CROSS COMPILING WITH JAVASCRIPT

Miguel Angel Pastor
Halfbrick

Presentation

Miguel Angel Pastor Manuel

- 15+ years game development (Pyro Studios, Sony, Lucas Arts…)
- Optimization skills, C++/ASM
- Emuscener, worked on several hobbyist emulators
- Onan Games co-founder
- Halfbrick Lead Web Developer

iOS Game Development

- Angry Birds
  - C++, OpenGL ES 1.1, Box2D
- Fruit Ninja
  - C++, OpenGL ES 1.1
  - Fast pace game
  - 3D assets

C++

- Why C++?
  - Speed
  - Excellent debugging tools
  - Tons of Open Source libraries
  - Statically typed
  - Compiled
  - "Cross platform"
  - Traditionally consoles only supported C++/ASM
  - Personally, don’t like javascript

Web Development

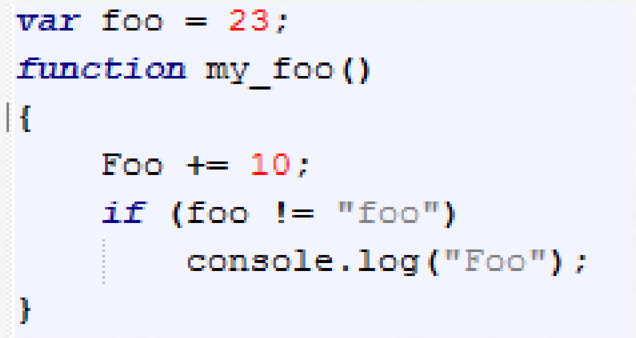
- No C++ support
- HTML5 & WebGL Open formats
  - Javascript
  - WebGL 3D API
- Flash
  - AS3
  - Stage3D 3D API

Web Development

- Angry Birds Chrome
  - Manually converted C++ -> Java
  - Java -> Javascript using GWT
  - Several months development
  - Hard to update

Javascript

- Dynamic
- Weakly typed
- Garbage collector
- Slow
    - V8 engine
    - Best case ~5x slower than C++
- Debugging tools not mature
- Easy to introduce bugs



```
var foo = 23;
function my_foo()
{
    Foo += 10;
    if (foo != "foo")
        console.log("Foo");
}
```

Javascript

```
var g_foo = 23; // number
function my_foo()
{
    g_foo +=10; //number
    g_foo = "foo"; //string
    g_foo = []; //array
    g_foo.m_foo = 10; //dynamic member
}
```

HTML5/WebGL

- PC ready
  - Chrome/Firefox
- iPhone 3GS quality games
- Javascript is bottle neck
- PC GPU faster than mobile
- C++ not supported
  - use cross compiling techniques

Mandreel

Our technology:

mandreel

What is Mandreel?

- It's a Platform, not only a compiler
- Converts C++/ObjC to HTML5 and Flash
- It works in all the browsers
- Automatic source code conversion
  - Same game, same functionality
  - Add new features in the iOS version
  - Feature in the web version automatically
- Conversion process only a few days
  - Less time, less money, publish faster

Published videogames

- Monster Dash
  - Developer: Halfbrick
  - Source: iOS
  - Target: HTML5

- Band Stars
  - Developer: Six Foot Kid
  - Target: HTML5

Published videogames

- Bug Village
  - Developer: Glu Mobile
  - Source: iOS
  - Target: HTML5

- A Space Shooter for free
  - Developer: Frima Studio
  - Source: iOS
  - Target: HTML5

Cross platform

- Mandreel platform
  - OpenGL ES 1.1/2.0
  - Custom audio API
  - Custom XHR API
  - C/C++
  - Custom event processing API
  - Visual studio integration

Cross platform

- Mandreel targets
  - PC
  - Android
    - Visual studio integration(debugging+compiling)
  - Windows 8 metro
    - OpenGL ES emulator on top of DX 11.1
  - Flash Stage3D
    - OpenGL ES emulator on top of Stage3D
  - HTML5 + WebGL

Cross compiling

- LLVM
  - C++ frontend
  - BSD type license
  - Custom JS Backend
- Visual studio Integration
  - New Mandreel platform

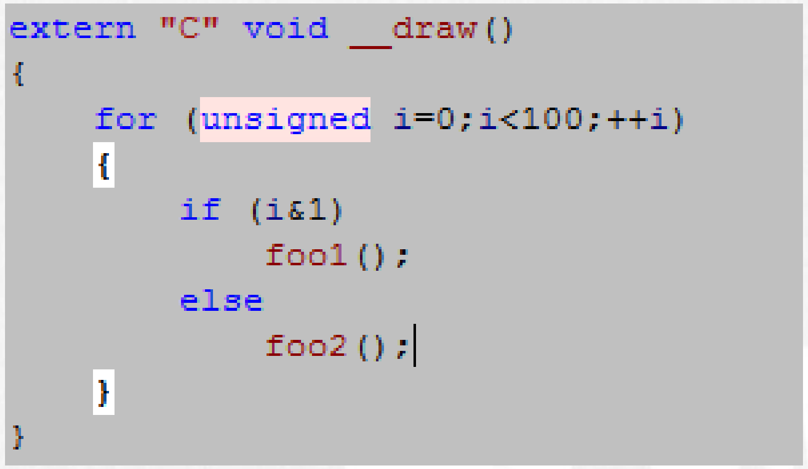
Cross compiling

- Javascript LLVM target
  - 32bit CPU
  - 32 integer registers
  - 32 floating point registers
  - Stack based function calling
  - aligned memory access ONLY
  - Float ops -> double precission

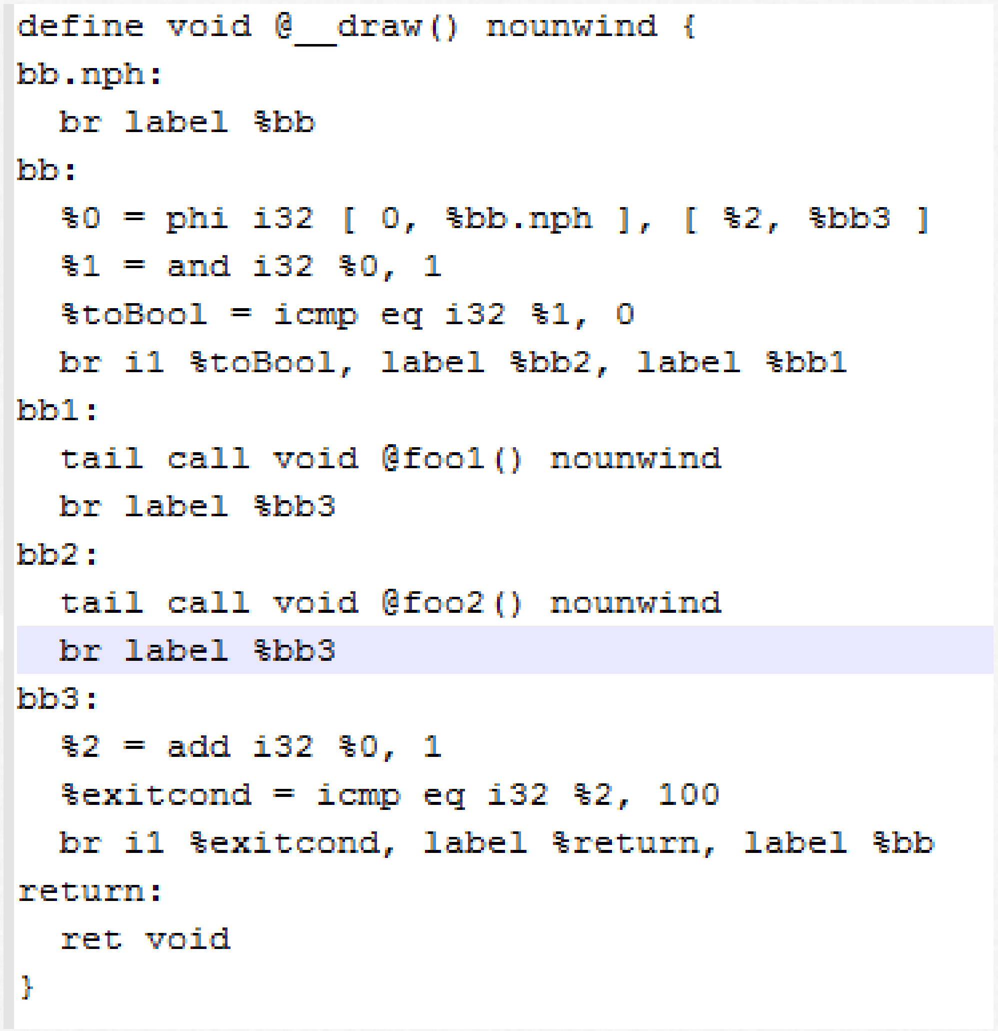
Cross compiling

- Javascript no goto sentence
  - Clever use of continue label/ break label
  - most complex piece
  - Inspired on emscripten relooper
    - download paper

Cross compiling

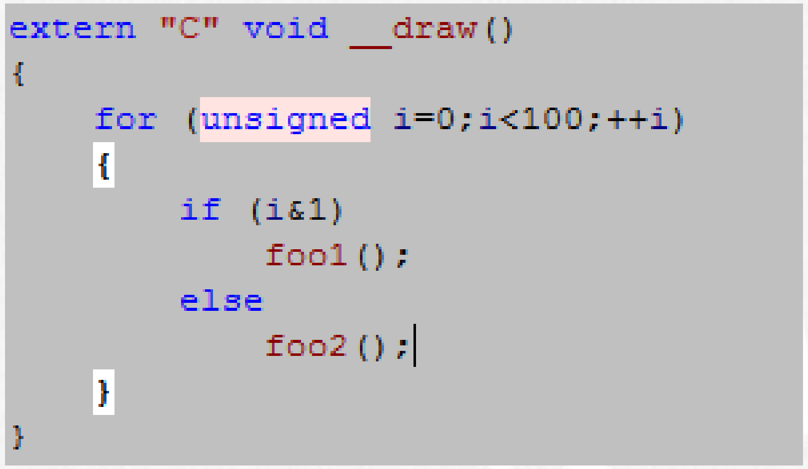


```
extern "C" void __draw()
{
    for (unsigned i=0;i<100;++i)
    {
        if (i&1)
            foo1();
        else
            foo2();
    }
}
```

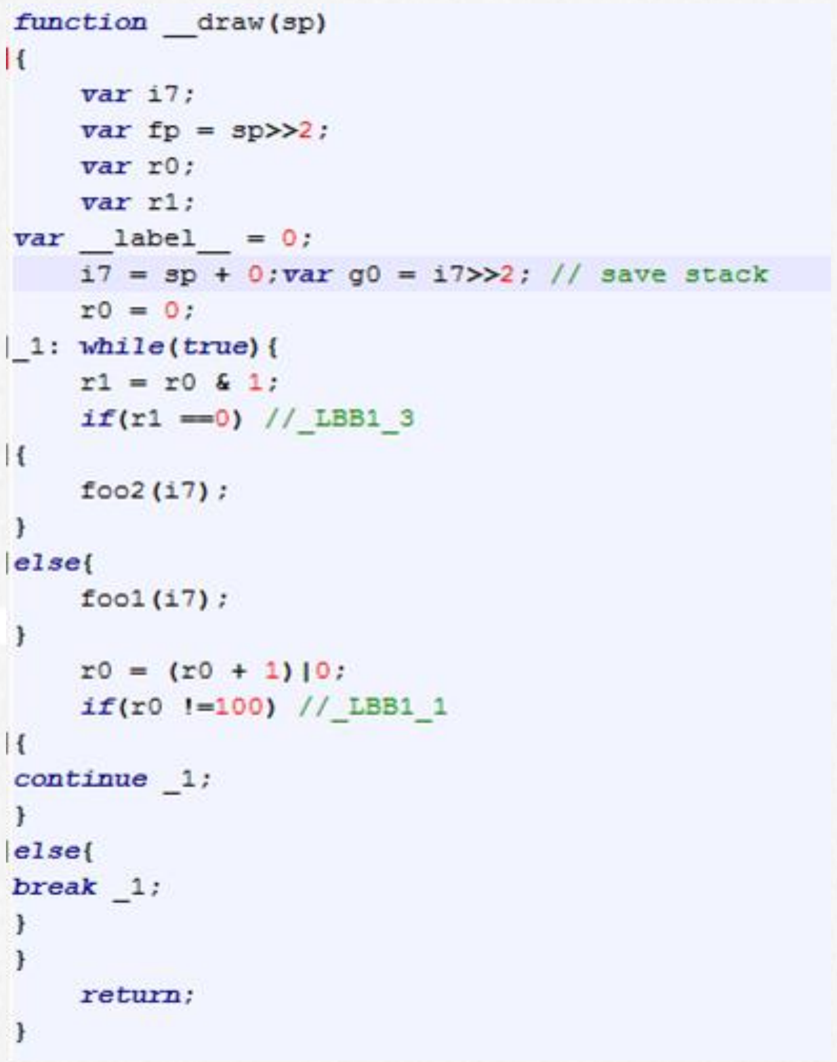


```
define void @__draw() nounwind {
bb.nph:
  br label %bb
bb:
  %0 = phi i32 [ 0, %bb.nph ], [ %2, %bb3 ]
  %1 = and i32 %0, 1
  %toBool = icmp eq i32 %1, 0
  br i1 %toBool, label %bb2, label %bb1
bb1:
  tail call void @foo1() nounwind
  br label %bb3
bb2:
  tail call void @foo2() nounwind
  br label %bb3
bb3:
  %2 = add i32 %0, 1
  %exitcond = icmp eq i32 %2, 100
  br i1 %exitcond, label %return, label %bb
return:
  ret void
}
```

Cross compiling



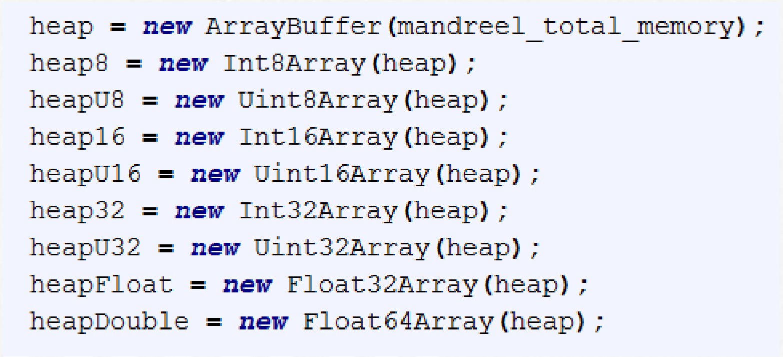
```
extern "C" void __draw()
{
    for (unsigned i=0;i<100;++i)
    {
        if (i&1)
            foo1();
        else
            foo2();
    }
}
```



```
function __draw(sp)
{
    var i7;
    var fp = sp>>2;
    var r0;
    var r1;
var __label__ = 0;
    i7 = sp + 0;var g0 = i7>>2; // save stack
    r0 = 0;
_1: while(true){
    r1 = r0 & 1;
    if(r1 ==0) //_LBB1_3
{
    foo2(i7);
}
else{
    foo1(i7);
}
    r0 = (r0 + 1)|0;
    if(r0 !=100) //_LBB1_1
{
continue _1;
}
else{
break _1;
}
}
    return;
}
```

Cross compiling

- Memory access
  - Typed arrays
  - IE 10/Chrome/Firefox/Safari
  - No support for unaligned access
- Memory model
  - Big ArrayBuffer
  - Allocated during init
  - can't grow/shrink
  - malloc/free use that buffer
  - pessimistic allocation



```
heap = new ArrayBuffer(mandreel_total_memory);
heap8 = new Int8Array(heap);
heapU8 = new Uint8Array(heap);
heap16 = new Int16Array(heap);
heapU16 = new Uint16Array(heap);
heap32 = new Int32Array(heap);
heapU32 = new Uint32Array(heap);
heapFloat = new Float32Array(heap);
heapDouble = new Float64Array(heap);
```

Cross compiling

```
struct TData
{
    float f_data;
    unsigned int i_data;
};
extern "C" TData* data;
extern "C" void __draw()
{
    data->f_data = 192.f;
    data->i_data = 1234;
}
```

```
define void @__draw() nounwind {
entry:
  %0 = load %struct.TData** @data, align 4
  %1 = getelementptr inbounds %struct.TData* %0, i32 0, i32 0
  store float 1.920000e+002, float* %1, align 4
  %2 = load %struct.TData** @data, align 4
  %3 = getelementptr inbounds %struct.TData* %2, i32 0, i32 1
  store i32 1234, i32* %3, align 4
  ret void
}
```

```
function __draw(sp)
{
    var i7;
    var fp = sp>>2;
    var r0;
    var r1;
var __label__ = 0;
    i7 = sp + 0;var g0 = i7>>2;
    r0 = data;
    r0 = r0 >> 2;
    r1 = heap32[(r0)];
    r1 = r1 >> 2;
    heap32[(r1)] = 1128267776;
    r0 = heap32[(r0)];
    r0 = r0 >> 2;
    heap32[(r0+1)] = 1234;
    return;
}
```

Cross compiling

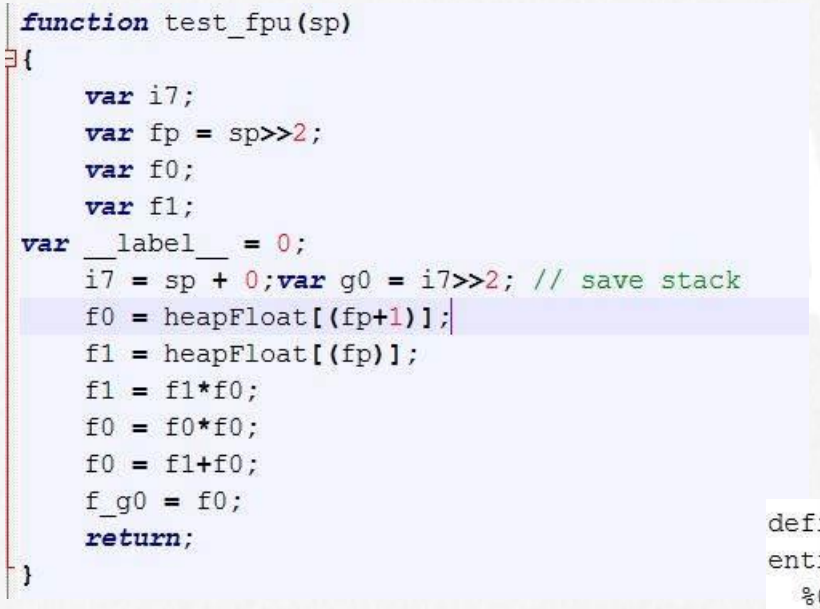
- No GC problems
  - No dynamic allocation during runtime = stable framerate
  - Malloc/free uses preallocated buffer
  - Faster than JS hand written code

Cross compiling

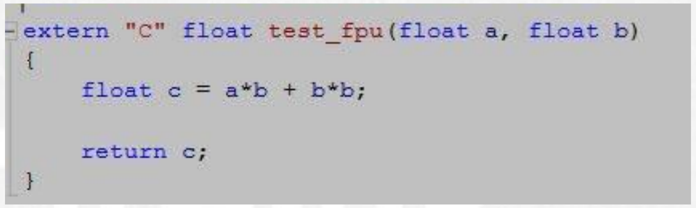
- Javascript variables double precision
  - Integer arithmetic operation problems
  - JS maximum integer 2^53
  - r0 = 0xffffffff, r3 = 0x1
  - r2 = r0 + r3 -> integer add can fail
  - r2 = 0x100000000 -> bad result
  - r2 = (r0 + r3)|0 -> force integer result
  - r2 = 0x0 -> good result

Cross compiling

- Javascript variables double precision
  - Floating point ops double promoted
  - Result different from C/C++ native

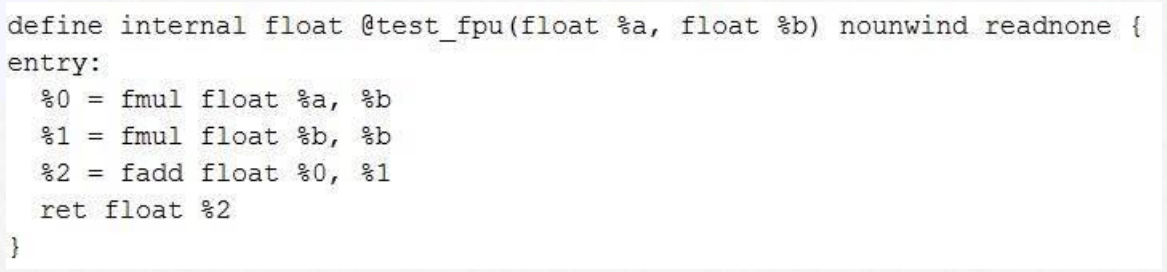


```
function test_fpu(sp)
{
     var i7;
     var fp = sp>>2;
     var f0;
     var f1;
var __label__ = 0;
     i7 = sp + 0;var g0 = i7>>2; // save stack
     f0 = heapFloat[(fp+1)];
     f1 = heapFloat[(fp)];
     f1 = f1*f0;
     f0 = f0*f0;
     f0 = f1+f0;
     f_g0 = f0;
     return;
}
```



```
extern "C" float test_fpu(float a, float b)
{
    float c = a*b + b*b;

    return c;
}
```



```
define internal float @test_fpu(float %a, float %b) nounwind readnone {
entry:
  %0 = fmul float %a, %b
  %1 = fmul float %b, %b
  %2 = fadd float %0, %1
  ret float %2
}
```

Cross compiling

- Big JS files
  - some games ~20 megabytes
  - Obfuscated + YUI compressor ~8 MB
  - Using LZMA ~ 1.5 MB
  - Decompress at load
  - Cache locally using FileSystem/IndexedDB

C++, why we care

- It's fast on iOS/Android
- Console development main language
- HTML5 mobile not ready yet
- Use C++ on mobile and JS on PC
  - Use Mandreel
  - PC users and installing = problem

C++, why we care

- JS not suitable for large projects
- Primitive debugging tools
- JS no typed, easy to introduce bugs
- JS is dynamic, errors spotted at runtime
- Garbage collector, no solid framerate

JS and cross compiling

- Hard to optimize
  - Everything is dynamic
- Browser dependent performance
  - User experience browser dependent
  - FLASH same experience across browsers
- No solid framerate
  - New code comes in, slow down

Questions

Contact

Miguel Angel Pastor

mpastor@halfbrick.com

www.halfbrick.com

www.baktery.com