

# 1.. INTRODUCTION

---

TODO: add sources, make more readable for a general audience, make it more professional.

**I describe the main goal of the field of geomatics to be: *Give as many people as possible as much insight in their surroundings as possible.*** This is why we scan the earth, why we 'geo-process' these raw findings into more clean, meaningful forms, why we put these results in databases the size of dozens of terabytes, and why we build applications to view this data. All of it should be in the service of the general public, to give them the tools and data they need to gain meaningful information and insight into our surroundings, our Earth.

**The web plays a vital role in pursuing this goal.** The vast majority of geodata end-products are web applications. this seems to stem from the web's excellent ability to publish cross-platform, and the fact that web-apps require no installations.

**However, web-applications have their limitations.** It is usually only a visualization, without any user interactivity other than panning & zooming, toggling layers, or maybe adding annotations.

**This has its reasons..** It makes sense from the "dumb-client + smart-server" design principle, prevalent in most geodata web applications. All complex operations should be done server-side, where these tools can be central, calculated with more powerful languages such as C++, and we don't have to worry about the differences between the clients. It leads to a concrete division of labour: The server deals with processing and pre-rendering, and the clients only purpose is visualization, a window into preprocessed data.

**But, this hard divide between processing & visualization causes problems.** The "dumb-client + smart-server" design principle leaves the end-users of web applications with static, non-interactive, and downright slow tools.

- Static and non-interactive, because all data shown needs to be pre-processed and pre-rendered. All possibilities granted to end-users will have to be thought about beforehand by the creators, and the possibilities are often limited since every additional option takes up vital database storage space. This leaves the user no room for experimentation, exploration, or personalization.
- And slow, since any post-processing needed by a "dumb-client" requires additional web calls. It requires the server in question to:
  1. Be activated by the client by means of a web call.
  2. Posses of the exact same data the client is looking at. If this is not the case, it requires additional web calls to acquire this.
  3. deliver all the resulting data back to the client using even more web-calls.

All the while, the client has little to no insight in the going-ons on the server. Status updates would require yet additional web calls.

**These problems could be fixed by introducing client-side geoprocessing.** If the tools used traditionally at the server-side, like the C++ geoprocessing libraries CGAL and GDAL, could be utilized client-side, the discrepancy between visualization & processing in web-apps could be bridged. This would allow a new range of interactive, dynamic web applications, in which geodata can be post-processed quickly, uniquely, and on demand.

**This poses its own set of problems.** Normally, only the `javascript` programming language can be utilized in client-side web applications. This would mean that the aforementioned geoprocessing libraries often containing tens of thousands lines of code, would have to be rewritten in javascript, or would have to be compiled to javascript. The first option would be a time-consuming task, and would have to be repeated every time the underlying libraries change. the second option is possible. C++-based libraries such as CGAL could technically be converted to a special, fast subset of javascript called `asm.js` using the `emscripten` compiler. Even this, however, is often time consuming. The rather large javascript files take a long time to download, to scan, and to be properly optimized by a javascript Just In Time (JIT) Compiler.

**A recent, emergent technology poses a third option: WebAssembly.** WebAssembly, shortend as `wasm`, is a compilation target meant to be fast, save, and platform & source independent (SOURCE). It performs better than `asm.js` in almost all aspects: it loads quicker, it is scanned quicker, and since it is far closer to bytecode than javascript, it can perform at a speed comparable to its native counterpart.

This development means that technically, there is not much preventing a client-side application to be almost just as powerful as a server-side application.

The question remains:

- How well would WebAssembly enable client-side Geoprocessing? and
- What would a web-application equipped with client-side geoprocessing look like? How could it be used in practice?

This paper

This research will explore the possibility of WebAssembly to answer the first question, and poses an experiment to explore the second question. The experiment is roughly this: Instead of minimizing the responsibilities of a web-client application, what happens if we do the reverse and maximize the responsibilities?

(write a bit about a vpl)

## 2.. BACKGROUND & 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 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

### x.x On Native tools with web-publishing support

Why many geodata processing tools such as FME and QGIS are integrating ways of web-publishing.

This research proposes to reverse this reasoning. Instead of giving native applications tools to publish, it proposes to grant the most common distribution destination, the web, the tools needed to perform geodata processing.

### x.x On Web Processing Services

This is covered briefly in the introduction, but Web Processing Services ....

Very applicable to big-data

but even so, even server side processing would benefit from the speed, security and containerization WebAssembly poses over workflows which utilize a combination of python and C++. This, sadly, is outside the scope of this research.

### 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 dissecting 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 necessary 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 is 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 it 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 On Interactive Web Applications

### x.x.x VAT: A Scientific Toolbox for Interactive Geodata Exploration

## x.x Relevant WebAssembly Tools

### x.x.x Emscripten

Emscripten is a tool PAPERRRR

### x.x.x Wasm-Pack

wasm-pack can be seen as the emscripten 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	<a href="https://www.safe.com/fme/fme-desktop/">https://www.safe.com/fme/fme-desktop/</a>
The graphical modeler	QGIS Contributors	Free	Open	QGIS users	Geoprocessing	<a href="https://www.safe.com/fme/fme-desktop/">https://www.safe.com/fme/fme-desktop/</a>
Houdini	SideFX	~€1,690 p.y.	Closed	3D modellers & SFX	Procedural Modelling & Special effects	<a href="https://www.sidefx.com/">https://www.sidefx.com/</a>
Geometry Nodes	Blender Foundation & Contributors	Free	Open	3D modellers & SFX	Procedural Modelling & Special effects	<a href="https://docs.blender.org/manual/en/latest/modeling/geometry_nodes/index">https://docs.blender.org/manual/en/latest/modeling/geometry_nodes/index</a>
Grasshopper	David Rutten / McNeel	€995 one time	Closed	3D modellers & architects	Parametric Design	<a href="https://docs.blender.org/manual/en/latest/modeling/geometry_nodes/index">https://docs.blender.org/manual/en/latest/modeling/geometry_nodes/index</a>
GeoFlow	Ravi Peter	Free	Open	Geoprocessing experts	Geoprocessing: Rapid prototyping & Visualizing in between steps	<a href="https://github.com/geoflow3d/geoflow">https://github.com/geoflow3d/geoflow</a>
Dynamo	Autodesk	+revit €3,330 p.y.	Semi-open	Expert Revit Users	BIM automation	<a href="https://dynamobim.org/">https://dynamobim.org/</a>

- 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 these goals differ, all of these vpls have a lot in common. All of them have some representation of vectors, points, line segments, polygons, surfaces and solids, in one way or another.
- huge differences in availability. If they are not free (QGIS / Blender / Geoflow), these vpls are extremely expensive. this availability roughly correlates to the open / closed source nature of the packages.

## 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 it 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 between 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 + jupyter, 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

- Emscripten
- 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