GLSL continued

EECS 487

January 31, 2007

today

- presentation: Chay Beng Tan
- coordinate systems, transforms
- more on GLSL
- OpenGL lighting details

coordinate system overview

- object space
  - “natural” coordinates for defining object
- world space
  - coordinate system for whole scene
- eye space
  - coordinates are relative to camera

object space

- convenient for defining an object
- e.g.: “canonical cylinder”:
  - base at origin
  - aligned with Y-axis (say)
  - height 1, radius 1

world space

- coordinate system used to define a scene
- origin and axes are arbitrary (but fixed)
- scene objects are located/oriented relative to world origin and axes

eye space

- origin at camera
- X direction points right
- Y direction points up
- -Z direction points forward along line of sight
  - i.e., Z direction points behind camera
- in jot called “cam space”

digression: lights in jot

- lights can be defined in eye space or world space
- e.g.: “headlight”
  - always pointing forward along line of sight
  - trivial to define in eye space
  - in world space, would have to update the light each frame

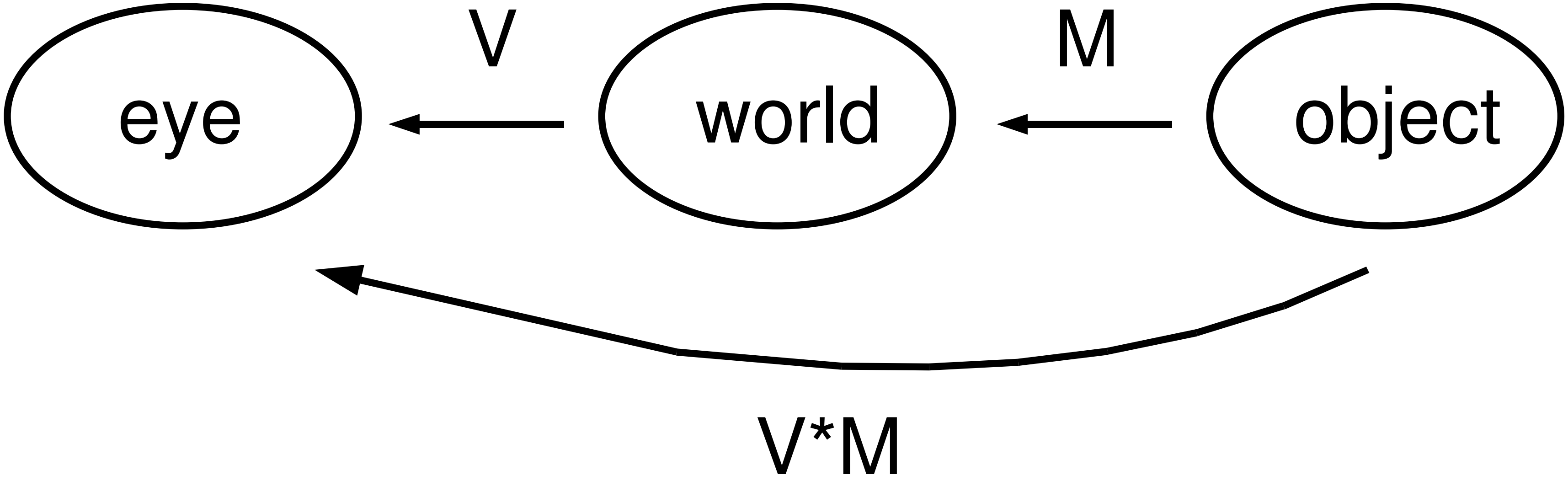
transforms

- modeling transform “M”
  - maps object space to world space
  - e.g., scale, rotate, & translate canonical cylinder to make it appear as a telephone pole, say

transforms

- viewing transform “V”
  - maps world space to eye space
- modelview transform: V*M
  - combines M and V
  - maps object space to eye space

transforms



V

M

eye

world

object

V*M

representation

- transforms represented as 4x4 matrices
- 3D points and vectors have xyz coordinates, plus a 4th coordinate: w
- points: w = 1
- vectors: w = 0
- allows translation via matrix multiplication

combining transforms

say **x** is an object space location

(4x1 column vector)

w = M**x** is same location in world coords

e = V**w** is same location in eye coords

e = V**w** = VM**x**

VM is matrix product: V*M

- maps object coords to eye coords

more on transforms
next week

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 (e.g.: `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]
- examples on next slide...

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 Varying

```
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

Example: lighting.vp

```
varying vec3 N;  // per-fragment normal in eye coords
varying vec4 P;  // per-fragment position in eye coords

void main()
{
   // compute the vertex normal and position in eye
   coordinates:
   N = gl_NormalMatrix * gl_Normal;
   P = gl_ModelViewMatrix * gl_Vertex;

   // output vertex position in clip coordinates
   gl_Position = ftransform();
}

// lighting.vp
```

Example: lighting.fp

next few slides...

```
// Fragment program to compute OpenGL lighting model
// for case of directional or point lights 0 thru 4.
//
// Note: Assumes lights 0 - 3 are enabled (and others disabled).

varying vec3 N;  // per-fragment normal, eye space
varying vec4 P;  // per-fragment location, eye space
```

```
void main()
{
   // material ambient, diffuse, and specular colors:
   vec4 a = gl_FrontMaterial.ambient;
   vec4 d = gl_FrontMaterial.diffuse;
   vec4 s = gl_FrontMaterial.specular;

   // N is interpolated across the triangle, so normalize it:
   vec3 n = normalize(N);

   // unit view vector in eye space:
   vec3 v = normalize(-P.xyz);

   // base color: global ambient light:
   vec4 color = gl_LightModel.ambient * a;

   // some ATI cards have problems with for loops
   color += light_contribution(gl_LightSource[0],a,d,s,n,v);
   color += light_contribution(gl_LightSource[1],a,d,s,n,v);
   color += light_contribution(gl_LightSource[2],a,d,s,n,v);
   color += light_contribution(gl_LightSource[3],a,d,s,n,v);

   gl_FragColor = color;
}
```

```
vec4 light_contribution(
   in gl_LightSourceParameters light, // light parameters
   in vec4 a,                         // material ambient color
   in vec4 d,                         // material diffuse color
   in vec4 s,                         // material specular color
   in vec3 n,                // per-pixel unit normal, eye space
   in vec3 v                 // per-pixel unit view vector, eye space
   )
{
   // compute unit vector to light,
   // handle point or directional case:
   vec3 l = normalize(
      (light.position.w == 0.0 ?
       light.position :          // directional
       light.position - P).xyz   // positional
      );

   // cont'd...
```

```
// attenuation:
float t = 1.0;
if (light.position.w != 0.0) { // if positional
   float d = distance(light.position, P);
   float k0 = light.constantAttenuation;
   float k1 = light.linearAttenuation;
   float k2 = light.quadraticAttenuation;
   t = 1.0 / (k0 + d*(k1 + d*k2));
}

// spotlight:
if (light.spotCutoff < 180.0) { // if spotlight in effect
   float sl = dot(normalize(light.spotDirection), -l);
   t *= (sl < light.spotCosCutoff) ?
      0.0 :                                   // out of cone
      pow(max(sl,0.0), light.spotExponent);   // inside cone
}

// cont'd...
```

```
   // n dot l used in diffuse part:
   float nl = max(dot(n,l), 0.0);

   // h dot n raised to shininess power; used in specular part
   vec3  h = normalize(l + v);
   float hn = pow(max(0.0, dot(h,n)), gl_FrontMaterial.shininess);

   return t*((light.ambient  * a)      + // ambient contribution
             (light.diffuse  * d) * nl + // diffuse contribution
             (light.specular * s) * hn); // specular contribution
} // light_contribution
```

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

demos

- project 2 shaders
  - provided in support code