Grafica al calcolatore
Laboratorio - 4

Andrea Giachetti andrea.giachetti@univr.it
Marco Fattorel, Fabio Marco Caputo
Department of Computer Science, University of Verona, Italy

Reminder:

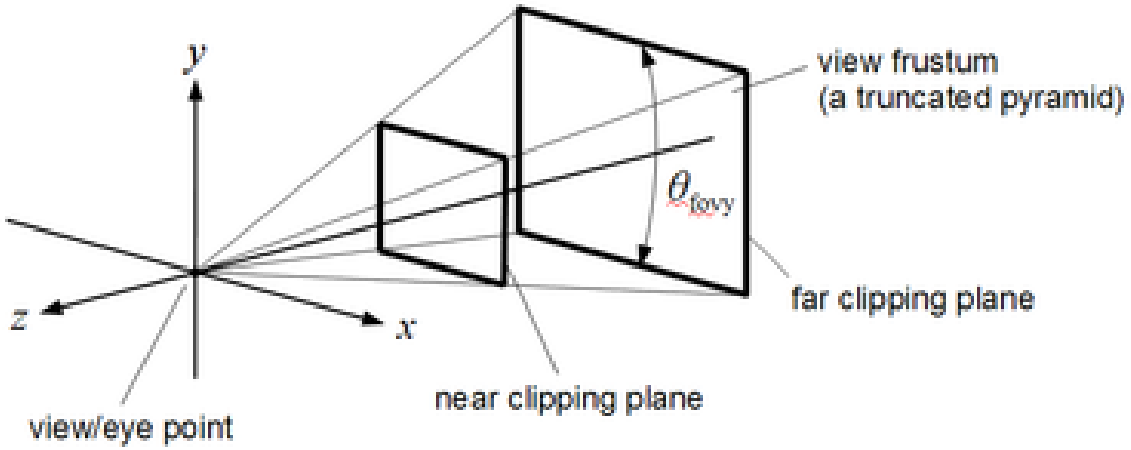
- Ricordare il path per le librerie GLFW
  - export LD_LIBRARY_PATH=lib/lin
- Per usare l'emulazione mesa
  - export LIBGL_ALWAYS_SOFTWARE=1
- Per far funzionare i codici su alcune macchine con scheda video intel su ubuntu
  - export MESA_GLSL_VERSION_OVERRIDE=130
  - Potrebbe essere provato in alternativa al comando precedente

Passiamo (finalmente) al 3D

- Caricare l'esercizio e04
- Il modello adesso è 3D: vertici e edge di un cubo
- Ci appiccichiamo una texture che mappa i lati a regioni di un'immagine creata ad hoc
- Introduciamo le matrici: model matrix, view matrix e proiezione prospettica
  - Sono gestite con la libreria glm. Passate al vertex shader come uniform
  - L'operazione è svolta poi nel vertex shader
- Usiamo l'algebra delle matrici in coordinate omogenee che abbiamo visto in teoria

Matrici

- Model Matrix: la applichiamo ai modelli per muoverli nella scena.
- View Matrix: muoviamo il sistema di riferimento standard associato con la telecamera
- Projection matrix: applichiamo la proiezione per ottenere le coordinate standardizzate sul piano immagine
- Poi c'è la viewport che abbiamo già visto



Codice

```
glm::mat4 projection = glm::perspective(PI/4,    // vertical FOV
1.f/1.f,          // aspect ratio
1.0f,             // near clipping plane
10.0f             // far clipping plane
);
```

Codice

```
glm::mat4 view = glm::lookAt(
glm::vec3(2.5f, 2.5f, 2.5f),  // punto da cui guardo
glm::vec3(0.0f, 0.0f, 0.0f),  // punto a cui guardo
glm::vec3(0.0f, 1.0f, 0.0f)   //up vector
);
```

Codice

```
glUniformMatrix4fv(glGetUniformLocation(shaderProgram,
"projection"), 1, GL_FALSE, &projection[0][0]);
glUniformMatrix4fv(glGetUniformLocation(shaderProgram,
"view"), 1, GL_FALSE, &view[0][0]);
glUniformMatrix4fv(glGetUniformLocation(shaderProgram,
"model"), 1, GL_FALSE, &model[0][0]);
```

Riepilogo: glm

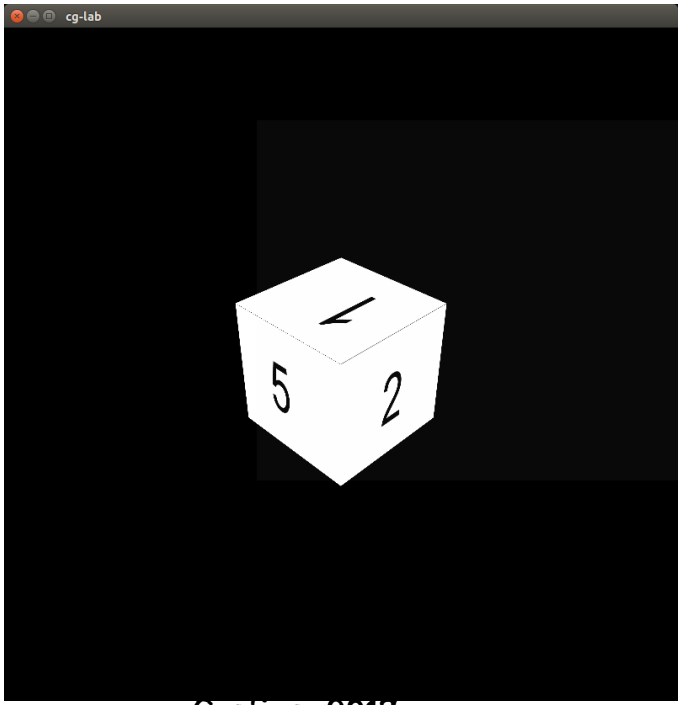
- Si può applicare
  - mat4 translate(mat4, vec3)
  - mat4 scale(mat4, vec3)
  - mat4 rotate(mat4, vec3)
- Se mat4 è fornito come primo argomento il risultato è moltiplicato e ritornato dalla funzione
  - m=glm::translate(m,glm::vec3(1,1,1));
  - m=glm::rotate(m,a,glm::vec3(0,0,1));
  - m=glm::translate(m,glm::vec3(1,1,1));
- È più efficiente di
  - const glm::mat4 i(1.0); //identity matrix
  - m=m*glm::translate(i,glm::vec3(1,1,1));
  - m=m*glm::rotate(i,a,glm::vec3(0,0,1));
  - m=m*glm::translate(i,glm::vec3(1,1,1));

Ricordiamo

- Possiamo per ogni oggetto nella scena memorizzare una model martix
- Se abbiamo più istanze di oggetti uguali, o con semplicemente attributi da cambiare, usiamo lo stesso buffer e facciamo i vari disegni dopo le rispettive trasformazioni
- La composizione delle trasformazioni segue le regole che abbiamo studiato a teoria
  - Es: verificare che la composizione dipende dall'ordine! Applicare al quadrato fermo una model matrix composta e invertire l'ordine. Che succede?

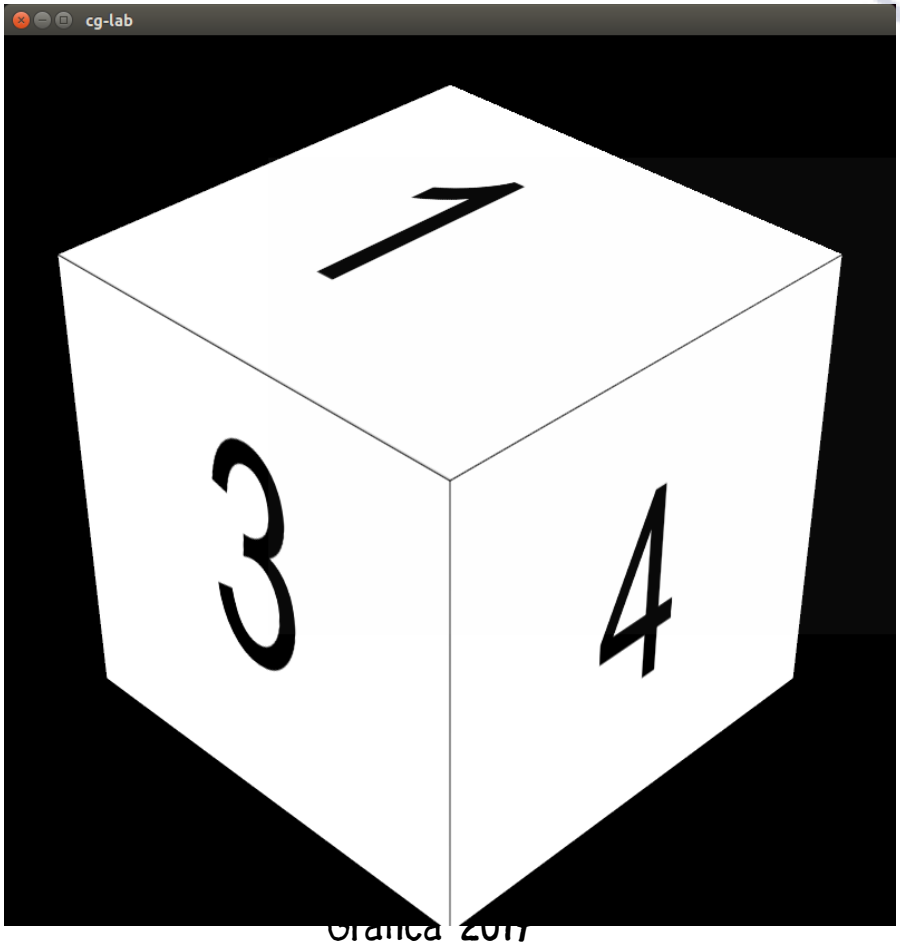
Esercizio

- A partire dal codice originale, modificare la model matrix per ruotare il cubo in modo che mostri le facce 1,2,5
- Nota abbiamo forzato l'uso dell'angolo in radianti
- A partire dal codice originale, ottenere lo stesso effetto modificando invece i parametri di lookat



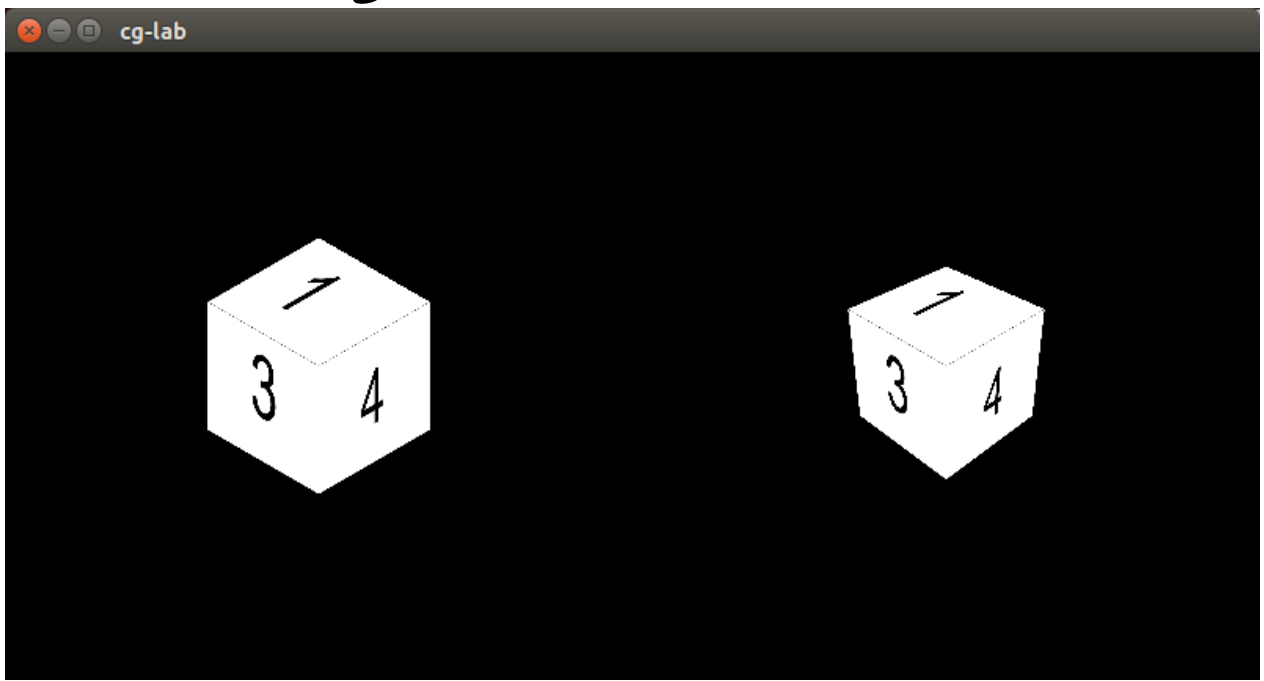
Esercizio

- Sempre dall'originale e04 cambiare i parametri della proiezione per avere un risultato simile a quello qui sotto



Esercizi

- Provare a sostituire la proiezione prospettica con quella ortografica
- glm::ortho (T const &left, T const &right, T const &bottom, T const &top, T const &zNear, T const &zFar)
- Provare a mettere in una finestra 800x400 due viewport sui quali mappare un rendering con proiezione ortografica e prospettica come nella figura sotto



Depth buffer e backface culling

- Cosa succede se si commentano le corrispondenti righe? Verificare!

- //Qui si attiva lo z-buffer!
- glEnable(GL_DEPTH_TEST); check(__LINE__);
- //qui si attiva il back face culling
- glEnable(GL_CULL_FACE); check(__LINE__);

Tempo e animazione

- Per animare e muovere interattivamente gli oggetti e la telecamera occorre affidarsi al tempo macchina anche per evitare problemi dovuti alle differenti prestazioni dell'HW
  - Provate a confrontare cosa succederebbe se usaste rotazione proporzionale al numero di frame calcolati...
- #include <chrono>
- using timer = std::chrono::high_resolution_clock;
- Nel draw calcoliamo il tempo intercorso dal disegno precedente
- t = (timer::now()-start_time).count() * (float(timer::period::num)/float(timer::period::den));
- dt = (timer::now()-last_time).count() * (float(timer::period::num)/float(timer::period::den));
- glm::mat4 model = glm::rotate(glm::mat4(1.f), t, glm::vec3(0.0f, 1.0f, 0.0f)); //anima il cubo centrale, dt si usa per la navigazione

Uso texture multiple

- glGenTextures(2, textures);
- ...
- glActiveTexture(GL_TEXTURE1);
- glBindTexture(GL_TEXTURE_2D, textures[1]);
-
-
- glUniform1i(glGetUniformLocation(shaderProgram, "textureSampler"), 0);
- glUniform1i(glGetUniformLocation(shaderProgram, "textureSampler2"), 1);

Interpolazione tra 2 valori

- //GLSL provides the mix function. This function should be used where possible:
- resultRGB = mix(colorRGB_0, colorRGB_1, alpha);

Esercizio

- Caricare il file e04b.cpp
- Aggiungere seconda texture dal file cube2.png
  - Fare sì che il cubo cambi texture alla pressione di un tasto
  - Fare sì che alla pressione di un altro tasto la direzione di rotazione cambi
  - Fare sì che al premere di un tasto le tessiture delle due immagini siano mescolate
  - Fare sì che al premere di un tasto la telecamera si muova verso l'alto

La prossima settimana o la successiva

- Esercizio da consegnare
- Specifiche all'inizio dell'ora
- Consegna entro fine ora sull'area di elearning
- Lavoro individuale