

CPSC 314 Computer Graphics

Dinesh K. Pai

Nuts and bolts of
Graphics programming

1

Announcements

- Assignment 1 now out. Due Jan 23.
 - See coursepage/resources.html
 - There was a small fix at 11:15am today. So if you downloaded before that, please get the code again.
- Today: an introduction to programming with GLSL and WebGL

2

Introduction to Assignment 1

- Switch to demo

3

How to get started..

- First download assignment template and ensure that it runs in your preferred browser. See <https://github.com/mrdoob/three.js/wiki/How-to-run-things-locally>
- Work on the different parts in sequence. Later parts will need material covered later this week.

4

The good news

- Even though there are lots of details and options, a few useful things go a long way.
- After initial setup, most of your effort will be on translating graphics concepts into code
- For Assignment 1, this is already setup for you. You mainly have to focus on the vertex shader.

6

```

/**
 * UBC CPSC 314, Vjan2015
 * Outline of a Three.js program for this course
 */
// SCENE
var scene = new THREE.Scene();
// RENDERER
var renderer = new THREE.WebGLRenderer();
// CAMERA
var camera = new THREE.PerspectiveCamera(30, 1, 0.1, 1000);
// SHADERS
var gemMaterial = new THREE.ShaderMaterial({
  uniforms: { gemPosition: gemPosition},
  vertexShader: <VertexShaderSource>,
  fragmentShader: <FragmentShaderSource>
})
// OBJECT GEOMETRY
var gemGeometry = new THREE.SphereGeometry(1, 32, 32);
// OBJECT MESH
var gem = new THREE.Mesh(gemGeometry, gemMaterial);

scene.add(gem);

// SETUP UPDATE CALL-BACK
function update() {
  requestAnimationFrame(update);
  renderer.render(scene, camera);
}
update();

```

animation loop

7

Minimalist shaders

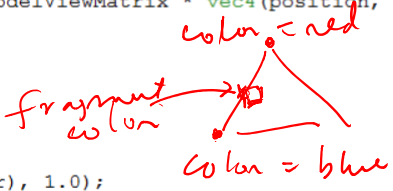
```

vertex shader
-----
uniform vec3 gemPosition;
varying vec3 color;

void main() {
    color = normal;
    gl_Position = projectionMatrix * modelViewMatrix * vec4(position, 1.0);
}

fragment shader
-----
varying vec3 color;
void main() {
    gl_FragColor = vec4(normalize(color), 1.0);
}

```



8

A closer look at GLSL shaders

9

GLSL

- OpenGL shading language
- C-like, w. data types and functions useful for graphics
 - vec3, vec4, dvec4, mat4, sampler2D ...
(OpenGL data are floats unless qualified)
 - <matrix-vector multiplication>, reflect, refract
- Used for both vertex shaders and fragment shaders, with small differences
- WebGL uses GLSL 1.0, which doesn't have many features of versions available in OpenGL
https://www.khronos.org/registry/gles/specs/2.0/GLSL_ES_Specification_1.0.17.pdf

10

Creating a Shader Program

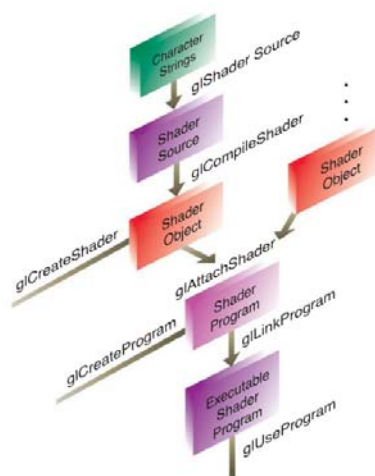


Figure 2.1 Shader-compilation command sequence

Source: OpenGL programming guide, 8th edition

11

Vertex Shader from textbook's hw2d example

```
#version 130

uniform float uVertexScale;

in vec2 aPosition;
in vec3 aColor;
in vec2 aTexCoord0, aTexCoord1;

out vec3 vColor;
out vec2 vTexCoord0, vTexCoord1;

void main() {
    gl_Position = vec4(aPosition.x * uVertexScale, aPosition.y, 0,1);
    vColor = aColor;
    vTexCoord0 = aTexCoord0;
    vTexCoord1 = aTexCoord1;
}
```

12

Summary of Key GLSL Concepts (1)

- 'uniform' type qualifier
 - Same for all vertices
- "in", "out", "varying" type qualifiers configure data flow in the pipeline.
- "in" type qualifiers
 - Input from previous shader stage
 - For vertex shaders, these are per-vertex attributes
- "out" type qualifiers
 - Outputs to next stage
 - gl_position is built-in output variable that must be set before rasterization

13

Summary of Key GLSL Concepts (2)

- “varying” type qualifiers
 - Equivalent to “out” for VS, “in” for FS
 - Deprecated since GLSL 1.3
 - Three.js only supports it at present
- Support for geometry, vector and matrix arithmetic
 - length, distance, dot, cross, normalize, reflect
- Compiled by the WebGL, at runtime

14

Three.js support

- THREE.ShaderMaterial() lets you set shaders, uniforms
- Built-in uniforms and attributes. See <http://threejs.org/docs/#Reference/Renderers/WebGL/WebGLProgram>
- Some vertex attributes
 - position, normal, and uv
- Some uniforms
 - modelView matrix and cameraPosition

15

ShaderMaterial Example

```
var material = new THREE.ShaderMaterial( {  
    uniforms: {  
        time: { type: "f", value: 1.0 },  
        resolution: { type: "v2", value: new THREE.Vector2() }  
    },  
    attributes: {  
        vertexOpacity: { type: 'f', value: [] }  
    },  
    vertexShader: document.getElementById( 'vertexShader' ).textContent,  
    fragmentShader: document.getElementById( 'fragmentShader' ).textContent  
} );
```

<http://threejs.org/docs/#Reference/Materials/ShaderMaterial>

16

Animation (infinite) Loop

```
// SETUP UPDATE CALL-BACK  
function update() {  
    requestAnimationFrame(update); // next frame  
    renderer.render(scene, camera);  
}  
  
// Do this last  
update();
```

17

Debugging your program

- Debugging GLSL programs can be challenging. Keep calm. Many problems are due to strict typing. E.g., float literals must use decimal point
- Good news: easy to run and see results. No compilation step. Test code as you write it.
- Browsers provide some tools for JavaScript debugging, but not for GLSL programs
 - Toggle console with, e.g., <F12>
 - Reload page with CTRL-R

18

Next class

- Back to 3D Math for Graphics
 - Read Chapter 2, Chapter 3 up to 3.5.

19