Image Processing using JavaScript and Web Assembly

Student Name: Sam Robbins

Supervisor Name: Tom Friedetzky

Submitted as part of the degree of BSc Computer Science to the

Board of Examiners in the Department of Computer Sciences, Durham University

Abstract —

A Context/Background

Web Assembly is a relatively new method for computation on the web, allowing for the use of a much wider range of languages. This is being used in high intensity contexts to improve computation time.

B Aims

To determine in which cases implementing algorithms in web assembly is the right choice to make

C Method

Implement a range of image processing algorithms in both JavaScript and Web Assembly and measure using a variety of metrics.

D Results

E Conclusions

Keywords — Put a few keywords here.

I INTRODUCTION

Image processing is a widely used technique for a range of problems. In most modern cameras there will be some aspect of image processing at the time of capture, whether it is enhancements to the image or just the formatting to a file type, such as JPEG. After capture there is often additional image processing, be it applying filters or compression for upload online. Sophisticated image processing techniques don't just have benefit for the sharing of photos, but are also used medically, such as for processing CT images (Zhang et al. 2017).

The performance of image processing was greatly improved with the introduction of Digital Signal Processors (DSP), these are specialised chips for performing signal processing tasks, such as the discrete cosine transform. In mobile devices these are now often integrated into the System on a Chip (SOC) (Angoletta 2008), however with the increase in computational performance since DSPs were introduced, these algorithms can also be ran on the main processor.

Traditionally image processing has been done in applications like Adobe Photoshop, however as browser market share is increasing web based photo editors are becoming more common. In a study by Forrester Consulting, workers are spending 1/3 of the work day on average in a web browser, so it is an area lots of companies are targeting to launch new products (Forrester 2018).

In 1995 JavaScript was introduced as a method for introducing interactivity to web applications, since then there were some improvements made for programmers who wanted to perform computationally intensive tasks with their website, such as Web Workers, introduced in 2009 (Hickson 2009). However one of the biggest steps forward is the introduction of Web Assembly in 2017, allowing for assembly code to be executed by web browsers (Haas et al. 2017).

Using Web Assembly over JavaScript proposes advantages such as being able to use your existing codebase, rather than having to translate it into JavaScript along with benefits such as type safety using static types, where JavaScript uses dynamic types. TypeScript works to try and solve this problem, but is just a transpiler to JavaScript, where Web Assembly allows for type checking at runtime.

A Objectives

For this project I want to implement a range of image processing algorithms in both Web Assembly and JavaScript and compare their performance. For Web Assembly I will be using the Rust programming language as it is one of the most popular languages for web assembly and contains a range of features to make creating web assembly easier.

B Research Question

My Research question is to find in which cases Web Assembly offers a benefit over JavaScript, finding this out by implementing image processing algorithms using both mechanisms.

II RELATED WORK

A Language options for web assembly

B Web Assembly Image Processing in use

Squoosh by Google Chrome is a tool to compress images and implements many of its codecs using Web Assembly, this approach was also adopted by Next.js for their image component to improve performance, reducing the installed size by 27.3 MB (Haddad 2021). As this was replacing a package with code in the project, it led to a large increase in the amount of code to maintain, but this could be abstracted to a package. One example of such a package is `photon` which provides abstractions on top of the `image` Rust library (Odwyer 2019).

C Performance of Web Assembly

Alongside image processing, Web Assembly is also used for a range of other purposes. Figma is a tool for designers, and implemented Web Assembly to improve their load times $3\times$ compared to their previous solution which translated assembly to JavaScript (Wallace 2017).

However, web assembly is not always the best solution, as when the same author was creating a JavaScript bundler they recommend against using Web Assembly (Wallace 2020). This is because JavaScript bundlers are ran natively on the computer, rather than in a browser, a method enabled by solutions such as Node.js. Node.js has very similar performance to browsers as it uses the same v8 browser engine as Google Chrome does, however Node.js also means that native code can be ran, which the author observed to provide a $10\times$ speed up.

This behaviour was also observed in (Jangda et al. 2019), where peak slowdowns were observed of $2.5\times$ in Google Chrome compared to the native code. These benchmarks are taken from SPEC CPU2006 and SPEC CPU2017. A sample of these results are shown in Table 1

Table 1: Comparison of algorithms between native and wasm

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

As one can see from these results, Native isn't guaranteed to be faster than Web Assembly, however this is often the case.

D Current Image Processing Techniques

One of the most popular image processing libraries for JavaScript is Sharp with over 1,700,000 weekly downloads, this uses native code in the form of libvips and claims to be the fastest module for resizing. A slightly less popular library is jimp with over 1,400,000 weekly downloads, the difference with this library is that it is written entirely in JavaScript and so can be ran in the browser. The difference between these and other libraries was measured on a task of resizing an image (Fuller 2020). These results are shown in Table 2

Table 2: Performance of Image Processing Libraries

Library	Best ops/sec	Best speedup
jimp	0.77	1.0
mapnik	3.39	4.4
gm	4.33	5.6
imagemagick	4.39	5.7
sharp	25.60	33.2

This shows native libraries being significantly faster than the solution in pure JavaScript, which means that even with the slowdown of web assembly compared to native code as discussed before, web assembly should still outperform JavaScript in this regard.

E Image formats

There are a range of image formats in use, and the encoding and decoding of images is a computationally intensive task. The Web Almanac studies over 7 million websites and found that 40.26% of images are of the jpg image format and 26.90% in png (Levien et al. 2020). JPEG is a high performance image codec with a study by Cloudinary showing to to have an encoding speed of 49 MP/s and a decoding speed of 108 MP/s (Sneyers 2020). JPEG has both lossless and lossy versions, and in a study of lossless compression of 382 images, these results are shown in Table 3 (Ukrit et al. 2011)

Table 3: Compression speed and ratio for various algorithms

Algorithm	Compression Speed	Compression Ratio
Lossless JPEG	11.9	3.04
JPEG-LS	19.6	4.21
JPEG 2000	4.0	3.79
PNG	3.6	3.35

As one can see from these results, the compression ratio is very comparable between the various implementations of JPEG and PNG, however the compression speed greatly differs, with some JPEG algorithms significantly outperforming PNG.

III SOLUTION

This section presents the solutions to the problems in detail. The design and implementation details should all be placed in this section. You may create a number of subsections, each focussing on one issue.

This section should be between 4 to 7 pages in length.

IV RESULTS

this section presents the results of the solutions. It should include information on experimental settings. The results should demonstrate the claimed benefits/disadvantages of the proposed solutions.

This section should be between 2 to 3 pages in length.

V EVALUATION

This section should between 1 to 2 pages in length.

VI CONCLUSIONS

This section summarises the main points of this paper. Do not replicate the abstract as the conclusion. A conclusion might elaborate on the importance of the work or suggest applications and extensions. This section should be no more than 1 page in length.

The page lengths given for each section are indicative and will vary from project to project but should not exceed the upper limit. A summary is shown in Table 4.

Table 4: SUMMARY OF PAGE LENGTHS FOR SECTIONS

	Section	**Number of Pages**
I.	Introduction	2–3
II.	Related Work	2–3
III.	Solution	4–7
IV.	Results	2–3
V.	Evaluation	1-2
VI.	Conclusions	1

References

Angoletta, M. E. (2008), 'Digital signal processor fundamentals and system design'.

Forrester (2018), 'Rethink technology in the age of the cloud worker'.

Fuller, L. (2020), 'Sharp performance'.
URL: *https://sharp.pixelplumbing.com/performance*

Haas, A., Rossberg, A., Schuff, D. L., Titzer, B. L., Holman, M., Gohman, D., Wagner, L., Zakai, A. & Bastien, J. (2017), Bringing the web up to speed with webassembly, *in* 'Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation', pp. 185–200.

Haddad, J. (2021), 'Remove sharp for wasm variant'.
URL: *https://github.com/vercel/next.js/pull/22253*

Hickson, I. (2009), Web workers, W3C note, W3C. https://www.w3.org/TR/2009/WD-workers-20090423/.

Jangda, A., Powers, B., Berger, E. D. & Guha, A. (2019), Not so fast: analyzing the performance of webassembly vs. native code, *in* '2019 {USENIX} Annual Technical Conference ({USENIX}{ATC} 19)', pp. 107–120.

Levien, R. et al. (2020), *Web Almanac*, 2020 edn, HTTP Archive.

Odwyer, S. (2019), 'Photon'.
URL: *https://github.com/silvia-odwyer/photon*

Sneyers, J. (2020), 'How jpeg xl compares to other image codecs'.
URL: *https://cloudinary.com/blog/how_jpeg_xl_compares_to_other_image_codecs*

Ukrit, M. F., Umamageswari, A. & Suresh, G. (2011), 'A survey on lossless compression for medical images', *International Journal of Computer Applications* **31**(8), 47–50.

Wallace, E. (2017), 'Webassembly cut figma's load time by 3x'.
URL: *https://www.figma.com/blog/webassembly-cut-figmas-load-time-by-3x/*

Wallace, E. (2020), 'esbuild'.
URL: *https://esbuild.github.io/getting-started*

Zhang, H., Zeng, D., Zhang, H., Wang, J., Liang, Z. & Ma, J. (2017), 'Applications of nonlocal means algorithm in low-dose x-ray ct image processing and reconstruction: A review', *Medical physics* **44**(3), 1168–1185.