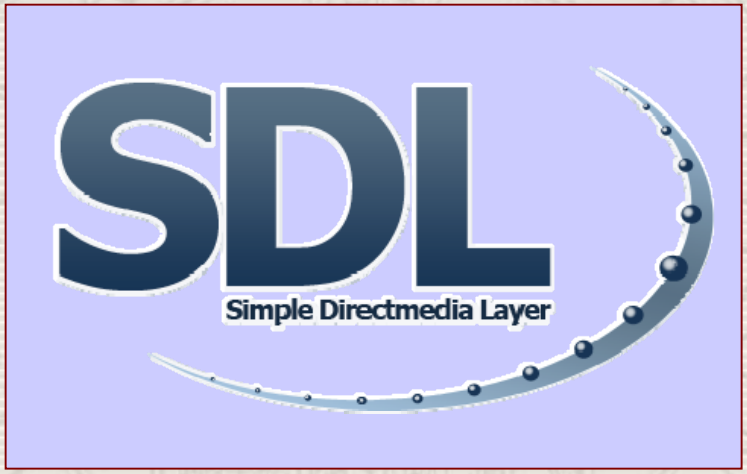
Libreria Grafica SDL 2.0



http://www.libsdl.org/

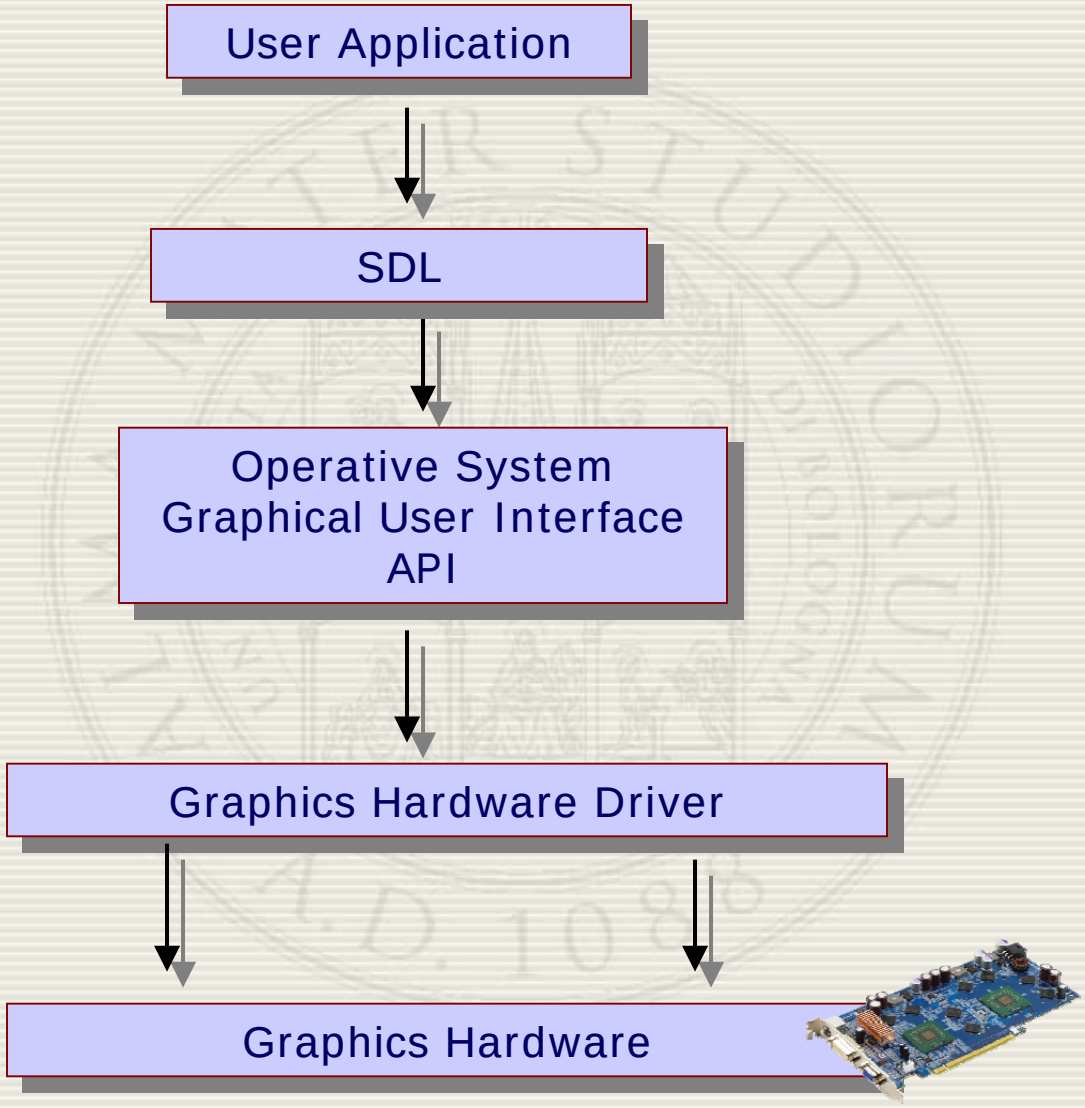
Libreria Grafica SDL 2.0

una libreria scritta in C “cross-platform multimedia” cioè che si presenta come una astrazione su varie piattaforme

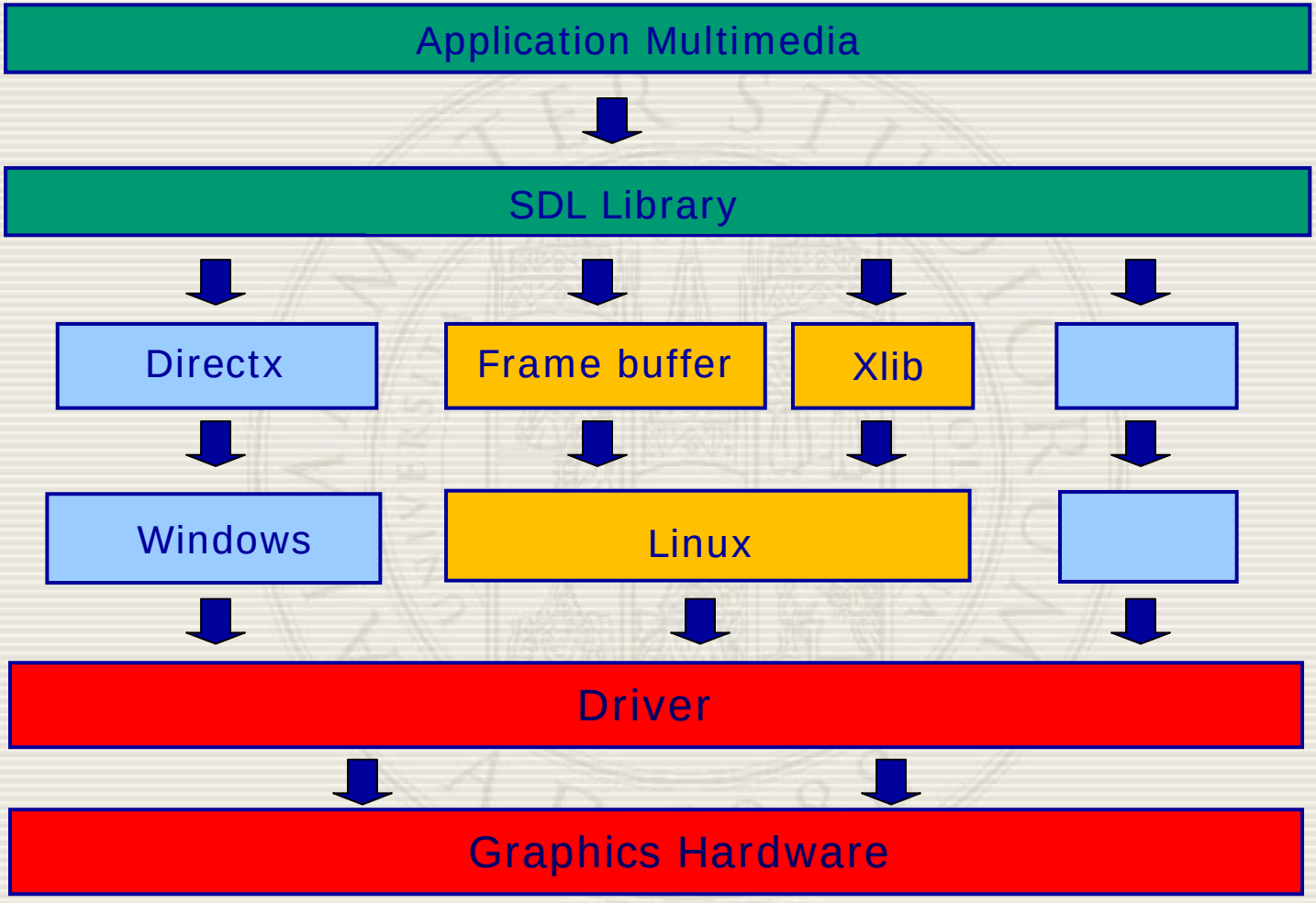
Simple DirectMedia Layer

- cross-platform multimedia library
- GNU LGPL
- fornisce accesso (a livello sufficientemente basso) a
  - audio,
  - keyboard, mouse, joystick,
  - 3D hardware via OpenGL
  - 2D video framebuffer.
- gira su: Linux, Windows, MacOS X, iOS, Android
- completo di librerie "figlie" per vari scopi (e.g. SDL_image, SDL_ttf, ecc.)
- C/C++

Struttura Applicazione SDL



Struttura Applicazione SDL



SDL: Inizio e Fine

SDL è composta da otto sottosistemi: Audio, CDROM, Gestione Eventi, I/O File, Gestione Joystick, Processi, Tempo e Video.
Prima di essere usati, questi sottosistemi devono essere inizializzati.

L'argomento, specifica che cosa si inizializza

SDL_Init(SDL_INIT_VIDEO)

Prima di terminare si devono anche chiudere chiamando alla fine

SDL_quit( )

Esempio:

```
/* Initialize SDL */
if (SDL_Init(SDL_INIT_EVERYTHING) < 0) {
    fprintf(stderr, "Error initializing SDL: %s\n", SDL GetError());
    /* Fail */
}
```

SDL_Window

Dopo aver inizializzato SDL, è necessario creare una window/viewport.
Questo viene fatto chiamando

SDL_CreateWindow( )

L'esempio seguente crea un'area di disegno 640x480 posizionata a partire da 0,0;
come ultimo argomento si usi uno dei seguenti flag:
SDL_WINDOW_SHOWN
SDL_WINDOW_FULLSCREEN: usa full screen mode
SDL_WINDOW_RESIZABLE
……
SDL_CreateWindow ritorna un puntatore a SDL_Window;

Esempio:

```
SDL_Window *p_win;
  ……
  p_win = SDL_CreateWindow("Title",0, 0, 640, 480, SDL_WINDOW_SHOWN);
  if ( p_win == NULL) {
    fprintf(stderr, "Create Window Error: %s\n", SDL GetError());
    /* Fail! */
  }
```

SDL_Renderer

Dopo aver creato una SDL_Window, è necessario creare un 2D Rendering Context per la window. Questo viene fatto chiamando

SDL_CreateRenderer( )

Esempio:

```
SDL_Renderer *p_ren;
  ……
  p_ren = SDL_CreateRenderer(p_win, -1,SDL_RENDERER_ACCELERATED);
  if ( p_ren == NULL) {
    fprintf(stderr, “Create Renderer Error: %s\n", SDL GetError());
    /* Fail! */
  }
```

Codice C di esempio

```
#include <SDL2/SDL.h> /*SDL defs */

int main(int argc, char *argv[]) {
SDL_Window *p_win;
SDL_Renderer *p_ren;

  /* Initialize SDL */
  if (SDL_Init(SDL_INIT_EVERYTHING) < 0) {
    fprintf(stderr, "Error initializing SDL: %s\n", SDL_GetError());
    return 1;
  }
  p_win = SDL_CreateWindow(“Hello World”,0,0,640, 480, SDL_WINDOW_SHOWN);
  if ( p_win == NULL) {
    fprintf(stderr, “Create Window Error: %s\n", SDL_GetError());
    return 1;
  }
  p_ren = SDL_Renderer(p_ win, -1, SDL_RENDERER_ACCELERATED);
  if ( p_ren == NULL) {
    fprintf(stderr, “Craete Renderer Error: %s\n", SDL_GetError());
    return 1;
  }
  ....(continua)
```

Codice C di esempio

```
....(continua)

  /* Fill with a color, update the window and wait 5 secs */
  /* Set the color for drawing operations */
  SDL_SetRendererDrawColor(p_ren,50,50,50,255);

  /* Clear the current rendering target */
  SDL_RenderClear(p_ren);

  /* Update the screen */
  SDL_RenderPresent(p_ren);

  /* Wait 5000 milliseconds before continuing */
  SDL_Delay(5*1000);

  /* Cleanup SDL */
  SDL_DestroyRenderer(p_ren);
  SDL_DestroyWindow(p_win);
  SDL_Quit();
  return 0;
}
```

Codice C di esempio (variante)

```
....(continua)

   /* Fill with a color, update the window and wait 5 secs */
  /* Set the color for drawing operations */
   SDL_SetRendererDrawColor(p_ren,50,50,50,255);

  /* Fill a rectangle on the current rendering target with the drawing color */
   SDL_RenderFillRect(p_ren, NULL);

   /* Update the screen */
   SDL_RenderPresent(p_ren);

/* Wait 5000 milliseconds before continuing */
   SDL_Delay(5*1000);

   /* Cleanup SDL */
   SDL_DestroyRenderer(p_ren);
   SDL_DestroyWindow(p_win);
   SDL_Quit();
   return 0;
}
```

Codice C di esempio (variante)

```
....(continua)

SDL_Rect dest_rect;

   dest_rect.x=10; dest_rect.y=10;
   dest_rect.w=100; dest_rect.h=100;

   /* Fill with a color a rectangle, update the window and wait 5 secs */
  /* Set the color for drawing operations */
   SDL_SetRendererDrawColor(p_ren,50,50,50,255);

  /* Draw a rectangular of the window dimension */
   SDL_RenderFillRect(p_ren, &dest_rect);

  /* Wait 5000 milliseconds before continuing */
   SDL_Delay(5*1000);

   /* Cleanup SDL */
   SDL_DestroyRenderer(p_ren);
   SDL_DestroyWindow(p_win);
   SDL_Quit();
   return 0;
}
```

```
typedef {
  int x,y;
  int w,h;
} SDL_Rect;
```

SDL_Render functions

La libreria SDL2 mette a disposizioni le seguenti primitive di disegno 2D:

SDL_RenderClear( )
SDL_RenderDrawLine[s]()
SDL_RenderDrawPoint[s]() //disegna un pixel su SDL_Window
SDL_RenderDrawRect[s]()
SDL_RenderFillRect[s]()
SDL_RenderPresent()

Ed altre ancora:

SDL_RenderReadPixels() //legge pixels da SDL_Window
....
SDL_RenderCopy()
....

SDL_Surface

La struttura SDL_Surface rappresenta un'area grafica di disegno, in memoria RAM.
Una Surface ha una width, una height (attributi w, h), un pixel format (color depth, alpha-channel, attribute format) ecc. Per creare una surface:

SDL_CreateRGBSurface( )

Una SDL_Surface deve essere liberata (free della memoria) con una chiamata a

SDL_FreeSurface( )

dopo aver finito di usarla.

Molte SDL function operano su SDL_Surface; per esempio

SDL_FillRect: riempie un rettangolo di una Surface con un colore.
SDL_BlitSurface: copia una Surface o una porzione di una Surface su un'altra o su una SDL_Window con un certo Rendering context.

La SDL_Surface era l'area di disegno nelle SDL 1.2, ed è mantenuta nelle SDL2 per compatibilità.

SDL_Surface

SDL non ha built-in function per il setting o getting di pixel su una SDL_Surface (cioè per disegnare un pixel con un colore o leggere il colore di un pixel).

E' però possibile accedere al "frame buffer" in memoria RAM e modificarne/leggerne il contenuto. Costruiamo una funzione di disegno di un punto/pixel ossia che scriva in memoria:

```
/*
* Set the pixel at (x, y) to the given value
* NOTE: The surface must be locked before calling this!
*/
void GC_PutPixel1(SDL_Surface *surface, int x, int y, Uint32 pixel)
{
    int bpp = surface->format->BytesPerPixel;
    /* Here p is the address to the pixel we want to set */
    Uint8 *p = (Uint8 *)surface->pixels + y * surface->pitch + x * bpp;
    ......
    *p = pixel
    .....
}
```

Slide 18 di **Hardware e Software per Grafica Interattiva:**

st_fm + M * y + x

SDL_Surface

Costruiamo una funzione per leggere un colore dalla memoria video:

```
/*
* Return the pixel value at (x, y)
* NOTE: The surface must be locked before calling this!
*/
Uint32 GC_GetPixel1(SDL_Surface *surface, int x, int y)
{
   int bpp = surface->format->BytesPerPixel;
   /* Here p is the address to the pixel we want to retrieve */
   Uint8 *p = (Uint8 *)surface->pixels + y * surface->pitch + x * bpp;
   ....
   return *p;
}
```

Esempio:

```
SDL_Surface *p_screen;
Uint32 col_pixel;
   SDL_LockSurface(p_screen);
   col_pixel=GC_GetPixel1(p_screen, x, y);
   /* modify the pixel color and write the pixel */
   GC_PutPixel1(p_screen, x, y, col_pixel);
   SDL_UnlockSurface(p_screen);
```

Attenzione: SDL_LockSurface

SDL_LockSurface blocca una surface affinché si possa accedere direttamente alle informazioni pixel. Fra chiamate a

SDL_LockSurface e SDL_UnlockSurface

si può scrivere e leggere dall'area *surface->pixels*, usando il formato pixel descritto in *surface->format*.

Alla fine si deve usare SDL_UnlockSurface per sbloccarla.

Non tutte le surface richiedono di essere bloccate. Se

SDL_MUSTLOCK(*surface*)

ritorna 0, allora si può leggere e scrivere in ogni istante.

```
if (SDL_MUSTLOCK(surface)) SDL_LockSurface(surface);
... accesso a info/operazioni sui pixel ...
if (SDL_MUSTLOCK(surface)) SDL_UnlockSurface(surface);
```

Fra una coppia di chiamate per bloccare/sbloccare non si devono fare chiamate di sistema operativo o chiamate a funzioni, in quanto si potrebbero avere blocchi di sistema.

SDL_Surface

Nota: Si noti che in SDL i lock di surface sono ricorsivi; questo significa che si possono fare più lock, ma che per ogni lock deve esserci un corrispondente unlock.

Esempio: funzione SDL_BlitSurface

```
SDL_BlitSurface(SDL_Surface *src, SDL_Rect *src_rect, SDL_Surface
                *dst, SDL_Rect *dst_rect):
```

```
typedef {
  int x,y,;
  int w,h;
} SDL_Rect;
```

Disegna/copia il rettangolo definito da src_rect della SDL_Surface src nel rattangolo definito da dst_rect sulla SDL_Surface dst.

Sebbene questa funzione legga e scriva pixel, NON deve essere fatta una chiamata alla SDL_LockSurface prima di essere usata

```
SDL_Surface *p_screen, *p_surface;
Uint32 col_pixel;
   SDL_BlitSurface(p_surface, src_rect, p_screen, dst_rect);
```

SDL_Surface

In SDL2, dopo aver “disegnato” su una SDL_Surface *p_surf, come si fa a visualizzarla? Si utilizza la seguente funzione che definisce una SDL_Surface associata ad una window.

```
SDL_Surface *screen;
   screen=SDL_GetWindowSurface(SDL_Window *win)
```

Quindi si copia il contenuto della SDL_Surface p_surf sulla SDL_Surface screen associata alla window, quindi si aggiorna la window

```
SDL_BlitSurface(p_surf, src_rect, screen, dst_rect);
SDL_UpdateWindowSurface(win);
```

In alternativa, per disegnare su una SDL_Surface, si può crea un Rendering Context per la SDL_Surface screen associata alla window, e si disegna direttamente su questa:

```
SDL_Renderer *ren;
   ren=SDL_CreateSoftwareRenderer( screen);
```

Libreria GCGraLib2

Una piccola libreria C per interfacciarsi con la libreria SDL2 e fare grafica 2D in modo semplice:

```
void GC_PutPixel1(SDL_Surface *surface, int x, int y, Uint32 pixel);
Uint32 GC_GetPixel1(SDL_Surface *surface, int x, int y);

void GC_DrawLine1(SDL_Surface *s, int x0, int y0, int x1, int y1);
void GC_DrawRect1(SDL_Surface *s,int ax,int ay,int width,int height);
void GC_DrawCircle1(SDL_Surface *s, int xin, int yin, int rad,Uint32 color);
void GC_FillCircle1(SDL_Surface *s, int xin, int yin, int rad,Uint32 color);

void GC_DrawRect(SDL_Renderer *r,int ax,int ay,int width,int height);
void GC_DrawCircle(SDL_Renderer *r, int xin, int yin, int rad);
void GC_FillCircle(SDL_Renderer *r, int xin, int yin, int rad);
```

(continua)

Libreria GCGraLib2 (continua)

```
void GC_DrawText(SDL_Render *r, TTF_Font *fonttodraw,
                 char fgR, char fgG, char fgB, char fgA,
                 char bgR, char bgG, char bgB, char bgA,
                 char text[ ], int x, int y, enum textquality quality)
```

dove:

```
enum textquality{ solid, shaded, blended };
```

Libreria GCGraLib2 (continua)

```
Uint32 GC_GetPixelImage(SDL_Surface *surface, int x, int y);

SDL_Surface* GC_LoadImage(char *file, int *exitstate);

void  GC_DrawImage(SDL_Surface *srcimg, int sx, int sy, int sw,
                   int sh, SDL_Surface *dstimg, int dx, int dy);
```

NOTA: la funzione GC_LoadImage è in grado di leggere una immagine in uno qualunque dei formati previsti dalle SDL (bmp, gif, ppm, jpg, png, ecc.), ma la GC_GetPixelImage è in grado di acquisire correttamente solo informazioni colore da immagini in formato ppm, jpg e png.

Esempi di codice

Cartella GCGraLib2: README_GCGraLib2 e i file

GCGraLib2.c

GCGraLib2.h

Cartella SDL2prg0: Makefile e vari programmi di esempio:

polygon2.c

polygonf2ren.c

....

Nota: nel seguito, per ogni argomento, verranno indicati i programmi di esempio da provare.

Gestione Input

La tastiera e il mouse generano eventi (memorizzati in una struttura SDL_Event)

```
typedef union{
   Uint32 type;
   SDL_WindowEvent window;
   SDL_KeyboardEvent key;

   SDL_MouseMotionEvent motion;
   SDL_MouseButtonEvent button;
   SDL_MouseWheelEvent wheel;
   SDL_JoyAxisEvent jaxis;
   SDL_JoyBallEvent jball;
   SDL_JoyHatEvent jhat;
   SDL_JoyButtonEvent jbutton;
   ....
} SDL_Event;
```

```
SDL_Event event;
```

La funzione `int SDL_PollEvent(SDL Event *ev)`

Controlla se ci sono eventi nella coda e ritorna il valore 1 se ce ne sono, 0 se non ce ne sono; se ce ne sono, il primo evento viene rimosso dalla coda FIFO e memorizzato nella struttura SDL_Event puntata da ev.

Gestione Input: Keyboard

Gli eventi sono differenziati per key pressed e released (SDL_KEYDOWN e SDL_KEYUP)

```
typedef struct{
  Uint32 type;
  Uint32 WindowID;
  Uint8 state;
  Uint8 repeat;
  SDL_keysym keysym;
} SDL_KeyboardEvent;
```

- event.key è una struttura SDL_KeyboardEvent
- event.key.keysym.sym è il codice virtuale in SDL per i tasti (SDLK_xxx (sia per press che release))

Gestione Input

SDLKey	**valori ASCII o nome comune**
SDLK_ESCAPE	'^[' escape
SDLK_0	'0'
SDLK_1	'1'
SDLK_2	'2'
....	
SDLK_a	'a'
SDLK_b	'b'
....	
SDLK_KP0	keypad 0
SDLK_KP1	keypad 1
SDLK_UP	up arrow
SDLK_DOWN	down_arrow
SDLK_RIGHT	right arrow
SDLK_LEFT	left arrow

Valori costanti che può assumere il campo:
event.key.keysym.sym

Gestione Input

```
SDL_Event event;
/* Handle all pending events */
while( SDL_PollEvent(&event) ){
  switch(event.type){
    case SDL_KEYDOWN: /* Key pressed */
      switch (event.key.keysym.sym) {
        case SDLK_LEFT:
          move_left();
          break;
        case SDLK_RIGHT:
          move_right();
          break;
      }
      break;
    case SDL_KEYUP: /* Key released */
      switch (event.key.keysym.sym) {
        case SDLK_LEFT:
        ....
      }
    case SDL_QUIT: /* Ctrl-C etc */
     ...
  }
}
```

Gestione Input: Mouse

Per il mouse ci sono due tipi di eventi: motion e button (press/release)

```
typedef struct{
  Uint32 type;
  Uint32 windowID;
  Uint32 state;
  Sint32 x, y;
  Sint32 xrel, yrel;
}
SDL_MouseMotionEvent;
```

```
typedef struct{
  Uint32 type;
  Uint32 windowID;
  Uint32 button;
  Uint32 state;
  Sint32 x, y;
}
SDL_MouseButtonEvent;
```

```
typedef struct{
  Uint32 type;
  Uint32 WindowID;
  Sint32 x, y;
  Uint32 direction;
}
SDL_MouseWheelEvent;
```

Si può controllare lo stato corrente del mouse con:

SDL_BUTTON( )

Gestione Input: Mouse

```
switch(event.type) {
    case SDL_MOUSEMOTION:
        /* event.motion is a SDL_MouseMotionEvent */
        player.x = event.motion.x; /* Xpos */

        /* Change y if left button is pressed */
        if (event.motion.state & SDL_BUTTON(SDL_BUTTON_LEFT))
          player.y = event.motion.y;
    case SDL_MOUSEBUTTONUP:
        /* event.button is a SDL_MouseButtonEvent */
    case SDL_MOUSEBUTTONDOWN:
        /* event.button is a SDL_MouseButtonEvent */
}
```

Gestione Input: Joystick

Per il joystick ci sono quattro tipi di eventi: axis motion, button, hat position e trackball motion:

```
typedef struct{
  Uint32 type;
  SDL_joystickID which;
  Uint8 axis;
  Sint16 valuel;
} SDL_JoyAxisEvent;
```

```
typedef struct{
  Uint32 type;
  SDL_joystickID which;
  Uint8 button;
  Uint8 state;
} SDL_JoyButtonEvent;
```

```
typedef struct{
  Uint32 type;
  SDL_joystickID which;
  Uint8 hat;
  Uint8 value;
} SDL_JoyHatEvent;
```

```
typedef struct{
  Uint32 type;
  SDL_joystickID which;
  Uint8 ball;
  Sint16 xrel, yrel;
} SDL_JoyBallEvent;
```

Esempi di codice

Nei seguenti codici si possono trovare esempi di ciclo degli eventi e gestione eventi:

polygon2.c

polygonf2ren.c

inter_polygon2ren.c

mouse_polygon2ren.c

ball.c

ball2.c

ball_time.c

Gestione Testo

Con la libreria SDL_ttf si può gestire il disegno di testo su una finestra grafica usando fonti truetype.
Il testo viene reso su una SDL_Surface che sarà copiata sullo schermo.

SDL_ttf deve essere inizializzato con

```
TTF_Init( )
```

carica una font e ritorna un puntatore a TTF_Font

```
TTF_OpenFont(char *name, int pts)
```

generare una SDL_Surface con un certo testo char* ed uno specifico color

```
TTF_RenderText_xx(TTF_Font*, char *, SDL_Color)
```

xx == Solid stamperà la fonte senza alpha-blending
xx == Blended usa alpha-blending (più lento ma migliori risultati)

Si copia la SDL_Surface con il testo su una SDL_Texture e questa la si rende/copia sulla window:

```
SDL_Texture *tex = SDL_CreateTextureFromSurface(ren, resulting_text);
SDL_RenderCopy(ren, tex, NULL, &dstrect);
```

Per rilasciare la texture memory e la font memory si usa

```
SDL_DestroyTexture(tex);
TTF_FreeFont(TTF_Font *)
```

Infine si chiude con:

```
TTF_Quit( )
```

Esempi di codice

Nei seguenti codici si possono trovare esempi di gestione testo:

provatext.c

mouse_polygon2ren.c

draw_data.c

Gestione Immagini

Con la libreria SDL_image si può gestire il caricamento e disegno di immagini in molti formati grafici come BMP, GIF, JPG, PPM, ecc.

SDL_Surface *IMG_Load(const char **file*)

Carica un *file* per essere usato come un'immagine in una nuova surface.
Questa chiama la IMG_LoadTyped_RW, con l'estensione utilizzata per il file e questo permette di includere tutti i tipi supportati.
Ritorna un puntatore all'immagine come una nuova SDL_Surface.

```
/* load sample.png into image */
SDL_Surface *image;

 image=IMG_Load("sample.png");
  if(!image) {
     printf("IMG_Load: %s\n", IMG_GetError());
    /* handle error */
  }
```

Esempi di codice

Nei seguenti codici si possono trovare esempi di gestione immagini:

provasdl.c

mouse_polygon2ren.c

draw_data.c

Gestione del Tempo

La libreria SDL mette a disposizioni alcune funzioni cross-platform per la gestione del tempo. Fondamentalmente fornisce due modi per gestire l'animazione di un oggetto ogni Tot millisecondi:

- usa una "timer callback function"
- si rileva il numero di millisecondi passati e si muove l'oggetto se sono passati i fissati Tot millisecondi.

SDL_GetTicks( ) rileva il numero di millisecondi trascorsi dalla inizializzazione della libreria SDL

SDL_Delay( ) attende uno specificato numero di millisecondi prima di ritornare

SDL_AddTimer( ) aggiunge un timer che chiamerà una callback function dopo uno specificato numero di secondi

SDL_RemoveTimer( ) rimuove un timer precedentemente aggiunto con la SDL_AddTimer

Semplice Applicazione SDL: header

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

#include <stdlib.h>
#include <SDL2/SDL.h>
#include <SDL2/SDL_image.h>
#include <SDL2/SDL_ttf.h>
```

Semplice Applicazione SDL: main

```
int main(int argc, char **argv) {
SDL_Window *win;
SDL_Renderer *ren;
SDL_Event event;
    SDL_Init(SDL_INIT_VIDEO);
    SDL_CreateWindow(“Title”,0,0,640, 480,SDL_WINDOW_SHOWN);
    SDL_CreateRenderer(win,-1,SDL_RENDERER_ACCELERATED);
    int done = 0;
    while ( ! done ) { /* Loop, drawing and checking events */
      if (SDL_PollEvent(&event) {
        switch(event.type){
          case SDL_QUIT : done = 1;   break ;
          case SDL_KEYDOWN :
                if ( event.key.keysym.sym == SDLK_ESCAPE )
                 done = 1;
                break;
        }else
        myDrawScene(win);
    }
    SDL_DestroyRenderer(ren);
    SDL_DestroyWindow(win);
    SDL_Quit();
    return 1;
  }
```

Semplice Applicazione SDL: la parte che disegna

```
void myDrawScene(SDL_Window *win)
{

      /* disegna tutto quello che si vuole, quindi…*/

    SDL_RenderPresent(win);
}
```

Cerchiamo di fare meglio

- Ridisegna la scena troppe volte!!
- Gestiamo un timer per ridisegnare
- Nel ciclo degli eventi mettiamo:

```
case GC_VIDEOEXPOSE:
        myDrawScene();
        break;
```

- Togliamo myDrawScene dal ciclo degli eventi
- Usiamo una nuova funzione redraw( )

```
void redraw(){
/* ci automandiamo un messaggio che ci fara' ridisegnare la finestra */
SDL_Event e;
  e.type=GC_VIDEOEXPOSE;
  SDL_PushEvent(&e);
}
```

Semplice Applicazione SDL: main

```
#define GC_VIDEOEXPOSE 999
int main(int argc, char **argv) {
SDL_Event event;
SDL_Renderer *ren;
SDL_Event event;
   SDL_Init(SDL_INIT_VIDEO);
   SDL_CreateWindow("Title",0,0,640, 480,SDL_WINDOW_SHOWN);
   SDL_CreateRenderer(win,-1,SDL_RENDERER_ACCELERATED);
   int done = 0;
   while ( ! done ) { /* Loop, drawing and checking events */
     if (SDL_PollEvent(&event) {
       switch(event.type){
        case GC_VIDEOEXPOSE:
              myDrawScene(win);
              break;
         case SDL_QUIT : done = 1;   break ;
         case SDL_KEYDOWN :
               if ( event.key.keysym.sym == SDLK_ESCAPE )
                done = 1;
               break;
        }
     else myTimer();
    }
```

Semplice Applicazione SDL: timer per ridisegnare

```
void myTimer()
{
Uint32 timeNow;

   timeNow=SDL_GetTicks();
/* si vuol disegnare una volta ogni tot secondi */
  if (timeNow-timeLast > elapsed)
  {
    redraw( );
    timelast=timeNow;
 }
  else
   ....
}
```

```
void myTimer()
{
   SDL_Delay(elapsed);
    redraw( );
}
```

Esempi di codice

Nei seguenti codici si possono trovare esempi di gestione del tempo:

ball.c

ball2.c

ball_time.c

polygonf2ren.c

Libreria SDL2

NOTA: la libreria SDL2 permette di creare più di una window per processo/programma.

Vedi esempio: mouse_polygon2ren.c