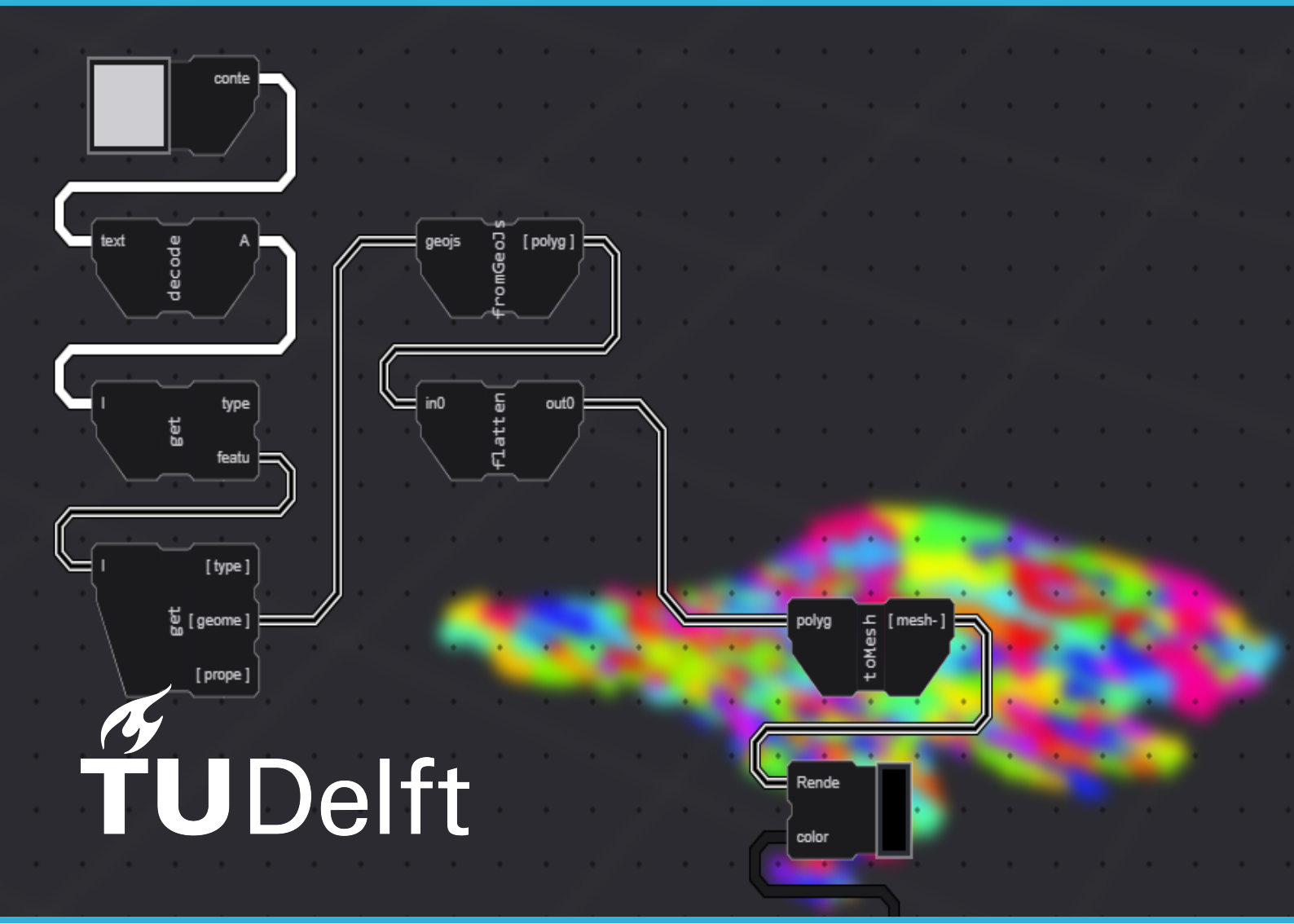
MSc thesis in Geomatics

GEOFRONT: A browser based visual programming language for geo-computation

Jos Feenstra

2022



TU Delft

MSc thesis in Geomatics

GEOFRONT: A browser based visual programming language for geo-computation

Jos Feenstra

June 2022

A thesis submitted to the Delft University of Technology in partial fulfillment of the requirements for the degree of Master of Science in Geomatics

Jos Feenstra: *GEOFRONT: A browser based visual programming language for geo-computation* (2022)

The work in this thesis was carried out in the:

3D geoinformation group
Delft University of Technology

Supervisors:	Ir. Stelios Vitalis
	Dr. Ken Arroyo Ohori
Co-reader:	Associate. Prof. Hugo Ledoux

Abstract

NOTE: this is a salespitch, not an abstract :)

This thesis presents GeoFront, a web-based visual programming language, similar to Grasshopper & FME. With GeoFront, geoprocessing & computational design flowcharts can be viewed, run, and shared using the web. The full flowchart runs client-side in a browser, and both end results and intermediate products can be inspected in a 3D viewer.

GeoFront offers several functionalities such as the parametric creation of 2D and 3D primitives, triangulation, isocurve extraction, and more. These functionalities can be expanded upon though a plugin system which utilizes the existing "Node Package Manager" infrastructure. Together with WebAssembly, this enables the utilization of industry standard geoprocessing libraries such as 'CGAL', 'GDAL' and 'PROJ', and data parsing libraries such as 'IFC.js' and 'laz-rs'.

Geofront has been created as an experiment to explore if visual, browser-based geo-computation can make geo-computation in general more accessible. The advantages and disadvantages of browser based geo-computation, compared to native or server-side geo-computation, are examined in several scenario's. Both quantitative indicators, like loading and runtime performance, as well as qualitative indicators, like the fitness for an intended use-case, are measured in each of these cases. This study concludes that based on these measurements, browser-based geo-computation is not only possible, but fast, and an enabler of many promising use-cases, such as on-demand geodata processing apps, educational demo apps, and code sharing. However, extensive user-group testing is required before any definitive statements on accessibility and fitness for geo-computation can be made.

x are continuously becoming more popular among practitioners due to [...] However, there is no [...] to allow [...]

This thesis attempts to address this gap by proposing and implementing a general Web-based visual programming language for geo-computation. This 'geo-web-vpl' is designed to fulfill what [...] Those requirements were identified after [...]

[what]

Following the implementation, the project was tested by simulating use-case scenarios. The tests demonstrate the feasibility of [...] At the same time some key parameters of [...] identified which if tuned properly they can optimize the performance, behavior and robustness of the geo-web-vpl. With the project being a prototype solution, the vpl is far from operational and there is certainly a lot of space for improvement regarding both components.

Utilizing the workflow in practice would be the ideal way for collecting useful feedback. Besides that, This is already a useful environment for demo's, education, and small-scale geometry processing.

Acknowledgements

- Stelios
- Ken
- Martin, Current employer, GeoDelta
- Sybren, Previous employer, Sfered
- Nadja
- Tim
- Friends & Family

...

Contents

Acronyms

1 Introduction

Notes:
– I still use browser–based geoprocessing and the geo–web–vpl quite interchangably in the introduction. Both need to be linked up better.

In order to specify the contribution of this thesis, I wish to bring two fields of study to the reader's attention: Visual programming, and Web GIS. After this, the subject and goal this thesis can be made clear.

Visual programming

What is a VPL and how is it used within the field of geo-information?

A Visual Programming Language (vpl), or visual programming environment, is a type of programming language represented in a graphical, non-textual manner. A VPL often refers to both the language and the Integrated Development Environment (IDE) which presents this language.

Within the field of geo informatics, VPL's are most often used for specifying geodata transformations and performing spatial analyses. SaveSoft's FME (SOURCE) is a good example of this. This Extract Load Transform (ETL) platform automates data integration, and is widely used by GIS experts. VPLs like these offers users a chance to interactively automate workflows & processing pipelines, while requiring little to no programming knowledge. In between results can be inspected intuitively, and the processes can be changed on the fly, often with immediate feedback. This advantage of interactive, low-code automation is why the VPL continues to be a popular interface within the field of GIS, as well as all many other use-cases in need of both low-code automation and visual debugging, like BIM, CAD, Shader Programming and Procedural Geometry. A VPL done right can make automation available to a very large audience.

Web GIS

What is web GIS and why is browser-based geo-computation relevant?

Web Geographic Information Systems (gis) forms an indispensable component of the wider geospatial software landscape. For the average person, an interactive gis 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 accessibility, since no installment or app-store interaction is required to run or update the app. (src: vpl 2019, src: hybrid) As soon as it can be found, it can be used. The ability to share a full application with a link, or to embed it within the larger context of a webpage

is also not trivial. Together, these aspects have made the browser a popular host for many important geographical applications, especially when accessibility is vital.

Despite the popularity of geographical web applications, the range of actual gis abilities these applications are capable of is very limited. Geospatial data computation (geo-computation) abilities, like CRS translations, bundle adjustment, 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 geo-computation 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 geo-computation and analysis tools, which end-users could use to post-process and analyze geodata quickly, uniquely, and on demand.

This is why geo-computation within a web application, also known as Browser-based geoprocessing (bbg), 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 have been 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 geo-computation within a web application as of today still almost nowhere to be found?*

1.1 Problem Statement

Taken together, the proven usefulness of a vpl for geodata computation, together with the theorized potential of browser-based geo-computation, raise the question why a true browser-based visual programming environment for geo-computation (geo-web-vpl) does not exist yet.

Prior studies give an indication to where the disconnect between the field of geo-VPLs and web-GIS may be. We encounter either geo-VPLs which were not able to be properly used in a browser, or web-based VPLs unable to support geo-computation functionalities. This suggest that geo-computation functionalities:

- Were not able to be properly compiled into a format functional on the web
- Were not able to be properly consumed and used by a web-based VPL
- Were not able to be properly facilitated by the interface of a non-geo web-VPL

Moreover, the real reason of this disconnect may not lie in one of these areas, but in the interplay between all three of these factors.

1.2 Research Objective

This study seeks to close the gap between geo-VPLs and web-GIS by designing and implementing a geo-web-vpl. The study starts from the presupposition that proper utilization of existing geo-computation functionalities and libraries within a VP environment is key to making a geo-web-vpl succeed. As such, overcoming this technical challenge is the focal point of this study.

1.3 Research Questions

Based on this objective, the research question is formulated as follows:

> *How to design and create a web-based vpl which can utilize existing geoprocessing libraries?*

Sub Questions

The following sub-research questions are needed in order to answer the main question. These are directly based upon the components of the problem statements.

- *Libraries: To what extend can existing geoprocessing libraries be made available for web consumption?*
- *Interface: How to implement a visual programming environment on the web which supports geometry?*
- *Distribution: How can web-consumable libraries be loaded and used within a web-vpl?*
- *Utilization: What are the advantages and disadvantages of using existing geoprocessing libraries through a geo-web-vpl?*

In order to obtain answers to these questions 1) an literature review is done, 2) a prototype software application is developed as a proof of concept and 3) this application is tested and compared to existing methods using real-world datasets.

1.4 Scope

The scope of this thesis is cornered in the following ways:

Only frontend geo-computation

This study excludes any *backend* based geo-computation. A web application *could* be used to orchestrate geo-computation web-services, which could also deliver a form of browser-based geo-computation to end-users. However, for the scope of this thesis, we limit ourselves to purely client-side solutions, with calculations literally happening within the clients browser. This is also why this study excludes the OGC standard of the Web Processing Service (wps) OGC [2015].

Adding backend-based geo-computation to a geo-web-VPL would be an excellent follow-up investigation to this study. Future work could research the possibility of utilizing a hybrid strategy of both client-side and server-side geo-computation, following in the footsteps of Panidi et al. [2015].

No Usability Analysis

While accessibility / usability is a motivation of the development of a vpl, no claims will be made that this method of geo-computation is more usable as opposed to existing geo-computation methods. This research attempts to solve practical inhibitions in order to discover whether or not browser-based, vpl-based geo-computation is *a* viable 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 browser-based geo-computation in comparison to native geo-computation must also be left to subsequent research. While this would gain us tremendous insight, client-side geo-computation is too new to make a balanced comparison. Native environments like QGIS, FME or ArcGIS simply have a twenty year lead in research and development.

Only WebAssembly-based containerization

This thesis examines a WebAssembly-based approach to containerization and distribution of geo-computation functionalities. Containerization using Docker is also possible for server-side applications, but is not usable within a browser. For this reason, Docker-based containerization is left out of this studies' examination. And to clarify: Docker and WebAssembly are not mutually exclusive models, and can be used in conjunction on servers or native environments.

Mostly Point Cloud/ DTM focussed geo-computation

We are also required to concentrate on the scope of 'geo-computation', which is a sizable phenomenon. The term is generally used to cover all operations on geodata, from rasters, tabular datasets, 'object-oriented datasets' such as the CityJSON or IndoorGML, and point clouds. Due to time limitations, we are forced to focus on particular type of geo-computation. 3D-based geo-computation is chosen, with a particular focus on pointclouds and and DTMs. The hypothesis is that these types of data may benefit the most from 'geo-web-vpl' based geo-computation.

Only core browser features

Lastly, the implementation of the geo-web-vpl will limit itself to core browser features, keeping dependencies at a minimum, in an attempt to generalize the results of the study. If the study would use very specific frameworks and technologies to solve key issues, questions might arise if the results of the study counts in general for browser-based geo-computation or geo-VPLs, or if they only count in this very specific scenario. "Core browser-" - Only using basic HTML5 features.

This study defines "Core browser features" as the set of features present in most major, contemporary browser vendors by default, by which we mean Firefox, Safari, and chromium-based web browsers (src). This includes modern HTML features such as WebGL, the Canvas API, Web Workers, Web Components, and WebAssembly.

1.5 Reading Guide

This is still a sketch, this must be tweaked as the the thesis evolves.

-> chapter 2: Related Work

Covers related studies in the field of geo-VPL and web GIS.

VPL: To what extend does this study overlap with other geo-vpl's?

GEOWEB: Which technologies are available for browser-based geoprocesing?

-> chapter 3: Methodology

Explain precisely in what way the sub-questions will be answered.

-> chapter 4: Implementation

Provide an answer for the *Libraries, Interface, and Distribution* sub-questions.

Answer of "Libraries": Show all wasm-considerations: C++ vs Rust vs rewriting in Js. Emscripten vs wasm-bindgen. Conclude with the preference for rust, and explain why.

Answer of "Interface": Show the full application implementation. Conclude with what about this implementation is relevant for geo-computation.

Answer of "Distribution": the plugin system schema.

-> chapter 5: Experiments

Provide an answer for the question of Utilization.

Show the results of usage of GeoFront & the results of experiments.

-> chapter 6: Discussion

discuss to which extend the solution was able to satisfy main research question.

discuss unaddressed aspects of the thesis, such as the massive nature of geodata, and the utility of a vpl for geo-computation.

-> chapter 7: Conclusion

conclude to which extend the solution was able to satisfy the main research question.

-> chapter 8: Future work

Show the potential reach of this research, and how it might be expanded

...

2 Related work

This thesis takes place at the intersection of two fields of study: the VPL, and Web GIS. The two topics have already been mentioned briefly during the introduction. Specific challenges

These fields

- diverse challenges exist within both fields

- this chapter -¿ show how these challenges relate to features refine

2.1 Visual Programming

```
write something about :
- cognitive_1996
- advances_2004
- characterizing_2021

An increasing number of software applications are being written by end users
without formal software development training.
This inspired large technology companies such as Microsoft [91] and Amazon [90]
to invest in low-code development environments empowering end users to
create web and mobile applications.
According to the 2019 Q1 Forrester report, the low-code market will witness an
annual growth rate of 40\%,
with spending forecast to reach \$21.2 billion by 2022 [102].
End-User Development (EUD) has emerged as a field that is concerned with
tools and activities allowing end users
who are not professional software developers to write software applications [11].
This is promising as end users know their own domain and needs more than anyone else,
and are often aware of specificities in their respective contexts.
Further, as end users outnumber developers with professional software development
training by a factor of 30-to-1,
EUD enables a much larger pool of people to participate in software development [12].
A visual programming language (VPL), among other EUD techniques,
allows end users to create a program by piecing together graphical elements
 rather than textually specifying them [9].
Traditionally, visual programming has been successfully used to help novices
 learn basics of programming by visualizing elements of a program.
However, visual programming is increasingly being used by end
users in various domains to create and tailor applications that are useful
beyond the realm of education.
```

```
For instance, VPLs are now being used in fields such as
the Internet of Things (IoT) [3], [10], mobile
application development [51], robotics [8], and Virtual/Augmented Reality [4].
}

From characterizing_2021

other major use cases:
- PLC: Ladder
```

Dataflow modelling

```
- Dataflow modelling is a field closely related to visual programming.
- However, where visual programming is concerned with many aspects,
interface and usability being one of them, dataflow modelling is
primairly concerned with the correct representation of data transformation.
- within the field of dataflow modelling, it turned out that certain
visual programming paradigms are advantageous, since they make parallel
programming explicit. Thus, these fields are often named in conjunction.
- The important take-away is that visual programming is not
just a matter of UI or a stylistic choice.

- By more correctly representing dataflow and communicating
opportunities for parallel computation, it can lead to faster applications.
```

In Geometry Computation & Visualization

```
  - Besides the use cases already mentioned, a significant number of
  visual programming applications are emerging in fields concerned
  with 2D and 3D geometry creation \& visualization.

PROCEDURAL GEOMETRY
  - Rhino: Grasshopper
  - Revit: Dynamo
  - Blender: Geometry Nodes
  - Houdini: Procedural Modelling

TEXTURES AND SHADERS
  - Blender: Shader Nodes
  - Adobe: Substance Designer
  - Unreal Engine: Material Nodes
  - Unity: Shader Graph
  - Houdini: FX

These are all popular applications, many users, multiple courses and tutorials

The persistence of visual programming within the field of shaders and
```

geometry, suggests that visual programming languages are advantageous in situations where a 'visual' product requires debugging during development.

In field of geo-informatics

(explain the use-case of vpl's within the field of geo-informatics, why they are significant to us)
within geomatics

- data translation: FME
- cloud-native computation: modellab in rasterfoundry
  - link to dataflow modelling
- debugging & experimentation: GeoFlow

Challenges

- several challenges exist within the research field of visual programming.
- literature study (knowling what is and isnt being researched)
- based upon : prior meta-analysis of 'characterizing_2021'

Difficulty in assessment of 'usability'

- usability is a nebulous phenomenon, hard to measure empirically.
- no consensus on evaluation frameworks among VPL researchers.

BUT (leave this part, mention it later)
- we will make no such attempt. usability serves as background motivation.
- this study assumes vpl's are 'in general' more usable to end-users than text-based alternatives, based on the positive results of most of the studies analysed by communicating_2021.

web based visual programming is underresearched

large challenge
- not many examples
  of these examples:
  - no suitable starter projects
  - none are concerned with geometry

  - Visual programming environment for geo-computation \emph{in a browser} has not been tried before.

"Finally, 53.3% (16) of the tools were available publicly with some document
We strongly recommend that future tools are made available for end users as
-> not fully related, but making this vpl web-based could seriously help this

VPL availabilty and life cycle

Finally, (communicating_2021) names the 'life cycle' of apps created with the VPL as one of the most overlooked aspects within VPL research.
- extend
- debug
- publish
- run

2.2 Web GIS

This is dated. this section includes many old introductions, and requires refinement.

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.

Browser-based geoprocessing

This study is related to a series of studies concerned with browser-based geoprocessing, also refered to as 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]. To fix this ambiguity, this study refers to client-side geoprocessing as 'browser-based geoprocessing'.

Browser-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 webpages. 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, like 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. 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. Instead, the authors propose a hybrid approach, combining the advantages of server-side and client-side geoprocessing. They also present the 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 the web 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 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 recent browser features, 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++/Rust geoprocessing libraries to the web.

WebAssembly

From all browser-based features, WebAssembly is one of the newest, and turned out 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 geo-computation 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 reffig fig:wasm-trajectory, 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 WebAssembly (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.

2.2.1 (More on webassembly)

Not just open source: process sharing using fully containerized instances. Think .

current vision / direction: containerized, sharable processes, together with web-based, front end visual programming environments (RasterFoundry). Docker is usually named as a vision for these sharable processes.

-> We do have examples of cloud-native geodata formats, and some examples of cloud-based geo-computation (RasterFoundry , Google Earth Engine, more). However, these approaches have not yet tried to use truly sharable, containerized geoprocesses using Docker or WebAssembly.

-> WebAssembly as a whole is underresearched. WebAssembly is not a fully virtualized container image, but just a binary set of instructions, meant to be executed on a virtual machine. Think of safe, cross-platform dll's. WebAssembly is in this regard more simple than docker, but this gives it more opportunities. WebAssembly runs in the browser for instance.

-> This opportunity to run in the browser would enable these cloud-native frontend environments to "dry-run" these processes from within the browser, completely detached from the server, as a means to experiment with processes on a small scale before applying them to a cloud native environment.

-> However, no implementations exist yet which combines containerized processes with these frontend computation environments.

Browser based geoprocessing software

- ¿ These are all similar efforts, also hooked up to the OGC cloud native developments Geofront will differ, for it focusses on 3D & Point clouds instead of rasters & map algebra, and focusses on client-side geodata consumption & processing instead of being a server-side pipeline configurer.

ModelLab says this : "Widespread access to frequent, high-resolution Earth observation imagery has created the need for innovative tools like ModelLab that will help individuals and organizations to effectively access, analyze, edit, and visualize remotely sensed data in transformative new ways without years of specialized training or ongoing investments in proprietary software and technology infrastructure. "

Client-side geoprocessing

TODO: Figure out how to phrase this better

While these studies pose a strong theoretical case for client-side geoprocessing, their practical implementations were less convincing . The implementations of Panidi et al. [2015]; Hamilton [2014] were written in a time before WebAssembly & major javascript optimizations, and the study of Kulawiak et al. [2019] prioritized theory over practice.

This study recognizes a need for a new, practical attempt at client-side geoprocessing. Client-side geoprocessing is a promising prospect with potentially many use cases. Previous attempts are either dated due to the web's rapid advancements, or chose theory over practice.

This needs a better phrase, but it really is the most direct way of putting it

In addition, the implementations lacked creativity .

The applications were either meant as small demo's, without a clear target audience or use-case in mind (just a way to demo performance), or as highly specified debugging tool for the authors. But, as mentioned before, a major advantage of Web applications over native

applications is *Accessibility*, and a major advantage of client-focussed web apps over server-focussed web apps is *Interactivity*. By creating a client-side web application which is neither accessible nor interactive, the main incentive for creating a client-side web app is lost. Additionally, by forgoing the question of accessibility, many auxiliary use-cases of client-side geoprocessing where overlooked.

The Cloud Native Geospatial movement

The Open Geospatial Consortium (OGC)... Mission: FAIR Geodata

A prominent development within the OGC is the recent effort towards a **"Cloud Native Geospatial"** future. This initiative aims to radically simplify geodata storehouses to static servers serving large, singular binary geodata files. All processing and analysis of this geodata can then be performed by separate cloud-based web services. This architecture has many advantages over current geodata storage and analysis methods:

- These new Cloud Native geodata formats are much cheaper to access by front-end and back-end services, compared to active services.
- Substituting active SQL or noSQL databases by static binary files is easier and cheaper for data providers, leading to more and more readily available geodata.
- By using supercomputers (Microsoft Planetary Computer) and cloud-storage (AWS), Geodata processes could make use of near-infinite computational and storage resources.
- By having all data centralized in one location or type of location, new, large scale patterns within our geodata could be discovered.
- For web GIS, this would offer direct data streaming options, similar to services like "Netflix" or "Spotify".

These features may have a far reaching impact on society. Chris Holmes, forerunner of the cloud-native geospatial movement, envisions what the movement could mean for even non-GIS users: *With the introduction of accessible, centralized data, and the dramatically different workflows that follow, Cloud Native Geospatial has the potential to introduce new, non-specialized users to the power of geospatial information that GIS practitioners have enjoyed for decades. [...]. The ecosystem of geospatial experts will collaborate to create analyses and insight, but any non-expert user will be able to select and apply those to the geographic area they care about. C̃hris Holmes* All these reasons explain why the OGC and many other parties are now actively pursuing this vision.

But while this vision is in active development, many large-scale challenges are still in its way. One of the most important challenges is the required paradigm shift within geo-computation / geoprocessing workflows. The current, common geo-computation workflow of retrieving online data, only to run it through a local process and send the resulting data back into servers, will have to be reversed: In a cloud-native future, we will not retrieve data for our local process, but we will upload our process to the data. This introduces a sizable challenge: **Portable, Containerized Geo-computation**.

The challenge of sharing and chaining together containerized fragments of geoprocesses to a variety of environments will require more than just open source collaboration. This study interprets the challenge of portable geo-computation by means of the FAIR paradigm. If geodata processes need to be just as portable as the geodata forming the input and output,

then perhaps the FAIR paradigm should extend from FAIR geodata to FAIR geodata *processing* as well. The challenge facing the cloud-native vision then becomes: **How to make geo-computation Findable, Accessible, Interoperable, and Reusable?** This links back to containerization, for containerization is a very powerful method of making geo-computation more Interoperable and Reusable.

Improve this intro

The current state of the art is far removed from either portable or FAIR geo-computation.

- current methods: Docker, and some geo-computation platforms.
- Not many implementations using WebAssembly, while this is a prime candidate: Even the guy who made Docker said so.
- ignore the cloud: focus on the act of containerizing geoprocesses using webassembly an sich

FAIR geoprocessing may have many auxiliary advantages.

CITYJSON VALIDATOR ARGUMENT: THE WEB CAN BE USED TO IMMEDIATELY MAKE SOME TOOL / SOME RESEARCH PROJECT OPERATIONAL 'IN THE REAL WORLD'. THIS WAY, DATA CAN BE GATHERED, USER FEEDBACK CAN BE GATHERED, AND THE TOOL CAN BE EVALUATED IN TERMS OF REPRODUCABILITY. FINALLY, IT OFFERS THE POSSIBILITY OF THE TOOL BEING ACTUALLY USED, IN PRACTICE. z

2.2.2 Challenges

```
Relevant Challenges & developments within web GIS:
- cloud-native movement
-
```

2.3 Aknoledgements

- Name similar studies or things - Ravi Peter: awesome c++ based application - not web - not formal reseach

3 Methodology

3.0.1 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.

3.0.2 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 Graphical User Interface (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.

3.0.3 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?* .

3.0.4 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.

3.1 Nature

The prior works on browser-based geoprocessing indicate that a theoretical framework for browser-based geoprocessing vpl is in place. But, and this is especially evident in the studies regarding client-side geoprocessing, the practical implementation of these theories

This necessitates a practical approach in response.

, the advantages of a vpl,

3.2 Inhibitors of a geo-web-vpl

VPL - Closed-source nature - Difficulty to integrate with 'regular' software

Browser-based geoprocessing - Nature of geodata processing: - Big data - lots of IO operations (writing files away)

- Javascript: - different library ecosystem - hard to utilize for geodata which requires a strongly typed environment

3.3 Proposed Model

(a diagram showing relationships between vpl, libraries, javascript, webassembly)

- javascript-subset library wrappers - - - magic methods

- different javascript-subset for vpl runtimes

3.4 Features

Q: Which features are required for a 'usable' VPL intended for geo-computation?

We define three major features.

These features are in turn based upon related works within VPL research, as well as web GIS studies. This section outlines these requirements, and explains why they are used. Additionally, certain use-cases are envisioned as proxies of these requirements, to test certain requirements in a realistic scenario.

...

Feature A: Interactive

A VPL should be Interactive.

```
  Interactivity is the defining factor of the vpl.
  a list of standard VPL features & application features required as a base-line
- Users must be able to construct a script by visual means.
- Dataflow Modelling
- Dragging and dropping is a ui which

(geo-vpl features:)
- read data from user-submitted files
- write data to files, downloadable by the user
- debug / inspect data in a 3D viewer
- draw geometry in a 3D viewer
```

Feature B: Extendable

A VPL should be Extendable.

REASON:
- VPL: Important take away from meta-analysis: extendability
- WEB: Geoweb: FAIR principles -> fair geoprocessing

CRITERIA
- ease of creating and using plugins
- ease of using existing geo-libraries
- written in non-js languages

```
  The quality of plugin creation and utilization
The extend in which existing, non-js, industry-standard libraries can be used
```

Feature C: Publicability (the ability to publish / operationalize)

A VPL should be Publicable. What is meant by that, is that scripts designed by using the VPL should be

REASON:
- VPL: application life-cycle is important: how to publish & use applications created u
- WEB: cloud-native geoprocessing: requires server-side / cloud based geo-computation interoperability:
  meaning the scripts must be able to be used without any of the interactivity features.

CRITERIA:
- ease of sharing apps as a visual program
- ease of compiling and running the app headless

3.5 Evaluation

Q: Usage: Who benefits from a web-geo-vpl, and how?

A web-based visual programming language for geo-computation as described by this study does not exist yet. This requires us to be clear about its intended use-case. However, because of this same novelty, a singular, definitive use-case can not be given. To solve this problem, this study names four possible use-cases. These cases are not mutually exclusive, but will be judged separately, based on specific criteria During assessment, we will use these four profiles and accompanying criteria to judge how well the geo-web-vpl meets up this use-case.

Case 1: Educational Sandbox

" insight within these processes are vital for communication, voor 'overdracht' the 'jonathan blow educational argument' "

- This use-case can be fully realized within the current state of geofront - "Geoprocessing for kids" - "What is a delaunay triangulation?" - "Let people play / experience / traverse a nef polyhedron" - Using something helps with understanding

Criteria

Used primairly as a test for **Criterium A**.

Case 2: Web Demo Environment

- Reproducibility toolkit: - Workflow: - Load your own code from a CDN - Build a demo setup around it - load a custom graph from a public json file - share a url pointing to this json (which contains the CDN address) - You can now share a rust / C++ program as a fully usable web demo, and analyze its performance using different datasets, test parameters, etc. - interdisciplinary exchange of ideas - MISSING FEATURE: dependency list inside of the graph.json save file

Criteria

Used primairly as a test for **Criterium B**. show that criteria B can be used to strip down the environment to purely that which needs to be demonstrated.

Case 3: End User Geoprocessing Environment

- Lightweight FME, but open source & on the web. - The tool in itself can be regarded as an end-user application: - Load file, do something with the file, download resulting file - REQUIRES WAY MORE SUPPORTING LIBRARIES AND TOOLS - Flowcharts can be exchanged by means of url's. - Future work: export flowchart to a process which can be run natively or server side.

Criteria

Used primairly as a test for **Criterium C**.

Case 4: Rapid-Prototyping Environment

– Web geoflow
– To be used within a regular software development process.
– Ravi's GeoFlow, but on the web
– Meant to visually debug a certain process, after which this process can be 'co

WHY:
– all three the previous use-cases combined: demo, test, educate yourself on your own algorithms, then operationalize the code to use it in a serious environment.

Criteria

A total test of **Criteria A, B and C**.

4 Implementation

This chapter explains the significant implementation details of the geo-web-vpl solution. This solution is titled ”GEOFRONT”, as a concatenation of ”geometry” and ”front-end”.

GeoFront exists as a set of loosely coupled repositories, all published on the version control platform GitHub under the MIT license. These repositories are grouped under the GitHub Organization `thegeofront` : https://github.com/thegeofront . The `app` repository holds the main GeoFront application. It contains the source code for most aspects, such as the basic UI, the visual programming interface, and the standard components. The `engine` repository is the library used for the 3D visualizations, some of the logic within the graph components, and some helper functions. The repositories prefixed with a `gfp` are Geofront Plugins. These plugins can be consumed by this The `GeoFront` organization also contains several repositories which are loaded as plugins, such as the wasm-bindings for startin.

4.1 App

The Geofront Application (`app`) is set up using a `TypeScript` codebase. It uses `webpack` to compile this to a singular javascript file, and this file practically represents the full application. the repository used around 9.000 lines of code, divided into the following core categories and functionalities:

I `shims`: Models to reason about ’programming language features’, such as functions and variables.

II `nodes-canvas`: Contains the model, renderer and controller of the visual programming graph itself.

III `modules`: holds all classes dedicated to dynamically loading plugins.

IV `std`: The default plugin, baked into GeoFront.

V `html`: a ’web framework’ of reusable html components making up the UI visuals.

VI `menu`: a menu system which takes care of the model & logic of the UI.

VII `viewer`: The 3D renderer accompanying GeoFront.

VIII `util`: Utility functions.

What follows is a clarification of some of these categories.

I. Shims

Since GeoFront is partially a programming language, a way was needed to reason about some of the features of a programming language, such as functions, types, variables, and modules / libraries / plugins. This is why the `FunctionShim`, `TypeShim`, `VariableShim` and `ModuleShim` classes exist, respectively. For example, the `FunctionShim` offers the name, description, number of inputs, and number of outputs of a function, as well as ways to invoke the function it represents. These inputs and outputs have corresponding `TypeShims`, which are formal representations of TypeScript / JavaScript-based types. TypeShims can be structured recursively to define a `List of List of strings` for example. The `TypeShim` is also used to reason about type checking and type compatibility.

the shims are a set of classes used in almost all other categories within GeoFront. They are located within the `modules` category, for this is where most shims are instantiated. The shim classes are designed using the Object Type design pattern (SOURCE: BIG FOUR). In short, this means that one function corresponds to exactly one `FunctionShim` instance, and that this instance is shared with any object wishing to use the function. The name `shim` was taken from `wasm-bindgen`'s (SOURCE) naming convention for the auto-generated `JavaScript` and `TypeScript` files. These classes were first created as a representation of those shims.

II. Nodes Canvas

From the Shims, the core of the visual programming language can be constructed. The `nodes-canvas` category contains all logic & visualization of the visual programming graph, and writing this category was equivalent to developing the visual programming language itself. The other categories can be understood as auxiliary to this category.

Representation

The architecture of this GeoFront's visual programming language is first and foremost a Model View Controller setup. The Model is at its core a Directed Acyclic Graph (DAG). This DAG is an object-oriented, graph-like representation of the data flow of a regular programming language.

This architecture is implemented using multiple classes. The Viewer and Controller are represented by the `NodesCanvas`, which has access to a `Graph` Model, containing `Nodes` and `Cables`.

Nodes are analogous to the "Functions" of normal programming languages. As such, a Node knows about the function they invoke through a `FunctionShim` reference. The node contains a number of input and output sockets based on this information, and each socket contains exactly one optional reference to a `Cable` (SEE PICTURE). As the name implies, these Nodes form the nodes of the DAG. However, they differ from a pure DAG implementation, in that they also provide pointers back in the reverse direction, forming essentially a normal graph, or a doubly linked list.

The Cables are GeoFronts analogy to normal "Variables". Cables know about the type they represent through a `TypeShim`. A Cable must have exactly one origin, which is an output socket of a Nodes, and must have one or more destinations, which are the input sockets of other Nodes (SEE PICTURE).

From these ingredients, a graph can be constructed. To reason about this graph as a whole, the `Graph` class was introduced. It contains methods and functions to add or remove nodes, to add and remove cables between nodes, to parse the graph from and to a json, and to calculate the graph. This study chose to centralize most logic to this Graph, instead of adding complex logic to individual Nodes. This centralized approach was deemed more clear: Many of the required functionalities cover multiple nodes, and this is better presented from an "overview" perspective of theb Because of the way Cables and Nodes reference each other, the graph has characteristics of a doubly linked list data structure. Using normal references in these types of situations could easily lead to memory management issues such as Dangling Pointers. Even though the JavaScript runtime is garbage collected, it is still subject to memory leak issues or runtime errors. Substituting references with ids makes it easier to prevent these types of problems.

Finally, this entire graph object is accessed by a larger `NodesCanvas` object, and this canvas has the responsibilities of "View" and "Controller". More?

Rendering

This nodes data model must be rendered to the screen so users can comprehend and edit the graph. This visualization is achieved by using the HTML5 Canvas Api. The Canvas API is a raster-based drawing tool, offering an easy to use, high-level api to draw 2D shapes such as lines, squares, circles, and polygons. The Nodes Canvas uses this API to draw polylines and polygons at runtime, to represent the cables and nodes respectively. These basic shapes and their styles change dynamically, based on features like the length of a cable, how many input sockets a node requires, or whether or not a node is selected.

Like other HTML5 features, the main advantage is that this API is included and implemented within the browser itself. This method is fast thanks to its C++ based implementation, and can be used without the need to include anything within the source code of the application.

The disadvantage of the Canvas API is that it uses many CPU-based draw calls. it is primarily CPU based, Making it much less fast and efficient than WebGL for instance, which is based on the GPU. This is quickly noticeable within interactive applications which need to redraw often, such as GeoFront. The Canvas is refreshed and redrawn often, and does not distinguish between unchanged and changed features. This comes down to a performance linear to the amount of nodes and cables drawn. For the current implementation and scale of geofront, this performance is acceptable.

Style

Special care has been put into the stylization of the graph. The visuals take inspiration from various programming languages, such as Blender's GeometryNodes, McNeel's Grasshopper, and Ravi Peter's GeoFlow (SOURCES). A few notable exceptions, however. Firstly, the entire graph is placed on a grid, and all nodes strictly adhere to this grid. For example, a node with three inputs will always occupy three grid cells in height. This grid is applied for much of the same reason as terminals & source code are displayed using monospaced fonts. Consistent sizing encourages organization and clarity, for this makes it easy for components to line up, and predict how much space something requires. Cables also adhere to the grid. They

alter their shape in such a way to remain as octagonal as possible, as an attempt to make connections between nodes more readable. This takes some additional inspiration from subway maps, as well as the design of computer chips. This makes for a good fit, since both these spatial configurations and the Geofront Graph are focussed on clear connectivity.

Interaction

User Interaction is made possible through the HTML DOM Events. This api provides ways to listen to many events, including keyboard and mouse events. When the mouse is moved, its screen-space position is transformed to a grid position, which allows the user to select one or multiple objects. GeoFront's user interface strives to match features of regular desktop applications. As such, the Geofront Graph supports features like undoing, redoing, duplication, copying, and pasting. These functionalities can be used with the expected keyboard combinations (Ctrl + C / Ctrl + V).

Graph Manipulation

In order to support these features, especially undo / redo support, we are required to explicitly track the history of the graph. This is why a Command Pattern (Big Four) was implemented. Instead of directly editing the graph, all actions are represented by `Action` objects. Each Action can 'do' and 'undo' a specific action, and the data needed to make this do and undo are stored within the action. By then introducing a bridge class we decouple the graph and controller, only allowing interaction with the graph by giving this bridge Action objects. The Bridge maintains a stack of undo and redo actions, which represents this history.

Additionally, editing any graph data structure is never trivial. Special care must be taken to ensure the validity of a graph before and after changes, and is especially the case with GeoFronts' Doubly linked graph. To the best of the authors knowledge, these is no "trick" or "pattern" to ease this. Instead, the Graph and Graph Decoupler classes both have to be designed in such a way

Calculation

When regarding the geofront graph, or any other programming language, we see many functions requiring variables which are the result of other functions. This is why a graph like this can also be called a dependency graph. If we wish to calculate the result of the graph, these dependencies must be taken into account. We must sort the functions the graph in such a way that all dependencies are known before a function is calculated. This problem is known as a topological sorting problem, and can be solved using Kahn's algorithm:

```
Step -1:
  Make an 'order' list

Step 0:
  Make a 'visisted' counter, initialized at 0

Step 1:
```

```
  Make a 'dependency' counter for each node, initialized at 0

Step 2:
  Add 1 to this counter for each input edge of this node.

Step 3:
  Fill a queue with all dependency 0 nodes. These are the starters.

Step 4:
  Remove a node from the queue (Dequeue operation) and then:
  add the nodes' id to the 'order' list.
  Increment 'visisted' counter by 1.
  Decrease 'dependency' counter by 1 for all dependent nodes.
  If one 'dependency' counter reaches 0,
    add it to the queue.

Step 5:
  Repeat Step 4 until the queue is empty.

Step 6:
  If 'visisted' counter is not equal to the number of nodes,
    then the graph was errorous, probably cyclical.
```

This algorithm has several important qualities.

...

If all intermediate steps are cached, this same algorithm can also be used for performing partial recalculations of the graph. The starting positions of the algorithm then simply become the altered parameter, after which only the required functions will recalculate.

III. Modules

The name "Modules" is also used to refer to plugins within the context of geofront.

Shims are automatically created

explain the system to dynamically load plugins at runtime

The Dynamic nature of javascript makes it possible to load plugins at runtime.

4.1.1 Plugins

What does a plugin need in order to be acceptable one javascript file, one type definitions file, and one optional wasm file are accepted.

ts -¿ normal build works out of the box rust -¿ wasm pack compile target web -¿ works out of the box c++ -¿ emscripten -¿ more manual work needed

Features

- IO can use: - basic types (boolean / number / string) - jsons, objects, interfaces (typesystem will pick up on them) - ArrayBuffers (vital for performant data transfer)

4.1.2 IV. STD / The Standard Library

The standard library is a 'baked in plugin'. Points, lines, triangles, vectors, etc. -¿

Shims are manually created -¿ eventually, large parts of the STD can become a plugin

4.1.3 V. Menu

Web Framework

Every web application requires some amount of html and css to make up its content, and GeoFront is no exception.

`html`

- No Framework - Web Components - Typesave Events -¿ needed, because of the Lapsed listener problem. - Webpack + Typescript

"... WebComponents have not been used just as a curiosity. The Graph is intended as a sequence of containerized processes, which whould match WebComponents containerized nature perfectly, just like it matches WebAssembly ..."

with these html components, the basics of the webapp can be created: menu's side menu, and the two main canvases: the graph canvas, and the viewer canvas GeoFront's use-case is, however, slightly different. We wish to use html as a type of IMGUI. Several cur

4.1.4 VI. Viewer

Finally, the VPL requires some way of visualizing 3D geometry.

explain how the viewer was created from scratch

lightweight three.js clone

supports basic things

fast, due to direct buffer types

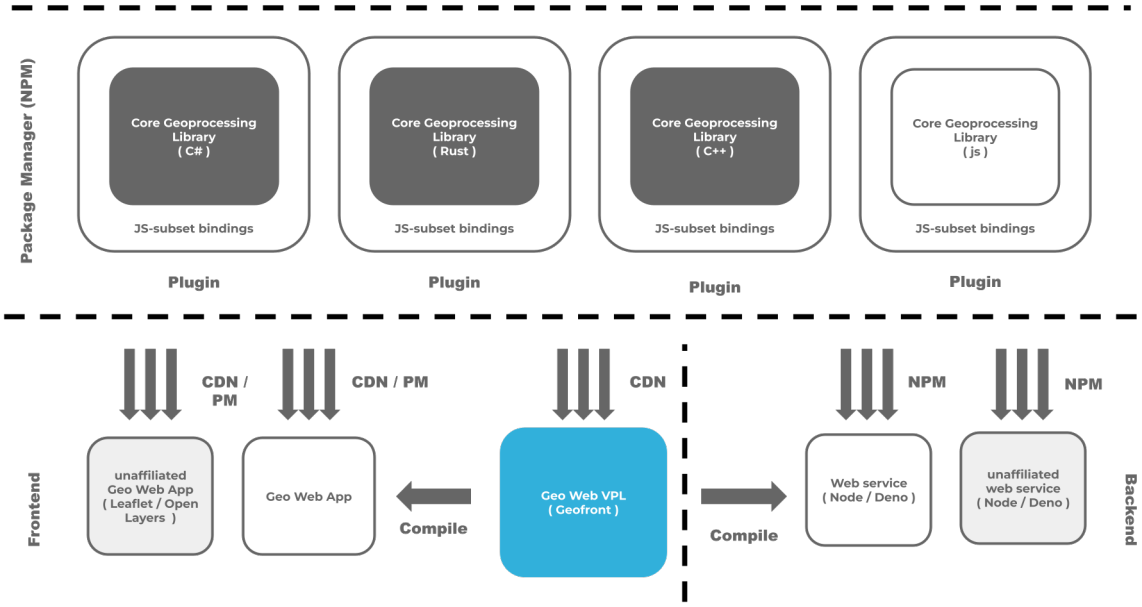


Figure 4.1: Plugin Model

4.2 Engine

[be brief about this]

The `Engine` is a typescript library of about 20.000 lines of code.

It can be used both as a header-only library, or it can be compiled to a normal javascript module. Header only is an unusual

- Various functionalities, not only created for GeoFront. - A custom engine was used, so it could be specialized to the needs of geoprocessing. - Three.js inspired, but very strip-down

- show webmap demo (maybe) - show

4.3 Plugins

The last implementation feature which requires elaboration is the plugin system. A full overview of this system can be given

Using this system.

A system for

...

Using this

Startin

- web assembly based

CGAL

5 Experiments

5.1 Use Case: Educational Sandbox

WHAT:
- show the behaviour of a simple ransac algorithm, fitting a plane through a point cloud
- show it beharivourly: show in-between steps
- make parameters ajustable (number of high scores, minimum high score, number of tries, etc.)
- add least squares adjustment, and compare.

SIMILAR EXAMPLES:
- the geometric predicates explanation
(https://observablehq.com/@mourner/non-robust-arithmetic-as-art)
-

ASSESS ON:
- educational value
- ease of usage (the promise of **Criterium A**)

ASSESSMENT (hypothesis):
+ indeed very insightful for analying the behaviour
  and operation of certain
  algorithms & parameters. Not many applications can show
  this level of insight.
+ feature B can be used to strip the tool down
  to the bare minimum, helping
  with not overwhelming the user with features

- This VPL is not easy to operate.
  It remains difficult to communicate what needs to
  be done, how things work. This is not an expert tool,
  but also not a beginners' tool.
- No built-in tutorialization
- hard to discover the code underneath,
  obfuscating the link between process and code.

5.2 Use Case: Web Demo Environment

```
WHAT:
 - Show the startin delaunay triangulator
 - Accept user-submitted Laz files as input
   - filter the ground
 - Accept a randomly generated point cloud based on perlin noise.
 - Visualize the generated mesh, and make it available for download

EXAMPLES:
 - hugo's demo

 ASSESS ON:
 - **Criterium B**: extendability:
   - how does foreign and native codebases interact?
 - clarity
 - reproducibility
 - performance
 - scope (raster / vector / 2D / 3D)

JUDGEMENTS (hypothesis):
 + Clarity is fine
 + Reproducability is
 + Performance is decent

 ~ clarity is fine, vpl allows users to 'play around' and try
   different configurations, even run their own data through the
   demonstrated functions

 - Application is less suited for 2D
```

5.3 Use Case: Geoprocessing Environment

```
WHAT:
  - Query an area of a point cloud with a polygon
  - turn that area of points into a triangulation
  - turn triangulation into isocurves
  - save this as a geojson
  - turn this whole thing into a function, which takes a PC,
    polygon and isocurve range, and spits out the geojson
  - share this using a link
  - turn this into an app?
  - turn this into a script?

EXAMPLES:
 -

 ASSESS ON:
```

```
- **Criterium C**: Publicability:
  - the ability to operationalize the application
  - can it be used by end-users? clarity? too much clutter

JUDGEMENTS (hypothesis):
  ~ sharing by link is possible, but for end-user usage,
    its very cluttered
  ~ compiling to js only partially works
  ~ compiling into an app is not possible, but
    since everything runs client-side, it could be implemented.
```

6 Discussion

The qualitative and quantitative measurements have a degree of subjectivity - research topic is sizable -

7 Conclusion

In this article we described the design, creation and evaluation of GeoFront, a web-based point-cloud processing tool.

...

Geofront has been created as an experiment to explore if visual, browser-based geo-computation can make geo-computation in general more accessible. The advantages and disadvantages of browser based geo-computation, compared to native or server-side geo-computation, are examined in several scenario's. Both quantitative indicators, like loading and runtime performance, as well as qualitative indicators, like the fitness for an intended use-case, are measured in each of these cases. This study concludes that based on these measurements, browser-based geo-computation is fast enough that it can enable many promising use-cases, such as on-demand geodata processing apps, educational demo apps, and code sharing. However, extensive user-group testing is required before any definitive statements on accessibility and fitness for geo-computation can be made.

...

Contributions:
– a web based visual programming language for geometry computation.

7.1 Answer to main research question?

???

8 Post-Conclusion

This is a dumping ground for text which should belong in a 'reflection' chapter or an 'on the future' chapter. In any case, not included within the main body of the thesis.

8.1 Personal Motivation

During my internship I was tasked with creating a parametric 3D CAD model.

- local usage -¿ quick, direct feedback

- We needed to make this a product for end users.

- Industry-standard choice: cloud -¿ smart-server dumb-client setup, cloud-native architecture

- Problems -¿ continuously downloading new resulting CAD files after every change created a lot of web traffic. -¿ slow, not at all the same experience. -¿ cloud host was even more slow in cold-start scenario's -¿ cloud host monetization scheme: pay for every time the script runs, -¿ meant that consumers had to be discouraged to 'play around' with the tool too much.

This made me question the cloud-based paradigm, at least for the use case of calculating geometry by end users.

At the same time, many of our parametric designers could use grasshopper, and only grasshopper.

This led me to think about a vpl which can run client-side in a browser, and can produce client-side applications.

if the data is geodata or CAD data, does not matter besides the fact that geodata is often big data.

8.2 On the future

8.3 Auxiliary Objectives

8.4 Motivations

Motivation 1: Client-side Geoprocessing

Despite the popularity of geographical web applications, the range of actual gis abilities these applications are capable of is very limited. **geoprocessing!** (**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, whereby mentionned as bbg, 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?*

se concerns represent the three main obstacles preventing a smooth, widespread adoption of bbg.

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.

Motivation 2: Accessible Geoprocessing Libraries

Most industry-standard geoprocessing libraries such as CGAL are difficult to use by anyone but experts in the field. A steep learning curve combined with relatively complex installation procedures hinders quick experimentation, demonstration, and widespread utilization of these powerful tools. It also limits the interdisciplinary exchange of knowledge, and compromises the return of investment the general public may expect of publicly funded research.

Geofront could improve the accessibility of existing geodata processing and analysis libraries, without adding major changes to those tools, by loading webassembly-compiled versions of them, similar to [other web demo's](todo).

Fair Geodata Processing

-¿ OGC: Fair Geodata -> FME : data integration tool -> Grasshopper: 3D CAD: visually creating and debugging geometries L -> Used within the field of AEC for solar analysis, routing, permit checks. These visual programming environments Due to this need for [X], the goal of this thesis is to develop a method for geo-computation in a browser-based front-end.

Explore Web Capabilities

- HTML5 with WebGL and Canvas Api -¿ Celsium, Leaflet, D3 - javascript runtime improvements -¿ shift from backend-heavy to frontend-heavy web applications - http range request -¿ cloud native geospatial movement - WebAssembly -¿ ...

WebAssembly is a potentially revolutionary technology. We don't know yet what it fully means or what it can do, so we should explore.

Many things where discovered as side effects of the main goal of this thesis. We wish to discuss these other aspects here.

8.4.1 Improvements to FME & grasshopper

If this study would have stated to create a FOSS, web based alternative to FME or Grasshopper, that would perhaps be enough to base a thesis on.

-¿ FME: run scripts on the server -¿ Grasshopper: (industry sollutions) Shapediver, White Lioness,

-¿ The fun thing is that GeoFront has been created with this perspective from scratch. This could lead to a much more clean implementation

8.4.2 FAIR geodata Processing by using browser-based, front-end geo-computation

8.4.3 Exploring WebAssembly

8.4.4 Exploring Client-side geo-computation

8.4.5 Notes

/ experiment to assess: - The fitness of the web in general for client-side geo-computation - If new features of modern browsers mean anything for the field of geo-informatics at large - The topic of accessible geoprocessing.

now, answer this to the best of your ability

Many considerations

While conducting this research, I came across various key insights from various studies, and there seemed to be a link between them

Most important effort I saw is the "denichification" of the geospatial world. - Hugo's keynote - Linda van den Brink's PHD - cloud-native geospatial

8.5 FAIR

observation: - a lot of attention for FAIR geodata, linking of geodata, using common standards, etc.

BUT: We don't seem to put as much emphasis on FAIR geodata *Processing*. - The means to filter a dataset, to process it or convert it to a required format or CRS, could become more FAIR

We focus on Accessibility accessibility

-> the fact of being able to be reached or obtained easily: *Two new roads are being built to increase accessibility to the town centre.*

-> the quality of being easy to understand: *The accessibility of her plays means that she is able to reach a wide audience.*

...

- The front-end browser technologies are a vital component of the modern geospatial software. - Like how the entire cloud-native moment is only possible because of the HTTP range request feature.

THE MAIN THING I WOULD LIKE YOU TO GET AWAY FROM ALL OF THIS: - a vital component of the cloud-native geospatial moment is the "HTTP range request" web feature, and chris said as much. - this feature has been out for some time (html1/1,) - What I'm saying, is that we have a whole range of similar, 'game changing technologies' recently added to web browsers, and I have a feeling these features could be the birthing grounds for new, ground breaking ideas and movements of ideas.

We have not fully envisioned these new trends, nor do we have a catchy, powerful name such as *Cloud Native Geospatial*, But I have no doubt that something revolutionary will come of this.

Nevertheless, I will attempt to name and envision a trend from these technologies. *"FAIR Geodata Processing"*.

Vision: - Portable, cross-platform, binary geoprocessing libraries, which can be used on the cloud / on servers, natively, and in the browser, without any changes. -> we can use that to build standards for geodata processing itself. Every `GP` library interoperable with every other library, at least on a language and package manager level. - This also eliminates the need of platform specific plugins (QGIS plugins, ARCGIS plugins, Blender Plugins, 'web plugins'). - This could lead to a generalized geoprocessing library portal like NPM / cargo / WAPM with an attached content delivery network, Or these infrastructures can just be utilized, with just an UI sprayed on top.

- I am aware that these types of efforts have been attempted many times before, but WebAssembly might be a missing link

-> webassembly has a good balance between portability and performance.

->

If I were to attempt to name this trend, I would

8.5.1 Use Cases of Browser Based Geoprocessing

- For which use-cases might such an application be beneficial? (Old Phase 4) - Name all reasons for browser based geoprocessing, and the 4 audiences.

Discuss

8.6 Reproducibility

Reproducibility Results themselves are insanely reproducable. Software can be used, you can reproduce results easely by dumping versions of geofront in a folder.

...

Software can also be build without too many difficulties, but the procedure has some unconventional build steps:

...

BUT, the code is not the cleanest, nor the most conventional. minus points on open-source accessibility.

8.6.1 Environment

- github organization - repo for engine - repo for app - repo for each plugin - build procedure

8.6.2 Usage

basically, write a tutorial

This is how one can use Geofront

8.6.3 Creating & Using your own code

show the insane (rust + wasm + npm) workflow

Locally: 1. Write a geoprocessing / analysis library using a system-level language (rust, C++). 2. Expose certain functions as public, using 'wasm-pack' or 'emscripten'. 3. Compile to '.wasm' + 'd.ts' + '.js'. 4. publish to npm (very easy to do with wasm-pack, can also be done with emscripten)

Alternatively: 1. Write a library using typescript, 3. Compile to 'd.ts' + '.js'. 4. publish to npm

In Geofront: 4. Reference the CDN (content delivery network) address of this node package.

Congrats, this is now a publicly accessible geofront plugin! The library is loaded, A component is created for each function, with inputs for input parameters, and a singular output. If

a 'typescript tuple type' is exported, the plugin loader will create multiple outputs according to each component of the tuple.

8.7 Limitations

Limitations of usage right now

8.8 Future Work

Boulevard of broken dreams :)

This thesis has only scratched the surface of all that

- Geofront as web-based version of grasshopper - Geofront as lightweight QGIS replacement

8.8.1 App Features

Constrain plugin support

- Its becoming too hard to cater to both. The more constraint plugin support becomes, the more powerful and well-integrated we can make the plugins - It would be really nice to make rust the only way of creating plugins. - Build a separate plugin loader for C / C++, or only support those through C++ bindings (dont know if wasm-pack can handle C bindings)

Adding first-class type support to plugin types, by creating a Rust crate

We couldn't do this, because we wanted to cater to .ts, .js and C++- based plugins.

8.9 Reflection

One of the goals of this study was to investigate and explore browser-based geo-computation, and there are many ways of conducting exploitive studies. This study chose for a practical approach: investigation by means of creating an application. The advantage of this approach is that it leads to tangible results.

The disadvantage of this approach is that software development can lead a study astray, if the development needs of the application are put before the needs of the study itself.

8.10 Design Choices

8.10.1 From First Principles

A part of this study is gathering a clear picture of the capabilities present in the browser itself. We do not perform a study on the capabilities of the *Ecosystem*. it aid the results of this study if they can be detached from specific frameworks. This study has therefore developed GeoFront with minimal dependencies in mind, and does not make use of

8.10.2 Web based

- accessible - immediately usable -¿ no installation - cross platform - easy to integrate with end-user applications (often web applications). - easy maintainability (just update website, no need to distribute installers)

- one-of-software argument

- makes conceptual sense for end-users with certain applications: - "You download something from the internet by using an internet browser".

The "one of" software argument: QGIS is excellent for users who use it daily or at least weekly.

(use the QGIS user data you found)

BUT, users who want to access and process geodata *once in a while*, you ideally want something more temporary. Web Applications make more sense in this regard: No updates, no background processes, no 'presence' on the machine itself. Just go to the website, do what you need to do, and close the browser again. Similar to webshops.

This is in addition to the obvious advantages, like no need to install, easy maintainability, and cross-platform distribution by default.

Finally, using the web ensures that the code will run on all devices: native, mobile, desktop, IOT devices

Accessibility - Findable == Accessible - Reproduce & validate research results - Interdisciplinary exchange of ideas - Educational Web Demo's - "Getting a feel" for data - Before / After - Hands-on experience

¡!– donald knuth argument: by keeping geodata raw, and posponing the consumtion of geospatial data to the last possible moment, we can –¿ Reduce web trafic

8.10.3 Meant for generic 3D usage

Sorry for this rant, this will be more nuanced

Today, we see the need for collaboration between CAD, BIM and GIS. Entire industries (Speckle, FME) have been introduced to bridge the gaps.

All three of CAD, BIM and GIS want the ability to join solids together, desire to give certain spatial objects metadata, and want to run automated workflows in the cloud. These Individual fields are constantly reinventing features the other fields have already figured out. BIM is starting to open up to the idea of streaming only a part of the building instead of the whole

thing, something which the GIS world has been doing for years. On the other hand, GIS is only now starting to make the transition to 3D, a transition not unlike to how BIM is replacing 2D CAD in the AEC industry.

We will allow geofront to be fully customized by different plugins. By unloading all GIS plugins, and adding all BIM plugins, we turn GeoFront from a GIS to a BIM tool.

8.10.4 Visual Programming interface

The choice for a Visual Programming Language(vpl) is made to further explore this idea of accessible geoprocessing.

demonstrate the advantage of making a geoprocessing tool web based, and thus potentially accessible to a larger audience. Using visual programming, the geoprocessing sequence can be altered on the fly, and in-between products can be inspected quickly, as both data and in a 3D viewer. 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.

To enable the interplay of the following three features:

1. Alter the process without recompiling 2. Use UI to quickly and easily alter input data. 3. Visualize in-between products in 3D.

Each of these steps is individually possible with regular programming. Feature 1 can be achieved using hot-reloading. For feature 2, a regular GUI debug menu can be used. For feature 3, we can write and save in-between products, and open them up a 3D viewer of choice.

What makes a VPL special, is its ability to seamless integrate all three of these aspects, and allow interplay *between* these aspects.

... VPL's pop up all the time in all fields of computer science, But they intent to stick within 3d applications. (blender, rhino, unity, unreal)

Why? - a need for 3D visual debugging. - arbitrary parameters that require to be 'toyed' with, aka, to find a certain balance interactively and empirically. - Inverse distance weighting - Tolerances - Size of smoothing kernel (can also be achieved with settings panels)

... - Visual Scripting on the web - "A geodata processing sequence is often conceptualized as a pipeline. Then lets make it an actual pipeline. " -

this is still informal

Scalability

Geodata is big data. Will this web application be scalable to handle big datasets?

One of the problems to address when considering the ergonomics of geodata geo-computation, is the fact that geodata is almost always big data. A web application cannot be expected to process huge datasets. So how does geofront address this fundamental aspect of geoprocessing?

First, lets give the devil it's due. - Even when processing "smaller" datasets of, lets say 4 GB, most of the 'flowchart niceties' of geofront will cease to be useful. Inspecting this data will take more time than its worth, and reconfiguring the flowchart will take a long time. This can be mitigated by using web workers, but it will still not be very ergonomic to work with. - This is why performance is everything within geomatics.

BUT:

- A case can be made for on-demand, browser-based geoprocessing.

- Even when we want to write a tool to deal with large datasets, we often test and develop this tool in a smaller context, with a smaller dataset first. The same thing is possible with geofront:

- Geofront is mostly meant as a sandboxing tool for experimentation: An environment try out different procedures, parameters, and different datasets.

- The flowcharts created with geofront are compilable to javascript. this allows any processing operation created with geofront to be executed from the command line using node.js. This is a way of how geofront can integrate with large-scale geodata pipelines.

The point is that even if we use cloud-based computation, we still want to be able to be able to ergonomically and correctly configure these geoprocesses. Geofront could still assist with that.

BUT MOREOVER:

The possibility of client-side geoprocessing also allows for an entirely new geoprocessing workflow, which could replace some use-cases that now require big-data processing and storage. Instead of storing big datasets of pre-processed results, by using client-based, on demand geoprocessing, an application could take a general big-data base layer, and process it on-demand, with a scope and settings determined by the end-users.

This type of *Process Streaming* is certainly not a drop-in replacement for all big-data use cases. But, in cases which can guarantee a 'local correctness', this should be possible. Examples of this are a delaunay triangulation, TIN interpolation or image filter-based operations. This could be a more cost-effective outcome, as server farms & Terabytes of storage are time consuming, expensive phenomenon.

8.10.5 WebAssembly

Why WebAssembly? to complete the major thing geofront set out to do: making low-level scripts accessible on the web.

To allow for the previous two (VPL + WEB) without a compromise to speed

On its own: WebAssembly is useful for being containerized binary code. - Binary: WebAssembly is close to machine code, making it very performant. - Containerized: the main advantage of WebAssembly over normal binaries is security. wasm can be reasoned about in a virtual, containerized manner, since it uses virtual memory and a system of incremental privileges. WebAssembly binaries cannot access memory outside of its designated memory pool, making segmentation errors harmless. The incremental privileges also ensure that binaries cannot access anything the user did not explicitly allow for.

Taken together, this makes WebAssembly a more secure alternative to regular binaries. This is also why browsers added support for WebAssembly, but not for regular binaries: Adding support for regular binaries would be a substantial risk to the security of all internet users.

Only core components

Why not build everything as a local application, and publish the entire thing as wasm?

That would be: - more performant (probably) - Better native experience - Better compilation to standard executable

BUT: - The current setup allows for javascript interoperability. - This is useful for the purposes of UI, GUI, Web requests & Responses, jsons, WebGL. - These are all aspects that would have needed to be part of the C++ application, that we now get 'for free', since the implementation of these features are present within the browsers of clients. - javascript can now also serve as its scripting language, making custom, scriptable components a possibility.

8.10.6 Minimal Dependencies

Goal: assess raw web technologies, not the web ecosystem.

1. Minimize dependencies. - Maximize usage of standard HTML5 features. - We want to access core web technologies, not the javascript ecosystem, thats a whole different question. - We are also under the presupposition that the less this project depends on existing project, the more portable this project, or portions of it, will become.

2. Separate geoprocessing tools into plugins as much as possible: - ideally, if you are not using rasterization tools: do not load rasterization tools. - This means: Divide all needed functionalities up in plugins. - Then load these plugins lazily: only when needed.

- This also aids the purpose of geofront: Making low level code accessible.

8.10.7 application design

Nielsen and Molichs 10 User Interface Design Guidelines

1. Model geofront after a 'normal' desktop application. - Make users forget that they are looking at a website - Undo / Redo support - Cut / Copy / Paste support

user interface strives to be as 'normal' as possible within its constraints and features. its what users have come to expect

8.11 3D, Point-cloud focussed geoprocessing

The adoption of COPC could mean the same for point cloud data, but it remains unknown what accessible analyze, edit, and visualize means for point cloud processing. How to present the complex endeavour of point-cloud processing in the format of an accessible web application is unknown.

Bibliography

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

Community, Q. (2022). QGIS Homepage.

Esri (2022). ArcGIS Homepage.

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

Mozilla (2013). asm.js.

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

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

Colophon

This document was typeset using LaTeX, using the KOMA-Script class `scrbook`. The main font is Palatino.