

# Computer Graphics

Discussion 2

Garett Ridge, Sam Amin

[garett@cs.ucla.edu](mailto:garett@cs.ucla.edu), [samamin@ucla.edu](mailto:samamin@ucla.edu)

# Part I: Practical concepts - Shapes

Getting ready to code shapes

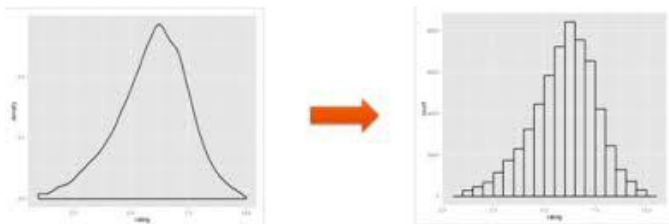
# We'll make shapes out of math.

- Remember your high school teachers making you do it with graphing calculators? This will be the more advanced version of that.

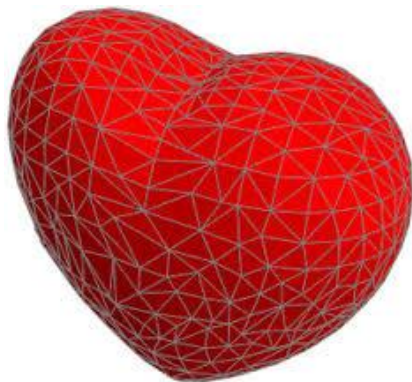
The difference: We're mostly trying to draw functions that are not linear or even polynomial.



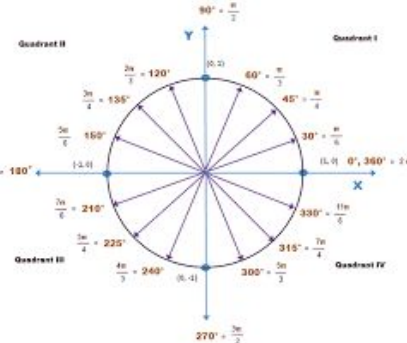
# Discretization



- We don't know how to tell a computer to draw most shapes because of their complicated non-linear formulas.
- Instead, we linearize those shapes: Break them up into a finite number of line segments between discrete points



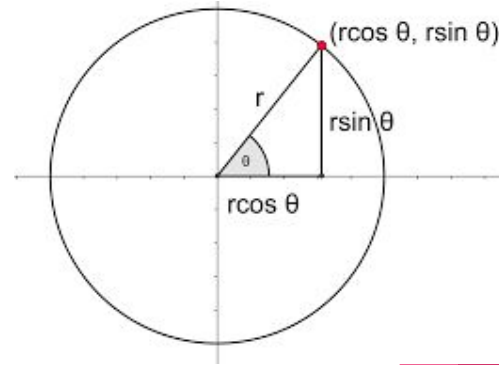
# Let's list N # of points around a circle.



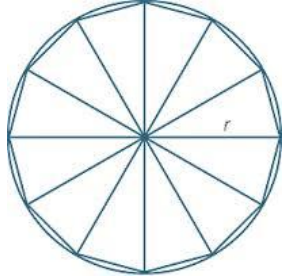
- First point: (1,0,0)
- The next point: Wherever it is, when interpreting its position as a vector, we can split it into one vector per each axis so that we have right triangles. Now we can use trig on it
- We know the diagonal's length
  - (It's the radius  $r$ , distance to the circle)
- By definition  $\cos(\theta) = x/r$  and  $\sin(\theta) = y/r$

# Let's list N # of points around a circle.

$x = r \cdot \cos(\theta)$ ,  $y = r \cdot \sin(\theta)$  where  $\theta$ , our parameter, takes N tiny steps from  $0 \dots 2\pi$



# Triangles



- We want to draw the whole 2D area, not just some points, so to discretize the shape with its approximate area let's use triangles
  - Simplest 2d shape (remove any points and it will make it 1d) - this makes triangles the "2D simplex"
- List the points in triangle order - two approaches:
  - Sort list into triples of points, or
  - Make a separate list of sorted triples of indices
    - Indices are shorter to write, so more can fit in a CPU cache. Our lists of points can be huge so this matters.



# Triangles

So, we have either...

$(0,0), (1,0), (0.479, 0.878), (0,0), (0.479, 0.878), (0.841, 0.540)...$

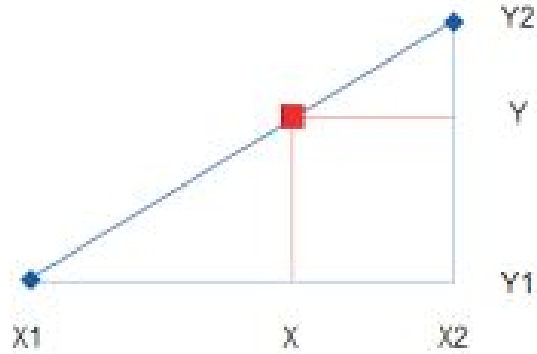
or...

0,1,2,0,2,3,0,3,4,0,4,5,0,5,6,0,6,7...





## Another exercise:



$$\frac{(X - X1)}{(X2 - X1)} = \frac{(Y - Y1)}{(Y2 - Y1)}$$

$$Y = Y1 + (X - X1) \frac{(Y2 - Y1)}{(X2 - X1)}$$

- List some points along a line from one point to another - This process is called convex interpolation



## Another exercise:

- List some points along a line from one point to another - This process is called convex interpolation
- The formula to do that is quite short:

$$p_{\text{interpolated}} = (1-a) * p_1 + a * p_2$$

- You'll be seeing a lot of math with that form
  - It's only convex and an interpolation if  $0 \leq a \leq 1$ , otherwise it's an extrapolation
- Let (a) vary from 0 to 1 in steps - this is a parametric equation. Instead we could imagine a parameter time (t) rather than (a) -- at each time t between 0 sec and 1 sec there's a different point on the line segment.



# What's a matrix?

- A system of equations
- Our mathematical hammer, with our 3D-object shaped functions as the nail
- What does a matrix mean?

$$\begin{bmatrix} 2 & 3 \\ 4 & 5 \end{bmatrix} \begin{bmatrix} x_{old} \\ y_{old} \end{bmatrix} = \begin{bmatrix} x_{new} \\ y_{new} \end{bmatrix} \Rightarrow \begin{aligned} x_{new} &= 2x_{old} + 3y_{old} \\ y_{new} &= 4x_{old} + 5y_{old} \end{aligned}$$



# Matrix Concepts

- Guiding a shape into place in a scene
- Matrix types



## Elementary Matrices

An elementary matrix  $E$  is an  $n \times n$  matrix that can be obtained from the identity matrix  $I_n$  by one elementary row operation.

$$E = e(I)$$

where  $e$  is an elementary row operation.

- All elementary matrices are invertible, an inverse exists.
- The inverse of an elementary matrix is also an elementary matrix.

Recall identity matrices

$$[1] \quad \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \dots$$

Row Replacement:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -4 & 0 & 1 \end{bmatrix} \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ g-4a & h-4b & i-4c \end{bmatrix}$$

Interchange of Rows:

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} = \begin{bmatrix} d & e & f \\ a & b & c \\ g & h & i \end{bmatrix}$$

Multiplication by a constant:

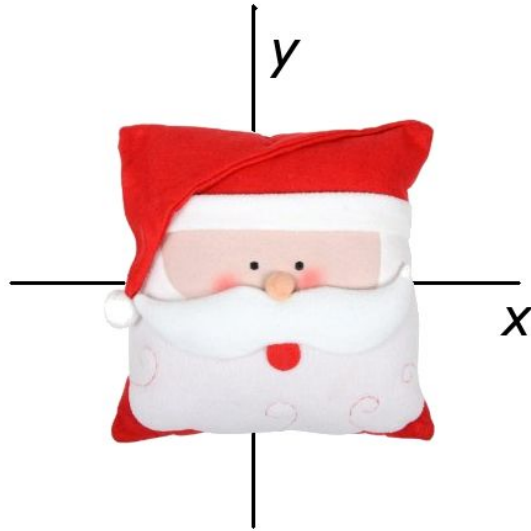
$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 5 \end{bmatrix} \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} = \begin{bmatrix} d & e & f \\ a & b & c \\ 5g & 5h & 5i \end{bmatrix}$$

OOPS Pretend these  
two rows are not  
swapped

What if we apply  $e_1$  to all the points of an image?

Interchange of Rows:

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = ?$$



# What if we apply $e_1$ to all the points of an image?

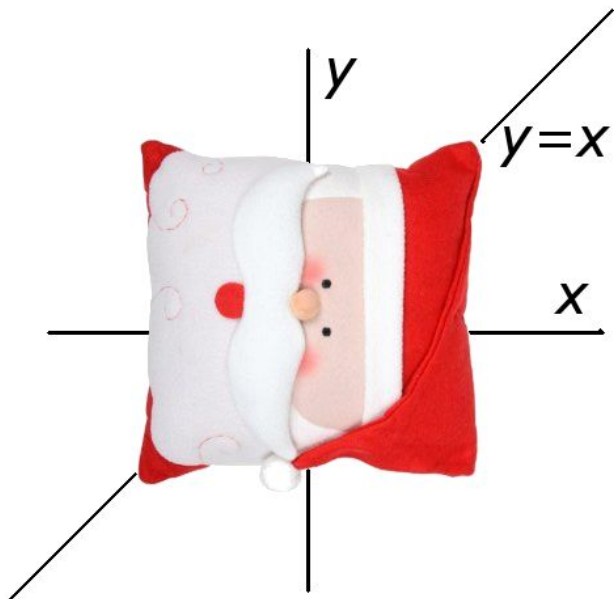
We get an (axis) inversion.

$$\text{new}_x = y, \text{ new}_y = x$$

So the inversion matrix is:

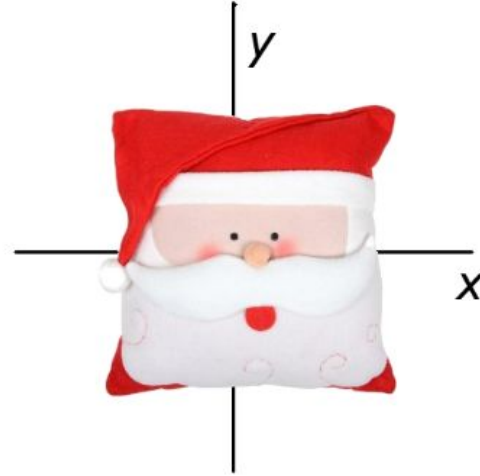
Interchange of Rows:

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = ?$$





# What if we apply $e_2$ to all the points of an image?



Multiplication by a constant:

$$\begin{bmatrix} 5 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = ?$$

# What if we apply $e_2$ to all the points of an image?

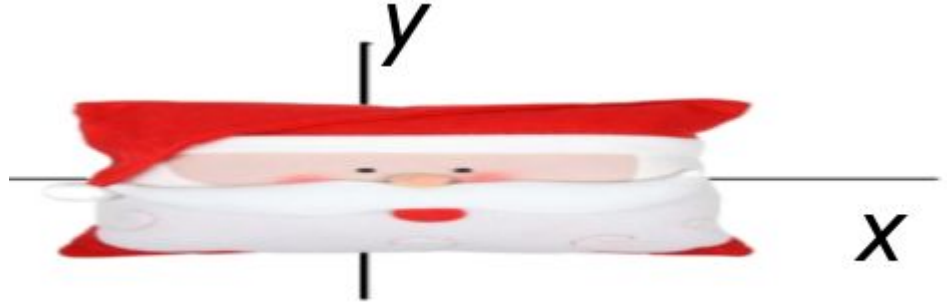
We get a scale.

$$\text{new}_x = 5x$$

So the scale matrix is:

Multiplication by a constant:

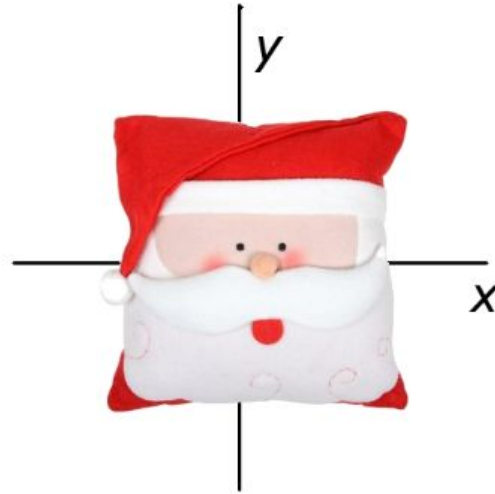
$$\begin{bmatrix} 5 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = ?$$



What if we apply  $e_3$  to all the points of an image?

Row Replacement:

$$\begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = ?$$



# What if we apply $e_3$ to all the points of an image?

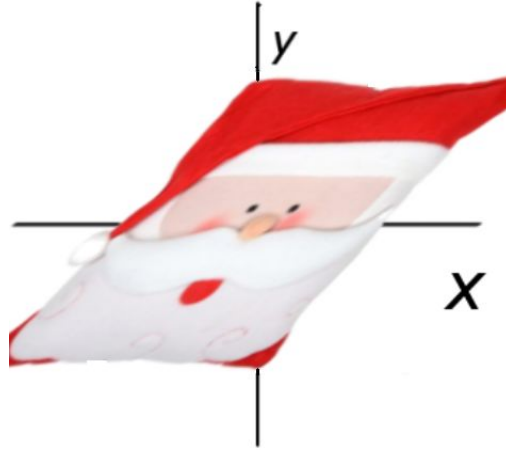
We get a shear.

$$\text{new}_x = x + y$$

So the shear matrix is:

Row Replacement:

$$\begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = ?$$




# Rotation Matrices

A rotation is two shears.

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

But certain properties have to hold. This one is not a pure rotation matrix (it has some scaling effect too). Rotations have to have determinant = 1.



# Rotation Matrices

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

A rotation also has to have orthogonal columns. This one passes that!

Our simpler shear matrix from earlier did not (check the columns):

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



# Rotation Matrices

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

This one is more like it. **Trig** identities ensure this matrix always has determinant=1. The two columns are also always perpendicular. Suppose theta is 45 degrees:



# Rotation Matrices

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

This one is more like it. **Trig** identities ensure this matrix always has determinant=1. The two columns are also always perpendicular. Suppose theta is 45 degrees:

$$\begin{bmatrix} \cos 45^\circ & -\sin 45^\circ \\ \sin 45^\circ & \cos 45^\circ \\ & & 1 \end{bmatrix} = \begin{bmatrix} .5\sqrt{2} & -.5\sqrt{2} \\ .5\sqrt{2} & .5\sqrt{2} \\ & & 1 \end{bmatrix}$$






# Rotation Matrices

Let's try our rotation matrix, but we don't want to multiply ugly radicals so let's combine it with a scale matrix for testing.

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

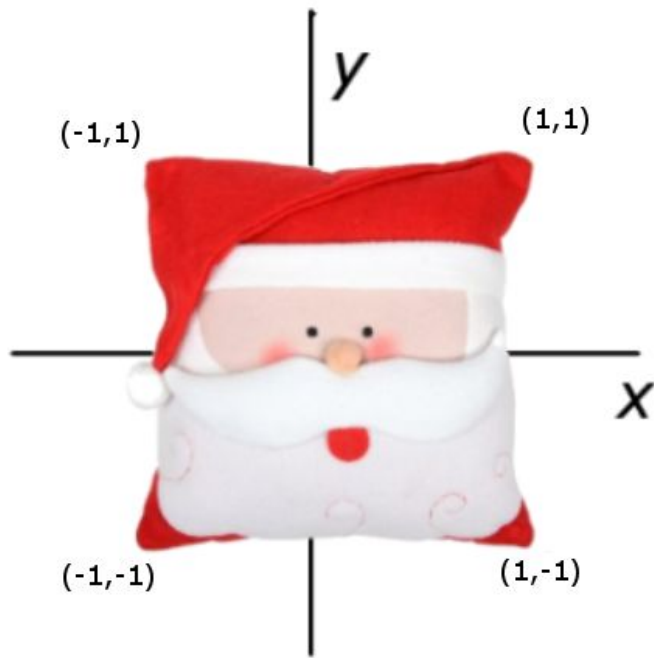
There, much better to work with. (This was our matrix from earlier that had a scaling influence).



# Rotation Matrices

What it means:

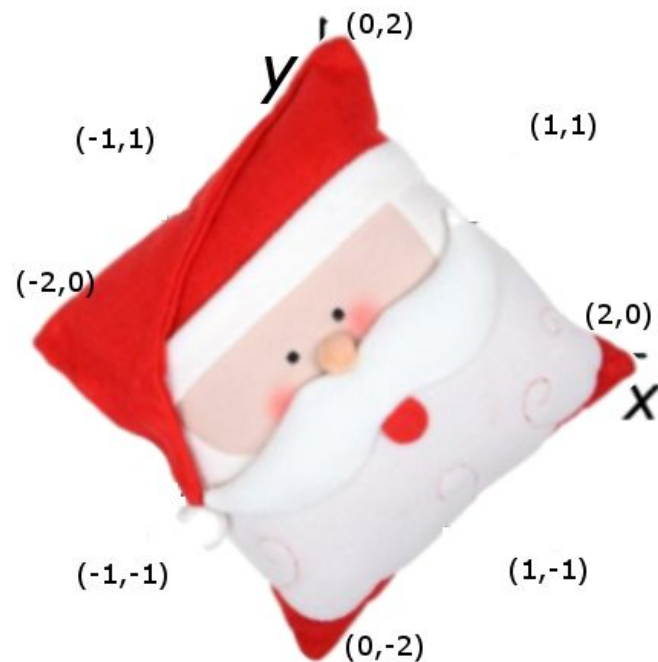
$$\begin{bmatrix} 1 & -1 \\ 1 & 1 \\ & & 1 \end{bmatrix} \Rightarrow \begin{aligned} x_{new} &= x_{old} - y_{old} \\ y_{new} &= x_{old} + y_{old} \end{aligned}$$



# Rotation Matrices

What it means:

$$\begin{bmatrix} 1 & -1 \\ 1 & 1 \\ & & 1 \end{bmatrix} \Rightarrow \begin{aligned} x_{new} &= x_{old} - y_{old} \\ y_{new} &= x_{old} + y_{old} \end{aligned}$$



# All this time...

SANTA'S NOSE HAS NEVER LEFT THE ORIGIN.



# Translation Matrices

- Every matrix we've discovered using the elementary ones only has a linear effect, stretching and warping the image around the origin in various ways
- How can we accomplish translation of Santa to other places?
- We can't with our current machinery. Why?
- Every point always takes a contribution of  $x_{\text{old}}$ ,  $y_{\text{old}}$ , and  $z_{\text{old}}$  to get its value, and so the origin point (0,0,0) will always contribute zero to each axis for the new point, mapping onto itself.
- We need to be able to take a component of some other quantity besides  $x_{\text{old}}$ ,  $y_{\text{old}}$ , and  $z_{\text{old}}$
- Our vector we're multiplying needs to grow by 1 term.

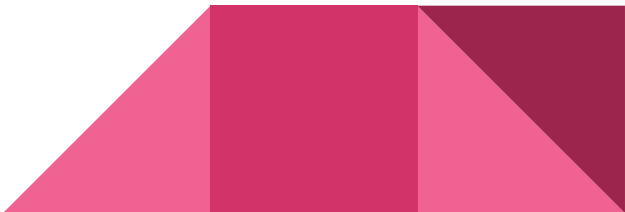


# Translation Matrices

Solution:

$$\begin{bmatrix} ? & ? & ? & ? \\ ? & ? & ? & ? \\ ? & ? & ? & ? \\ ? & ? & ? & ? \end{bmatrix} \begin{bmatrix} x_{old} \\ y_{old} \\ z_{old} \\ 1 \end{bmatrix} = \begin{bmatrix} x_{new} \\ y_{new} \\ z_{new} \\ 1 \end{bmatrix}$$

One of the components that contributes to the answer is now the constant 1, which we can pull from even when all others are zero. This allows us to do a non-linear, affine operation to the picture - moving it.




# Translation Matrices

Solution:

$$\begin{bmatrix} ? & ? & ? & ? \\ ? & ? & ? & ? \\ ? & ? & ? & ? \\ ? & ? & ? & ? \end{bmatrix} \begin{bmatrix} x_{old} \\ y_{old} \\ z_{old} \\ 1 \end{bmatrix} = \begin{bmatrix} x_{new} \\ y_{new} \\ z_{new} \\ 1 \end{bmatrix}$$

We'd like our new vector to have "1" in the 4<sup>th</sup> coordinate as well, so we can use matrices on it right away. What does our matrix have to be like to ensure that will happen?




# Translation Matrices

Solution:

$$\begin{bmatrix} ? & ? & ? & ? \\ ? & ? & ? & ? \\ ? & ? & ? & ? \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{old} \\ y_{old} \\ z_{old} \\ 1 \end{bmatrix} = \begin{bmatrix} x_{new} \\ y_{new} \\ z_{new} \\ 1 \end{bmatrix}$$

Just make sure the bottom row is like above. This 4th row is an equation that literally says “ $1_{new} = 1_{old}$ ”.



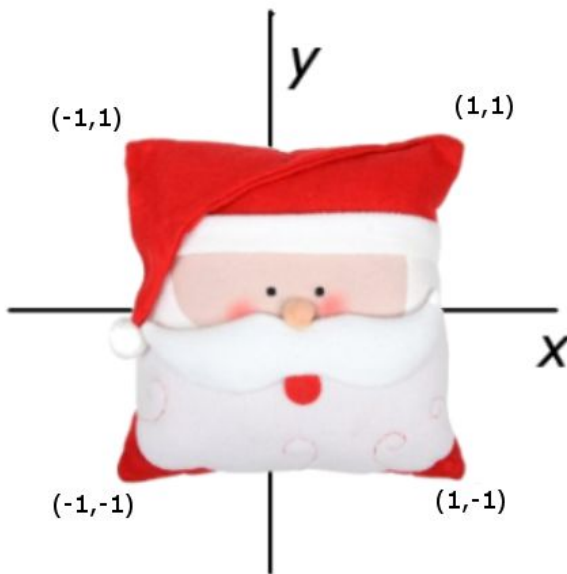


# Translation Matrices

3D translations are the reason we use 4x4 matrices instead of 3x3.

Let's do one:

$$\begin{bmatrix} 1 & & & \\ & 1 & & \\ & & 1 & \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{old} \\ y_{old} \\ z_{old} \\ 1 \end{bmatrix} = \begin{bmatrix} x_{new} \\ y_{new} \\ z_{new} \\ 1 \end{bmatrix}$$

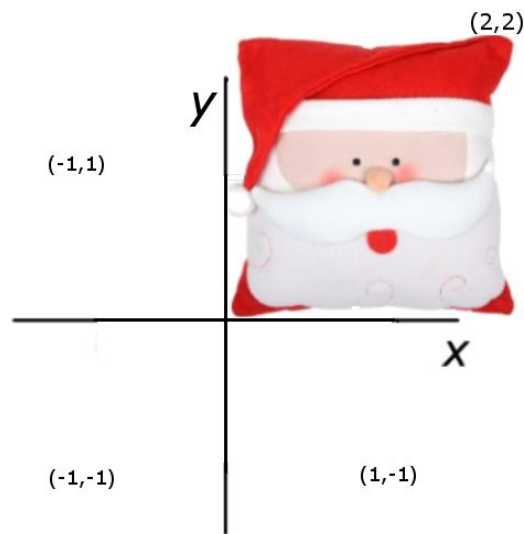


# Translation Matrices

3D translations are the reason we use 4x4 matrices instead of 3x3.

Let's do one:

$$\begin{bmatrix} 1 & & & \\ & 1 & & \\ & & 1 & \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{old} \\ y_{old} \\ z_{old} \\ 1 \end{bmatrix} = \begin{bmatrix} x_{new} \\ y_{new} \\ z_{new} \\ 1 \end{bmatrix}$$



# Matrix Order

- The hardest 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.
- Two approaches. Multiply starting from:
  - Right to left or,
  - Left to right

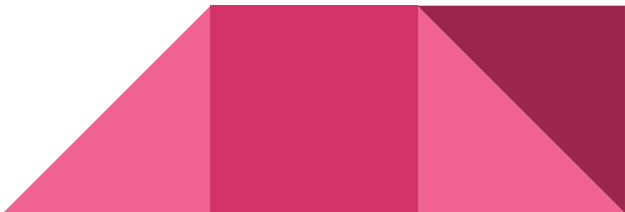




# Part II: Javascript Crash Course

Self-explanatory

# Similarities with C++

- Flow control commands like if/else/switch/for/while/do/break/continue are still all the same.
  - Functions are still called with (), and you still put input parameters in there.
  - Sequences of operations (statements) are still separated by ; semicolons...
  - Assignment still uses =.
- 

# Similarities with C++

- Blocks of multiple statements are still made by { } brackets, and are still necessary around functions, or if there are multiple statements you'd like to be included in an if / for / while / etc.
- Boolean comparisons are still the same.
- Objects still have member variables and members are still obtained through . (dot).
- "this" still refers to the current object.
- Code comments are still // or /\* \*/.
- Always use ==

## Similarities with C++

- Following your code line by line in a debugger continues to be practically more important than running it, although now you'll be using a built-in debugger in your web browser.



# Most Obvious Difference

Variable declaration:

- “var” for every variable
- Javascript figures out type for you

Examples:

- a for-loop -- "var counter = 0" instead of "int counter = 0"
- a whole matrix --just say var, as in "var x = model\_transform".

When writing a function, **don't** say "var" before each argument, that won't compile.

Ints and floats are treated the same -- no truncation errors






# More Differences

- Don't have to warn javascript that your function is going to return a certain type of value. Just do it by saying "return" whatever when it's time to.
- Javascript loses track of its "this" pointer quite easily, forcing you re-name "this" as another variable and pass it in to your function yourself (I usually re-name it to "self")
- The keyword "this" isn't automatically implied in javascript code even if you're in a member function.
  - For example, you have to fully spell out "this.draw\_flower()" when calling your function, you have to fully spell out "this.graphicsState.animation\_time" instead of just "graphicsState.animation\_time" when you're using that variable to make things move.
- To use sine and cosine in javascript, you have to look in the Math namespace; you say Math.sin and Math.cos. These expect radians

# Relevant to You


- Your time measurements are given in milliseconds.
  - We use a matrix library our textbook author wrote (where our math data types are called `vec2`, `vec3`, `vec4`, and `mat4`).
  - Some of your matrices will be made automatically in loops where naming them all won't be easy. Because of that you will probably want a "stack" of matrices keeping track of the most current, second most current, etc.
    - To make a stack in javascript, you just make an array. It will look like: `this.stack = []`;
    - Afterwards, `this.stack.push( model_transform )` saves the current matrix
    - `this.stack.pop()` returns the most recent (and takes it off the stack - if you don't read from the last array element)
- 

# One More Thing

One of the requirements for homework 1 will be to break up your code into functions. In order to keep track of your "current" matrix even as you enter and leave functions, it's a good idea to have all your functions use matrices as in and out variables.

- Take `model_transform` as a parameter, and return the new `model_transform` as the return value. This lets you keep a "current" matrix with you as you pass from one function to another, while still letting you break up your code into functions

Javascript doesn't have "pass by reference" variables. Using return values to send the updated matrix back to the caller is the simplest way to keep your matrix current.



# Writing Your Own Function

For now, to write your own function, use the following syntax. Insert it into the list of class Animation's (public, inheritable) methods this way:

```
Animation.prototype.draw_flower = function( model_transform )  
{  
    ... (code for drawing your scene's flower goes here)  
  
    return model_transform;  
}
```

All placed at the bottom of animation.js

