(semi-formal attempt at an introduction)

1. INTRODUCTION

Standards

Geodata experts are often concerned with the creation and adoption of common standards. (WFS WMS, cityJSON). This is done to prevent an *interrelating mess*: a graph with each node connecting to every other node.

Software engineers are taught to avoid O(n^2) algorithms as much as possible, and this is a similar phenomenon. Sometimes, this problem can be solved by introducing one intermediary node, after which all different nodes only need to be concerned about its read-write relationship to only that intermediary. (name a vivid example)

WebAssembly

WebAssembly is an emergent technology / standard which the exact same goal (SOURCE: WASM paper). It is a compilation target meant to be platform & source independent. It attempts to be the ultimate intermediary between software and hardware, which would be a dream for developers in that sense: "Run Anything Anywhere".

"Run Anything" means that a platitude of languages (C, C++, Rust) can be compiled to WebAssembly, with the promise that these wasm-binaries are almost as fast as native binary compilations of those same languages.

"Run Anywhere" means that it is possible to run wasm-binaries on Windows, Mac & Linux Desktops, natively on mobile devices, on servers, and even client-side in web-browsers. This runtime is containerized, improving privacy, security against malware, an user control.

Applications

These two properties together give WebAssembly many interesting applications. Like Docker, it can be used to run foreign software in a save, containerized manner (SOURCE: WASI). This is why it is now officially supported by all major browsers, making it the 4th type of code to run in a browser, alongside javascript, css and html.

Why is that interesting? Well, Google Earth for example is using WebAssembly(SOURCE: ...). They could just take the desktop application of Google Earth, compile it to WebAssembly, and then publish it on the web, making it much more accessible.

FAIR

An important side-note is the relationship of WebAssembly and the FAIR principles. The FAIR principles are a collection of four well-established assessment criteria used for judging the usability of software applications (SOURCE). They stand for Findable, Accessible, Interoperable, and Reusable. WebAssembly has the potential to improve all four of those criteria for a piece of software:

WASM web apps: There is no delay between Findability and Accessibility. As soon as it can be found, it can be accessed.

WASM containerized: If the core logic of something is compiled into a wasm library, than this logic becomes Interoperable and Reusable. We can be sure that it will produce the same results, wherever it is run. Write once, use anywhere <-> Collect once, use multiple times

Geomatics

It is unclear what WebAssembly exactly means for the geospatial community, but potential of improving the FAIR qualities of geoprocessing software is promising: What if the exact same code could be used client-side and server-side? What if all C++ based libraries such as CGAL could be utilized from a browser, without needing to be installed? What if processes which were previously hard to chain together could suddenly work together perfectly?

The potential benefits of WebAssembly make research into utilizing wasm crucial. Both the technical capabilities of `wasm` for geomatics purposes need to be research, as well as the possibilities in utilization. This is exactly what this paper will attempt to research.

Paper

This paper will perform research into the possibilities and effectiveness of compiling C++ geoprocessing libraries such as CGAL & 3dfier into WebAssembly. To explore the utilization of WebAssembly, this research will involve creating an application using these libraries. This can be seen as a "geo-wasm case study", as this application would be impossible to create without wasm. The application will take the shape of a visual programming language, or VPL for short.

2. RELATED WORK

other geo-vpls:

RESEARCH QUESTIONS

How well does WebAssembly support a client-side geoprocessing vpl?

1. **GEO-WEB-VPL**: How to make a web-based, client-side, vpl geoprocessing environment?
    - 1a. **GEO**: What basic features does a geoprocessing environment need?
        - *answer: MOSKOW*
    - 1b. **WEB**: What options and limitations does a HTML5, CSS & JS based environment give us?
        - *answer: Canvas, HTML as ui, webgl*
    - 1c. **VPL**: What are the advantages and disadvantages of using a vpl?
        - *answer from the vpl papers*
2. **GEO-WASM**: How well can C++ geoprocessing libraries such as CGAL & 3dfier be used within a web browser without needing to be installed, by using WebAssembly?

    - 2a: How well do WebAssembly compiled geoprocessing (geo-wasm) libraries perform compared to native, cli usage?
    - 2b: How to handle types / data models between multiple, unrelated `wasm` libraries?
    - 2c: How do C++ geoprocessing libraries differ from all other C++ libraries?
    - 2d: What does this difference mean for `wasm` compilation and usage?

3. **GEO-WEB-VPL + GEO_WASM**: How well can geo-wasm libs be used within the context of a geo-web-vpl?

    - 3a: What data must a geo-wasm provide in order to become usable within a geo-web-vpl?
        - *answer: descriptors of operations and variables, how many inputs and outputs, etc.*
    - 3b: How can this data be utilized by the geo-web-vpl?
        - "
    - 3c: How are the geo-wasm libraries distributed?
        - *answer: web service like npm or pip? Something like an app store?*

- How to design a `wasm` plugin ecosystem?

    - Marketplace Website ?
    - Npm ?
-