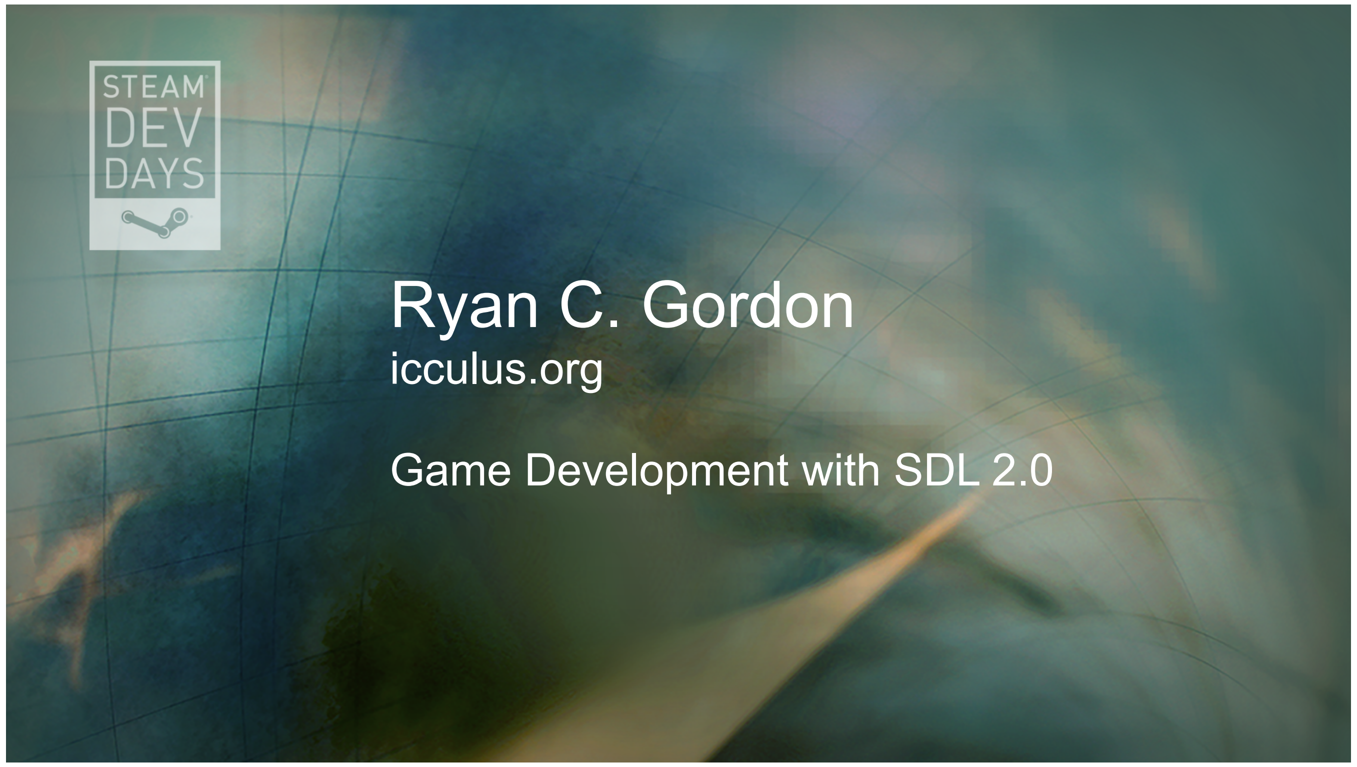
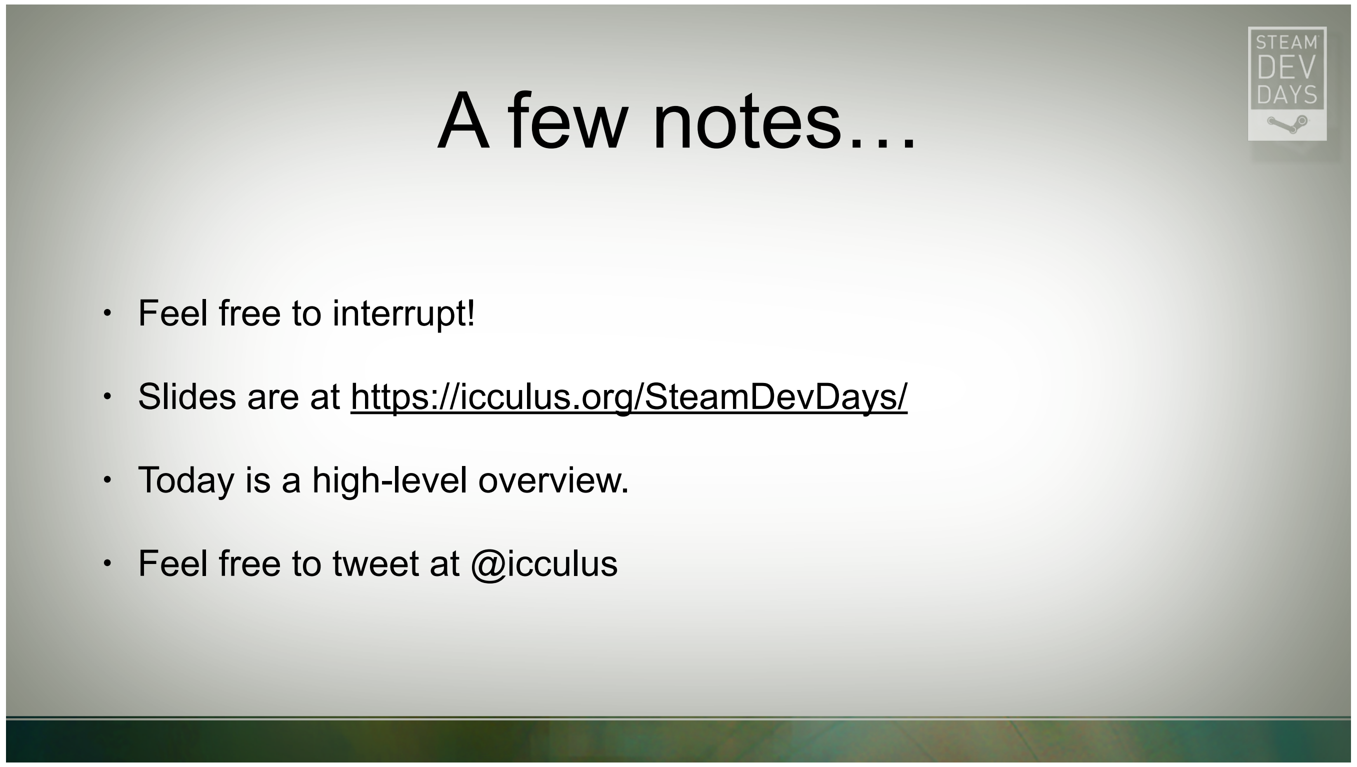


Game Development with SDL 2.0

Ryan C. Gordon
icculus.org

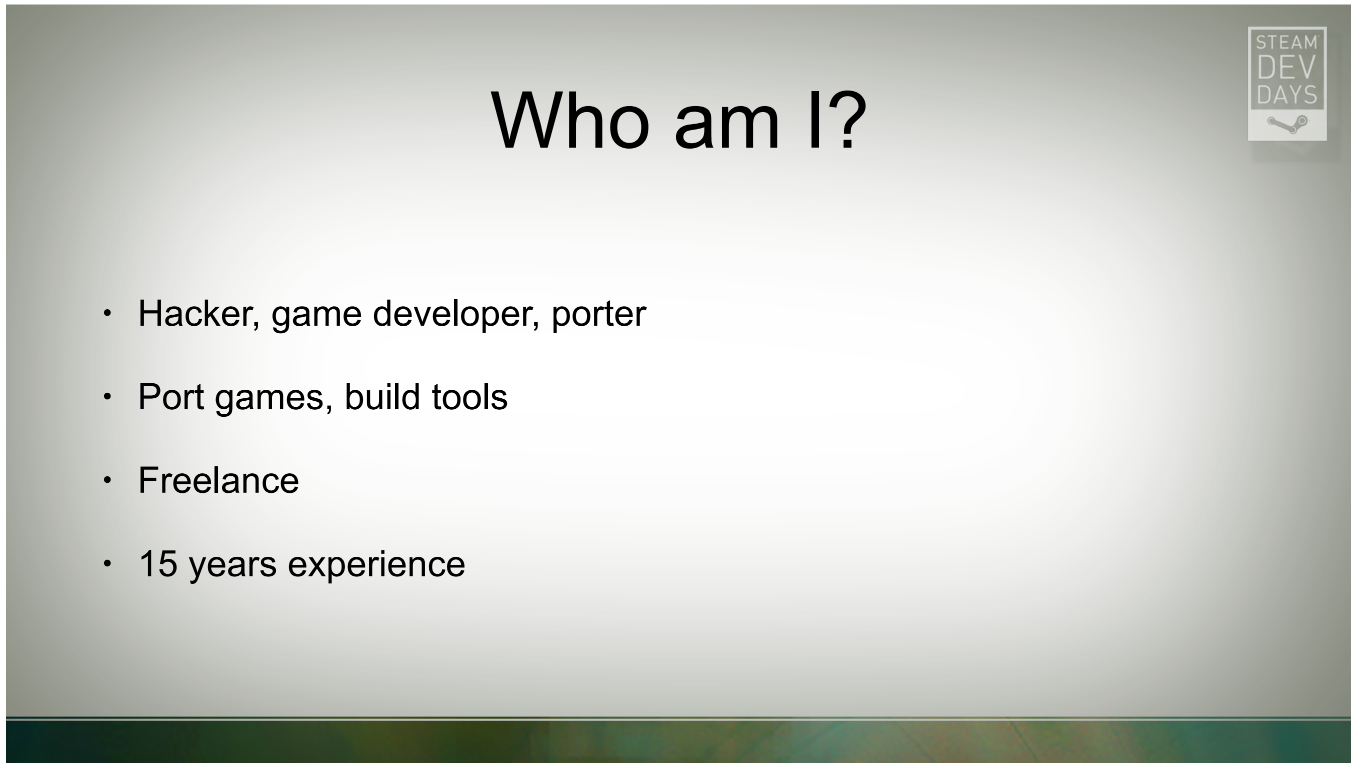
Are you in the right room?



A few notes…

- Feel free to interrupt!
- Slides are at https://icculus.org/SteamDevDays/
- Today is a high-level overview.
- Feel free to tweet at @icculus

I could talk about the nitty-gritty of this for _hours_, but this talk won't have many code samples, etc. Mostly I want you to walk out of here knowing what you can do with SDL2, and do further research in our excellent documentation and friendly community.



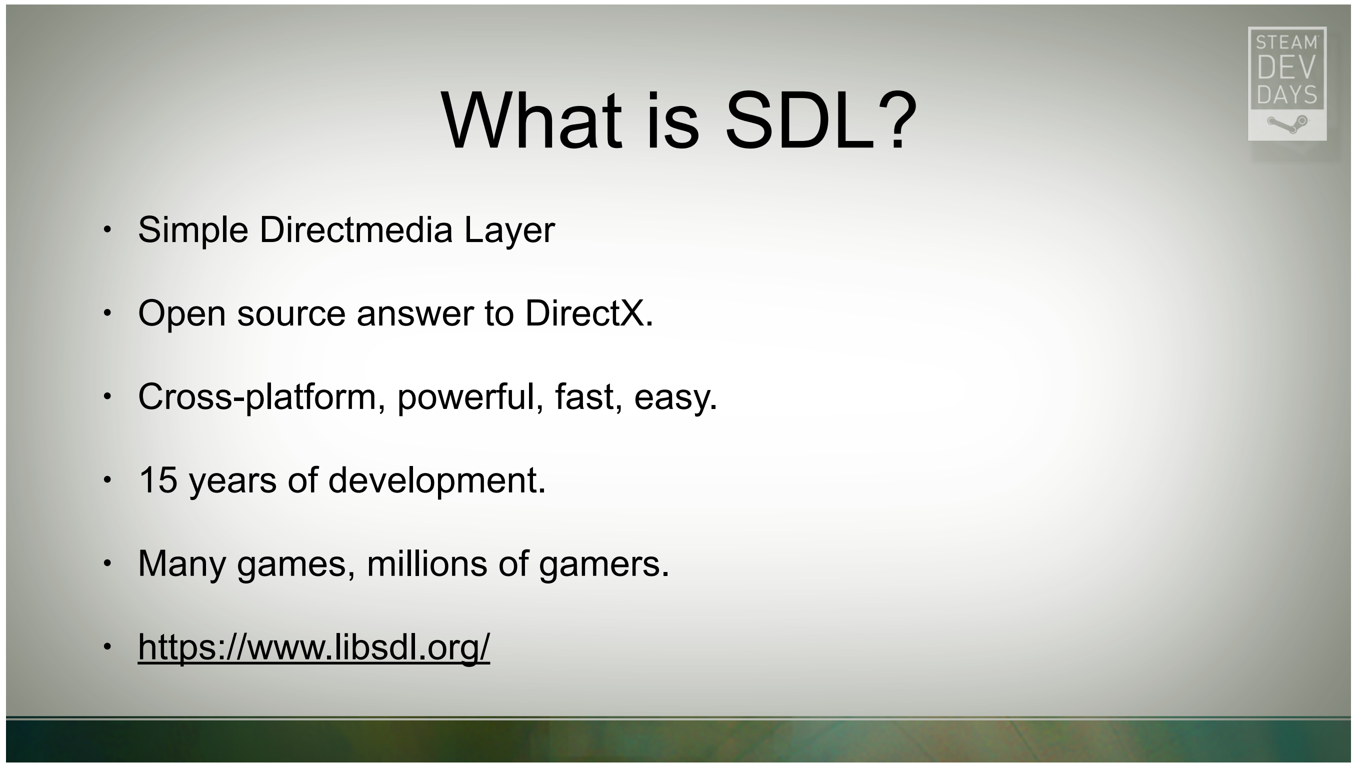
Who am I?

- Hacker, game developer, porter
- Port games, build tools
- Freelance
- 15 years experience

And I can be bought! Drop me an email at icculus@icculus.org if you need games ported, bugs fixed, problems solved.

This is my Nascar jacket.

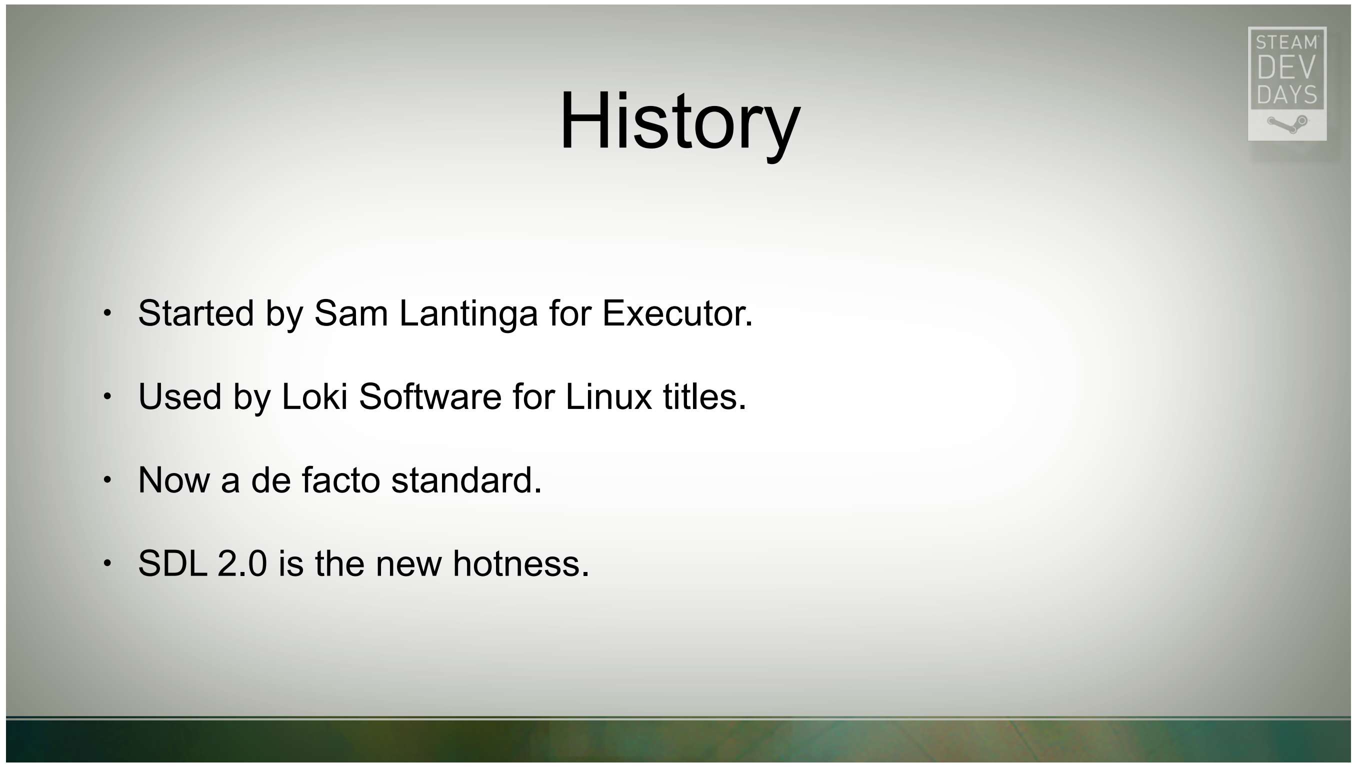
My most demanding customer.



What is SDL?

- Simple Directmedia Layer
- Open source answer to DirectX.
- Cross-platform, powerful, fast, easy.
- 15 years of development.
- Many games, millions of gamers.
- https://www.libsdl.org/

SDL has powered most triple-A game engines, at least for Mac or Linux ports, if not Windows, too.
UnrealEngine 1 through 3, idTech 1 through 4, Unity3D, Source, and many others, have shipped SDL-based builds.
Valve is using it for all of their games, as well as the Steam Client.
Almost every Linux game uses it.

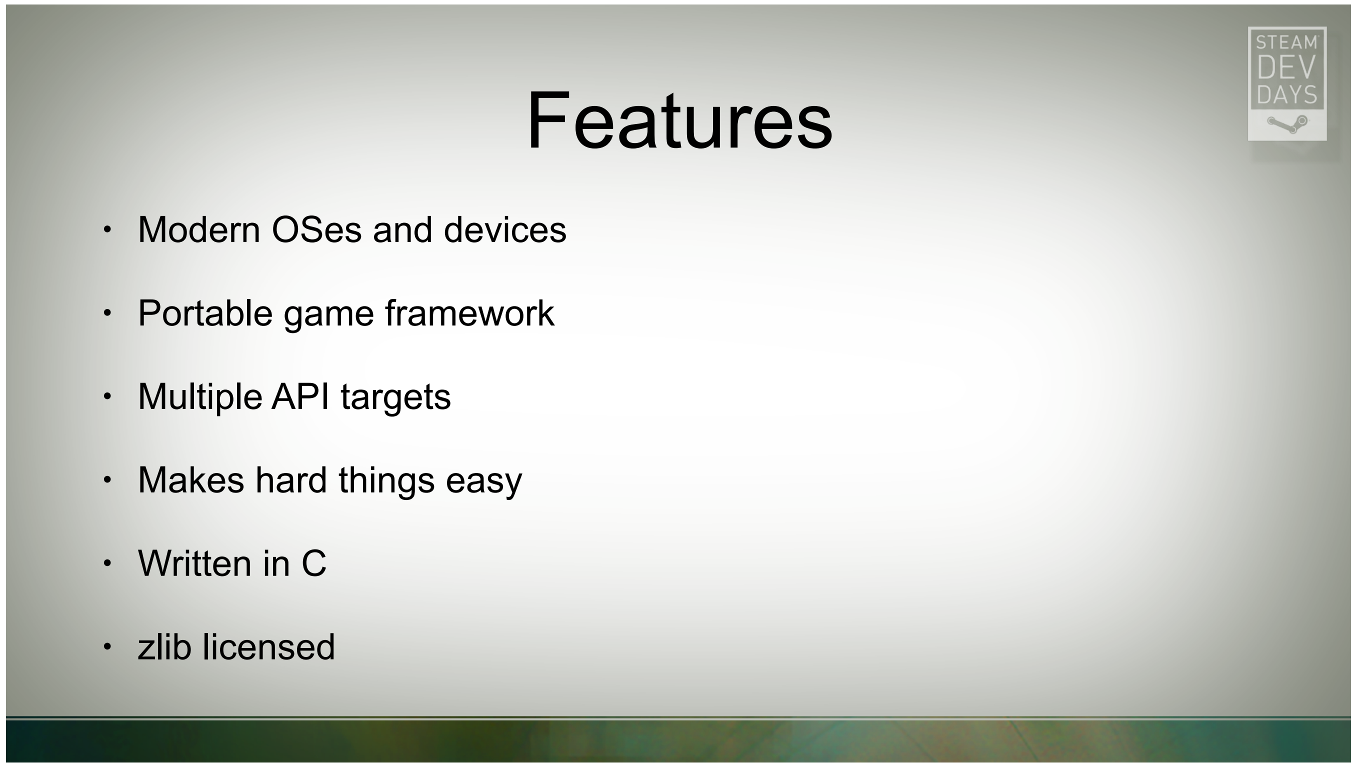


History

- Started by Sam Lantinga for Executor.
- Used by Loki Software for Linux titles.
- Now a de facto standard.
- SDL 2.0 is the new hotness.

Sam started working on SDL around 1998, for a Mac Classic emulator named Executor. Loki hired Sam and based almost all their game ports on it, using it to replace DirectX code, and had several engineers improve it, not to mention contributions from the open source community.

For many years, we improved SDL 1.2, but the past few years have been dedicated to developing SDL 2.0, which removes many of the limitations of 1.2 and adds lots of great new features.

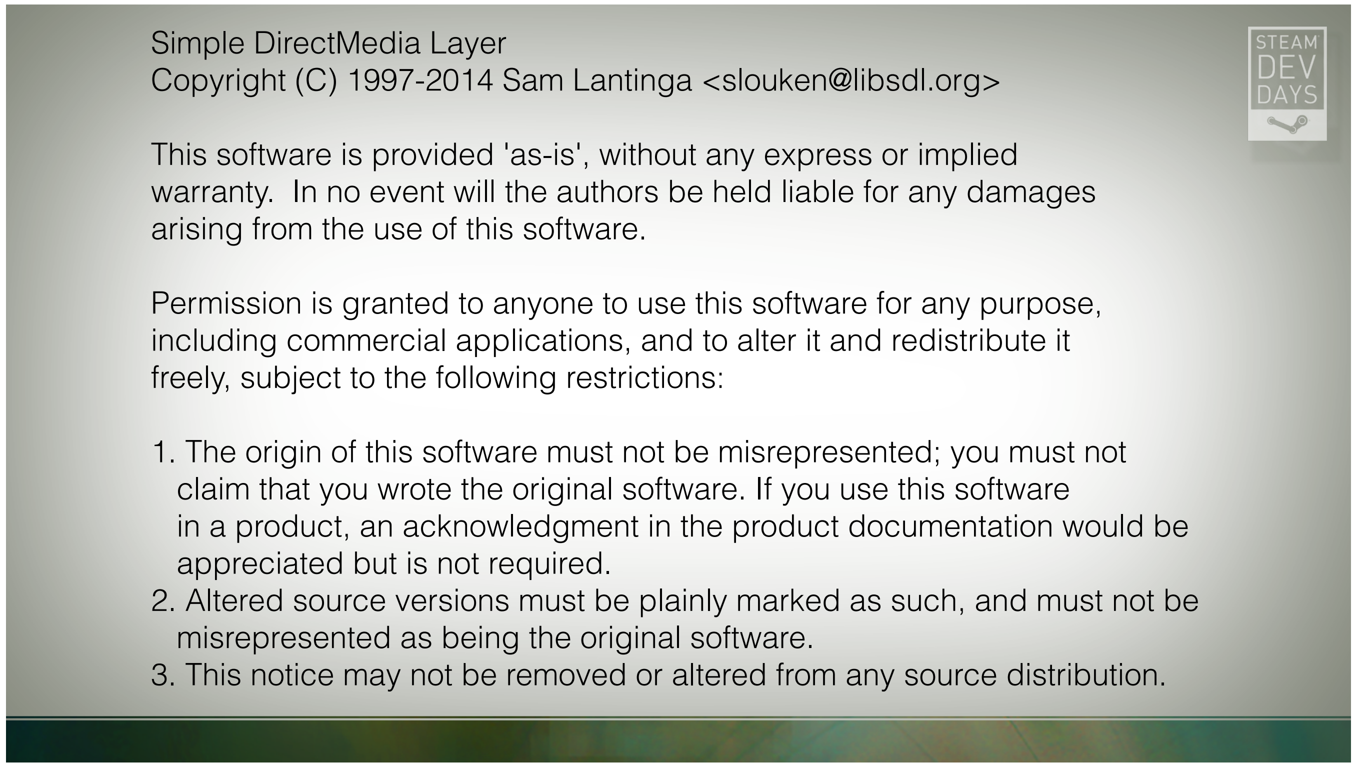


Features

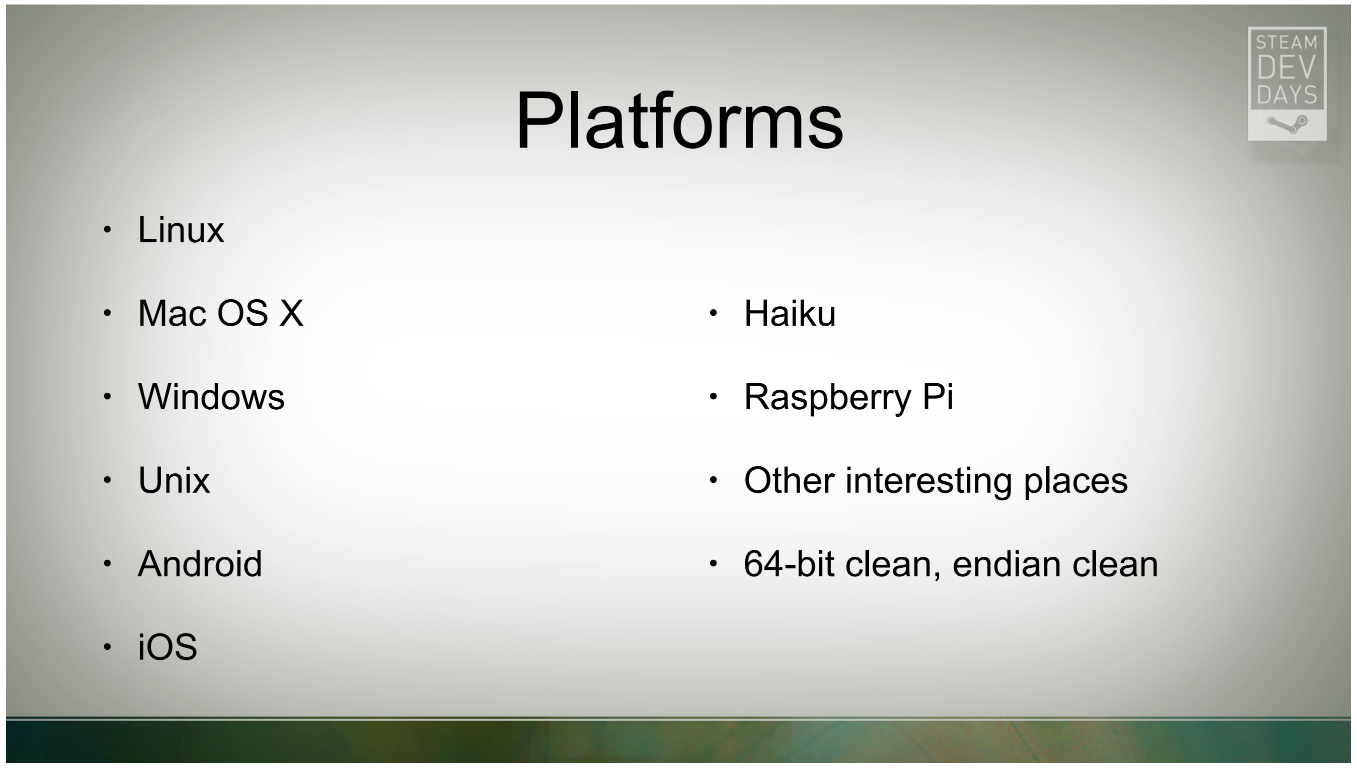
- Modern OSes and devices
- Portable game framework
- Multiple API targets
- Makes hard things easy
- Written in C
- zlib licensed

We wrote this in C for several reasons: first, we like C. But also it helped portability in times where there were many different compilers with varying degrees of quality. Also, C++ binary compatibility was sketchy on Linux at the time, and C code is easy to call from just about anywhere: scripting language bindings, dlsym() lookups, and of course, C++.

We switched to the zlib license to make things easier for everyone, and access platforms that are hostile to the LGPL.



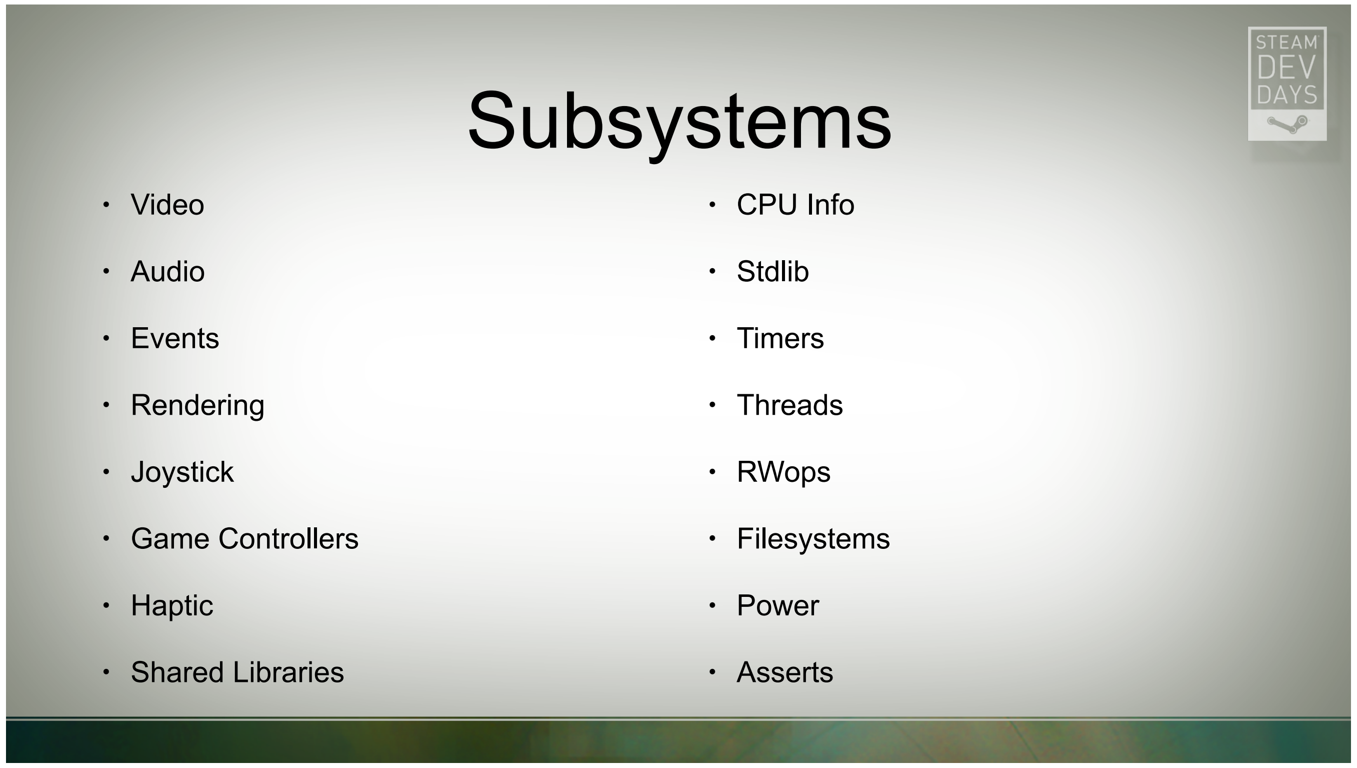
SDL 1.2 used the GNU LGPL 2.1 license. I was going to put it in this presentation, but it took 22 slides to fit it all. We moved away from it for SDL 2.0; Sam and a few others had written the bulk of the original code, and for the rest, we got permission to relicense from contributors, and replaced and removed a lot of code, in order to have a clean codebase.



Platforms

- Linux
- Mac OS X
- Windows
- Unix
- Android
- iOS
- Haiku
- Raspberry Pi
- Other interesting places
- 64-bit clean, endian clean

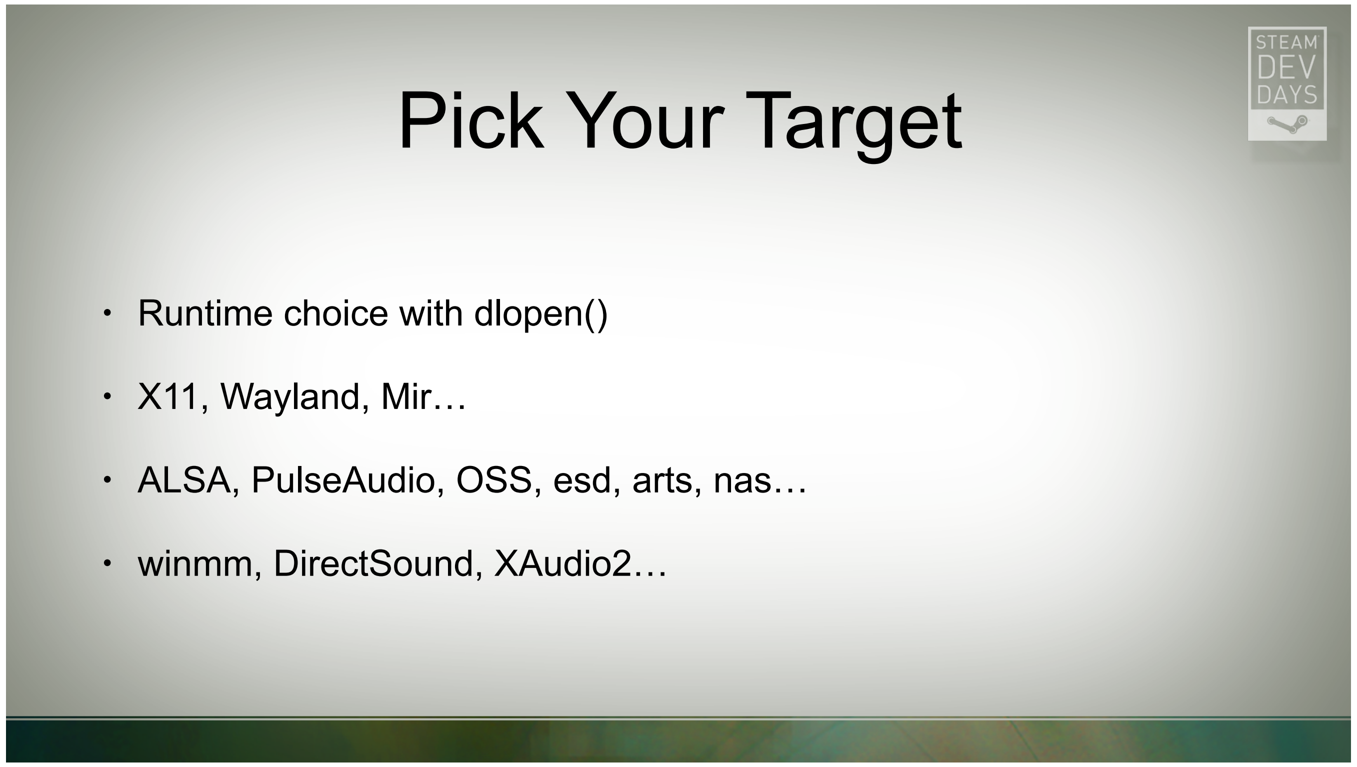
SDL 2 has, at times, also worked on the Sony PSP and Nintendo DS. If you have a platform you want supported, send patches!



Subsystems

- Video
- Audio
- Events
- Rendering
- Joystick
- Game Controllers
- Haptic
- Shared Libraries
- CPU Info
- Stdlib
- Timers
- Threads
- RWops
- Filesystems
- Power
- Asserts

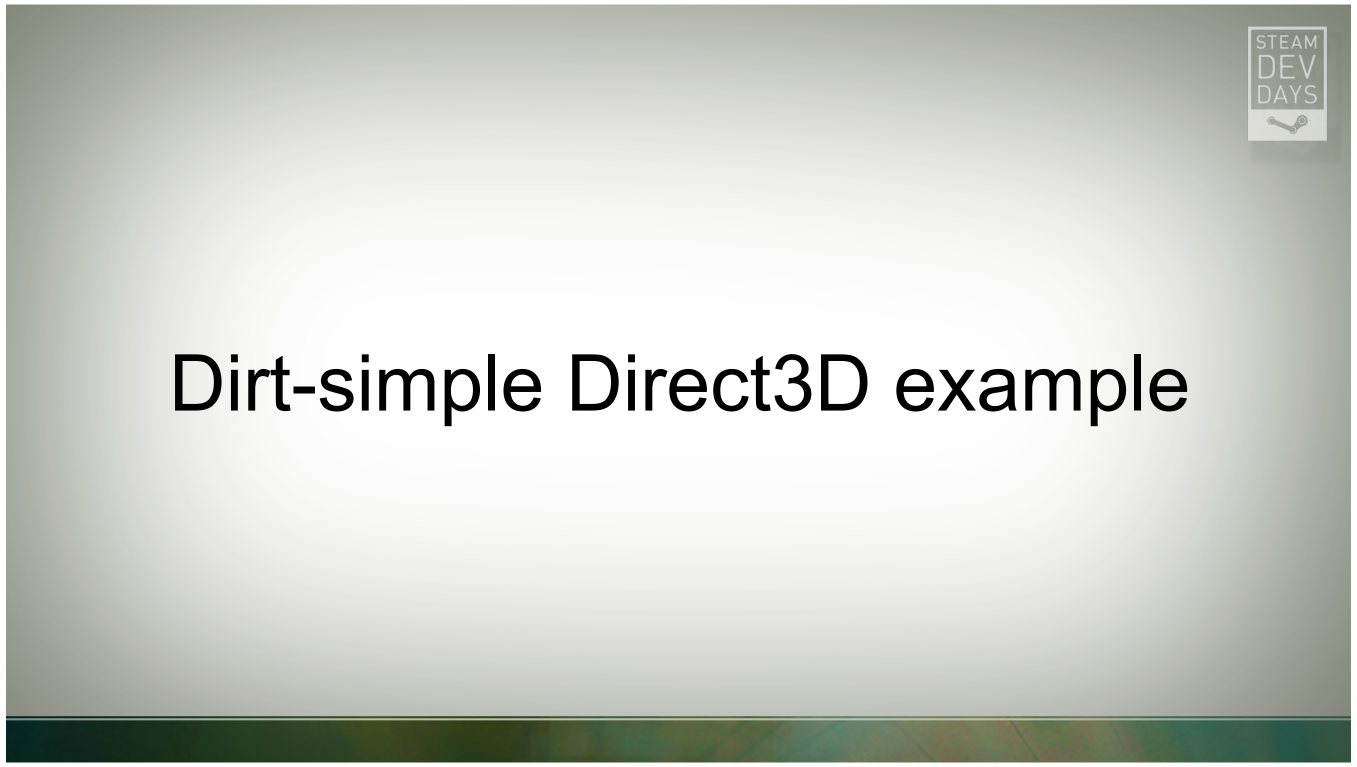
For SDL 1.2, this slide had 8 bullet points. Now there’s 16, which seems right when moving from 1.x to 2.0.



Pick Your Target

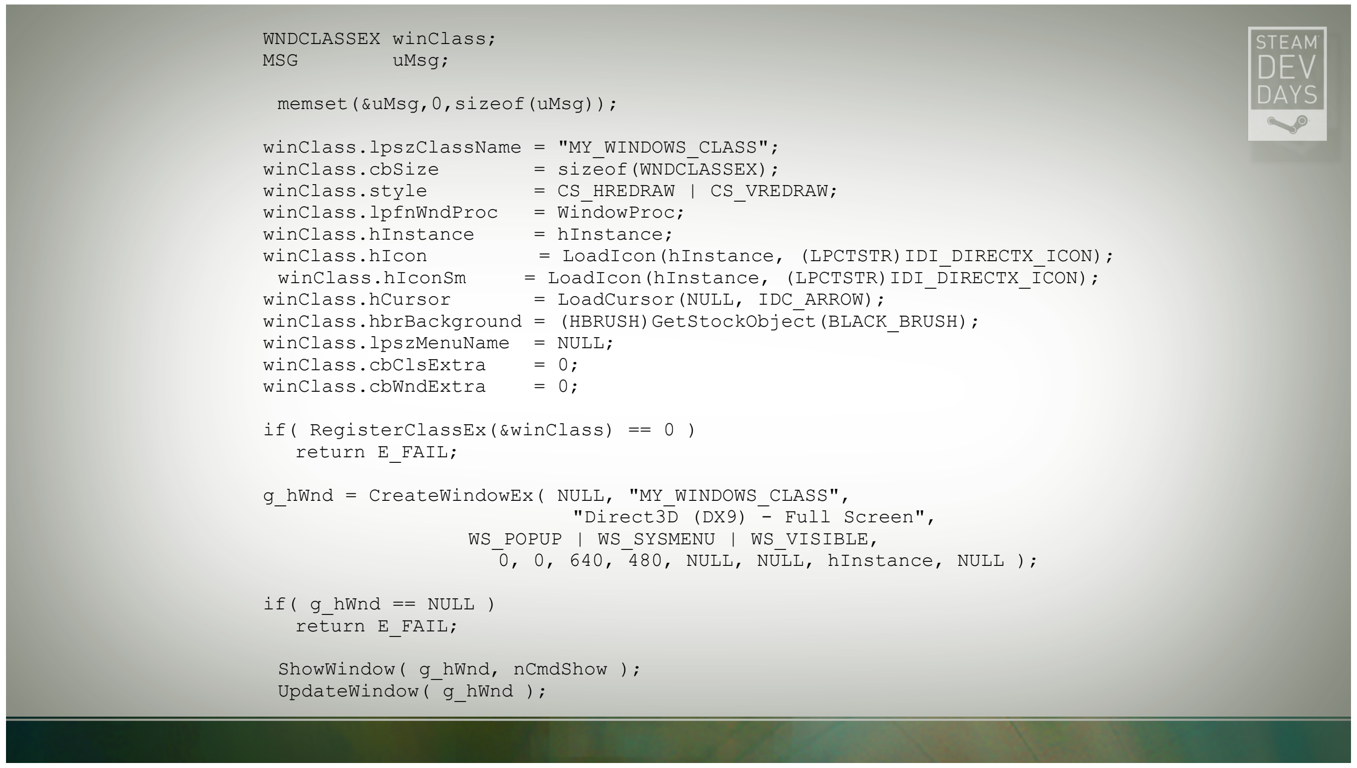
- Runtime choice with dlopen()
- X11, Wayland, Mir…
- ALSA, PulseAudio, OSS, esd, arts, nas…
- winmm, DirectSound, XAudio2…

SDL links against nothing but the C runtime (on Windows, it doesn't even link to that). Everything else is detected and chosen at runtime, based on what is available on a given user's system.



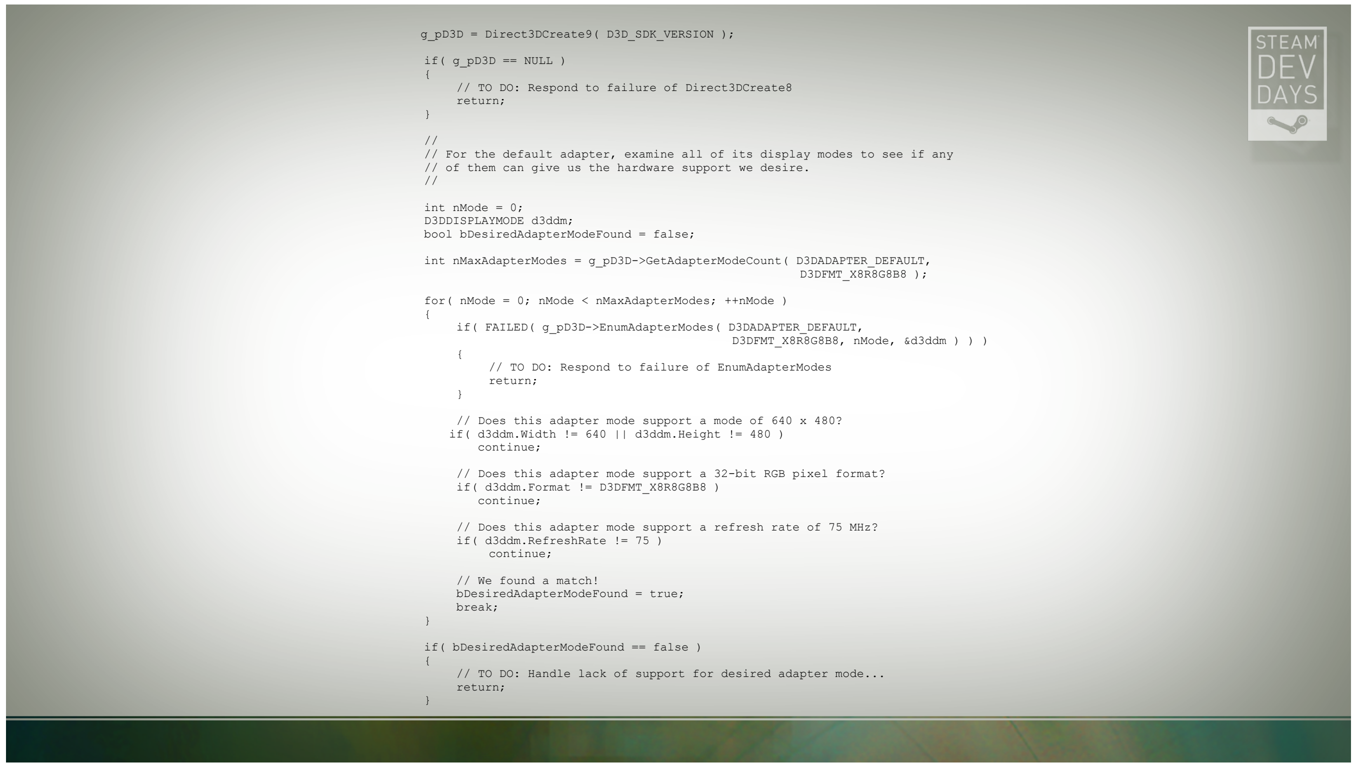
Dirt-simple Direct3D example

Sarcasm!



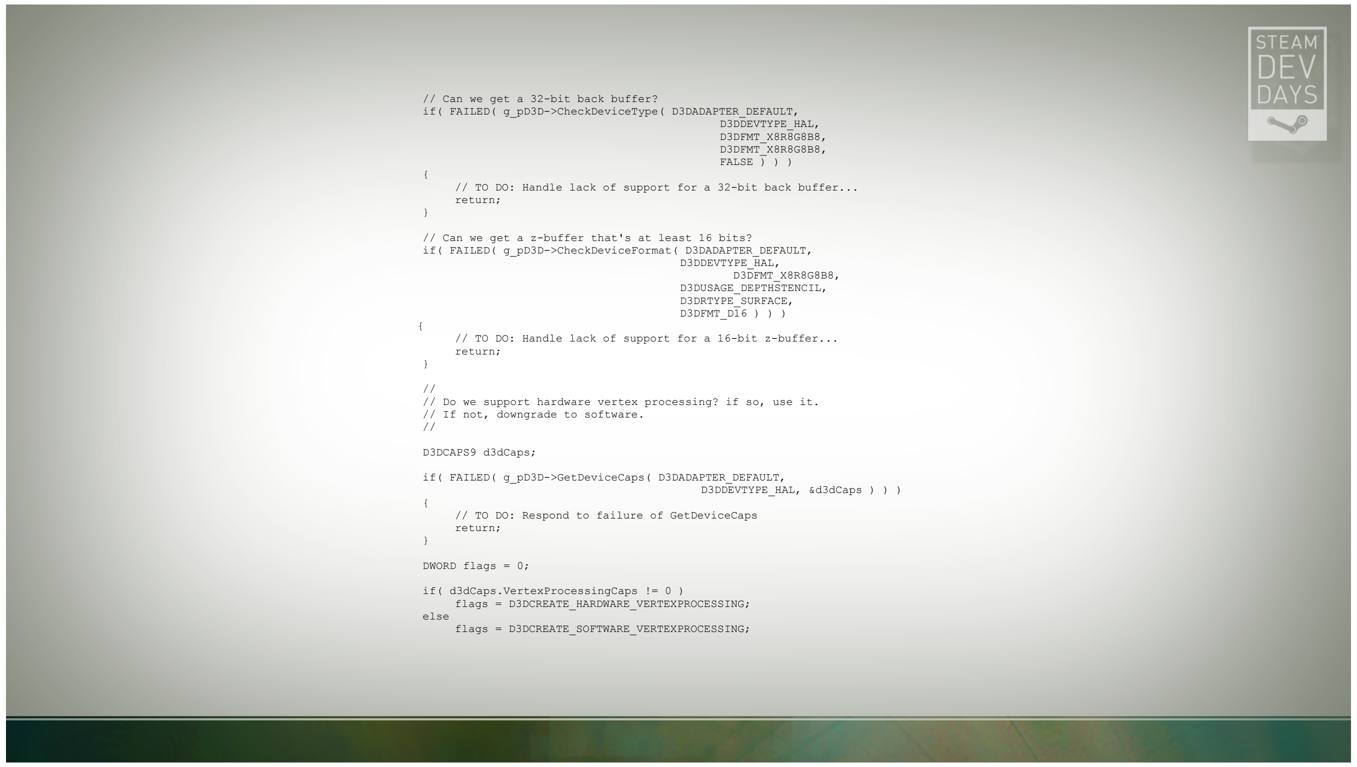
```
WNDCLASSEX winClass;
MSG        uMsg;

 memset(&uMsg,0,sizeof(uMsg));

winClass.lpszClassName = "MY_WINDOWS_CLASS";
winClass.cbSize        = sizeof(WNDCLASSEX);
winClass.style         = CS_HREDRAW | CS_VREDRAW;
winClass.lpfnWndProc   = WindowProc;
winClass.hInstance     = hInstance;
winClass.hIcon          = LoadIcon(hInstance, (LPCTSTR)IDI_DIRECTX_ICON);
 winClass.hIconSm     = LoadIcon(hInstance, (LPCTSTR)IDI_DIRECTX_ICON);
winClass.hCursor       = LoadCursor(NULL, IDC_ARROW);
winClass.hbrBackground = (HBRUSH)GetStockObject(BLACK_BRUSH);
winClass.lpszMenuName  = NULL;
winClass.cbClsExtra    = 0;
winClass.cbWndExtra    = 0;

if( RegisterClassEx(&winClass) == 0 )
   return E_FAIL;

g_hWnd = CreateWindowEx( NULL, "MY_WINDOWS_CLASS",
                          "Direct3D (DX9) - Full Screen",
                 WS_POPUP | WS_SYSMENU | WS_VISIBLE,
                    0, 0, 640, 480, NULL, NULL, hInstance, NULL );

if( g_hWnd == NULL )
   return E_FAIL;

 ShowWindow( g_hWnd, nCmdShow );
 UpdateWindow( g_hWnd );
```

This isn't a pathological case. More or less, this is what the example code in the Direct3D documentation looks like.



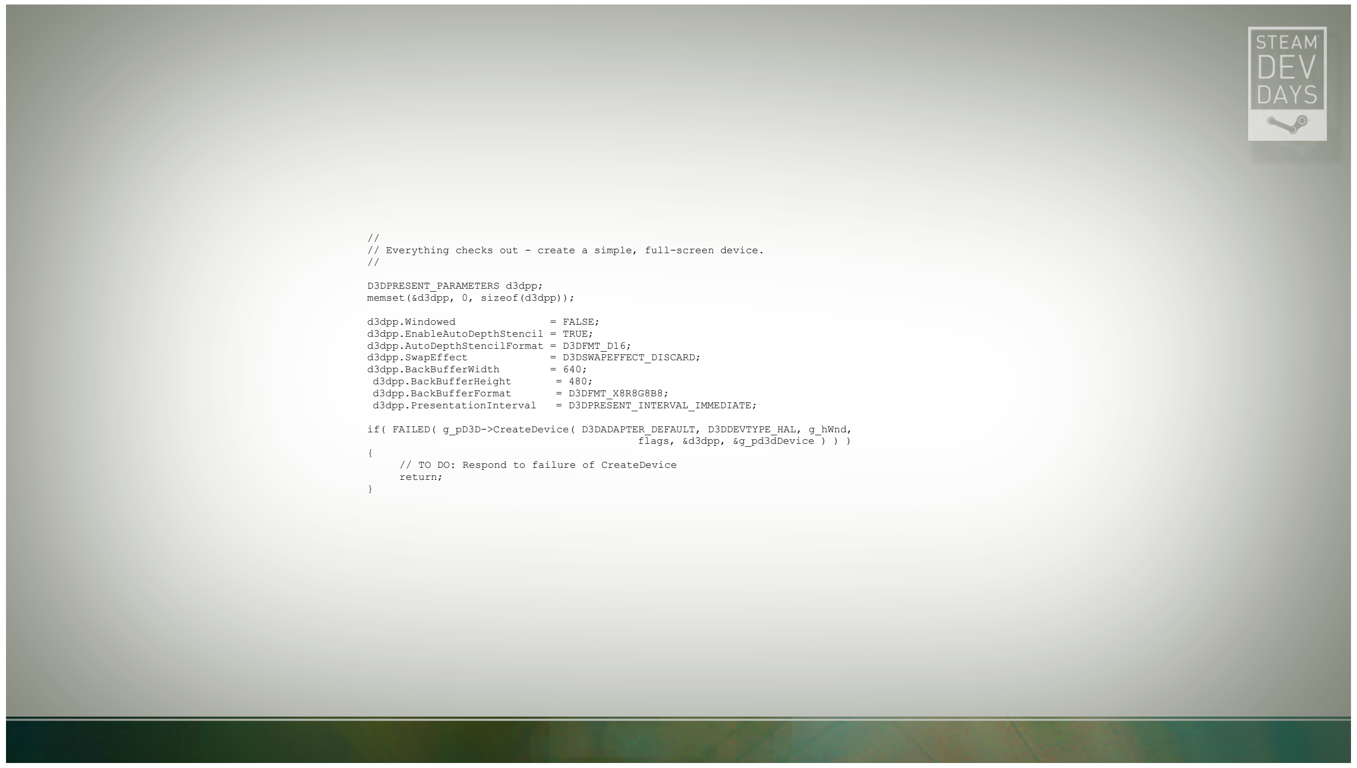
```
g_pD3D = Direct3DCreate9( D3D_SDK_VERSION );

if( g_pD3D == NULL )
{
    // TO DO: Respond to failure of Direct3DCreate8
    return;
}

//
// For the default adapter, examine all of its display modes to see if any
// of them can give us the hardware support we desire.
//

int nMode = 0;
D3DDISPLAYMODE d3ddm;
bool bDesiredAdapterModeFound = false;

int nMaxAdapterModes = g_pD3D->GetAdapterModeCount( D3DADAPTER_DEFAULT,
                                                    D3DFMT_X8R8G8B8 );

for( nMode = 0; nMode < nMaxAdapterModes; ++nMode )
{
    if( FAILED( g_pD3D->EnumAdapterModes( D3DADAPTER_DEFAULT,
                                          D3DFMT_X8R8G8B8, nMode, &d3ddm ) ) )
    {
        // TO DO: Respond to failure of EnumAdapterModes
        return;
    }

    // Does this adapter mode support a mode of 640 x 480?
   if( d3ddm.Width != 640 || d3ddm.Height != 480 )
      continue;

    // Does this adapter mode support a 32-bit RGB pixel format?
    if( d3ddm.Format != D3DFMT_X8R8G8B8 )
      continue;

    // Does this adapter mode support a refresh rate of 75 MHz?
    if( d3ddm.RefreshRate != 75 )
        continue;

    // We found a match!
    bDesiredAdapterModeFound = true;
    break;
}

if( bDesiredAdapterModeFound == false )
{
    // TO DO: Handle lack of support for desired adapter mode...
    return;
}
```

If you can read this, you're sitting too close to the screen.



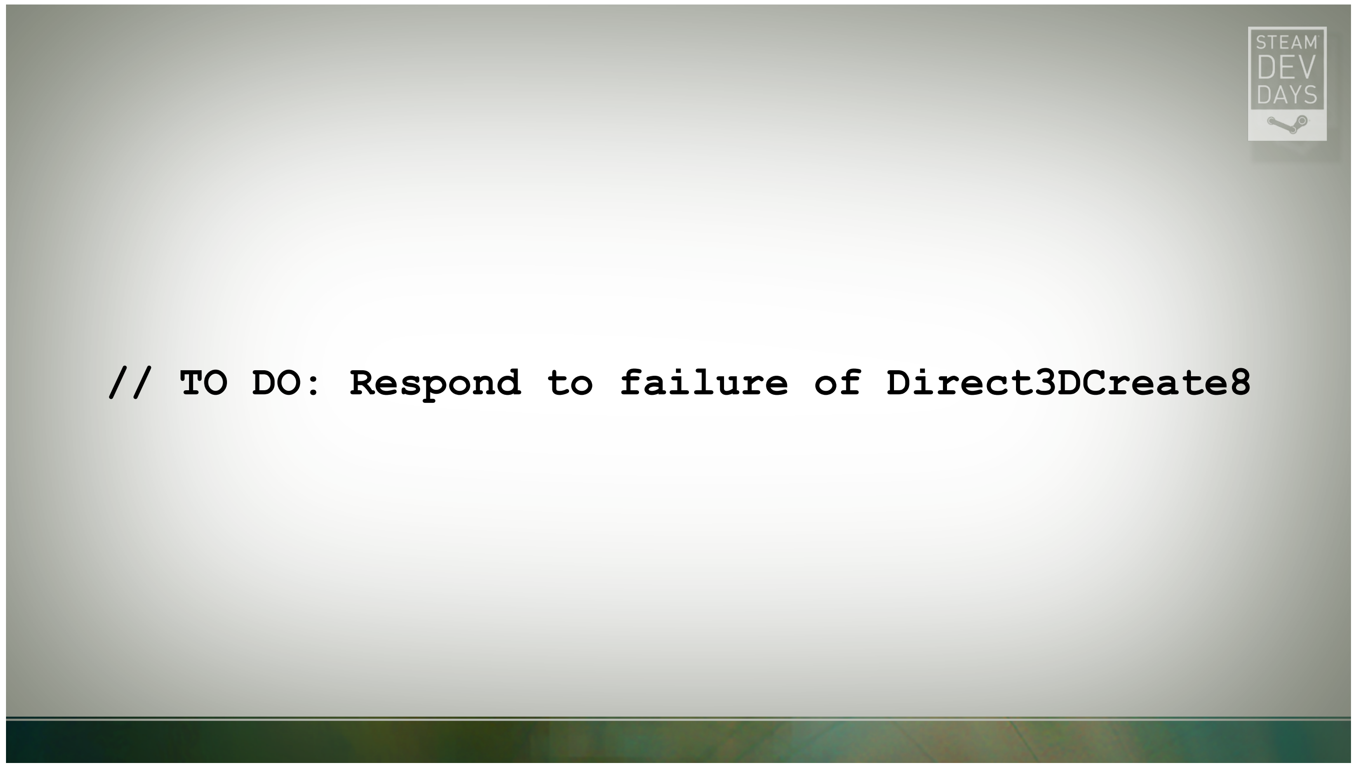
```
// Can we get a 32-bit back buffer?
if( FAILED( g_pD3D->CheckDeviceType( D3DADAPTER_DEFAULT,
                                     D3DDEVTYPE_HAL,
                                     D3DFMT_X8R8G8B8,
                                     D3DFMT_X8R8G8B8,
                                     FALSE ) ) )
{
     // TO DO: Handle lack of support for a 32-bit back buffer...
     return;
}

// Can we get a z-buffer that's at least 16 bits?
if( FAILED( g_pD3D->CheckDeviceFormat( D3DADAPTER_DEFAULT,
                                       D3DDEVTYPE_HAL,
                                               D3DFMT_X8R8G8B8,
                                       D3DUSAGE_DEPTHSTENCIL,
                                       D3DRTYPE_SURFACE,
                                       D3DFMT_D16 ) ) )
{
     // TO DO: Handle lack of support for a 16-bit z-buffer...
     return;
}

//
// Do we support hardware vertex processing? if so, use it.
// If not, downgrade to software.
//

D3DCAPS9 d3dCaps;

if( FAILED( g_pD3D->GetDeviceCaps( D3DADAPTER_DEFAULT,
                                      D3DDEVTYPE_HAL, &d3dCaps ) ) )
{
     // TO DO: Respond to failure of GetDeviceCaps
     return;
}

DWORD flags = 0;

if( d3dCaps.VertexProcessingCaps != 0 )
     flags = D3DCREATE_HARDWARE_VERTEXPROCESSING;
else
     flags = D3DCREATE_SOFTWARE_VERTEXPROCESSING;
```

At this point, we’re still deciding if the system supports 3D acceleration at all.



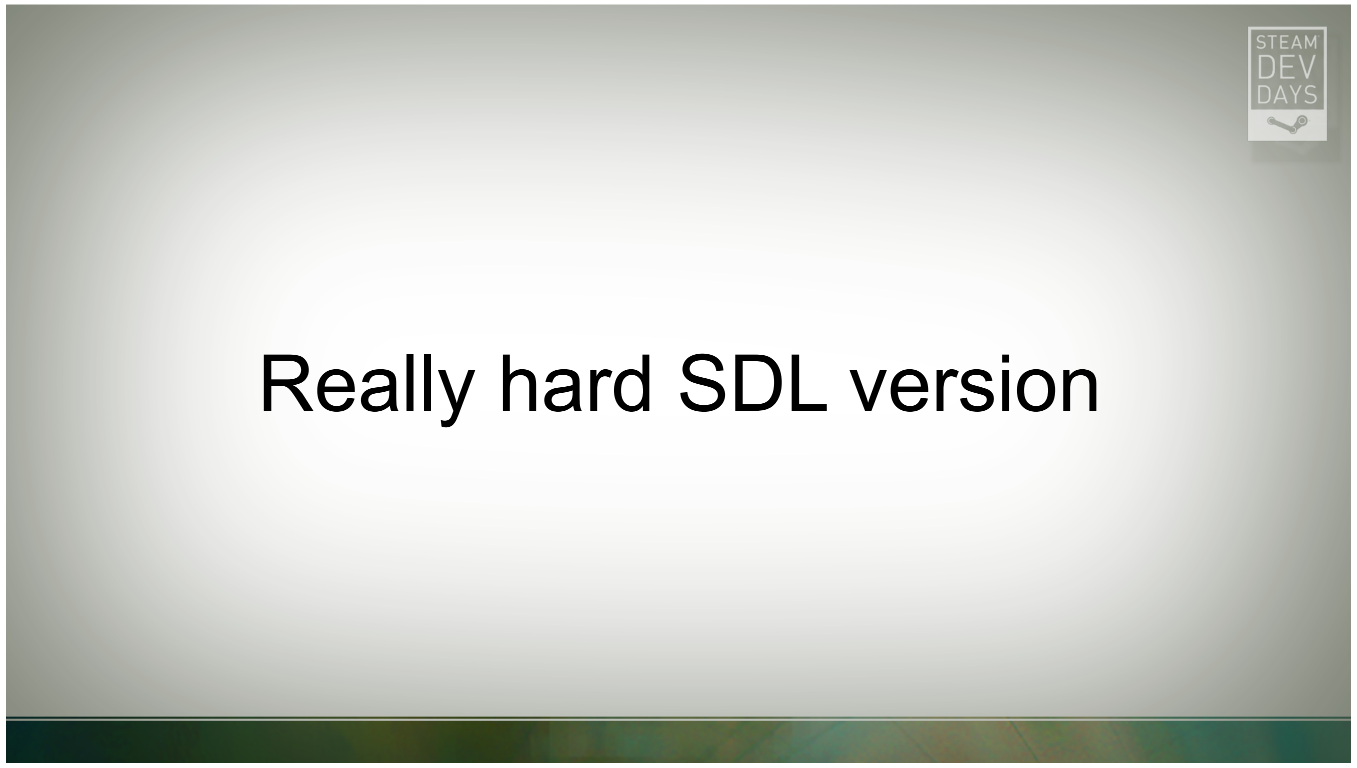
```
//
// Everything checks out - create a simple, full-screen device.
//

D3DPRESENT_PARAMETERS d3dpp;
memset(&d3dpp, 0, sizeof(d3dpp));

d3dpp.Windowed               = FALSE;
d3dpp.EnableAutoDepthStencil = TRUE;
d3dpp.AutoDepthStencilFormat = D3DFMT_D16;
d3dpp.SwapEffect             = D3DSWAPEFFECT_DISCARD;
d3dpp.BackBufferWidth        = 640;
 d3dpp.BackBufferHeight       = 480;
 d3dpp.BackBufferFormat       = D3DFMT_X8R8G8B8;
 d3dpp.PresentationInterval   = D3DPRESENT_INTERVAL_IMMEDIATE;

if( FAILED( g_pD3D->CreateDevice( D3DADAPTER_DEFAULT, D3DDEVTYPE_HAL, g_hWnd,
                                          flags, &d3dpp, &g_pd3dDevice ) ) )
{
     // TO DO: Respond to failure of CreateDevice
     return;
}
```

What cracks me up about this is the “create a simple, full-screen device” comment. Oh, I see, this was the SIMPLE way!!



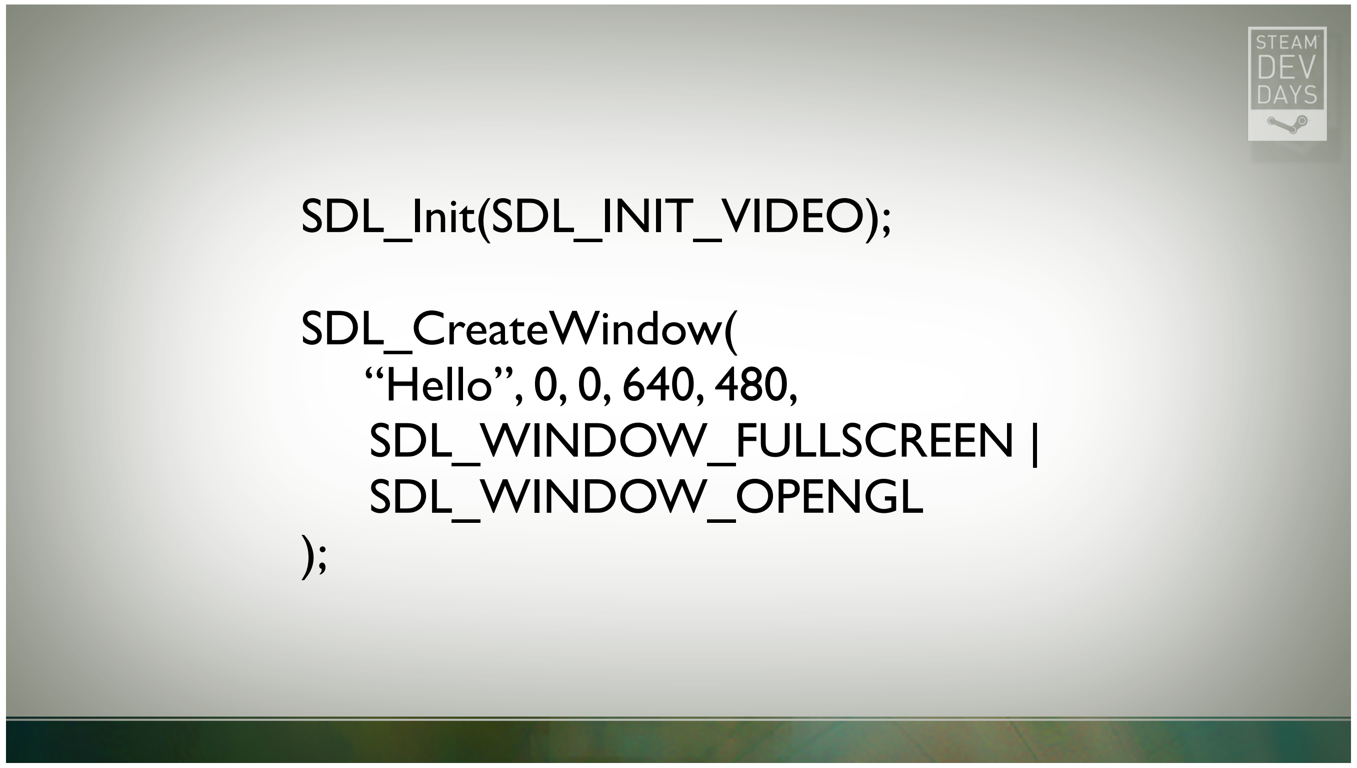
```
// TO DO: Respond to failure of Direct3DCreate8
```

Look of disapproval.



Really hard SDL version

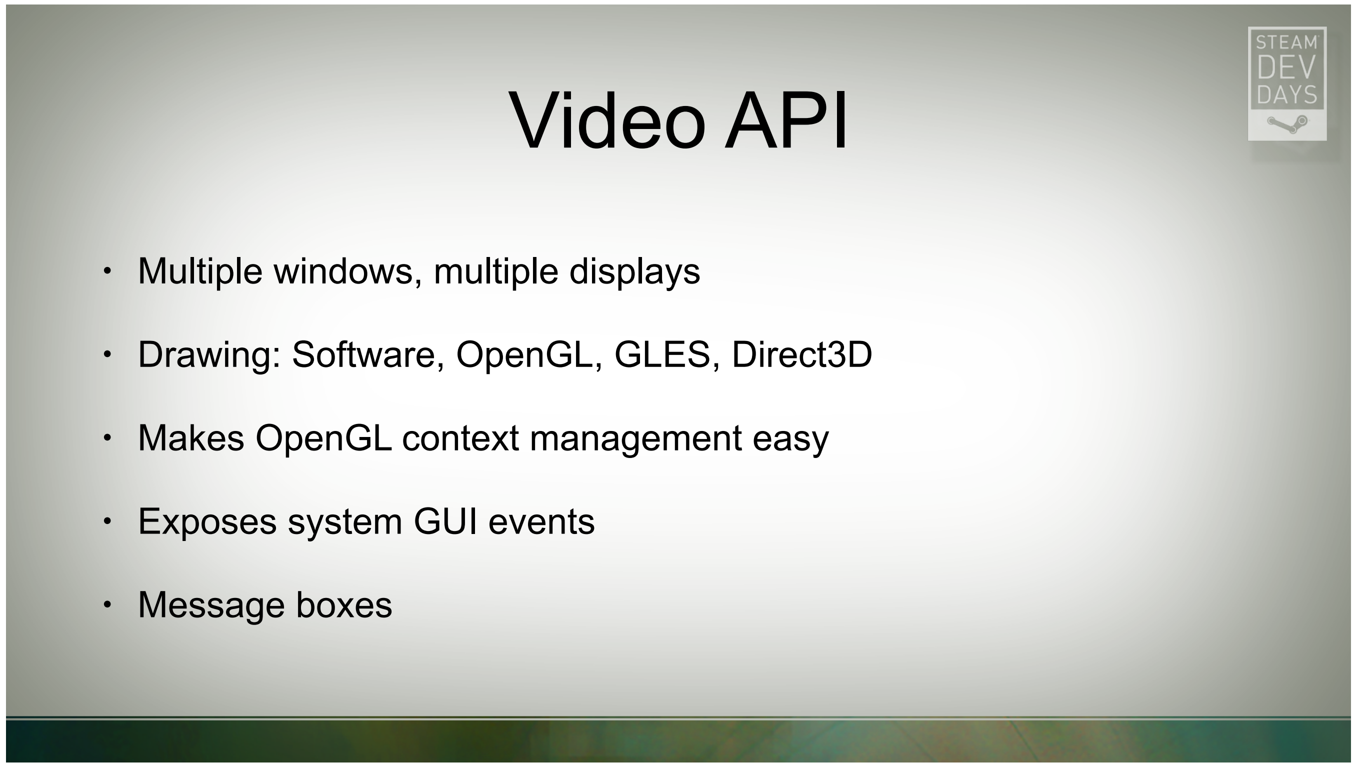
SARCASM!!



```
SDL_Init(SDL_INIT_VIDEO);

SDL_CreateWindow(
     “Hello”, 0, 0, 640, 480,
      SDL_WINDOW_FULLSCREEN |
      SDL_WINDOW_OPENGL
);
```

I also fail to check for errors. Look of disapproval!

Also: SDL_WINDOW_FULLSCREEN_DESKTOP is awesome, use it instead of SDL_WINDOW_FULLSCREEN, if you can. The Render API makes that a no-brainer (one extra function call to set logical rendering size). With OpenGL, you might need a little more effort.

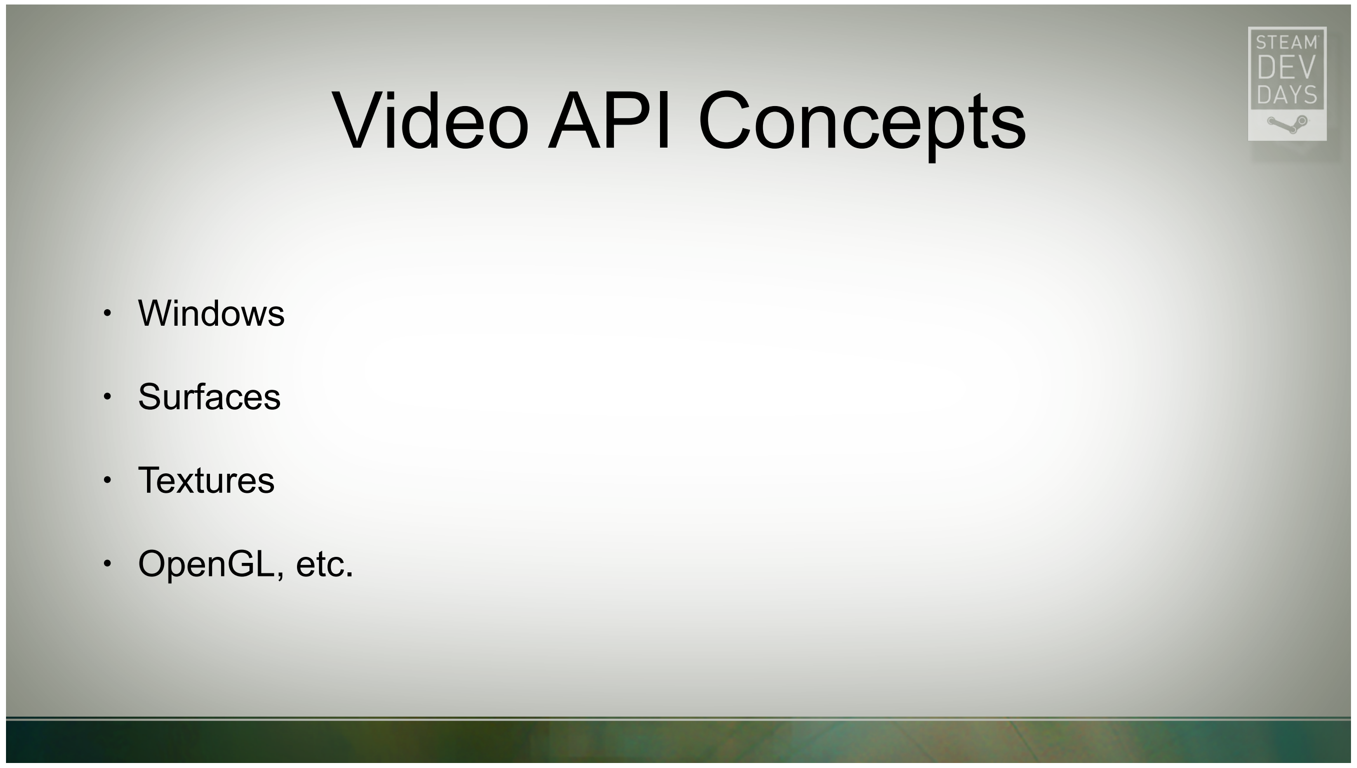


Video API

- Multiple windows, multiple displays
- Drawing: Software, OpenGL, GLES, Direct3D
- Makes OpenGL context management easy
- Exposes system GUI events
- Message boxes

You usually need a window to make a good portion of SDL useful, but most platforms can do audio and joysticks without one. Not to mention threads and a dozen other things.

You have no idea how hard it is to get a message box on the screen on some platforms.

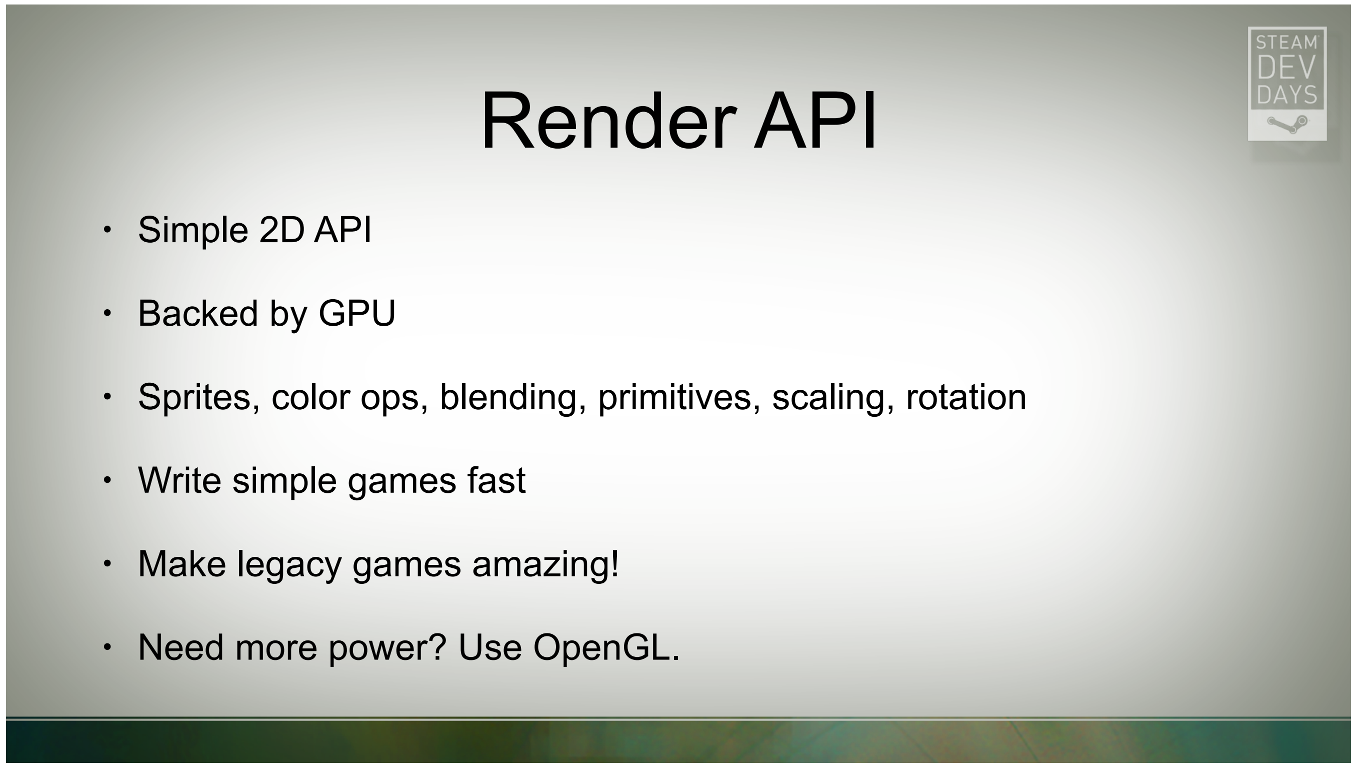


Video API Concepts

- Windows
- Surfaces
- Textures
- OpenGL, etc.

SDL 1.2 dealt only with Surfaces, which might be (but usually weren't) in video RAM. SDL 2.0 makes surfaces always in software (but they still need to be "locked" in case we've RLE-encoded them for efficiency). Textures are in GPU memory, unless you're using the software renderer, which you probably aren't.

tl;dr: Surface == system RAM, Texture == GPU RAM.

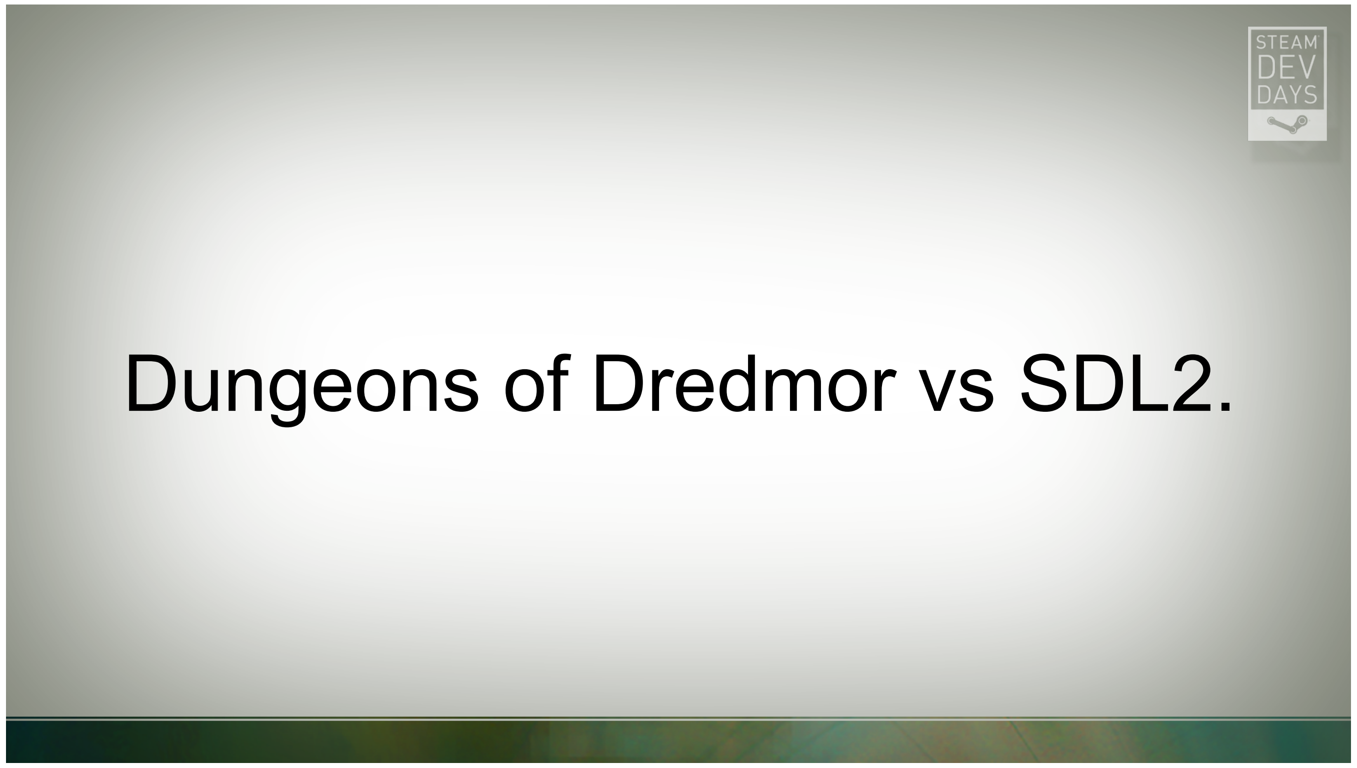


Render API

- Simple 2D API
- Backed by GPU
- Sprites, color ops, blending, primitives, scaling, rotation
- Write simple games fast
- Make legacy games amazing!
- Need more power? Use OpenGL.

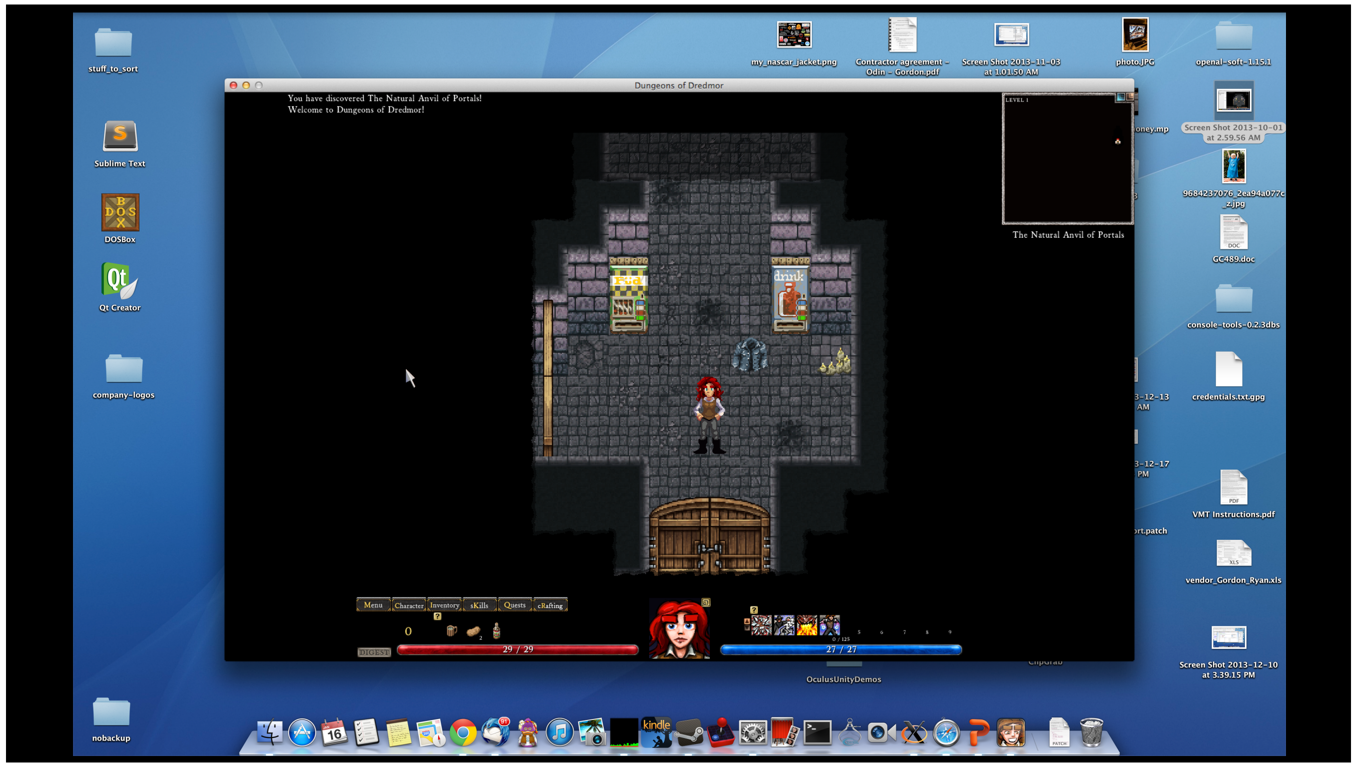
We've practiced enormous restraint with the Render API. It's nearly irresistible to build something that wraps OpenGL and Direct3D and exposes all their functionality, but that way lies madness.

What we decided we wanted was something that makes simple games simple…think of a really powerful computer with the rendering primitives of a Super Nintendo. We also wanted it to make legacy, software-rendered games easy to get onto the GPU, so we could upscale them, accelerate them, and plop a Steam Overlay on top of them.



Dungeons of Dredmor vs SDL2.

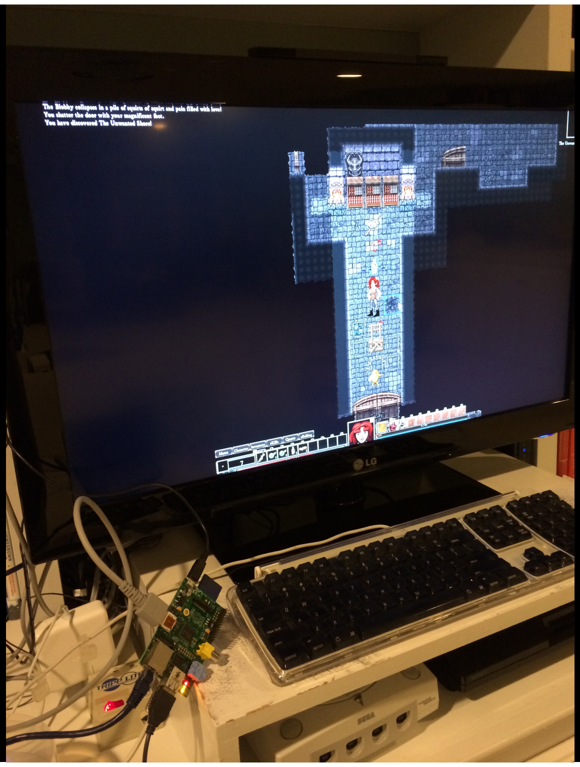
But hey, screeeeeeeeeeeeeeenshot demooooooooooooo…!


Dungeons of Dredmor
You have discovered The Natural Anvil of Portals!
Welcome to Dungeons of Dredmor!
The Natural Anvil of Portals
29 / 29
27 / 27

This is Dungeons of Dredmor, ported to SDL2, using the Render API, on Mac OS X. It looks the same on Linux and Windows, but the desktop is less pretty. But it’s just as cluttered. :/

This is Dungeons of Dredmor, ported to SDL2, using the Render API, on iOS. Sorry for the grainy photo.

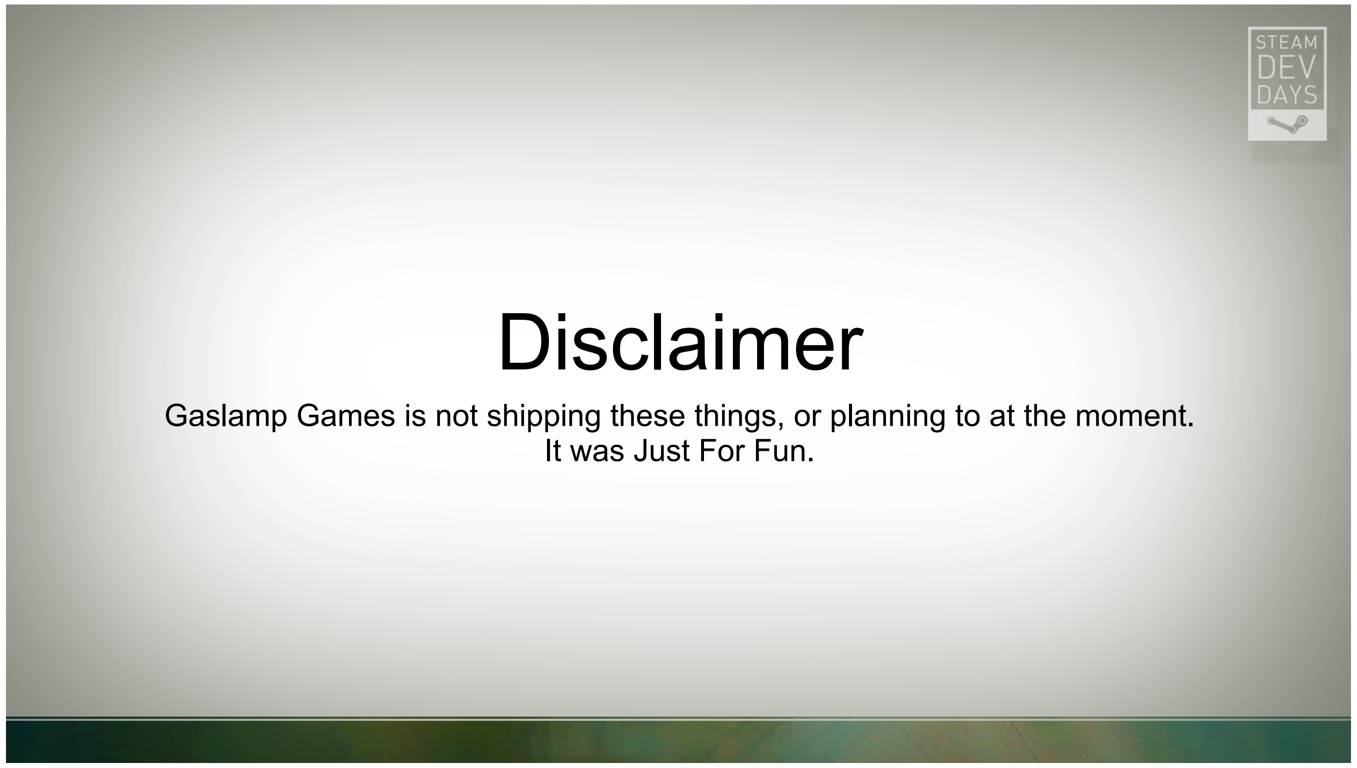
Once we had the SDL2 port running on desktops, the total lines of code changes to get this far: zero.



This is Dungeons of Dredmor, ported to SDL2, using the Render API, on a Raspberry Pi. This is using OpenGL ES on a VT, without X11.

Once we had the SDL2 port running on desktops, the total lines of code changes to get this far: zero.

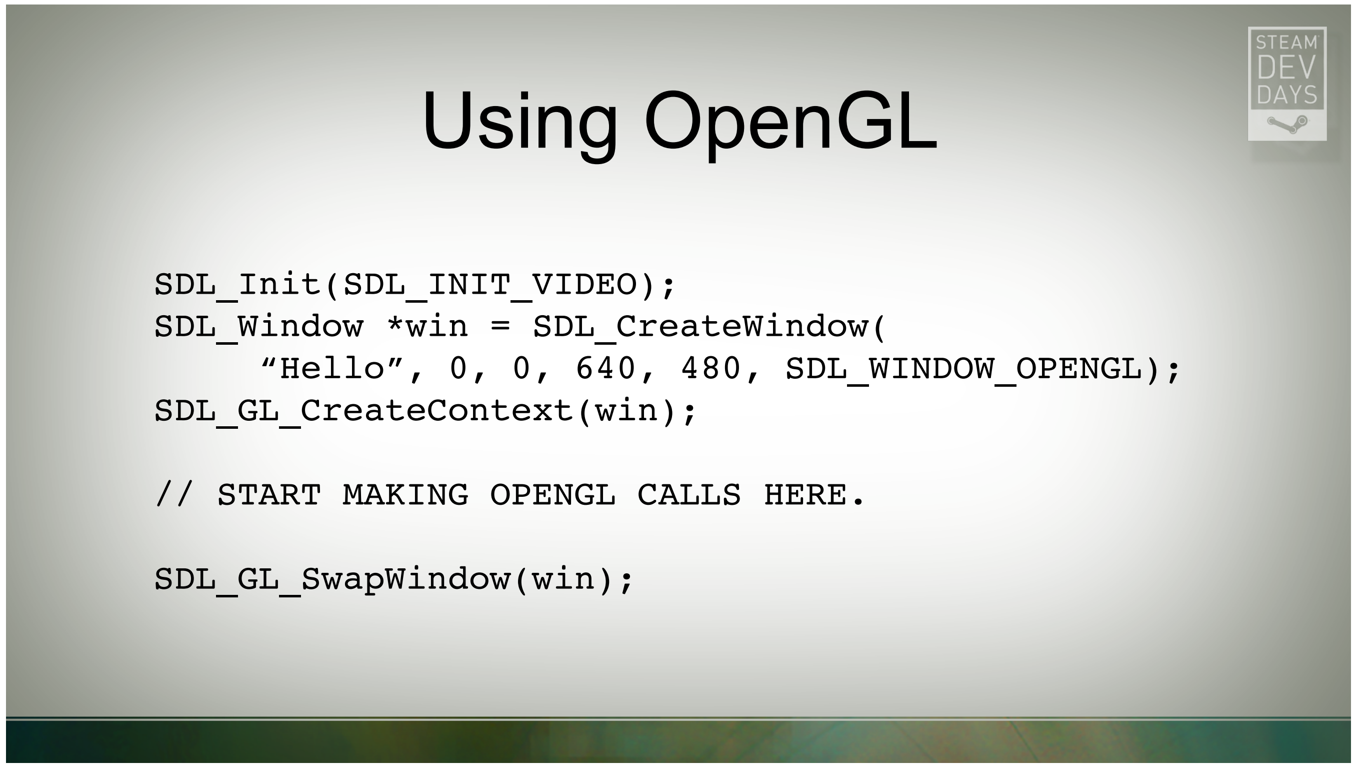
That Dreamcast under the keyboard? Not quite there yet.



Disclaimer

Gaslamp Games is not shipping these things, or planning to at the moment.
It was Just For Fun.

But anything is possible!



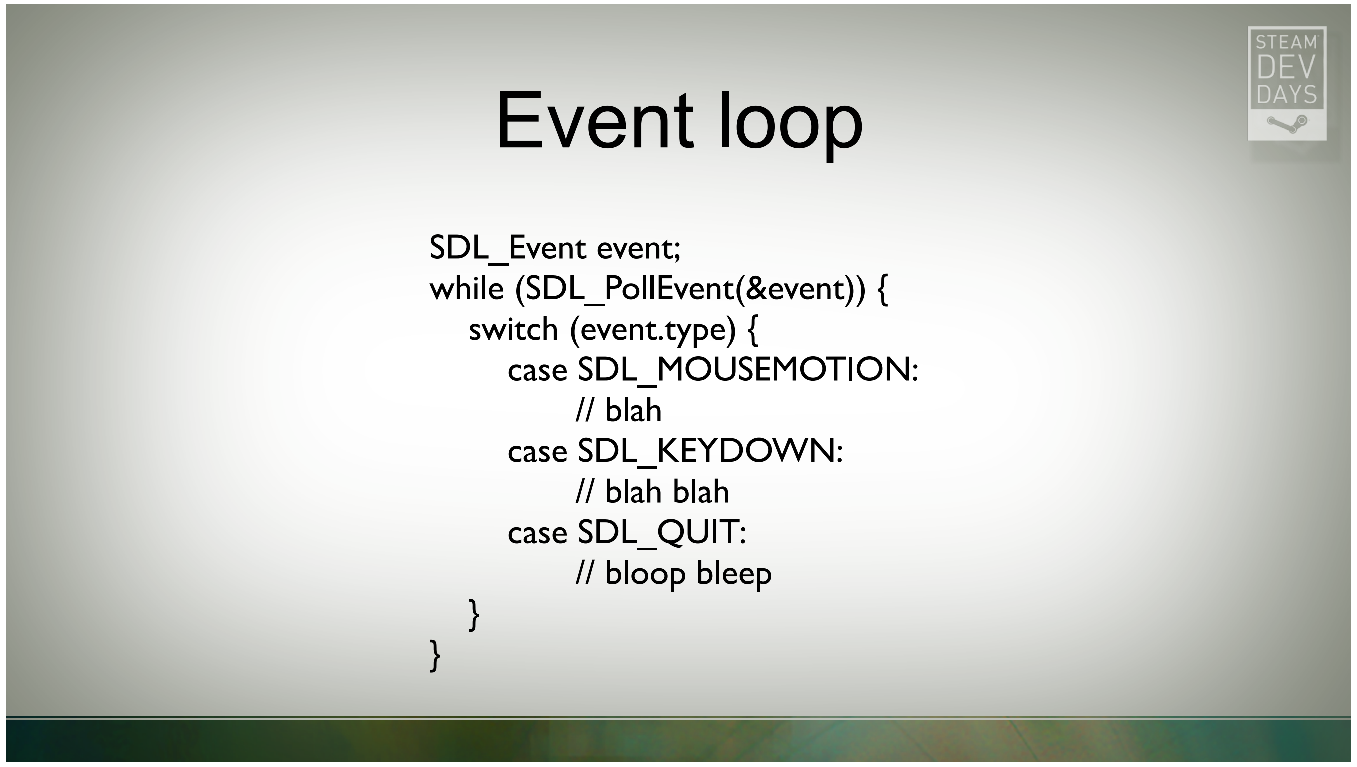
Using OpenGL

```
SDL_Init(SDL_INIT_VIDEO);
SDL_Window *win = SDL_CreateWindow(
     “Hello”, 0, 0, 640, 480, SDL_WINDOW_OPENGL);
SDL_GL_CreateContext(win);

// START MAKING OPENGL CALLS HERE.

SDL_GL_SwapWindow(win);
```

Extra credit for researching SDL_GL_SetAttribute() and SDL_GL_SetSwapInterval().

Events

- OS Events (mouse, keyboard, window)
- Relative mouse mode
- Touch API
- Gestures
- Joysticks and Game Controllers
- Timers



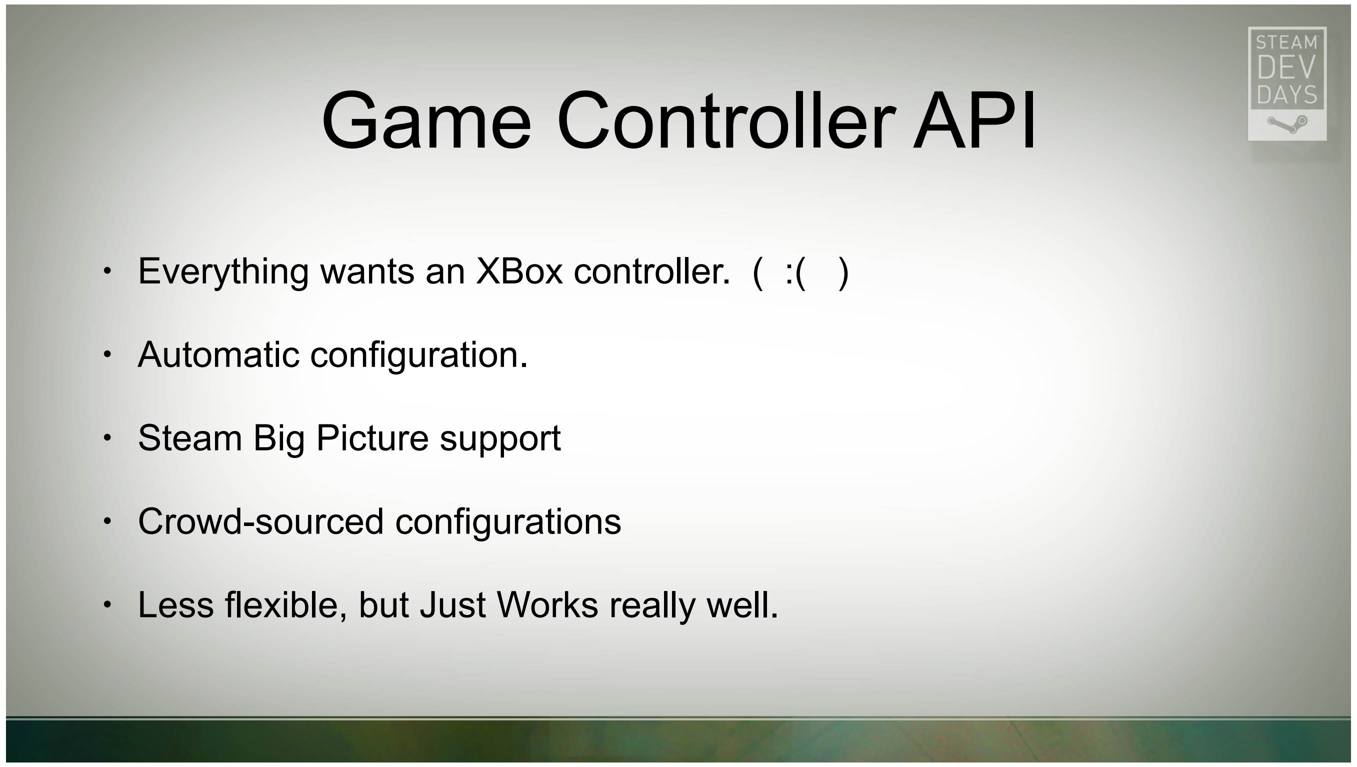
Event loop

```
SDL_Event event;
while (SDL_PollEvent(&event)) {
    switch (event.type) {
        case SDL_MOUSEMOTION:
               // blah
        case SDL_KEYDOWN:
               // blah blah
        case SDL_QUIT:
               // bloop bleep
    }
}
```

Don’t forget the “break;” statements, though.

Joystick API

- Multiple sticks
- Polling or events
- Query axes, buttons, hats, names
- Connect and disconnect notifications



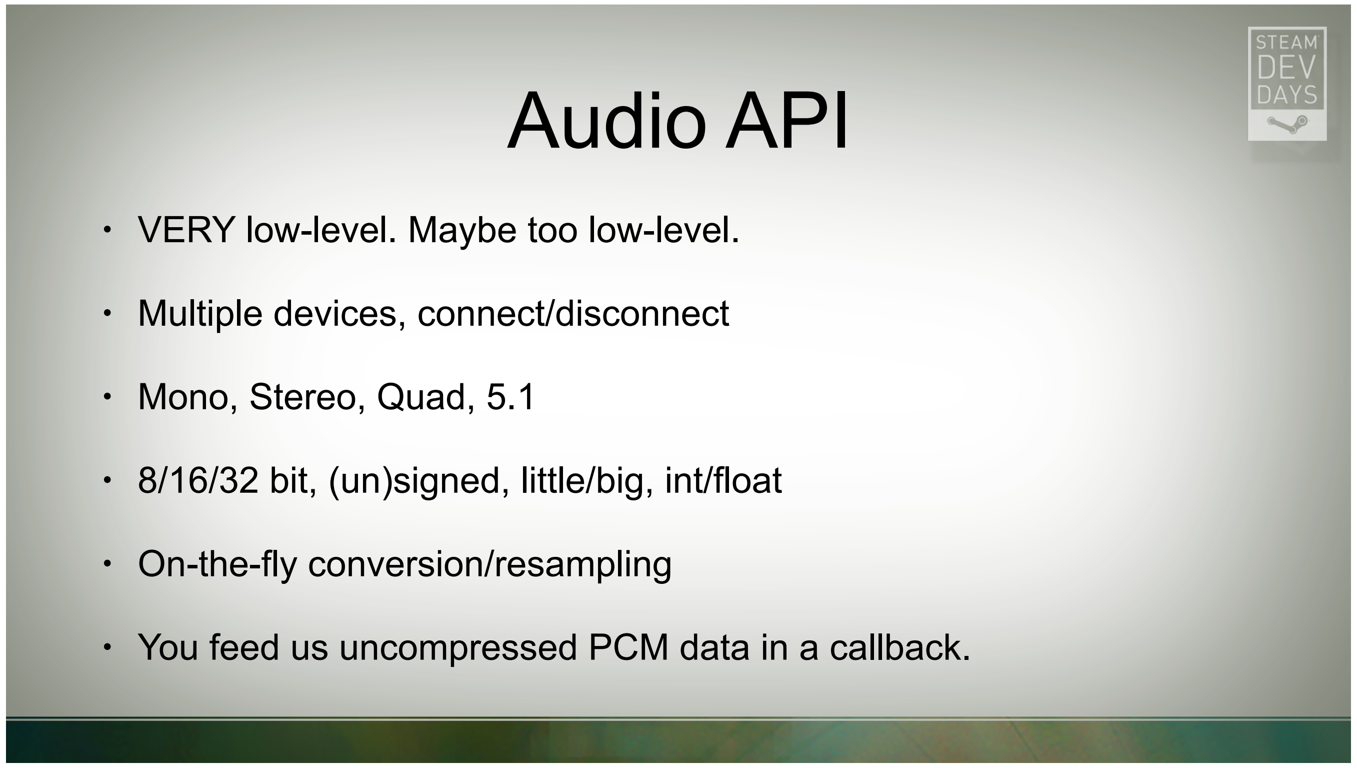
Game Controller API

- Everything wants an XBox controller. (:(   )
- Automatic configuration.
- Steam Big Picture support
- Crowd-sourced configurations
- Less flexible, but Just Works really well.

I didn’t know the proper protocol for wrapping an emoticon in parentheses.

Haptic API

- “Haptic” == “Force feedback”
- Supports controllers *and* mice!
- Complex effects, simple rumble, left/right
- Fire and forget

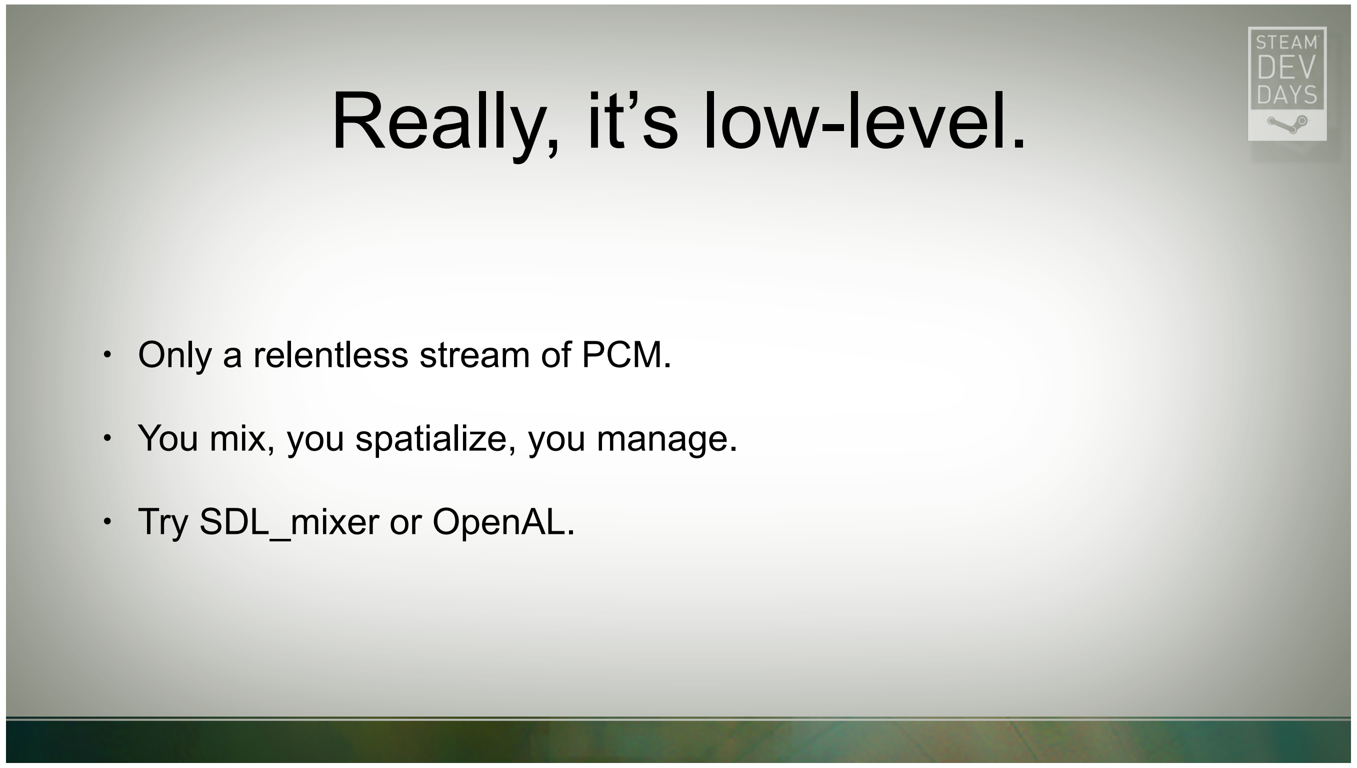


Audio API

- VERY low-level. Maybe too low-level.
- Multiple devices, connect/disconnect
- Mono, Stereo, Quad, 5.1
- 8/16/32 bit, (un)signed, little/big, int/float
- On-the-fly conversion/resampling
- You feed us uncompressed PCM data in a callback.

You can tune out for this slide if you aren't hardcore.

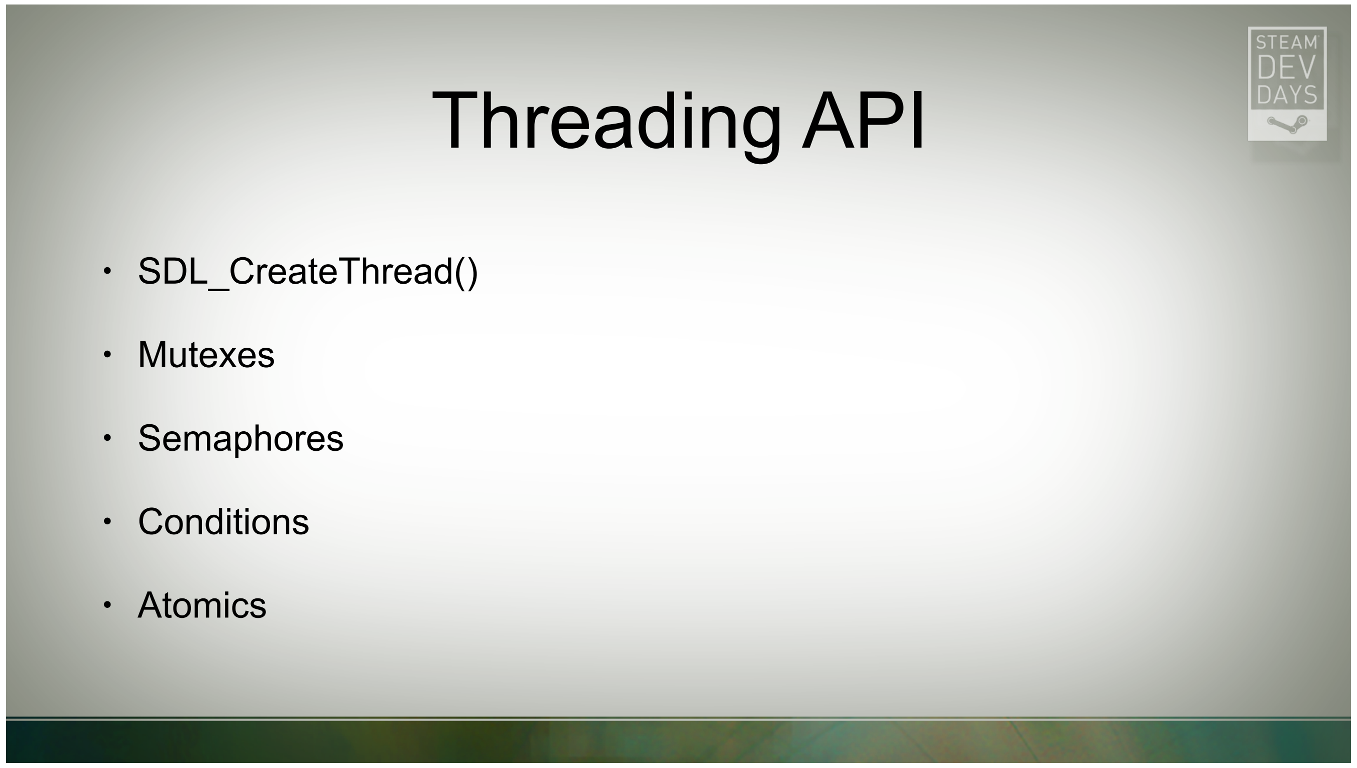
The callback runs in a separate thread that SDL spins internally. In SDL 1.2, some platforms, like Mac OS Classic, ran the callback in a hardware interrupt! I don't miss those days.



Really, it's low-level.

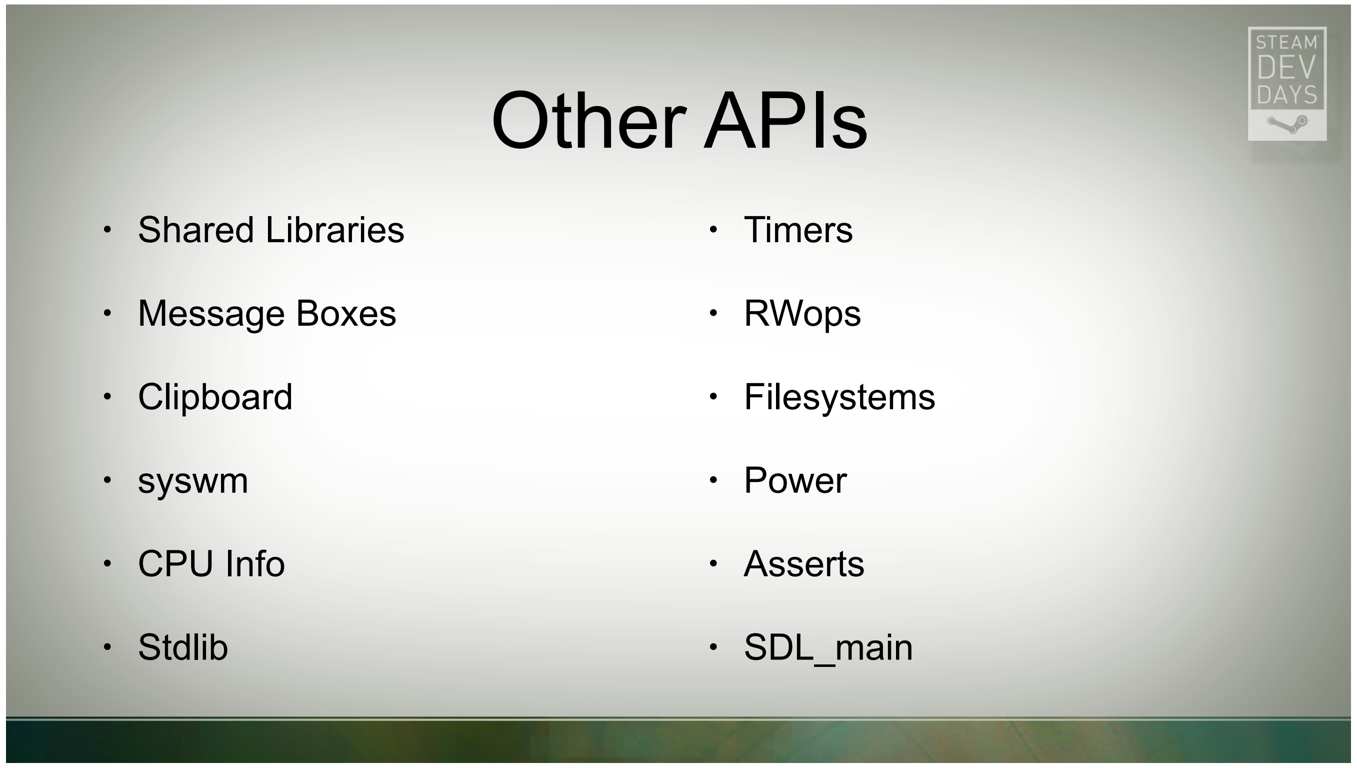
- Only a relentless stream of PCM.
- You mix, you spatialize, you manage.
- Try SDL_mixer or OpenAL.

Several titles have used it, and several audio libraries have been built on top of it, to get portability while providing a more powerful API.


Threading API

- SDL_CreateThread()
- Mutexes
- Semaphores
- Conditions
- Atomics

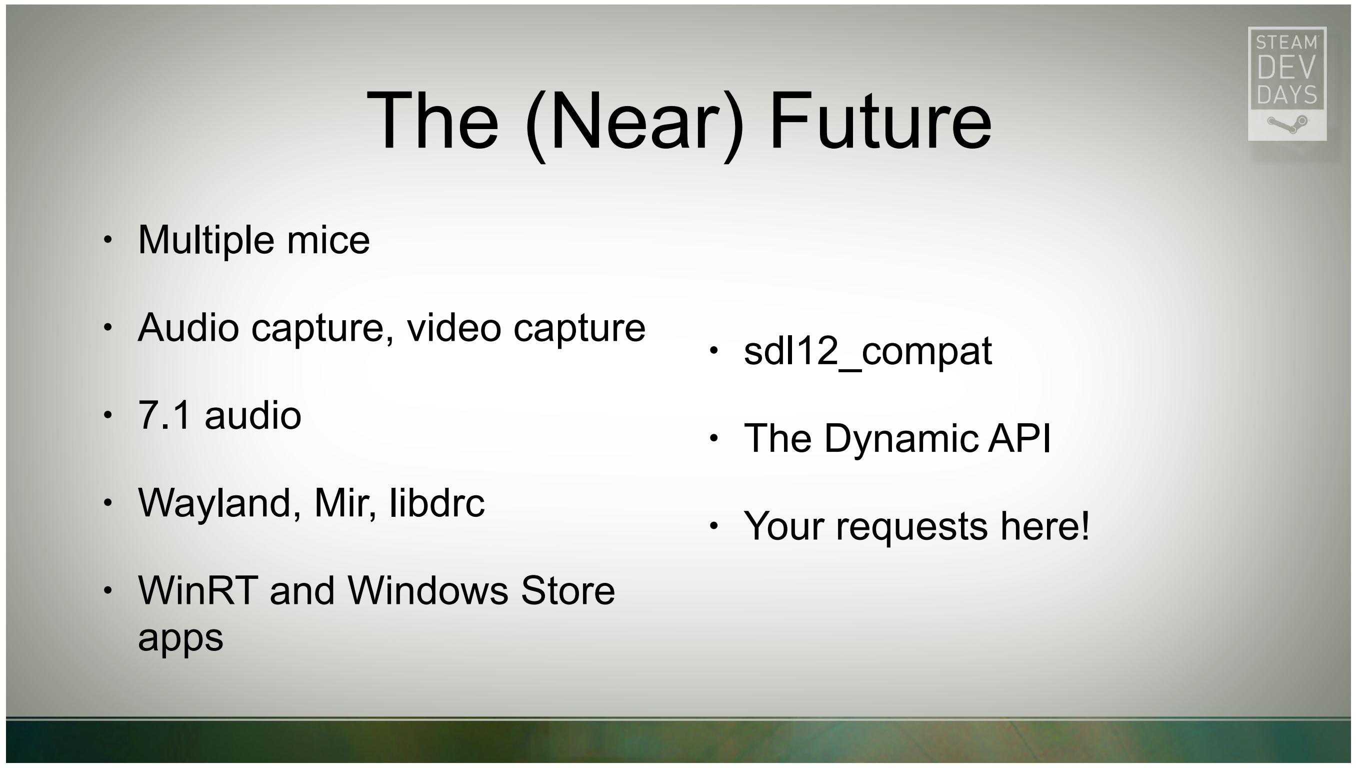
SDL_CreateThread() takes a string to name for the thread. We try to convince your OS to show you this name in the debugger if possible.



Other APIs

- Shared Libraries
- Message Boxes
- Clipboard
- syswm
- CPU Info
- Stdlib
- Timers
- RWops
- Filesystems
- Power
- Asserts
- SDL_main

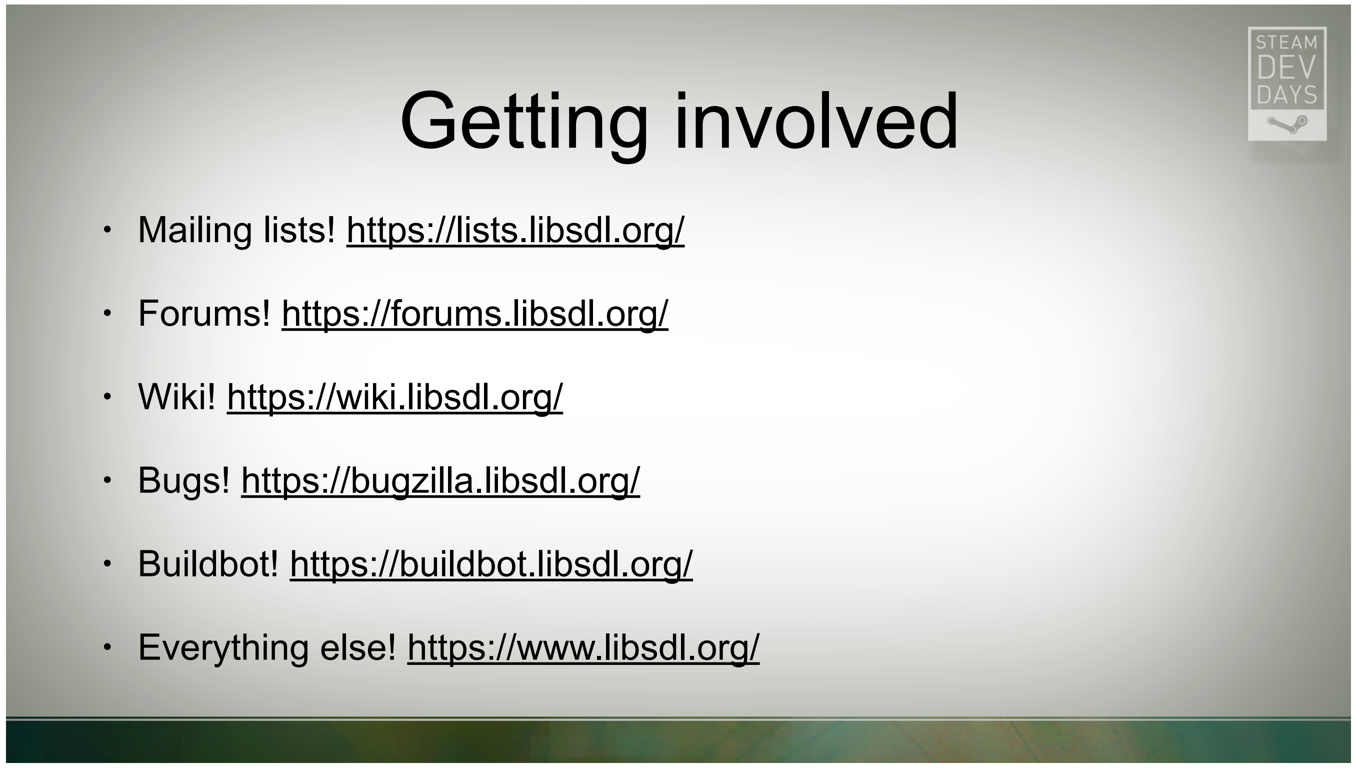
This slide is another hour-long talk in itself.



The (Near) Future

- Multiple mice
- Audio capture, video capture
- 7.1 audio
- Wayland, Mir, libdrc
- WinRT and Windows Store apps
- sdl12_compat
- The Dynamic API
- Your requests here!

We have more crazy ideas on the wishlist than we have engineers to implement them in a timely fashion. If you consider yourself a hacker, we'd love your contributions!



Getting involved

- Mailing lists! https://lists.libsdl.org/
- Forums! https://forums.libsdl.org/
- Wiki! https://wiki.libsdl.org/
- Bugs! https://bugzilla.libsdl.org/
- Buildbot! https://buildbot.libsdl.org/
- Everything else! https://www.libsdl.org/

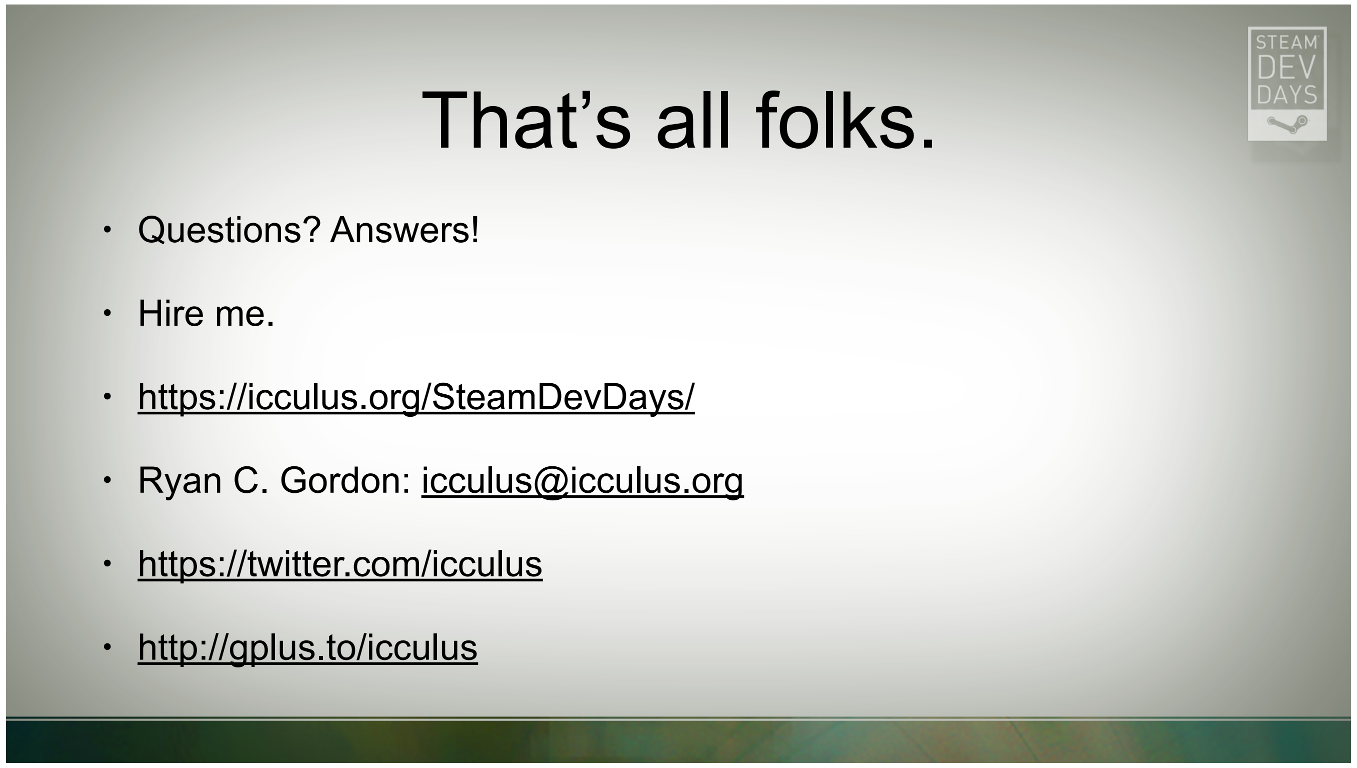
https for the win. http works too, if you're one of those types.

We have spent a lot of time on the wiki, and consider it the best source of documentation. The public headers are heavily documented, but we tend to expand on the details more on the wiki. Don't be afraid to use it (and complain to us if something needs improvements).

Bugzilla is also heavily used; we've been through over 2000 bug reports on it, and tend to turn there first when looking for something to do, and tend to forget to do things that haven't made it into the bug tracker.

The forums are wired up to the mailing lists, bidirectionally. You can pick your favorite form of communication, and it will show up on both outlets.

Buildbot lets us know when we broke something immediately.



That's all folks.

- Questions? Answers!
- Hire me.
- https://icculus.org/SteamDevDays/
- Ryan C. Gordon: icculus@icculus.org
- https://twitter.com/icculus
- http://gplus.to/icculus

No, really, throw money and retweets at me.

I listed a Twitter and Google+ account here. If you're using Facebook, you should reevaluate your life choices.