Simple DirectMedia Layer (SDL)

by Kyle Smith

Introduction

* SDL - Simple DirectMedia Layer
* From the website: "Simple DirectMedia Layer is a cross-platform multimedia library designed to provide low level access to audio, keyboard, mouse, joystick, 3D hardware via OpenGL, and 2D video framebuffer.“
* Acts as a medium between a number of I/O interfaces and the developer

Introduction 2

* Easy to imagine uses for this
  * Games
  * Media players
  * Multimedia chat
* Open Source
  * Originally developed by one person
  * Lots of third parties have contributed
    * Ports to different operating systems
    * Different plugins and libraries

History

* Sam Latinga, Creator
  * Employee at Loki Software from 1998-2001
  * Ported Popular games from Windows to Linux
    * Unreal Tournament
    * Civilization
    * Heroes III
  * Got the idea to create a cross-platform media interface
  * Released first version of SDL in 1998
  * Later became Lead Software Engineer for Blizzard Entertainment

History 2

* 2001 - Loki Software goes out of business
  * Latinga continues work on SDL with growing support for his project
  * 2008, Latinga leaves Blizzard to start his own company, Galaxy Gameworks
* SDL is still written and maintained largely by one man.
* Many popular apps and games have been written on the platform

Advantages

* Can write code once and compile it on most platforms!
  * SDL is a wrapper
  * Hides low-level hardware access from the developer
  * Provides an easy-to-use API to access sound, graphics, keyboard, etc.
  * Even officially supports Android and iOS mobile apps!

Advantages 2

* Great for small and independent developers
  * Free and open source
  * Cross-platform support reaches a larger market
  * Software distribution technology makes this easier
    * Steam
    * App Store
    * Google Play
  * No-cost, low-overhead software development platform

Advantages 3

* Doesn't require installing runtime libraries like Java or Adobe AIR
  * Easier for the end user
  * Less bloated can potentially mean better performance
  * But provides enough abstraction to make it easy on the developer

* Again, and most importantly: Free!

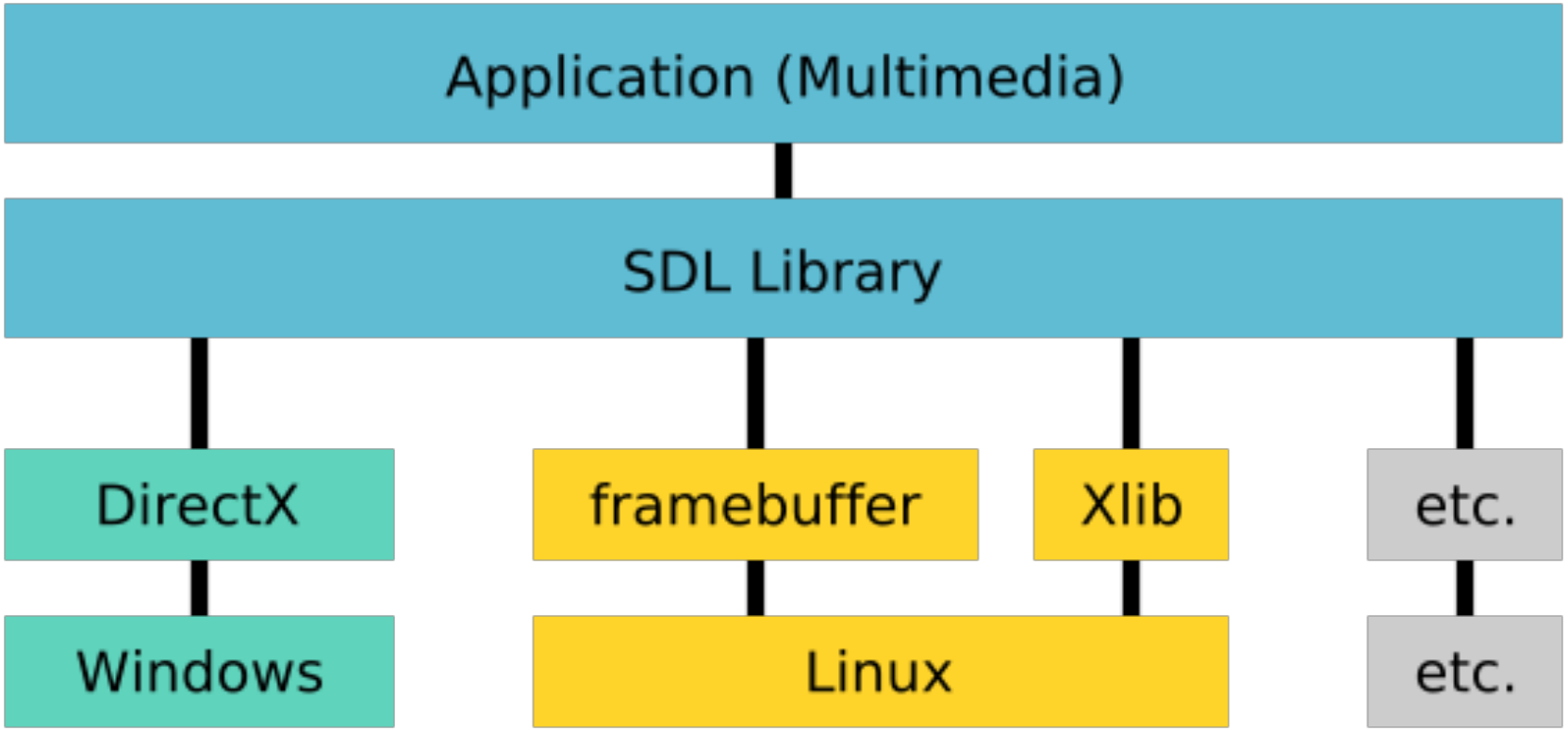
Disadvantages

* Not designed for enterprise software
  * Open source means no guarantees
  * Developed and maintained largely by one man
  * Run into a bug? Report it and hope it gets fixed.
* No cutting-edge technology
  * No DirectX 11.1 if you want cross-platform!
  * OpenGL shader support is limited

Disadvantages 2

* Hides the hardware level
  * Generally a good thing to reduce complexity
  * But you have to trust SDL on low-level optimizations and tweaks
  * If the behavior is not what you want, nothing you can do.
  * Can also be slower :(

SDL Layers of Abstraction



Features

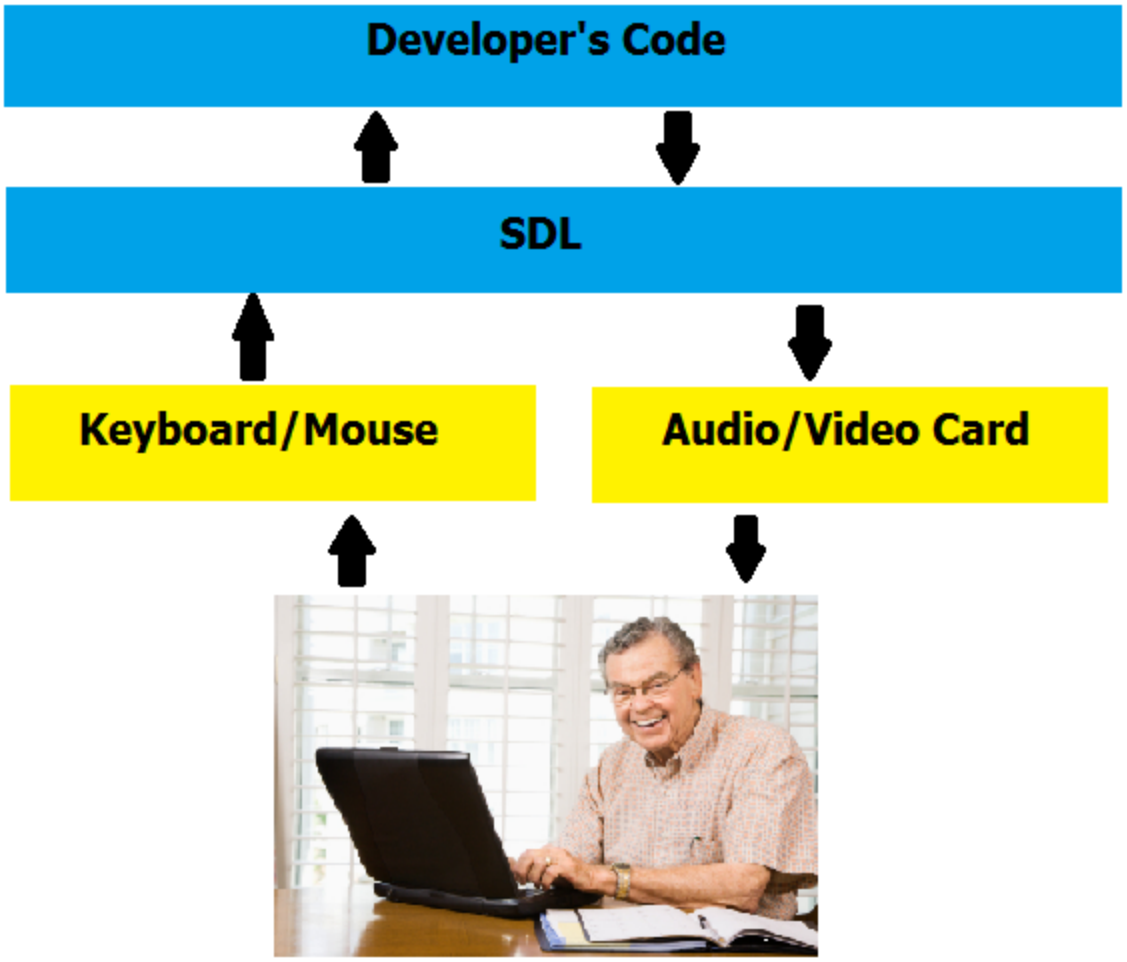
* Provides an API to access multiple different kinds of hardware
  * Keyboard/Mouse
  * Joystick
  * Audio
  * 3D graphics through OpenGL
  * 2D video framebuffer

Features 2

* Supports extensions!

* Several things are out of the scope of SDL, but can be plugged into it
  * Networking
  * Image libraries (PNG, JPG, etc.)
  * Sound libraries (MP3, OGG, etc.)
  * Lots of GUIs

SDL Hides the Hardware Layer



Example Program

* This is a program I created to demonstrate the capabilities of SDL.
* Utilizes keyboard and mouse input and produces 3D graphics via OpenGL.
* Some code samples taken from various tutorials on OpenGL and SDL.
* **(demo)**

What's Going On?

* There's a lot going on in the program, but SDL makes the implementation a lot simpler.
* Some SDL-specific initialization is required (setup_sdl())
* Keyboard and mouse inputs are handled in main_loop()
* Then, lots of OpenGL graphics code.

SDL Graphics Handling

```
static void setup_sdl() {
const SDL_VideoInfo* video;

 if ( SDL_Init(SDL_INIT_VIDEO) < 0 ) {
  fprintf(stderr, "Couldn't initialize SDL: %s\n", SDL_GetError());
  exit(1);
 }
atexit(SDL_Quit);
video = SDL_GetVideoInfo( );
 if( video == NULL ) {
  fprintf(stderr, "Couldn't get video information: %s\n", SDL_GetError());
  exit(1);
 }
SDL_GL_SetAttribute( SDL_GL_RED_SIZE, 5 );
 SDL_GL_SetAttribute( SDL_GL_GREEN_SIZE, 5 );
 SDL_GL_SetAttribute( SDL_GL_BLUE_SIZE, 5 );
 SDL_GL_SetAttribute( SDL_GL_DEPTH_SIZE, 16 );
 SDL_GL_SetAttribute( SDL_GL_DOUBLEBUFFER, 1 );
if( SDL_SetVideoMode( WIDTH, HEIGHT, video->vfmt->BitsPerPixel, SDL_OPENGL ) == 0 ) {
  fprintf(stderr, "Couldn't set video mode: %s\n", SDL_GetError());
  exit(1);
 } }
```

Tells SDL that we want to use its graphics engine

Obtains info about our graphics hardware

We want to use OpenGL to power our graphics!

SDL Graphics Handling 2

* That's all we need to do to set up OpenGL for use in SDL!
* Now, we have full access to the video card to do all the standard OpenGL commands.
* Shaders require more work (unfortunately)

* Now, how about input handling?

SDL Input Handling

```
SDL_Event event;  Uint8 *keystate;
 while (1) {
  keystate = SDL_GetKeyState(NULL);
  /* process pending events */
  while( SDL_PollEvent( &event ) ) {
   switch( event.type ) {
   case SDL_KEYDOWN:
    switch ( event.key.keysym.sym ) {
    case SDLK_ESCAPE:
     exit(0);
     break;
    default: break;
    case SDLK_g:
     land->garaud = !land->garaud;
     break;
    case SDLK_m:
     moveLight = !moveLight;
    }
    break;
   case SDL_QUIT:
    exit (0);
    break;
  } }
```

Loop forever, and check for user input.

If we receive an event (user input), check what it was.

Switch/case based on key input. SDL has built-in constants for each possible key (e.g. SDLK_g = the 'g' key).

Closing the window is also considered an 'event' by SDL. Exit the program if this happens.

SDL Input Handling 2

```
if (keystate[SDLK_RIGHT])
  camAngleH += 4;
if (keystate[SDLK_LEFT])
  camAngleH -= 4;
if (keystate[SDLK_UP])
  camAngleV += 4;
if (keystate[SDLK_DOWN])
  camAngleV -= 4;
if (keystate[SDLK_EQUALS])
  distance -= 0.3;
if (keystate[SDLK_MINUS])
  distance += 0.3;
if (distance < 0)
  distance = 0;
camAngleH = camAngleH % 360;
camAngleV = camAngleV % 360;
if (moveLight)
  lightAngle += 5;
repaint();
SDL_Delay( 50 );
```

These keys are outside the event handler. Why?

They are not switch/case, which means key presses are not mutually exclusive. This allows simultaneous input from any or all of these keys.

Tell OpenGL to redraw the image.

SDL also provides time functions, so we can create hardware-independent timing. This is good so we don't wear out our CPU running the main loop unnecessarily fast.

SDL Input Handling 3

* SDL provides its own syntax for accessing keyboard commands.
* Can either access input through the event handler or the SDL_GetKeyState.
  * Event handler is good for boolean switches (e.g. moving the light).
  * SDL_GetKeyState is good for continuous changes and/or handling simultaneous key changes (e.g. moving the camera).
  * The latter would be more commonly used for games and anything that requires real-time interaction.
* SDL_Delay is very useful for hardware-independent timing.

More about Graphics

* So what do we do now that we've initialized our graphics in SDL? What do we have to do to actually start drawing things?
* SDL hides the hardware layer from us, so any interaction with the graphics card must go through it.

More About Graphics 2

```
land = glGenLists(1);
glNewList(land,GL_COMPILE);
glColor3f(0.55f,0.27f,0.09f);
for (i=0;i<SIZE;i++)
 for (j=0;j<SIZE;j++)
 {
  glBegin(GL_QUADS);
  glNormal3f(landarray[i][j]-landarray[i+1][j],landarray[i][j]-landarray[i][j+1],1.f);
  glVertex3f((GLfloat)i-SIZE/2,(GLfloat)j-SIZE/2,landarray[i][j]);
  glVertex3f((GLfloat)i-SIZE/2+1,(GLfloat)j-SIZE/2,landarray[i+1][j]);
  glVertex3f((GLfloat)i-SIZE/2+1,(GLfloat)j-SIZE/2+1,landarray[i+1][j+1]);
  glVertex3f((GLfloat)i-SIZE/2,(GLfloat)j-SIZE/2+1,landarray[i][j+1]);
  glEnd();
 }
glEndList();
```

Generate an OpenGL display list for quick access later. These are stored on the graphics card and can be drawn very quickly when called.

Set a bunch of vertices and normal vectors to generate a terrain.

Note: there is a lot of graphics code in this program, so this is just a small sample. Most of the code looks similar to this.

More About Graphics 3

* So what?

* There are no calls to SDL functions in that code!
* Can plug that code into GLUT, Qt, or any other interface that supports OpenGL and it will compile.
* Once OpenGL is initialized, SDL steps out of the way and lets you take over.

* Fortunately, most graphics interfaces behave similarly.

SDL on Other Platforms

* SDL was designed with the goal of cross-platform programming in mind.
* However, that doesn't mean "favoritism" hasn't emerged.
* SDL supports DirectX (sort of)
  * Great for Windows users!
  * But using DirectX completely kills all cross-platform capability.
* Even though mobile platforms are officially supported, the usability is rather limited.
  * SDL 1.3 (yet to be released) is hoped to improve mobile support

SDL on Other Platforms 2

* Cross-platform programming is a great goal, but it comes at a cost.
  * Functionality on more platforms = lower possible complexity in applications.
  * More advanced applications = fewer platforms can support it.
* Truly cross-platform applications:
  * Calculator? Solitaire? Emacs?
* Partially cross-platform applications:
  * OpenGL games
  * OpenOffice
  * Android and iOS support OpenGL, but can they run games designed for desktop computers?

SDL on Other Platforms 3

* Very few applications need to be run on literally *any* platform.
  * Every application has its niche.
  * We don't need OpenOffice on our phones.
  * And we don't need SMS on our desktops.
* The beauty of platforms like SDL is not the ability to publish an app on every platform.
  * Someone can learn SDL and publish an app anywhere with very small learning curve for each platform.
  * Publish a OSX/Linux app today…
  * Publish a iOS/Android app tomorrow.

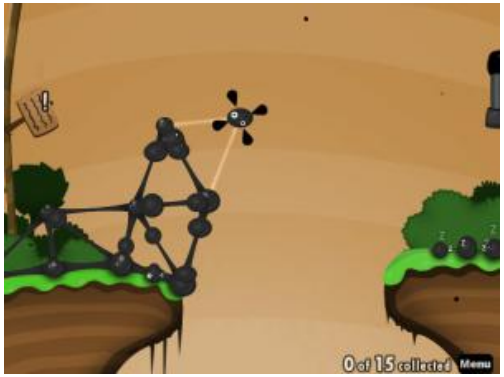
What About Extensions?

* SDL maintains a database of extension libraries on its website.
* Some of them provide access to hardware not natively supported by SDL
  * Microphone
  * Webcam
  * Network card
* Some provide more abstraction
  * Game development engines
  * Collision detection
  * “How to program” tutorials
* Font libraries, GUIs, sound engines, etc.

Summary

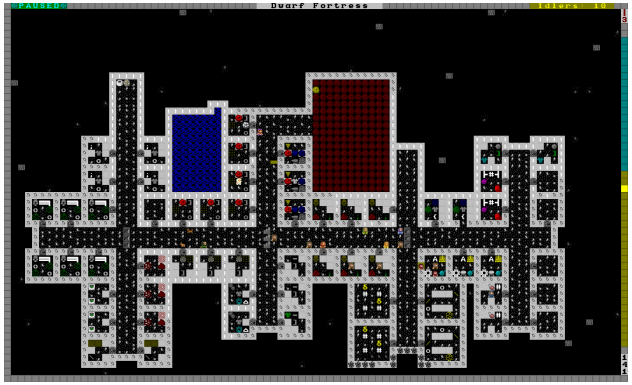
* Even though SDL is a relatively small project, has a lot of potential power.
* A clever developer could build a career using only SDL.
* Given the variety of platforms and the number of extensions, an application could be written for almost any purpose.
* Not designed for “beginner” programmers, but designed with simplicity and ease of use in mind.

Lastly…



World of Goo

Neverwinter Nights



Dwarf Fortress

All made with SDL!