Computer Graphics

Discussion 2

Garett Ridge garett@cs.ucla.edu,
Sam Amin samamin@ucla.edu,
Theresa Tong theresa.r.tong@gmail.com,
Quanjie Geng szmun.genggj@gmail.com

Today’s topic: The Matrix Chain

Sending triangles to their final places

Part I: Combining Transformations

Transformations and Syntax Suggestions

Moving Objects

Your shader program applies the final matrix product to each point in your shape.

- The matrices used are 4x4; 3 dimensions for x, y, and z and the 4th to allow translations

Translation

$$\begin{bmatrix} 1 & 0 & 0 & X \\ 0 & 1 & 0 & Y \\ 0 & 0 & 1 & Z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Scale

$$\begin{bmatrix} SX & 0 & 0 & 0 \\ 0 & SY & 0 & 0 \\ 0 & 0 & SZ & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotate about Z

$$\begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Moving Objects

- Applying transformations in different orders often produces different results
  - ex: non-uniform scale followed by rotate produces shear instead of rotated object

Example transformation: scale(2), then translate(1, 2, 3)

$$M_{\text{translate}} \cdot M_{\text{scale}} = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & 3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 2 & 0 & 0 & 1 \\ 0 & 2 & 0 & 2 \\ 0 & 0 & 2 & 3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} 2x+1 \\ 2y+2 \\ 2z+3 \\ 1 \end{pmatrix}$$

Order of Transformations

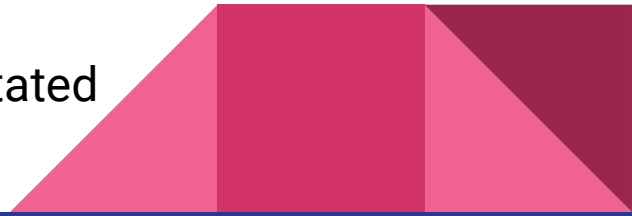
An example of transformation order:

```
[...]rotation(90, 0, 1, 0));
this.cube.draw(...);
[...]translation(10, 0, 0));
this.cube.draw(...);
[...]scale(1, 1, 3));
this.cube.draw(...);
```

Your three core transformations are:

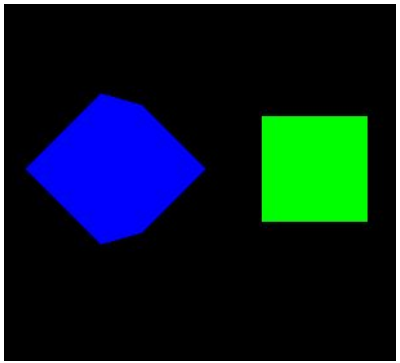
```
rotation(angle, x, y, z)
translation(x, y, z)
scale(x, y, z);
```

- The first cube will be rotated
- The second cube will be translated, then rotated
- The last cube will be scaled, then translated, then rotated



Order of Transformations

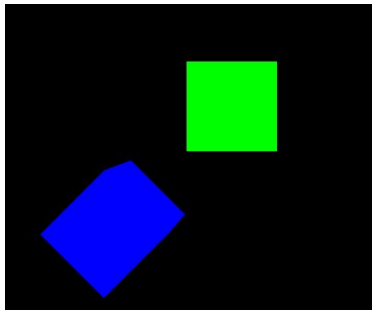
Another example-- a green cube followed by a blue cube:



```
this.cube.draw(..., greenMat);
```

```
[...]translation(-2, 0, 0));
[...]rotation(45, 0, 0, 1));
this.cube.draw(..., blueMat);
```

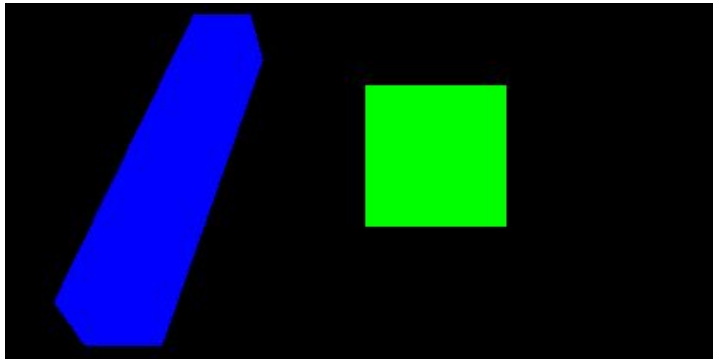
```
[...]rotation(45, 0, 0, 1));
[...]translation(-2, 0, 0));
this.cube.draw(..., blueMat);
```



Order of Transformations

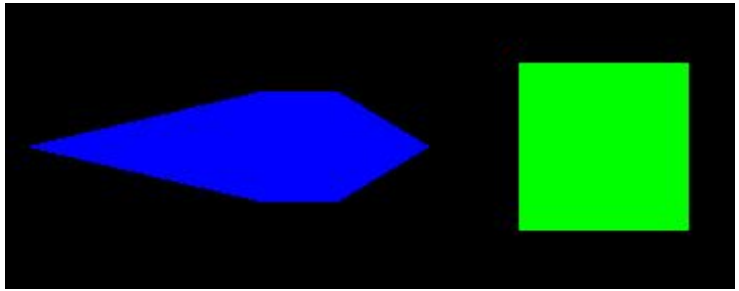
Building on the previous slide:

```
[...]rotation(45, 1, 0, 0));
[...]scale(0.5, 0.5, 3));
this.cube.draw(..., blueMat);
```



Shear vs. Not Shear

```
[...]scale(0.5, 0.5, 3));
[...]rotation(45 , 1, 0, 0));
this.cube.draw(..., blueMat);
```



Transformations

Transformations are matrix operations

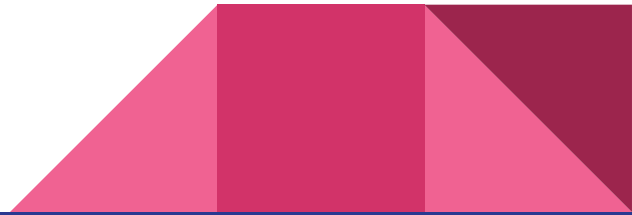
- Reading from top to bottom in your code: coordinate systems thinking
- Reading from bottom to top in your code: points thinking

Instead of undoing and redoing transformations, best to use a shortcut such as a stack to save states for you.

Transformations

model_transform = mult(model_transform, translation(0,0,-2));

- You use mult to apply a transformation to your current model transformation matrix
- Your options for transformations are translation, scale, and rotation. Rotation takes an extra parameter:
  - rotation(degree, x, y, z)
- After setting up your model_transform matrix how you like, make a call to draw



Part II: Mat Multiplication is NOT COMMUTATIVE

Ok but really

Matrix Multiplication Review (Sorry)

Use dot products for all 4 cells:

$$\begin{bmatrix}1 & 2\\3 & 4\end{bmatrix} * \begin{bmatrix}4 & 3\\2 & 1\end{bmatrix} = ?$$

$$* \begin{bmatrix}4 & 3\\2 & 1\end{bmatrix}$$

$$\begin{bmatrix}1 & 2\\3 & 4\end{bmatrix} = \begin{bmatrix} ? \end{bmatrix}$$

Matrix Multiplication Review

Use dot products for all 4 cells:

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} * \begin{bmatrix} 4 & 3 \\ 2 & 1 \end{bmatrix} = ?$$

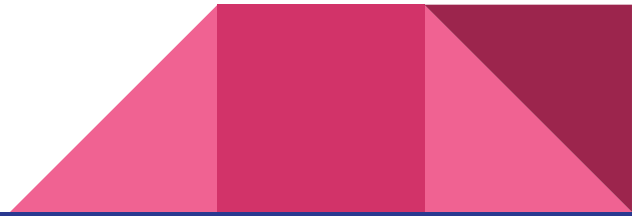
$$* \begin{bmatrix} 4 & 3 \\ 2 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} = \begin{bmatrix} 8 & 5 \\ 20 & 13 \end{bmatrix}$$

Matrix Multiplication is NOT commutative.

Given Matrix A and Matrix B that are non-trivial / diagonal,

AB != BA



Matrix Multiplication is NOT commutative.

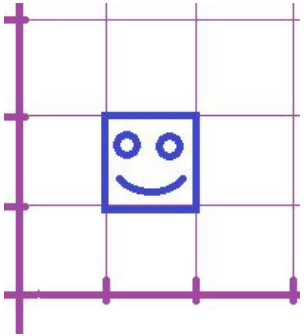
Remember our old rotation matrix:

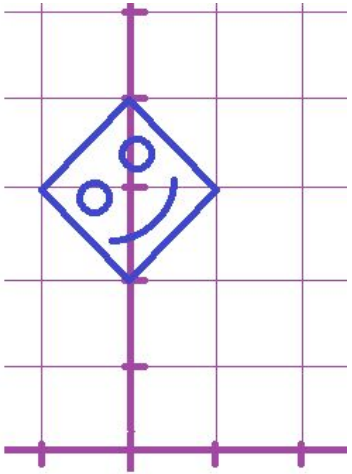
$$scale(\sqrt{2}) * rotate_z(45°) = ?$$

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

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

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

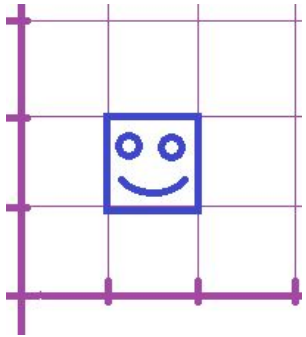




Matrix Multiplication is NOT commutative.

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

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

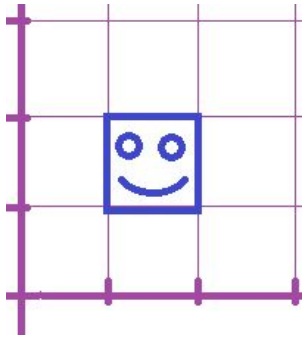


Where do the corners of the face go if we use this one?

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?

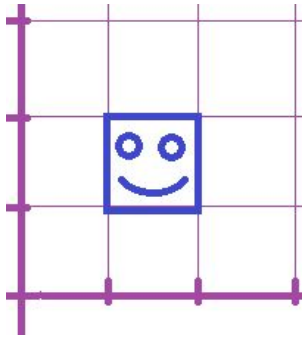
Matrix Multiplication is NOT commutative.

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:

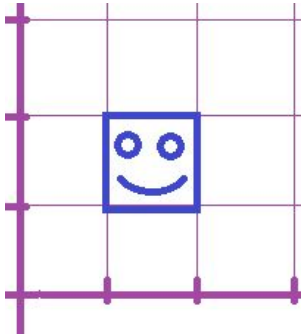
We sheared it!

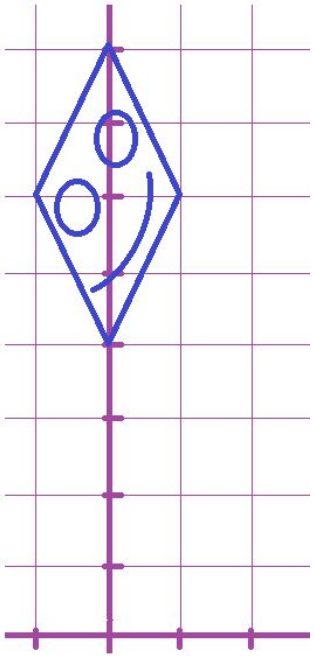
$$\begin{bmatrix} 1 & -1 \\ 2 & 2 \end{bmatrix} * \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 4 \end{bmatrix}$$

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

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

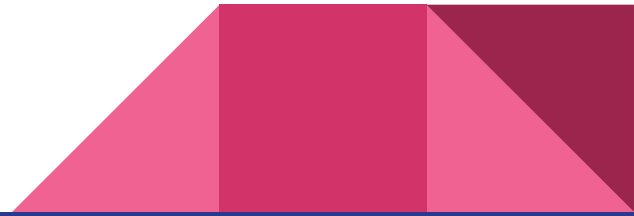
$$\begin{bmatrix} 1 & -1 \\ 2 & 2 \end{bmatrix} * \begin{bmatrix} 2 \\ 2 \end{bmatrix} = \begin{bmatrix} 0 \\ 8 \end{bmatrix}$$





Matrix Multiplication is NOT commutative.

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

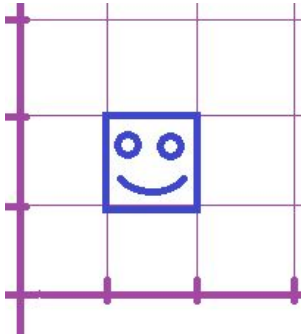


Matrix Multiplication is NOT commutative.

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?

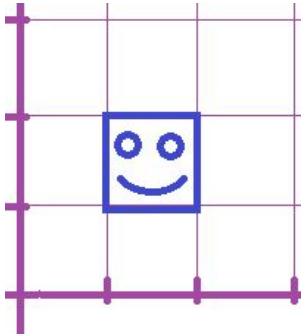


Matrix Multiplication is NOT commutative.

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?



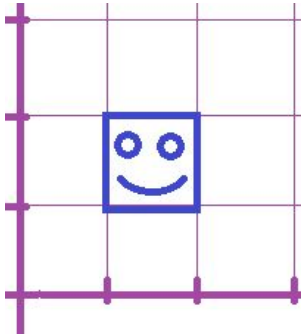
Matrix Multiplication is NOT commutative.

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:

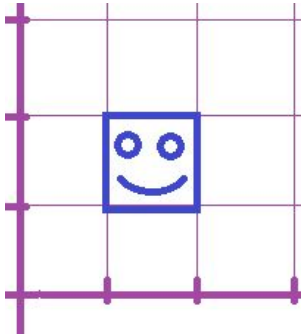
We didn't shear it!

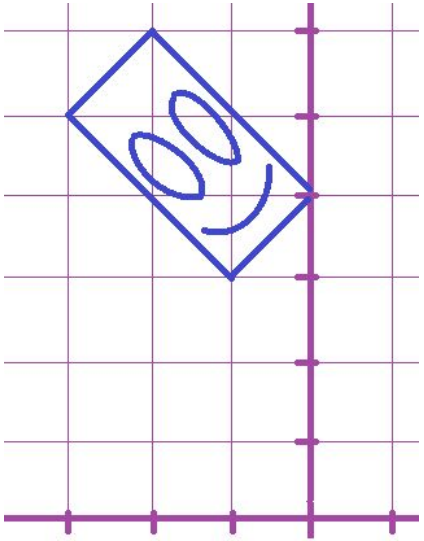
$$\begin{bmatrix} 1 & -2 \\ 1 & 2 \end{bmatrix} * \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} -1 \\ 3 \end{bmatrix}$$

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

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

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

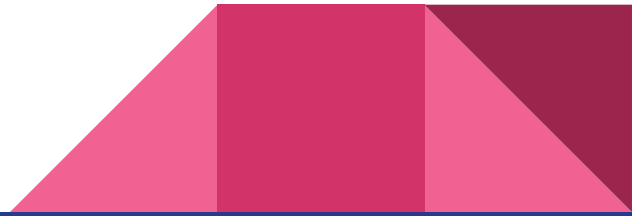




Matrix Multiplication is NOT commutative.

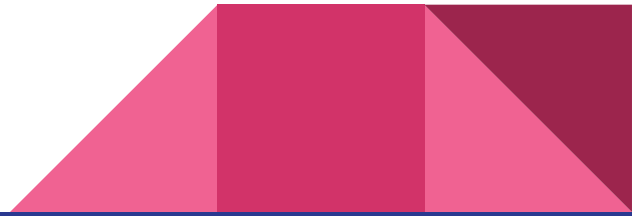
Non-uniform scales anywhere in the matrix chain could produce a shear later.

Since shears are rarely desired, non-uniform scales are typically put at the very end of a matrix chain (to the immediate left of the point) and then we undo that most recent part of the transform before drawing the next shapes.



Matrix Multiplication is NOT commutative.

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



Part III: Matrix Orderings

Matrix Multiplication is NOT commutative.

Let’s look at another instance of scale matrices misbehaving…

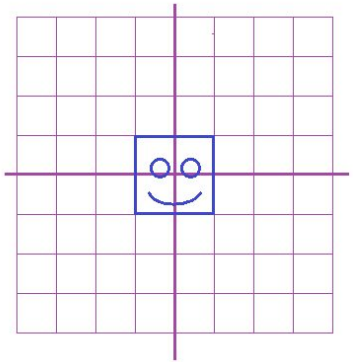
Translate * Scale

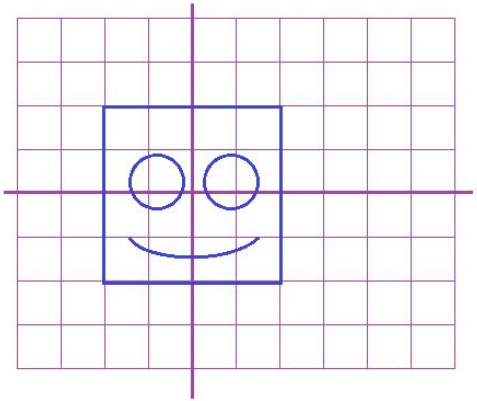
vs

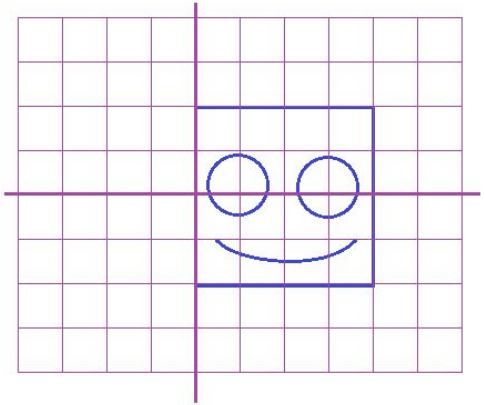
Scale * Translate

Matrix Multiplication is NOT commutative.

$Translate_x(2) * Scale(2) * p$



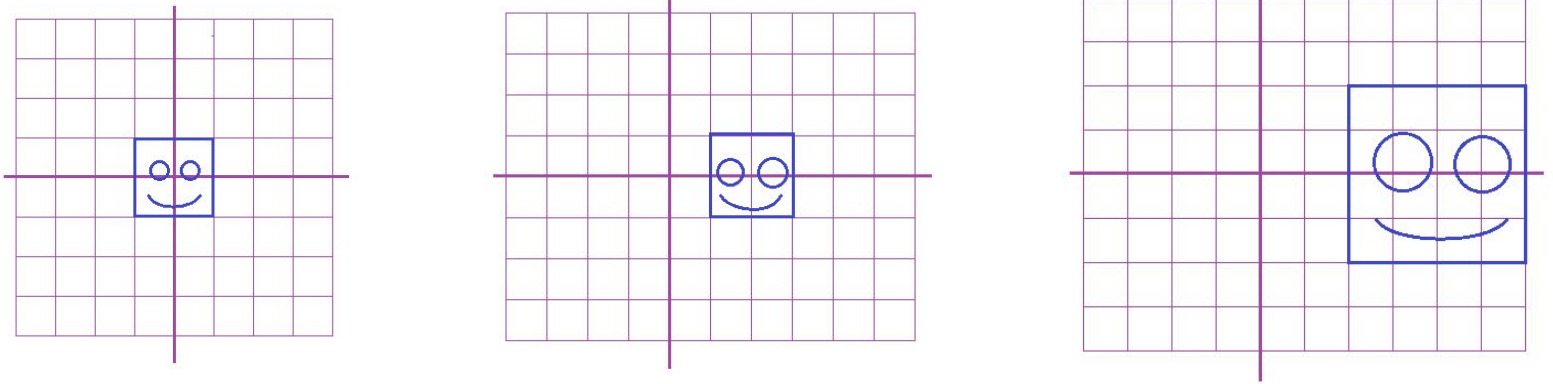




(Read it backwards -- the scale happens to points first because it’s closer to the point in the equation)

Matrix Multiplication is NOT commutative.

Scale(2) * Translate$_x$(2) * p



(Read it backwards -- the translate happens to points first because it's closer to the point in the equation) Notice how the gap scales away from the origin same as the face!

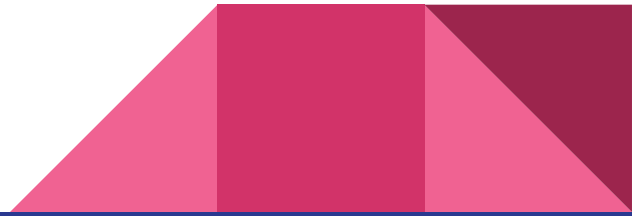
Matrix Multiplication is NOT commutative.

Another way to think of the previous two slides:

Read the multiplication forwards instead, and this time let the coordinate system axes follow you instead of remaining static on each transform.

You should come up with the same final picture even though you did the operations backwards, because unlike last time, now a scale affects how far a future translate moves - it scales your basis.

This would be thinking in **bases** instead of **points**.



Matrix Order

- When our projects start using lines of code like this one order suddenly becomes a thing:

```
model_transform = mult( model_transform, translation( 0, 0, .25 ) );
```

- That's the general way to accumulate new little pieces into model and camera matrices.

Matrix Order

```
model_transform = mult( model_transform, translation( 0, 0, .25 ) );
```

- It takes your variable's current matrix and multiplies another mat4 to the right of it, and that becomes the "new" matrix.
- Since each of them sticks another matrix onto the right side of the chain, changing the order of these lines in your code changes the left-to-right ordering of your multiplication formula!!!!

Matrix Order

- That’s the only reason order matters in your code.
- On an exam, you can just write the whole chain of matrices out left-to-right into a complete formula so there is no longer any concept of "first" or "last“, then you can’t mess up order

THE END