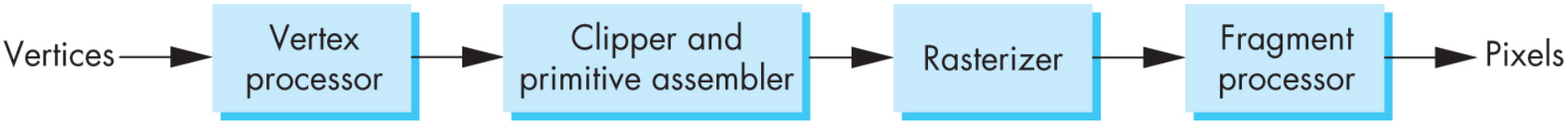
2D graphics with WebGL

Some material contained here is adapted from the book's slides.

September 6, 2016

Graphics Pipeline



Shaders

So far we've seen trivial vertex and fragment shaders:

```
<script id="vertex-shader" type="x-shader/x-vertex">
attribute vec4 vPosition;
void main()
{
  gl_PointSize = 1.0;
  gl_Position = vPosition;
}
</script>


<script id="fragment-shader" type="x-shader/x-fragment">
precision mediump float;
void main()
{
  gl_FragColor = vec4( 1.0, 0.0, 0.0, 1.0 );
}
</script>
```

This vertex shader simply passes the vertex position attribute (from the application) through the pipeline. The fragment shader hardcodes every fragment red.

Attributes

Vertex Attributes

The only mandatory attribute for a vertex is its position:

```
1 <script id="vertex-shader" type="x-shader/x-vertex">
2 attribute vec4 vPosition;
3 void main()
4 {
5   gl_PointSize = 1.0;
6   gl_Position = vPosition;
7 }
8 </script>
```

We will add others:

- Color
- Normal vector (the vector perpendicular to the plane to which the vertex belongs to)
- Texture coordinate

Attributes

Vertex Attributes

Let's add color:

```
 <script id="vertex-shader" type="x-shader/x-vertex">
attribute vec4 vPosition;
attribute vec4 vColor; // in: the attribute is what the application sends to the shader
varying vec4 fColor; // out: varying qualifier interpolates the color between the vertices
void main()
{
    gl_PointSize = 1.0;
    gl_Position = vPosition; // pass-through
    fColor = vColor; // pass-through
}
</script>

<script id="fragment-shader" type="x-shader/x-fragment">
precision mediump float;

varying vec4 fColor;

void main()
{
    gl_FragColor = fColor; // pass-through
}
</script>
```

Attributes

From Application to GPU

The process of sending information to the GPU consists of:

- Creating a buffer
- Binding it (sets the buffer currently worked on). Remember WebGL is a state machine.
- Populating it
- Associating the buffer with shader variable and enabling it

```
1  var vertices = [new vec2(0, 1), new vec2(1, 1), new vec2(1, -1), new vec2(-1, 1), new vec2(-1, -1), new
       vec2(0, -1)];
2  // Create buffer
3  var bufferId = gl.createBuffer();
4  // Bind it
5  gl.bindBuffer( gl.ARRAY_BUFFER, bufferId );
6  // Populate it with data
7  gl.bufferData( gl.ARRAY_BUFFER, flatten(vertices), gl.STATIC_DRAW );
8  // Associate our shader variables with our data buffer
9  var vPosition = gl.getAttribLocation( program, "vPosition" );
10 gl.vertexAttribPointer( vPosition, 2, gl.FLOAT, false, 0, 0 );
11 gl.enableVertexAttribArray( vPosition );
```

Color is just another attribute, we follow the same process to send it to the GPU.

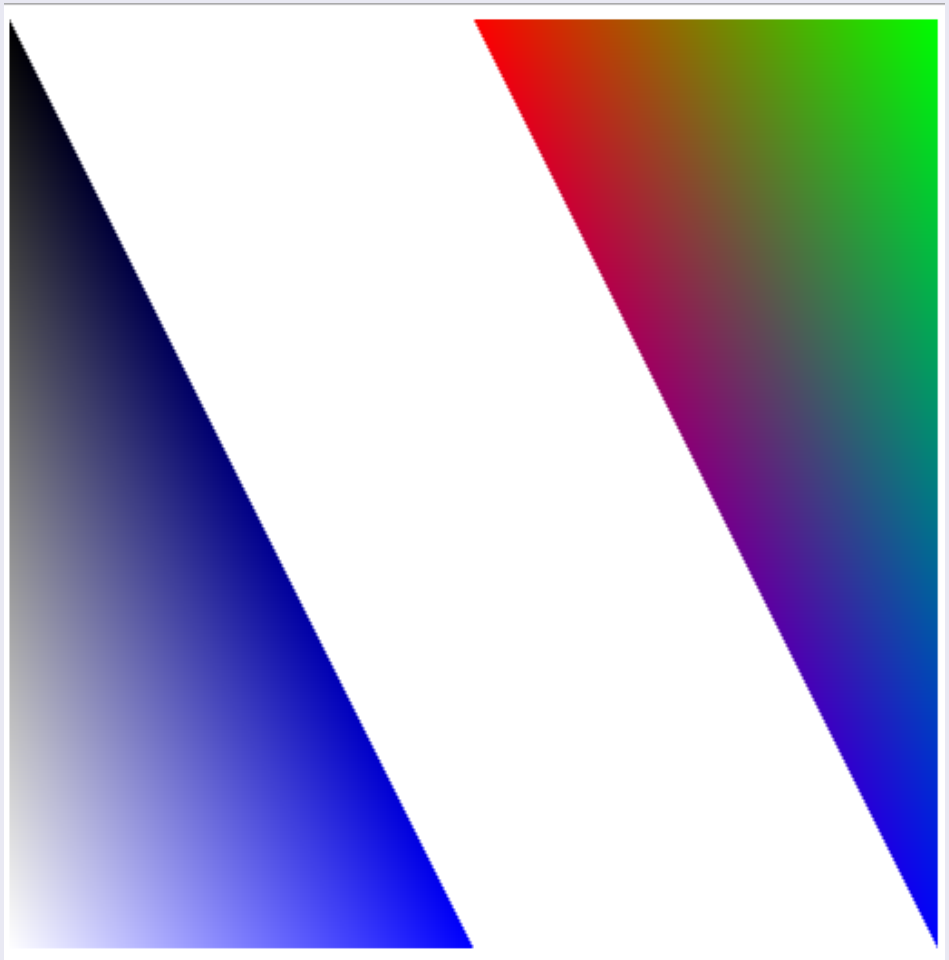
Attributes

Color Attribute to GPU

```
1  var colors = [new vec4(1, 0, 0, 1), new vec4(0, 1, 0, 1), new vec4(0, 0, 1, 1), new vec4(0, 0, 0, 1), new
       vec4(1, 1, 1, 1), new vec4(0, 0, 1, 1)];
2  // Create buffer that will hold color data
3  var colorBufferId = gl.createBuffer();
4  gl.bindBuffer( gl.ARRAY_BUFFER, colorBufferId );
5  // Populate it
6  gl.bufferData( gl.ARRAY_BUFFER, flatten(colors), gl.STATIC_DRAW );
7  // Associate our shader variables with our data buffer
8  var vColor = gl.getAttribLocation( program, "vColor" );
9  gl.vertexAttribPointer( vColor, 4, gl.FLOAT, false, 0, 0 );
10 gl.enableVertexAttribArray( vColor );
```

Attributes

Example: Two colored triangles



Notice how colors are interpolated. This example is available to download on the course web page.

Attributes

Vertex Attributes

Texture

- Part of an image mapped onto a triangle. We need to:
  - Load the image from the a file on the web server.
  - Send it to the GPU.
  - Instruct the GPU which part of the image gets mapped onto the triangle.

Attributes

Loading the texture

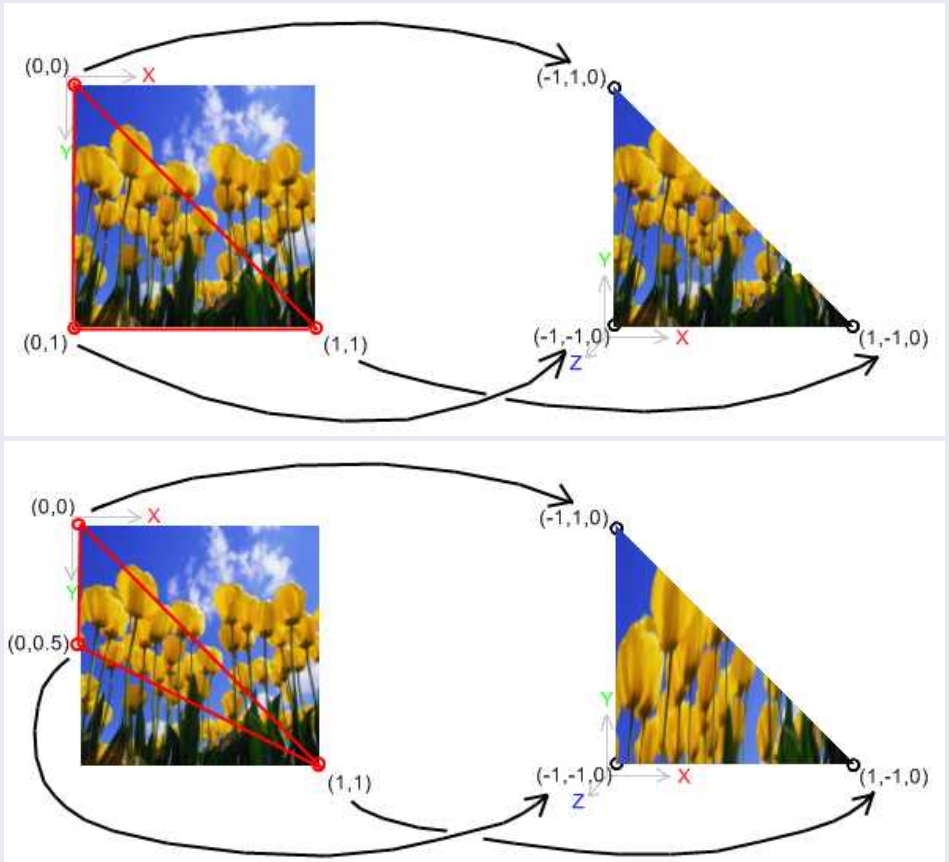
The texture is loaded asynchronously, so mechanics are more complicated.

```
1  function initTexture() {
2    texture = gl.createTexture();
3    gl.bindTexture( gl.TEXTURE_2D, texture);
4    gl.pixelStorei(gl.UNPACK_FLIP_Y_WEBGL, true);
5    texture.image = new Image();
6    texture.image.src = "trip.jpg";
7    texture.image.onload = function() {
8        gl.bindTexture(gl.TEXTURE_2D, texture); // set the texture we're currently working on
9        gl.pixelStorei(gl.UNPACK_FLIP_Y_WEBGL, true); // turn upside down
10       gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, gl.RGBA, gl.UNSIGNED_BYTE, this); // upload to GPU
11       gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER, gl.LINEAR);
12       gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.LINEAR);
13   }
14 }
```

- Textures must be square, sized power of two!

Texture Coordinates

Triangles are textured from a unit area.



Images retrieved from: http://blogs.msdn.com/b/danlehen/archive/2005/11/06/489627.aspx

Attributes

Texture Coordinate Attributes[a]

[a] See textured triangles example on course web page for full source code.

```
1  //for reference, the vertices are as follows
2  var vertices = [new vec2(-1, 1), new vec2(0, 1), new vec2(-1, -1), new vec2(0, 1), new vec2(1, 1), new vec2
       (1, -1), new vec2(0, 0.8), new vec2(-0.9, -1), new vec2(0.9, -1)];
3
4
5  var texCoords = [new vec2(0, 0), new vec2(0.5, 0), new vec2(0, 1), new vec2(0.5, 0), new vec2(1, 0), new
       vec2(1, 1), new vec2(0.5, 0), new vec2(0, 1), new vec2(1, 1)];
6
7  var texCoordBufferId = gl.createBuffer();
8  gl.bindBuffer( gl.ARRAY_BUFFER, texCoordBufferId );
9  gl.bufferData( gl.ARRAY_BUFFER, flatten(texCoords), gl.STATIC_DRAW );
10 // Associate our shader variables with our data buffer
11 var vTextureCoord = gl.getAttribLocation( program, "vTextureCoord" );
12 gl.vertexAttribPointer( vTextureCoord, 2, gl.FLOAT, false, 0, 0 );
13 gl.enableVertexAttribArray( vTextureCoord );
```

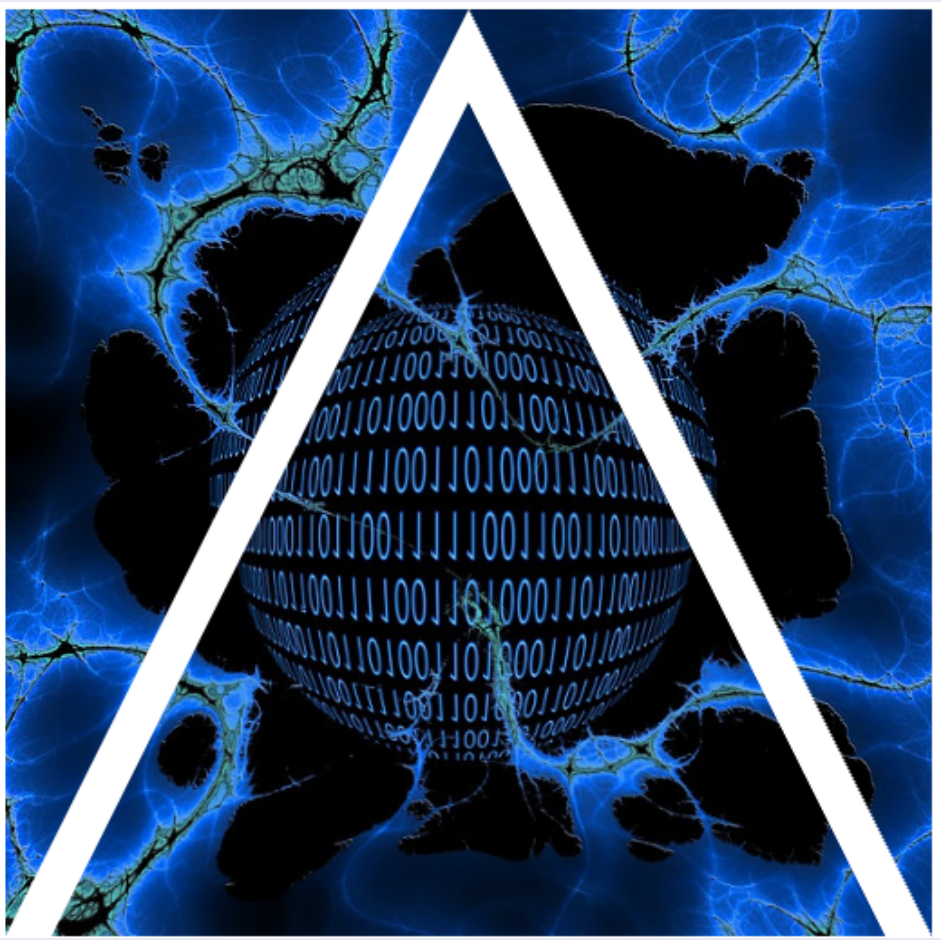
Attributes

Shaders

```
<script id="vertex-shader" type="x-shader/x-vertex">
attribute vec4 vPosition;
attribute  vec2 vTextureCoord; // in
varying vec2 fTextureCoord; // out
void main()
{
    gl_PointSize = 1.0;
    gl_Position = vPosition; // pass-through
    fTextureCoord = vTextureCoord; // pass-through

}
</script>

<script id="fragment-shader" type="x-shader/x-fragment">
precision mediump float;

varying  vec2 fTextureCoord;
uniform sampler2D texture;


void main()
{
    gl_FragColor = texture2D(texture, fTextureCoord); // sample from the image
}
</script>
```

Attributes

Example: Three textured triangles



GLSL

Data Types

- C types: int, float, bool
- Vectors:
  - float types: vec2, vec3, vec4
  - int types: ivec, and boolean: bvec
- Matrices: mat2, mat3, mat4. They are stored by columns and referenced m[row][column].
- C++ style constructors and casting:
  - vec3 a = vec3(1.0, 2.0, 3.0);
  - vec2 b = vec2(a);

GLSL

No Pointers

- There are no pointers in GLSL.
- Can use C structs that are populated from the application.
- Since matrices and vectors are basic types, they can be passed into and out-of GLSL functions. E.g.: mat3 foobar(mat3 a).
- Variables are passed by copying (value)

GLSL

Qualifiers

- GLSL has many of the same qualifiers as C++, such as const
- Need others due to the nature of execution model
- Variables can change:
  - Once per primitive
  - Once per vertex
  - Once per fragment
  - At any time in the application
- Vertex attributes are interpolated by the rasterizer into fragment attributes.

Attribute Qualifier

- Attribute-qualified variables can change at most once per vertex
- There are a few built-in variables such as gl_Position, but most have been deprecated
- Most are user-defined, in the application
  - attribute float temperature;
  - attribute vec3 velocity;

GLSL

Uniform qualified

- Variables are constant for an entire primitive
- Can be changed in application and sent to shaders
- Most are user-defined, in the application
- Cannot be changed in shader
- Used to pass information to shader such as time, a bounding box or transformation matrices

GLSL

Varying qualified

- Variables that are passed from vertex shader to fragment shader
- Automatically interpolated by the rasterizer
- With WebGL, GLSL uses the varying qualifier in both shaders varying vec4 color;
- More recent versions of WebGL use out in vertex shader and in in the fragment shader
  - out vec4 color; //vertex shader
  - in vec4 color; // fragment shader

Variable Naming

- Attributes passed to vertex shader have names beginning with v (v Position, vColor) in both the application and the shader. Note that these are different entities with the same name.
- Variable variables **must have the same name in both shaders**
- Uniform variables can have the same name in both shaders, not required.

GLSL

Selection

- Can refer to array elements using [] or selection (.) operator with:
  - x, y, z, w
  - r, g, b, a
  - s, t, p, q
- a[2], a.b, a.z, a.p are the same

GLSL

Swizzling

- Allows us to manipulate components, for example:
  vec4 a, b;
  a.yz = vec2(1.0, 2.0);
  b = a.yxzw;
- More details later in the course