Tutorial SDL-OpenGL (Parte I)

di Alexander "nanaki" Traverso | *21-04-2002 17:51:13*

Tutorial introduttivo sulle librerie SDL: come creare una finestra adatta ad usare l'OpenGL

Download | Commenta/Vota | Segnala | Stampa | Preferiti | Cerca simili

Tutorial SDL-OpenGL (Parte I)

INTRODUZIONE SDL

Iniziamo con una piccola descrizione delle SDL: quest'ultime sono librerie grafiche, simili alle DirectX, difatti, attraverso esse si può cambiare la risoluzione dello schermo, si possono caricare immagini e si possono anche usare file audio per aggiungere un pò di atmosfera nel proprio demo/videogioco; ma soprattutto le SDL possono creare per voi un buffer OpenGL pronto all'uso, su cui disegnare i vari oggetti 3D, e tutto questo in poche istruzioni, come vedrete.

Per compilare con le SDL, ovviamente le dovrete installare (basta che cercate su internet i pacchetti rpm delle SDL: libSDL.rpm, libSDL-devel.rpm), e poi passare il parametro *sdl-config --cflags --libs* a gcc in questo modo:

```
gcc -Wall -o foo foo.c 'sdl-config --cflags --libs' -lGL -lGLU
```

Le librerie da aggoiungere sono poi GL (OpenGL) e GLU (OpenGL utils); facile no?

NOTE SUL CODICE

Il codice sorgente di esempio nel tutorial è in linguaggio C, al contrario dei sorgenti accompagnati con il tutorial, che sono in C++, per il semplice fatto che il C è migliore per spiegare le cose, e non è detto che chiunque legga questo tutorial sappia il C++.

In questo tutorial il codice di esempio è di questo colore, mentre il codice delle definizioni delle funzioni o delle strutture è di questo colore.

INIZIALIZZAZIONE SDL

Anche le SDL devono essere inizializzate, ma non preoccupatevi, è tutto molto semplice, difatti basta chiamare questa funzione, che carica le librerie dinamiche che fanno parte delle SDL, e eseguono le prime inizializzazioni:

```
int SDL_Init ( Uint32 flags );
```

Le possibili flags sono:

SDL_INIT_TIMER
inizializza i timer (x calcolare fps, e altre operazioni da eseguire con i secondi)
SDL_INIT_AUDIO
inizializza la scheda audio...
SDL_INIT_VIDEO
inizializza la scheda video (attraverso X) per cambiare risoluzione, disegnare sulle superfici, ecc...
SDL_INIT_CDROM
inizializza il CDROM...
SDL_INIT_JOYSTICK
stranamente questa flag inizializza il joystick...
SDL_INIT_NOPARACHUTE
questa flag fa si che le SDL ignorino certi errori che farebbero chiudere il programma: i "fatal signals", questi errori si verificano quando si va ad utilizzare una parte di memoria non allocata x il nostro programma.
SDL_INIT_EVENTTHREAD
ad essere sincero non lo so...:(
SDL_INIT_EVERYTHING
è abbastanza esplicativo... (everything = tutto)

Per ora la flag che useremo è SDL_INIT_VIDEO, quindi:

```
#include "SDL.h"    // includo il file header principale

// InitSDL - inizializzazione SDL
// screen = la tua superficie
// title = titolo della finestra creata dalle SDL
// w = larghezza schermo in pixel
// h = altezza schermo in pixel
// bpp = bits per pixel (16 o 32), ovvero il numero di colori
// 16 bit = 2^16 = 65536 colori
// 32 bit = 2^32 = troppi colori... (più di 4 miliardi)
// fullscreen = schermo intero o in finestra?
int InitSDL ( SDL_Surface* screen, const char* title, int w, int h, int bpp, int
fullscreen )
{
   SDL_Init ( SDL_INIT_VIDEO );
```

NOTA1: Se le flags fossero più di una, non preoccupatevi, basta unirle con un OR logico:

```
   SDL_Init ( SDL_INIT_VIDEO | SDL_INIT_AUDIO |
SDL_INIT_NOPARACHUTE );
```

NOTA2: Dopo aver chiamato la funzione SDL_Init, si possono inizializzare altri sistemi, attraverso questa funzione:

```
   SDL_InitSubSystem ( Uint32 flags );
```

per poi disattivarli basta usare questa funzione:

```
   SDL_QuitSubSystem ( Uint32 flags );
```

Ora, per avere più informazioni sulla scheda grafica in uso sul vostro computer, useremo questa funzione:

```
   SDL_VideoInfo* SDL_GetVideoInfo ( void );
```

che ritorna un puntatore a una struttura che contiene informazioni sull'hardware video, come la memoria video, la possibilità di eseguire blit hardware e la possibilità di contenere le superfici (aree di memoria dove è possibile disegnare) nella memoria video invece che nella memoria RAM; la struttura è questa, copiata pari-pari dal file SDL.h...

```
typedef struct {
    Uint32 hw_available :1; /* Flag: Can you create hardware surfaces? */
    Uint32 wm_available :1; /* Flag: Can you talk to a window manager? */
    Uint32 UnusedBits1 :6;
    Uint32 UnusedBits2 :1;
    Uint32 blit_hw :1; /* Flag: Accelerated blits HW --> HW */
    Uint32 blit_hw_CC :1; /* Flag: Accelerated blits with Colorkey */
    Uint32 blit_hw_A :1; /* Flag: Accelerated blits with Alpha */
    Uint32 blit_sw :1; /* Flag: Accelerated blits SW --> HW */
    Uint32 blit_sw_CC :1; /* Flag: Accelerated blits with Colorkey */
    Uint32 blit_sw_A :1; /* Flag: Accelerated blits with Alpha */
    Uint32 blit_fill :1; /* Flag: Accelerated color fill */
    Uint32 UnusedBits3 :16;
    Uint32 video_mem; /* The total amount of video memory (in K) */
    SDL_PixelFormat *vfmt; /* Value: The format of the video surface */
} SDL_VideoInfo;
```

A questo punto bisogna controllare "hw_available" e "blit_hw", per poi agire di conseguenza sulle flag che poi useremo per creare la superficie principale:

```
    Uint32 flags;
    SDL_VideoInfo* vInfo;

    vInfo = SDL_GetVideoInfo ( );
    if ( vInfo == NULL ) {
        // errore
        // codice per visualizzare errore
        return -1;    // init non riuscito
    }

    if ( vInfo->hw_available )
        flags = SDL_HWSURFACE;
    else
        flags = SDL_SWSURFACE;

    if ( vInfo->blit_hw )
        flags |= SDL_HWACCELL;
```

A questo punto bisogna controllare "fullscreen", se questa è vera aggiungiamo la flag SDL_FULLSCREEN altrimenti non facciamo niente.

```
    if ( fullscreen )
        flags |= SDL_FULLSCREEN;
```

Ora aggiungiamo le flag di default per far funzionare OpenGL:

```
    flags |= SDL_OPENGL |            // questa superficie è adatta all?OpenGL
            SDL_GL_DOUBLEBUFFER |    // questa superficie usa un buffer
con cui
                                     // fare il flip (tipo il backbuffer DDraw)
            SDL_HWPALETTE;           // la palette dev'essere contenuta nella
                                     // memoria video

    SDL_GL_SetAttribute ( SDL_GL_BACKBUFFER, 1 );    // attiva il double-
buffer
```

Finalmente possiamo creare la nostra superficie, utilizzando la funzione

```
    SDL_Surface* SDL_SetVideoMode ( int width, int height, int bpp, Uint32
flags );
```

Credo che sia abbastanza esplicativa...
Come variabile di ritorno ha un puntatore a una struttura SDL_Surface, contenente tutte le sue informazioni (larghezza, altezza, il pitch: lunghezza in byte di una riga della superficie), e un puntatore alla sua memoria (void* pixels).
Se volete saperne di più date un'occhiata al file SDL.h in */usr/include/SDL*.

```
    screen = SDL_SetVideoMode ( w, h, bpp, flags );
    if ( screen == NULL ) {
        // errore
        // codice per visualizzare errore
        return -1; // init non riuscito
    }

    SDL_WM_SetCaption ( title, title );

    return 0; // init riuscito
}
// fine InitSDL
```

CONTROLLARE L'INPUT

Ora che abbiamo la nostra superficie dobbiamo anche sapere come gestire tutti gli eventi di quest'ultima: pressione pulsanti tastiera, pressione polsanti mouse, movimento mouse, ecc...
Per tutte queste operazioni useremo la struttura SDL_Event:

```
typedef union {
    Uint8 type;
    SDL_ActiveEvent active;
    SDL_KeyboardEvent key;
    SDL_MouseMotionEvent motion;
    SDL_MouseButtonEvent button;
    SDL_JoyAxisEvent jaxis;
    SDL_JoyBallEvent jball;
    SDL_JoyHatEvent jhat;
    SDL_JoyButtonEvent jbutton;
    SDL_ResizeEvent resize;
    SDL_QuitEvent quit;
    SDL_UserEvent user;
    SDL_SysWMEvent syswm;
} SDL_Event;
```

Questa struttura contiene tutti i possibili eventi che la nostra finestra può gestire, in forma di altre strutture che potete trovare nel file */usr/include/SDL/SDL_events.h.*

Per ora useremo gli eventi inerenti alla tastiera e al mouse, attraverso delle funzioni “callback”, ovvero funzioni che verrano richiamate al momento opportuno; si definiscono come funzioni normali, e dovrete cambiarle ogni qual volta i controlli cambino, per fare un esempio, all’inizio di un gioco, le frecce della tastiera serviranno a spostarsi sulle varie opzioni, ma quando inizierà il gioco le frecce cambieranno funzione...

Passando oltre ecco un esempio di funzione x la gestione della tastiera:

```
// k = numero del tasto della tastiera
// p = TRUE:    il tasto è stato schiacciato
//     FALSE:   il tasto è stato rilasciato
void keyb_func ( SDLKey k, int p )
{
```

```
   if ( k == SDL_ESCAPE && p == TRUE )    // se viene schiacciato il tasto
ESC
      SDL_Quit ( );                    // rilascia le SDL
}
```

Per avere la lista dei tasti basta dare uno sguardo al file */usr/include/SDL/SDL_keysym.h*, e cercare l'enum SDLKey.

Ora passiamo alla funzione x controllare il mouse, ecco un altro esempio:

```
// b = SDL_BUTTON_LEFT: tasto sinistro del mouse
//     SDL_BUTTON_MIDDLE: tasto centrale del mouse
//     SDL_BUTTON_RIGHT: tasto destro del mouse
// s = stato dei tasti del mouse:
//     SDL_PRESSED: tasto premuto
//     SDL_RELEASED: tasto non premuto
// x, y: posizione del mouse nello schermo
void mouse_func ( int b, int s, int x, int y )
{
   if ( b == SDL_BUTTON_LEFT && s == SDL_PRESSED ) // se viene
premuto il tasto
      SDL_Quit ( );                // sx in qualunque parte
                  // dello schermo, esci
}
```

Se volete sapere di più sul controllo del mouse da parte delle SDL vai a dare uno sguardo al file */usr/include/SDL/SDL_mouse.h*.

Questi erano degli esempi che possono essere cambiati in base alle vostre esigenze, ora bisogna scrivere una funzione che chiami keyb_func e mouse_func, in base agli eventi, ed eccola qua:

```
// ProcessEvents - controllo degli eventi SDL
// Keyboard = funzione di controllo della tastiera
// Mouse    = funzione di controllo del mouse
void ProcessEvents ( void (*Keyboard) ( SDLKey, int ),
                  void (*Mouse) ( int, int, int, int ) )
{
   SDLKey    key;
   SDL_Event event;

   if ( SDL_PollEvent ( &event ) ) {
      switch ( event.type ) {
         case SDL_KEYDOWN:
         if ( Keyboard ) {
            key = event.key.keysym.sym;
            (*Keyboard) ( key, TRUE );
         }
         break;

         case SDL_KEYUP:
         if ( Keyboard ) {
            key = event.key.keysym.sym;
            (*Keyboard) ( key, FALSE );
         }
         break;

         case SDL_MOUSEBUTTONDOWN:
         case SDL_MOUSEBUTTONUP:
         if ( Mouse ) {
            (*Mouse) ( event.button.button,
                  event.button.state,
```

```
                    event.button.x,
                    event.button.y );
            }
            break;
        }
    }
}
```

NOTA: Questa funzione dovrà essere chiamata a ogni frame.

FINALMENTE OPENGL

Finalmente possiamo passare alla programmazione OpenGL, partendo con una funzione che inizializzi propriamente le librerie OpenGL; le operazioni da fare non sono molte, ma essenziali per disegnare anche solo un triangolo sullo schermo.

```
// InitGL - inizializzazione OpenGL
// w = larghezza dello schermo
// h = altezza dello schermo
// minz, maxz = z maggiore e minore per i piani di clipping
void InitGL ( int w, int h, float minz, float maxz )
{
```

Per prima cosa bisogna attivare la possibilità di usare le texture e lo zbuffer:

```
    glEnable ( GL_TEXTURE );
    glEnable ( GL_DEPTHTEST );
```

per poi fare i settaggi iniziali per zbuffer, tipo di shading e colore di sfondo:

```
    glDepthFunc ( GL_LEQUAL );    // disegna solo se z è minore o uguale
    glClearDepth ( 1.0f );        // esegui il clear dello zbuffer con la profondità
                                  // maggiore ( 0.0f minimo - 1.0f massimo )
    glShadeModel ( GL_SMOOTH );   // gouraud shading
    glClearColor ( 0.0f, 0.0f, 0.0f, 0.0f );  // colore di sfondo nero
```

ora viene la parte più importante: settare il viewport e la matrice di proiezione, che spiegano all'OpenGL come comportarsi al momento della proiezione dei poligoni, ovvero alla trasformazione da 3d a 2d; il viewport non è niente di speciale: sono quattro interi che indicano la parte di superficie da usare, ovvero i due vertici estremi del rettangolo su cui disegnare, invece la matrice è qualcosa di molto più complesso, ma per fortuna ci viene in aiuto questa funzione:

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

fovy = angolo di visuale sull'asse y in gradi, normalmente è 45
aspect = la divisione tra la larghezza e l'altezza dello schermo
zNear = il piano più vicino di clipping
zFar = il piano più lontano di clipping

Se sapete poco di matrici, di piani e di vettori, vi consiglio di scaricare qualche tutorial che ve li spieghi (su internet è pieno), perchè io non ho voglia di farlo :P
Ora andiamo avanti con la funzione:

```
    glViewPort ( 0, 0, w, h );
    glMatrixMode ( GL_PROJECTION ); // attiva la matrice di proiezione
    glLoadIdentity ( );
```

```
    // qua sarà meglio controllare che h non sia zero, per non avere
    // un errore a runtime (Division by zero)
    gluPerspective ( 45.0f, (float)w / (float)h, minz, maxz );
    glMatrixMode ( GL_MODELVIEW );  // attiva la matrice oggetto
    glLoadIdentity ( );
}
```

E con questo abbiamo finito l'inizializzazione dell'OpenGL, ora vi chiederete come si fa a disegnare qualcosa, vero? È molto semplice, ecco un esempio:

```
    glBegin ( GL_QUADS ); // inizio il disegno di 1 o più rettangoli
        // 1° rettangolo
        glVertex3f ( -0.1f, -0.1f, 0.0f ); // 1° vertice
        glVertex3f ( 0.1f, -0.1f, 0.0f ); // 2° vertice
        glVertex3f ( 0.1f, 0.1f, 0.0f ); // 3° vertice
        glVertex3f ( -0.1f, 0.1f, 0.0f ); // 4° vertice
        // 2° rettangolo
        glVertex3f ( -0.1f, -0.1f, 1.0f ); // 1° vertice
        glVertex3f ( 0.1f, -0.1f, 1.0f ); // 2° vertice
        glVertex3f ( 0.1f, 0.1f, 1.0f ); // 3° vertice
        glVertex3f ( -0.1f, 0.1f, 1.0f ); // 4° vertice
        // e così via...
    glEnd ( ); // fine disegno di rettangoli
```

Ogni vertice deve essere formato da un vettore: glVertex** (x, y, z);
successivamente può essere formato da una normale: glNormal** (nx, ny, nz);
da un colore: glColor** (r, g, b, a);
da coordinate u e v per il texture mapping: glTexCoord** (u, v);

I due asterischi possono essere:
1d, 1f, 1i, 1s = un double, un float, un integer, un short
2d, 2f, 2i, 2s = due double, due float, due integer, due short
fino a 4.
Alla fine del nome della funzione se c'è anche una 'v', vuol dire che i dati sono in forma di array, ecco un esempio:

```
float x, y, z;
float v [3];

// ...
x = 0.0f; y = 0.0f; z = 1.0f;
// x
v[0] = 0.0f;
// y
v[1] = 0.0f;
// z
v[2] = 1.0f;
// ...
glBegin ( GL_TRIANGLES );
    // ...
    glVertex3f ( x, y, z );
    // oppure
    glVertex3fv ( v );
    // ...
glEnd ( );
```

Comunque non basta questo per disegnare, perche a ogni frame ci sono varie operazioni da fare oltre a quelle inerenti al disegno, due le più importanti:

```
    GLvoid glClear ( GLenum );          // pulisce lo schermo e lo zbuffer
    void SDL_GL_SwapBuffers ( void );   // esegue il flip tra il buffer e la
superficie primaria
```

Ecco un esempio di funzione che disegna un rettangolo:

```
// DrawFrame - disegna la scena (un rettangolo...)
// x1, y1, x2, y2 - coordinate rettangolo
void DrawFrame ( int x1, int y1, int x2, int y2 )
{
    glClear ( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );

    glBegin ( GL_QUADS );
        // non viene chiamato nessun glColor, quindi il rettangolo sarà bianco
        glVertex3i ( x1, y1, 1 );
        glVertex3i ( x2, y1, 1 );
        glVertex3i ( x2, y2, 1 );
        glVertex3i ( x1, y2, 1 );
    glEnd ( );

    SDL_GL_SwapBuffers ( );
}
```

Credo che con questo sia tutto, per ora...
Per ulteriori chiarimenti date uno sguardo al codice sorgente allegato o scrivetemi su nanaki_1984@hotmail.com

---

Visualizzo 0 commenti.

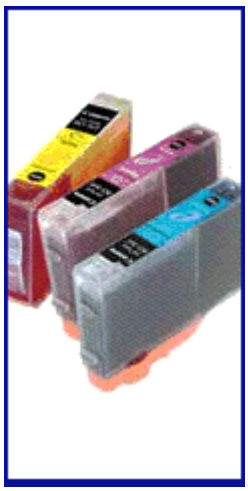
Devi effettuare il login o **registrarti** per poter commentare una risorsa.

DATA

Valutazione

VOTI
6

5 Vota !



TIPS: Monitorare le connessioni attive.