PRESENTATION SCRIPT

0 pre-intro

- Hello everyone, thank you all for coming
- Today I would like to give you a small, 10 minute presentation to explain my P4 Draft Thesis in broad strokes.
- Content will be as can be expected:

Introduction

Motivation

- **GIS SOFTWARE NORMALLY COMES IN ONE OF TWO FORMS**: Applications, and Libraries [REF: Elliott, 2007]

  Both have different intensions and strengths Software libraries are tailored to programmers, and generally offer rich functionality which the programmer can use to compose new software. Applications are tailored towards end-users, and offer a subset of functionality, but offer much more in terms of interactivity and visualization.

  - HOWEVER: This split in intensions has drawbacks:
    - libraries are not end-user friendly:
      - functionalities are not directly accessible.
      - no GUI \ visualization support out of the box
    - applications are not programmer friendly:
      - applications limit access to functionality
      - applications are not usually composable

- **For GIS: we desire Visualization & Composability**

- **THERE ARE MULTIPLE WAYS** researchers and developers have tried to bring software applications and libraries closer together.

- **ONE OF THESE WAYS IS THE VISUAL PROGRAMMING LANGUAGE**
  - A visual programming language, or VPL for short, is both a programming language and a graphical application.
  - Because of this, it can both be used to
    - interact with and visualize geodata
    - compose automated procedures
  - **new development: Web VPLS**
    - 'live' alongside web GIS
    - (as more and more GIS applications are starting to become web-based, it makes sense that 'configuring automation' in a web browser becomes a need).
    - Accessibility advantages: no installation

Problem

- **PROBLEM: library portability**

  - many of the native VPLS use critical 'geocomputation' libraries which make these VPLs usable for GIS.
  - However, these same libraries cannot be used on the web by normal means.
  - This hinders the original goal of 'bringing Libraries and (Web) Applications closer together'
  - web based alternatives exist, but
    - time consuming to synchronize and test features [Ref: mapbox.rs]
    - which in turn hinders innovation

- **Goal: Solving the library portability problem for web-based VPLs.**

  - make it so native, system level libraries can be used in a web-based VPL.

Objective

- **The Objective of the study** is to attempt to solve the library portability problem for web-based VPLs.

  - In doing so: Contributing to the quality of web-based VPLs
  - And: Contributing to closing the gap between library and (web) applications.

- **Research Question**:

  - How can native geocomputation libraries be compiled, loaded, and utilized within a browser-based dataflow-VPL?

- **How**:

- Practical: This study has Designed, implemented, and discussed a possible solution

- Three challenges are identified** to using native library in a vpl:

  - Firstly, **I. Compilation**:
  - Secondly, **II. Loading**:
  - Lastly **III. Utilization**:

- The libraries we intend to bring to the web-VPL are industry-standard geo-libraries, written in system level languages, like C++ or Rust

Background

Dataflow VPL

A dataflow VPL is a specific type of Graph VPL with a set of constraints:

- Dataflow VPL are defined as Graph based VPL,
  - with only pure, stateless, functions without side-effects. (not triggering anything special, using global properties, etc.)
  - and with immutable variables, (meaning that variables are not allowed to change after initialization)

which allows it to gain qualities akin to 'Functional Programming'

These qualities can roughly be grouped as 'clarity for both programmer and machine'

- Leading to better performance and usage

- and maybe because of this, almost all VPLs handling Geometry that I have looked at as part of the background study, are implemented as 'semi' dataflow VPLs:

- This observation had consequences for the methodology.

Methodology

- To understand the methodology of this study, it is important to recognize all the language models a Geo-library needs to traverse to go from native library, to a 'plugin library' incorporated in a web-based VPL.
- Moreover, the language model of the VPL itself also count as one of those language models, as it is a programming language to a degree.
  - The model of a dataflow VPL was chosen for all the reasons of the previous chapter.
  - However, this model did not have a web implementation yet, and had to be created from scratch.
- With those things said, the methodology is defined as follows:
  - First, a **custom web VPL** is designed and implemented as a **host** for the libraries.
  - Second, a **plugin system** is designed and implemented.
    - The system consists of:
      - A Plugin Loader on the side of this VPL, and
      - A Plugin model the plugin libraries must adhere to.
  - Finally, Two sets of **tests** are performed:
    - one set to analyse to what extent this plugin system allows for the compilation of native libraries.
    - one set of tests to analyse to what extent this solution allows for proper utilization of those native libraries in the VPL environment.

Results

1 Geofront

It turned out that this design could indeed be implemented on the web in TypeScript (JavaScript). This implementation had advantages and disadvantages:

- The big advantage of a TypeScript implementation is that a great number of features do not need to be included within the source code of the application, leading to quick load times.
  - WebGL, UI (HTML), 2D Canvas API
- Also, Javascript's flexibility proved to be useful to support features like dynamically loading and using libraries at runtime.

- However, TypeScript does not have any runtime support for types, leading to reflection problems and problems with javascript types (no integer atomic, only number),

- Also, javascript is not a good host for enforcing dataflow-VPL constraints.

  - for example, the absence of explicit immutability made it so that all functions will need to be 'thrusted' to not alter their input data.

PICTURES

To begin with the base VPL:

- This is what the created dataflow VPL looks like.

  - Just like a native VPL, it offers inspectable in-between variables

- Different than most VPLs, it makes a strong distinction between computation nodes: which are pure functions

- And so-called Widgets, which offer usability niceties, but can produce side effects.

- offers a 3D viewer to inspect geodata

2 Plugin System

- A novel plugin system was designed to load a library compiled to WebAssembly into this prototype VPL almost without explicit configuration.

  - This method allows one library to serve **Three formats**: as **VPL plugin**, as **native C++ / Rust library**, and as **Js library**.
  - Moreover, this method allows users to develop a library locally, and then quickly experiment and test its usage online.

- This design was implemented successfully, albeit with some limitations, and certain aspects which did need explicit configuration.

- The biggest drawback was that by designing a lenient, non-restrictive loader, the 'dataflow' qualities could again not be enforced. (Libraries loaded can produce side effects, can point some object loaded in memory, there is no way of preventing them)

PICTURES

This is a minimal example of a Rust library, which is loaded by the plugin loader into the dataflow VPL format.

- The loader incorporates aspects like Types and parameter names.

3 Compilation Tests

The aforementioned plugin system was tested by attempting to load multiple libraries. The conclusion of these tests raise a dilemma between Rust and C++:

Rust

- The study could successfully expose multiple native geocomputation library in a manner properly consumable by a web-vpl.
- This lead to a variety of applications, like loading a `.laz / .copc` point cloud, and triangulating it.
- Regrettably, the Rust language is too new to contain libraries which can be considered 'industry standard'.

C++

- C++ based geocomputation libraries cannot be sufficiently compiled for web consumption, at least not for the purposes of loading the functionalities within a web-VPL.
- The Emscripten compiler did not provide the same level of features as Rust's `wasm-pack`.
- A series of workarounds had to be created. These workarounds came close to successful utilization, but the time frame of this study did not allow a final workaround to be implemented.
- C++ Has a strong foundation of existing geocomputation libraries compared to newer languages like Rust.
    - However, this same legacy inhibits its portability, which makes for Larger binaries, less performant 'wrapper' features, and makes it overall harder to compile to web browsers.

IMAGES

Discuss what you see...

Conclusion

Q: "How can native geocomputation libraries be compiled, loaded, and utilized within a browser-based dataflow-VPL?" A: The key to successfully compiling, loading and using geo-libraries is to address the frictions between the four required groups of languages:

1. Existing geocomputation libraries (C++, Rust)
2. WebAssembly
3. JavaScript / Typescript

- wasm wrappers
- VPL & loader source code

4. The dataflow VPL

- Pure functions, immutable variables

This study focussed on the disconnect between a dataflow VPL model on the one hand, and existing (geocomputation) libraries on the other hand, and tried to mitigate the discrepancy between these two, and all intermediate languages required for web compilation.

The proposed solution managed to sufficiently address these frictions for the Rust language. However, Rust geo-libraries are too young to be considered industry standards. Further study is required for incorporating C++.