

Corso di
Informatica Grafica

Introduzione
a
OpenGL

parte 1

Introduzione

- OpenGL è una specifica che definisce una API:
 - Una libreria grafica 3D
 - Uno standard grafico
 - Indipendente dal sistema operativo
 - Indipendente dal sistema di window management
 - Indipendente dall'hardware
- Non tratta la gestione di eventi (mouse, tastiera, window management) se non in modo molto “semplice”
- È una API che richiede la programmazione in C
- È una macchina a stati

Introduzione

- OpenGL è un'evoluzione di IRIS GL, l'interfaccia 3D di SGI (Silicon Graphics Inc.).
- Nato per supportare via software le caratteristiche non supportate dall'hardware grafico.
- Offre al programmatore una API unica ed uniforme.
- Nasconde l'interfacciamento con acceleratori 3D.
- Il compito di OpenGL è quello di ricevere primitive come punti, linee e poligoni, e di convertirli in pixel (rasterizzazione) attraverso una pipeline grafica.
- Riferimento: OpenGL Programming Guide (Addison-Wesley Publishing Company) o “The Red Book”.

Introduzione

- 1992 OpenGL 1.0
- 1997 OpenGL 1.1 – vertex array
- 1998 OpenGL 1.2 – 3d textures
- 2001 OpenGL 1.3 – cube maps, multitexturing
- 2002 OpenGL 1.4 – automatic mipmap generation
- 2003 OpenGL 1.5 – vertex and fragment programs
- 2004 OpenGL 2.0 – GL shading language
- 2006 OpenGL 2.1 – GLSL changes
- 2008 OpenGL 3.0 – vengono deprecate molte funzioni originali
- 2009 OpenGL 3.1 – buffer objects
- 2010 OpenGL 4.0 – tessellation shaders

Introduzione

- Componenti di OpenGL

OpenGL è costituita da alcune librerie:

- gl nucleo di OpenGL
- glu OpenGL utility library
- glut OpenGL utility toolkit

- gl

- Rendering 3D (o 2D)
- Lighting
- Hidden surface removal
- Stenciling
- Alpha blending
- Antialiasing
- Texture mapping
- Fog
- ...

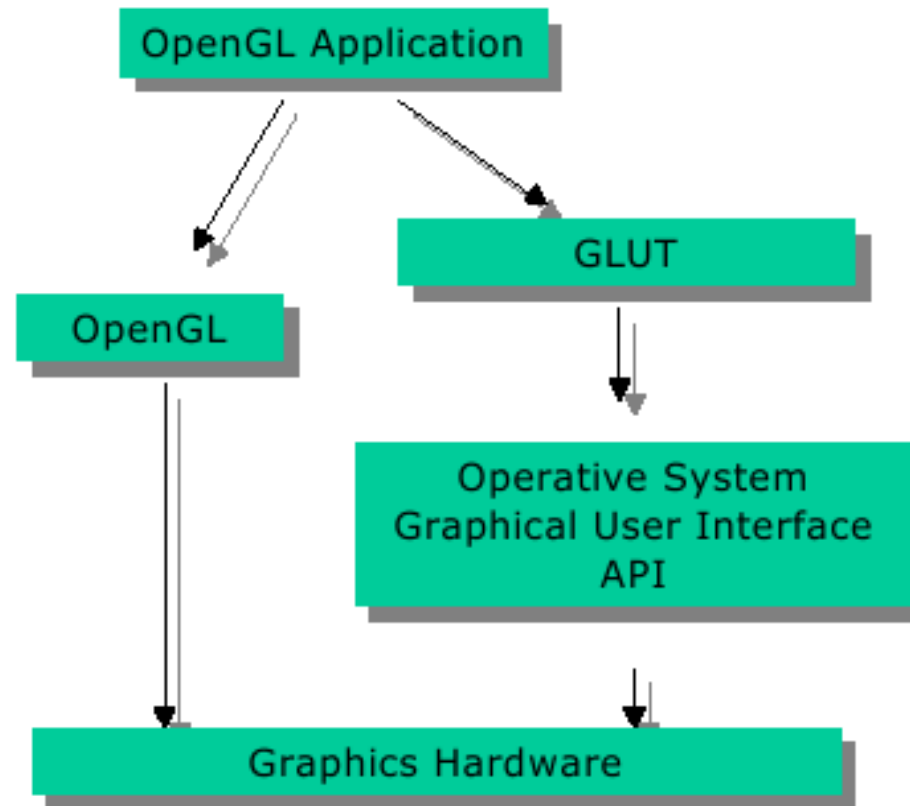
Introduzione

- **glu** (insieme di funzioni utility)
 - gestione parametri viewing
 - gestione texture mapping
 - polygon tessellation
(decompositore generico di poligoni concavi)
 - curve e superfici parametriche
 - gestione errori
- **glut** (toolkit di interfaccia sistema operativo)
 - Funzionalità base di window management (pur conservando l'indipendenza dal s.o.)
 - Gestione di eventi (mouse, tastiera)
 - Primitive 3D (cubi, sfere, tori...)
 - Permette di creare in modo semplice e rapido un contesto in cui disegnare con OpenGL (utile se si vogliono testare porzioni di programmi più complessi o routine grafiche indipendenti)

OpenGL vs DirectX

Feature:	OpenGL	DirectX
Sviluppo	consorzio di aziende non-profit “Khronos Group”	Microsoft
Interfaccia	chiamate procedurali	COM
Aggiornamenti	Annuali	Annuali
portabilità	multiplatforma	solo piattaforme Microsoft
Semplicità d'uso	+	-
Prestazioni	+	+
Utilizzo prevalente	professionale	videogames

Introduzione



Introduzione

- Formato comandi

<libreria><comando><signature>

<libreria> = **gl** o **glu** o **glut** o **wgl** o...

<comando> = ad es. **Color**, **Vertex**...

<signature> = numero di argomenti +
lettera distintiva del tipo di
argomenti (distinguono diverse
implementazioni di una
funzionalità)

es. : **glColor3f**(...), **glVertex2i**(...)

gluLookAt(...) (qui non c'è signature)

- Tipi di dato

- **GLbyte** b 8 bit

- **GLubyte** ub 8 bit

- **GLshort** s 16 bit

- **GLushort** us 16 bit

- **GLint** i 32 bit

- **GLuint** ui 32 bit

- **GLfloat** f 32 bit

- **GLdouble** d 64 bit

Istruzioni di base GLUT

void **glutInit**(int *argc, char **argv)

(prima istruzione da chiamare, opzionale se usate windows)

void **glutInitDisplayMode**(unsigned int *mask*)

inizializza il modo di visualizzazione di un contesto OpenGL creato con glut

void **glutInitWindowPosition**(int *x*, int *y*)

determina in che posizione dello schermo verrà creata la finestra di glut (angolo superiore sinistro)

void **glutInitWindowSize**(int *width*, int *height*)

inizializza la dimensione della finestra da creare

int **glutCreateWindow**(char* *title*)

crea la finestra con le impostazioni precedenti e restituisce un identificatore della finestra (utile nella gestione di programmi con finestre multiple)

Istruzioni di base gl

void glClearColor(GLfloat *red*, GLfloat *green*, GLfloat *blue*, GLfloat *alpha*)
setta il colore con cui “pulire” il color buffer (valori di default 0.0)

void glBegin(GLenum *mode*)
delimita i vertici di una “primitiva”.E' usato congiuntamente a **glEnd()**

void glVertex2f(GLfloat *x*, GLfloat *y*)
definisce un vertice di una primitiva. Le funzioni glVertex* possono essere chiamate solo da glBegin/glEnd. Esistono versioni che accettano 1,2,3 o 4 argomenti

void glFlush()
richiede che si inizi l'esecuzione di tutti gli eventuali comandi OpenGL ancora in sospeso, cioè che le operazioni richieste vengano terminate in un tempo finito

Istruzioni di base gl

`void glFinish()`

come nel caso precedente, ma non ritorna finchè l'esecuzione di tutti i comandi è stata completata

Tipi di primitiva di OpenGL (argomento di glBegin)

OpenGL prevede dieci costanti che determinano altrettanti modi con cui costruire una primitiva (punti, linee, poligoni). Le differenze riguardano il modo e l'ordine in cui i vertici vengono collegati (glVertex*):

GL_POINTS

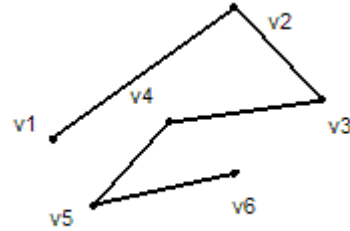


GL_LINES

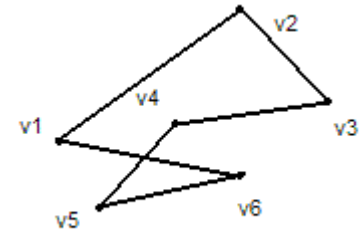


Funzionalità di base

GL_LINE_STRIP

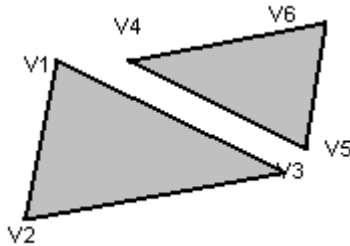


GL_LINE_LOOP

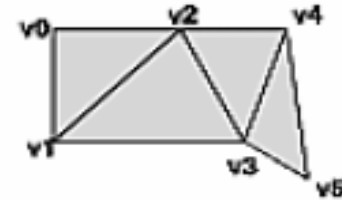


GL_TRIANGLES

verso antiorario



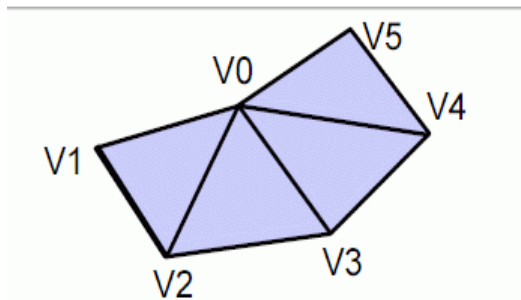
GL_TRIANGLE_STRIP



i triangoli vengono disegnati automaticamente considerando i vertici nel seguente ordine v0, v1, v2, poi v2, v1, v3, poi v2, v3, v4 eccetera. Tale ordine assicura che i triangoli vengano disegnati con la stessa orientazione (verso antiorario)

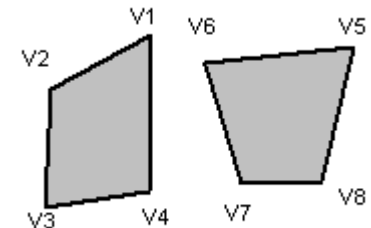
GL_TRIANGLE_FAN

verso antiorario



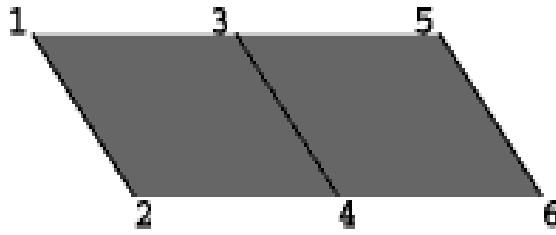
GL_QUADS

verso antiorario



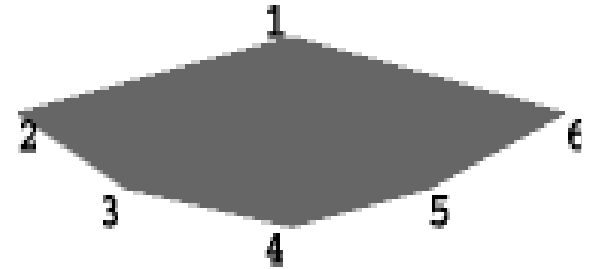
Funzionalità di base

GL_QUAD_STRIP



GL_POLYGON

verso antiorario



Tutti i poligoni devono essere convessi, con vertici appartenenti allo stesso piano, con perimetro che non si interseca con se stesso, con un solo perimetro (non sono ammessi poligoni con perimetro esterno ed interno)

Schema di un programma OpenGL

Un programma C/OpenGL ha tipicamente le seguente struttura:

.....

funzioni callback che contengono codice scritto dal programmatore

.....

main ()

{

 inizializzazione

 creazione delle finestre (almeno una)

 registrazione di callback per gli eventi di interesse nelle finestre

 eventuale creazione di menu' (e sottomenu')

 innesco del "main loop" (**glutMainLoop**)

}

La funzione **glutMainLoop** attiva il ciclo di ascolto di eventi; tale funzione viene richiamata solo una volta e ha lo scopo di gestire tutti gli eventi per i quali esiste un callback function che sia stata registrata.

Compilazione

Linux

```
gcc -o esempio1 esempio1.cpp -lglut -lGL -lGLU
```

la libreria GLUT puo' essere installata dal pacchetto “freeglut”

Windows

con MS VISUAL STUDIO

- 1) Creare un progetto (“console application”) **vuoto**
- 2) Impostare le librerie statiche da linkare glut32.lib opengl32.libglu32.lib nelle impostazioni del progetto (sottomenu linker->input)
- 3) Caricare un file sorgente nel progetto e compilare

Esempio1: creazione di una finestra

```
#include<GL/glut.h>
#include<GL/glu.h>
#include<GL/gl.h>
void redraw(){
glClearColor(0.0f,0.0f,0.0f);
glClear(GL_COLOR_BUFFER_BIT);
glBegin(GL_POINTS);
glVertex2f(0.0f,0.0f);
glEnd(); glFlush();
}

void main(int argc, char** argv)
{
glutInitDisplayMode(GLUT_SINGLE);
glutInitWindowPosition(100,200);
glutInitWindowSize(320,200);
glutCreateWindow("esempio 1");

glutDisplayFunc(redraw);
glutMainLoop();
}
```

Il colore

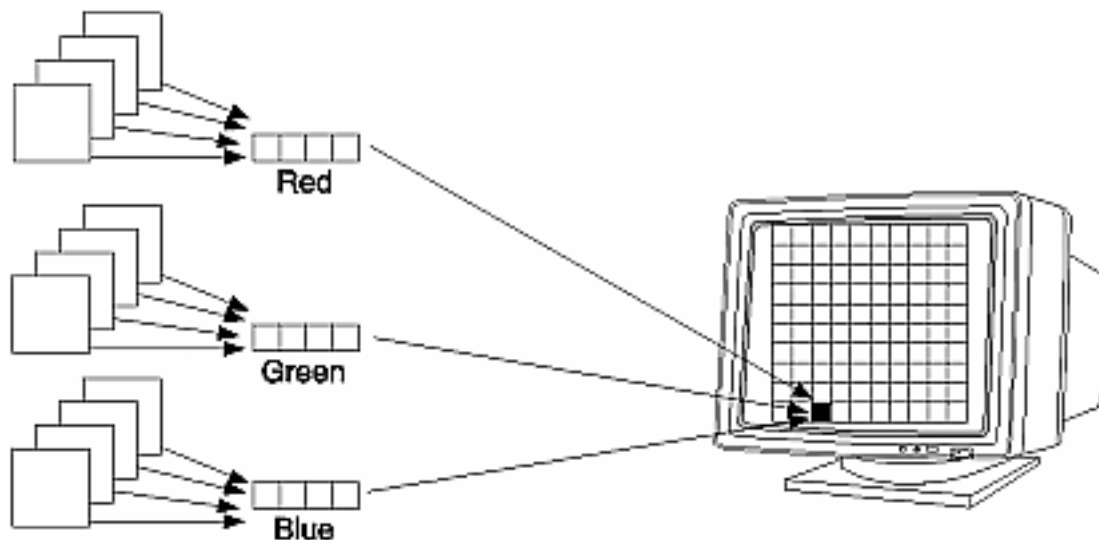
- OpenGL prevede due modi per specificare il colore:
 - Modalità indicizzata
 - Modalità RGB
- Per attivare la prima è necessario:
 - Selezionare la modalità con : **glutInitDisplayMode**(GLUT_INDEX)
 - Definire i colori che si vogliono nella “tavolozza” con:
`void glutSetColor(GLint index,GLfloat red,GLfloat green,GLfloat blue)`
 - Definire il colore per il clearing del color buffer con
`void glClearIndex(GLint index)`
 - Selezionare il colore corrente con :
`void glIndexi(GLint index)`

Il colore

- Per attivare la modalità RGB è necessario:
 - Selezionare la modalità con: **glutInitDisplayMode(GLUT_RGB)**
 - Settare il colore per il clearing con :
`void glClearColor(GLfloat red, GLfloat green, GLfloat blue)`
 - Settare il colore corrente con :
`void glColor3f(GLfloat red, GLfloat green, GLfloat blue)`
oppure
`void glColor4f(GLfloat red, GLfloat green, GLfloat blue, GLfloat alpha)`
o altre versioni di glColor
- In generale la modalità RGB permette un numero maggiore di colori, è la sola che funziona con le texture, funziona meglio con effetti quali fog, alpha blending antialiasing...

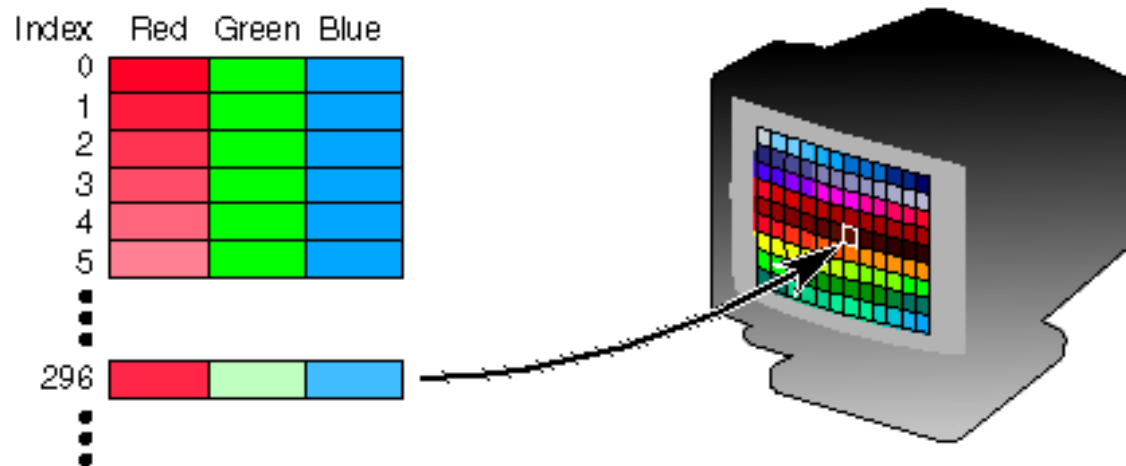
Gestione del colore in modalità RGB

In modalità RGB l'hardware riserva un certo numero di bit alla memorizzazione delle componenti rossa, verde e blu di ciascun colore . Solitamente i valori delle componenti R G B sono memorizzati come interi e poi convertiti in numeri floating point compresi tra 0 e 1. Ad es. in un sistema RGB a 24 bit solitamente sono riservati 8 bit a ciascun colore (i bit sono memorizzati in bitplanes, una bitplane è un'area di memoria che contiene un bit di informazione per ogni pixel sullo schermo), quindi 8 bit per la componente R significa che tale componente potrà assumere $2^8 = 256$ valori diversi (tonalità di R). I corrispondenti valori float corrisponderanno quindi a $0.0 = 0/255 \dots 255/255 = 1.0$



Gestione del colore in modalità indicizzata

In modalità indicizzata OpenGL usa una mappa dei colori (lookup table), simile alla tavolozza di un pittore, in cui prima di iniziare a dipingere vengono preparati i colori necessari a partire dai tre colori fondamentali R G B. Ad ogni colore preparato viene assegnato un indice intero utilizzato per recuperare il colore corrispondente. L'indice del colore viene memorizzato nelle bitplanes di ciascun pixel. Il numero di colori disponibili è quindi limitato dalla dimensione della color map e dal numero di bitplanes. La dimensione della color map è sempre una potenza di 2, se contiene 2^n indici e i bitplanes disponibili sono m , il numero di colori effettivamente disponibili è $\min(2^n, 2^m)$



Modalità
indicizzata

Modalità RGB

Esempio 2 : il colore



```
void redraw( void ){
glutSetColor(0,0.0,0.4,0.0);
glutSetColor(1,1.0,0.0,1.0);
glPointSize(3.0);
glClearIndex(0);
glClear(GL_COLOR_BUFFER_BIT);
glBegin(GL_POINTS);
glIndexi(1);
glVertex2f(0.0, 0.0);
glEnd();glFlush();
}
```

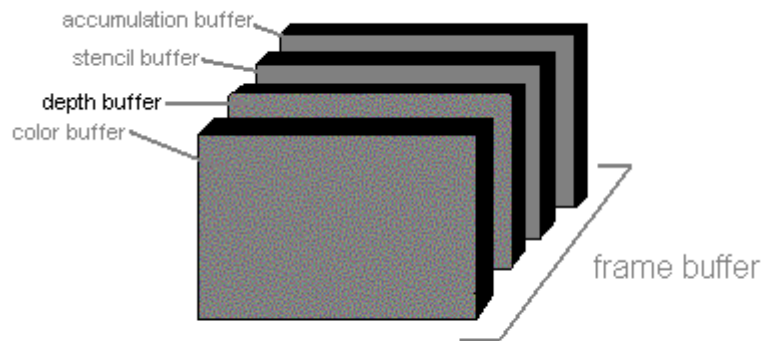


```
void redraw( void ){
glPointSize(3.0);
glClearColor(0.0,0.4,0.0,1.0);
glClear(GL_COLOR_BUFFER_BIT);
glBegin(GL_POINTS);
glColor3f(1.0,0.0,1.0);
glVertex2f(0.0,0.0);
glEnd();
glFlush();
}
```

Frame Buffer

L'insieme della memoria che OpenGL utilizza per immagazzinare informazioni riguardo a quante e quali parti disegnare del modello va sotto il nome di Frame Buffer ed è costituito da:

- Color : contiene i dati relativi al colore di ciascun pixel su schermo. In esso finisce una il risultato di una parte delle chiamate della funzione di drawing (redraw negli esempi precedenti)
- Depth : utilizzato nella rimozione delle superfici nascoste
- Stencil : utilizzato per limitare il drawing ad aree di forma qualsiasi
- Accumulation : utilizzato per ottenere effetti particolari come motion blur



Frame Buffer

Ogni implementazione i OpenGL determina la dimensione del Frame Buffer. Per conoscere la dimensione in bits per pixel di ogni buffer si utilizza il comando

void glGetIntegerv(*GLenum pname, GLint* params*)

che accetta numerosi parametri per il campo *pname* tra cui:

GL_RED_BITS, GL_GREEN_BITS,

GL_BLUE_BITS, GL_ALPHA_BITS Numero di bits per le componenti R, G, B, o A del colore

GL_INDEX_BITS Numero di bits per indice nel color buffer

GL_DEPTH_BITS Numero di bits per pixel nel depth buffer

GL_STENCIL_BITS Numero di bits per pixel nello stencil buffer

GL_ACCUM_RED_BITS, GL_ACCUM_GREEN_BITS,

GL_ACCUM_BLUE_BITS, GL_ACCUM_ALPHA_BITS Numero di bits per le component R, G, B, o A del accumulation buffer

Buffer Clearing

- Solitamente prima di iniziare a disegnare si inizializza il contenuto di alcuni buffer con un valore predefinito con le chiamate:

void **glClearColor**(GLfloat *red*, GLfloat *green*, GLfloat *blue*, GLfloat *alpha*)

per Color Buffer (se si è in modalità RGB, altrimenti si usa **glClearIndex**)

void **glClearDepth**(GLdouble *depth*)

per Depth Buffer

void **glClearStencil**(GLint *s*)

per Stencil Buffer

void **glClearAccum**(GLfloat *red*, GLfloat *green*, GLfloat *blue*, GLfloat *alpha*)

per Accumulation Buffer

- Infine con void **glClear**(GLbitfield *mask*) si inizializza il contenuto del buffer

Buffer Clearing (2)

mask può valere:

- GL_COLOR_BUFFER_BIT
- GL_DEPTH_BUFFER_BIT
- GL_STENCIL_BUFFER_BIT
- GL_ACCUMULATION_BUFFER_BIT

eventualmente le costanti sopraelencate possono essere sommate con un'operazione di “or” bit a bit (operatore |). Ad esempio

```
glClearColor( 0.0,0.0,1.0,1.0 );
```

```
glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );
```

resetta il contenuto del color buffer (lo sfondo della finestra, per intenderci) al valore blu e il contenuto del depth buffer al valore di default (1.0)

OpenGL: una macchina a stati

OpenGL è una macchina a stati in quanto le operazioni eseguite nel processare dati geometrici e pixel sono guidate dallo stato in cui si trovano i parametri interni, variando i parametri si ottengono risultati differenti. I valori dei parametri sono modificabili attraverso opportune funzioni ed è quindi possibile cambiare lo stato della macchina in ogni punto del codice. Questo significa che durante le operazioni di disegno si può abilitare e disabilitare più volte una determinata opzione.

Esempio (modifica della variabile di stato colore)

```
set_current_color(red);  
draw_object(A); // oggetto A disegnato in rosso  
draw_object(B); // oggetto B disegnato in rosso  
set_current_color(green);  
set_current_color(blue);  
draw_object(C); // oggetto C disegnato in blu
```

Esempio 3: linee

```
void redraw( void ){
    glClearColor(0.0,0.0,0.0,1.0);
    glClear(GL_COLOR_BUFFER_BIT);
    glBegin(GL_LINES);
    glColor3f(1.0,1.0,1.0);
    glVertex2f(-0.8,-0.8);
    glColor3f(1.0,0.0,0.0 );
    glVertex3f(0.5,0.2,-0.3);
    continua...
    glVertex4f(-0.5,0.4,0.7,1.0);
    glColor3f(0.0,1.0,0.0);
    // 3.0 divide x,y e z
    glVertex4f(-0.5,0.4,0.7,3.0);
    glEnd();
    glFlush();
}
```

glVertex*() accetta al piu' 4 parametri (x,y,z,w), se non si specificano z=0 e w=1. Se si specifica w il suo valore divide (x,y,z)

Esempio 4: Triangle Strip

```
void redraw( void ){
    glClearColor(0.0,0.0,0.0,1.0);
    glClear(GL_COLOR_BUFFER_BIT);
    glBegin(GL_TRIANGLE_STRIP);
    glColor3f(1.0,1.0,1.0);
    glVertex2f(-0.8,-0.8);
    glColor3f(1.0,0.0,0.0);
    glVertex2f(0.5,0.2);
    glColor3f(0.0,1.0,0.0);
    glVertex2f(-0.16,0.11);
    glColor3f(0.0,0.0,1.0);
    glVertex2f(-0.5,0.6);
    glVertex2f(-0.4,0.0);
    glEnd();
    glFlush();
}
```

Esempio 5: Polygon

```
void redraw( void ){
    glClearColor(0.0,0.0,0.0,1.0);
    glClear(GL_COLOR_BUFFER_BIT);
    glBegin(GL_POLYGON);
    glColor3f(0.0,1.0,0.0);
    glVertex2f(0.8,0.2);
    glColor3f(1.0,1.0,0.0);
    glVertex2f(0.7,0.5);
    glColor3f(0.0,0.0,1.0);
    glVertex2f(-0.5,0.6);
    glColor3f(1.0,1.0,0.0);
    glVertex2f(-0.7,0.0);
    glColor3f(1.0,1.0,1.0);
    glVertex2f(-0.5,-0.6);
    glColor3f(1.0,1.0,1.0);
    glVertex2f(0.5,-0.5);
    glEnd();glFlush();
}
```

Cosa accade se inserisco **glShadeModel(GL_FLAT)** prima di iniziare a disegnare?

Colore delle primitive OpenGL

Le primitive OpenGL possono essere disegnate in modalità “flat” o “smooth” (defaults). Lo smooth shading disegna l'intera primitiva (es: poligono) interpolando il colore dei vertici. Il flat shading disegna l'intera primitiva con un solo colore selezionato dal colore di uno dei vertici secondo la seguente tabella (con $i \geq 1$)

<i>primitive type of polygon i</i>	<i>vertex</i>
Single Polygon ($i \equiv 1$)	1
Triangle strip	$i + 2$
Triangle fan	$i + 2$
Independent triangle	$3i$
Quad strip	$2i + 2$
Independent quad	$4i$

colore poligono = colore primo vertice
colore triangolo = colore del terzo vertice

glRect

`glRect[dfis](x1, y1, x2, y2)` disegna un rettangolo ($z=0$) con vertici opposti
(x1,y1), (x2,y2)

equivalente a

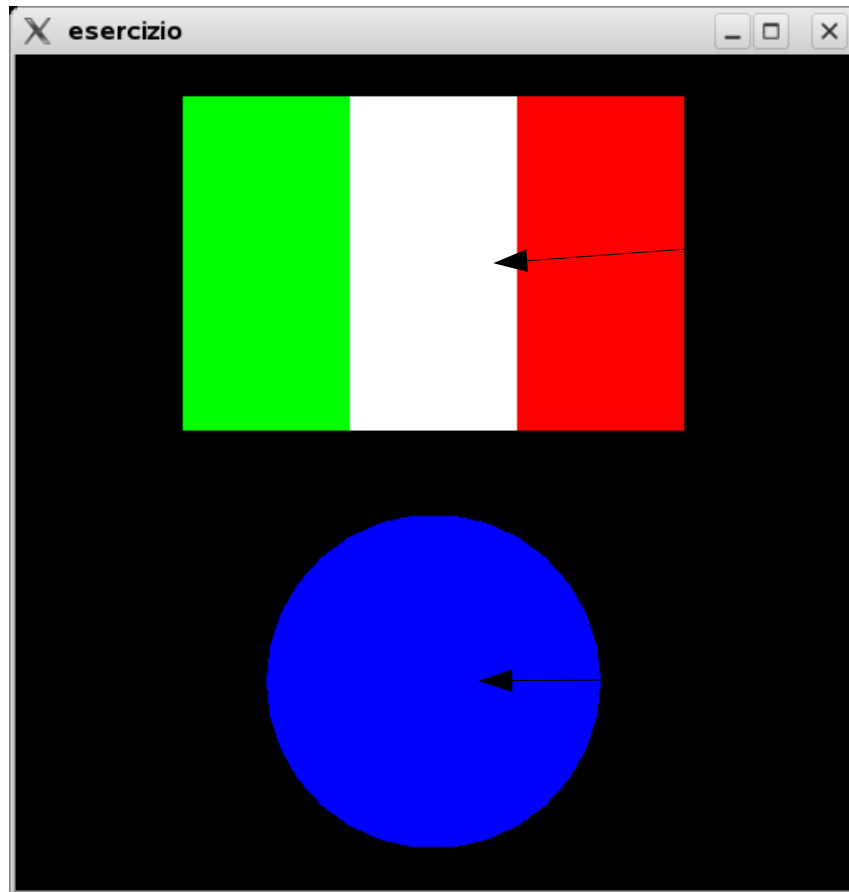
```
glBegin(GL_POLYGON);  
    glVertex2(x1, y1);  
    glVertex2(x2, y1);  
    glVertex2(x2, y2);  
    glVertex2(x1, y2);  
glEnd();
```

esempio

```
void glRectf(GLfloat x1,  
             GLfloat y1,  
             GLfloat x2,  
             GLfloat y2)
```

Esercizio

disegnare la bandiera dell'Italia e un cerchio blu sottostante.



GL_QUAD_STRIP

GL_TRIANGLE_FAN

Linee: dimensioni

E' possibile specificare lo spessore con cui le linee vengono disegnate con

`void glLineWidth(GLfloat width)`

dove *width* rappresenta lo spessore della linea, che per default vale 1.0. Lo spessore della linea viene (ovviamente) influenzato dalla risoluzione dello schermo (dal numero di pixel che formano lo schermo. Tale funzione influenza l'aspetto delle linee disegnate con GL_LINES, GL_LINE_STRIP, GL_LINE_LOOP.

Per conoscere con esattezza il valore dello spessore con cui vengono disegnate le linee e il range di valori ammessi OpenGL mette a disposizione la funzione

`void glGetFloatv(GLenum enum, float* value);`

esempio:

```
float curvalue, sizes[2];
```

```
glGetFloatv(GL_LINE_WIDTH, &curvalue);
```

```
glGetFloatv(GL_LINE_WIDTH_RANGE, sizes); // range di spessori ammesso
```

Esempio 6: dimensione della linea

```
void redraw() {  
    GLfloat y, fCurrSize;  
    GLfloat fSizes[2];  
    glGetFloatv(GL_LINE_WIDTH_RANGE, fSizes);  
    fCurrSize = fSizes[0];  
    for (y = -0.9f; y < 0.9f; y += 0.2f) {  
        glLineWidth(fCurrSize);  
        glBegin(GL_LINES);  
        glVertex2f(-0.8f, y);  
        glVertex2f(0.8f, y);  
        glEnd();  
        fCurrSize += 1.0f;  
    }  
}
```

Linee: tratteggio

Un'altra interessante caratteristica delle linee è il tratteggio, utile ad esempio in un contesto di sistema CAD in cui l'aspetto simbolico/funzionale prevale su quello realistico. Il tratteggio si abilita con :

void **glEnable**(GLenum *feature*) con feature=GL_LINE_STIPPLE

Una volta abilitato, è necessario specificare lo schema del tratteggio con :

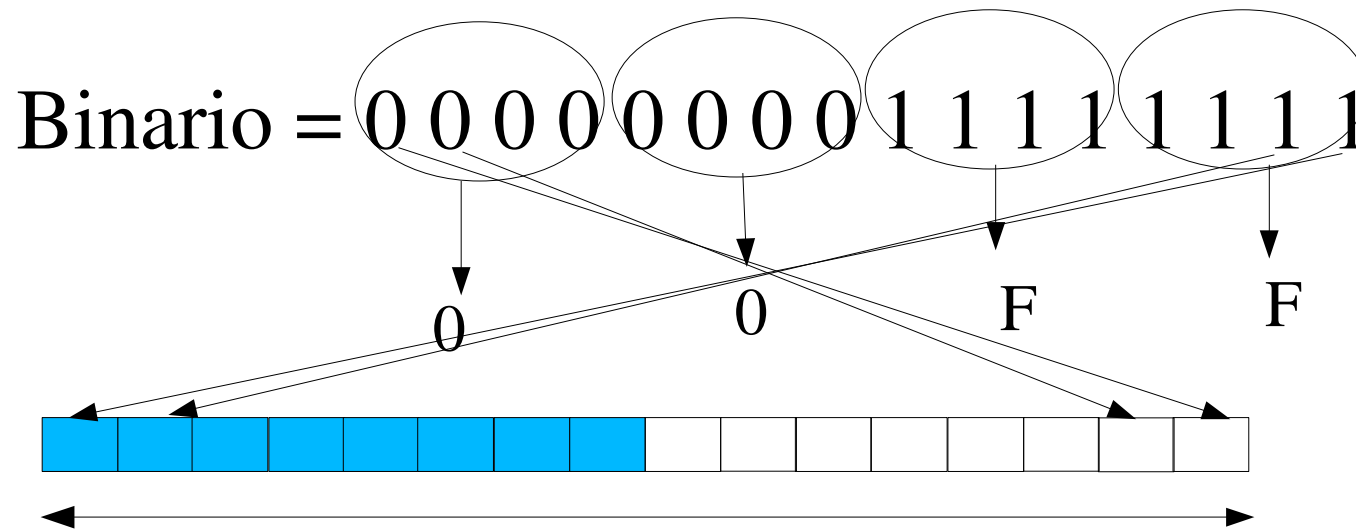
void **glLineStipple**(GLint *factor*, GLushort *pattern*)

in cui:

- *factor* specifica un moltiplicatore che specifica quanti pixel saranno modificati da ciascun bit di pattern. I valori accettati variano da 1 (default) a 255.
- *pattern* specifica lo schema del tratteggio come sequenza di 16 bit. Il tratteggio inizia col bit meno significativo. A questo parametro si assegna di solito un valore esadecimale corrispondente al numero binario che si vuole usare come schema

Creare il pattern per il tratterggio

Pattern = 0x00FF = 255



PATTERN	FACTOR
0x00FF	1
0x00FF	2
0x0C0F	1
0x0C0F	3
0xAAAA	1
0xAAAA	2
0xAAAA	3
0xAAAA	4

Esempio 7: tratteggio

```
void redraw() {  
    GLfloat y;  
    GLint factor=1;  
    GLushort pattern=0x5555;  
    glEnable(GL_LINE_STIPPLE);  
    for(y=-0.9f;y<0.9f;y+=0.2f) {  
        glLineStipple(factor,pattern);  
        glBegin(GL_LINES);  
        glVertex2f(-0.8f,y);  
        glVertex2f(0.8f,y);  
        glEnd();  
        factor++;  
    }  
}
```

Modi di disegnare i poligoni

Esistono diversi modi con cui disegnare i poligoni con OpenGL:

- Poligoni “pieni” GL_FILL
- Poligoni “wireframe” GL_LINE
- Soltanto i vertici GL_POINT

E' possibile inoltre indicare su quale faccia del poligono applicare il modo selezionato:

- Solo il davanti (senso antiorario dei vertici) GL_FRONT
- Solo il dietro (senso orario dei vertici) GL_BACK
- Entrambi GL_FRONT_AND_BACK

La funzione che permette di fare queste scelte è:

void glPolygonMode(GLenum *face*, GLenum *mode*)

Modi di disegnare i poligoni (2)

Se sappiamo a priori che di un solido non verrà mai visualizzata una certa parte (ad esempio l'interno), possiamo fare in modo che le superfici mai visibili non vengano mai disegnate => Surface Culling

glEnable(GL_CULL_FACE)

in tale modo la “back face” (vertici in senso orario) di ciascun poligono non è disegnata.

Se invece vogliamo disegnare un poligono “complesso”, costituito ad esempio da tanti triangoli, in modo tale che appaia come poligono “semplice”, possiamo indicare quale dei segmenti che costituiscono il poligono verrà disegnato e quale no, con la funzione

void glEdgeFlag(GLboolean *flag*)

Se un vertice viene marcato “true” viene considerato come vertice iniziale di un segmento visibile, altrimenti viene considerato come vertice iniziale di un segmento invisibile.

Esempio 8 : poligono “complesso”

```
void redraw() {  
    glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);  
    bool bEdgeFlag=false;  
    glBegin(GL_TRIANGLES);  
        glEdgeFlag(bEdgeFlag);  
        glVertex2f(-0.2f,0.0f);  
        glEdgeFlag(TRUE);  
        glVertex2f(0.2f,0.0f);  
        glVertex2f(0.0f,0.4f);  
        glVertex2f(-0.2f,0.0f);  
        glVertex2f(-0.6f,-0.2f);  
        glEdgeFlag(bEdgeFlag);  
        glVertex2f(-0.2f,-0.4f);  
        glEdgeFlag(TRUE);  
        glVertex2f(-0.2f,-0.4f);  
        glVertex2f(0.0f,-0.8f);  
        glEdgeFlag(bEdgeFlag);  
        glVertex2f(0.2f,-0.4f);  
        glEdgeFlag(TRUE);  
    glEnd();  
}
```

Esempio 8: poligono “complesso” (2)

```
glVertex2f(0.2f,-0.4f);      glVertex2f(-0.2f,-0.4f);
glVertex2f(0.6f,-0.2f);      glVertex2f(0.2f,-0.4f);
glEdgeFlag(bEdgeFlag);      glVertex2f(0.2f,0.0f);
glVertex2f(0.2f,0.0f);      glEdgeFlag(TRUE);

                             glEnd();

glVertex2f(-0.2f,0.0f);      glFlush();
glVertex2f(-0.2f,-0.4f);    }
glVertex2f(0.4f,0.0f);
glEdgeFlag(bEdgeFlag);
```

abilitare/disabilitare le caratteristiche di OpenGL

glEnable, glDisable

esempio: glEnable(GL_LIGHTING)

GL_ALPHA_TEST	Attiva il test del canale Alpha
GL_BLEND	Attiva funzione blending
GL_CLIP_PLANEi	Abilita piano di clipping (i = numero del piano)
GL_COLOR_MATERIAL	Tiene traccia del colore corrente per il materiale
GL_CULL_FACE	Attiva eliminazione facce nascoste
GL_DEPTH_TEST	Attiva il test della profondità
GL_LIGHTi	Attiva la sorgente luminosa i (0..GL_MAX_LIGHT-1)
GL_LIGHTING	Attiva il calcolo dell'illuminazione per i vertici
GL_LINE_SMOOTH	Disegna linee con antialiasing
GL_LINE_STIPPLE	Disegna le linee con pattern
GL_SCISSOR_TEST	Attiva rettangolo di clipping
GL_STENCIL_TEST	Attiva il test per lo Stencil Buffer

Introduzione alle primitive 3D

glut fornisce alcune primitive per generare solidi 3D:

```
void glutSolidSphere(GLdouble radius, GLint slices, GLint stacks);  
void glutWireSphere(GLdouble radius, GLint slices, GLint stacks); //slices =# linee di longitudine  
void glutSolidCube(GLdouble size); // stacks=# linee di latitudine  
void glutWireCube(GLdouble size);  
void glutSolidCone(GLdouble base, GLdouble height, GLint slices, GLint stacks);  
void glutWireCone(GLdouble base, GLdouble height, GLint slices, GLint stacks);  
void glutSolidTorus(GLdouble innerRadius, GLdouble outerRadius, GLint nsides, GLint rings);  
void glutWireTorus(GLdouble innerRadius, GLdouble outerRadius, GLint nsides, GLint rings);  
void glutSolidTeapot(GLdouble size);  
void glutWireTeapot(GLdouble size);
```

Per default l'osservatore è posto nell'origine ed è rivolto nella direzione negativa dell'asse z. Viene visualizzato tutto quello che e' contenuto in un cubo di lato 2 centrato nell'origine.

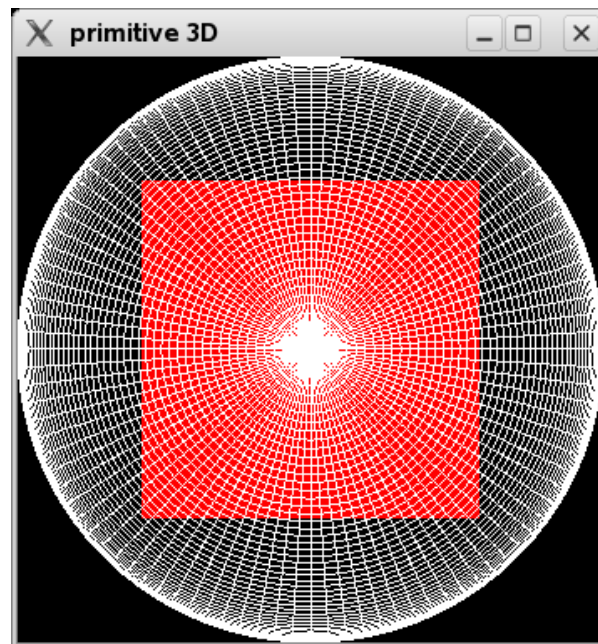
Esercizio

disegnare un cubo solido inscritto in una sfera wireframe di raggio 1 centrata nell'origine.

Nota: utilizzare le seguenti chiamate (nel main)

```
glutInitDisplayMode ( GLUT_RGB | GLUT_DEPTH );  
glEnable(GL_DEPTH_TEST); (prima di glutMainLoop)
```

e `glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);` nella funzione per il rendering della scena



cosa succede se la sfera ha un raggio maggiore di 1? perchè?