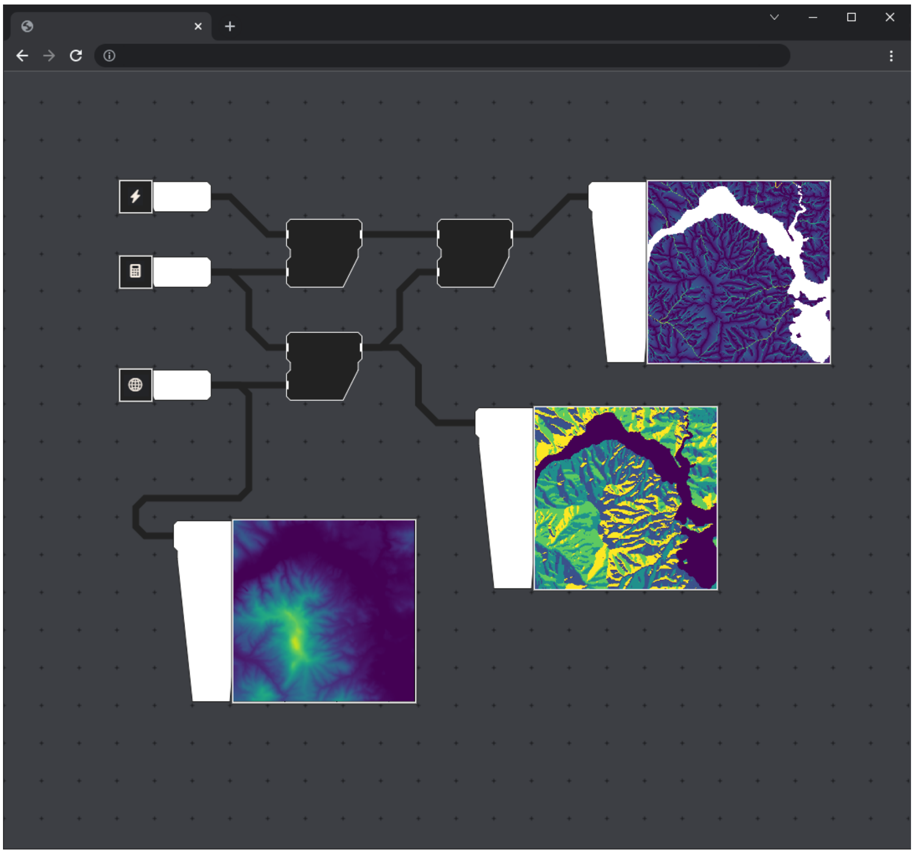


Thesis Proposal: Accessible geoprocessing in a browser using WebAssembly and Visual Programming

Jos Feenstra
MSc Geomatics | TU Delft
#4465768

1st supervisor: Stelios Vitalis
2nd supervisor: Ken Arroyo Ohori

January, 2022

Key words: Geomatics, WebAssembly, 3D geometry, Geo processing, Client side, Visual programming,

Acronyms

CSG client-side geoprocessing

DAG Directed Acyclic Graph

geoprocessing Geospatial data processing

GIS Geographic Information Systems

GUI Graphical User Interface

UI User Interface

UX User Experience

VPL Visual Programming Language

W3C World Wide Web Consortium

Wagl WebAssembly-compiled geoprocessing library

Wasm WebAssembly

WFS Web Feature Service

WMS Web Mapping Service

WPS Web Processing Service

Contents

1 Introduction

1.1 Motivation

Interactive, browser-based Geographic Information Systems (GIS) form an indispensable component of the modern geospatial software landscape. For the common person, a web application is often their first and only exposure to a GIS, be it a web mapping service, a navigation system, or a pandemic outbreak dashboard. A web application is cross-platform by nature, and offers ease of maintainability and access, since no installment or app-store interaction is required to run or update the app. The ability to be shared with a link, or to be embedded within the larger context of a webpage is also not trivial. These aspects have made the browser a popular host for many geographical applications, especially when targeting end-users.

Despite the popularity of geographical web applications, the range of actual GIS abilities these applications are capable of is very limited. Geospatial data processing (geoprocessing) abilities, like CRS translations, interpolation or boolean operators, are usually not present within the same software environment as the web app. Consequently, current geospatial web applications serve for the most part as not much more than viewers; visualizers of pre-processed data.

This limited range of capabilities inhibits the number of users and use cases geographical web applications can serve, and with it the usefulness of web GIS as a whole. If web applications gain geoprocessing capabilities, they could grow to be just as diverse and useful as desktop GIS applications, with the added benefits of being a web application. It would allow for a new range of highly accessible and sharable geoprocessing and analysis tools, which end-users could use to post-process and analyze geodata quickly, uniquely, and on demand.

This is why geoprocessing within a web application, also known as client-side geoprocessing (CSG), is slowly gaining traction during the last decade [Kulawiak et al., 2019,Panidi et al., 2015, Hamilton, 2014]. Interactive geospatial data manipulation and online geospatial data processing techniques are described as "current highly valuable trends in evolution of the Web mapping and Web GIS" [Panidi et al., 2015]. But this also raises the question: *Why is geoprocessing within a web application as of today still nowhere to be found?*

By studying the prior research mentioned, three obstacles are identified, hindering a smooth, widespread adoption of CSG:

- 1: CSG is technically challenging.
- 2: CSG is immature.
- 3: CSG can be considered unnecessary.

The study proposed by this paper seeks the advancement of web GIS & client-side geoprocessing by attempting to overcome these obstacles. It will do this by researching possible solutions to key components of all three of them. However, we must first regard each obstacle more closely, so that the significance of these key components can be made clear.

1.2 Obstacles

Obstacle 1: CSG is technically challenging

During the prior studies on client-side geoprocessing, serious technical drawbacks where encountered [Panidi et al., 2015, Hamilton, 2014]. Browser-based geoprocessing suffers from the fact that it will need to be written in the JavaScript programming language. Previous attempts at client-side geoprocessing have shown that JavaScript based geoprocessing libraries do not offer the performance required for proper geodata processing [Hamilton, 2014]. Moreover, the JavaScript library ecosystem does not offer viable alternatives to industry-standard libraries like CGAL and GDAL.

An emergent technology poses a possible solution to both problems. At the end of 2019, the World Wide Web Consortium (W3C) officially pronounced WebAssembly as the fourth programming language of the web [w3c, 2019]. Since then, all major browsers have added official WebAssembly support. WebAssembly (Wasm) is a binary compilation target meant to be small, fast, containerized, and platform & source independent [Haas et al., 2017]. Wasm surpasses javascript in almost all performance aspects: it loads quicker, it is scanned quicker, and since it is close to bytecode, it can often run at a speed comparable to native system bytecode [Jangda et al., 2019].

While these features sound promising, this does not automatically mean this obstacle can be overcome. WebAssembly brings along many practical uncertainties such as:

- Do geoprocessing libraries directly compile into WebAssembly? If not, which workarounds are needed?
- Will a WebAssembly-compiled geoprocessing library (Wagl) load efficiently, or should they be split up into parts, and loaded lazily?
- 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?

Obstacle 1 remains in place as long as these uncertainties remain unanswered.

Obstacle 2: CSG is immature

Secondly, geoprocessing within a web browser is currently too novel. That is to say, no supporting software exists for it, and the way geoprocessing is supposed to be performed within the context of a web-browser remains unknown. Obstacle 1's list of implementation considerations cover only how Wagl's can be *run*. Potential answers do not indicate how Wagl's can be *used*. Jangda et al. warn against assessing WebAssembly in a vacuum, and notes how its performance is highly dependent on the way it is used, and the context in which it is used. This context of geoprocessing within a web-browser brings up many considerations as well:

- How will users upload or specify the input in a web browser?
- Can the transformations offered by Wagl's be used like functions? Or do they require special services, such as a wrapper library, virtual file system, or a virtual command line interface?
- How can a sequence of geoprocessing steps be chained together in a web-browser?
- How can users recieve and evaluate the output in a web-browser?
- How can a browser-based User Interface (UI) or Graphical User Interface (GUI) facilitate all these functionalities?

Obstacle 2 can be overcome if these questions recieve a satisfying answer.

Obstacle 3: CSG can be considered unnecessary

The third obstacle is presented by the older counterpart of client-side geoprocessing: server-side geoprocessing. server-side geoprocessing seems to offer a solution to the problem of client-side geoprocessing, and because of this, CSG can be considered unnecessary.

The OGC standard of the Web Processing Service (WPS) offers a set of protocols to standardize geoprocessing on a server. By specifying input data and instructions to one of these services, polling the status of the process, and then downloading the results once finished, a user can process geodata on a server. If a client-side application creates an interface to use such a service, it can offer geoprocessing to a client.

This can be preferable. A Web Processing Service is an excellent solution when client-side hardware is limited, and when server-side resources are abundant. It is also more efficient if the datasets required for geoprocessing are already located on these servers, and when working with datasets too large to store locally.

However, this does not mean server-side processing is in all cases preferable over client-side geoprocessing. the choice of which one is 'preferred' is situational, and this cuts both ways. Using a server for geoprocessing will make a client application not self-contained, but distributed and dependent. It leads to a hard divide between visualization and processing, easily disrupted by connectivity failure, or changes within either the client or server. Such a system also fails to take advantage of client-side hardware improvements, and servers performing geoprocessing calculations could lead to high operational costs.

The choice between client or server also does not need to be binary. For example, hybrid strategies have already been theorized and (partially) implemented [Panidi et al., 2015].

Nonetheless, developers implementing client-side geoprocessing must be conscious of this situational nature. Obstacle 3 can be overcome if client-side geoprocessing can demonstrate clear advantages over server-side geoprocessing in specific situations.

1.3 Problem Statement

From these three obstacles, the following problem can be summarized: Client-side geoprocessing as a concept is not being adopted by geoweb applications, because it is considered technically impractical, immature, and unnecessary. Prior studies into CSG show its potential, and if WebAssembly performs as described by Haas et al, it could theoretically be the missing link to make client-side geoprocessing viable. However, these technologies are underresearched due to their novelty, and as a consequence, almost no existing libraries or applications exist using CSG. Practical implementation details are absent, and guidance regarding CSG is lacking.

Instead of focussing on only one of these obstacles, this study recognizes the causal and interdependent nature of all three problems: The absence of existing client-side geoprocessing applications makes it so the theoretical benefits of WebAssembly cannot be tested contextually, and this in turn offers us no way to examine the theoretical benefits of client-side geoprocessing over server-side geoprocessing. Taken together, these three obstacles form a barrier preventing web GIS applications of adopting client-side geoprocessing.

1.4 This Study

Therefore, This study will make a new, wholistic attempt at actualizing client-side geoprocessing. It will differ from previous attempts by adjusting its methodology to tackle key components of all three obstacles, in order to enable the widespread adoption of CSG. The first obstacle of CSG being impractical will be addressed by researching how well WebAssembly can host existing, industry-standard geoprocessing libraries. Both the ability of existing geoprocessing libraries to be compiled to WebAssembly will be researched, as well as the performance of the resulting binaries. The second obstacle of CSG being novel will be addressed by developing a use case application to assist this research. This application serves to contextualize the research, and to force the research to solve the practical inhibitions of how CSG can actually be used. This use case will also be used to address the third obstacle of CSG being regarded as unnecessary. The use case will be a unique type of application, one which would be highly impractical without client-side geoprocessing. If successful, the application can demonstrate the advantages of CSG over server-side equivalents, as well as offer guidance to other applications seeking the benefits of CSG.

1.5 Use Case

The use case application for this study is the so called "GeoFront" environment. GeoFront will be developed as a web-based GIS environment, offering users the ability to load, process, and then visualize or save various types of geodata. The entire application will run client-side in a browser, and uses a visual programming language as its primary GUI. For input, the environment offers Web Mapping Service (WMS) and Web Feature Service (WFS) support, as well as ways for users to load local geodata. For processing, geofront offers geoprocessing functions provided by GDAL and CGAL. Being a visual programming language, GeoFront can be used to chain multiple processing steps together, while still being able to retrieve in-between products. For output, the environment can be used to either save data to the user's local machine, or to visualize the results within the geofront application using a WebGL based viewport.

The choice for a Visual Programming Language (VPL) is made to demonstrate the advantage of client-side geoprocessing, and thus overcome the third obstacle. Using visual programming & CSG, the geoprocessing sequence can be altered on the fly, and in-between products can be inspected quickly. This way, a user can easily experiment with different methodologies and parameters which, hypothetically, improves the quality of the processed geodata. Additionally, a VPL forms a balance between a programming language and a full GUI, making the tool accessible to both programmers and non-programmers alike.

1.6 Research Questions

This study intends to discover if contemporary web technologies can facilitate a full-scale client-side geoprocessing tool, by seeking an answer to the following question:

How to ***design and create*** *a browser-based GIS environment which can* ***effectively utilize existing geoprocessing libraries****, using only the* ***current state*** *of* ***standard client-side web technologies****?*

Explanation

- **design and create**: The wording 'design and create' is used to signal that this will consider the theoretical design , as well as the practicalities of creating this design.

- **Standard client-side web technologies**: This phrase is meant to limit the scope to only the standard, core technologies of major browsers (Chrome, Edge, Safari, Firefox). This means the four languages Wasm, CSS, JavaScript and HTML. Additionally, HTML5 gives us WebGl, the 2d Canvas API, SVG's, and Web Components to work with.

- **Current state**: The study will use contemporary, even bleeding edge features of the modern web, but its findings will nonetheless be bound to this time of writing, as web technologies in particular quickly change.

- **Existing geoprocessing libraries**. This wording expresses this studies desire to explore the usage of existing geoprocessing libraries, rather than to recreate geoprocessing libraries from scratch.

- **Effectively utilize**: The study intends to not only find out how Wagl's can be *run* in a browser, but also how Wagl's can be *used*. 'Effective' is used in the wholistic sense, since a balance will have to be found between several aspects such as load-time, run-time, and less concrete usability aspects.

Sub Questions

The following sub-research questions are needed in order to answer the main question. The methodology chapter will explain the choices of these sub-questions.

1 : What is the most fitting methodology of compiling C++ geoprocessing libraries to WebAssembly?

2 : How to design and create a client-side geoprocessing environment for data-users?

3 : How can wasm-compiled geoprocessing libraries be distributed and used in a client-side geoprocessing environment?

4 : What are the advantages and disadvantages of GIS applications created using a client-side geoprocessing environment powered by WebAssembly?

Assessment

At the final conclusion of the proposed thesis, The Thesis will answer if the designed and created GIS environment can indeed effectively utilize these geo-libraries. This will be answered by quantitative and qualitative means:

Quality

- Have all design goals been met?
- Can data users 'effectively' handle input, process and output?
- Can the load & run times be regarded as acceptable to use?

Quantity

- Have all required features been implemented?
- Which libraries can / could be used?
- What are the load & run times of these libraries, compared to native execution?

1.7 Scope

Will Include

The 'will include' scope is represented by the Methodology chapter.

Will not include

Server-side or Native WebAssembly

This study will limit itself to the *client-side* usage of WebAssembly. A powerful case can be made for *server-side*, or native level usage of WebAssembly, especially in conjunction with a programming language such as Rust. Rust compiled to WebAssembly 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 [Clack, 2019].

Server-side or native 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 WebAssembly is important for more than just client-side geoprocessing. All geoprocessing could benefit from it.

Web Processing Service

Similarly, this study will exclude the OGC standard of the WPS [OGC, 2015], since these services do not offer *client* side geoprocessing, but instead offer *server* side geoprocessing. A client-side application *could* create an interface to use such a service, to essentially offer geoprocessing to clients, but this study regards a solution like that as a workaround, not a true solution to the problem of client-side geoprocessing.

This is not to say that client-side geoprocessing replaces the need for WPS. future work could research the possibility of utilizing a hybrid strategy of both client-side and server-side geoprocessing, following in the footsteps of [Panidi et al., 2015].

Usability Analysis

While usability is a major component of this research, no claims will be made that the developed use-case is more usable to native GIS applications or geoprocessing methods. This research attempts to solve practical inhibitions in order to discover whether or not client-side is *a* usable option. If it turns out that this method is viable technically, future research will be needed to definitively proof *how* usable it is compared to all other existing methods.

Similarly, 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 a tremendous amount of insight, client-side geoprocessing is too new to make a balanced comparison. Native environments like GRASSGIS, QGIS, FME or ArcGIS simply have a twenty year lead in research and development.

2 Related work

The introduction already touched upon a number of prior works on the topic of CSG as well as Wasm. However, we are required to examine these studies more closely as well as introduce others, in order to further define our methodology. This study's methodology will draw from prior studies on three topics: The topic of the Geoweb, client-side geoprocessing, and WebAssembly.

2.1 Related works on the Geoweb

The main research question speaks of 'utilization'. Utilization or 'Usability' can mean different things depending on the context used, and which user this applies to. We will define both the type of *usability* and *user* this study speaks of by studying the larger context this study is part of: the Geoweb.

The Geoweb, or Geospatial Web, covers a broad collection of topics located at intersection of the field of geo-information and the web. A noteworthy study on the Geoweb is Van den Brink's phd titled "Geospatial Data on the Web". [Brink, 2018]. She claims that even though geodata is vital to a diverse range of applications and people, the ability to properly retrieve geodata remains almost exclusive to experts in the field. This is to the determent of all these applications and people, jeopardizing value, opportunity, and decision making. She makes this argument by using the concept of FAIR geodata. Coined by [Mark D Wilkinson et al., 2016], The FAIR principles are a collection of four assessment criteria used to judge the usability of (scientific) data: Findable, Accessible, Interoperable, and Reusable.

We argue that if these concerns count for geodata *retrieval*, they should just as well count for geodata *processing*. After all, if a user is unable to convert the retrieved geodata to their particular use case, then the information they seek remains inaccessible. Therefore, this study introduces the concept of *FAIR geoprocessing*.

Based on the arguments presented by [Brink, 2018], we can also extrapolate that a GIS environment shouldn't exclusively be used by only experts. Van den Brink mentions a group called 'data users', presented as "web developers, data journalists etc. who use different kinds of data, including geospatial data, directly to create applications or visualizations that supply information to end users (citizens)".

We use both extrapolations to define the users and 'usability' for the context of this study. We will judge the proposed use case application as 'usable', if it is deemed Findable, Accessible, Interoperable, and Reusable. The user group intended to use this environment is defined as both experts in the field of geo-information and this more general group of data users.

2.2 Related works on client-side geoprocessing

This study will be the latest in a series of studies concerned with client-side geoprocessing. As such, it is important to relate to previous attempts and efforts, to learn from their findings, and to make sure the exact same research will not be performed twice.

As a side note, client-side geoprocessing is not to be confused with native geoprocessing clients, which would include applications like QGIS [Community, 2022]or ArcGIS [Esri, 2022].

Client-side, JavaScript based geoprocessing has seen some level academic interest throughout the last decade. The papers [Hamilton, 2014,Panidi et al., 2015,Kulawiak et al., 2019] all speak of an emergent trend of thick-client websites. Proponents of this trend argue that for certain applications, end users can benefit from dynamic, interactive websites which can immediately respond to a users input, rather than waiting on server round-trips necessary on static web-pages.

And by still being a webpage rather than a native application, users can access these applications without installment. The trend is made possible because of hardware improvements of client devices, as well as software improvements, such as HTML5.

The aforementioned papers each try to apply this trend to the field of geo-informatics. Hamilton et. al. created a such a 'thick-client', capable of replacing certain elements of server-side geoprocessing [Hamilton, 2014]. However, the results are unfavorable towards JavaScript. The paper states how "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.". While these findings are insightful, they are not directly applicable to the efforts of this study proposal. Three reasons for this:

- The paper stems from 2014. Since then, web browsers have seen a significant increase in performance thanks to advancements in JavaScript JIT compilers [Haas et al., 2017, Kulawiak et al., 2019].
- The paper does not use compile-time optimizations. WebAssembly wasn't invented yet, but the authors could have utilized 'asm.js' [Mozilla, 2013] which did exist at the time.
- The paper uses a javascript library which was never designed to handle large datasets.

The same statements can be made about similar efforts of Panidi et. al. [Panidi et al., 2015]. However, Panidi et. al. never proposed client-side geoprocessing as a replacement of server-side geoprocessing, and spend no time on comparing the performance of the two. Instead, the authors propose a hybrid approach, combining the advantages of server-side and client-side geoprocessing. They also present the interesting observation that client-side versus server-side geoprocessing shouldn't necessarily be a compassion of performance. "User convenience" as they put it, might dictate the usage of client-side geoprocessing in certain situations, despite speed considerations [Panidi et al., 2015].

This concern the general web community would label as User Experience (UX), is shared by a more recent paper [Kulawiak et al., 2019]. Their article examines the current state of web-technologies from the point of view of developing cost-effective Web-GIS applications for companies and institutions. Their research reaches a conclusion favorable towards client-side data processing: "[Client-side data processing], in particular, shows new opportunities for cost optimization of Web-GIS development and deployment. The introduction of HTML5 technologies has permitted for construction of platform-independent thick clients which offer data processing performance which under the right circumstances may be close to that of server-side solutions. In this context, institutions [...] should consider implementing Web-GIS with client-side data processing, which could result in cost savings without negative impacts on the user experience.".

From these papers we can conclude a true academic and even commercial interest in client-side geoprocessing in the last decade. However, researchers quickly encounter problems during practical implementations in the past. This might not hold up thanks to the recent developments in browser technology, but these papers still show how small, practical implementation details can relate to considerable changes in UX.

Additionally, to the best of the authors's knowledge, all papers concerned with browser-based geoprocessing either tried to use existing JavaScript libraries, or tried to write their own WebAssembly / JavaScript libraries. No studies have been performed on the topic of compiling existing C++ geoprocessing libraries to the web.

2.3 Related works on WebAssembly

From all client-side web technologies, WebAssembly is one of the newest, and is most likely to be a deciding factor of this study. This requires us to be aware of the state of WebAssembly, its performance considerations, and how this relates to geoprocessing performance. But first, an introduction on the compilation target itself is in order.

The original paper on WebAssembly was published on June 14, 2017 [Haas et al., 2017]. The authors write that the reason behind the creation of WebAssembly is the observation that certain web applications started using JavaScript as a compile target, using a high-performance subset of JavaScript called 'asm.js' [Mozilla, 2013]. However, JavaScript remains a high-level, highly abstract programming language, which never intended to be used as a compile target. The discrepancy between intended use and actual use led to many complications for developers using JavaScript this way, but also for the developers of JavaScript itself [Haas et al., 2017]. In order to relieve javascript of the responsibility of being a 'low-level' compilation target, developers of the four major browser vendors Mozilla, Google, Apple and Microsoft eventually joined up, and created WebAssembly and its corresponding paper.

Performance

The initial performance benchmarks look promising. The majority of performance comparisons show that WebAssembly only takes 10% longer than the native binary it was compared to [Haas et al., 2017]. A later study confirms this by reproducing these benchmarks [Jangda et al., 2019]. It even notices that improvements have been made in the two years between the studies. However, Jangda et. al. criticize the methodology of these benchmarks, stating that only scientific operations where benchmarked, each containing only 100 lines of code. The paper then continues to show WebAssembly is much more inefficient and inconsistent when it comes to larger applications which use IO operations and contain less-optimized code. These applications turn out to be up to twice as slow compared to native, according to their own, custom benchmarks. Jangda et. al. reason that some of this performance difference will disappear the more mature WebAssembly becomes, but state that WebAssembly has some unavoidable performance penalties as well. One of these penalties is the extra translation step, shown in Figure 1, which is indeed unavoidable when utilizing an in-between compilation target.

Even though this proposed study falls in the category of scientific computation, these performance considerations will still have to be taken into account. The most important conclusion to to take away from prior research on WebAssembly is that Wasm must not be regarded as a 'drop-in replacement', as [Melch, 2019] puts it. Just like any language, WebAssembly has strengths and weaknesses. While Wasm is designed to be as unassumptious and unopinionated about its source language as possible, the implementations of host environments do favor certain programming patterns and data structures over others, and this will have to be taken into account during the proposed study.

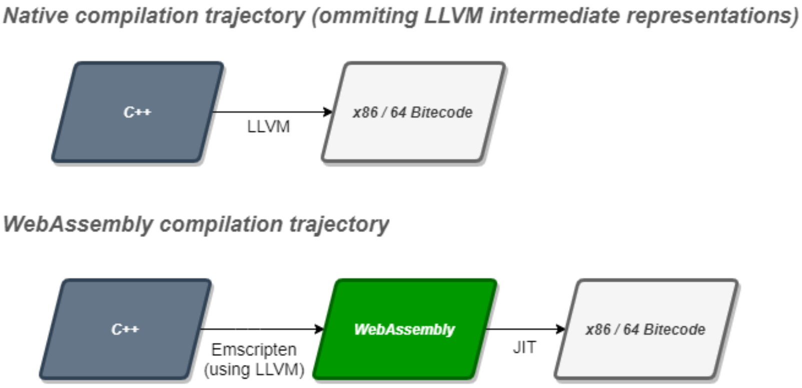

Native compilation trajectory (ommiting LLVM intermediate representations)
C++
LLVM
x86 / 64 Bitecode
WebAssembly compilation trajectory
C++
Emscripten
(using LLVM)
WebAssembly
JIT
x86 / 64 Bitecode

Figure 1: *Comparison of compilation trajectories*

2.4 Conclusion

Based on the studies on WebAssembly, we can conclude that the compilation peculiarities of WebAssembly have to be taken into account, as it cannot be regarded as a 'drop in replacement'. There is also a significant difference between using WebAssembly theoretically, and using it realistically. The studies on Client-side geoprocessing tell us that these implementation details can have vast consequences on user experience, and studies on the Geoweb express that this user experience is vital to FAIR, cross-community geoprocessing.

What this means for the methodology, is that a significant portion of this study's attention will have to go to experimenting with different ways of compiling to WebAssembly, while making sure it can still be used in a realistic scenario. If it turns out that the use-case app can only be used by experienced end-users who take special Wasm considerations in mind, a big reason of using the web, namely its accessibility, would be lost.

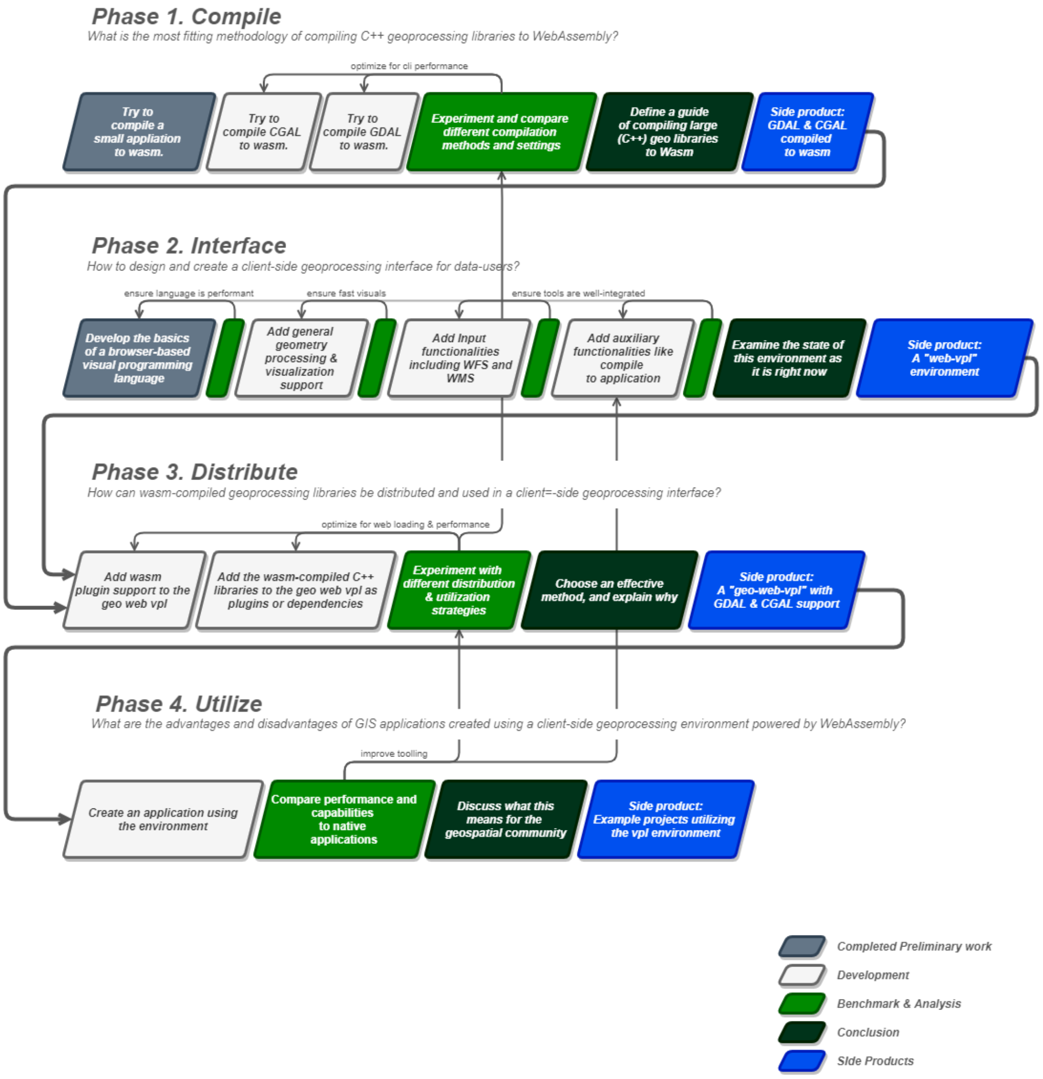

Phase 1. Compile
What is the most fitting methodology of compiling C++ geoprocessing libraries to WebAssembly?
optimize for cli performance
Try to compile a small appliation to wasm.
Try to compile CGAL to wasm.
Try to compile GDAL to wasm.
Experiment and compare different compilation methods and settings
Define a guide of compiling large (C++) geo libraries to Wasm
Side product: GDAL & CGAL compiled to wasm
Phase 2. Interface
How to design and create a client-side geoprocessing interface for data-users?
ensure language is performant
ensure fast visuals
ensure tools are well-integrated
Develop the basics of a browser-based visual programming language
Add general geometry processing & visualization support
Add Input functionalities including WFS and WMS
Add auxiliary functionalities like compile to application
Examine the state of this environment as it is right now
Side product: A "web-vpl" environment
Phase 3. Distribute
How can wasm-compiled geoprocessing libraries be distributed and used in a client=-side geoprocessing interface?
optimize for web loading & performance
Add wasm plugin support to the geo web vpl
Add the wasm-compiled C++ libraries to the geo web vpl as plugins or dependencies
Experiment with different distribution & utilization strategies
Choose an effective method, and explain why
Side product: A "geo-web-vpl" with GDAL & CGAL support
Phase 4. Utilize
What are the advantages and disadvantages of GIS applications created using a client-side geoprocessing environment powered by WebAssembly?
improve toolling
Create an application using the environment
Compare performance and capabilities to native applications
Discuss what this means for the geospatial community
Side product: Example projects utilizing the vpl environment
Completed Preliminary work
Development
Benchmark & Analysis
Conclusion
SIde Products

Figure 2: *Methodology schema*

3 Methodology

This brings us to the methodology of this study. To find answers to the main research question, this study will be divided into four phases, each matching a certain sub-research question. A phase has two goals. The primary goal is to answer its corresponding research question, and to cover the process towards this answer. The secondary goal is to describe the development of the software required to form this answer. This software can be seen as a "side product", and will be a required starting point to answer subsequent questions, as well as a vital component of the use-case application.

Figure 2 offers an overview of the full methodology. The schema acts as a general guide, and is subject to change, based on the obstacles encountered during the study. Please note the causality between the different phases, represented by thick arrows. The Compile and Interface phases can be executed in parallel, and require no particular order. The Synthesize and Utilize phases depend on side products from phases before it.

The methodology used can be characterized as both incremental and iterative. Its incremental nature means that every phase, and every step within a phase, will produce meaningful in-between products & results. This ensures the study will be insightful, even in the case the full scope of this study might become unfeasible. It is also vital for making the methodology iterative.

The study's iterative nature, represented by the cyclic paths within Figure 2, means that analysis and benchmarks will not be postponed until the end of the study, and will instead be an intricate part of each step. This makes sure obstacles are discovered early, and the trajectory of the study can be adjusted dynamically.

Every phase contains a number of development & analysis steps. The reasoning behind these steps are covered by the following sub-chapters.

3.1 Phase 1: Compile

The first sub-question is dedicated to overcoming Obstacle 1, and uses the related works on Wasm. It goes: **"What is the most fitting methodology of compiling C++ geoprocessing libraries to WebAssembly?"**. We specify ourselves to C++, since this seems to be the language of choice for almost all well established geoprocessing libraries. 'Fitting' in this context refers to the specific requirements posed by geoprocessing libraries. The libraries are large and complex, and the geodata used as input even more so. This means special attention will be given to these aspects. Standard compiler effectiveness criteria, such as portability (smallest file size), and performance, will also be considerations during the assessment of methodologies.

While the question poses to find the *best* compilation method, if it turns out that only one method makes it possible to compile sizable geo-libraries, this phase will nonetheless regard itself as successful. The question will have to be rephrased in that case.

Performance

The performance benefit of WebAssembly is an important component of why WebAssembly might be beneficial for client-side geoprocessing. As such, this phase is interested in confirming whether this is the case for geoprocessing applications. Once a sufficient compilation method is found, individual functions of geoprocessing libraries will be benchmarked using three different methods:

- Compiled and run as native binary (g++),
- Compiled to wasm, run natively (WASI),
- Compiled to asm.js, run natively (NODE.js),

Test Cases

CGAL & GDAL will be used as examples of "C++ geoprocessing libraries" for this phase. For one, these libraries are well established and relevant to geoprocessing as a whole. Many other geo-libraries depend on them. Moreover, they are large and complex, making it highly likely the problems described by related works will be encountered. We could choose more simple libraries, but this will not be representative of most C++ geoprocessing libraries.

While CGAL & GDAL will be this phase's primary subject, the answer of this sub-question aims to be applicable to all geoprocessing libraries.

3.2 Phase 2: Interface

Phase 2 is dedicated to overcoming Obstacle 2, assisted by the related works on the geoweb. Phase 2 seeks to not only make geoprocessing runnable, but actually usable, and usable to a wide audience of "data-users". This quest is posed using the research question: **How to design and create a client-side geoprocessing environment for data-users?**.

Phase 2 will primarily be about implementing the foundations of the use-case application 'GeoFront'. None of the existing web-based VPL's were deemed acceptable for the scope of this study, so the application will have to be created from scratch. The application will be created using JavaScript, or its type-save equivalent TypeScript. A choice can be made to write the whole environment as a native application which then, as a whole, can be published to the web using WebAssembly. This would probably be the most performant. However, referring back to the FAIR principles of [Mark D Wilkinson et al., 2016], this would be detrimental to the concept of Interoperability and Reusability. We wish the environment to contain small, standalone components which could be useful in and of themselves. For example, users might want to integrate a geodata process on their own website without the full GUI attached.

The modern web contains many technologies we can possibly use to facilitate all required features. WebGL offers the ability to efficiently visualize 3D data. The 2d canvas API and SVG's can be used to visualize 2D data. HTML can be used to build the interface.

Steps

Just like the entire study, the development trajectory during phase 2 will be done incrementally, ensuring results can be shown during all steps of the development. The first step of the phase will consist of creating the basics of the GUI itself. a basic VPL will be created which can only process boolean statements. The second step adds types, geometry, and the visualization of this geometry in 3D, as well as textures / images in 2d. The third step will add geospatial data support, like Web Feature Services, Web Map Services, and coordinate reference systems.

3.3 Phase 3: Distribute

The third phase is characterized by harmonizing the results of Phase 1 and 2. The related works pointed out that WebAssembly can only truly be tested within a realistic use-case scenario, So this Phase intends to do just that. The research question goes: **What is the best way of distributing wasm-compiled geoprocessing libraries, in order to use them within a client-side geoprocessing environment?**. This phase can be seen as a continuation of phase 1, but where the compilation research of phase 1 limits itself to native, CLI usage of WebAssembly, this phase introduces the web, and the developed interface during phase 2 as new factors to this research. Given this as the desired way of processing geodata, how can WebAssembly facilitate these desires?

This will result in new benchmarks, and new analyses, now including factors like client-side (down)load times, compilation, and utilization. Answers will have to be given to questions such as *Where do the wasm-compiled libraries live?* and *how are they cached?* .

3.4 Phase 4: Utilize

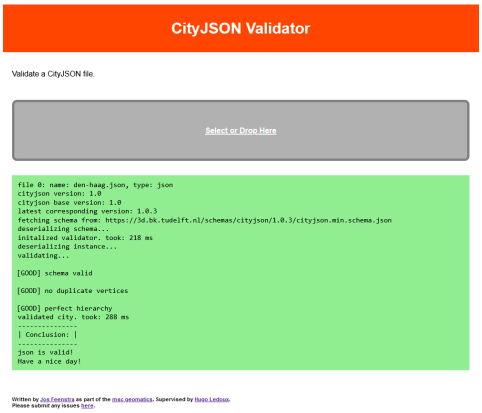
Finally, When the VPL contains all tools necessary to be used to properly process geodata, a final assessment can be made by using the environment to serve as an application. this assessment will try to overcome Obstacle 3 by posing the question: **What are the advantages and disadvantages of GIS applications created using a client-side geoprocessing environment powered by WebAssembly?**. This question requires a native but comparable GIS application to test this against.

We hypothesize that applications equipped with client-side geoprocessing open up a whole range of new possibilities posing both academic & commercial benefits. These aspects of the study can be discussed during this phase.

Case Study

This is a sketch of an application which could be created if the use-case application performs as intended. A series of steps & geoprocessing operations are given to give an impression of the complexity of such an application.

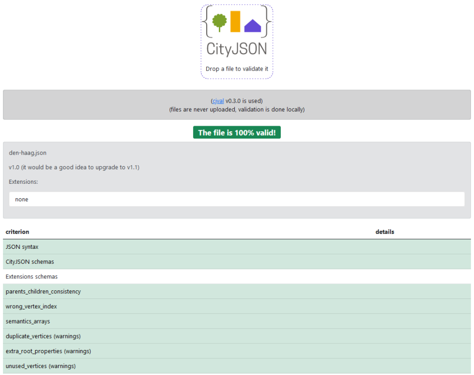
```
# Demo Application: On Demand Isocurves Generator

# 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)
```



```
file 0: name: den-haag.json, type: json
cityjson version: 1.0
cityjson base version: 1.0
latest corresponding version: 1.0.3
fetching schema from: https://3d.bk.tudelft.nl/schemas/cityjson/1.0.3/cityjson.min.schema.json
deserializing schema...
initalized validator. took: 218 ms
deserializing instance...
validating...

[GOOD] schema valid

[GOOD] no duplicate vertices

[GOOD] perfect hierarchy
validated city. took: 288 ms
---------------
| Conclusion: |
---------------
json is valid!
Have a nice day!
```

Written by Jos Feenstra as part of the msc geomatics. Supervised by Hugo Ledoux.
Please submit any issues here.

Figure 3: *The prototype*



(cjval v0.3.0 is used)
(files are never uploaded, validation is done locally)

The file is 100% valid!

den-haag.json

v1.0 (it would be a good idea to upgrade to v1.1)

Extensions:

none

criterion	details
JSON syntax	
CityJSON schemas	
Extensions schemas	
parents_children_consistency	
wrong_vertex_index	
semantics_arrays	
duplicate_vertices (warnings)	
extra_root_properties (warnings)	
unused_vertices (warnings)	

Figure 4: *The official validator*

4 Preliminary Results

As visualized in Figure 2, two steps of this proposed methodology has already been carried out: A small-scale wasm application, and the basics of a browser-based visual programming language. These where both part of the preliminary work done in preparation of this thesis proposal. This chapter will elaborate upon these works.

4.1 Wasm Application

WebAssembly is a major component of this proposed thesis. However, the compilation target is still widely regarded as being "bleeding edge" [Jangda et al., 2019]. This makes it all the more vital to gain some level of experience using the compile target before using it in more complicated scenario's.

The Geomatics curriculum this thesis proposal is part of, offers a subject called "Research Assignment". This course offers students the opportunity to gain experience in a geomatics-related topic of choice, and this made it the perfect context to try out WebAssembly for a geomatics web application.

The assignment given was to deliver a prototype for an online cityJSON validator. WebAssembly was to be used to perform a very fast JSON schema check over a user submitted json, among other things. This assignment was carried out successfully, and the delivered prototype (Figure 3) played a role in the official CityJSON validator (Figure 4), which uses WebAssembly.

Creating this application offered many key insights of working with WebAssembly, which will undoubtedly play an important role in the proposed thesis.

4.2 VPL Prototype

The 'GeoFront' use case application has also received preliminary attention, in the form of a demo build. The focus of this demo was on the GUI, and as such, it supports only basic types.

The current application looks like Figure 5. Input, output and operators can be placed on a canvas, and lines can be drawn in between these components. The graph then determines the order of execution by means of a Directed Acyclic Graph (DAG), and executes this any time an input has altered, or when the graph has been reconfigured. All graphics are provided by using the 2d Canvas API, native to modern web browsers.

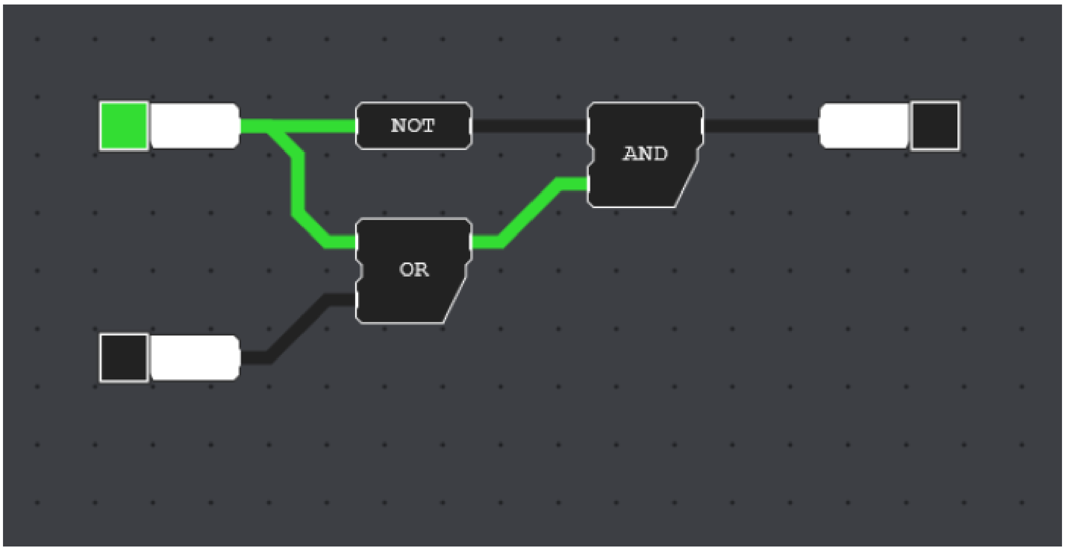


Figure 5: *The 'Geofront' environment.*

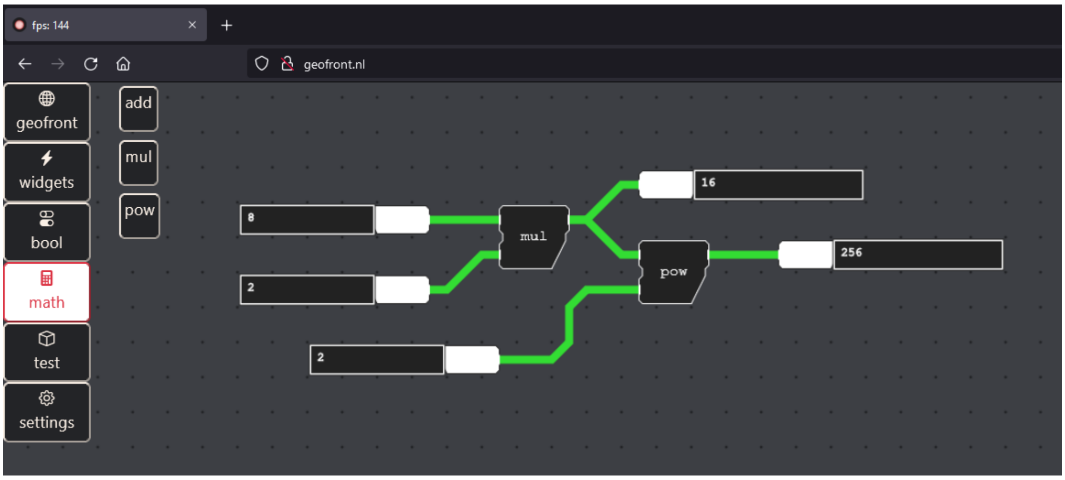
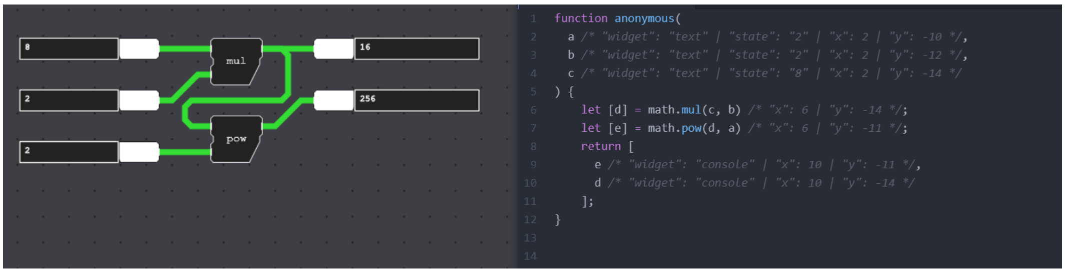


Figure 6: *GeoFront, showcasing basic mathematics.*



```javascript
function anonymous(
  a /* "widget": "text" | "state": "2" | "x": 2 | "y": -10 */,
  b /* "widget": "text" | "state": "2" | "x": 2 | "y": -12 */,
  c /* "widget": "text" | "state": "8" | "x": 2 | "y": -14 */
) {
    let [d] = math.mul(c, b) /* "x": 6 | "y": -14 */;
    let [e] = math.pow(d, a) /* "x": 6 | "y": -11 */;
    return [
      e /* "widget": "console" | "x": 10 | "y": -11 */,
      d /* "widget": "console" | "x": 10 | "y": -14 */
    ];
}
```

Figure 7: *GeoFront, showcasing javascript interoperability.*

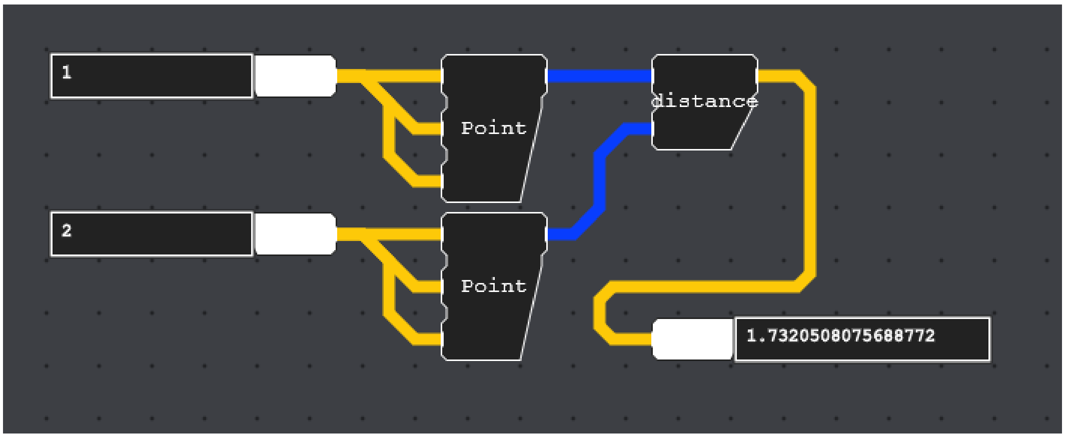


Figure 8: *GeoFront, showcasing support for basic structs. Types are color coded.*

Besides boolean values and operations, basic mathematical operations are also supported (see Figure 6). Panels utilizing html are used to provide inputs and showcase outputs.

Figure 7 shows how a graph can be saved as JavaScript. This script can be run completely independent of the geofront application, as long as the libraries used by the individual operation are included. The script can also itself be turned into an operation. This is meant for encapsulation, just like how regular programming languages support the creation and re-use of functions.

Lastly, Figure 8 shows that the first steps are made towards structs, objects, and other complex data types.

5 Planning and Organization

The study proposed serves as a MSc Geomatics Thesis. As such, several assessment dates are in order:

- The Graduation plan presentation (P2) will take place at January 28th.
- The Midterm progress meeting (P3) will take place at the end of March / beginning of April.
- The Go/no-go assessment (P4) will take place mid May.
- The Public presentation and final assessment (P5) will take place at the end of June.

A Gantt chart (Figure 9) explains how these assessment dates relate to the overall methodology.

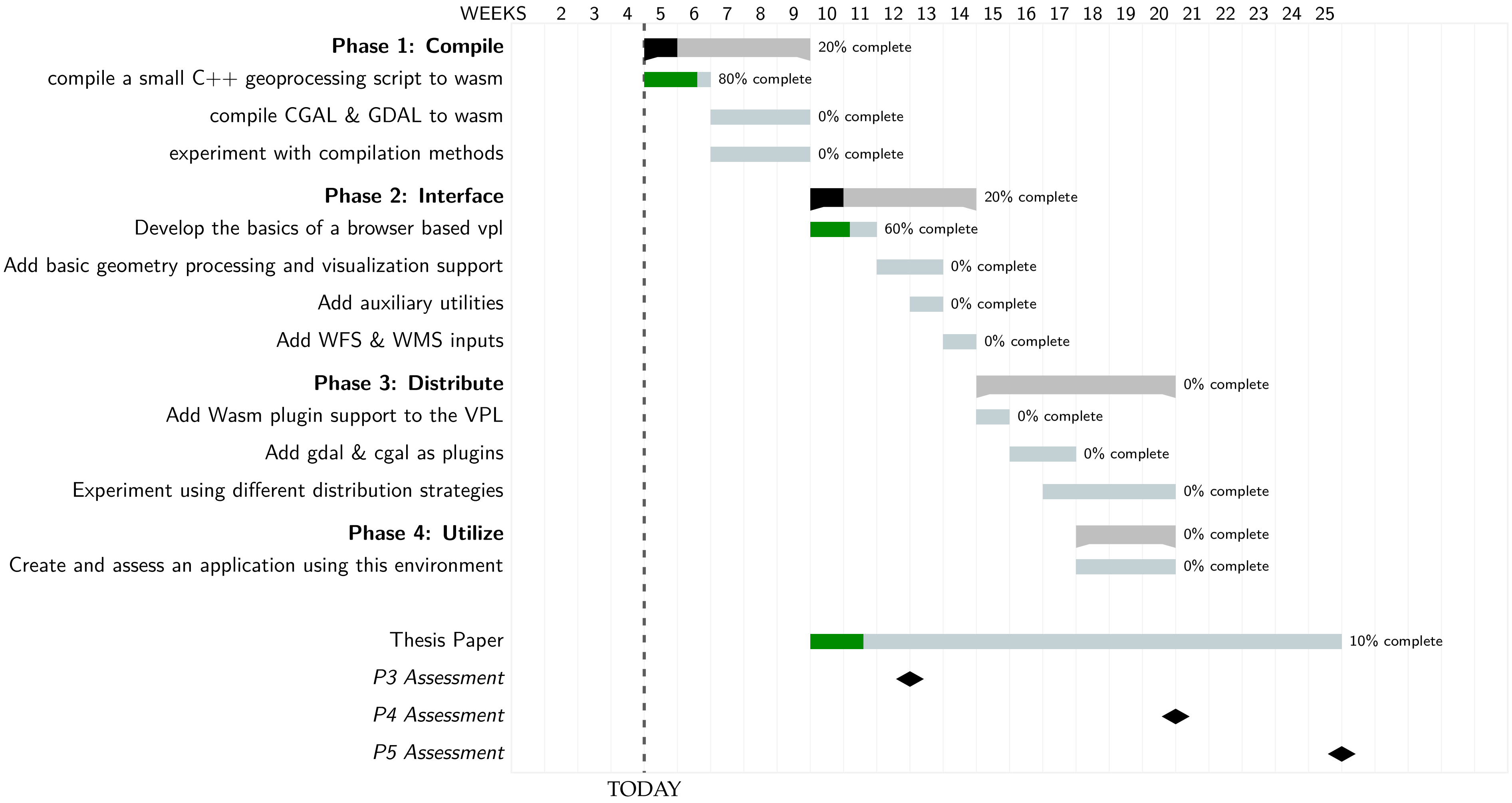


Figure 9: *Gantt chart of the methodology, made with pgfgantt*

6 Tools and Data

6.1 Tools

In order to perform the study proposed, several tools are needed. All tools mentioned are free and open source.

Typescript

Typescript will be the programming language used for creating the main visual programming language application. Typescript is chosen over javascript, since the safeguards and type checks provided at compile time are very beneficial when creating medium to large scale applications.

Depending on the size of the application, a framework such as React or Vue might be used to offer structure.

HTML5

HTML5 provides adequate features in order to create the actual GUI. Buttons and panels can be created using just HTML. The Canvas 2d API is an excellent method of drawing 2d shapes, and can be used to draw the components and cables of the VPL itself. Alternatively, this can also be done using Scalable Vector Graphics (SVG's).

WebGl

WebGl allows for performant web visualizations using syntax similar to OpenGL. This will be used to make preview visualizations of 2D and 3D data. Modern web browsers are equipped with WebGl by default.

WebAssembly

WebAssembly binaries, and its surrounding tooling, will of course be used. Modern browsers provide WebAssembly support by default. For running WebAssembly files locally, WASI, a native WebAssembly Runtime will be utilized.

C++ & Emscriptem

Most geoprocessing libraries commonly used within the geospatial community are C++ based libraries, such as CGAL & GDAL. A toolchain called Emscripten will be used to compile these libraries into WebAssembly. Additional C++ wrapper libraries might be written if direct compilation proves to be difficult.

Rust & wasm-pack

Rust is the community preference for developing programs targeting wasm. The rust ecosystem contains a number of powerful tools such as wasm-pack, and the standard rust compiler includes wasm as a compilation target by default. If during this study a need arises for newly written geoprocessing libraries, Rust will be preferred over C++ for these reasons. But, by default, Rust will not be used during this study, since no well-known geoprocessing libraries exist within its ecosystem.

6.2 Data

This study is not dependent on specific datasets. However, in order to use Geodata processing libraries, we will have need of some data to start with. Demo data of the netherlands will be used, using the WFS and WMS services provided by the Dutch geodata portal PDOK.

References

[Brink, 2018] Brink, L. v. d. (2018). *Geospatial Data on the Web*. PhD thesis. original-date: 2018-10-12T08:52:14Z.

[Clack, 2019] Clack, L. (2019). Standardizing WASI: A system interface to run WebAssembly outside the web – Mozilla Hacks - the Web developer blog.

[Community, 2022] Community, Q. (2022). QGIS Homepage.

[Esri, 2022] Esri (2022). ArcGIS Homepage.

[Haas et al., 2017] Haas, A., Rossberg, A., Schuff, D. L., Titzer, B. L., Holman, M., Gohman, D., Wagner, L., Zakai, A., and Bastien, J. (2017). Bringing the web up to speed with WebAssembly. In *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI 2017, pages 185–200, New York, NY, USA. Association for Computing Machinery.

[Hamilton, 2014] Hamilton, E. L. (2014). *Client-side versus Server-side Geoprocessing: Benchmarking the Performance of Web Browsers Processing Geospatial Data Using Common GIS Operations*. Thesis. Accepted: 2016-06-02T21:00:45Z.

[Jangda et al., 2019] Jangda, A., Powers, B., Berger, E. D., and Guha, A. (2019). Not So Fast: Analyzing the Performance of WebAssembly vs. Native Code. pages 107–120.

[Kulawiak et al., 2019] Kulawiak, M., Dawidowicz, A., and Pacholczyk, M. E. (2019). 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. *Computers & Geosciences*, 129:26–37.

[Mark D Wilkinson et al., 2016] Mark D Wilkinson, Appleton, Myles Axton, Arie Baak, Niklas Blomberg, Jan-Willem Boiten,, and Luiz Bonino da Silva Santos, Philip E Bourne, et al. (2016). The FAIR Guiding Principles for scientific data management and stewardship.

[Melch, 2019] Melch, A. (2019). Performance comparison of simplification algorithms for polygons in the context of web applications.

[Mozilla, 2013] Mozilla (2013). asm.js.

[OGC, 2015] OGC, O. G. C. (2015). Web Processing Service.

[Panidi et al., 2015] Panidi, E., Kazakov, E., Kapralov, E., and Terekhov, A. (2015). Hybrid Geoprocessing Web Services. pages 669–676.

[w3c, 2019] w3c (2019). World Wide Web Consortium (W3C) brings a new language to the Web as WebAssembly becomes a W3C Recommendation.