
STEAM
DEV
DAYS
Ryan C. Gordon
icculus.org
Getting Started with Linux Game Development

Here we go.

A few notes…

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

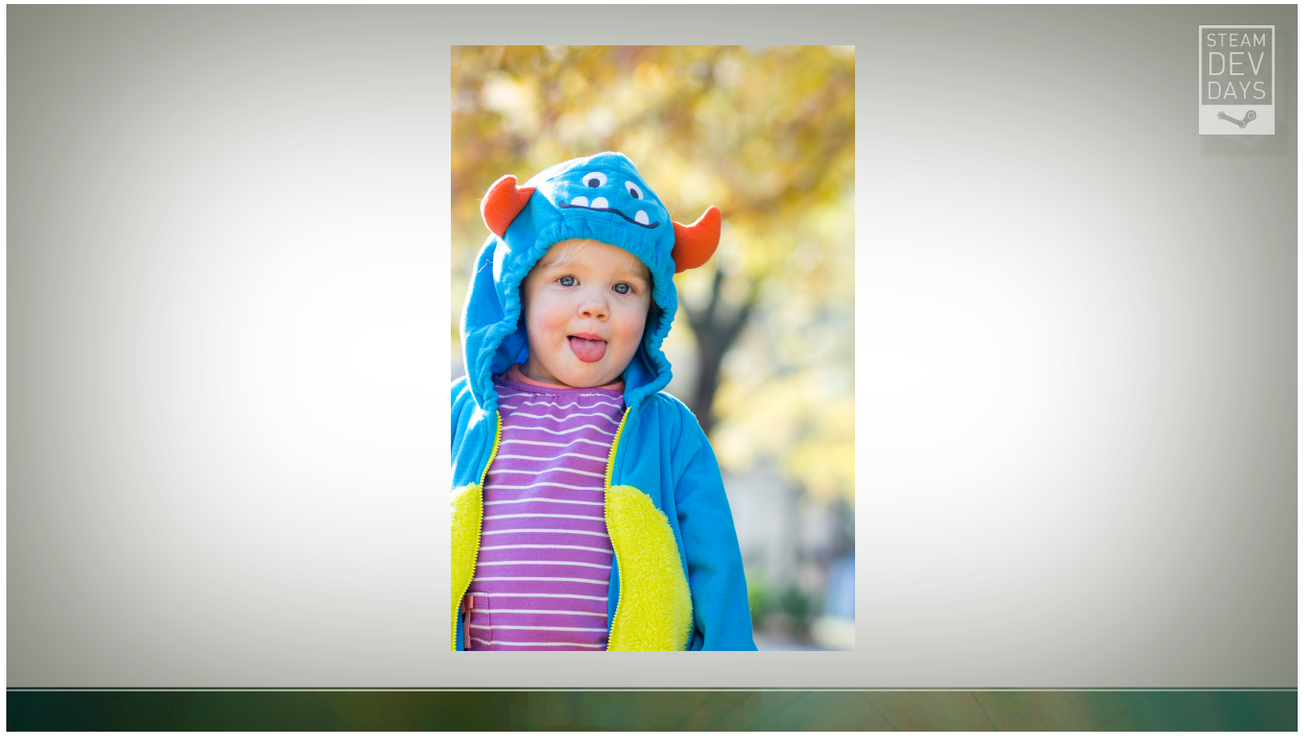
I could talk about the nitty-gritty of this for _hours_, but this talk won't have many code samples, etc. Mostly I want you to walk out of here knowing what you can do, and see the later talks for more depth on various subjects.

Who am I?

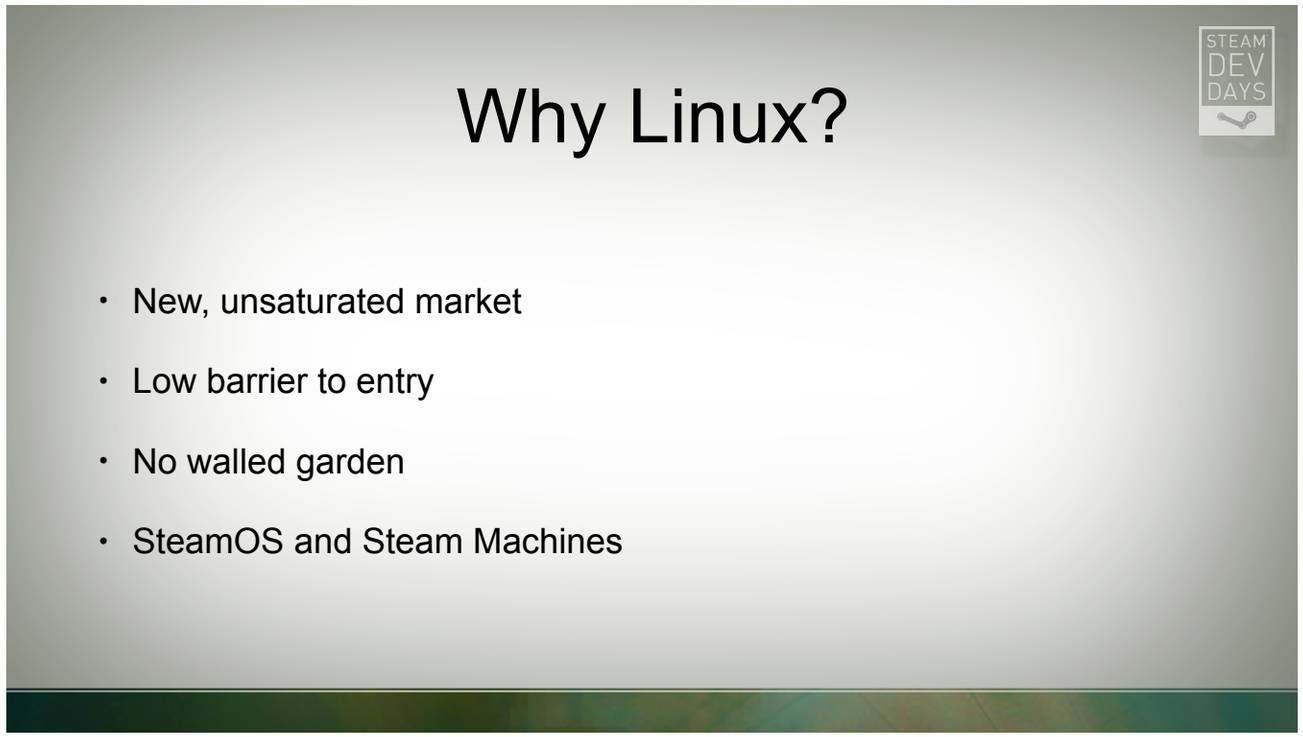
- Hacker, game developer, porter
- Port games, build tools
- Freelance
- 15 years experience

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…



…and this is my other full time job.



The Linux ecosystem isn’t flooded with cheap product, like Windows. You don’t have to compete to be heard over the mob.

No one dictates what you can publish, when you can update, what you can charge. No one takes a cut from you. There’s an app store on some Linux systems, but unlike other systems, you aren’t forced to use them to deliver software, so you aren’t beholden to them in any case.

And, uh, naturally, if you want to run on SteamOS, here you are.

MythBusting

- Distro fragmentation isn’t real
- Most hardware is supported
- GPU drivers are good
- Linux users spend money on software
- Tech you need is available for Linux

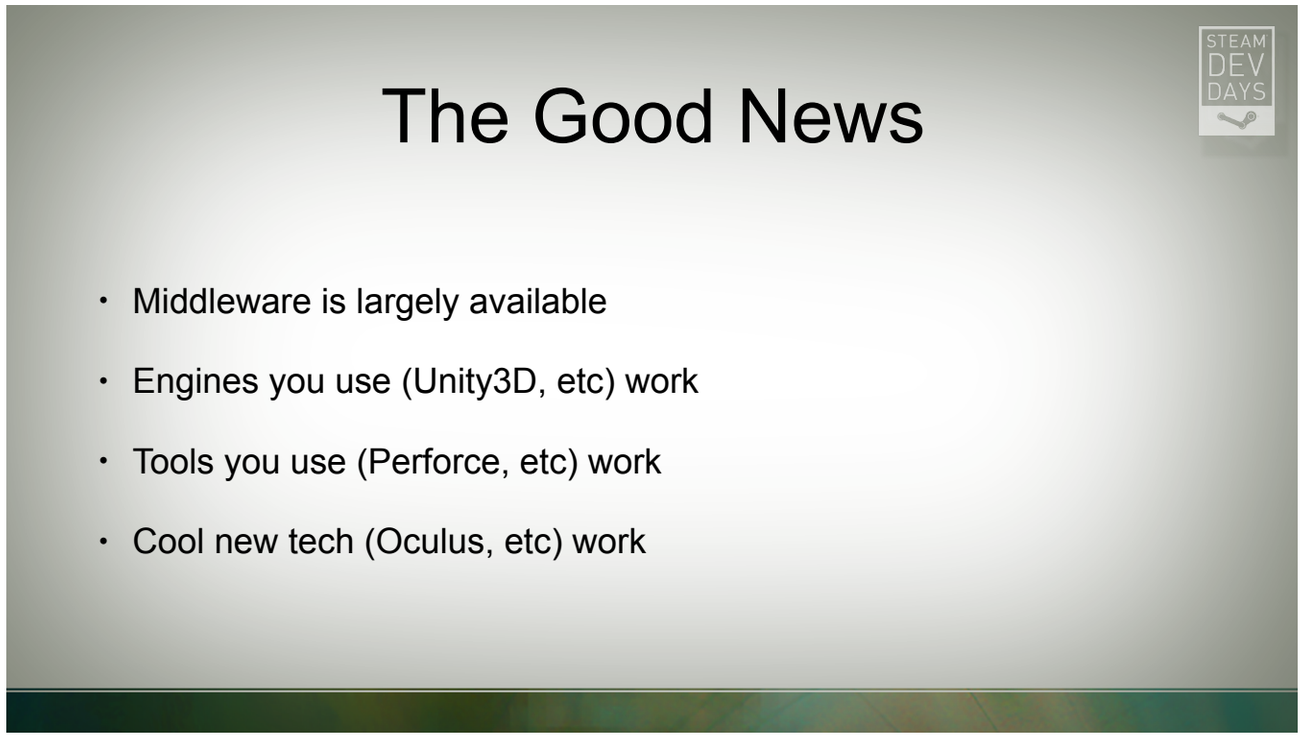
Truth about distro fragmentation: I’ve shipped dozens of games, over 15 years, never did anything but a generic Linux build for any game. SDL handles most of the differences for you, the rest are avoidable and largely unimportant anyhow.

Since most drivers live in the Linux kernel tree, and continue to be maintained, whereas old hardware that had a Win95 driver never got updated for XP and later, one could argue Linux supports more hardware than Windows at this point, collectively. More to the point: most things you buy off the shelf tend to Just Work on Linux now, which was not true if you last tried it years ago.

Nvidia and AMD ship their own GPU drivers, and there are open source ones too. All of them are somewhere between pretty functional and world-class.

Linux users consistently spend more per capita than Mac and Windows users do, across every Humble Bundle promotion (including ones where there aren’t actually any Linux games for sale).

Cool, disruptive tech, like the Oculus Rift, or the Leap Motion controller, are shipping Linux SDKs.



Middleware is probably available, much of it as source code. It would almost be easier to list things that aren't available.

Engines are there: Unity3D pretty much offers a "click here for Linux port" button, Unreal Engine 3 has been ported by outside developers at least three times now (Dungeon Defenders, Painkiller HD, Papo Y Yo, all ported by different developers, with more to come), idTech 1-4 all shipped Linux versions. Source Engine runs on Linux. Leadworks ran a successful Kickstarter to bring their tech to Linux this year. MonoGame-SDL2 helps get XNA games running. This list goes on.

Tools you use, Perforce, Subversion, scripting languages, etc…they probably have Linux ports, if they weren't Linux projects to start with.

The Bad News

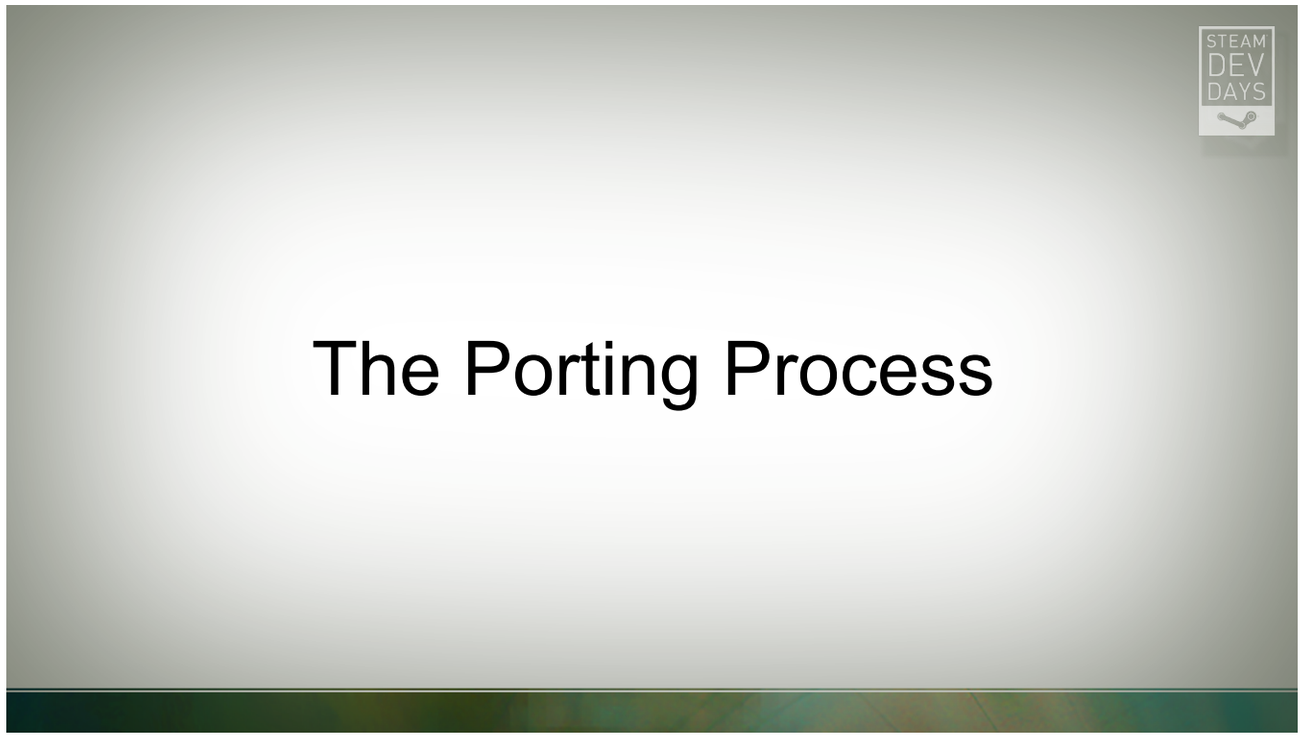
- If you ***need*** Visual Studio, this is harder.

http://en.wikipedia.org/wiki/Stockholm_syndrome

So wait…no Visual Studio?!

- Emacs and vi
- Sublime Text
- QtCreator
- Code::Blocks
- Eclipse
- Many other options

Many of these work on Windows, Linux, and Mac OS X, and some are much better than Visual Studio.



insert dramatic music here

DUN DUN DUUUUUUUUUN

Start with Windows

- Move to SDL2 on Windows
- Move to OpenGL on Windows
- *Then* start port to Linux

If you’re primarily a Windows developer, you can do 90% of the hard part without even installing Linux.

Get it compiling

- GCC, Clang, Intel C++
- Makefiles, CMake, Premake, scons

…but really GCC or Clang.

Mining from VS2008 .vcproj

```
<File
  RelativePath="src\audio\player.cpp"
  >
```

Visual Studio project files are just XML documents, so you could write a tool that formally parses these to get build information, but it's hard to do this well, and generally you only have to do it once, so no one ever wrote it. That being said, a little shell script and perl will get you a list of files used in the project.

```
grep "RelativePath=" myproj.vcproj |perl -pi -e 's/\r//; s/\A\s*RelativePath="//; s/\"//; s#\\#/#g;'
```

Note that header files also get caught in this, so you might want to filter for ".cpp" or whatnot, too. VS2010 and later specify headers elsewhere and don't have that problem. See next slide.

Mining from VS2010 .vcxproj


```
<ClCompile Include="src\audio\player.cpp" />
```

grep “ClCompile“ proj.vcxproj |perl -pi -e ’s/\r//; s/\A\s*\<ClCompile Include=“//; s#”\s*/\>##; s#\\#/#g;’

When in doubt, stub it out

#define STUBBED(x) printf("STUBBED: %s\n", x)

This is simple enough:

STUBBED(“Need a way to read .dds files”);

When in doubt, stub it out

```
#define STUBBED(x) do { \
    static bool seen_this = false; \
    if (!seen_this) { \
        seen_this = true; \
        fprintf(stderr, "STUBBED: %s at %s (%s:%d)\n”, \
                x, __FUNCTION__, __FILE__, __LINE__); \
    } \
} while (0)
```

You can get way fancier, though. This is the macro I personally use in my work.

When in doubt, stub it out

MessageBox(hwnd, “Out of memory”, “Error”, 0);

So this won’t fly…

When in doubt, stub it out


```
#if WINDOWS
MessageBox(hwnd, “Out of memory”, “Error”, 0);
#else
STUBBED(“Need a Linux msgbox”);
#endif
```

…but now you know to come back to this later. You’d be surprised: lots of games ship with STUBBEDs in them still, because we found out the code was never actually used. When shipping, you can change the STUBBED macro to be a no-op, or make it generate an error, depending on whether you want to ignore the remaining ones or be forced to clean them out.

Also useful: you can grep for STUBBED to know roughly what the remaining workload looks like.

p.s.: SDL_ShowSimpleMessageBox().

Don’t do this.

```
#if LINUX
    some_non_windows_thing();
#endif
```

I call this “Third Platform Syndrome.” You had a game, you ported it somewhere else, and the third platform could have benefitted from that work, but you gated all the stuff that was specific to the first platform with checks for the second. The third platform loses. This code compiles out, leaving you with no warning; it’s not there anymore.

Do this!

```
#if !WINDOWS
    some_non_windows_thing();
#endif
```

…so say what you actually meant.

Definitely don’t do this.

```
#if WINDOWS
    some_windows_thing();
#elif PLAYSTATION
    some_ps_thing();
#endif
```

Same deal for places where you have to have platform-specific code for each case…you are basically guaranteeing this will cause you problem with each new platform.

Do this!

```
#if WINDOWS
    some_windows_thing();
#elif PLAYSTATION
    some_ps_thing();
#else
    #error Oh no.
#endif
```

…so complain loudly when this happens, so you know to come back and do something about it.

Inline assembly

```
_asm {
    mov ecx, 1
    mov eax, ecx
}
```

This asm doesn’t do anything you care about, except maybe corrupt the current function’s register state. It’s just for example purposes. This is how it looks for Visual Studio.

Inline assembly

```
__asm__ __volatile__ (
    “movl $1, %%ecx \n”
    “movl %%ecx, %%eax \n”
    : “a”
    : /* no inputs */
    : “ecx”
);
```

This is the equivalent in GCC. I’d explain what this means, but I’m trying to show you that you shouldn’t want anything badly enough to have to deal with this pile of crap.

Inline assembly

- Don’t use inline asm. Ever.
- Seriously, don’t do it.
- Compiler intrinsics
- SDL2 atomics, SDL_GetTicks(), SDL_GetPerformanceCounter(), etc.
- nasm

Everything you want for inline asm is handled better elsewhere, so don’t do it.

Get it compiling

- Stub out problem code
- Fix simple things now
- Don’t touch anything else (for now)

Fix simple things now instead of STUBBED’ing them. If you can just twiddle a line of code, just do it. Big jobs should get STUBBED. Don’t lose momentum when you’re just trying to get it to build at all.

The urge to clean things up, refactor, and improve code is really strong, because you’re looking at pieces of your code you probably haven’t in a long time. Fight this urge. You want to minimize the changes you make right now, both to make merging easier and to make your diffs (what did I _actually change in here?) easier to read.

Definitely feel free to refactor later. If this is someone else’s game, feel free to refactor never.

Get it linking

- Middleware
- System dependencies

Middleware is often a matter of making sure you have the right versions, and if they have Linux ports (or source code so you can port yourself, which in many cases is just a matter of compiling it, even if it never really had a formal port before. YMMV.). If you don't have middleware, plan to replace or remove it, which is always an interesting and unique problem.

System dependencies: in a perfect world, you want to link against the C runtime and nothing else. Ship your own copy of SDL2 with your program and let it solve the rest of it.

Get it running

- Use SDL2!
- Use OpenGL!
- (maybe) Use OpenAL!
- Use the Steam Runtime!

You should use SDL2 for lots and lots of reasons. See the other talk I'm giving at Steam Dev Days.

Filesystem gotchas

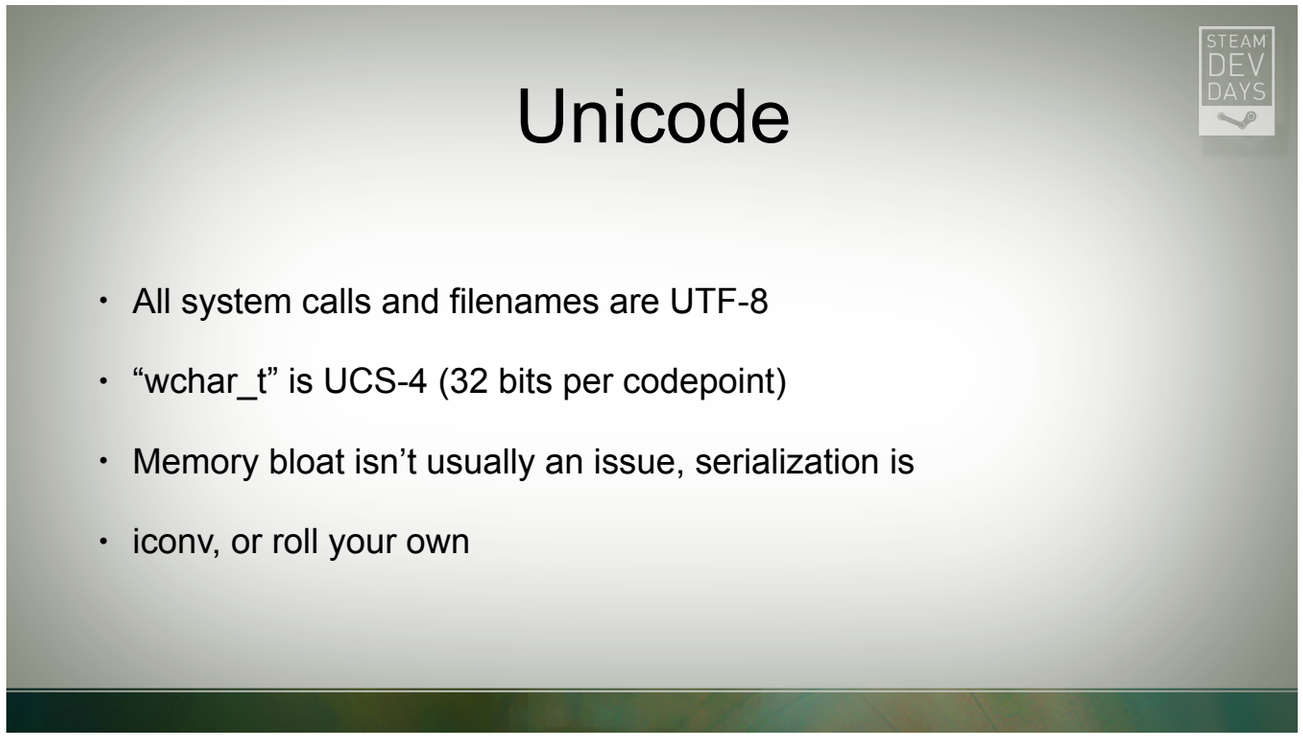
- Its paths are '/' instead of '\\'
- It's single root, no drive letters
- It's more locked down
- It's multiuser
- It's always Unicode (sort of)
- It's case-sensitive!

This is trickier than it looks, because sometimes the bugs aren't obvious.

Use SDL_GetPrefPath() and SDL_GetBasePath() to find out where you can write files, and where you game is installed. More than one login on the system might use the same install of the game, but they should each get their own savegame/config/etc directories (that's SDL_GetPrefPath()). Assume they can only write to files under the SDL_GetPrefPath() directory.

Filenames can use any byte, except '/' (that's the path separator) and a null char (that's the string terminator), but the convention is that filenames are strings encoded in UTF-8. This gives you Unicode support, and works with the same fopen() you've always used. Don't use Latin1 encoding. Ideally: try to stick to US-ASCII to keep it simple, as that happens to also be valid UTF-8.

Case-sensitivity can be a bitch. Best practice is to put your game in a handful of lowercase packfiles (which will be findable with SDL_GetBasePath()), so your game only has to find a few things on the physical filesystem, and then can do whatever it likes to extract data from those files.



UTF-8 is variable length, but it fits in a single char (often) and is identical to ASCII below codepoints 128, so it’s good for legacy code and Americans. It also happens to work with strcmp() and sorts correctly, and never contains a null char in the middle of a string (at least, not for encoding purposes).

It’s a good hack, from the people that brought you C strings.

You need to convert to UTF-8 for system calls and filenames if you have a wide string.

Windows uses UTF-16, but even developers that care about Unicode often screw this up and treat it as UCS-2. It works in most places on Earth anyhow. In such a case, you can do a simple cast from your 2-byte windows wchar_t to a 4-byte Linux wchar_t.

Since neither wchar_t is variable in size, your code (probably) just works…except when serializing to disk or a network. You might have to catch these places and convert the 2 byte data to 4.

iconv is part of the standard C runtime on Linux. SDL2 provides SDL_iconv that works everywhere. For simple conversions, you can cheat and roll your own converter.

Someone pointed out at the talk that C++11 has better wide-character support, and that might be a good cross-platform option to explore, if you are using C++11.

(This slide at the talk said “UTF-4” for Linux…that should have been UCS-4, or UTF-32. Sorry!

Get it debugged

- GDB 7
- UndoDB
- WinGDB
- VisualGDB
- QtCreator
- Valgrind
- LLVM’s AddressSanitizer
- See next two talks in this room!

GDB7 has “Reverse debugging” (tivo for bug fixes).

UndoDB is a commercial product with a similar feature, that’s faster at the moment.

QtCreator is a good option for Visual Studio users, looking for a similar GUI debugger.

Valgrind is powerful as hell, and really slow. AddressSanitizer is faster, but requires you to rebuild with instrumentation.

Clang users should try -O1 —fsanitize=address for AddressSanitizer. If you aren’t using clang, you really should anyhow.

Not listed here but worth exploring: tcmalloc

Bruce Dawson’s talk covers debugging in much more detail.

Debugging OpenGL

- ApiTrace
- gDebugger
- See Jason/Rich/Dan’s talk, immediately following this one.

Definitely go see VOGL at the OpenGL talk.

Get it optimized

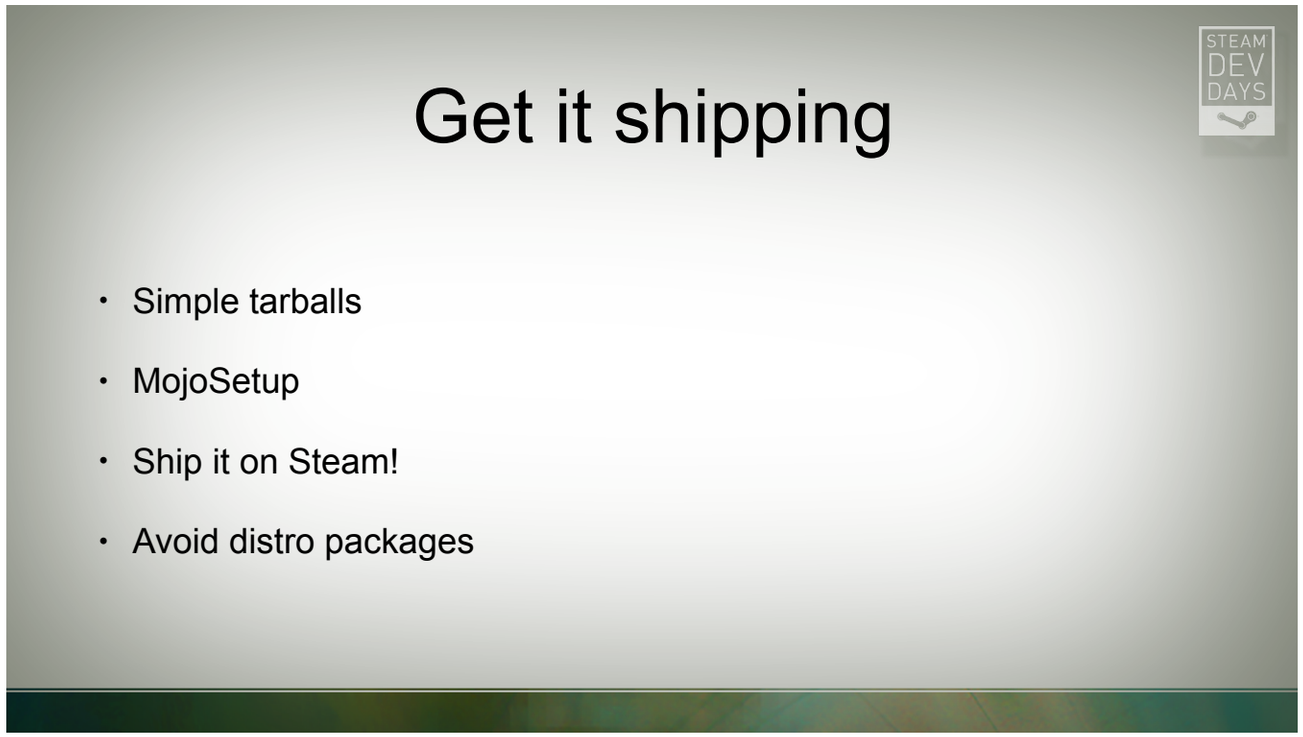
- perf
- Zoom
- ApiTrace
- Telemetry

perf is a command line tool. It’s good stuff, but you’ll be reading the documentation.

Zoom is awesome (but closed source and costs money). It looks a little like Apple’s Shark tool, because it was written by ex-Apple employees that built Shark in the first place.

Telemetry is also commercial software. It can visualize your CPU hotspots really well, and the visualization tools run natively on Linux, too.

ApiTrace functions a rudimentary OpenGL profiler. It’s better than nothing, but I think all hopes are pointed to VOGL, as of this week.



(This slide accidentally got deleted from my talk when we were trying to get the laptop to talk to the projector.)

You can ship your game as a simple tarball, or if you want/need a GUI installer, MojoSetup (which I wrote) is awesome and powerful.

Of course, shipping on Steam solves all your distribution woes, if you have that option.

Don’t ever ship .deb or .rpm, etc, packages. It’s not worth the trouble.

Contact me

- https://icculus.org/SteamDevDays
- icculus@icculus.org
- @icculus on Twitter
- http://gplus.to/icculus
- (If you hire me, you can forget this whole talk!)

I love when you give me money. In lieu of money, I accept retweets.

Other talks to see

- Moving your game to OpenGL: 3pm today, this room
- Getting started Debugging on Linux: 4pm today, this room
- Breakout: Programming on Linux, OpenGL: 5pm today, this room
- Beyond Porting: 3pm tomorrow, this room
- Game Development with SDL 2.0: 5pm tomorrow, Room 611/613

All of these talks are great, go check out their slides and videos.