GLSL cont'd

see also: separate slides on texture mapping

EECS 487

October 4, 2006

to do

- SA sign-up
- texture mapping (separate slides)
- environment mapping
- bump mapping
- more on GLSL
- texturing in GLSL

The following slides are adopted from:

An Introduction to the OpenGL Shading Language

by Keith O'Conor

Image Synthesis Group
Trinity College, Dublin

http://isg.cs.tcd.ie/oconork/presentations/GLSLseminar.ppt

Bypassing pipeline

- Vertex processes bypassed
  - Vertex Transformation
  - Normal Transformation, Normalization
  - Lighting
  - Texture Coordinate Generation and Transformation
- Fragment processes bypassed
  - Texture accesses & application
  - Fog

Previous programmability

- Assembly programs
  - ARB_vertex_program
  - ARB_fragment_program
  - Messy!
- Needed general, readable & maintainable language

Types

void

float vec2 vec3 vec4

mat2 mat3 mat4

int ivec2 ivec3 ivec4

bool bvec2 bvec3 bvec4

sampler*n*D, samplerCube,
samplerShadow*n*D

Types

- Structs
- Arrays
  - One dimensional
  - Constant size (ie `float array[4];`)
- Reserved types
  - **`half hvec2 hvec3 hvec4`**
  - **`fixed fvec2 fvec3 fvec4`**
  - **`double dvec2 dvec3 dvec4`**

Type qualifiers

- attribute
  - Changes per-vertex
    - eg. position, normal etc.
- uniform
  - Does not change between vertices of a batch
    - eg light position, texture unit, other constants
- varying
  - Passed from VS to FS, interpolated
    - eg texture coordinates, vertex color

Operators

- grouping: ()
- array subscript: []
- function call and constructor: ()
- field selector and swizzle: .
- postfix: ++ --
- prefix: ++ -- + - !

Operators

- binary: * / + -
- relational: < <= > >=
- equality: == !=
- logical: && ^^ ||
- selection: ?:
- assignment: = *= /= += -=

Reserved Operators

- prefix: ~
- binary: %
- bitwise: << >> & ^ |
- assignment: %= <<= >>= &= ^= |=

Scalar/Vector Constructors

- **No casting**

```
float f; int i; bool b;
vec2 v2; vec3 v3; vec4 v4;

vec2(1.0 ,2.0)
vec3(0.0 ,0.0 ,1.0)
vec4(1.0 ,0.5 ,0.0 ,1.0)
vec4(1.0)                    // all 1.0
vec4(v2 ,v2)
vec4(v3 ,1.0)

float(i)
int(b)
```

Matrix Constructors

```
vec4 v4; mat4 m4;

mat4( 1.0, 2.0, 3.0, 4.0,
      5.0, 6.0, 7.0, 8.0,
      9.0, 10., 11., 12.,
      13., 14., 15., 16.)   // row major

mat4( v4, v4, v4, v4)       // 4 columns
mat4( 1.0)                  // identity matrix
mat3( m4)                   // upper 3x3
vec4( m4)                   // 1st column
float( m4)                  // upper 1x1
```

Accessing components

- component accessor for vectors
  - xyzw rgba stpq [i]
- component accessor for matrices
  - [i] [i][j]

Vector components

```
vec2 v2;
vec3 v3;
vec4 v4;

v2.x    // is a float
v2.z    // wrong: undefined for type
v4.rgba // is a vec4
v4.stp  // is a vec3
v4.b    // is a float
v4.xy   // is a vec2
v4.xgp  // wrong: mismatched component sets
```

Swizzling & Smearing

- R-values

```
vec2 v2;
vec3 v3;
vec4 v4;

v4.wzyx // swizzles, is a vec4
v4.bgra // swizzles, is a vec4
v4.xxxx // smears x, is a vec4
v4.xxx  // smears x, is a vec3
v4.yyxx // duplicates x and y, is a vec4
v2.yyyy // wrong: too many components for type
```

Vector Components

- L-values

```
vec4 v4 = vec4( 1.0, 2.0, 3.0, 4.0);

v4.xw = vec2( 5.0, 6.0); // (5.0, 2.0, 3.0, 6.0)
v4.wx = vec2( 7.0, 8.0); // (8.0, 2.0, 3.0, 7.0)
v4.xx = vec2( 9.0,10.0); // wrong: x used twice
v4.yz = 11.0;            // wrong: type mismatch
v4.yz = vec2( 12.0 );    // (8.0,12.0,12.0, 7.0)
```

Flow Control

- expression ? trueExpression : falseExpression
- if, if-else
- for, while, do-while
- return, break, continue
- discard (fragment only)

Built-in variables

- Attributes & uniforms
- For ease of programming
- OpenGL state mapped to variables
- Some special variables are required to be written to, others are optional

Special built-ins

- Vertex shader

```
vec4  gl_Position;      // must be written
vec4  gl_ClipPosition;  // may be written
float gl_PointSize;     // may be written
```

- Fragment shader

```
float gl_FragColor;     // may be written
float gl_FragDepth;     // may be read/written
vec4  gl_FragCoord;     // may be read
bool  gl_FrontFacing;   // may be read
```

Attributes

- Built-in

```
attribute vec4  gl_Vertex;
attribute vec3  gl_Normal;
attribute vec4  gl_Color;
attribute vec4  gl_SecondaryColor;
attribute vec4  gl_MultiTexCoordn;
attribute float gl_FogCoord;
```

- User-defined

```
attribute vec3  myTangent;
attribute vec3  myBinormal;
Etc…
```

Built-in Uniforms

```
uniform   mat4  gl_ModelViewMatrix;
uniform   mat4  gl_ProjectionMatrix;
uniform   mat4  gl_ModelViewProjectionMatrix;
uniform   mat3  gl_NormalMatrix;
uniform   mat4  gl_TextureMatrix[n];

struct gl_MaterialParameters {
  vec4  emission;
  vec4  ambient;
  vec4  diffuse;
  vec4  specular;
  float shininess;
};
uniform gl_MaterialParameters gl_FrontMaterial;
uniform gl_MaterialParameters gl_BackMaterial;
```

Built-in Uniforms

```
struct gl_LightSourceParameters {
  vec4  ambient;
  vec4  diffuse;
  vec4  specular;
  vec4  position;
  vec4  halfVector;
  vec3  spotDirection;
  float spotExponent;
  float spotCutoff;
  float spotCosCutoff;
  float constantAttenuation
  float linearAttenuation
  float quadraticAttenuation
};
Uniform gl_LightSourceParameters
   gl_LightSource[gl_MaxLights];
```

Built-in Varyings

```
varying   vec4  gl_FrontColor        // vertex
varying   vec4  gl_BackColor;        // vertex
varying   vec4  gl_FrontSecColor;    // vertex
varying   vec4  gl_BackSecColor;     // vertex

varying   vec4  gl_Color;            // fragment
varying   vec4  gl_SecondaryColor;   // fragment

varying   vec4  gl_TexCoord[];       // both
varying   float gl_FogFragCoord;     // both
```

Built-in functions

- Angles & Trigonometry
  - **radians, degrees, sin, cos, tan, asin, acos, atan**
- Exponentials
  - **pow, exp2, log2, sqrt, inversesqrt**
- Common
  - **abs, sign, floor, ceil, fract, mod, min, max, clamp**

Built-in functions

- Interpolations
  - **mix**(x,y,a) **x*(1.0-a) + y*a**)
  - **step**(edge,x) **x <= edge ? 0.0 : 1.0**
  - **smoothstep**(edge0,edge1,x)

```
t = (x-edge0)/(edge1-edge0);
t = clamp( t, 0.0, 1.0);
return t*t*(3.0-2.0*t);
```

Built-in functions

- Geometric
  - **length, distance, cross, dot, normalize, faceForward, reflect**
- Matrix
  - **matrixCompMult**
- Vector relational
  - **lessThan, lessThanEqual, greaterThan, greaterThanEqual, equal, notEqual, notEqual, any, all**

Built-in functions

- Texture
  - **texture1D, texture2D, texture3D, textureCube**
  - **texture1DProj, texture2DProj, texture3DProj, textureCubeProj**
  - **shadow1D, shadow2D, shadow1DProj, shadow2Dproj**
- Vertex
  - **ftransform**

Integrating GLSL with OpenGL

- 2 basic types of “object”
  - Shader object
  - Program object
- Create vertex & fragment shader objects
  - Load source code from file
  - Compile each
- Create program object
  - Attach shaders
  - Link program
- Use program

Creating objects

```
GLuint glCreateProgram();
  // Allocates a shader program.

GLuint glCreateShader(Glenum type);
  // Allocates a shader object. type must be
  // GL_VERTEX_SHADER or GL_FRAGMENT_SHADER.
```

Load the source

```
void glShaderSource(
  GLuint shader,
  GLsizei nstrings,
  const GLchar** strings,
  const GLint* lengths);
   // if lengths==NULL, strings are
   // null-terminated
   // need utility function to read
   // shader source from file and
   // write it into a string
```

Compile

```
void glCompileShader(GLuint shader);
// compiles the source code previously loaded
// for given shader.

// check for success/failure using:
GLint status = 0;
glGetShaderiv(shader, GL_COMPILE_STATUS, &status);
if (status != GL_TRUE)
   // see next slide!
```

Report errors

```
// Return the log associated with the last
// compilation of a shader.
void glGetShaderInfoLog(
   GLuint shader,
   Glsizei buf_size,
   Glsizei* length,
   char* info_log);

// E.g.:
const GLsizei BUF_SIZE = 4096;
char info_log[BUF_SIZE] = {0};
GLsizei len = 0;
glGetShaderInfoLog(shader, BUF_SIZE, &len, info_log);
cerr << "ShaderInfoLog: ”<< endl
     << info_log << endl;
```

Attaching & Linking

```
void glAttachShader(GLuint program, GLuint shader);
// twice: once for vertex shader and
// once for fragment shader

void glLinkProgram(GLuint program);
// After attaching all shaders, link program.
// If no errors, program is now ready to use.

// check for errors:
GLint status = 0;
glGetProgramiv(program, GL_LINK_STATUS, &status);
if (status != GL_TRUE)
   // see next slide
```

Report errors

```
// Return the log associated with the last
// link attempt for a program:
void glGetProgramInfoLog(
   GLuint program,
   GLsizei buf_size,
   GLsizei* length,
   char* info_log);

// E.g.:
const GLsizei BUF_SIZE = 4096;
char info_log[BUF_SIZE] = {0};
GLsizei len = 0;
glGetProgramInfoLog(program, BUF_SIZE, &len, info_log);
cerr << "ProgramInfoLog: "<< endl
     << info_log << endl;
```

Activating the program

```
void glUseProgram(GLuint program);
// switches on shader, bypasses FFP
// if program == 0, shaders turned off
```

Other functions

```
void glDeleteShader(GLuint shader);
void glDeleteProgram(GLuint program);
// release resources associated with
// given shader or program, once they
// are no longer being used.

void glValidateProgram(GLuint program);
// validates program against current OpenGL
// state settings

// E.g.:
GLint status = 0;
glGetProgramiv(program, GL_VALIDATE_STATUS, &status);
if (status == GL_TRUE)
   // program is activated and will execute
```

Loading uniform variables

```
// Returns index of uniform variable name
// associated with the shader program:
GLint glGetUniformLocation(
   GLuint program,
   const char* name);

// E.g.: for variable named “delta”:
GLint index = glGetUniformLocation(prog, “delta”);

// Set the value of a given uniform variable:
glUniform{1234}{if}(GLint index, TYPE value);

// E.g.:
void glUniform3f(GLint location, GLfloat v0,
                 GLfloat v1, GLfloat v2);
```

Loading attribute variables

```
// Returns index of attribute variable name
// associated with the shader program:
GLint glGetAttribLocation(
   GLuint program,
   const char* name
   );

// Set the value of a given uniform variable:
glVertexAttrib{1234}{sfd}(GLint index, TYPE values);

// E.g.:
void glVertexAttrib3f(
   GLint location, GLfloat v0, GLfloat v1, GLfloat v2
   );
```

Check documentation for details. E.g.:

http://developer.3dlabs.com/documents/GLmanpages/glUniform.htm

http://developer.3dlabs.com/documents/GLmanpages/glVertexAttrib.htm

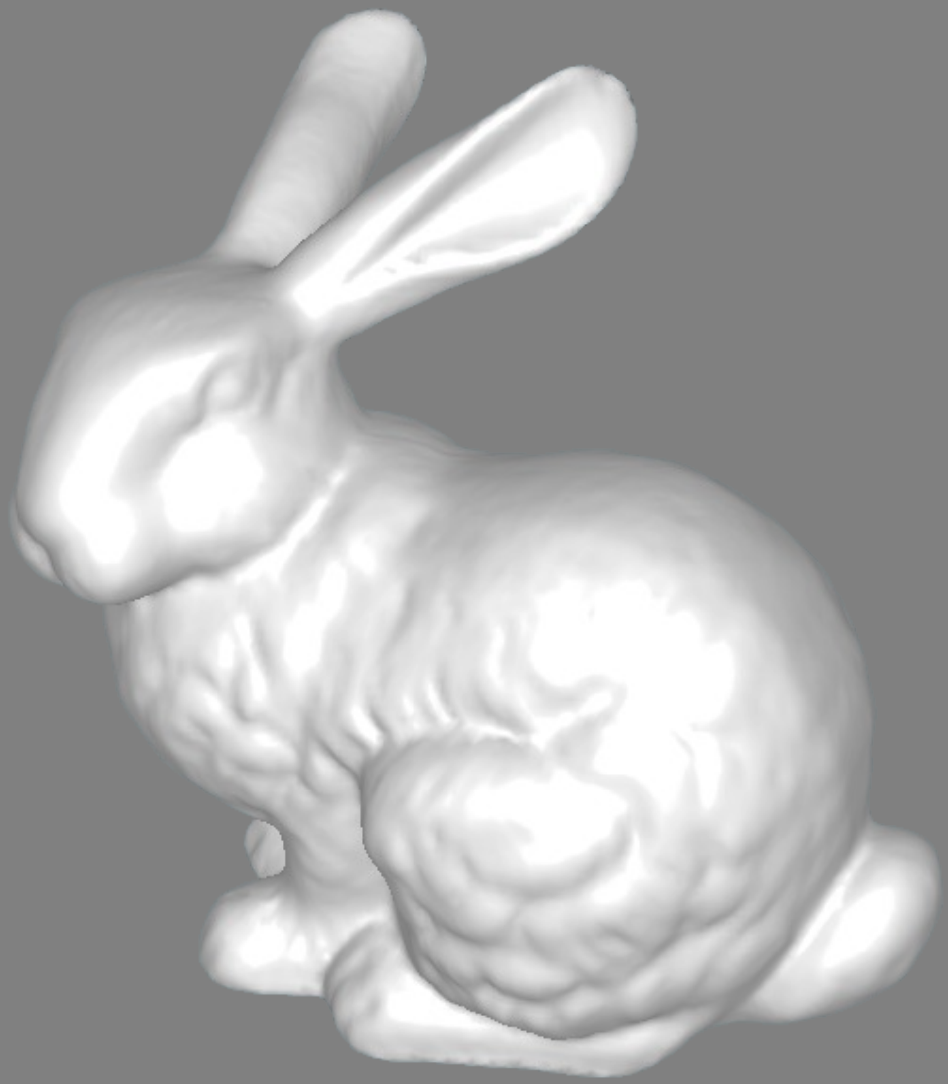
Ivory – vertex shader

```
uniform vec4 lightPos;

varying vec3 normal;
varying vec3 lightVec;
varying vec3 viewVec;

void main(){
   gl_Position = gl_ModelViewProjectionMatrix * gl_Vertex;
   vec4 vert = gl_ModelViewMatrix * gl_Vertex;

   normal   = gl_NormalMatrix * gl_Normal;
   lightVec = vec3(lightPos - vert);
   viewVec  = -vec3(vert);
}
```

Ivory – fragment shader

```
varying vec3 normal;
varying vec3 lightVec;
varying vec3 viewVec;

void main(){
   vec3 norm = normalize(normal);

   vec3 L = normalize(lightVec);
   vec3 V = normalize(viewVec);
   vec3 halfAngle = normalize(L + V);

   float NdotL = dot(L, norm);
    float NdotH = clamp(dot(halfAngle, norm), 0.0, 1.0);

    // "Half-Lambert" technique for more pleasing diffuse term
   float diffuse  = 0.5 * NdotL + 0.5;
   float specular = pow(NdotH, 64.0);

   float result = diffuse + specular;

   gl_FragColor = vec4(result);
}
```

Ivory – output



Toon shading

- Let d = dot product of unit light direction and normal
  - define your own light direction, e.g. in eye space
- Decide a “dark” color and a “light” color
- Return output color based on cut-off d_0:

  `return (d < d0) ? dark_color : light color;`
- Can get a soft transition using:

  ```
  float t = smoothstep(d0, d1, d);
  ```

  to define interpolation between dark and light color

Toon shading

Example: accessing textures

proj1/env-map.fp