WebAssembly
Of Portability, and Performance

Deepti Gandluri, Google
WebAssembly (abbreviated Wasm) is a safe, portable, low-level code format designed for efficient execution and compact representation.
In a Nutshell

1. Fast, safe, and well-defined semantics
2. Efficient and portable representation
3. MVP Supported on 4 Major Browsers, node.js, and other embeddings
4. Open Standard, developed in by a W3C community group
WebAssembly is here!

“The AutoCAD team took a 35 year old code base...and got it to compile directly in a browser using WebAssembly.... So now the power of AutoCAD is just a link away.” #io18 #io2018keynote

WebAssembly in Blockchains #206

Moving “VirtualWorld3d” from pure JavaScript to WebAssembly

Virtual World 3d

As a non-web platform, blockchains have distinct requirements. We have compiled notes from projects across the space (Ethereum, Parity, Dfinity, and Truebit), and will use this session to give an overview of approaches and challenges to the wider community. Referenced in CG-04.
WebAssembly, but not for the Web?
WebAssembly and Node.js

1. WebAssembly module can be imported into a Node.js app.

2. Sandboxed environment separated from the host runtime using fault isolation techniques.

3. Build once, run on the server or the client, ship single binary to any platform.

4. Not faster than native, no direct access to other node.js modules, and native libraries.
BRING BACK THE 90's
The CImg Library is a small and open-source C++ toolkit for image processing, designed with these properties in mind:

**Usefulness**

CImg defines classes and methods to manage images in your own C++ code. You can use CImg to load/save various file formats, access pixel values, display/transform/filter images, draw primitives (text, faces, curves, 3d objects,...), compute statistics, manage user interactions on images, and so on.

**Genericity**

CImg defines a single image class able to represent datasets having up to 4-dimensions (from 1d scalar signals to 3d hyperspectral volumetric images), with template pixel types (bool,char,int,float,...). It also handles image collections and sequences.

**Portability**

CImg is self-contained, thread-safe and highly portable. It fully works on different operating systems (Linux, Windows, MacOS X, *BSD,...) and is compatible with various C++ compilers (Visual C++, g++, clang++, icc,...).

**Simplicity**

CImg is lightweight. It is made of a single header file cimg.h that must be included in your C++ source. It defines only four different classes, encapsulated in the namespace cimg_library. It can be compiled

http://cimg.eu/
https://github.com/dtschump/CImg
Compiling from C++ to WebAssembly

- Setup the toolchain
- Identify which parts of the application need to be built
- Compile to Wasm with em++
- JS Integration

On the Web

- Host a simple server
- Launch!
Tiles and Triangles

Refactor C code to align with memory/export requirements, Compile void* item_filled_triangles()

```
em++ --bind --std=c++11 -I/Users/gdeepti/Source/wasm-jpeg-ijg/CImg/ triangles.cpp -s WASM=1 -s ALLOW_MEMORY_GROWTH=1 -s "EXTRA_EXPORTED_RUNTIME_METHODS=['addOnPostRun']" -o triangles.js
```

Figure out what to expose to JS, exported functions, embind etc.

```javascript
console.time('triangles');
const triangles = Module.draw_triangles(height, width);
console.timeEnd('triangles');

console.time('canvas put image data');
const imageData = new ImageData(new Uint8ClampedArray(triangles), width, height);
const context = canvas.getContext('2d');
context.putImageData(imageData, 0, 0);
console.timeEnd('canvas put image data');
```
Intercept Key presses

Vary the number of tiles, and triangles

Rotation speed, color variations

Use other library functions for image processing
A quick peek into V8 Internals
JavaScript Pipeline

Script Text → Parser → AST → Ignition → Bytecode → Execution

- Parsing
- Optimization
- TurboFan
- Type Feedback
- Interpreter
- Execution

Optimized Code → deopt
WebAssembly Pipeline

Wasm Module → Decode → Validate → Liftoff

Graph builder → TurboFan → Optimized Code

Baseline Code
Performance Caveats

1. Emerging standard, so there are going to be cases that are not optimized yet.

2. Works well for computation intensive workloads, like image/video processing, game engines, encryption, scientific visualizations, and simulations.

3. Performance nuanced, and has many aspects.
Post MVP Features
WebAssembly Threads and Atomics

1. Low-level building blocks for pthreads-style shared memory: shared memory between threads, atomics and futexes.

2. New atomic memory operators, including loads/stores, will follow a model compatible with the C++11 memory model (with some differences).

3. The WebAssembly Threads feature allows multiple WebAssembly instances in separate Web Workers to share a single WebAssembly.Memory object.

4. Implementations in multiple browsers, ongoing work to make applications work in a multi-threaded context.
Future Features, and Proposals in Flight

- Fixed-width SIMD
- Exception handling
- Garbage collection
- ECMAScript module integration
- Tail Calls
- Host bindings

And more at: https://webassembly.org/docs/future-features/
Thanks!

WebAssembly Information - http://webassembly.org/

Community Group - https://www.w3.org/community/webassembly/

Code labs:

An Introduction to Web Assembly - https://codelabs.developers.google.com/codelabs/web-assembly-intro