Discussion 1 Computer Graphics 174a

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

Part I: Course Logistics

Forum, Textbook, Projects

Piazza Forum

- Join it on piazza.com now - ask if you need help
- All our homework will be largely solved on there
- Also will have the assignment materials for download on there, including your code template (the skeleton you'll fill in to do homework)
- The code template will be posted on Piazza ~Sunday. A web document and a few short javascript files.
- Pay attention to Piazza this week, because that's where announcements will happen about setting up your hosting & debugging

Which language will we use?

- Three common languages for graphics are C++, Java, and WebGL.
- All of these use OpenGL (graphics card API)
- WebGL = JavaScript plus OpenGL calls
- C++ plus OpenGL calls, on the other hand, is sometimes just called OpenGL

WebGL is the one we're going to use.

- JavaScript is the language of the web. Nice because everything is done in a browser.
- We'll provide a crash course for beginners
- Not much relation to Java. JavaScript has unusual stuff like closures, prototypes, and self-executing functions

Which language will we use?

- JavaScript handles a little bit easier than the C++ version of OpenGL.
  - Less typing / some saved steps
  - Modern web browsers have a built in graphics debugger that gives great error feedback from the GPU
  - The internet's not going away
  - No setup required
    - Runs from a file immediately
    - Edit each file directly
  - Running all demos is as simple as visiting links
  - Your final projects are easy for others to play with

Which language will we use?

- The way to organize a graphics program is the same across languages
- The code skeleton is easy to translate to C++ or Java
- Most lines you'll type for graphics commands look the same in each language
  - Making a scene can mean copying and pasting these commands instead of typing
  - When you're not copying and pasting, you can look through the template to see how javascript does things and mimic.

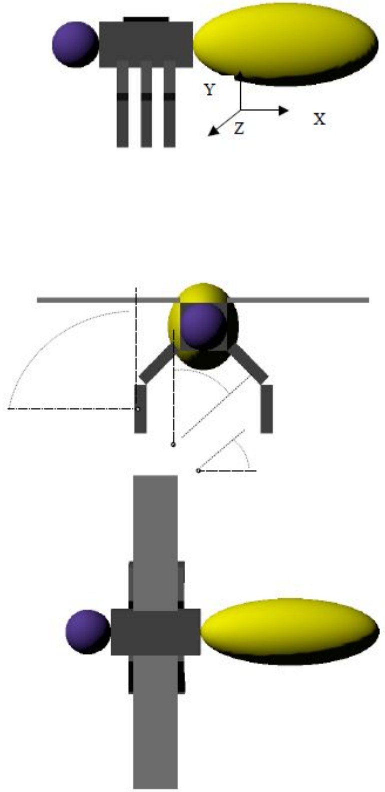
Which textbook?

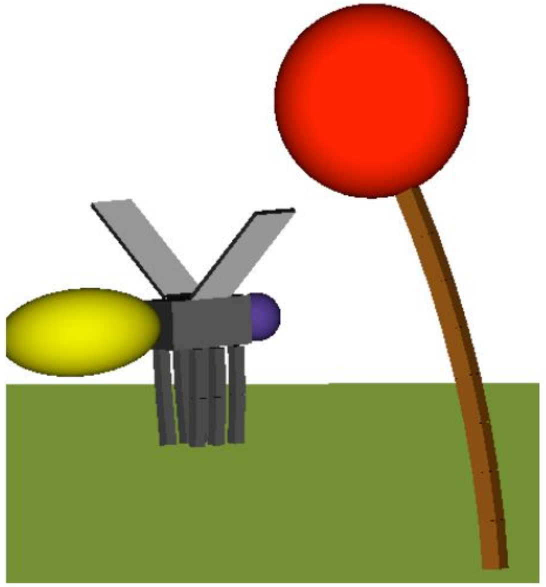
- Even though the bookstore says “optional”, this course closely follows the chapters of “Interactive Computer Graphics” by Edward Angel.
- The course updated to JavaScript when the textbook did
- Only the newest (7th) edition is in our language

Assignment 1: Animate a bee

- Mimic our animation as closely as you can
- Connect boxes and balls correctly into a bee, flower, and ground
- Required:
  - Bee parts must connect at exact corner edges
  - Your scene must be noticeably unique from everyone else's
- You'll be graded on:
  - Shapes must stay connected the same way when pivoting
  - Fluidity of movements

Assignment 1: Animate a bee





Assignment 2: Animate anything

- Make any animation or game as long as it includes all the required techniques
- Graded on creativity, complexity, and attention to detail
- Yours will be played and voted on by the class -- the best receive extra credit towards the course grade
- Yours can join prior years' showcases online. Example:

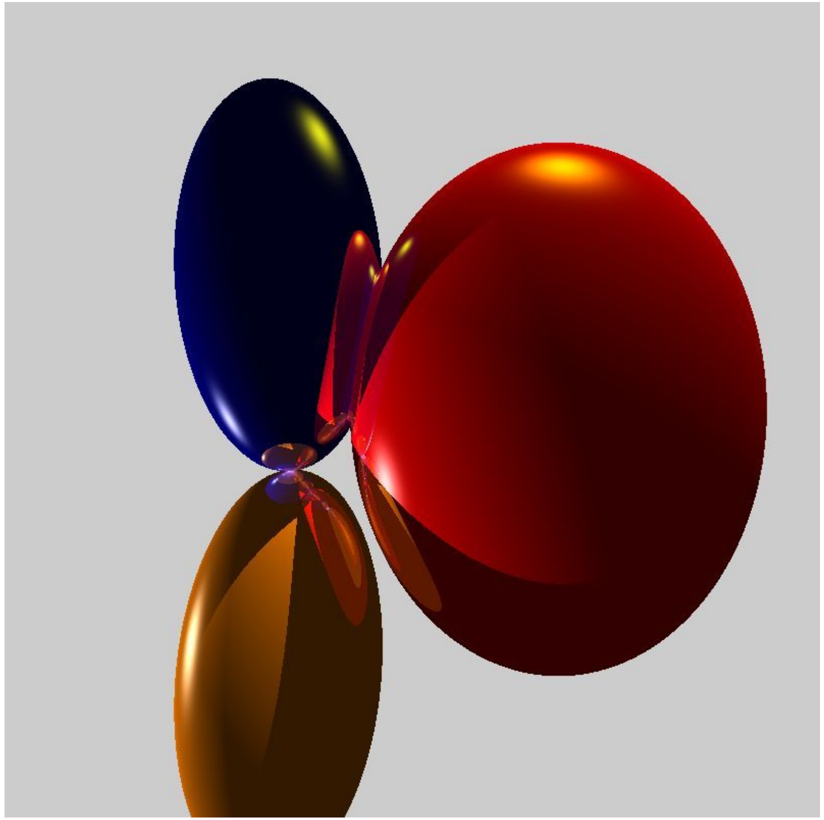
  http://web.cs.ucla.edu/~dt/courses/CS174A/animations/assignment2-best-16s/

Assignment 3: Trace Rays through our scenes

- We give you a full description of a simple scene
- You'll build an image by mathematically following rays of light that bounce around the objects in the scene
- You'll be graded on implementing all the features (reflections, refractions, shadows, lighting) and correctness of images

Project 3

A test image:



Part II: Practical concepts - Making Shapes

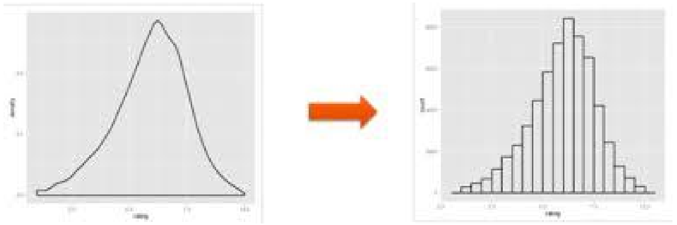
With math and code

We'll make shapes out of math.

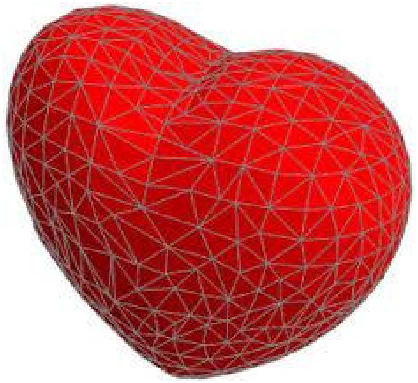
- Remember graphing shapes with a calculator? This will be the more advanced version of that.

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

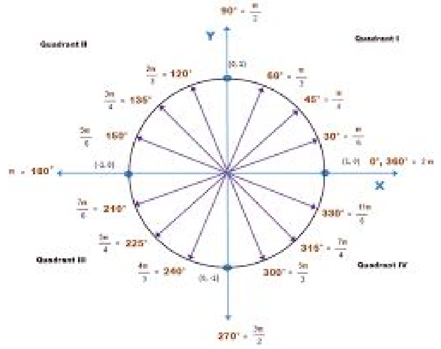
Discretization



- 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 N discrete points



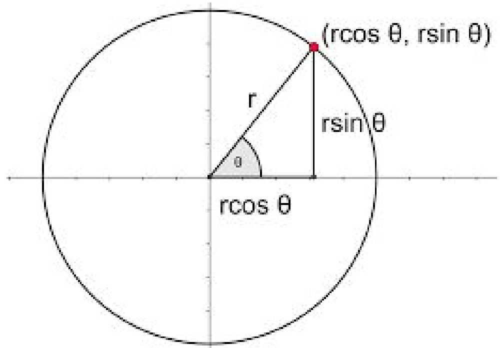
Let's list N points around a circle.



- 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

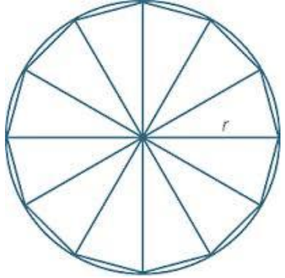
Let's list N points around a circle.

x = r*cos(Θ), y = r*sin(Θ) where theta is as shown below.



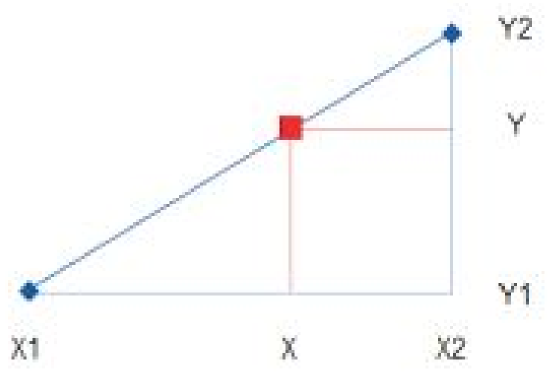
Using Θ as a variable input parameter, take N tiny steps from 0...2*PI.

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
    - (0,0), (1,0),(0.479, 0.878), (0,0), (0.479, 0.878), (0.841,0.540)...
  - Or, make a separate list of sorted triples of indices
    - Indices are shorter to write, so more can fit more triangles in a CPU cache:
    - 0,1,2,0,2,3,0,3,4,0,4,5,0,5,6,0,6,7...

Another exercise:



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

Linear interpolation

- The formula to do that is quite short:

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

  - It’s only an interpolation (i.e. convex) if $0<=a<=1$
  - Otherwise it's an extrapolation
- You’ll be seeing that form a lot this quarter!!!
- Let (a) vary from 0 to 1 in steps - this is a parametric equation.
- Or we could imagine a parameter time (t) rather than (a) -- at each time t between 0 sec and 1 sec we reach a different point on the line segment.

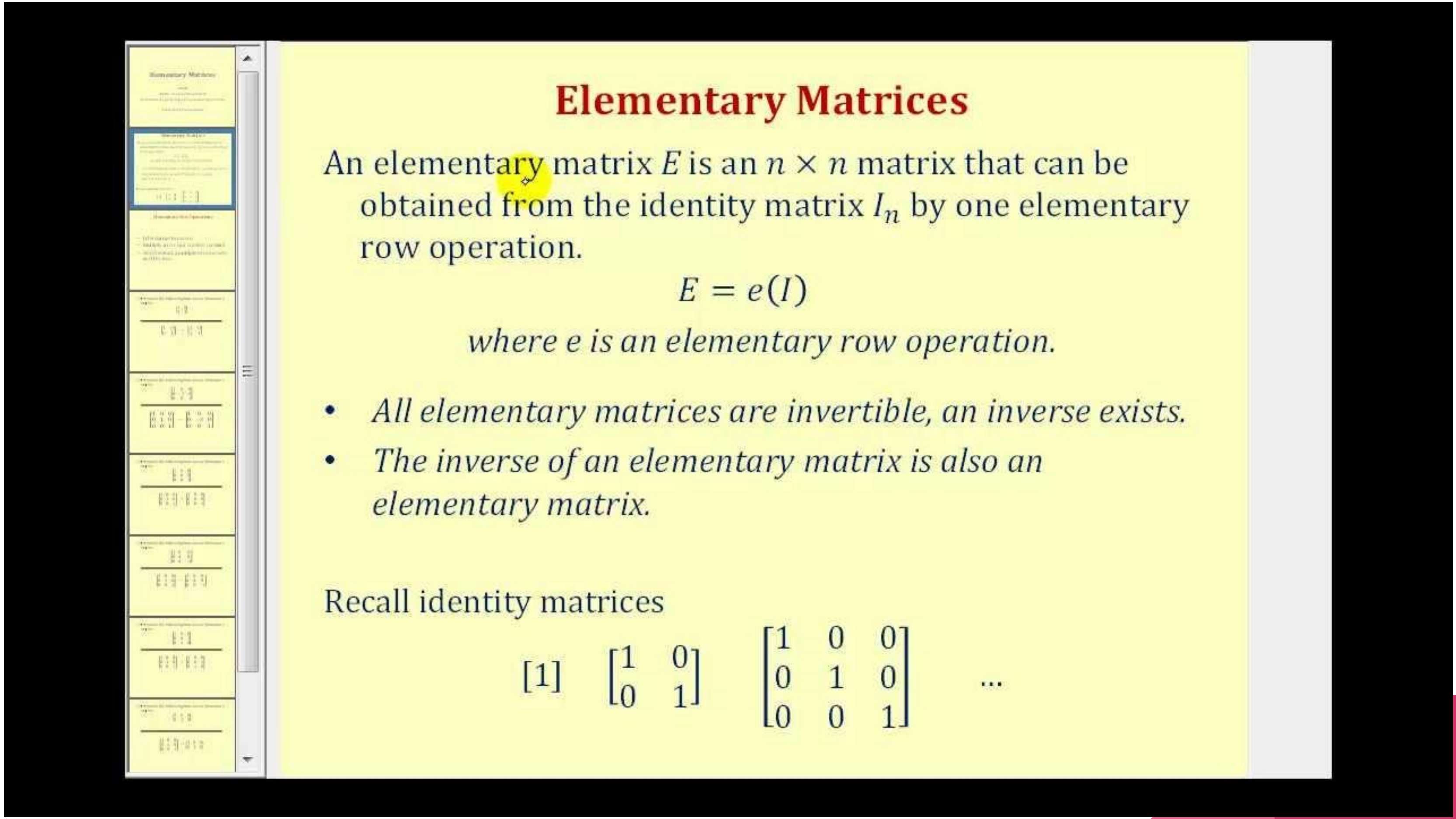
What's a matrix?

- A system of (first order) equations
- Our mathematical tool to tweak 3D functions or point clouds with
- Remember that a matrix is shorthand:

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

Matrix Concepts

- Guiding a shape into place in a scene
- A few common matrices perform movement or warp space.
- Recall elementary row operations:



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.

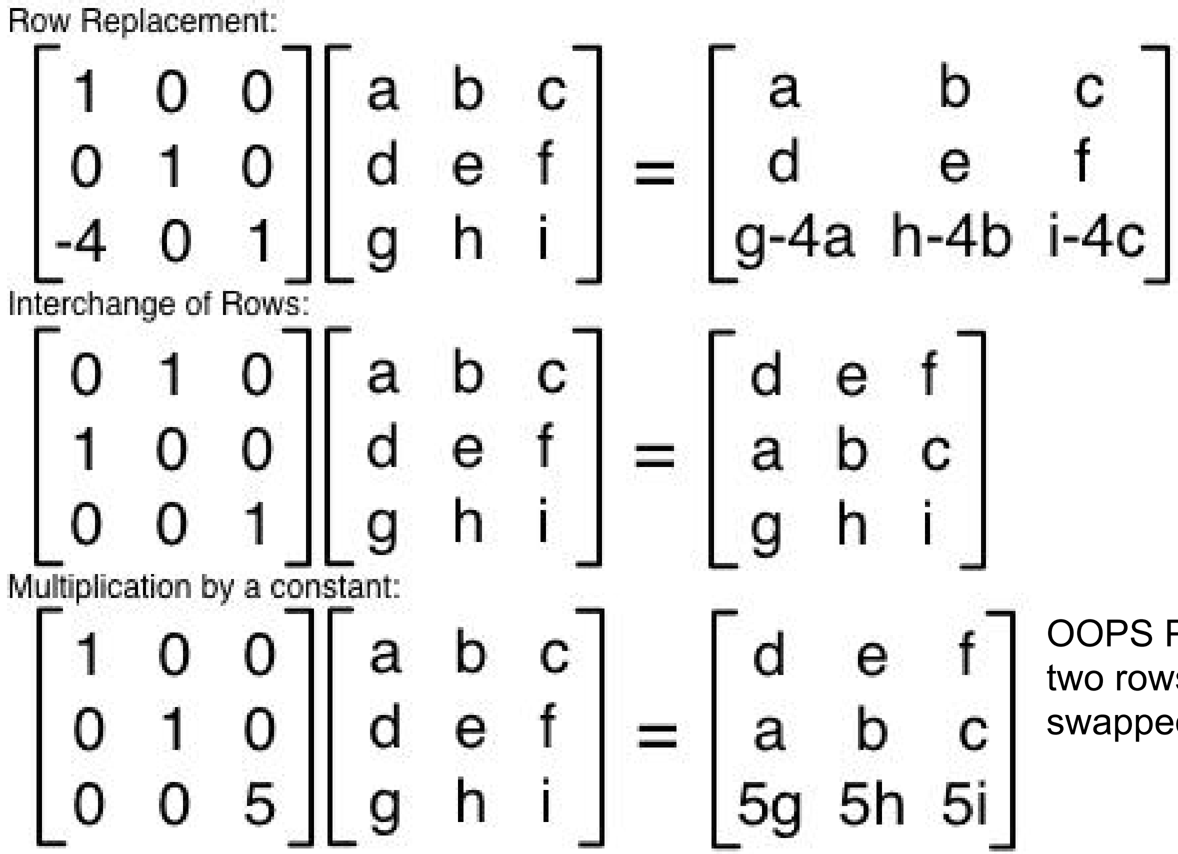
$$E = e(I)$$

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

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



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:

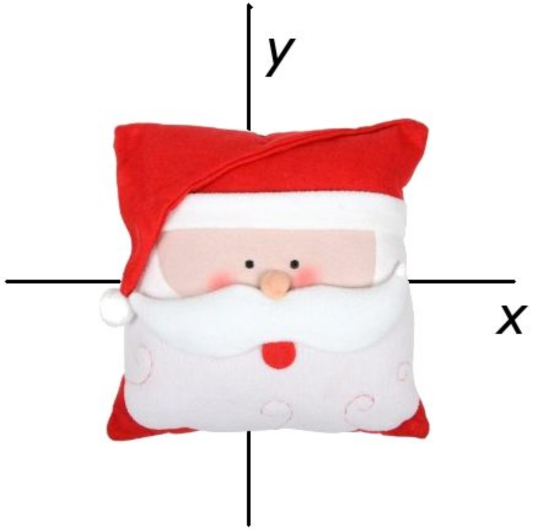
$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 5 \end{bmatrix}\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} = \begin{bmatrix} d & e & f \\ a & b & c \\ 5g & 5h & 5i \end{bmatrix}$$

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?

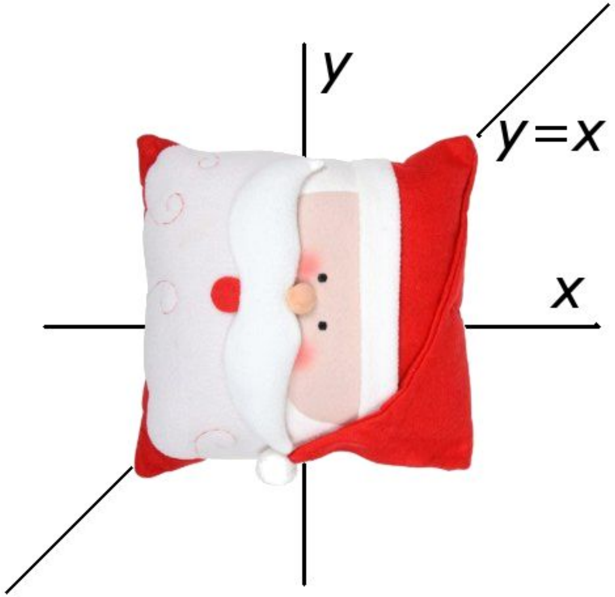
We get an axis swap.

$new_x = y$, $new_y = x$

Matrix form of that:

Interchange of Rows:

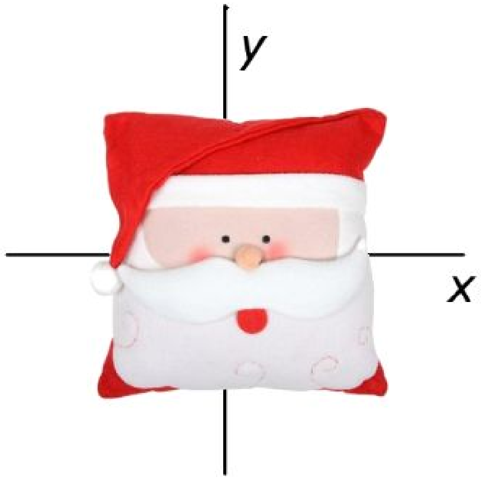
$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = ?$$



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?

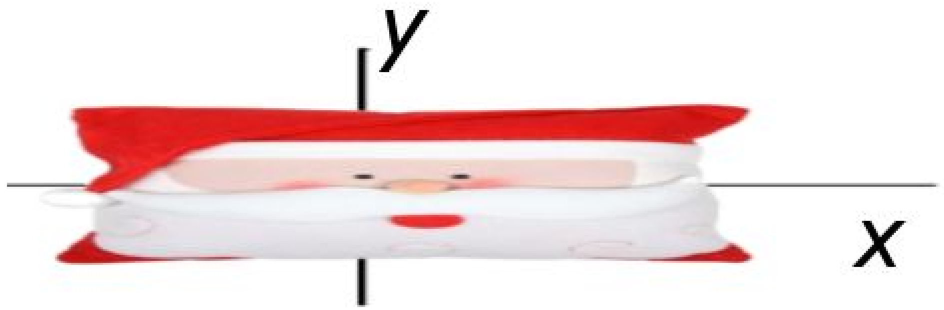
We get a scale.

$new_x = 5x$

Matrix form of that:

Multiplication by a constant:

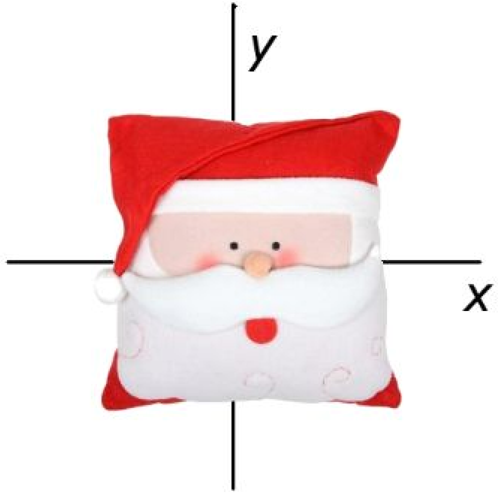
$$\begin{bmatrix} 5 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = ?$$



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?

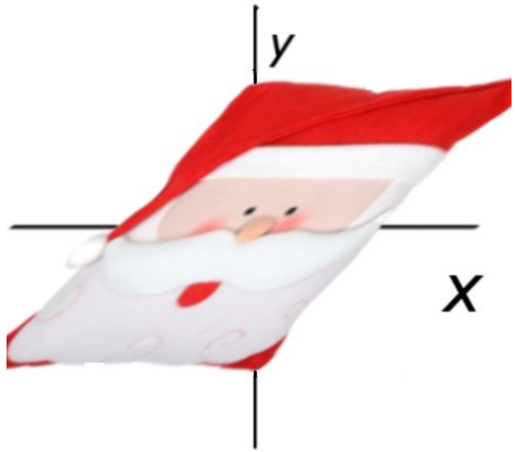
We get a shear.

$new_x = x + y$

Matrix form of that:

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:

$$\begin{bmatrix} cos45^\circ & -sin45^\circ & \\ sin45^\circ & cos45^\circ & \\ & & 1 \end{bmatrix} = \begin{bmatrix} .5\sqrt{2} & -.5\sqrt{2} & \\ .5\sqrt{2} & .5\sqrt{2} & \\ & & 1 \end{bmatrix}$$

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.

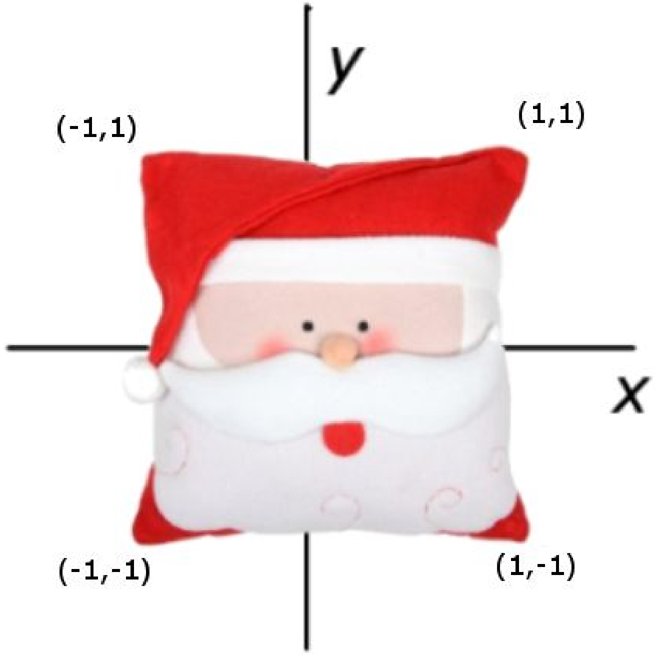
$$\begin{bmatrix} .5\sqrt{2} & -.5\sqrt{2} & \\ .5\sqrt{2} & .5\sqrt{2} & \\ & & 1 \end{bmatrix} * \begin{bmatrix} \sqrt{2} & & \\ & \sqrt{2} & \\ & & 1 \end{bmatrix} = \begin{bmatrix} 1 & -1 & \\ 1 & 1 & \\ & & 1 \end{bmatrix}$$

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

Rotation Matrices

What it means:

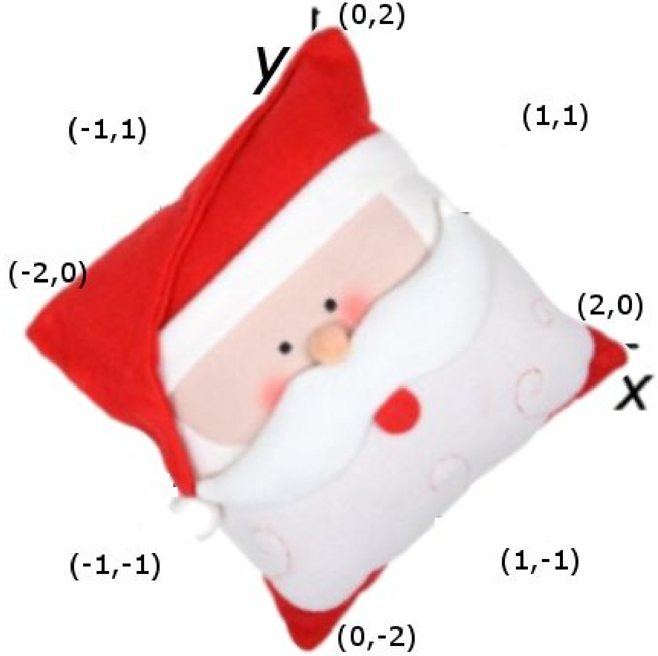
$$\begin{bmatrix} 1 & -1 & \\ 1 & 1 & \\ & & 1 \end{bmatrix} \Rightarrow \begin{array}{l} x_{new} = x_{old} - y_{old} \\ y_{new} = x_{old} + y_{old} \end{array}$$



Rotation Matrices

What it means:

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



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
- Why can't we accomplish translation of Santa to other places with our current machinery?
- Every point always takes a contribution of x_{old}, y_{old}, and z_{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_{old}, y_{old}, and z_{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 4th 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:

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

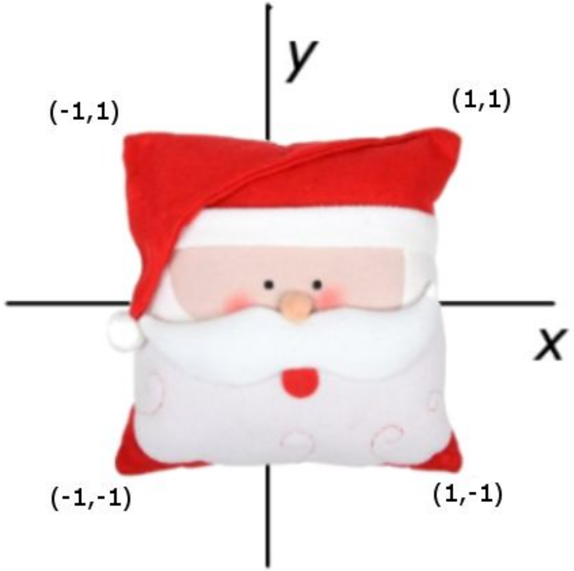
Just make sure the bottom row is like above. This 4th row is an equation that literally says “$1_{new}=1_{old}$”.

Translation Matrices

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

Let's do one:

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

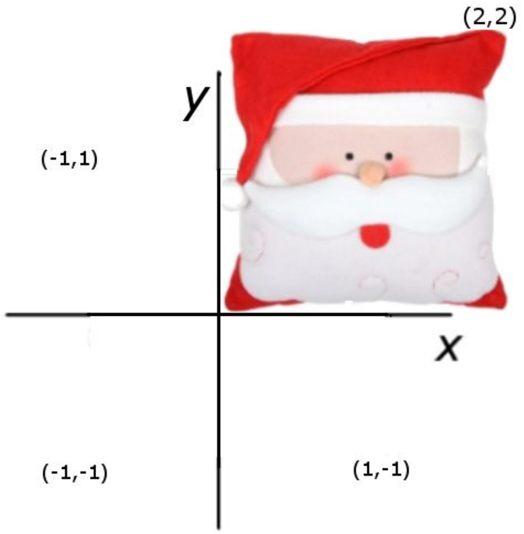


Translation Matrices

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

Let's do one:

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



Matrix Order

- The 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 products can only be <u>written</u> in one left-right order. Changing the order changes the answer.
- Matrix products can be <u>evaluated</u> in any left-right order you want though.
- 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?).

Matrix order (example)

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

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

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

etc

VS

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

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

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

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)

Part III: Writing JavaScript from a C++ background

Similarities with C++

- Flow control commands are still all the same: if/else/switch/for/while/do/break/continue
- Functions are still called with () or (...input parameters...)
- Sequences of operations (statements) are still mostly separated by ;
- Assignment still uses =

Similarities with C++

- Brackets { } are still necessary around subroutines, or if there are multiple statements you'd like in a code block or an if / for / while / etc.
- Boolean math and comparisons are still the same.
- Objects still have member variables and members are still obtained with . (dot).
- "this" still refers to the current object.
- Code comments are still // or /* */.

Similarities with C++

- Following your code line by line in a debugger continues to be critical for getting it to work, although now you'll be using a built-in debugger in your web browser.

Main Difference

Variable declaration

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

  Examples:
  - Loop counter: "var counter = 0" instead of "int counter = 0"
  - Custom matrix: "var x = mat4(); // (the identity)".

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 about return types when declaring functions. Just say “function” and return any expression when it’s time to.
- Javascript loses track of its “this” pointer quite easily; sometimes you need to pass it in manually under a different name (such as “self”)
- The keyword "this" isn't automatically implied in javascript code even if you're in a member function.
    - You have to fully spell out "this.draw_flower()" when calling your functions. Or "this.animation_time" instead of just "animation_time" when accessing members.
- Sine and Cosine are in the Math namespace; you say Math.sin and Math.cos. These expect degrees

More Differences

- Your time measurements are given in milliseconds.
- We use a tiny matrix library for using 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.
    - All javascript arrays can call stack functions
    - Making an array 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' read from the last array element)

For Assignment 1

- One of the requirements will be to break up your code into functions.
- Use matrices as both in & 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; only return values can send the updated matrix back to the caller).