LinuxFocus article number 288
http://linuxfocus.org

Concevoir un effet de défilement de texte avec SDL

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

L´auteur:

Etudiant ingénieur en télécommunications à l'école Polytechnique de Milan, il travaille comme administrateur réseau et s'intéresse à la programmation (surtout en langage assembleur et en C/C++). Depuis 1999 il ne travaille que sous Linux/Unix.

Traduit en Français par:
Guy Passemard
<g.passemard(at)free.fr>

Résumé:

Cette série d'article a pour but d'introduire le lecteur dans le monde de la production multimedia, également connu sous le nom de "démos". L'internet abonde d'informations sur le sujet, mais peu de personnes écrivent de très belles choses pour Linux: mon but est de décrire la théorie de quelques effets graphiques et sonores et de les réaliser en utilisant la bibliothèque SDL. Vous trouverez plus ample information en suivant ces liens :

- www.libsdl.org: la lecture du code des démos et des jeux open source est le meilleur moyen d'apprendre de nouvelles solutions.
- www.lnxscene.org: un nouveau site, mais aussi un bon endroit pour échanger des connaissances; vous pouvez (parfois) m'y rencontrer sous le pseudo "muaddib"

Prérequis pour comprendre cet article :

- Les bases du langage C (syntaxe, boucles, bibliothèques)
- Les rudiments de la bibliothèque SDL (les principales fonctions, l'initialisation) --> www.libsdl.org

Le "scroller"

Vraiment un très grand merci à Sam Lantinga pour la bibliothèque SDL.

Un "scroller" est la partie d'une démo qui écrit une phrase défilante sur l'écran: c'est un des effets

élémentaires que vous pouvez trouver dans une production multimedia et qui ajoute un peu de dynamique au texte que vous voulez montrer à l'utilisateur. Dans cet article nous verrons comment construire un "scroller" simple, qui déplace le texte de la droite vers la gauche.

L'idée à la base d'un "scroller" est de copier une partie de l'écran, un ou plusieurs pixels vers la gauche (ou dans une autre direction). En exécutant cette opération à une bonne vitesse vous obtenez l'illusion d'un mouvement continu, et c'est tout.

La théorie de base n'est pas trop complexe; regardons comment traduire tout ceci en terme de code : nous allons nous référer à partir de maintenant au concept de la surface, bien connu de ceux qui possèdent quelques notions de programmation SDL. En travaillant avec la bibliothèque SDL, il faut toujours se rappeler que l'on peut utiliser la puissante fonction

`SDL_BlitSurface()` qui nous permet de copier une partie d'une `SDL_Surface` identifiée par une `SDL_Rect` sur une autre `SDL_Surface` identifiée par une autre `SDL_Rect` .

Par exemple, imaginez que nous définissions et initialisions deux surfaces et deux 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)};
```

où nous supposons que le masque de couleurs est déjà correctement initialisé. La copie de deux surfaces pleines nécessite un minimum d'effort

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

Notez que sur la surface de destination, seules les coordonnées d'origine du rectangle sont prises en compte et non la dimension : ceci montre que l'opération

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

est parfaitement correcte et effectue la copie de la moitié gauche de la surface `Src` sur sa moitié droite `Dst`

Maintenant l'exécution d'un défilement de surface ne devrait plus être un mystère: il suffit de copier une partie d'une surface et de la décaler dans un rectangle sur la même surface; pour plus de clarté tout ce code doit être placé dans une boucle, ainsi le programme devient un peu plus complexe, mais le concept de base reste simple. A chaque itération de boucle nous utilisons deux rectangles, le second est décalé par rapport au premier en respectant la direction du "scrolling", et nous copions la surface sur elle même du premier au 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);
}
```

Comme vous pouvez le voir, il ne suffit pas de faire défiler la surface vers la gauche: nous devons aussi réinsérer depuis la droite les pixels sortis de l'écran, sinon la surface de défilement laissera derrière elle les copies de la dernière colonne, générant un effet de "dragging". Nous sommes censés avoir déjà préparé une surface liée à l'écran. Maintenant, nous allons voir un programme complet qui ouvre une fenêtre de 480x200 et y effectue le défilement continu d'une image.

```
#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
```

Cet ensemble de définitions est standard dans (presque) toutes les productions multimedia.

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

Voici les trois surfaces que nous utiliserons : `img_surface` contiendra une image chargée à partir d'un fichier, `scroll_surface` l'image décalée et `temp_surface` les pixels que nous devons réintroduire à partir de la droite.

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

Une structure `SDL_VideoInfo` contient des informations concernant le matériel vidéo, quant à la surface `screen` elle pointera vers l'écran réel.

```
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;
}
```

La fonction `init_video()` initialise le sous-sytème vidéo SDL et remplit la structure `info` La fonction `set_video()` essaye de définir un mode vidéo donné (dimensions et profondeur des couleurs).

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

  SDL_Quit( );

  exit( code );
}
```

C’est la fonction essentielle pour terminer, elle libère toutes les ressources utilisées et appelle `SDL_Quit` .

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

Un évènement provoqué en appuyant sur une touche : dans ce cas la touche ESC.

```
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;
        }
    }
}
```

La non moins essentielle fonction de gestion d'évènements.

```
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);
```

Travaillons avec une surface `tmp` pour initialiser `scroll_surface` et `temp_surface` . Les deux sont converties au format du "framebuffer" vidéo par la fonction `SDL_DisplayFormat` .

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

Ici nous chargeons dans img_surface, l'image sauvegardée dans un fichier.

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

  SDL_ShowCursor(SDL_DISABLE);
}
```

Tous les évènements sont ignorés sauf l'appui sur une touche et la sortie du programme; de plus le curseur est désactivé.

```
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};
```

Ce sont les trois rectangles décrits dans l'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);
```

Ce code initialise le tout: choisit le mode vidéo, écrit le titre de la fenêtre, appelle `init()` et prépare `scroll_surface` .

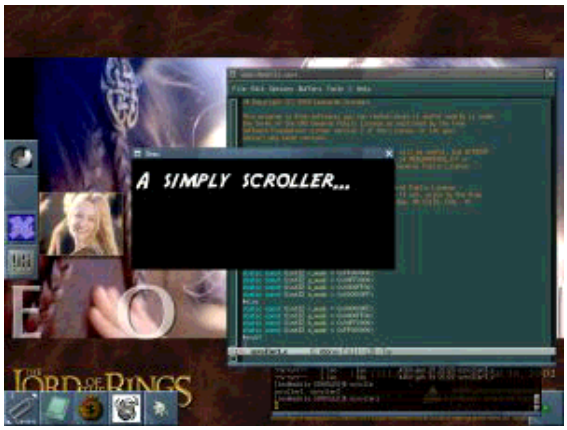
```
  while( 1 )
    {
      process_events();

      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;
}
```

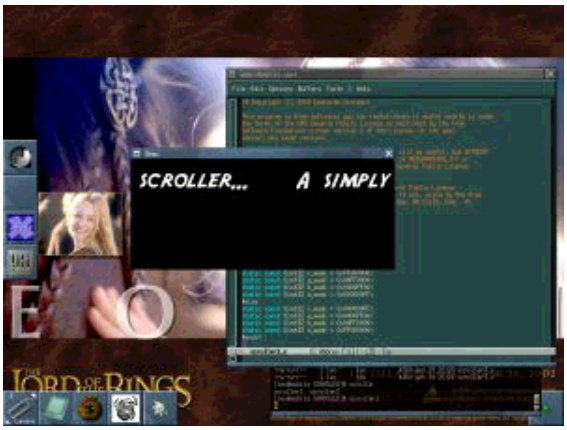
Ceci est la boucle présentée dans l'article: seules les fonctions de contrôle d'évènements et la rotation de la surface de l'écran ont été ajoutées.

Comme vous pouvez vous en apercevoir, l'initialisation de la bibliothèque n'est pas courte, mais a l'avantage d'être commune à toute la démo, et lorsque la taille du code augmentera, l'initialisation deviendra une partie de moindre importance dans un programme entier car elle est réutilisable et portable.

Une demo

Voici des photos d'écran du "scroller" à des moments différents:





ps: Vous pouvez m’addresser vos commentaires, corrections et questions à mon adresse de courrier électronique ou sur la page de discussion. S’il vous plaît, veuillez m’écrire en anglais, allemand, ou italien.

2005-01-14, generated by lfparser_pdf version 2.51