OpenGL Mathematics (GLM)

Libreria GLM

Libreria matematica C++ per la programmazione grafica

https://glm.g-truc.net/0.9.9/index.html

Latest release version: 0.9.9.7 (2020/1/05)

GLM Setup

```
#include <glm/glm.hpp>    //Tipi di matrici e vettori

// Matrici di trasformazione
#include <glm/gtc/matrix_transform.hpp>
#include <glm/gtx/transform.hpp>
#include <glm/gtc/type_ptr.hpp>
```

Vettori GLM

Per impostazione predefinita e seguendo le specifiche GLSL, **i costruttori predefiniti di vettori e matrici inizializzano i componenti a zero**. Questo è un comportamento affidabile ma l'inizializzazione ha un costo e non è sempre necessaria. Questo comportamento può essere disabilitato in fase di compilazione definendo LM_FORCE_NO_CTOR_INIT prima di qualsiasi inclusione di <glm/glm.hpp> o altra inclusione GLM.

- Vettori di tipo intero: ivec{2,3,4}
- • Vettori di tipo float: vec{2,3,4}
- ivec2 mivec2; // mivec2 = (0, 0)
- ivec3 mivec3(mivec2, 0); // mivec3 = (0, 0, 0)
- ivec4 mivec4(mivec3, 0); // mivec4 = (0, 0, 0, 0)
- vec2 mvec2; // mvec2 = (0.0f, 0.0f)
- vec3 mvec3(1.0f, 2.0f, 3.0f); // mvec3 = (1.0f, 2.0f, 3.0f)
- vec3 V(3.0f, 4.0f, 5.0f); // V è un vettore di lunghezza non unitaria
- V.lenght() → misura la lunghezza del vettore ||V||
- V_unit = normalize(V); // la funzione normalize divide il vettore V per la sua norma 2 rendendolo di lunghezza unitario

- Prodotto Scalare tra due vettori (dot product)
- vec3 A(x1, y1, z1), B(x2, y2, z2);
- // Calcolo facendo uso della definizione
- float dotProduct = A.x * B.x + A.y * B.y + A.z * B.z;
- dotProduct = dot(A, B);   //facendo uso della funzione dot di GLM

- Prodotto vettoriale
- vec3 A(x1, y1, z1), B(x2, y2, z2);
- // Calcolo facendo uso della definizione
- vec3 crossProduct = vec3(
- A.y * B.z – A.z * B.y,
- A.z * B.x – A.x * B.z,
- A.x * B.y – A.y * B.x
-);
- crossProduct = cross(A, B); //facendo uso della funzione cross di GLM

Come si dichiara una matrice in GLM

- mat4 M; //inizializza la matrice M di dimensione 4x4 a valori tutti nulli
- mat4 M1(1.0); // crea la matrice identità
- mat3 M2(2.0f); // crea una matrice diagonale 3x3 con elementi sulla diagonale posti a 2.0
- mat4 M3(vec4(1, 2, 3, 4), vec4(1, 2, 3, 4), vec4(1, 2, 3, 4), vec4(1, 2, 3, 4)); // crea una matrice 4x4 utilizzando 4 vettori colonna
- float c00 = M3[0][0]; // accesso alle componente della matrice
- Applicare la matrice identità ad un vettore
- vec4 V(1, 2, 3, 1);
- mat4 M4(1.0f); // crea la matrice identità 4x4
- vec4 V2 = M4 * V; // moltiplica la matrice identità per il vettore.

Matrice di Traslazione

- • Costruisce la matrice di traslazione con spostamenti in x,y e z dati da (T_x, T_y, T_z)
- #include <glm/gtx/transform.hpp>
- vec4 V(1, 2, 3, 1);
- mat4 M4(1.0);
- M4 = translate(M4, vec3(Tx, Ty, Tz));
- vec4 V2 = M4 * V; // Moltiplica la matrice di traslazione per il vettore V

Matrice di scalatura

- • Costruisce la matrice di scalatura con fattori di scala lungo le tre dimensioni (S_x, S_y, S_z)

- #include <glm/gtx/transform.hpp>
- vec4 V(1, 2, 3, 1);
- mat4 M4(1.0);
- M4 = scale(M4, vec3(Sx, Sy, Sz));
- vec4 V2 = M4 * V;

Rotazione intorno ad un dato asse

- Costruisce la matrice di rotazione dato l'asse (R_x, R_y, R_z) e l'angolo theta
- #include <glm/gtx/transform.hpp>
- #define PI 3.14159265358979323846

- vec4 V(1, 2, 3, 1);
- mat4 M4(1.0);
- M4 = rotate(M4, radians(theta), vec3(Rx, Ry, Rz));
- vec4 V2 = M4 * V;
- radians funzione di glm che fa la conversion da gradi a radianti.

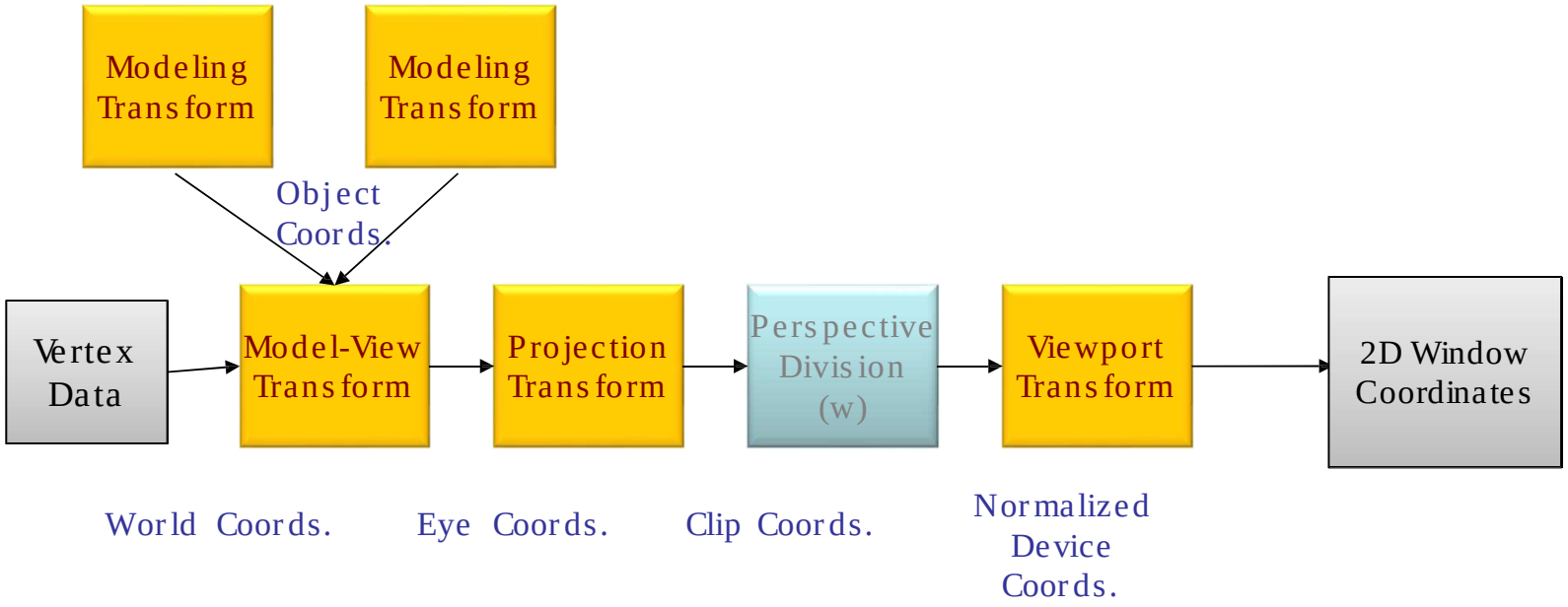
Trasformazioni

- Modellazione
- Vista
- Proiezione

Trasfomazioni in OPENGL

Le trasformazioni ci permettono di passare da uno spazio ad un altro, e vengono usate per realizzare una scena 3D e la pipeline del rendering.

Usiamo trasformazioni rappresentate da matrici 4x4



Trasformazioni di Modellazione

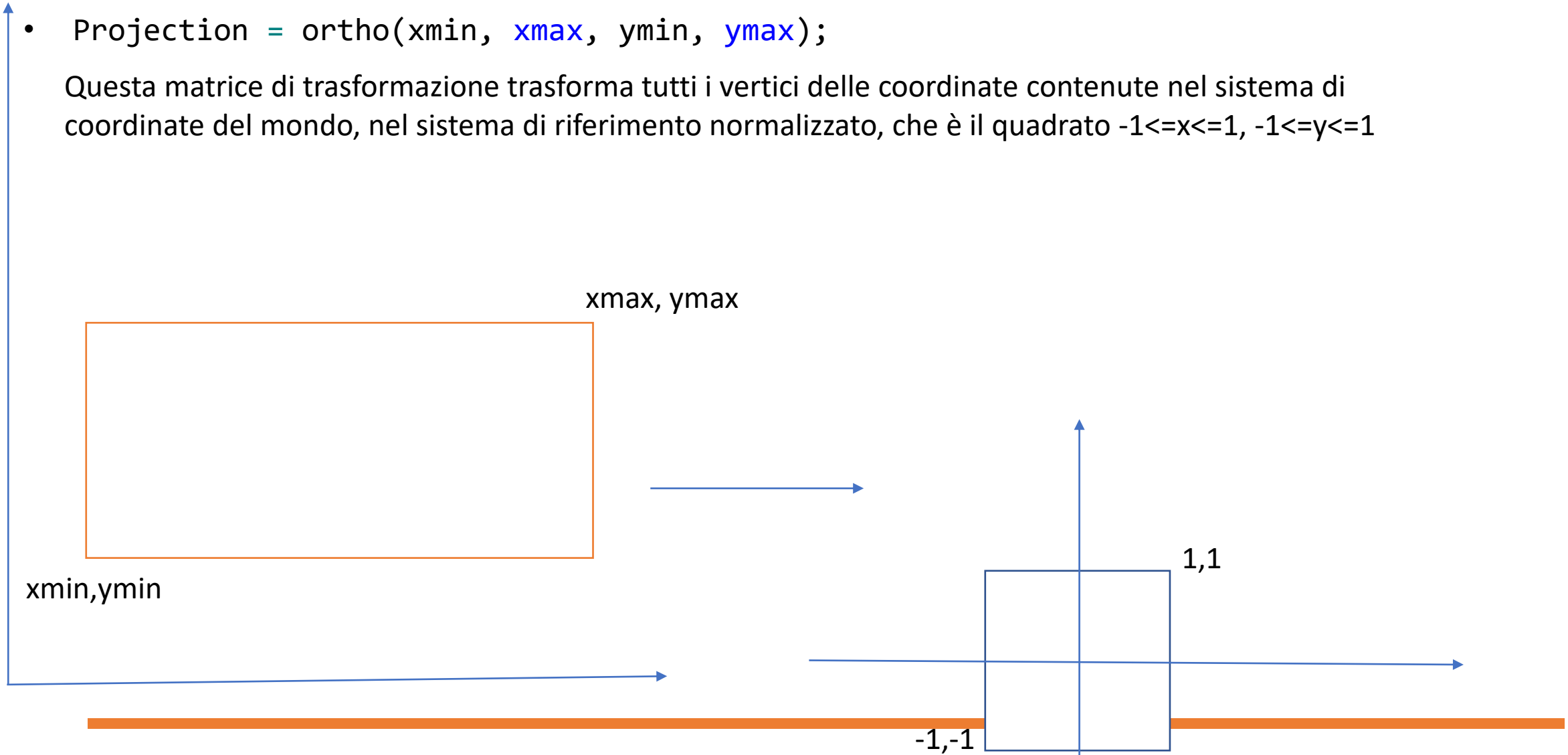
- Prima definiamo le trasformazioni che devono essere effettuate sul modello geometrico, successivamente il modello viene disegnato.
- //initialize identity matrix and transformations
- mat4 Model = mat4(1.0f);
- Model = translate(Model, dx, dy, dz); //matrix T
- Model = scale(Model, sx, sy, sz); //matrix S
- Model = rotate(Model, angle, rx,ry,rz); //matrix R
- //comunica la Uniform matrice Model variabile Uniform, allo shader
- // Draw the object
- glDrawArrays(GL_TRIANGLES, 0, nvertices);

- Model = T*S*R

Proiezione (caso 2D ortografico)

- `Projection = ortho(xmin, xmax, ymin, ymax);`

Questa matrice di trasformazione trasforma tutti i vertici delle coordinate contenute nel sistema di coordinate del mondo, nel sistema di riferimento normalizzato, che è il quadrato -1<=x<=1, -1<=y<=1



Matrice di Traslazione per traslare il centro della finestra in coordinate del mondo in (0,0,0)

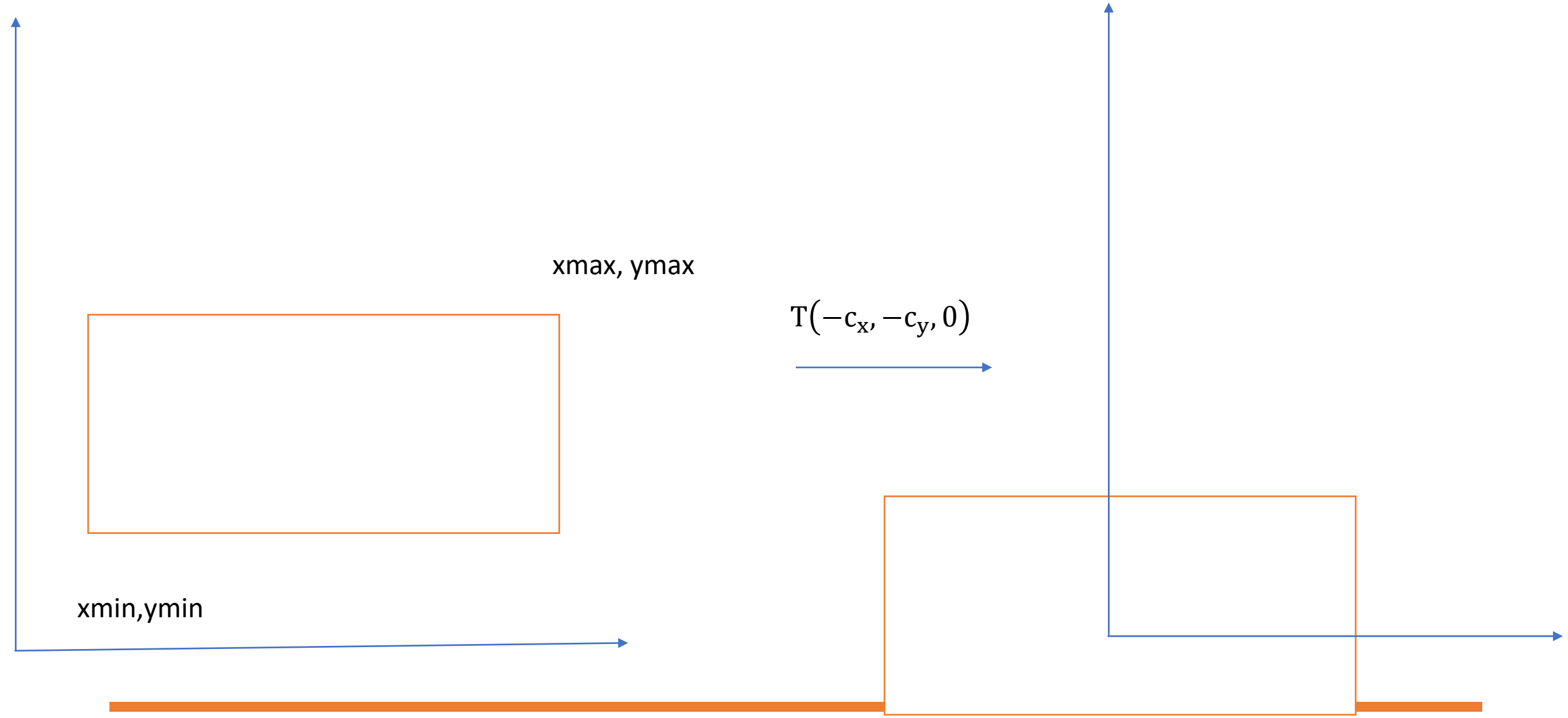
Centro della finestra in coordinate del mondo:

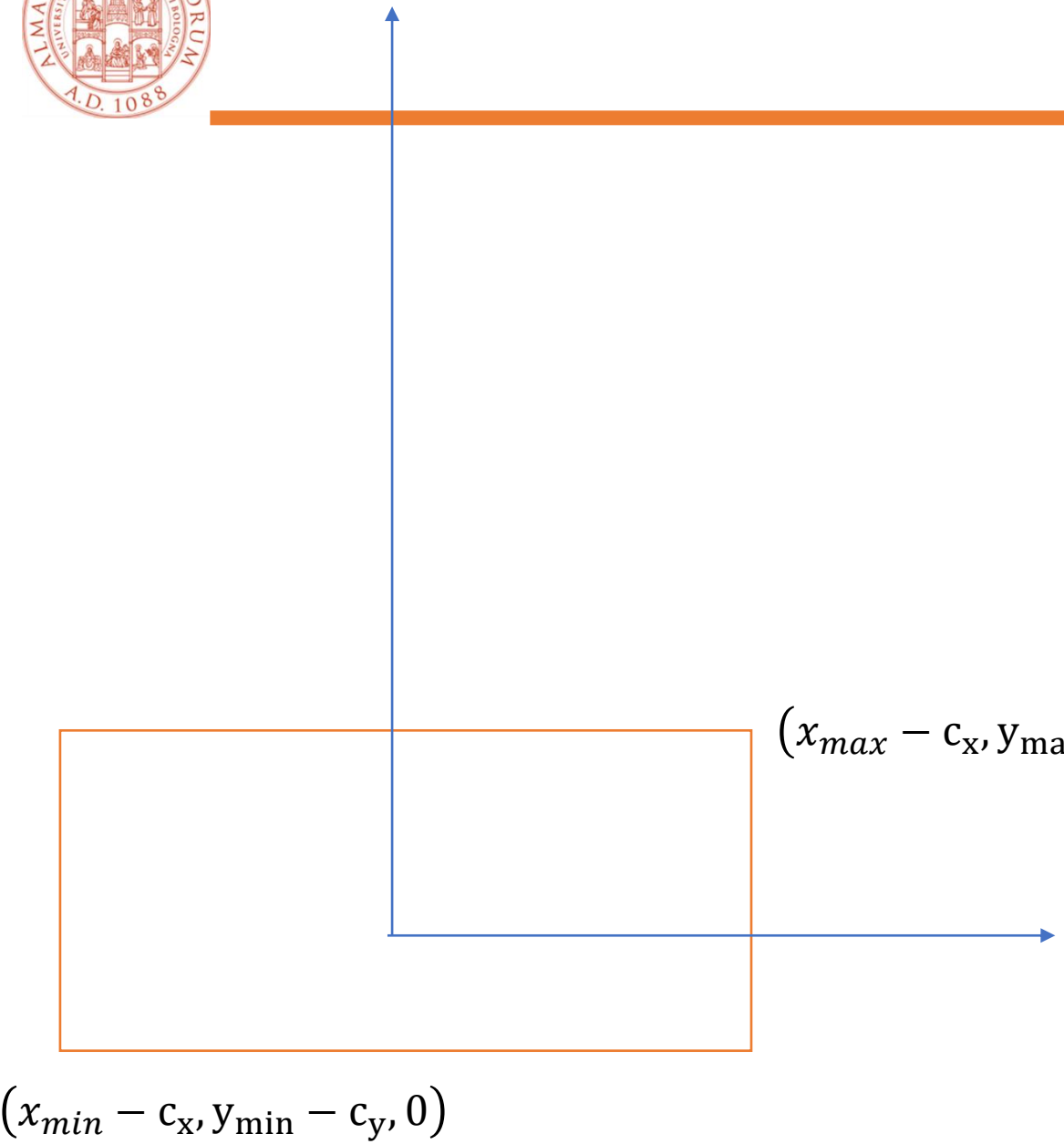
$$C_x = \frac{x_{max} + x_{min}}{2}$$
$$C_y = \frac{y_{max} + y_{min}}{2}$$

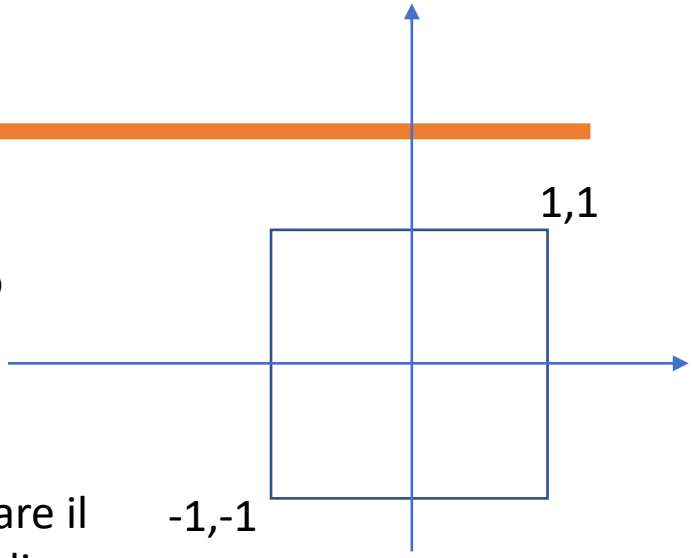
$$\mathrm{T}(-\mathrm{c_x}, -\mathrm{c_y}, 0) = \begin{bmatrix} 1 & & & -c_x \\ & 1 & & -c_y \\ & & 1 & 0 \\ & & & 1 \end{bmatrix}$$

Applicare questa matrice al centro della finestra mondo lo riporta all'origine

$$\begin{bmatrix} 1 & & & -c_x \\ & 1 & & -c_y \\ & & 1 & 0 \\ & & & 1 \end{bmatrix} \begin{bmatrix} c_x \\ c_y \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$


xmax, ymax
$T(-c_x, -c_y, 0)$
xmin,ymin





Ricordiamo che vogliamo scalare questa finestra in maniera tale che la sua larghezza e la sua altezza valgano 2. Vogliamo trovare il fattore di scala sx ed sy tali che

$$s_x(x_{max} - x_{min}) = 2$$

$$s_y(y_{max} - y_{min}) = 2$$

Da cui segue che:

$$s_x = \frac{2}{x_{max} - x_{min}}$$

$$s_y = \frac{2}{y_{max} - y_{min}}$$

$$S(s_x, s_y, 1) = \begin{bmatrix} s_x & & & 0 \\ & s_y & & 0 \\ & & 1 & 0 \\ & & & 1 \end{bmatrix}$$

Quindi la matrice di trasformazione che trasforma tutti i vertici delle coordinate contenute nel sistema di coordinate del mondo, nel sistema di riferimento normalizzato è data dal prodotto $S(s_x, s_y, 1) \cdot T(-c_x, -c_y, 0)$

$$P = \begin{bmatrix} s_x & & & 0 \\ & s_y & & 0 \\ & & 1 & 0 \\ & & & 1 \end{bmatrix} \begin{bmatrix} 1 & & & -c_x \\ & 1 & & -c_y \\ & & 1 & 0 \\ & & & 1 \end{bmatrix}$$

Nel caso 2D, Viewing è la matrice identità.

V’ = Projection * Viewing * Modeling * V

drawScene()

```
mat4 model;
mat4 view;
mat4 proj;
…
mat4 mvp = proj * view * model;
glUniformMatrix4fv(0, 1, GL_FALSE, value_ptr(mvp));
```

Spedisce la matrice allo shader:
Quante matrici: 1
Trasposta: FALSE

vertex_shader.glsl

```
#version 330 core
layout (location = 0) in vec3 aPos;

uniform mat4 Umvp;
in vec4 vertex;

void main()
{
gl_Position = Umvp * vertex;
}
```

Trasformazioni di Viewport.

Viewport = regione rettangolare dello schermo dove l'immagine viene proiettata
Default: la viewport coincide con la finestra aperta sullo schermo

• E' possibile modificare la regione della finestra sullo schermo su cui andare a visualizzare il disegno con la funzione

glViewport(x,y,larghezza,altezza)

x,y specifica l'angolo inferiore sinistro del rettangolo della finestra, in pixel