Analysis and comparison of acceleration methods in JavaScript environments based on simplified standard Hough transform algorithm

Damian Koper[1*] and Marek Woda[1]

[1*]Faculty of Information and Communication Technology, Wroclaw University of Science and Technology.

*Corresponding author(s). E-mail(s): kopernickk@gmail.com;
Contributing authors: marek.woda@pwr.edu.pl;

Abstract

JavaScript has become one of the most widely used programming language in the world. However it has not been widely adopted to perform numerical computing by a community and maintainers for a long time. In this paper, we present analysis of acceleration methods of choice in JavaScript execution environments including browser, Node and Deno. We focus evenly on adopting the same codebase to take advantage of every method, benchmarking our solutions and analysis of a toolchain building libraries as compatible as possible with multiple environments. To compare performance, we use a simplified standard Hough transform algorithm with threshold as voting phase. As a reference points of our benchmarks, we use sequential version of the algorithm written in both JavaScript and C++.

Keywords: javascript, acceleration, sht, standard hough transform, node, browser, deno, webgl, webpack, wasm, simd, workers

1 Introduction

TODO: JS in general, JS in numerical computing.
TODO: Environments briefly.

TODO: Acceleration methods briefly.

TODO: Building methods and complex architectures.

TODO: Target: * overall comparison, * to identify the most promising acceleration method and environment * to identify fields and aspects to conduct further research

2 Benchmarking

TODO: SHT in general briefly

TODO: Implemented SHT, Non-LUT and LUT variants

TODO: Env x Methods matrix

TODO: library interface and building method

TODO: Comparing benchmarks across environments and methods, js sequential and C++ as reference point

TODO: JIT and cold/warm start, real-world vs synthetic

- TODO: CoV
- TODO: possibility of function extraction

TODO: Testing environment

- TODO: Hybrid CPU architecture and bench details

3 Methods and results

3.1 Sequential

TODO: Algorithm description, variants, method description, other methods are only necessary modification of this method

TODO: Accumulator difference (one pixel differs - float64 vs float32) - single figure with 4 differences referred to later

TODO: Results, C++ as reference

TODO: Notice Firefox optimization after 5th run and chrome overall better optimization

TODO: Performance from devtools, show times in Math.sin and Math.cos

3.2 Node C++ addon

TODO: Method description, building process

TODO: Results, C++ and Sequential as reference

3.3 WebAssembly and Asm.js

TODO: Method description, building process

TODO: Results, C++ and Sequential as reference

3.4 WebAssembly SIMD

TODO: Method description, building process

TODO: refer to accumulator differences

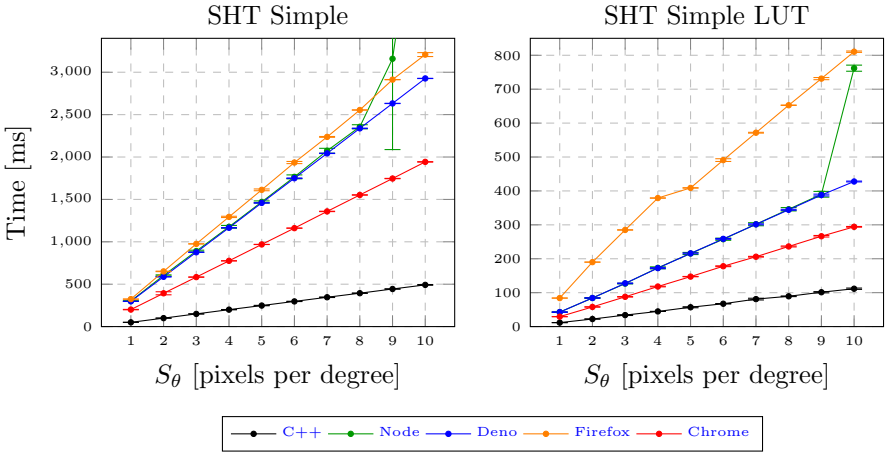
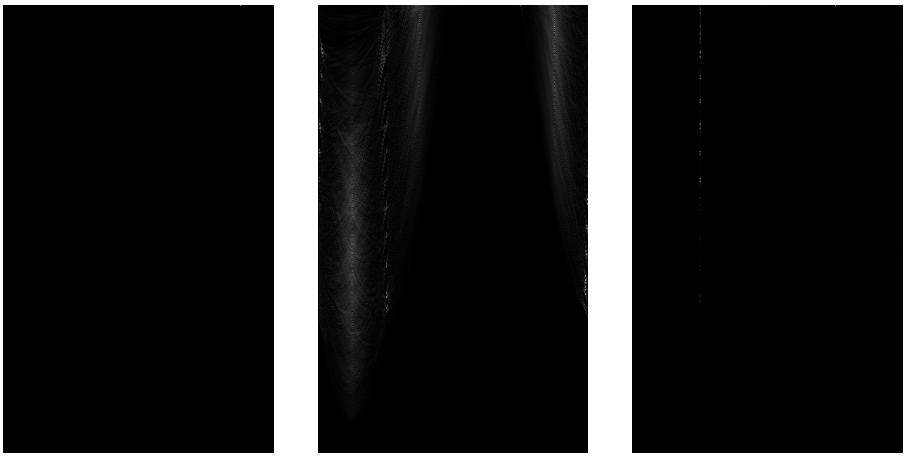


Fig. 1 Sequential SHT execution benchmark results. The difference between performance on Chrome and the rest of JavaScript environments is substantial.



(a) Sequential LUT (b) WebGL (c) WASM

Fig. 2 Normalized absolute accumulator difference from sequential non-LUT variant.

TODO: Instead of performance of implicit SIMD (llvm vectorization), mention that it is not better than standard WASM, but SIMD instructions are used to load/store data.

TODO: Performance from devtools if interesting

3.5 Workers

TODO: Method description, building process

TODO: Results, C++ and Sequential as reference

TODO: Performance from devtools

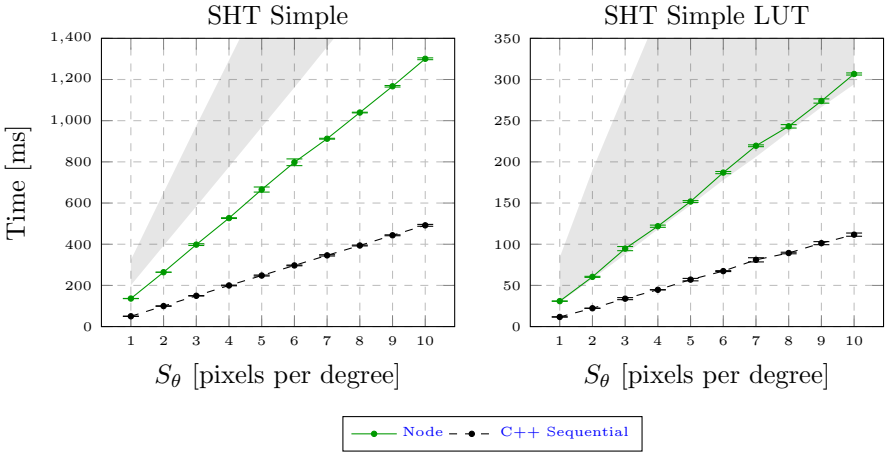


Fig. 3 Node C++ addon SHT execution benchmark results. Gray area shows sequential JavaScript execution performance range.

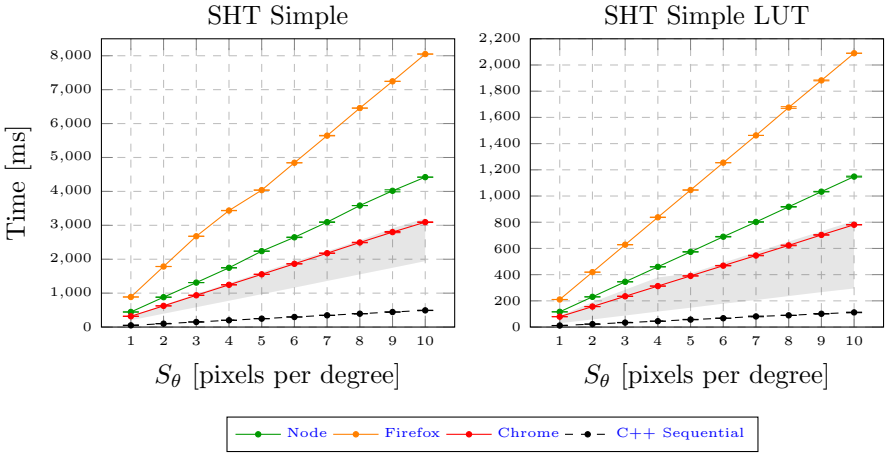


Fig. 4 Asm.js SHT execution benchmark results. Gray area shows sequential JavaScript execution performance range.

TODO: speedup math + speedup efficiency

3.6 WebGL

TODO: Method description, building process (minification caveats)
TODO: refer to accumulator differences
TODO: Results, C++ and Sequential as reference
TODO: Performance from devtools, readpixel time factor

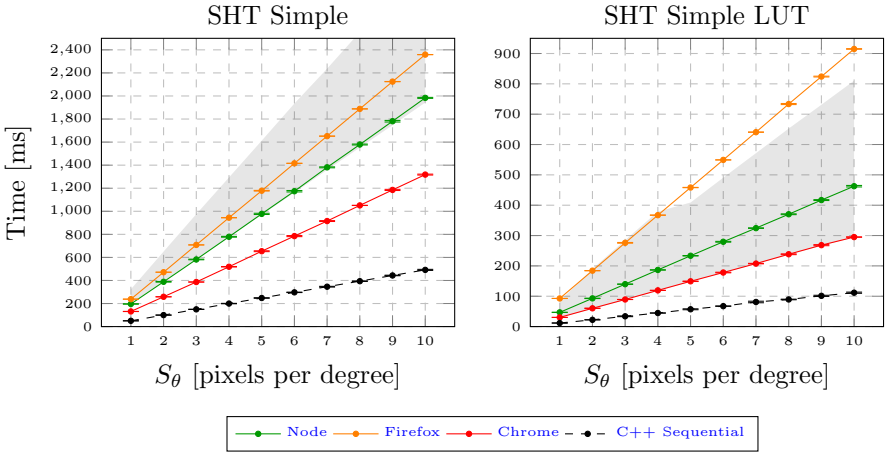


Fig. 5 WASM SHT execution benchmark results. Gray area shows sequential JavaScript execution performance range.

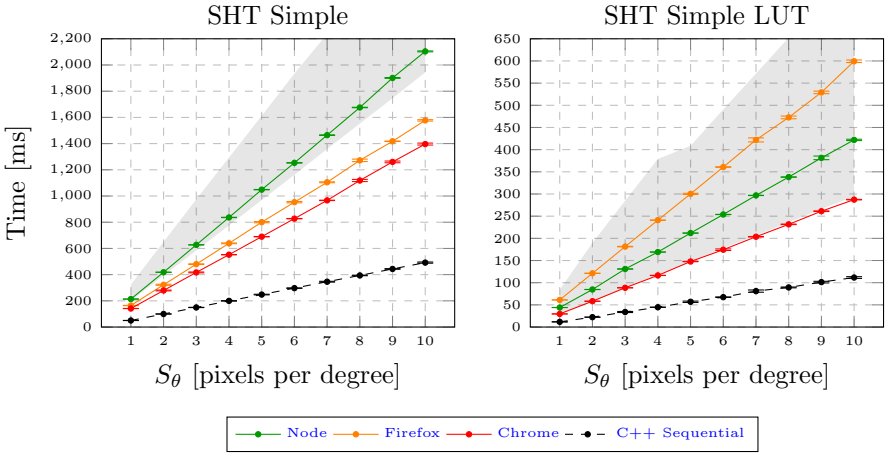


Fig. 6 WASM SIMD (explicit) SHT execution benchmark results. Gray area shows sequential JavaScript execution performance range.

4 Comparison

TODO: why

4.1 Coldstart

TODO: how and why

TODO: choose best comparison

TODO: speedup math

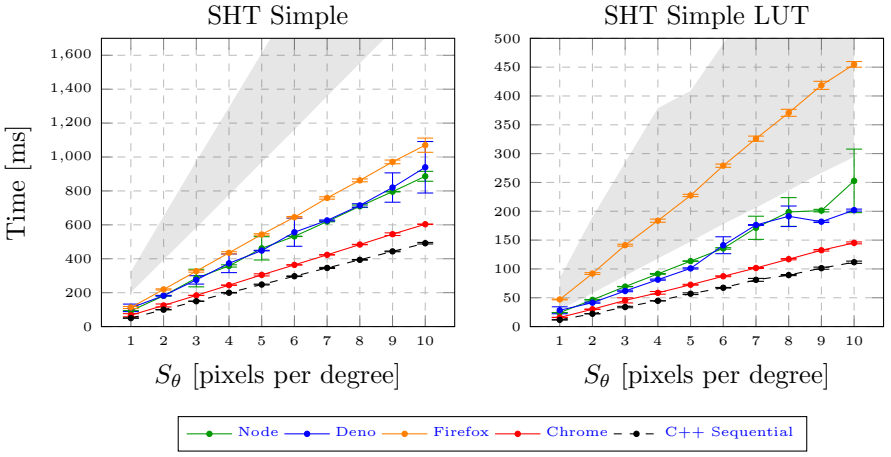


Fig. 7 Workers SHT execution benchmark results with concurrency $n = 4$. Gray area shows sequential JavaScript execution performance range. Non-LUT variant performs better than sequential execution. On the other hand the LUT one has gained only small increase in performance.

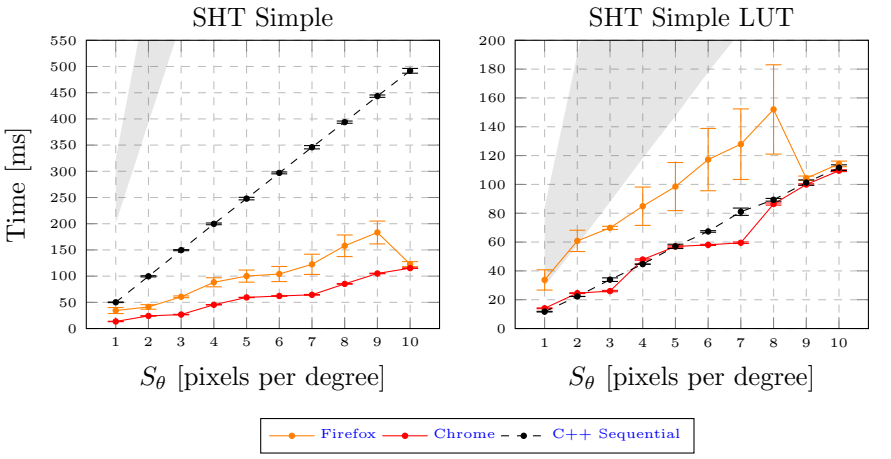


Fig. 8 WebGL SHT execution benchmark results. Gray area shows sequential JavaScript execution performance range.

4.2 Environments

TODO: how and why

5 Discussion

TODO: identified interesting aspects

TODO: further research

TODO: state of multi-environment numerical computing in JS

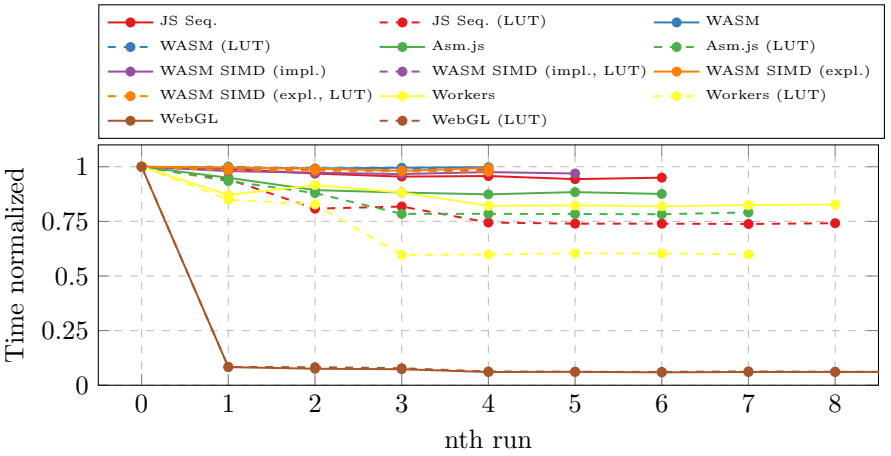


Fig. 9 Normalized coldstart execution times for methods implemented in the Chrome environment. Precompiled methods do not benefit from runtime optimization. The LUT variant is more optimizable than the standard one. The performance of WebGL's first run suffers from the need for compiling GPU programs.

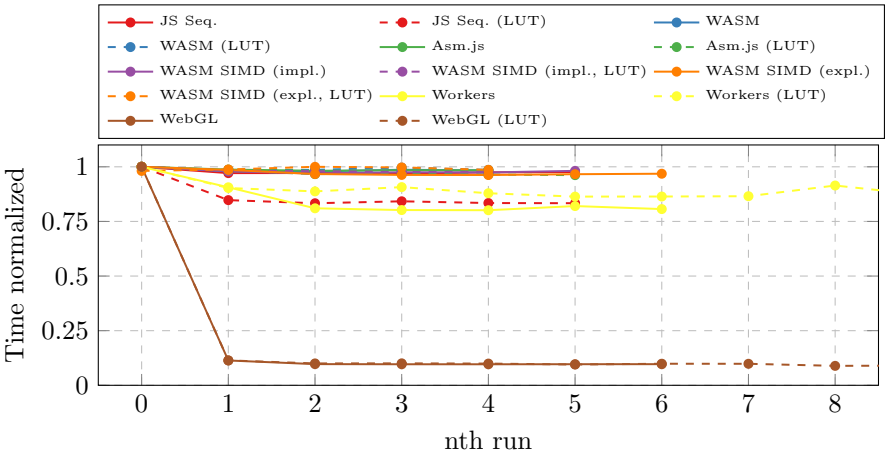


Fig. 10 Normalized coldstart execution times for methods implemented in the Firefox environment. Precompiled methods do not benefit from runtime optimization. The LUT variant is more optimizable than the standard one. The performance of WebGL's first run suffers from the need for compiling GPU programs.

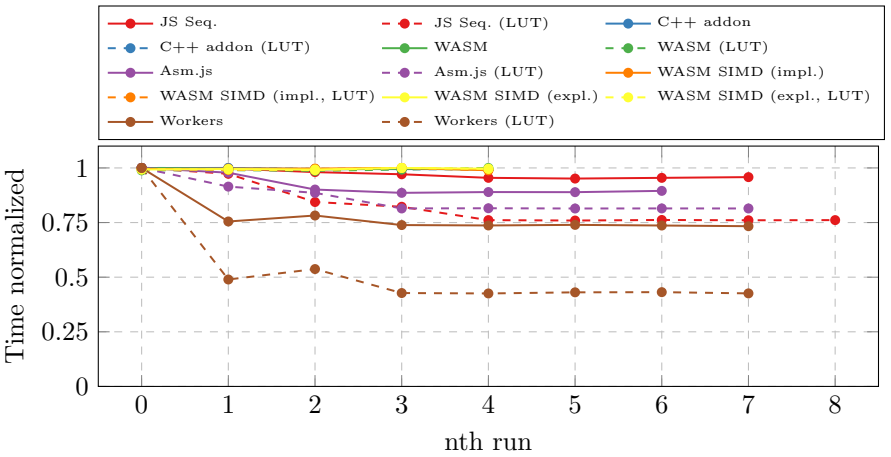


Fig. 11 Normalized coldstart execution times for methods implemented in Node environment. Precompiled methods do not benefit from runtime optimization. The LUT variant is more optimizable than the standard one.

Table 1 Comparison of implemented methods in analyzed environments. The general comparison was done using Chrome as a reference point and geometric mean for times comparable with Chrome.

	Execution time[ms]			
Method	**Chrome**	**Firefox**	**Node**	**Deno**
C++ addon (LUT)	- (-)	- (-)	31 (-)	- (-)
C++ addon	- (-)	- (-)	136 (-)	- (-)
Asm.js	317 (1.00)	887 (2.80)	444 (1.40)	- (-)
Asm.js (LUT)	79 (1.00)	210 (2.66)	116 (1.47)	- (-)
WebGL	13 (1.00)	34 (2.62)	- (-)	- (-)
WebGL (LUT)	14 (1.00)	34 (2.43)	- (-)	- (-)
JS Sequential	200 (1.00)	324 (1.62)	302 (1.51)	298 (1.49)
JS Sequential (LUT)	29 (1.00)	84 (2.90)	43 (1.48)	43 (1.48)
WASM SIMD (expl.)	141 (1.00)	164 (1.16)	214 (1.52)	- (-)
WASM SIMD (expl., LUT)	30 (1.00)	61 (2.03)	44 (1.47)	- (-)
WASM SIMD (impl.)	130 (1.00)	239 (1.84)	197 (1.52)	- (-)
WASM SIMD (impl., LUT)	30 (1.00)	93 (3.10)	47 (1.57)	- (-)
WASM	131 (1.00)	238 (1.82)	196 (1.50)	- (-)
WASM (LUT)	30 (1.00)	93 (3.10)	47 (1.57)	- (-)
Workers	67 (1.00)	113 (1.69)	93 (1.39)	110 (1.64)
Workers (LUT)	16 (1.00)	47 (2.94)	24 (1.50)	28 (1.75)
Geometric mean (non-LUT)	(1.00)	(1.86)	(1.47)	(1.56)
Geometric mean (LUT)	(1.00)	(2.71)	(1.51)	(1.61)
Geometric mean (all)	(1.00)	(2.25)	(1.49)	(1.59)

References