Linux Game Development

Written in LaTeXby Michael Durrer

[1]

[1] Last update on 24. Januar 2007

Inhaltsverzeichnis

Teil I

Einführung

Kapitel 1

Programmieren unter Linux

1.1 Einleitung

Sie haben dieses Buch gekauft, von meiner Seite als PDF runtergeladen oder wie auch immer erhalten und fragen sich nun vielleicht Folgendes:

Wie programmiere ich ein Spiel oder eine Anwendung für Linux?
Womit fange ich überhaupt an?
Kann ich das überhaupt?

So zumindest waren meine überlegungen, als ich vor vielen Jahren angefangen habe, mich mit der Interna von Rechnern und der Digitaltechnik herumzuschlagen. Und in Erinnerung an die Problematik dieses Beginns (der heute in der riesigen Fälle an Informationen im Internet immerhin erleichtert worden ist), möchte ich dieses Buch all denen zur Verfügung stellen, die sich gerade inmitten jener Phase befinden und dringend Informationen suchen oder bereits diverse Grundlagen besitzen und darauf aufbauen möchten. Natürlich sei dieses Buch auch all Jenen ans Herz gelegt, die sich schlichtweg weiterbilden möchten in **C/C++** oder der **SDL-Bibliothek**.

Da sich hier aus obigen Gründen einige vom Wissensstand doch sehr verschiedene Zielgruppen herauskristallisieren, habe ich das Buch in mehrere spezifische Sektionen aufgeteilt:

- Programmieren unter Linux
- Die Grundlagen von ANSI-C und C++
- Die Grundlagen von Python
- Einstieg in SDL mit Beispielen in ANSI-C & C++ & Python
- Emulation - Fremde Systeme simulieren
- Open Source Lizenzen - Was ist das eigentlich genau?

Dabei ist der Schwerpunkt auf Linux gelegt, doch habe ich auch durchgehend Wert auf Portabilität gelegt.
Speziell in der Spielewelt wäre eine häufigere Verwendung der SDL angebracht, ist sie doch portabel auf allen gängigen Plattformen wie Windows Vista, Linux oder Mac OS X.
Portabel heisst in diesem Sinne, dass durch Verwendung von SDL für Steuer-, Eingabe-, Audio- und Videogeräte diese universal programmierbar werden. Eine einheitliche API (*Application Programming Interface*) steuert die Geräte nun an, der Programmierer sieht nur noch die API und deren Dokumentation. Spezifische Eigenheiten für jeweilige Geräte und Betriebssysteme, z.B. verschiedene Grafikkarten, übernimmt nun die SDL-Schicht komplett und übersetzt sie in die Sprache der jeweiligen Betriebssysteme

Zusätzlich habe ich mich für dieses Buch auf 3 Programmiersprachen festgelegt: **C, C++ & Python**.
C/C++ ist quasi ein Industrie-Standard und wird seit Jahren in der Applikationsentwicklung da eingesetzt, wo man schnelle und übersichtliche Anwendungen benötigt, zu diesen gehören auch grafiklastige Applikationen. Seit einigen Jahren sind aber die Prozessoren mittlerweile an einem Punkt angelangt, wo sich die Entwicklung nicht mehr so stark an Geschwindigkeit multipliziert wie früher.

Die meisten Rechner sind heutzutage schon bei einer Geschwindigkeit angelangt, mit der die meisten Spiele auf dem Markt lauffähig sind, zudem wird immer mehr Rechenleistung auf die enorm leistugnsfähigen Grafikkarten ausgelagert, die auch auf langsameren Rechnern eine gute Grafikleistung hervorzaubern können und nicht mehr zwingend eine 'schnelle Programmiersprache' benötigen.

An dieser Stelle springt Python ein: Python ist eine skriptbasierte Sprache, die ebenfalls objektorientiert aufgebaut ist und somit eine exzellente übersicht bietet. Desweiteren ist sie sehr einfach zu erlernen und kann ebenfalls mit SDL umgehen, es richtet sich daher eher an die Programmieranfänger, was jedoch nicht heissen soll, dass man damit nicht genauso komplexe Applikationen bauen könnte.
Wie auch immer Ihre Fähigkeiten und Ihr Wissen derzeit liegen, Ich hoffe Sie finden mit diesem Buch ein Themengebiet, dass Sie weiterbringt und Ihnen einen einfachen Einstieg in die Thematik ermöglicht. Beachten Sie jedoch bitte, dass ich Ihnen nahelege, gute Vorkenntnisse in ANSI-C oder C++ mitzubringen, da ich den Grossteil dieses Buches doch darauf stütze und auf jeden Fall früher oder später für professionelle Anwendungsentwicklung gelernt werden muss.

An den Teil über *Emulation*, sollten sich nur echte Profis ranwagen, welche C/C++ bereits in- und auswändig kennen. Auch wenn vielleicht nicht alles beim ersten Durchlesen verständlich sein mag für Anfänger, interessant ist es dennoch allemal und vermittelt Fach- und Hintergrundwissen, dass sehr hilfreich beim Verständnis der Programmierung sein kann. Auch wenn dies nach nun nach viel Arbeit anhören mag, bringt es doch viel Spass mit sich. Und eben diesen wünsche ich Ihnen nun mit diesem Buch!

Ihr Michael Durrer

Teil II

SDL - Der Simple DirectMedia Layer

Kapitel 2

SDL-Initialisierung

2.1 Einige Hintergrundinformationen...

SDL steht, wie schon im Titel erwähnt, für die Simple DirectMedia Layer Library. Der Name sagt uns schon, dass uns SDL einen *simplen* und *direkten* Zugriff auf Medien, bzw. Multimedia-Hardware bieten soll.

Sie wurde entwickelt von Sam Latinga, während er bei Loki Software, bekannt für ihre Linux-Portierungen von Windows-Spielen, als leitender Programmierer angestellt war. SDL diente als Basis für die Portierung vieler Windows-Spiele, darunter *Civilization: Call to Power*und *Descent 3*, um nur Einige zu nennen.
Dank der vielen Fähigkeiten der Bibliothek und ihren OpenGL-Erweiterungen hat, vorallem aber auch dank der LGPL-Lizenzierung (mehr dazu im Kapitel **Open Source Lizenzen - Was ist das eigentlich genau?**), haben zu einer enormen Verbreitung dieser Entwicklungsbibliothek geführt. Mittlerweile entwickeln ein ganzer Haufen an professionellen als auch private Entwickler an SDL weiter und sorgen dafür, dass auch weiterhin aktuelle Technologien leicht ansprechbar bleiben über ein vereinfachtes Interface.

Durch die grosse Verbreitung ergibt sich noch ein besonders interessanter Vorteil: Die Plattformunabhängigkeit. Mittlerweile unterstützt SDL mehr als nur alle gängigen Betriebssysteme wie Linux, Windows und Mac OS X sowie die meisten Hochsprachen wie C/C++ sondern auch viele Nischen-Betriebssysteme und -Plattformen. Unter Anderem AmigaOS, SEGA Dreamcast, Microsoft XBox, Sony Playstation u.v.m.

Ganz im Gegensatz zu DirectX von Microsoft, welches ja alles Andere als cross-platform-fähig ist...

Unter Windows hat SDL zudem einen kleinen (selbst unverschuldeten) Haken: Microsoft lässt den Zugriff auf die Multimedia-Hardware, insbesondere Grafikkarten, nur über ihr eigenes Entwicklungs-Kit zu, nämlich DirectX. Somit kann auch die SDL-Schicht nur auf die DirectX-Schicht aufbauen und ist somit gezwungenermassen leicht langsamer unter Windows, wenn man das Programm mit einem Kompilat auf dem gleichen Rechner, jedoch unter einem anderen Betriebssystem testet. Dieser fällt jedoch nicht allzustark ins Gewicht und seien wir doch ehrlich:
Uns ist die einfache Programmierung und das Erreichen eines grösseren Zielpublikums durch Cross-Platform-Kompatibilität ein paar Frames pro Sekunde wert, oder?

2.2 Installation

Zuerst brauchen wir natürlich eine saubere SDL-Installation für unseren Compiler (in unserem Fall GCC).

2.2.1 Header-Dateien

Um SDL benutzen zu können, müssen wir natürlich in unserer Hauptdatei (z.B. *main.c* oder *main.cpp*) die SDL.h Header-Datei inkludieren:

#include <SDL.h>

Nun stehen uns alle Funktionen und Prozeduren der SDL-Welt zur Verfügung! Willkommen in SDL! Als nächsten Schritt müssen wir einen Modus initialisieren, zum Beispiel um Video-spezifische Sachen darzustellen den Video-Modus.

2.3 SDL Modi initialisieren

SDL kann in mehreren Modi initialisiert werden, jedoch werden nicht immer alle benötigt. Dafür kann man mehrere Modi kreuzen. So braucht man beispielsweise in einigen Programmen gar keinen Audio-Modus oder es wird bei selbstablaufenden Programmen nichteinmal Input/Eingabe benötigt.

Deswegen kann man Ressourcen (und damit Leistung!) sparen, indem wir nur das initialisieren, was wir auch wirklich benötigen.

Als Allererstes müssen wir SDL selber initialisieren, wir nehmen ersteinmal den Video-Modus. Dafür benötigen wir den Befehl *SDL_Init()*:

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

Wir ersehen daraus, dass wir sogenannte Flags übergeben, hier nehmen wir den Video-Modus:

```
SDL_Init(SDL_INIT_VIDEO);
```

Nun haben wir den Video-Modus initialisiert und können bereits andere Video-Modi initialisieren. Es gibt einige solcher Flags, hier die komplette Liste:
SDL_INIT_INIT Timer initialisieren.
SDL_INIT_VIDEO Video initialisieren.
SDL_INIT_INPUT Eingabegeräte (Joystick, Maus, Tastatur) initialisieren.

Doch zuvor wollen wir noch sehen, wie man mehrere Modi miteinander kombiniert, wir wollen den Input-Modus zusätzlich! Dies bewerkstelligen wir, indem wir die Werte miteinander in einer ODER-Tabelle verknüpfen. Die ODER-Tabelle verknüpft Bits nach folgendem Schema:

0	0	0
0	1	1
1	0	1
1	1	1

Das heisst, wenn mindestens eine der beiden Werte, bzw. deren jeweilige Bits die miteinander verknüpft werden, 1 ist, ist das Ergebnis (C) 1. So kann man gut Werte miteinander verknüpfen, bzw. fehlende Bits auffüllen. Das ODER-Zeichen ist in C & C++ die Pipe: |
Ein kleines Beispiel: Wir wollen Wert A mit Wert B verknüpfen, da wir den Wert 255 (= 8 Bit) haben möchten.

```
Wert A: 1111 0000 (binär) = F0 (hexadezimal) = 128
Wert B: 0000 1111 (binär) = 0F (hexadezimal) = 15
```