CS174A Lecture 6

Announcements & Reminders

- *10/16/22: A2 due; will be discussed during this week's TA session*
- *10/26/22 and 10/27/22: Office hours, ???, Zoom*
- *10/27/22: Midterm Exam: 6:00 – 7:30 PM PST, in person, in class*
- *11/08/22: Team project proposals due, initial version*
- *11/09/22: A3 due*
- *11/10/22: Midway demo, online zoom*
- *Updated syllabus on Canvas*

Last Lecture Recap

- ***Polygons (triangles)***
- ***Transformations: translation, scaling, rotation, shear***
  - Geometrical representation
  - Mathematical representation
  - Homogeneous representation
- ***Inverse of Transformations***

Next Up

- *Concatenation of transformations*
- *Spaces: Model, Object/World, Eye/Camera, Screen*
- *Projections: parallel and perspective*
- *Midterm*
- *Lighting*
- *Flat and Smooth Shading*

SIGGRAPH trailers from 2014

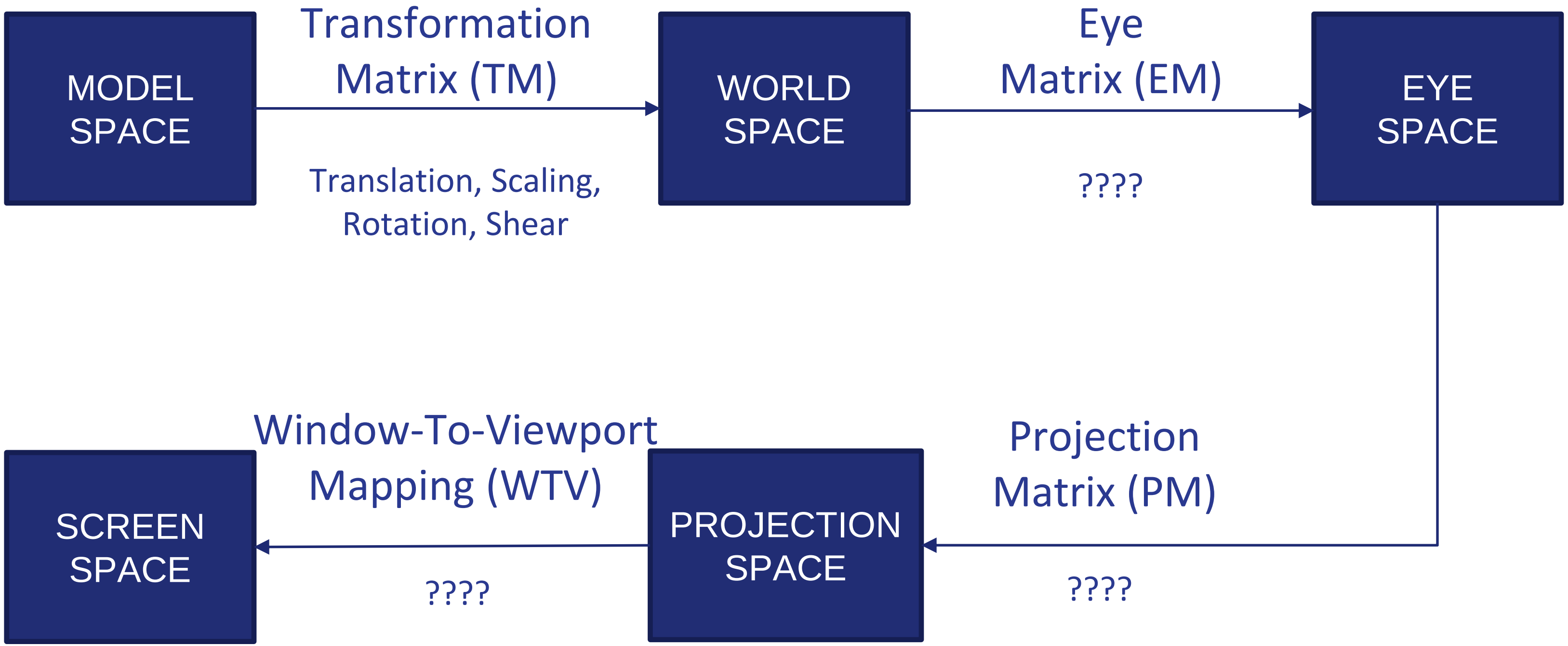
Going backwards,

https://www.youtube.com/watch?v=s8lzXMWMngU

And

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

Rendering Pipeline



Linear Combination of Vectors

Definition

A linear combination of the m vectors $\mathbf{v}_1, \ldots, \mathbf{v}_m$ is a vector of the form:

$$\mathbf{w} = a_1\mathbf{v}_1 + \ldots + a_m\mathbf{v}_m, \qquad a_1, \ldots, a_m \text{ in } R$$

Special Cases

Linear combination

$\mathbf{w} = a_1\mathbf{v}_1+\ldots a_m\mathbf{v}_m$, $\qquad a_1, \ldots, a_m$ in R

Affine combination:

A linear combination for which $a_1+\ldots+a_m=1$

Convex combination

An affine combination for which $a_i \geq 0$ for $i=1,\ldots,m$

Linear Combination of Points

Points P, Q scalars a, b:

$$P' = a*P + b*Q$$
$$= a\,[p_1, p_2, p_3, 1]^T + b[q_1, q_2, q_3, 1]^T$$
$$= [ap_1+bq_1,\; ap_2+bq_2,\; ap_3+bq_3,\; a+b]^T$$

Affine combination ⇒ a + b = 1

$$P' = (1\text{-}a)*P + a*Q$$

Convex combination ⇒ a + b = 1 AND a, b ≥ 0

Linear Operations

1. ***Makes sense for linear as well as affine:***
   a. Addition of 2 vectors
   b. Multiplication of vector by a scalar
   c. Addition of a vector and a point

2. ***Doesn't make sense for linear, but ok for affine:***
   a. Addition of 2 points
   b. Multiplication of a point by a scalar

Affine Transformations/Interpolations

Examples: translations, rotations, scaling, shear

Preserves:

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

Linear Vs. Affine Transformations

- A linear transformation only takes linear combinations of x, y, and z
  - Point at origin stays on the origin forever
- An *affine* transformation is more general and more powerful
  - It’s called an affine transformation because it **preserves** affine combinations (line segment interpolations don’t get messed up before vs. after)

Summary: Affine is Useful

- Affine transformations are the main modeling tool in graphics
    - They are applied as matrix multiplications
    - Any affine transformation can be performed as a series of <u>elementary</u> affine transformations
    - We can now do object placement
        - Model entire scenes

Affine Combinations of Points

$$W = a_1 P_1 + a_2 P_2$$
$$T(W) = T(a_1 P_1 + a_2 P_2) = a_1 T(P_1) + a_2 T(P_2)$$

Proof: from linearity of matrix multiplication

$$\mathbf{M}W = \mathbf{M}(a_1 P_1 + a_2 P_2) = a_1 \mathbf{M} P_1 + a_2 \mathbf{M} P_2$$

Preservations of Lines and Planes

$$L(t) = (1-t)P_1 + tP_2$$
$$T(L) = (1-t)T(P_1) + tT(P_2) = (1-t)\mathrm{M}P_1 + t\mathrm{M}P_2$$

$$Pl(s,t) = (1-s-t)P_1 + tP_2 + sP_3$$
$$\begin{aligned} T(Pl) &= (1-s-t)T(P_1) + tT(P_2) + sT(P_3) \\ &= (1-s-t)\mathrm{M}P_1 + t\mathrm{M}P_2 + s\mathrm{M}P_3 \end{aligned}$$

Preservation of Parallelism

$$L(t) = P + t\mathbf{u}$$

$$\mathrm{M}L = \mathrm{M}(P + t\mathbf{u}) = \mathrm{M}P + \mathrm{M}(t\mathbf{u}) \rightarrow$$
$$\mathrm{M}L = \mathrm{M}P + t(\mathrm{M}\mathbf{u})$$

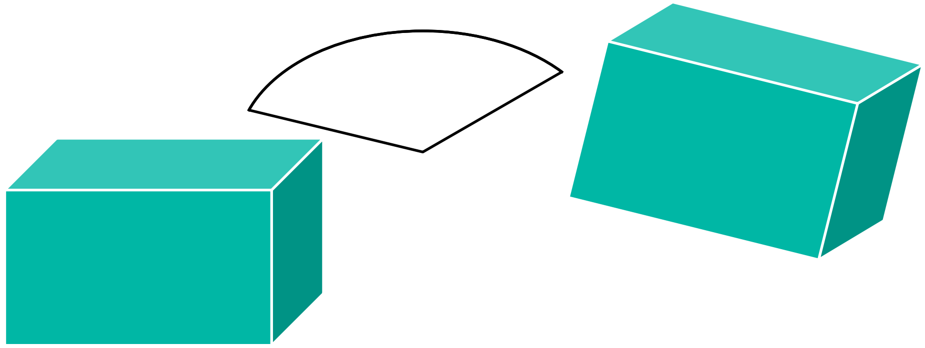
$\mathbf{Mu}$ independent of P

Similarly for planes

Rigid Body Transformations

Examples: translations and rotations

Preserves lines, angles and distances



Affine Transforms Review

Affine Transformations in 3D

General form

$$\begin{bmatrix} Q_x \\ Q_y \\ Q_z \\ 1 \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} P_x \\ P_y \\ P_z \\ 1 \end{bmatrix}$$

Or:

$$Q = \mathrm{M}P$$

General Form

Rotation / Scaling / Shearing

Translation

$$\mathbf{M} = \begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

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 vectors (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.

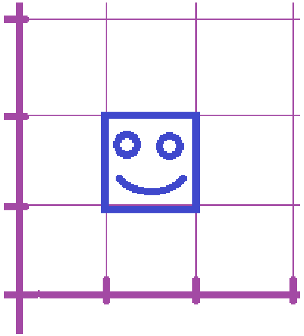
Remember our old rotation matrix:

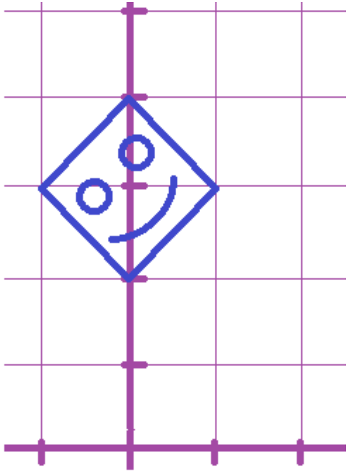
$$scale(\sqrt{2}) * rotate_z(45^\circ) = ?$$

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

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

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

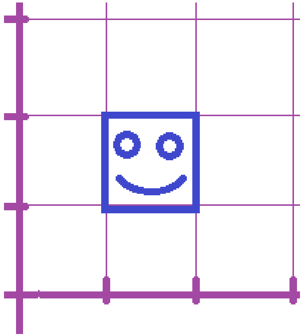




Matrix Multiplication is NOT commutative.

Suppose we modify it with a non-uniform scale matrix from the <u>left</u>:

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

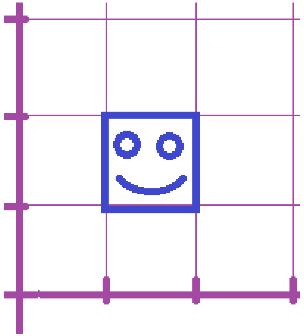


Where do the corners of the face go if we use this one?

Matrix Multiplication is NOT commutative.

Suppose we modify it with a non-uniform scale matrix from the <u>left</u>:

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



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

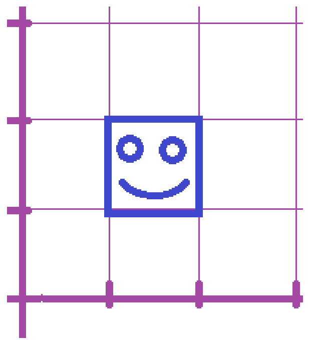
Matrix Multiplication is NOT commutative.

Suppose we modify it with a non-uniform scale matrix from the <u>left</u>:

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

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

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 <u>left</u>:

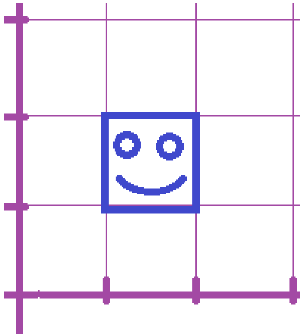
We sheared it!

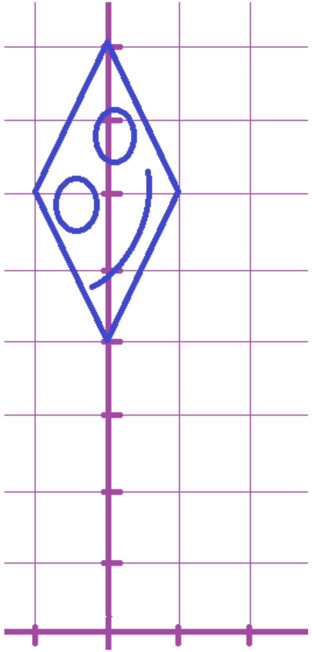
$$\begin{bmatrix} 1 & -1 \\ 2 & 2 \end{bmatrix} * \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 4 \end{bmatrix}$$

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

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

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





Matrix Multiplication is NOT commutative.

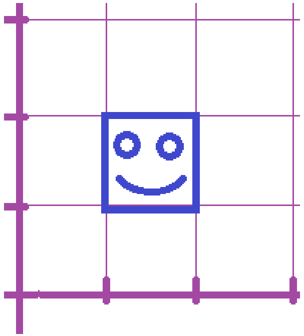
Let’s try the product the other way around now...

Matrix Multiplication is NOT commutative.

Suppose we modify it with a non-uniform scale matrix from the <u>right</u>:

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

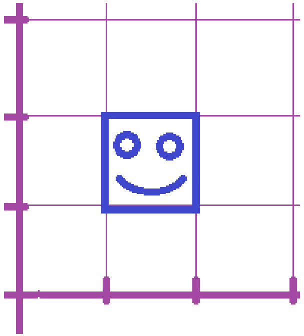
Where do the corners of the face go if we use this one?



Matrix Multiplication is NOT commutative.

Suppose we modify it with a non-uniform scale matrix from the <u>right</u>:

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



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

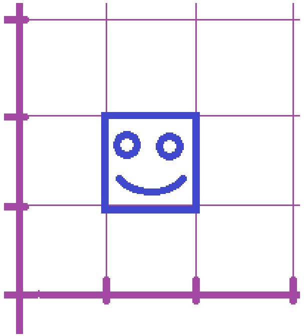
Matrix Multiplication is NOT commutative.

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 <u>right</u>:

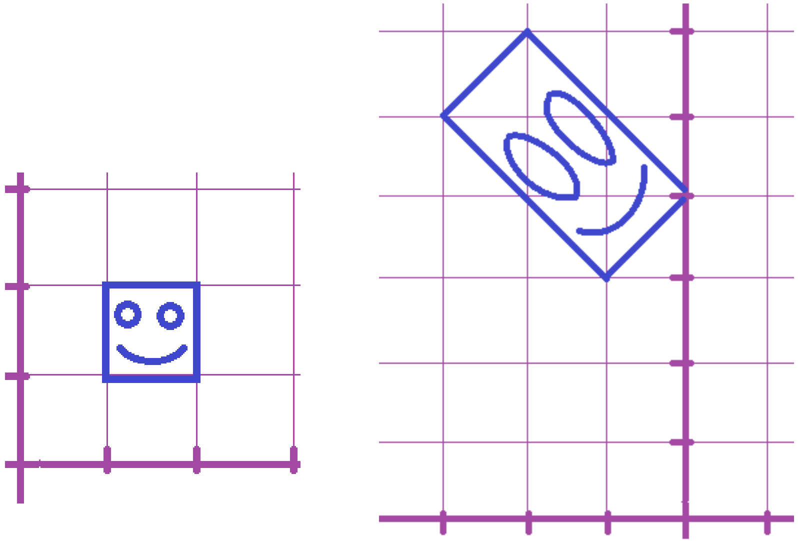
We didn't shear it!

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

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

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

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



Matrix Order

- Remember the rules:
- Non-Commutativity:
  - ABCDE != BACDE != EDCBA
  - Matrix products can only be <u>written</u> in one left-right order. Changing the order changes the answer.
- Associativity:
  - Matrix products can be <u>evaluated</u> 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

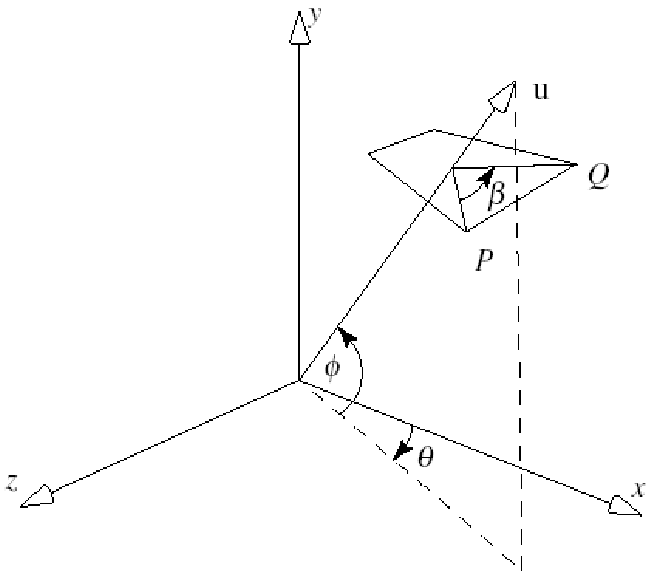
Let's derive the transformation matrix for rotation around an arbitrary axis **u**

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*

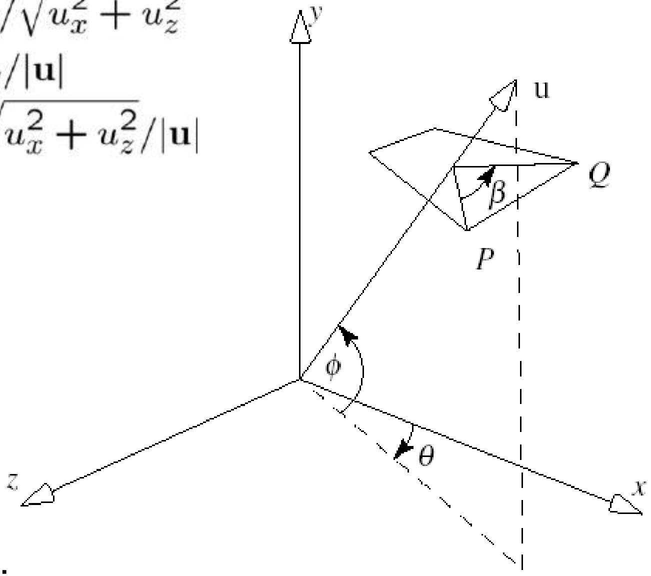
Derivation

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: $\mathbf{R}_{\mathbf{u}}(\beta)$ =

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



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

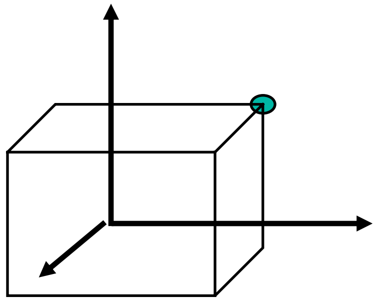
Transformations are represented by affine matrices

$$\begin{array}{c} \begin{array}{cc} \text{Rotate/Scale/Shear} & \text{Translate} \end{array} \\ \mathbf{M} = \begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{array}$$

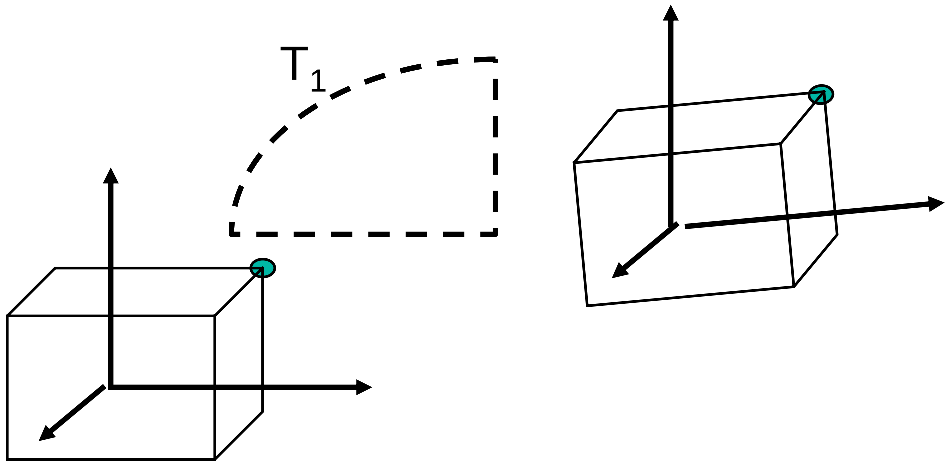
Coordinate systems too:

$$\begin{array}{c} \begin{array}{cccc} \text{Basis Vector 1} & \text{Basis Vector 2} & \text{Basis Vector 3} & \textit{Origin Point} \end{array} \\ \mathbf{M} = \begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{array}$$

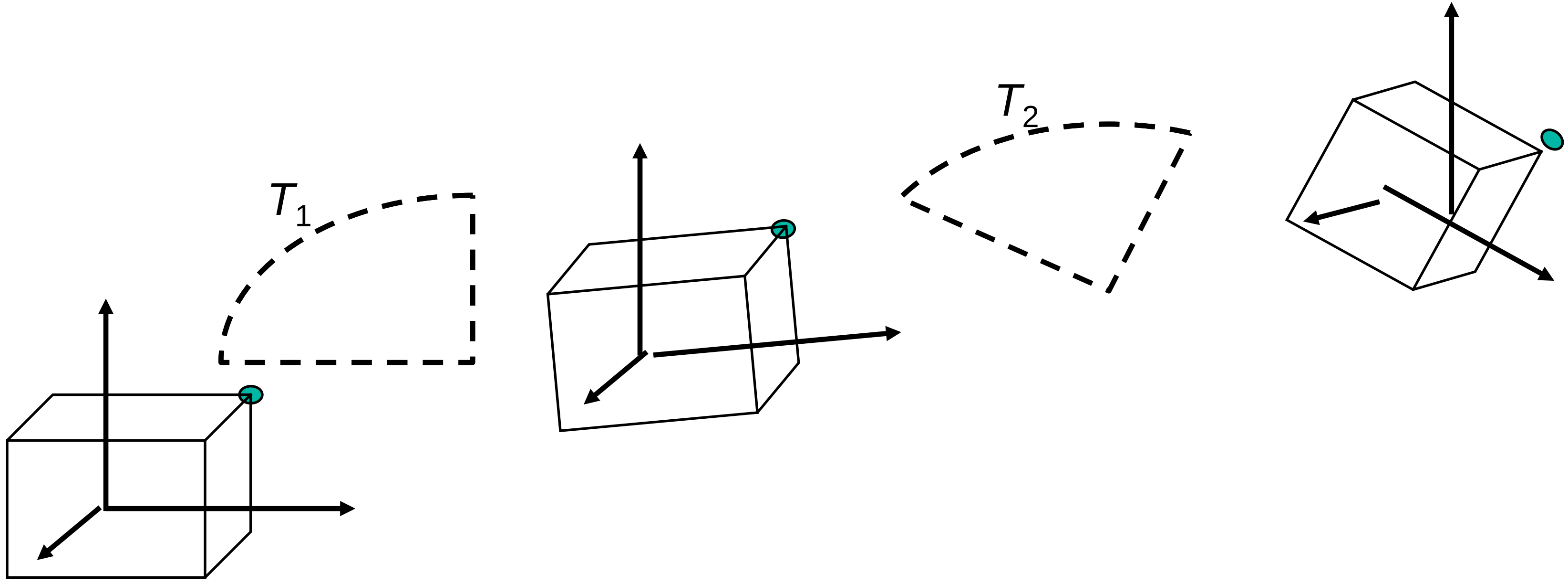
Transforming a Point by Transforming Coordinate Systems



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:

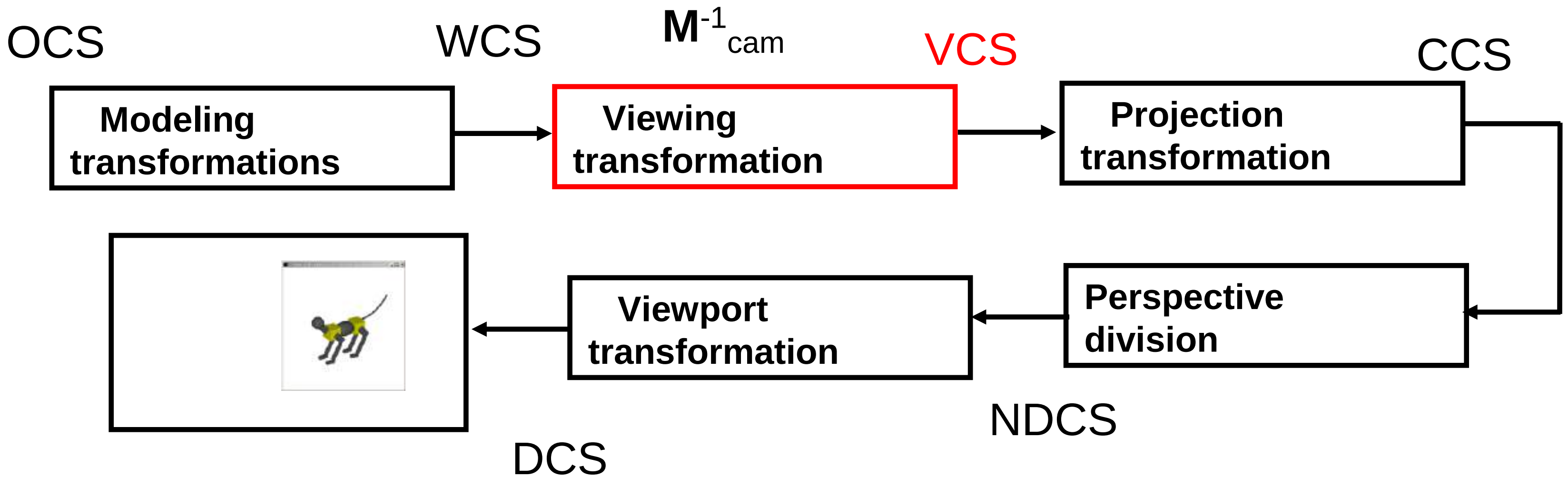
$$\underset{Viewport}{\begin{bmatrix} ? & 0 & 0 & ? \\ 0 & ? & 0 & ? \\ 0 & 0 & ? & ? \\ 0 & 0 & 0 & 1 \end{bmatrix}} * \underset{Projection}{\begin{bmatrix} ? & 0 & ? & ? \\ 0 & ? & ? & ? \\ 0 & 0 & ? & ? \\ 0 & 0 & ? & ? \end{bmatrix}} * \underset{Camera}{\begin{bmatrix} ? & ? & ? & ? \\ ? & ? & ?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

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.

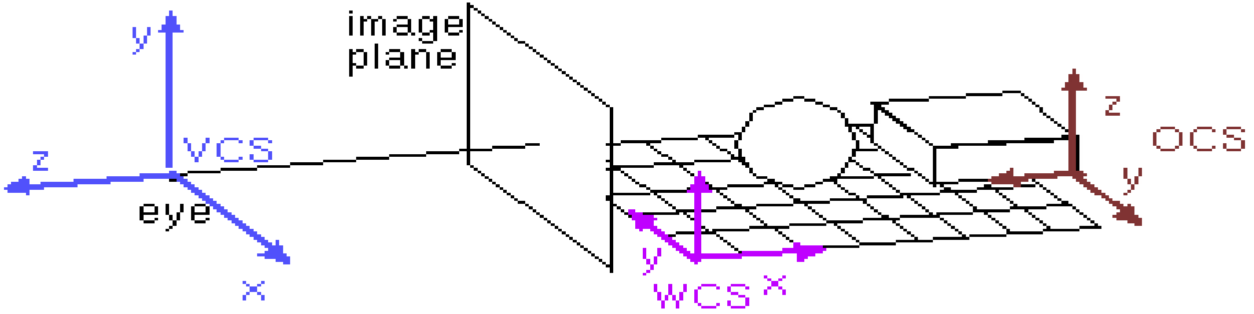
Graphics Pipeline



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

Camera Transformation

Transforms objects to camera coordinates



$$\left.\begin{array}{l} P_{\text{WCS}} = \mathrm{M}_{\text{cam}} P_{\text{VCS}} \rightarrow P_{\text{VCS}} = \mathrm{M}_{\text{cam}}^{-1} P_{\text{WCS}} \\ P_{\text{WCS}} = \mathrm{M}_{\text{mod}} P_{\text{OCS}} \end{array}\right\} \rightarrow$$

$$P_{\text{VCS}} = \underbrace{\mathrm{M}_{\text{cam}}^{-1} \mathrm{M}_{\text{mod}} P_{\text{OCS}}}_{\text{Modelview Transformation}}$$

Transform Process

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

- 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