GLSL

<a.soroa@ehu.eus>

EHU

1 Introducción

2 Sintaxis

3 Tipos de datos

4 Funciones

GLSL

- GLSL (*OpenGL Shading Language*): lenguaje de programación para escribir *shaders*.
  - Se utiliza para los shaders de vértices y fragmentos.
- Es parecido al lenguaje *C*, pero más sencillo.
  - Sintaxis parecida a *C*, mismas operaciones de control de flujo del programa.
  - Es más estricto que *C* en los tipos de datos.
  - Calificadores de tipo en vez de lectura y escritura para el manejo de la entrada y la salida.

GLSL: Hello world

Ejemplo de vertex shader:

```
// variables uniform (entrada)
uniform   vec3 color;
uniform   mat4 modelToClipMatrix;

/* atributos (entrada) */
attribute vec3 position;

// variables varying (salida)
varying   vec3 vColor;

void main() {
  vec4 position4 = vec4( position, 1.0);
  gl_Position = modelToClipMatrix * position4; // necesario
  vColor = color;
}
```

GLSL: Hello world

Ejemplo de fragment shader:

```
// variables varying (entrada)
varying   vec3 vColor;

void main() {
  gl_FragColor = vec4(vColor, 1.0); // necesario
}
```

Comentarios

- Los comentarios están delimitados por `'/*'` y `'*/'` o por `'//'` y una nueva línea.

```
/* Un comentario */
vec2 pos = vec2(1.0, 2.0);
pos += vec2(2.0, 2.0); // otro comentario
```

Tipos de datos

- escalares:
  - `float, int, bool`
- vectores (float, int, boolean):
  - `vec2, vec3, vec4`
  - `ivec2, ivec3, ivec4`
  - `bvec2, bvec3, bvec4`
- matrices (float):
  - `mat2, mat3, mat4`
- texturas:
  - `sampler2D, samplerCube, sampler2DShadow, ...`

Tipos de datos

GLSL data type	C data type	Description
bool	int	Conditional type, taking on values of **true** or **false**.
int	int	Signed integer.
float	float	Single floating-point scalar.
vec2	float [2]	Two component floating-point vector.
vect3	float [3]	Three component floating-point vector.
vec4	float [4]	Four component floating-point vector.
bvec2	int [2]	Two component Boolean vector.
bvec3	int [3]	Three component Boolean vector.
bvec4	int [4]	Four component Boolean vector.
ivec2	int [2]	Two component signed integer vector.
ivec3	int [3]	Three component signed integer vector.
ivec4	int [4]	Four component signed integer vector.
mat2	float [4]	2×2 floating-point matrix.
mat3	float [9]	3×3 floating-point matrix.
mat4	float [16]	4×4 floating-point matrix.
sampler1D	int	Handle for accessing a 1D texture.
sampler2D	int	Handle for accessing a 2D texture.
sampler3D	int	Handle for accessing a 3D texture.
samplerCube	int	Handle for accessing a cubemap texture.
sampler1DShadow	int	Handle for accessing a 1D depth texture with comparison.
sampler2DShadow	int	Handle for accessing a 2D depth texture with comparison.

Tipos de datos

Example

```
float f = -0.5;
int i = 5;
vec3 pos = vec3(1.0, 0.5, -0.5);        = [1.0, 0.5, -0.5]
ivec4 v = ivec4(5);                     = [5, 5, 5, 5]
mat2 m = mat2(1.0, 2.0, 3.0, 4.0);      = [1.0 3.0
                                           2.0 4.0]
```

Escalares

Example

```
float f = 5.0;
int i = 5;
float f = 5;              ERROR
float f = float(5);       OK
float f = 5.0;            OK
bool b = true;
```

Matrices

- `mat2`, `mat3`, `mat4`

Example

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

```
mat4 identity = mat4(1.0);
```
= [1.0 0.0 0.0 0.0
0.0 1.0 0.0 0.0
0.0 0.0 1.0 0.0
0.0 0.0 0.0 1.0]

Matrices: Acceso a elementos

- Los índices comienzan en cero
- Si tenemos una matriz `m`:

  `m[1]`: segunda columna
  `m[1][0]`: primer elemento de la segunda columna

Example

```
mat2 m = mat2(1.0, 2.0, 3.0, 4.0);      = [1.0 3.0
                                           2.0 4.0]
vec2 v2 = m[1];                         = [3.0, 4.0]
float f = m[1][0];                      = 3.0
```

Matrices: Multiplicación

- Soportadas nativamente (eficientes)
- Matriz por vector, vector por matriz, matriz por matriz
- Si las dimensiones no concuerdan: error de compilación

Matrices: Multiplicación

- Matriz por matriz

Example

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

```
mat2 m2 = mat2(0.5, 0.6, 0.7, 0.8);
```
= [0.5 0.7
0.6 0.8]

```
mat2 r = m * m2;
```
= [2.3 3.1
3.4 4.6]

```
mat2 r2 = m2 * m;
```
= [1.9 4.3
2.2 5.0]

Vectores

- Tipo fundamental en GLSL:
  - coordenadas
  - colores
  - ...

Example

```
vec2 uv = vec2(0.7, 0.2);          = [0.7, 0.2]
vec3 pos = vec3(1.0, 0.5, -0.5);   = [1.0, 0.5, -0.5]
ivec3 iv = ivec3(1, 5, -5);        = [1, 5, -5]
vec2 v2 = vec2(1, 2.0);            // error "1" es entero
vec2 v2 = vec2(1.0, 2.0);          // OK
```

Acceso a elementos

- Las componentes de un vector se acceden mediante:
  - índice: `[índice]` (comenzando en cero)
  - color: `.rgba`
  - posición: `.xyzw`
  - coordenadas de textura: `.stpq`

Acceso a los elementos

Example

```
vec4 baz;
baz.rgba;   // es lo mismo que baz
baz.xy;     // es un vec2
baz.st;     // igual que el anterior
baz.b;      // es un solo float
baz[2];     // es lo mismo que baz.b
baz.xb;     // ILEGAL (por mezclar xyzw con rgba)
baz.xxx;    // es un vec3
```

Acceso a los elementos

Example

```
vec4 vecA = vec4(1.0, 2.0, 3.0, 4.0);      = [1.0, 2.0, 3.0, 4.0]
vec2 vecB = vecA.xz;                       = [1.0, 3.0]
vec3 vecC = vecA.brb;                      = [3.0, 1.0, 3.0]
vec4 Acev = vecA.wzyx;                     = [4.0, 3.0, 2.0, 1.0]
vec4 Bdup = vecB.xxyy;                     = [1.0, 1.0, 3.0, 3.0]
```

Acceso a los elementos. Asignación.

Example

```
vec4 pos = vec4(1.0, 2.0, 3.0, 4.0);      = [1.0, 2.0, 3.0, 4.0]
pos.xw = vec2(5.0, 6.0);                  = [5.0, 2.0, 3.0, 6.0]
pos.wx = vec2(7.0, 8.0);                  = [8.0, 2.0, 3.0, 7.0]
pos.xx = vec2(3.0, 4.0);                  //ILEGAL: x usado
                                          //dos veces
```

Operaciones

Example

```
vec3 v, u;
vec3 a = v * u;          a = [v.x*u.x; v.y*u.y; v.z*u.z]
float d = dot(u,v);      producto escalar
vec3 a = cross(u,v);     producto vectorial
```

Multiplicación

- Matriz por vector

Example

```
vec2 pos = vec3(1.0, 0.5);                  = [1.0, 0.5]
mat2 m = mat2(1.0, 2.0, 3.0, 4.0);          = [1.0 3.0
                                               2.0 4.0]
vec2 r = m * pos;                           = [2.5, 4.0]
```

$$\texttt{r} = \begin{pmatrix} 1 & 3 \\ 2 & 4 \end{pmatrix} \begin{pmatrix} 1 \\ 0.5 \end{pmatrix} = \begin{pmatrix} 2.5 \\ 4.0 \end{pmatrix}$$

Multiplicación

- Vector por matriz

Example

```
vec2 pos = vec3(1.0, 0.5);                  = [1.0, 0.5]
mat2 m = mat2(1.0, 2.0, 3.0, 4.0);          = [1.0 3.0
                                               2.0 4.0]
vec2 r = pos * m;                           = [2.0, 5.0]
```

$$\texttt{r} = \begin{pmatrix} 1 & 0.5 \end{pmatrix} \begin{pmatrix} 1 & 3 \\ 2 & 4 \end{pmatrix} = \begin{pmatrix} 2.0 & 5.0 \end{pmatrix}$$

Multiplicación: adaptando dimensiones

Example

```
vec2 pos = vec2(1.0, 0.5);
mat3 m = ...
vec3 r = m * pos;
vec3 r = m * vec3(pos, 1.0);
vec2 r2 = m * vec3(pos, 1.0);
vec2 r2 = (m * vec3(pos, 1.0)).xy;
```

Vector dimensión 2
Matrix (3×3)
Error: $(3 \times 3)(2 \times 1)$
OK: $(3 \times 3)(3 \times 1)$
Error: el resultado es (3×1)
OK

Variables tipo *sampler*

- Variables especiales que acceden a la información de una textura (*texel*)
- La textura puede ser 1D, 2D o 3D
- Estas variables son las entradas a funciones como `texture2D`, que obtienen la información requerida.

Example

```
uniform sampler2D sampler;
vec2 uv = vec2(0.5,0.5);
vec3 texel = texture2D(sampler, uv);
```

Cualificadores

- Cualificadores especiales que necesitan enviarse junto con las variables cuando se establece la comunicación entre OpenGL y GLSL o entre un vertex shader y un fragment shader.
  - `attribute`
  - `uniform`
  - `varying`

Cualificadores: attribute

- Sólo para el *vertex shader*
- Datos que la aplicación envía al *vertex shader* **por cada vértice**. Por ejemplo:
  - Posición
  - Normal
  - Coordenada de textura
- Sólo lectura.
- Escalarres, vectores y matrices
  - Normalmente vectores

Example

```
attribute vec3 position;
attribute vec3 normal;
attribute vec2 texCoord;
```

Cualificadores: uniform

- Variables qye la aplicación envía al *vertex shader* y al *fragment shader*.
- Sólo lectura.
- Por ejemplo:
  - matriz *modelview*
  - matriz de proyección
  - información del material (color, ...)
  - información de luces (posición, color, ...)

Example

```
uniform vec3 color;
uniform mat4 modelToClipMatrix;
```

Cualificadores: varying

- Variables que el *vertex shader* envía al *fragment shader*
  - la salida del *vertex shader* es por cada vértice
  - la entrada del *fragment shader* es por cada fragmento
- Se produce una interpolación (*varying*)

Cualificadores: varying

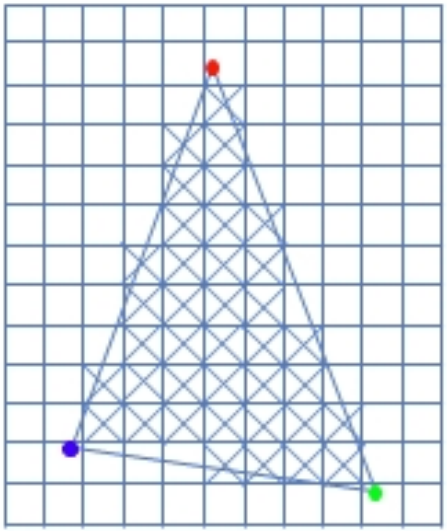
Ejemplo de vertex shader:

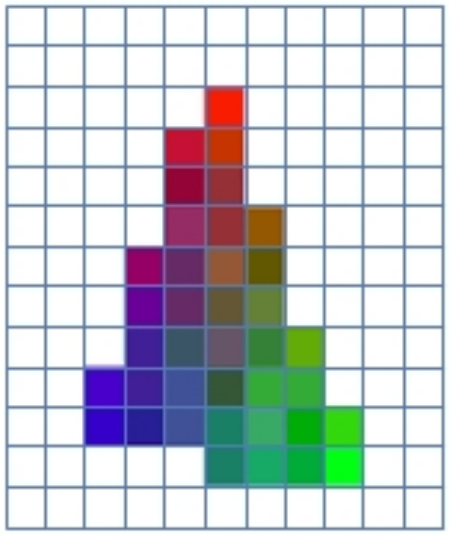
```
// color como atributo
attribute vec3 color;

// variables varying (salida)
varying   vec3 vColor;

void main() {
  gl_Position = ...;
  vColor = color;
}
```

```
// variables varying (entrada)
varying   vec3 vColor;
void main() {
  gl_FragColor = vec4(vColor, 1.0);
}
```





Variables especiales

- Deben ser asignadas con un valor
  - Si no, error de compilación
- *Vertex shader*: `gl_Position (vec4)`
  - Posición del vértice en *Clip coordinates*
- *Fragment shader*: `gl_FragColor (vec4)`
  - Color RGBA del fragmento

Hello world de nuevo

vertex shader:

```
// variables uniform (entrada)
uniform   vec3 color;
uniform   mat4 modelViewMatrix;
uniform   mat4 projectionMatrixx;

/* atributos (entrada) */
attribute vec3 position;

// variables varying (salida)
varying   vec3 vColor;

void main() {
  gl_Position = projectionMatrix * modelViewMatrix * vec4(position, 1.0);
  vColor = color;
}
```

Hello world de nuevo

fragment shader:

```
// variables varying (entrada)
varying   vec3 vColor;

void main() {
  gl_FragColor = vec4(vColor, 1.0); // necesario
}
```

Bucles y condicionales

- Se permiten bucles al estilo de C:
  - `for`, `while`, `do-while`, `if`, `if-else`, `?:` (selección), `continue`, `break`, `return`, `return <expresion>`, `discard`
- Especial: `discard`
  - En los *fragment shader*, indica que el fragmento debe ser descartado.

Funciones GLSL

- GLSL tiene muchas funciones implementadas
- Conviene utilizarlas (eficiencia)
- Generales:
  - `abs`, `sign`, `floor`, `ceil`, `fract`, `mod`, **`min`**, **`max`**, `clamp`, `mix`, `step`, `smoothstep`, ...
- Trigonométricas:
  - `radians`, `degrees`, `sin`, `cos`, `tan`, `asin`, `acos`, `atan`, ...)
- Exponenciales:
  - `pow`, `exp`, `log`, `exp2`, `log2`, `sqrt`, `inversesqrt`, ...
- Geométricass:
  - `length`, **`distance`**, **`dot`**, `cross`, **`normalize`**, `faceforward`, `reflect`, `refract`, `ftransform`, ...

Funciones GLSL

- Entre matrices:
  - `lessThan`, `lessThanEqual`, `greaterThan`, `greaterThanEqual`, `equal`, `notEqual`, `not`, `all`, `any`, ...
- Funciones de derivadas para fragmentos:
  - `dFdx`, `dFdy`, `fwidth`
- Acceso a texturas:
  - `texture1D`, `texture1DProj`, `texture1DLod`, `texture1DProjLod`
  - `texture2D`, ...
  - `texture3D`, ...
  - `textureCube`, `textureCubeLod`
  - `shadow1D`, `shadow2D`, `shadow1DProj`, `shadow2DProj`, `shadow1DLod`, `shadow2DLod`, `shadow1DProjLod`, `shadow2DProjLod`
- Funciones de ruido:
  - `noise1`, `noise2`, `noise3`, `noise4`