THE PENNSYLVANIA STATE UNIVERSITY
SCHREYER HONORS COLLEGE

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

# Performance Analysis of WebAssembly and JavaScript Engines for Common Geospatial Algorithms

MICHAEL BULLINGTON
SPRING 2021

A thesis
submitted in partial fulfillment
of the requirements
for a baccalaureate degree
in Computer Science
with honors in Computer Science

Reviewed and approved* by the following:

Dr. Danfeng Zhang
Assistant Professor, Dept. of Computer Science and Engineering
Thesis Supervisor

Dr. Jesse Barlow
Professor, Dept. of Computer Science and Engineering
Honors Adviser

* Electronic approvals are on file.

## ABSTRACT

Geospatial analysis is used across a variety of technology industries and mediums. Web applications which use location and/or mapping as a part of their product are plentiful and varied, from supply chain optimization to finding restaurants on social platforms. One of the main open-source libraries and baselines for performance is TURF.js—an "advanced geospatial analysis for browsers and Node.js." TURF.js implements algorithms such as the area of a polygon, the distance between two points, and Boolean operations (AND, OR, XOR) between two polygons.

In February 2018, the WebAssembly Working Group published the working draft for WebAssembly—a "binary instruction format for a stack-based virtual machine. WASM was created with the intent of being executable in a web browser context. This thesis will propose, test, and analyze if rewriting targeted geospatial algorithms from TURF.js in C++, compiled with the Emscripten toolchain for LLVM, will provide performance benefits to web applications.

For web applications targeting primarily Safari platforms, WASM performance of geospatial algorithms saw dramatic gains over TURF.js. For Chromium-based browsers, TURF.js performance superceded WASM in all circumstances. TURF.js also had more consistent, and often faster, performance results compared to WASM in Mozilla Firefox.

# TABLE OF CONTENTS

LIST OF FIGURES ... iii

LIST OF TABLES ... iv

ACKNOWLEDGEMENTS ... v

Chapter 1 Introduction ... 1

Chapter 2 Background ... 4

2.1 Methodology ... 11

Chapter 3 Evaluation ... 13

3.1 Implementation ... 13
3.2 Configuration 1 ... 17
3.3 Configuration 2 ... 20
3.4 Configuration 3 ... 23
3.5 Configuration 4 ... 26
3.6 Analysis ... 29

Chapter 4 Conclusion ... 32

Chapter 5 Future Work ... 34

Appendix A ... 36

Appendix B ... 37

Appendix C ... 38

# LIST OF FIGURES

Figure 1 Documentation for @turf/area .................................................................................. 6

Figure 2 Documentation for @turf/union................................................................................ 7

Figure 3 Documentation for @turf/intersect ............................................................................ 8

Figure 4 Documentation for @turf/difference.......................................................................... 8

Figure 5 Example usage of Astro.js functions.......................................................................... 11

Figure 6 Options for Astro build ............................................................................................. 14

Figure 7 dist/ folder after a successful build............................................................................ 14

Figure 8 Non-fatal error on macOS systems ........................................................................... 15

Figure 9 Configuration 1 Area Results.................................................................................... 17

Figure 10 Configuration 1 Union Results................................................................................ 17

Figure 11 Configuration 1 Difference Results ........................................................................ 18

Figure 12 Configuration 1 Intersect Results............................................................................ 18

Figure 13 Configuration 2 Area Results.................................................................................. 20

Figure 14 Configuration 2 Union Results................................................................................ 21

Figure 15 Configuration 2 Difference Results ........................................................................ 21

Figure 16 Configuration 2 Intersect Results............................................................................ 21

Figure 17 Configuration 3 Area Results.................................................................................. 23

Figure 18 Configuration 3 Union Results................................................................................ 23

Figure 19 Configuration 3 Difference Results ........................................................................ 24

Figure 20 Configuration 3 Intersect Results............................................................................ 24

Figure 21 Configuration 4 Area Results.................................................................................. 26

Figure 22 Configuration 4 Union Results................................................................................ 26

Figure 23 Configuration 4 Difference Results ........................................................................ 27

Figure 24 Configuration 4 Intersect Results............................................................................ 27

Figure 25 Screenshot of example playground ........................................................ 29

Figure 26 JavaScript Profiler Chart ........................................................................ 30

Figure 27 Profiler bottom-up tree ........................................................................... 30

# LIST OF TABLES

Table 1 Configuration 1 ops/sec averages ........ 18

Table 2 Configuration 2 ops/sec averages ........ 21

Table 3 Configuration 3 ops/sec averages ........ 24

Table 4 Configuration 4 ops/sec averages ........ 27

## ACKNOWLEDGEMENTS

Thank you to Dr. Danfeng Zhang, who aided in and was receptive to the thesis topic I was invested in exploring. Dr. Zhang was incredibly generous with his time and knowledge. Thank you to my friends and family for their continued support of my academic, professional, and personal endeavors. Thank you to my professional colleagues for their advice, knowledge, and inspiration.

# Chapter 1

# Introduction

Geospatial analysis is used across a variety of technology industries and mediums. Since Google Maps' foundational work in 2005, advances in web technology have been used to their fullest extent in geospatial applications. Web applications which use location and/or mapping as a part of their product are plentiful and varied, from supply chain optimization to finding restaurants on social platforms.

Outside of Google Maps' ecosystem, one of the main open-source libraries in the web geospatial space is TURF.js—"a modular geospatial analysis engine written in JavaScript" [12]. TURF.js implements algorithms such as the area of a polygon, the distance between two points, and Boolean operations (AND, OR, XOR) between two polygons. As of April 4, 2021, TURF.js is downloaded over 96,000 times per week on NPM, a package repository for JavaScript development [12]. TURF.js development is also sponsored by companies such as Mapbox and Triplebyte.

In February 2018, the WebAssembly Working Group published the working draft for WebAssembly—a "binary instruction format for a stack-based virtual machine."— "WASM" for short. WASM was introduced prior in "*Bringing the Web up to Speed with WebAssembly,"* a collaboration between researchers at Google, Microsoft, Mozilla, and Apple. As such, WASM was created with the intent of being executable in a web browser context. For C/C++ targets, a popular toolchain for compiling to WebAssembly is Emscripten, an open-source compiler based on Clang/LLVM infrastructure.

Since WASM's conception, its promoted performance increase has been a source of conflicting, and often domain-specific, research and evidence. Prior work such as "*Performance comparison of simplification algorithms for polygons in the context of web applications*" (2019), which analyzes a specific algorithm (polygon simplification) implemented by TURF.js, shows promising results and warrants investigation of other geospatial domains.

This thesis will propose, test, and analyze if rewriting targeted geospatial algorithms from TURF.js in C++, compiled with the Emscripten toolchain, will provide performance benefits to web applications. In the scope of this thesis, the targeted algorithms will be as follows:

1. "@turf/area," derived from "*Some Algorithms for Polygons on a Sphere*" (2007) by Robert G. Chamberlain, William H. Duquette. [8]
2. "@turf/union," "@turf/intersect," "@turf/difference." Collectively, these algorithms are derived from "*A new algorithm for computing Boolean operations on polygons*." (2013) by Francisco Martínez, Carlos Ogayar, Juan R. Jiménez, Antonio J. Rueda. [1]

Re-implementation of these algorithms, matching the feature set and test suite of TURF.js, will be accomplished through an accompanying open-source project named "Astro." The accompanying source code for this project will be made available on GitHub as well as a supplemental file to this paper.

This thesis also will analyze real-world size thresholds for each targeted algorithm and make recommendations on which algorithms may benefit most from WASM.

**Chapter 2** covers the literature review of previous research in field upon which this research is based. **Chapter 3** discusses the implementation of the experiment as well as statistical results from conducting the experiment. This will also include a CPU profile to find computational bottlenecks.

**Chapter 4** will cover conclusions that can be drawn from the research and may be summarized as follows. For web applications targeting primarily Safari platforms, the default

browser on macOS and only web rendering engine available on iOS/iPadOS, Astro/WASM performance of geospatial algorithms saw dramatic gains over TURF.js for each targeted algorithm. For Chromium-based browsers (Google Chrome, Microsoft Edge, Opera, etc.), TURF.js performance supersedes Astro/WASM in all circumstances. TURF.js is also recommended for Mozilla Firefox browsers, where TURF.js and Astro/WASM have varying performance characteristics for each targeted algorithm with JavaScript performance being more consistent. Lastly, **Chapter 5** will cover future work that may be done upon this research for further improvements.

# Chapter 2

## Background

According to the *Geographic Information Science and Technology Body of Knowledge*, GIS (geographic information systems) are collectively defined as a way of managing and analyzing information about geographic land, location, and characteristics [2]. Broadly, GIS software augments use cases for mapping with computation available since the 1960s [2]. A sub-domain of Geographic Information Science (GIScience), and the component of a larger GIS the research focuses on, is Geospatial Technology (GT). GT includes data storage, manipulation, and analysis [2].

To appropriately store and manipulate geospatial data a standard for a coordinate system and approximation of Earth's dimensions is needed. The research will exclusively use the WGS 84 datum which was developed by the *United States Department of Defense* [5]. The WGS 84 datum provides a coordinate system in terms of latitude and longitude, which can both be specified in decimal degrees [9].

Storing coordinates under WGS 84 (or any datum) requires a structured format that maps representation to higher-level constructs such as points, polygons, etc. The structured format can also be used for the transport of geospatial data, say from a TCP/IP server to a client. The *OpenGIS Simple Features Implementation Specification for SQL (SFSQL)* outlines these higher-level constructs, which are as follows [9]:

> 0-dimensional Point and MultiPoint; 1-dimensional curve LineString and MultiLineString; 2-dimensional surface Polygon and MultiPolygon; and the heterogeneous GeometryCollection.

SFSQL additionally outlines a storage/transport plain text format known as Well-Known Text (WKT) and a binary format known as Well-Known Binary (WKB).

In the space of web geospatial applications, for storage/transport the GeoJSON specification is both popular and highly used in JavaScript web-mapping libraries [9]. GeoJSON uses the WGS 84 datum and constructs defined by SFSQL [9]. An example of The GeoJSON format is shown in **Appendix C**.

GeoJSON conversely uses the JSON data-interchange format, which is both widely adopted within the JavaScript community and a subset of the *ECMA-262* Specification itself (of which JavaScript is a flavor) [20]. The JSON format supports objects and arrays, with a limited number of value types: string, number, "true," "false," and "null" [20].

As of the *ECMA-262 5.1th Edition* specification, JSON is a first-class transport in the JavaScript API with *JSON.parse* and *JSON.stringify*, functions browsers supply and heavily optimize [21]. Version 5.1 of ECMAScript was released in June 2011. *JSON.parse* and *JSON.stringify* have been implemented in Google Chrome since version 3.0 and Mobile Safari since iOS 4 [22][23].

As previously mentioned, TURF.js is a JavaScript library that provides geospatial analysis [12]. TURF.js offers a set of pure functions that perform polygon or point measurement/transformation [12], often with specific algorithms making sure the calculations are accurate on the WGS 84 datum. TURF.js does not use GeoJSON plaintext directly, however since JSON is a subset of the ECMAScript specification, the result of *JSON.parse* (a JavaScript object) can still follow both the JSON and GeoJSON standards [20]. TURF.js works with GeoJSON as its basic data type, and GeoJSON is often used for both function input and output.

The research will focus on two algorithms in the TURF.js software package: area and Boolean operations.

The first of these algorithms, area, is implemented referencing "*Some Algorithms for Polygons on a Sphere*" (2007) by Robert G. Chamberlain, William H. Duquette [8][24]. The area algorithm uses WGS 84's equatorial radius *a* as the radius of the circle [5][24]. The area algorithm is wrapped in a reduction function, *@turf/area*, that takes GeoJSON as a parameter.

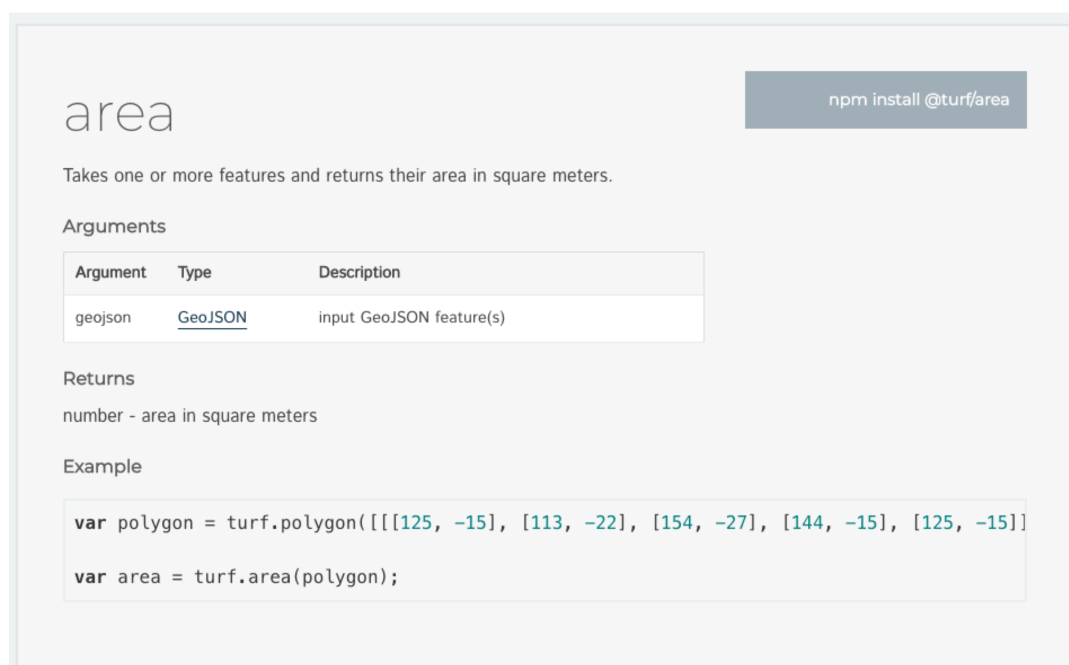



**Figure 1 Documentation for @turf/area**

The second of these algorithms, Boolean operations, is implemented in a library named "*polygon-clipping*" created by Mike Fogel and Alexander Milevski [25]. The algorithm is based on "*A new algorithm for computing Boolean operations on polygons*." (2013) by Francisco Martínez, Carlos Ogayar, Juan R. Jiménez, Antonio J. Rueda [1]. The previously mentioned paper includes sample C++ code implementing the algorithm, of which "polygon-clipping" is loosely based [25], and as discussed later is modified for this body of research.

polygon-clipping accepts GeoJSON as an argument and returns GeoJSON objects for all Boolean operations, making TURF.js wrappers small comparatively. TURF.js wraps polygon-clipping in the form of the following functions: *@turf/union*, *@turf/intersect*, and *@turf/difference.*

union

npm install @turf/union

Takes two (Multi)Polygon(s) and returns a combined polygon. If the input polygons are not contiguous, this function returns a MultiPolygon feature.

Arguments

| Argument | Type | Description |
| --- | --- | --- |
| polygon1 | Feature <(Polygon\|MultiPolygon)> | input Polygon feature |
| polygon2 | Feature <(Polygon\|MultiPolygon)> | Polygon feature to difference from polygon1 |
| options | Object | Optional Parameters |

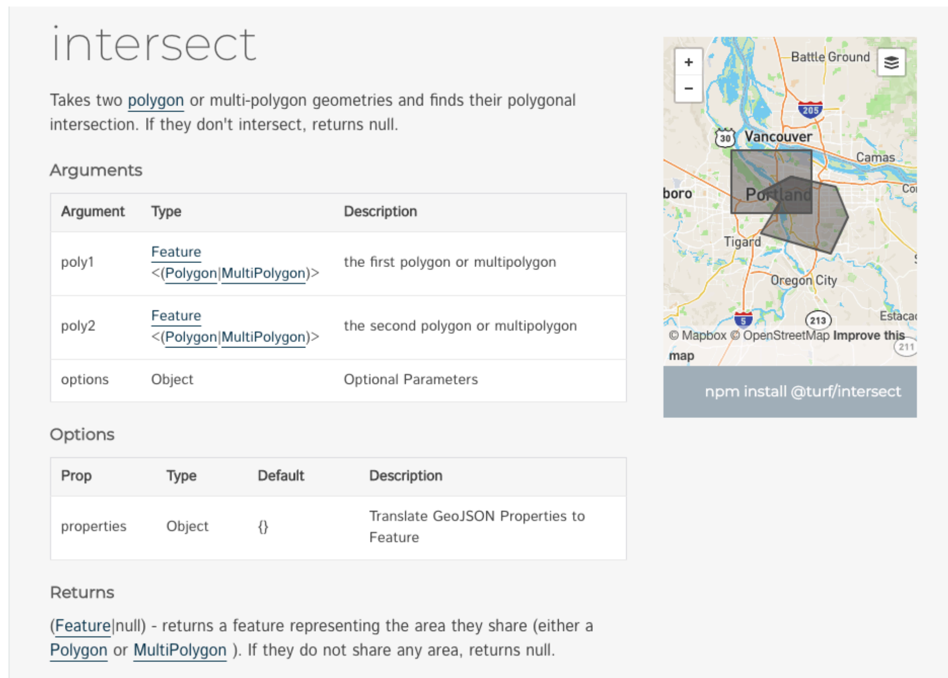
Options

| Prop | Type | Default | Description |
| --- | --- | --- | --- |
| properties | Object | {} | Translate Properties to output Feature |

Returns

Feature <(Polygon|MultiPolygon)> - a combined Polygon or MultiPolygon feature

**Figure 2 Documentation for @turf/union**



# intersect

Takes two polygon or multi-polygon geometries and finds their polygonal intersection. If they don't intersect, returns null.

Arguments

| Argument | Type | Description |
|---|---|---|
| poly1 | Feature <(Polygon\|MultiPolygon)> | the first polygon or multipolygon |
| poly2 | Feature <(Polygon\|MultiPolygon)> | the second polygon or multipolygon |
| options | Object | Optional Parameters |

Options

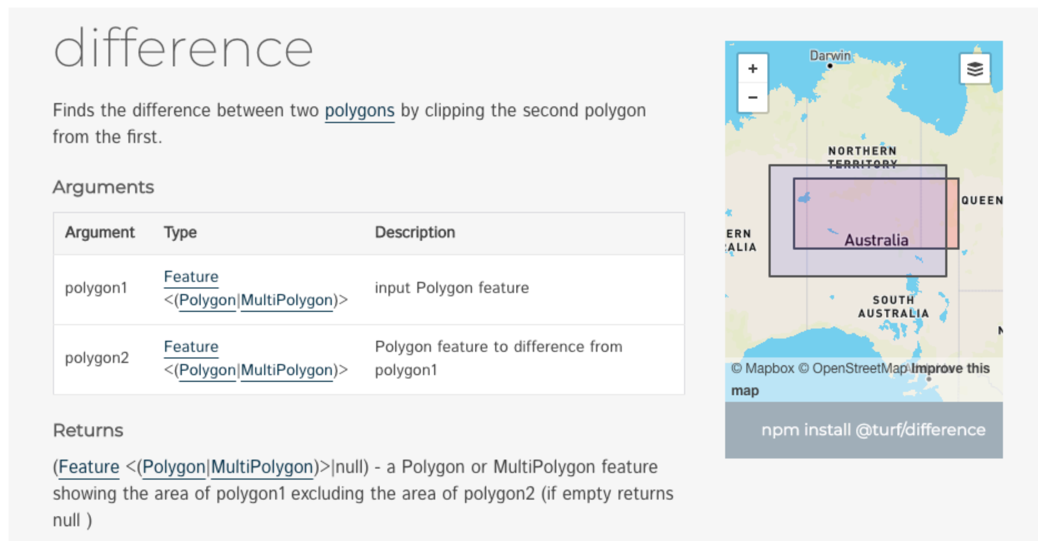
| Prop | Type | Default | Description |
|---|---|---|---|
| properties | Object | {} | Translate GeoJSON Properties to Feature |

Returns

(Feature|null) - returns a feature representing the area they share (either a Polygon or MultiPolygon ). If they do not share any area, returns null.



**Figure 3 Documentation for @turf/intersect**



# difference

Finds the difference between two polygons by clipping the second polygon from the first.

Arguments

| Argument | Type | Description |
|---|---|---|
| polygon1 | Feature <(Polygon\|MultiPolygon)> | input Polygon feature |
| polygon2 | Feature <(Polygon\|MultiPolygon)> | Polygon feature to difference from polygon1 |

Returns

(Feature <(Polygon|MultiPolygon)>|null) - a Polygon or MultiPolygon feature showing the area of polygon1 excluding the area of polygon2 (if empty returns null )



**Figure 4 Documentation for @turf/difference**

Historically JavaScript has been the main target for web developers due to its wide availability in browsers [6]. Compiling to JavaScript from other languages, such as Java or Python, have long been an area of interest [6]. For C/C++ specifically, a compiler from C/C++ to

JavaScript was created named Emscripten [4]. A core technology enabling Emscripten is LLVM, an umbrella project for various compiler and toolchain technologies including an intermediary IR format and various frontends [4]. Specifically, Emscripten uses LLVM at various layers, including compiling LLVM IR to JavaScript [4]. Emscripten originally compiled to a subset of JavaScript that was later formalized as *asm.js*, which enabled browsers such as Firefox to make asm.js specific optimizations not available with traditional JavaScript code [6].

With the working draft of WebAssembly, employees from all major browsers worked on a proposal to expand asm.js's use case into a separate stack-based virtual machine running inside the browser (or JS engine such as V8) [3]. The binary format used as bytecode for this virtual machine is known as WebAssembly, or WASM for short. Over asm.js, WASM has the potential for smaller file sizes with its binary format and further performance benefits [3].

WASM is loaded asynchronously in a browser, typically from a .wasm file extension, then initialized and ran from a JavaScript context. Interacting with WASM is very low-level and requires calling exported symbols from WASM code, with the only value types being i32, i64, f32, and f64 [3]. Because of this, WASM applications often ship with JavaScript bindings, which wrap friendly APIs around these constructs and handle things like converting value types, manual memory management, etc.

WebAssembly 1.0 has currently shipped in all four major browsers [30]. As of Emscripten v2.0.0, released in October 2020, the only backend supported is a new LLVM backend that creates WASM bytecode directly. As such, Emscripten no longer directly supports asm.js [28].

Comparing performance characteristics of JavaScript geospatial equations with equivalents ran in WASM have already been attempted, namely with "*Performance comparison*

*of simplification algorithms for polygons in the context of web applications*" by Alfred Melch [7]. This prior art focuses on the simplification algorithm for polygons, which TURF.js wraps similarly to a library named Simplify.js [7]. The goal of researching other algorithms in the TURF.js software package is to expand on Melch's work in this area and assess performance benefits since 2019 in both JavaScript engine performance and WebAssembly virtual machine performance.

Additionally, unexplored in Melch's paper, WebAssembly algorithms may benefit from the WebAssembly SIMD proposal, which was based on the SIMD.js proposal and originally from Dart SIMD. SIMD provides instructions for the WebAssembly VM that can parallelize math operations on vectors. SIMD instructions are supported by most modern processor architectures which enable this feature. SIMD operations can either be written directly to take advantage of the instructions, or the LLVM compiler can make a best-effort optimization using "autovectorization." Autovectorization can be enabled in Emscripten with the "*-msimd128*" flag [10]. As of the date of publication, Google Chrome is the only browser to have shipped SIMD support in the form of a feature flag *chrome://flags/#enable-webassembly-simd* [10], with Mozilla Firefox signaling development [11].

## 2.1 Methodology

To assess the performance characteristics between TURF.js and WASM, equivalent algorithms for area, convex hull, and Boolean operations will be implemented in C++. A best-effort attempt will be used to make correctness and computational similarities with the TURF.js implementation, including any deviations TURF.js has from the source algorithms.

Serializing GeoJSON objects to WASM-compatible value types and back will be handled in the setup and teardown of the tests and should not be considered for analysis. This is a strong ergonomic change from TURF.js but was recommended in the Future Work of "*Performance comparison of simplification algorithms for polygons in the context of web applications*" [7]. WASM can interact with SFSQL constructs through pre-populated *std::vector* pointers that have double floating-point precision. For higher-level constructs such as polygon, *std::vector* instantiations are nested. Additionally, file size considerations were not taken into account for the sake of this research.

```
turf.area(FEATURE)

turf.union(FEATURE_1, FEATURE_2)

withAstro(FEATURE, a => {
    a.area()
})

withAstro(FEATURE_1, FEATURE_2, (a1, a2) => {
    a1.union(a2)
})
```

**Figure 5 Example usage of Astro.js functions**

Benchmarking both types of functions will be done using Benchmark.js, a popular benchmarking suite for JavaScript [19]. Benchmark.js will run each suite multiple times to return

statistically significant results [19] and will be configured to report findings in ops/second (higher being better/faster).

Benchmarking suites will be created based on polygon complexity from polygons n = {3,18,33…,153}. These polygons will be n-circles in GeoJSON format. Therefore, each geospatial function will be tested with 10 different polygons of increasing complexity. For functions that take multiple arguments such as *union*, the benchmark will compare two n-circles, one of which is translated 90deg north by the circle's radius. As mentioned above, at each polygon size the suite will be running multiple times.

Astro and TURF.js functions will be benchmarked independently on different browsers to track performance. The benchmark code will be tested on the 3 of the largest browsers: Google Chrome, Safari, and Firefox. Additionally, Google Chrome will be tested in a second pass with LLVM autovectorization and SIMD support enabled. Microsoft Edge was left outside the test suite as it is based on the Chromium project and uses the V8 engine [29]. Likewise, it is possible to run the benchmarks inside Node.js, however, this was omitted as it also uses the V8 engine [13].

Non-optimized builds of Astro functions will also be CPU profiled on Google Chrome to provide insights into computational bottlenecks.

# Chapter 3

# Evaluation

## 3.1 Implementation

The source code, testbench, and analysis tools are versioned in the source-code-management (SCM) tool Git [15]. The project is structured as an NPM package that requires Node.js [13] and Yarn [14] to gather dependencies. The author used Node.js and Yarn versions 15.7.0 and 1.17.3 respectively, however subsequent minor versions should also be compatible. NPM packages are versioned in Git via “yarn.lock”, and all packages can be retrieved with Yarn.

The C++ code written for Astro, alongside JS wrapper code to allow for serialization, has been stored in src/. The project is hosted on GitHub at the link in **Appendix A**. A checkout of the Git repository has also been included with this paper.

Astro is compiled using the “js-wasmc” (WASMC) toolchain, which is a higher-level abstraction on-top of Emscripten that adds reproducible builds using Docker. As such, Docker is required for building Astro. WASMC will execute “emcc,” the Emscripten compiler, which has been configured to use the flags in **Appendix B**. This occurs inside Docker, which uses Docker image “mbullington/emsdk” [17] and EMSDK version 2.0.14. mbullington/emsdk is a downstream version of the “emscripten/emsdk” Docker image but has added Ninja compiler, a requirement for WASMC, as a dependency from the Ubuntu APT repositories.

To build Astro, Docker must be running, and an internet connection is required to download the Docker image. By default, Astro will be built in production mode. To change this behavior, assign the *DEBUG* constant to **true** inside *wasmc.js*. LLVM autovectorization is disabled by default. To enable autovectorization, which uses the WebAssembly SIMD proposal

and cannot be run on browsers without support, assign the *SIMD* constant to **true** inside *wasmc.js*.

```
const DEBUG = false
// const DEBUG = true

const SIMD = false
// const SIMD = true

const FLAGS = ['-std=c++11', '-flto', '-s ALLOW_MEMORY_GROWTH=1', '-s MALLOC=emmalloc']
if (!DEBUG) {
    FLAGS.push('-O3')
    FLAGS.push('-fno-rtti')
    FLAGS.push('-fno-exceptions')
} else {
    FLAGS.push('-O0')
    FLAGS.push('--profiling')
}

if (SIMD) {
    FLAGS.push('-msimd128')
}
```

**Figure 6 Options for Astro build**

Building the project can then be accomplished by executing “yarn && yarn build” from a Command Prompt, which will populate the “*dist*” folder. Building has been tested successfully on both macOS and Linux, however, was not tested on Windows platforms. A non-fatal error sometimes occurs while building Astro, however, it has not been noted to affect build output.

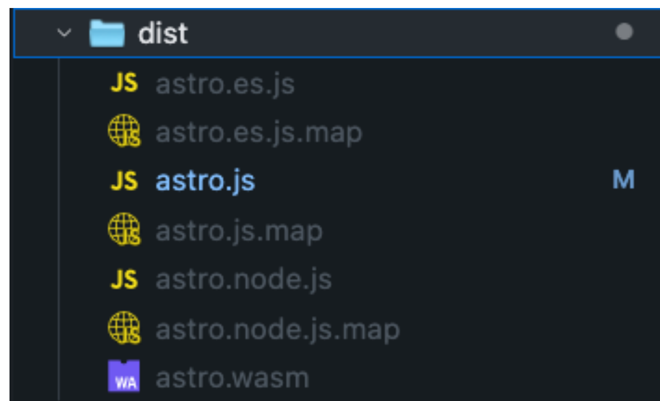



**Figure 7 dist/ folder after a successful build**

```
build failed: Error: ENOENT: no such file or directory, copyfile '/Users/plumbus/Projects/astr
o/build/release/obj/astro.wasm' -> '/Users/plumbus/Projects/astro/dist/astro.wasm'
error Command failed with exit code 1.
info Visit https://yarnpkg.com/en/docs/cli/run for documentation about this command.
```

**Figure 8 Non-fatal error on macOS systems**

All benchmarking is available in the bench/ folder. To initiate a run, the "yarn bench" command is used. On a successful run, JSON files for each run will be placed in the bench/run/ folder with *"{function}-{number representing polygon complexity}.json"* format (for example, "*area-1.json*"). The benchmark will be running with the user's Node.js binary. For browser support, Astro bundles the benchmark using Parcel [18]. Parcel will take various packages written for Node.js, specifically using CommonJS format, and bundle them in a file suitable for execution in browsers [18].

To initiate a run in a browser, run the "yarn bench-browser" command in a command prompt then go to http://localhost:5000 in the browser being tested. The Developer Tools console can be used to track the success of a browser run. On run completion, a TAR archive will be prompted to download that you can extract into "*bench/run*" manually using the system archive tool (on macOS and Linux, untested on Windows). The following versions of each software were used for analysis: Google Chrome Canary 91, Safari Technology Preview 122, Mobile Safari on iOS 14.4, and Firefox Developer Edition 87.0b9.

After running the benchmark code in various browsers and/or Node.js, a successful run can be analyzed by opening the "*visualization.nb*" file at the root directory in Wolfram Mathematica [16]. Additional instructions are included inside the notebook. All figures in Results were generated from this notebook. Mathematica 12.1.1.0 was used for analysis. Running the "yarn serve" command in a command prompt also serves an HTTP server with a playground available at http://localhost:5000/example/.

The third_party/ folder includes code from other projects, which may be under a different license than Astro itself. A version of the code from "*A new algorithm for computing Boolean operations on polygons*." [1] has been included in "*third_party/martinez*" under the Public Domain.

A modified version of Benchmark.js has been included in "*third_party/benchmark*" under the MIT License, available at "*third_party/benchmark/LICENSE*." Benchmark.js was modified to allow compatibility with the Parcel bundler.

To represent geometric types, Astro has included (third_party/geometry.hpp) the "*geometry.hpp*" library from Mapbox, sourced under the ISC license. The project is self-described as "C++ geometry types" and features generics-precision data structs for points, multi-points, line strings, multi-line strings, and polygons. By design, these data structures share conceptual meaning with their GeoJSON counterparts.

The results were computed with the following specifications: macOS Catalina 10.15.6, AMD Ryzen 5600X, 32 GB 3000MHz DDR4 RAM.

### 3.2 Configuration 1

Configuration 1 runs in the **Google Chrome** browser with Astro.js debug flag set to **false**. The x-axis scales linearly with n-complexity (to convert to n-complexity, use *3 + index*15*). The y-axis represents operations/sec, with higher being better/faster.

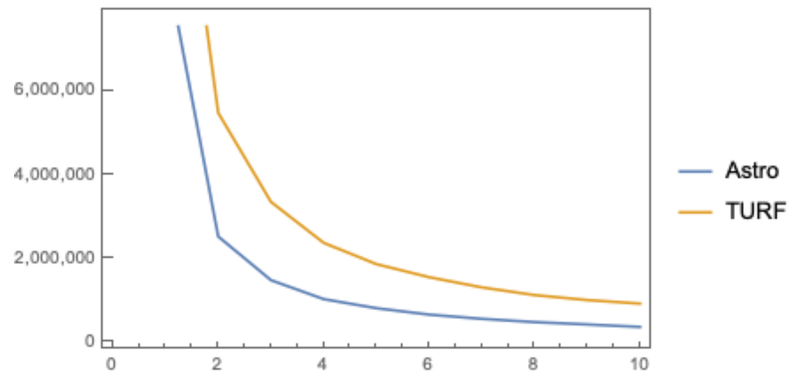



Figure 9 Configuration 1 Area Results

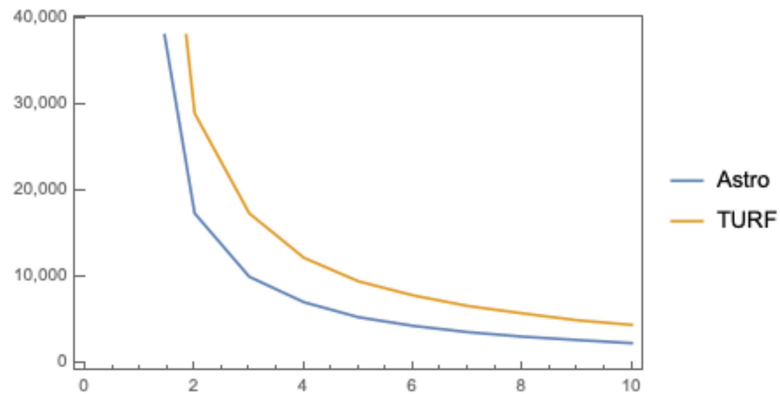



Figure 10 Configuration 1 Union Results

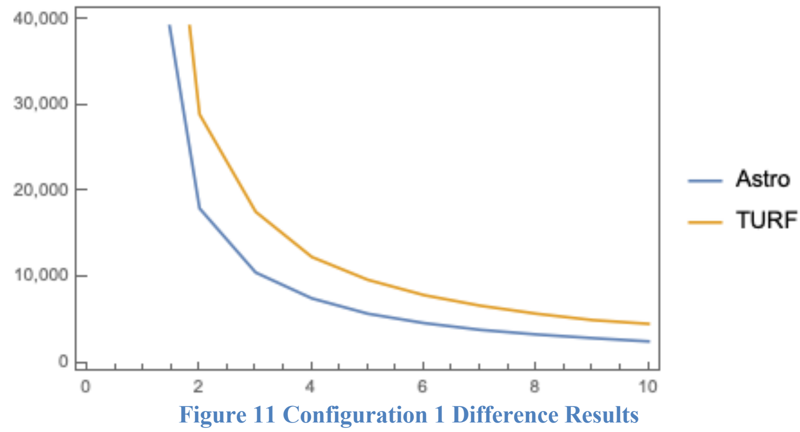



Figure 11 Configuration 1 Difference Results

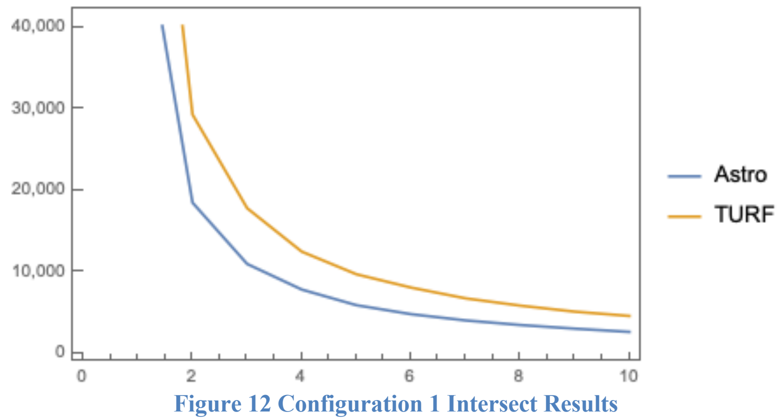



Figure 12 Configuration 1 Intersect Results

As seen in **Table 1** below, in Configuration 1, TURF.js was faster in all cases than the Astro/WASM implementation. Performance degraded similarly between the TURF.js and Astro implementations as polygon complexity increased.

Table 1 Configuration 1 ops/sec averages

| **Function** | **Average ops/sec** |
|---|---|
| astro.area | 1 748 894 |
| turf.area | 3 379 035 |
| astro.union | 11 066.4 |
| turf.union | 18 511.1 |

| astro.difference | 11 639.4 |
|---|---|
| turf.difference | 18 468.3 |
| astro.intersect | 11 852.1 |
| turf.intersect | 18 785.3 |

### 3.3 Configuration 2

Configuration 2 runs in the **Google Chrome** browser with Astro.js debug flag set to **false**. In Configuration 2, the Chrome flag chrome://flags/#enable-webassembly-simd is set to **enabled**, and the Astro.js SIMD flag is set to **true**. The x-axis scales linearly with n-complexity (to convert to n-complexity, use *3 + index*15*). The y-axis represents operations/sec, with higher being better/faster.

As seen in **Table 2** below, in Configuration 2, TURF.js was faster in all cases than the Astro/WASM implementation. There is little discernable difference in performance compared to **Configuration 1**. Performance degraded similarly between the TURF.js and Astro implementations as polygon complexity increased.

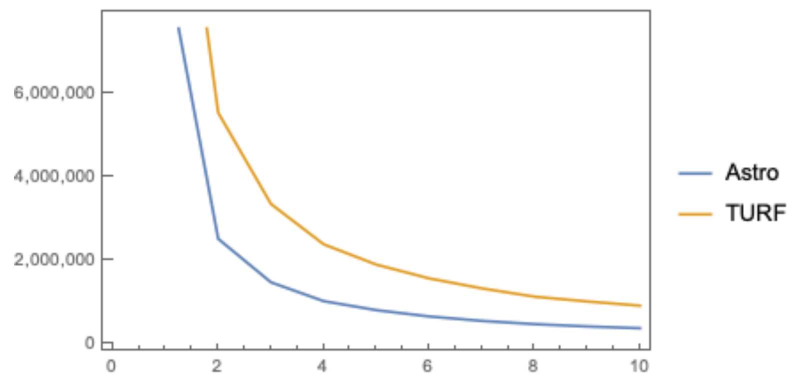


Figure 13 Configuration 2 Area Results

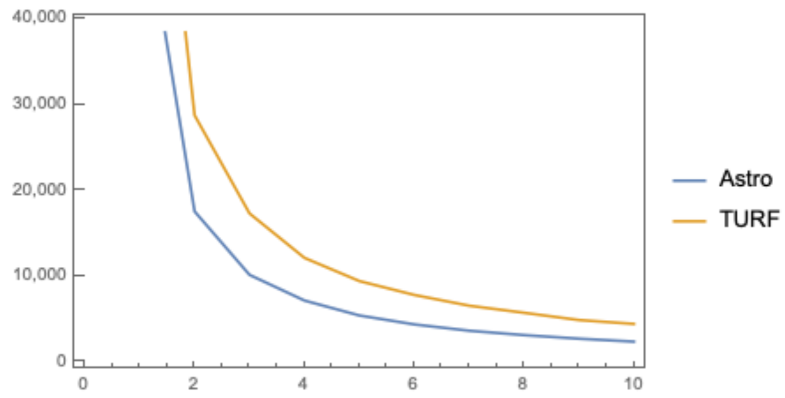



Figure 14 Configuration 2 Union Results

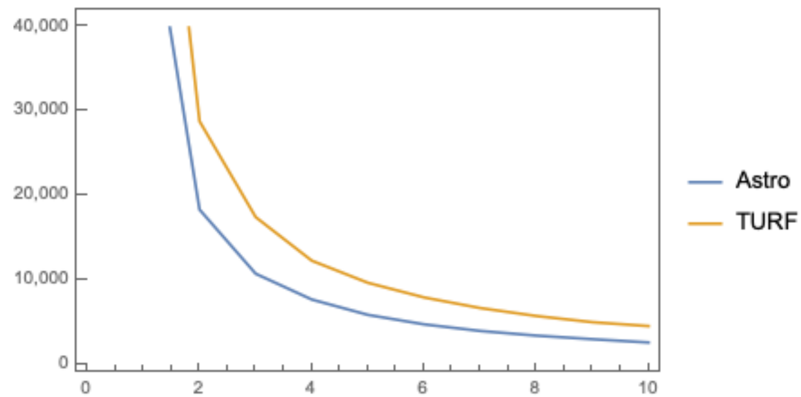



Figure 15 Configuration 2 Difference Results

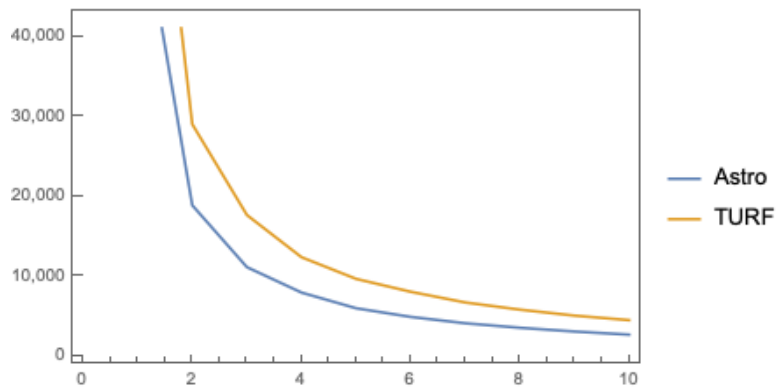



Figure 16 Configuration 2 Intersect Results

Table 2 Configuration 2 ops/sec averages

| **Function** | **Average ops/sec** |
|---|---|
| astro.area | 1 760 991 |

| | |
|---|---|
| turf.area | 3 406 072 |
| astro.union | 11 257.6 |
| turf.union | 18 345.7 |
| astro.difference | 11 854.5 |
| turf.difference | 18 430.1 |
| astro.intersect | 12 054.1 |
| turf.intersect | 18 663.9 |

## 3.4 Configuration 3

Configuration 3 runs in the **Safari** browser with Astro.js debug flag set to **false**. The x-axis scales linearly with n-complexity (to convert to n-complexity, use *3 + index*15*). The y-axis represents operations/sec, with higher being better/faster.

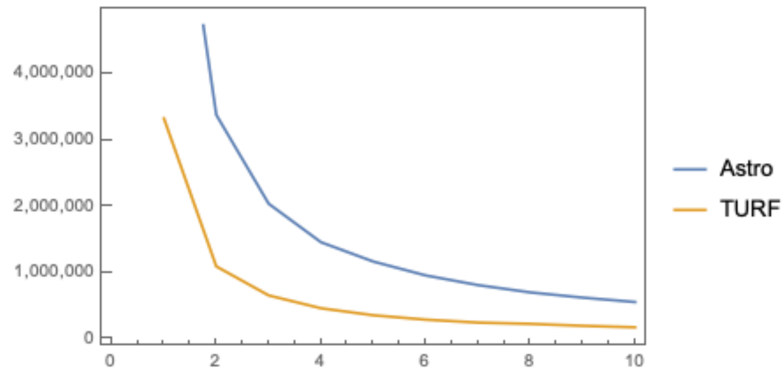



Figure 17 Configuration 3 Area Results

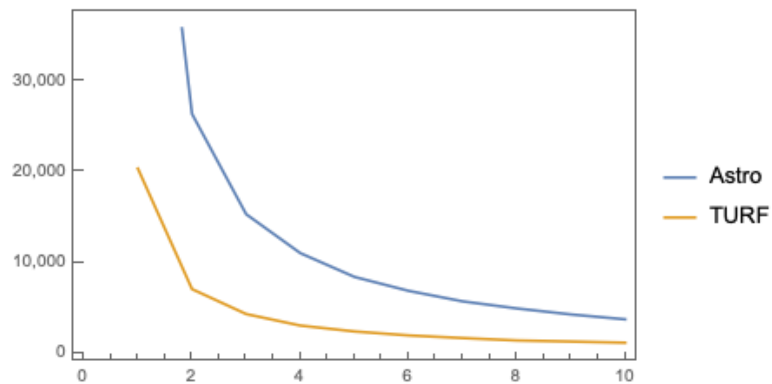



Figure 18 Configuration 3 Union Results

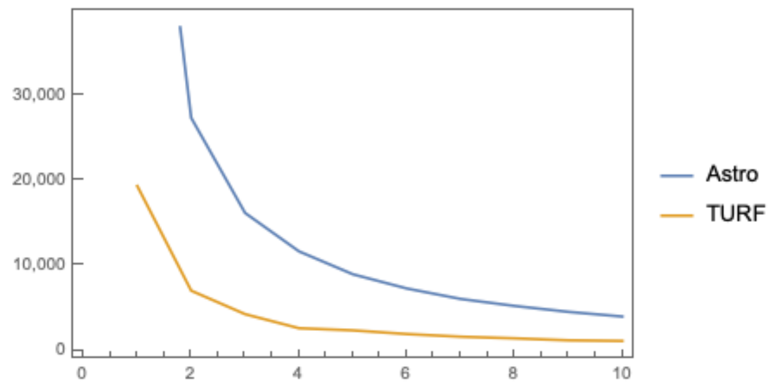



Figure 19 Configuration 3 Difference Results

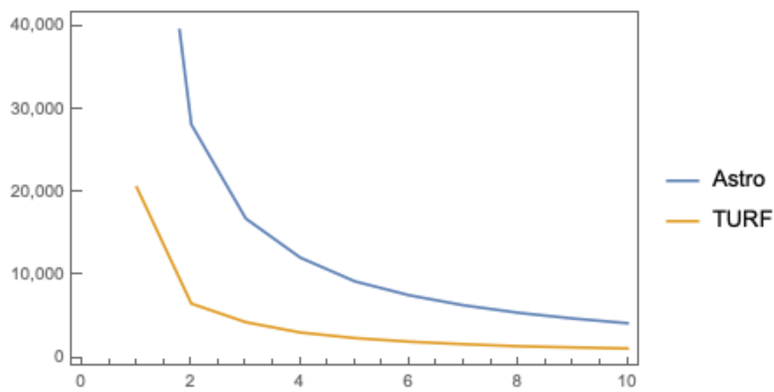



Figure 20 Configuration 3 Intersect Results

As seen in **Table 3** below, in Configuration 3, Astro/WASM was significantly faster in all cases than TURF.js. Performance degraded similarly between the TURF.js and Astro implementations as polygon complexity increased.

Table 3 Configuration 3 ops/sec averages

| Function | Average ops/sec |
| --- | --- |
| astro.area | 2 048 620 |
| turf.area | 703 621 |
| astro.union | 16 374.1 |
| turf.union | 4 419.83 |

| | |
|---|---|
| astro.difference | 17 018.4 |
| turf.difference | 4 271.31 |
| astro.intersect | 17 534.2 |
| turf.intersect | 4 417.94 |

## 3.5 Configuration 4

Configuration 4 runs in the **Mozilla Firefox** browser with Astro.js debug flag set to false. The x-axis scales linearly with n-complexity (to convert to n-complexity, use *3 + index*15*). The y-axis represents operations/sec, with higher being better/faster.

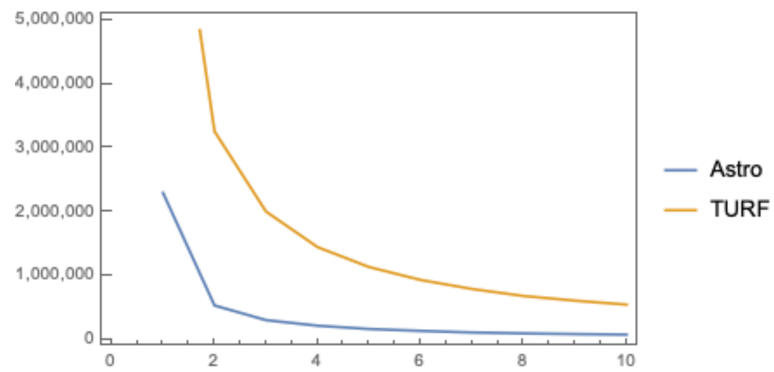



Figure 21 Configuration 4 Area Results

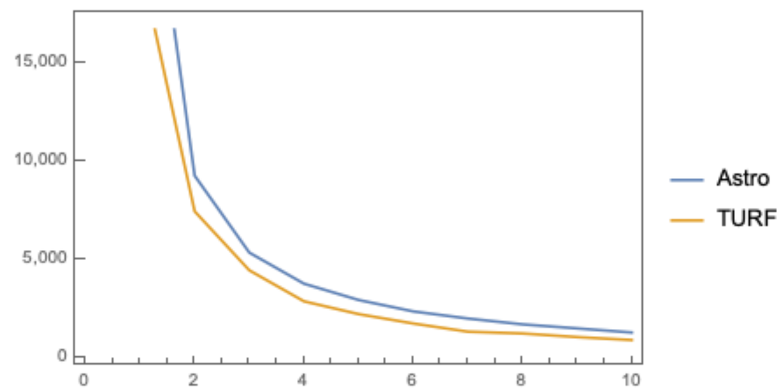



Figure 22 Configuration 4 Union Results

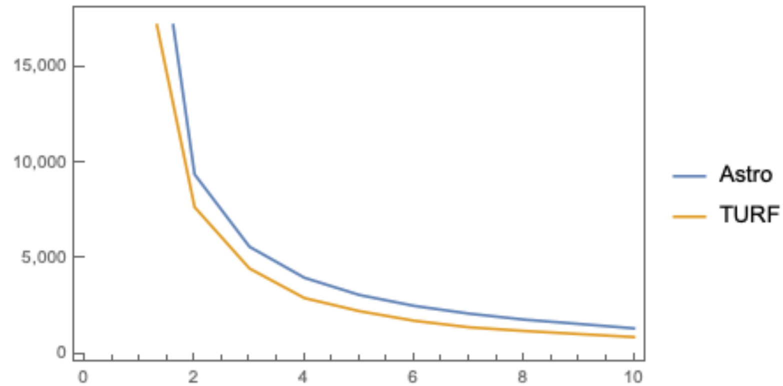



Figure 23 Configuration 4 Difference Results

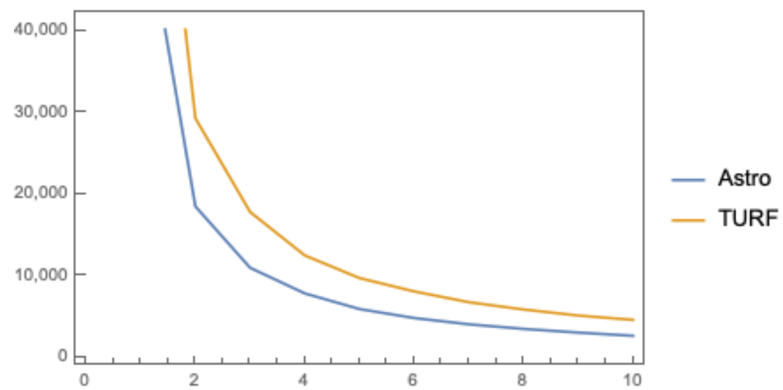



Figure 24 Configuration 4 Intersect Results

As seen in **Table 4** below, in Configuration 4, TURF.js was significantly faster than Astro/WASM for area calculations. For Boolean operators, both had similar performance but with Astro/WASM being consistently higher. Performance degraded similarly between the TURF.js and Astro implementations as polygon complexity increased in all cases.

Table 4 Configuration 4 ops/sec averages

| **Function** | **Average ops/sec** |
|---|---|
| astro.area | 397 131 |
| turf.area | 2 019 771 |
| astro.union | 5 901.77 |

| | |
|---|---|
| turf.union | 4 302.2 |
| astro.difference | 6 072.81 |
| turf.difference | 4 500.4 |
| astro.intersect | 6 283.98 |
| turf.intersect | 4 412.17 |

## 3.6 Analysis

The analysis was run in the **Google Chrome** browser with Astro.js debug flag set to true. Analysis was conducted via CPU Profiling the playground as described in **Implementation** by executing “yarn serve” in a command prompt then navigating to http://localhost:5000/example/. Clicking on “*Button that fires astro1.union(astro2)*” will start a CPU profile that can be analyzed.

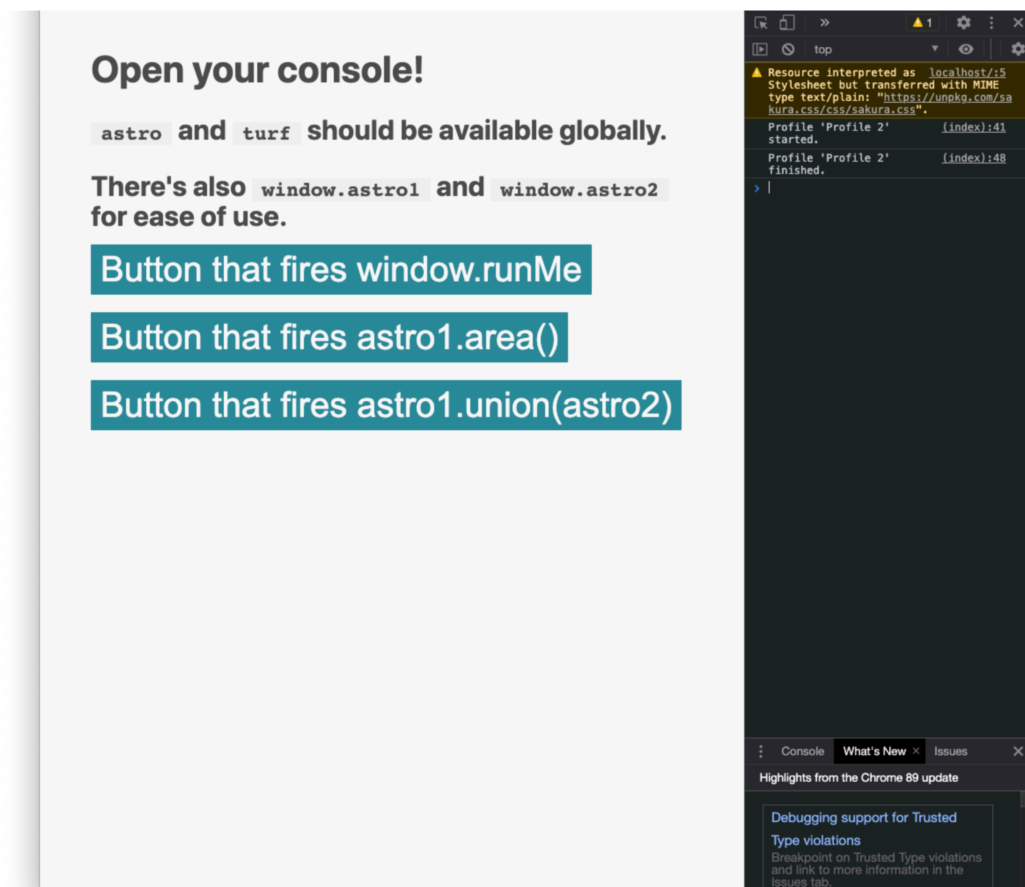



Figure 25 Screenshot of example playground

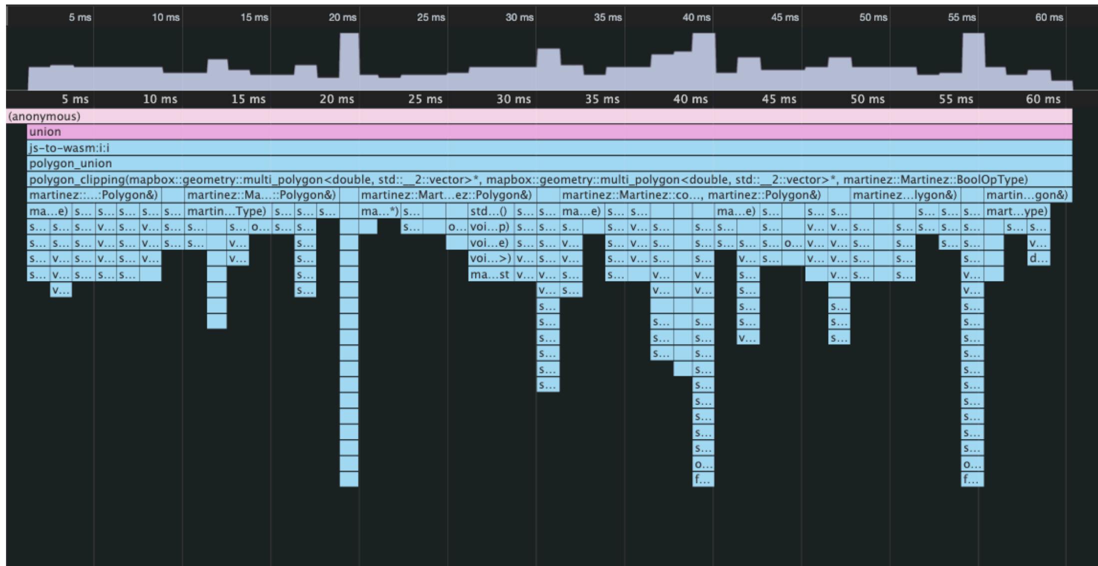
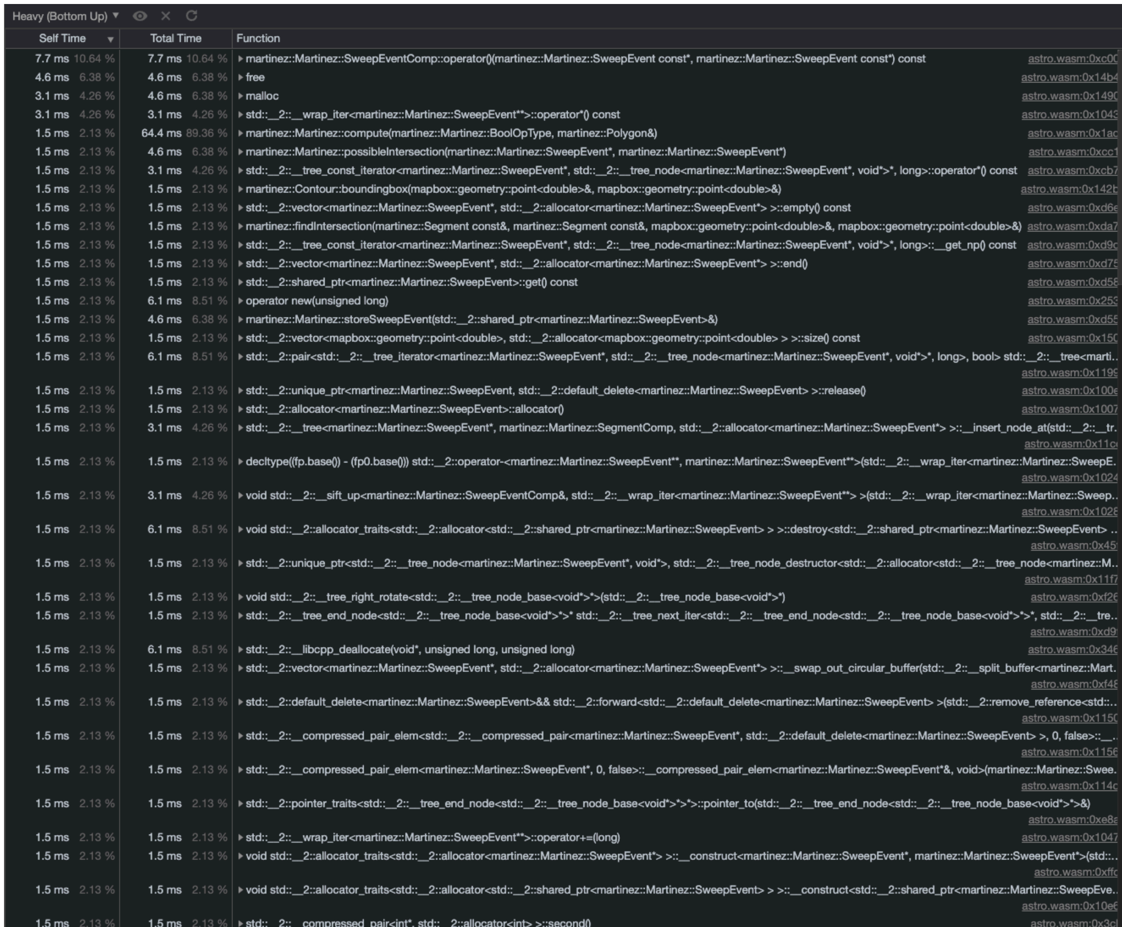



Figure 26 JavaScript Profiler Chart



Heavy (Bottom Up)

| Self Time | | Total Time | | Function | |
|---|---|---|---|---|---|
| 7.7 ms | 10.64 % | 7.7 ms | 10.64 % | martinez::Martinez::SweepEventComp::operator()(martinez::Martinez::SweepEvent const*, martinez::Martinez::SweepEvent const*) const | astro.wasm:0xc00 |
| 4.6 ms | 6.38 % | 4.6 ms | 6.38 % | free | astro.wasm:0x14b4 |
| 3.1 ms | 4.26 % | 4.6 ms | 6.38 % | malloc | astro.wasm:0x1490 |
| 3.1 ms | 4.26 % | 3.1 ms | 4.26 % | std::__2::__wrap_iter<martinez::Martinez::SweepEvent**>::operator*() const | astro.wasm:0x1043 |
| 1.5 ms | 2.13 % | 64.4 ms | 89.36 % | martinez::Martinez::compute(martinez::Martinez::BoolOpType, martinez::Polygon&) | astro.wasm:0x1ac |
| 1.5 ms | 2.13 % | 4.6 ms | 6.38 % | martinez::Martinez::possibleIntersection(martinez::Martinez::SweepEvent*, martinez::Martinez::SweepEvent*) | astro.wasm:0xcc1 |
| 1.5 ms | 2.13 % | 3.1 ms | 4.26 % | std::__2::__tree_const_iterator<martinez::Martinez::SweepEvent*, std::__2::__tree_node<martinez::Martinez::SweepEvent*, void*>*, long>::operator*() const | astro.wasm:0xcb7 |
| 1.5 ms | 2.13 % | 1.5 ms | 2.13 % | martinez::Contour::boundingbox(mapbox::geometry::point<double>&, mapbox::geometry::point<double>&) | astro.wasm:0x142b |
| 1.5 ms | 2.13 % | 1.5 ms | 2.13 % | std::__2::vector<martinez::Martinez::SweepEvent*, std::__2::allocator<martinez::Martinez::SweepEvent*> >::empty() const | astro.wasm:0xd6e |
| 1.5 ms | 2.13 % | 1.5 ms | 2.13 % | martinez::findIntersection(martinez::Segment const&, martinez::Segment const&, mapbox::geometry::point<double>&, mapbox::geometry::point<double>&) | astro.wasm:0xda7 |
| 1.5 ms | 2.13 % | 1.5 ms | 2.13 % | std::__2::__tree_const_iterator<martinez::Martinez::SweepEvent*, std::__2::__tree_node<martinez::Martinez::SweepEvent*, void*>*, long>::__get_np() const | astro.wasm:0xd9c |
| 1.5 ms | 2.13 % | 1.5 ms | 2.13 % | std::__2::vector<martinez::Martinez::SweepEvent*, std::__2::allocator<martinez::Martinez::SweepEvent*> >::end() | astro.wasm:0xd75 |
| 1.5 ms | 2.13 % | 1.5 ms | 2.13 % | std::__2::shared_ptr<martinez::Martinez::SweepEvent>::get() const | astro.wasm:0xd58 |
| 1.5 ms | 2.13 % | 6.1 ms | 8.51 % | operator new(unsigned long) | astro.wasm:0x253 |
| 1.5 ms | 2.13 % | 4.6 ms | 6.38 % | martinez::Martinez::storeSweepEvent(std::__2::shared_ptr<martinez::Martinez::SweepEvent>&) | astro.wasm:0xd55 |
| 1.5 ms | 2.13 % | 1.5 ms | 2.13 % | std::__2::vector<mapbox::geometry::point<double>, std::__2::allocator<mapbox::geometry::point<double> > >::size() const | astro.wasm:0x150 |
| 1.5 ms | 2.13 % | 6.1 ms | 8.51 % | std::__2::pair<std::__2::__tree_iterator<martinez::Martinez::SweepEvent*, std::__2::__tree_node<martinez::Martinez::SweepEvent*, void*>*, long>, bool> std::__2::__tree<marti... | astro.wasm:0x1199 |
| 1.5 ms | 2.13 % | 1.5 ms | 2.13 % | std::__2::unique_ptr<martinez::Martinez::SweepEvent, std::__2::default_delete<martinez::Martinez::SweepEvent> >::release() | astro.wasm:0x100e |
| 1.5 ms | 2.13 % | 1.5 ms | 2.13 % | std::__2::allocator<martinez::Martinez::SweepEvent>::allocator() | astro.wasm:0x1007 |
| 1.5 ms | 2.13 % | 3.1 ms | 4.26 % | std::__2::__tree<martinez::Martinez::SweepEvent*, martinez::Martinez::SegmentComp, std::__2::allocator<martinez::Martinez::SweepEvent*> >::__insert_node_at(std::__2::__tr... | astro.wasm:0x11c |
| 1.5 ms | 2.13 % | 1.5 ms | 2.13 % | decltype((fp.base()) - (fp0.base())) std::__2::operator-<martinez::Martinez::SweepEvent**, martinez::Martinez::SweepEvent**>(std::__2::__wrap_iter<martinez::Martinez::SweepE... | astro.wasm:0x1024 |
| 1.5 ms | 2.13 % | 3.1 ms | 4.26 % | void std::__2::__sift_up<martinez::Martinez::SweepEventComp&, std::__2::__wrap_iter<martinez::Martinez::SweepEvent**> >(std::__2::__wrap_iter<martinez::Martinez::Sweep... | astro.wasm:0x1028 |
| 1.5 ms | 2.13 % | 6.1 ms | 8.51 % | void std::__2::allocator_traits<std::__2::allocator<std::__2::shared_ptr<martinez::Martinez::SweepEvent> > >::destroy<std::__2::shared_ptr<martinez::Martinez::SweepEvent> ... | astro.wasm:0x45 |
| 1.5 ms | 2.13 % | 1.5 ms | 2.13 % | std::__2::unique_ptr<std::__2::__tree_node<martinez::Martinez::SweepEvent*, void*>, std::__2::__tree_node_destructor<std::__2::allocator<std::__2::__tree_node<martinez::M... | astro.wasm:0x11f7 |
| 1.5 ms | 2.13 % | 1.5 ms | 2.13 % | void std::__2::__tree_right_rotate<std::__2::__tree_node_base<void*>*>(std::__2::__tree_node_base<void*>*) | astro.wasm:0xf26 |
| 1.5 ms | 2.13 % | 1.5 ms | 2.13 % | std::__2::__tree_end_node<std::__2::__tree_node_base<void*>*>* std::__2::__tree_next_iter<std::__2::__tree_end_node<std::__2::__tree_node_base<void*>*>*, std::__2::__tre... | astro.wasm:0xd9 |
| 1.5 ms | 2.13 % | 6.1 ms | 8.51 % | std::__2::__libcpp_deallocate(void*, unsigned long, unsigned long) | astro.wasm:0x346 |
| 1.5 ms | 2.13 % | 1.5 ms | 2.13 % | std::__2::vector<martinez::Martinez::SweepEvent*, std::__2::allocator<martinez::Martinez::SweepEvent*> >::__swap_out_circular_buffer(std::__2::__split_buffer<martinez::Mart... | astro.wasm:0xf48 |
| 1.5 ms | 2.13 % | 1.5 ms | 2.13 % | std::__2::default_delete<martinez::Martinez::SweepEvent>&& std::__2::forward<std::__2::default_delete<martinez::Martinez::SweepEvent> >(std::__2::remove_reference<std::... | astro.wasm:0x1150 |
| 1.5 ms | 2.13 % | 1.5 ms | 2.13 % | std::__2::__compressed_pair_elem<std::__2::__compressed_pair<martinez::Martinez::SweepEvent*, std::__2::default_delete<martinez::Martinez::SweepEvent> >, 0, false>::__... | astro.wasm:0x1156 |
| 1.5 ms | 2.13 % | 1.5 ms | 2.13 % | std::__2::__compressed_pair_elem<martinez::Martinez::SweepEvent*, 0, false>::__compressed_pair_elem<martinez::Martinez::SweepEvent*&, void>(martinez::Martinez::Swee... | astro.wasm:0x114c |
| 1.5 ms | 2.13 % | 1.5 ms | 2.13 % | std::__2::pointer_traits<std::__2::__tree_end_node<std::__2::__tree_node_base<void*>*>*>::pointer_to(std::__2::__tree_end_node<std::__2::__tree_node_base<void*>*>&) | astro.wasm:0xe8a |
| 1.5 ms | 2.13 % | 1.5 ms | 2.13 % | std::__2::__wrap_iter<martinez::Martinez::SweepEvent**>::operator+=(long) | astro.wasm:0x1047 |
| 1.5 ms | 2.13 % | 1.5 ms | 2.13 % | void std::__2::allocator_traits<std::__2::allocator<martinez::Martinez::SweepEvent*> >::__construct<martinez::Martinez::SweepEvent*, martinez::Martinez::SweepEvent*>(std::... | astro.wasm:0xffc |
| 1.5 ms | 2.13 % | 1.5 ms | 2.13 % | void std::__2::allocator_traits<std::__2::allocator<std::__2::shared_ptr<martinez::Martinez::SweepEvent> > >::__construct<std::__2::shared_ptr<martinez::Martinez::SweepEve... | astro.wasm:0x10e6 |
| 1.5 ms | 2.13 % | 1.5 ms | 2.13 % | std::__2::__compressed_pair<int*, std::__2::allocator<int> >::second() | astro.wasm:0x3cb |

Figure 27 Profiler bottom-up tree

As shown in **Figure 26**, the bulk of the computational time is spent inside WASM, showing the solution works in minimizing serialization time between JavaScript and WASM. In **Figure 27**, the most Self Time is spent on the functions SweepEventComp, free, and malloc. Potential optimizations in the future may include analyzing *emmalloc* performance vs. *dlmalloc* in the EMCC options of **Appendix B** [27]. Significant time spent inside STL containers such as set show possibilities for further optimization as well.

# Chapter 4

# Conclusion

Having rewritten targeted geospatial algorithms from TURF.js in C++, intending to analyze performance benefits to web applications, the benchmarks in **Chapter 3** can give nuanced but informational guidance. The outcomes of all algorithms degraded similarly as polygon complexity increased, showing a correlation. In each configuration, the faster implementation for a given algorithm was faster for every polygon complexity. As this is the case, the research does not suggest a threshold of polygon complexity at which one implementation would be more performant than the other.

For the area algorithm implemented by "@turf/area," all configurations except **Configuration 3** (Safari) showed better performance for TURF.js over the Astro/WASM implementation. Contrasting for Boolean operations, **Configuration 3** (Safari) and **Configuration 4** (Mozilla Firefox) saw dramatic and marginal gains in performance respectively for Astro/WASM when compared to TURF.js.

**Configuration 1** and **Configuration 2** (both Google Chrome, with and without LLVM autovectorization enabled) showed TURF.js as significantly faster for all algorithms and polygon sizes.

The research shows that an optimal solution is based heavily on browser configuration. For web applications targeting primarily Safari platforms, the default browser on macOS and only web rendering engine available on iOS/iPadOS, Astro/WASM performance of geospatial algorithms saw dramatic gains over TURF.js for each targeted algorithm.

For Chromium-based browsers (Google Chrome, Microsoft Edge, Opera, etc.), which have a combined over 78% browser market share as of October 2020 [26], TURF.js continues to

have excellent JavaScript performance that supersedes Astro/WASM in all circumstances. TURF.js is also recommended for Mozilla Firefox browsers, where Turf.js and Astro/WASM have varying performance characteristics for each targeted algorithm, with the upper bound of Astro/WASM performance being similar to JavaScript.

# Chapter 5

# Future Work

The paper presents a library named Astro, which rewrites TURF.js in C/C++ to analyze their performance in the WebAssembly VM compared to TURF.js itself. Astro can be extended/improved in many ways.

Two algorithms are currently covered by Astro, area and Boolean operations. Increasing the surface area of the algorithms could provide more data points and a more holistic picture of the performance of Astro compared to TURF.js.

Astro may benefit from time spent further analyzing and optimizing C++ code. Astro is currently competitive or better than TURF.js in multiple configurations, however, both the JavaScript and WASM VMs are the subject of large amounts of collaboration and development work. The benchmark may be evaluated with the same configurations in the future, with updated versions of each browser, to have different results and recommendations.

Astro may also benefit from a more fundamental shift in its structure. Comparing **Configuration 1** and **Configuration 2**, LLVM's autovectorization did not provide meaningful performance benefits over the configuration without SIMD instructions enabled. Re-writing algorithms to take advantage of SIMD instructions directly may be possible or provide performance benefits.

Lastly compared to TURF.js, Astro/WASM has a more complex structure and higher maintenance cost, partially due to the underlying language choice of C++. C++ may be harder than JavaScript to contribute, track memory/logical bugs, etc. This issue does not exist with the TURF.js project but could be potentially mitigated by maximizing reuse and/or moving to a language with different semantics. This is potentially possible due to WebAssembly's definition

as a bytecode language with other backends, including one from LLVM that converts LLVM IR to WASM [4]. A few languages with potential in this area include Swift, Rust, and Go.

# Appendix A

## Resources

Astro.wasm is provided as a GitHub repository at this location: https://github.com/mbullington/astro-wasm

A checkout of the repository at commit 7e0cf5d512fcd32d52da5473cff56a4b653a1d8c has been included as a supplemental file (ZIP format) in publication. This commit was also used to compute all results and analysis.

# Appendix B

## EMCC Compiler Flags

```
-std=c++11
-Wno-shorten-64-to-32
-Wno-unused-function
-Wno-unused-parameter
-Wno-unused-variable
-Wno-null-conversion
-Wno-c++11-extensions
-Wtautological-compare
-Dexport="__attribute__((used))"
-DNDEBUG
-flto
-fno-rtti
-fno-exceptions
-O3
-I/src/third_party/geometry.hpp/include
-s ALLOW_MEMORY_GROWTH=1
-s MALLOC=emmalloc
```

# Appendix C

## GeoJSON Example


```
{
  "type": "FeatureCollection",
  "features": [{
    "type": "Feature",
    "geometry": {
      "type": "Point",
      "coordinates": [102.0, 0.5]
    },
    "properties": {
      "prop0": "value0"
    }
  }, {
    "type": "Feature",
    "geometry": {
      "type": "LineString",
      "coordinates": [
        [102.0, 0.0],
        [103.0, 1.0],
        [104.0, 0.0],
        [105.0, 1.0]
      ]
    },
    "properties": {
      "prop0": "value0",
      "prop1": 0.0
    }
  }, {
    "type": "Feature",
    "geometry": {
      "type": "Polygon",
      "coordinates": [
        [
          [100.0, 0.0],
          [101.0, 0.0],
          [101.0, 1.0],
          [100.0, 1.0],
          [100.0, 0.0]
        ]
      ]
    },
    "properties": {
      "prop0": "value0",
      "prop1": {
        "this": "that"
      }
    }
  }]
}
```

# BIBLIOGRAPHY


[1] F. Martínez, C. Ogayar, J. R. Jiménez, and A. J. Rueda, “A simple algorithm for Boolean operations on polygons,” *Advances in Engineering Software*, vol. 64, pp. 11–19, Oct. 2013, doi: 10.1016/j.advengsoft.2013.04.004.

[2] D. DiBiase, University Consortium for Geographic Information Science, Model Curricula Task Force, and Body of Knowledge Advisory Board, Eds., Geographic information science and technology body of knowledge, 1st ed. Washington, D.C: Association of American Geographers, 2006.

[3] A. Haas *et al.*, “Bringing the Web up to Speed with WebAssembly.”

[4] A. Zakai, “Emscripten: an LLVM-to-JavaScript compiler,” in *Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion*, New York, NY, USA, Oct. 2011, pp. 301–312, doi: 10.1145/2048147.2048224.

[5] J. A. Slater and S. Malys, “WGS 84 — Past, Present and Future,” in Advances in Positioning and Reference Frames, Berlin, Heidelberg, 1998, pp. 1–7.

[6] A. Zakai, “Big Web App? Compile It!,” Accessed: Mar. 31, 2021. [Online]. Available: https://kripken.github.io/mloc_emscripten_talk/#/.

[7] A. Melch, “Performance comparison of simplification algorithms for polygons in the context of web applications,” Aug. 2019, [Online]. Available: https://mt.melch.pro/mt-polygon-simplification.pdf.

[8] R. G. Chamberlain and W. H. Duquette, “Some Algorithms for Polygons on a Sphere,” p. 32, 2007.

[9] *The GeoJSON Format.*, RFC-7946, 2016.

[10] Deepti Gandluri and Thomas Lively, “Fast, parallel applications with WebAssembly SIMD,” V8, Jan. 30, 2020. https://v8.dev/features/simd (accessed Apr. 03, 2021).

[11] Deepti Gandluri and Ng Zhi An, “WebAssembly SIMD,” Chrome Platform Status. https://www.chromestatus.com/feature/6533147810332672 (accessed Apr. 03, 2021).

[12] “@turf/turf,” npm. https://www.npmjs.com/package/@turf/turf (accessed Apr. 04, 2021).

[13] Node.js, “Node.js,” Node.js. https://nodejs.org/en/ (accessed Apr. 04, 2021).

[14] “Home,” Yarn - Package Manager. https://yarnpkg.com/ (accessed Apr. 04, 2021).

[15] “Git.” https://git-scm.com/ (accessed Apr. 04, 2021).

[16] “Wolfram Mathematica: Modern Technical Computing.” https://www.wolfram.com/mathematica/ (accessed Apr. 04, 2021).

[17] “mbullington/emsdk.” https://hub.docker.com/r/mbullington/emsdk (accessed Apr. 04, 2021).

[18] Devon Govett, “JavaScript,” Parcel. https://parceljs.org/javascript.html (accessed Apr. 04, 2021).

[19] Mathias Bynens and John-David Dalton, “Benchmark.js.” https://benchmarkjs.com/ (accessed Apr. 04, 2021).

[20] *The JSON data interchange syntax*, 2nd edition, ECMA-404, 2017.

[21] *ECMAScript® Language Specification*, 5.1 edition, ECMA-262, 2011.

[22] “JSON.parse() - JavaScript,” MDN Web Docs. https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/JSON/parse (accessed Apr. 04, 2021).

[23] “JSON.stringify() - JavaScript,” MDN Web Docs. https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/JSON/stringify (accessed Apr. 04, 2021).

[24] "turf/index.ts at 2e9d3d51f765a814c2cad90e88ff86e27c9e066f," Turfjs/turf, Dec. 08, 2020. https://github.com/Turfjs/turf/blob/2e9d3d51f765a814c2cad90e88ff86e27c9e066f/packages/turf-area/index.ts (accessed Apr. 04, 2021).

[25] M. Fogel, mfogel/polygon-clipping. 2021. https://github.com/mfogel/polygon-clipping (accessed Apr. 04, 2021).

[26] "Browser market share," NetMarketShare. https://netmarketshare.com/browser-market-share.aspx (accessed Apr. 04, 2021).

[27] A. Zakai, "emmalloc option (#6249) · emscripten-core/emscripten@78d3f20," GitHub. /emscripten-core/emscripten/commit/78d3f20b8d515a4e1b92434519dbe7b088628fea (accessed Apr. 04, 2021).

[28] "Release Notes," Emscripten 2.0.16 documentation. https://emscripten.org/docs/introducing_emscripten/release_notes.html (accessed Apr. 04, 2021).

[29] J. Belfiore, "Microsoft Edge: Making the web better through more open source collaboration," Windows Experience Blog, Dec. 06, 2018. https://blogs.windows.com/windowsexperience/2018/12/06/microsoft-edge-making-the-web-better-through-more-open-source-collaboration/ (accessed Apr. 04, 2021).

[30] "Roadmap," WebAssembly. https://webassembly.org/roadmap/ (accessed Apr. 04, 2021).

# ACADEMIC VITA OF MICHAEL BULLINGTON

## Education

**Pennsylvania State University; Schreyer Honors College** **August 2017 – May 2021**

- **Computer Science**, Bachelor of Science **State College, PA**

## Work Experience

**DJI Technology Ltd.** **May 2019 – Dec 2020**

**Front-End Web Developer Intern** **Remote; Palo Alto, CA**

- **Product Manager** focusing on the North American market for Public Safety and Enterprise. Responsibilities included planning roadmap/releases for our 4-person team, initiating compliance with the **NIST 800-53** framework (authoring over 75 controls), and creating high-fidelity designs to collaborate with stakeholders.
- Architected and open-sourced team JavaScript infrastructure, style guide, and all non-proprietary reusable code. Contributed to **Turf.JS** and other open-source projects representing DJI.
- Lead front-end web development for a government/private industry project using **Vue** and **Mapbox**. The product required close compliance with government specification and verification.
- Re-built the DJI AirWorks experience to adapt to an online-first conference amid the **COVID-19** pandemic. Duties were shared across the US Web Team and include: design, development, QA testing, and deployment on AWS East.
- Helped to interview candidates for web positions in the Palo Alto office.

**Wolfram Research Inc.** **May 2018 – May 2019**

**Intern, Core Engine R&D** **Remote; Champaign, IL**

- Overhauled typesetting engine for Wolfram Cloud, improving aesthetic and better matching **TeX** metrics.
- Created a library-agnostic mapping solution with default **Leaflet** driver, combining Wolfram's rich computational intelligence with mapping on the web.

**Acuity Brands; DGLogik Inc.** **April 2015 – May 2018**

**IoT Software Engineer** **Remote; Oakland, CA**

- Helped develop a secure protocol for IoT devices with SDK implementations. It is now an integral part of the company's stack and ships to thousands of customers worldwide such as **Cisco**, **IBM**, and **Intel**.
- Shipped multiple foundational features for Acuity Brands' Atrius suite using **D3, THREE.js**, and **Mapbox**. Product has since been used by customers such as **Microsoft** and **Target**.

**HackPSU** **September 2019 — Present**

- MLH hackathon organized by Penn State students that runs each semester. Created and designed a mobile application for the event using **React Native** and **Firebase**.

## Miscellaneous

**German-American Partnership Program** **June 2016**