Programming with OpenGL
Part 5: More GLSL

CS 432 Interactive Computer Graphics
Prof. David E. Breen
Department of Computer Science
Objectives

• Coupling shaders to applications
  - Reading
  - Compiling
  - Linking
• Vertex Attributes
• Setting up uniform variables
• Example applications
Linking Shaders with Application

• Read shaders
• Compile shaders
• Create a program object
• Link everything together
• Link variables in application with variables in shaders
  - Vertex attributes
  - Uniform variables
Attribute and Varying Qualifiers

• Starting with OpenGL GLSL 1.4 *attribute* and *varying* qualifiers have been replaced by *in* and *out* qualifiers

• No changes needed in application

• Vertex shader example:

```
# WebGL version 1.0
# GLSL ES version 1.0
# GLSL version 1.2
attribute vec3 vPosition;
varying vec3 color;

# WebGL version 2.0
# GLSL ES version 3.0
# GLSL version 1.4
in vec3 vPosition;
out vec3 color;
```
Coloring Each Vertex  
(Old)

attribute vec3 vPosition, vColor;

varying vec3 fcolor;

void main()
{
    gl_Position = vec4(vPosition, 1);
    fcolor = vColor;
}
Coloring Each Vertex (New)

```glsl
in vec3 vPosition, vColor;
out vec3 fcolor;

void main()
{
    gl_Position = vec4(vPosition, 1);
    fcolor = vColor;
}
```
Coloring Each Fragment (Old)

```glsl
precision mediump float;
varying vec3 fcolor;

void main()
{
   gl_FragColor = vec4(fcolor, 1);
}
```
Coloring Each Fragment (New)

```glsl
precision highp float;
in vec3 fcolor;
out vec4 fragcolor;

void main()
{
    fragcolor = vec4(fcolor, 1);
}
```

Go to square.html
Reading a Shader

- Shaders are added to the program object and compiled
- Usual method of passing a shader is as a null-terminated string using the function
  - `gl.shaderSource(fragShdr, fragElem.text);`
- If shader is in HTML file, we can get it into application by `getElementById` method
- If the shader is in a file, we can write a reader to convert the file to a string
Adding a Vertex Shader

```javascript
var vertShdr;
var vertElem =
    document.getElementById( vertexShaderId );

vertShdr = gl.createShader( gl.VERTEX_SHADER );

gl.shaderSource( vertShdr, vertElem.text );
gl.compileShader( vertShdr );

// after program object created
gl.attachShader( program, vertShdr );
```
Adding a Fragment Shader

var fragShdr;
var fragElem =
    document.getElementById( fragmentShaderId );

fragShdr = gl.createShader( gl.FRAGMENT_SHADER );

// after program object created
gl.attachShader( program, fragShdr );
Shader File Reader

• Following code may be a security issue with some browsers if run locally

```javascript
function getShader(gl, shaderName, type) {
    var shader = gl.createShader(type);
    shaderScript = loadFileAJAX(shaderName);
    if (!shaderScript) {
        alert("Could not find shader source: "+shaderName);
    }
}
```

• Best to keep shader code in html file for now

Angel and Shreiner: Interactive Computer Graphics 7E © Addison-Wesley 2015
Program Object

• Container for shaders
  - Can contain multiple shaders
  - Other GLSL functions

```javascript
var program = gl.createProgram();

gl.attachShader( program, vertShdr );
gl.attachShader( program, fragShdr );
gl.linkProgram( program );
```

Go to `initShaders.js`
Vertex Attributes

- Vertex attributes are named in the shaders
- Linker forms a table
- Application can get index from table and ties it to an application variable
- Similar process for uniform variables
Adding Color

• If we set a color in the application, we can send it to the shaders as a per vertex attribute or as a uniform variable depending on how often it changes.

• Let’s associate a color with each vertex.

• Set up an array of same size as positions.

• Send to GPU as a buffer object.
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, 3, gl.FLOAT, false, 0, 0 );
gl.enableVertexAttribArray( vColor );
var vBuffer = gl.createBuffer();
gl.bindBuffer( gl.ARRAY_BUFFER, vBuffer );
gl.bufferData( gl.ARRAY_BUFFER, flatten(vertices),
    gl.STATIC_DRAW );

var vPosition = gl.getAttribLocation( program, "vPosition" );
gl.vertexAttribPointer( vPosition, 3, gl.FLOAT, false, 0, 0 );
gl.enableVertexAttribArray( vPosition );

Go to square.js
Sending a Uniform Variable

// 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;
}
attribute vec3 vPosition, vColor;
varying vec3 fcolor;

void main()
{
    gl_Position = vec4(vPosition, 1);
    fcolor = vColor;
}
in vec3 vPosition, vColor;
out vec3 fcolor;

void main()
{
    gl_Position = vec4(vPosition, 1);
    fcolor = vColor;
}
precision mediump float;
varying vec3 fcolor;

void main()
{
    gl_FragColor = vec4(fcolor, 1);
}
precision highp float;
in vec3 fcolor;
out vec4 fragcolor;

void main()
{
    fragcolor = vec4(fcolor, 1);
}

Go to square.html
function render() {

    gl.clear( gl.COLOR_BUFFER_BIT);
    gl.drawArrays( gl.TRIANGLE_FAN, 0, 4 );

}
Review Code

Go to square.html
webgl-utils.js
initShaders.js
MV.js
square.js
Vertex Shader Applications

• Moving vertices
  - Morphing
  - Wave motion
  - Fractals

• Lighting
  - More realistic models
  - Cartoon shaders
in vec4 vPosition;
out vec4 color;
uniform float xs, zs, // frequencies
uniform float h; // height scale
uniform float time; // time from app
void main()
{
    vec4 t = vPosition;
    t.y = vPosition.y
        + h*sin(time + xs*vPosition.x)
        + h*sin(time + zs*vPosition.z);
    gl_Position = t;
}
in vec3 vPosition;
uniform mat4 ModelViewProjectionMatrix;
uniform vec3 vel;
uniform float g, m, t;
void main()
{
    vec3 object_pos;
    object_pos.x = vPosition.x + vel.x*t;
    object_pos.y = vPosition.y + vel.y*t + g/(2.0*m)*t*t;
    object_pos.z = vPosition.z + vel.z*t;
    gl_Position = ModelViewProjectionMatrix*vec4(object_pos,1);
}
Vertex vs Fragment Lighting

- Per vertex lighting
  (Gouraud shading)

- Per fragment lighting
  (Phong shading)
Fragment Shader Applications

Texture mapping

Procedural textures  environment mapping  bump mapping

Programming with OpenGL
Part 6: Three Dimensions
Objectives

• Develop a more sophisticated three-dimensional example
  - Sierpinski gasket: a fractal

• Introduce hidden-surface removal
Three-dimensional Applications

• In OpenGL, two-dimensional applications are a special case of three-dimensional graphics

• Going to 3D
  - Not much changes
  - Use `vec3`, `glUniform3f`
  - Have to worry about the order in which primitives are rendered or use hidden-surface removal
Sierpinski Gasket (2D)

• Start with a triangle

• Connect bisectors of sides and remove central triangle

• Repeat
Example

• Five subdivisions
The gasket as a fractal

• Consider the filled area (black) and the perimeter (the length of all the lines around the filled triangles)

• As we continue subdividing
  - the area goes to zero
  - but the perimeter goes to infinity

• This is not an ordinary geometric object
  - It is neither two- nor three-dimensional

• It is a fractal (fractional dimension) object
Gasket Program

- HTML file
  - Same as in other examples
  - Pass through vertex shader
  - Fragment shader sets color
  - Read in JS file
var points = [];
var NumTimesToSubdivide = 5;

/* initial triangle */

var vertices = [
    vec2(  -1,  -1 ),
    vec2(   0,    1 ),
    vec2(   1,  -1 )
];

divideTriangle( vertices[0], vertices[1],
    vertices[2], NumTimesToSubdivide);
Draw one triangle

/* display one triangle */

function triangle( a, b, c ){
    points.push( a, b, c );
}

Angel and Shreiner: Interactive Computer Graphics 7E © Addison-Wesley 2015
function divideTriangle( a, b, c, count ){
  // check for end of recursion
  if ( count === 0 ) {
    triangle( a, b, c );
  }
  else {
    // bisect the sides
    var ab = mix( a, b, 0.5 );
    var ac = mix( a, c, 0.5 );
    var bc = mix( b, c, 0.5 );
    --count;
    // three new triangles
    divideTriangle( a, ab, ac, count-1 );
    divideTriangle( c, ac, bc, count-1 );
    divideTriangle( b, bc, ab, count-1 );
  }
}
```javascript
var program = initShaders(gl, "vertex-shader", "fragment-shader");

gl.useProgram( program );
var bufferId = gl.createBuffer();
gl.bindBuffer( gl.ARRAY_BUFFER, bufferId );
gl.bufferData(gl.ARRAY_BUFFER, flatten(points), gl.STATIC_DRAW );
var vPosition = gl.getAttribLocation( program, "vPosition" );
gl.vertexAttribPointer( vPosition, 2, gl.FLOAT, false, 0, 0 );
gl.enableVertexAttribArray( vPosition );
render();
```
function render() {
    gl.clear( gl.COLOR_BUFFER_BIT );
    gl.drawArrays( gl.TRIANGLES, 0, points.length );
}
Example

- Five subdivisions
Moving to 3D

• We can easily make the program three-dimensional by using
  \[
  \text{point3 } v[3]
  \]
  and we start with a tetrahedron
3D Gasket

• We can subdivide each of the four faces

• Appears as if we remove a solid tetrahedron from the center leaving four smaller tetrahedra

• Code almost identical to 2D example
Almost Correct

- Because the triangles are drawn in the order they are specified in the program, the front triangles are not always rendered in front of triangles behind them.

get this

want this
Hidden-Surface Removal

• We want to see only those surfaces in front of other surfaces

• OpenGL uses a *hidden-surface* method called the z-buffer algorithm that saves depth information as objects are rendered so that only the front objects appear in the image
Z-buffering

• **Z-buffering (depth-buffering)** is a visible surface detection algorithm
• Implementable in hardware and software
• Requires data structure (**Z-buffer**) in addition to frame buffer.
• **Z-buffer** stores values \([0 \ldots ZMAX]\) corresponding to depth of each point.
• If the point is closer than one in the buffers, it will replace the buffered values
# Z-buffering

<table>
<thead>
<tr>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

+ 

<table>
<thead>
<tr>
<th>5</th>
<th>5</th>
<th>5</th>
<th>5</th>
<th>5</th>
<th>5</th>
<th>5</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>

= 

<table>
<thead>
<tr>
<th>5</th>
<th>5</th>
<th>5</th>
<th>5</th>
<th>5</th>
<th>5</th>
<th>5</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>5</th>
<th>5</th>
<th>5</th>
<th>5</th>
<th>5</th>
<th>5</th>
<th>5</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>

+ 

<table>
<thead>
<tr>
<th>3</th>
<th>4</th>
<th>3</th>
<th>5</th>
<th>4</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>8</td>
</tr>
<tr>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
</tr>
</tbody>
</table>

= 

<table>
<thead>
<tr>
<th>5</th>
<th>5</th>
<th>5</th>
<th>5</th>
<th>5</th>
<th>5</th>
<th>5</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>5</th>
<th>5</th>
<th>5</th>
<th>5</th>
<th>5</th>
<th>5</th>
<th>5</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>
Z-buffering w/ front/back clipping

for (y = 0; y < YMAX; y++)
    for (x = 0; x < XMAX; x++) {
        F[x][y] = BACKGROUND_VALUE;
        Z[x][y] = -1; /* Back value in NPC */
    }

for (each polygon)
    for (each pixel in polygon’s projection) {
        pz = polygon’s z-value at pixel coordinates (x,y)
        if (pz < FRONT && pz > Z[x][y]) { /* New point is behind front plane & closer than previous point */
            Z[x][y] = pz;
            F[x][y] = polygon’s color at pixel coordinates (x,y)
        }
    }
Using the z-buffer algorithm

• The algorithm uses an extra buffer, the z-buffer, to store depth information as geometry travels down the pipeline.

• Depth buffer is required to be available in WebGL.

• It must be
  - Enabled
    • `gl.enable(gl.DEPTH_TEST)`
  - Cleared in for each render
    • `gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT)`
Surface vs Volume Subdivision

• In our example, we divided the surface of each face.
• We could also divide the volume using the same midpoints.
• The midpoints define four smaller tetrahedrons, one for each vertex.
• Keeping only these tetrahedrons removes a volume in the middle.
• See text for code.
Volume Subdivision