1.. INTRODUCTION

> TODO: add sources, make it more professional.

> TODO: add more 'timely' comments like "The last decade has seen a rapid evolution of processing, analysis and visualization of freely available geographic data using Open Source Web-GIS. "

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 (I NEED A SOURCE). this seems to stem from the web's excellent ability to publish cross-platform, and the fact that web-apps require no installation besides the browser.

However, geo web applications are functionally very limited, compared to native geo applications. They are mostly used as data visualizers. User interactivity is limited to moving the camera, toggling layers, adding annotations.

This is explainable from the "thin client fat server" design principle, prevalent in most geodata web applications (SOURCE: GEOWEB). This paradigm states that complex operations should be done server-side, where these tools can be central, where calculations can use more powerful languages such as C++, and where the differences between low-end and high-end clients can be mitigated. It leads to a concrete division of labour: The server deals with geoprocessing and pre-rendering, and the clients only purpose is visualization, a window into preprocessed data.

Still, this hard divide between processing & visualization causes problems. The thin-client web applications users are left with are static, non-interactive, and slow tools.

- Static and non-interactive, because post-processing in a thin client is not possible. this means that all options presented need to be pre-processed, pre-rendered, and stored in a server-side database. All possibilities granted to end-users will have to be thought about beforehand by the creators of the tool, and these possibilities are often limited since every additional option takes up vital database storage space. This leaves the user little room for experimentation, exploration, or personalization.
- And slow, since any type of interaction between a thin client and corresponding thick server requires many steps:
    1. The server must be activated by the client by means of a web call.
    2. It must 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 data.
    3. The server can then perform the desired function. While this is happening, the client often has no insight in the progress of the server. Status updates can be given, but would yet again require more web traffic, especially when containing visualizations.
    4. After this is done, The server has to deliver all the resulting data back to the client using even more web-calls.

These problems could be mitigated by introducing client-side geoprocessing. If the tools used traditionally at the server-side, like the mature 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.

Insight is more than just visualizing data. I pose that interaction and dynamic experimentation with geodata leads to a higher level of understanding into our geospatial surroundings. And, if the overarching goal of geomatics is to improve and **share** geospatial insight, then client-side geoprocessing is a worthwhile pursuit.

However, client-side geoprocessing 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 also a possibility. C++-based libraries such as CGAL can be converted to a special, fast subset of javascript called `asm.js` using the `emscripten` compiler [SOURCE: emscriptem]. This, however, can result in inefficiencies. The rather large javascript files usually take a long time to download, to scan, and to be properly optimized by a javascript Just In Time (JIT) Compiler [SOURCE: wasm].

A recent, emergent technology poses a third option. WebAssembly, shortened as wasm, is a binary compilation target meant to be small, fast, save, and platform & source independent [SOURCE: wasm].

It outperforms `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 often perform at a speed comparable to its native counterpart [SOURCE: wasm, not-so-fast].

This development means that theoretically, there is not much preventing a wasm-powered client-side application to be almost as powerful as a server-side application. The question remains, however, if this is also practically the case. Several practical uncertainties remain, such as:

- Do geoprocessing libraries directly compile into WebAssembly? If not, which workarounds are needed?
- Will WebAssembly-compiled geoprocessing libraries (`wagl`'s), load efficiently, or should they be split up into parts, and loaded lazily?
- Can the tools offered by `wagls` be directly used like functions? Or do they require special services, such as a virtual file system?
- How well do `wagl` operations perform in a browser, compared to their native counterparts? What can be done to make this difference as small as possible?

Performing geoprocessing in a browser seriously transforms the nature of a geo-web application. This raises another set of questions, of a sightly different nature:

- Instead of minimizing the responsibilities of a web-client application, what happens if we do the reverse and maximize the responsibilities?

- What would a thick client look like?
- and what should a thick client be able to do?
- What should the user experience be?

Since no wasm-powered, client-side **geoprocessing** applications exist yet, there is no way to directly answer these questions, and no way to confirm the theoretical benefits of WebAssembly for the geospatial community.

This Study

The aim of this study is to provide an environment meant for client-side geoprocessing using WebAssembly. This environment will be used to demonstrate if and how wasm-based, client-side geoprocessing is possible. At the same time, by presenting this environment, the study aims to explore the design possibilities of a web-GIS application equipped with such tools.

This will require research into the technical effectiveness of WebAssembly. C++ geoprocessing libraries such as CGAL & GDAL will be tested on their ability to be compiled, loaded, and used from a browser. This is compared against their compilation by other means, such as native binaries or the aforementioned `asm.js`. This research is complemented by an extensive 'case study' to explore the design possibilities of a web-application equipped with client-side geoprocessing. A thick-client web application will be created, and this will serve as platform for testing the aforementioned `wagl`'s. The tool and can be additionally used for acquiring, visualizing, and saving geodata.

(WebAssembly geoprocessing libraries -> `wagl`'s)

Limits & Future Work

Client-side WebAssembly Only

This study will limit itself to the **client-side** usage of WebAssembly. A powerful case can be made for **server-side** usage of WebAssembly, especially in conjunction with a language like Rust. WebAssembly + Rust could, compared to using python, java or C++, make geoprocessing more maintainable and reliable, while at the same time ensuring memory safety, security, and performance [SOURCE: wasi, wasm-ai]. Server-side wasm is beyond the scope of this paper, but would be an excellent starting point for future work.

Note that this also means that research into `wagl`'s is important for more than just client-side geoprocessing. All geoprocessing could benefit from it.

No Surveys

Additionally, a survey analyzing how users experience client-side geoprocessing in comparison to native geoprocessing must also be left to subsequent research. While this would gain us tremendous insight, client-side geoprocessing is insufficiently researched and developed to make a balanced comparison. Native environments like GRASSGIS, QGIS, FME or Esri simply have a 20+ year lead. This paper seeks to first close this gap, limiting itself to overcoming the technical and design boundaries in the pursuit of practical client-side geoprocessing. It seeks to present client-side geoprocessing as **an** option. Afterwards, Future research will have to be done to discover if this is **the** option (for a particular use case).

2.. BACKGROUND & RELATED WORK

x.x On Client-side vs server-side geoprocessing

x.x.x 2014 Client-side versus Server-side Geoprocessing ...

These results demonstrated that the current implementation of web browsers are limited in their ability to execute JavaScript geoprocessing and not yet prepared to process data sizes larger than about 7,000 to 10,000 vertices before either prompting an unresponsive script warning in the browser or potentially losing the interest of the user.

This paper is very similar to what i'm doing, and it makes a conclusion that scared me at first glance. Then I saw that this is a paper out of 2014.

The results of this paper are insightful, but do not directly applicable to this paper because of three reasons:

1. The paper used javascript-based geoprocessing, not `asm.js` optimized. This is known to be inefficient.
2. The paper stems from 2014. is before an incredible industry-wide performance increase of javascript interpreters. This is the result of technological development in the form of an arms race between the major browswer vendors.
3. This paper will introduce WebAssembly to speed things up.

x.x.x Hybrid Geoprocessing Web Services

This paper proposes a hybrid strategy, using the OGC Web Processing services as a starting point, and building client-side tools around it. This is different from the approach offered by this study, which starts from the well-known CGAL and GDAL geoprocessing libraries. The environment proposed by this thesis might offer OGC Web Processing services, inspired by this paper.

x.x.x Analysis of server-side and client-side Web-GIS data processing methods on the example of JTS and JSTS using open data from OSM and geoportal

This is a very relevant source

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 of a program:

WASM web apps: If an application is published on the web without login requirements, makes it so 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 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 On Interactive Web Applications

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

x.x On advanced 3d web applications

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 modeller	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›

Name	Author	Availability	Source	Audience	Purpose	Link
Grasshopper	David Rutten / McNeel	€995 one time	Closed	3D modellers & architects	Parametric Design	https://docs.blender.org/manual/en/latest/modeling/geometry_nodes/inde›
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 Modeller). 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.

Their similarities end there. huge differences exist between them:

- 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.
- differences in "usefulness". ... ...

3.. RESEARCH QUESTIONS

> Do this again, this is old

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?

Some question I would like to answer:

- Is there a difference between compiling the full CGAL / GDAL library into one binary and loading this in one go, versus compiling parts of it into several smaller binaries, which can be downloaded and loaded lazely?

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 Design Case Study

A Visual Programming Language (VPL) will be created

5.3 Use 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