Discussion 1 Computer Graphics 174a

Ruiming Cao ruimingc@g.ucla.edu,
Arjun Lakshmipathy arjun.lakshmipathy@cs.ucla.edu,
Garett Ridge garett@cs.ucla.edu

Part I: Course Logistics

Forum, Textbook, Projects

Piazza Forum

- Join it on piazza.com now - ask if you need help
- All homework and coding questions will be discussed there
- Assignment materials will be announced on there, including your code template (the skeleton you'll fill in to do homework)
- The code template shows how you organize a graphics program and will be posted next week: A web document (~75 lines) and two javascript files (~500 lines each).
- Pay attention to Piazza next week, because that's where announcements will happen about setting up your code & environment

Which language will we use?

- Three common languages for graphics are C++, Java, and WebGL.
- All of these use OpenGL (API for talking to your graphics card)
- WebGL = JavaScript plus OpenGL calls

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. Instead, JavaScript has unusual stuff like closures, dynamic typing, lots of shorthand, prototypes, and anonymous functions.

Which language will we use?

- JavaScript handles a little bit easier than the C++ version of OpenGL.
    - Less to type / some saved steps
    - Modern web browsers have *excellent* code debuggers built right in - even better than Visual Studio / other C++ IDE’s
        - Can even give error feedback from the GPU
    - Most programming is moving over to the internet anyway
    - 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 code skeleton demonstrates how to organize a graphics program
  - The way to do that is the same across languages
  - Most lines you'll type for graphics commands even look the same in each language
- Making a scene can mean copying and pasting 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.
- This course updated to JavaScript (from C++) when the textbook did
- Only the newest (7th) edition of the book is in our language

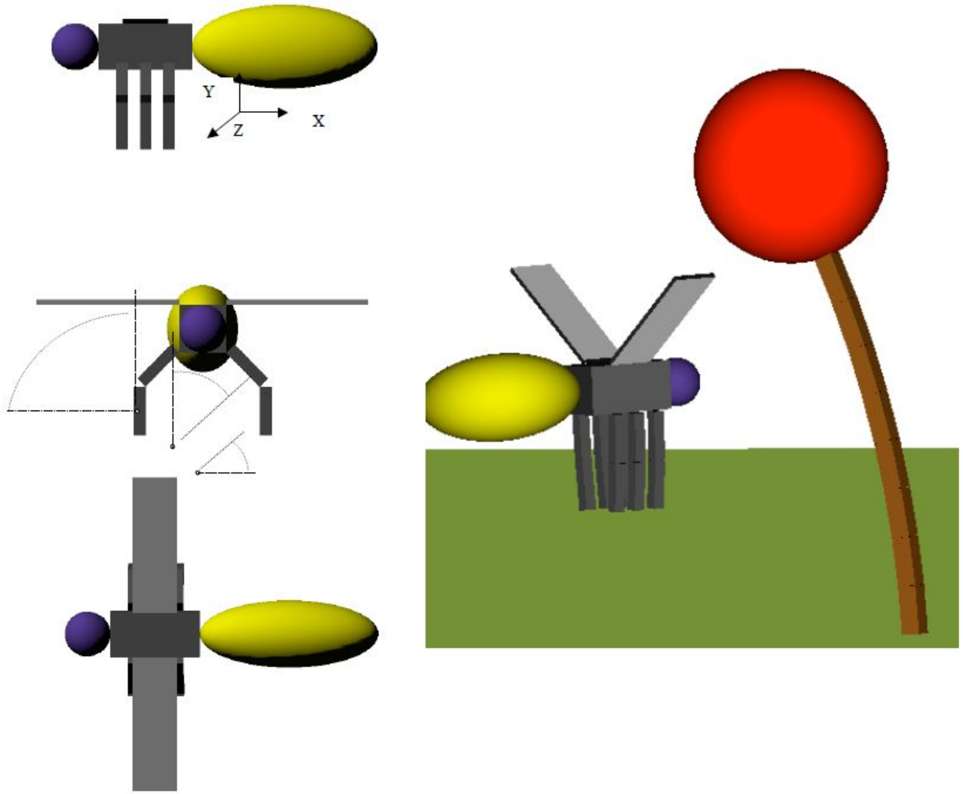
Part II: Project logistics

Probable assignments based on the past

Likely Assignment 1: Animate a scene we describe

- Mimic our animation as closely as you can
- Connect boxes and balls correctly
- Required:
    - Box 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

Example from a previous quarter



Assignment 2: Animate anything

- Make any animation or game as long as it includes certain 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-17s/

Assignment 2: Animate anything

http://web.cs.ucla.edu/~dt/courses/CS174A/animations/assignment2-best-17s/

Why do CS 174a projects have the same "look" to them?

- Students usually don't build very irregular shapes, or customize the shaders

Assignment 2: Animate anything

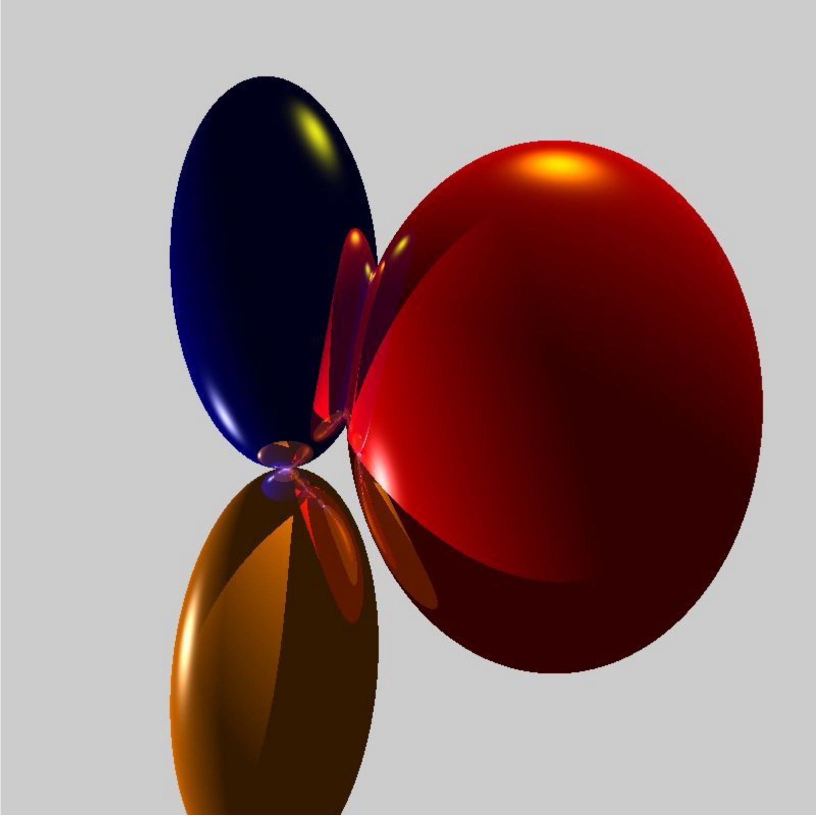
- The biggest limitation is resources, not technique
- Assignment 2 shows what can be done by:
  - One person,
  - in one quarter,
  - from scratch in plain JavaScript plus 500 lines of helper code
- Whereas modern films and games are done by:
  - A multitude of workers
    - (dozens of programmers, hundreds of artists to manually tweak things),
  - supercomputer warehouses,
  - a few years of development, &
  - pooling resources with other companies by writing software API hooks
- The end results are still close enough

Assignment 3: Trace Rays through our scenes

- We give you some text that fully encodes 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

Example:



Part III: Practical concepts - Making Shapes

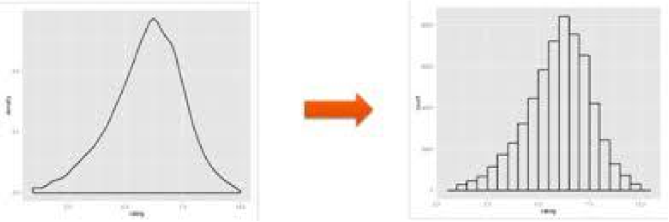
With math and code

We'll make shapes out of math.

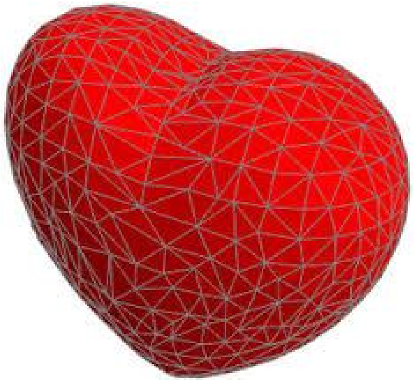
- Remember graphing shapes with a calculator? This will be the more advanced version of that.

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

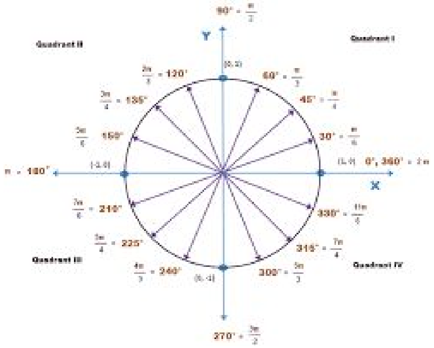
Discretization



- 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



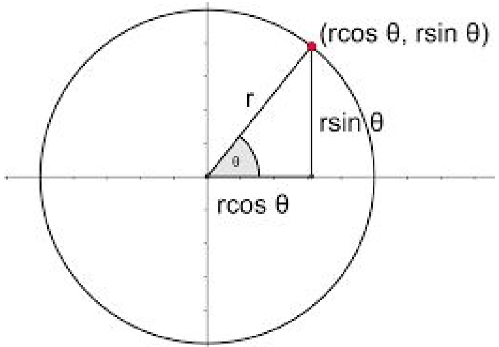
Let's list N points around a circle.



- 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

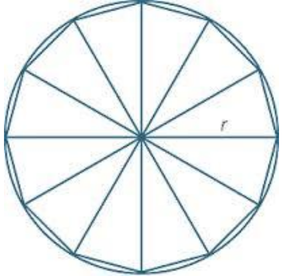
Let's list N points around a circle.

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



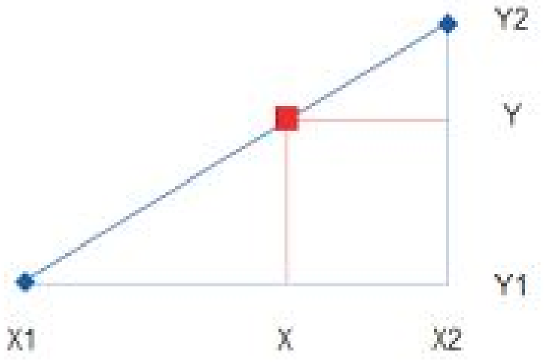
Using Θ as a variable input parameter, take N tiny steps from 0...2*PI.

Triangles



- We want to draw the whole 2D area, not just some points, so to discretize the shape with its approximate area, 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 triangles can fit 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:

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

  - It’s only an interpolation (and called “convex”) if 0<=a<=1
  - Otherwise it's an extrapolation
- You’ll be seeing that equation a lot
- 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. Now it’s animated.

Part IV: 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) happen one after another, and can be separated by ; or whitespace
- 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++

- As with C++, following your code line by line in a debugger continues to be critical for getting it to work
- Now you'll be using a built-in debugger in your web browser, like Chrome’s really good one:

Similarities with C++

- All the debugging features you're (hopefully) used to from C++ are there in Chrome
  - Like for running your program one line at a time (pausing in between, to hover your mouse over each variable or expression and observe as they change)
- It has a lot of features that are better than what you're used to from C++
  - A console for typing arbitrary statements and seeing the result
  - If a line of code does multiple things, you can put a breakpoint on any of the parts
  - No way to drag around the instruction pointer though
    - (one thing C++ IDE's do better)

Main Difference from C++: Variable Declaration.

- JavaScript is a dynamically typed language
- It figures out type for you implicitly at run time
- Use “let” or “const” as the types for every variable

  Examples:
  - A loop counter: "let counter = 0" instead of "int counter = 0"
  - A custom matrix: "let matrix = Mat4.identity()

When writing a function, leave off the "let" before each argument, though.

Ints and floats are treated the same in JS -- no truncation errors.

More JavaScript Differences

- Don't have to warn javascript about return types when declaring functions. Just say “function” to declare a function.
- Return any expression you want when it’s time to.
- Inside of Classes, for methods you even leave off the word “function”.
- You can use shorthand for declaring functions when you want:

```
function (x,y) { return x + y; }
```

Can be:

```
(x,y) => x+y
```

```
function x { return x + 9 }
```

Can be:

```
x => x+9
```

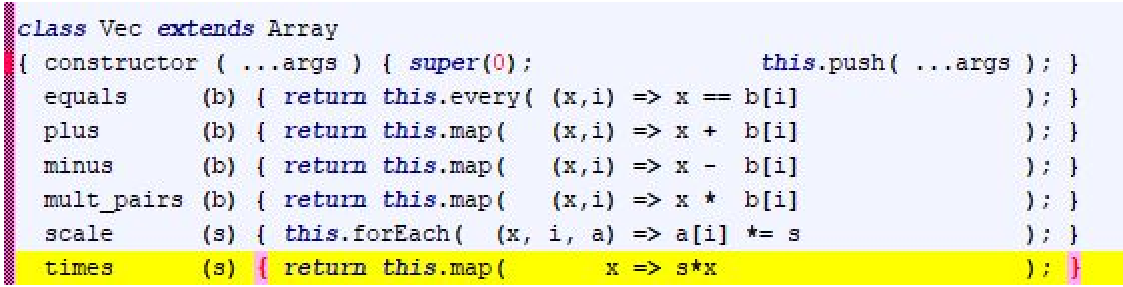
- Unlike C++, all semicolons are optional in JavaScript.

More JavaScript Differences

- Functions can be declared right in the middle of whatever you’re typing, so it’s really convenient in JavaScript to pass functions around as callbacks to other functions
    - Here’s a function call from your template that makes a click-able button. As part of this call, we define another function (a callback) for what the button should do.

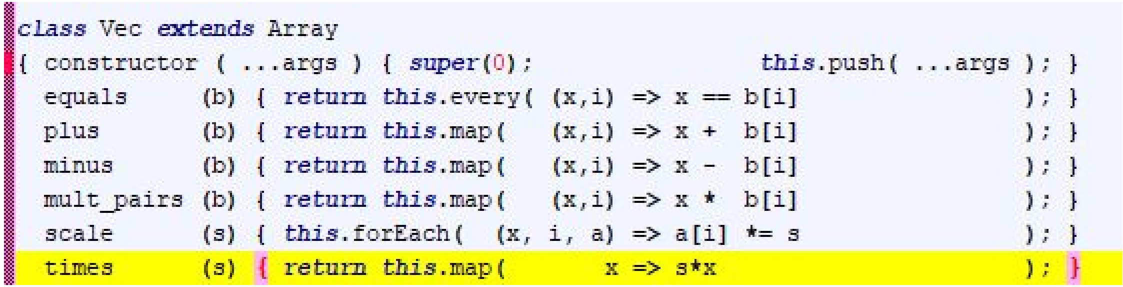
```
this.key_triggered_button( "Go to world origin", "r",
      function() { this.target().set_identity( 4,4 ) },
      "orange" );
```

The first few lines of the 174a template:



```
class Vec extends Array
{ constructor ( ...args ) { super(0);                 this.push( ...args ); }
  equals     (b) { return this.every( (x,i) => x == b[i]                 ); }
  plus       (b) { return this.map(   (x,i) => x +  b[i]                 ); }
  minus      (b) { return this.map(   (x,i) => x -  b[i]                 ); }
  mult_pairs (b) { return this.map(   (x,i) => x *  b[i]                 ); }
  scale      (s) { this.forEach(  (x, i, a) => a[i] *= s                 ); }
  times      (s) { return this.map(       x => s*x                       ); }
```

- The template might seem mysterious to you for a while.
- That’s OK; remember that your projects largely amount to just changing numbers and copy/pasting lines of code to move shapes.



```
class Vec extends Array
{ constructor ( ...args ) { super(0);                 this.push( ...args ); }
  equals     (b) { return this.every( (x,i) => x == b[i]                 ); }
  plus       (b) { return this.map(   (x,i) => x +  b[i]                 ); }
  minus      (b) { return this.map(   (x,i) => x -  b[i]                 ); }
  mult_pairs (b) { return this.map(   (x,i) => x *  b[i]                 ); }
  scale      (s) { this.forEach(   (x, i, a) => a[i] *= s                ); }
  times      (s) { return this.map(       x => s*x                       ); }
```

It’s new, unfamiliar syntax, but try to spot the following:

- Object-Oriented concepts from C++, like “class” “constructor” and “super”
- Extending a class (the built-in Array class) to use its built-in functions
  - “push”, “every”, “map”, “foreach”
- Although it’s not beginner JavaScript, it’s still possible to at least scan the list of methods available to a “Vec”: (“equals” “plus”, “minus”, etc.) and guess what they do with the Array
- Lots of Callbacks
- The “arrow” shorthand shown earlier
- Javascript shorthand “...” for expanding arrays

More JavaScript Differences

- Unlike C++, some things are buried in the Math namespace. For trig you say Math.sin, Math.cos, etc. These expect radians.
  - More: Math.min, Math.pow, Math.max, Math.PI, Math.log, Math.exp
- Your time measurements are given in milliseconds.
- 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 class 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.

More JavaScript Differences

- For math, we will give you three classes that define a Vector (Vec), Matrix (Mat), and a graphics-specific 4x4 matrix (Mat4)
- Javascript arrays:
  - Declare one yourself (a literal) like this [a, b, c]. They behave like C++ std::vector<>’s, except they can combine multiple types of values.
- Javascript has a data type called an object (not the class called Object, also found in JavaScript).
  - Declare one yourself (a literal) like this { “a”: 1, “b”: 2, “c”: 3 }. They behave like C++ std::map<string, >’s, except they can combine multiple types of values.

Stacks in Graphics

- Sometimes your code deals with values (especially math matrices) that change a lot of times and even outlive function calls. Because of that you will might want a "stack" to maintain the history of that value, for you to unwind as you please.

Stacks in Graphics

- All javascript arrays have stack functions
  - Making any array data member will look like: **this.history_stack = [ ];**
  - Afterwards, **this.history_stack.push(some_matrix)** saves your current matrix
  - **this.history_stack.pop()** returns the most recent one (and takes it off the stack - if you don't want that, just read from the last array element)
- Old GPUs used to manage huge "stacks" for you (for history of your variables) because it is so common to design a scene hierarchically (like a tree).
- Modern GPUs have "programmable shaders" you make yourself instead of those stacks. We'll learn about those.