# Client-side geo-processing using WebAssembly and Visual Programming : Context Definition


| | |
|---|---|
| name | Jos Feenstra |
| nr | 4465768 |
| email | feenstrajos@gmail.com |
| date | June 11, 2021 |

# Meta

This document is a very early draft for the eventual thesis. It aims to frame the exact problem this thesis will tackle, and to pose a sketch of the solution. It also offers a critical reflection on the proposed solution to this problem in the form of a Q&A.

# Problem Statement

## Problem 1 : Client-side geoprocessing is underdeveloped

Geodata processing client-side in a browser is very underdeveloped. This is a huge inhibition for many applications. Client-side geoprocessing offers tremendous potential:

- Users will never have to install anything except a web-browser.
    - This will make geoprocessing more accessible & operational to a large, non-geodata expert audience. It allows more people to do more things with geodata, and reach more interesting
      conclusions quicker.
- Client-side geoprocessing would offload postprocessing to the end user, allowing the end user to tailor geodata to their exact needs.
- Client-side geoprocessing allows direct user feedback unlike server-side geoprocessing. Users can be on top of the calculations, look at in betweens steps, etc.
- Instead of having large, preprocessed datasets, geodata could be processed on demand from the source. If a user is only interested in a small area of the source dataset, this could save vast amounts of time, storage space and computational resources.

The last point gives client side geoprocessing both an ecological & economical reason. The accelerated effects of climate change gives geomatics experts a moral reason to avoid huge render farms and other power consuming methods whenever possible. Additionally, client-side can also be much cheaper, since all of the processing and rendering will happen on the machines of the user, and not on the servers of the organization.

> [JF]: underdeveloped in which way?
>
> the client-side software ecosystem is run by `javascript` and various languages which compile to `js`, such as typescript. this ecosystem lacks powerful geoprocessing tools found in python and C++ ecosystems like `cgal`, `geopanda`, `3dfier`, etc. Javascript is also not a very fast language compared to C++. So, developing these tools from scratch in `js` would be both a lot of redundant work, and probably not very effective.

## Problem 2 : Geoprocessing overall lacks ergonomics

Geodata processing is very unfriendly to non-experts:

- It is a notoriously hard endeavour. You pretty much require an expert.
- Software is often hard to find, sometimes hard to install.
- Software is often difficult to operate.

To expand upon that last point. experts often solve specific geo-processing problems by creating cli tools. This is fast, and makes sense from a programmer's perspective, but has a few drawbacks in terms of ergonomics:

- Strong relationship of problem & solution.
    - Hard to scale or repurpose.
    - Difficult to parameterize. (settings.json / settings.yaml)
- Native Code: Not trivial to make cli tools cooperate with other environments.
- difficult to debug
    - Often hard to track how algorithms are behaving, due to the difficulty of visualizing in between steps.
- 'relatively hard' to share methodologies among colleagues.
    - It takes a while to clone a repo, get all dependencies right, build the project, get the exact same dataset the colleague used, possibly setup the right databases and fill those with test data, etc.
- relatively hard to 'publish' the tools to end-users.

> [JF]: The main problem with Qgis is how ui heavy and static it is. People often quickly reach for other appoaches such as python + geopanda's, to have a more direct relationship with the geodata, and to create processes which can be reused, rather than to create one map, and one map only.

# Solution

> [JF]: I need to rephrase this. Not : *the goal is to build an application* But: *the goal is to explore the possibility of improving client-side geoprocessing*

This research intents to solve both problems by researching and developing a new geodata processing application. The first problem will be solved by making the application run in a web browser, and by using `WebAssembly` to make this environment access the libraries needed for professional geoprocessing. The second problem will be solved by making this application operable by means of a visual programming language. The aim of this research is to give give both geomatics experts & non-expert users alike an effective and ergonomic way to process and visualize geodata client-side in a web browser. The research question is: "*How to create an `effective` and `ergonomic` web-geo-VPL?*"

The qualities `effective` and `ergonomic` are important to re-emphasize. They refer to the two problems this research simultaneously attempts to tackle. This is also reflected in the two main categories of sub-research questions. This duality is a necessity since both the two problems as well as these two aspects are interrelated (the Q/A will cover more).

The most direct way to answer the main question is to practically design and create the type of application suggested, and analyse its effectiveness and ergonomics afterwards. Many design decisions will occur, and these will be documented within the written thesis.

# Research Question

*"How to create a web-geo-VPL which is both `effective` and `ergonomic`?"*

## sub questions

> JF: most of these are design decision questions. Don't know if that is a good or bad thing

- A : Technical Aspect
    - How to use wasm to compile an existing geoprocessing library?
        - and to make it ready for web usage
    - When is `javascript` faster, and when is `WebAssembly` faster?
    - How to design a `wasm` plugin ecosystem?
        - Marketplace Website ?
        - Npm ?
    - How to handle types between multiple, unrelated `wasm` libraries?
- B : Ergonomic Aspect
    - What are existing ways of geodata processing?
        - *Who* uses *What* and *Why*?
    - What lessons can be learned from existing VPL's?
        - How do geomatics users use existing VPLs?
        - How would geomatics users **want** to use a VPL?
    - How to design a new VPL?
        - What needs to be different for a web vpl?
        - What needs to be different for a vpl consering geomatics activities?

# Q/A

## Why a Visual Programming Language (VPL)?

I would argue that VPL's offer a perfect UI for geodata processing. Code is only a process. VPL's can have input, process and output within the same environment. This poses an advantage for geometry processing, or other coding with a strong visual component, such as textures or shaders.

> [JF]: it also enforces a functional programming style.

example:

- input:
    - draw a polyline on a map
    - slider to test a range of values
    - directly use a WFS / WMS service
- process:
    - **iteration**: visualize in-between steps easely
    - quickly demonstrating the effects of certain parameters
    - quickly demonstrating the result of a tweak in the process
- output:
    - visualize geometry on a map

## What can this proposed VPL do exactly?

- on demand geoprocessing
- parametric modelling similar to other geo-VPL's
- see **Applications**

## Why does this research need to solve both problem 1 and 2? Is solving one of them not enough?

I think these two problems are mutually dependent: one cannot truly solve one without solving the other. Getting the technicalities right of client-side geoprocessing is only part of the problem. One of the big reasons why anyone would want client-side geoprocessing would be the fact that it is accessible to a very large audience. If

client-side geoprocessing technically becomes available, but in the form of a 'classic' web-cli, why would anyone want to use it? what was gained by publishing this online? why not do this offline, where it is quicker, and we have better tools available? The reverse is also true: if we would dodge the web and develop a new geo-vpl natively to solve the geo-processing ergonomic problems mentioned above, then the research would be 'just' another native geoprocessing tool. then why not use one of the many other geoprocessing-vpl's and improve it, instead of making a new one?

However, the two problems taken together forms a perfect 'why' answer. The technicalities and the ergonomics enforce each other. Ergonomic must-haves give direction to what is needed technically. Technical side of things has an ergonomic dimension: If a tool is technically very complex or not completely 'foolproof', then one can either choose to still 'act' like the tool can do anything, and do a lot of behind the scenes patchwork with unexpected behavior, or one can choose to communicate these technical limitations clearly, and present it in a way which gives the end user complete control, but also makes him/her responsible in some way.

## why do we need a new geo-VPL?

To answer this question, a bit of analysis of the current geo-vpl's are needed:

### Related visual programming languages focussed on geometry:

| Name | Author | Availability | Source | Audience | Purpose | Link |
|---|---|---|---|---|---|---|
| FME | Save Software | €2,000 one time | Closed | Geoprocessing intermediaries | Geoprocessing | https://www.safe.com/fme/fme-desktop/ |
| Houdini | SideFX | ~€1,690 p.y. | Closed | 3D modellers & SFX | Procedural Modelling & Special effects | https://www.sidefx.com/ |
| Geometry Nodes | Blender | Free | Open | 3D modellers & SFX | Procedural Modelling & Special effects | https://docs.blender.org/manual/en/latest/modeling/geometry_nodes/index.ht |
| Grasshopper | David Rutten / McNeel | €995 one time | Closed | 3D modellers & architects | Parametric Design | https://docs.blender.org/manual/en/latest/modeling/geometry_nodes/index.ht |
| geoflow | Ravi Peter | Free | Open | Geoprocessing experts | Geoprocessing: Rapid prototyping & Visualizing in between steps | https://github.com/geoflow3d/geoflow |
| Dynamo | Autodesk | +revit €3,330 p.y. | Semi-open | Expert Revit Users | BIM automation | https://dynamobim.org/ |

Drawbacks of all of these: They are not FAIR:

1. not `findable` & `accessible`: most have a large barrier of entry: the tool needs to be installed \ Requires an account \ Requires several thousands of euros per year.
2. not `interoperable`: plugins must always be specifically created for these environments. no way of directly using a repo, like you can with a regular programming language.
3. not `re-usable`: all vpl's are strongly tied to their host environment / no way to turn a script into a CLI or web-app.

> [JF]:
>
> idea to fix (2): wasm as plugin: just a collection of functions / classes / etc.
>
> - automatic js library from wasm builder
> - updating a list of all available plugins idea to fix (2): direct exposure: make all functions found within wasm binaries callable from visual programming canvas. with these two ideas, there is no need to specifically create plugins, any wasm binary can be utilized.

> idea to fix (3): make the flowchart compilable to javascript, or an IFC-like json format. make the flowchart loadable from javascript or IFC-like json. make the flowchart sharable using a link / hash.

> comments on why ravi created geoflow: (expand upon this later!!!)
>
> - rapid prototyping
> - fast visualizing of in-between steps
> - why c++? -> fast & many libraries.

## So: why do we need a new geo-VPL?

So why why not grasshopper / FME / ravi's geo-flow? Three improvements need to be made to solve the drawbacks mentioned above:

- (1) To solve `findable` & `accessible` -> **Web Based**
    - No need to install anything.
    - Instantly sharable with others.

    - Context-agnostic: desktop / mobile / web-client / server

- (2) To solve `interoperable` -> **WebAssembly**
    - Language-agnostic architecture:
        - plugins can be created using C++, Rust, and anything compilable to WebAssembly.
    - Wasm binaries can directly be loaded as plugins, no / minimal wrapper code needed
    - these libraries can be mixed

- (3) To solve `re-usable` -> **(Java)script interoperability**
    - Define the graph in the form of a textual script (subset javascript).
    - Graph can be saved to textual script form.
    - Graph can be loaded from textual script form.
    - Script can be run without a VPL environment, (because it is just javascript)
    - Graph has a one-to-one relationship to javascript.
        - One Operation / Node has a one-to-one relationship to a javascript function.
    - Enables the best of both worlds.
        - js: Encapsulation, version control, run time, loops, if statements,
        - vpl: Insight, Overview, UI

# Applications

Geoprocessing remains a rather vague term used throughout these descriptions. I will define concrete geo-processing applications, which I aim to create using the VPL:

## Application 1: On Demand Triangulator + Isocurves:

*difficulty to code: Hard*

### Input:

- Point Cloud

### Output

- Line Curves / .png render of line curves

### Steps:

- Load ahn3 point-cloud (WFS Input Widget | WFS Preview Widget)
- Visualize point cloud on top of base map of the netherlands (WMS Input Widget | WMS Preview Widget)
- Only select terrain points (list filter Operation)
- Construct a 2d polygon by clicking points on a map (Polygon Input Widget)
- Select Area of interest using a 2d polygon (Boundary Include Operation)
- Triangulate point cloud with a certain resolution (Triangulate Operation)
- Intersect the mesh surface with a series of planes (Isocurves from Mesh Operation)
- Preview data (MultiLine Preview Widget)
- Export data (MultiLine export Widget)

## Application 2: Get geometry of a street based on a street name

*difficulty to code: Medium*

### Input

- adress string

### Output

- Vector (center)
- cityjson of buildings in this street

### Steps

- query the adress coorindate using openstreetmap (Open Streetmap Fetch Operation)

> [JF] : Async operations????

- query the geometry with the coorindate using 3d bag (3D Bag Query)
- perform operations to prune away all buildings not in the same street (????)
- visualize / export geometry

## Application 3: Query ahn3 height

*difficulty to code: Easy*

### Input

- Csv file with `latitude` and `longitude` column

### Output

- Csv file with `latitude`, `longitude` and `height` column

### Steps

- Iterate per item in the csv (figure out how looping is gonna work on the canvas)
- load AHN3 in raster format (WMS Input Widget)
- convert the coordinate to an x / y coordinate in the image
- query the raster image (image.query() Operation)
- construct a new CSV (csv.new() Operation)
- make it downloadable (csv.download() Widget)

Description

This is something I would use geopandas for

# Random Notes

## note on applications

I'm starting to realise that the specific types of geoprocessing applications this tool would be excellent in, are processes where you want to make geodata interactive: move the cursor, and things change. Select a different building, and things change.

## note on VPL

Still consider a non-vpl method. a web-ide where selecting a variable will make it light up on the map. This will be more programmer-friendly, at the cost of being more non-programmer-unfriendly.

consider a code / vpl tab, like in `blockly`. This way you could make both.

## some alternative title:

*FAIR geodata processing in a `Browser` using `WebAssembly`.*

## some knowledge gap

- GIS Processing should be more operational & less obscure without losing any potency.

## note on project title

`geon` is quite general. `geon.nl` apparently exists already: geon.nl. It is not the most original name in the universe. I cooked up a new name : `geofront`, as in `geometry / geodata frontend`. I bought geofront.nl just in case :).

## notes on existing geodata processing methods

- Python + Jupyter + geopandas approach
    - sort of parametric: you can trace back, and rerun certain parts
    - still very CLI-like, output is static
- QGIS approach
- Google earth engine approach
- ( Grasshopper approach )