Simple DirectMedia Layer (SDL)

Fabian Fagerholm

Seminar:Software engineering and Computer Games

Spring 2006

Department of Computer Science

University of Helsinki

(English translation of original, 12th June 2006.)

Contents

- **Introduction**
  - What is SDL?
  - A very brief history
  - Open Source
- API: a look into the functionality of SDL
- Architecture
- Bindings to higher-level languages
- Case

Introduction: What is SDL?

- SDL is a low-level multimedia library.
  - Sound playback
  - Keyboard reading
  - Joystick/game controller reading
  - Graphics (2D)
  - OpenGL-compatible (3D)
  - System routines (parallel programming, event handling, etc.)
- SDL is portable, which means it works on several different hardware platforms.
  - Enables game development for several platforms at the same time.
  - Enables porting of the game to a new platform later.
  - Works with more than 10 operating systems and on more than 10 hardware platforms. (Is this useful? On the other hand, does it hurt?)

Introduction: A very brief history

- The first version of SDL was developed by Sam Lantinga, then an employee of Loki Software.
  - Loki made Linux versions of Windows and Mac games (Sim City 3000 Unlimited, Quake III Arena, among others).
  - SDL was among the tools used in the games.
  - The company ceased activitites in the beginning of 2002.
  - Sam Lantinga is still the lead developer of SDL, and currently works at Blizzard Entertainment (World of Warcraft, Diablo II, etc.).
- SDL is written in C.
  - Very portable and efficient.
  - Works directly with C++.
  - Bindings to other languages are easy to make.

Introduction: Open Source

- SDL is Open Source software (licensed under the LGPL)
  - Publising the source code of your own game is not required under certain conditions. The goal is to keep SDL freely available.
  - Development is different from a closed software product. This requires practical awareness about the open development model.
  - Development can be followed in real time through the SDL project source code repository (http://www.libsdl.org/cgi/cvsweb.cgi)
  - The lack of a background organization might lead to the thought that the project cannot be trusted.
  - However, the license ensures that you can even develop your own version.
  - Many companies are interested in free availability and open development models. As an example, Nokia develops and publishes Open Source software.
- Before using: be aware of what this means!
  - Free Software / Open Source is a cross-disciplinary phenomenen about which there is a lot of conflicting information.

Contents

- Introduction
- **API: a look into the functionality of SDL**
  - General functionality
  - Graphics
  - Event handling
  - Sound
  - Parallel programming
  - OpenGL
- Architecture
- Bindings to higher-level languages
- Case

API: General functionality

- The API of SDL consists of around 200 functions and a set of structs.
- Initialization: SDL_Init
  - Initializes SDL and its subsystems (graphics, souns, etc.)
  - Not all subsystems have to be used:
    - SDL_Init(SDL_INIT_AUDIO | SDL_INIT_VIDEO);
- Loading of shared libraries: SDL_LoadObject(”libGL.so”);
  - Enables plugins to be loaded at run-time.
- Freeing resources: SDL_Quit();
  - You have to free the resources you allocated yourself, however.

API: Graphics

- Most of the API is graphics-related.
  - 3D graphics is a separate issue, these routines operate on a 2D frame buffer.
- Display and graphics card features, settings the display mode
- Partial updating of the frame buffer; swapping buffers in double-buffering
- Colour management and conversions between different colour systems
- Controlling transparency with key colour or alpha channel
- Image block operations (BLIT, Block Image Transfer)
- Operations related to the mouse pointer
- OpenGL operations
- Overlay operations for video

API: Graphics

- Example

```
SDL_Surface *image = IMG_Load("SDL_logo.png");
```

```
typedef struct SDL_Surface {
  Uint32 flags;                /* surface flags */
  SDL_PixelFormat *format;     /* pixel format (bits per pixel, etc.) */
  int w, h;                    /* width, height */
  Uint16 pitch;                /* length of surface scanline in bytes */
  void *pixels;                /* pointer to pixel data */
  SDL_Rect clip_rect;          /* surface clipping rectangle */
  int refcount;                /* used when freeing surface */
  /* This structure also contains private fields not shown here */
} SDL_Surface;
```

API: Event handling

- Events are caused by the user or the system external to the game, for example
  - Pressing a key on the keyboard
  - Clicking or moving the mouse
  - Using the joystick / game controller
  - Resizing the game window
- Events are stored in a queue.
- The game processes the queue at regular intervals or pauses to wait for events to arrive.
  - Filters can be applied to process only certain kinds of events.
- User-defined events are possible: SDL_UserEvent
  - Could be used to transmit movements by an AI player or for synthesized events by a testing program.
  - Not practical for game logic?

API: Sound

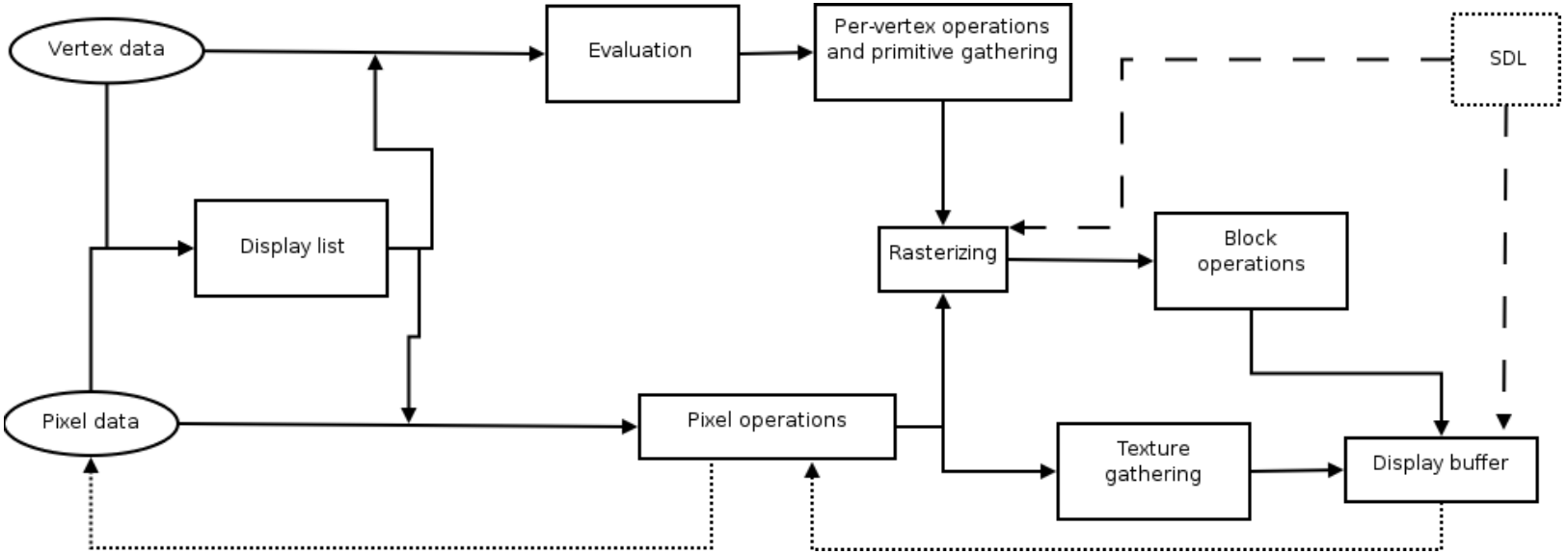
- The sound features of SDL are modest
- Initializing of sound hardware, freeing of resources
- Conversion of sound data, e.g. changing frequency and sample rate to match hardware capabilities
- Simple mixing
  - Repeated iterations cause distortion and clipping, because the mixing is not based on an accumulator and uses simple addition.
- In 3D games, it is often desirable to position the sound in 3D space, which requires a suitable library such as OpenAL.
  - Simulation of sound direction
  - Sound volume depending on distance
  - Effect of motion on sound (Doppler-effect)

API: Parallel programming

- Parallel programming is tedious to implement in a portable way.
- SDL offers the most common tools for parallel programming:
  - Threads
  - Mutex constructs
  - Semaphores
  - Condition variables
- In Linux, for example, implementation using POSIX threads, which are created by the operating system and can use SMP hardware.
- In other environments, an internal implementation may be used (threads are simulated in the SDL process).
- More details in parallel programming literature or, for example, the Computer Science department course Concurrent Programming.

API: OpenGL

- SDL itself does not offer 3D features
  - Acceleration is platform dependent, development is very quick
- OpenGL can be used for programming 3D graphics
- SDL can initialize the display for use with OpenGL
  - SDL_GL_LoadLibrary, SDL_GL_SetAttribute, ...



Contents

- Introduction
- API: a look into the functionality of SDL
- **Architecture**
  - Modularity
  - SDL as part of a game engine
- Bindings to higher-level languages
- Case

Architecture: modularity

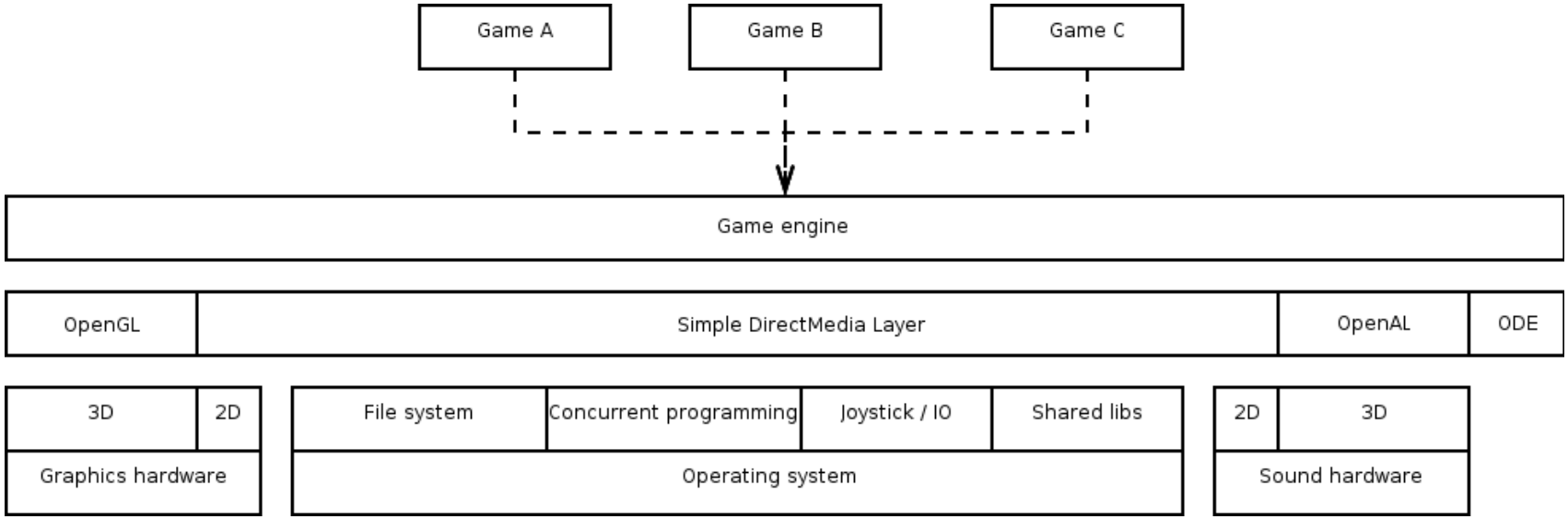
- SDL is modular
  - The API defines data structures, on which the functions operate
  - A module can provide creation and modification operations for the interface
  - The modularity is contract-based – C and C++ do not prevent wrongful use of the data strucures
  - A standard library extends the feature set: SDL_Image, SDL_mixer, SDL_net, SDL_rtf, SDL_ttf
- Example: support for image formats in the SDL_Image module
  - Supports GIF, JPEG, PNG formats, among others
  - Returns an SDL_Surface structure regardless of the file format
  - The SDL_Surface structure is the basic data structure that the 2D graphics operations operate on

Architecture: SDL as part of a game engine

- From a software engineering perspective, code reuse is essential
- The game engine isolates the reusable parts of a game
  - Graphics engine (2D and/or 3D)
  - Sound engine
  - Game world physics
  - General game logic, e.g. board game logic, first-person shooter logic, ...
  - Etc.
- SDL is not a game engine, but
- it can be part of one
  - Loki Software games worked this way

Architecture: SDL as part of a game engine

- A sketch of an Open Source game engine
  - OpenGL: partly abstracted; loading objects from files, etc.
  - OpenAL: each object acts as a sound source
  - ODE: Open Dynamics Engine, game world physics modeling
  - SDL: connects the pieces, provides operating system and portability routines



Contents

- Introduction
- API: a look into the functionality of SDL
- Architecture
- **Bindings to higher-level languages**
  - Overview
  - Python / PyGame
- Case

Bindings: overview

- Development is faster with a higher-level language
  - No need to do memory management
  - Quick prototyping
  - The game engine can include all resource-intensive functionality, the game logic is programmed in a higher-level language
- Bindings provide the operations of SDL as operations of the higher-level language
- Typically, the bindings form a very thin layer – the same functions are available, but
- sometimes higher-level operations are included
- SDL bindings are available for 20 languages, including C#, Java, Lua, Perl, PHP, Python, Ruby and Smalltalk

Bindings: Python / PyGame

- As an example, PyGame for Python
  - Important and interesting; Python works on the Symbian Series 60 phones and Nokia 770 device
- PyGame offers both the low-level functionality of SDL and its own higher-level functionality
  - The Sprite class: a graphical object in 2D games
  - The Group class: contains many Sprite-objects
  - Surfarray: pixel-level graphics operations, could be used for e.g. special visual effects
- Instead of a traditional game look, PyGame suggests a Model-View-Controller design pattern
  - An thread for each object
  - Project-level benefits, when game logic and graphics are separate
  - Distributed data structures in network games are easier to code separately
  - Testing is easier: unit tests for each part (Model, View) using PyUnit

Contents

- Introduction
- API: a look into the functionality of SDL
- Architecture
- Bindings to higher-level languages
- **Case**
  - Battle for Wesnoth, an adventure game

Case: Battle for Wesnoth

- A turn-based board game with an adventure theme
  - Open Source project, started 2003
  - Translated to 20+ languages
- Portable: Linux, Windows, Mac OS X, BeOS, Solaris, FreeBSD, OpenBSD, NetBSD
- Written in C++
- 60000+ lines of code
  - 2 000 000 USD (COCOMO-Basic)

Case: Battle for Wesnoth

- An object model built in C++ using SDL functionality
- Example: threading::thread
  - Uses thread functionality of SDL

```
...
thread::thread(int (*f)(void*), void* data) :
    thread_(SDL_CreateThread(f,data))
{}
...
void thread::join()
{
    if(thread_ != NULL) {
        SDL_WaitThread(thread_,NULL);
        thread_ = NULL;
    }
}
...
```

Summary

- SDL is a multimedia library
  - low-level operations
  - portability is central
  - freely available, open development model
- Simple API, no high-level operations
  - OpenGL for 3D graphics
- Modular architecture
  - contract-based
  - suited for use in a game engine
- Lots of bindings to high-level languages
- In real use
  - requires a lot of effort due to its low-level design

Sources

- Loki Software Inc. and John R. Hall: *Programming Linux Games*. No Starch Press, Inc., San Fransisco, CA, USA, 2001.
- Erik Yuzwa: *Game Programming in C++: start to finish*. Charles River Media, Inc., Massachussets, USA, 2006.
- Simple DirectMedia Layer Project, Simple DirectMedia Layer: http://www.libsdl.org/
- OpenAL Project, OpenAL: http://www.openal.org/
- SDL Logo by Arne Claus
- Battle for Wesnoth: http://www.wesnoth.org/