UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Centre de la Imatge i la Tecnologia Multimèdia

GLSL for Beginners

Introduction

- OpenGL Shading Language (GLSL)
  - High-level language similar to C used to write shaders
  - Created by OpenGL ARB (Architecture Review Board)
  - Formally included into OpenGL 2.0 core in 2004
  - Cross-platform compatibility in many OS
    - Windows, Linux, macOS, …
  - Included in the drivers of each hardware vendor
    - Nvidia, AMD (ATI), …

Features

- Syntax similar to C/C++
- Use of the function void main() as a main routine of the program
- Extensions
  - Data types
    - int, float, bool
    - vec2, vec3, vec4 (arrays)
    - mat2, mat3, mat4 (2x2, 3x3 and 4x4 matrices)
    - sampler1D, sampler2D and sampler3D (1D, 2D and 3D textures)
  - Qualifiers
    - attribute, uniform, varying

Features

- **Built-in functions**
  - Maths (abs, max, min, sin, cos, tan…)
  - Geometric (dot product, cross product…)
  - Texture lookup
  - …
- **Differences with C/C++**
  - No automatic conversion of data types
  - There are no pointers, char, double, short, long…
  - Function's parameters: all of them are passed by value
    - in → Input parameters
    - out → Output parameters
    - inout → I/O parameters

Data Types

- **Data types shared with C/C++**
  - **int** `int numTextures = 4;`
  - **float** `float a = 2.4e5;`
  - **bool** `bool found = true;`
- **Others**
  - **Vectors of 2, 3, and 4 components of float, int or bool**

    vec2, vec3, vec4 ivec2, ivec3, ivec4 bvec2, bvec3, bvec4
  - **Vectors can represent**
    - **Points/Vectors: access to components via .x, .y, .z, .w**
    - **Colors: access to channels via .r, .g, .b, .a**
    - **Texture coord: access to coords via .s, .t, .p, .q**


Data Types

- **Matrices**
  - **2x2, 3x3 and 4x4 of float**

mat2, mat3, mat4

- **Example**

```
mat4 transform;
vec4 col = transform[2];              // 3rd column of the matrix
float v = transform[column][row]
```


Data Types

- **Samplers**
  - **Represent 1D, 2D and 3D textures**
    - **sampler1D, sampler2D, sampler3D**
    - **samplerCube → cube-mapping texture**
    - **sampler2DShadow → shadow-mapping texture**

- **Example**

```
uniform sampler2D myTexture;

vec4 color = texture2D(myTexture, gl_TexCoord[0].st);
```

Data Types

- **Structs**
  - **Behaviour similar to C/C++**

- **Example**

```
struct MyLight
{
        vec3 position;
        vec3 color;
};

MyLight light0;
```



Data Types

- **Arrays**
  - **Arrays can be defined for any data type**
  - **When passing arrays as parameters, it behaves in the same way as copying the whole array**

- **Example**

```
struct MyLight
{
        vec3 position;
        vec3 color;
};

MyLight lights[2] = MyLight[2](vec3(1.0,1.0,1.0), vec3(1.0,0.0,0.0));
```


Variables

- **Except from attribute, uniform and varying variables, they can be initialized**

```
float b = 2.6;

vec4 v = vec4(1.0,2.0,3.0,4.0);

MyLight l = MyLight(v, v);
```

- **Matrices are initialized with column's values**

```
Mat2 m = mat2(1.0,2.0,3.0,4.0);
```

$$m = \begin{bmatrix} 1.0 & 3.0 \\ 2.0 & 4.0 \end{bmatrix}$$


Variable Qualifiers

- **attribute**
  - Passed from App → VP (value may vary per vertex)
  - They are global variables and read-only
  - They can be float, vec2, vec3, vec4, mat2, mat3 or mat4
- **uniform**
  - Passed from App → VP/FP (value may vary per primitive)
  - They are global variables and read-only
  - They can be of any type
- **varying**
  - Passed from VP → FP
  - They are global, output vars. for the VP and read-only for the FP
- **const**
  - Read-only constant value used in the VP/FP
- **Without qualifier**
  - Global or local value that can be read or written
  - "Lifetime" limited to the execution of the VP/FP

Flow Control

- **Similar to C/C++**
  - **Conditions**
    - **if, if…else, if…else if…else**
  - **Loops**
    - **while, for**
  - **Loops control**
    - **break, continue**
  - **discard → used to discard the fragment (to avoid updating the frame buffer)**

Operators

- **Similar to C/C++**
  - **Arithmetic, relational, logic operators…**
- **Others**
  - **Swizzling “.”**
    - **To access the elements of a vector**
    - **It can be used to access the elements in a specific order**

```
vec4 v;
v.rgba;                    // vec4
v.rgb;                     // vec3
v.xy;                 // vec2
v.abgr;                    // vec4, different order
v.xy = vec2(2.0,3.0);                // only change x and y
```

Operators

- **When operators are applied to vectors, they are applied to its components individually**

```
vec4 u, v, w;

float a;

w = u + v;          // Sum of 2 vectors

v = a + u;          // Sum the value a to each component

w++;            // Increase all the components of w
```


Functions

- Defined in a similar way to C/C++
- The difference lies in the qualifiers of the parameters
  - in: input parameter (passed by value)
    - Copy in but don't copy back out; still writable within the function
  - out: output parameter(return by value)
    - Only copy out; readable but undefined at entry to function
  - inout: input/output parameter (passed and returned by value)
    - Copy in and copy out

Functions

- **Example**

```
void computeCoord(in vec3 normal, inout vec3 coord)
{
    coord = coord + normal;
}

vec3 computeCoord(in vec3 normal, in vec3 coord)
{
    return coord + normal;
}
```

Additional Information

- **The API with the complete specification of types, functions, … can be downloaded from the WebGL 2.0 reference card (or OpenGL reference cards)**

https://www.khronos.org/developers/reference-cards/

Questions?

www.citm.upc.edu