CS174A Lecture 6

Announcements & Reminders

- *Project #2 due on Sunday 10/20/19 midnight*
- *Team project info posted on Piazza*
- *Midterm: Oct 29*

Last Lecture Recap

- *Polygons (triangles)*
- *Transformations: translation, scaling, rotation, shear*

Next Up

- ***Examples of transformations***
- ***Spaces:***
  - Model space
  - Object/world space
  - Eye/camera space
  - Screen space
- ***Projections: parallel and perspective***
- ***Lighting***
- ***Flat and Smooth Shading***

SIGGRAPH trailers from 2014

Going backwards,

https://www.youtube.com/watch?v=s8IzXMWMngU

And

https://www.youtube.com/watch?v=u3Z1hDwGEmM

Affine Transformations

Examples: translations, rotations, scaling, shear

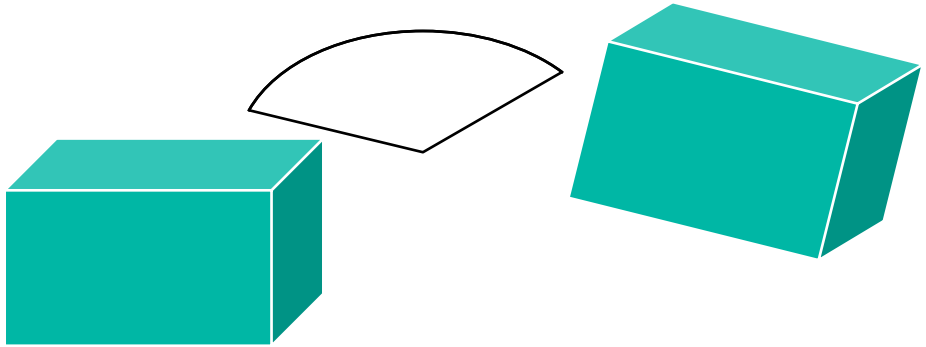
Preserves:

- Collinear points
- Planarity
- Parallelism of lines and planes
- Relative ratios of edge lengths

Rigid Body Transformations

Examples: translations and rotations

Preserves lines, angles and distances



Examples of Tranformation Composition

- ***Rotation followed by translation vs. translation followed by rotation***
  - Commutative and associative properties
- ***Rotation about a random point (not the origin)***
- ***Rotation about a random axis***
- ***Transforming a vector/normal***
  - For vertices/points transformation matrix = M
  - For normals = $(M^T)^{-1}$

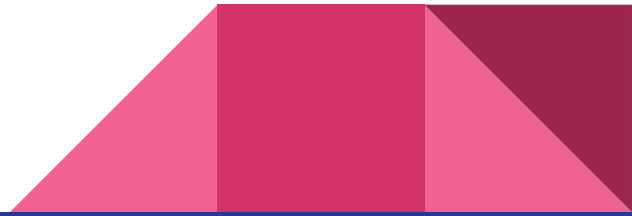
Matrix Order

- Remember the rules:
- Non-Commutativity:
  - ABCDE != BACDE != EDCBA
  - Matrix products can only be written in one left-right order. Changing the order changes the answer.
- Associativity:
  - Matrix products can be evaluated in any left-right order you want, though.
  - ABCDE = A(B(C(DE))) = (((AB)C)D)E

Matrix Multiplication is NOT commutative.

Given Matrix A and Matrix B that are non-trivial nor diagonal,

AB != BA



Matrix Multiplication is NOT commutative.

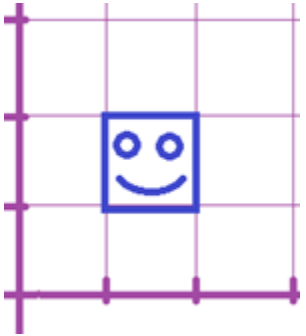
Remember our old rotation matrix:

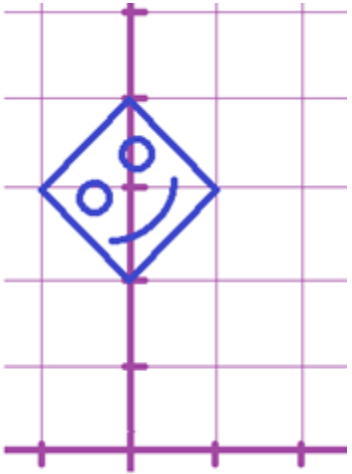
$$scale(\sqrt{2}) * rotate_z(45°) = ?$$

$$\begin{bmatrix} \sqrt{2} & \\ & \sqrt{2} \end{bmatrix} * \begin{bmatrix} \cos(45°) & -\sin(45°) \\ \sin(45°) & \cos(45°) \end{bmatrix} = ?$$

$$\Rightarrow \begin{bmatrix} \sqrt{2} & \\ & \sqrt{2} \end{bmatrix} * \begin{bmatrix} \sqrt{2}/2 & -\sqrt{2}/2 \\ \sqrt{2}/2 & \sqrt{2}/2 \end{bmatrix} = ?$$

$$\Rightarrow \begin{bmatrix} 1 & -1 \\ 1 & 1 \end{bmatrix}$$

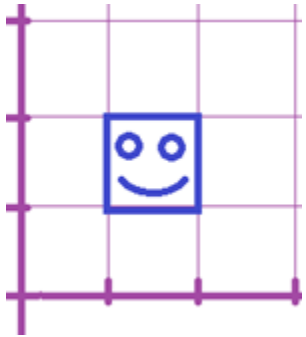




Matrix Multiplication is NOT commutative.

Suppose we modify it with a non-uniform scale matrix from the left:

$$\begin{bmatrix} 1 & \\ & 2 \end{bmatrix} * \begin{bmatrix} 1 & -1 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} & \\ ? & \end{bmatrix}$$

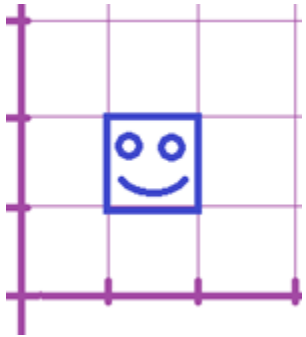


Where do the corners of the face go if we use this one?

Matrix Multiplication is NOT commutative.

Suppose we modify it with a non-uniform scale matrix from the left:

$$\begin{bmatrix} 1 & \\ & 2 \end{bmatrix} * \begin{bmatrix} 1 & -1 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & -1 \\ 2 & 2 \end{bmatrix}$$



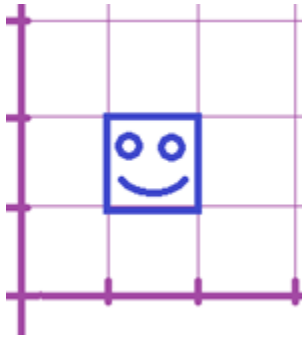
Where do the corners of the face go if we use this one?

Matrix Multiplication is NOT commutative.

Suppose we modify it with a non-uniform scale matrix from the left:

$$\begin{bmatrix} 1 & -1 \\ 2 & 2 \end{bmatrix} * \begin{bmatrix} 1 \\ 1 \end{bmatrix} = [?]$$

$$\begin{bmatrix} 1 & -1 \\ 2 & 2 \end{bmatrix} * \begin{bmatrix} 2 \\ 2 \end{bmatrix} = [?]$$



Where do the corners of the face go if we use this one?

Matrix Multiplication is NOT commutative.

Suppose we modify it with a non-uniform scale matrix from the left:

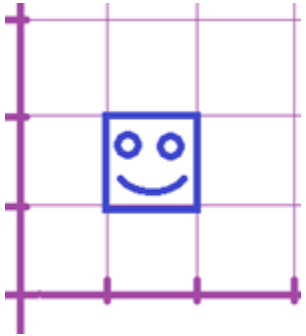
We sheared it!

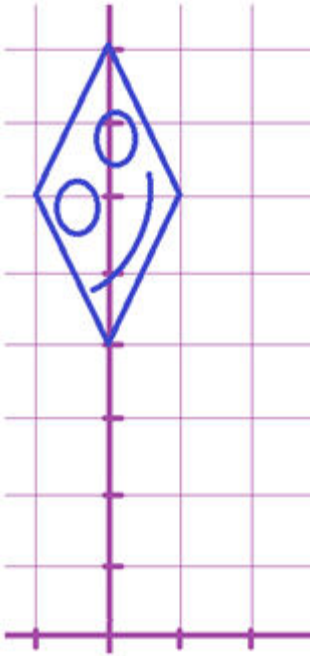
$$\begin{bmatrix} 1 & -1 \\ 2 & 2 \end{bmatrix} * \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 4 \end{bmatrix}$$

$$\begin{bmatrix} 1 & -1 \\ 2 & 2 \end{bmatrix} * \begin{bmatrix} 1 \\ 2 \end{bmatrix} = \begin{bmatrix} -1 \\ 6 \end{bmatrix}$$

$$\begin{bmatrix} 1 & -1 \\ 2 & 2 \end{bmatrix} * \begin{bmatrix} 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 6 \end{bmatrix}$$

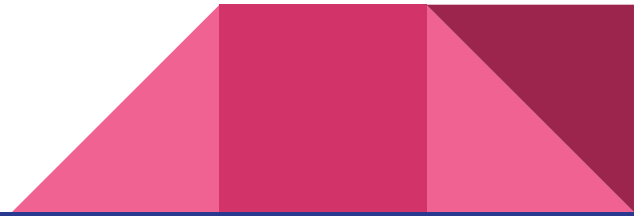
$$\begin{bmatrix} 1 & -1 \\ 2 & 2 \end{bmatrix} * \begin{bmatrix} 2 \\ 2 \end{bmatrix} = \begin{bmatrix} 0 \\ 8 \end{bmatrix}$$





Matrix Multiplication is NOT commutative.

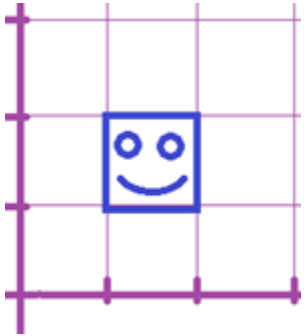
Let’s try the product the other way around now...



Matrix Multiplication is NOT commutative.

Suppose we modify it with a non-uniform scale matrix from the right:

$$\begin{bmatrix} 1 & -1 \\ 1 & 1 \end{bmatrix} * \begin{bmatrix} 1 & \\ & 2 \end{bmatrix} = \begin{bmatrix} & \\ ? & \end{bmatrix}$$



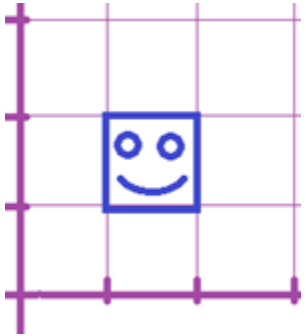
Where do the corners of the face go if we use this one?

Matrix Multiplication is NOT commutative.

Suppose we modify it with a non-uniform scale matrix from the right:

$$\begin{bmatrix} 1 & -1 \\ 1 & 1 \end{bmatrix} * \begin{bmatrix} 1 & \\ & 2 \end{bmatrix} = \begin{bmatrix} 1 & -2 \\ 1 & 2 \end{bmatrix}$$

Where do the corners of the face go if we use this one?



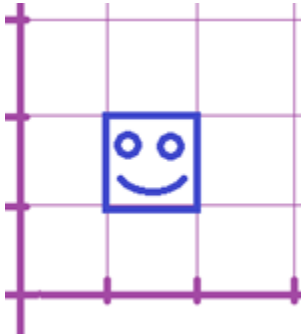
Matrix Multiplication is NOT commutative.

Suppose we modify it with a non-uniform scale matrix from the right:

$$\begin{bmatrix} 1 & -2 \\ 1 & 2 \end{bmatrix} * \begin{bmatrix} 1 \\ 1 \end{bmatrix} = [?]$$

$$\begin{bmatrix} 1 & -2 \\ 1 & 2 \end{bmatrix} * \begin{bmatrix} 2 \\ 2 \end{bmatrix} = [?]$$

Where do the corners of the face go if we use this one?



Matrix Multiplication is NOT commutative.

Suppose we modify it with a non-uniform scale matrix from the right:

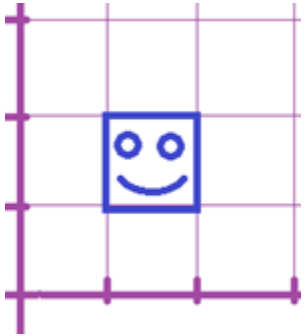
We didn't shear it!

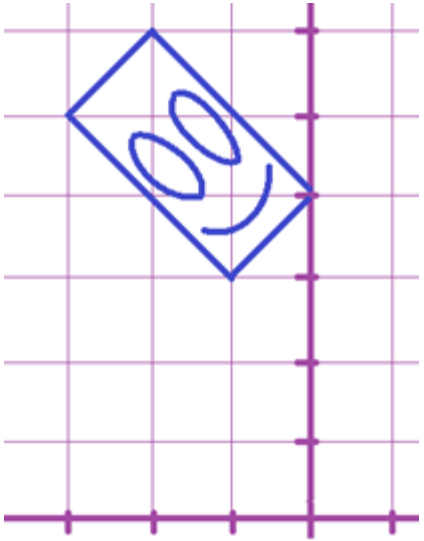
$$\begin{bmatrix} 1 & -2 \\ 1 & 2 \end{bmatrix} * \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} -1 \\ 3 \end{bmatrix}$$

$$\begin{bmatrix} 1 & -2 \\ 1 & 2 \end{bmatrix} * \begin{bmatrix} 1 \\ 2 \end{bmatrix} = \begin{bmatrix} -3 \\ 5 \end{bmatrix}$$

$$\begin{bmatrix} 1 & -2 \\ 1 & 2 \end{bmatrix} * \begin{bmatrix} 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 4 \end{bmatrix}$$

$$\begin{bmatrix} 1 & -2 \\ 1 & 2 \end{bmatrix} * \begin{bmatrix} 2 \\ 2 \end{bmatrix} = \begin{bmatrix} -2 \\ 6 \end{bmatrix}$$





Matrix Order

- Remember the rules:
- Non-Commutativity:
  - ABCDE != BACDE != EDCBA
  - Matrix products can only be written in one left-right order. Changing the order changes the answer.
- Associativity:
  - Matrix products can be evaluated in any left-right order you want, though.
  - ABCDE = A(B(C(DE))) = (((AB)C)D)E

Rotation Around an Arbitrary Axis

Euler's theorem:

Any rotation or sequence of rotations around a point is equivalent to a single rotation around an axis that passes through the point

Let's derive the transformation matrix for rotation around an arbitrary axis **u**

Rotation Around an Arbitrary Axis

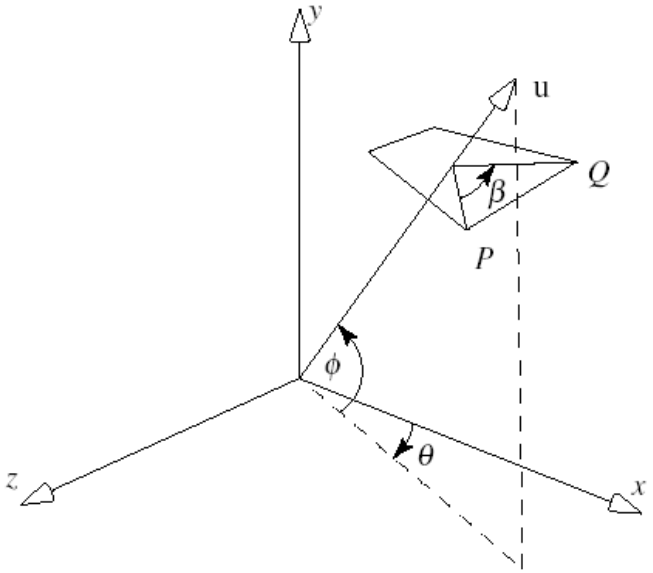
Vector (axis): $\mathbf{u} = [u_x, u_y, u_z]^T$

Rotation angle: β

Point: P

Method:

1. *Two rotations to align* **u** *with x-axis*
2. *Do x-roll by* β
3. *Undo the alignment*



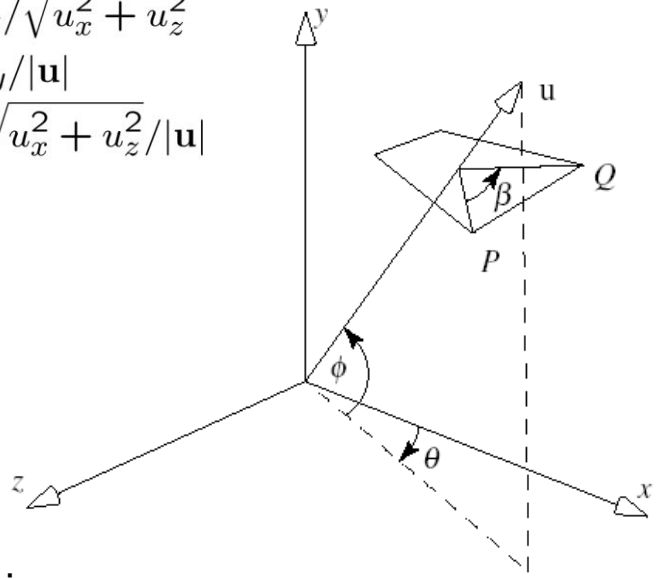
Derivation

1. $\mathbf{R}_z(-\phi)\ \mathbf{R}_y(\theta)$
2. $\mathbf{R}_x(\beta)$
3. $\mathbf{R}_y(-\theta)\ \mathbf{R}_z(\phi)$

$$\cos(\theta) = u_x/\sqrt{u_x^2 + u_z^2}$$
$$\sin(\theta) = u_z/\sqrt{u_x^2 + u_z^2}$$
$$\sin(\phi) = u_y/|\mathbf{u}|$$
$$\cos(\phi) = \sqrt{u_x^2 + u_z^2}/|\mathbf{u}|$$

All together: $\mathbf{R}_u(\beta)$ =

$\mathbf{R}_y(-\theta)\ \mathbf{R}_z(\phi)\ \mathbf{R}_x(\beta)\ \mathbf{R}_z(-\phi)\ \mathbf{R}_y(\theta)$



We should add translation too if the axis is not through the origin

Transformations of Coordinate Systems

Coordinate systems consist of basis vectors and an origin (point)

They can be represented as affine matrices

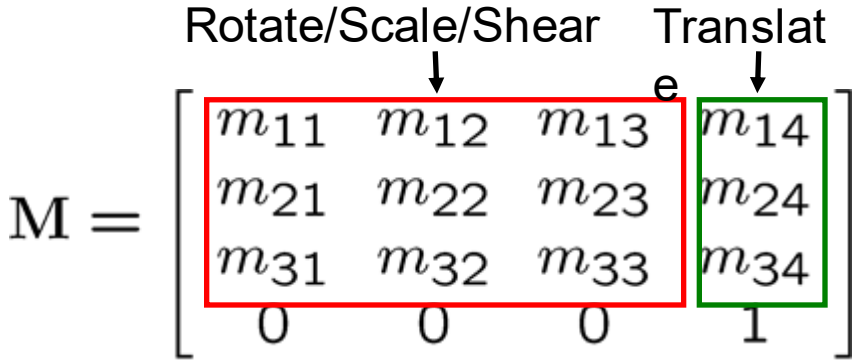
Therefore, we can transform them just like points and vectors

This provides an alternative way to think of transformations:

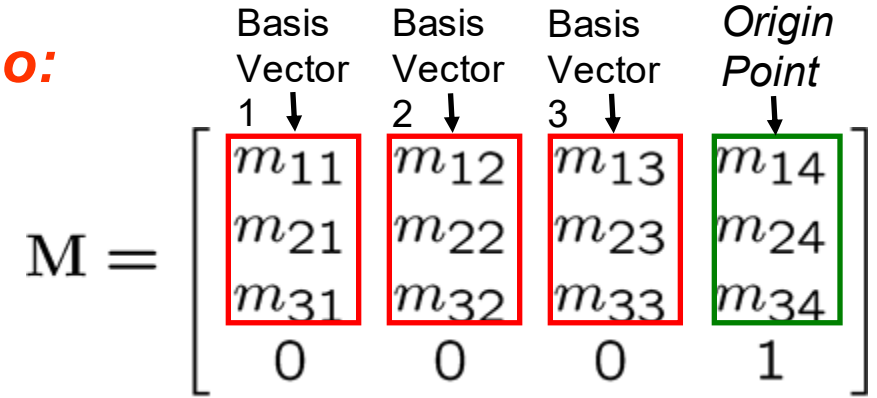
As changes of coordinate systems

Remember

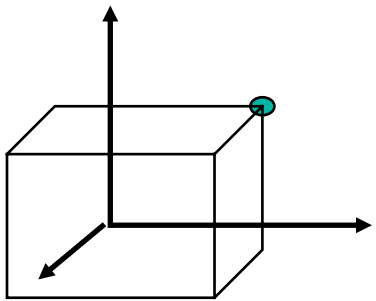
Transformations are represented by affine matrices



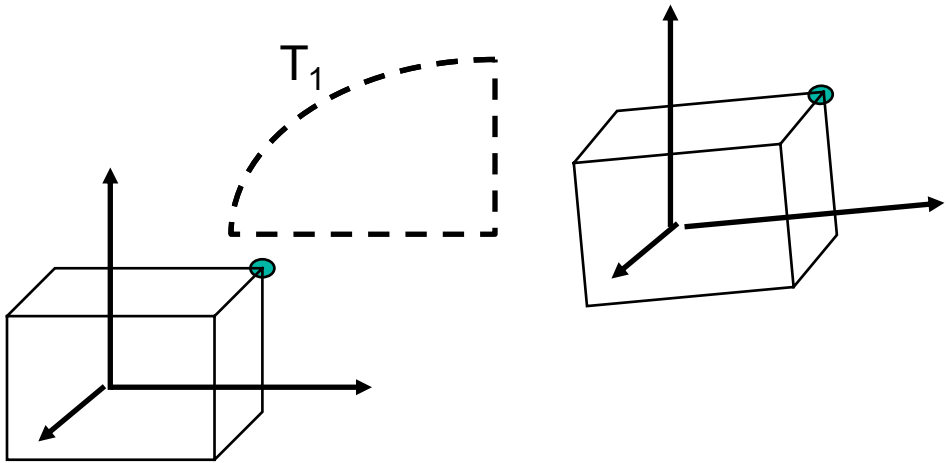
Coordinate systems too:



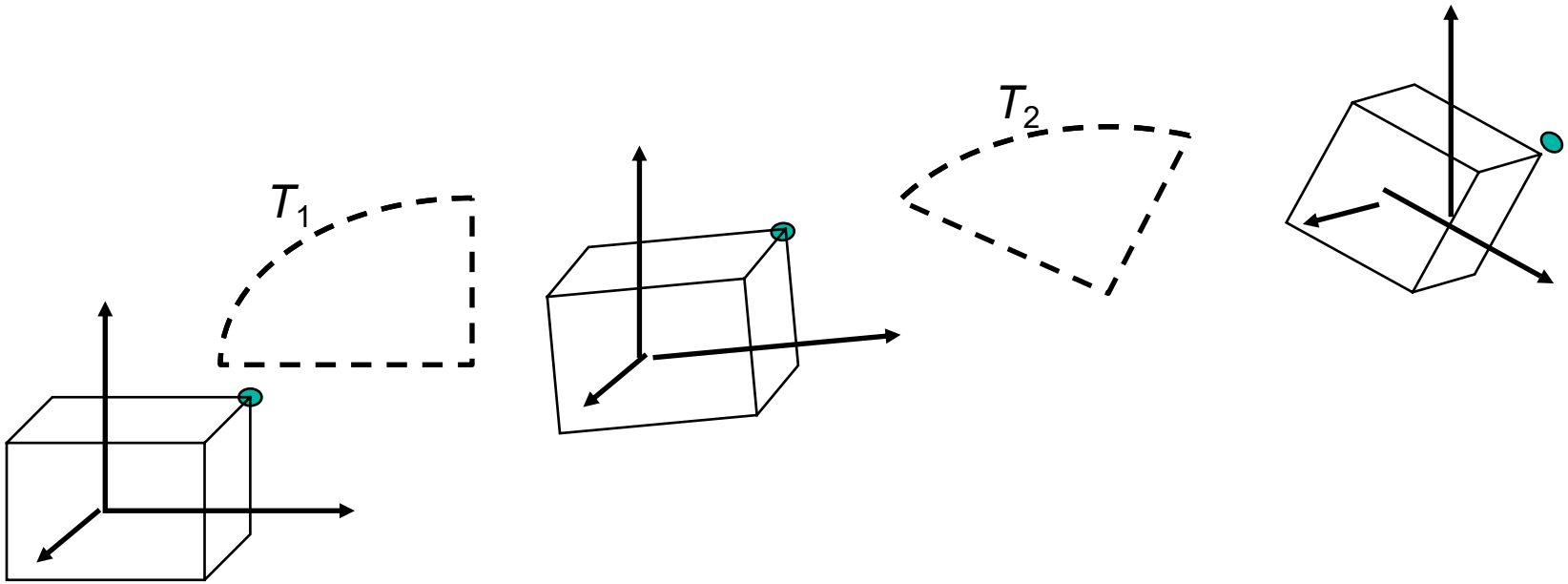
Transforming a Point by Transforming Coordinate Systems



Transforming a Point by Transforming Coordinate Systems



Transforming a Point by Transforming Coordinate Systems



Matrix Review

- All the objects you draw on screen are drawn one vertex at a time, by starting with the vertex's xyz coordinate and then multiplying by a matrix to get the final xy coordinate on the screen.

$$\underbrace{\begin{bmatrix} ? & ? & ? & ? \\ ? & ? & ? & ? \\ ? & ? & ? & ? \\ 0 & 0 & ? & ? \end{bmatrix}}_{M} * \underbrace{\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}}_{p}$$

Transforms

$$\underbrace{\begin{bmatrix} ? & ? & ? & ? \\ ? & ? & ? & ? \\ ? & ? & ? & ? \\ 0 & 0 & ? & ? \end{bmatrix}}_{M} * \underbrace{\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}}_{p}$$

- Before that matrix, the xyz coordinate is always some trivial value like (.5, .5, .5)
  - In the reference system of the shape itself
  - For example, a cube's own coordinates for its corners
- After that matrix, it's some different xy pixel coordinate denoting where that vertex will show up on the screen.
  - And z for depth, and a fourth number for translations / perspective effects
- That mapping is all that the transform does.

Transform Process

- The transform is always just one 4x4 matrix.
- But calculating what it should be involves multiplying out a big chain of intermediate special matrices. That chain is always:

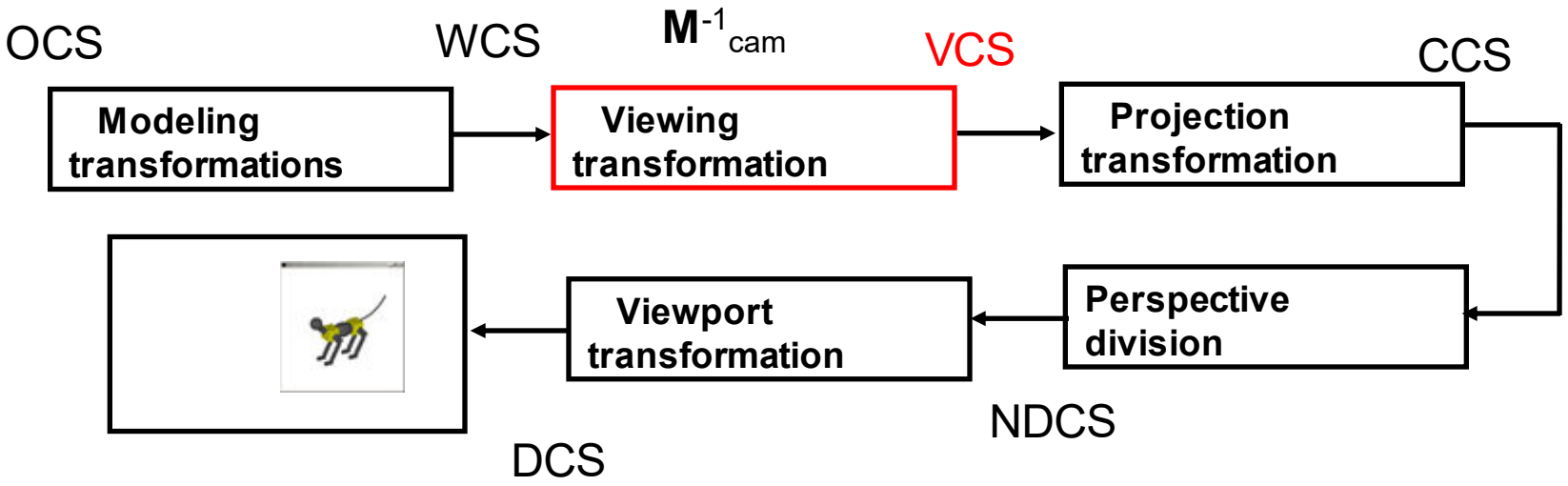
$$\underbrace{\begin{bmatrix} ? & 0 & 0 & ? \\ 0 & ? & 0 & ? \\ 0 & 0 & ? & ? \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{Viewport} * \underbrace{\begin{bmatrix} ? & 0 & ? & ? \\ 0 & ? & ? & ? \\ 0 & 0 & ? & ? \\ 0 & 0 & ? & ? \end{bmatrix}}_{Projection} * \underbrace{\begin{bmatrix} ? & ? & ? & ? \\ ? & ? & ? & ? \\ ? & ? & ? & ? \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{Camera} * \underbrace{\begin{bmatrix} ? & ? & ? & ? \\ ? & ? & ? & ? \\ ? & ? & ? & ? \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{Model} * \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Transform Process

$$\underbrace{\begin{bmatrix} ? & 0 & 0 & ? \\ 0 & ? & 0 & ? \\ 0 & 0 & ? & ? \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{Viewport} * \underbrace{\begin{bmatrix} ? & 0 & ? & ? \\ 0 & ? & ? & ? \\ 0 & 0 & ? & ? \\ 0 & 0 & ? & ? \end{bmatrix}}_{Projection} * \underbrace{\begin{bmatrix} ? & ? & ? & ? \\ ? & ? & ? & ? \\ ? & ? & ? & ? \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{Camera} * \underbrace{\begin{bmatrix} ? & ? & ? & ? \\ ? & ? & ? & ? \\ ? & ? & ? & ? \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{Model} * \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

- Note: We never actually see the viewport matrix.
  - The viewport matrix is automatically applied for you at the end of the vertex shader.
  - Early during initialization, javascript set it up, calling gl.viewport(x,y,width,height).
- All the other special matrices you do manage.

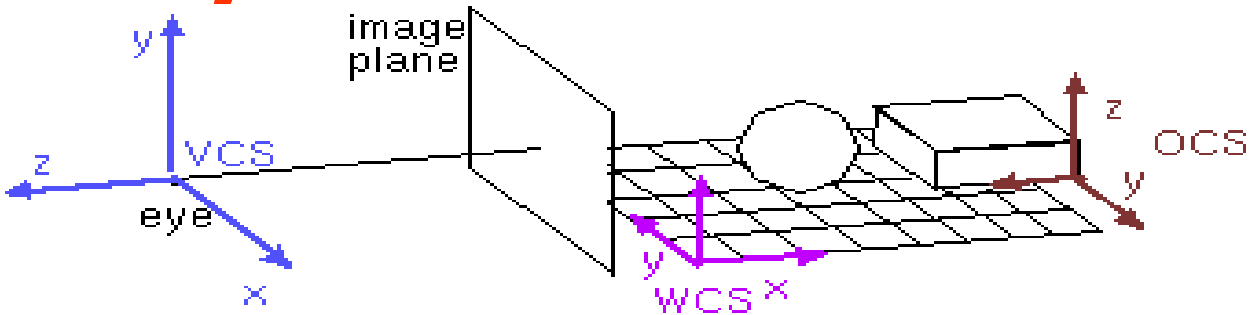
Graphics Pipeline



Rendering a 3D Scene From the Point of View of a Virtual Camera

Camera Transformation

Transforms objects to camera coordinates



$$\left.\begin{array}{l} P_{\mathrm{wcs}} = \mathbf{M}_{\mathrm{cam}} P_{\mathrm{vcs}} \rightarrow P_{\mathrm{vcs}} = \mathbf{M}_{\mathrm{cam}}^{-1} P_{\mathrm{wcs}} \\ P_{\mathrm{wcs}} = \mathbf{M}_{\mathrm{mod}} P_{\mathrm{ocs}} \end{array}\right\} \rightarrow$$

$$P_{\mathrm{vcs}} = \underbrace{\mathbf{M}_{\mathrm{cam}}^{-1} \mathbf{M}_{\mathrm{mod}}} P_{\mathrm{ocs}}$$

Modelview Transformation

Transform Process

$$\underbrace{\begin{bmatrix} ? & 0 & 0 & ? \\ 0 & ? & 0 & ? \\ 0 & 0 & ? & ? \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{Viewport} * \underbrace{\begin{bmatrix} ? & 0 & ? & ? \\ 0 & ? & ? & ? \\ 0 & 0 & ? & ? \\ 0 & 0 & ? & ? \end{bmatrix}}_{Projection} * \underbrace{\begin{bmatrix} ? & ? & ? & ? \\ ? & ? & ? & ? \\ ? & ? & ? & ? \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{Camera} * \underbrace{\begin{bmatrix} ? & ? & ? & ? \\ ? & ? & ? & ? \\ ? & ? & ? & ? \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{Model} * \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

- The camera matrix is very much like the model transform matrix for placing shapes. But:
  - The shape being placed is the scene's observer
  - You actually use the **inverse matrix** of what you would have done to a 3D model of an actual camera