in world
  - Ok, for shading (mostly)
  - Bad for texture

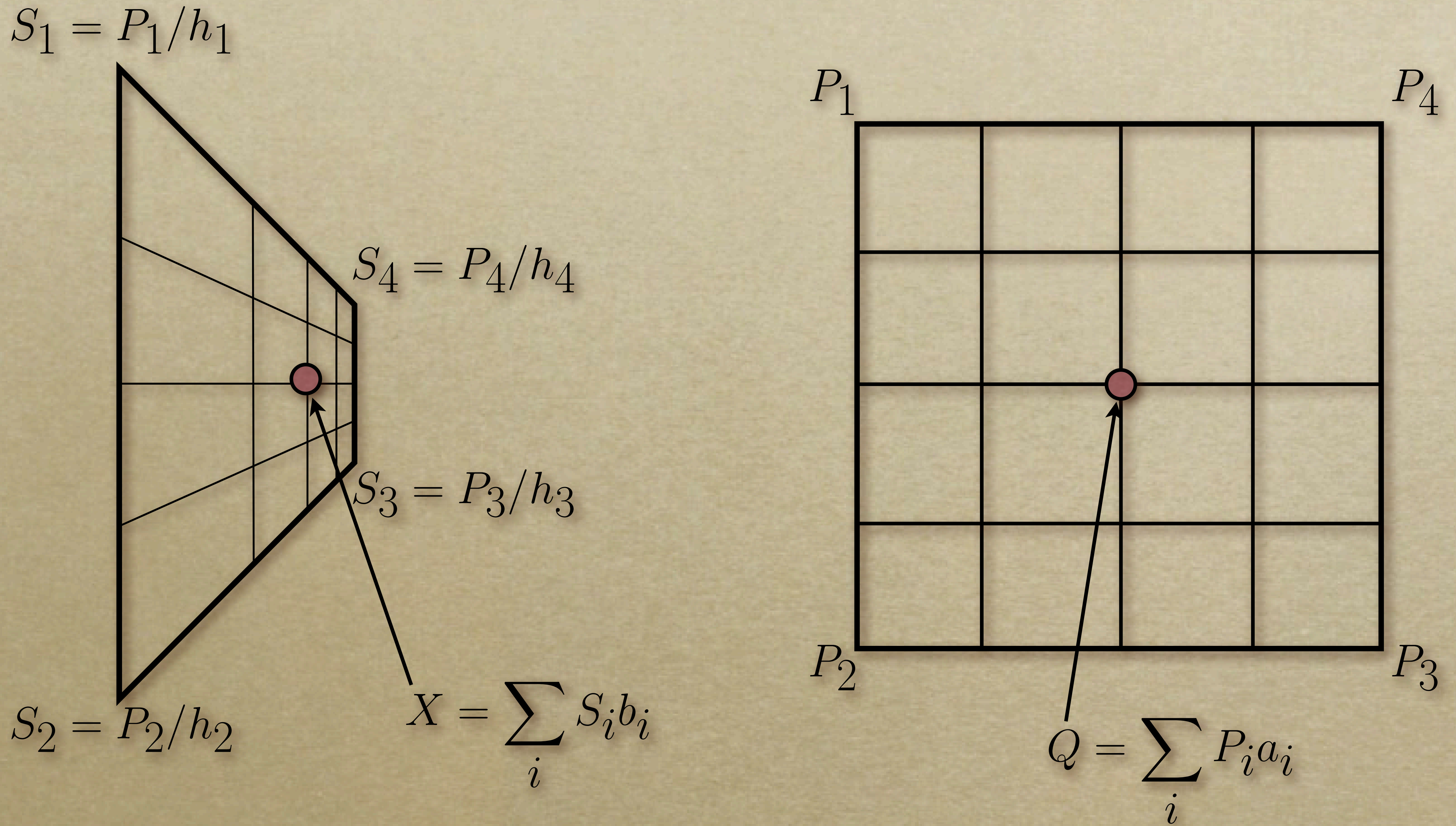


Depth Distortion



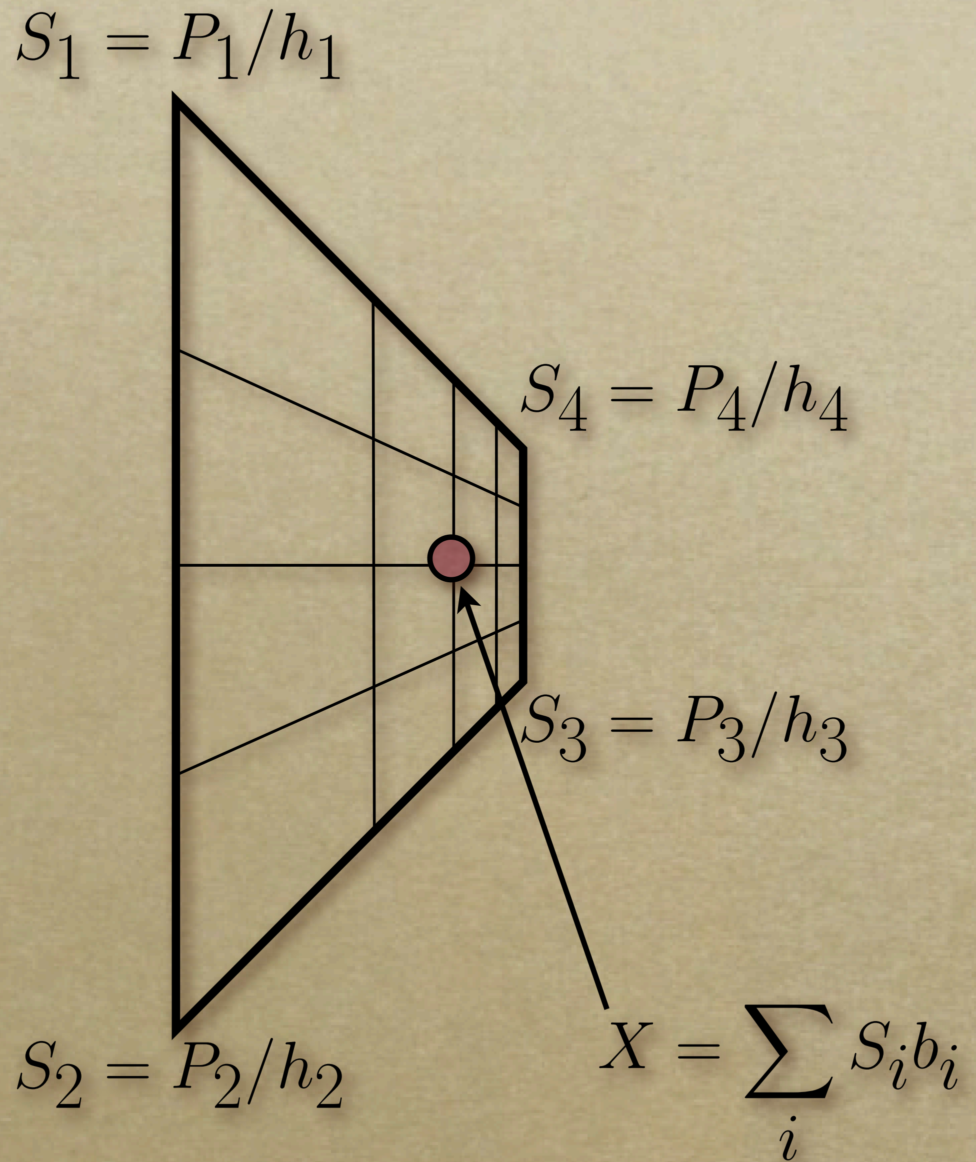


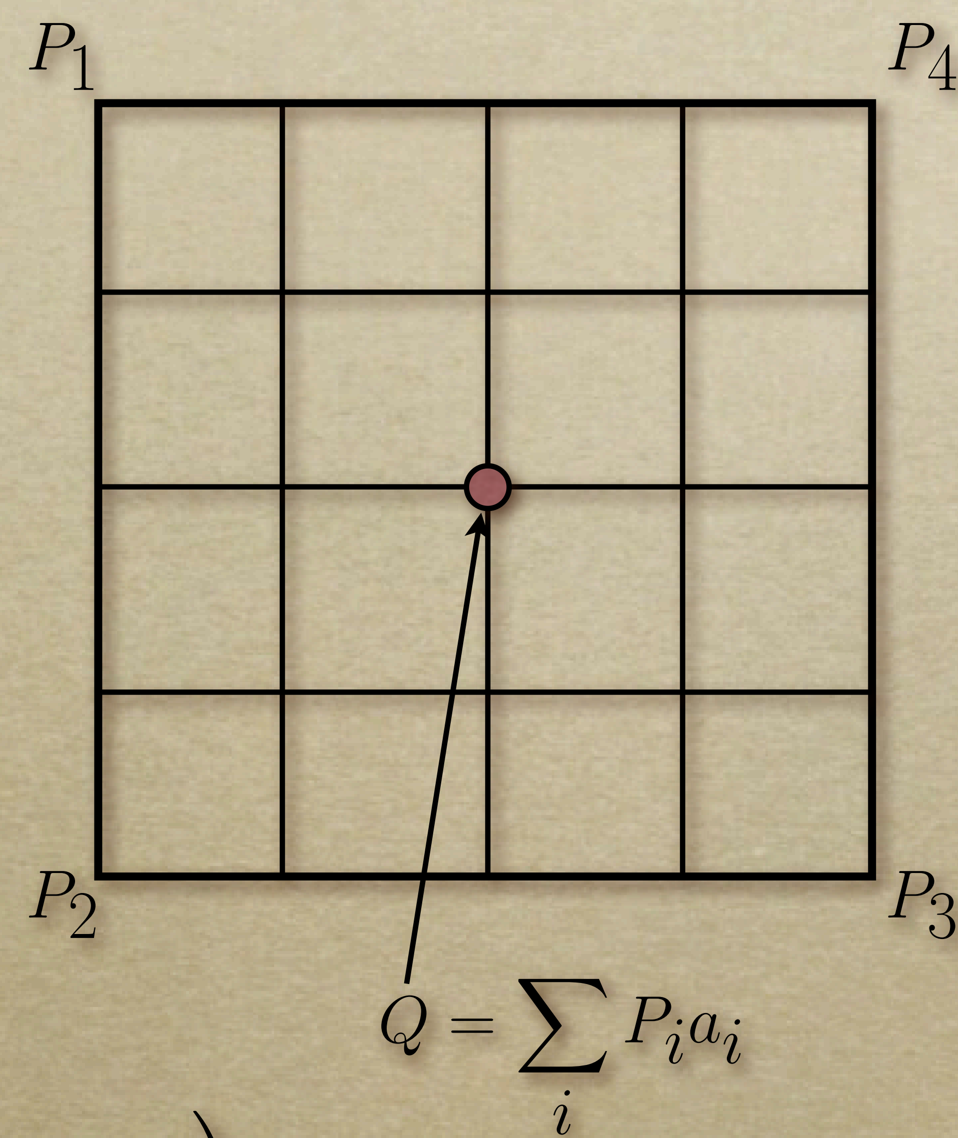
Depth Distortion



We know the S_i, P_i, and b_i, but not the a_i.

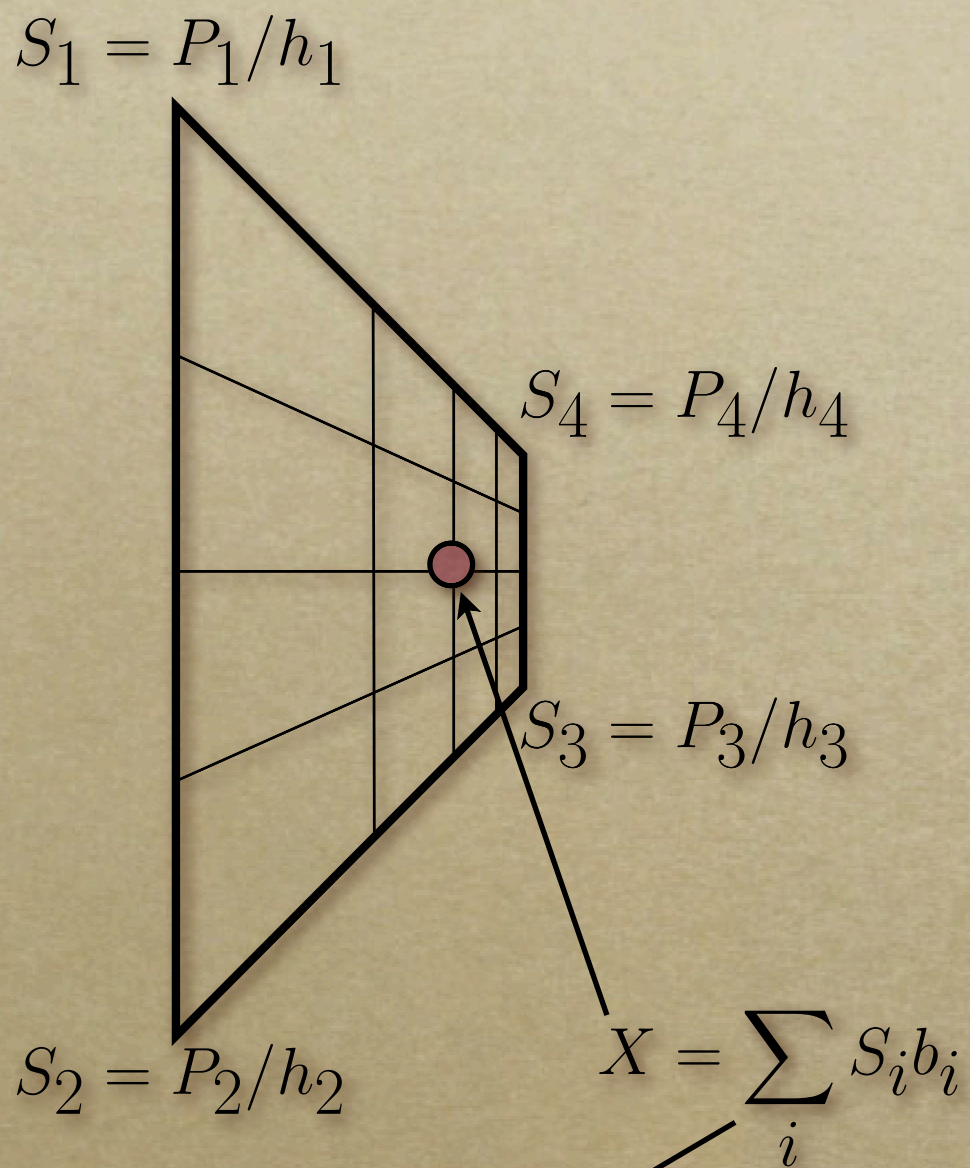
Depth Distortion

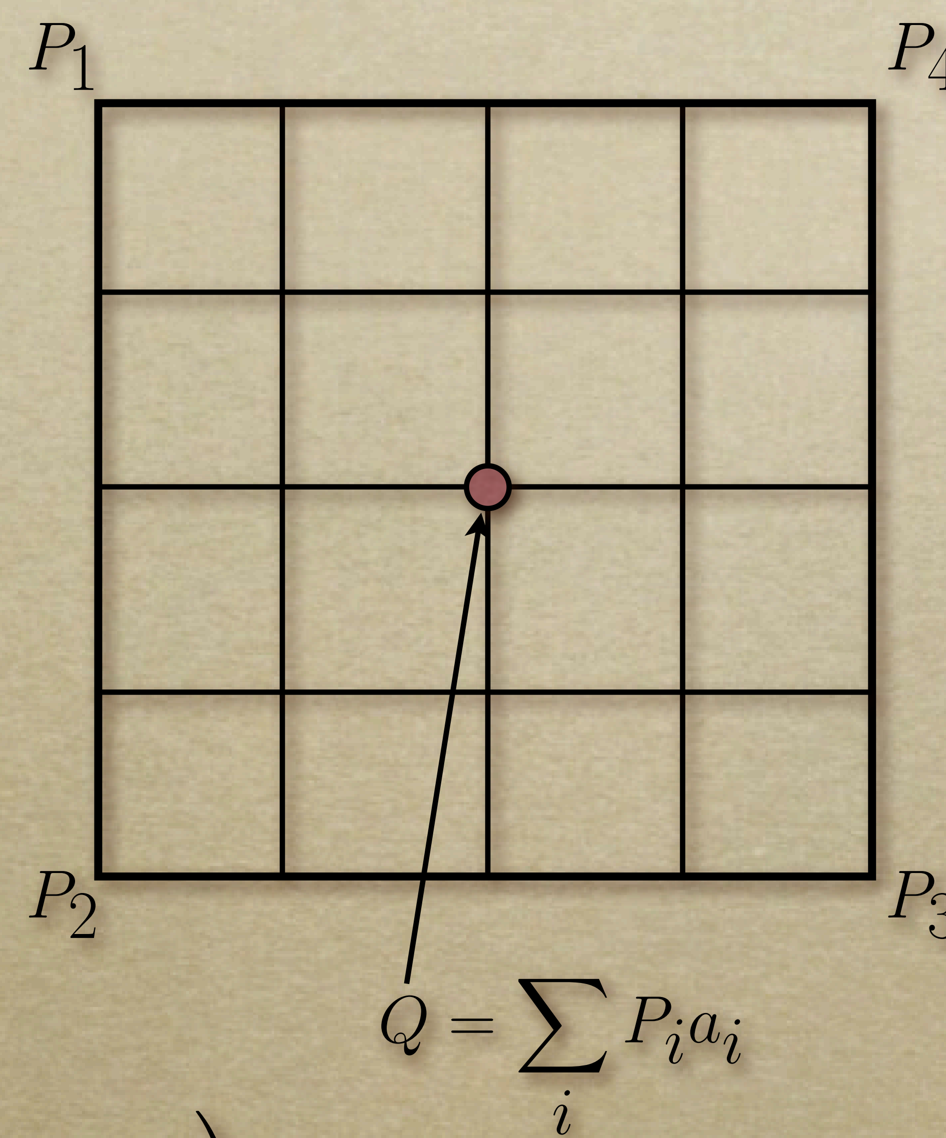




$$X = Q/h = \left(\sum_i P_i a_i\right) / \left(\sum_j h_j a_j\right)$$

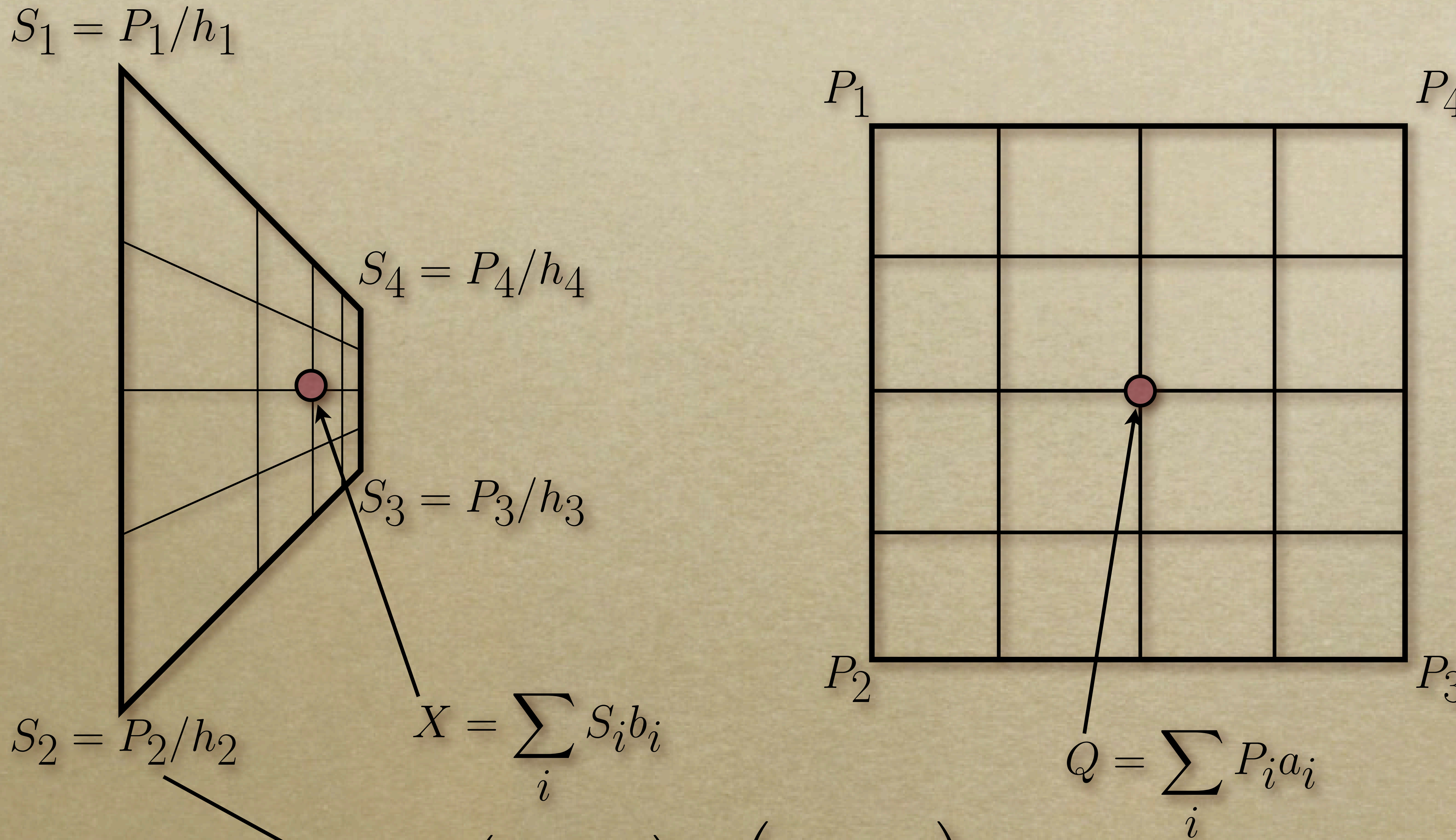
Depth Distortion





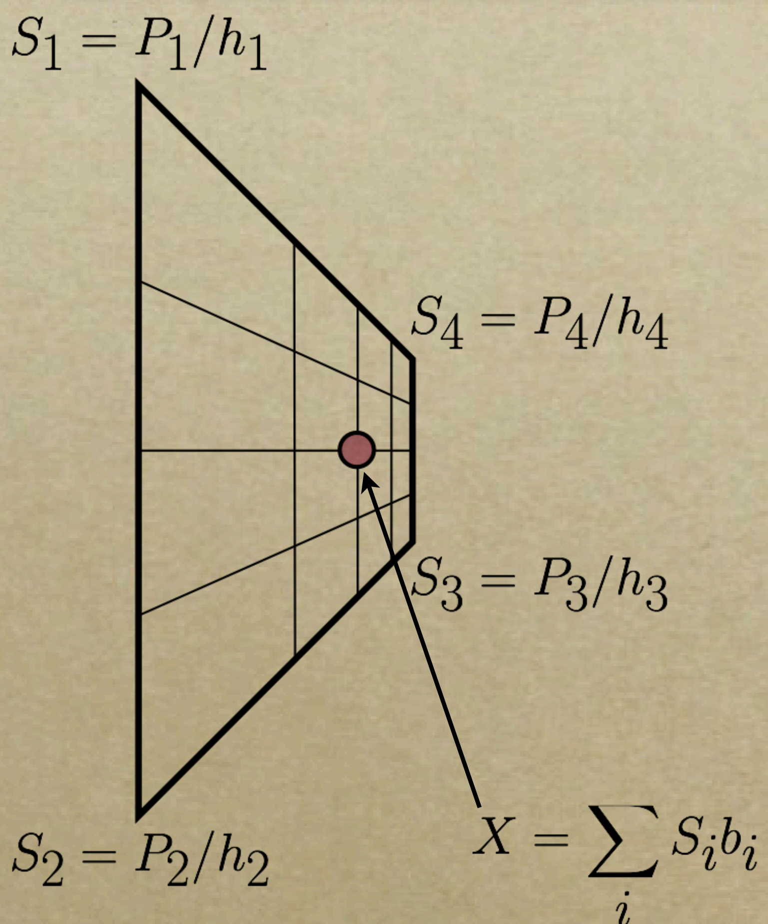
$$\sum_i S_i b_i = \left(\sum_i P_i a_i \right) / \left(\sum_j h_j a_j \right)$$

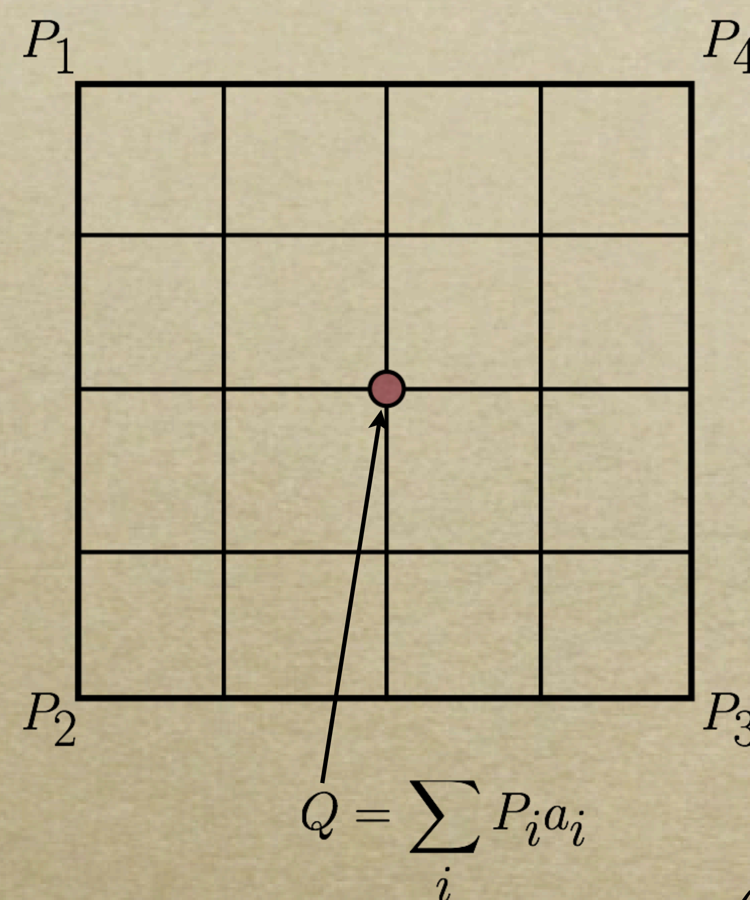
Depth Distortion



$$\sum_i P_i b_i / h_i = \left(\sum_i P_i a_i \right) / \left(\sum_j h_j a_j \right)$$

Depth Distortion



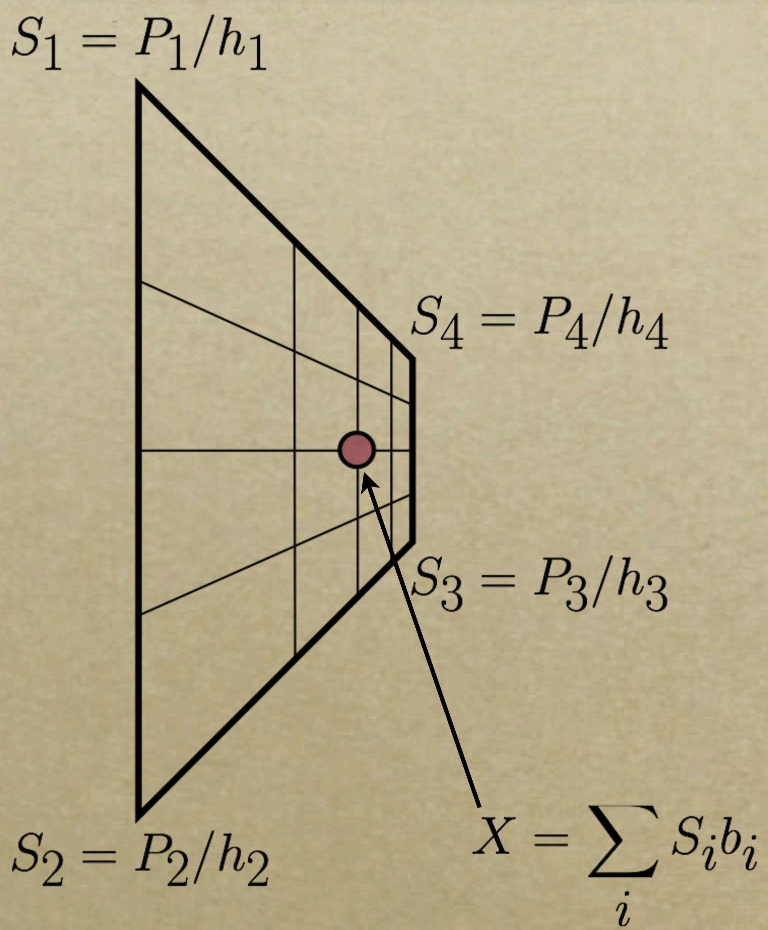


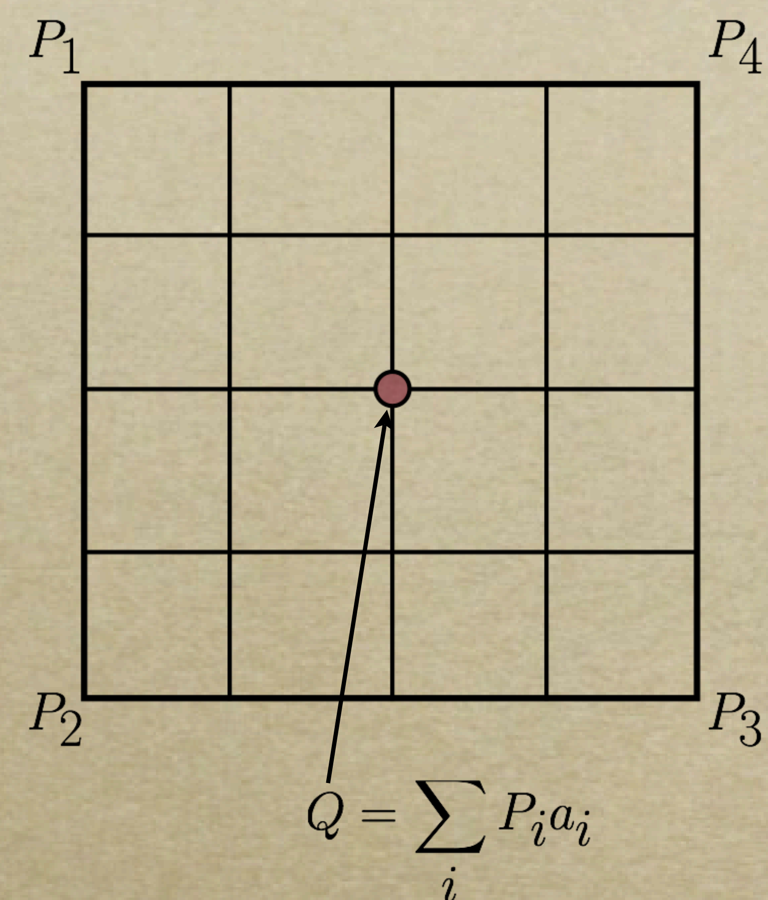
Independent of given vertex locations.

$$\sum_i P_i b_i / h_i = \left(\sum_i P_i a_i \right) / \left(\sum_j h_j a_j \right)$$

$$b_i / h_i = a_i / \left(\sum_j h_j a_j \right) \quad \forall i$$

Depth Distortion



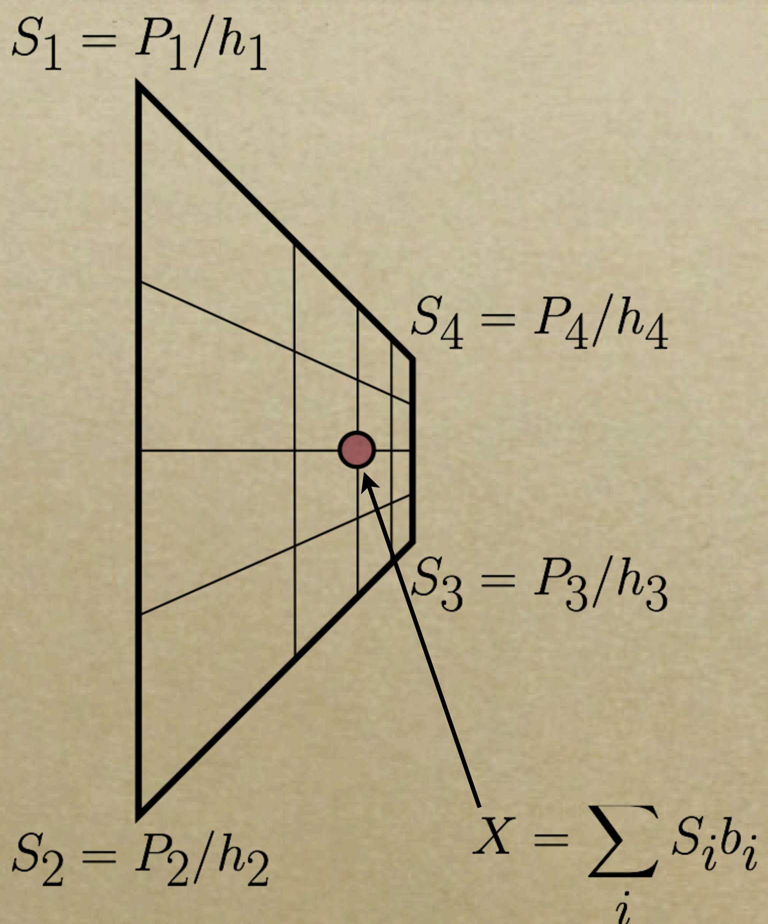


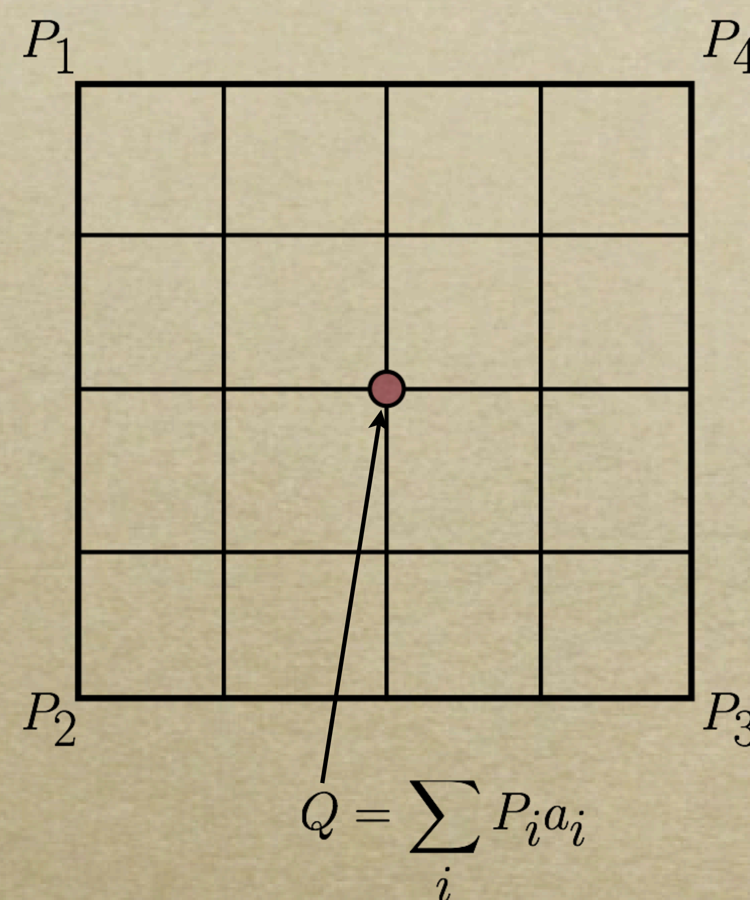
$$b_i/h_i = a_i / \left(\sum_j h_j a_j \right) \quad \forall i$$

Linear equations in the a_i .

$$\left(\sum_j h_j a_j \right) b_i/h_i - a_i = 0 \quad \forall i$$

Depth Distortion





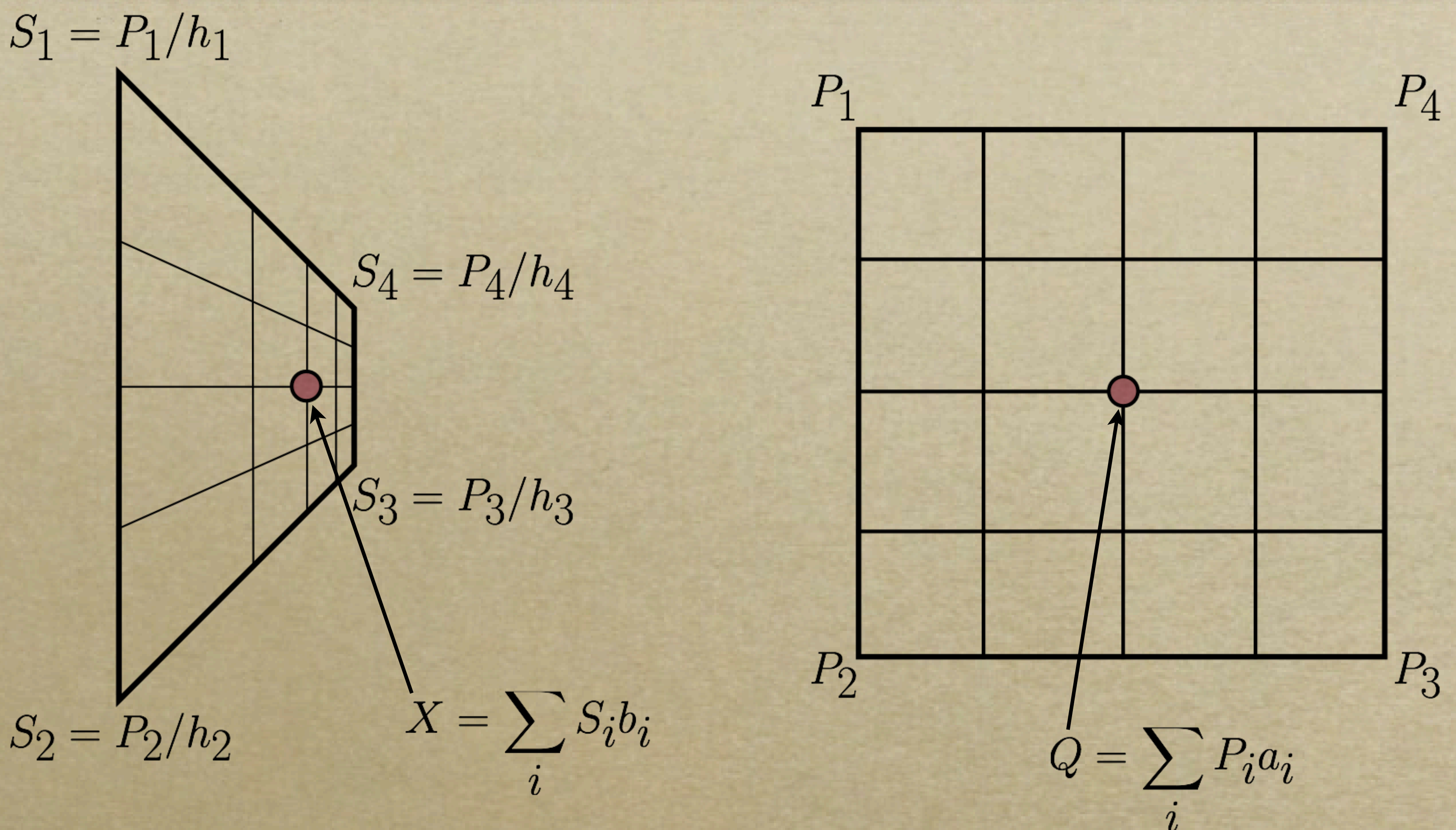
Linear equations in the a_i.

$$\left(\sum_j h_j a_j\right) b_i/h_i - a_i = 0 \quad \forall i$$

Not invertible so add some extra constraints.

$$\sum_i a_i = \sum_i b_i = 1$$

Depth Distortion



For a line: $a_1 = h_2 b_i / (b_1 h_2 + h_1 b_2)$

For a triangle: $a_1 = h_2 h_3 b_1 / (h_2 h_3 b_1 + h_1 h_3 b_2 + h_1 h_2 b_3)$

Obvious Permutations for other coefficients.