Project Synopsis

Team B

05 October 2021

1 Title

Image processing web app

2 Statement of the Problem

Most image manipulation web apps require you to upload images to the cloud before performing the transformations. But for people with lower bandwidth limits, this approach may not be practical for editing high quality images, since for each image an upload and a download would have to occur.

Alternatively the images could be manipulated in the browser using Javascript, but since Javascript is an interpreted language, for high quality images, operations could be slow.

3 Justification

Our solution solves both the problem of high bandwidth usage and would use, WebAssembly running in the users browser so that editing can take on the user's side, without need for uploading. Additionally, since WebAssembly is compiled, this approach would be more performant than a pure js approach.

4 Scope and limitations of the project

4.1 Scope:

Initially we plan to implement the simpler image transformations from scratch in WebAssembly. These transformations include but are not limited to, flipping, rotating, color channel extraction, conversion to black and white and conversion to negative.

4.2 Limitations

1. Implementing interactive transformations like cropping in WebAssembly would be highly inefficient due to increase in need of message passing between Javascript and WebAssembly.

2. Due to the low level nature of WebAssembly, implementing more complex transformations like filters would be quite difficult without the use of external libraries

5 Review of literature

A paper by Herrera, Chen et. al. [1] found that the performance of Wasm on the ostritch numerical benchmark is nearly 2x faster than the speed of Javascript, and on Microsoft edge it is at least 2.5x faster than Javascript(due to poor js optimisation on edge).

As found by Alevärn [2], using WebAssembly for in browser image processing is faster than server side image processing. For large images, they found that even for a very low Round-Trip-Time(RTT) of 1 ms a rather high average throughput of 226 (Firefox) or 249 Mbps (Google Chrome) is required. In addition, in Google Chrome a RTT lower than 46 ms is required, and in Firefox a RTT lower than 50.5 ms is required to give the server-side solution in native code a chance to be as fast as the client-side solution in WebAssembly.

6 Methodology Used (tech requirement/algorithms)

6.1 Image Processing

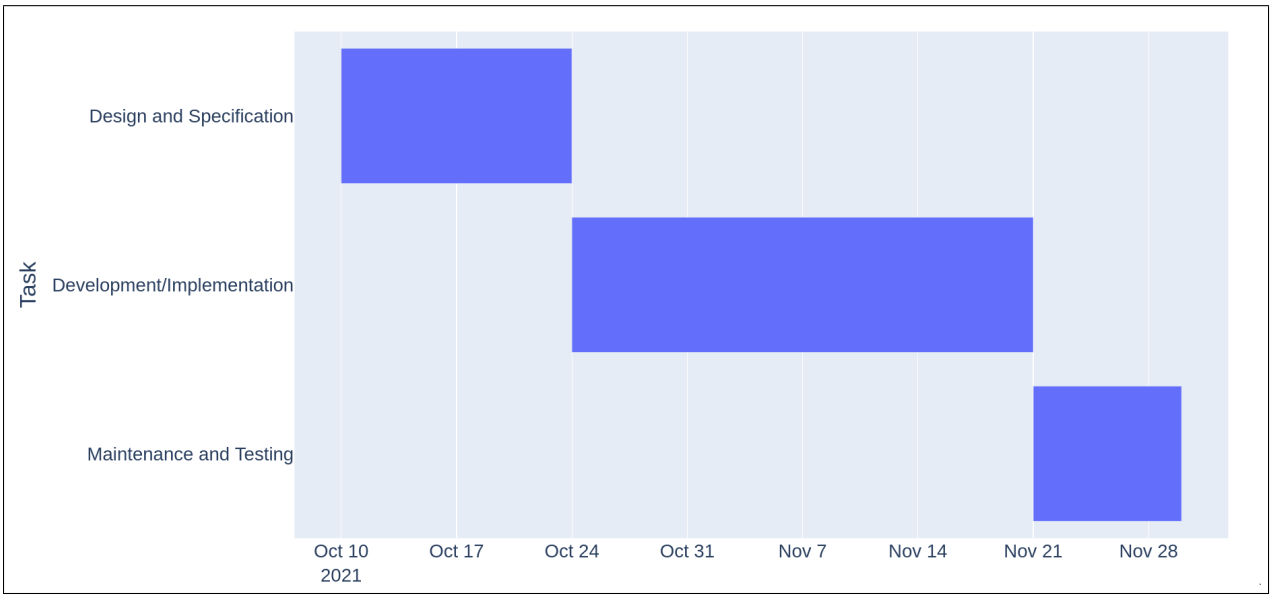
Standard image processing algorithms for flipping, rotating, color channel extraction etc will be used.

6.2 Web Assembly

Most image transformations will be done using WebAssembly, with Javascript being used for those transformations wherein implementation in WebAssembly is non-trivial (eg. cropping). WebAssembly is a binary instruction format for a stack-based virtual machine. Wasm is designed as a portable compilation target for programming languages, enabling deployment on the web for client and server applications.

7 Time schedule/work plan

- Design and Specification Phase: 2 weeks
- Implementation Phase: 4 weeks
- Maintenance and Testing Phase: 1.5 weeks



gantt chart

References

[1] D. Herrera, H. Chen, E. Lavoie, and L. Hendren, "Webassembly and javascript challenge: Numerical program performance using modern browser technologies and devices," 2018.

[2] M. Alevärn, "Server-side image processing in native code compared to client-side image processing in webassembly," 2021.

8 Bibliography

- Memory management in webassembly: https://blog.knoldus.com/memory-management-in-webassembly-with-rust/
- Image Processing in AssemblyScript: https://blog.ttulka.com/learning-webassembly-10-image-processing-in-assemblyscript