LinuxFocus article number 288
http://linuxfocus.org

Implementing a scroller in SDL graphics

by Leonardo Giordani
<leo.giordani(at)libero.it>

About the author:

I just received my diploma from the Faculty of Telecommunication Engineering in Politecnico of Milan. Interested in programming (mostly in Assembly and C/C++). Since 1999 works almost only with Linux/Unix.

Translated to English by:
Leonardo Giordani
<leo.giordani(at)libero.it>

Abstract:

This series of articles has the purpose of introducing the reader to the world of multimedia productions, also known as "demos". The whole Internet is full of informations about the matter, but few people does write such beautiful thing for Linux: my goal is to describe the theory of some graphical and sound effects and their implementation using the SDL library. Further informations can be found at

- www.libsdl.org: reading the code of open source demos and games are the best way to learn new solutions.
- www.lnxscene.org: a new site, but can be a nice place to exchange knowledge; you can find me there (sometimes) as "muaddib"

Prerequisites for the understanding of the article are:

- Basic knowledge of C language (sintax, loops, libraries)
- Basic knowledge of SDL library (basic functions, initializing) --> www.libsdl.org

The scroller

Many many thanks to Sam Lantinga for the SDL library.

A scroller is a part of a demo that write a moving sentence on the screen: it is one of the basic effect that you can find in a multimedia production and adds a bit of dynamicity to the text you want show to the user. In this article we will see how to build a basic scroller that moves the text from right to left.

The basic idea of a scroller is to copy a part of of the screen one or more pixel to the left (or to some direction). Executing this operation with a good speed you obtain the illusion of a continuous movement, and that's all.

The basic theory is not very complex; let's see how to translete all this in code terms: we will refer from now on to the concept of surface, well known to those who have some basic SDL programming skills. Working with the SDL library we have always to remember that we can use the powerful

`SDL_BlitSurface()` function, which let us copy part of an `SDL_Surface` identified by an `SDL_Rect` on another `SDL_Surface` identified by another `SDL_Rect` .

For example, imagine we defined and initialized two surfaces and two rectangles

```
#define WIDTH 800
#define HEIGHT 600

SDL_Surface *Src;
SDL_Surface *Dst;

Src = SDL_CreateRGBSurface(SDL_HWSURFACE, WIDTH, HEIGHT, 8,
       r_mask, g_mask, b_mask, a_mask);
Dst = SDL_CreateRGBSurface(SDL_HWSURFACE, WIDTH, HEIGHT, 8,
       r_mask, g_mask, b_mask, a_mask);

SDL_Rect BigArea = {0, 0, (WIDTH / 2), HEIGHT};
SDL_Rect SmallArea = {(WIDTH / 2) + 1, 0, (WIDTH / 2), (HEIGHT / 2)};
```

where we supposed the color mask to be already well initialized. Copying the two full surfaces implies a minimal effort

```
SDL_BlitSurface(Src, 0, Dst, 0);
```

Note that on the destiantion surface only the origin coordinates of the rectangle are considered and not the dimension: this means that the operation

```
SDL_BlitSurface(Src, &BigArea, Dst, &SmallArea);
```

is perfectly legal and copies the left half of the `Src` surface on the right half of the `Dst` surface.

Now performing the scroll of a surface should no more be a mistery: it is sufficient copying a part of the surface in a shifted rectangle on the same surface; clearly all this code have to be but in a loop, so the resulting program becomes a bit more complex, but the basic concept is simple. At every tick of the loop we use two rectangles, the second shifted respects to the first in the direction of the scrolling, and we copy the surface on itself from the first to the second rectangle.

```
SDL_Surface *temp;

SDL_Rect From = {1, 0, WIDTH, HEIGHT};
SDL_Rect First = {0, 0, 1, HEIGHT};
SDL_Rect Last = {WIDTH-1, 0, 1, HEIGHT};

temp = SDL_CreateRGBSurface(SDL_HWSURFACE, 1, HEIGHT, 8,
       r_mask, g_mask, b_mask, a_mask);
```

```
while(1){
  SDL_BlitSurface(Src, &First, temp, 0);
  SDL_BlitSurface(Src, &From, Src, 0);
  SDL_BlitSurface(temp, &First, Src, &Last);
  SDL_BlitSurface(Src, 0, Screen, 0);
}
```

As you can see, it is not sufficient to scroll the surface towards left: we have to reinsert from the right the pixels exited from the screen, or the scrolling surface will let behind itself the copies of the last column, generating a sort of "dragging" effect. We supposed to have already a surface linked to the screen. Now let's see a complete program that opens a 480x200 window and performs the continuous scolling of an image on it.

```
#include "SDL/SDL.h"
#include "SDL/SDL_image.h"

#define SCREEN_WIDTH 480
#define SCREEN_HEIGHT 200

#if SDL_BYTEORDER == SDL_BIG_ENDIAN
static const Uint32 r_mask = 0xFF000000;
static const Uint32 g_mask = 0x00FF0000;
static const Uint32 b_mask = 0x0000FF00;
static const Uint32 a_mask = 0x000000FF;
#else
static const Uint32 r_mask = 0x000000FF;
static const Uint32 g_mask = 0x0000FF00;
static const Uint32 b_mask = 0x00FF0000;
static const Uint32 a_mask = 0xFF000000;
#endif
```

This set of definitions is standard in (almost) every multimedia production.

```
static SDL_Surface* img_surface;
static SDL_Surface* scroll_surface;
static SDL_Surface* temp_surface;
```

These are the three surface we will use: `img_surface` will contain the image loaded from file, `scroll_surface` the shifted image and `temp_surface` the pixels we have to let enter again from right.

```
static const SDL_VideoInfo* info = 0;
SDL_Surface* screen;
```

A `SDL_VideoInfo` structure contains informations about video hardware, while the `screen` surface will point to the real screen.

```
int init_video()
{
  if( SDL_Init( SDL_INIT_VIDEO) < 0 )
    {
      fprintf( stderr, "Video initialization failed: %s\n",
               SDL_GetError( ) );
      return 0;
    }

  info = SDL_GetVideoInfo( );
```

```
  if( !info ) {
    fprintf( stderr, "Video query failed: %s\n",
             SDL_GetError( ) );
    return 0;
  }

  return 1;
}

int set_video( Uint16 width, Uint16 height, int bpp, int flags)
{
  if (init_video())
    {
      if((screen = SDL_SetVideoMode(width,height,bpp,flags))==0)
        {
          fprintf( stderr, "Video mode set failed: %s\n",
                   SDL_GetError( ) );
          return 0;
        }
    }
  return 1;
}
```

The `init_video()` function initializes the SDL video subsystem and fills up the `info` structure. The `set_video()` function tries to set a given video mode (dimensions and color depth).

```
void quit( int code )
{
  SDL_FreeSurface(scroll_surface);
  SDL_FreeSurface(img_surface);

  SDL_Quit( );

  exit( code );
}
```

This is the essential termination function, which deallocates all the resources we used and calls `SDL_Quit .`

```
void handle_key_down( SDL_keysym* keysym )
{
    switch( keysym->sym )
      {
      case SDLK_ESCAPE:
        quit( 0 );
        break;
      default:
        break;
      }
}
```

A "key pressed" event: in this case the ESC key.

```
void process_events( void )
{
    SDL_Event event;

    while( SDL_PollEvent( &event ) ) {
```

```
        switch( event.type ) {
        case SDL_KEYDOWN:
            handle_key_down( &event.key.keysym );
            break;
        case SDL_QUIT:
            quit( 0 );
            break;
        }
    }
}
```

The not lesser essential event management function.

```
void init()
{
  SDL_Surface* tmp;
  Uint16 i;

  tmp = SDL_CreateRGBSurface(SDL_HWSURFACE, SCREEN_WIDTH,
       SCREEN_HEIGHT, 8, r_mask, g_mask, b_mask, a_mask);

  scroll_surface = SDL_DisplayFormat(tmp);


  SDL_FreeSurface(tmp);
```

Let's work with a `tmp` surface to initialize `scroll_surface` and `temp_surface` . Both are converted in the video framebuffer format through the `SDL_DisplayFormat` function.

```
  img_surface = IMG_Load("img.pcx");
```

Here we load into img_surface the image saved in the file.

```
  for (i = 0; i < SDL_NUMEVENTS; ++i)
    {
      if (i != SDL_KEYDOWN && i != SDL_QUIT)
        {
          SDL_EventState(i, SDL_IGNORE);
        }
    }

  SDL_ShowCursor(SDL_DISABLE);
}
```

All events are ignored except for the pressure of a key and the exit from the program; moreover we disable the cursor.

```
int main( int argc, char* argv[] )
{
  SDL_Rect ScrollFrom = {1, 0, SCREEN_WIDTH, SCREEN_HEIGHT};
  SDL_Rect First = {0, 0, 1, SCREEN_HEIGHT};
  SDL_Rect Last = {SCREEN_WIDTH - 1, 0, 1, SCREEN_HEIGHT};
```

These are the three rectangles described in the article.

```
  if (!set_video(SCREEN_WIDTH, SCREEN_HEIGHT, 8,
      SDL_HWSURFACE | SDL_HWACCEL | SDL_HWPALETTE /*| SDL_FULLSCREEN*/))
   quit(1);
```

```
  SDL_WM_SetCaption("Demo", "");

  init();

  SDL_BlitSurface(img_surface, 0, scroll_surface, 0);
```

This code initializes the whole thing: set the video mode, writes the window title, calls `init()` and prepares `scroll_surface` .

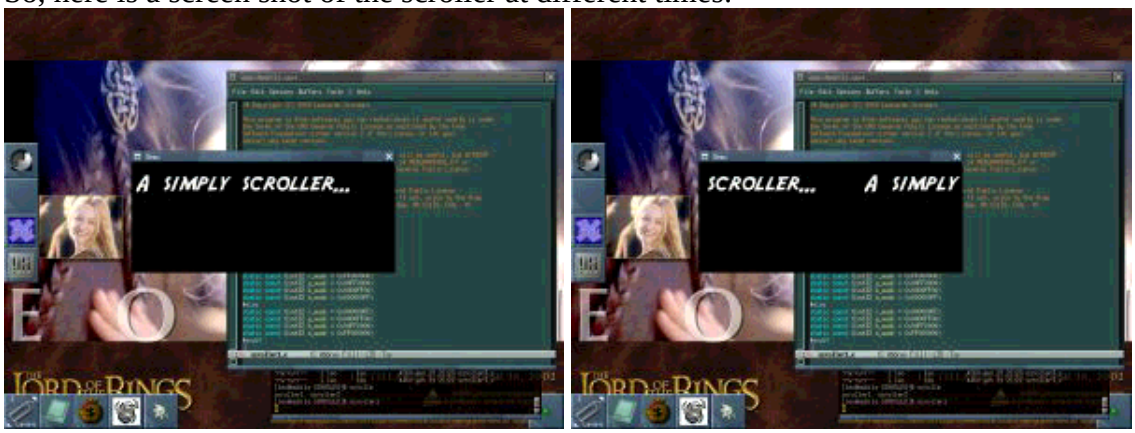
```
  while( 1 )
    {
      process_events();

      SDL_BlitSurface(scroll_surface, &First, temp_surface, 0);

      SDL_BlitSurface(scroll_surface, &ScrollFrom, scroll_surface, 0);

      SDL_BlitSurface(temp_surface, &First, scroll_surface, &Last);

      SDL_BlitSurface(scroll_surface, 0, screen, 0);

      SDL_Flip(screen);
    }

  return 0;
}
```

This is the loop described in the article: only the event control functions and the screen surface flip have been added.

As you can see the initialization work for the library is not short, but with the advantage that it is common to the whole demo, so while the code increases, initialization will become even more a small part of the full program, and it is a reusable and portable one.

A demo

So, here is a screen shot of the scroller at different times:



ps: You can send me comments, corrections and questions at my mail address or through the Talkback

page. Plese write me in english, german or italian.

Translation information:
it --> -- : Leonardo Giordani <leo.giordani(at)libero.it>
it --> en: Leonardo Giordani <leo.giordani(at)libero.it>

2005-01-14, generated by lfparser_pdf version 2.51