# Lecture 7

# SIGGRAPH trailers from 2012

https://www.youtube.com/watch?v=cKrng7ztpog

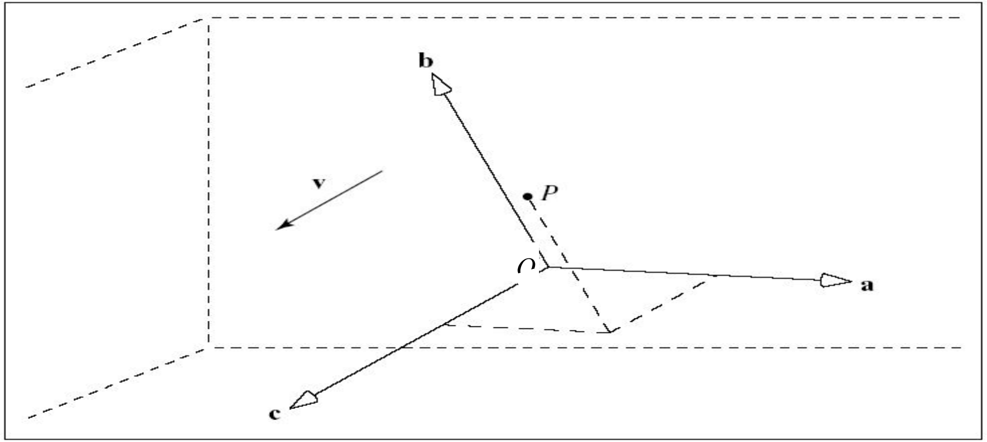
# Announcements

- Assignment 3 is released!
- Due Date

# Can we view Transformations as Coordinate Systems?

# Reminder: Coordinate Systems

Coordinate system:
*O*, **a**, **b**, **c**,



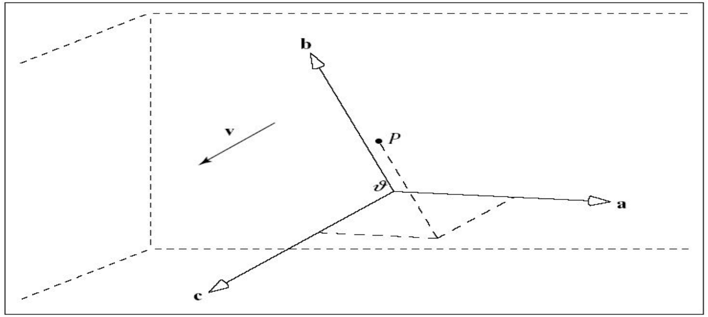

$$\mathbf{v} = [v_1 \;\; v_2 \;\; v_3]^T \rightarrow \mathbf{v} = v_1\mathbf{a} + v_2\mathbf{b} + v_3\mathbf{c}$$

$$P = [p_1 \;\; p_2 \;\; p_3]^T \rightarrow P - O = p_1\mathbf{a} + p_2\mathbf{b} + p_3\mathbf{c}$$
$$P = O + p_1\mathbf{a} + p_2\mathbf{b} + p_3\mathbf{c}$$

# Reminder: Coordinate Systems

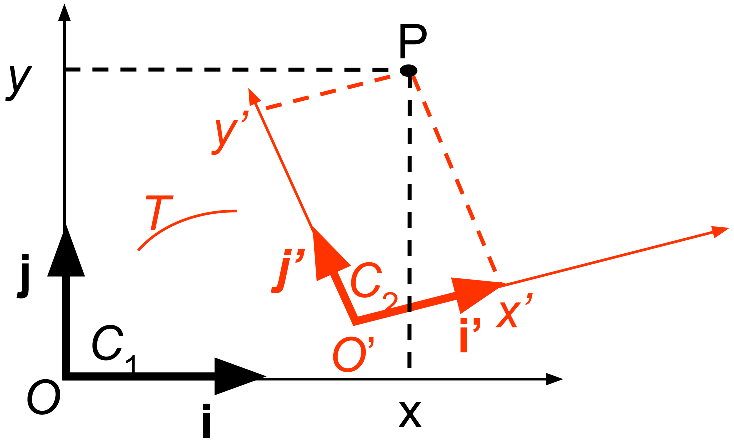
$$\mathbf{v} = v_1\mathbf{a} + v_2\mathbf{b} + v_3\mathbf{c} \rightarrow \mathbf{v} = [\mathbf{a}\ \mathbf{b}\ \mathbf{c}\ O]\begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ 0 \end{bmatrix}$$

$$P = O + p_1\mathbf{a} + p_2\mathbf{b} + p_3\mathbf{c} \rightarrow P = [\mathbf{a}\ \mathbf{b}\ \mathbf{c}\ O]\begin{bmatrix} p_1 \\ p_2 \\ p_3 \\ 1 \end{bmatrix}$$



# Transforming $C_1$ into $C_2$

***What is the relationship between P in $C_2$ and P in $C_1$ if $T(C_1) \mapsto C_2$?***




$$C_1 : P = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

$$C_2 : P = \begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix}$$

$$O' = T(O),$$
$$\mathbf{i}' = T(\mathbf{i}),$$
$$\mathbf{j}' = T(\mathbf{j}),$$
$$\mathbf{k}' = T(\mathbf{k})$$

# Derivation

By definition $P$ is the linear combination of vectors $\mathbf{i}', \mathbf{j}', \mathbf{k}'$ and point $O'$.

$$P = x'\mathbf{i}' + y'\mathbf{j}' + z'\mathbf{k}' + O'$$

In coordinate system $C_1$:

$$P_{C_1} = x'\mathbf{i}'_{C_1} + y'\mathbf{j}'_{C_1} + z'\mathbf{k}'_{C_1} + O'_{C_1}$$

# Derivation

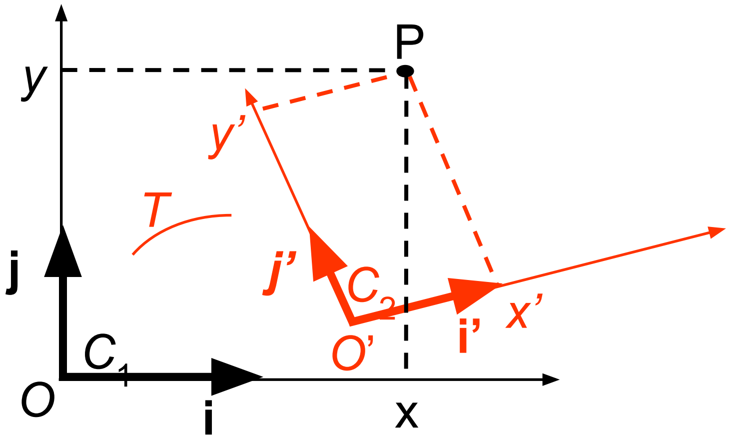
$$P_{C_1} = x'\mathbf{i}'_{C_1} + y'\mathbf{j}'_{C_1} + z'\mathbf{k}'_{C_1} + O'_{C_1}$$

We know that $[\mathbf{i}'_{C_1}, \mathbf{j}'_{C_1}, \mathbf{k}'_{C_1}, O'_{C_1}] = T([\mathbf{i}, \mathbf{j}, \mathbf{k}, O])$

$$\begin{aligned}
P_{C_1} &= x'T(\mathbf{i}) + y'T(\mathbf{j}) + z'T(\mathbf{k}) + T(O) \\
&= x'\mathbf{M}\mathbf{i} + y'\mathbf{M}\mathbf{j} + z'\mathbf{M}\mathbf{k} + \mathbf{M}O \\
&= x'\mathbf{M}\begin{bmatrix}1\\0\\0\\0\end{bmatrix} + y'\mathbf{M}\begin{bmatrix}0\\1\\0\\0\end{bmatrix} + z'\mathbf{M}\begin{bmatrix}0\\0\\1\\0\end{bmatrix} + \mathbf{M}\begin{bmatrix}0\\0\\0\\1\end{bmatrix} \\
&= \mathbf{M}\begin{bmatrix}x'\\0\\0\\0\end{bmatrix} + \mathbf{M}\begin{bmatrix}0\\y'\\0\\0\end{bmatrix} + \mathbf{M}\begin{bmatrix}0\\0\\z'\\0\end{bmatrix} + \mathbf{M}\begin{bmatrix}0\\0\\0\\1\end{bmatrix} \\
&= \mathbf{M}\left(\begin{bmatrix}x'\\0\\0\\0\end{bmatrix} + \begin{bmatrix}0\\y'\\0\\0\end{bmatrix} + \begin{bmatrix}0\\0\\z'\\0\end{bmatrix} + \begin{bmatrix}0\\0\\0\\1\end{bmatrix}\right) = \mathbf{M}\begin{bmatrix}x'\\y'\\z'\\1\end{bmatrix}
\end{aligned}$$

# $P$ in $C_1$ vs $P$ in $C_2$

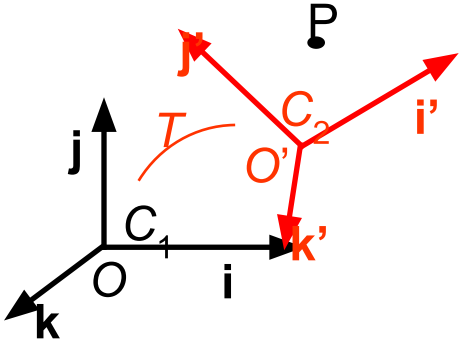
$$\underset{T}{C_1 \mapsto C_2}$$




$$P_{C_1} = \mathrm{M} P_{C_2}$$

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \mathrm{M} \begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix}$$

# Transformations as a Change of Basis




**So, we know that**

$$P_{C_1} = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \mathrm{M} \begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \mathrm{M} P_{C_2}$$

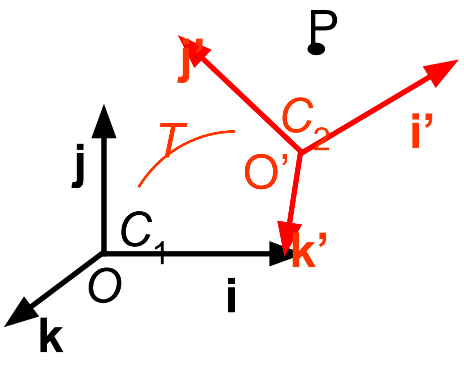
**Now, what is $\mathrm{M}$ with respect to the basis vectors?**

$$P_{C_2} = x'\mathbf{i}'_{C_2} + y'\mathbf{j}'_{C_2} + z'\mathbf{k}'_{C_2} + O'_{C_2} = x' \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} + y' \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} + z' \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

$$P_{C_1} = x'\mathbf{i}'_{C_1} + y'\mathbf{j}'_{C_1} + z'\mathbf{k}'_{C_1} + O'_{C_1} = x' \begin{bmatrix} i'_x \\ i'_y \\ i'_z \\ 0 \end{bmatrix} + y' \begin{bmatrix} j'_x \\ j'_y \\ j'_z \\ 0 \end{bmatrix} + z' \begin{bmatrix} k'_x \\ k'_y \\ k'_z \\ 0 \end{bmatrix} + \begin{bmatrix} O'_x \\ O'_y \\ O'_z \\ 1 \end{bmatrix}$$

$$P_{C_1} = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} i'_x & j'_x & k'_x & O'_x \\ i'_y & j'_y & k'_y & O'_y \\ i'_z & j'_z & k'_z & O'_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \mathrm{M} P_{C_2}$$

# Transformations as a Change of Basis




$$P_{C_1} = \mathrm{M} P_{C_2}$$

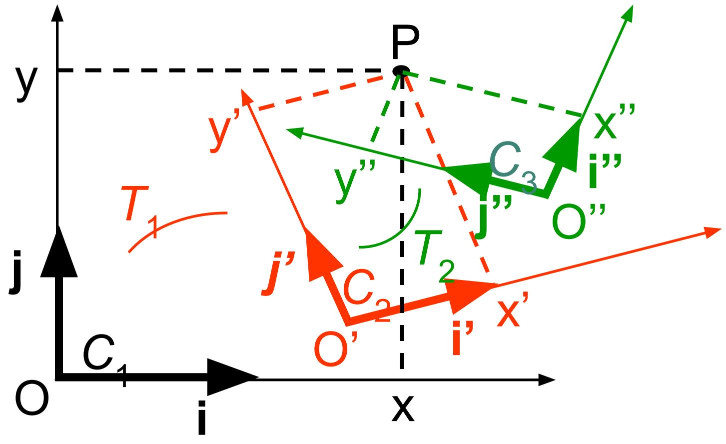
$$P_{C_1} = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} i'_x & j'_x & k'_x & O'_x \\ i'_y & j'_y & k'_y & O'_y \\ i'_z & j'_z & k'_z & O'_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \mathrm{M} P_{C_2}$$

**That is:**

We can view transformations as a change of coordinate system

# Successive Transformations of the Coordinate System

$$C_1 \xmapsto{T_1} C_2 \xmapsto{T_2} C_3$$




Working backwards:

$$P_{C_2} = \mathrm{M}_2 P_{C_3} \rightarrow \begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \mathrm{M}_2 \begin{bmatrix} x'' \\ y'' \\ z'' \\ 1 \end{bmatrix}$$

$$P_{C_1} = \mathrm{M}_1 P_{C_2} \rightarrow \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \mathrm{M}_1 \begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \mathrm{M}_1 \mathrm{M}_2 \begin{bmatrix} x'' \\ y'' \\ z'' \\ 1 \end{bmatrix}$$

# GuerrillaCG Series:  Hierarchies

https://vimeo.com/2159127

# Matrix Order: Two Mindsets

“Points” mindset and “Bases” mindset

# Matrix Order

- The trickiest concept in the class - we'll look at it as many times as possible until it's clear.  Might as well start seeing it now.

# 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

# Matrix Multiplication is NOT commutative.

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

AB != BA

# Matrix Multiplication is NOT commutative.

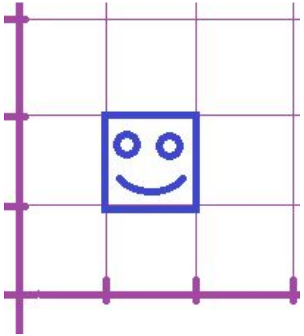
Remember our old rotation matrix:

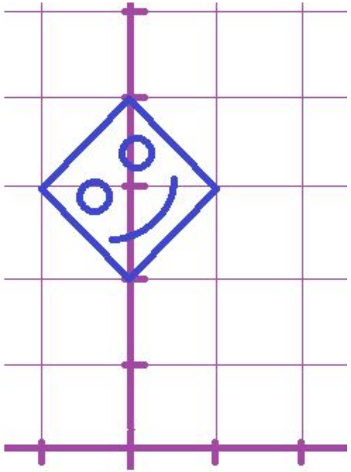
$$scale(\sqrt{2}) * rotate_z(45^\circ) = ?$$

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

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

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

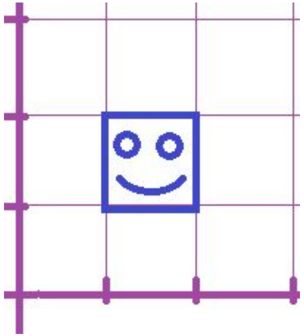




# Matrix Multiplication is NOT commutative.

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

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

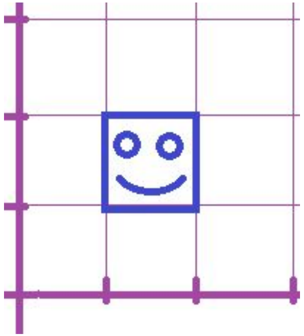


Where do the corners of the face go if we use this one?

# Matrix Multiplication is NOT commutative.

Suppose we modify it with a non-uniform scale matrix from the <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?

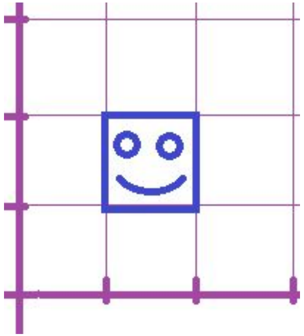
# Matrix Multiplication is NOT commutative.

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

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

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

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



# Matrix Multiplication is NOT commutative.

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

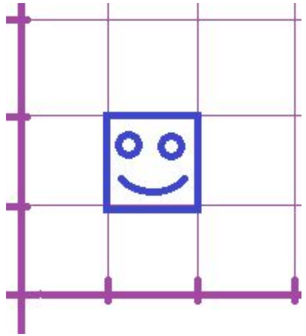
$$\begin{bmatrix} 1 & -1 \\ 2 & 2 \end{bmatrix} * \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 4 \end{bmatrix}$$

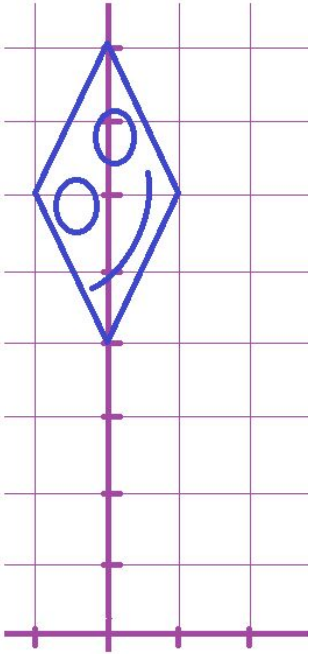
$$\begin{bmatrix} 1 & -1 \\ 2 & 2 \end{bmatrix} * \begin{bmatrix} 1 \\ 2 \end{bmatrix} = \begin{bmatrix} -1 \\ 6 \end{bmatrix}$$

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

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

We sheared it!





# Matrix Multiplication is NOT commutative.

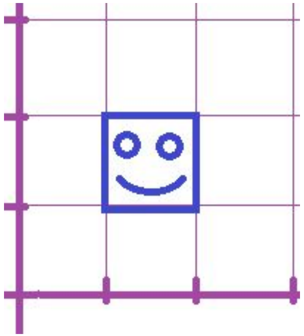
Let's try the product the other way around now...

# 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?

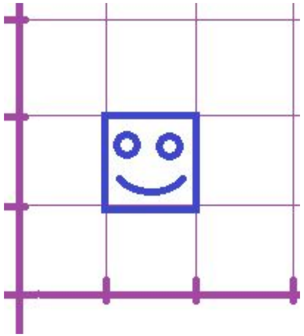


# Matrix Multiplication is NOT commutative.

Suppose we modify it with a non-uniform scale matrix from the <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?



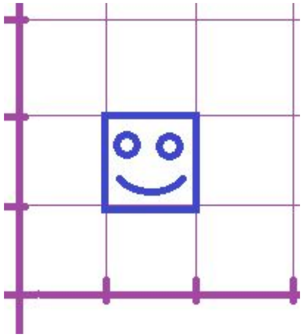
# Matrix Multiplication is NOT commutative.

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 right:

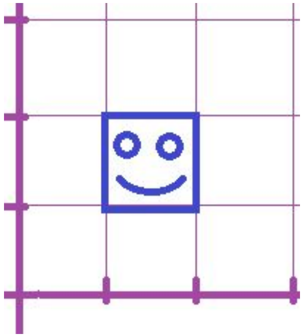
We didn't shear it!

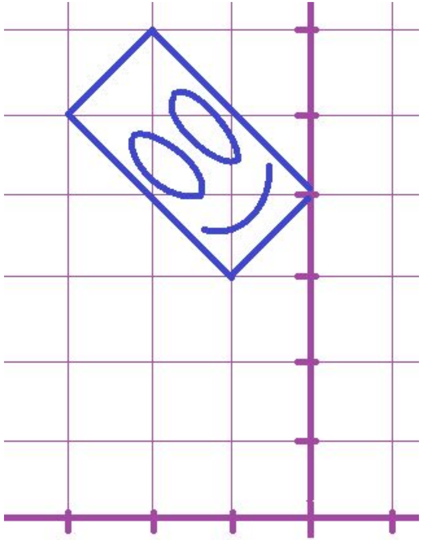
$$\begin{bmatrix} 1 & -2 \\ 1 & 2 \end{bmatrix} * \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} -1 \\ 3 \end{bmatrix}$$

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

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

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





# Matrix Order

- Remember the rules:
- Non-Commutativity:
  - ABCDE != BACDE != EDCBA
  - Matrix products can only be <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

# Storing Matrix Products

- Typical choice in a computer:  Maintain an accumulator variable and store your total product so far in there.
- But what type should our accumulator variable be?
    - Each point of the shape?
        - No, that's too much iteration for every new matrix term
    - Basis vector set?
        - Yes.  Just maintain in a single matrix ("current" / "model transform")

# Matrix Order

- That still leaves a choice of which direction to accumulate new terms from.
  - Pre-multiply new terms from the left?
  - Post-multiply new terms from the right?
- Either way is mathematically equivalent (evaluation order of a product is up to you). We could even start accumulating from the middle.

# Two possible mindsets:

1. Starting from the right side of your product and moving leftwards:
   - This is like accumulating changes to the final picture - that is, warping the whole shape around by all its points until it's finally in place. "Points perspective"
   - Not often useful.
2. Starting from the left side of your product and moving rightwards:
   - This is like accumulating changes to your XYZ basis vectors (your local reference frame), visit places in your scene with it. "Bases perspective"
   - More often useful.

# Matrix order (example)

$$\underbrace{\begin{bmatrix}1 & 2\\ 3 & 1\end{bmatrix}}_{A}\underbrace{\begin{bmatrix}1 & 2\\ 2 & 1\end{bmatrix}}_{B}\underbrace{\begin{bmatrix}2 & 1\\ 1 & 1\end{bmatrix}}_{C}\underbrace{\begin{bmatrix}x\\ y\end{bmatrix}}_{\bar{x}}$$

$$\underbrace{\begin{bmatrix}5 & 4\\ 5 & 7\end{bmatrix}}_{(AB)}\underbrace{\begin{bmatrix}2 & 1\\ 1 & 1\end{bmatrix}}_{C}\underbrace{\begin{bmatrix}x\\ y\end{bmatrix}}_{\bar{x}}$$

$$\underbrace{\begin{bmatrix}14 & 9\\ 17 & 12\end{bmatrix}}_{(ABC)}\underbrace{\begin{bmatrix}x\\ y\end{bmatrix}}_{\bar{x}}$$

etc

vs

$$\underbrace{\begin{bmatrix}1 & 2\\ 3 & 1\end{bmatrix}}_{A}\underbrace{\begin{bmatrix}1 & 2\\ 2 & 1\end{bmatrix}}_{B}\underbrace{\begin{bmatrix}2 & 1\\ 1 & 1\end{bmatrix}}_{C}\underbrace{\begin{bmatrix}x\\ y\end{bmatrix}}_{\bar{x}}$$

$$\underbrace{\begin{bmatrix}1 & 2\\ 3 & 1\end{bmatrix}}_{A}\underbrace{\begin{bmatrix}1 & 2\\ 2 & 1\end{bmatrix}}_{B}\underbrace{\begin{bmatrix}2x+y\\ x+y\end{bmatrix}}_{(C\bar{x})}$$

$$\underbrace{\begin{bmatrix}1 & 2\\ 3 & 1\end{bmatrix}}_{A}\underbrace{\begin{bmatrix}4x+3y\\ 5x+3y\end{bmatrix}}_{(BC\bar{x})}$$

Option 1: Starting from left, post-multiply each matrix in turn before finally applying to point (moves the universe’s bases around for drawing a stationary point set)

Option 2: Starting on right, multiply each matrix onto the point in turn (moves points around a stationary universe)

# Matrix Order

- Bases perspective: Left to right
- **Why do we usually choose to post-multiply (grow the product rightward)?**
  - We make hierarchical shapes!
  - On the way to the leaf nodes of our shapes, parent nodes' matrices happen to equal subsets of our leaf node's total product:

`Grandparent:` M*N * (each point)

`Parent:` M*N*O*P * (each point)

`Child` = $\boxed{\boxed{M*N}*O*P}$*Q * (each point)

# Matrix Order

`Child` = $\boxed{\boxed{M*N}*O*P}*Q$ * (each point)

- These subsets are always the first N terms inside our product, starting from the left
- We want our intermediate products on the way to the final answer to conveniently equal our parent node matrices so we can draw the parents.
    - Therefore, we start from the left side and multiply rightwards.

# Matrix Order

- This isn't a strict rule; we can still pre-multiply sometimes.
  - Maybe we're not building a hierarchical shape right now
  - Maybe we're traversing the hierarchy backwards (starting at a leaf node) and willing to do the duplicate work that entails
    - Can't save parent matrices on the way to child nodes this way:

`Child` = M*N*O*P*Q * (each point)

  - Maybe we're checking our answer from post-multiplying

# Matrix Order

- This isn't a strict rule; we can still pre-multiply sometimes.
  - Maybe we're checking our answer from post-multiplying
    - If you're taking an exam, try both ways!
    - It's important to be able to understand both points and bases mindsets so they can corroborate each other.

# Summary

- Two common approaches. Multiply starting from:
  - Right to left (pre-multiply all new terms onto the product) or,
  - Left to right (post-multiply)
- The choice determines what your intermediate products are (points vs matrices?), and what each intermediate step intuitively means
  - An updated image

    vs.

  - An updated basis to draw it in

# Rule of Thumb

## *Transforming a point $P$:*

Transformations: $T_1$, $T_2$, $T_3$

Matrix: $\mathbf{M} = \boxed{\mathbf{M}_3 \boxed{\mathbf{M}_2\ \mathbf{M}_1}}$

Point is transformed by $\mathbf{M}P$

Each transformation happens with respect to the **same** coordinate system

## *Transforming a coordinate system:*

Transformations: $T_1$, $T_2$, $T_3$ (not generally the same as the ones above)

Matrix: $\mathbf{M} = \boxed{\boxed{\mathbf{M}_1\ \mathbf{M}_2}\ \mathbf{M}_3}$

A point has coordinates $\mathbf{M}P$ in the original coordinate system

Each transformation happens with respect to **previous** coordinate system

# Matrix Game

http://bases-game.glitch.me

# Practice Example

# Matrix Order in Practice

- We normally use post-multiplication, which means:
  - Reading from top to bottom in your code: “Bases” thinking
  - Reading from bottom to top in your code: “Points” thinking
- To go backwards in your history:
  - Apply the opposite of your transforms <u>in opposite order</u> (remember the rule of matrix inverse of products)
  - Or assign your matrix to a saved value
    - A backup matrix variable (watch out for aliasing, use copy())
    - A JavaScript “stack” of copies of previous values

# Remember

- Some of the best test questions require reasoning about long transform sequences on 2D drawings or graphs.
- The *order* of transformation is by far the hardest concept to get consistently right throughout the projects.

# Remember

- To think in points picture *starting* by drawing your shape
  - Then stretch it into place with global transforms
- To think in axes picture *finishing* by drawing your shape
  - After applying transformations to move your axis / origin

# Matrix Order Example

- Suppose we wanted to swing at a distance of 10 around some point (x, y, z).
- We'll show <u>two pieces of code</u> that do that.
- The difference between them will be:
  - One pre-multiplies new terms to the chain,
  - and the other post-multiplies.
  - <u>Both will produce the same product order!</u>

# Matrix Order Example

Pre-multiplying is “Thinking in points” :

Building a shape first (in this case an orbit shape) and then moving the whole shape’s points to the arbitrary xyz point

```
                                    // Send the object to the orbit’s edge:
model_transform = Mat4.translation([ 0,0,10 ]);
                                    // Rotate everything within the orbit based on time:
model_transform.pre_multiply( Mat4.rotation( t, [ 0,1,0 ] );
                                    // Send the orbit to an arbitrary xyz point:
model_transform.pre_multiply( Mat4.translation([ x,y,z ]) );
                                    // Draw the shape there:
this.shapes.cube.draw( ... );
```

# Matrix Order Example

Post-multiplying is "Thinking in axes": Opposite ordering of code lines

Bringing your origin over to the arbitrary xyz point, rotating there, then move this new origin out 10 units away from the pivot point:

```
                                   // Move coordinate system to the arbitrary pivot point
model_transform = Mat4.translation([ x,y,z ]);
                                   // Rotate our coordinate system over time
model_transform.post_multiply( Mat4.rotation( t, [ 0,1,0 ] );
                                   // Travel out along our newly rotated system
model_transform.post_multiply( Mat4.translation([ 0,0,10 ]) );
                                   // Draw the shape there:
this.shapes.cube.draw( ... );
```

# Tip: Rotations become shears

- Non-uniform scales anywhere in the matrix chain turn all rotations to the right of them into shears.
- Since shears are rarely desired, non-uniform scales are typically put at the very right end of a matrix chain (to the immediate left of the point).
  - Then we undo that most recent part of the transform before drawing the next shapes.

# Tip: Backtracking your operations

Non-commutativity is the reason that we have to unwrap our matrices in **reverse order** whenever we backtrack through our scene.  Any other order would have a different effect and wouldn't go back to the prior state.