Implementation eines Scrollers mittels SDL−Grafik

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

About the author:

Vor kurzem erhielt ich mein Diplom von der Fakultät für Telekommunikations − Engineering am Politecnico in Milano. Mein Hauptinteresse liegt im Programmieren (Assembler und C/C++). Seit 1999 arbeite ich fast ausschliesslich mit Linux/Unix.

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

Abstract:

Diese Artikelserie möchte den Leser in das Gebiet der Multimediaproduktion einführen, welches auch als "Demos" bekannt ist. Im Internet finden wir eine Fülle von Informationen darüber, doch nur wenige Leute schreiben diese wunderbaren Sachen für Linux: mein Ziel ist es, die Theorie einiger der Graphik− und Audioeffekte zu beschreiben sowie deren Implementation unter der Verwendung der DSL−Bibliothek. Weitere Informationen finden wir bei:

- http://www.libsdl.org/: Eine der besten Methoden, um neue Lösungen zu finden, ist das Studium des Codes von Open−Source−Demos und −Spielen.
- http://www.lnxscene.org/: Eine neue Webseite, aber sie kann eine Fundstelle sein, um Wissen auszutauschen, ich bin dort (manchmal) als "muaddib" zu finden.

Vorbedingungen, um diesen Artikel zu verstehen:

- Grundkenntnisse in C (Syntax, Schleifen, Bibliotheken)
- Grundkenntnisse der SDL−Library (grundlegende Funktionen, Initialiserung) −−> www.libsdl.org

Der Scroller

Recht herzlichen Dank an Sam Lantinga für die SDL−Bibliothek.

Ein Scroller ist der Teil eines Demos, welcher einen sich bewegenden Text auf den Bildschirm schreibt: einer der einfachsten Effekte einer Multimediaproduktion. Er lässt den Text für den Betrachter etwas dynamischer

erscheinen. Dieser Artikel wird uns zeigen, wie man einen einfachen Scroller baut, der den Text von rechts nach links bewegt.

Die Grundidee eines Scrollers ist es, einen Teil des Bildschirms einen oder mehrere Pixel nach links (oder irgendeine Richtung) zu kopieren. Ausführen dieser Operation mit einer vernünftigen Geschwindigkeit gibt uns die Illusion einer gleichmäßigen Bewegung, das ist alles.

Die grundlegende Theorie ist nicht sehr komplex, wir wollen all das jedoch in Code−Terminologie ausdrücken: von hier an beziehen wir uns nur noch auf das Konzept der Surface − denen mit Grundkenntnissen von SDL bestens bekannt. Die SDL−Bibliothek erlaubt uns die wirkungsvolle

">SDL_BlitSurface() −Funktion zu benutzen. Diese ermöglicht uns, einen Teil eines ">SDL_Surface indentifiziert als Rechteck, ">SDL_Rect, auf ein anderes Surface,">SDL_Surface, ebenfalls als Rechteck ,">SDL_Rect, zu kopieren.

Zum Beispiel stell dir vor, wir definieren und initialisieren zwei Surfaces und zwei Rechtecke:

```
#define WIDTH 800
#define HEIGHT 600
 !supportEmptyParas]>]>p>p>
SDL_Surface *Src;
SDL_Surface *Dst;
 !supportEmptyParas]>]>p>p>
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);
 !supportEmptyParas]>]>p>p>
SDL_Rect BigArea = {0, 0, (WIDTH / 2), HEIGHT};
SDL_Rect SmallArea = {(WIDTH / 2) + 1, 0, (WIDTH / 2), (HEIGHT / 2)};
```

für welche die Colormask bereits initialisiert wurde. Das Kopieren der zwei vollständigen Surfaces beansprucht wenig Aufwand.

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

Hier ist anzumerken, dass auf der Ziel−Surface nur die Ursprungskoordinaten des Rechtecks und nicht dessen Abmessungen in Betracht gezogen werden: d.h. dieser Schritt

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

ist vollständig legal, er kopiert die linke Hälfte des ">Src −Surface auf die rechte Hälfte des ">Dst −Surface.

Damit sollte die Ausführung des Scrolls kein Geheimnis mehr sein: es genügt, einen Teil des Surface in ein örtlich verschobenes Rechteck auf der gleichen Surface zu kopieren. Es ist klar, dass all der Code sich in einer Schleife befinden muss, damit wird das Programm etwas komplexer − das Grundkonzept ist jedoch immer noch einfach. Bei jedem Durchlauf der Schleife benutzen wir zwei Rechtecke, das zweite verschiebt sich relativ zum ersten in der Richtung des Scrollens und wir kopieren die Surface vom ersten zum zweiten Recheck auf sich selbst.

```
SDL_Surface *temp;
 !supportEmptyParas]>]>p>p>
SDL_Rect From = {1, 0, WIDTH, HEIGHT};
SDL_Rect First = {0, 0, 1, HEIGHT};
SDL_Rect Last = {WIDTH-1, 0, 1, HEIGHT};
```

```
 !supportEmptyParas]>]>p>p>
temp = SDL_CreateRGBSurface(SDL_HWSURFACE, 1, HEIGHT, 8,
r_mask, g_mask, b_mask, a_mask);
 !supportEmptyParas]>]>p>p>
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);
}
```

Wie leicht ersichtlich ist, reicht es nicht aus, die Surface nach links scrollen zu lassen: wir müssen auch von rechts die weggewanderten Pixel wieder auffüllen oder die scrollende Surface hinterlässt eine Kopie der letzten Spalte, das erzeugt eine Art "dragging"−Effekt. Wir setzen voraus, dass wir schon eine Surface auf unseren Bildschirm gelinkt haben. Als nächstes werfen wir einen Blick auf ein vollständiges Programm, das ein 480x200 Fenster öffnet und den kontinuierlichen Scroll eines Bildes darauf durchführt.

```
#include "SDL/SDL.h"
#include "SDL/SDL_image.h"
 !supportEmptyParas]>]>p>p>
#define SCREEN_WIDTH 480
#define SCREEN_HEIGHT 200
 !supportEmptyParas]>]>p>p>
#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
```

Diese Liste von Definitionen ist in (fast) allen Multimediaproduktionen Standard.

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

Dieses sind die drei Surfaces, die wir benutzen werden: `">img_surface` enthält das Image der Datei, `">scroll_surface` das verschobene Image und `">temp_surface` die Pixel, die wir von rechts wieder einfügen müssen.

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

Eine `">SDL_VideoInfo`−Struktur enthält Informationen über Video−Hardware, während die `">screen` −Surface auf den wirklichen Bildschirm zeigt.

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

```
 info = SDL_GetVideoInfo( );
 !supportEmptyParas]>]>p>p>
 if( !info ) {
 fprintf( stderr, "Video query failed: %s\n",
 SDL_GetError( ) );
 return 0;
 }
 !supportEmptyParas]>]>p>p>
 return 1;
}
 !supportEmptyParas]>]>p>p>
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;
}
```

Die ">init_video()–Funktion initialisiert das SDL–Video–Subsystem und füllt die ">info–Struktur. Die ">set_video()–Funktion versucht, einen existierenden Video–Modus einzustellen (Dimensionen und Farbtiefe).

```
void quit( int code )
{
 SDL_FreeSurface(scroll_surface);
 SDL_FreeSurface(img_surface);
 !supportEmptyParas]>]>p>p>
 SDL_Quit( );
 !supportEmptyParas]>]>p>p>
 exit( code );
}
```

Das ist die notwendige Beendigungsfunktion, sie gibt alle Ressourcen, die wir benutzt haben, wieder frei und ruft ">SDL_Quit auf.

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

Ein "Taste drücken"– Ereignis: in diesem Fall die ESC – Taste.

```
void process_events( void )
{
 SDL_Event event;
 !supportEmptyParas]>]>p>p>
 while( SDL_PollEvent( &event ) ) {
 !supportEmptyParas]>]>p>p>
```

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

Die nicht weniger wichtige "event management"– Funktion.

```
void init()
{
 SDL_Surface* tmp;
 Uint16 i;
 !supportEmptyParas]>]>p>p>
 tmp = SDL_CreateRGBSurface(SDL_HWSURFACE, SCREEN_WIDTH,
SCREEN_HEIGHT, 8, r_mask, g_mask, b_mask, a_mask);
 !supportEmptyParas]>]>p>p>
 scroll_surface = SDL_DisplayFormat(tmp);
 !supportEmptyParas]>]>p>p>
 !supportEmptyParas]>]>p>p>
 SDL_FreeSurface(tmp);
```

Laßt uns mit einer "`>tmp`–Surface arbeiten, um "`>scroll_surface` und "`>temp_surface` zu initialisieren. Beide werden in das Video–Framebuffer– Format mit der "`>SDL_DisplayFormat`–Funktion umgewandelt.

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

Hiermit laden wir unsere gespeicherte Image–Datei in img_surface.

```
 for (i = 0; i < SDL_NUMEVENTS; ++i)
 {
 if (i != SDL_KEYDOWN && i != SDL_QUIT)
 {
 SDL_EventState(i, SDL_IGNORE);
 }
 }
 !supportEmptyParas]>]>p>p>
 SDL_ShowCursor(SDL_DISABLE);
}
```

Alle Ereignisse – ausser dem Drücken einer Taste und dem Beendigen des Programms – werden ignoriert. Ausserdem setzen wir den Cursor ausser Funktion.

```
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};
```

Hier sind die drei vorgehend beschriebenen Rechtecke:

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

```
 SDL_WM_SetCaption("Demo", "");
 !supportEmptyParas]>]>p>p>
 init();
 !supportEmptyParas]>]>p>p>
 SDL_BlitSurface(img_surface, 0, scroll_surface, 0);
```

Dieser Code initialisiert den ganzen Vorgang: er stellt den Video−Mode ein, schreibt den Window−Titel, ruft `">init()` auf und bereitet `">scroll_surface` vor.

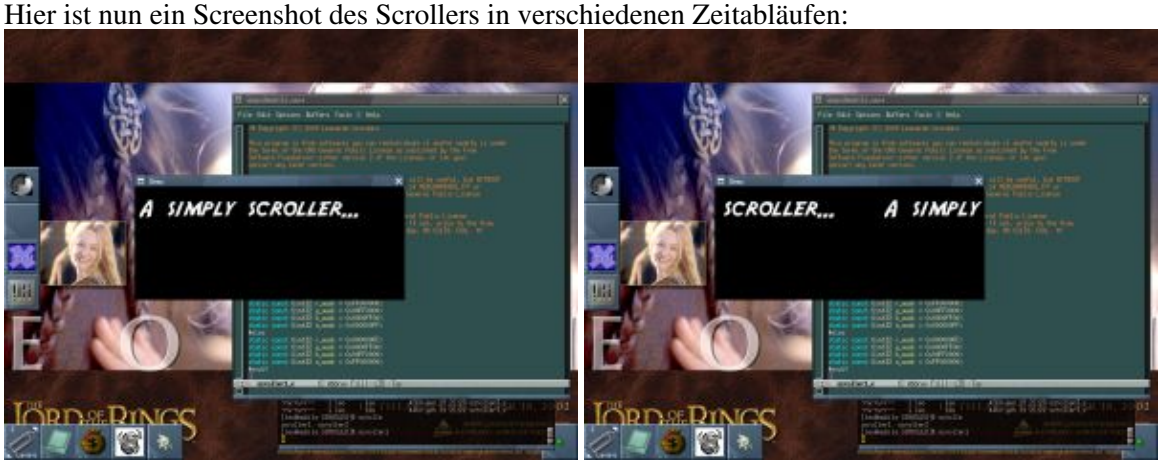
```
 while( 1 )
 {
 process_events();
 !supportEmptyParas]>]>p>p>
 SDL_BlitSurface(scroll_surface, &First, temp_surface, 0);
 !supportEmptyParas]>]>p>p>
 SDL_BlitSurface(scroll_surface, &ScrollFrom, scroll_surface, 0);
 !supportEmptyParas]>]>p>p>
 SDL_BlitSurface(temp_surface, &First, scroll_surface, &Last);
 !supportEmptyParas]>]>p>p>
 SDL_BlitSurface(scroll_surface, 0, screen, 0);
 !supportEmptyParas]>]>p>p>
 SDL_Flip(screen);
 }
 !supportEmptyParas]>]>p>p>
 return 0;
}
```

Das ist der oben erwähnte Loop: nur die Ereignis−Kontrollfunktionen und der Screen−Surface−Flip wurden hinzugefügt.

Wie wir sehen können, sind die Vorbereitungen für die Bibliothek nicht gerade kurz, hat aber den Vorteil, dass sie für die gesamte Demo zuständig ist. Die Initialisation wird mit dem wachsenden Code ein kleinerer Teil des Gesamtprogramms − sie ist wiederbenutzbar und portabel.

Ein Demo

Hier ist nun ein Screenshot des Scrollers in verschiedenen Zeitabläufen:



P.S. Ihr könnt mir Kommentare, Korrekturen und Fragen an meine Emailadresse oder durch die Talkbackseite schicken. Bitte schreibt in Englisch, Deutsch oder Italienisch.

Translation information:
it −−> −− : Leonardo Giordani <leo.giordani(at)libero.it>
it −−> en: Leonardo Giordani <leo.giordani(at)libero.it>
en −−> de: Jürgen Pohl <sept.sapins(at)verizon.net>

2005−01−11, generated by lfparser_pdf version 2.51