Introduction

CS 385 - Class 1
25 January 2022

Welcome

Welcome to CS 385

- *Topics in Computer Science*
  - Our topic for this semester is *computer graphics* 😎
  - The field of *computer graphics* has potentially many definitions

Welcome to CS 385

- *Topics in Computer Science*
  - Our topic for this semester is *computer graphics* 😎
  - The field of *computer graphics* has potentially many definitions

Application Domains

Graphic Design

Visualization

Gaming

Art

Visual Effects

Simulation

Welcome to CS 385

- *Topics in Computer Science*
  - Our topic for this semester is *computer graphics* 😎
  - The field of *computer graphics* has potentially many definitions
- In this class, we'll focus on fundamental techniques:
  - Interactive, GPU-accelerated, three-dimensional applications
  - We'll also touch on some other domains, notably ray tracing, and image processing

Application Domains

Graphic Design

Visualization

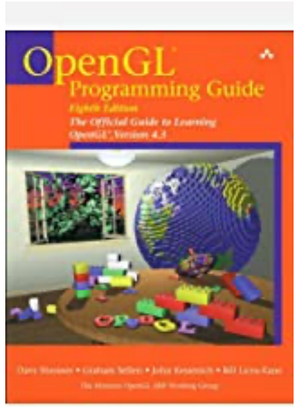
Gaming

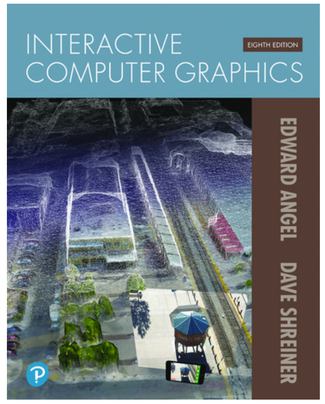
Art

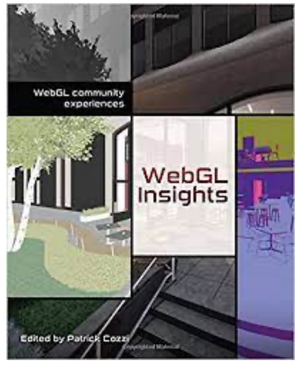
Visual Effects

Simulation

A little about me …







Some considerations …

- User interfaces and drawing graphics is very computing-platform dependent
  - there are some some cross-platform solutions, like Qt
- Requires headers (imports) and system-dependent libraries
- Working on native development isn't practical for a 15 week course

Operating System	GUI toolkits	Graphics APIs
Microsoft Windows	Win32, MFC	D3D11, D3D12, OpenGL, Vulkan
Linux	GTK, KDE, Qt	OpenGL, Vulkan
macOS	Cocoa	Metal
iOS	UIKit	Metal, OpenGL ES
Android	Android Framework	OpenGL ES, Vulkan

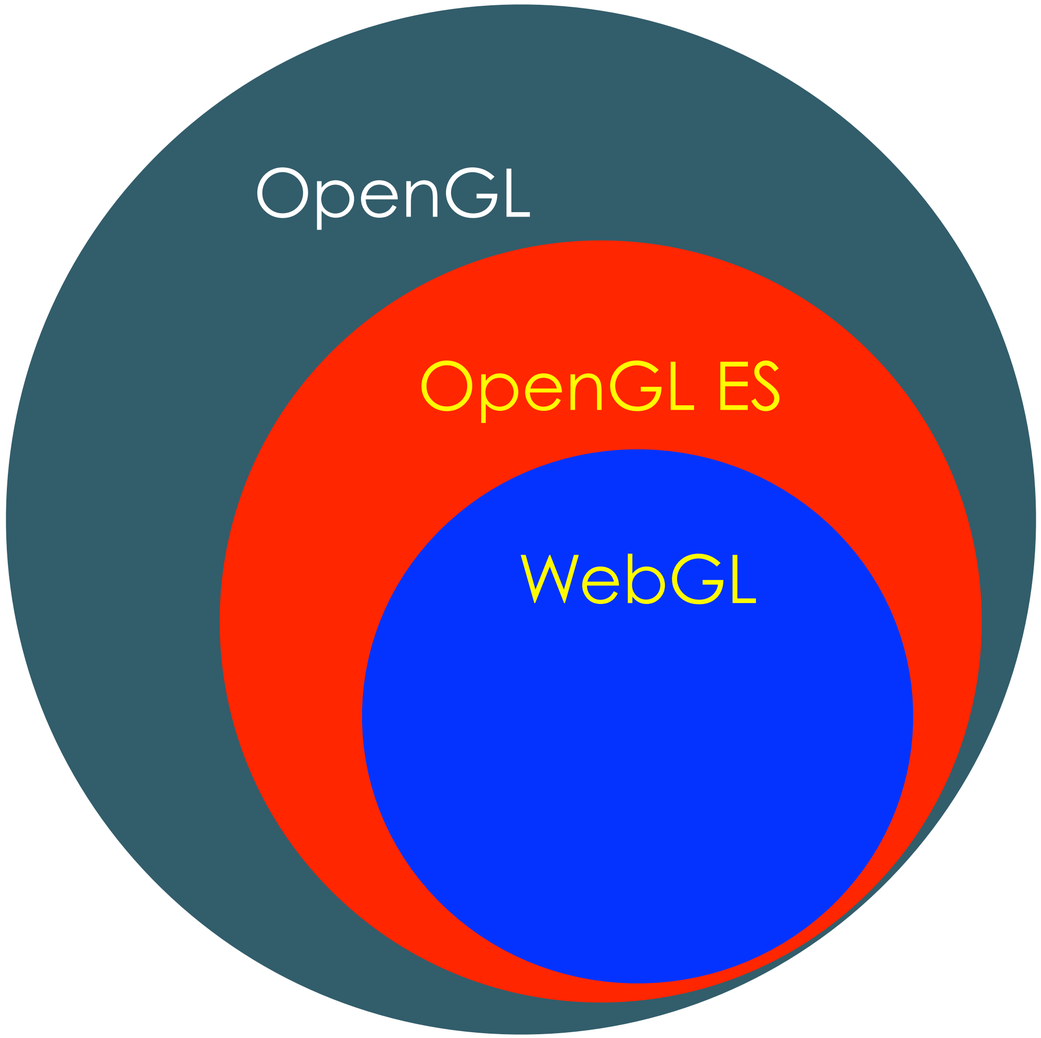
A similar, but different approach

- We're going to use web technologies
- Much simpler for developing applications
  - an application is a web page
  - GUI elements (e.g., buttons) are native HTML elements
  - input is processed through JavaScript
  - graphics are drawn using WebGL
- All of these are supported in most modern web browsers

Task	Technology
Application Window	HTML5
GUI	HTML5 & JavaScript
Graphics	WebGL & JavaScript

Are you messing with us, Dave?

- In terms of graphics, what we'll learn works very similarly between OpenGL, OpenGL ES, and WebGL.
  - the function names are very similar between each of these APIs
  - WebGL makes some things much simpler
- What about "modern graphics APIs" like Vulkan, Metal, or DX12?



Course Mechanics

Course Information

- Syllabus contains all of this information
- Assignments and exams will be distributed and collected through Canvas
- Collaboration & academic honesty policy

Grading Criteria	% of Grade
Assignments	20%
Midterm Exam	20%
Final Exam	20%
Final Project	30%
Participation	10%
Total	100%

Assignments, Final Project, and Participation

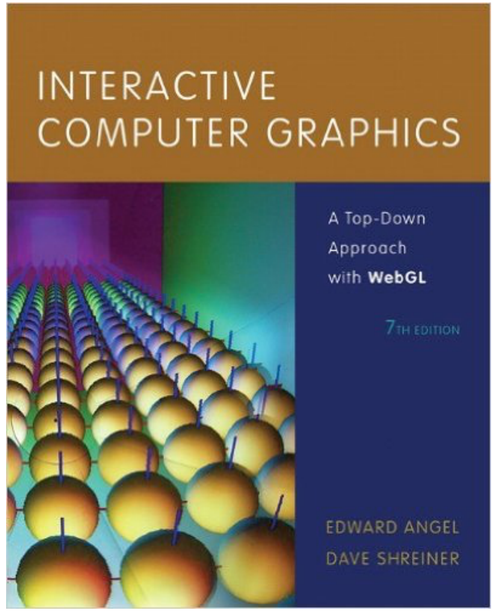
- Coding assignments
  - should be completable in a week's time
- Final Project
  - submit a project plan (including schedule)
  - project presentation
- Class participation

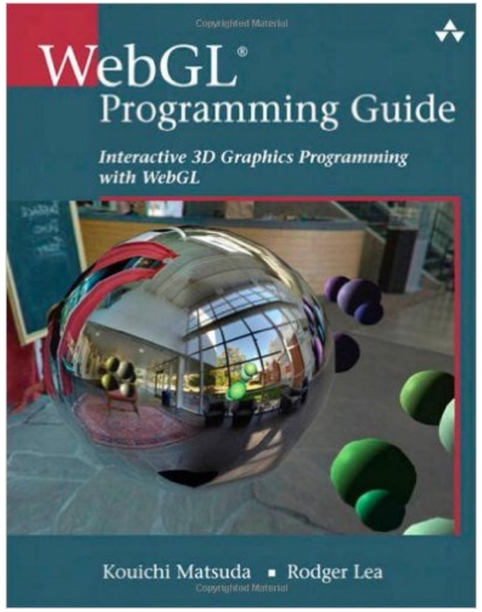
Exams

- Questions derived from course materials
- Will include some code-based questions
  - either writing or analyzing code

Reference Resources

- Interactive computer Graphics — A Top-Down Approach with WebGL
- WebGL Programming Guide





Getting Started

Our First Application

- We need an HTML5 page
- We really only need two sections:
  - *head* where we'll include our JavaScript files, and other
  - *body* where we'll have our "windows" and UI elements

```
<!DOCTYPE html>
<html>
<head>
...
</head>

<body>
...
</body>
</html>
```

Opening a Window

- WebGL uses an HTML5 canvas element as its window
- As with many HTML elements, you can specify attributes
  - id
  - width, height
    - default sizes are in pixels
    - you can also specify in percentage of the enclosing element
  - styles
  - scripts
- The *id* will be particularly important for us

```
<!DOCTYPE html>
<html>
<head>
...
</head>

<body>
<canvas id=“webgl-canvas"
    width="512" height=“512">
  </canvas>
</body>
</html>
```

Clearing the Canvas' Background

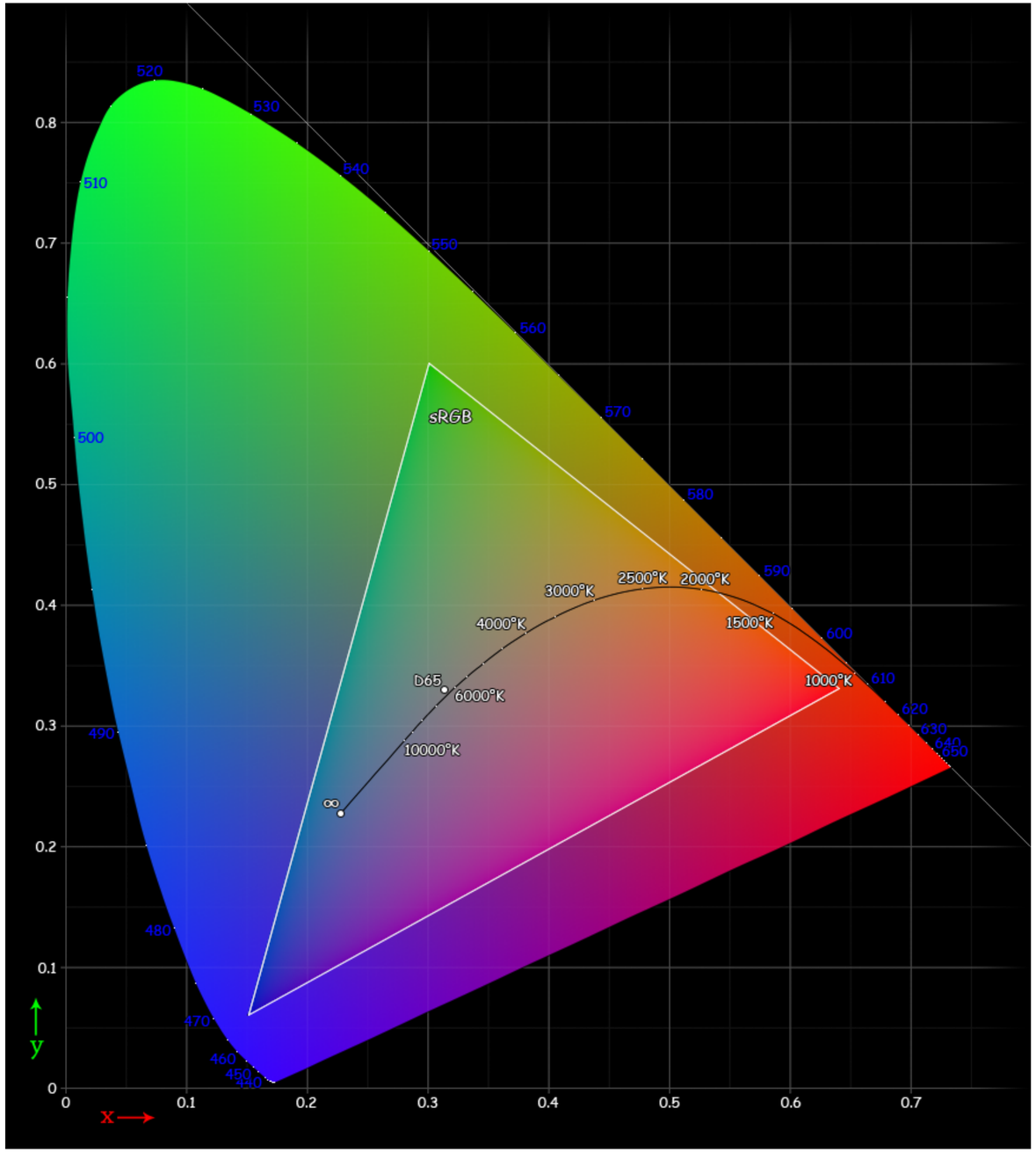
- We can specify an *attribute* to set the background color
- Note the format of the color

```
<!DOCTYPE html>
<html>
<head>
...
</head>

<body>
<canvas id="webgl-canvas"
    width="512" height="512"
    style="background-color: #0000FF">
  </canvas>
</body>
</html>
```

Color & Computer Graphics

- Computer displays use the *RGB color-model*
  - Colors on the screen are a combination of three colors: red, green, and blue
  - The set of representable colors is called a *gamut*
- Most graphics APIs conceptually use values in the range [0.0, 1.0]
  - although values may be mapped into the representable range of a variable's data type
- High-dynamic range (HDR) images can use values above 1.0



Structuring our Web Applications

- JavaScript code can either be inlined, or imported from a file
- We'll most often use a separate JavaScript file to hold our application-specific code

```
<!DOCTYPE html>
<html>
<head>
<script type="text/JavaScript"
    src="clear.js">
  </script>
</head>

<body>
<canvas id="webgl-canvas" width="512"
height="512"></canvas>
</body>
</html>
```

Structure of all of our WebGL Applications

- We'll essentially have two functions in our applications
  - `init()` — WebGL setup, and calls to one-time initialization operations
  - `render()` — our rendering loop
    - it does the drawing
    - called for each frame of the application's operation
    - uses the current state of our application to position, *shade*, and animate the objects in our scene
- Think of `window.onload` as a C++ program's `main()` function
  - it's called when the window's loaded and things are ready to go

```
function init() {
...
}

function render() {
 ...
}

window.onload = init;
```

Graphics Contexts

- A *graphics context* is:
  - an entity that stores API-specific graphics data, and
  - may (depending on the programming language) manage the function interface

```
gl.clearColor( 1.0, 0.0, 1.0, 1.0 );
```

Creating a WebGL Context

- Recall that the canvas's *id* attribute was going to be important
  - we use it to find our window from our application code
- Once we have the canvas, we can get *contexts* from it, including one for WebGL
  - `webgl2` is the tag for a WebGL 2 context.

```
function init() {
var canvas =
document.getElementById("webgl-canvas");

gl = canvas.getContext("webgl2");
if (!gl) { … ; return; }
...
}
window.onload = init;
```

Clearing the Window

- Use the `clearColor()` method of our WebGL object to specify the *clear* (i.e., background) color
- `clear()` will clear all of the buffers of the window
  - here we have the *color buffer*
  - we'll see other buffers later

```
function init() {
var canvas =
document.getElementById("webgl-canvas");

gl = WebGLUtils.setupWebGL(canvas);
 if (!gl) { return; }

gl.clearColor(1.0, 0.0, 1.0, 1.0);
gl.clear(gl.COLOR_BUFFER_BIT);
}

 window.onload = init;
```

"Running" the Application

- The `init()` function is executed when the window completes loading in the browser
- Populate `render()` and other functions to control our app's operation
- Use JavaScript *event listener* callback functions
  - mouse, keyboard, or touch events