LEC11: GLSL Operators

Ku-Jin Kim
School of Computer Science & Engineering
Kyungpook National University

Contents

- OpenGL Shading Language (GLSL)
  - Operators and Functions
  - Constructors
  - Qualifiers
- Program Example
  - Translation with changing vertex position
  - Translation with Vertex Shader

OpenGL Shading Language (GLSL)

- Operators and Functions
- Constructors
- Qualifiers

https://www.khronos.org/registry/OpenGL/specs/gl/GLSLangSpec.1.20.pdf

GLSL Operators

- Standard C operators without
  - address-of (&) or dereference (*)
  - type casting

Precedence	Operator Class	Operators	Associativity
1 (highest)	parenthetical grouping	( )	NA
2	array subscript function call and constructor structure field or method selector, swizzler post fix increment and decrement	[ ] ( ) . ++ --	Left to Right
3	prefix increment and decrement unary (tilde is reserved)	++ -- + - ~ !	Right to Left
4	multiplicative (modulus reserved)	* / %	Left to Right
5	additive	+ -	Left to Right
6	bit-wise shift (reserved)	<< >>	Left to Right
7	relational	< > <= >=	Left to Right
8	equality	== !=	Left to Right
9	bit-wise and (reserved)	&	Left to Right
10	bit-wise exclusive or (reserved)	^	Left to Right
11	bit-wise inclusive or (reserved)	|	Left to Right
12	logical and	&&	Left to Right
13	logical exclusive or	^^	Left to Right
14	logical inclusive or	||	Left to Right
15	selection	? :	Right to Left
16	Assignment arithmetic assignments (modulus, shift, and bit-wise are reserved)	= += -= *= /= %= <<= >>= &= ^= |=	Right to Left
17 (lowest)	sequence	,	Left to Right

GLSL Built-in Functions (1)

- Angle and Trigonometry functions
  - ex) radians, degrees, sin, cos, ...
- Exponential functions
  - ex) pow, log, sqrt, ...
- Common functions
  - ex) abs, floor, ceilng, min, max, ...
- Geometric functions
  - ex) length, dot, cross, normalize, ...

GLSL Built-in Functions (2)

- Matrix functions
  - ex) matrixCompMult, transpose, ...
- Vector relational functions
  - ex) lessThan, lessThanEqual, ...
- Texture lookup functions
  - ex) texture1D, texture2D, texture3D, ...
- Fragment processing functions
  - ex) dFdx, dFdy, ...
- Noise functions
  - ex) noise1, noise2, ...

Pointers

- There are no pointers in GLSL
- We can use ‘struct’ which can be copied back from functions
- Because matrices and vectors are basic types they can be passed into and output from GLSL functions

  Ex) mat3 func(mat3 a)

Function calls

- Functions are ‘called by value-return’
  - input arguments are copied into the function at call time
  - output arguments are copied back to the caller before function exit

Operator overloading

- Overloading of vector and matrix types

```
mat4 m;
vec4 b, c, d;
d = m*b; // a column vector as 1D array
c = b*m; // a row vector as 1D array
```

Vector Operations (1)

- Vector operations are component-wise usually

Ex)
```
vec3 u, v, w;
float f;
 ...
v = u + f;
            // v.x = u.x + f;
            // v.y = u.y + f;
            // v.z = u.z + f;
w = u + v;
            // w.x = u.x + v.x;
            // w.y = u.y + v.y;
            // w.z = u.z + v.z;
```

Vector Operations (2)

- The same vector can be a row vector or a column vector according to the usage
  - Ex)

```
vec2 v = vec2(10., 20.);
mat2 m = mat2(1., 2., 3., 4.);
vec2 w = m * v;   // v is a column vector
      // w = vec2(1. * 10. + 3. * 20., 2. * 10. + 4. * 20.)
w = v * m;              // v is a row vector
      // w = vec2(1. * 10. + 2. * 20., 3. * 10. + 4. * 20.)
```

Swizzling and Selection (1)

- Can refer to array elements by element using [ ] or selection (.) operator with
  - x, y, z, w // accessing point/normal vector
  - r, g, b, a // accessing color vector
  - s, t, p, q // accessing texture coord. vector
  - a[2], a.b, a.z, a.p are the same
- Swizzling operator
  - You can use any combination of up to 4 of the letters to create a vector

```
vec4 a;
a.yz = vec2(1.0, 2.0);
a.xwz = vec3(1.0, 2.0, 2.0);
```

Swizzling and Selection (2)

Ex1)
```
vec2 c;
c.x        // is legal
c.z        // is illegal
vec4 pos = vec4(1.0, 2.0, 3.0, 4.0);
vec4 swiz = pos.wzyx;   // swiz = (4.0, 3.0, 2.0, 1.0)
vec4 dup = pos.xxyy;    // dup = (1.0, 1.0, 2.0, 2.0)
```

Ex2)
```
vec4 pos = vec4(1.0, 2.0, 3.0, 4.0);
pos.xw = vec2(5.0, 6.0);  // pos = (5.0, 2.0, 3.0, 6.0)
```

Constructors (1)

- Constructor usage
  - type_name(value1, value2, ...)
- Conversion constructors
  - int(bool) // converts a Boolean value to an int
  - int(float) // converts a float value to an int
  - float(bool) // converts a Boolean value to a float
  - float(int) // converts an integer value to a float
  - bool(float) // converts a float value to a Boolean
  - bool(int) // converts an integer value to a Boolean

Constructors (2)

- Vector constructors

  Ex) vec3 a =vec3(1.0, 2.0, 3.0);
  vec2 b = vec2(a);
  vec3(1.0); // vec3(1.0) == vec3(1.0, 1.0, 1.0)

- Matrix constructors

  Ex) mat2 m = mat2(1.0, 2.0, 3.0, 4.0);
  mat2 a = mat2(vec2(1.0, 2.0), vec2(3.0,4.0));
  mat4 b = mat4(1.0); // 4x4 identity matrix

Constructors (3)

- Structure constructors

  Ex)
```
struct light {
    float intensity;
    vec3 position;
};
light lightVar = light(3.0, vec3(1.0, 2.0, 3.0));
```

- Array constructors

  Ex)
```
const float c[3] = float[3](5.0, 7.2, 1.1);
```

Qualifiers

- used to modify the storage or behavior of global and local variables
- Qualifiers we will mainly use
  - const
    - ex) const vec4 point = vec4(1.0, 2.0, 3.0, 1.0);
  - uniform
  - in, out (attribute, varying)

Uniform Qualifier

- Uniform Qualifier
  - form the linkage between a shader and the application
  - Can be decided the value in application and sent to shaders
  - Cannot be changed in shader
- Used to pass information which is read-only in the shader
  - Ex) bounding box of a primitive,
    translation distance for the primitive,
    rotation angle for the primitive,
    ....

In Qualifiers

- for function parameters passed into a function
- Usages at vertex/fragement shader:
  - vertex shader: 'in'
    - is used to input per-vertex data such as vertex attribute from application
  - fragment shader: 'in'
    - is used to get per-fragment values, typically interpolated from a previous stage's outputs

Out Qualifiers

- for output interface between the declaring shader and the subsequent stages of the OpenGL pipeline

Example of in/out

```
// Vertex Shader
const vec4 red = vec4(1.0, 0.0, 0.0, 1.0);
in vec4 vPosition;
out vec4 color_out;
void main(void)
{
  gl_Position = vPosition;
  color_out = red;
}

// Fragement Shader
in vec4 color_out;
out vec4 FragColor;
void main(void)
{
  FragColor = color_out;
}
```

Attribute/Varying Qualifier

- Included in GLSL 1.2
- Deprecated in GLSL 1.3
- Attribute qualifier
  - linkage between a vertex shader and OpenGL application for transferring per-vertex data
- Varying qualifier
  - linkage between a vertex shader and a fragment shader for interpolated data

Example of attribute/varying– with deprecated features

```
// Vertex Shader
const vec4 red = vec4(1.0, 0.0, 0.0, 1.0);
attribute vec4 vPosition;
varying vec4 color_out;
void main(void)
{
  gl_Position = vPosition;
  color_out = red;
}

// Fragment Shader
varying vec4 color_out;
void main(void)
{
  gl_FragColor = color_out;
}
```

Program Example

- Translation with changing vertex position
- Translation with vertex shader

Usage of Idle Callback

- void myidle()
- glutIdleFunc(myidle);
- glutPostRedisplay();

If we animate with a single Buffer

- We can have broken-up display with pieces of the triangle
- Why this problem happens?
  - Difference of speed of the computation and rendering in display
    - Frame buffer is redisplayed at 60-100 Frames per Second (refresh rate)
    - Application program operates asynchronously and can cause changes to the frame buffer at any time
  - a redisplay of the frame buffer can occur when its contents are still being altered by the application → partially drawn display

Double Buffering for Animation

- The process of writing into the frame buffer is decoupled from the process of reading the frame buffer's contents for display.
- Using two frame buffers
  - Front buffer
    - displaying
  - Back buffer
    - constructing what we would like to display
- After drawing one frame
  - swap the front and back buffers
  - clear the new back buffer and can start drawing into it.

Double Buffer in OpenGL

- in Main function
  - glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);

- in Display Callback
  - glutSwapBuffers();

Program Examples

- Please prepare the program files in the lecture note board
  - LEC11.0_static.c
  - LEC11.1_translate_vertex_position.c
  - LEC11.2_translate_vs.c

HW#11 Triangle Translation with Keyboard Callback (1)

- Due date: This Friday 6:00pm
- Program Specification:
  - Create a window with size 500x500 (title: your student number and name)
  - Draw a triangle with vertices:

    v0: 0.0, 0.0, 0.0, 1.0

    v1: 0.3, 0.0, 0.0, 1.0

    v2: 0.0, 0.3, 0.0, 1.0
  - Initially, the triangle is drawn (without animation) in the window.
  - With step size 0.0001f,
    - Once you type 'w', the triangle continuously translates along +y axis
    - Once you type 's', the triangle continuously translates along –y axis
    - Once you type 'a', the triangle continuously translates along –x axis
    - Once you type 'd', the triangle continuously translates along +x axis
  - Whenever you type 'i', the triangle is drawn at initial position (without animation).

HW#11 Triangle Translation with Keyboard Callback (2)

- You have to use 'uniform' variables to implement this. (Please refer LEC11.2_translate_vs.c)
  - I could implement the program with TWO uniform variables.
- Do not use 'glBegin' and 'glEnd'. – This applies to ALL HOMEWORKS.
- Submit the .c file through LMS