SOMMARIO

- Introduzione alle librerie grafiche e a OpenGL
- Implementazioni hw e sw
- Supporto multipiattaforma
- Esempio di programma con OpenGL
- Sistemi di coordinate 2D e 3D – Proiezioni
- Primitive 3D

Introduzione

DEFINIZIONE: OpenGL puo' essere definita come un'interfaccia software ad hardware grafici

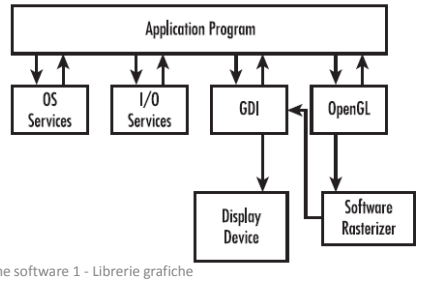
- E' una libreria per la grafica e la modellazione 3D.
- Non e' un linguaggio di programmazione (come C/C++/C#), eccetto per l'OpenGL shading language.
- E' una "application programming interface" (API). E' quindi possibile richiamare le funzioni della libreria direttamente dalla maggior parte dei linguaggi di programmazione.
- Non esiste un formato di file specifico di OpenGL per descrivere i modelli o gli ambienti virtuali.

Implementazioni sw e hw

- E' una libreria rivolta all'utilizzo con dispositivi hardware progettati e ottimizzati per la visualizzazione e la manipolazione di grafica 3D.
- E' caratterizzata da un'elevata velocita' e portabilita'.
- Esistono implementazioni software (Microsoft, Mesa3D, Apple). Nelle implementazioni software il rendering non puo' essere effettuato velocemente come nelle implementazioni hardware, ma i programmi possono essere eseguiti anche senza un hardware grafico accelerato.

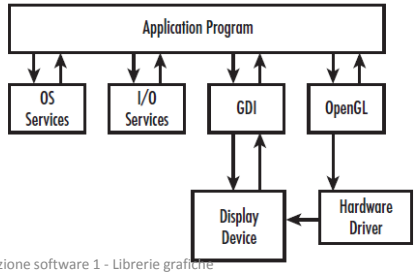
Implementazioni sw

Esempio: applicazione Windows. Un'implementazione software di OpenGL riceve le richieste grafiche da un'applicazione e costruisce (rasterize) l'immagine dell'oggetto grafico 3D. Fornisce quindi al graphics device interface (GDI) l'immagine da visualizzare a monitor. Su altri OS il GDI e' sostituito dall'equivalente API per la gestione del display.

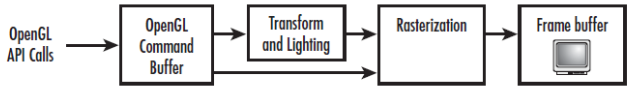


Implementazioni hw

Esempio: scheda grafica. Le chiamate alle API OpenGL sono passate al driver dell'hw. Il driver non passa i suoi output alle API Windows GDI, ma si interfaccia direttamente con l'hardware di visualizzazione (implementazione accelerata). Tuttavia, alcune funzionalita' sono implementate in software come parte del driver, altre sono passate direttamente all'hardware.



Implementazioni hw



A seconda dell'hardware grafico porzioni maggiori o minori della fase di "trasformazione" possono essere eseguite in hardware. Cio' comporta la possibilita' di calcolare e visualizzare modelli sempre piu' dettagliati in tempi minori.

Indipendenza dalla piattaforma

- OpenGL non ha funzioni per la gestione delle finestre, l'interazione con l'utente, l'I/O su file. Tali funzioni sono rimandate a quelle fornite dal sistema operativo.
- Questo permette di avere un'astrazione dell'hardware grafico indipendente dalla piattaforma.
- Funzionalita' quali la gestione delle finestre, sono fornite dall'OpenGL utility toolkit (GLUT), una libreria croos-piattaforma, disponibile per distribuzioni Linux, supportata in maniera nativa da Mac OS X e disponibile attraverso l'implementazione freeglut per Windows.
- GLUT non sostituisce le funzionalita' delle API dello specifico OS, ma fornisce la base per creare programmi portabili sotto diversi OS.

Esempi

I seguenti esempi sono creati utilizzando

- Microsoft Visual Studio 2008
- OpenGL fornito da Windows
- Libreria GLUT

Struttura di un programma

```
#include <iostream>
#include <glut.h>
using namespace std;

void RenderScene();
void MyInit();

int main(int argc, char* argv[])
{
	glutInit(&argc, argv);
	glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
	glutCreateWindow("Esempio OpenGL");
	glutDisplayFunc(RenderScene);
	MyInit();
	glutMainLoop();
	return 0;
}
```

Display callback function: GLUT chiama questa funzione ogni volta che la finestra deve essere ridisegnata.

Struttura di un programma

```
void MyInit()
{
	glClearColor(1,1,1,1);
}

void RenderScene()
{
	glClear(GL_COLOR_BUFFER_BIT);
	glFlush();
}
```

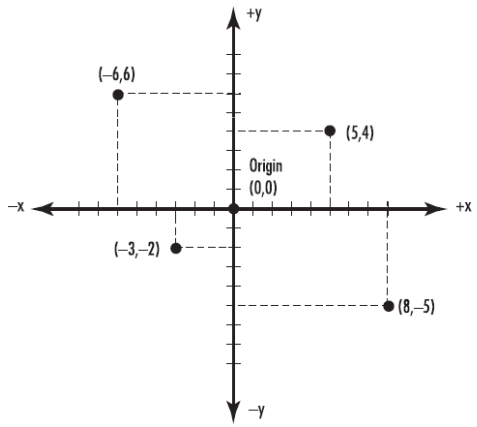
Definisce il colore usato per cancellare la finestra (RGB + alpha)

Cancella un particolare buffer, in questo caso il “color buffer”.

Esegue i comandi OpenGL non ancora eseguiti.

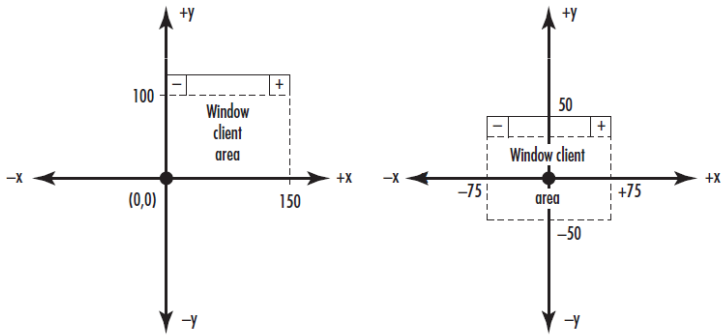
Sistemi di coordinate 2D

Esempio: sistema di coordinate 2D cartesiano



Sistemi di coordinate 2D

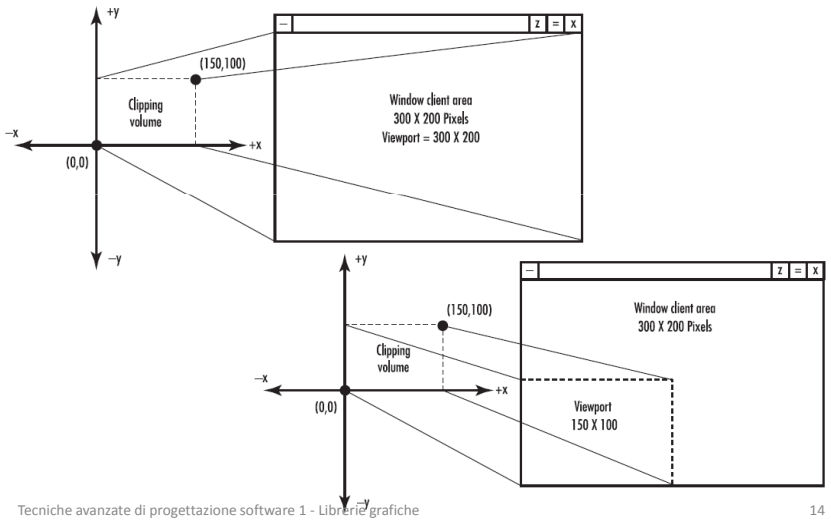
E’ necessario stabilire come trasformare le coordinate cartesiane in coordinate schermo , specificando la regione occupata dalla finestra. Questa regione e’ chiamata *regione di clipping*.



Viewport

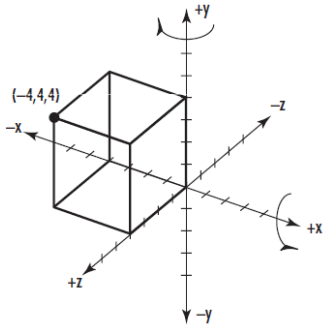
- L'altezza e la larghezza dell'area di clipping potrebbero non corrispondere all'altezza e alla larghezza della finestra (in pixels). Il sistema di coordinate deve quindi essere mappato da coordinate cartesiane "logiche" a coordinate schermo "fisiche" (in pixels). Questo mapping e' noto come "definizione di una *viewport*".
- La viewport mappa l'area di clipping ad una regione della finestra.
- Le dimensioni della viewport possono essere pari a quelle della finestra, maggiori o inferiori.

Viewport



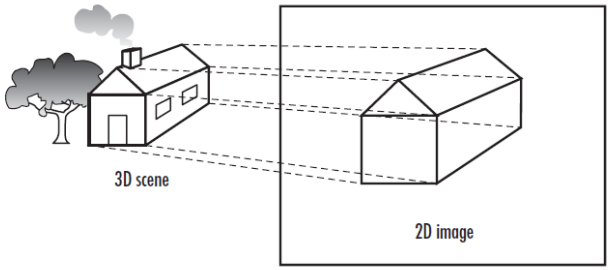
Sistemi di coordinate 3D

Esempio: sistema di coordinate 3D cartesiano



Proiezioni

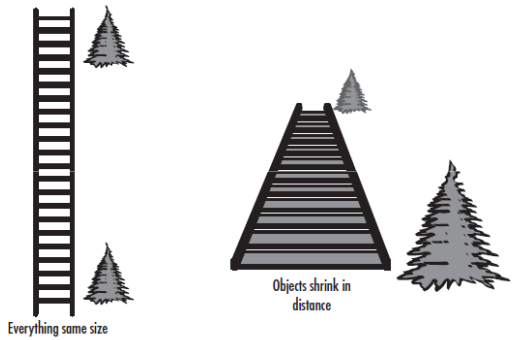
- E' necessario rappresentare oggetti definiti in 3 dimensioni su una superficie bidimensionale (lo schermo). La trasformazione da 3D a 2D e' nota come proiezione.



Proiezioni

- Specificando la proiezione viene specificato il *viewing volume*, ovvero la porzione di spazio che si vuole trasformare e come esso deve essere trasformato.
- Esistono due diversi tipi di proiezione: ortografica e prospettica

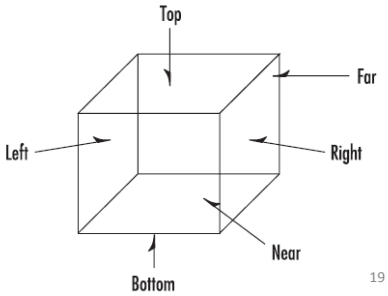
Proiezioni



PROIEZIONE ORTOGRAFICA **PROIEZIONE PROSPETTICA**

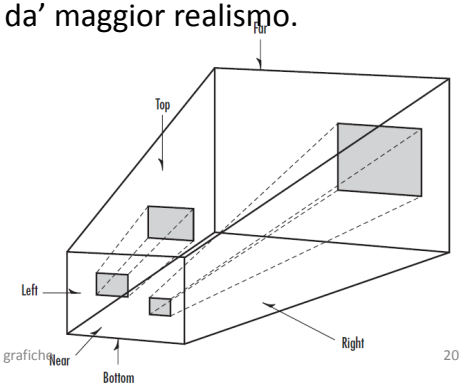
Proiezione ortografica

- Nella proiezione ortografica (o parallela) viene specificato un viewing volume cubico (o a parallelepipedo). Tutto cio' che e' al di fuori di questo volume non viene rappresentato.
- Gli oggetti con la stessa dimensione vengono rappresentati con la stessa dimensione indipendentemente dalla loro posizione (usata da programmi CAD)



Proiezione prospettica

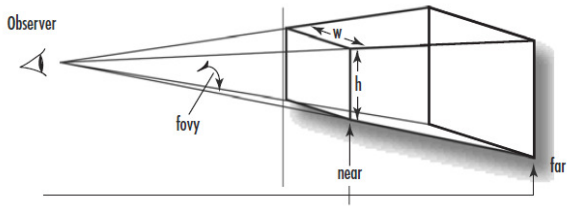
- Nella proiezione prospettica il viewing volume ha la forma di un tronco di piramide (*frustum*).
- Gli oggetti lontani appaiono piu' piccoli degli oggetti (con la stessa dimensione) vicini.
- Questo tipo di proiezione da' maggior realismo.



Proiezione prospettica

- Le proiezioni prospettiche sono definite utilizzando la seguente funzione:

```
void gluPerspective(Gldouble fovy, Gldouble aspect,
                    Gldouble zNear, Gldouble zFar);
```



Esempio

```
int main(int argc, char* argv[]){
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB );
  glutCreateWindow("Esempio OpenGL – Viewport");
  glutDisplayFunc(RenderScene);
  glutReshapeFunc(ChangeSize);
  MyInit();
  glutMainLoop();
  return 0;
}
void RenderScene(){
       glClear(GL_COLOR_BUFFER_BIT);
       glColor3f(0,1,0);
       glRectf(-25,25,25,-25);
       glFlush();
}

void MyInit(){glClearColor(1,1,1,1);}
```


Esempio

Questa funzione viene chiamata dalla libreria GLUT quando la finestra cambia dimensioni.

```
void ChangeSize(GLint w, GLint h)
{
       GLfloat aspectRatio;
       if(h==0)
              h=1;
       glViewport(0,0,w,h);
       glMatrixMode(GL_PROJECTION);
       glLoadIdentity();
       aRatio = (GLfloat)w/(GLfloat)h;

       if(w<=h)
              glOrtho(-100,100,-100/aRatio, 100/aRatio,1,-1);
       else
              glOrtho(-100*aRatio,100*aRatio,-100, 100,1,-1);
       glMatrixMode(GL_MODELVIEW);
       glLoadIdentity();
}
```

Si setta la viewport alle dimensioni della finestra

Si resetta il sistema di coordinate

Disegnare oggetti in 3D

- Gli oggetti piu' complessi sono costruiti a partire da primitive.
- Le primitive in OpenGL sono oggetti a 1-2-3 dimensioni.
- Un punto in 3D e' definito da

```
glVertex*(GLfloat x, GLfloat y, GLfloat z)
```

- GL_POINTS: per definire un insieme di primitive da interpretare e disegnare come punti.

```
glBegin(GL_POINTS)
  glVertex3f(0.0f,0.0f,0.0f);
  glVertex3f(10.0f,10.0f,10.0f);
glEnd()
```

Disegnare oggetti in 3D

- Per disegnare una linea, specificandone gli estremi, si utilizza la primitiva GL_LINES:

```
glBegin(GL_LINES)
  glVertex3f(0.0f,0.0f,0.0f);
  glVertex3f(10.0f,10.0f,10.0f);
glEnd()
```

- E' da notare che in questo caso due vertici descrivono una singola primitiva.
- Altre primitive:

  GL_TRIANGLE, GL_QUADS, GL_POLYGON

Esempio – Cubo 3D

```
class MyCube{
    float Color[3];
    float PPos;
    float MPos;
    float trans;
    public:
    float Rot;

    MyCube(float r, float g, float b, float rot, float mpos,
                                                  float  ppos){
       Color[0]=r;
       Color[1]=g;
       Color[2]=b;
       Rot=rot;
       MPos = mpos;
       PPos = ppos;
       trans= 0;
 }
```


Esempio – Cubo 3D

```
...

void DrawCube(){
      glTranslatef(0.0f,0.0f, ViewPort);
      glRotatef(Rot, 0.0f, 1.0f, 0.0f);
      glBegin(GL_QUADS);
          glColor3fv(Color);
          glVertex3f(PPos, PPos, MPos);
          glVertex3f(PPos, PPos, PPos);
          glVertex3f(PPos, MPos, PPos);
          glVertex3f(PPos, MPos, MPos);
       ...
       glEnd();
   }
};
```

Il cubo e' costruito utilizzando la primitiva GL_QUADS

Analogamente per le altre facce del cubo

Esempio – Cubo 3D

```
MyCube* c;

int main(int argc, char* argv[])
{
       glutInit(&argc, argv);
       glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
       glutCreateWindow("Esempio OpenGL");
       glutDisplayFunc(RenderScene);
       MyInit();
       glutMainLoop();
       delete c;
       return 0;
}
```

Esempio – Cubo 3D

```
void MyInit()
{
	glClearColor(1,1,1,1);
	c = new MyCube(1,0,1,50,-1,1);
	glEnable(GL_DEPTH_TEST);
	glMatrixMode(GL_PROJECTION);
	gluPerspective(40.0, 1.0, 1.0,10.0);
	glMatrixMode(GL_MODELVIEW);
	gluLookAt(0.0, 5.0, 5.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.);

}
void RenderScene()
{
	glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
	c->DrawCube();
	glutSwapBuffers();
}
```

Proiezione prospettica

Pipeline delle trasformazioni geometriche

- Esistono 3 tipi di trasformazioni che mettono in relazione i punti nello spazio 3D e cio' che appare sullo schermo:
  - Viewing
  - Modeling
  - Projection
- Su definiscono "Eye coordinates" le coordinate dal punto di vista dell'osservatore, indipendentemente dalle trasformazioni che avvengono.

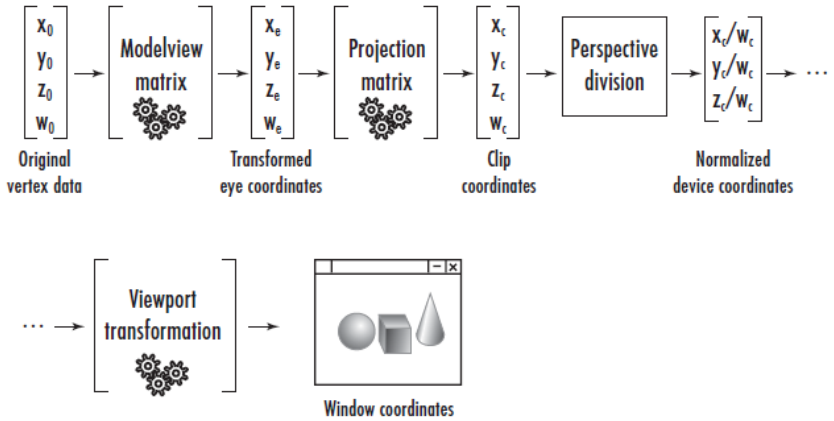
Pipeline delle trasformazioni geometriche

- ***Viewing transformation:*** stabilisce il punto di osservazione della scena. E' come posizionare una camera nella scena. Specificando tale trasformazione si definisce un nuovo sistema di coordinate, rispetto al quale avverranno tutte le successive trasformazioni.
- ***Modeling transformations:*** sono usate per manipolare il modello e gli oggetti all'interno di esso. Queste trasformazioni muovono, ruotano e scalano gli oggetti. Nota: dato che ogni trasformazione e' calcolata rispetto all'ultima trasformazione effettuata, l'aspetto finale della scena puo' dipendere dall'ordine in cui le trasformazioni sono effettuate.

Pipeline delle trasformazioni geometriche

- ***Dualita' modelview***: dal momento che i movimenti del punto di osservazione e degli oggetti nella scena avvengono in maniera relativa, le due trasformazioni (viewing e modeling) sono combinate in un'unica matrice chiamata *modelview*.
- ***Projection transformation:*** specifica come la scena e' proiettata sullo schermo. Determina il *viewvolume* e come sono effettuate le proiezioni.
- ***Viewport transformation:*** e' il mapping alle coordinate schermo fisiche.

Pipeline delle trasformazioni geometriche



Colori

- I colori in OpenGL sono definiti utilizzando la funzione

```
void glColor4f(float red, float green, float
                                  blue,float alpha)
```

- La componente `alpha` specifica la traslucenza del colore.
- La funzione `glColor` setta il colore per tutti gli oggetti che seguono. E' tuttavia possibile specificare diversi colori per i diversi vertici di una linea o un poligono. Il colore risultante dipendera' dal modello di shading adottato.
- Con il *smooth shading* (`GL_SMOOTH`) il passaggio da un colore all'altro avviene seguendo una retta nel cubo RGB.
- Con il *flat shading* (`GL_FLAT`) non viene calcolato il shading e la primitiva viene disegnata utilizzando il colore specificato per l'ultimo vertice (eccetto GL_POLYGON).

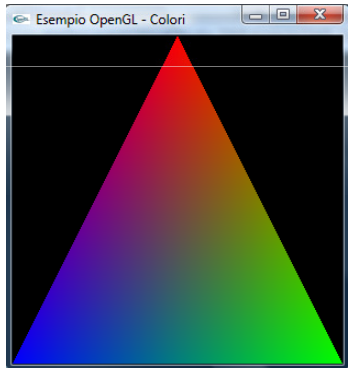
Colori - Esempio

```
void RenderScene()
{
	glClear(GL_COLOR_BUFFER_BIT);
	glShadeModel(GL_SMOOTH);
	glBegin(GL_TRIANGLES);
		glColor3f(1,0,0);
		glVertex3f(0.0f,1.0f,0.0f);
		glColor3f(0,1,0);
		glVertex3f(1.0f,-1.0f,0.0f);
		glColor3f(0,0,1);
		glVertex3f(-1.0f,-1.0f,0.0f);
	glEnd();
	glFlush();
}

void MyInit()
{
	glClearColor(0,0,0,1);
}
```


Colori - Esempio

```
int main(int argc, char* argv[])
{
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB );
  glutCreateWindow("Esempio OpenGL – Colori");
  glutDisplayFunc(RenderScene);
  MyInit();
  glutMainLoop();
  return 0;
}
```



Illuminazione della scena

- E' possibile approssimare le condizioni di illuminazione del mondo reale, utilizzando 3 modelli di illuminazione: *ambient*, *diffuse* e *specular*. Un effetto realistico e' ottenuto unendo opprotunamente queste tre componenti.
- La *ambient light* non e' caratterizzata da una particolare direzione. Gli oggetti risultano illuminati su tutte le superfici e in tutte le direzioni.
- La *diffuse light* ha una componente direzionale ed e' riflessa dalle superfici in maniera proporzionale alll'angolo con cui la superficie incide la superficie.
- La *specular light* ha una componente direzionale e viene riflessa dalle superfici in un'unica direzione.

Texture mapping

- E' possibile mappare immagini bidimensionali alle primitive definite in OpenGL (e quindi anche agli oggetti tridimensionali) in maniera tale da aumentare ulteriormente il realismo. Questo procedimento e' noto come *texture mapping*.
- Dopo aver applicato una texture ad una primitiva, OpenGL si occupa di effettuare le modifiche necessarie ad un corretto rendering (ad esempio dopo aver effettuato una trasformazione).
- Una texture puo' essere "incollata" ad una superficie o usata per modulare il colore presente nella superficie.

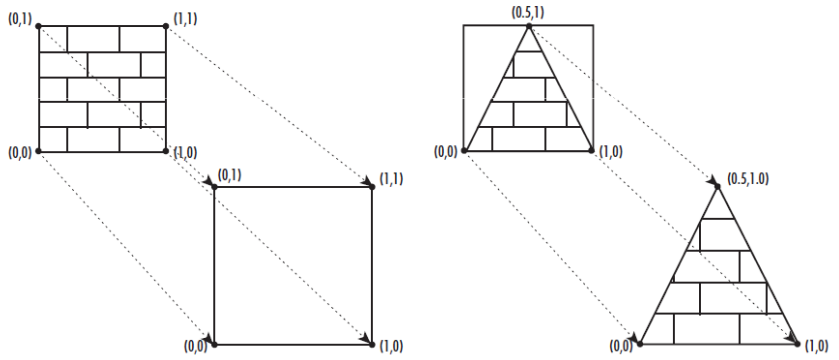
Texture mapping

Per applicare una texture ad una superficie sono necessari i seguenti passi:

- Creare una texture (ad esempio a partire da un'immagine) e una primitiva.
- Indicare come la texture deve essere applicata a ciascun pixel (ad esempio con GL_DECAL la texture e' "incollata" sulla superficie)
- Abilitare il texture mapping `glEnable(GL_TEXTURE_2D)`.
- Disegnare la scena, indicando la texture e le coordinate geometriche.

Texture mapping

E' necessario specificare le coordinate della texture per ogni vertice della primitiva su cui vogliamo mappare la texture.



Texture - esempio

```
int main(int argc, char** argv)
{
   glutInit(&argc, argv);
   glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);
   glutInitWindowSize(250, 250);
   glutInitWindowPosition(100, 100);
   glutCreateWindow("Texture mapping");
   MyInit();
   glutDisplayFunc(RenderScene);
   glutMainLoop();
   return 0;
}
```


Texture - esempio

```
void MyInit(){
   glClearColor (0.0, 0.0, 0.0, 0.0);
   glShadeModel(GL_FLAT);
   glEnable(GL_DEPTH_TEST);
   glGenTextures(1, &texName);
   glBindTexture(GL_TEXTURE_2D, texName);
   glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
   glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
   glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
   glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
   CMyImage c("eye.pgm");
   GLubyte* data = new GLubyte[c.GetCol()*c.GetRow()*3];
   for (int i=0, j=0; i< c.GetCol()*c.GetRow(); i++,j=j+3){
        data[j]=c.GetData()[i];
        data[j+1]=c.GetData()[i];
        data[j+2]=c.GetData()[i];
   }
   glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, c.GetCol(), c.GetRow(),
                               0, GL_RGB, GL_UNSIGNED_BYTE, data);
   delete [] data;
}
```

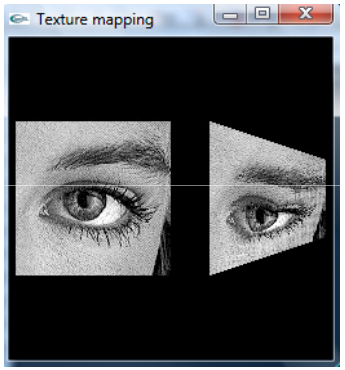

Texture - esempio

```
void RenderScene(){
   glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
   glEnable(GL_TEXTURE_2D);
   glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_DECAL);

   glBegin(GL_QUADS);
   glTexCoord2f(1.0, 1.0); glVertex3f(-2.0, -1.0, 0.0);
   glTexCoord2f(1.0, 0.0); glVertex3f(-2.0, 1.0, 0.0);
   glTexCoord2f(0.0, 0.0); glVertex3f(0.0, 1.0, 0.0);
   glTexCoord2f(0.0, 1.0); glVertex3f(0.0, -1.0, 0.0);

   glTexCoord2f(1.0, 1.0); glVertex3f(0.5, -1.0, 0.0);
   glTexCoord2f(1.0, 0.0); glVertex3f(0.5, 1.0, 0.0);
   glTexCoord2f(0.0, 0.0); glVertex3f(2, 0.5, 0);
   glTexCoord2f(0.0, 1.0); glVertex3f(2, -0.5, 0);   glEnd();
   glFlush();
   glDisable(GL_TEXTURE_2D);
}
```


Texture - esempio



Imaging

- OpenGL supporta la lettura/scrittura da e sul color buffer.
- Pertanto e' possibile leggere dal buffer mediante la funzione:

```
glReadPixels(GLint x, GLint y, GLsizei
width, GLsizei height, GLenum format,
GLenum type, const void *pixels)
```

e, ad esempio, salvarne il contenuto su file o leggere i pixel di un'immagine da un file e ridirigerli sul color buffer, mediante la funzione :

```
glDrawPixels(GLsizei width, GLsizei
height, GLenum format, GLenum type, const
void *pixels)
```


Imagin - Esempio

```
int main(int argc, char* argv[]){
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB );
  glutInitWindowSize(200 ,200);
  glutCreateWindow("Esempio OpenGL – Imaging");
  glutDisplayFunc(RenderScene);
  MyInit();
  glutMainLoop();
  return 0;
}
```


Imaging - Esempio

```
void MyInit(){
      glClearColor(0,0,0,1);
      //lettura da file dell'immagine
}

void RenderScene(){
  GLint iViewport[4];
  glClear(GL_COLOR_BUFFER_BIT);
  glRasterPos2i(-1,1);
  glGetIntegerv(GL_VIEWPORT, iViewport);
  glPixelZoom((GLfloat) iViewport[2] / width,
                  -(GLfloat)iViewport[3] / (GLfloat)height);
  glDrawPixels(height, width,GL_LUMINANCE, GL_UNSIGNED_BYTE,
                                              pImage);
  glutSwapBuffers();
}
```

Per riscalare l'immagine alle dimensioni della finestra, e invertire l'ordine sull'asse y