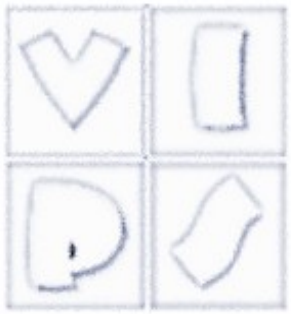
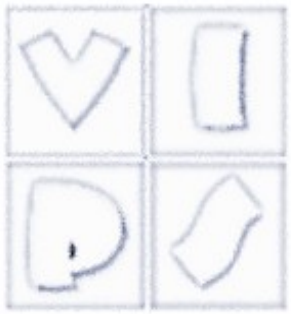


Grafica al calcolatore
Laboratorio - 3

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

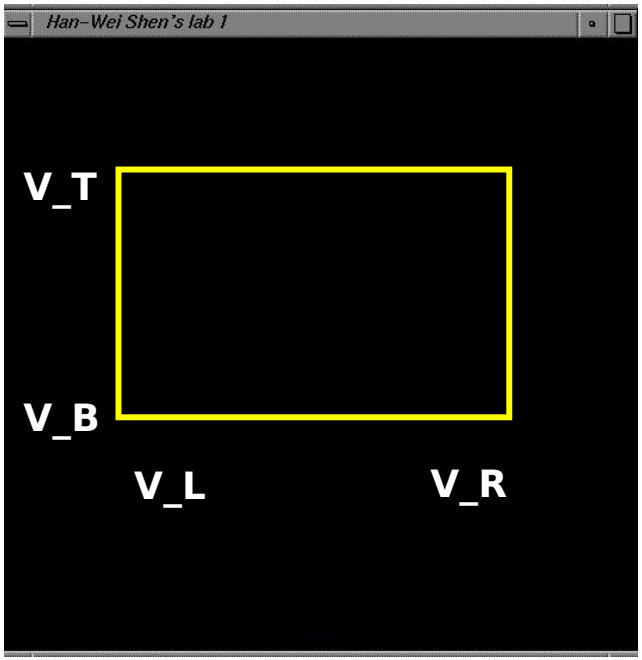


Note

- I possibili valori per glDrawElements sono
  - GL_POINTS, GL_LINE_STRIP, GL_LINE_LOOP, GL_LINES, GL_TRIANGLE_STRIP, GL_TRIANGLE_FAN, and GL_TRIANGLES
- GlClearColor(R,G,B,A) cambia il colore con cui viene ripulito il frame buffer

Viewport

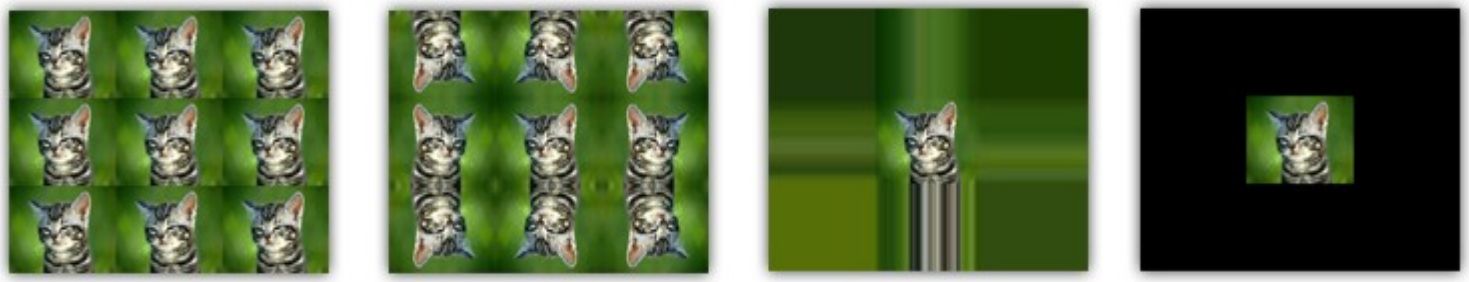
- The rectangular region in the screen that maps to our world window
- Defined in the window's (or control's) coordinate system

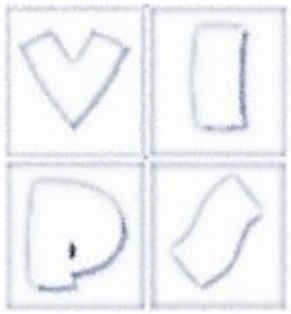



```
glViewport(int  left, int  bottom,
           int  (right-left),
           int (top-bottom));
```

Textures

- Abbiamo visto in teoria il meccanismo del texture mapping
- Ora lo vediamo in pratica. Dobbiamo
  - Avere un modo di mettere in memoria l'immagine texture
  - Associare un attributo ai vertici con le coordinate (ricordate che erano teoricamente nell'intervallo [01] [01] che corrispondevano agli estremi dell'immagine, anche se poi avevamo visto che potevamo mettere coordinate fuori estendendo la parametrizzazione con le modalità
    - Tile: repeat (OGL); ripetizione periodica
    - Mirror:ripetizione periodica ma specchiata a ogni ripetizione
    - Clamp to edge – valori esterni sono prolungati dal bordo vicino
    - Clamp to border – tutti i valori esterni sono attribuiti ad un valore a parte





Esercizio e03

- Sempre geometria2D come prima, ma mappiamo texture
- Utilizziamo la libreria http://lodev.org/lodepng/
- Togliamo il colore e mettiamo le coordinate texture al suo posto: 2 float Le definiamo sui vertici

```
const GLfloat vertices[] = {
//   Position      Texcoords
     -0.5f,  0.5f, 0.0f, 0.0f,// Top-left
      0.5f,  0.5f, 1.0f, 0.0f, // Top-right
      0.5f, -0.5f, 1.0f, 1.0f,  // Bottom-right
     -0.5f, -0.5f, 0.0f, 1.0f// Bottom-left
};
```

- Definiamo l'oggetto texture: GLuint texture;

- Nel main si chiama la funzione
  - initialize_texture();

```
glGenTextures(1, &texture);
	std::vector<unsigned char> image;
	unsigned width, height;

	unsigned error = lodepng::decode(image, width, height, "image.png");
	if(error) std::cout << "decode error " << error << ": " << lodepng_error_text(error) <<
std::endl;

	// il bind della texture avverra' sulla texture unit numero 0 (l'indice che dovra' essere
passato agli shader)
	glActiveTexture(GL_TEXTURE0);
	glBindTexture(GL_TEXTURE_2D, texture);
	glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, width, height, 0, GL_RGBA,
GL_UNSIGNED_BYTE, image.data());
	// shaderProgram must be already initialized
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
}
```

  - La funzione collega la texture a un indice e definisce parametri
    - comportamento per i parametri st fuori [0 1]
    - Filtri per magnification e minification
      - (si potrebbe fare mipmapping)

glUniform1i(glGetUniformLocation(shaderProgram, "textureSampler"), 0);

- Nel draw() avviene il collegamento della texture da usare (ce ne potrebbero essere diverse) alla variabile “sampler” di tipo uniform con cui si accede dal GLSL

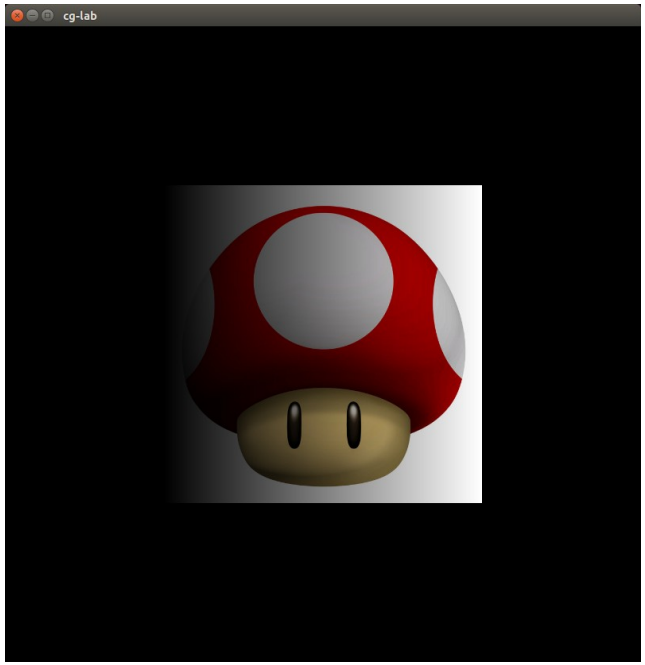
```
"uniform sampler2D textureSampler;"
"void main() {"
" outColor = texture2D(textureSampler, Coord);"
```

- Alla fine la texture deve essere distrutta

```
glDeleteTextures(1, &texture);
```

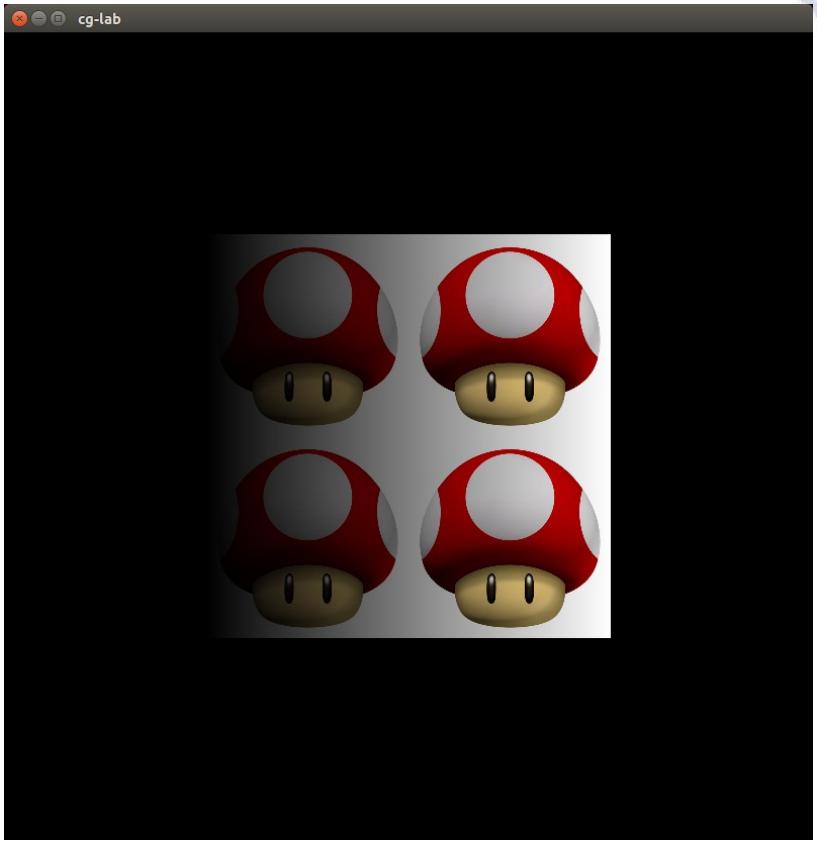
Esercizio

- Caricare ora il file es03b.cpp
- E' come prima, ma si passano allo shader sia coordinate texture e texture sia colore vertici (tutti bianchi)
- L'output è un blending moltiplicativo
- Modificare il programma per ottenere l'effetto in figura



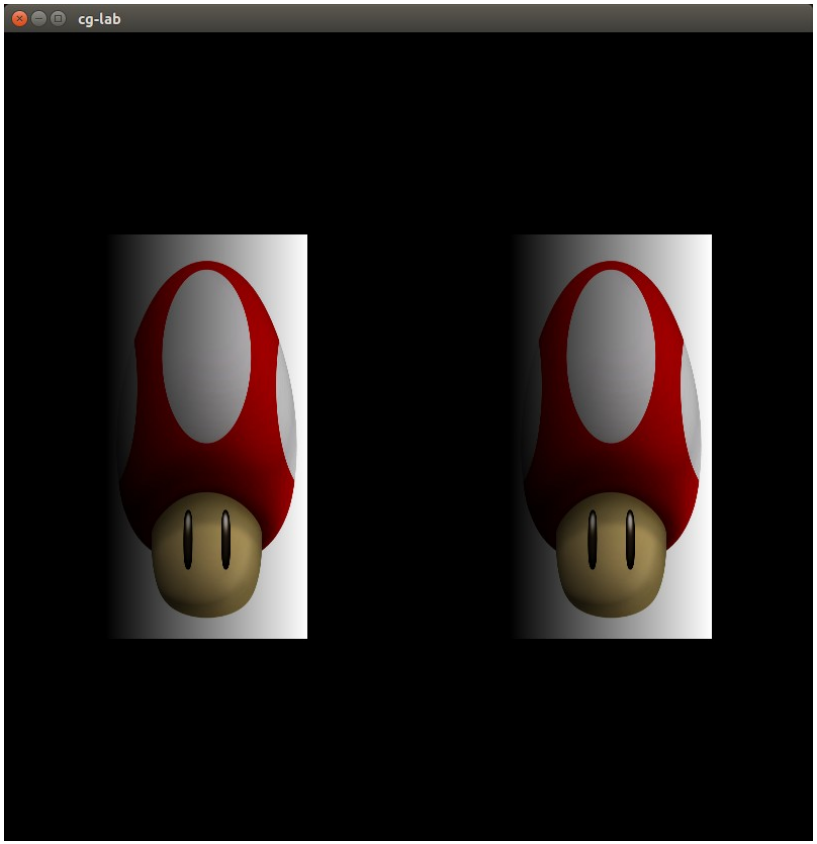
Esercizio

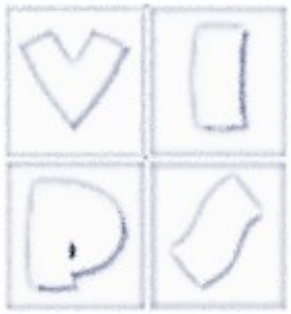
- Modificare ulteriormente il programma per ottenere l'immagine qui sotto



Esercizio

- Usare due viewport per ottenere la figura qui sotto



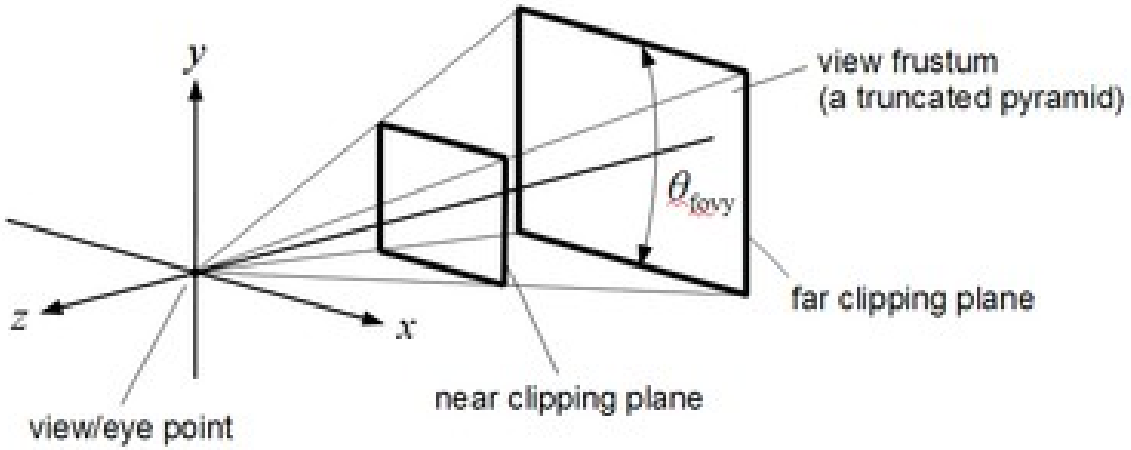


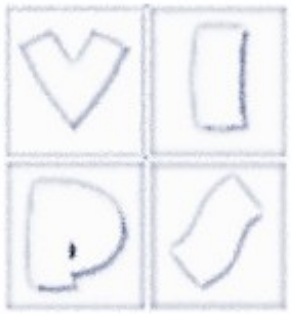
Passiamo (finalmente) al 3D

- 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

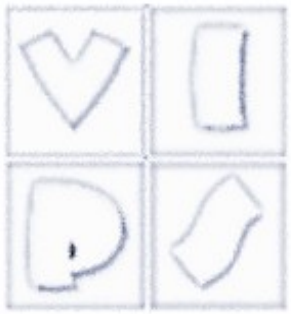
- Model Matrix: la applichiamo ai modelli per muoverli nella scena.
- View Matrix: muoviamo il sistema di riferimento standard associato con la telecamera
- Projection matrix: applichiamo la proiezione per ottenere le coordinate standardizzate sul piano immagine
- Poi c'è la viewport che abbiamo già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





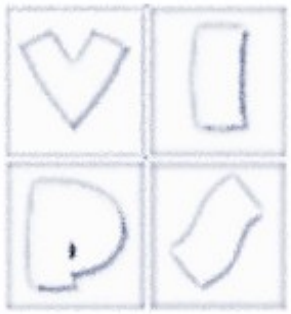
Codice

```
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
);
```

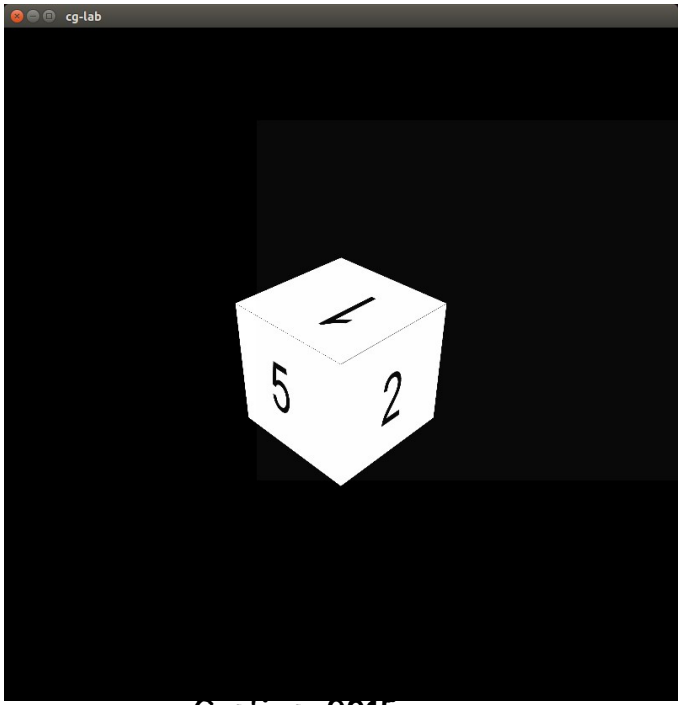


Codice

```
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]);
```

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

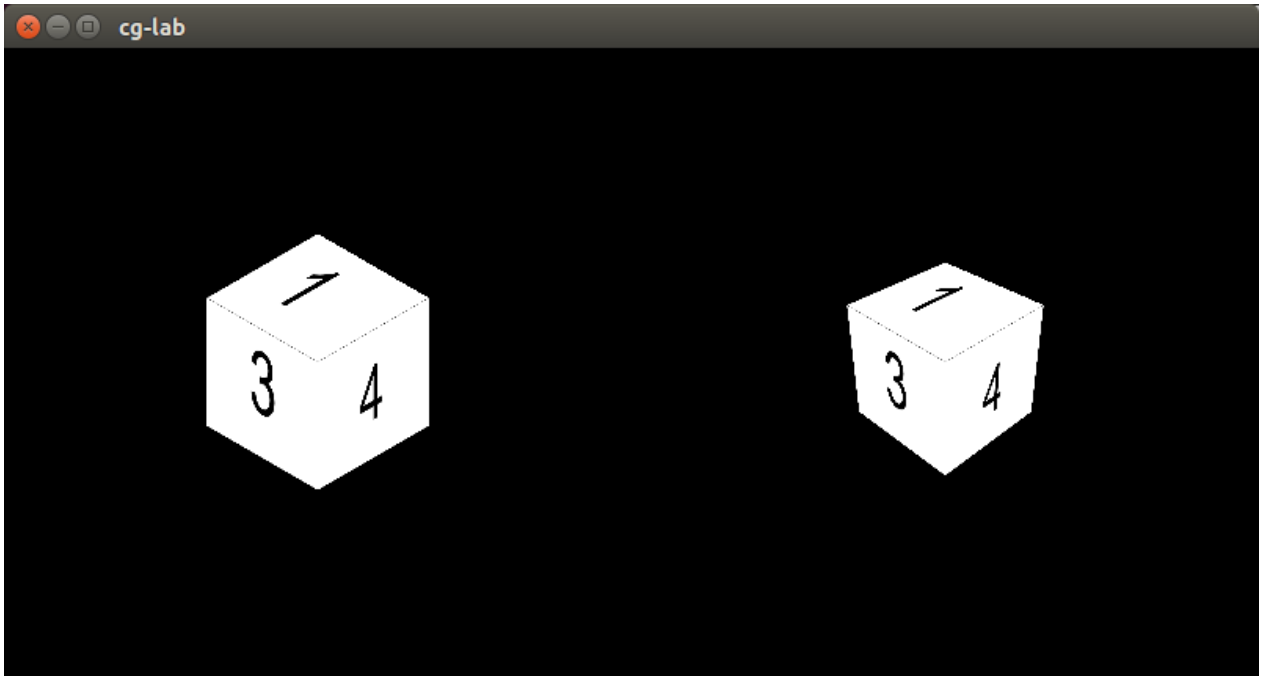


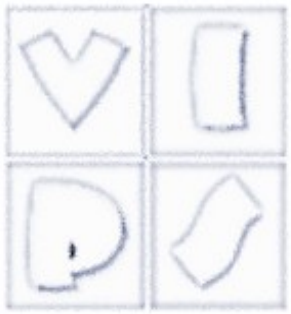
Esercizio

- 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





Riferimenti

- http://www.opengl.org
- http://www.khronos.org/opengl/
- http://www.glfw.org/
- http://antongerdelan.net/opengl
- http://www.opengl-tutorial.org/
- http://www.arcsynthesis.org/gltut/
- http://glm.g-truc.net/0.9.5/index.html