# Benchmarking on WebAssembly

**20th September 2019**

## PROJECT BACKGROUND

### Historical Background

In the past, JavaScript, HTML and CSS were mandatory and de-facto standards to developing a website. At first, JavaScript was designed for simple interactions and decorations. It was treated as a simple language for simple usages. These days, our demands have changed significantly, even server side frameworks like Node.js uses JavaScript. However, it is still far away from satisfying our needs. For this reason, people were trying to create new ways to replace or complement the traditional web technologies.

In 2011, Google developed Google Native Client(NaCl) which allows C and C++ executables to run in a sandbox environment on Chrome. Unfortunately, it failed to gain attention due to limited support in other browsers[1].

In 2013, Mozilla released asm.js for translating C or C++ code into JavaScript code. This allows further development of running C and C++ codes on the web browsers. This also provided solid foundation for WebAssembly[1].

In 2015, WebAssembly was first announced and demonstrated. Two years later, the minimum viable product of it was officially released.

### WebAssembly and JavaScript

WebAssembly uses assembly-like language dedicated to take the native performance of local computers for processing a web application in a secured environment[3]. Unlike JavaScript and web workers, it supports multithreading in a flexible way.

WebAssembly does not serve as a replacement of JavaScript. Instead, it complements JavaScript by combining their high flexibility and performance advantage.

### WebAssembly and Emscripten

Currently, there is no Garbage Collection support from WebAssembly[1]. Using languages like Java or Python will have difficulties in implementation. By comparison, languages like

C and C++ which does not need Garbage Collection are more suitable to be used at this stage.

Therefore, this project takes C++ and JavaScript as the main programming languages. A toolchain called Emscripten will be used to compile the C++ code into WebAssembly code[4].

**Use Cases**

According to different articles, use cases of WebAssembly include image/video editor, games, peer-to-peer application, local web server and so on[1][5].

In June 2019, Autodesk migrated one of their famous products, AutoCAD, to the web using WebAssembly[2]. It is foreseeable that the use cases of WebAssembly will continue to expand.

**Motivation to Benchmarking**

WebAssembly favours migrating existing software to the web. However, there may be some performance loss as sacrifice. This project aims to evaluate the performance of a standalone image editor, the WebAssembly and JavaScript versions of it. Details are discussed in Project Objectives section.

## PROJECT OBJECTIVES

1. Many image processing routines are computationally intensive tasks. This characteristic is beneficial towards the benchmarking and evaluation on WebAssembly. An image processor with various image processing routines will be created for these purposes.
2. The performance of WebAssembly varies on different platforms and settings. The benchmarking and evaluation should be done on different environments, including the type of the web browsers and those three versions of image processors.
3. Give suggestions on improving the performance on working with WebAssembly and JavaScript. The benchmarking results may lead to some hints on this.

## METHODOLOGY

### Image Processing

To implement the core image processor, the Image Filtering module provided by OpenCV will be used. More modules will be added for evaluation if they are feasible. As OpenCV supports C++, both the standalone and WebAssembly versions of the processor can be created with it.

For the JavaScript version, CamanJS and CamanJS Plugins will be used for the comparison. These two packages use JavaScript and CoffeeScript in its implementation. Other image processing libraries will be added if they are feasible or needed.

### Benchmarking

At first, a microbenchmark support library provided by Google will be used inside the C++ and WebAssembly versions of the image processor. At the same time, JetStream2 will be used for the JavaScript version. Other benchmarking tools may be applied in the future for further evaluation.

As different web browsers have different support towards WebAssembly, the evaluation will be carried on different web browsers.

### Result Visualization

During benchmarking process, other processes and threads running in the background can affect its accuracy. It is suggested to allow only one benchmarking process to run at a time for more accurate results.

The results will later be gathered and represented in different forms. Python and related libraries are suitable for this purpose.

## PROJECT SCHEDULE & MILESTONE

- September
    - Project Plan
- October
    - Adapt to OpenCV in C++ and WebAssembly
    - Adapt to CamanJS and CamanJS Plugins in JavaScript
    - Implements an image processing function from OpenCV, CamanJS and CamanJS
- November
    - Adapt to benchmarking libraries in C++, WebAssembly and JavaScript environments
    - Implements a few benchmarking functions
- December
    - First evaluation to the performance of the image processors
- January
    - Add more image processing and benchmarking functions
- February
    - Add more image processing and benchmarking functions
- March
    - Second evaluation to the performance of the image processors
- April
    - Wrap up and recheck all works

## REFERENCES


[1]M. Rourke, *Learn WebAssembly*, 1st ed. Birmingham - Mumbai: Packt, 2018.

[2]"WebAssembly", *Webassembly.eu*, 2019. [Online]. Available: https://webassembly.eu/. [Accessed: 20- Sep- 2019]

[3]"WebAssembly", *Webassembly.org*, 2019. [Online]. Available: https://webassembly.org/. [Accessed: 20- Sep- 2019]

[4]"Main — Emscripten 1.38.45 documentation", *Emscripten.org*, 2019. [Online]. Available: https://emscripten.org/. [Accessed: 20- Sep- 2019]

[5]"Use Cases - WebAssembly", *Webassembly.org*, 2019. [Online]. Available: https://webassembly.org/docs/use-cases/. [Accessed: 20- Sep- 2019]