Hello, Sierpinski!

Today

- How in the what now is the gasket being drawn to the screen?

Immediate mode vs. retained mode

- Two major types of graphics APIs
- Immediate mode: draws things to the screen as soon as told to; doesn't remember things frame-to-frame
- Retained mode: remembers information from frame-to-frame; data stored in memory for quick access

Immediate mode example

- Drawing the same thing multiple frames in a row requires the data to be sent multiple times
- Examples of immediate mode graphics APIs:
  - Processing
  - OpenGL < 2.0
  - "Turtle" graphics
  - Canvas
  - QBasic
- Let's look at a Processing example...

Retained mode APIs

- In retained mode, we upload data to the graphics API to use in the rendering and then later tell it to be drawn
- With these APIs, redrawing something multiple frames in a row doesn't require data to be re-sent
- Examples:
  - Unity
  - Unreal Engine
  - OpenSceneGraph

WebGL

- WebGL is somewhere in the middle
- We send some data to be stored in a buffer on the graphics card
- We issue commands telling the GPU to draw something to the screen, using data from a specific buffer
- WebGL is also a **state machine**, in that it has some information that is "currently set"
- Let's look at Processing as an example...

WebGL example

- Let's see how this all comes together...

Working with coordinates in code

- In Javascript, we will be using the vec2/vec3/vec4 functions to create vectors representing coordinates in 2, 3, or 4 dimensions
- These are not built-in functions, but are instead provided by the MV library
- Let's take a look...

Math with vectors

- We can multiply a vector by a scalar
- We can multiply a vector by a vector (*)
- We can multiply a matrix by a vector

Shaders

- Shaders run on the GPU and control calculation of vertex positions and the calculation of colors for each pixel
- A **shader program** consists of a **vertex shader** and a **fragment shader**
- Vertex shader: Given per-vertex information, determine where the vertex should be placed in normalized screen coordinates
- Fragment shader: Given information of a particular pixel in a primitive, determine what color to draw

Creating a shader

- let vs = gl.createShader(gl.VERTEX_SHADER)
- gl.shaderSource(vs, "shader code here")
- gl.compileShader(vs)
- // same for fs and FRAGMENT_SHADER
- let shader = gl.createProgram()
- gl.attachShader(shader, vs)
- gl.attachShader(shader, fs)
- gl.linkProgram()

Buffer functions

- Buffers are things that we can create to store data on the graphics card
- Information includes things that can change **per-vertex** or information on what order to use vertices
- This includes:
  - Position
  - Color
  - Normal vector
  - Texture coordinates

Buffer functions

- gl.createBuffer()
- gl.bindBuffer(buffer, target)
- gl.bufferData(target, data, usage)
- gl.isBuffer(buffer)
- gl.deleteBuffer(buffer)

gl.createBuffer()

- Create a new buffer that can store data and return a WebGLBuffer that can be used to refer to it
- The only thing you're allowed to do with the return value is pass it to other WebGL functions

gl.bindBuffer(buffer, target)

- Sets the active buffer of target to be the passed-in buffer
- Target can be either:
  - gl.ARRAY_BUFFER: Used for setting data that changes per-vertex
  - gl.ELEMENT_ARRAY_BUFFER: Used for setting data that determines the order to connect vertices

gl.bufferData(target, data, usage)

- target: Whether to store data in gl.ARRAY_BUFFER or gl.ELEMENT_ARRAY_BUFFER
- data: A typed array containing the data to be stored
- usage: one of either
  - gl.STATIC_DRAW: data used often, doesn't change often
  - gl.DYNAMIC_DRAW: data used often, changes often
  - gl.STREAM_DRAW: data not used often

gl.isBuffer(buffer)

- Returns true if the parameter is a valid WebGLBuffer, false otherwise

gl.deleteBuffer(buffer)

- Tells WebGL to delete the indicated buffer and mark it as no longer usable

Setting attribute data

- Attributes are data sent as input to the vertex shader that can change per-vertex
- This include position, color, normal, texture coordinate, etc.
- We are able to store multiple attributes in a single buffer, or instead store attributes in separate buffers

gl.getAttribLocation(program, variable)

- Returns the index of the attribute named variable in the specified program

gl.vertexAttribPointer(...)

- vertexAttribPointer(index, size, type, normalized, stride, offset)
- Sets the attribute specified by *index* to take data from the currently bound ARRAY_BUFFER
- Let's look at MDN for what the parameters mean...

gl.vertexAttribPointer(...)

- vertexAttribPointer(index, size, type, normalized, stride, offset)
- Sets the attribute specified by *index* to take data from the currently bound ARRAY_BUFFER
- Let's look at MDN for what the parameters mean...
- gl.vertexAttribPointer(*variableName*, #, gl.FLOAT, false, 0, 0);

gl.enableVertexAttribArray(index)

- Enable the specified attribute

gl.drawArrays(type, first, count)

- type: What type of primitive to draw
  - gl.POINTS, gl.LINE_STIRP, gl.LINE_LOOP, gl.LINES, gl.TRIANGLE_STIRP, gl.TRIANGLE_FAN, gl.TRIANGLES
- first: First index to start drawing at
- count: How many vertices to draw