Introduction to Computer Graphics with WebGL

Ed Angel
Professor Emeritus of Computer Science
Founding Director, Arts, Research, Technology and Science Laboratory
University of New Mexico

Angel and Shreiner: Interactive Computer Graphics 7E © Addison-Wesley 2015
Programming with WebGL
Part 3: Shaders

Ed Angel
Professor of Emeritus of Computer Science
University of New Mexico
Data Types

- **C types:** int, float, bool
- **Vectors:**
  - float vec2, vec3, vec4
  - Also int (ivec) and boolean (bvec)
- **Matrices:** mat2, mat3, mat4
  - Stored by columns
  - Standard referencing m[row][column]
- **C++ style constructors**
  - vec3 a = vec3(1.0, 2.0, 3.0)
  - vec2 b = vec2(a)
No Pointers

• There are no pointers in GLSL
• We can use C structs which can be copied back from functions
• Because matrices and vectors are basic types they can be passed into and output from GLSL functions, e.g.

    mat3 func(mat3 a)

• variables passed by copying
Qualifiers

• GLSL has many of the same qualifiers such as `const` as C/C++

• Need others due to the nature of the 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
• User defined (in application program)
  - attribute float temperature
  - attribute vec3 velocity
  - recent versions of GLSL use in and out qualifiers to get to and from shaders
Uniform Qualified

• Variables that are constant for an entire primitive
• Can be changed in application and sent to shaders
• Cannot be changed in shader
• Used to pass information to shader such as the time or a bounding box of a primitive or transformation matrices
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
  ```glsl```
  ```
  varying vec4 color;
  ```
  ```
  • More recent versions of WebGL use `out` in vertex shader and `in` in the fragment shader
  ```glsl```
  ```
  out vec4 color;  // vertex shader
  in vec4 color;   // fragment shader
  ```
  ```
Our Naming Convention

• Attributes passed to vertex shader have names beginning with v (vPosition, vColor) in both the application and the shader
  - Note that these are different entities with the same name
• Fragment variables begin with f (fColor) in both shaders
  - Must have same name
• Uniform variables are unadorned and can have the same name in application and shaders
Example: Vertex Shader

```glsl
attribute vec4 vPosition;
attribute vec4 vColor;
varying vec4 fColor;
void main()
{
    gl_Position = vPosition;
    fColor = vColor;
}
```
precision mediump float;

varying vec4 fColor;

void main()
{
    gl_FragColor = fColor;
}

var cBuffer = gl.createBuffer();
gl.bindBuffer( gl.ARRAY_BUFFER, cBuffer );
gl.bufferData( gl.ARRAY_BUFFER, flatten( colors ), gl.STATIC_DRAW );

var vColor = gl.getAttribLocation( program, "vColor" );
gl.vertexAttribPointer( vColor, 4, gl.FLOAT, false, 0, 0 );
gl.enableVertexAttribArray( vColor );
// in application

vec4 color = vec4( 1.0, 0.0, 0.0, 1.0 );
colorLoc = gl.getUniformLocation( program, "color" );
gl.uniform4f( colorLoc, color );

// in fragment shader (similar in vertex shader)

uniform vec4 color;

void main()
{
  gl_FragColor = color;
}

Angel and Shreiner: Interactive Computer Graphics 7E © Addison-Wesley 2015
Operators and Functions

• Standard C functions
  - Trigonometric
  - Arithmetic
  - Normalize, reflect, length

• Overloading of vector and matrix types

```cpp
mat4 a;
vec4 b, c, d;
c = b*a; // a column vector stored as a 1d array
d = a*b; // a row vector stored as a 1d array
```