

# Comparing WebAssembly and JavaScript for a web based photo editor

Sam Robbins

Durham University

# What is the project about?

For a long time, JavaScript was the only method for doing computation in web browsers, however in 2017, Web Assembly was introduced as an alternative [1].

## Definition (Web Assembly)

Web Assembly is a language that allows for the execution of languages that compile to assembly on the web

# Research Question

Finding in which cases Web Assembly offers a benefit over JavaScript, by implementing image processing algorithms using both mechanisms.

# Deliverables

- ▶ **Basic aim**

Compare the performance of one complex algorithm between Web Assembly and JPEG, for a complex algorithm, it should not simply loop over the pixels of the image, and instead be performing more sophisticated techniques.

- ▶ **Intermediate aim**

Extend the basic aim to multiple complex algorithms.

- ▶ **Advanced aim**

Look into tweaks that can be used to improve performance, such as compile time optimizations for Web Assembly.

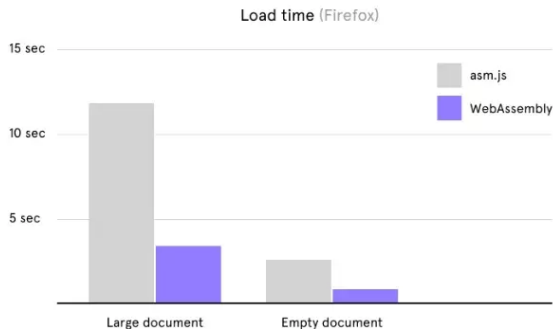
What has been done by others?

## Using Web Assembly for image processing — Next.js [2]

~~NEXT~~.JS

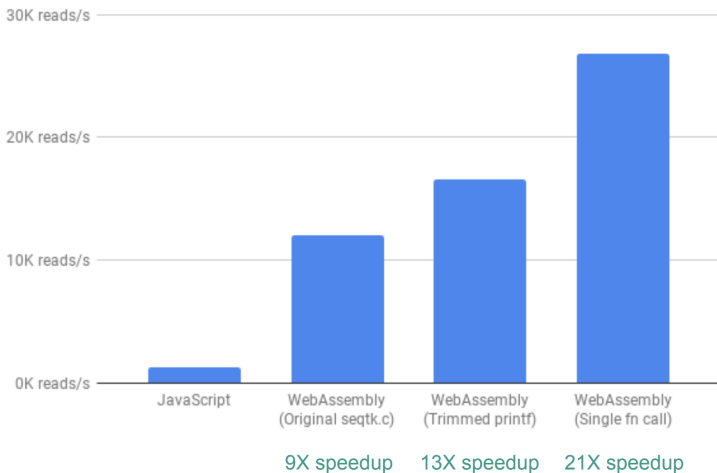
96.5 MB  $\Rightarrow$  69.2 MB

# Performance of Web Assembly — Figma [3]



3× faster load times

# Performance of Web Assembly — Fastq.bio [4]





## Performance of Web Assembly — Compared to native [5]

Benchmark	Field	Native time(s)	Google Chrome time(s)
bzip2	Compression	370	864
mcf	Combinatorial	221	180
milc	Chromodynamics	375	369
namd	Molecular Dynamics	271	369
gobmk	Artificial Intelligence	352	537

# How does it compare?

- ▶ Studies haven't been done on the performance of web assembly for image processing, just some metrics found during development of one project
- ▶ Often comparisons are being made between asm.js and Web Assembly, comparing native implementations gives a more fair comparison

## My Solution

# Choosing a language for Web Assembly

- ▶ Most languages have some support for Web Assembly
- ▶ Web Assembly doesn't include a garbage collector so languages that don't need them are preferred
- ▶ Lots of development work has been put into the Rust ecosystem regarding web assembly
- ▶ Rust is a relatively new language (released in 2010) by Mozilla, now community owned. Has similarities to C++ but is designed from the ground up, so deals with some of the issues C++ has.

# Choosing problems

- ▶ Common image processing tasks
- ▶ Range of computational complexity

# Platform

- ▶ Want results to be equivalent to Google Chrome (64% market share [6])
- ▶ Node.js uses the V8 browser engine (the same as Google Chrome)

# Brightness



# Contrast

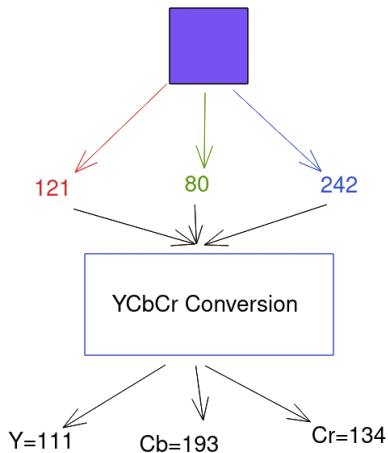




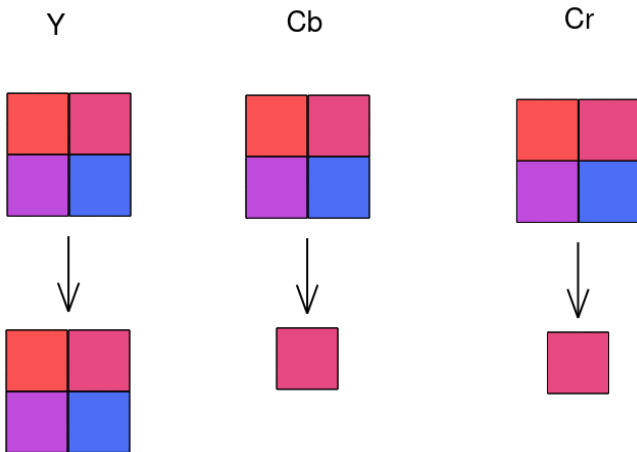
# Gaussian Blur



# JPEG — YCbCr Conversion [7]



# JPEG — Downsampling



## JPEG — DCT [8]

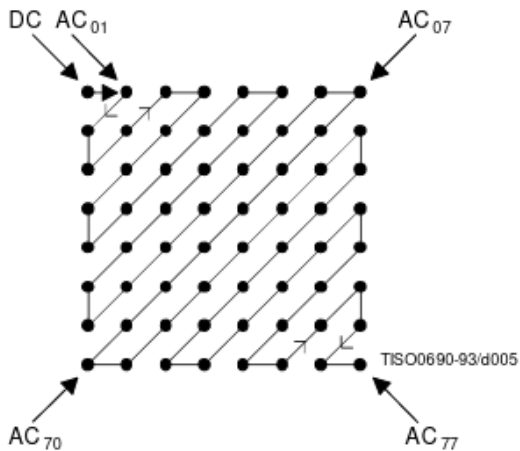
$$T_{i,j} = \begin{cases} 1/\sqrt{N} & \text{if } i = 0 \\ \sqrt{\frac{2}{N}} \cos(\frac{(2j+1)i\pi}{2N}) & \text{if } i > 0 \end{cases}$$

N is the dimension of the matrix (here 8)

## JPEG — Quantization

$$\begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix}$$

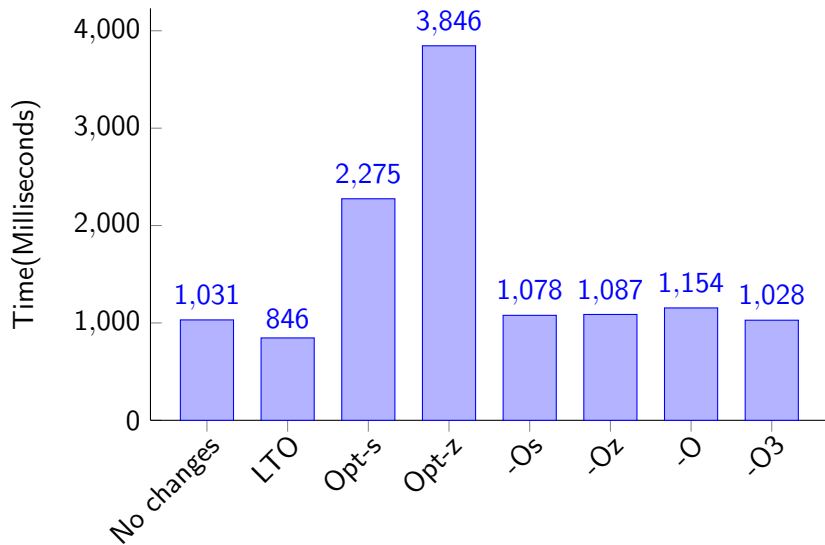
# JPEG — Zigzag



# Web Assembly Optimizations

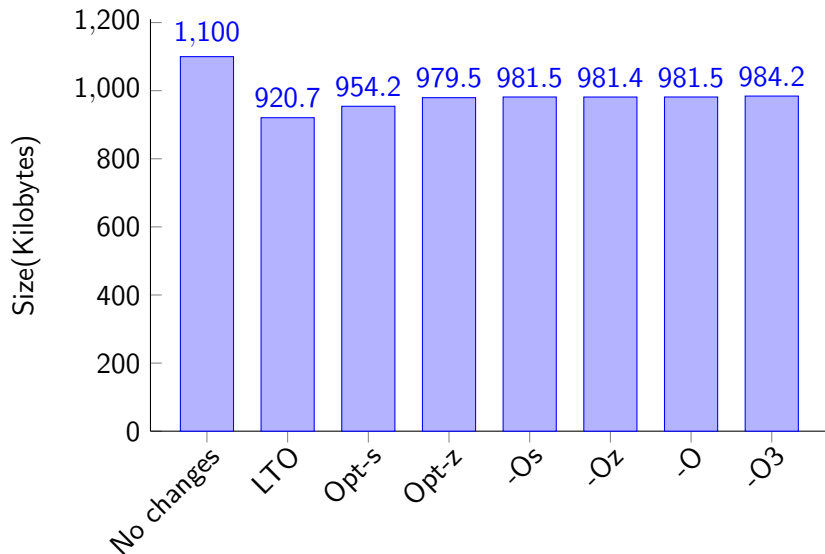
1. Link Time Optimization
2. Rust compiler opt-level
  - ▶ "s" : optimize for size
  - ▶ "z" : optimize aggressively for size
3. `wasm-opt`
  - ▶ `-Os` : optimize for size
  - ▶ `-Oz` : optimize aggressively for size
  - ▶ `-O` : optimize for speed
  - ▶ `-O3` : optimize aggressively for speed

## Web Assembly Optimizations - Time

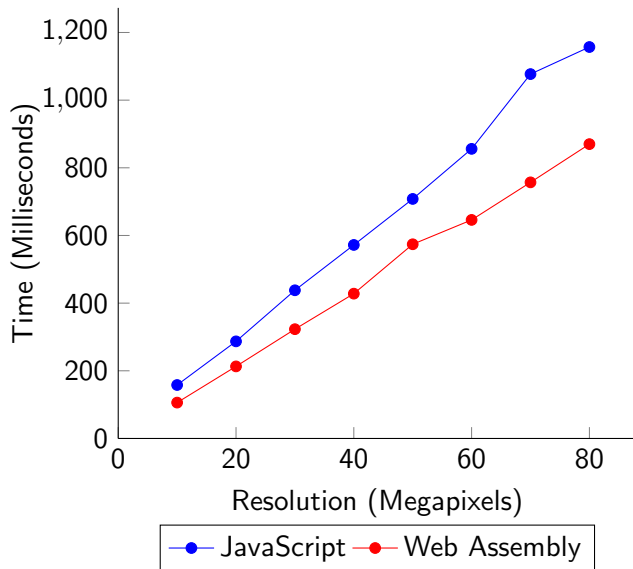




## Web Assembly Optimizations - Size



## Verifying the results

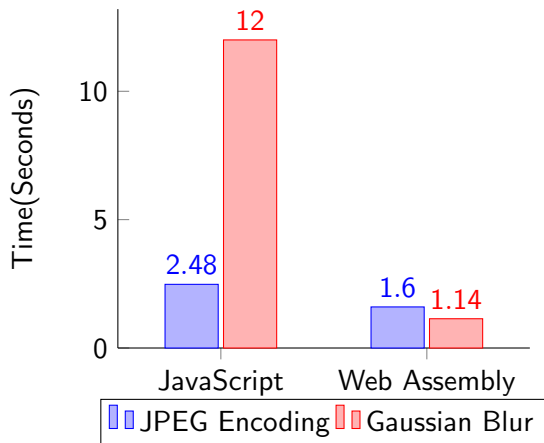


# Results

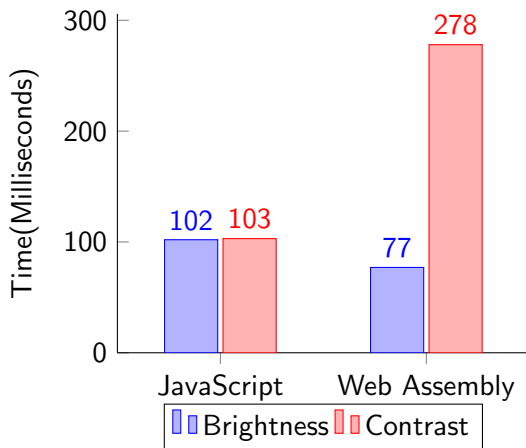
# Interesting metrics

- ▶ Run time
- ▶ Memory consumption

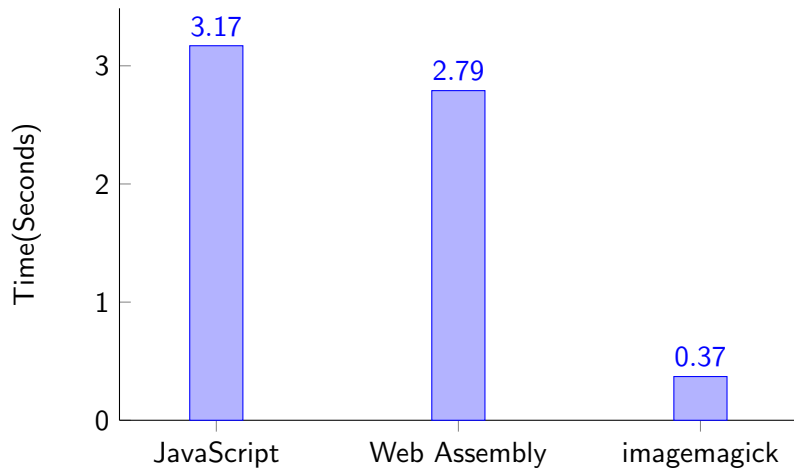
# Complex Algorithms



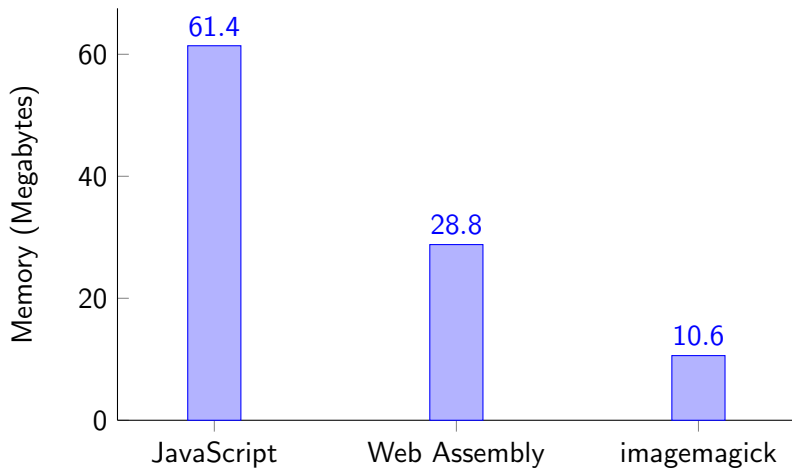
# Simple Algorithms



## Time for the full process



# Memory consumption of the full process





# Evaluation

- ✓ Faster
- ✓ Lower memory consumption
- ✓ Static types
- ✗ Static types
- ✗ Slower than native
- ✗ Need to use multiple languages
- ✗ Libraries have had less development

# Further Work

- ▶ WebGPU [9]
- ▶ Different methods
- ▶ Different languages

# References I

- [1] Andreas Haas et al. “Bringing the web up to speed with WebAssembly”. In: *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation*. 2017, pp. 185–200.
- [2] Joe Haddad. *Remove sharp for wasm variant*. Feb. 2021. URL: <https://github.com/vercel/next.js/pull/22253>.
- [3] Evan Wallace. *WebAssembly cut Figma’s load time by 3x*. en. URL: <https://www.figma.com/blog/webassembly-cut-figmas-load-time-by-3x/> (visited on 10/28/2020).
- [4] *How We Used WebAssembly To Speed Up Our Web App By 20X (Case Study)*. en. Apr. 2019. URL: <https://www.smashingmagazine.com/2019/04/webassembly-speed-web-app/> (visited on 11/27/2020).

# References II

- [5] Abhinav Jangda et al. “Not so fast: analyzing the performance of webassembly vs. native code”. In: *2019 {USENIX} Annual Technical Conference ({USENIX}{ATC} 19)*. 2019, pp. 107–120.
- [6] Statcounter. *Browser Market Share Worldwide*. 2021. URL: <https://gs.statcounter.com/browser-market-share> (visited on 04/09/2021).
- [7] *INFORMATION TECHNOLOGY – DIGITAL COMPRESSION AND CODING OF CONTINUOUS-TONE STILL IMAGES – REQUIREMENTS AND GUIDELINES*. The International Telegraph and Telephone Consultative Committee. 1993.
- [8] Cabeen Ken and Gent Peter. “Image compression and the discrete cosine transform”. In: *Math45 College of the Redwoods* (1998).

## References III

- [9] GPU for the Web Community Group. *WebGPU*. Standard. W3C, 2021. URL: <https://gpuweb.github.io/gpuweb/>.