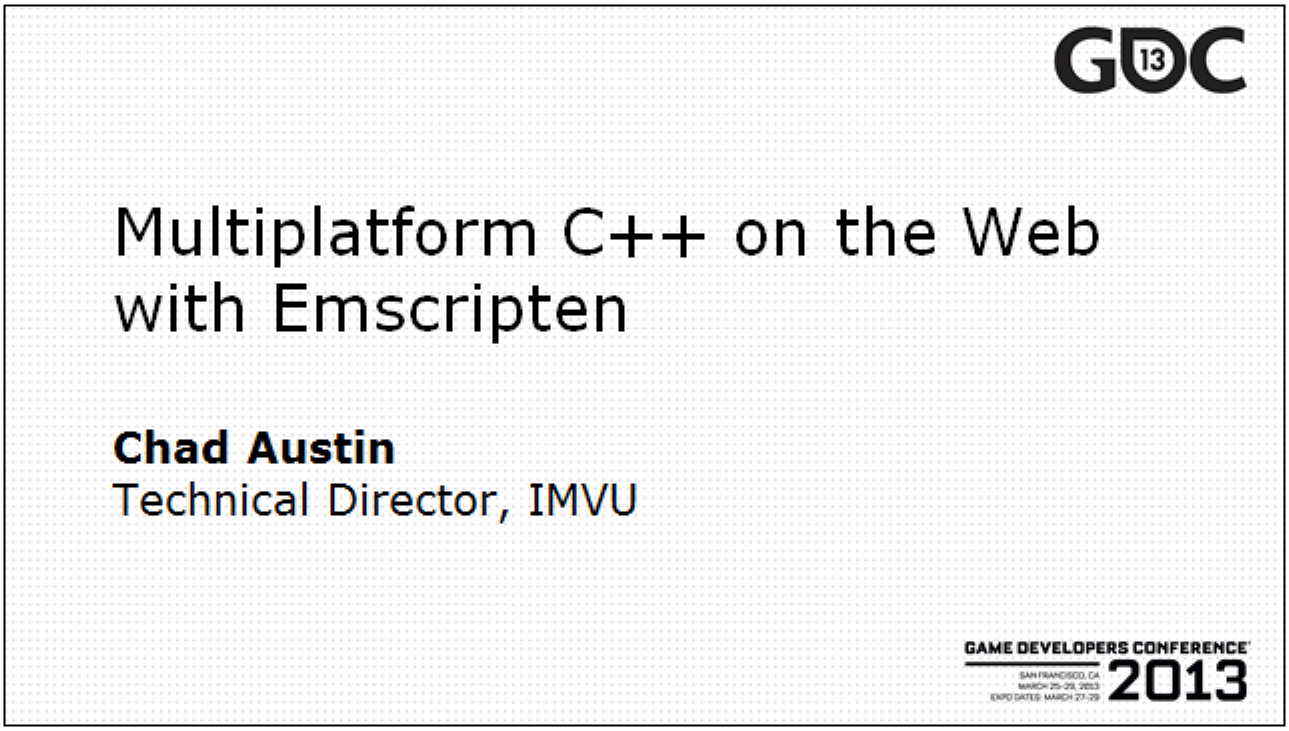
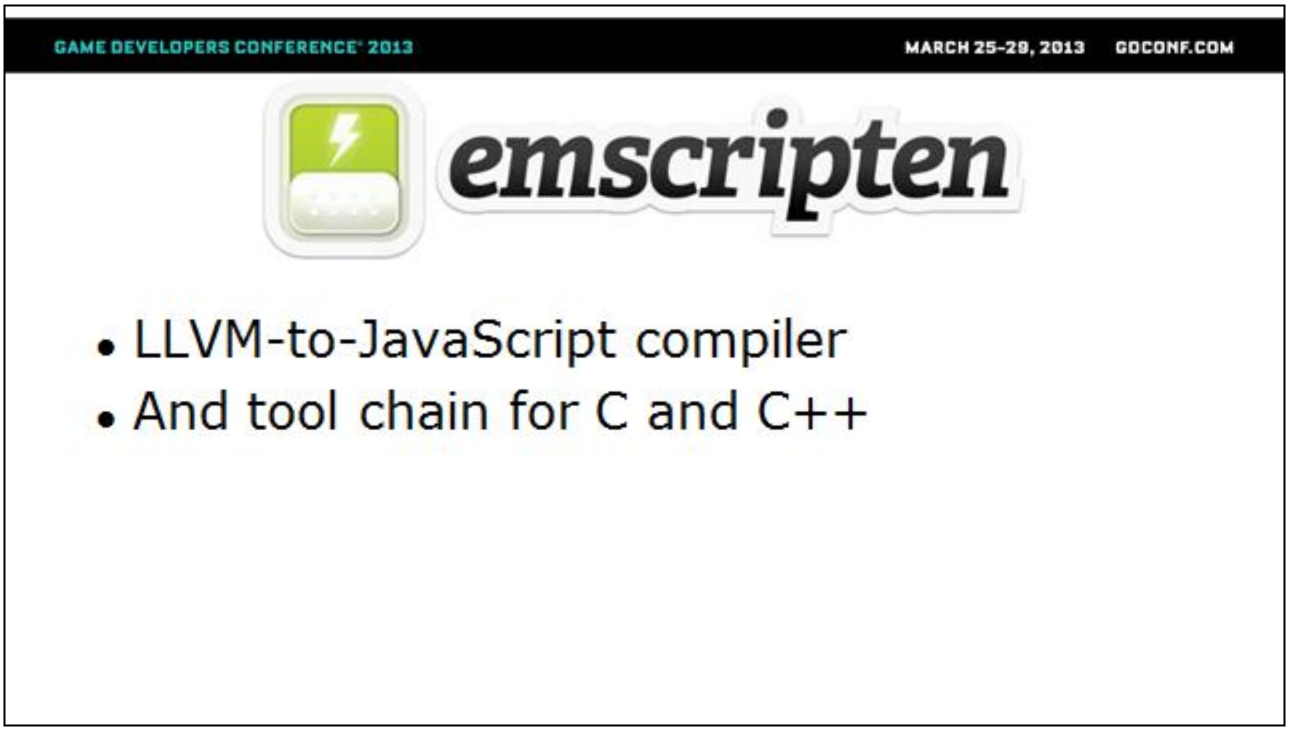


# Multiplatform C++ on the Web with Emscripten

**Chad Austin**
Technical Director, IMVU

GAME DEVELOPERS CONFERENCE
SAN FRANCISCO, CA
MARCH 25-29, 2013
EXPO DATES: MARCH 27-29
2013

Hi, my name is Chad Austin, and I'm here to tell you about what led IMVU to select Emscripten as a key part of our strategy to have a single C++ engine across all platforms, including web browsers.



GAME DEVELOPERS CONFERENCE® 2013 MARCH 25-29, 2013 GDCONF.COM

## emscripten

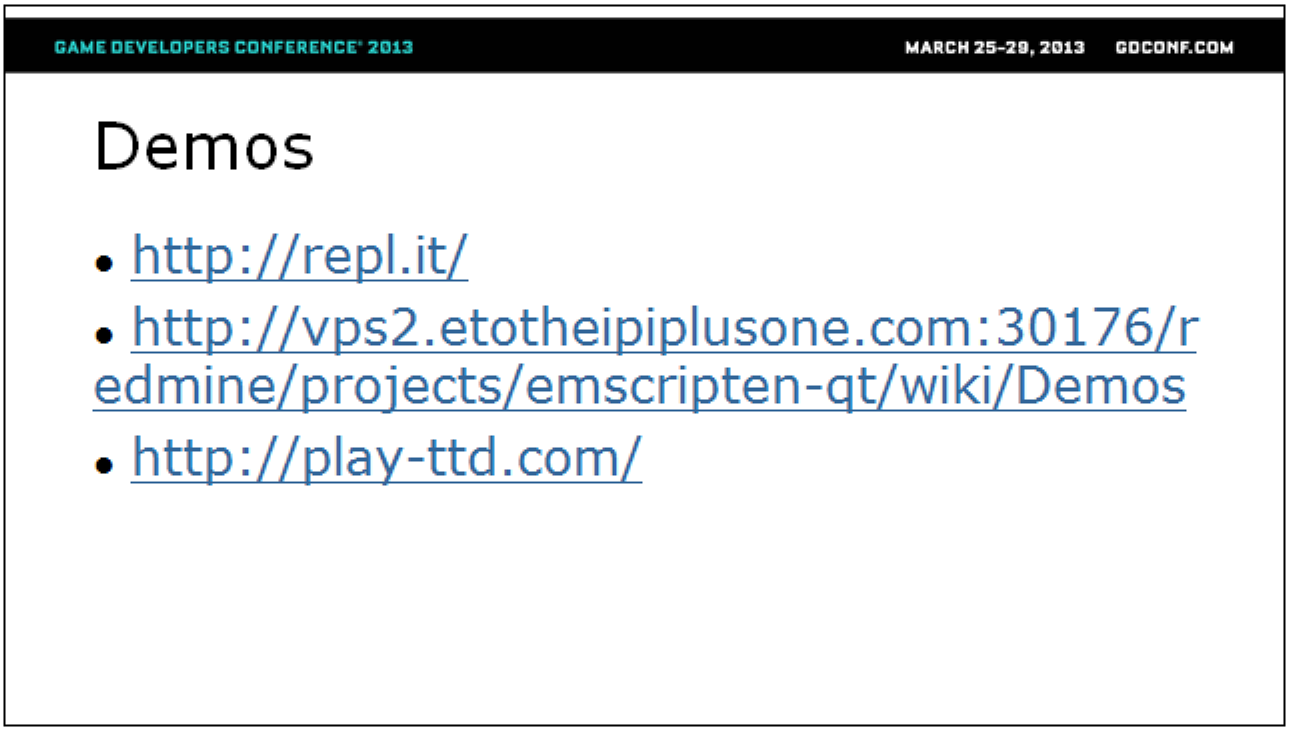
- LLVM-to-JavaScript compiler
- And tool chain for C and C++

LLVM is an open source compiler infrastructure, mostly commonly used as the foundation of the clang C/C++ compiler.

Emscripten translates LLVM into JavaScript suitable for web browsers.

## emscripten

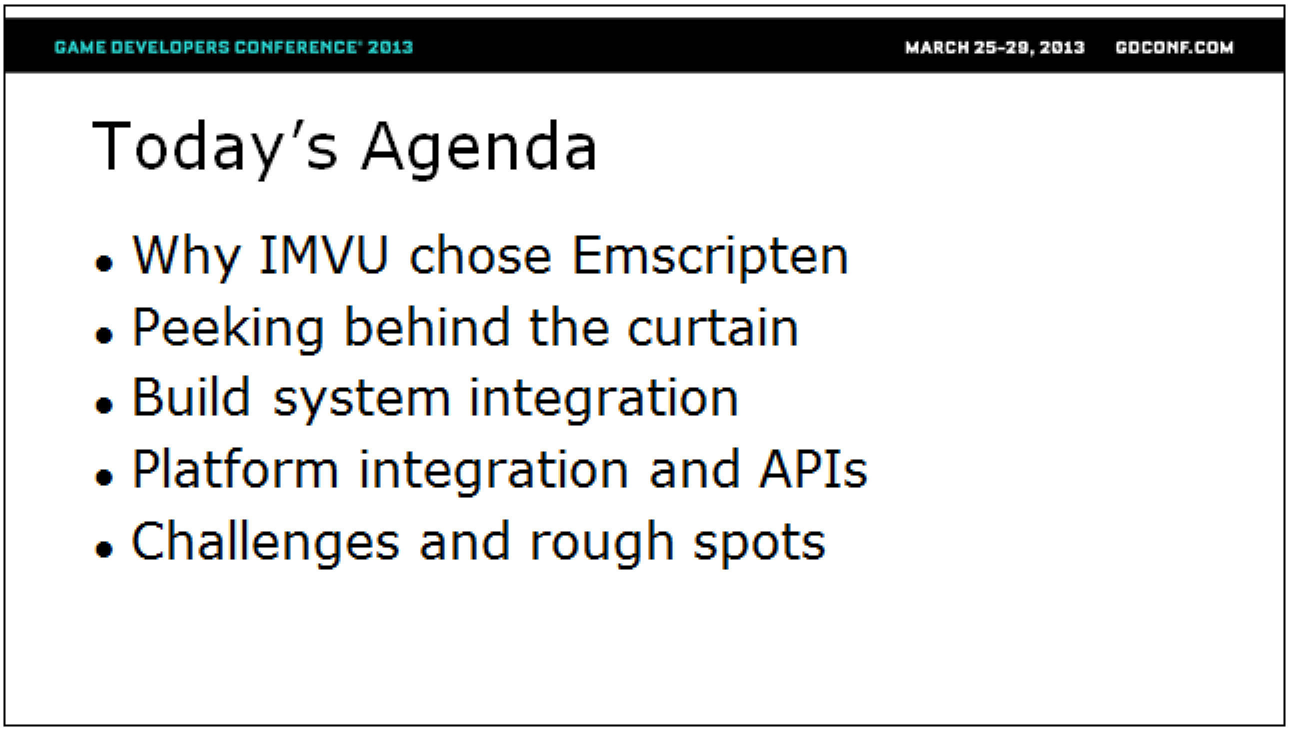
- Developed by Alon Zakai
- Now funded by Mozilla
- http://emscripten.org



GAME DEVELOPERS CONFERENCE® 2013 MARCH 25-29, 2013 GDCONF.COM

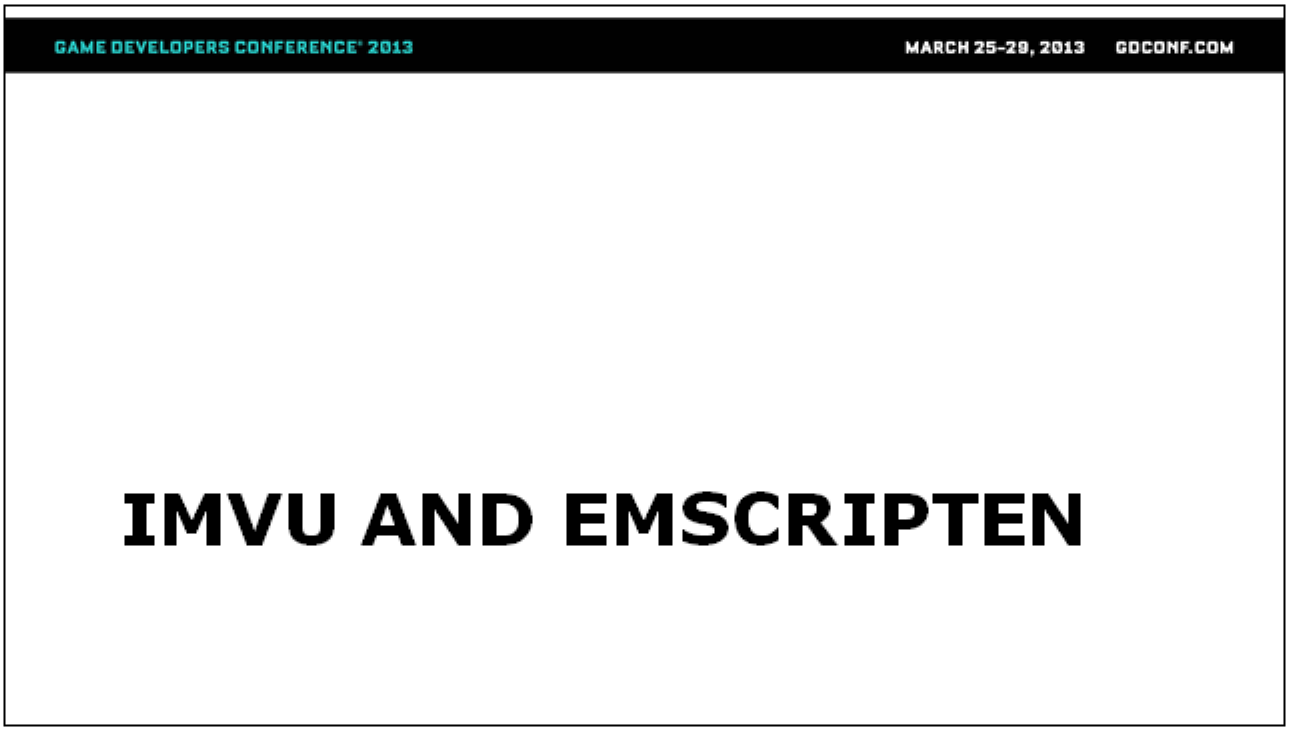
## Demos

- http://repl.it/
- http://vps2.etotheipiplusone.com:30176/redmine/projects/emscripten-qt/wiki/Demos
- http://play-ttd.com/

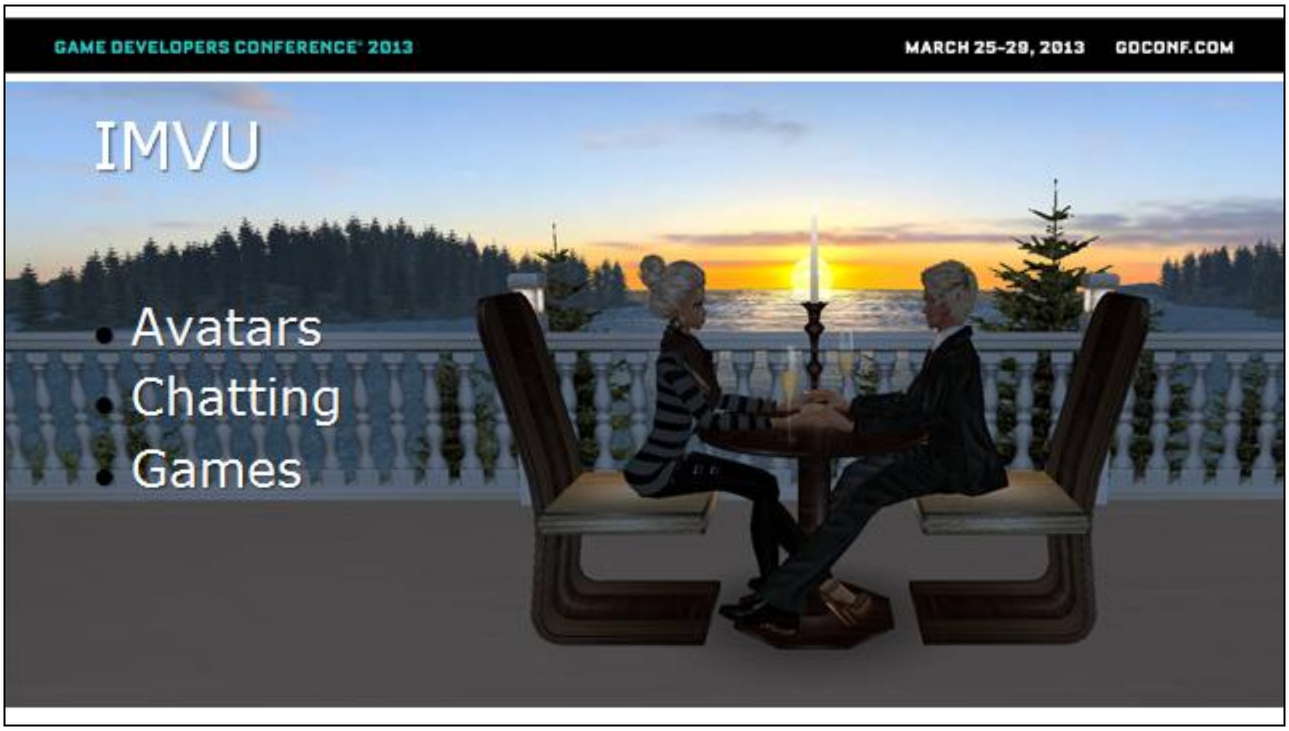


## Today's Agenda

- Why IMVU chose Emscripten
- Peeking behind the curtain
- Build system integration
- Platform integration and APIs
- Challenges and rough spots



## IMVU AND EMSCRIPTEN



GAME DEVELOPERS CONFERENCE® 2013 MARCH 25-29, 2013 GDCONF.COM

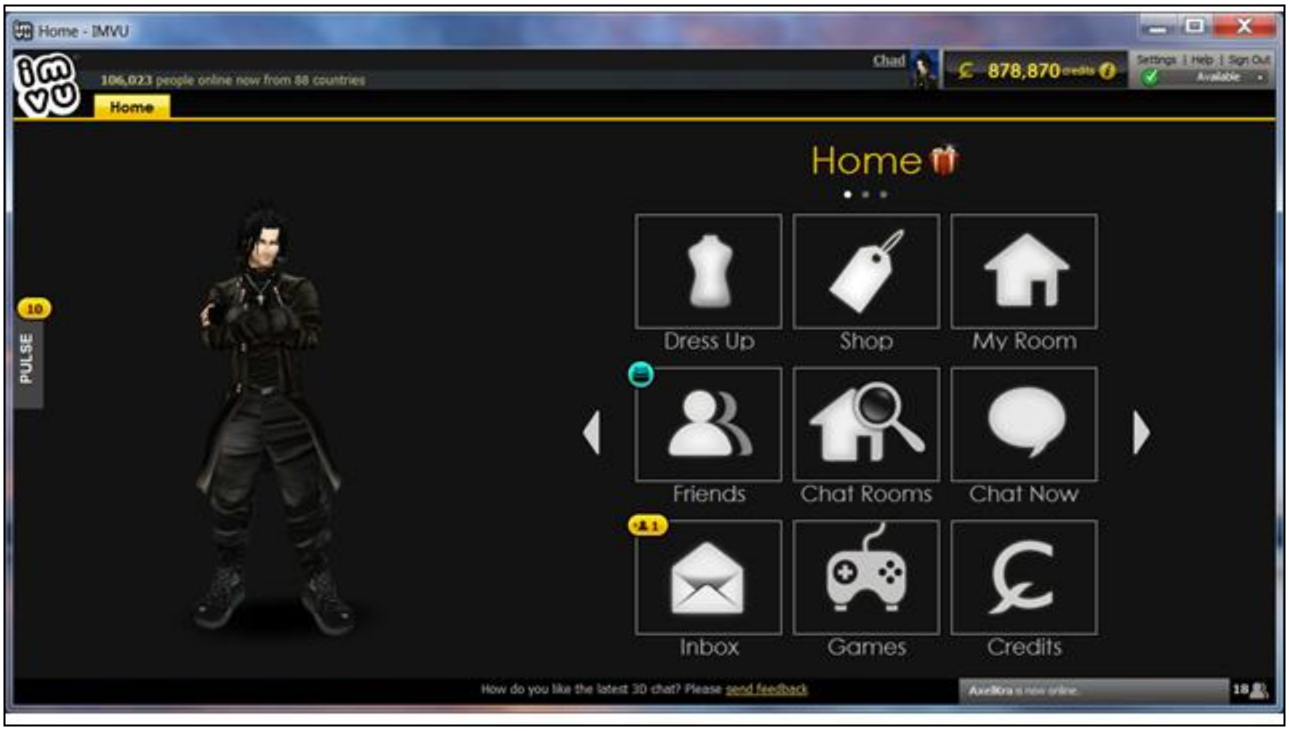
## IMVU

- Avatars
- Chatting
- Games

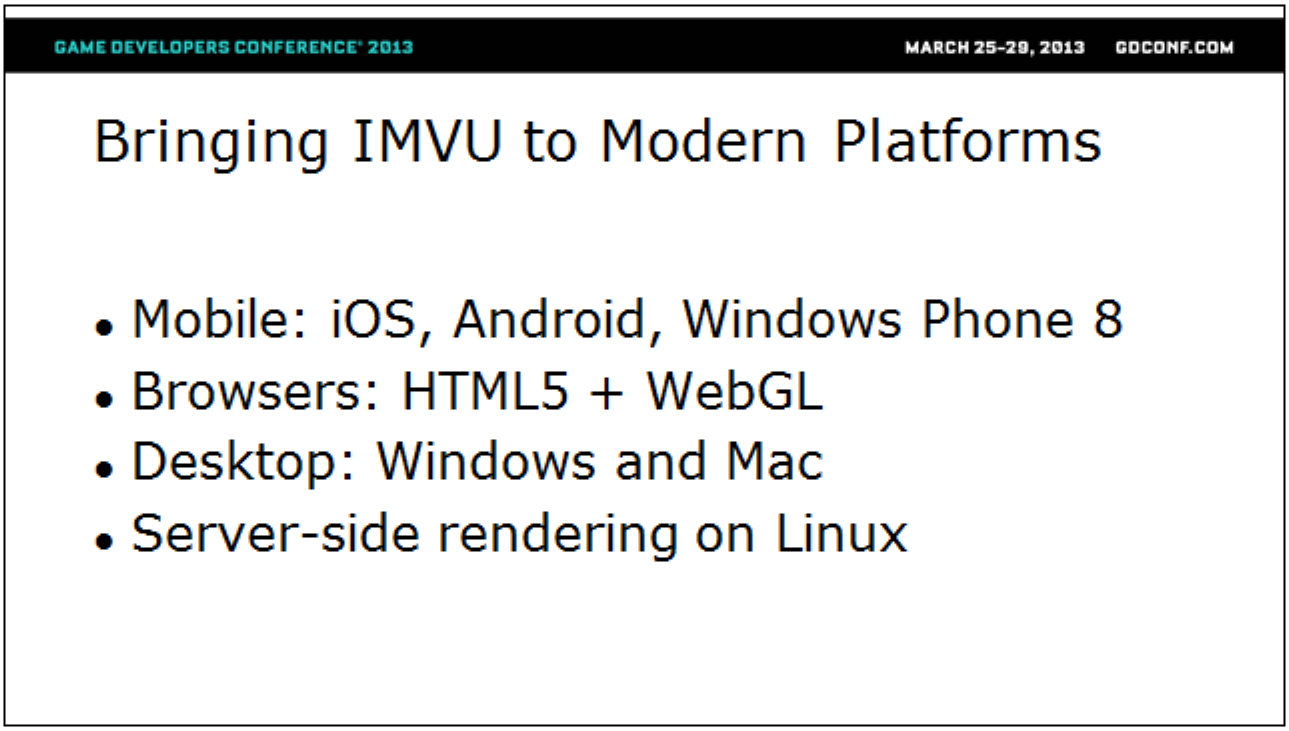
IMVU is a place where members use 3D avatars to meet new people, chat, and play games.

## IMVU

- 14M virtual goods
- 16,000 new items / day

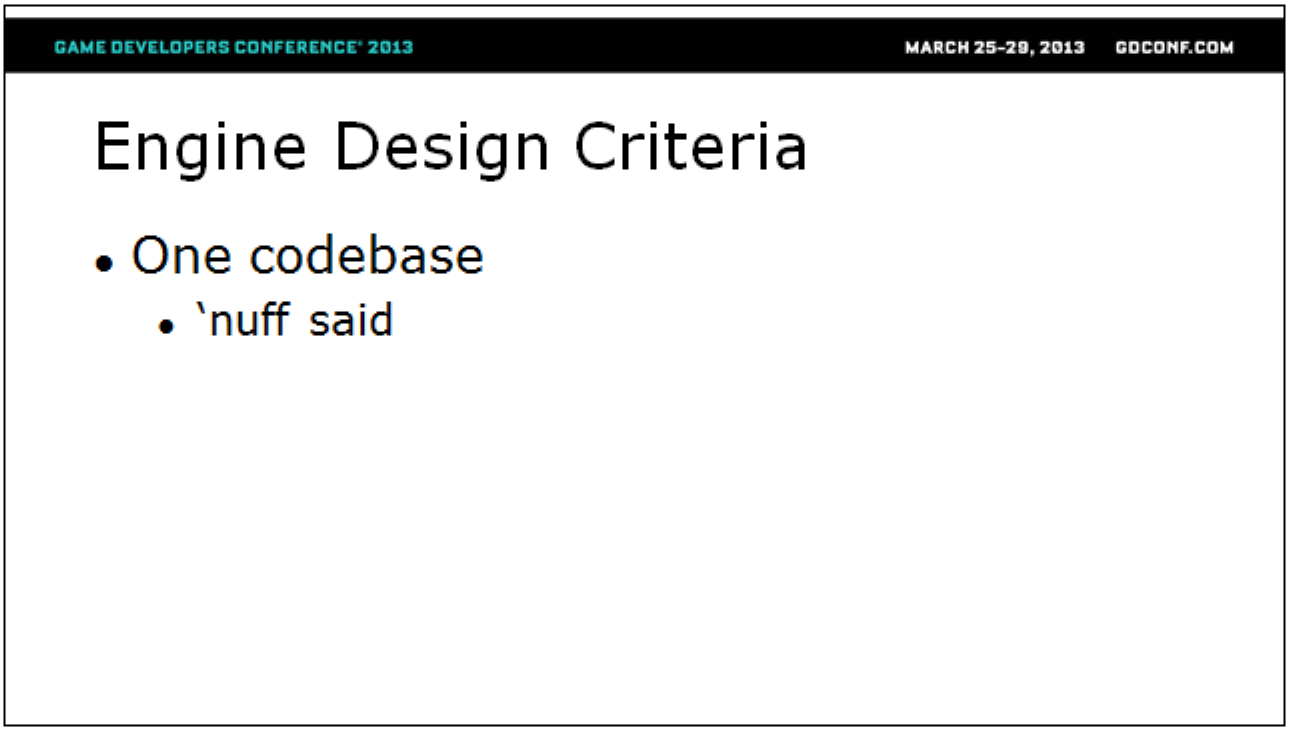


Today IMVU is a downloadable application for Windows and Mac.



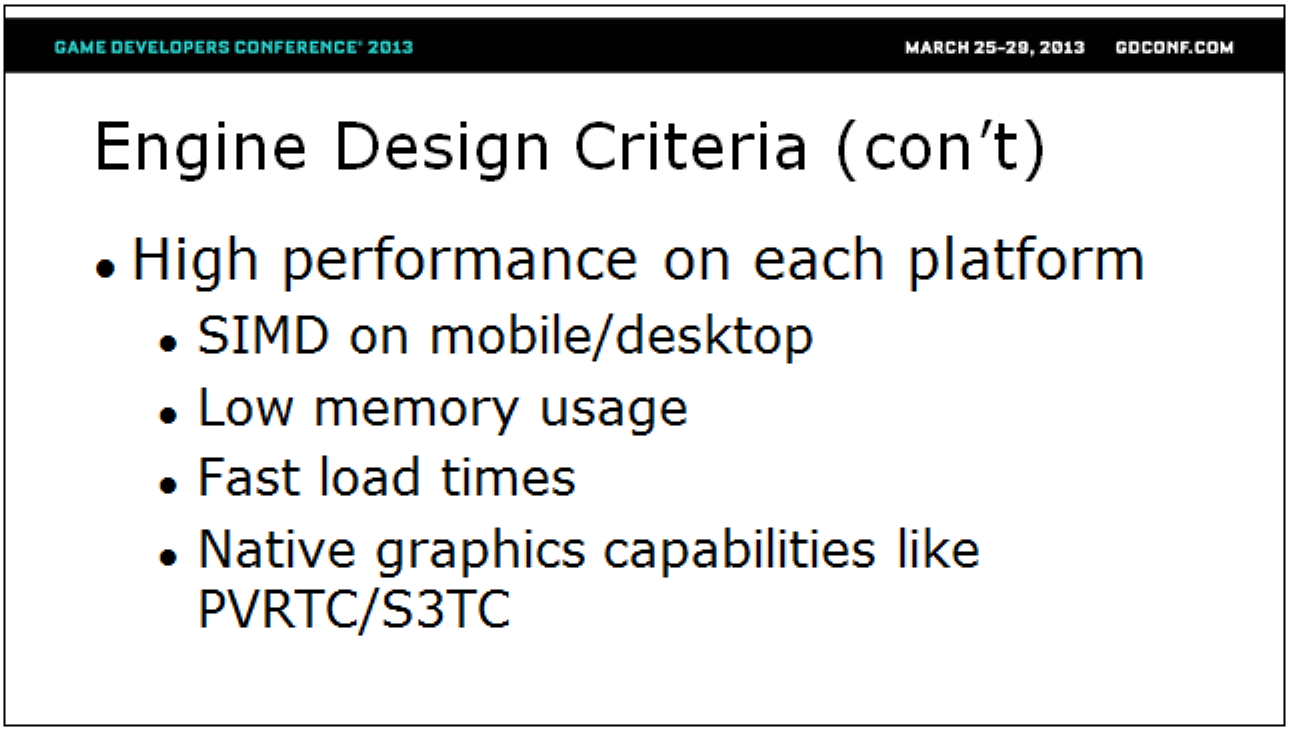
## Bringing IMVU to Modern Platforms

- Mobile: iOS, Android, Windows Phone 8
- Browsers: HTML5 + WebGL
- Desktop: Windows and Mac
- Server-side rendering on Linux



## Engine Design Criteria

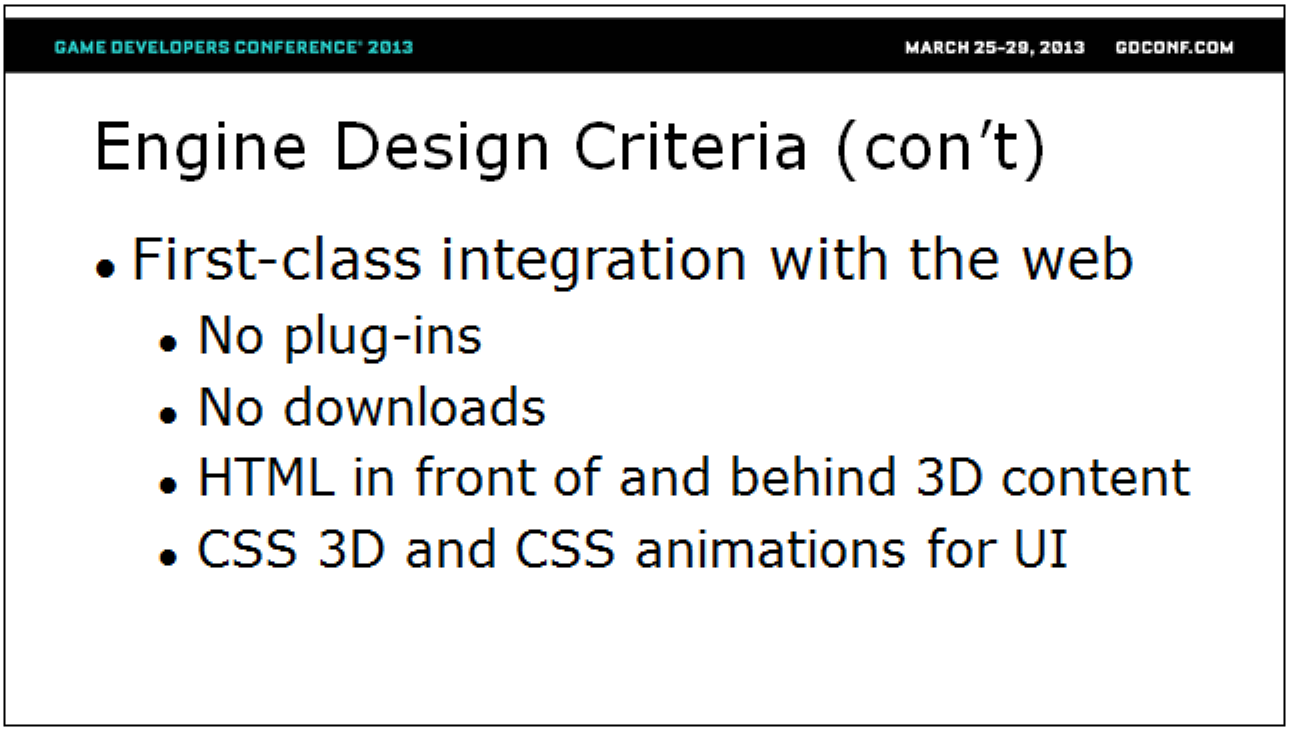
- One codebase
  - ‘nuff said



## Engine Design Criteria (con't)

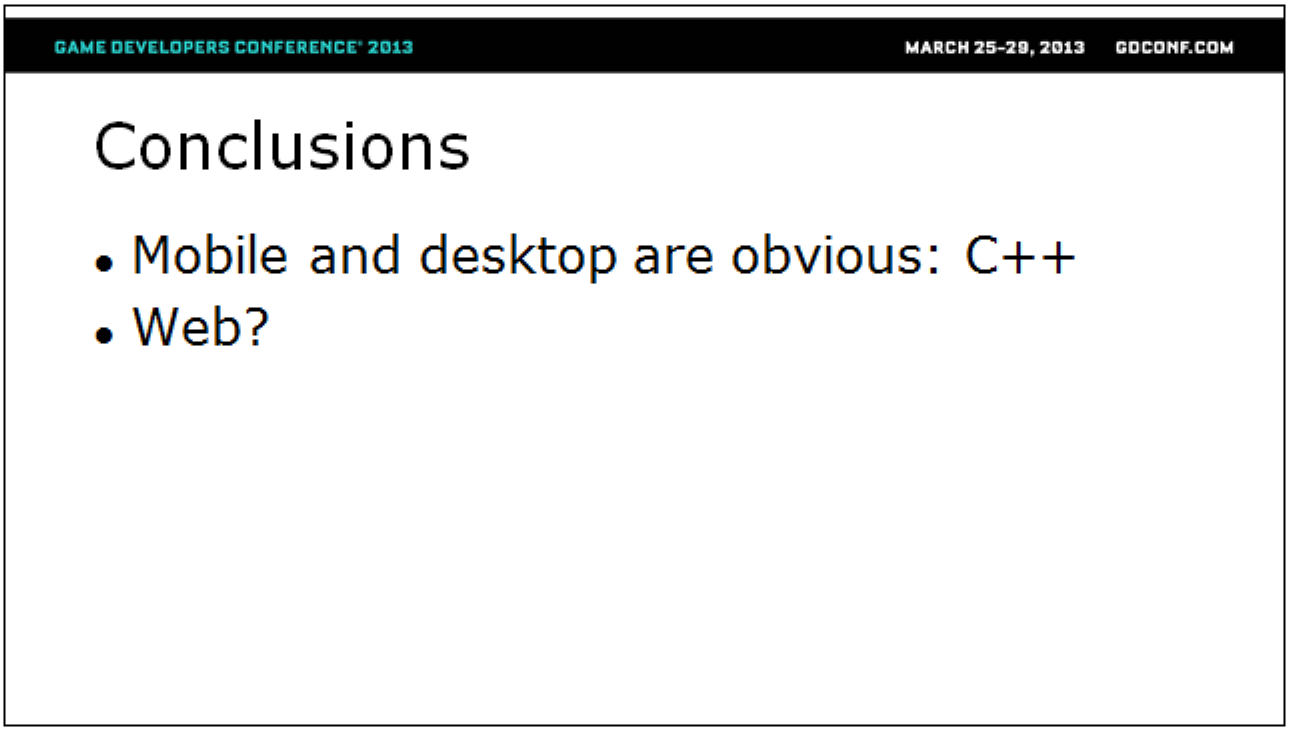
- High performance on each platform
  - SIMD on mobile/desktop
  - Low memory usage
  - Fast load times
  - Native graphics capabilities like PVRTC/S3TC

Making best use of the hardware is key on mobile devices.



## Engine Design Criteria (con't)

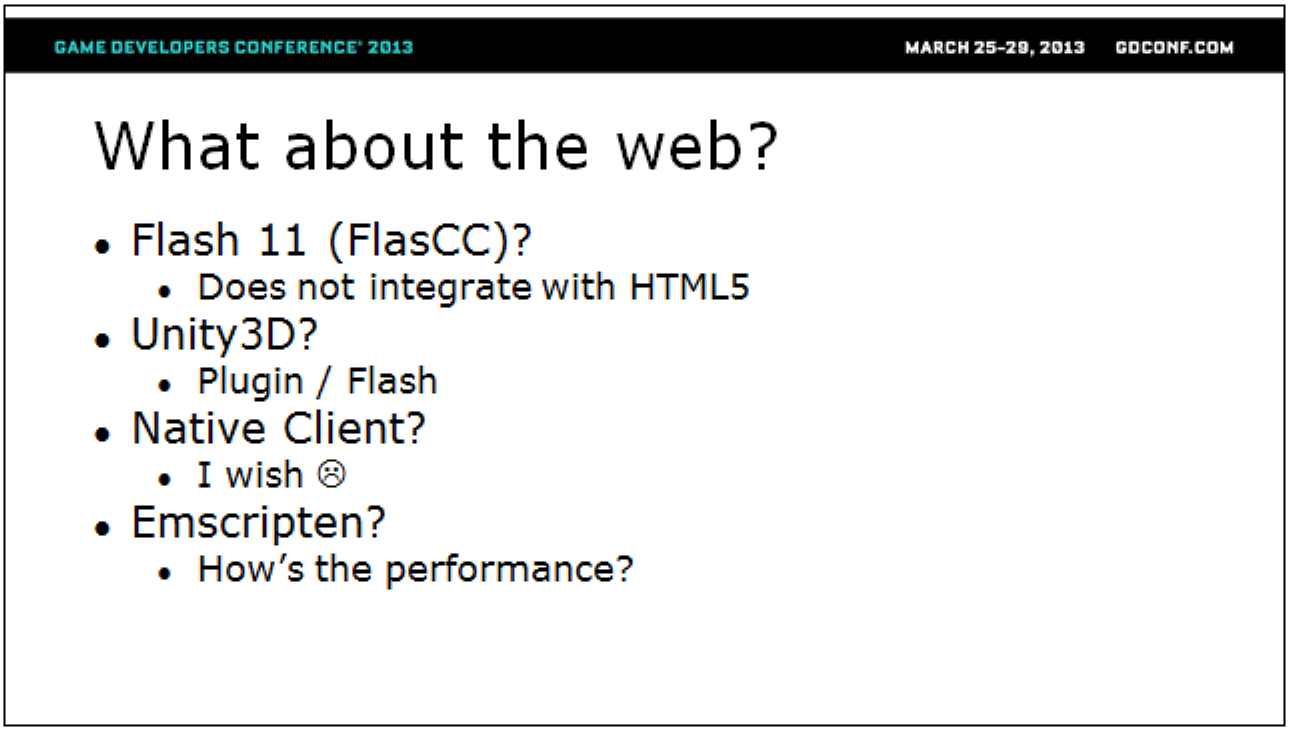
- First-class integration with the web
  - No plug-ins
  - No downloads
  - HTML in front of and behind 3D content
  - CSS 3D and CSS animations for UI



GAME DEVELOPERS CONFERENCE® 2013 MARCH 25-29, 2013 GDCONF.COM

## Conclusions

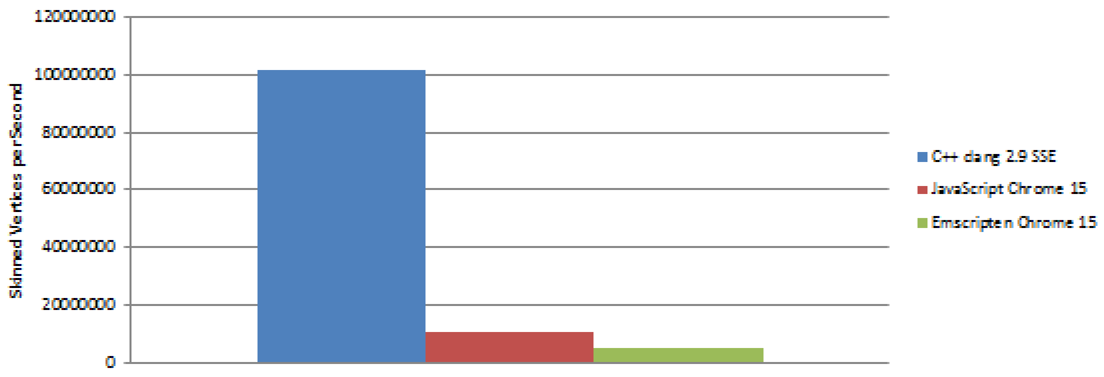
- Mobile and desktop are obvious: C++
- Web?

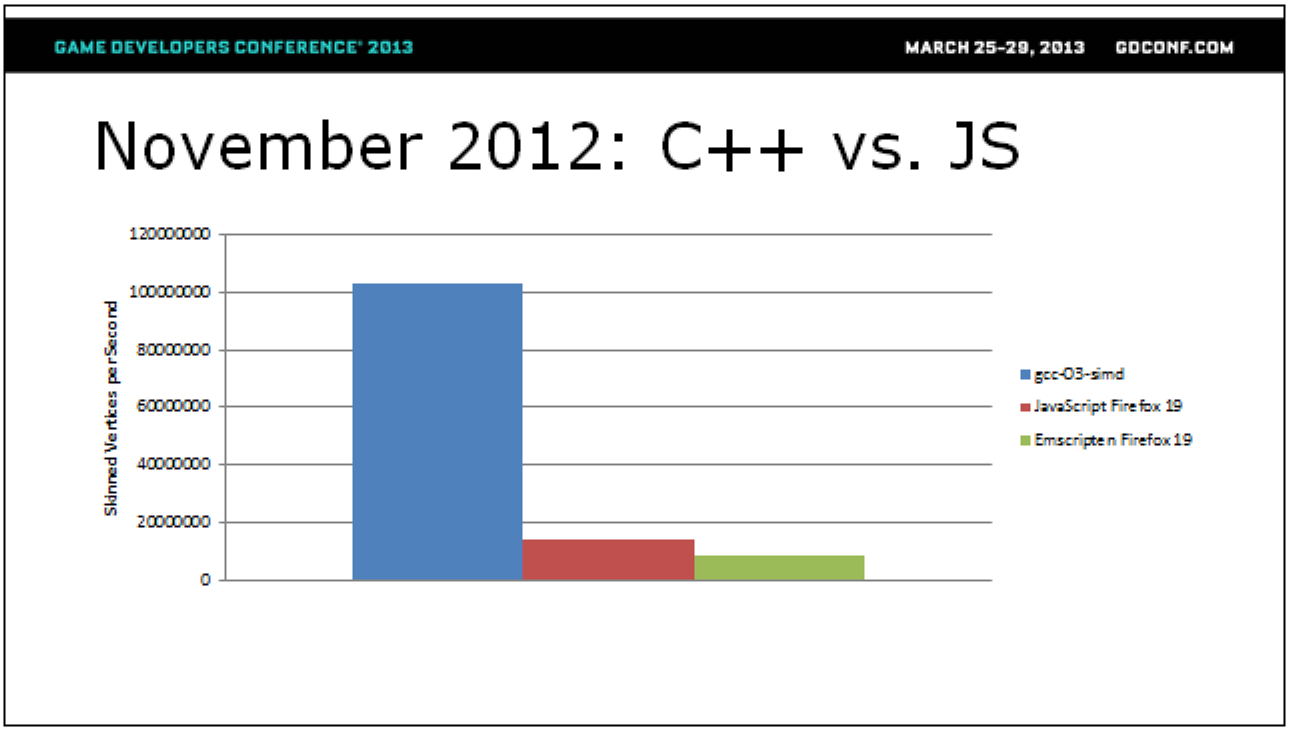


## What about the web?

- Flash 11 (FlasCC)?
  - Does not integrate with HTML5
- Unity3D?
  - Plugin / Flash
- Native Client?
  - I wish ☹
- Emscripten?
  - How's the performance?

## November 2011: C++ vs. JS

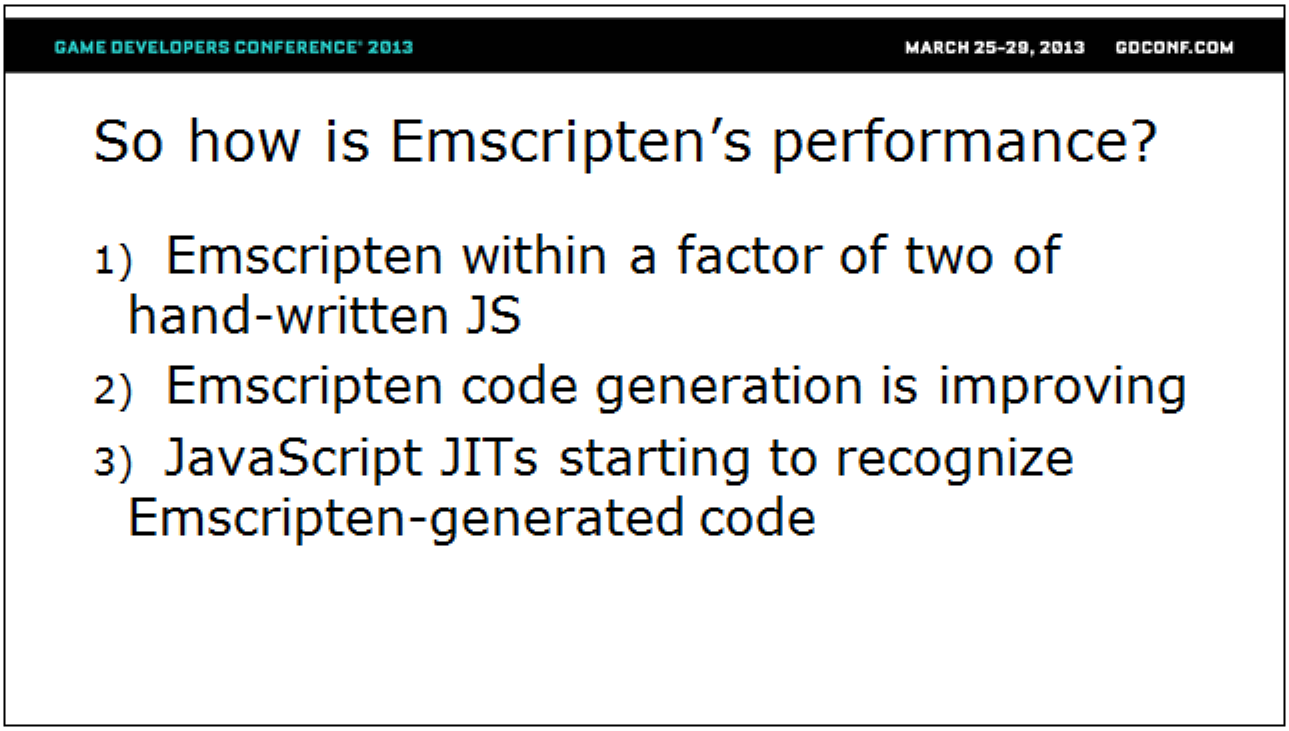




GAME DEVELOPERS CONFERENCE® 2013 MARCH 25-29, 2013 GDCONF.COM

## November 2012: C++ vs. JS

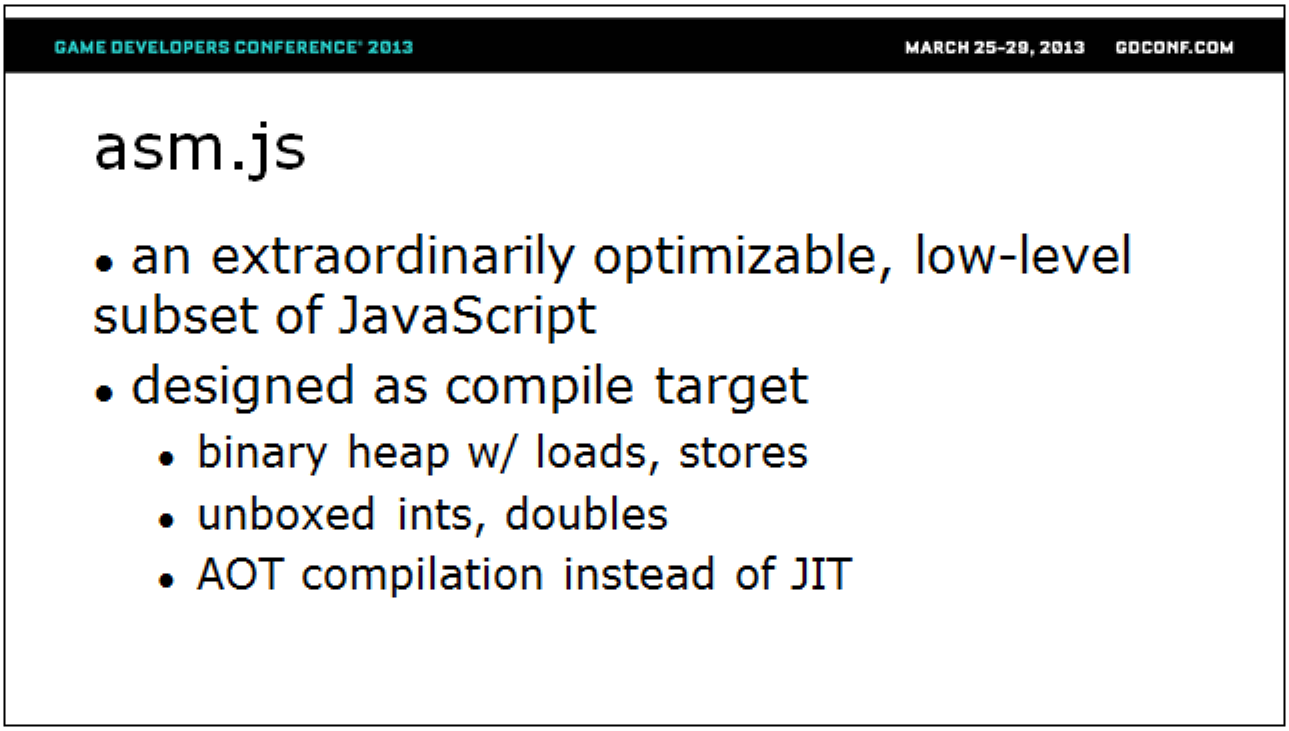
My testing methodology: run the benchmark in all compilers with all optimization settings and pick the fastest.


GAME DEVELOPERS CONFERENCE® 2013 MARCH 25-29, 2013 GDCONF.COM

## So how is Emscripten's performance?

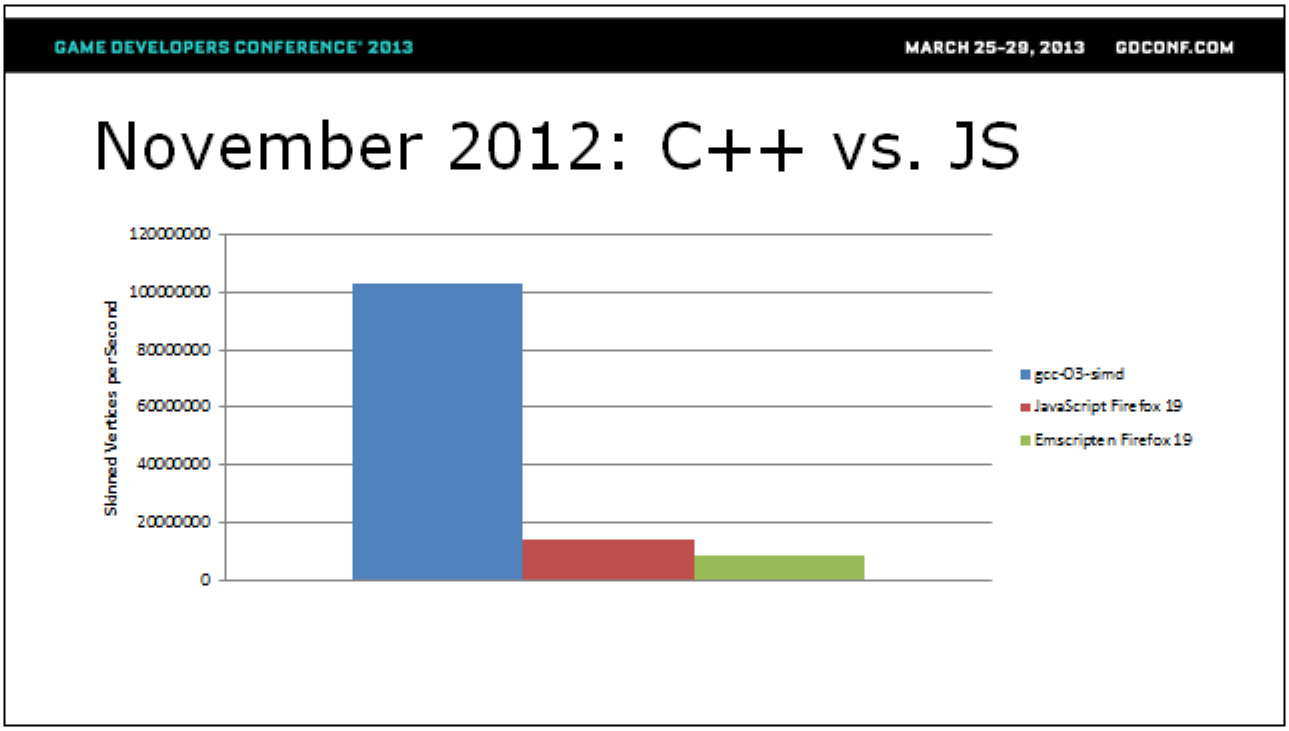
1) Emscripten within a factor of two of hand-written JS
2) Emscripten code generation is improving
3) JavaScript JITs starting to recognize Emscripten-generated code

At this point, I'm willing to pay a reduction in performance in order to have a single C++ engine codebase.



## asm.js

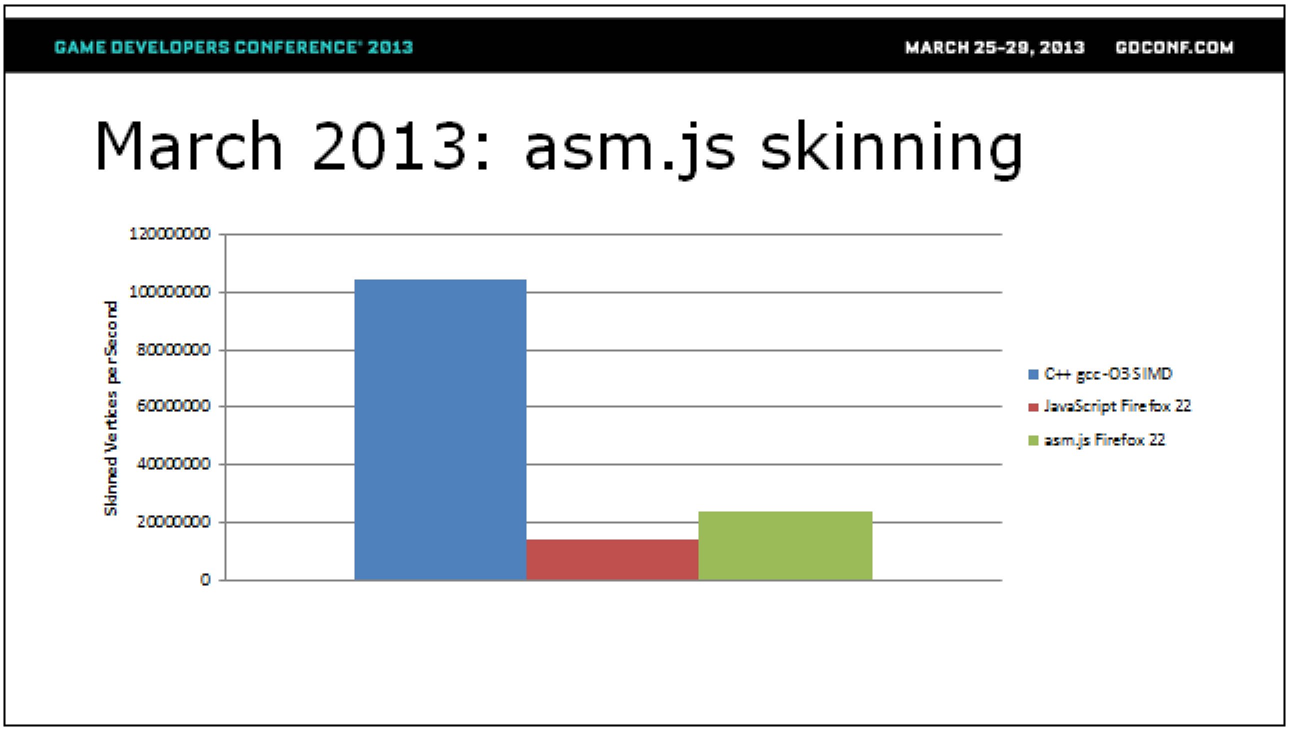
- an extraordinarily optimizable, low-level subset of JavaScript
- designed as compile target
  - binary heap w/ loads, stores
  - unboxed ints, doubles
  - AOT compilation instead of JIT



GAME DEVELOPERS CONFERENCE® 2013 MARCH 25–29, 2013 GDCONF.COM

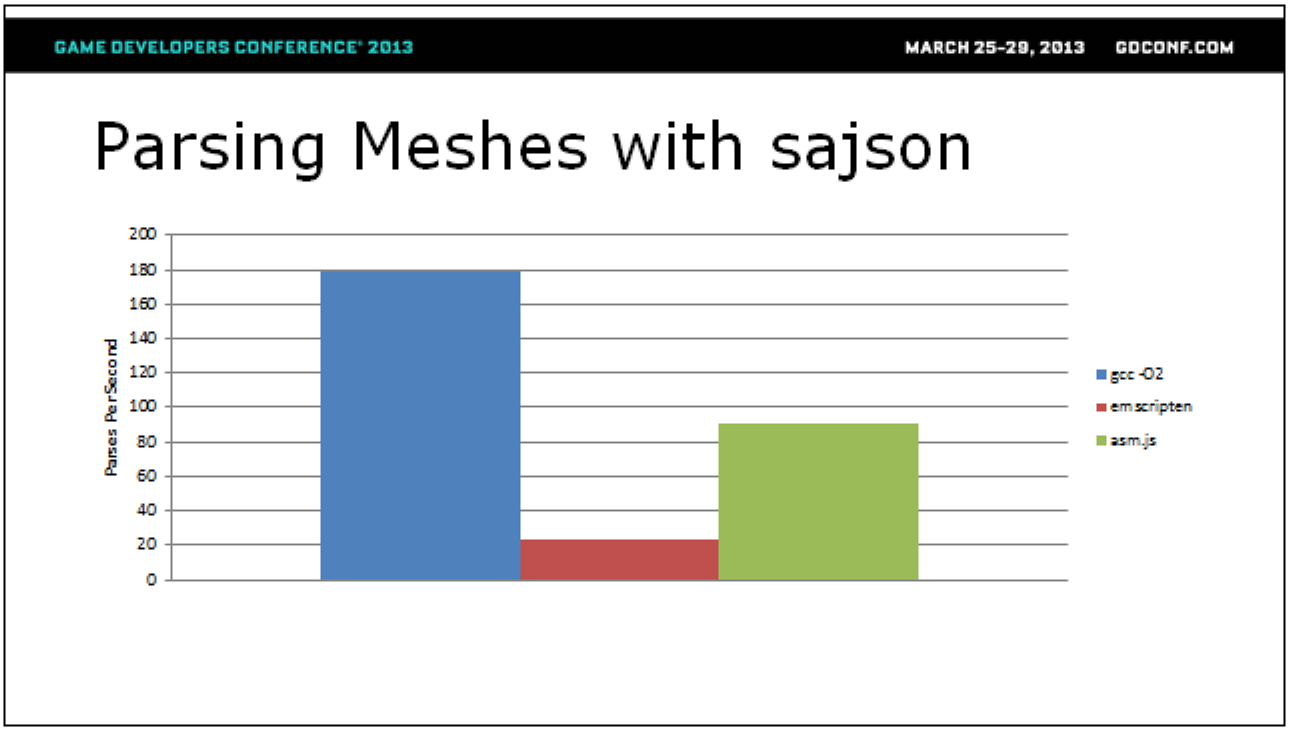
## November 2012: C++ vs. JS

Same benchmark as before for comparison purposes.



## March 2013: asm.js skinning

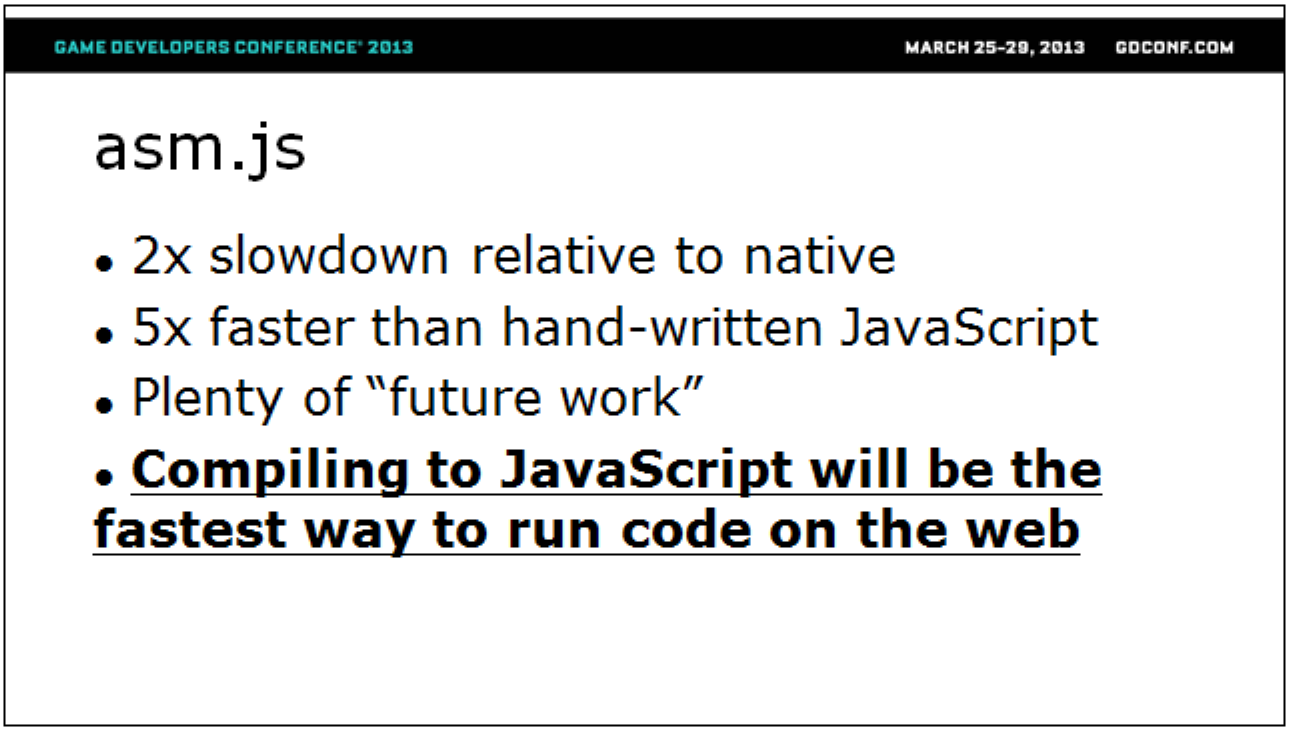
asm.js is a big win. (But lack of SIMD makes it still slower than native.)



GAME DEVELOPERS CONFERENCE® 2013 MARCH 25-29, 2013 GDCONF.COM

## Parsing Meshes with sajson

So let's look at a benchmark that doesn't rely so much on SIMD.

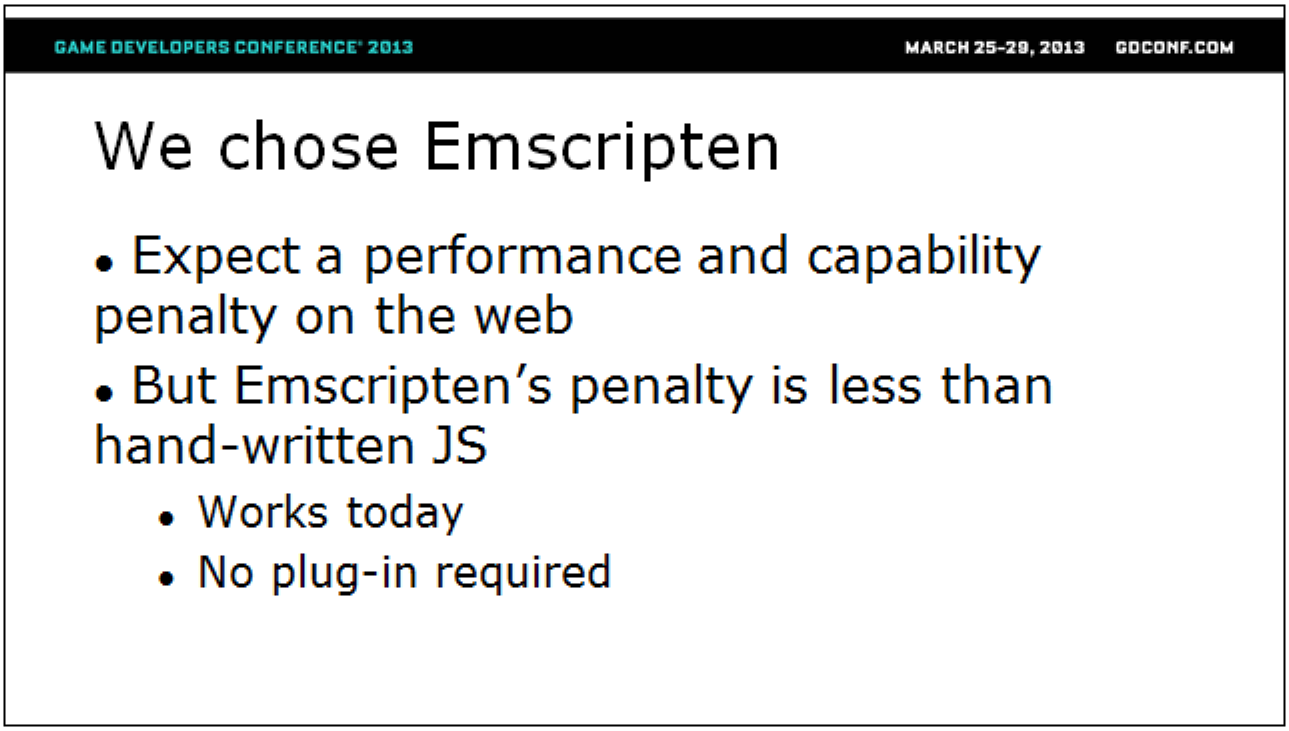


## asm.js

- 2x slowdown relative to native
- 5x faster than hand-written JavaScript
- Plenty of "future work"
- **Compiling to JavaScript will be the fastest way to run code on the web**

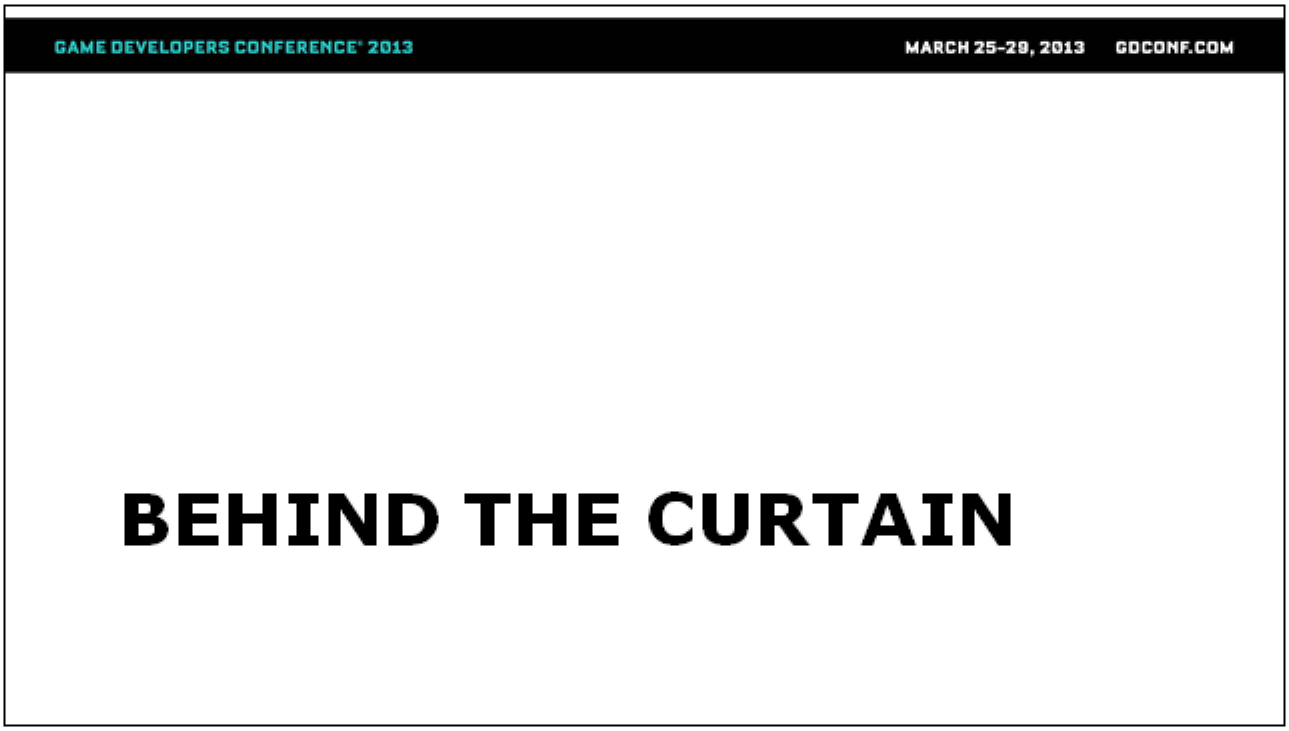
## Browser Support

- Emscripten runs on any browser with typed arrays
  - Firefox, Chrome, Safari, Opera
  - IE10+

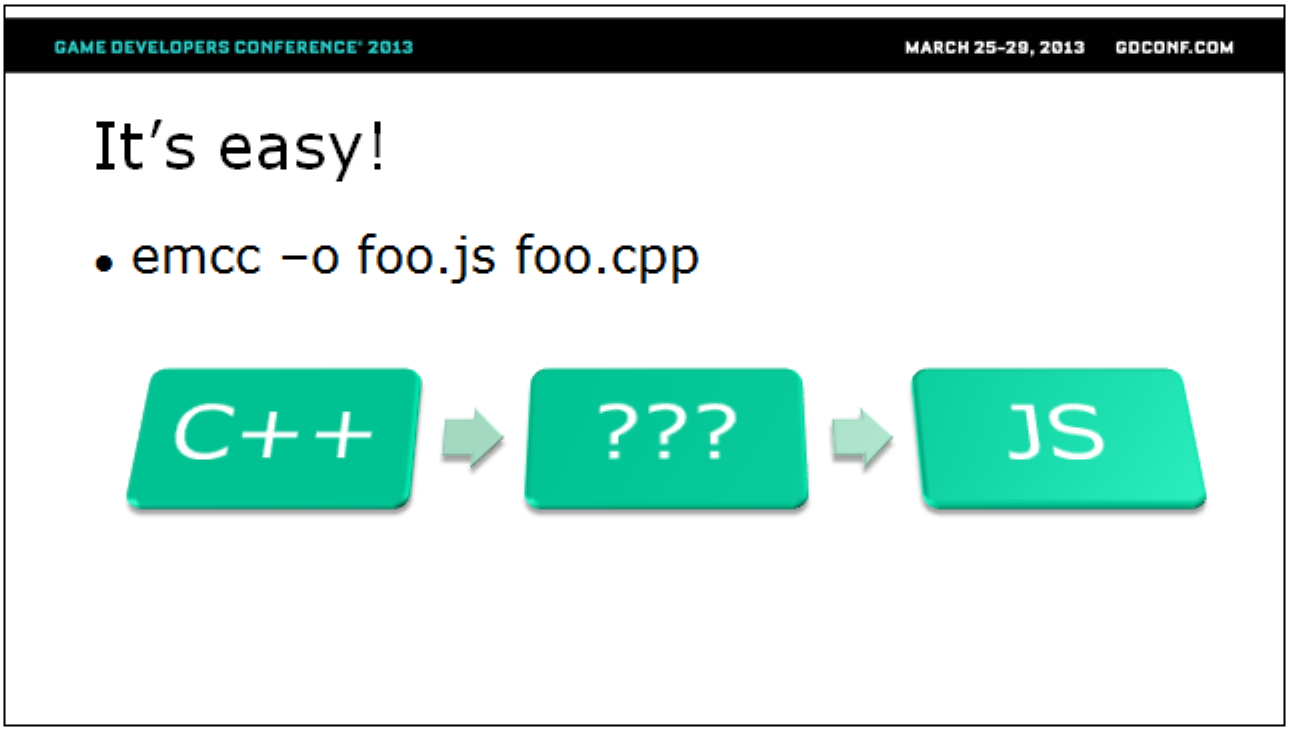


## We chose Emscripten

- Expect a performance and capability penalty on the web
- But Emscripten's penalty is less than hand-written JS
  - Works today
  - No plug-in required



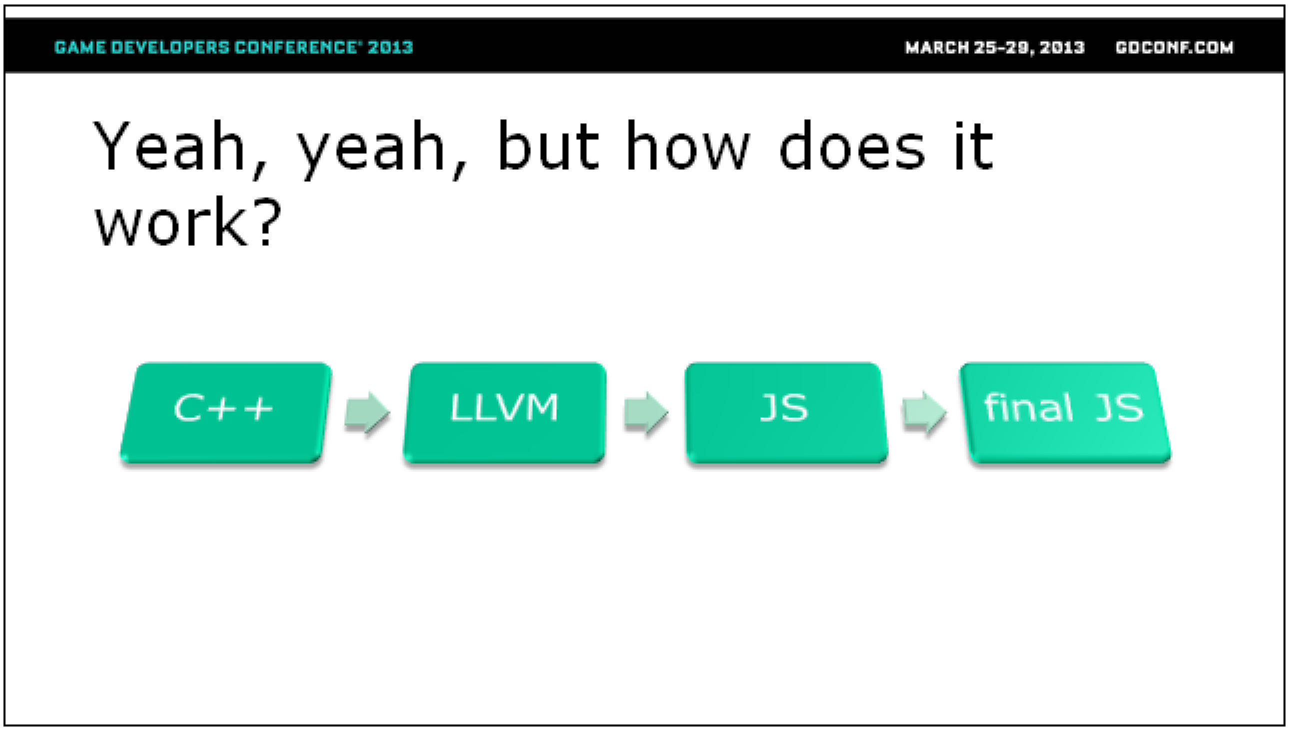
## BEHIND THE CURTAIN



## It’s easy!

- emcc –o foo.js foo.cpp

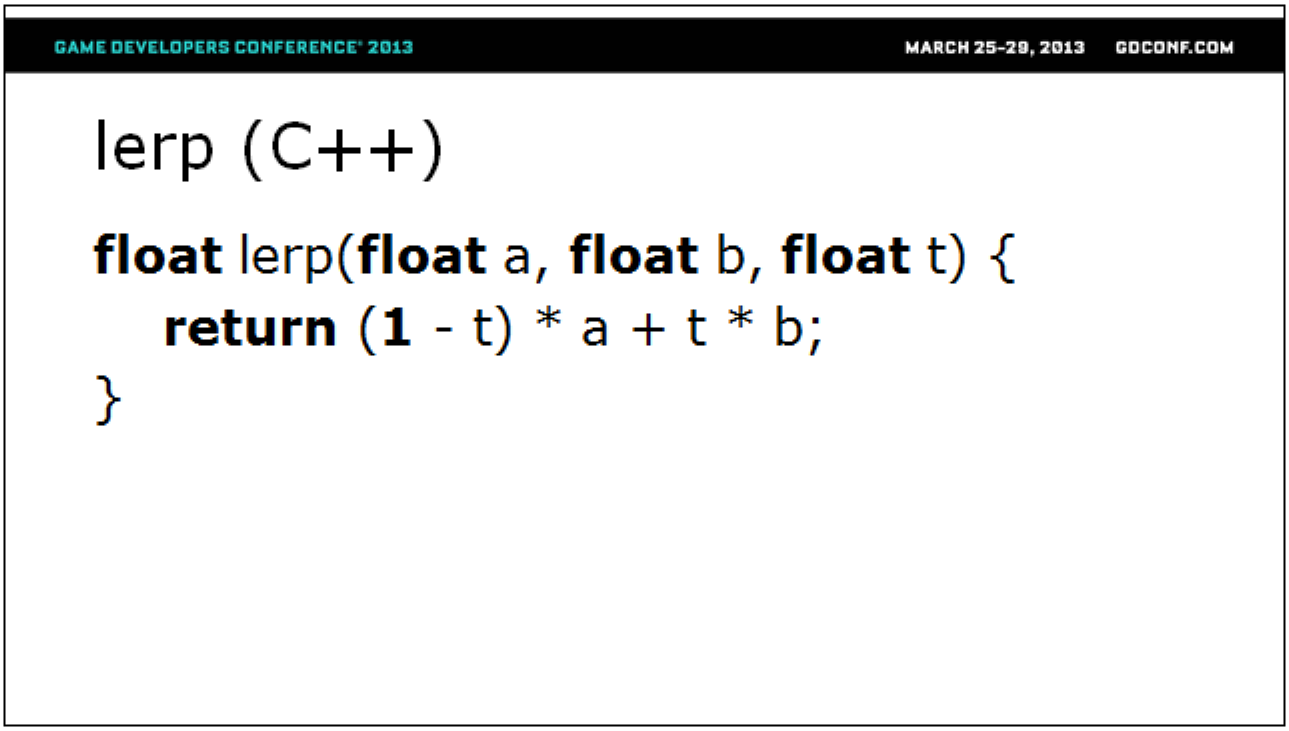
$$\begin{tikzcd}\text{C++} \arrow[r] & \text{???} \arrow[r] & \text{JS}\end{tikzcd}$$



GAME DEVELOPERS CONFERENCE® 2013 MARCH 25-29, 2013 GDCONF.COM

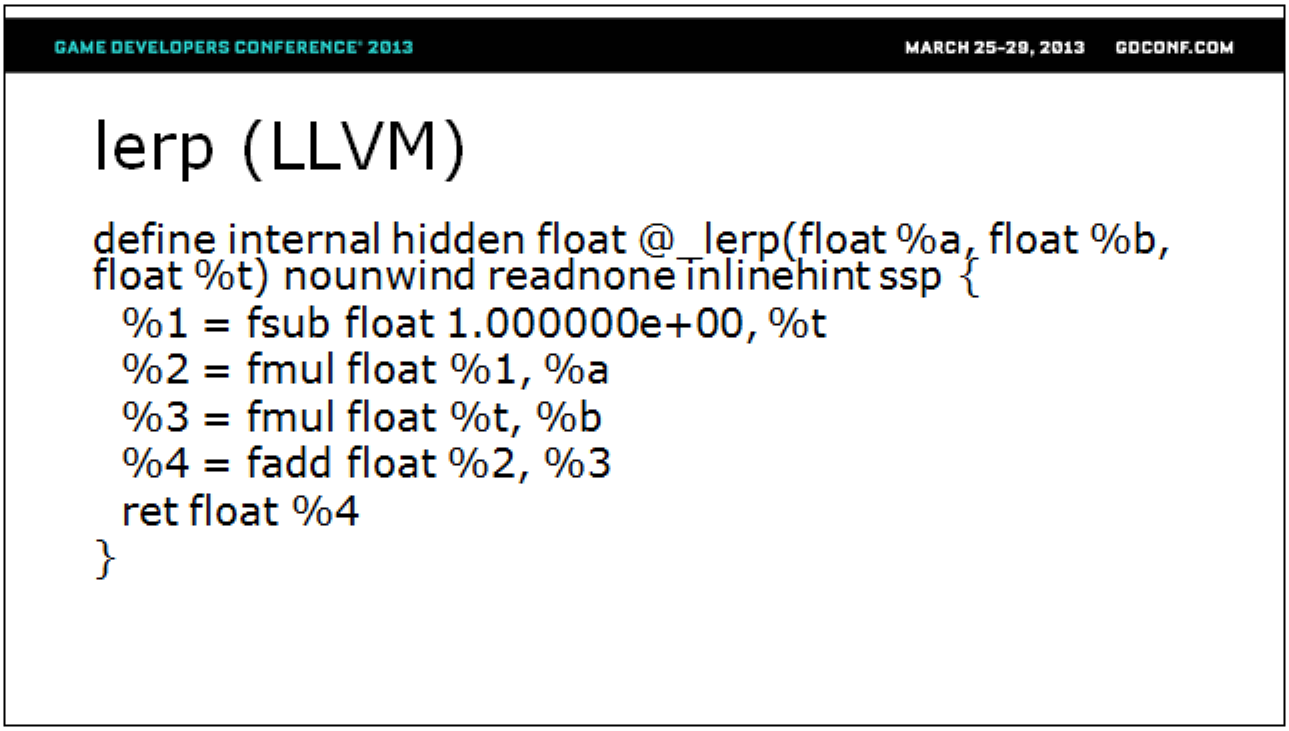
## Yeah, yeah, but how does it work?

$$\begin{tikzcd}\text{C++} \arrow[r] & \text{LLVM} \arrow[r] & \text{JS} \arrow[r] & \text{final JS}\end{tikzcd}$$



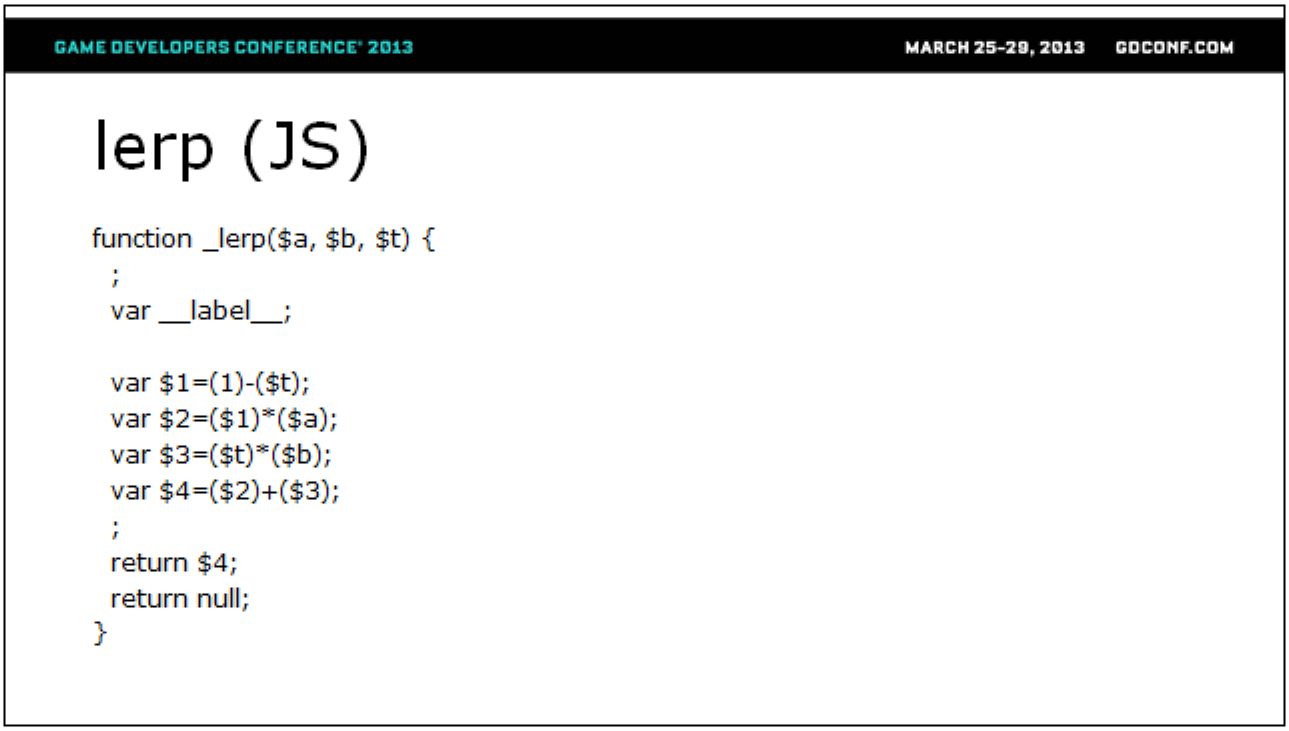
## lerp (C++)

```
float lerp(float a, float b, float t) {
    return (1 - t) * a + t * b;
}
```



## lerp (LLVM)

```
define internal hidden float @_lerp(float %a, float %b,
float %t) nounwind readnone inlinehint ssp {
  %1 = fsub float 1.000000e+00, %t
  %2 = fmul float %1, %a
  %3 = fmul float %t, %b
  %4 = fadd float %2, %3
  ret float %4
}
```

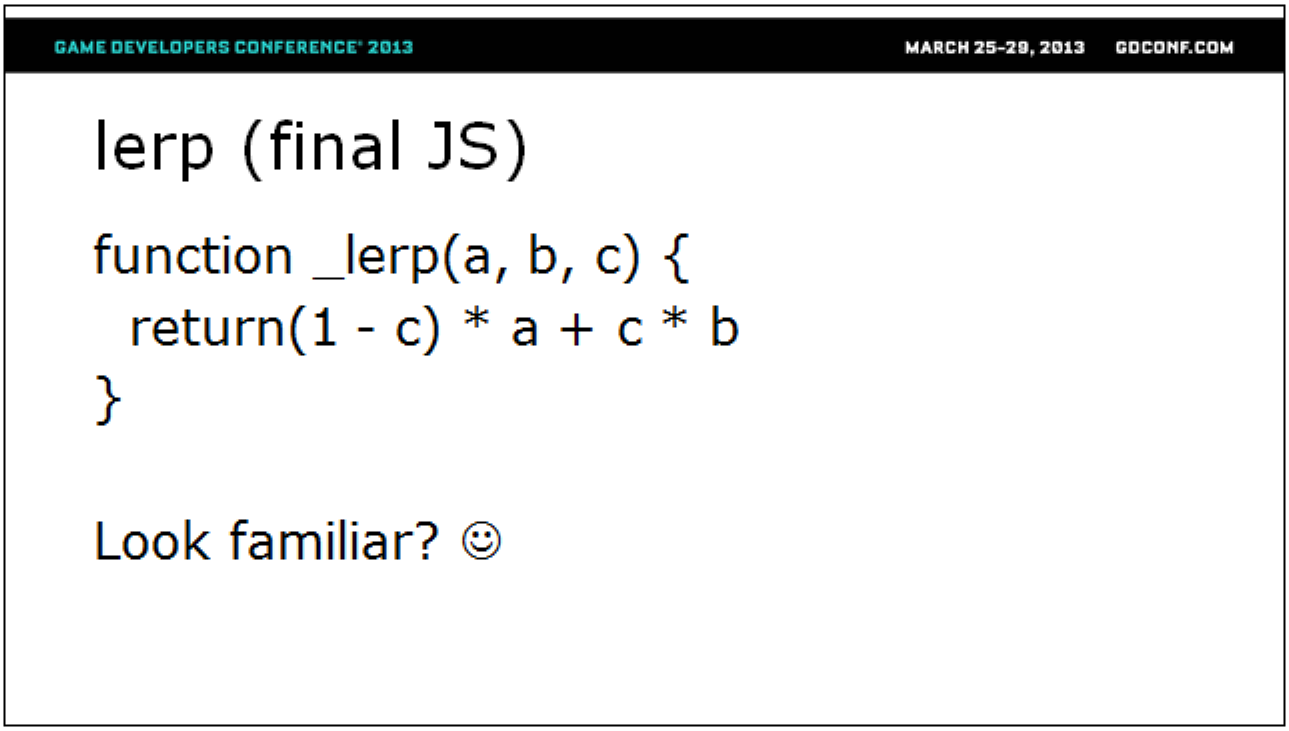

GAME DEVELOPERS CONFERENCE® 2013 MARCH 25-29, 2013 GDCONF.COM

## lerp (JS)

```
function _lerp($a, $b, $t) {
  ;
  var __label__;

  var $1=(1)-($t);
  var $2=($1)*($a);
  var $3=($t)*($b);
  var $4=($2)+($3);
  ;
  return $4;
  return null;
}
```

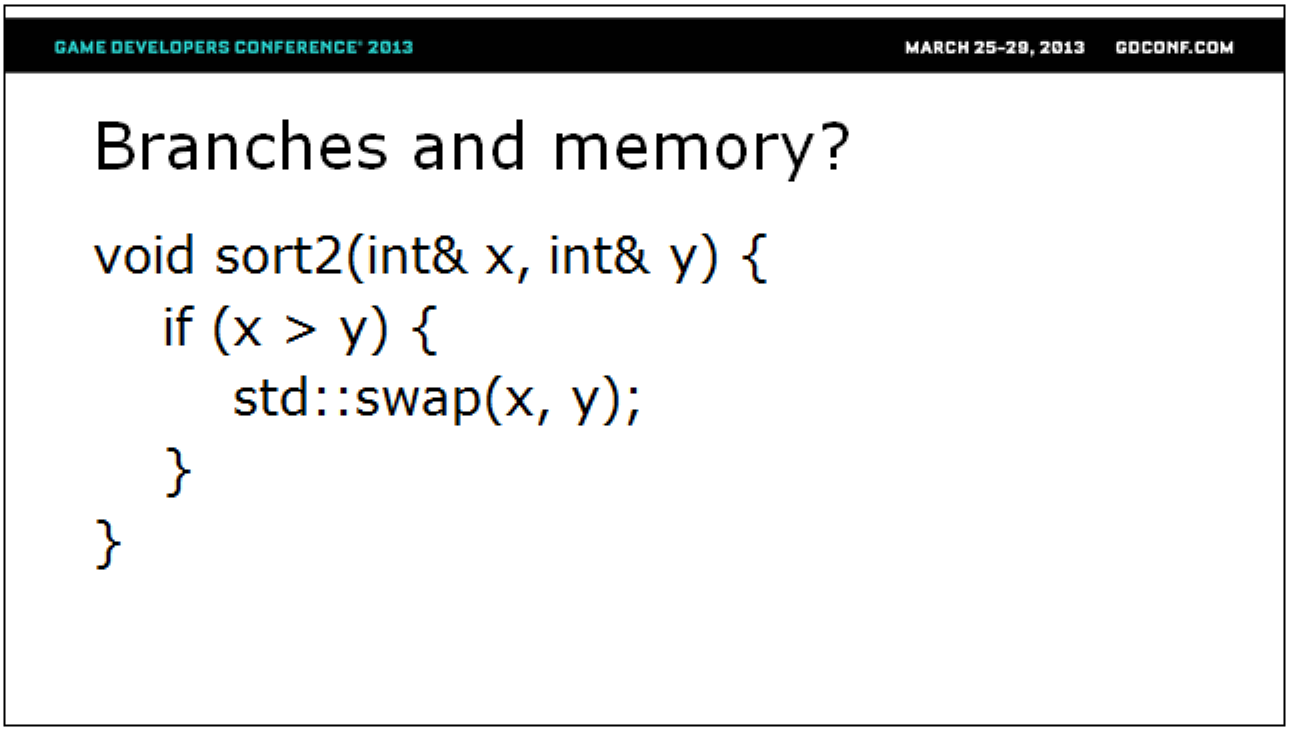
Note the excessive redundancy of the generated code. After Emscripten runs, a series of JavaScript optimizers transform the generated code until it looks like this:



## lerp (final JS)

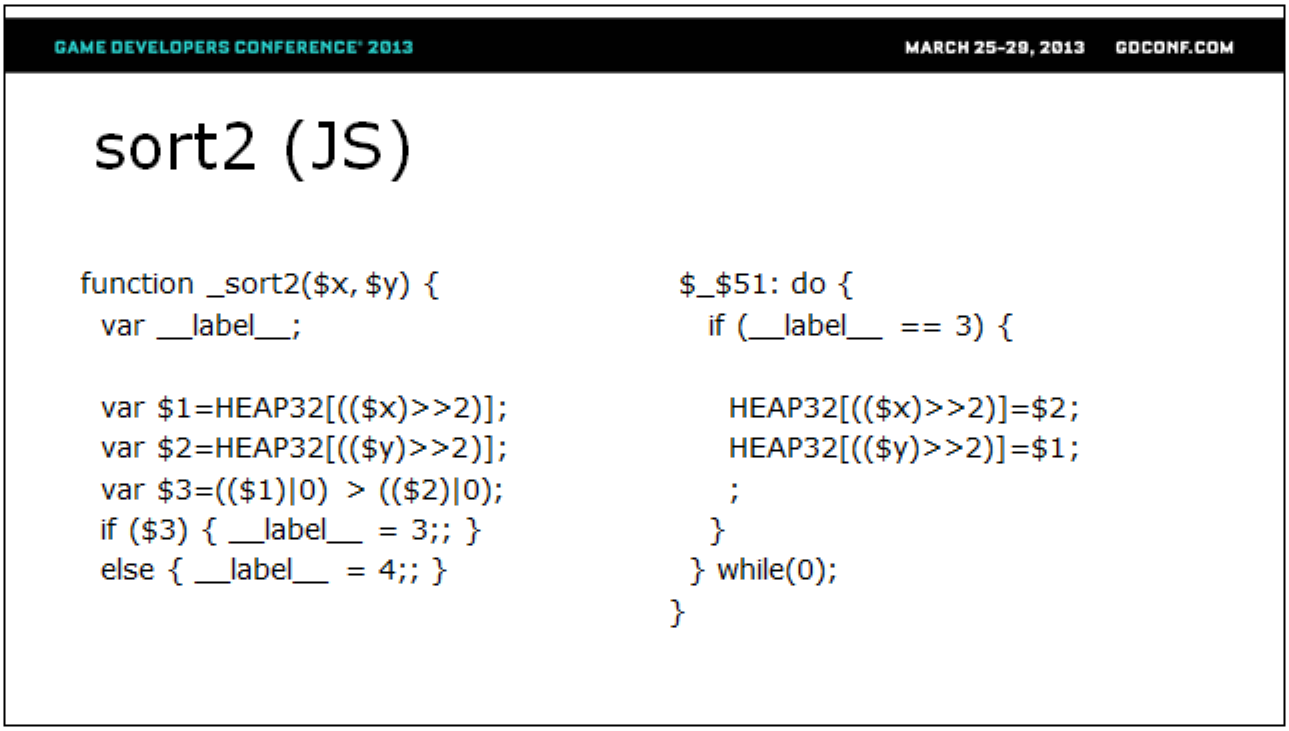
```
function _lerp(a, b, c) {
  return(1 - c) * a + c * b
}
```

Look familiar? ☺



## Branches and memory?

```
void sort2(int& x, int& y) {
    if (x > y) {
        std::swap(x, y);
    }
}
```



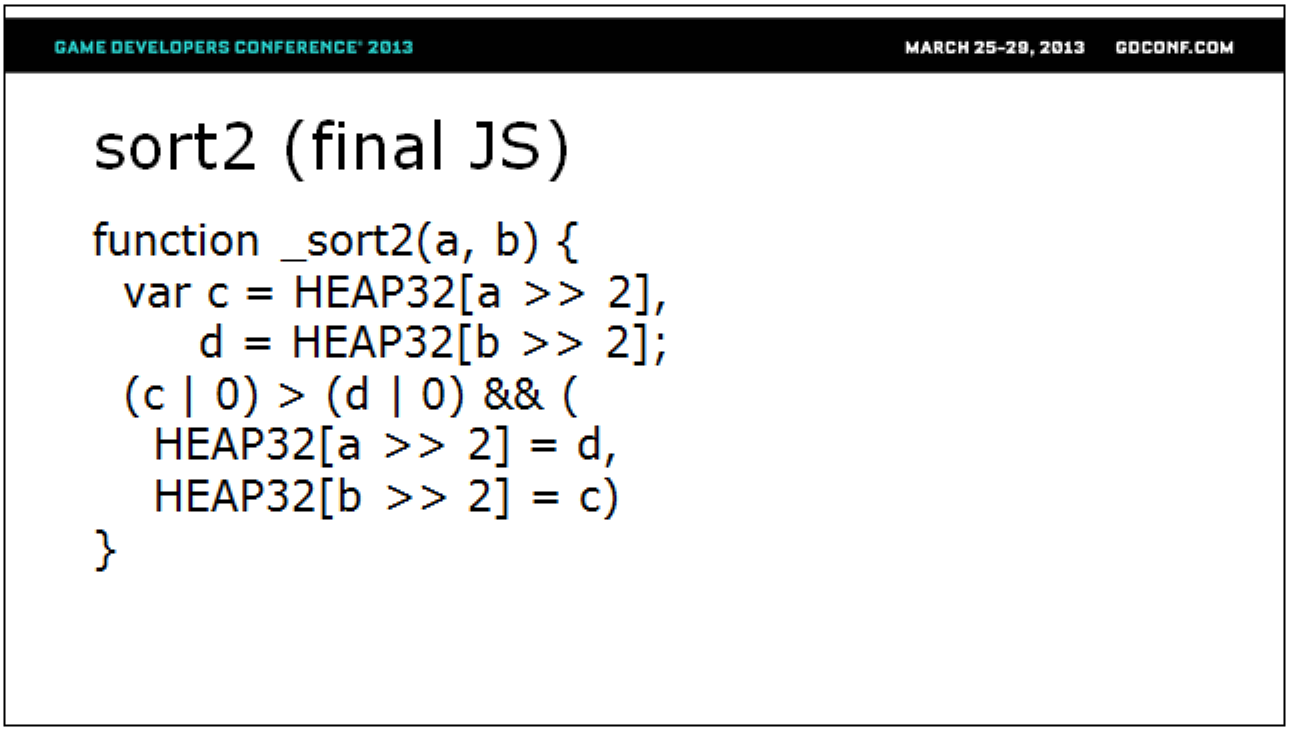
## sort2 (JS)

```
function _sort2($x, $y) {
  var __label__;

  var $1=HEAP32[(($x)>>2)];
  var $2=HEAP32[(($y)>>2)];
  var $3=(($1)|0) > (($2)|0);
  if ($3) { __label__ = 3;; }
  else { __label__ = 4;; }
```

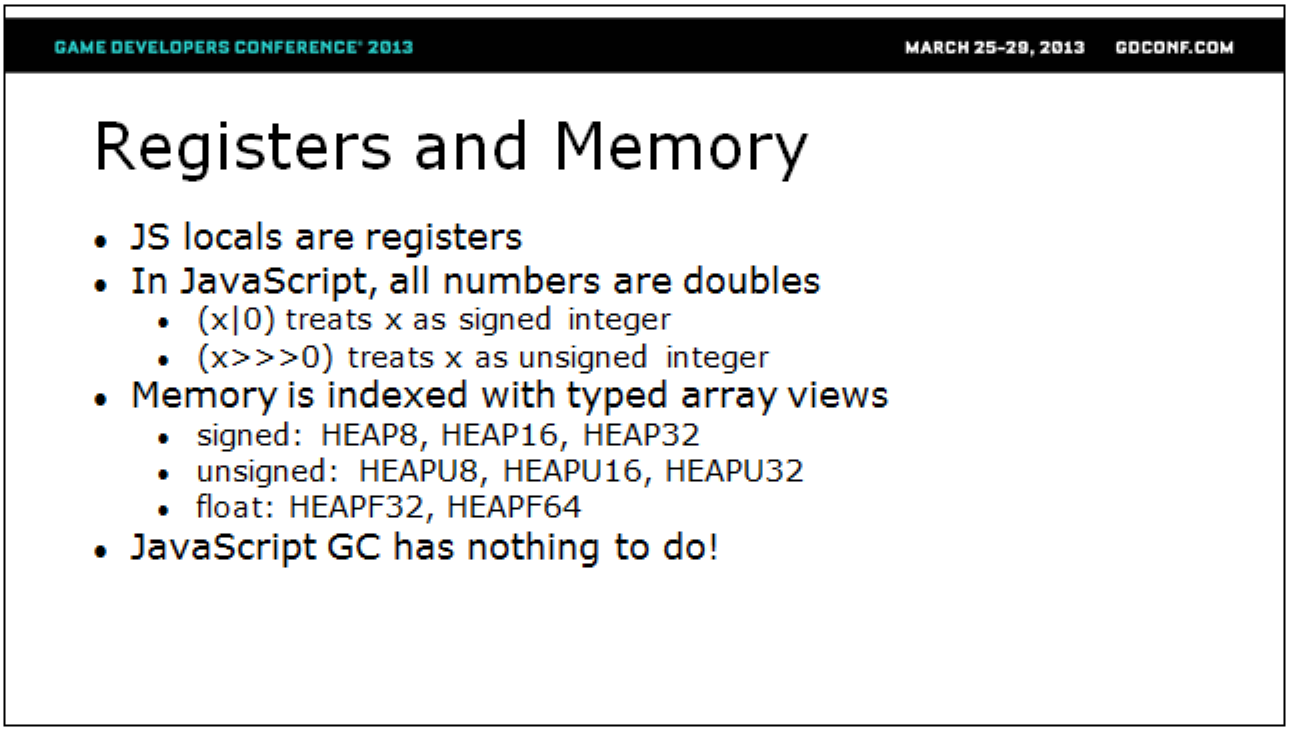
```
 $_$51: do {
   if (__label__ == 3) {

     HEAP32[(($x)>>2)]=$2;
     HEAP32[(($y)>>2)]=$1;
     ;
   }
 } while(0);
}
```



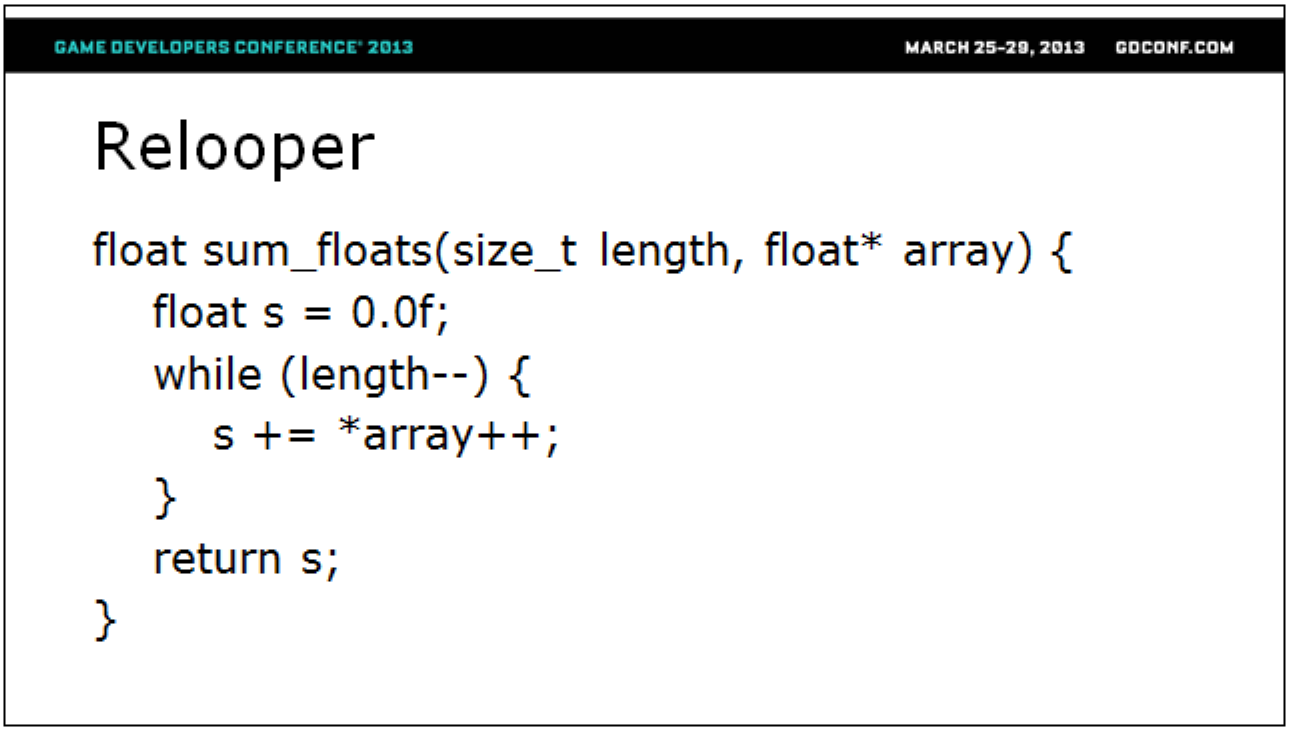
## sort2 (final JS)

```
function _sort2(a, b) {
  var c = HEAP32[a >> 2],
      d = HEAP32[b >> 2];
  (c | 0) > (d | 0) && (
    HEAP32[a >> 2] = d,
    HEAP32[b >> 2] = c)
}
```



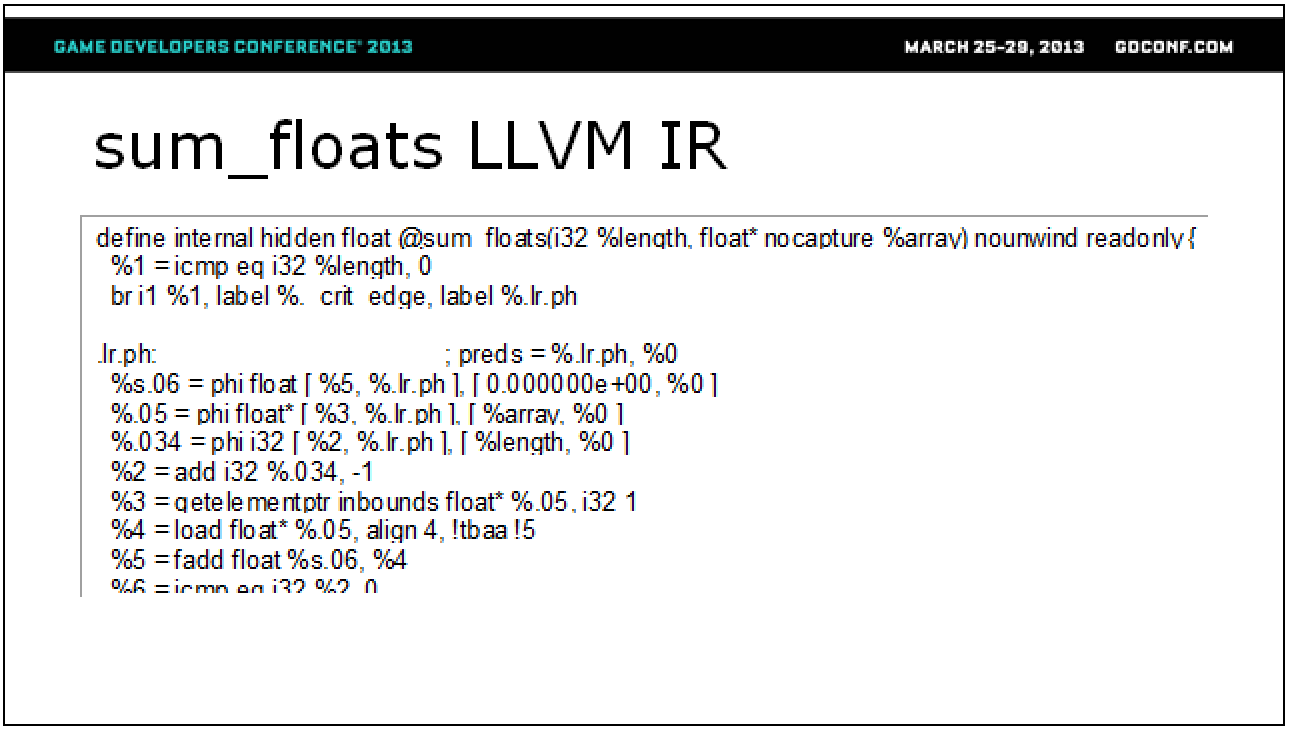
## Registers and Memory

- JS locals are registers
- In JavaScript, all numbers are doubles
  - (x|0) treats x as signed integer
  - (x>>>0) treats x as unsigned integer
- Memory is indexed with typed array views
  - signed: HEAP8, HEAP16, HEAP32
  - unsigned: HEAPU8, HEAPU16, HEAPU32
  - float: HEAPF32, HEAPF64
- JavaScript GC has nothing to do!



## Relooper

```
float sum_floats(size_t length, float* array) {
    float s = 0.0f;
    while (length--) {
        s += *array++;
    }
    return s;
}
```



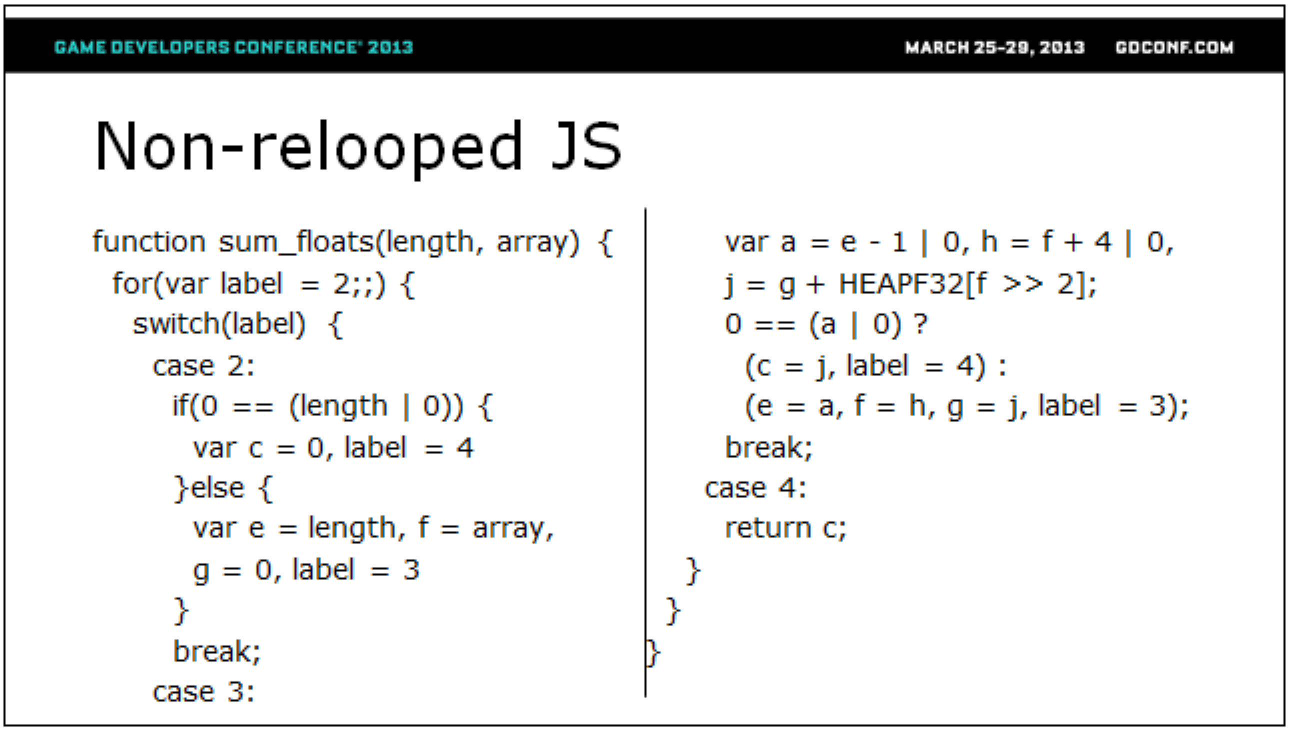
GAME DEVELOPERS CONFERENCE® 2013 MARCH 25-29, 2013 GDCONF.COM

## sum_floats LLVM IR

```
define internal hidden float @sum_floats(i32 %length, float* nocapture %array) nounwind readonly {
  %1 = icmp eq i32 %length, 0
  br i1 %1, label %._crit_edge, label %.lr.ph

.lr.ph:                                    ; preds = %.lr.ph, %0
  %s.06 = phi float [ %5, %.lr.ph ], [ 0.000000e+00, %0 ]
  %.05 = phi float* [ %3, %.lr.ph ], [ %array, %0 ]
  %.034 = phi i32 [ %2, %.lr.ph ], [ %length, %0 ]
  %2 = add i32 %.034, -1
  %3 = getelementptr inbounds float* %.05, i32 1
  %4 = load float* %.05, align 4, !tbaa !5
  %5 = fadd float %s.06, %4
  %6 = icmp eq i32 %2, 0
```

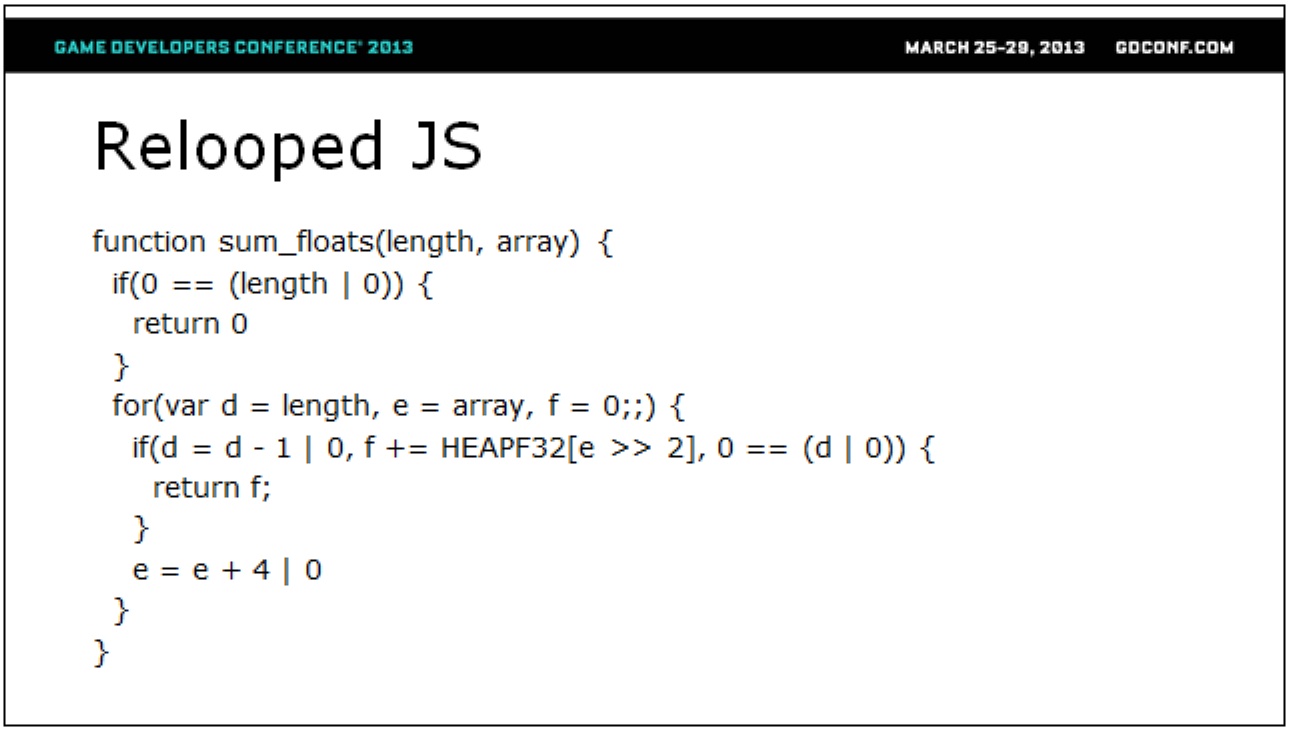
Note that LLVM has no high-level control flow, just branch instructions.


GAME DEVELOPERS CONFERENCE® 2013 MARCH 25-29, 2013 GDCONF.COM

## Non-relooped JS

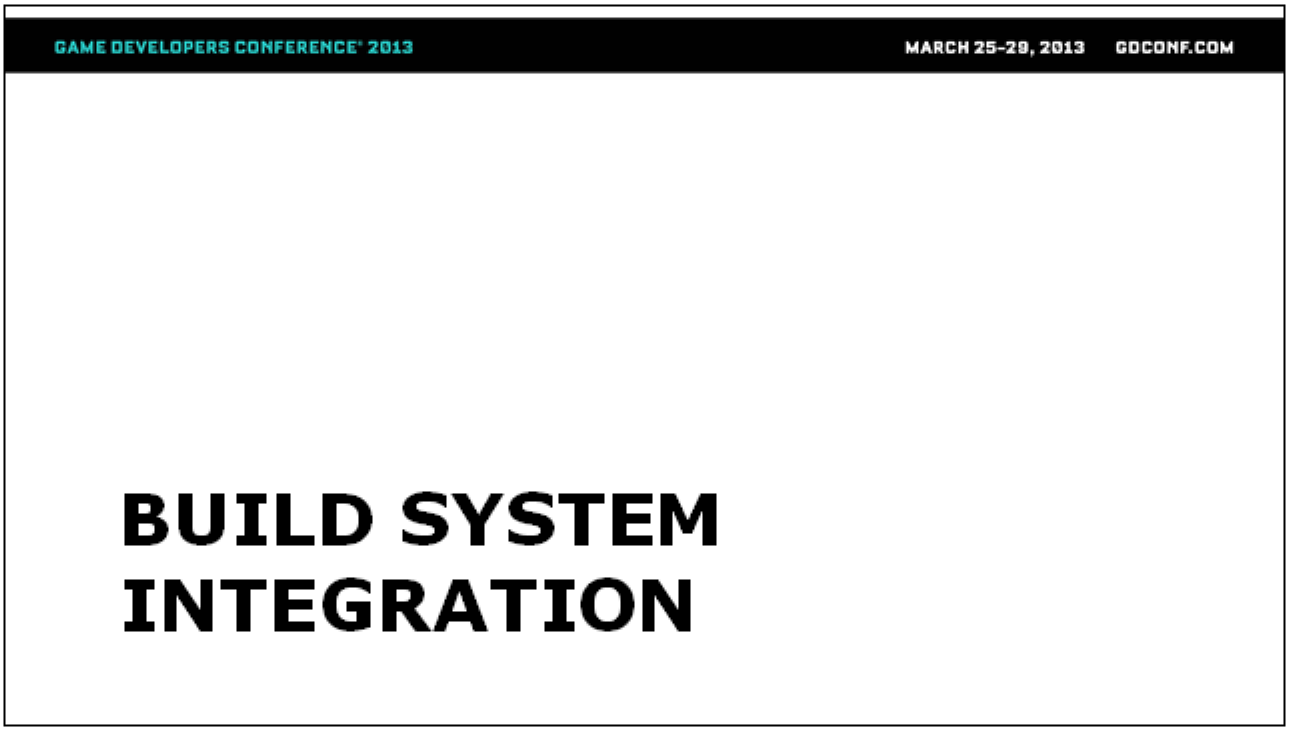
```
function sum_floats(length, array) {
 for(var label = 2;;) {
  switch(label) {
   case 2:
    if(0 == (length | 0)) {
     var c = 0, label = 4
    }else {
     var e = length, f = array,
     g = 0, label = 3
    }
    break;
   case 3:
    var a = e - 1 | 0, h = f + 4 | 0,
    j = g + HEAPF32[f >> 2];
    0 == (a | 0) ?
     (c = j, label = 4) :
     (e = a, f = h, g = j, label = 3);
    break;
   case 4:
    return c;
  }
 }
}
```

Naive translation to JS results in an infinite loop containing a switch. In effect, implementing goto.

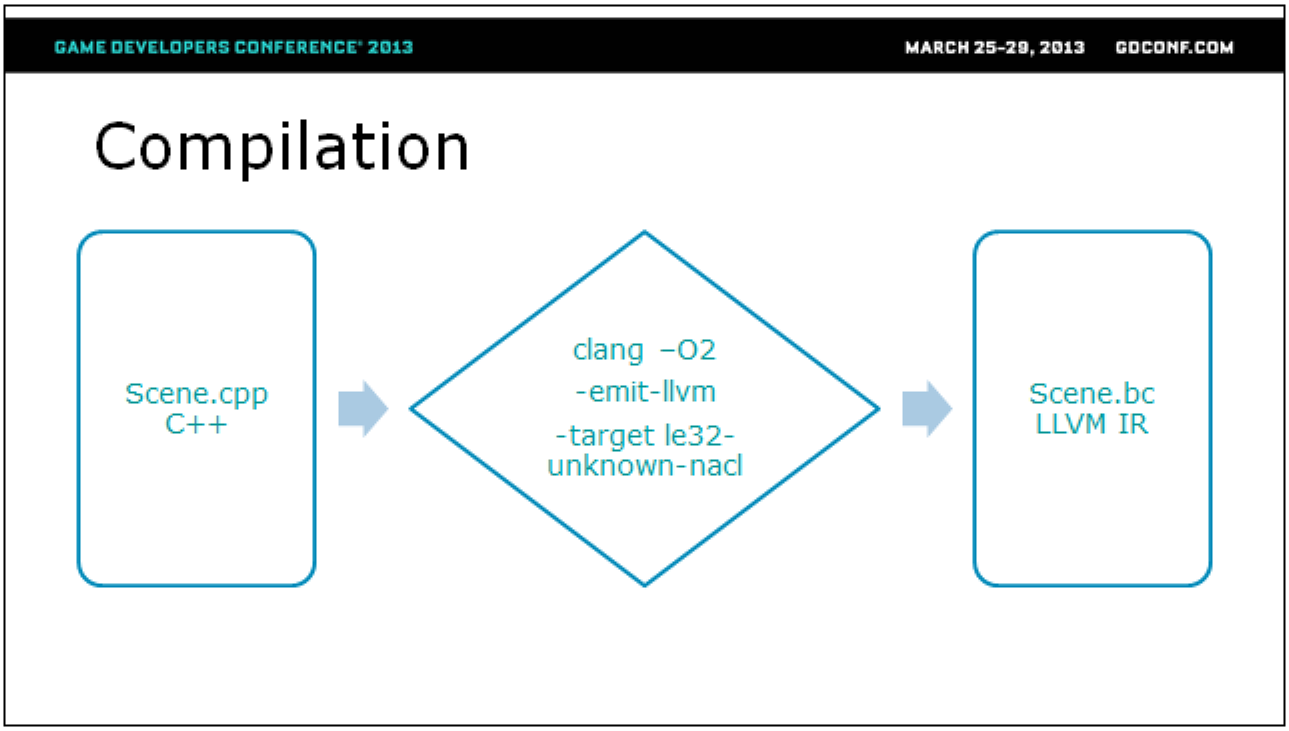


## Relooped JS

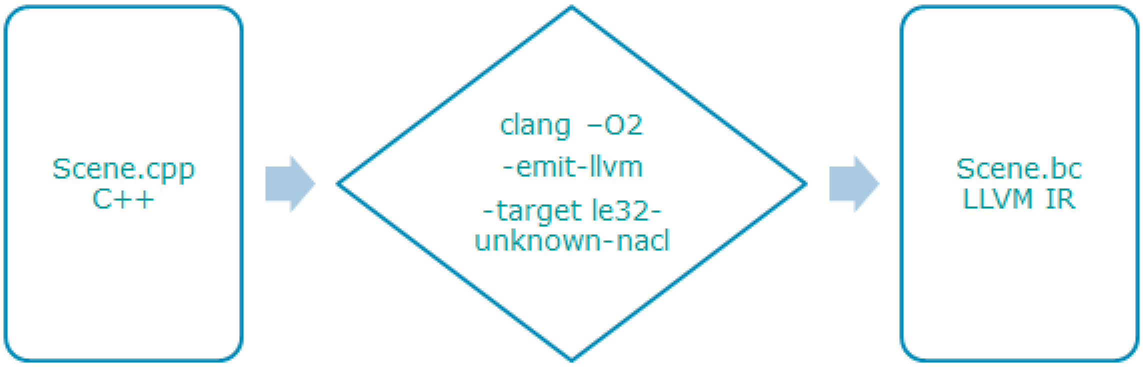
```
function sum_floats(length, array) {
  if(0 == (length | 0)) {
    return 0
  }
  for(var d = length, e = array, f = 0;;) {
    if(d = d - 1 | 0, f += HEAPF32[e >> 2], 0 == (d | 0)) {
      return f;
    }
    e = e + 4 | 0
  }
}
```

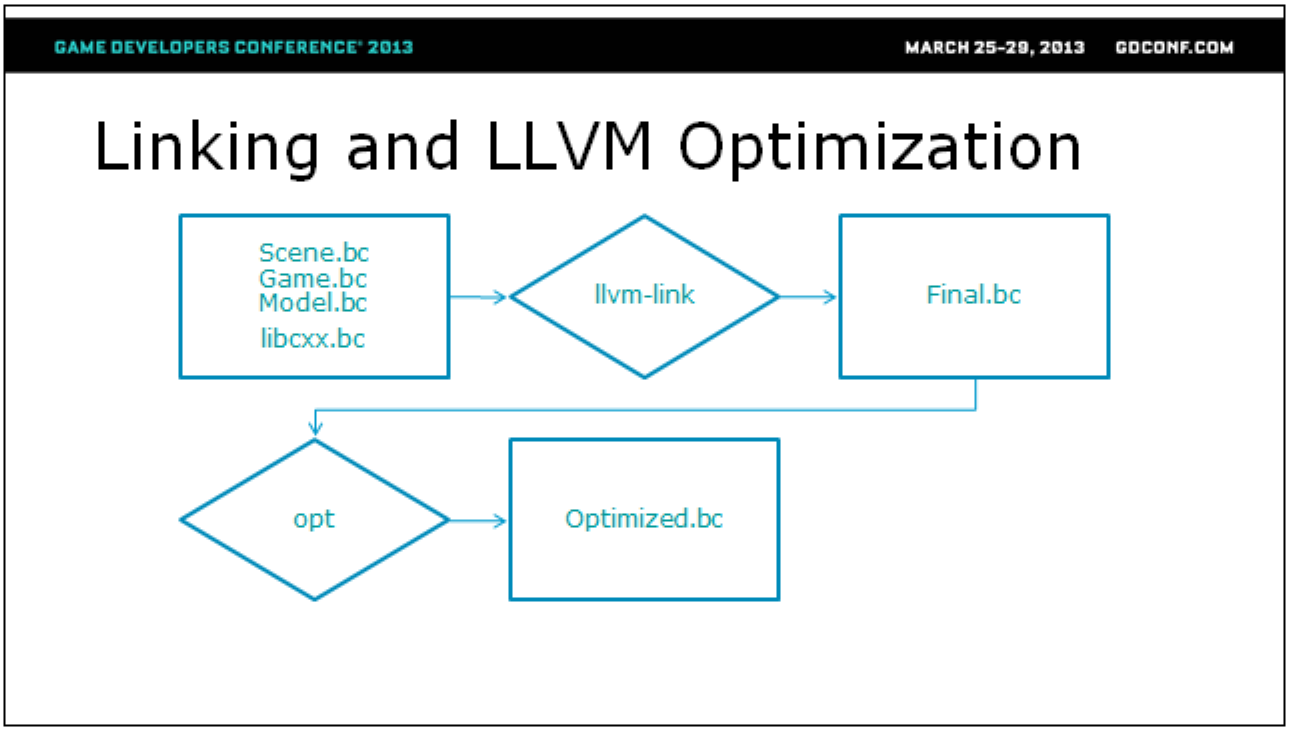


## BUILD SYSTEM INTEGRATION



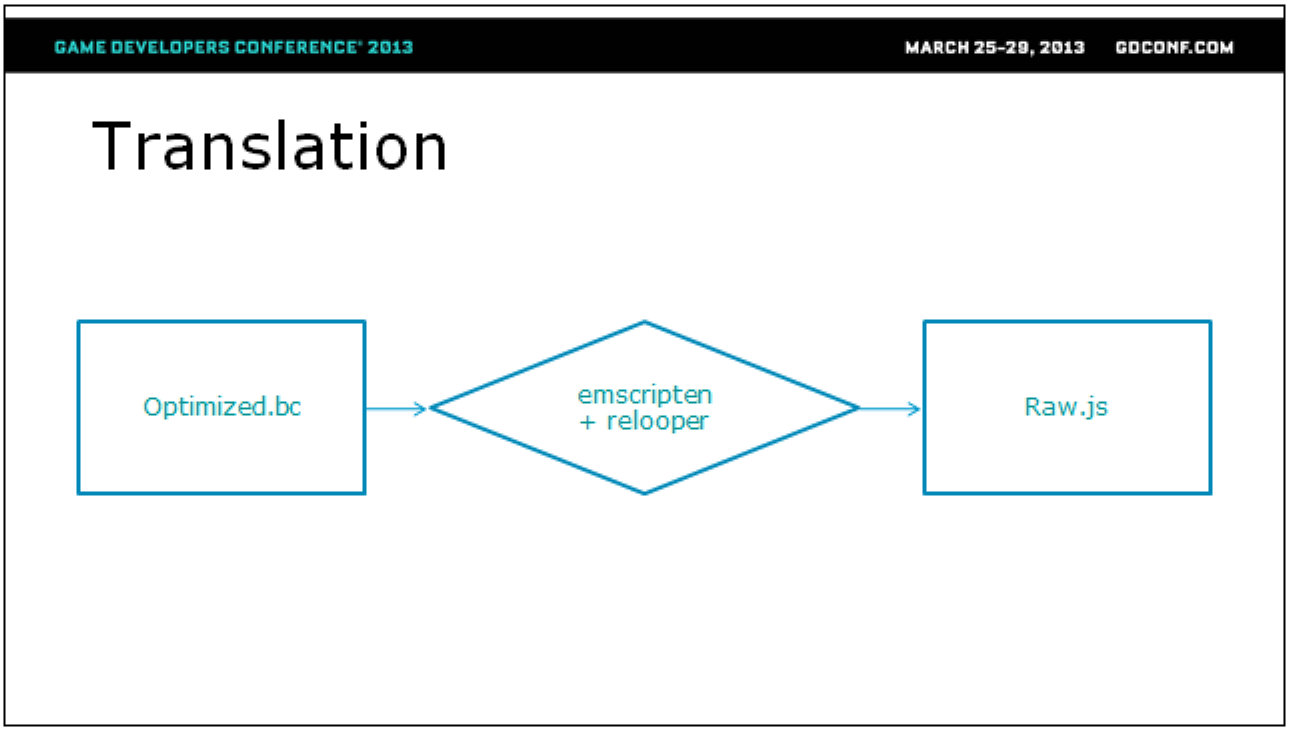
## Compilation



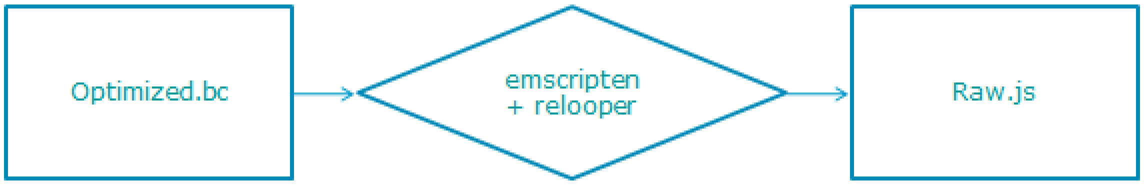


## Linking and LLVM Optimization

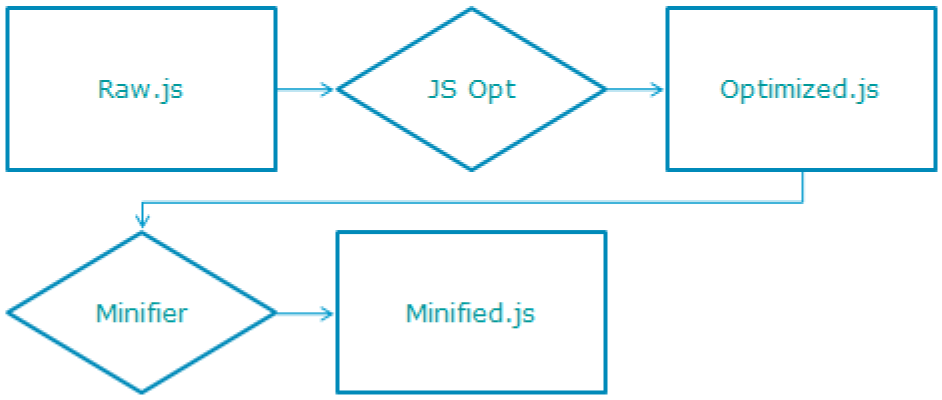
$$\begin{tikzcd}
\boxed{\text{Scene.bc, Game.bc, Model.bc, libcxx.bc}} \arrow[r] & \text{llvm-link} \arrow[r] & \boxed{\text{Final.bc}} \arrow[dll] \\
\text{opt} \arrow[r] & \boxed{\text{Optimized.bc}} &
\end{tikzcd}$$



## Translation

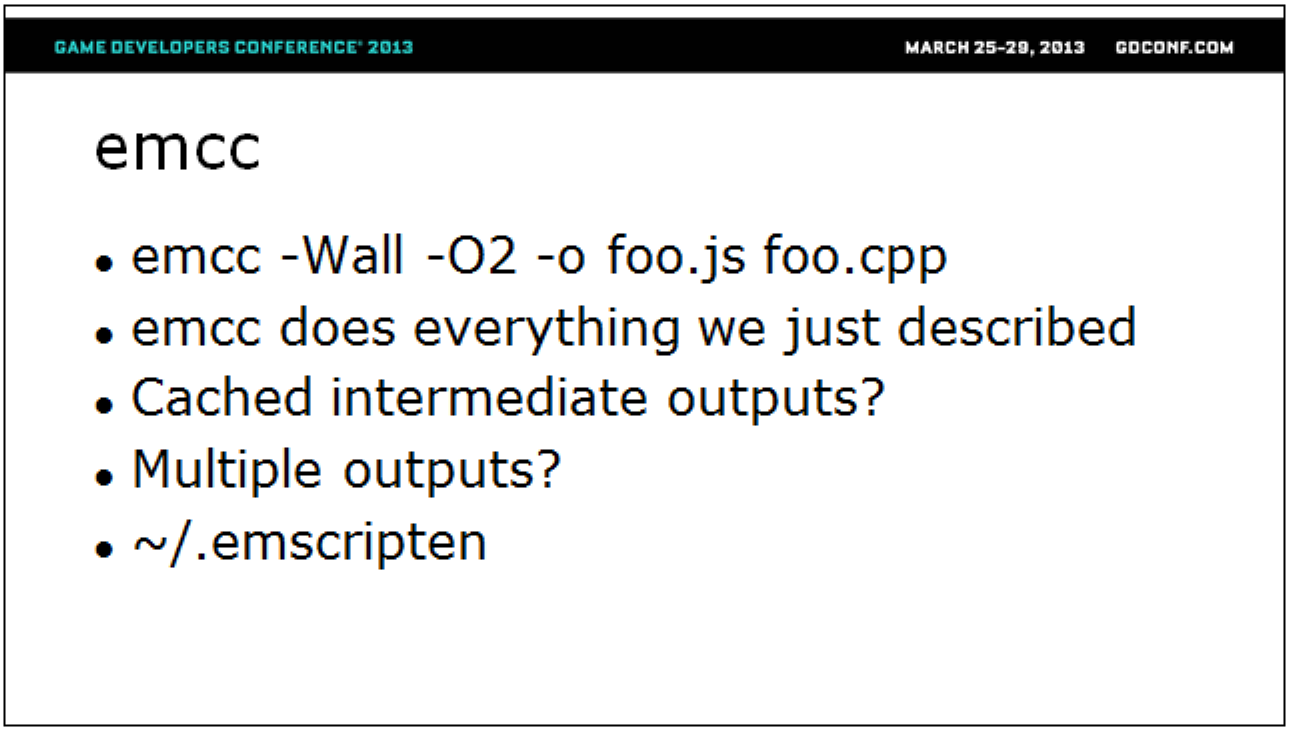


## JavaScript Optimization



## Convert to JavaScript Module

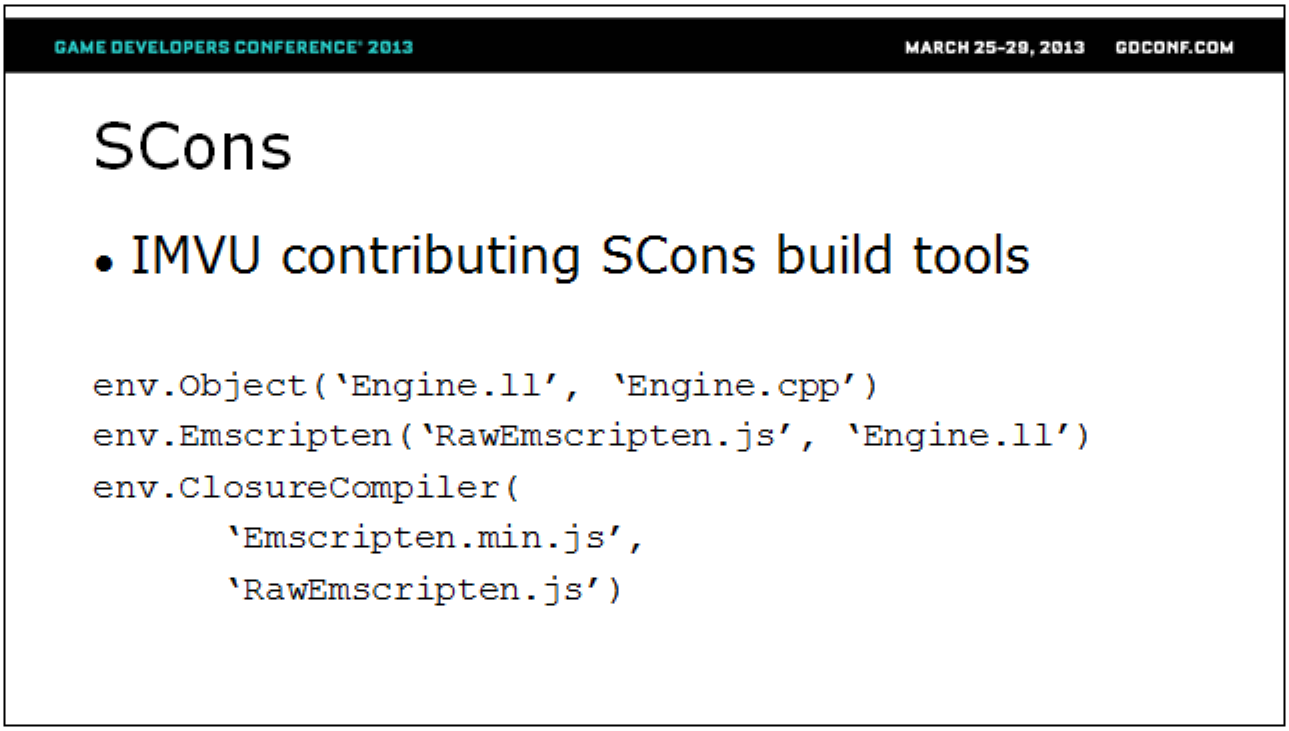
- RequireJS (AMD)
- CommonJS (NodeJS)
- module (IMVU)


GAME DEVELOPERS CONFERENCE® 2013 MARCH 25-29, 2013 GDCONF.COM

## emcc

- emcc -Wall -O2 -o foo.js foo.cpp
- emcc does everything we just described
- Cached intermediate outputs?
- Multiple outputs?
- ~/.emscripten

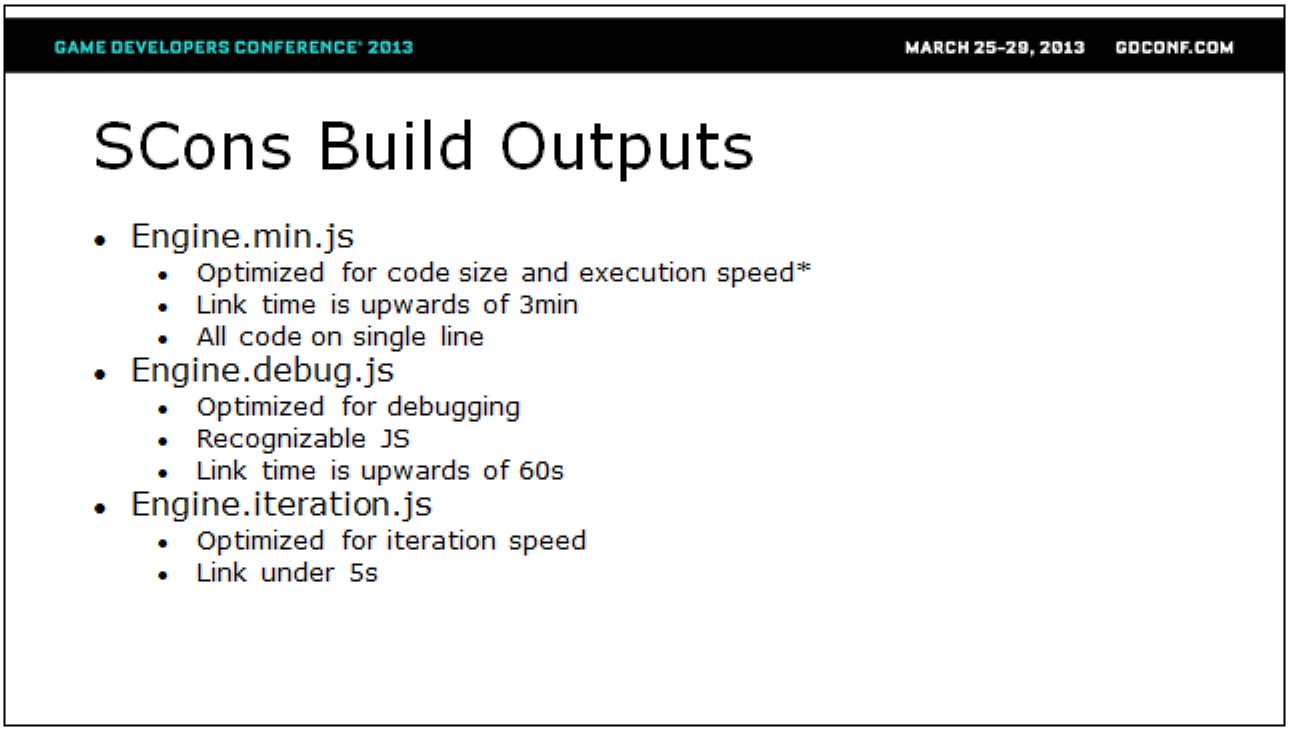
Another reason we prefer SCons over emcc is in the case where a change to some C++ doesn't necessarily modify the generated code. In that case, emcc would attempt to relink, the slowest part, whereas SCons would know that the linker inputs haven't changed and finish the build early.



## SCons

- IMVU contributing SCons build tools

```
env.Object('Engine.ll', 'Engine.cpp')
env.Emscripten('RawEmscripten.js', 'Engine.ll')
env.ClosureCompiler(
        'Emscripten.min.js',
        'RawEmscripten.js')
```


GAME DEVELOPERS CONFERENCE® 2013 MARCH 25-29, 2013 GDCONF.COM

## SCons Build Outputs

- Engine.min.js
  - Optimized for code size and execution speed*
  - Link time is upwards of 3min
  - All code on single line
- Engine.debug.js
  - Optimized for debugging
  - Recognizable JS
  - Link time is upwards of 60s
- Engine.iteration.js
  - Optimized for iteration speed
  - Link under 5s

Engine.min.js is the most important build, but it’s painful to build and work with.

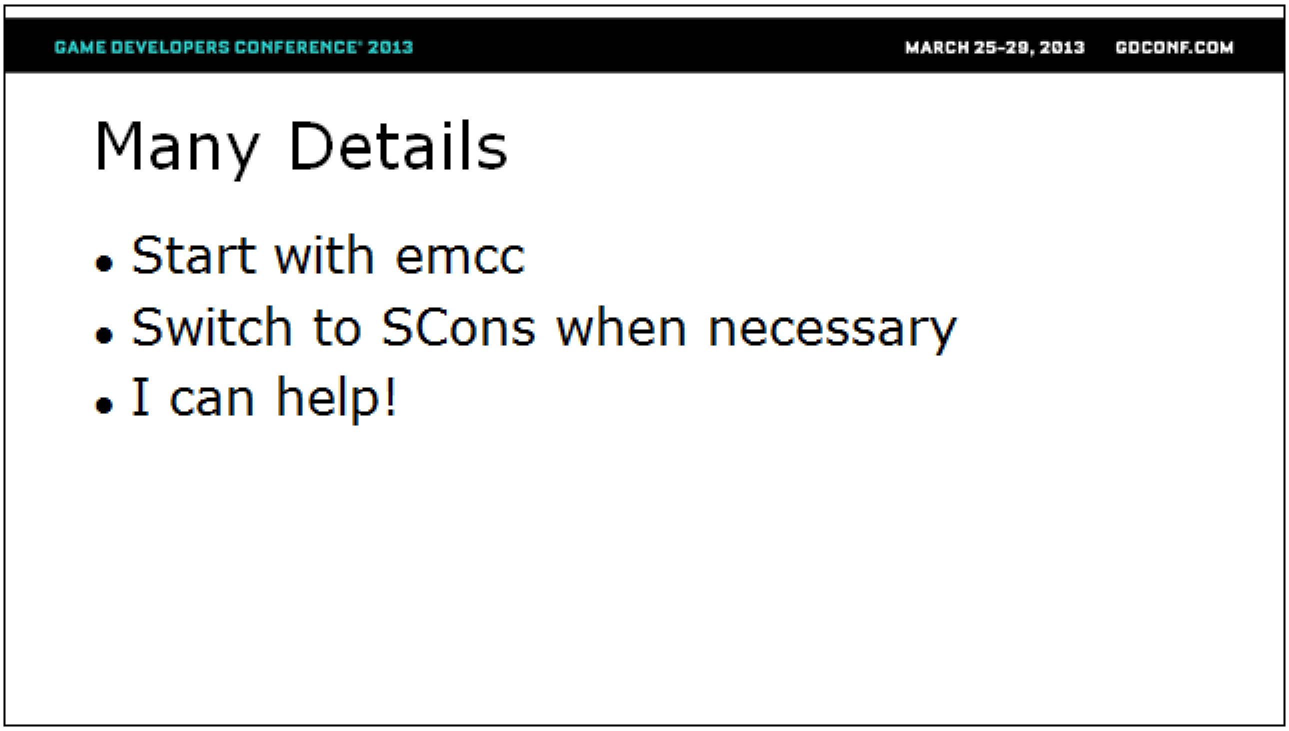
Engine.debug.js is ideal for debugging, but still takes time to build.

Engine.iteration.js is for running unit tests.

```
(e|0)!=(c|0))return  m=w,e-c|0;if(0==(e|0))return  m=w,0;if(0!=r[h+
36|0]<<24>>24)return  q=h|0,q=(G.multiply(a[q>>2],a[q+4>>2],e,0
>(e|0)?-1:0),a[u>>2]),q=-mm(d,q,a[u+4>>2])|0,m=w,q;
e=h|0;k=(e|0)>>2;b=(e+4|0)>>2;c=h+8|0;e=(c|0)>>2;var
f=d+16|0,g=d+24|0;Aj(a[k],a[b],a[e],a[(c+4|0)>>2],a[f>>2],a[f+4>>2],
a[g>>2],a[g+4>>2],w,A);e=h+16|0;c=h+24|0;k=d|0;b=d+8|0;Aj(a[e>>
2],a[e+4>>2],a[c>>2],a[c+4>>2],a[k>>2],a[k+4>>2],a[b>>2],a[b+4>
>2],J,Ca);c=A+8|0;e=a[c>>2];c=a[c+4>>2];b=Ca+8|0;k=a[b>>2];b=a[
b+4>>2];if(c>>>0<b>>>0|c>>>0==b>>>0&e>>>0<k>>>0)J=-1;else
if(c>>>0>b>>>0|c>>>0==b>>>0&e>>>0>k>>>0)J=1;else
if(e=A|0,k=(e|0)>>2,A=a[k],b=(e+4|0)>>2,k=a[b],c=Ca|0,e=(c|0)>>2,C
a=a[e],c=(c+4|0)>>2,e=a[c],k>>>0<e>>>0|k>>>0==e>>>0&A>>>0<
Ca>>>0)J=-1;else  if(Ca=k>>>0>e>>>0|k>>>0==e>>>0
&A>>>0>Ca>>>0)J=Ca&1;else  return A=w+8|0,k=(A|0)>>2,Ca=a[k],
b=(A+4|0)>>2,A=a[b],c=J+8|0,e=(c|0)>>2,e=a[e],c=(c+4|0)>>2,c=a[c]
,A>>>0<c>>>0|A>>>0==c>>>0&Ca>>>0<e>>>0?J=-
1:A>>>0>c>>>0|A>>>0==c>>>0&Ca>>>0>e>>>0?J=1:(A=w|0,k=(A|
0)>>2,Ca=a[k],b=
```

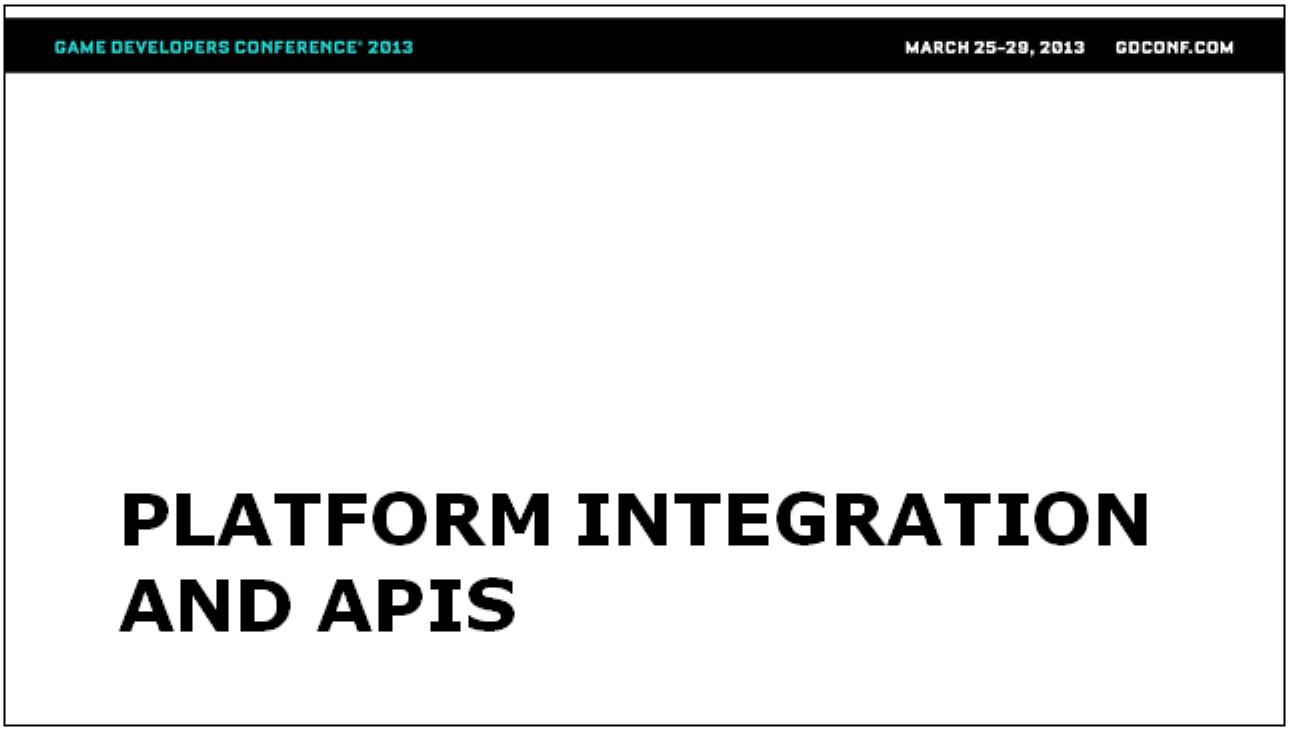
## Debug Build

```
function __Z22btAlignedAllocInternalji(a) {
  HEAP32[1310797] = HEAP32[1310797] + 1 | 0;
  return __ZL21btAlignedAllocDefaultji(a, 16)
}
function __Z21btAlignedFreeInternalPv(a) {
  0 != (a | 0) && (HEAP32[1310796] =
HEAP32[1310796] + 1 | 0,
__ZL20btAlignedFreeDefaultPv(a))
}
```

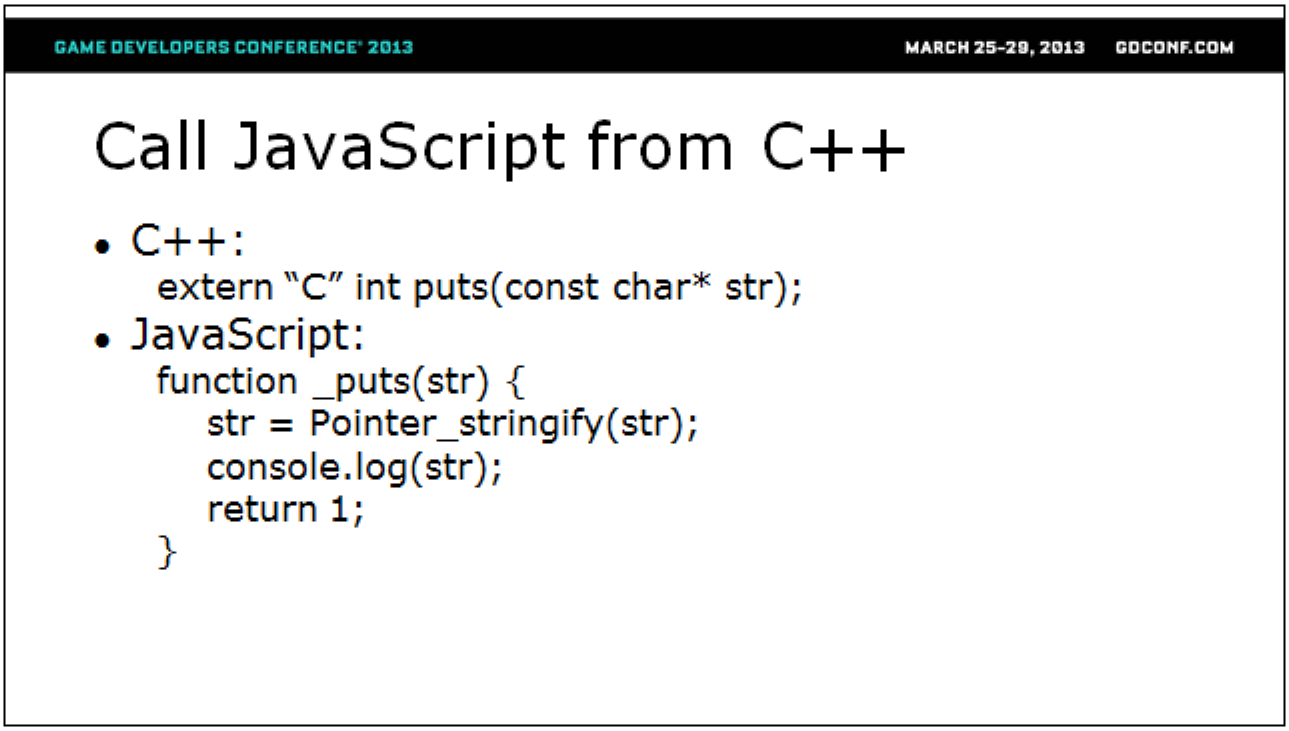


## Many Details

- Start with emcc
- Switch to SCons when necessary
- I can help!



## PLATFORM INTEGRATION AND APIS



## Call JavaScript from C++

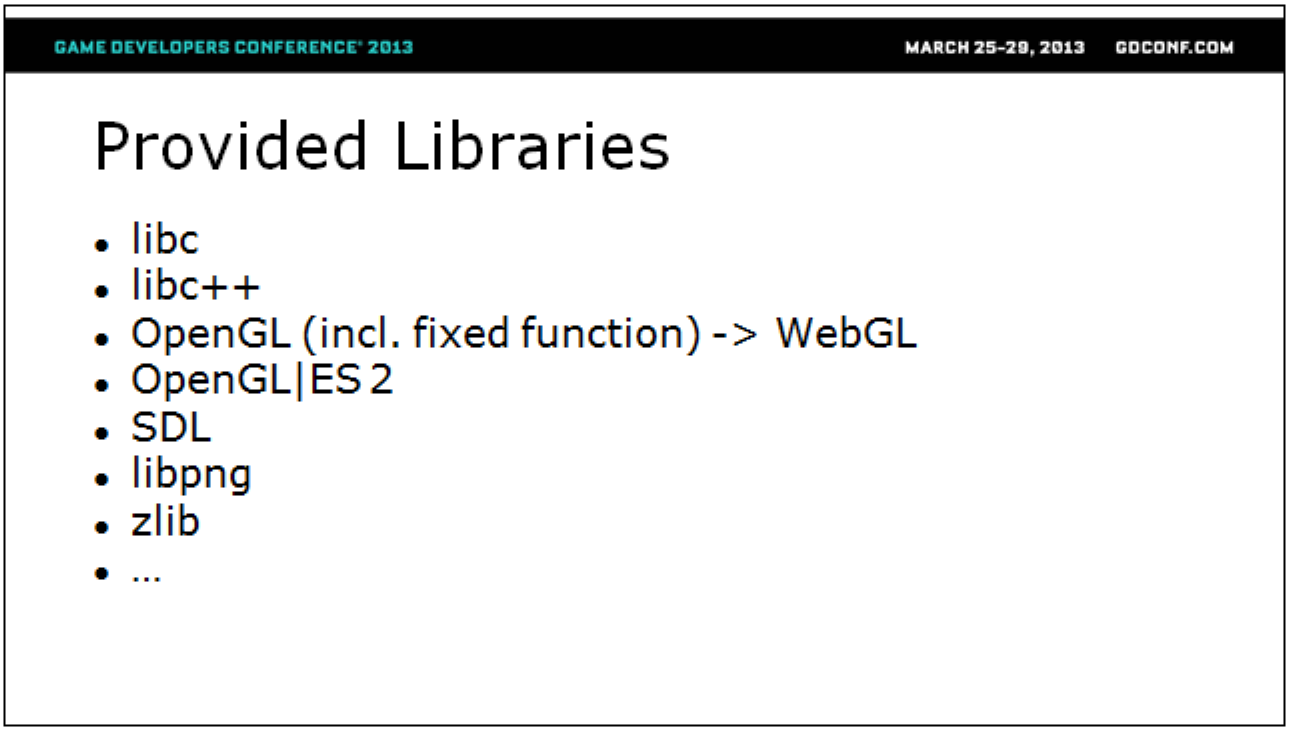
- C++:

```
extern "C" int puts(const char* str);
```

- JavaScript:

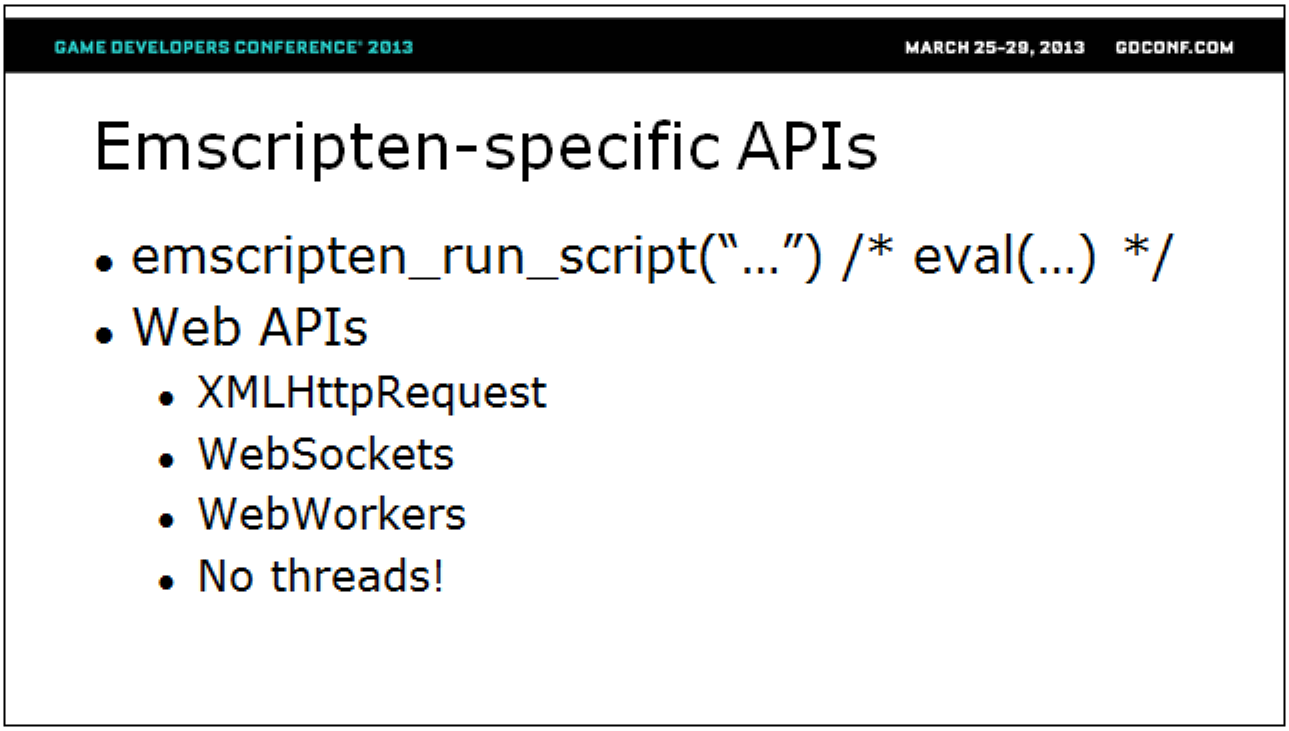
```
function _puts(str) {
    str = Pointer_stringify(str);
    console.log(str);
    return 1;
}
```

Pointer_stringify is a helper that takes the char* address and creates a JS string from the heap.



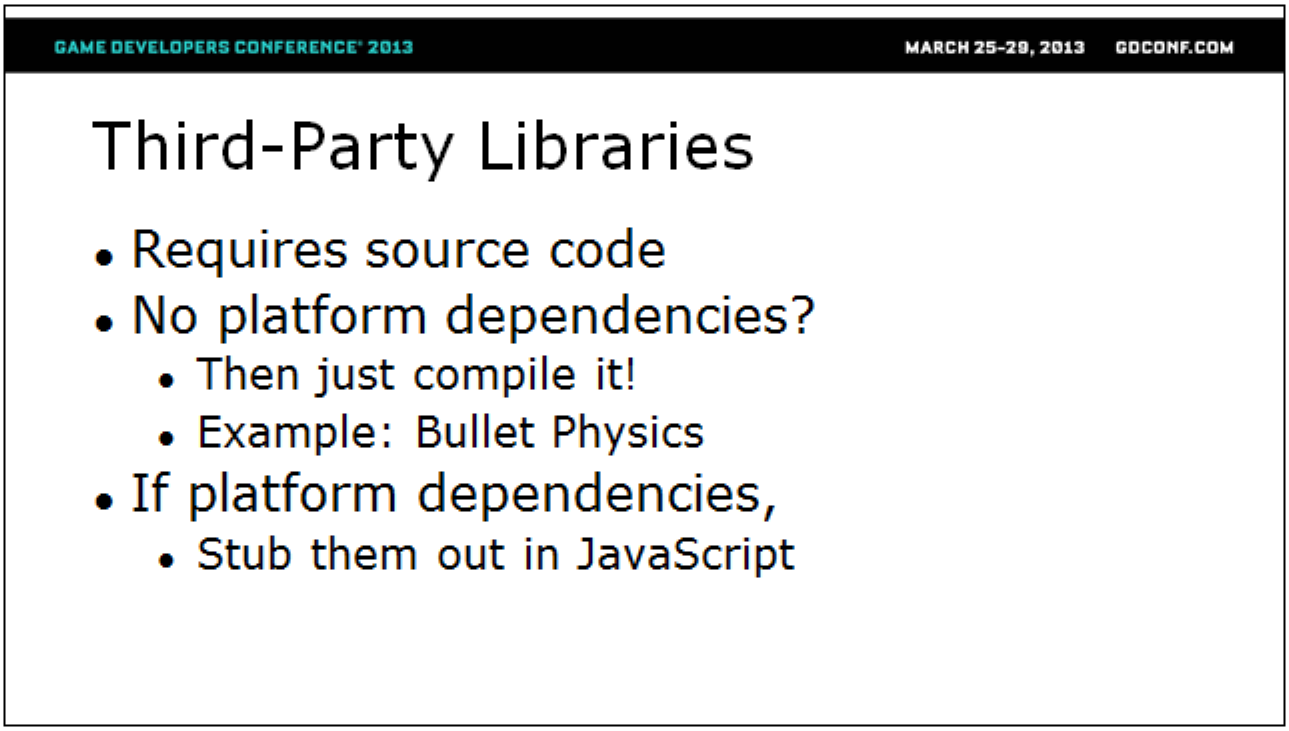
## Provided Libraries

- libc
- libc++
- OpenGL (incl. fixed function) -> WebGL
- OpenGL|ES 2
- SDL
- libpng
- zlib
- ...



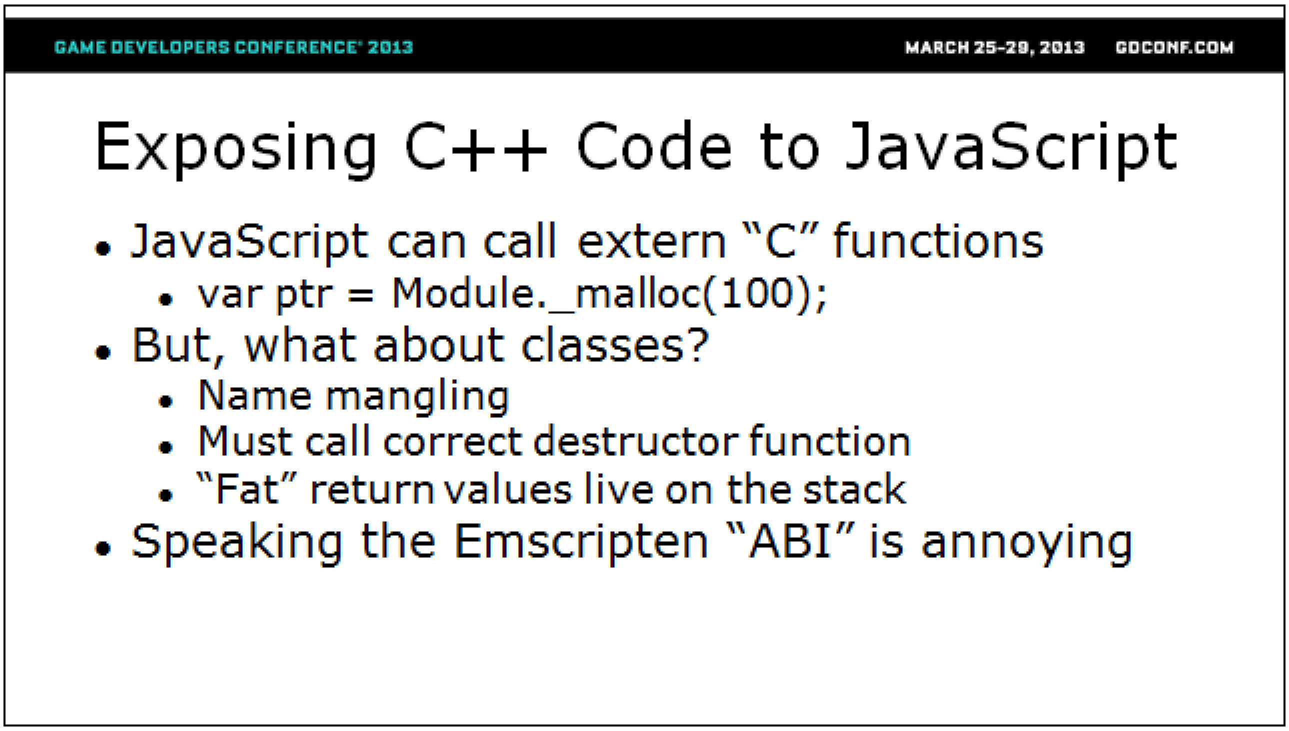
## Emscripten-specific APIs

- emscripten_run_script("...") /* eval(...) */
- Web APIs
  - XMLHttpRequest
  - WebSockets
  - WebWorkers
  - No threads!



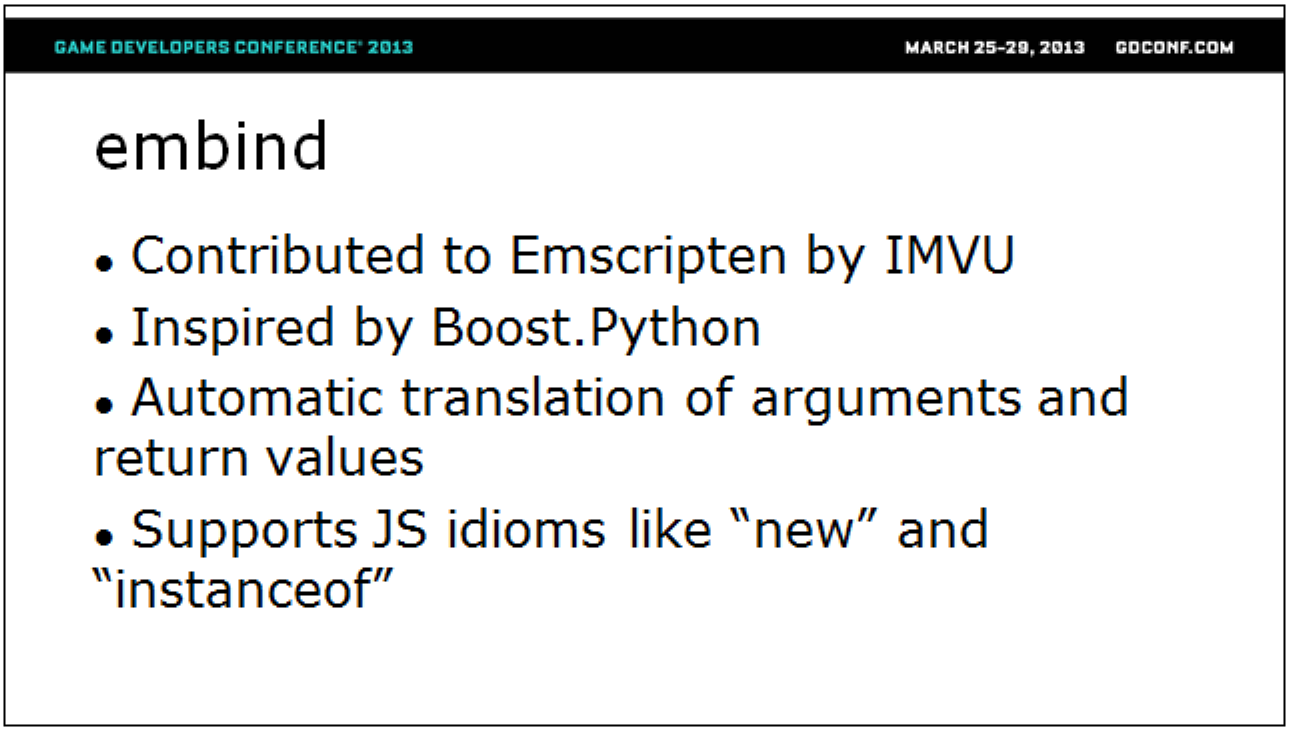
## Third-Party Libraries

- Requires source code
- No platform dependencies?
  - Then just compile it!
  - Example: Bullet Physics
- If platform dependencies,
  - Stub them out in JavaScript



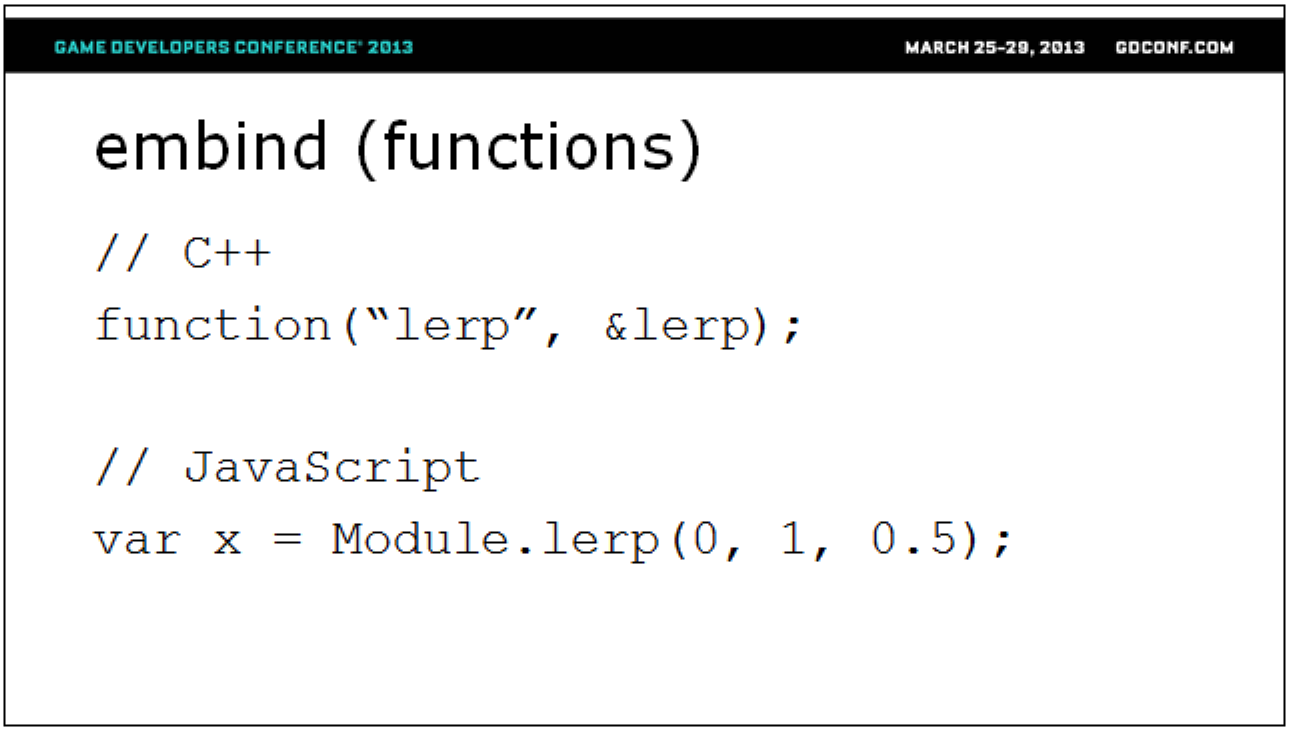
## Exposing C++ Code to JavaScript

- JavaScript can call extern "C" functions
  - var ptr = Module._malloc(100);
- But, what about classes?
  - Name mangling
  - Must call correct destructor function
  - "Fat" return values live on the stack
- Speaking the Emscripten "ABI" is annoying



## embind

- Contributed to Emscripten by IMVU
- Inspired by Boost.Python
- Automatic translation of arguments and return values
- Supports JS idioms like “new” and “instanceof”



## embind (functions)

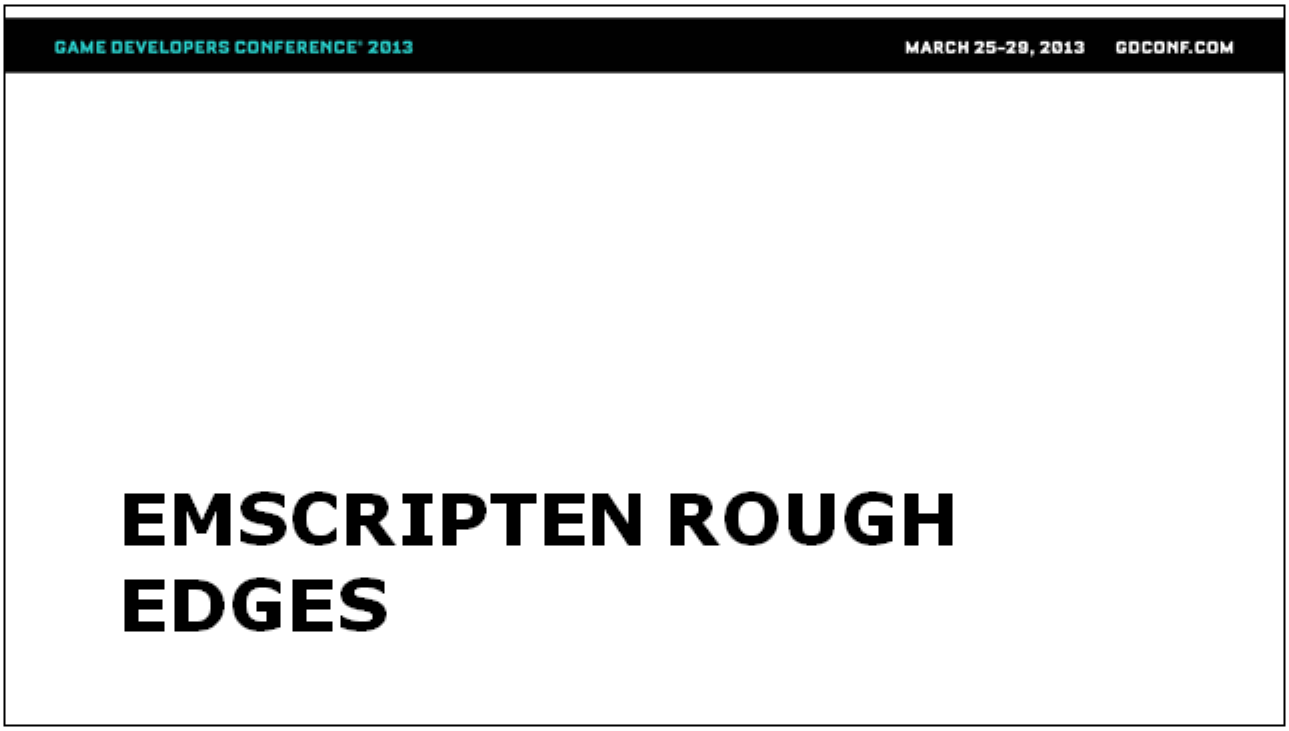
```
// C++
function("lerp", &lerp);

// JavaScript
var x = Module.lerp(0, 1, 0.5);
```

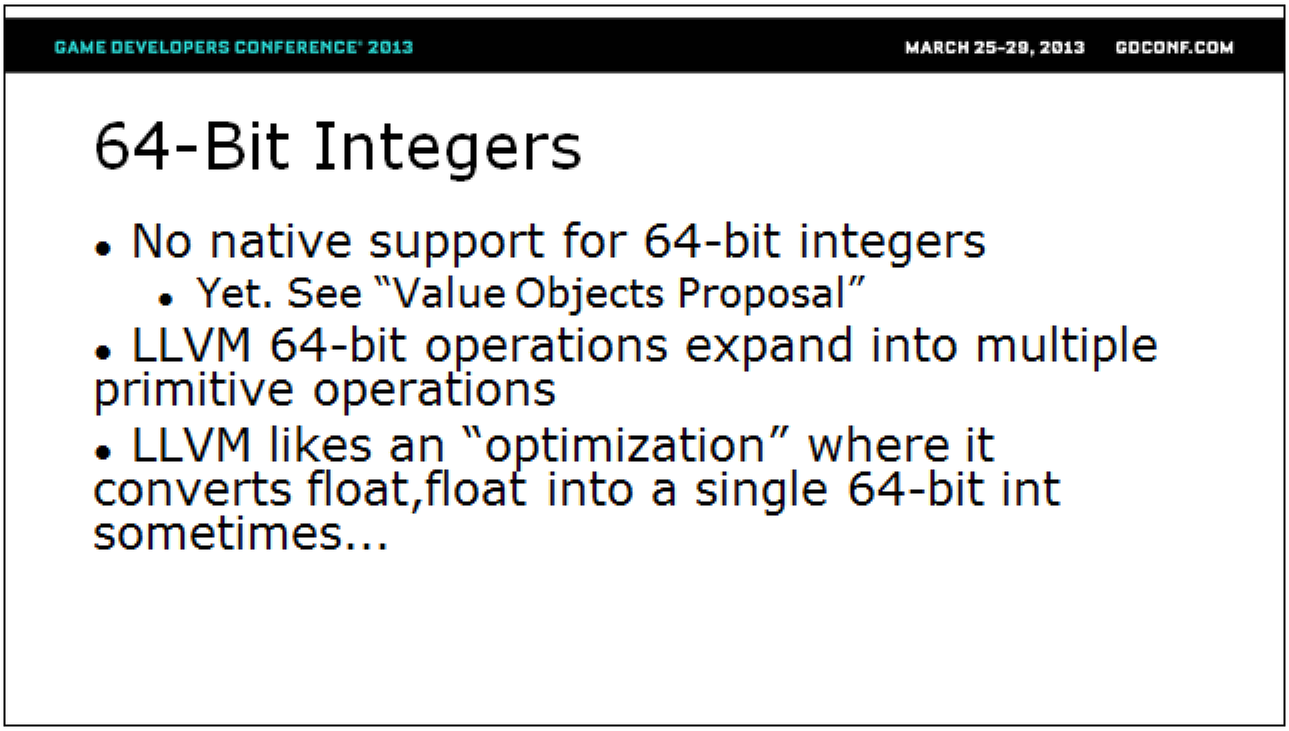
## embind (classes)

```
class_<Skeleton>("Skeleton") // C++
    .constructor<>()
    .function("addBone", &Skeleton::addBone)
    .property("bones", &Skeleton::bones)
    ;

var s = new Module.Skeleton(); // JavaScript
s.addBone(bone);
s.delete(); // Don't forget!
```

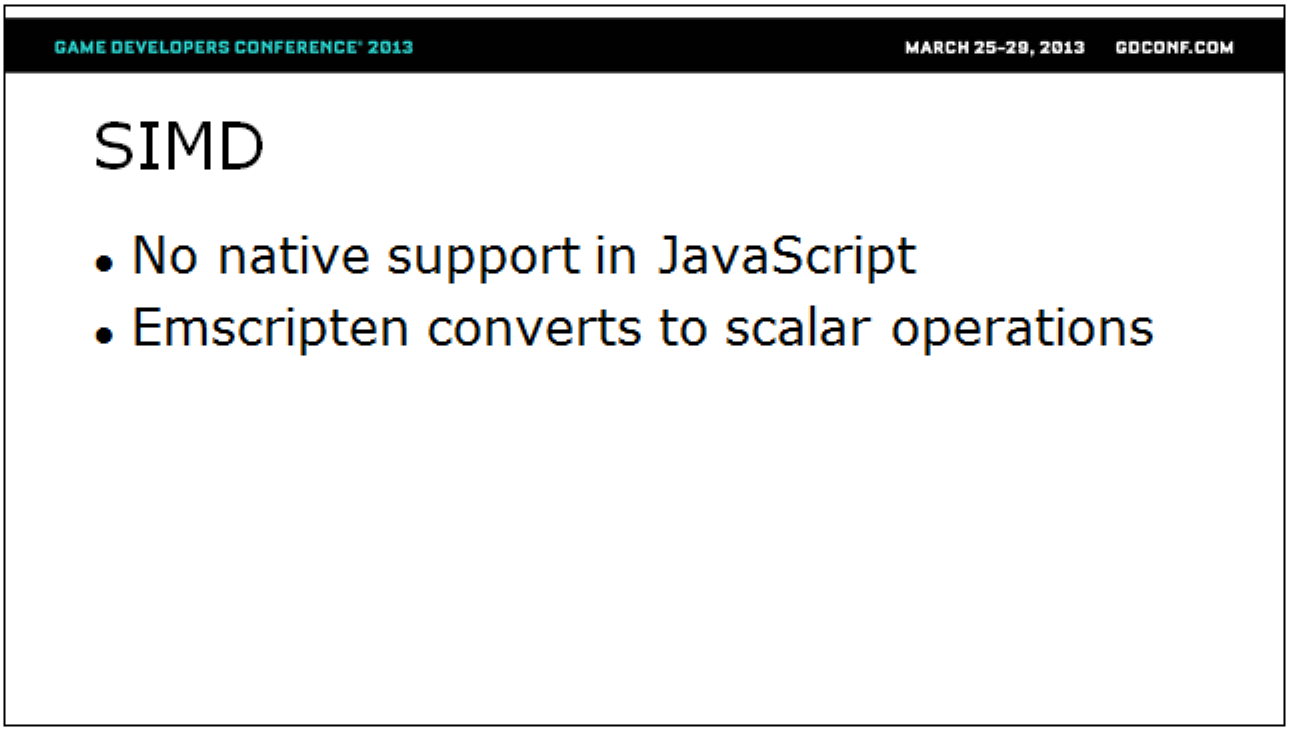


## EMSCRIPTEN ROUGH EDGES



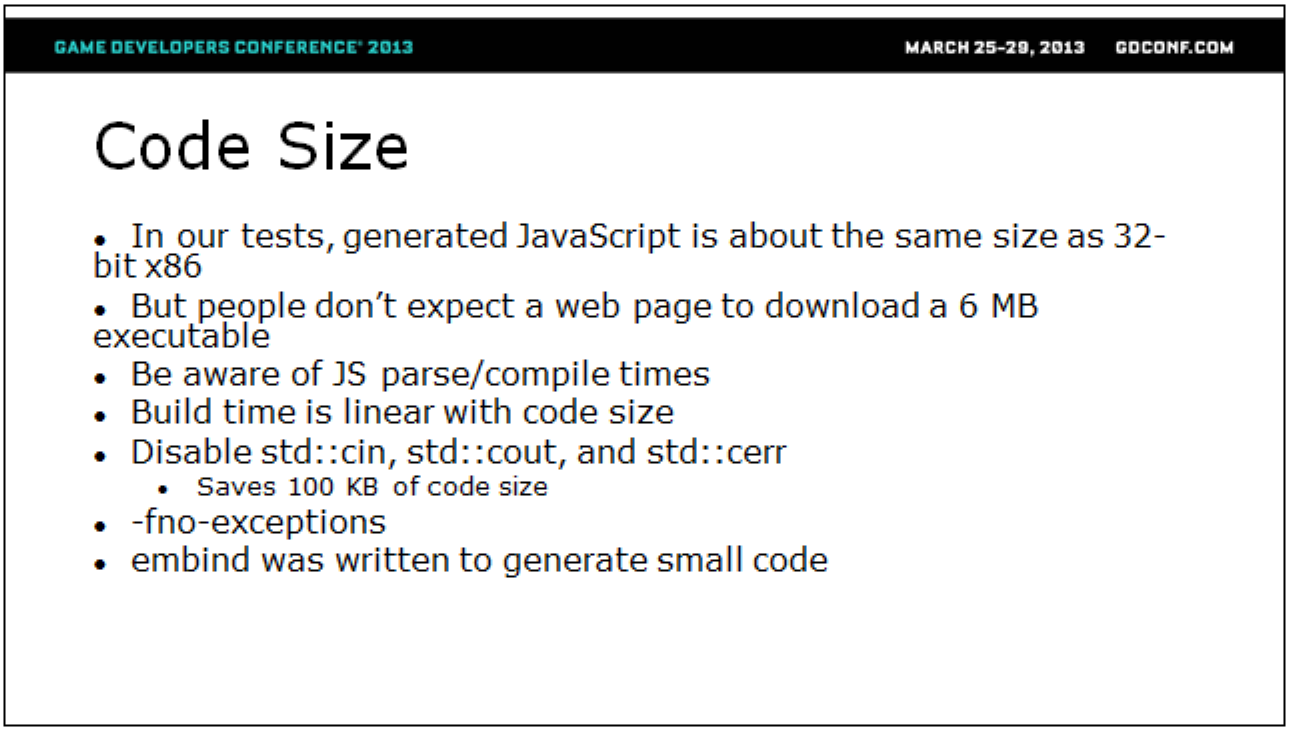
## 64-Bit Integers

- No native support for 64-bit integers
  - Yet. See "Value Objects Proposal"
- LLVM 64-bit operations expand into multiple primitive operations
- LLVM likes an "optimization" where it converts float,float into a single 64-bit int sometimes…



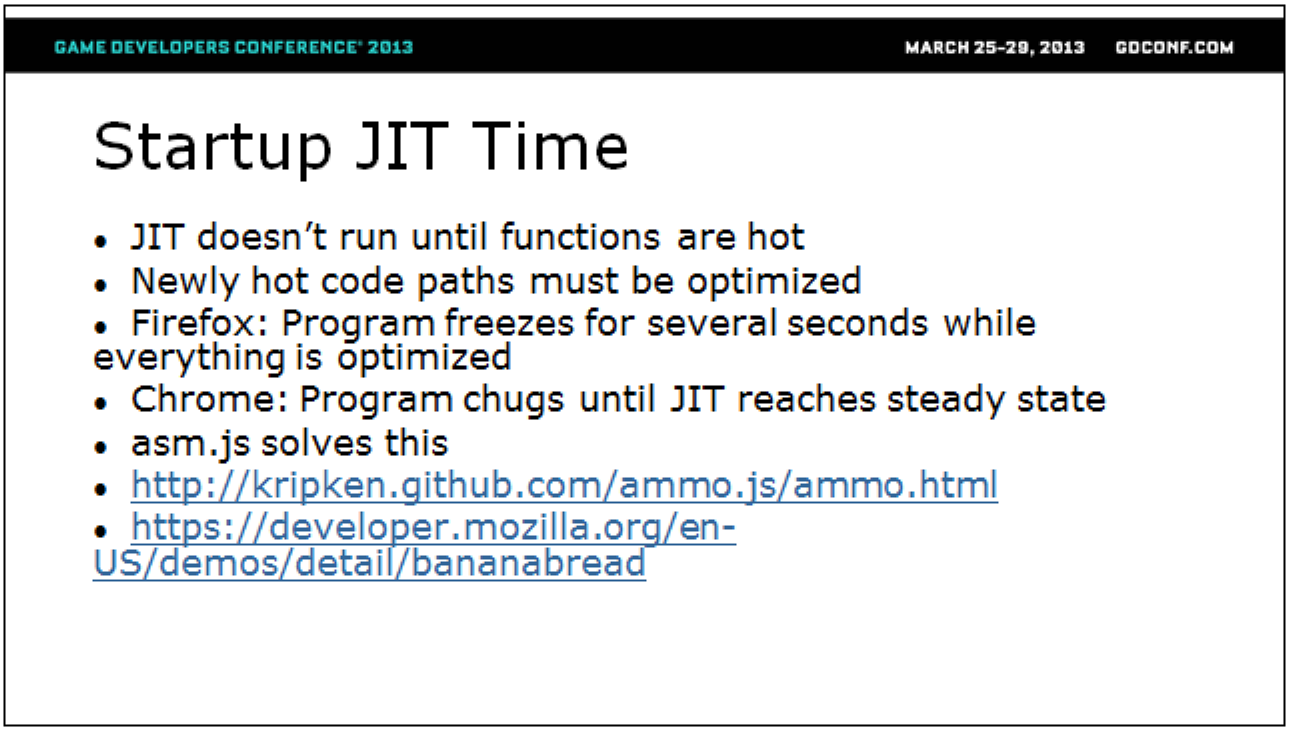
## SIMD

- No native support in JavaScript
- Emscripten converts to scalar operations



## Code Size

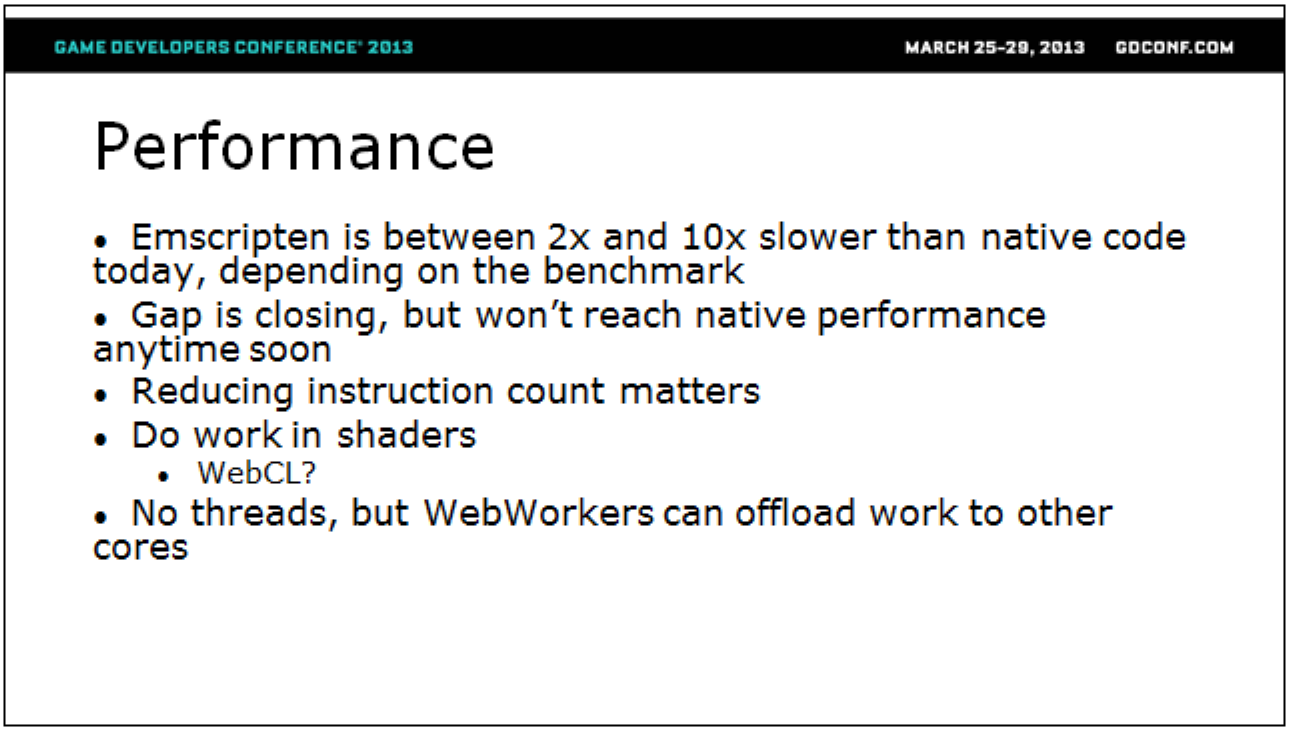
- In our tests, generated JavaScript is about the same size as 32-bit x86
- But people don't expect a web page to download a 6 MB executable
- Be aware of JS parse/compile times
- Build time is linear with code size
- Disable std::cin, std::cout, and std::cerr
  - Saves 100 KB of code size
- -fno-exceptions
- embind was written to generate small code



GAME DEVELOPERS CONFERENCE® 2013 MARCH 25-29, 2013 GDCONF.COM

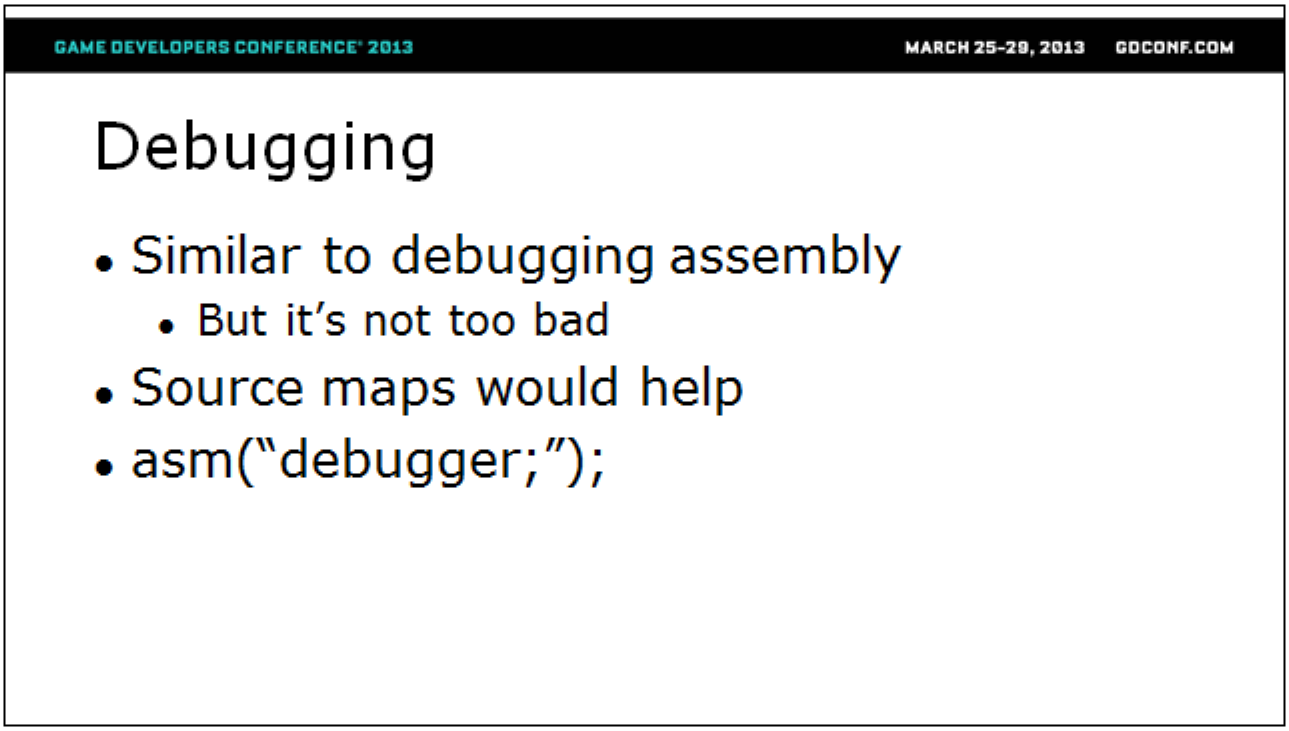
## Startup JIT Time

- JIT doesn't run until functions are hot
- Newly hot code paths must be optimized
- Firefox: Program freezes for several seconds while everything is optimized
- Chrome: Program chugs until JIT reaches steady state
- asm.js solves this
- http://kripken.github.com/ammo.js/ammo.html
- https://developer.mozilla.org/en-US/demos/detail/bananabread



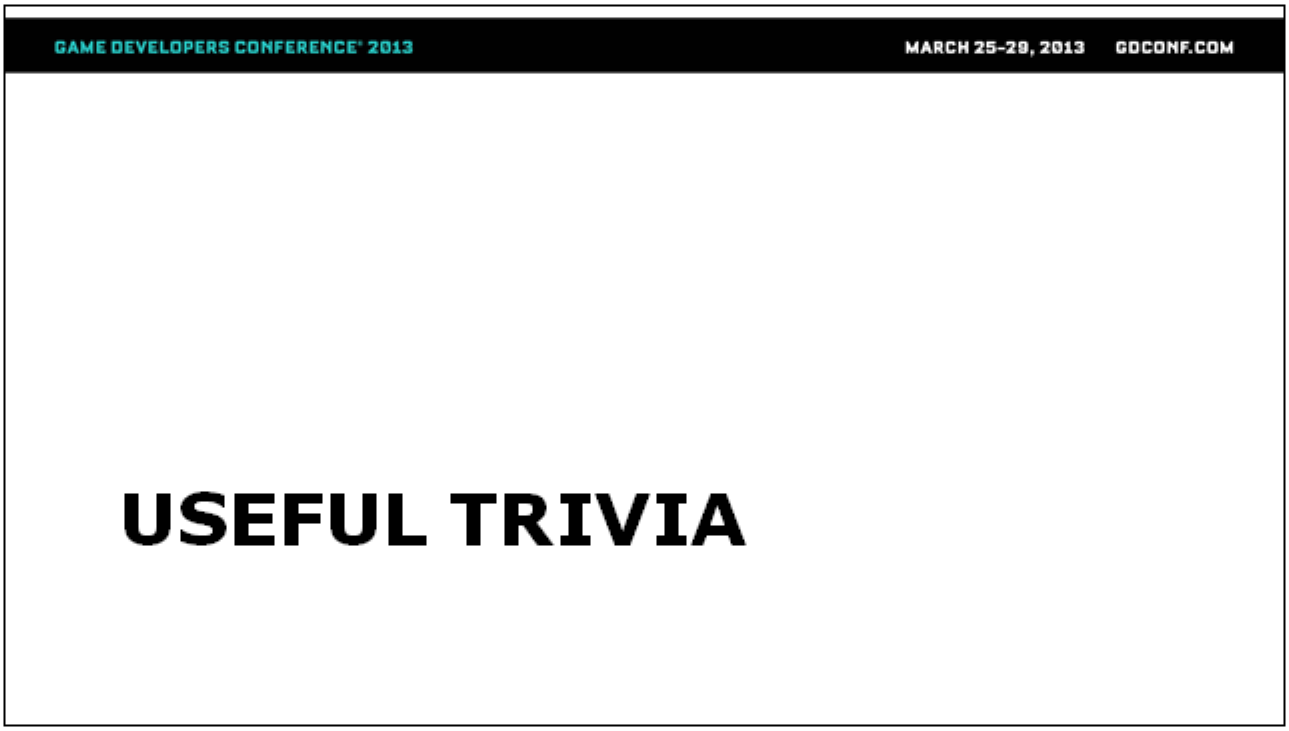
## Performance

- Emscripten is between 2x and 10x slower than native code today, depending on the benchmark
- Gap is closing, but won't reach native performance anytime soon
- Reducing instruction count matters
- Do work in shaders
  - WebCL?
- No threads, but WebWorkers can offload work to other cores

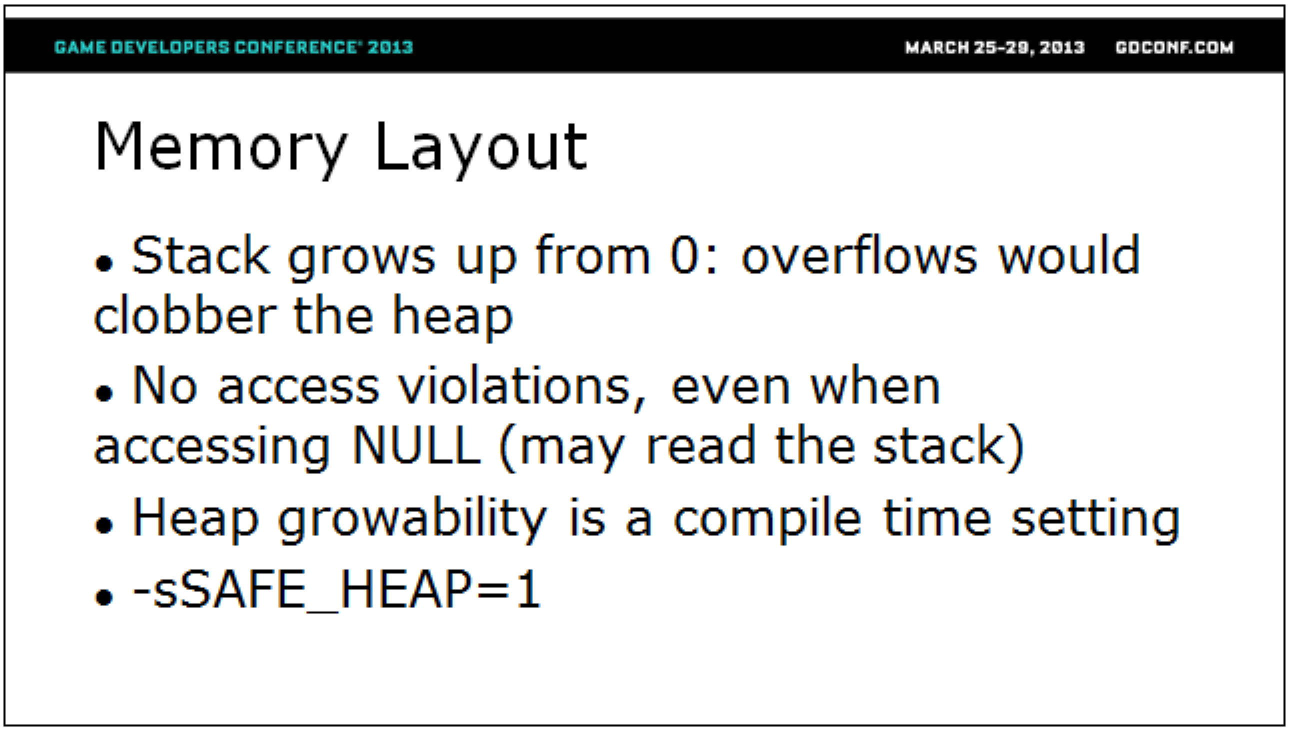


## Debugging

- Similar to debugging assembly
  - But it’s not too bad
- Source maps would help
- asm(“debugger;”);

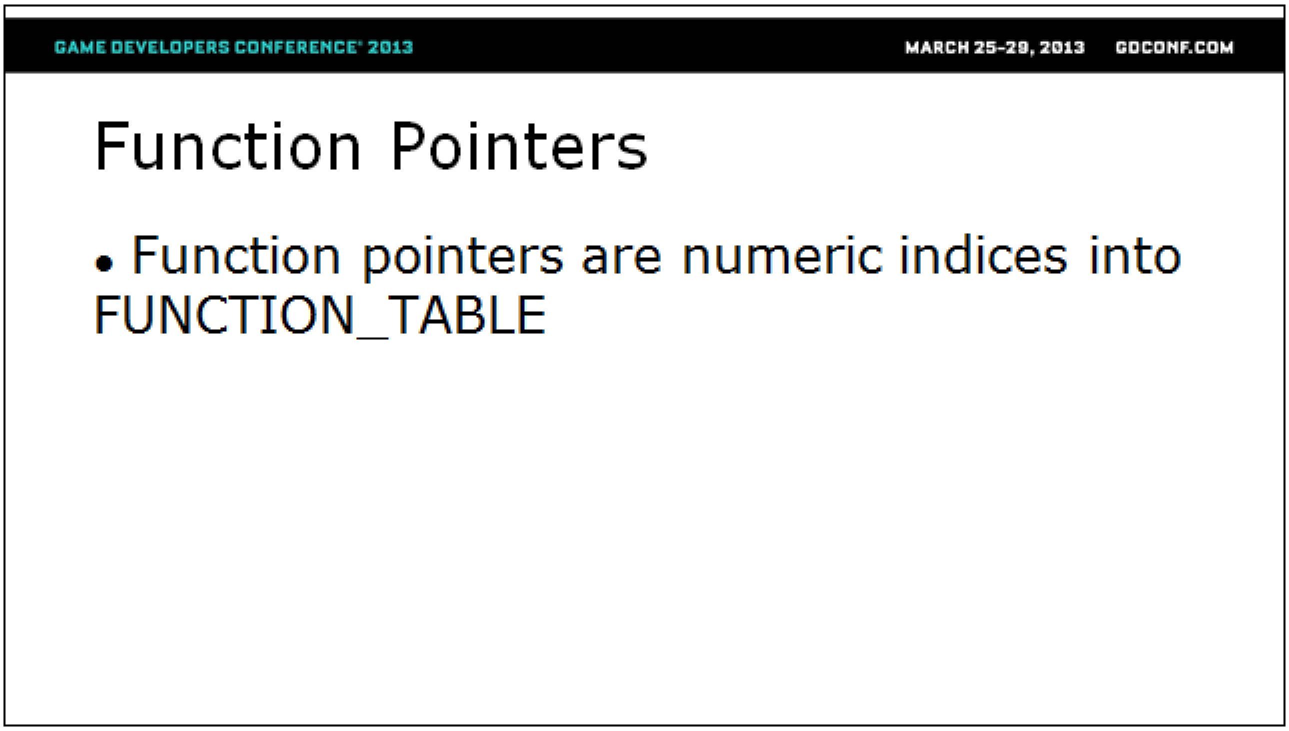


## USEFUL TRIVIA



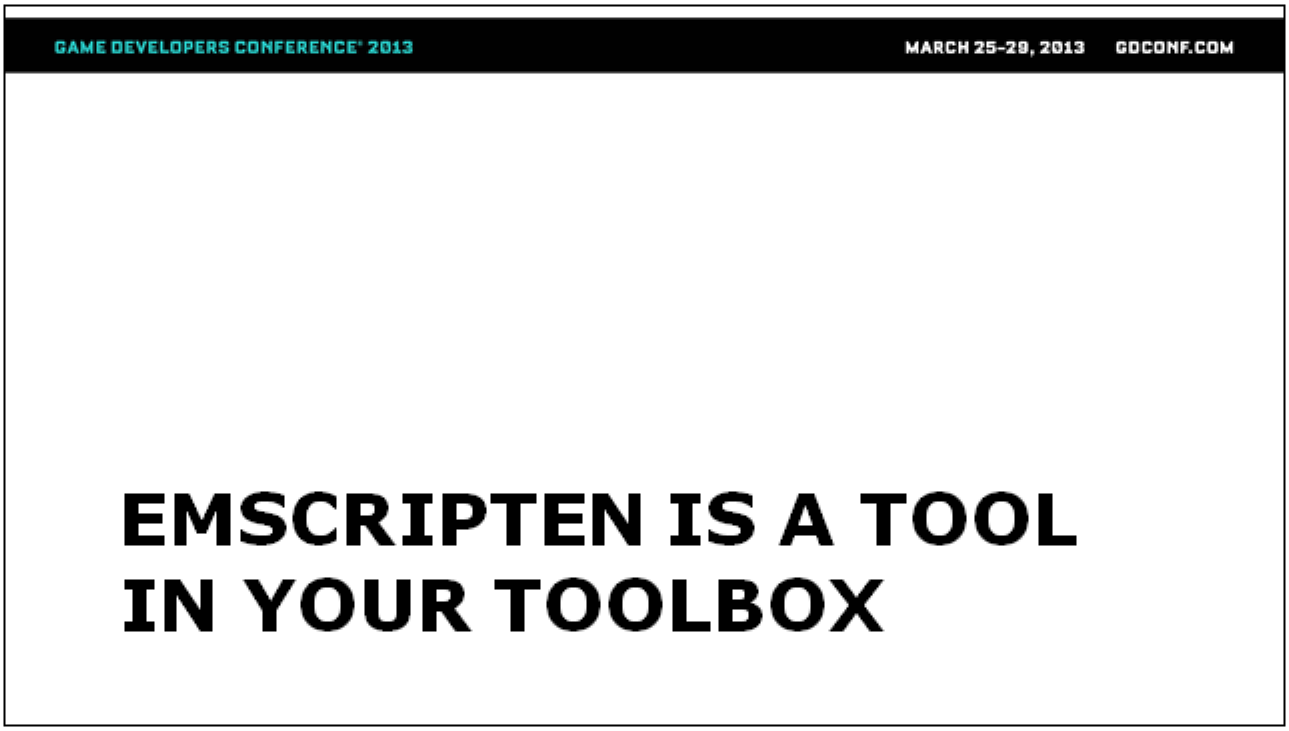
## Memory Layout

- Stack grows up from 0: overflows would clobber the heap
- No access violations, even when accessing NULL (may read the stack)
- Heap growability is a compile time setting
- -sSAFE_HEAP=1

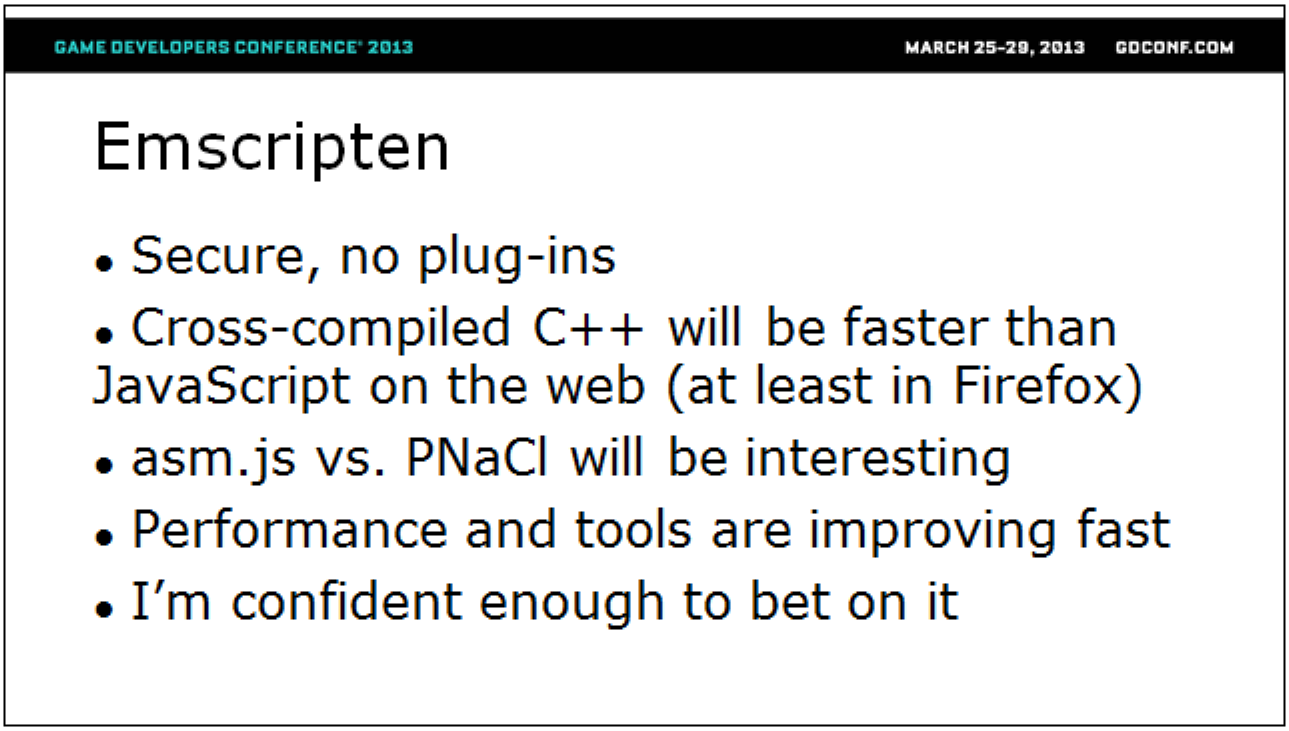


## Function Pointers

- Function pointers are numeric indices into FUNCTION_TABLE

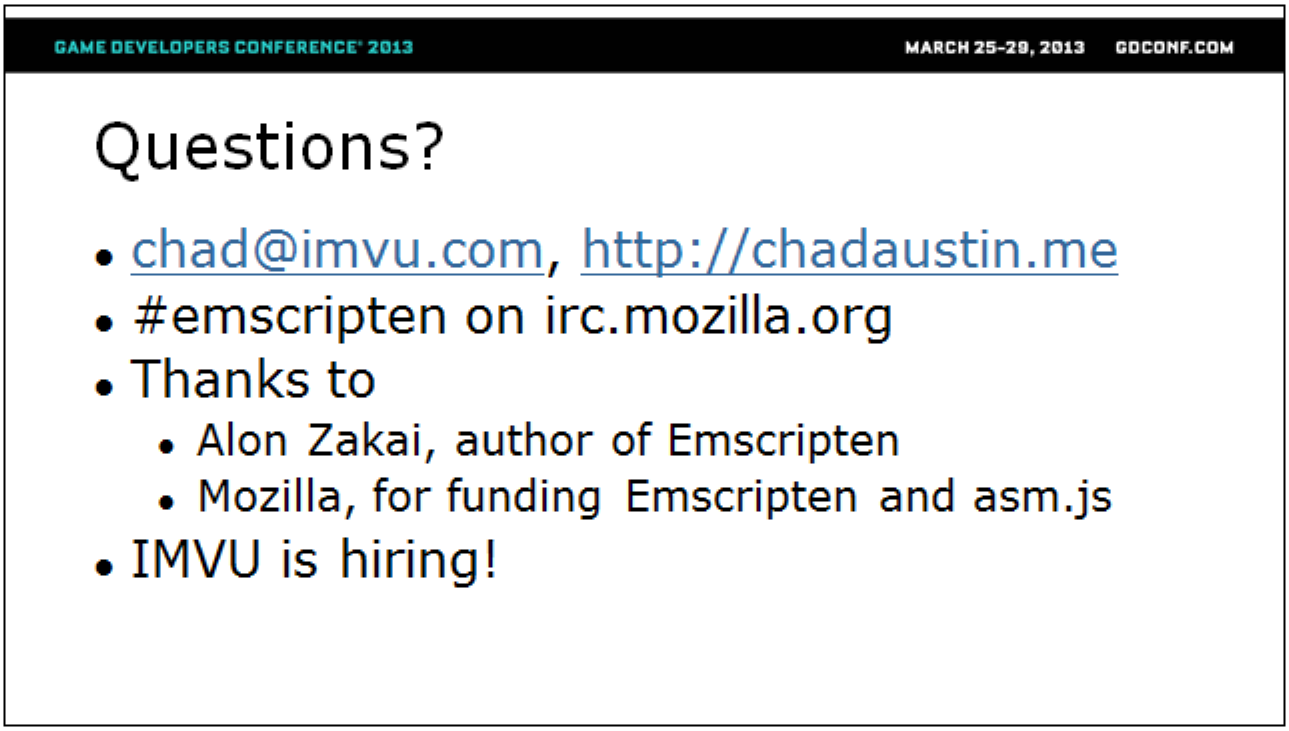


## EMSCRIPTEN IS A TOOL IN YOUR TOOLBOX



## Emscripten

- Secure, no plug-ins
- Cross-compiled C++ will be faster than JavaScript on the web (at least in Firefox)
- asm.js vs. PNaCl will be interesting
- Performance and tools are improving fast
- I'm confident enough to bet on it



## Questions?

- chad@imvu.com, http://chadaustin.me
- #emscripten on irc.mozilla.org
- Thanks to
  - Alon Zakai, author of Emscripten
  - Mozilla, for funding Emscripten and asm.js
- IMVU is hiring!