1.. INTRODUCTION

> [JF] NOTE: maybe this should even be more concrete. Like: CGAL in a browser would be nice. Can we do that?

--- *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)

[DIARGAM: INTERRELATION PROBLEM]

--- *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". (SOURCE: WASM PAPER | NUANCE THIS STATEMENT)

"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*

A save, platform-independent binary target which also targets the web gives makes many interesting things now possible. 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.

[DIAGRAM: EXPLAINING CONCEPTS OF WASM]

--- *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.

--- *Problem statement*

[DIAGRAM: PROBLEM STATEMENT]

It is unclear what WebAssembly exactly means for the geospatial community. It could potentially fix many problems:

- 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 defined technically / quantitatively by means of a performance analysis, as well as practically / qualitatively, by documenting the creation of a web-based geoprocessing application, and judging its capabilities and effectiveness in relation to other geoprocessing methods.

The research into the technical effectiveness of WebAssembly will involve compiling C++ geoprocessing libraries such as CGAL & GDAL into WebAssembly, and then comparing the performance of these libraries against their compilation by other means (native / asm.js).

The research into the practical effectiveness of WebAssembly will be done by creating a case study geoprocessing environment using these wasm-compiled geoprocessing libraries. The environment will take the shape of a visual programming language, or VPL for short. This, together with the web's advantage of accessibility, will hypothetically demonstrate how WebAssembly can make complex geoprocessing more easy to distribute, access and use.

2.. RELATED WORK

{DIAGRAM: DEPENDENCY TREE OF RELATED WORKS}

This section of the thesis proposal covers how this research relates to prior research.

The execution of this research requires adequate background knowledge on:

- wasm itself
- wasm performance
- Relevant wasm based applications
- wasm's surrounding tools and compilers

In addition, since the case study application contains the creation of a VPL, it is important to relate this work to other geometry-based visual programming languages, as well as a paper which analysed the advantages and disadvantages of using a VPL as opposed to a programming language.

x.x On WebAssembly & Wasm Performance

x.x.x Website

https://webassembly.org/ (THE WEBSITE OF WASM ITSELF??)

x.x.x Bringing the Web up to Speed with WebAssembly

This is the original paper introducing WebAssembly in 2017, co-written by software engineers from the major browser vendors Mozilla, Google, Apple and Microsoft. It defines that a low-level compilation target should be save, fast, portable and compact. It continues by showing how previous attempts at low-level code on the web fail in at least one of these criteria, and that WebAssembly is the first to delivers on all of them. The chapters following this up cover the design details of the language, and the decisions which had to be made to live up to the four criteria. These details will become relevant when reasoning about why WebAssembly might be faster in one case versus another.

Chapter 6 and 7 also require special attention. Chapter 6 shows the possibilities available to a host environment for compiling, instantiating and invoking wasm binaries.

Chapter 7 : Implementation:

- validate
- execution time
- binary size

Initial benchmarks look promising large portion of benchmarks within 10%

x.x.x Not So Fast WebAssembly Paper

Paper exploring performance of WebAssembly more thorough.

Starts out positive: current benchmarks (2019) are even better than those of the original paper (2017).

BUT

Those original papers cover a type of benchmark which uses mainly scientific operations as benchmarks. Each of these operations are roughly 100 lines of code. This paper created a way to compile full, large-scale applications into WebAssembly, and proceeds to benchmark them. They found that these types of applications run significantly slower and spikier.

BUT

This might not be a problem for the scope of this research. This research will deal with the originally criticized scientific purposes anyway. If it does turn out that wasm performs significantly slower the larger the binaries are, This research might explore disecting the C++ libraries into a number of tiny wasm Binaries, one per function for example, or per .cpp file. As stated in the Wasm paper (SOURCE), it is possible to inject precompiled wasm binaries within other wasm binaries. This way, the functionalities of one library could be lazy-initialized, so only the parts that are necessairy are being compiled and used. Food for thought...

...

A telling example of the cause of the loss in speed is this:

NATIVE: C --{CLANG}-> x86-64 code

WEB C --{EMSC}-> WASM --{JIT}-> x86-64 code

- Chapter 6 is very significant

x.x On WebAssembly Applications:

x.x.x Michael Yuan — Tensorflow inference on WebAssembly

Michael Yuan — Tensorflow inference on WebAssembly

https://www.youtube.com/watch?v=poe0Z7GR8uI

This talk by Dr. Michael Yuan explains the advantages of WebAssembly for especially the utilization (inference) of trained AI models. This is relevant, since the field of AI is, like the field of geo-informatics, concerned with complex calculations and the efficient processing of large datasets. Dr. Yuan states that, while python might be a fine choice for training AI models, the actual inference / usage of those models is very inefficient using contemporary tools. Python is very slow, does not run on edge devices, and offers limited support in (web) application frameworks. A native application ís fast, but offers different challenges. A native app is tied to its specific hardware platform, cannot be orchestrated, is very sensitive to bugs or attacks, is not save since it has OS-level access, and just like python, cannot easily be integrated in web or application frameworks.

The lecturer claims that WebAssembly solves these problems because it is containerized and thus save, while at the same time being very performant. Additionally, the fact that is is a language agnostic compile target, and can be used together with many (web) applications, makes it an excellent solution to the earlier mentioned problems.

this talk further supports the claims made that geodata processing would benefit from adopting WebAssembly. At the same time, it is mainly concerned with improving server side performance, which is outside the scope of this paper.

x.x Relevant WebAssembly Tools

x.x.x Emscriptem

Emscriptem is a tool PAPERRRR

x.x.x Wasm-Pack

wasm-pack can be seen as the emscriptem equivalent, but created to serve the `Rust` programming language.

NO PAPER

x.x Relevant Geoprocessing libraries

x.x.x CGAL

(SOURCE)

x.x.x GDAL

...

x.x VPL

The last topic requiring background knowledge is the topic of visual programming languages (VPL's).

x.x.x The relevant vpl paper

x.x.x Related visual programming languages focussed on geometry:

What follows is a brief analysis of existing visual programming languages. While many more exist, such as Unity's Shader Graph, This list limits itself on vpl's meant for generating & processing geometry.

Name	Author	Availability	Source	Audience	Purpose	Link
FME	Save Software	€2,000 one time	Closed	Geoprocessing intermediaries	Geoprocessing	https://www.safe.com/fme/fme-desktop/
The graphical modeler	QGIS Contributors	Free	Open	QGIS users	Geoprocessing	https://www.safe.com/fme/fme-desktop/
Houdini	SideFX	~€1,690 p.y.	Closed	3D modellers & SFX	Procedural Modelling & Special effects	https://www.sidefx.com/
Geometry Nodes	Blender Foundation & Contributors	Free	Open	3D modellers & SFX	Procedural Modelling & Special effects	https://docs.blender.org/manual/en/latest/modeling/geometry_nodes/inde›
Grasshopper	David Rutten / McNeel	€995 one time	Closed	3D modellers & architects	Parametric Design	https://docs.blender.org/manual/en/latest/modeling/geometry_nodes/inde›

Name	Author	Availability	Source	Audience	Purpose	Link
GeoFlow	Ravi Peter	Free	Open	Geoprocessing experts	Geoprocessing: Rapid prototyping & Visualizing in between steps	https://github.com/geoflow3d/geoflow
Dynamo	Autodesk	+revit €3,330 p.y.	Semi-open	Expert Revit Users	BIM automation	https://dynamobim.org/

- Of these seven vpl's, two are focussed on procedural design (Grasshopper / Dynamo), two are focussed on modelling in the context of special effects (Blender, Houdini), and three are focused on geo-processing (FME, Graphical Mod). I would argue that while the

3.. RESEARCH QUESTIONS

3.1 Objectives

[DIAGRAM: TECHNICAL & PRACTICAL ASPECTS???]

This paper's main objective is to judge the fitness of WebAssembly for client-side 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
- Geo-processing client-side

NOT:

- web processing services or server orchestration
- WASI

4.. MOTIVATION

4.1 'higher level' questions.

The research questions chosen for this research are part of a set of larger questions. While the research will not completely answer the following questions, I believe the questions are nonetheless important to adress.

> What should the field of geomatics do with WebAssembly?
>
> - Why should the field of geomatics be interested?
> - Can we technically use it for geomatics?
> - Can we practically use if for geomatics?

This also further explains the need for the vpl application within this research. I believe it necessary to develop an application whom's existence serves as a starting point for answering the more complicated "why should we", and "practical" sub-questions.

4.2 Additional problems the software tries to solve, and features it tries to present:

additionally,

- Real-time geodata processing

- A number of use-cases exist with a growing need for real-time geodata processing. (SOURCE: INCIDENT MAPPER)
- Moving tools like CGAL closer to the final product (Web Application) can create more dynamic applications.

- Improved Geoprocessing Ergonomics

- Insightful debugging: Client-side geoprocessing together with a VPL allows direct user feedback unlike server-side geoprocessing. Users can be on top of the calculations, look at in betweens steps, reconfigure the procedure without recompilation, see the immediate effects of parameter changes.
- Improved communications: Users will be able to share demo's and procedures with a link.
- Improved accessibility: Users will not have to install anything except a web-browser. This will make geoprocessing more accessible & operational to a larger audience. It allows more people to do more things with geodata, and reach more interesting conclusions quicker.

- Just In Time / Personal Geodata

- JIT: Instead of having large, preprocessed datasets, geodata could be processed on demand from the source client-side. If a user is only interested in a small area of the source dataset, this could save vast amounts of time, storage space and computational resources.
- Personal: It also allows the end user to tailor geodata to their exact needs.

5.. METHODOLOGY

(utilize pre-work)

5.1 Software

5.2 Tests

5.3 Case Study

> *Demo Application: On Demand Triangulator + Isocurves*
>
> Input:
>
> - Point Cloud
>
> Output
>
> - Line Curves / .png render of line curves
>
> Steps:
>
> - Load ahn3 point-cloud (WFS Input Widget | WFS Preview Widget)
> - Visualize point cloud on top of base map of the netherlands (WMS Input Widget | WMS > Preview Widget)
> - Only select terrain points (list filter Operation)
> - Construct a 2d polygon by clicking points on a map (Polygon Input Widget)
> - Select Area of interest using a 2d polygon (Boundary Include Operation)
> - Triangulate point cloud with a certain resolution (Triangulate Operation)
> - Intersect the mesh surface with a series of planes (Isocurves from Mesh Operation)
> - Preview data (MultiLine Preview Widget)
> - Export data (MultiLine export Widget)

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

- WebAssembly
    - As compile target
- C++
- Typescript / Javascript
    - Front-end code
    - WebGl & javascript Canvas api
        - visualization
- Rust ????

Libraries & Tools

- Emscriptem
- Wasm-Pack
- SSVM ???
    - : WebAssembly high performant virtual machine meant for server side

Data

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