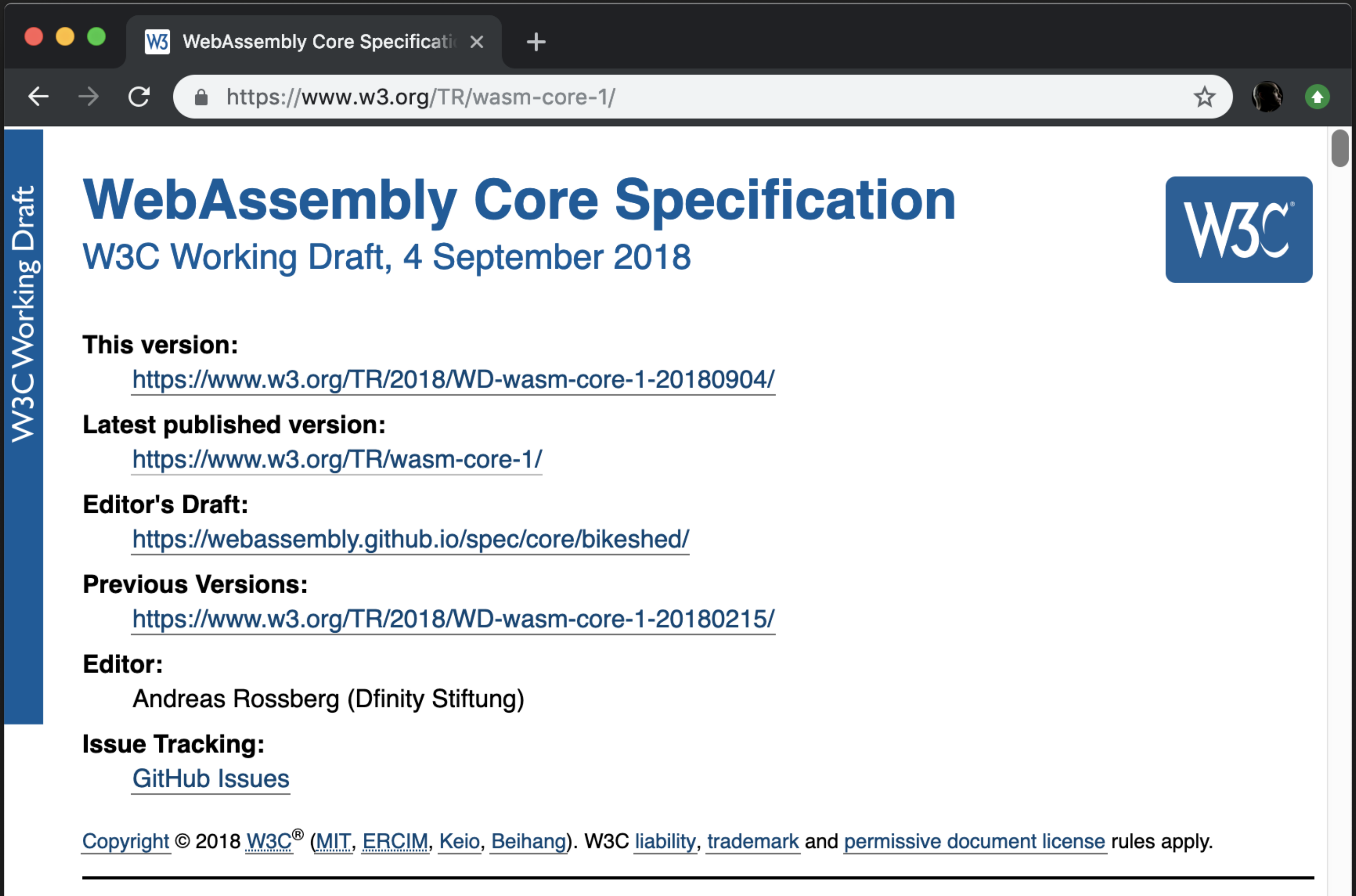
What is WebAssembly?

WebAssembly is a standard that defines a binary format and a corresponding assembly-like text format for executables used by web pages.

Let's pick that apart

Standard?



WebAssembly Core Specificati
https://www.w3.org/TR/wasm-core-1/
W3C Working Draft

WebAssembly Core Specification

W3C Working Draft, 4 September 2018

This version:
https://www.w3.org/TR/2018/WD-wasm-core-1-20180904/

Latest published version:
https://www.w3.org/TR/wasm-core-1/

Editor's Draft:
https://webassembly.github.io/spec/core/bikeshed/

Previous Versions:
https://www.w3.org/TR/2018/WD-wasm-core-1-20180215/

Editor:
Andreas Rossberg (Dfinity Stiftung)

Issue Tracking:
GitHub Issues

Binary format?

```
$ hexdump -C web/wasm/hello.wasm
00000000  00 61 73 6d 01 00 00 00  01 05 01 60 00 01 7f 03  |.asm.......`....|
00000010  02 01 00 05 03 01 00 10  06 11 02 7f 00 41 80 80  |.............A..|
00000020  c0 00 0b 7f 00 41 80 80  c0 00 0b 07 2d 04 06 6d  |.....A......-..m|
00000030  65 6d 6f 72 79 02 00 0b  5f 5f 68 65 61 70 5f 62  |emory...__heap_b|
00000040  61 73 65 03 00 0a 5f 5f  64 61 74 61 5f 65 6e 64  |ase...__data_end|
00000050  03 01 05 68 65 6c 6c 6f  00 00 0a 06 01 04 00 41  |...hello.......A|
00000060  2a 0b                                             |*.|
00000062
```

Assembly language?

```
$ wasm-dis web/wasm/hello.wasm
(module
 (type $0 (func (result i32)))
 (memory $0 16)
 (global $global$0 i32 (i32.const 1048576))
 (global $global$1 i32 (i32.const 1048576))
 (export "memory" (memory $0))
 (export "__heap_base" (global $global$0))
 (export "__data_end" (global $global$1))
 (export "hello" (func $0))
 (func $0 (; 0 ;) (type $0) (result i32)
  (i32.const 42)
 )
)
```

Executable?

The definition is basically something that makes computers do something

used by web pages?

Lies! Despite the name it can be used in all kinds of non-web scenarios

We'll be sticking to the web today though

Creating a WebAssembly module

It's possible to create modules from the assembly format.

But it's much easier to use a language and compile it to WebAssembly

There are **many** languages that can compile to WebAssembly

AssemblyScript, Astro, Brainfuck, C, C♯ C++, D, Elixir, Faust, Forest, Forth, Go, Grain, Haskell, Java, JavaScript, Julia, Idris, Kotlin/Native, Kou, Lua, Nim, Ocaml, Perl, PHP, Plorth, Poetry, Python, Prolog, Ruby, Rust, Scheme, Wah, Walt, Wam, Xlang, Zig

The examples will be in Rust

Because it was time to learn a new language anyway

...and Rust is pretty cool

...and the Rust support for WebAssembly is pretty great!

Small JavaScript to Rust dictionary

npm	cargo
package	crate
www.npmjs.com/package	crates.io

Hello World

A minimal program that outputs "42" into a `<div>`

Project contents

```
hello-wasm/
├── build.sh
├── crate
│   ├── Cargo.toml
│   └── src
│       └── lib.rs
└── web
    ├── index.html
    └── wasm
```

build.sh

```
mkdir -p web/wasm
cd crate
cargo build --release &&\
  wasm-opt --strip-producers --strip \
    --remove-unused-module-elements \
    target/wasm32-unknown-unknown/release/hello.wasm -o ../web
```

CODE BLOCK DELIMITER

hello-wasm/crate/cargo.toml

SOURCE FILE NOT FOUND

index.html

```
<!doctype html>
<html lang=en>
<head><meta charset="utf-8"/><title>Hello</title></head>
<body>
<div id="hello">-</div>
<script>
  WebAssembly.instantiateStreaming(fetch('wasm/hello.wasm'))
    .then(wasm => {
      const mod = wasm.instance
      const main = mod.exports.hello
      document.getElementById("hello").innerText = main()
    })
</script>
</body>
</html>
```

lib.rs

```rust
#[no_mangle]
fn hello() -> i32 {
    42
}
```

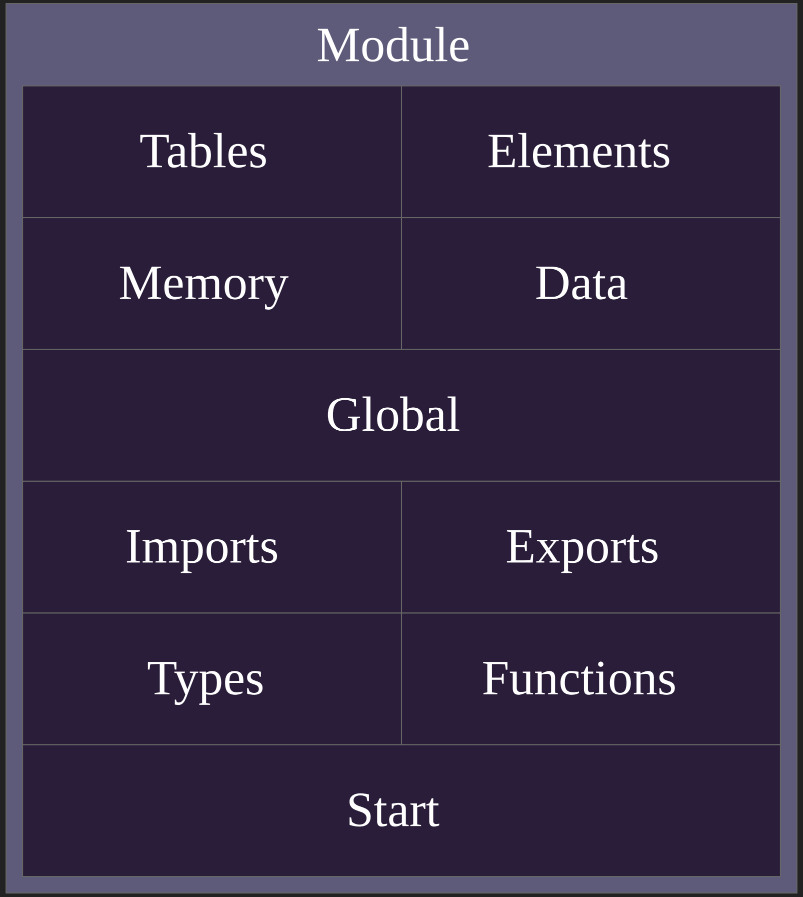
Let's see if it works...

That was...underwhelming.

Let's take a step back

```
$ wasm-dis web/wasm/hello.wasm
(module
 (type $0 (func (result i32)))
 (memory $0 16)
 (global $global$0 i32 (i32.const 1048576))
 (global $global$1 i32 (i32.const 1048576))
 (export "memory" (memory $0))
 (export "__heap_base" (global $global$0))
 (export "__data_end" (global $global$1))
 (export "hello" (func $0))
 (func $0 (; 0 ;) (type $0) (result i32)
  (i32.const 42)
 )
)
```

Anatomy of a WebAssembly Module



Module

Tables

Elements

Memory

Data

Global

Imports

Exports

Types

Functions

Start

Is it fast enough to use?

Let's device a test

- Use only features JS and WASM share
- Somewhat visually interesting

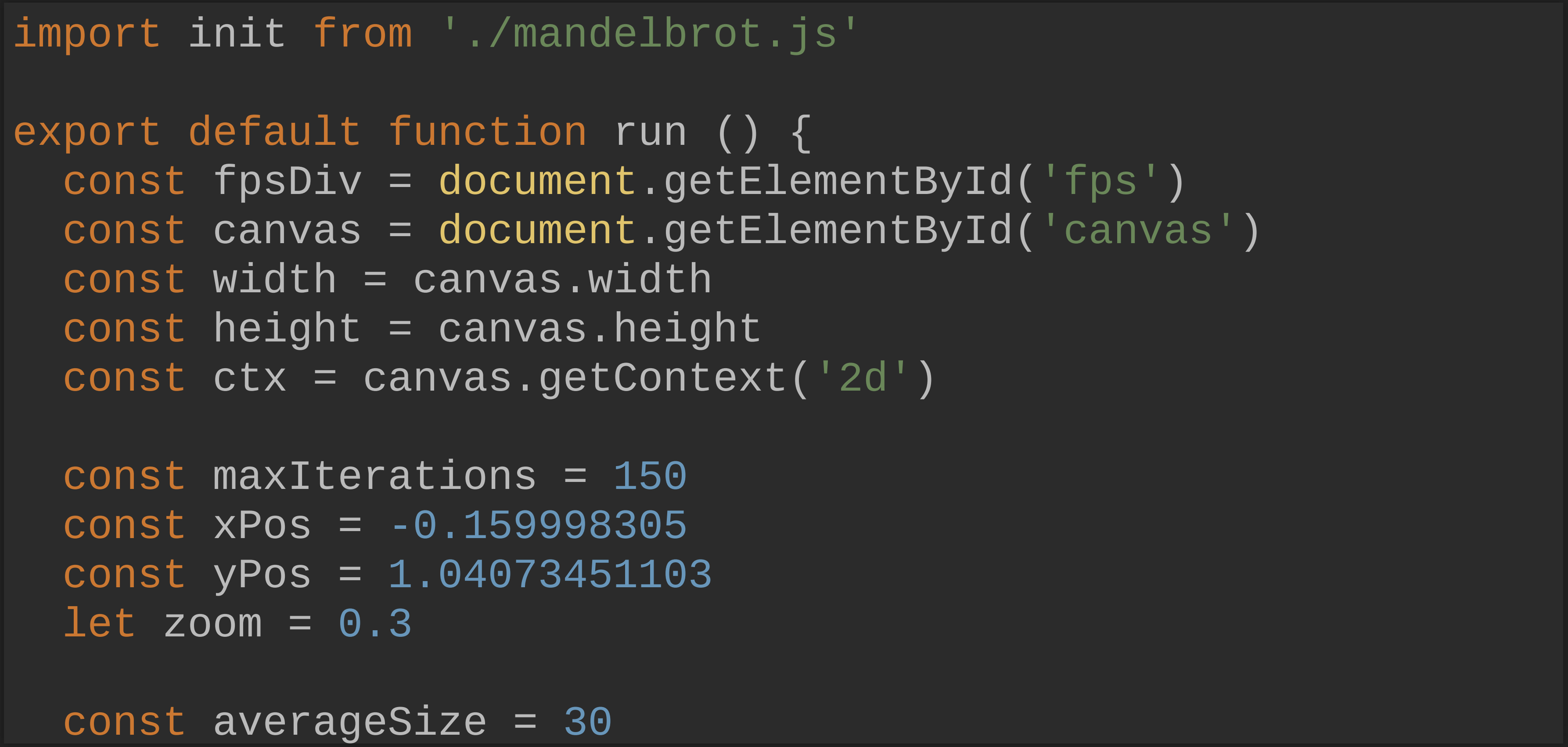
Mandelbrot rendering

- Render to canvas
- Use 64 bit floats for precision
- Max 150 iterations per pixel

Shared code

- `index.html`
- `main.js`

```
<!doctype html>
<html lang=en>
<head>
    <meta charset="utf-8"/>
    <title>JavaScript Mandelbrot</title>
    <style>
        canvas {
            padding: 0;
            margin: auto;
            display: block;
            border: black 1px solid;
            width: 1024px;
            height: 768px;
            position: absolute;
            top: 0;
```



```
import init from './mandelbrot.js'

export default function run () {
  const fpsDiv = document.getElementById('fps')
  const canvas = document.getElementById('canvas')
  const width = canvas.width
  const height = canvas.height
  const ctx = canvas.getContext('2d')

  const maxIterations = 150
  const xPos = -0.159998305
  const yPos = 1.04073451103
  let zoom = 0.3

  const averageSize = 30
```

JavaScript specific code

- `mandelbrot.js`

Rust specific code

- `mandelbrot.js`
- `lib.rs`

Let's see if it works...

That was...better!

Performance is pretty much identical

Can you do all the things you need to?

Let's device a test

- Somewhat visually interesting
- Do more in Rust

Project contents

```
mandelbrot-webgl
├── crate
│   ├── Cargo.toml
│   └── src
│       ├── lib.rs
│       ├── mandelbrot64.frag
│       ├── stats.rs
│       ├── util.rs
│       └── vertices.vert
├── dist
├── node_modules
│   └── You all know what's going on in here
├── package.json
├── web
│   ├── index.html
```

Using all the tools

- NPM
  - left-pad
- Webpack
  - wasm-pack
- Cargo
  - wasm-bindgen

Let's see if it works...

Did it work?

Yes...

How painful was it?

Quite.

Mostly due to WebGL APIs, Otherwise it was quite pleasant.

There are still a few sharp edges that needs cleaning up.

Most are to do with the JS language being untyped and Rust being strongly typed.

Rust also doesn't have a concept of `null`/`undefined` etc.

In particular the `requestAnimationFrame` callback code is horrendous

```
let f: Rc<RefCell<Option<_>>> = Rc::new(RefCell::new(None)
let g: Rc<RefCell<Option<_>>> = f.clone();
*g.borrow_mut() = Some(Closure::new(move || {

    ...
    request_animation_frame(f.borrow().as_ref().unwrap());
}));

request_animation_frame(g.borrow().as_ref().unwrap());
```

There are several projects creating wrapping API to improve these things

Browser support

caniuse says it's approaching 85%.

Probably enough for deploying internal apps today, and starting development of larger apps

But it's supposed to be faster? Why wasn't it faster?

It can be!

For larger apps, the payloads will be smaller

Streaming loading/compilation can make time-to-interactive shorter

Data heavy tasks, integer maths and tasks that can take advantage of SIMD instructions

It's very young tech still

Cool links I found

- A cartoon intro to WebAssembly
- Oxidizing Source Maps with Rust and WebAssembly
- Maybe you don't need Rust and WASM to speed up your JS
- Speed Without Wizardry
- Fast, Bump-Allocated Virtual DOMs with Rust and Wasm
- WebAssembly Load Times and Performance