POLITECNICO
MILANO 1863

GLSL ES Syntax

Computer Graphics 2021

Erica Stella (erica.stella@polimi.it)

+++VIEW BAIT VERY IMPORTANT SLIDE+++

- This pack of slides serves as introduction to the syntax of GLSL ES
- GLSL ES syntax is also summarised in the attached pdf from pag. 6 (webgl20-reference-guide.pdf)
- DON'T WORRY if you don't understand how the code snippets work overall, we will deal with that during the classes, just **focus on learning the syntax**

GLSL ES: variable types

The accepted basic ***variable types*** are:

void	no function return value or empty parameter list
bool	Boolean
int	signed integer
float	floating scalar
vec2, vec3, vec4	n-component floating point vector
bvec2, bvec3, bvec4	Boolean vector
ivec2, ivec3, ivec4	signed integer vector
mat2, mat3, mat4	2x2, 3x3, 4x4 float matrix
sampler2D	access a 2D **texture**
samplerCube	access cube mapped **texture**

```
#version 300 es
//global variables
in vec4 a_position;
in vec2 a_uv;
uniform float u_del_uv;
uniform mat4 u_mvp;
out vec2 uv_fs;

void main() {
  //vec2(..) creates a new vec2
  uv_fs = vec2(a_uv.x + u_del_uv, a_uv.y);
  vec3 n_Light_Dir; //local variable
  […]
  gl_Position = u_mvp * a_position;
}
```

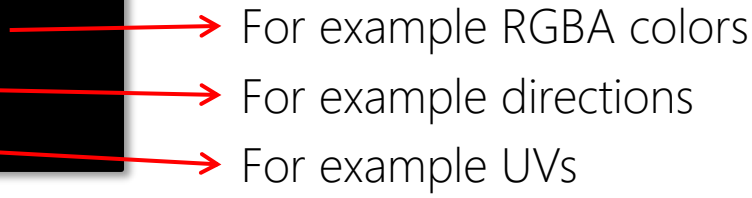
There are **no pointer types** as well as no ***new*** statements

GLSL ES Vectors

Since vectors are often used, let's see them into more details:

To define a vector of constants, it is possible to specify its values in a number matching with the vector's type.

```
vec4 light = vec4(1.0, 0.9, 0.5, 1.0);
vec3 nDir  = vec3(0.8, -0.3, -0.1);
vec2 uv    = vec2(0.1, 1.0);
```



It is possible to extend a vector adding extra elements. This is done by inserting the shorter vector in the constructor of the longer one…

```
vec3 n_Light_Dir; //(0.3, 0.5, 0.7)
float light_Color; //0.4
…
return vec4(n_Light_Dir, light_Color); //(0.3, 0.5, 0.7, 0.4)
```

… this property can be exploited, for example, to return more values from a function

GLSL ES Vectors

Single **vectors** elements can be extracted or referred to using field notations with **xyzw**, **rgba** or **stpq** field names.

```
vec4 light = vec4(1.0, 0.9, 0.5, 1.0);
```

These are equivalent

```
light.x=      light.r=      light.s       = 1.0
light.y=      light.g=      light.t       = 0.9
light.z=      light.b=      light.p       = 0.5
light.w=      light.a=      light.q       = 1.0
```

More than one letter can be used to refer to more elements, example

```
vec3 l1 = light.xyz;
vec2 l2 = light.rb;
```

GLSL ES Functions

GLSL **functions** are defined in a way similar to ***C***.

- We have the **`return`** type. It can be `void` in case we are defining a procedure.
- Then the function name, followed by typed function parameters
- The function body is held between { } brackets.
- A function can have local variables.

```
vec4 light_model(int light_type, vec3 position){

    vec3 n_Light_Dir;
    float light_Color;

        […]

    return vec4(n_Light_Dir, light_Color);
}
```

Precision qualifier

It is also possible to define the precision for *int* and *float* and *samplers* variables using a **precision qualifier**.

Qualifiers introduced in the OpenGL ES (do not exist in standard OpenGL specification) to increase shader efficiency and decrease memory requirements.

Precision Qualifiers	Descriptions	Default Range and precision	
		Float	int
highp	High precision. The minimum precision required for a vertex shader (default there)	$(-2^{62}, 2^{62})$	$(-2^{16}, 2^{16})$
mediump	Medium precision. Minimum precision in FS	$(-2^{14}, 2^{14})$	$(-2^{10}, 2^{10})$
lowp	Low precision. Still able to represent all colors.	$(-2, 2)$	$(-2^{8}, 2^{8})$

Precision qualifier

A precision qualifier can be added **before the type specification**, affecting only the variable.

```
mediump     vec3   col1;
lowp        int    var2;
higp        mat4   pMatrix;
```

Or stated for **ALL variables of a type** by using the **`precision`** statement at the beginning of a shader code

```
precision mediump float;

out vec4 outColor;

void main() {
   outColor = vec4(0.0,0.0,1.0, 1);
}
```

fragment shader code

Special Variables

Listed here for the sake of completeness, we will see what they mean during the class

Shader programs use **Special Variables** (global) to communicate with fixed-function parts of the pipeline.

Built-in, so no need to declare them.

vec4 gl_Position; (VS) Final transformed vertex position, computed in clip space coordinates.

vec4 gl_FragColor; (FS) Final fragment color output, in RGBA (from WebGL2 can be avoided)

```
#version 300 es
in vec4 a_position;

void main() {
    gl_Position = a_position;
}
```

Vertex shader

```
#version 300 es
precision mediump float;

out vec4 outColor;

void main() {
    gl_FragColor = vec4(0.0,0.0,1.0, 1);
    // preferred version
    outColor = vec4(0.0,0.0,1.0, 1);
}
```

Fragment shader