1. INTRODUCTION

--- *Standards*

Geodata experts are often concerned with the creation and adoption of common standards. (WFS WMS, cityJSON). This is done to prevent an *interrelating mess*: a graph with each node connecting to every other node. Software engineers are taught to avoid O(n^2) algorithms as much as possible, and this is a similar phenomenon. Sometimes, this problem can be solved by introducing one intermediary node, after which all different nodes only need to be concerned about its read-write relationship to only that intermediary. (name a vivid example + SOURCE)

--- *WebAssembly*

WebAssembly is an emergent technology / standard which the exact same goal (SOURCE: WASM paper). It is a compilation target meant to be platform & source independent. It attempts to be the ultimate intermediary between software and hardware, which would be a dream for developers in that sense: "Run Anything Anywhere".

"Run Anything" means that a platitude of languages (C, C++, Rust) can be compiled to WebAssembly, with the promise that these wasm-binaries are almost as fast as native binary compilations of those same languages.

"Run Anywhere" means that it is possible to run a wasm-binary on Windows, Mac & Linux Desktops, natively on mobile devices, on servers, and even client-side in web-browsers. This runtime is also containerized, improving privacy, security against malware, and user control.

--- *Applications*

The possibility of a save, platform-independent binary target which also targets the web gives WebAssembly many interesting applications. Like Docker, it can be used to run foreign software in a save, containerized manner (SOURCE: DOCKER, WASI). This is one of the reasons why wasm is also supported by most major browsers, making it the 4th type of 'code' to run in a browser, alongside javascript, css and html.

This means that all existing libraries written in any language are now able to be distributed by the web, enabling applications which are both powerful and accessible. The Google Earth web application uses WebAssembly for example (SOURCE: Google Earth). This way, the C++ codebase used for the desktop application could be re-used and repurposed for the web, instead of starting over again.

--- *FAIR*

An important side-note is the relationship of WebAssembly and the FAIR principles. The FAIR principles are a collection of four well-established assessment criteria used for judging the usability of software applications (SOURCE). They stand for Findable, Accessible, Interoperable, and Reusable. WebAssembly has the potential to improve all four of those criteria for a piece of software:

WASM web apps: There is no delay between Findability and Accessibility. As soon as it can be found, it can be accessed.

WASM containerized: If the core logic of something is compiled into a wasm library, than this logic becomes Interoperable and Reusable. We can be sure that it will produce the same results, wherever it is run. Write once, use anywhere <-> Collect once, use multiple times

--- *Uncertainty*

Many aspects of WebAssembly remain, however, uncertain. The performance gain over compiling to javascript, or native development of javascript, are highly application dependent (SOURCE: NOT SO FAST). The performance lost by using a 'virtual binary' like wasm over a native binary optimized specifically for certain hardware is also application dependent (SOURCE: NOT SO FAST). Lastly, since WebAssembly is very bare-bones and does not make many assumptions about its host environment, it is unclear how 'usable' wasm is in practice. Many tools around it exist to make working with wasm easier (wasm-pack & emscriptem), but it remains unsure what practical troubles could arise when using WebAssembly for certain applications.

--- *Relevance*

It is unclear what WebAssembly exactly means for the geospatial community. The potential of improving the FAIR qualities of geoprocessing software seems very promising:

- What if the exact same code could be used client-side and server-side?
- What if all C++ based libraries such as CGAL could be accessed from a browser, without needing to be installed?
- What if processes which were previously hard to chain together could suddenly work together perfectly?

At the same time, we do not know if these advantages mean anything if it turns out that wasm is too difficult to use in practice, or just not performant enough to be a viable alternative to native geoprocessing tools. Websites with many wasm files could take too long to load, or accessing certain old C++ libraries on the web might not yield any real benefits for end-users.

The potential benefits of WebAssembly, together with the many uncertainties, make research into utilizing wasm a crucial endeavour for the geospatial community. Both the technical capabilities of wasm for geomatics purposes need to be researched, as well as the capabilities in practical utilization.

--- *The Paper*

This paper attempts to judge the fitness of WebAssembly for web-geo-processing purposes. This fitness will be judged quantitatively by means of a performance analysis, as well as qualitatively by documenting the creation of a web-based geoprocessing application using WebAssembly.

This paper will perform research into the possibilities and effectiveness of compiling C++ geoprocessing libraries such as CGAL & 3dfier into WebAssembly. To explore the utilization of WebAssembly, this research will involve creating an application using these libraries. This can be seen as a "geo-wasm case study", as this

application would be impossible to create without wasm. The application will take the shape of a visual programming language, or VPL for short, to further build upon the web's advantage of accessibility.

2. RELATED WORK

2.1 On WebAssembly

- the not so fast paper

2.3 On geo-vpls:

- Ravi peter's geoflow
- the relevant vpl paper

Existing VPL's

3. RESEARCH QUESTIONS

3.1 Objectives

This paper main objective is to judge the fitness of WebAssembly for web-geo-processing purposes. This fitness will be judged quantitatively by means of a performance analysis, as well as qualitatively by documenting the creation of a web-based geoprocessing application using WebAssembly.

The main research question goes:

> ***How well does WebAssembly support a client-side geoprocessing vpl?***

This question contains two main components: WebAssembly for geo-processing, and a visual programming language. It then asks how well the one supports the other. These components are reflected in the sub-questions:

1. **GEO-WASM**: How well can C++ geoprocessing libraries such as CGAL & 3dfier be used within a web browser without needing to be installed, by using WebAssembly?
    - 2a: How well do WebAssembly compiled geoprocessing (geo-wasm) libraries perform compared to native, cli usage?
    - 2b: How to handle types / data models between multiple, unrelated `wasm` libraries?
    - 2c: How do C++ geoprocessing libraries differ from all other C++ libraries?
    - 2d: What does this difference mean for `wasm` compilation and usage?
2. **GEO-WEB-VPL**: How to make a web-based, client-side, vpl geoprocessing environment?
    - 1a. **GEO**: What basic features does a geoprocessing environment need?
    - 1b. **WEB**: What advantages and limitations does a HTML5, CSS & JS based environment and interface give us?
    - 1c. **VPL**: What are the advantages and disadvantages of using a vpl?
3. **GEO_WASM + GEO-WEB-VPL**: How well can geo-wasm libs be used within the context of a geo-web-vpl?
    - 3a: What data must a geo-wasm provide in order to become usable within a geo-web-vpl?
    - 3b: How can this data be utilized by the geo-web-vpl?
    - 3c: How are the geo-wasm libraries distributed?

3.2 Scope

LIMITED TO:

- WebAssembly for web-usage
-

NOT:

- web processing services or server orchestration
- WASI

•

4. Motivation

This paper contains auxiliary motivations, in addition to the main purpose of this research.

- Last-mile problem
- insight, sandboxing & Debugging

5. Methodology (mention pre-work)

5.4 Measure

6. Planning

TODO

- write P2 presentation
- build the VPL
- apply VPL to Case Study
- build a similar application using python + jupyler, or some other conventional method
- perform tests and compare the two

7. Tools used

Languages

- C++ & Rust
- Typescript / Javascript
    - Front-end code
- WebAssembly
    - As compile target

Libraries & Tools

- Emscriptem
- Wasm-Pack
- WebGl & javascript Canvas api
    - visualization

Data

- WMS & WFS services hosted by PDOK.
- sample Geojsons from the geojson site