Chapter 3

Linux Gaming APIs

I still remember the first game programming book I ever read. By Dave Roberts, it was entitled *PC Game Programming Explorer*, and it demonstrated game programming with a game called Alien Alley. This was actually a neat game, especially for one intended as a book example: its graphics were smooth, the artwork was top-notch, and it ran well on my rather underpowered system. It would be easy for me to write such a game today, even in a matter of a few hours. But to a neophyte game programmer, it seemed a towering monolith.

Back in the days of DOS-based gaming, programmers generally wrote games by issuing commands directly to the computer's hardware. There were only a few popular types of sound cards on the market, and many were at least partially compatible with Creative Labs' Sound Blaster. Input devices were trivial to program: accessing the mouse required only a few assembly language instructions (interrupt 33h, for those who remember), reading the joystick's position was a matter of a dozen lines of code, and there were several easy ways to collect keyboard input. Video programming was the hardest part of game development at the time: Although nearly every computer had a VGA-compatible display chip, coaxing fast and smooth graphics out of it took a significant amount of skill (due to some of the brain-dead limitations of the PC architecture). In fact, Alien Alley was mostly video code.

Times have changed, arguably for the better. Very few game programmers actually write register-level video code these days; instead, they rely on

prewritten interfaces (such as OpenGL, SDL, and DirectDraw). Direct hardware hacking is fun, but it slows down game development and usually produces unportable code (with the unfortunate effect that many "old school" games are extremely difficult to port to modern environments). Even if I could find the floppy disk that came with my copy of *PC Game Programming Explorer*, I doubt that I could port Alien Alley to Linux in any reasonable amount of time, simply because it depends on certain hardware-level features of the original VGA graphics adapter.

DOS programs are given free reign of the entire system; they can freely access memory or hardware ports, and they are effectively allowed to shove the operating system out of the way. Linux programs, on the other hand, are not generally allowed direct access to the system's hardware; they must either use interfaces provided by the Linux kernel or obtain special permissions, requiring the program to be executed under the godlike root account (a potential security risk). The Linux kernel also prevents programs from directly accessing certain areas of the system's memory. In return for these restrictions, Linux is able to prevent applications from interfering with one another, thereby ensuring the system's stability and security.

The bottom line is that we'll probably want to avoid talking directly to the system's multimedia hardware, but instead use one of many existing libraries for the purpose. It saves time and effort, and libraries are usually more fully developed and stable than code written for a particular game.

This chapter tours the variety of game-programming toolkits available under Linux. Most are free and open (indeed, I am wary of any Linux toolkit that isn't these days). If you intend to use these toolkits, familiarize yourself with the terms of the GNU Library General Public License (LGPL): It *is* possible to legally develop closed source, commercial software using LGPL libraries under certain conditions; something that has been a frequent source of confusion among developers.

Multimedia programming is a broad field, and so we have divided our tour into several categories. Some packages provide several types of functionality, and they will be mentioned more than once. Finally, some capabilities are provided by the Linux kernel itself, in which case we will simply refer to "the kernel" or "Linux."

Graphics APIs

Linux offers several options for graphics programming. Most of today's Linux games use the X Window System in some way, as it is almost universally available, well supported, and at least tolerably fast.[1] Recently the Linux framebuffer device interface has been making inroads into gaming, and this interface has a lot of potential. Finally, SVGALib provides a way to get extremely fast access to SVGA-compatible video devices.

SVGALib

As its name implies, SVGALib is a library for programming Super VGA-compatible video hardware, which is extremely fast because it directly accesses the system's video hardware. SVGALib has fallen out of favor recently, due to its inconvenient interface, its failure to fully support many of today's video chipsets, and its demand for root privileges. Furthermore, it is known to conflict with the X Window System, and in some cases it is incompatible with Linux's new framebuffer device system. While SVGALib is still under development, it is reasonable to predict that its use will continue to decline.

SVGALib is distributed with a sister library called vgagl (not to be confused with the OpenGL library). The vgagl library provides higher-level drawing and blitting functions that make an SVGALib programmer's life a bit easier. SVGALib also includes sublibraries for keyboard and mouse access.

If you really want to mess with SuperVGA video cards, don't mind locking up your console occasionally, and don't care too much about wide compatibility, SVGALib may be worth looking into. Otherwise, your hacking effort is probably better spent elsewhere.

---

[1] The X Window System is incredibly flexible, and it's really not a bad platform for gaming. However, its design requires all graphics data to pass through certain predefined channels, and the use of extensions is required to achieve acceptable game performance in most cases (among these extensions are shared memory access and the XVideo extension). X was never really intended for today's level of high-speed graphics processing. Some people think X should be replaced with a new system, but I believe that it just needs a bit of reworking in some areas. X has a lot going for it.

GGI

General Graphics Interface (GGI) is a massive, general-purpose, multitargeted graphics library that provides a complete graphics system for games and other applications. Its companion library, GII, provides portable input device support, and games that use it are meant to be easily portable to any platform. GGI does not depend on any one method of accessing graphics devices; instead, it provides a system of "back ends" that can support just about anything remotely resembling a graphics device. The GGI Project is also working on a kernel-based graphics infrastructure, KGI. GGI is free software, distributed under the GNU LGPL. The GGI Project's Web site is `http://www.ggi-project.org`.

SDL

Simple DirectMedia Layer (SDL) is a cross-platform multimedia library developed with commercial game porting in mind. (In fact, it has already been used to port a number of games from Windows to Linux, including most of Loki's titles.) SDL supports almost all of the major operating systems, including Linux, Windows, BeOS, and MacOS. In addition to fast graphics support, SDL provides interfaces for playing sound, accessing CD-ROM drives, and achieving portable multithreading.

SDL is also an excellent library for free software projects: Released under the GNU LGPL, it has everything a programmer needs to write fast, portable games. SDL has accumulated a collection of user-contributed libraries that provide additional functionality for game developers.

We will discuss the SDL library in detail later. SDL's Web site is `http://www.libsdl.org`, and a helpful group of SDL enthusiasts (including myself)[2] gathers on IRC at irc.openprojects.net, `#sdl`.

---

[2] My name on IRC is "overcode." I'm not difficult to find.

ClanLib

ClanLib is a C++ game-programming library that, like SDL, stresses platform independence and optimal use of the system's underlying multimedia resources. Released under the GNU LGPL, ClanLib's design is very clean and extensible.

ClanLib is a higher-level library than SDL: Whereas SDL provides a relatively small set of C functions for accessing the computer's hardware in a portable way, ClanLib provides a complete C++ infrastructure for game development. We will cover SDL rather than ClanLib in this book, but ClanLib is certainly a worthy contender. You can find more information about ClanLib at `http://www.clanlib.org`.

OpenGL

OpenGL is a 3D graphics API designed by Silicon Graphics and developed by an Architecture Review Board (ARB) of graphics industry leaders. Although it was not originally intended as a game-programming library, OpenGL has found a place as a convenient interface standard for hardware-accelerated 3D graphics, and therefore lends itself well to gaming. The Mesa 3D Graphics Library is a free implementation of the OpenGL specification, and there are Mesa-based Linux drivers for several popular 3D accelerator cards.

Unfortunately, we can't cover OpenGL here in the detail it deserves (3D graphics is a subject of its own), but we'll at least demonstrate how to gain access to OpenGL from within SDL programs. This particular combination allows us to use the rendering power of hardware-accelerated OpenGL with the various amenities provided by SDL, and it is an excellent platform for developing games. Loki Software has successfully used SDL and OpenGL to port several commercial games to Linux, including Heavy Gear II and Soldier of Fortune.

For more information on OpenGL, see the most recent version of the OpenGL ARB's *OpenGL Programming Guide*, or visit `http://www.opengl.org`.

Plib

Plib is the collective name for several individual game-programming libraries written by Steve Baker. The purpose of this library is to build a usable game

programming environment out of the OpenGL GLUT toolkit (more on GLUT in the next chapter). This collection includes sg ("Simple Geometry," routines for fast 3D math), ssg ("Simple Scene Graph," for manipulating 3D scene data), pui ("Picoscopic User Interface," a simple menu and dialog box system), sl ("Sound Library", a portable sound interface), and several other useful components. Plib is available at `http://plib.sourceforge.net`. These libraries are free software, available under the GNU LGPL.

Glide

Glide is 3Dfx's native 3D programing library, designed specifically for 3Dfx graphics chips. It is a much lower-level library than OpenGL, serving mainly as a consistent interface for all video cards based on 3Dfx chipsets. Since 3Dfx no longer has a virtual monopoly in the 3D accelerator business, Glide has lost a certain amount of popularity recently. With the advent of accelerated OpenGL under Linux, there are very few good reasons to use Glide for new game projects, and now that 3Dfx is out of business it's even less of an issue. It is mentioned here only because it has been an influential API during the past few years.

Xlib

Some game programmers eschew all of these "programmer-friendly" libraries in favor of using the X Window System directly (via the native Xlib API). While experienced programmers may achieve small performance gains this way, they do so at the expense of portability and simplicity.

Xlib is not particularly difficult to use, but it is meant to be used as a base for constructing other toolkits, rather than as a library for writing actual applications. Xlib is a bit too verbose for my taste, but you might find it enjoyable. If you've ever written an application with the Win32 API, you have a good idea of what Xlib programming is like, except that in most cases Xlib requires even more library calls to get anything done. Remember that toolkits such as SDL and ClanLib already use a number of Xlib's tricks to achieve their level of performance, and if you code for Xlib directly, you'll be duplicating this work.

If you're interested in learning Xlib (perhaps not a bad exercise, whether or not you actually intend to use it), you'll want to get one or two of the books from the official X Window System documentation series. See the Bibliography for more info.

Graphical User Interface Toolkits

Many games use menus and dialogs to let the user make configuration changes and select the type of game to play. In many cases it's practical to build an ad hoc interface for a particular project, but games with complex settings might benefit from a more substantial user interface toolkit. There are plenty of good GUI packages to choose from.

GTK+

Originally developed to serve as the GNU Image Manipulation Program's user interface, GTK+ (formerly just GTK) is an enormous GUI library that somewhat resembles the time-tested Motif toolkit. GTK+ is implemented on top of an abstraction layer called GDK, freeing GTK+ from low-level concerns like input gathering and pixel format conversion.

GTK+ is implemented in pure C, but C++ wrapper libraries are available. Its programming model takes a bit of getting used to, but it is powerful enough for building interfaces for large applications. It would be a major hassle to port GDK/GTK+ to work with anything but the X Window System or Microsoft Windows, so you can pretty much forget about using it to develop games for the framebuffer console.

The GTK+ project is online at `http://www.gtk.org`.

Tk

The Tk toolkit was originally created as a windowing interface for the Tcl scripting language, but it's since found its way into a number of other environments. It is an extensible and flexible GUI toolkit for X11, Windows, and MacOS. Tk is tied to Tcl, but you can still develop your application in C and only use Tcl to build the interface. (If you don't mind a bit of extra effort, you can bypass Tcl entirely, but Tk wasn't really designed for this.)

Tk is available under the same (extremely liberal) license as Tcl, and it can be modified and used in any type of application with very few restrictions. More information is available on `http://www.tcltk.org`.

Fltk

Fltk stands for "fast, light toolkit." It is a very small C++ GUI toolkit that works on several different platforms (and is easily portable to others). Fltk requires very little of the underlying platform, and this makes it a good candidate for integrating into existing graphics systems (games, for instance). This toolkit is released under the GNU LGPL, and more information is available from `http://www.fltk.org`.

Qt

Qt is a comprehensive, portable application development system for C++. It shares some similarities with Microsoft's MFC toolkit, but it's refreshingly different in implementation. Qt is portable between UNIX and Windows, and there is even an embeddable version of Qt for handheld devices. It's really not fair to call Qt a GUI toolkit; it does serve that purpose, but it also provides basic data structures, file I/O, networking, and image loading and saving.

TrollTech (a free software–friendly Norwegian company) created and maintains Qt as a commercial product, and you need to buy a license if you intend to use Qt in proprietary software. The Linux version is available under both the GNU General Public License and the custom Q Public License, but these require all unlicensed Qt applications to be free. Qt is great for creating free software and for serious commercial development, but it's probably not what you want if you're interested in small-scale, nonfree development.

More information on Qt is available at `http://www.trolltech.com`.

SDL GUI Support

There's no "official" SDL GUI toolkit, but there are a few user-contributed libraries that fill this niche. The SDL_gui library provides basic things like frames, menus, and widgets, while the SDL_console library implements a

Quake-like popup console system. Both of these libraries are free software, and you can hack them to your liking (provided, of course, that you contribute your modifications back to the community at large).

These and other user-contributed SDL addons are available on `http://www.libsdl.org`.

Audio APIs

Linux supports most of today's sound cards. There are two competing standards for kernel-level sound support—OSS and ALSA—but fortunately neither is difficult to work with, and games commonly support both.

OSS

The Open Sound System (OSS) is the original sound-programming interface for Linux. Maintained by 4Front Technologies, OSS provides a consistent kernel-based interface to sound hardware. Its API is not especially pretty, but if you close your eyes and pretend you're doing something fun you can almost forget about it.

OSS supports most of today's sound cards, but some of the newer drivers are not free and require a commercial OSS license. The free portions of OSS (OSS/Free) are included in the Linux kernel (and are no longer directly maintained by 4Front).

There are two types of OSS programs: "nice" and "rude." Nice OSS programs are likely to work on just about anything that remotely claims to be OSS compatible, including vendor-supplied drivers, FreeBSD's sound system, and ALSA's OSS emulation module. In fact, most OSS programs are basically nice. Rude OSS programs do unusual things with the driver, such as memory-mapping the driver's DMA buffer. While the maintainers of OSS discourage this, some people do it anyway (Quake 3 is a notable example). We'll discuss a variety of OSS programming techniques in Chapter 5.

More information on OSS is available from 4Front Technologies at `http://www.4front.com`.

ALSA

Advanced Linux Sound Architecture (ALSA) is a community project that seeks to surpass OSS in all areas. The ALSA team has created a complete set of kernel-level sound card drivers, an easy-to-use programming interface, and a facility for emulating OSS. ALSA is not without its fair share of quirks, but it is a viable alternative to OSS for sound support and, with few exceptions, games that support OSS are also compatible with ALSA. It would be good to see ALSA grow in popularity since it has a lot of functionality and a lot of promise. The only serious problem with ALSA is that it is somewhat of a moving target; its API changes frequently. For more information on ALSA, visit `http://www.alsa-project.org`. We'll address ALSA programming in Chapter 5.

ESD

The Enlightened Sound Daemon (ESD, also called EsounD) is a sound server that allows multiple applications to share a single sound card. ESD-aware applications send their sound streams to ESD, and ESD mixes them internally into a single output stream. Some people love ESD, and some hate it; it has its fair share of technical problems, but results are acceptable in most cases. The main problem with ESD (other than its bugginess and lack of documentation) is the basic fact that it takes time for audio data to travel over a network, and this results in a significant delay before sound actually gets to the soundcard. ESD currently uses a fixed-sized buffer, regardless of the type of network or sound card. This latency can be rather disruptive for gameplay, but it's usually not a problem for music playback and other things that don't need to be precisely timed.

Recently some sound card drivers have started to support multiple device opens; that is, the driver allows multiple programs to use the sound card at once. This renders ESD more or less obsolete, but these drivers are in the minority right now.

ESD is an excellent software package, but programming information is very sparse, other than a few spare comments in the header file and various ESD-enabled projects that users have written. We will cover the basics of ESD programming in Chapter 5.

OpenAL

The Open Audio Library (OpenAL) is an environmental 3D audio library that supports just about every major platform. It aims to provide an open replacement for proprietary (and generally incompatible) 3D audio systems such as EAX and A3D. OpenAL can add realism to a game by simulating attenuation (degradation of sound over distance), the Doppler effect (change in frequency as a result of motion), and material densities. OpenAL has been used in several Linux game ports, including Heavy Gear II and Sid Meier's Alpha Centauri.

The OpenAL Web site is `http://www.openal.org`, and we will cover its API in Chapter 5.

Scripting Libraries

Tcl

Tool Command Language (Tcl) is a very simple extension language designed to automate a variety of tools. It often loses out because some people try to use it as a replacement for Perl (which it is not), but its simple syntax and convenient extension mechanism make it an ideal candidate for game scripting. Tcl is good at processing strings, but it is a poor choice for high-volume number crunching and data manipulation. We will implement a game scripting engine with Tcl in Chapter 6.

The command-line Tcl interpreter and extension libraries are available as source and binaries from `http://www.scriptics.com`. Although Tcl is commercially maintained, it is free software.

Guile and MzScheme

Scheme is a modern programming language that draws heavily from Lisp. It was designed primarily by Guy Steele in 1979, and it has evolved quite a bit since then. Scheme tends to scare away novices due to its prefix notation, its heavy use of recursion, and the simple fact that it is a Lisp derivative, but advanced users generally find it an amazingly expressive language. Scheme can be parsed and executed very quickly, and it is sufficiently powerful to serve as an excellent game

scripting language. The decision to use Tcl instead of Scheme for our scripting examples was difficult, but I felt that Tcl would make for more straightforward examples. However, Scheme would probably provide better performance.

Guile is the official GNU extension language. It is a reasonably complete Scheme implementation, but its documentation is extremely sparse. You can find it at `http://www.gnu.org/guile`.

MzScheme is a complete and actively maintained Scheme system from Rice University (and others). It implements the latest official Scheme standard (R5RS) almost completely, and it extends the language in various ways to make it more practical as a general-purpose programming language. MzScheme functions both as a standalone Scheme interpreter and an embeddable scripting library. If you're interested in using Scheme as an extension language, MzScheme would be an excellent choice. It is available at `http://www.cs.rice.edu/PLT`.

Python and Perl

You're probably familiar with Python and Perl, and you may already be proficient in one of these languages. While most commonly used as standalone scripting languages, Python and Perl can also be embedded in applications to provide modular scripting support. We won't be using these languages in this book (we'll use Tcl instead), but their scripting interfaces are not terribly difficult (very similar to Tcl, which we'll discuss in Chapter 6). Perl is superb at string processing, and Python has a bit of an object-oriented slant. Which language is better suited to game development is anybody's guess.

Perl and Python are available from `http://www.perl.org` and `http://www.python.org`, respectively, and each language comes with plenty of online documentation.

Networking APIs

Networked gaming is big, and it is here to stay. There are several networking interfaces for Linux, but almost all of them revolve around the BSD sockets API that became a standard part of UNIX years ago.

BSD Sockets

A socket is a UNIX file descriptor that designates a network connection rather than a file on disk. Sockets can be thought of as telephone handsets; they are communication endpoints through which data can be transferred in either direction. Sockets are most commonly used with TCP/IP, the stack of protocols behind the Internet.

The advantage of programming with TCP/IP sockets is that TCP/IP is an incredibly versatile protocol. Some version of the BSD sockets API can be found in nearly every operating system, including Linux, Windows, BeOS, and Mac OS. TCP/IP can be used for both local (LAN) and wide-area (WAN) networking, and the protocol can be adapted to the nature of a particular game.

Chapter 7 focuses on socket programming. Even if you decide to use an additional toolkit for convenience, it is important to understand how sockets and the underlying network protocols operate.

OpenPlay

OpenPlay is the successor to NetSprocket, Apple's network gaming support library. It is a cross-platform library (implemented in C), and it compiles on Linux as well as Windows and MacOS. OpenPlay is released under the terms of the Apple Public Source License, which is a corporate-friendly license that seems to be remotely inspired by the GNU GPL. OpenPlay is a substantial API designed to compete with Microsoft's closed and proprietary DirectPlay. OpenPlay shows promise, but its Linux port is still under development.

It remains to be seen whether OpenPlay for Linux will catch on. Some Linux developers seem to distrust Apple (not always for rational reasons), but the finished port of OpenPlay will have a lot to offer. OpenPlay is available on Apple's public source site, `http://publicsource.apple.com`.

IPX and SPX

Internetwork Packet Exchange (IPX) is a simple networking protocol similar to the Internet's underlying IP protocol, and Sequenced Packet Exchange (SPX) is a higher-level protocol similar to the Internet's TCP protocol. These protocols

(often collectively referred to as IPX) were designed by Novell for its NetWare line of products. IPX has fallen out of favor, but it is still used in a number of games. IPX is fine for small private LANs, but it is not ideal for large networks. Should you choose to support IPX in your games, the Linux kernel provides the necessary networking code (via the normal BSD sockets interface). It is not terribly difficult to support both TCP/IP and IPX with the same networking code.

File Handling

Games often need to load images and audio samples from files. This can be a bit of a trick with today's complex file formats and compression techniques. Fortunately, you can usually avoid doing this decoding yourself—there are Linux-compatible libraries for just about every type of image or sound file you could possibly want to load. Many of these libraries are free software.

libpng and libjpeg

These two libraries allow you to load Portable Network Graphic (**.png**) and JPEG (**.jpg**) images, respectively. PNG is an excellent general-purpose image format that compresses images without loss in detail. It is based on a completely open specification, and it is widely supported by image manipulation programs. JPEG is an older, "lossy" image format that does a good job with landscapes and other natural scenes but produces noticeably lousy results with precise images such as line art. JPEG is also an open standard.

If you need to add support for PNG or JPEG images to a game, these libraries are the way to go. It would not be a good idea to try to implement either format yourself unless you have a lot of time on your hands. We'll use these libraries in this book, albeit indirectly: the SDL_image library (Chapter 4) links against them to provide seamless PNG and JPEG loading support.

libpng is the offical PNG reference library, and it is available at `http://www.libpng.org`. libjpeg is maintained by the Independent JPEG Group at `http://www.ijg.org`. These libraries are included in most Linux distributions.

libaudiofile and libsndfile

libaudiofile and libsndfile are libraries for loading audio data from files. Each can read and write a wide assortment of file formats. There is a lot of functional overlap between these two libraries, but they have different interfaces. libsndfile is probably the more convenient of the two, and we will use it for loading wave files in Chapter 5. libaudiofile has a slightly more arcane (but perhaps more powerful) interface, but it can be a bit annoying to use.

libsndfile was designed and written by Erik de Castro Lopo, and it is available under the GNU LGPL license. libaudiofile was originally implemented by Silicon Graphics for its multimedia workstations, but it has since been largely reimplemented as free software, and it has been officially adopted by the GNOME project.

You can find more information about libsndfile in Chapter 5 or at the library's home page, `http://www.zip.com.au/%7Eerikd/libsndfile/`.

libaudiofile is available at `http://www.68k.org/%7Emichael/audiofile/`, but it is included in most Linux distributions. You'll probably have to download libsndfile yourself. It's worth the trouble.

Ogg Vorbis

Ogg Vorbis is a new audio compression scheme designed to compete with MP3 and the upcoming (stymied) SDMI format. Vorbis is patent-free, and support for it can easily be dropped into an application with the libvorbis library. Although Ogg Vorbis is still under development, the bitstream format is finalized (meaning that future versions of Vorbis will not break compatibility), and, at this writing, it already compresses audio data slighly better than MP3 (with further improvements expected soon). Let's hear a round of applause for the people behind the Ogg project!

We will use Ogg Vorbis to implement game music in Chapter 5. The Vorbis library is available for free download online at `http://www.vorbis.com`.

The SDL MPEG Library, SMPEG

The SDL MPEG library is a free MPEG-1 video and audio library with a heavy SDL slant. If you want to add MPEG-1 video or MP3 audio playback to your SDL-based game or application, SMPEG is an excellent choice. It may or may not be a viable solution for non-SDL programs, though (since SMPEG outputs directly to SDL surfaces).

MPEG-1 is popular compressed video format based on the discrete cosine transform and motion prediction. It is lossy (that is, it discards video data that it judges to be of less importance), but it generally produces good results, and it is commonly used for game cinematics. MPEG-2 is a newer video codec that produces higher-quality results at the expense of a lower compression ratio, but it is encumbered by patents and is therefore not supported by SMPEG.

The SMPEG library is available in the Development section of `http://www.lokigames.com`. Loki Software commercially maintains it for use in its games, but SMPEG is free software.

zlib

zlib (pronounced *zee-lib* or *zeta-lib*) is a general-purpose data compression library that implements the gzip format. It features an interface very similar to the stdio codefopen and codefwrite functions, and it is often used as a drop-in replacement for such. zlib is a good option when you need decent compression and don't want to code it yourself.

This library is very widely used, and there's a very good chance that it's already present on your Linux installation. You can download zlib's source code from `http://www.gzip.org`.

On to the Code!

Enough groundwork. It's time to throw around some code. In the next chapter we'll talk about the SDL library, a one-stop shop for portable graphics and audio. We'll also get started on Penguin Warrior, a complete Linux game that we'll develop over several chapters.