Shaders and WebGL

February 23rd, 2017
Key Shader Concepts

Fragment Processing and Vertex Processing
Each does their own step in the pipeline
Vertex Shader

Process each vertex independently

Inputs:
• attribute variables (from buffers)
• uniform variables (constants from app)

Outputs:
• gl_Position (special magic variable)
• varying variables (interpolated for fragment shaders)
Fragment Shader

Process each fragment (pixel) independently

Inputs:
• varying variables (outputs from vertex shader)
• uniform variables (constants from app)

Outputs:
• gl_FragColor (magic variable)
• Other things for fragment tests
GLSL

• A special language for shaders
• Compiler built into graphics driver
• Used to write both kinds of shaders
GLSL Basics

main function

• always the entry point to a shader
• no arguments
• no return value
• inputs through variables
• outputs through variables
GLSL Type System

Strongly and Strictly Typed

- floats and ints are different
- need to be explicit about all conversions

Useful math types

- Short vectors: vec2, vec3, vec4
- Small matrices: mat3, mat4
Vector operations

```cpp
vec3 v;

float a = v.x;  // access a component
vec2 b = v.xy;  // any subset
vec2 c = v.yz;

vec3 d = v.zyx;  // any order (swizzle)
vec3 e = v.xxx;  // even repeats
```
Assembling vectors with type operators

vec2 a, b;
vec3 c;

vec3 f = vec3(1.0, 2.0, 3.0);
vec3 g = vec3(1, 2, 3);  // rare times an integer works
vec3 h = vec3(a, 1);
vec3 j = vec3(1, a);

vec4 k = vec4(a, b);
vec4 l = vec4(a, c.xy);
vec4 x;
vec3 p;
mat4 m;

vec4 y = M * x;
vec4 z = M * vec4(p, 1);

float a = dot(x, vec4(p, 0));
Yellow (simplest)

shdr.bkcore
precision highp float;
attribute vec3 position;
uniform mat4 modelViewMatrix;
uniform mat4 projectionMatrix;

void main()
{
    vec4 pos = modelViewMatrix * vec4(position, 1.0);
    gl_Position = projectionMatrix * pos;
}
Yellow (fragment)

```plaintext
precision highp float;

void main()
{
    gl_FragColor = vec4(1.0, 1.0, 0.0, 1.0);
}
```
Yellow Diffuse

The P4 shader (part of P5 as well)

Sortof: this does the lighting in the fragment shader

shdr.bkcore
Yellow Diffuse (vertex)

precision highp float;
attribute vec3 position;
attribute vec3 normal;
uniform mat3 normalMatrix;
uniform mat4 modelViewMatrix;
uniform mat4 projectionMatrix;
varying vec3 fNormal;

void main()
{
    fNormal = normalize(normalMatrix * normal);
    vec4 pos = modelViewMatrix * vec4(position, 1.0);
    gl_Position = projectionMatrix * pos;
}
Note the inputs

The application program sets the attributes and uniforms

We need to use the same names

Here the "application" is shdr.bkcore
Yellow Diffuse (fragment)

```glsl
precision highp float;
varying vec3 fNormal;

void main()
{
    vec3 dir = vec3(0,1,0); // high noon
    vec3 color = vec3(1,1,0); // yellow

    float diffuse = .5 + dot(fNormal,dir);
    gl_FragColor = vec4(diffuse * color, 1.0);
}
```
**Vertex Colors**

Compute the colors in the vertex shader
Pass to Fragment shader

shdr.bkcore
Vertex Colors (Fragment)

```cpp
precision highp float;
varying vec3 vColor;

void main()
{
    gl_FragColor = vec4(vColor, 1.0);
}
```
Vertex Colors (Vertex)

```cpp
precision highp float;
attribute vec3 position;
uniform mat4 modelViewMatrix;
uniform mat4 projectionMatrix;

varying vec3 vColor;

void main()
{
  vec4 pos = modelViewMatrix * vec4(position, 1.0);
  gl_Position = projectionMatrix * pos;
  vColor = vec3(0,.7,1);
}
```
Something different ...

Color the right side of the screen is different

GLSL built in variables (reference)

- Warning: `gl_FragCoord` is special (in pixels)
- Warning: resolution comes from `shdr.bkcore`

http://goo.gl/Hy9ir6
precision highp float;

uniform vec2 resolution;

void main()
{
   vec3 color;

   // gl_FragCoord is in pixels – so convert...
   float ndcx = (gl_FragCoord.x / resolution.x) - 1.0;

   if (ndcx > 0.0) {
      color = vec3(1.0, 1.0, 0.0);
   } else {
      color = vec3(1.0, 0.0, 1.0);
   }

   gl_FragColor = vec4(color, 1.0);
}

Control structures

If-then-else

```cpp
if (ndcx > 0.0) {
    color = vec3(1,1,0);
} else {
    color = vec3(1,0,1);
}
```

Step

```cpp
color = mix(vec3(1,1,0), vec3(1,0,1),
            step(ndcx, 0.0));
```
Step and Smoothstep

color = mix(vec3(1,1,0), vec3(1,0,1),
             step(0.0, ncdx) );

color = mix(vec3(1,1,0), vec3(1,0,1),
             smoothstep(-0.1, 0.1, ncdx) );
use 3D positions for colors

shdr.bkcore

What **coordinates system** to use position?

World coordinates?
Local coordinates?

Stripes
Checkers
Making cool shaders

Is actually hard in shdr.bkcore

• stuck with their attributes
• stuck with their uniforms

they do give time

Siren

consult the help
Complex Shader

Stripe Shader