

Grafica Computazionale

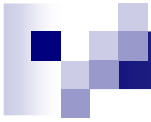
Laboratorio

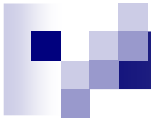
Fabio Ganovelli

fabio.ganovelli@gmail.com

a.a. 2006-2007

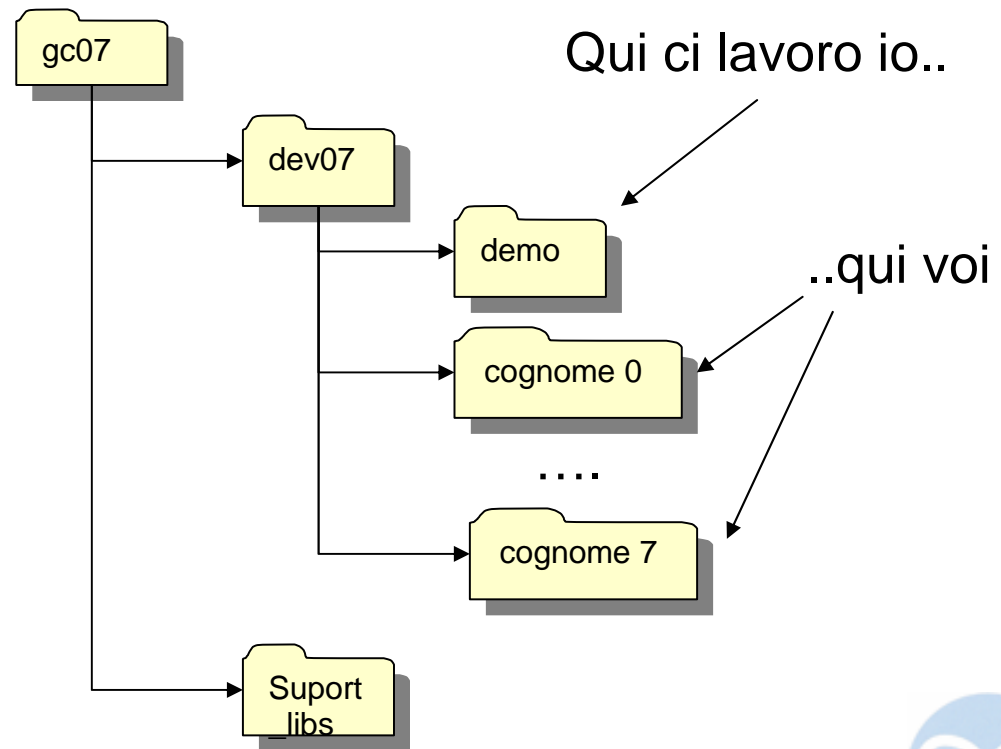
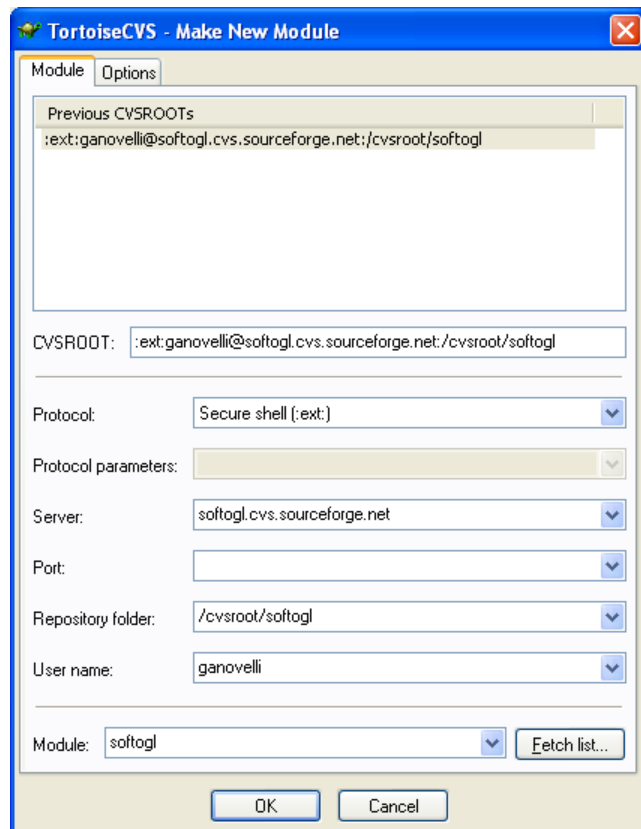
Credits: *Marco Tarini*



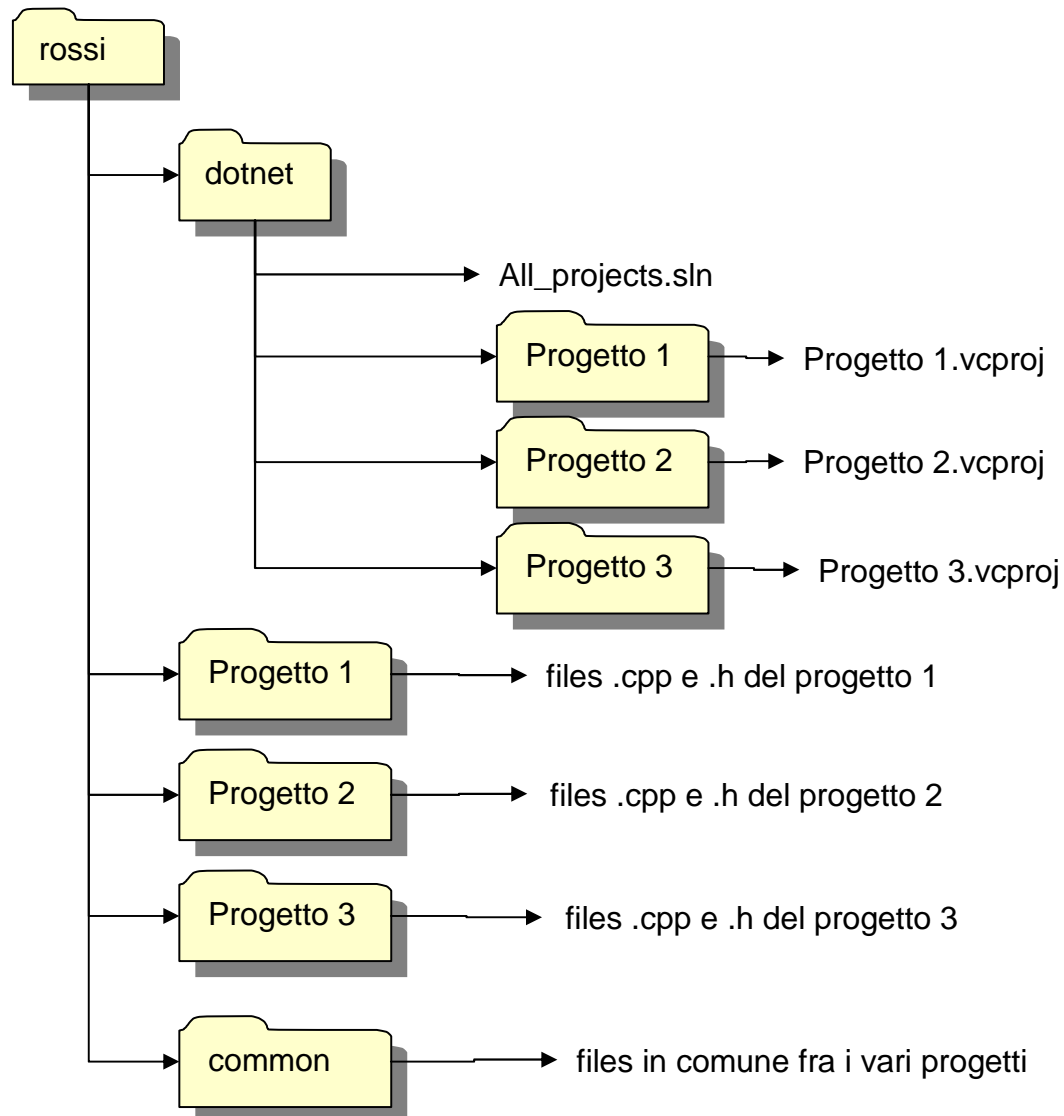


CVS

- Concurrent Versioning System
- Client per windows www.tortoisecvs.org.



Organizzazione del vostro folder





- Simple DirectMedia Layer

- cross-platform multimedia library
- GNU LGPL
- fornisce accesso (a livello abb basso) a

- audio,
- keyboard, mouse, joystick,
- windowing
- 3D hardware via OpenGL
- 2D video framebuffer.

- gira su: Linux, Windows, BeOS, MacOS X ...
- completato di librerie "figlie" per vari scopo (e.g. SDL_png per file png)
- C++

- <http://www.libsdl.org>





Struttura programma

- Struttura classica dei programmi a linea di comando:

```
main()  
{  
    init();  
    do_my_beautiful_algorithm();  
    exit();  
}
```

non va bene per
applicazioni
interattive !





Struttura programma

- Sistema a eventi
 - "a callback" (o "a message handlers" ecc)

```
main()  
{  
    init();  
    while (true) {  
        get_event();  
        process_event();  
    }  
}
```

eventi tipo:

- mouse, tastiera...
- sistema di finestre
 - reshape, minimizzazione...
- generati dall'applicazione stessa
- o da thread differenti





La minima applicazione SDL: headers

```
#ifdef WIN32
#define WIN32_LEAN_AND_MEAN
#include <windows.h>
#endif

#include <GL/gl.h>
#include <GL/glu.h>
#include <stdlib.h>
#include <SDL.h>
```





La minima applicazione SDL: main

```
int main(int argc, char **argv)
{
    SDL_Init(SDL_INIT_VIDEO);
    SDL_SetVideoMode(640, 480, 0, SDL_OPENGL);

    int done = 0;
    while ( ! done ) /* Loop, drawing and checking events */
    {
        myDrawGLScene(); /* Questa poi la vediamo */
        SDL_Event event;
        SDL_WaitEvent(&event);
        switch(event.type)
        {
            case SDL_QUIT      : done = 1;    break ;
            case SDL_KEYDOWN  :
                if ( event.key.keysym.sym == SDLK_ESCAPE )
                    done = 1;
                break;
        }
    }
    SDL_Quit();
    return 1;
}
```





Meglio:

- Ridisegna la scena troppe volte
- Gestiamo anche l'evento
“necessita’ di ridisegnare”
- Nel ciclo degli eventi:

```
case SDL_VIDEOEXPOSE      :  
    myDrawGLScene ( ) ;  
    break;
```

- E togliamo il `myDrawGLScene` dal ciclo degli eventi





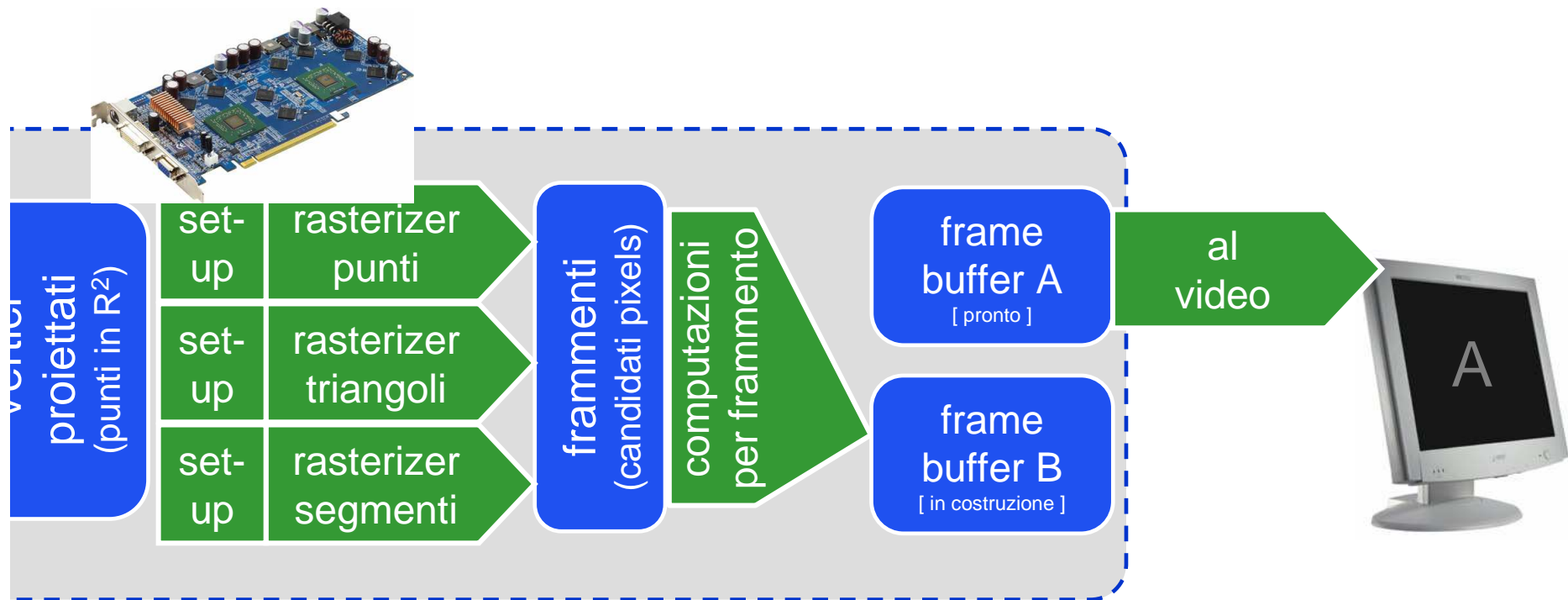
La minima applicazione SDL: la parte che disegna

```
void DrawGLScene()  
{  
    glClear(GL_COLOR_BUFFER_BIT);  
  
    /* disegna tutto */  
  
    glFinish(); /* aspetta che sia tutto finito */  
    SDL_GL_SwapBuffers(); /* questa fra un sec */  
}
```



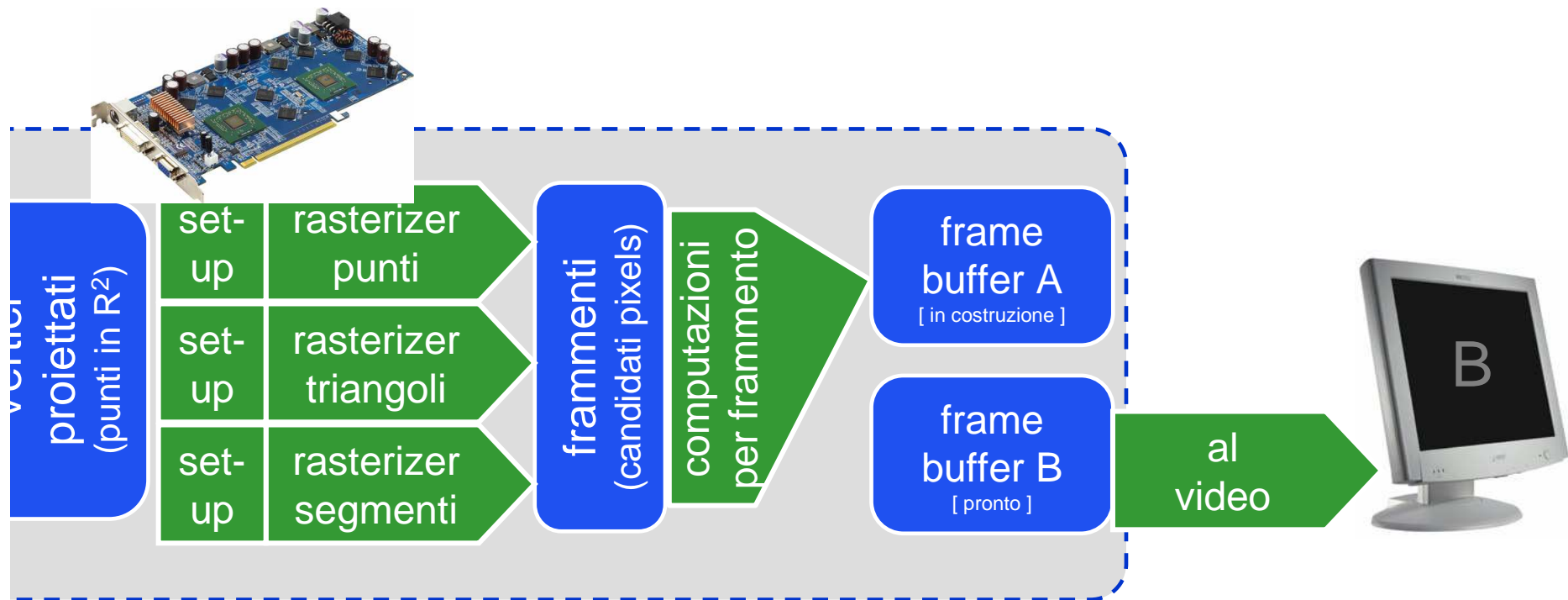
Double buffering

- Piccolo trucco utile alle applicazioni interattive
 - nascondere il frame buffer mentre viene riempito

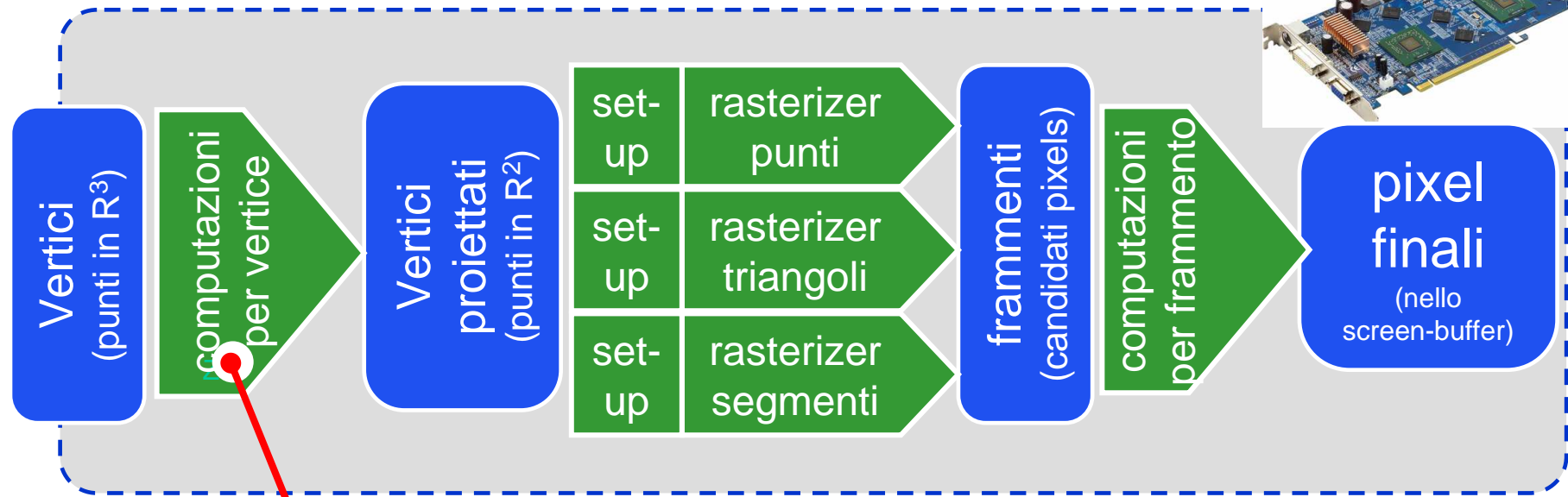


Double buffering

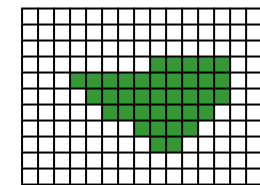
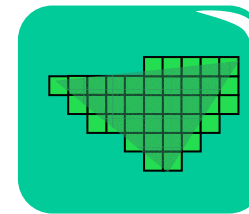
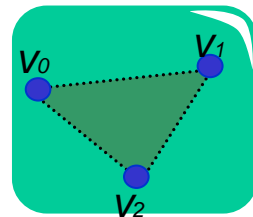
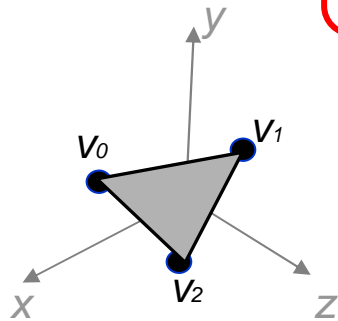
- Piccolo trucco utile alle applicazioni interattive
 - nascondere il frame buffer mentre viene riempito



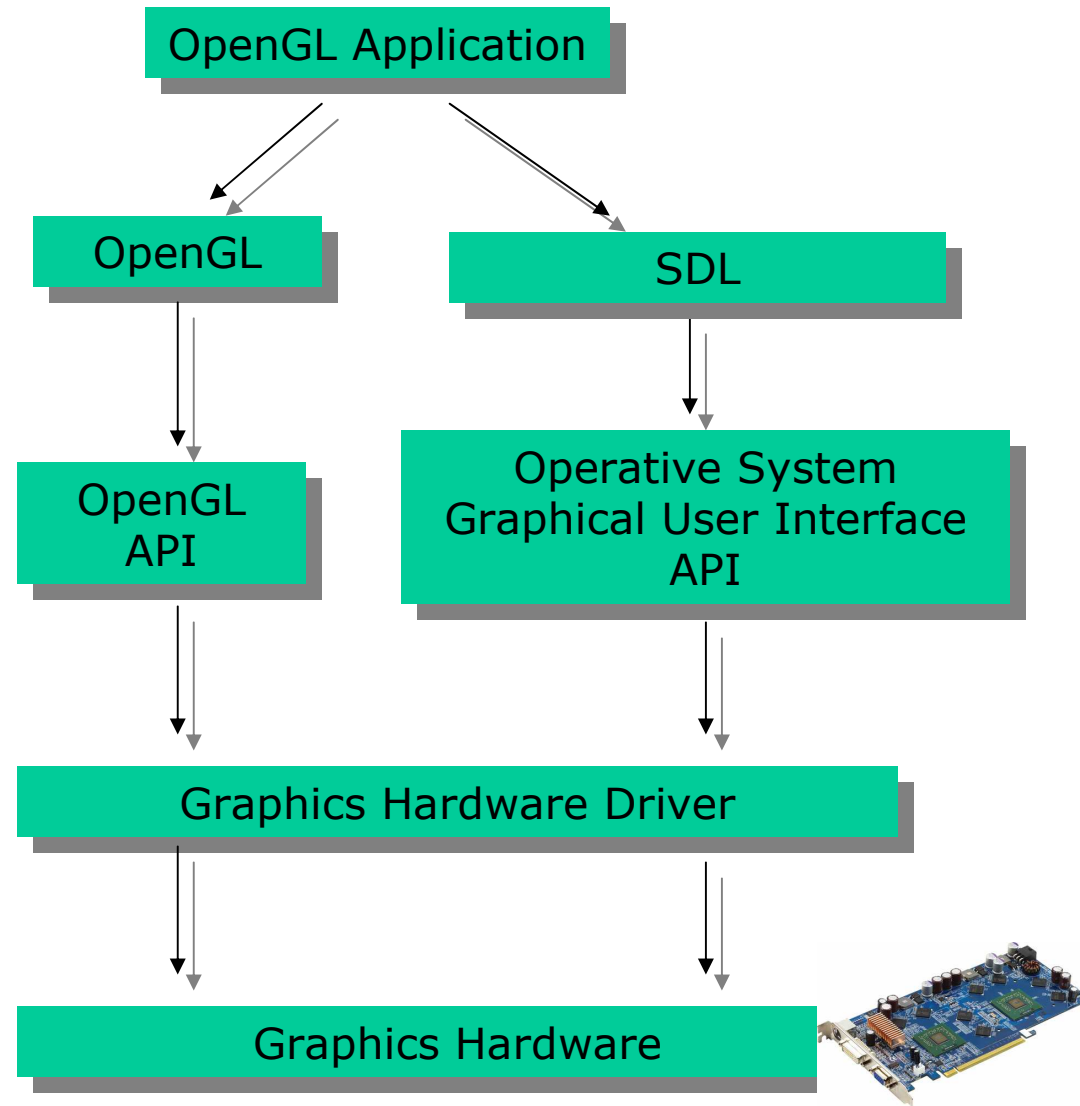
Riassunto puntate precedenti 1/3



ora di qui sappiamo tutto



Struttura Applicazione OpenGL+SDL



Non lo studiamo qui, ma esiste

- Direct3D
 - Parte di DirectX
 - Microsoft
 - Stessi scopi di OpenGL
 - un API per usare lo stesso hardware
 - struttura non dissimile
 - L'alternativa piu' comune a OpenGL





- Open Graphic Language
- Libreria C
 - Cross platform
 - Qualche centinaio di routines
- www.opengl.org
 - specifiche

Specifiche 

ver 2.0



OpenGL Storia

- inizialmente sviluppato da Silicon Graphics



- ora:

OpenGL **A**rchitecture **R**evision **B**oard

- mantiene e aggiorna le *specifiche*
- versione attuale: **2.0**
- una compagnia, un voto

ARB



- ci sono anche le *estensioni* private

- Soprattutto  e  **NVIDIA**.



Vicini di casa

- OpenGL e' il layer di base
- GLU (GL utilites)
 - insieme di funzioni di utility costruite sopra OpenGL, piu'comode da usare
 - esempio

```
void gluLookAt(eyex, eyey, eyez,  
              cx, cy, cz,  
              upx, upy, upz);
```
- GLUT e' il Toolkit di interfaccia con il SO
- Wgl e GLx sono i sottoinsiemi di OpenGL che dipendono dal SO



- Tutte le funzioni di OpenGL si chiamano:

glSomethingXXX

- Dove XXX specifica (numero) il tipo dei parametri:

- esempio:

```
glColor3f(float, float, float);
```

```
glColor3fv( float* );
```

f: float

d: double ...

v: vettore

- Non e' C++...



- Ma anche:

`glColor3b, glColor3d, glColor3f,
glColor3i, glColor3s, glColor3ub,
glColor3ui, glColor3us, glColor4b,
glColor4d, glColor4f, glColor4i,
glColor4s, glColor4ub, glColor4ui,
glColor4us, glColor3bv, glColor3dv,
glColor3fv, glColor3iv, glColor3sv,
glColor3ubv, glColor3uiv, glColor3usv,
glColor4bv, glColor4dv, glColor4fv,
glColor4iv, glColor4sv, glColor4ubv,
glColor4uiv, glColor4usv`





Basato sullo stato

- Una state machine
 - ad esempio
 - current color
 - posizione luci
 - matrici
 - fanno parte dello stato corrente
- Molti comandi OpenGL non fanno nulla
 - se non cambiare lo stato





Basato sullo stato

- Stato comprende 2 matrici (e due stacks)
 - **Model-View**
 - **Projection**
- Una di queste e' sempre la matrice di lavoro
 - la matrice corrente
- I comandi che modificano matrici lavorano su questa matrice





Matrici

- Nota: assume che siano memorizzate per colonne
 - detto anche in column major order

$$\begin{bmatrix} a_0 & a_4 & a_8 & a_{12} \\ a_1 & a_5 & a_9 & a_{13} \\ a_2 & a_6 & a_{10} & a_{14} \\ a_3 & a_7 & a_{11} & a_{15} \end{bmatrix}$$



- Per cambiare quale é la matrice di lavoro:

```
glMatrixMode(***);
```

```
GL_MODELVIEW
```

```
GL_PROJECTION
```

- Per rimpiazzare la matrice di lavoro

```
- glLoadIdentity();
```

```
- glLoadMatrixf( float* m );
```

- Tutti gli altri comandi modificano (moltiplicano per un'altra matrice) la matrice corrente.





in gradi

- Rotazioni

 - `glRotatef(angle, ax, ay, az);`

- Traslazioni

 - `glTranslatef(dx, dy, dz);`

- Scalature (non uniformi)

 - `glScalef(ax, ay, az);`

- Generica

 - `glMultMatrixf(float f*);`

asse di rotazione
passante per l'origine



- Vista:

```
void gluLookAt(eyex, eyey, eyez,  
              cx, cy, cz,  
              upx, upy, upz);
```

centro. La direzione
 e' ($c - eye$)



- Operazione sullo Stack:

glPushMatrix()

glPopMatrix()





Matrici di proiezione

- Matrici di proiezione:

```
glOrtho2D(left, right,  
          bottom top);
```

```
void gluPerspective(  
  fovy,
```

```
  aspect,
```

```
  zNear,
```

```
  zFar);
```

In
gradi



- Per settare il viewport:

```
glViewport(int x, int y,  
           int w, int h);
```

reminder: il rapporto fra w e h deve essere lo stesso specificato nella matrice di proiezione!



Evento Window Reshape

- Succede all'inizio
 - e ogni volta che l'utente cambia dimensioni alla finestra
 - devo permettere all'utente di farlo, durante l'inizializzazione:

```
SDL_SetVideoMode(640,480,0, SDL_OPENGL | SDL_RESIZABLE)
```

- gestione dell'evento: (devo fare di nuovo il set up del video)

```
...
```

```
case SDL_VIDEORESIZE :  
    SDL_SetVideoMode(event.resize.w,event.resize.h,  
        0, SDL_OPENGL |SDL_RESIZABLE);  
    myReshapeFunc(event.resize.w,event.resize.h);
```



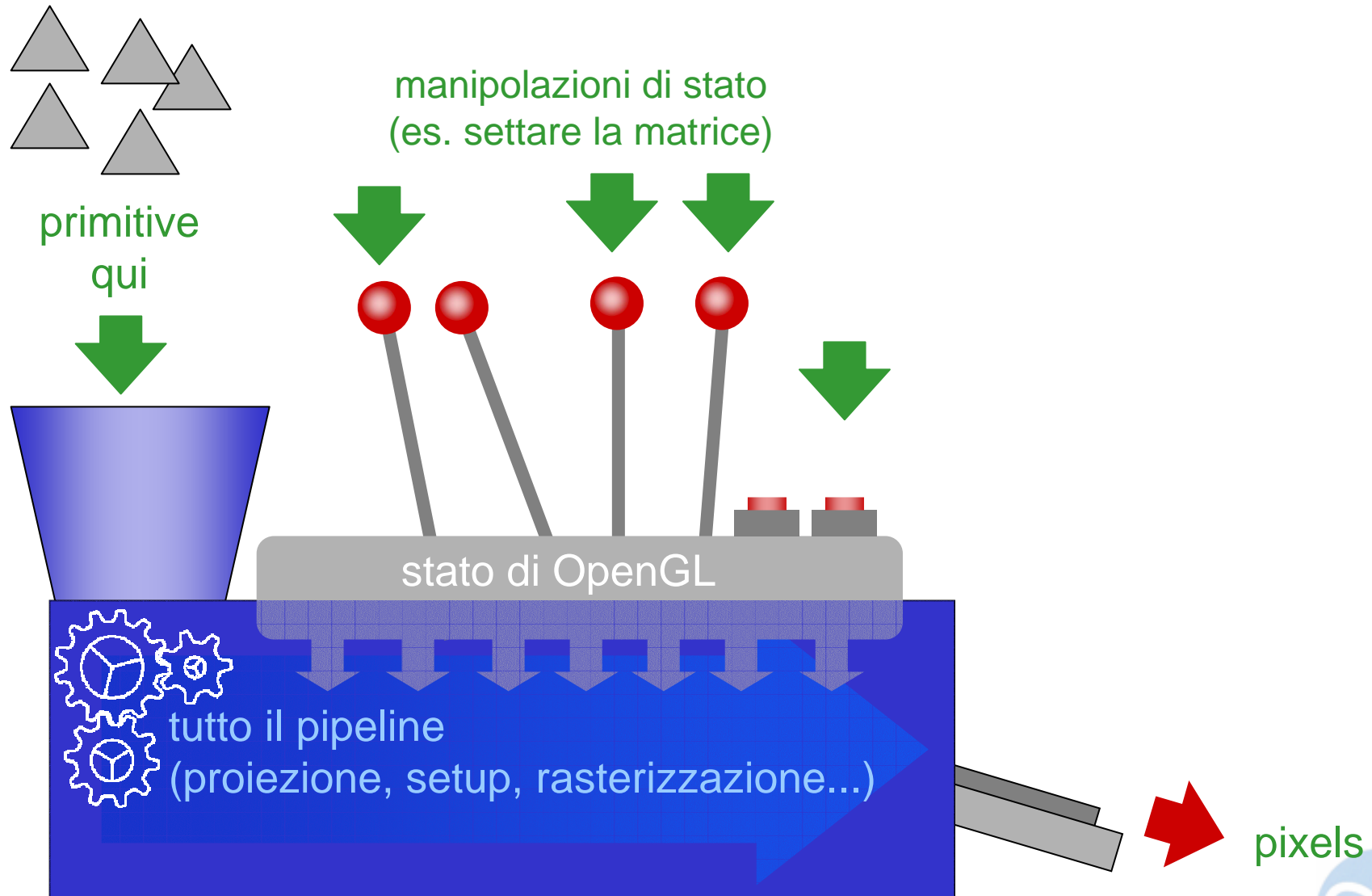
Adattare la camera alla finestra: proiezione ortografica

```
void myReshapeFunc(GLsizei w, GLsizei h)
{
    glMatrixMode (GL_PROJECTION);
    glLoadIdentity ();
    float ratio=(float)h/(float)w;
    glOrtho2D(-1,1,-ratio,ratio);

    glViewport (0, 0, (GLsizei) w, (GLsizei) h);
    glMatrixMode (GL_MODELVIEW);
}
```



Ricordare sempre: OpenGL = state machine



Come si "sparano" i triangoli nel pipeline

```
glBegin (GL_TRIANGLES);
```

```
glVertex3d(x1,y1,z1);
```

```
glVertex3d(x2,y2,z2);
```

```
glVertex3d(x3,y3,z3);
```

} primo triangolo

```
glVertex3d(x4,y4,z4);
```

```
glVertex3d(x5,y5,z5);
```

```
glVertex3d(x6,y6,z6);
```

} secondo triangolo

```
glVertex3d(x7,y7,z7);
```

```
glVertex3d(x8,y8,z8);
```

```
glVertex3d(x9,y9,z9);
```

} terzo triangolo

```
...
```

```
glEnd();
```



Come si "sparano" i triangoli nel pipeline

```
glVertex3d(x,y,z);
```

oppure `glVertex3f(x,y,z);`

oppure `glVertex3i(x,y,z);`

oppure `glVertex2d(x,y);`

oppure `glVertex4d(x,y,z,w);`

oppure `glVertex4dv(vett);`

oppure...

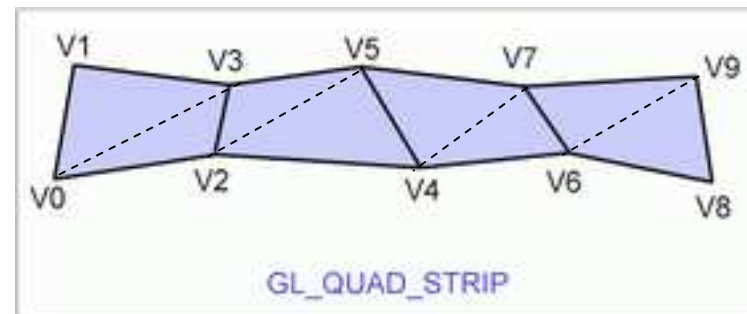
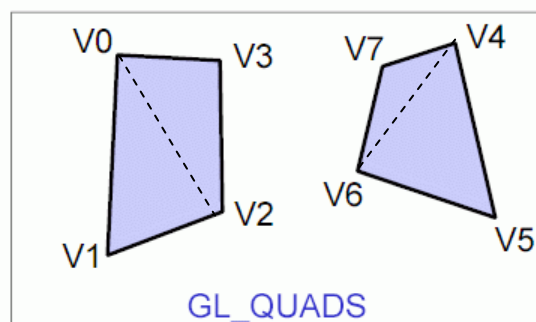
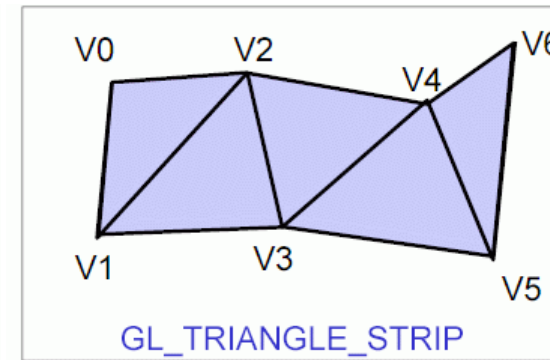
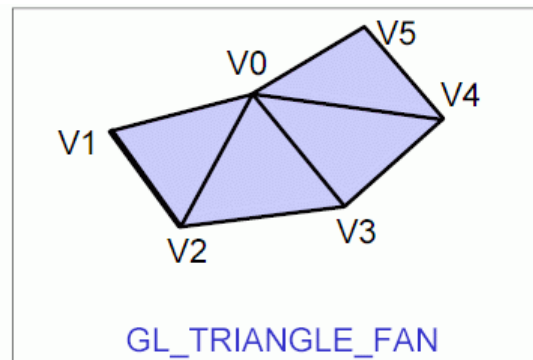
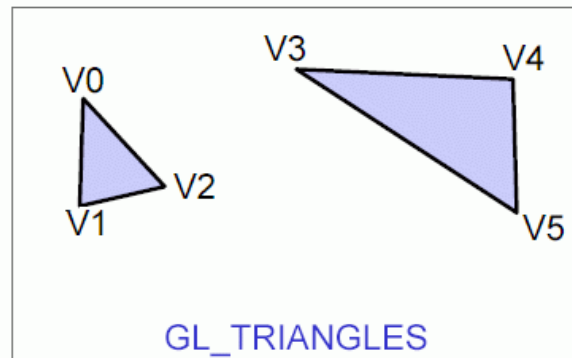
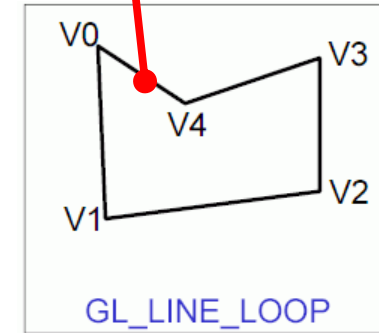
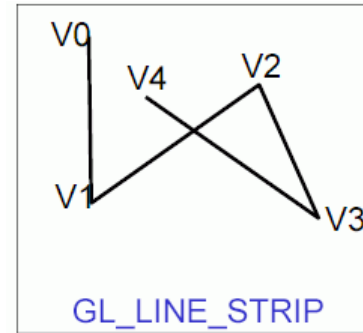
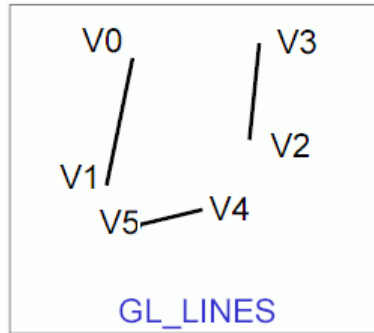
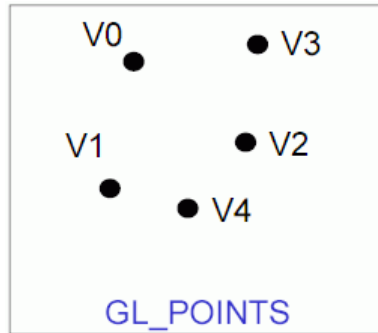
coordinata $w=1$
sottointesa!

coordinata $z=0$
sottointesa!



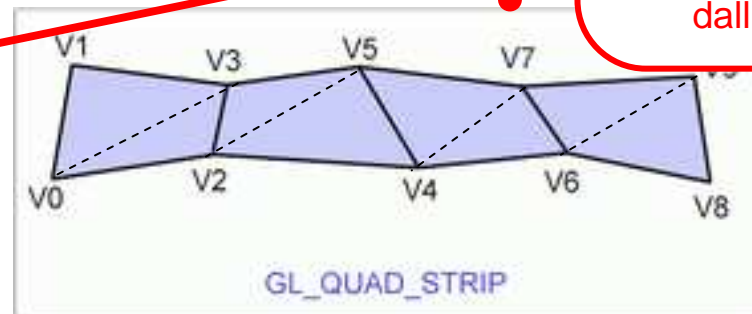
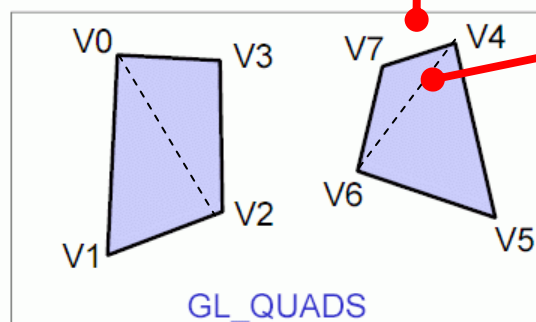
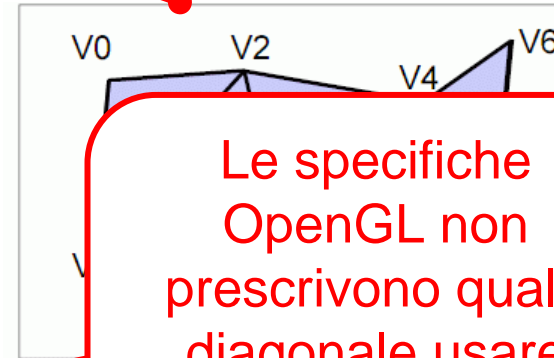
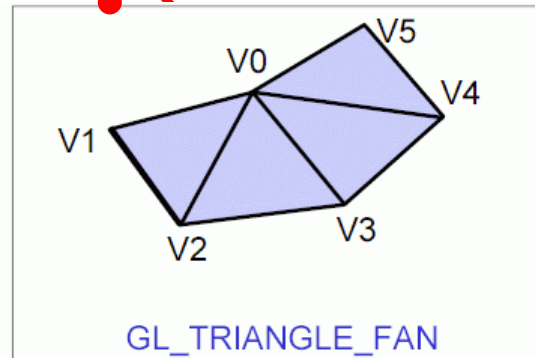
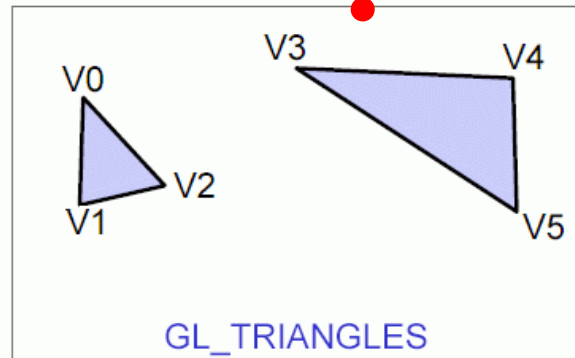
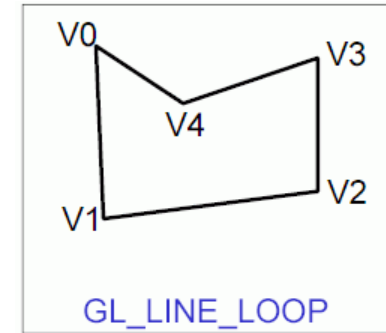
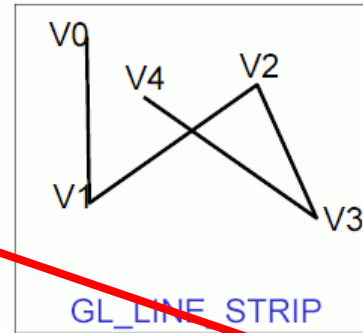
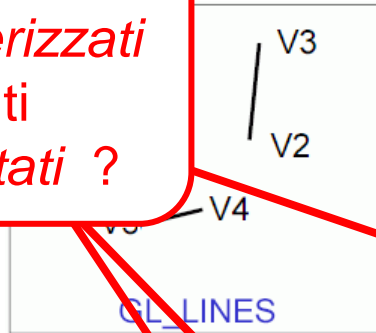
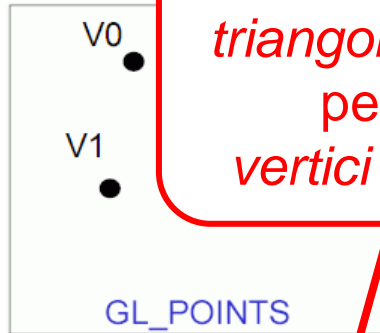
Non solo glBegin (GL_TRIANGLES);

linea finale quando si fa la glEnd()



Non solo glBegin (GL_TRIANGLES);

quanti
triangoli rasterizzati
per quanti
vertici proiettati ?



Le specifiche
OpenGL non
prescrivono quale
diagonale usare
(quindi dipende
dall'implementazione)

