Linguaggio GLSL

GLSL Sintassi

- Per scrivere uno shader si utilizzano linguaggi di shading C-like.
- I linguaggi di shading comuni
  - GLSL (OpenGL Shading Language)
  - HLSL (High Level Shader Language) (Directx)

GLSL Data Types

Tipi scalari: float, int, bool

Tipi vettori: vec2, vec3, vec4 (vettori di 2, 3 o 4 float)
ivec2, ivec3, ivec4 (vettori di 2,3,o 4 interi)
bvec2, bvec3, bvec4 (vettori di 2,3 o 4 boolenai)

Tipi matrice: mat2, mat3, mat4

Texture sampling: sampler1D, sampler2D,
sampler3D, samplerCube

GLSL Sintassi

- Tipi di dati - C-like
  - int
  - uint
  - float
  - double
  - bool (vero o falso)
  - struct

GLSL Sintassi

- Tipi di dati - vettore
  - vec {2, 3, 4} : Un vettore di 2, 3 o 4 float
  - dvec {2, 3, 4} : Per i double
  - bvec {2, 3, 4} : Per I booleani
  - ivec {2, 3, 4} : Per gli interi
  - uvec {2, 3, 4} : Per gli interi senza segno.

GLSL Sintassi

- Tipi di dati - vettore
  - Vector è una classe
  - È possibile accedere a un vettore per
    - .x, .y, .z, .w posizione o direzione
    - .r, .g, .b, .a colore
    - .s, .t, .p, .q coordinate di texture
      - color.rgb OK
      - color.xgb ERRATO

GLSL Syntax: Vettori

Costruttori

```
vec3  xyz  = vec3(1.0, 2.0, 3.0);
vec3  xyz  = vec3(1.0); // [1.0, 1.0, 1.0]
vec3  xyz  = vec3(vec2(1.0, 2.0), 3.0);
vec2  v =  vec2(1.0, 2.0);          // composto da valori
v = vec2(3.0, 4.0);
vec4   u = vec4(0.0); // inizializza tutti gli elementi a zero
vec2   t = vec2(u);  // prende le prime due componenti di u
vec3   vc = vec3(v, 1.0); // composto da un  vec2 ed un float
```

Swizzling e selezione

- Si accede alle componenti di un vettore usando [] oppure:
  - .x, .y, .z, .w
  - .r, .g, .b, .a
  - s, t, p, q
  - a[2] equivale ad a.b oppure a.z, oppure a.p
- L'operatore swizzling permette una selezione multipla di componenti da tipi vettore:

```
- vec4 a;
- a.yz = vec2(1.0, 2.0);
- a.xy = a.yx;    /* scambia di posto gli elementi*/
```

GLSL Vettori

- Swizzle: seleziona o ridefinisce elementi

```
vec4 c = vec4(0.5, 1.0, 0.8, 1.0);

vec3  rgb  = c.rgb;    // [0.5,  1.0,  0.8]
vec3  bgr  = c.bgr;    // [0.8,  1.0,  0.5]

vec3  rrr  = c.rrr;    // [0.5,  0.5,  0.5]

c.a = 0.5;             // [0.5,  1.0,  0.8,  0.5]
c.rb = 0.0;            // [0.0,  1.0,  0.0,  0.5]
```

GLSL Sintassi

- Tipi di dati - matrice
  - memorizzazione column-major (memorizzazione per colonne)
  - Tutti i tipi di matrice sono in virgola mobile,
    - mat*n* : Una matrice quadrata con *n* colonne e *n* righe
    - mat*nxm* : una matrice rettangolare con *n* colonne e *m* righe *n* = *m* = {2, 3, 4}

    Nella memorizzazione per colonne la matrice
    - | 1 5 9 13 |
    - | 2 6 10 14 |
    - | 3 7 11 15 |
    - | 4 8 12 16 |

    Viene memorizzata in memoria come:

    1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16

Questa convenzione è utilizzata in **OpenGL e GLSL** .

Questo rende più efficiente la comunicazione tra il codice della CPU (scritto in C/C++)
e il codice della GPU (scritto in GLSL) quando si utilizzano matrici con OpenGL.

Tuttavia, è importante notare che non tutte le librerie o linguaggi utilizzano questa convenzione. Alcuni utilizzano una memorizzazione "row major", dove gli elementi sono memorizzati per righe anziché per colonne. Pertanto, quando si lavora con dati di matrici tra diverse librerie o linguaggi, è importante essere consapevoli della convenzione di memorizzazione utilizzata per evitare errori di interpretazione dei dati.

GLSL Syntassi: Matrici

- Costruttori

```
mat3 i = mat3(1.0); // matrice identità 3x3

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

$$m = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$$

- Accesso agli elementi

```
float f = m[column][row];

float x = m[0].x; // x è la prima componente della prima colonna

vec2 yz = m[1].yz; // yz sono componenti della seconda
colonna
```

Tratta le matrici come array di vettori colonna

swizzling

GLSL Syntax: Vettori e Matrici

- Le operazioni tra matrici e vettori sono facili e veloci

```
vec3 xyz = // ...

vec3  v0  =  2.0 * xyz;   //  scale
vec3  v1  =  v0 + xyz;    //  component-wise
vec3  v2  =  v0 * xyz;    //  component-wise
```

```
 mat3 m = // ..
             .
 mat3 v = // ..
             .
mat3   mv =   v  * m;          // matrice  *  matrice

vec3   xyz2   =  mv * xyz; //    matrice  *  vettore
```

GLSL Sintassi

- Tipi di dati - sampler (texture)
  - sampler {1, 2, 3} D : {1, 2, 3} D texture
  - samplerCube : Per cubo texture map cube.
  - Esempio:
    - uniform sampler2D colorTexture;
    - vec4 texelColor = texture (colorTexture, textureCoord);

GLSL Sintassi

- funzioni
  - No ricorsione in GLSL
  - parametri
    - Proprio come C, ma con alcuni qualifier
    - in : variabile di input
    - out : variabile di output

GLSL Built-in Functions

- Selezionate funzioni trigonometriche

```
float  s = sin(theta);
float  c = cos(theta);
float  t = tan(theta);
float theta = asin(s);
// ...
```

Gli angoli sono misurati in radianti

```
vec3 angles = vec3(/* ... */);
vec3 vs = sin(angles);
```

Lavora sui vettori in modalità component-wise

GLSL Built-in Functions

- Funzioni esponenziale

```
float     xToTheY =        pow(x, y);
float     eToTheX =        exp(x);
float     twoToTheX        = exp2(x);

float      l = log(x);   // ln
float      l2 = log2(x); // log2

float     s = sqrt(x);

float is = inversesqrt(x);
```

GLSL Built-in Functions

- Funzioni comuni

```
float   ax  =  abs(x);     // absolute value
float   sx  =  sign(x); // -1.0, 0.0, 1.0

float   m0  =  min(x, y); // minimum value
float   m1  =  max(x, y); // maximum value

float c = clamp(x, 0.0, 1.0);

// altre: floor(), ceil(),

// step(), smoothstep(), ...
```

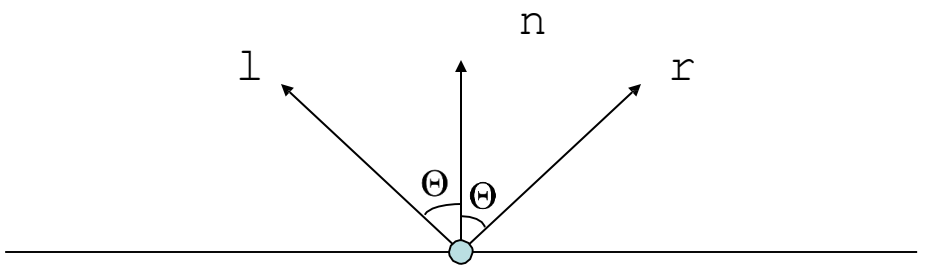
GLSL Built-in Functions

- Funzioni geometriche

```
vec3        l    =    //      ...
vec3        n    =    //      ...
vec3        p    =    //      ...
vec3        q    =    //      ...

float f = length(l);        // lunghezza di un vettore
float d =distance(p,  q); // distanza tra punti

float  d2 = dot(l, n);      // prodotto scalare
vec3   v2 = cross(l, n);    // prodotto vettoriale
vec3   v3 = normalize(l);   // normalizzazione di un vettore


vec3 v3 = reflect(l, n);   // reflect
```

GLSL Built-in Functions

- `reflect(-l, n)`
  - Dati l ed n, trova r, vettore riflesso di l rispetto ad n



GLSL Sintassi

- Funzioni built-in

Function Syntax	Description
float **length**(*TYPE x*)	Returns the length of vector *x*: **sqrt** ($x[0] \cdot x[0] + x[1] \cdot x[1] + \ldots$)
float **distance**(*TYPE p0*, *TYPE p1*)	Returns the distance between *p0* and *p1*: **length** (*p0* – *p1*)
float **dot**(*TYPE x*, *TYPE y*)	Returns the dot product of *x* and *y*: result = $x[0] \cdot y[0] + x[1] \cdot y[1] + \ldots$
vec3 **cross**(vec3 *x*, vec3 *y*)	Returns the cross product of x and y, i.e.,
TYPE **normalize**(*TYPE x*)	Returns a vector in the same direction as *x* but with a length of 1.
vec4 **ftransform**()	Returns the transformed input vertex such that it matches the output of the fixed-function vertex pipeline.
TYPE **reflect**(*TYPE I*, *TYPE N*)	Returns the reflection direction for incident vector I, given the normalized surface orientation vector *N*: result = $I - 2 \cdot$ **dot**(N, I) $\cdot N$

Flow Control

- if
- if else
- expression ? true-expression : false-expression
- while, do while
- for

Semplice Vertex Shader

```
in vec4 vPosition;
in vec4 vColor;
out vec4 color;
uniform mat4 Model;
uniform mat4 View;
uniform mat4 Projection;

 void main()
 {
 gl_Position =Projection*View*Model*vPosition
 color=vColor
 }
```

in: input dello shader che varia in base all'attributo del vertice

out: output dello shader

uniform: input dello shader che è costante lungo la chiamata `glDraw`

Un banale vertex shader che trasforma ogni vertice secondo la matrice Model View e Projection. Ogni esecuzione di glDrawArray() invoca lo shader con nuovi valori di vertice

Semplice Fragment Shader

```
in vec4 color;
out vec4 FragColor;

void main()
{
    FragColor = color;
}
```

in: generato nel rasterizer interpolando i colori dei vertici della primitva

Eseguito dopo il rasterizzatore, e quindi opera su ogni frammento di ogni primitiva visualizzata

ogni frammento è stato generato dal rasterizzatore, che è un built-in, non programmabile

Qualifiers delle variabili

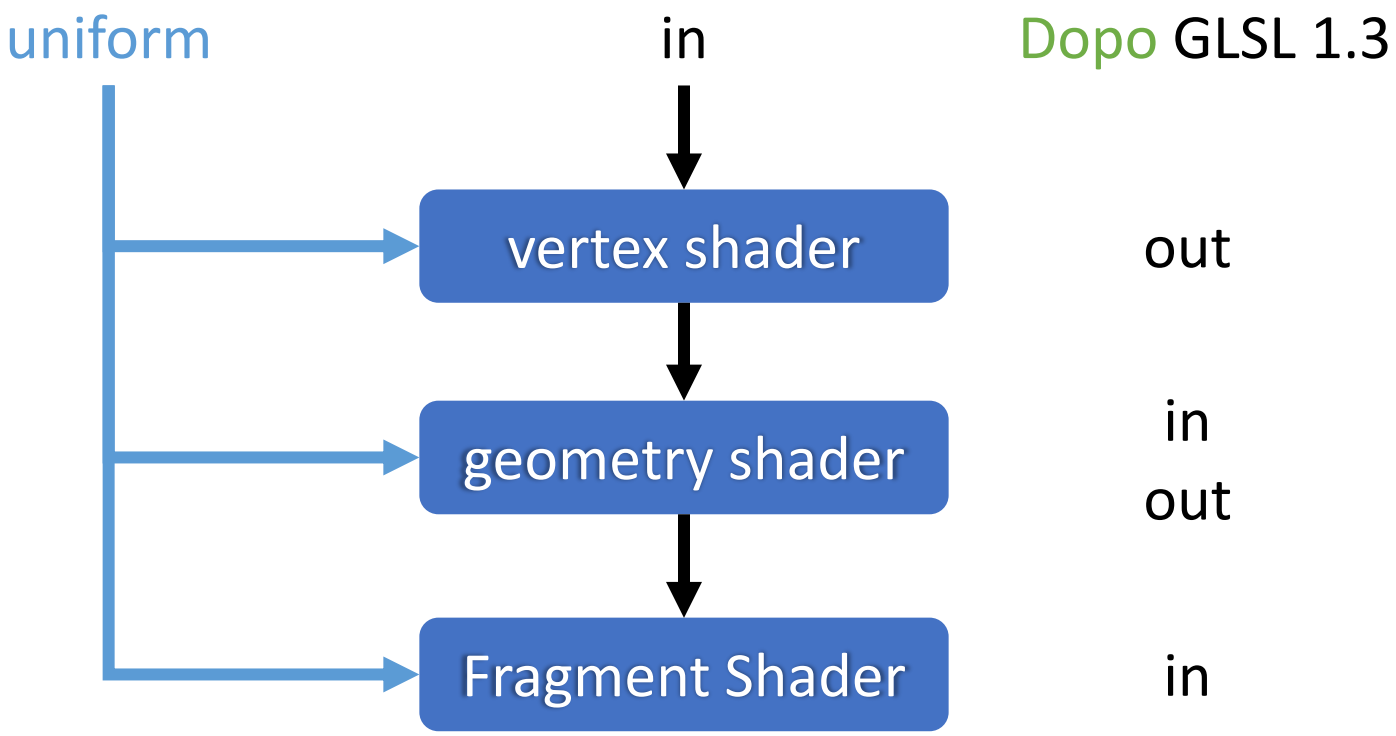
- Il qualificatore dà un significato speciale alla variabile.

- `in, out`

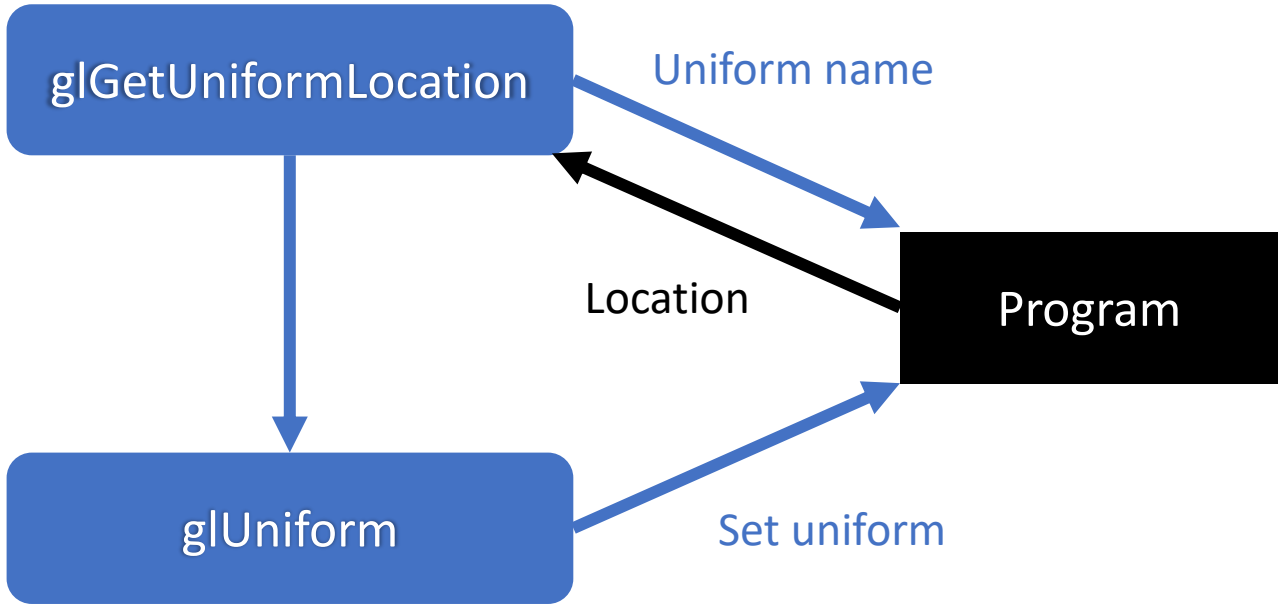
  Attributi di vertici input ed output degli shader, che **variano per ogni chiamata** di glDrawArrays

```
in      vec2   tex_coord;
out     vec4   color;
```

- `uniform` – Variabili globali il cui valore **rimane costante rispetto ad una chiamata di glDrawArrays, (cioè non cambia durante il rendering di una primitiva), che vengono passate dall'applicazione OpenGL agli shaders.**
  - Vengono utilizzate per condividere i dati tra un programma applicativo, vertex shader e fragment shader. Le variabili di tipo uniform possono essere lette ma non modificate dal vertex o fragment shaders.


uniform
in
Dopo GLSL 1.3
vertex shader
out
geometry shader
in
out
Fragment Shader
in

- Settare variabili uniform per gli shader



Come inizializzare i valori delle variabili uniformi nell'applicazione

Otteniamo i puntatori alle variabili uniform presenti all'interno dello ShaderProgram

```
GLint location = glGetUniformLocation(ShaderProgram, "name");
```

Successivamente le variabili uniformi vengono inizializzate con i loro valori usando i comandi

void **glUniformMatrix4fv**(	GLint *location*,
	GLsizei *count*,
	GLboolean *transpose*,
	const GLfloat **value*);

void **glUniform3f**(	GLint *location*,
	GLfloat *v0*,
	GLfloat *v1*,
	GLfloat *v2*);

glUniform — Specifica il valore da assegnare ad una variabile uniforme per il program corrente

È possibile assegnare il valore ad una variabile uniforme, nel caso si tratti di uno scalare (1f), un vettore di 2(2f), 3 o 4 componenti, specificandole una per una

void **glUniform1f**(	GLint *location*,
	GLfloat *v0*);

void **glUniform2f**(	GLint *location*,
	GLfloat *v0*,
	GLfloat *v1*);

void **glUniform3f**(	GLint *location*,
	GLfloat *v0*,
	GLfloat *v1*,
	GLfloat *v2*);

Oppure assegnando il valore tramite vettori, di una 2, 3 o 4 componenti

void **glUniform1fv**(	GLint *location*,
	GLsizei *count*,
	const GLfloat **value*);

void **glUniform2fv**(	GLint *location*,
	GLsizei *count*,
	const GLfloat **value*);

void **glUniform3fv**(	GLint *location*,
	GLsizei *count*,
	const GLfloat **value*);

void **glUniform4fv**(	GLint *location*,
	GLsizei *count*,
	const GLfloat **value*);

È possibile assegnare il valore ad una variabile uniforme, nel caso si tratti di una matrice 2x2, un vettore di 3x3, o una matrice 4x4

void **glUniformMatrix2fv**(	GLint *location*,
	GLsizei *count*,
	GLboolean *transpose*,
	const GLfloat **value*);

void **glUniformMatrix3fv**(	GLint *location*,
	GLsizei *count*,
	GLboolean *transpose*,
	const GLfloat **value*);

void **glUniformMatrix4fv**(	GLint *location*,
	GLsizei *count*,
	GLboolean *transpose*,
	const GLfloat **value*);

Parametri

location
Specifica la posizione della variabile uniforme da aggiornare.

count
Per i comandi (glUniform*v), specifica il numero di elementi che devono essere aggiornati.
Questo dovrebbe essere 1 se la variabile target non è un array e 1 o più se è un array.
Per i comandi matrice (glUniformMatrix*), specifica il numero di matrici da modificare.
Questo dovrebbe essere 1 se la variabile uniforme target non è una matrice di matrici,
e 1 o più se è una matrice di matrici.

transpose
Per i comandi che lavorano sulle matrici,
specifica se trasporre la matrice appena i valori vengono caricati nella variabile uniforme.

v0, v1, v2, v3
Per i comandi che lavorano sugli scalari, specifica i nuovi valori da utilizzare per la variabile uniforme specificata.

value
Per i comandi vettore e matrice, specifica un puntatore ad un array di **count** valori che verranno utilizzati per aggiornare la variabile uniforme specificata.

Esempio di variabile uniform

Application

```
GLuint loc_s;

void init() {
prog = createProgram("v.glsl", "f.glsl");
/* Otteniamo i puntatori alle variabili uniform per poterle utilizzare in seguito */
loc_s = glGetUniformLocation(prog,"s");
}

void drawScene(void){
float valore_s = 2.0;
/* Communicate the variable value to the shader */
glUniform1f(loc_s, valore_s);
...
}
```

numero di componenti

tipo di componenti

vertex shader

```
uniform float s;
vec3 out colore;

void main(){
    ...
}
```

corrispondenza

Fragment Shader

```
uniform float s;
in vec3 colore;

void main(){
    ...
}
```