Please wait for the presentation to start...

Geofront:

Directly accessible GIS tools

using

a web-based visual programming language

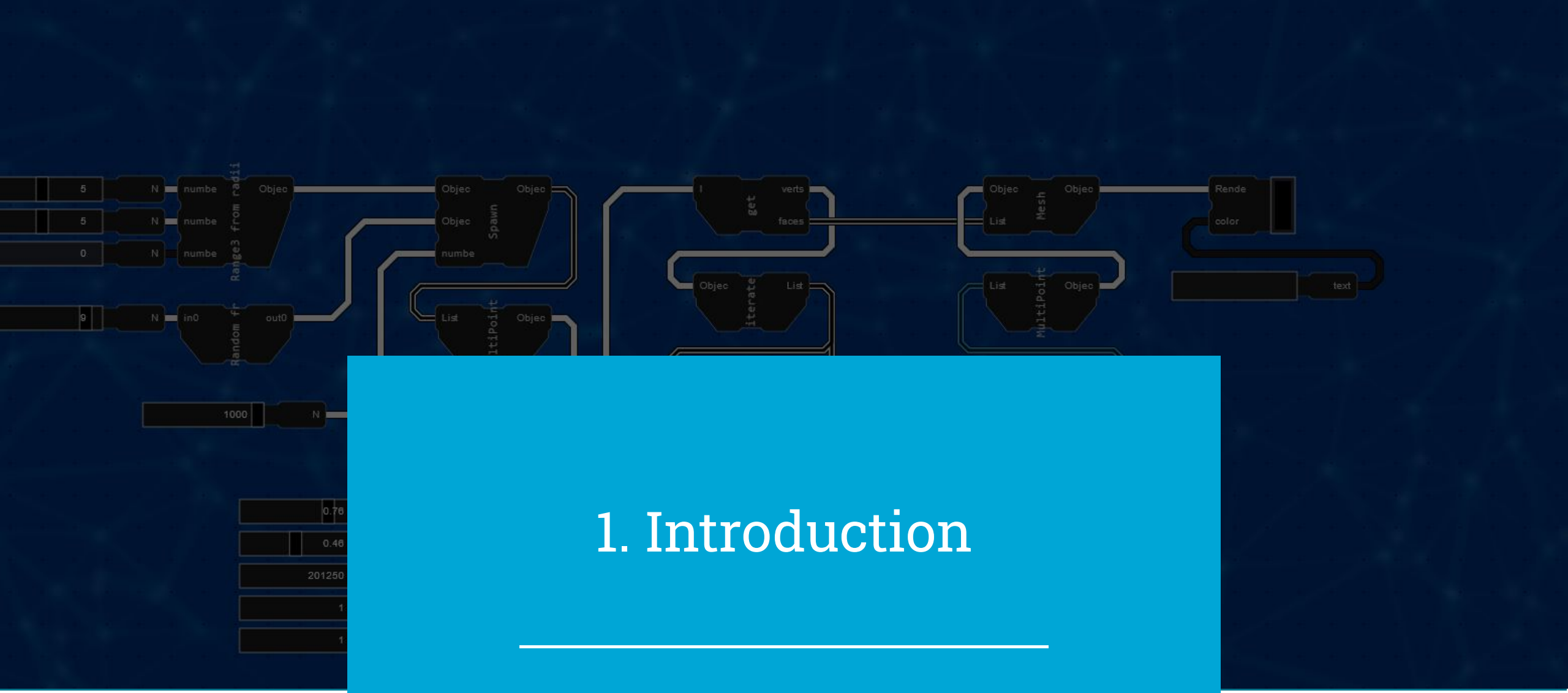
Master Thesis Geomatics | Final Presentation

Jos Feenstra | November 4th 2022

TU Delft

1. Introduction
2. Objective
3. Background
3. Methodology
4. Results
5. Conclusion

TU Delft



1. Introduction

TU Delft

GIS:

Geographical

Information

Science

End Users

Climatology

Infrastructure

Urban planning

Agriculture

Governance

Navigation

Military

(…)

TU Delft

Geographical

Information

Science

End Users

- Environmental studies
- Infrastructure
- Urban planning
- Governance
- Navigation
- The military
- Argiculture

Tools

TU Delft

Geographical

Information

Science

End Users

- Environmental studies
- Infrastructure
- Urban planning
- Governance
- Navigation
- The military
- Argiculture

Tools

TU Delft

Tools: Two forms of software:

Application

Library

TU Delft

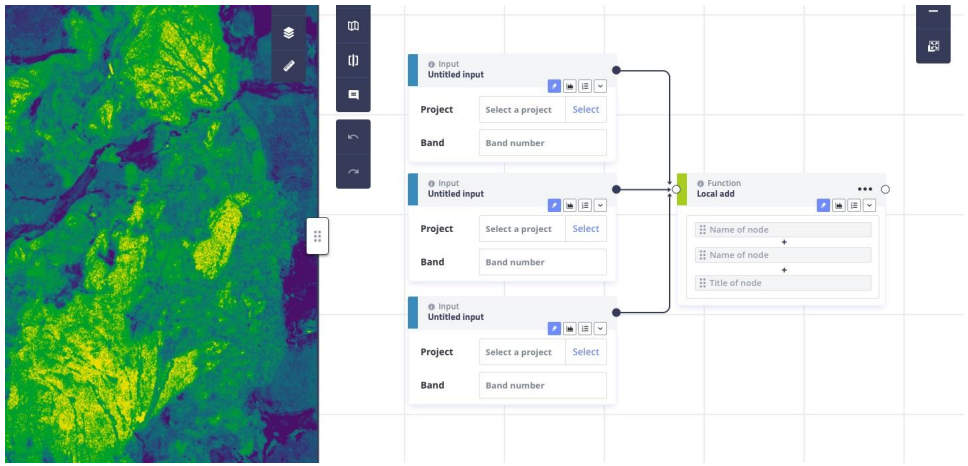
Application

Applications:

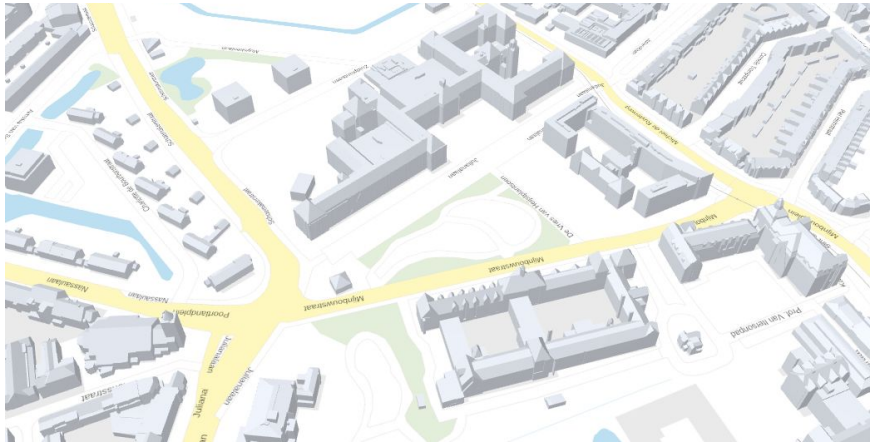
- End users
- Interaction

In GIS:

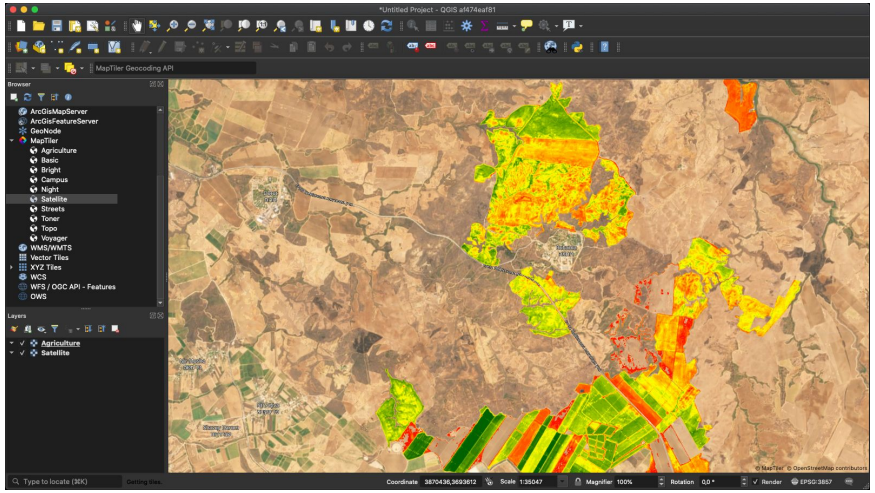
- Visualization
- Analysis



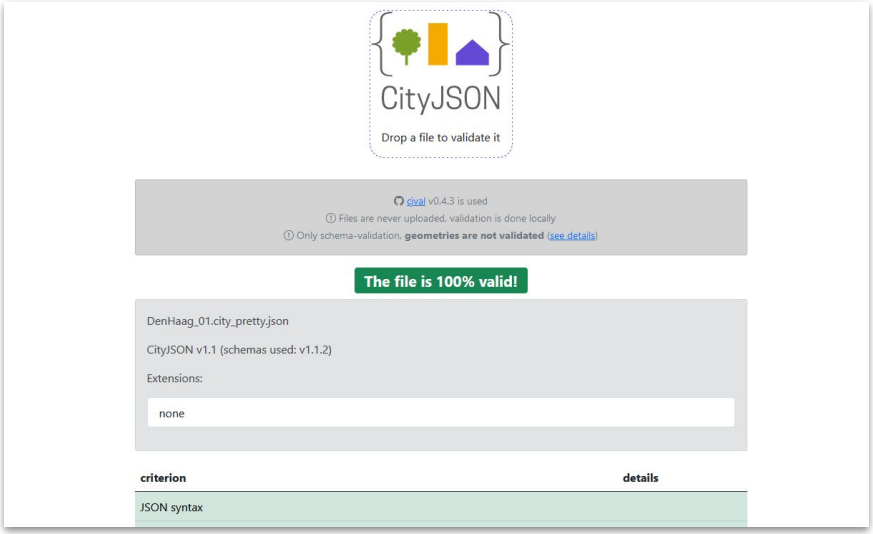
src: Model Lab



src: 3D bag viewer



src: QGIS



src: cjval

TU Delft

Libraries

Libraries:

- Reusable tools for applications (& other libraries)
- Rich functionality
- Cannot directly be used

In GIS:

- Transformation
- Analysis (Validation)

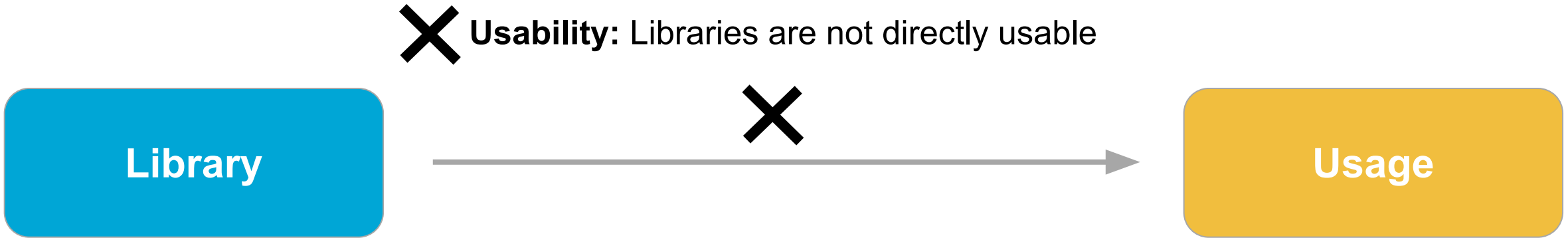
3D Geoinformation Libraries

- City3D
- PolyFit
- val3dity
- cgval
- cjio

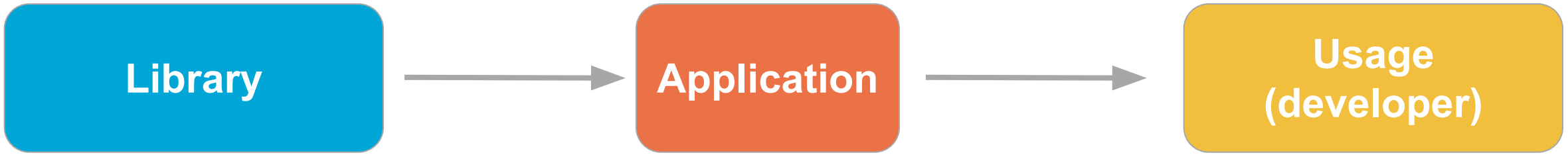
“Core” GIS Libraries

- PROJ
- GEOS
- GDAL
- (CGAL)
- (OpenCV)

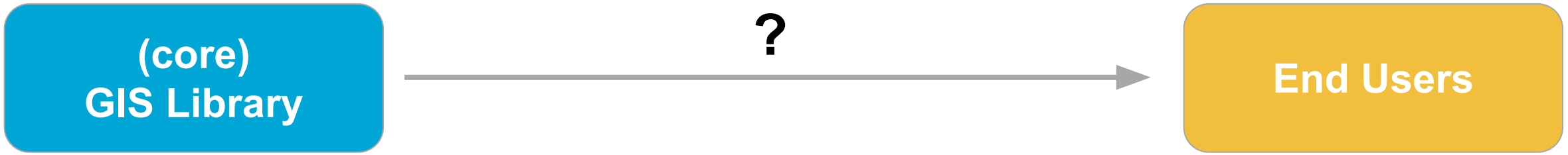
TU Delft


Usability: Libraries are not directly usable
Library
Usage

TU Delft


Library
Application
Usage
(developer)

Problem:



- Transformation
- Analysis
  - Validation

Environmental studies

Infrastructure

Urban planning

Governance

Navigation

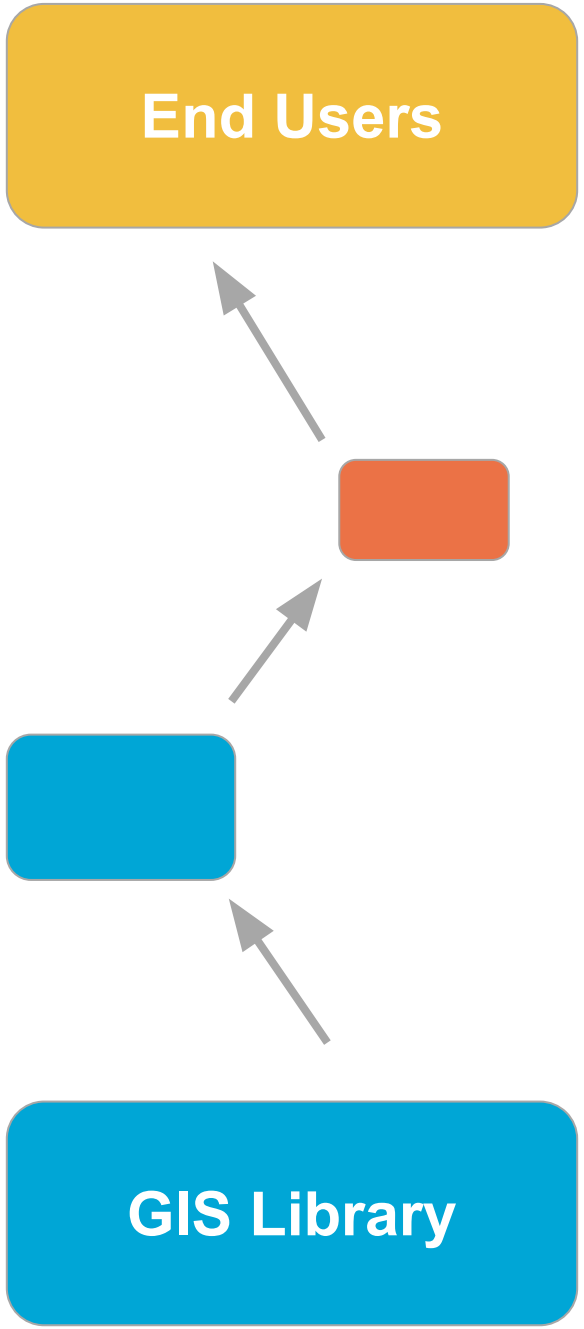
The military

Argiculture

TU Delft

Indirection

- Only indirect usage
- Dependent



TU Delft

Dependent on multiple layers of developers

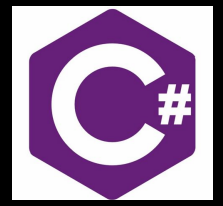
End Users

Application layer

*.exe

custom App

Bindings layer / Intermediate users



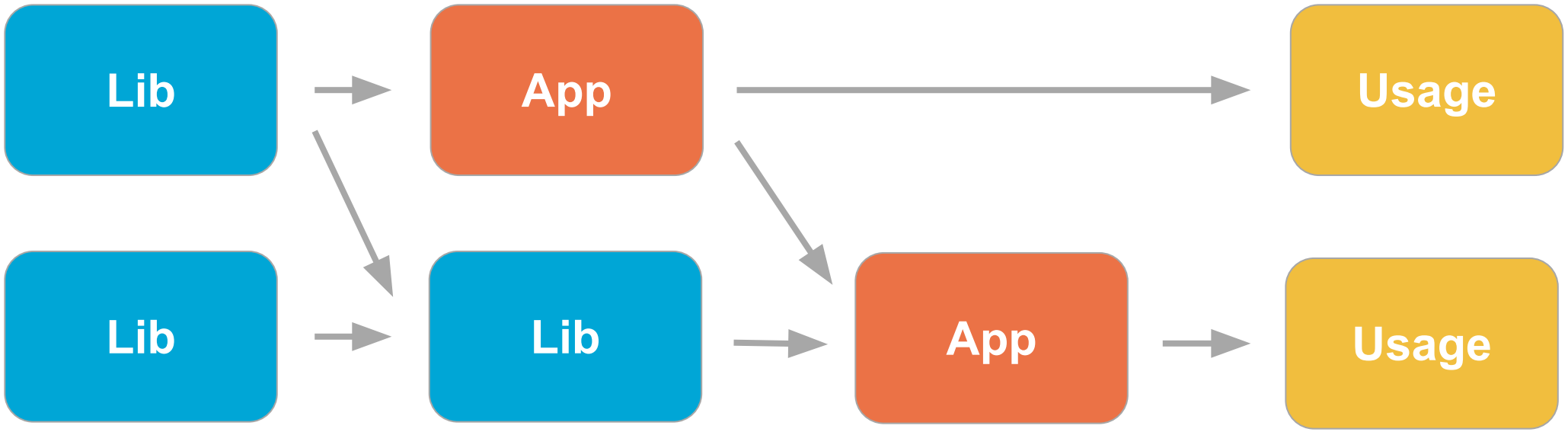
Maintain bindings

Maintain plugins

Maintain app

GIS Library

Moreover:

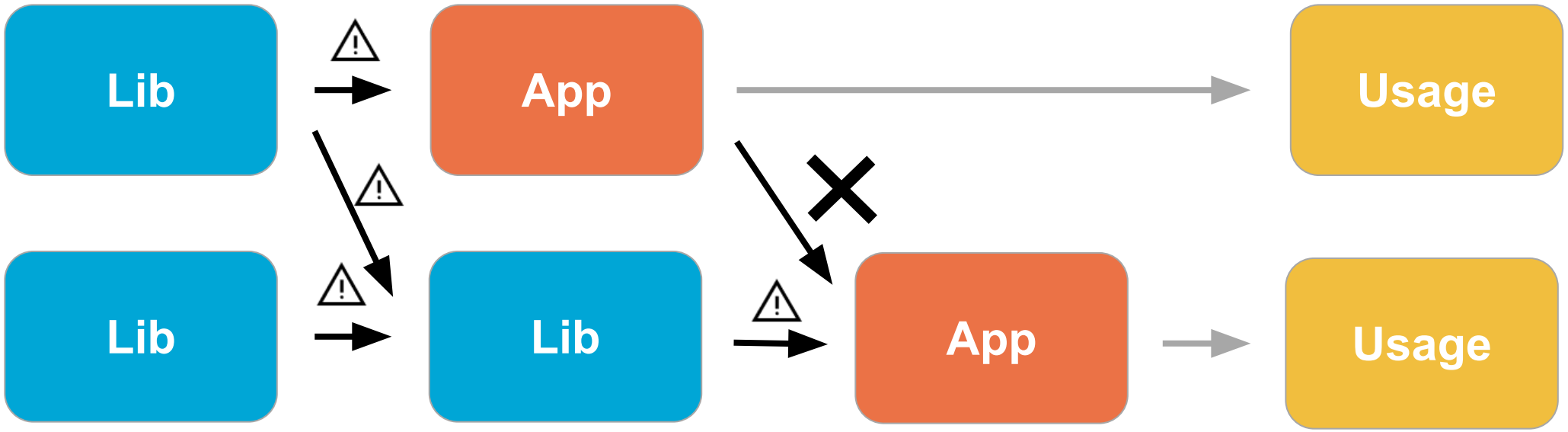


TU Delft

Moreover:

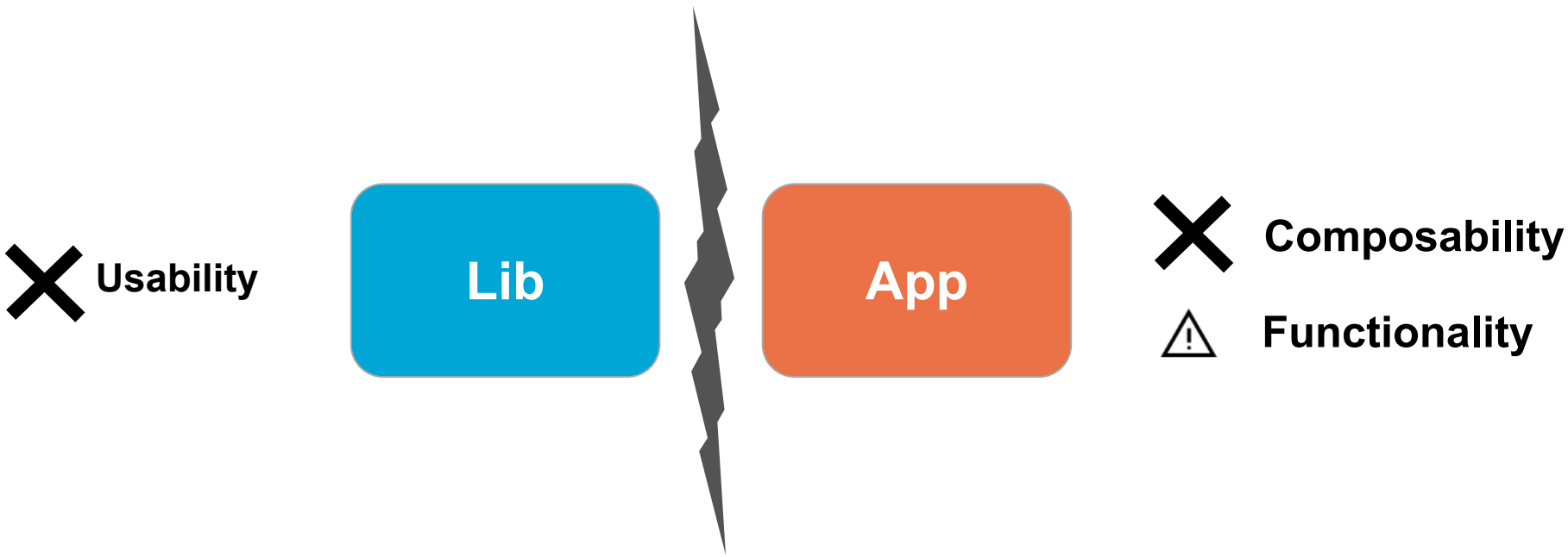
⚠ **Functionality:** capabilities may get lost at every step

✕ **Composability:** apps are **not** further composable



TU Delft

Conundrum:



Given this divide, how to achieve **Functionality,**
Composability, & **Usability** at the same time?

Adopted from Elliott C. (2007).
Tangible Functional Programming

Problem statement

Indirect access, leading to disadvantages…

… for **end users**:

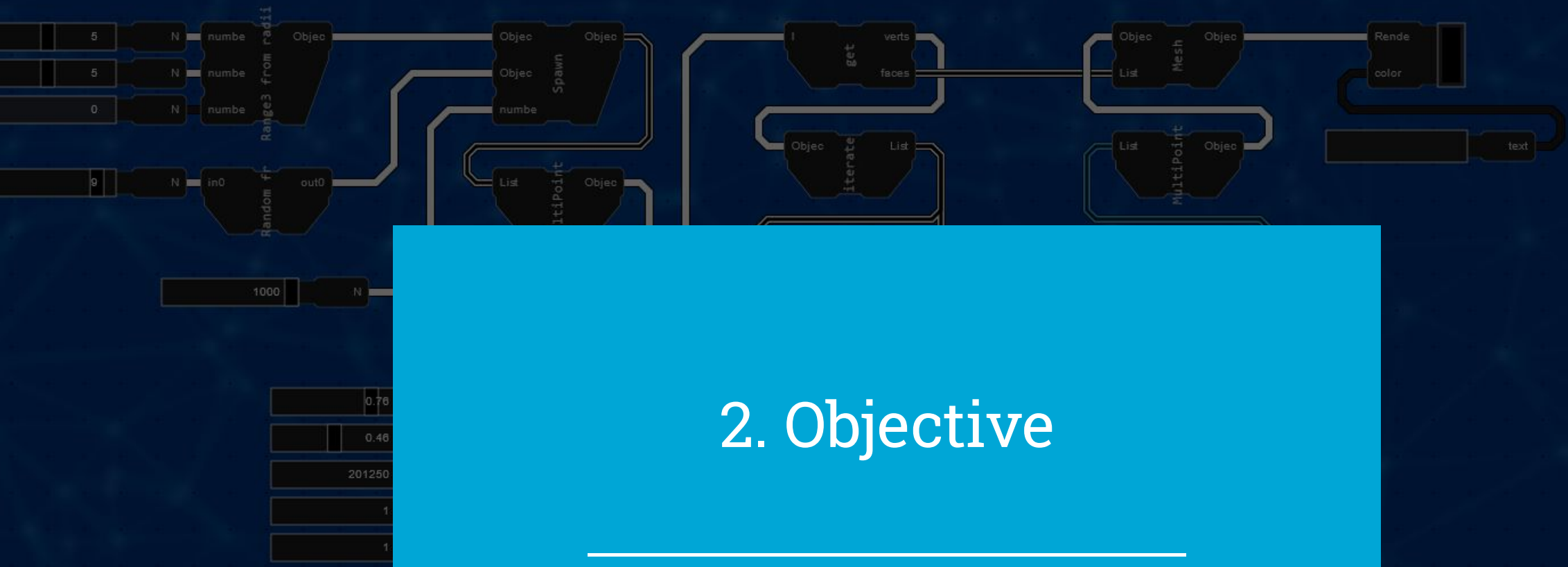
- *At the mercy* of in-between software
- Non-composable applications
- Features getting *lost in translation*

… for **library developers:**

- Synchronizing bindings, plugins, applications.

…for **society:**

- reduced impact of research


5
N
numbe
Objec
Range3 from radii
0
Random fr
in0
out0
Objec
Spawn
numbe
List
Objec
verts
faces
Objec
List
Mesh
List
Objec
MultiPoint
Rende
color
text
1000
0.76
0.46
201250
1

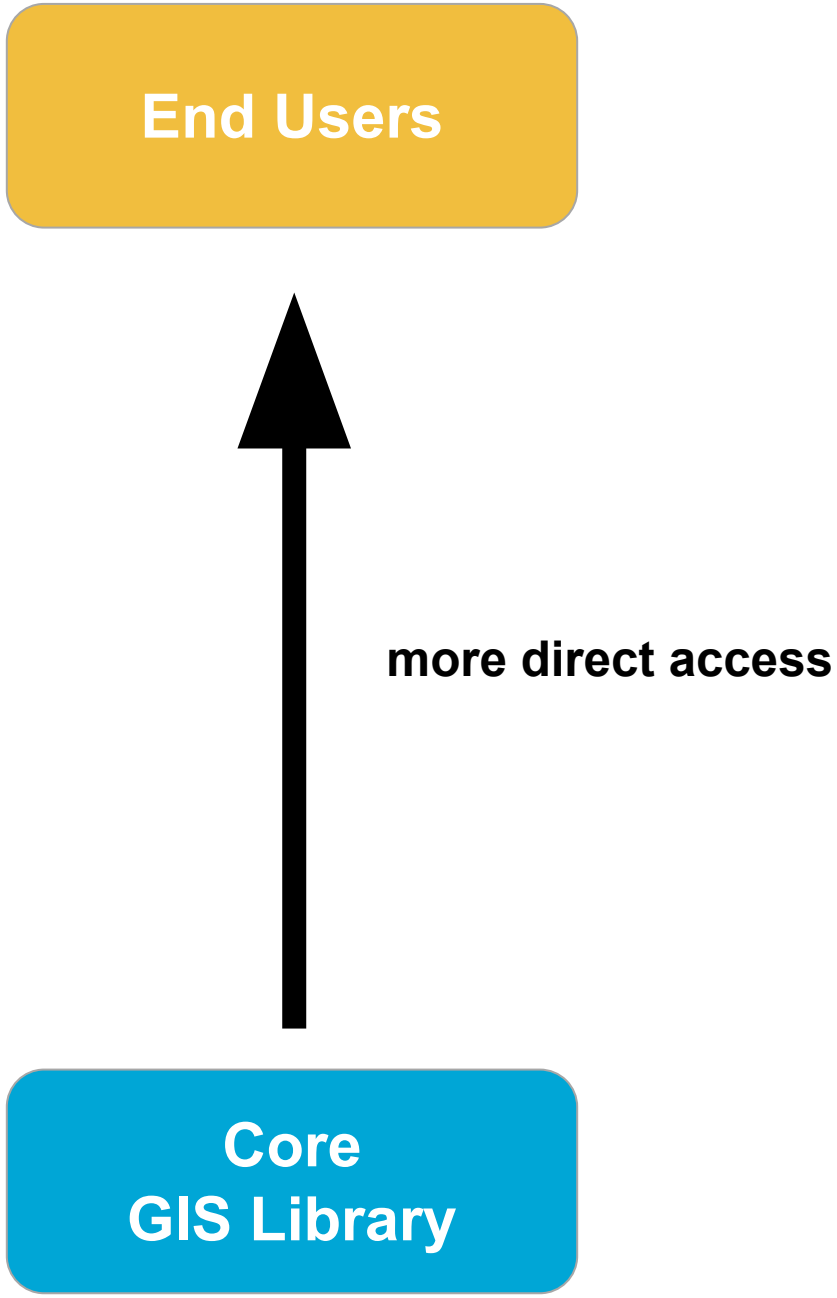
2. Objective

Goal of this study:

Allow GIS practitioners without a background in software development, to access the full potential of core transformation and analysis capabilities found in native GIS libraries.

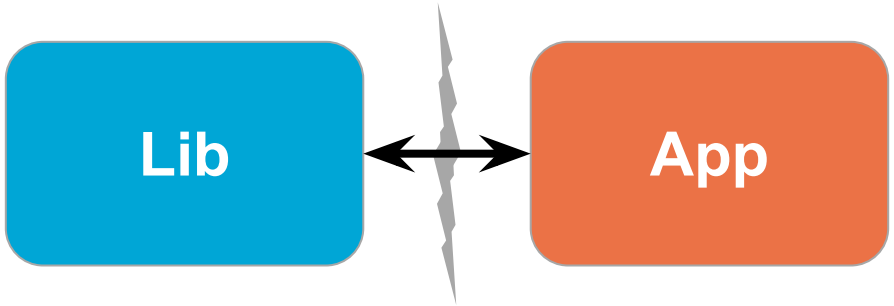
TU Delft

Goal:



TU Delft

Goal:



apps are an endpoint: Not further composable → Add **composability** and **automation** to apps

A **lib** offers no visualization or GUI. → Add **usability** and **GUI** to libs

A **lib** must be turned into an **app** before utilization.

Some **Lib** capabilities get lost when used in an **app**

TU Delft

How:

Presenting and prototyping a novel method:

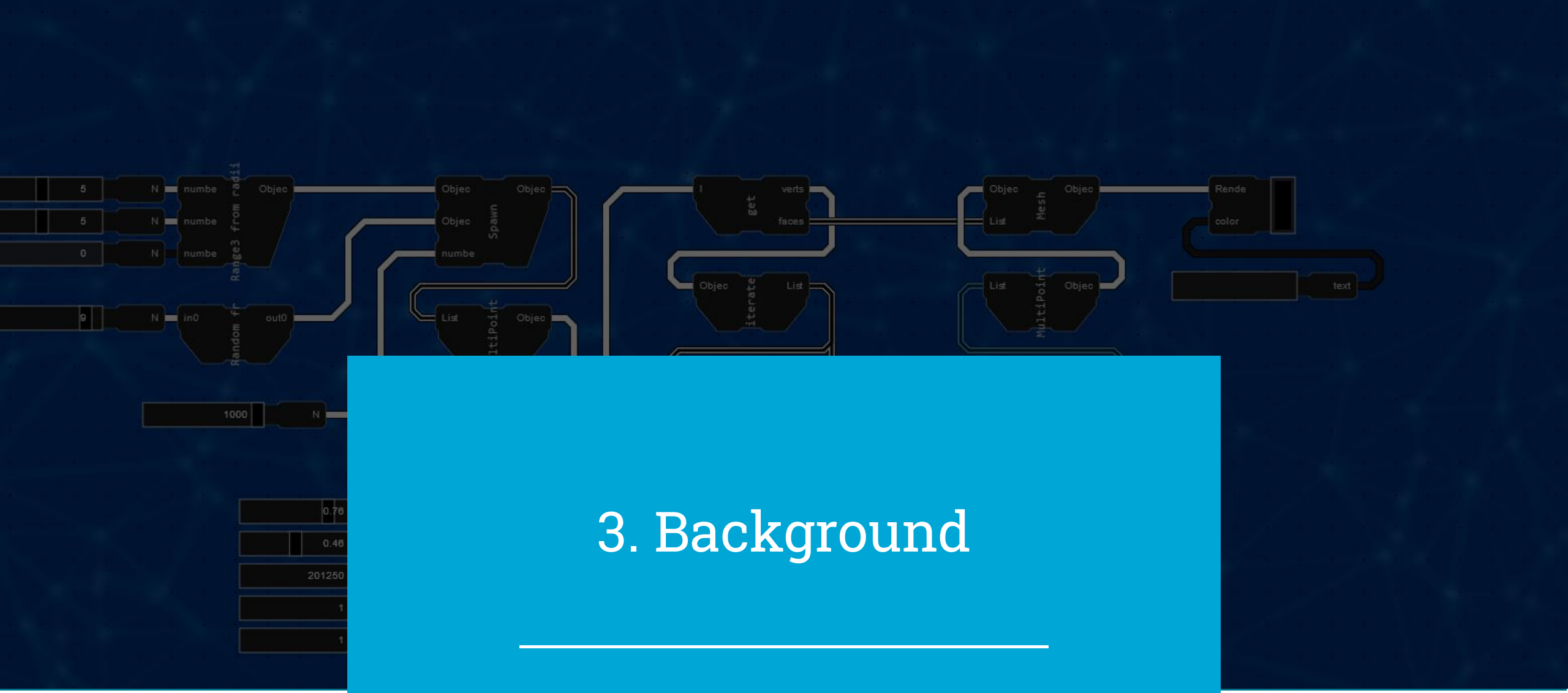
A **Web-based Visual Programming Language (VPL)** using **WebAssembly**

TU Delft

Research question:

Is a **web based VPL** a viable method for directly accessing native GIS libraries with a composable interface?

TU Delft



3. Background

“A Web-based VPL using WebAssembly”:

1. **Web Application**
2. **Visual Programming**
3. **WebAssembly**

TU Delft

1. Web Applications → Accessibility

Web Application → distribution

- No Installation
- Cross-platform

Static Web Application

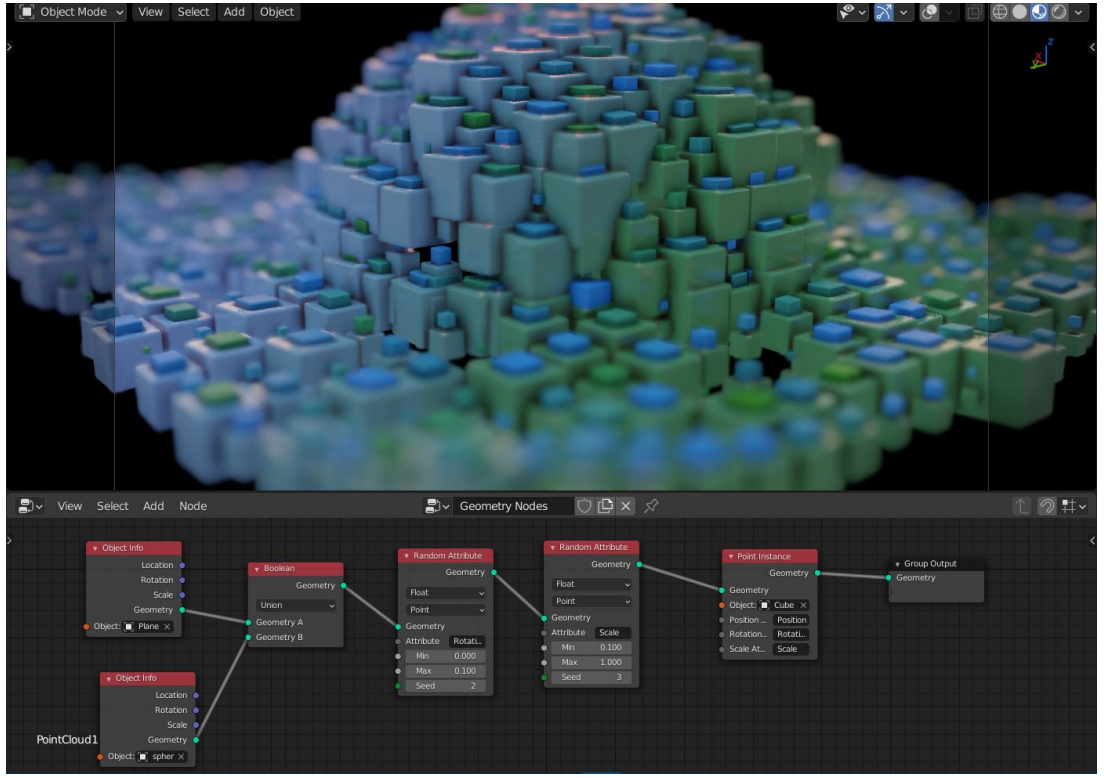
- More composable
- Cheap

TU Delft

2. Visual Programming → Composable applications

- Both a scripting language **and** application
- ‘programming’ by using GUI
  - Composable GUI
- Multiple types, O.A. Graph-based
  - Nodes
  - Edges

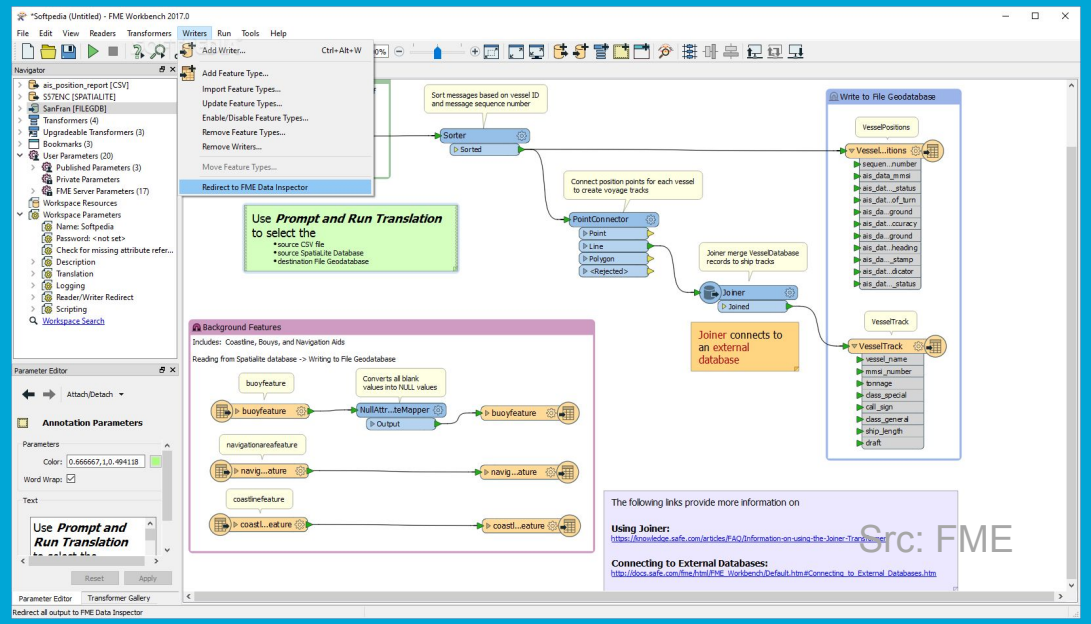
- “More usable than normal programming”
  - Kuhail et al. [2021]

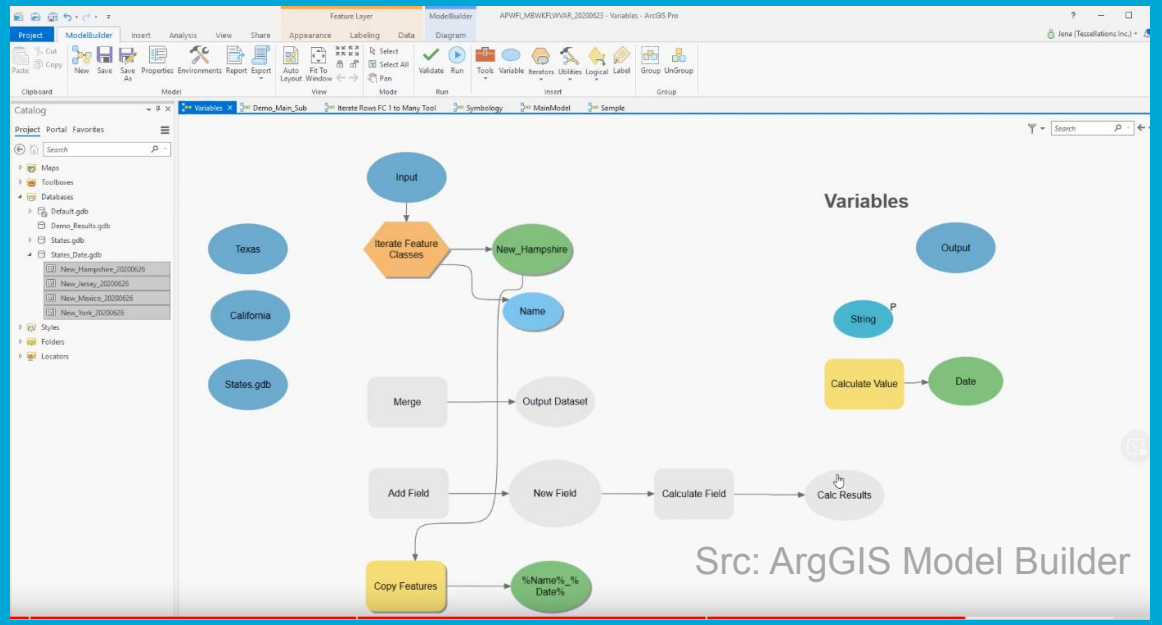


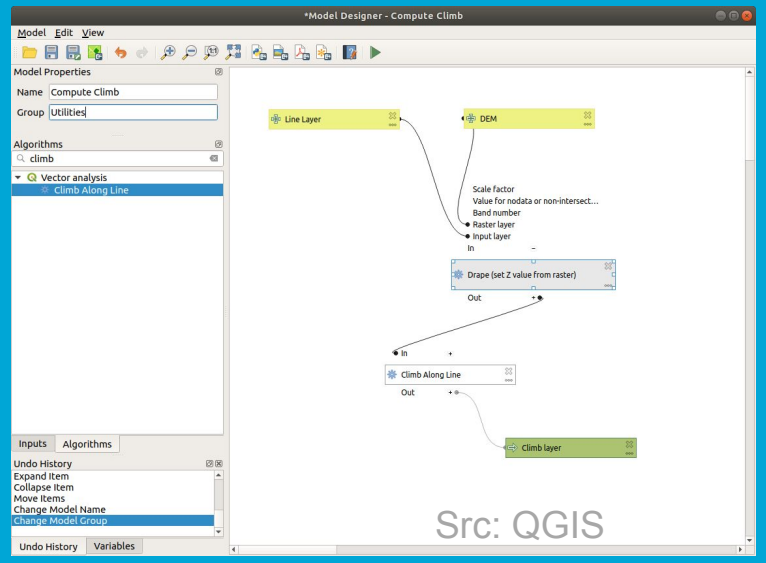
src: Blender Geometry Nodes

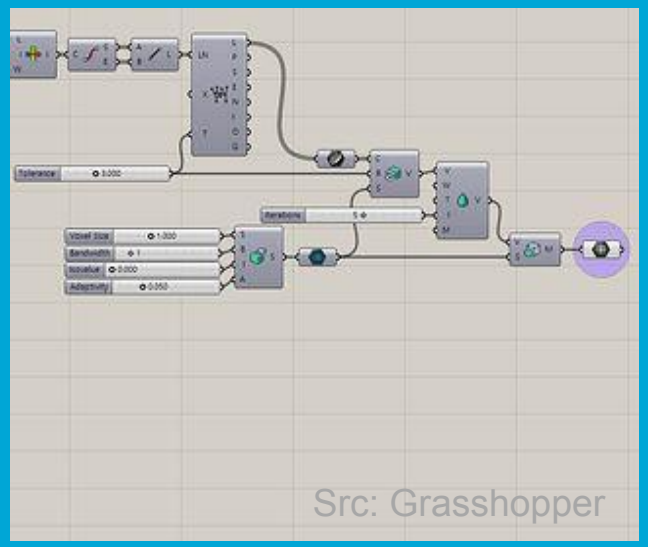
TU Delft

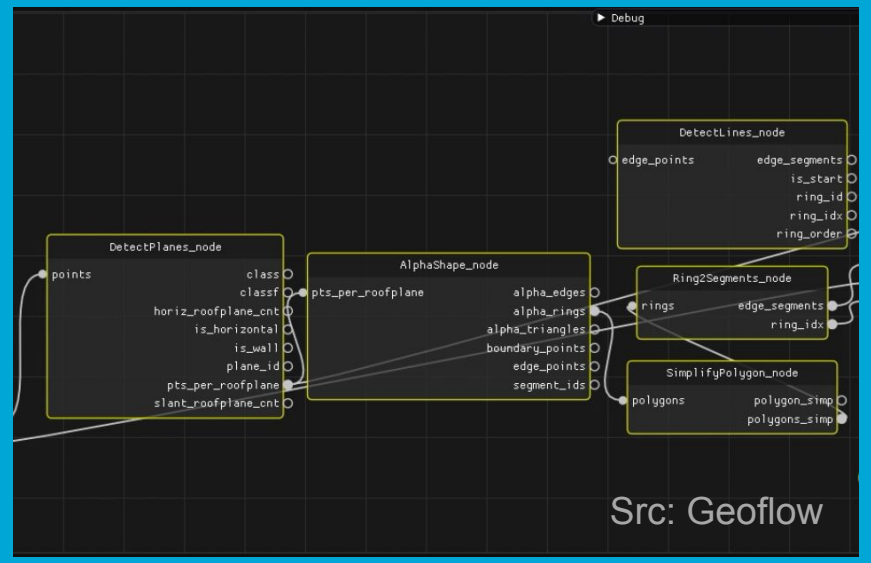
VPL within GIS







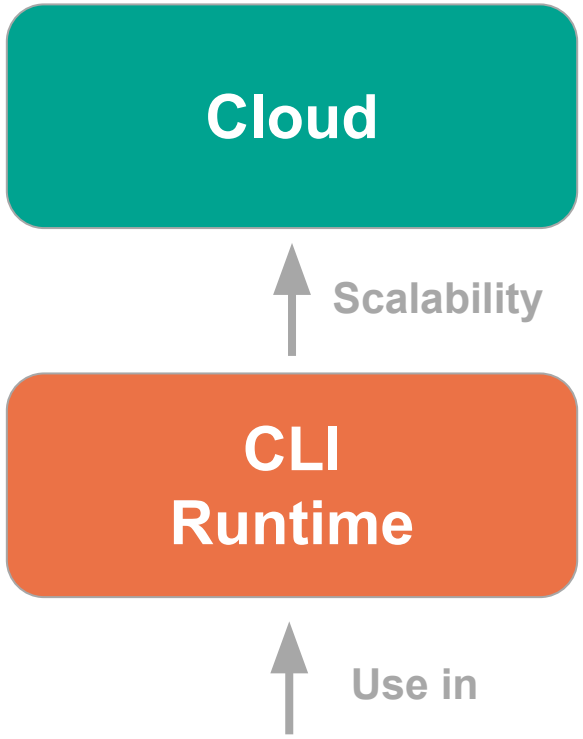


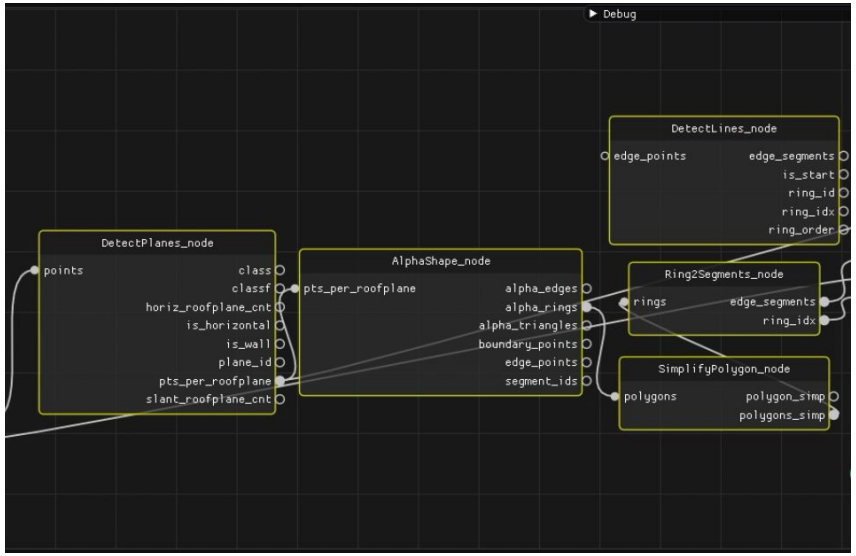


2. Visual Programming: GIS

requirements:

- Scalability
- Rich GUI

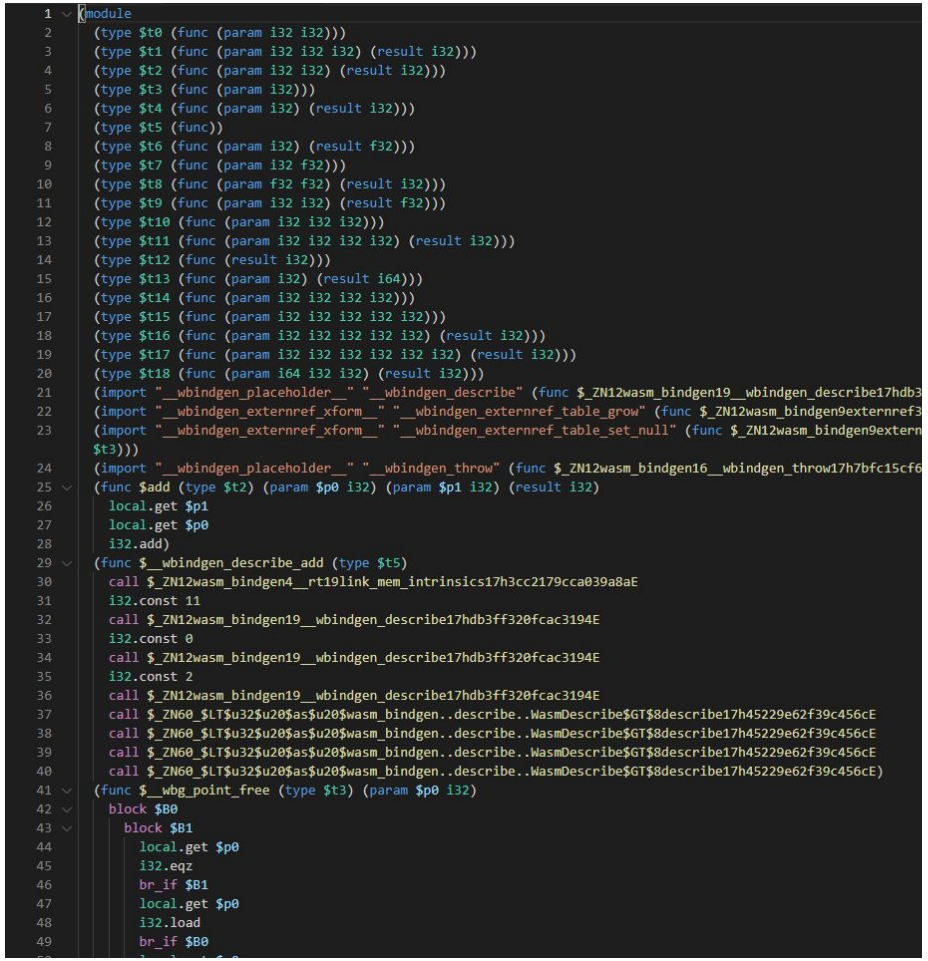




TU Delft

3. WebAssembly → direct access

- Exchangeable binaries
- Binary compilation target `library.wasm`
  - From multiple languages
  - To multiple runtimes
- Since 2017 (Haas, 2017)
- In browsers since 2019 (W3C, 2019)



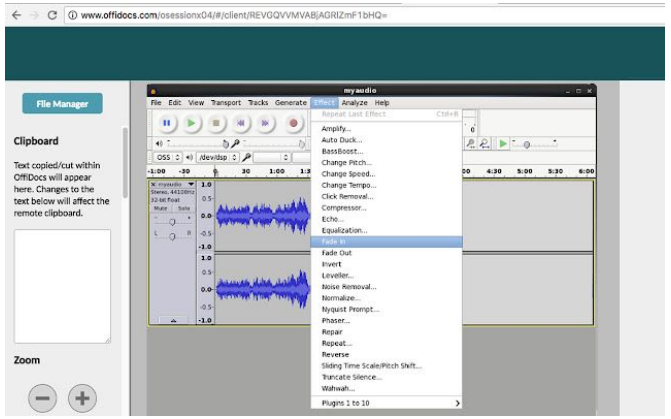
```
(module
  (type $t0 (func (param i32 i32)))
  (type $t1 (func (param i32 i32 i32) (result i32)))
  (type $t2 (func (param i32 i32) (result i32)))
  (type $t3 (func (param i32)))
  (type $t4 (func (param i32) (result i32)))
  (type $t5 (func))
  (type $t6 (func (param i32) (result f32)))
  (type $t7 (func (param i32 f32)))
  (type $t8 (func (param f32 f32) (result i32)))
  (type $t9 (func (param i32 i32) (result f32)))
  (type $t10 (func (param i32 i32 i32)))
  (type $t11 (func (param i32 i32 i32 i32) (result i32)))
  (type $t12 (func (result i32)))
  (type $t13 (func (param i32) (result i64)))
  (type $t14 (func (param i32 i32 i32 i32)))
  (type $t15 (func (param i32 i32 i32 i32 i32)))
  (type $t16 (func (param i32 i32 i32 i32 i32) (result i32)))
  (type $t17 (func (param i32 i32 i32 i32 i32 i32) (result i32)))
  (type $t18 (func (param i64 i32 i32) (result i32)))
  (import "__wbindgen_placeholder__" "__wbindgen_describe" (func $_ZN12wasm_bindgen19__wbindgen_describe17hdb3
  (import "__wbindgen_externref_xform__" "__wbindgen_externref_table_grow" (func $_ZN12wasm_bindgen9externref3
  (import "__wbindgen_externref_xform__" "__wbindgen_externref_table_set_null" (func $_ZN12wasm_bindgen9extern
  $t3)))
  (import "__wbindgen_placeholder__" "__wbindgen_throw" (func $_ZN12wasm_bindgen16__wbindgen_throw17h7bfc15cf6
  (func $add (type $t2) (param $p0 i32) (param $p1 i32) (result i32)
    local.get $p1
    local.get $p0
    i32.add)
  (func $__wbindgen_describe_add (type $t5)
    call $_ZN12wasm_bindgen4__rt19link_mem_intrinsics17h3cc2179cca039a8aE
    i32.const 11
    call $_ZN12wasm_bindgen19__wbindgen_describe17hdb3ff320fcac3194E
    i32.const 0
    call $_ZN12wasm_bindgen19__wbindgen_describe17hdb3ff320fcac3194E
    i32.const 2
    call $_ZN12wasm_bindgen19__wbindgen_describe17hdb3ff320fcac3194E
    call $_ZN60_$LT$u32$u20$as$u20$wasm_bindgen..describe..WasmDescribe$GT$8describe17h45229e62f39c456cE
    call $_ZN60_$LT$u32$u20$as$u20$wasm_bindgen..describe..WasmDescribe$GT$8describe17h45229e62f39c456cE
    call $_ZN60_$LT$u32$u20$as$u20$wasm_bindgen..describe..WasmDescribe$GT$8describe17h45229e62f39c456cE
    call $_ZN60_$LT$u32$u20$as$u20$wasm_bindgen..describe..WasmDescribe$GT$8describe17h45229e62f39c456cE)
  (func $__wbg_point_free (type $t3) (param $p0 i32)
    block $B0
      block $B1
        local.get $p0
        i32.eqz
        br_if $B1
        local.get $p0
        i32.load
        br_if $B0
```

wasm rendered at `.wat`
src: author

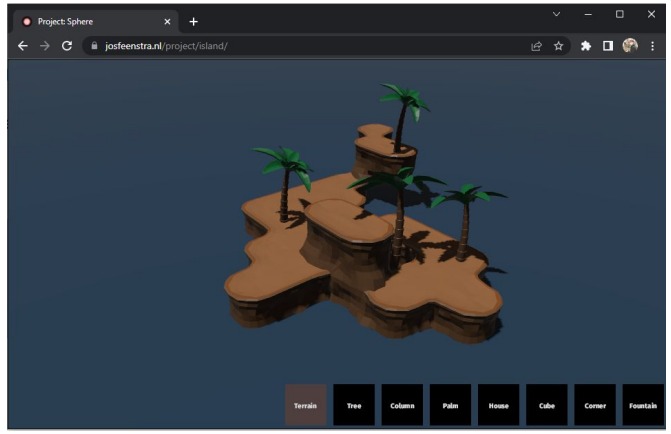
3. WebAssembly

Use case 1: Run native code on the web

src: Milica Mihajlija



src: audacity



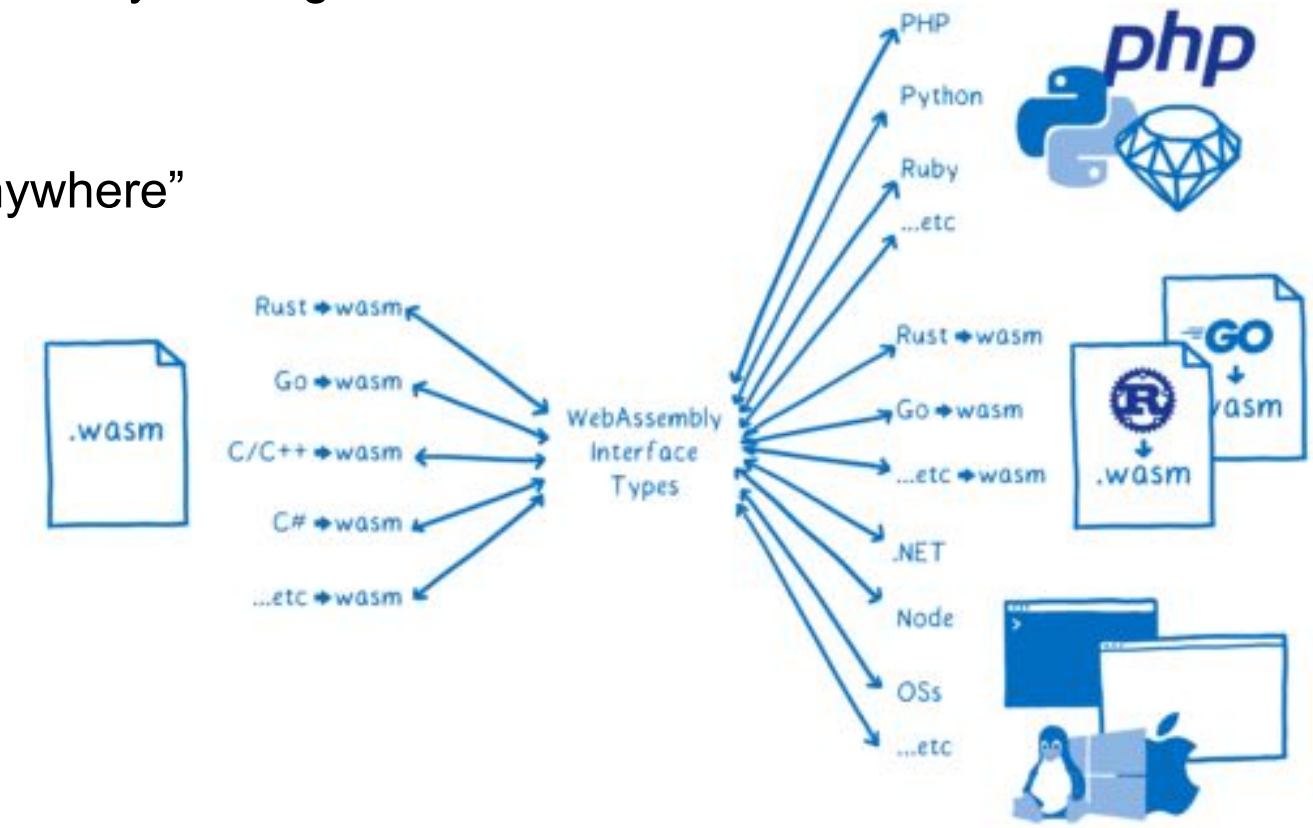
src: author

TU Delft

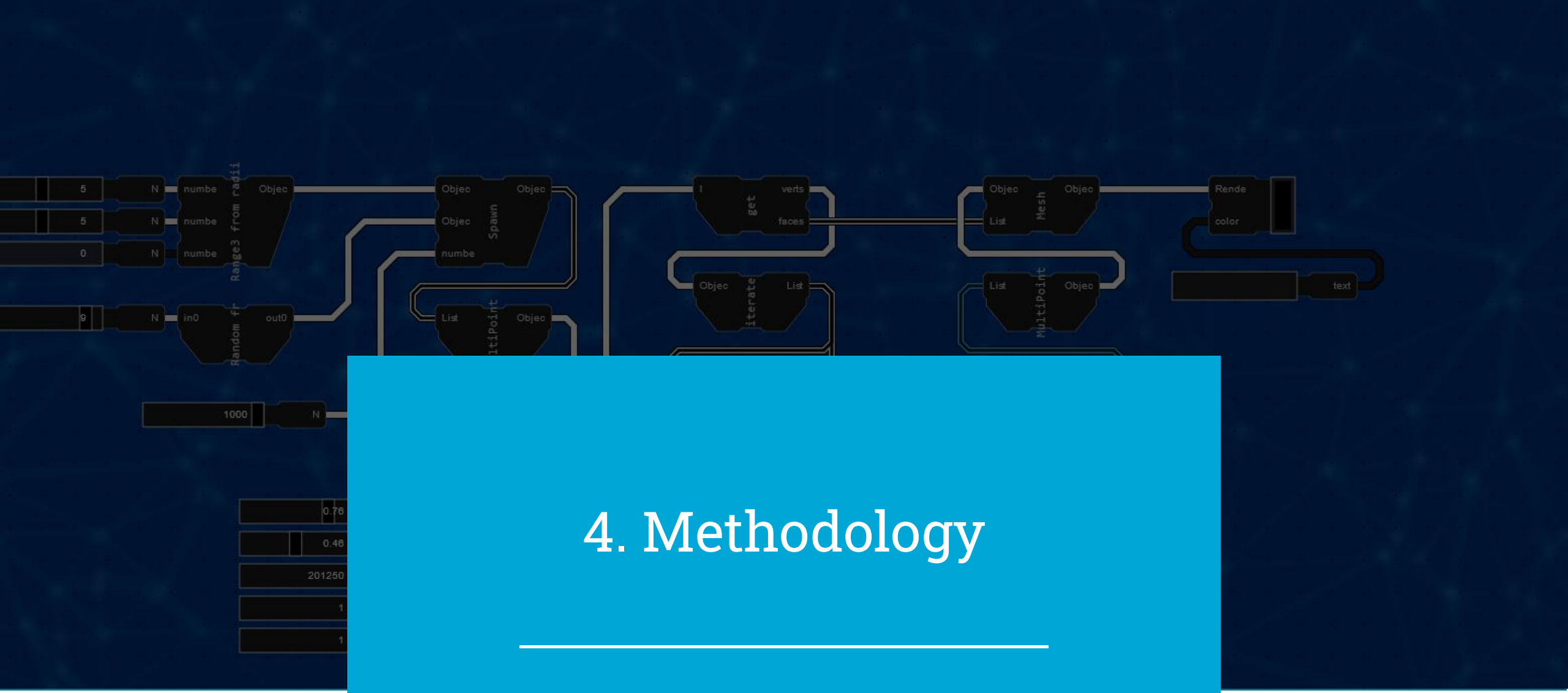
3. WebAssembly

Use case 2: Generic library binding

- Interface Types
- “run anything anywhere”



Clark, L. (2019)

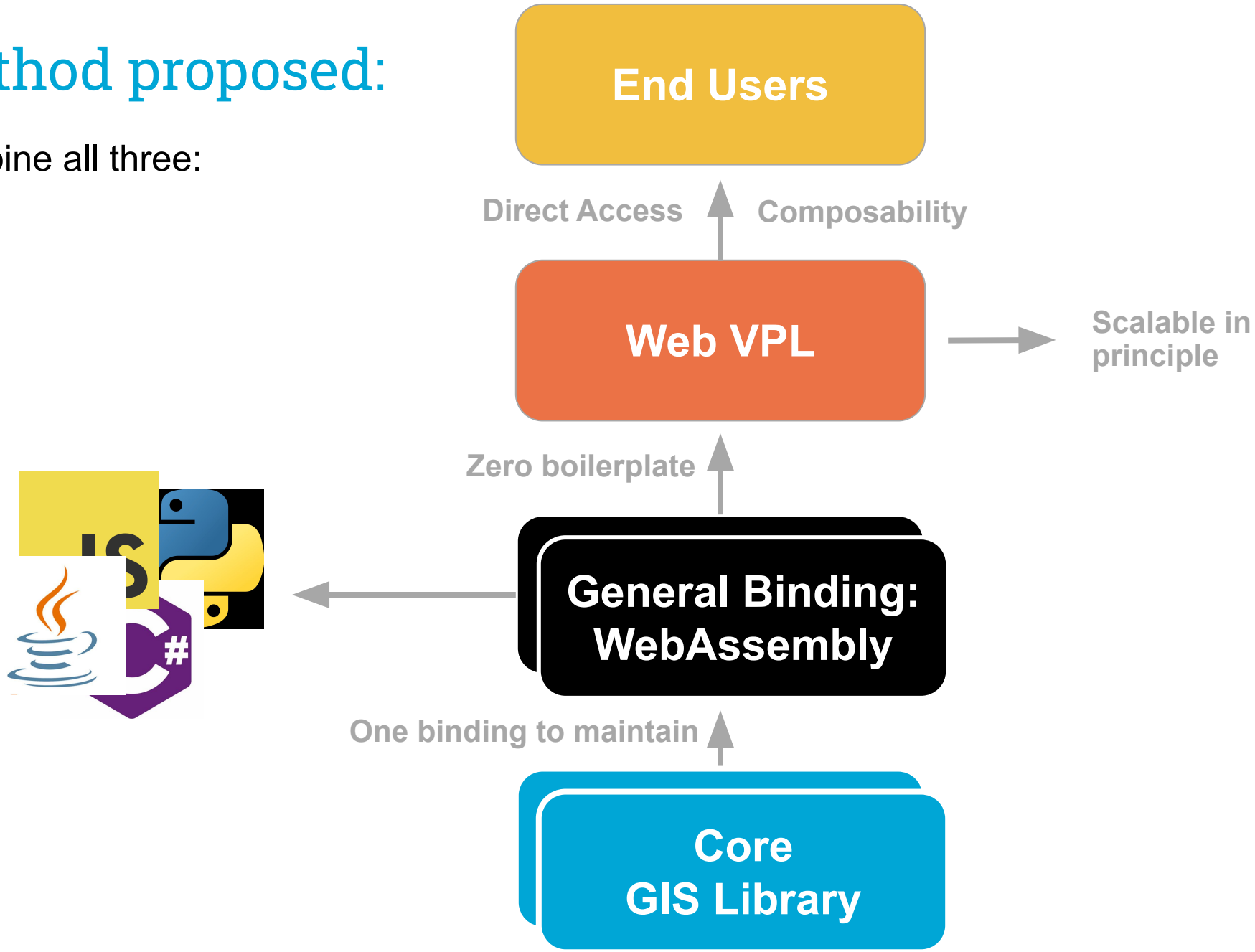


4. Methodology

TU Delft

method proposed:

Combine all three:

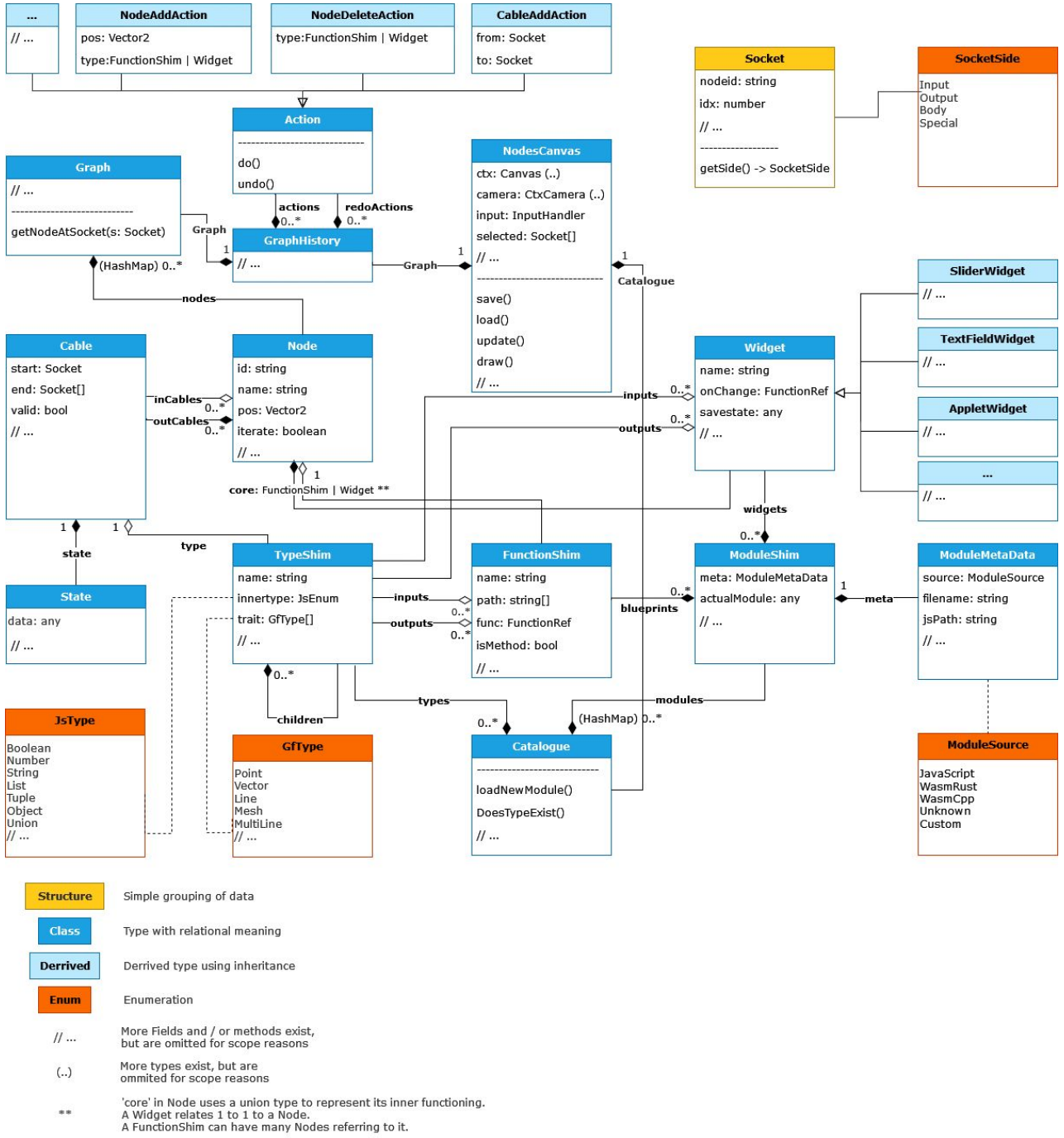


Study Overview:

1. Web VPL
2. Library Plugin system
    - Plugin loader
    - Plugin model
3. Tests
    - Test the plugin system with libraries written in different languages.
    - Compare functionality with existing VPLs
    - Use the environment in various scenarios

1. Web VPL: Design

- Essentially, a language
- Model View Controller
  - Model : syntax tree
- Shims

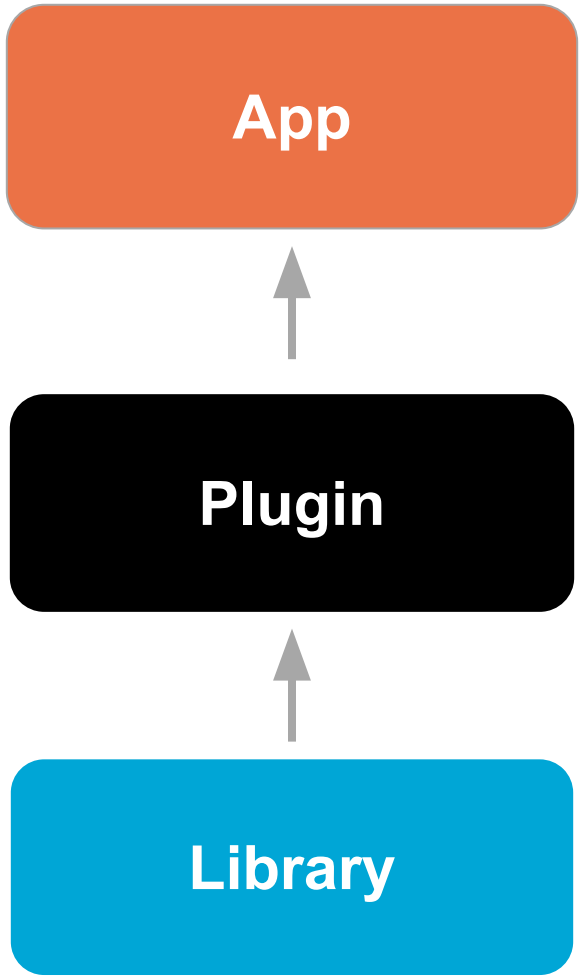


TU Delft

2. Plugin System

Regular case:

- Maintain separate project
- Explicitly state interface
- ‘boilerplate’



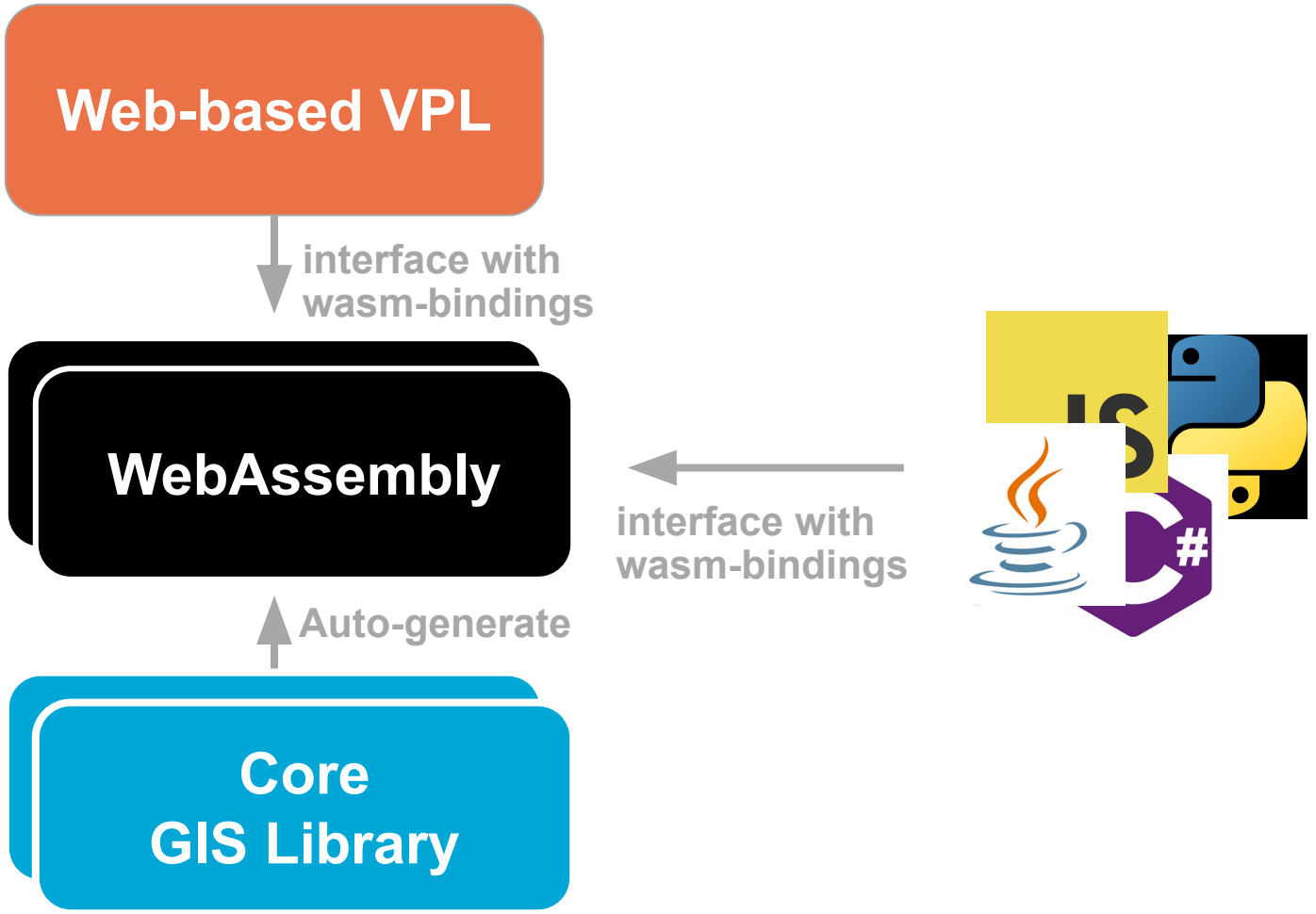
2. Plugin System

Our case:

- Leverage wasm compilers
- Mimic normal language
- Interprete bindings implicitly

Leads to:

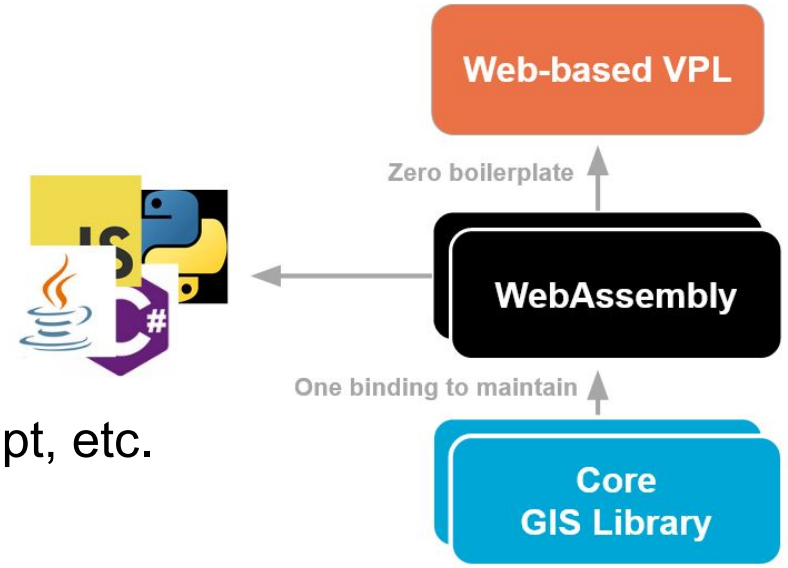
- No boilerplate
- Connect to existing infrastructure



2. Plugin System

Three elements:

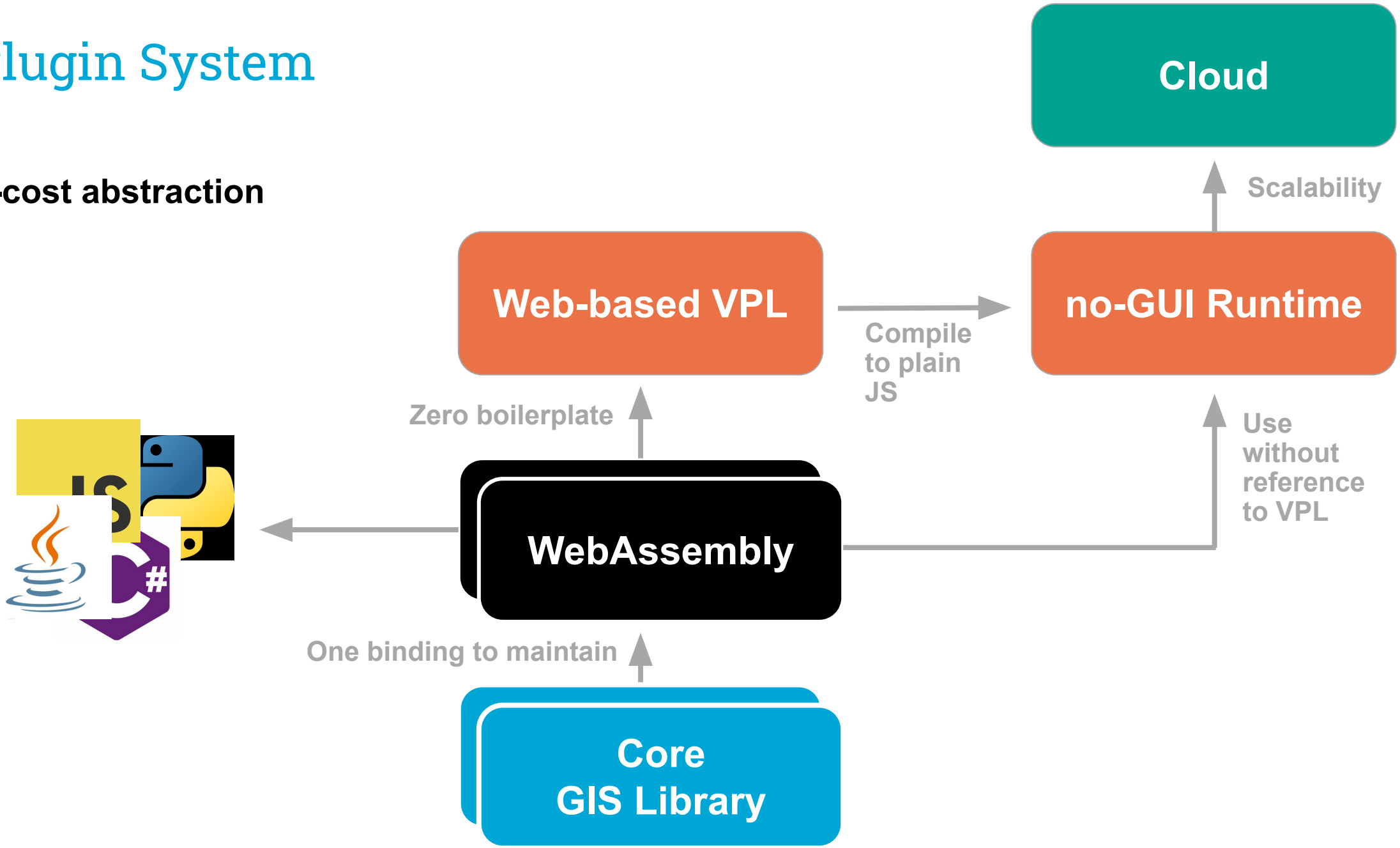
- **Direct utilization**→ Zero boilerplate
  - Leverage generic interface properties of WebAssembly
- **Portability**
  - Same behavior within this VPL as in python, C#, JavaScript, etc.
- **Scalability**
  - Zero-cost abstraction

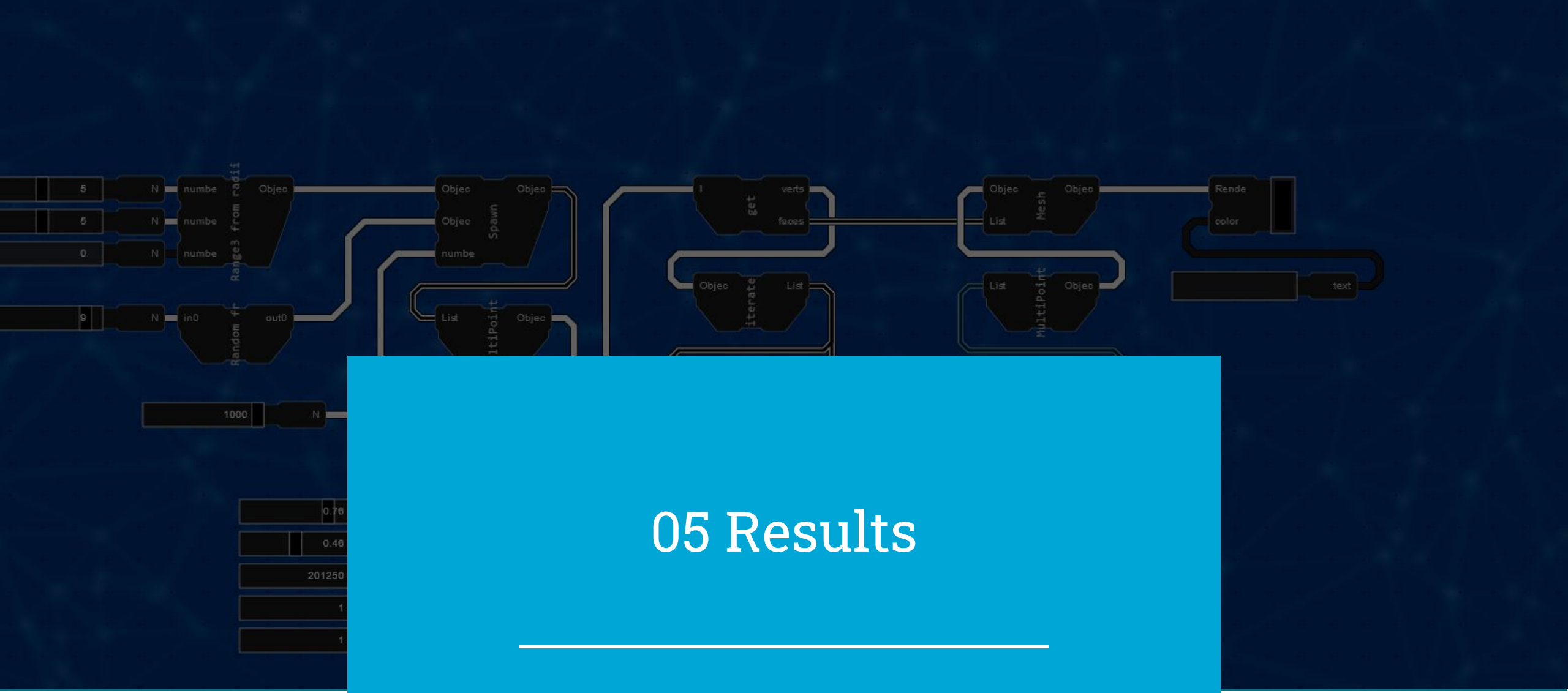


TU Delft

2. Plugin System

Zero-cost abstraction





05 Results

TU Delft

1. Web VPL implementation:
Geofront

2. Library Plugin System implementation

TU Delft

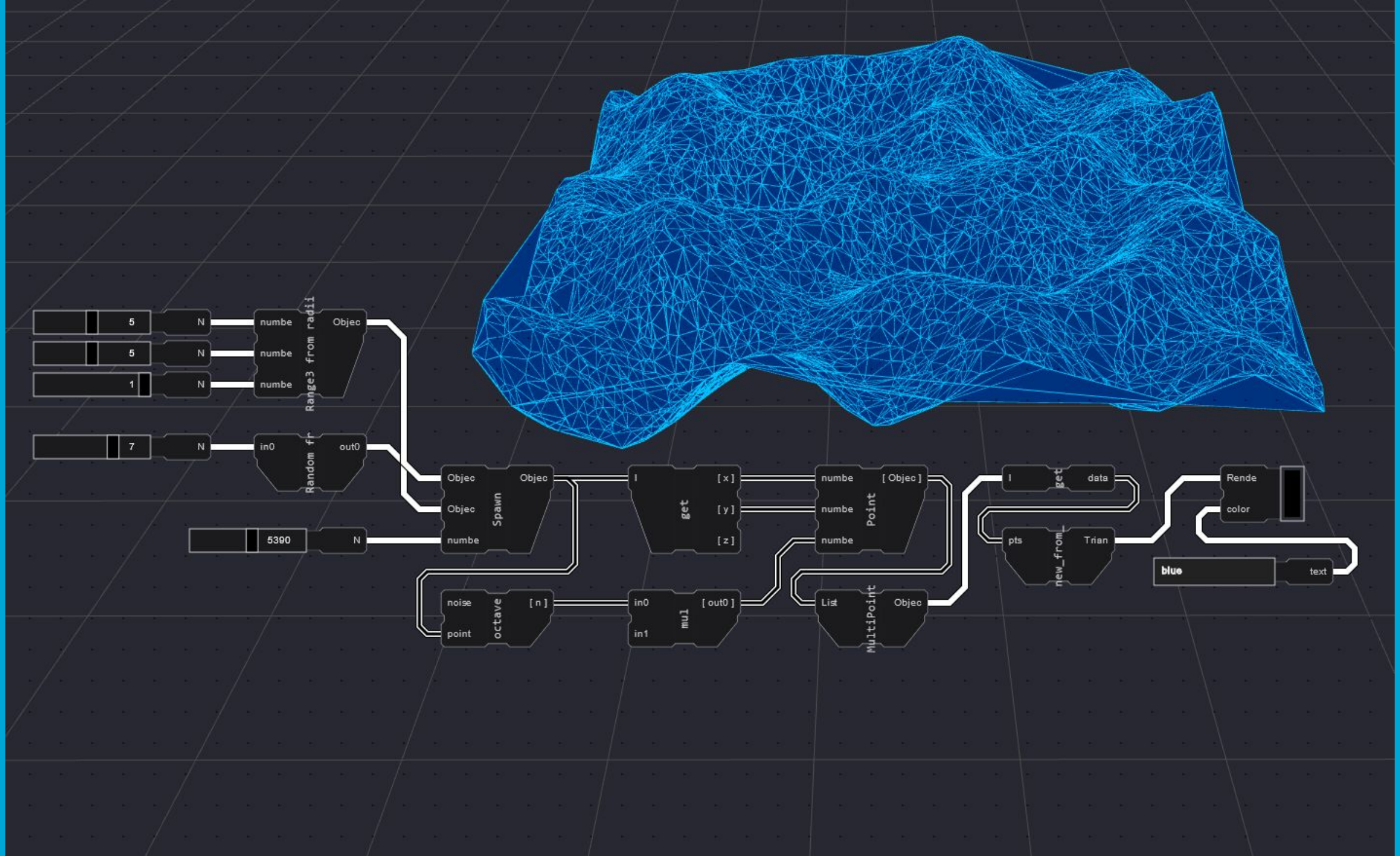
1. Web VPL implementation: **Geofront**

TU Delft

Web VPL: Geofront

Custom implementation:

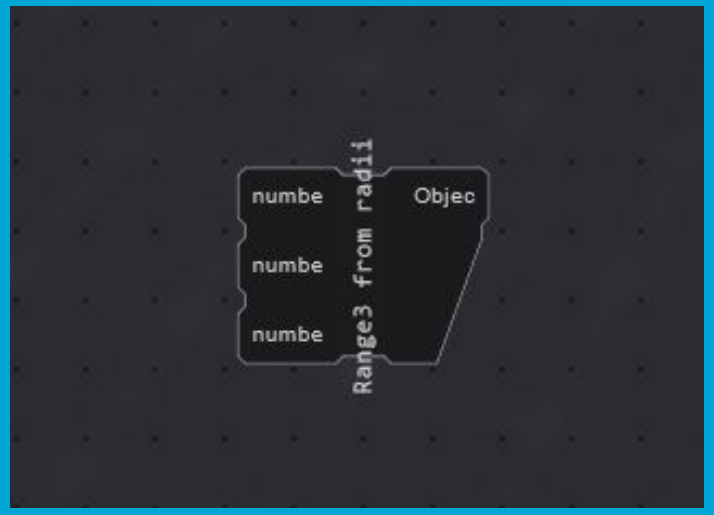
- Application Framework
- VPL model implementation
- renderer, Interaction, UI, etc.


5
N
5
N
1
N
numbe
numbe
numbe
Range3 from radii
Objec
7
N
in0
Random fr
out0
5390
N
Objec
Objec
numbe
Spawn
Objec
l
[x]
[y]
[z]
get
numbe
numbe
numbe
[Objec]
Point
noise
point
octave
[n]
in0
in1
mul
[out0]
List
MultiPoint
Objec
l
get
data
pts
new_from
Trian
Rende
color
blue
text

Main components

Node

Computation | Function

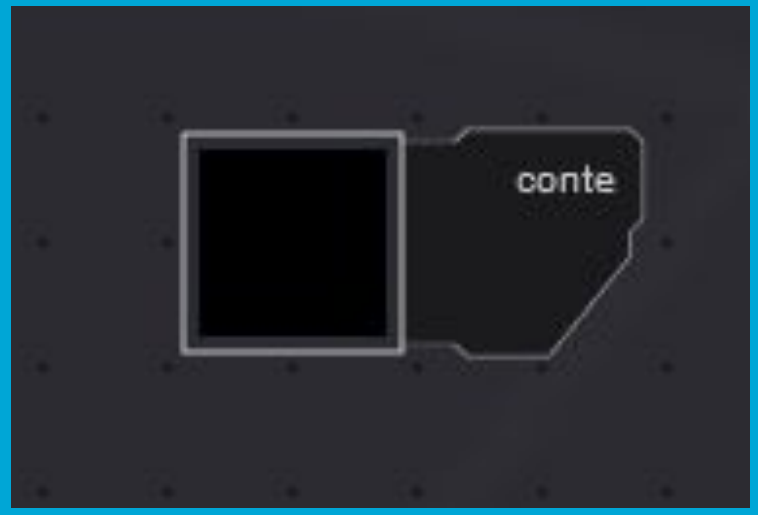


Cables

Variable | edges

Widget

GUI | Input Output



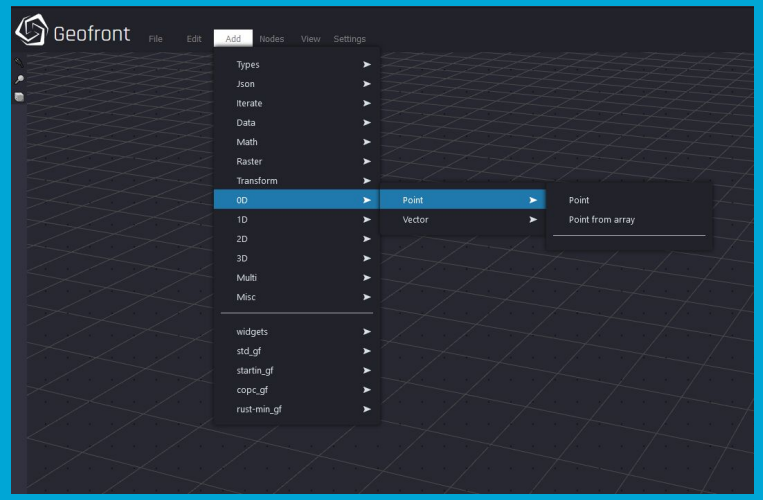
TU Delft

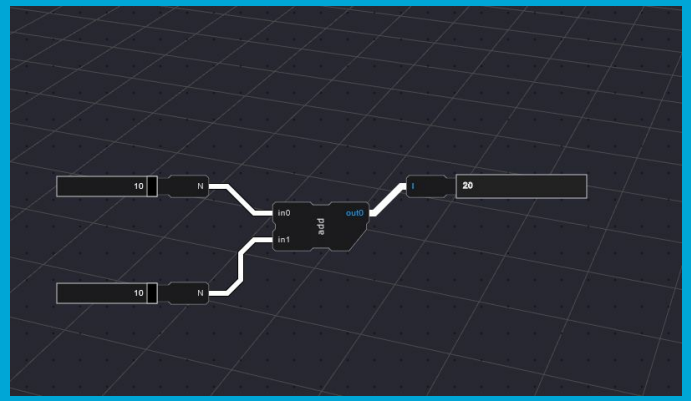
Workflow

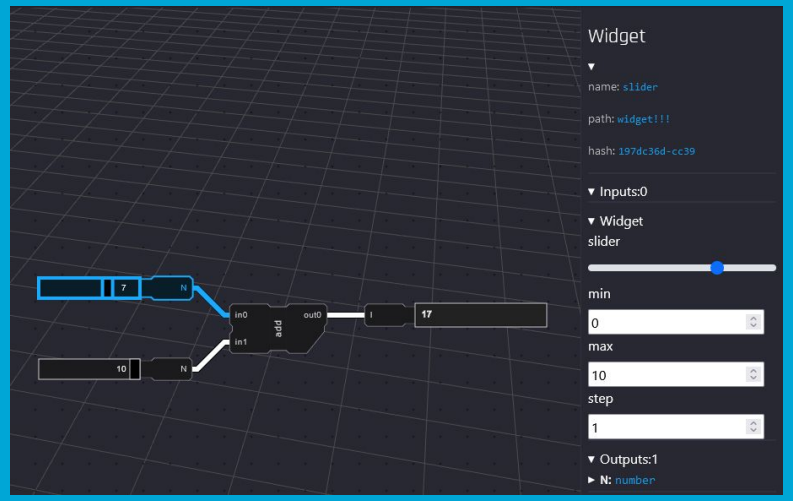
1. Add a node or widget from ‘add’ dropdown or fuzzy finder

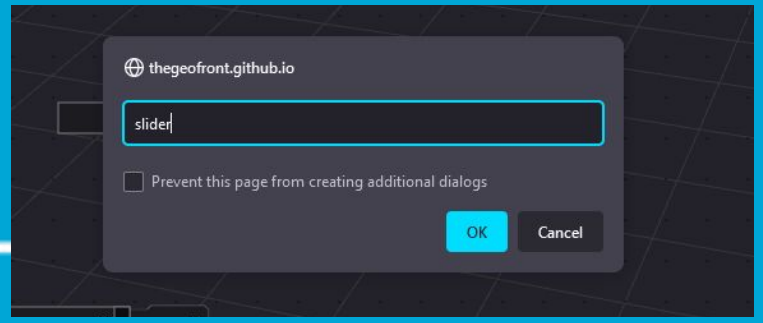
2. Connect nodes by dragging input to output sockets, to form graphs

3. To perform calculations, manipulate the input widgets using the canvas GUI, or a side menu



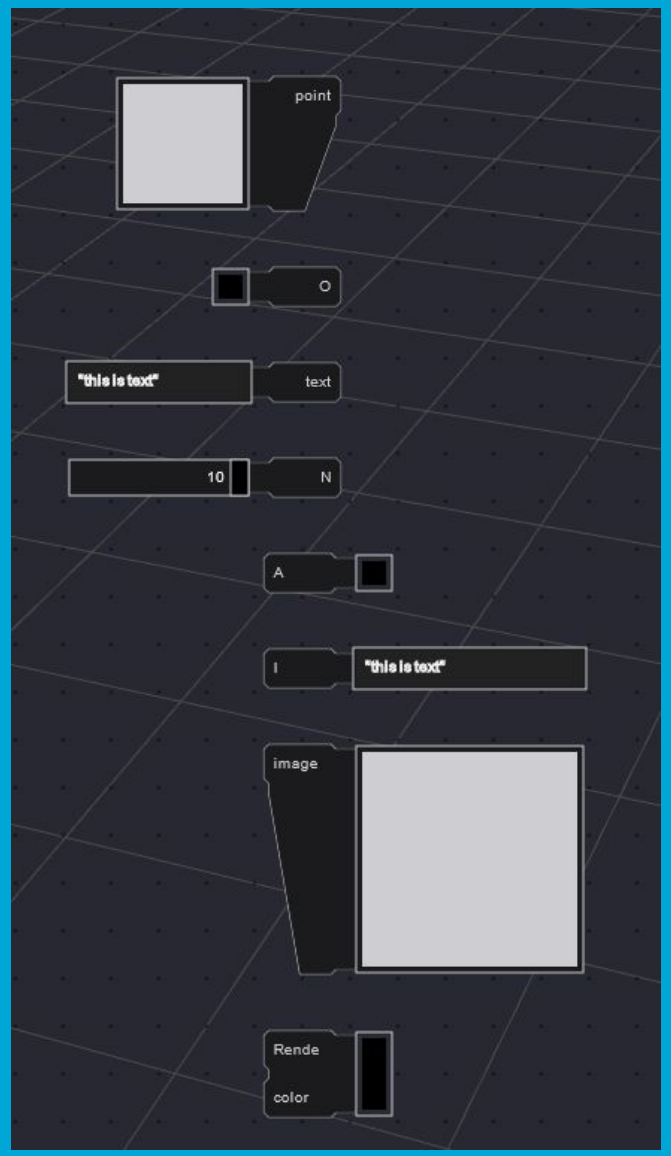


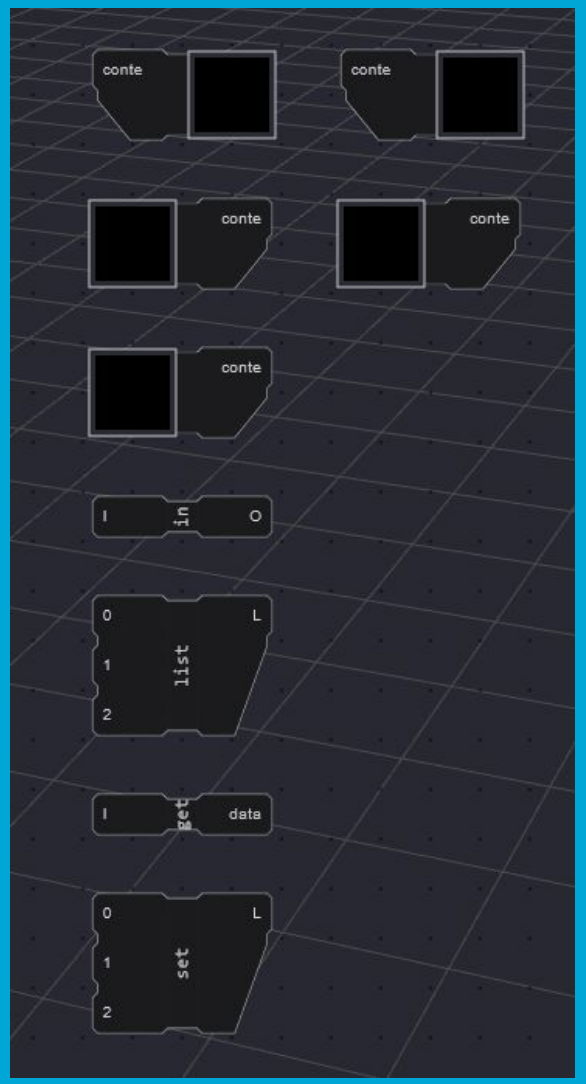




Widgets: Composable GUI

- “Applet”
- Boolean input
- Text field
- Number slider
- Boolean output
- Text output
- Image output
- Renderer





- File save as Blob | String
- File load as Blob | String
- File fetch as Blob | String
- Print to console
- Create list
- Get all properties from object
- Create object from properties

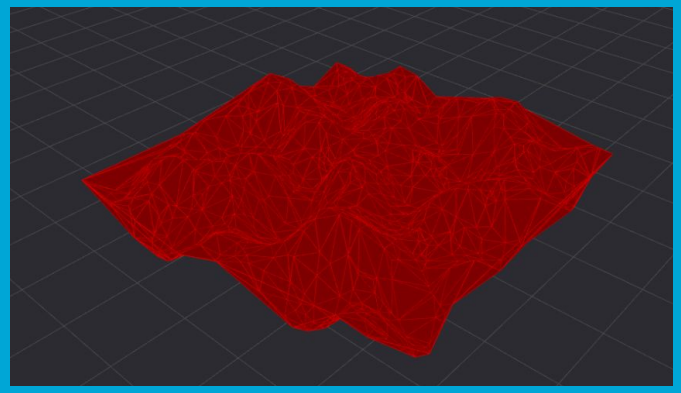
TU Delft

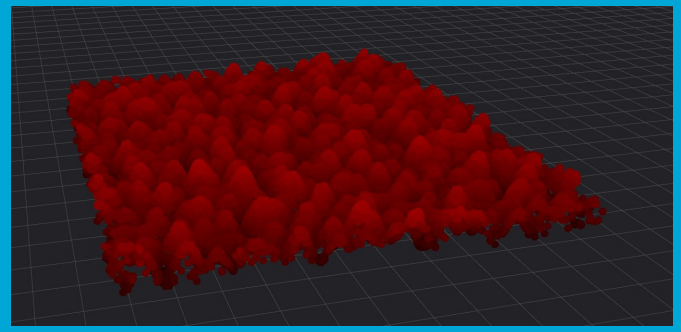
Visualization

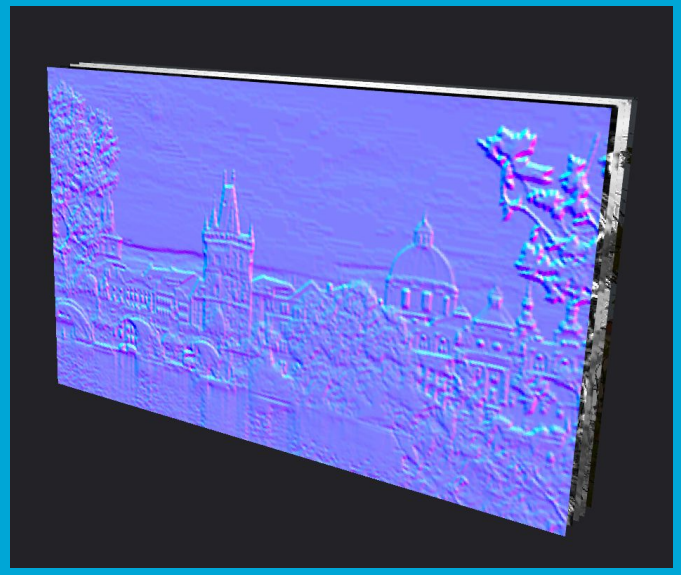
- Custom WebGL implementation

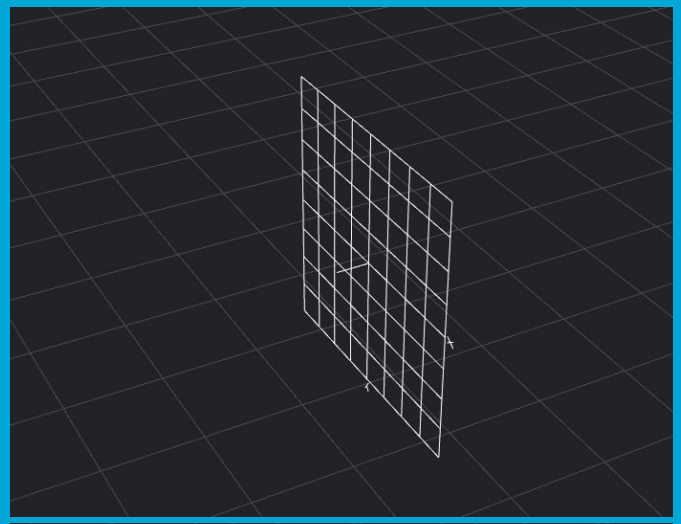
Support for:

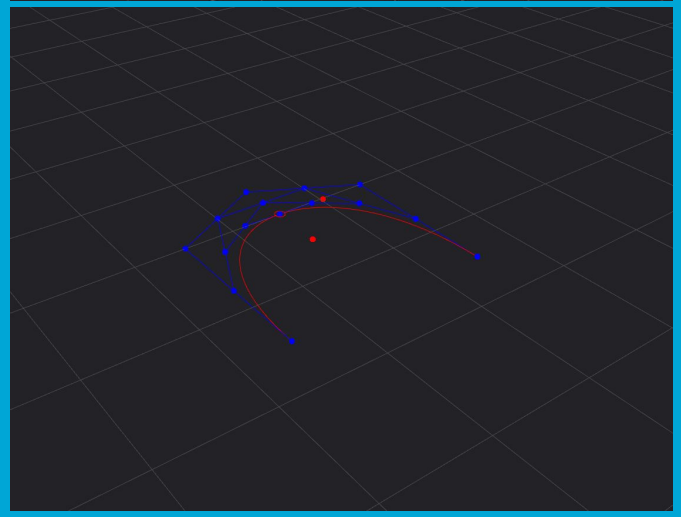
- Mesh
- Pointcloud
- Textures (images)
- Plane
- Bezier curve
- Bezier surface

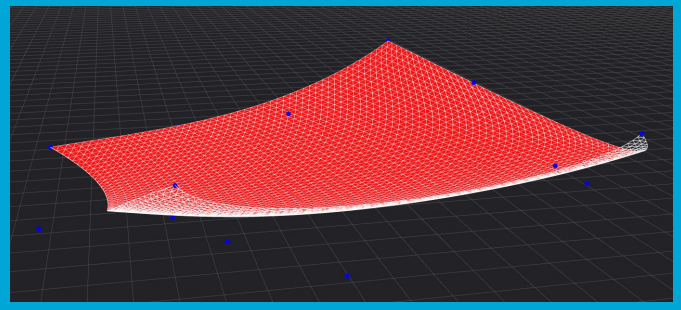






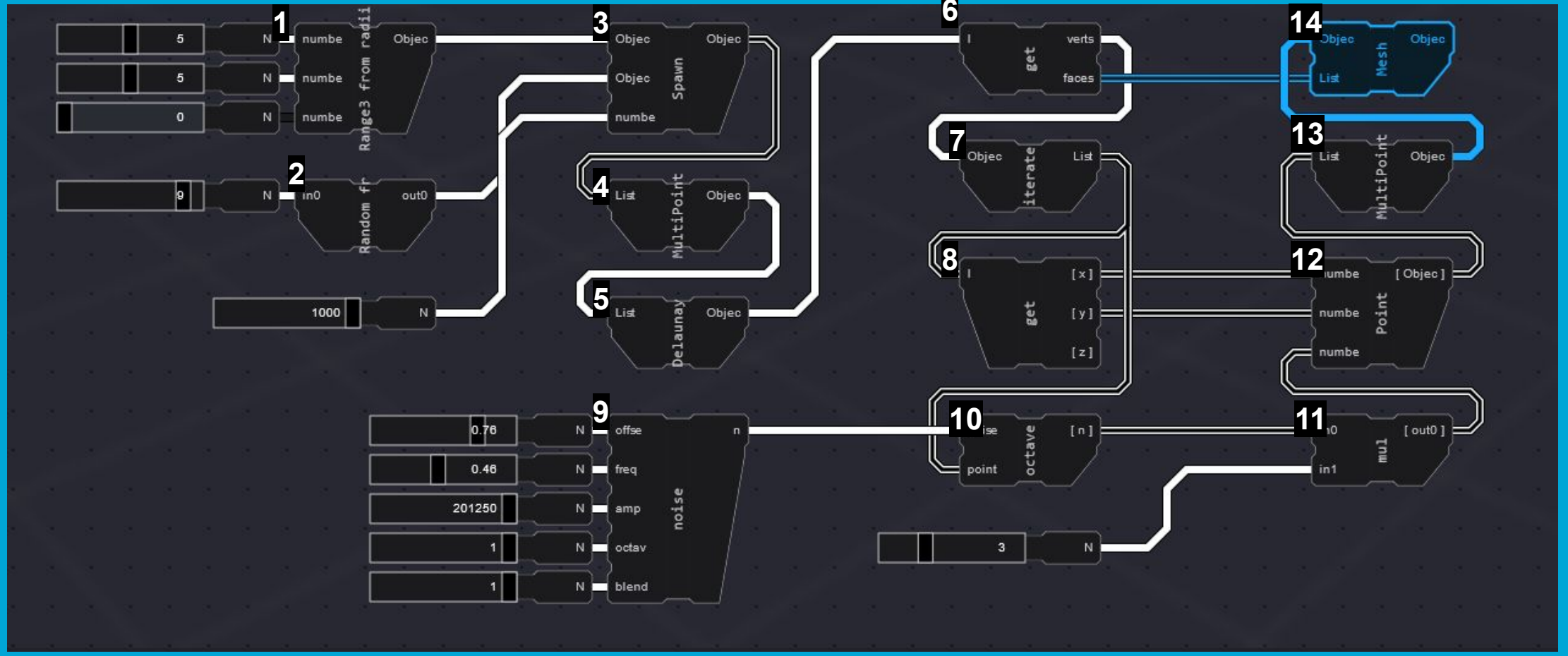






TU Delft

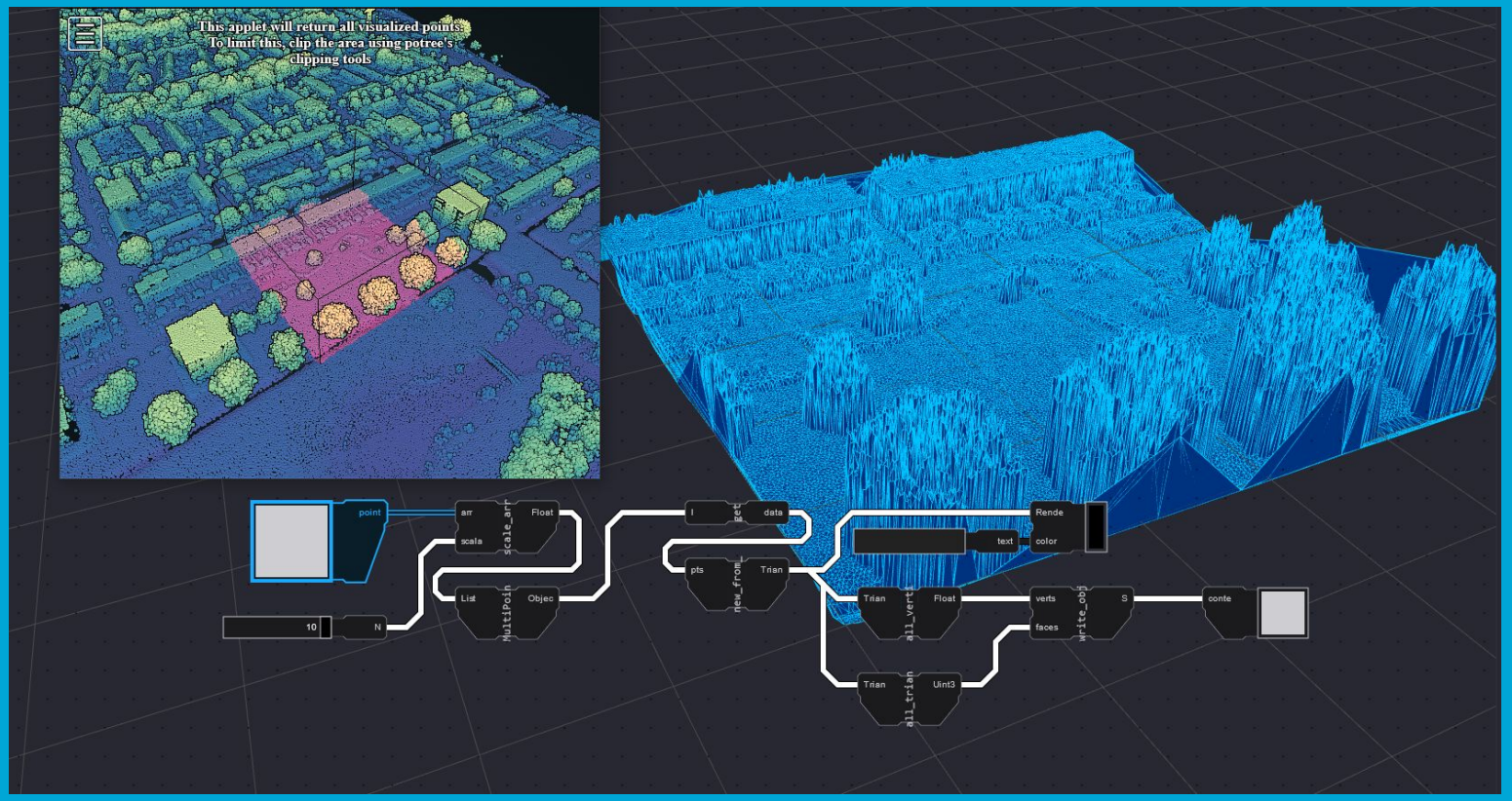
Calculation → Dependency sorting (kahn's algorithm)



TU Delft

Applet widget: sub-application support

Use output of one application, as input for Geofront



Video 1: Basic interaction

https://thegeofront.github.io/presentation/videos/geofront-1.mp4

TU Delft

Video 2: Basic composition & data inspection

https://thegeofront.github.io/presentation/videos/geofront-2.mp4

TU Delft

Video 3: A larger setup & parametrization

https://thegeofront.github.io/presentation/videos/geofront-3.mp4

TU Delft

Video 4: Geodata input → Obj output

https://thegeofront.github.io/presentation/videos/geofront-4.mp4

TU Delft

Geofront: Results

+ All major features implemented
+ Application composability

- Limited STD
- Types not interoperable
- Limited performance

TU Delft

Geofront: Feature comparison

Unique combination

	Grasshopper	*Blender*	*Mobius*	*Geoflow*	*Geofront*
Plugin support	**Yes**	No*	No	**Yes**	**Yes**
Plugin language	C#	-	-	C++	Rust/Js/Ts**
Plugin types	Partially	No	No	Unknown	**Yes**
Headless runtime	No	No	No	**Yes**	No
Web based	No	No	**Yes**	No	**Yes**
Base GIS Nodes	No	No	**Yes**	**Yes**	No
GUI nodes	**Yes**	**Yes**	No	No	**Yes**

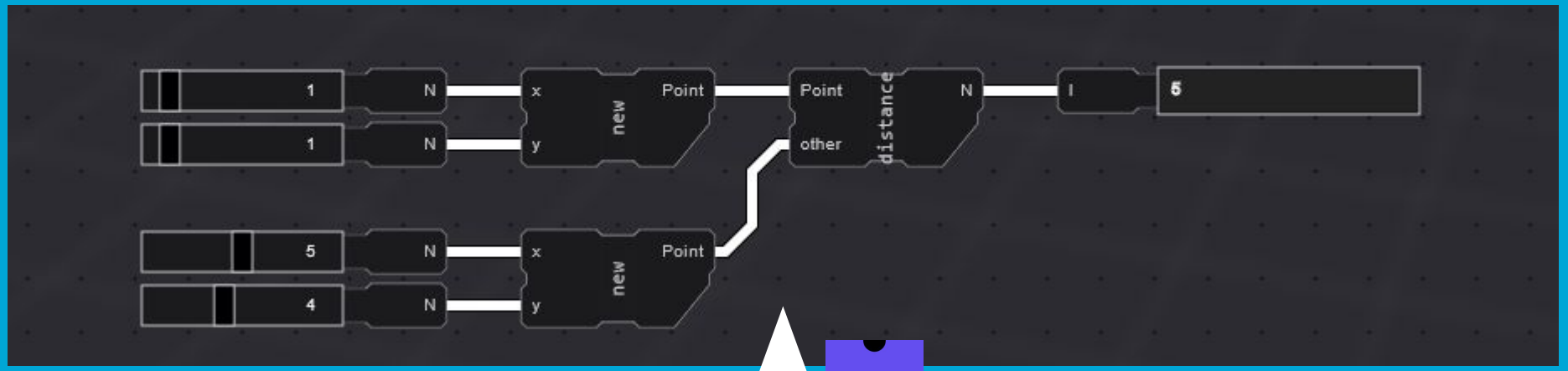
2. Library Plugin System implementation

Plugin System: Results

Automated extraction of:

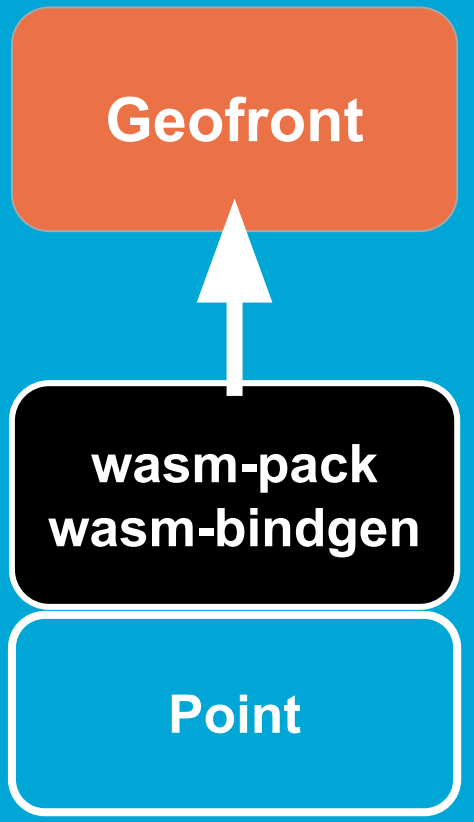
- A list of all functions present in the library
- A list of all custom types (structs / classes) present in the library
- Per function:
  - A list of all input parameters, name and type
  - An output type

Plugin System: Results



WA

```rust
#[wasm_bindgen]
pub struct Point {
    x: f32,
    y: f32,
}

#[wasm_bindgen]
impl Point {

    pub fn new(x: f32, y: f32) -> Self {
        Self { x, y }
    }

    pub fn distance(&self, other: &Self) -> f32 {
        ((self.x - other.x).powi(2) + (self.y - other.y).powi(2)).powf(0.5)
    }
}
```



Plugin System: Results

```
namespace MyPlugin
{
    public class AdderNode : GH_Component
    {
        public ComponentNodeFromString()
          : base("Add Integers",
            "Add",
            "This component adds two integer values",
            "My Plugin",
            "My Plugin Category")
        {
        }

        protected override void RegisterInputParams(GH_Component.GH_
    InputParamManager pManager)
        {
            pManager.AddIntegerParameter("a", "value A", GH_ParamAccess.item);

            pManager.AddIntegerParameter("b", "value B", GH_ParamAccess.item);

        }

        protected override void RegisterOutputParams(GH_Component.GH_
    OutputParamManager pManager)
        {
            pManager.AddIntegerParameter("R", "result", GH_ParamAccess.item);
        }

        protected override void SolveInstance(IGH_DataAccess DA)
        {
            int a;
            int b;
            DA.GetData(0, ref a);
            DA.GetData(1, ref b);
            int c = a + b;
            DA.SetData(0, c);
        }

        public override Guid ComponentGuid
        {
            get { return new Guid("197d2ec4-c3b1-47ed-8355-6af3b7612f01"); }
        }
    }
}
```

```
class AddNode : public Node
{
public:
  using Node::Node;

  void init()
  {
    add_input("a", typeid(int));
    add_input("b", typeid(int));
    add_output("result", typeid(int));
  }

  std::string info()
  {
    std::string s;
    if (output("result").has_data())
      s = std::to_string(output("result").get<int>());
    return s;
  }

  void process()
  {
    auto in1 = input("a").get<int>();
    auto in2 = input("b").get<int>();
    std::this_thread::sleep_for(std::chrono::microseconds(200));
    output("result").set(int(in1 + in2));
  }
};
```

Figure 55: Geoflow plugin

```
#[wasm_bindgen]
fn add(a: i32, b: i32) -> i32 {
    a + b
}
```

Figure 56: Geofront plugin

TU Delft

Plugin System: Tests

C++ → emscripten → WebAssembly

lib.cpp M

plugins > cpp-min-gf > src > lib.cpp

```cpp
// quick_example.cpp
#include <emscripten/bind.h>
#include <cmath>

using namespace emscripten;

float add(float left, float right) {
    return left + right;
}

class Point {
public:
    double x;
    double y;

    Point(double x, double y) :
        x(x),
        y(y) {}

    double distance(Point& other) {
        return std::pow(
            std::pow(x - other.x, 2) + std::pow(y - other.y, 2),
            0.5);
    }
};

EMSCRIPTEN_BINDINGS(cpp_min) {
    function("add", &add);
    class_<Point>("Point")
        .constructor<double, double>()
        .function("distance", &Point::distance)
        .property("x", &Point::x)
        .property("y", &Point::y);
}
```

Rust → wasm-pack → WebAssembly

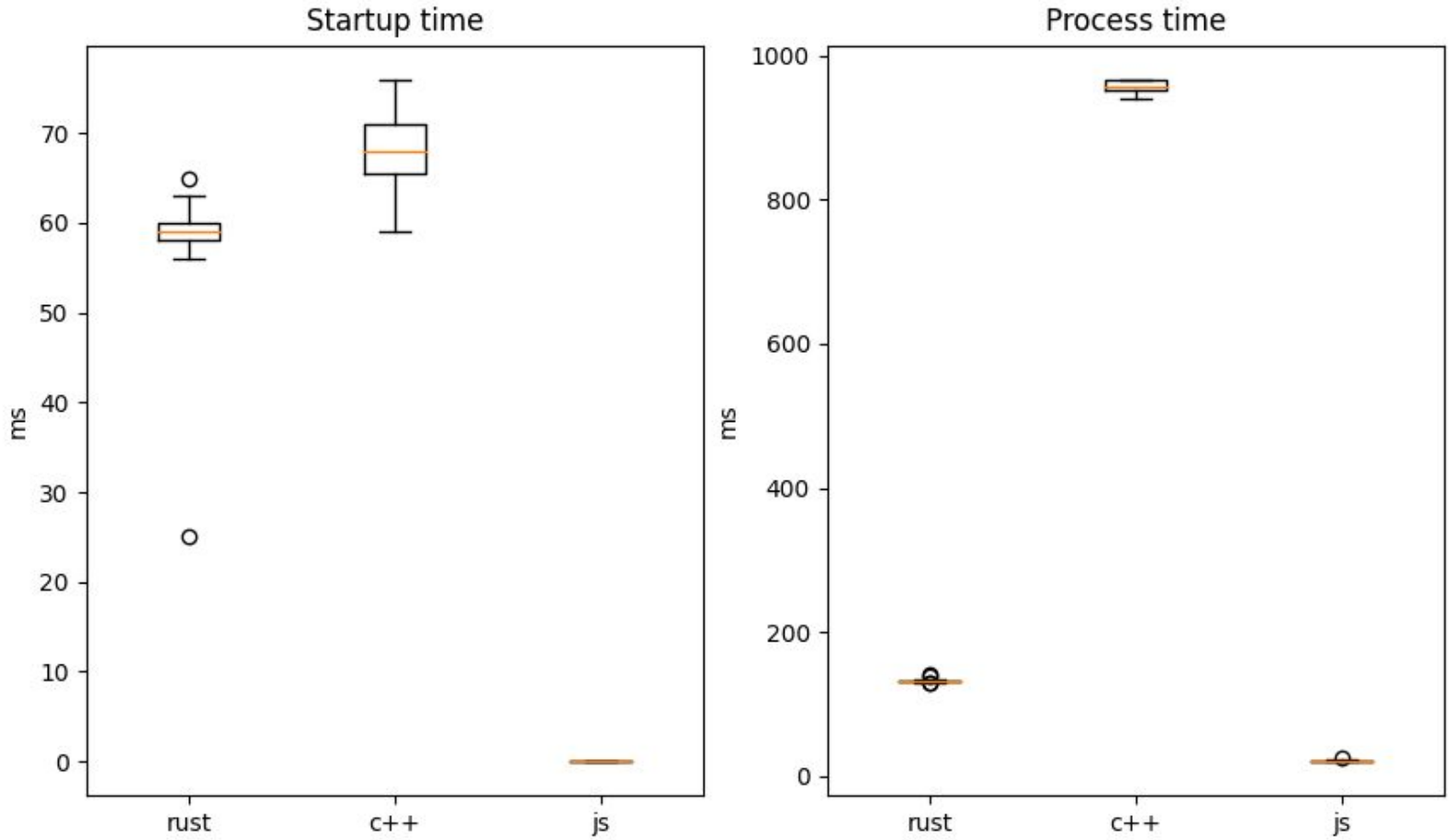
lib.rs

plugins > rust-min-gf > src > lib.rs > {} impl Point

```rust
use wasm_bindgen::prelude::*;

#[wasm_bindgen]
pub fn add(left: usize, right: usize) -> usize {
    left + right
}

#[wasm_bindgen]
pub struct Point {
    x: f32,
    y: f32,
}

#[wasm_bindgen]
impl Point {

    pub fn new(x: f32, y: f32) -> Self {
        Self { x, y }
    }

    pub fn distance(&self, other: &Self) -> f32 {
        ((self.x - other.x).powi(2) + (self.y - other.y).powi(2)).powf(0.5)
    }
}
```

Plugin System: Tests

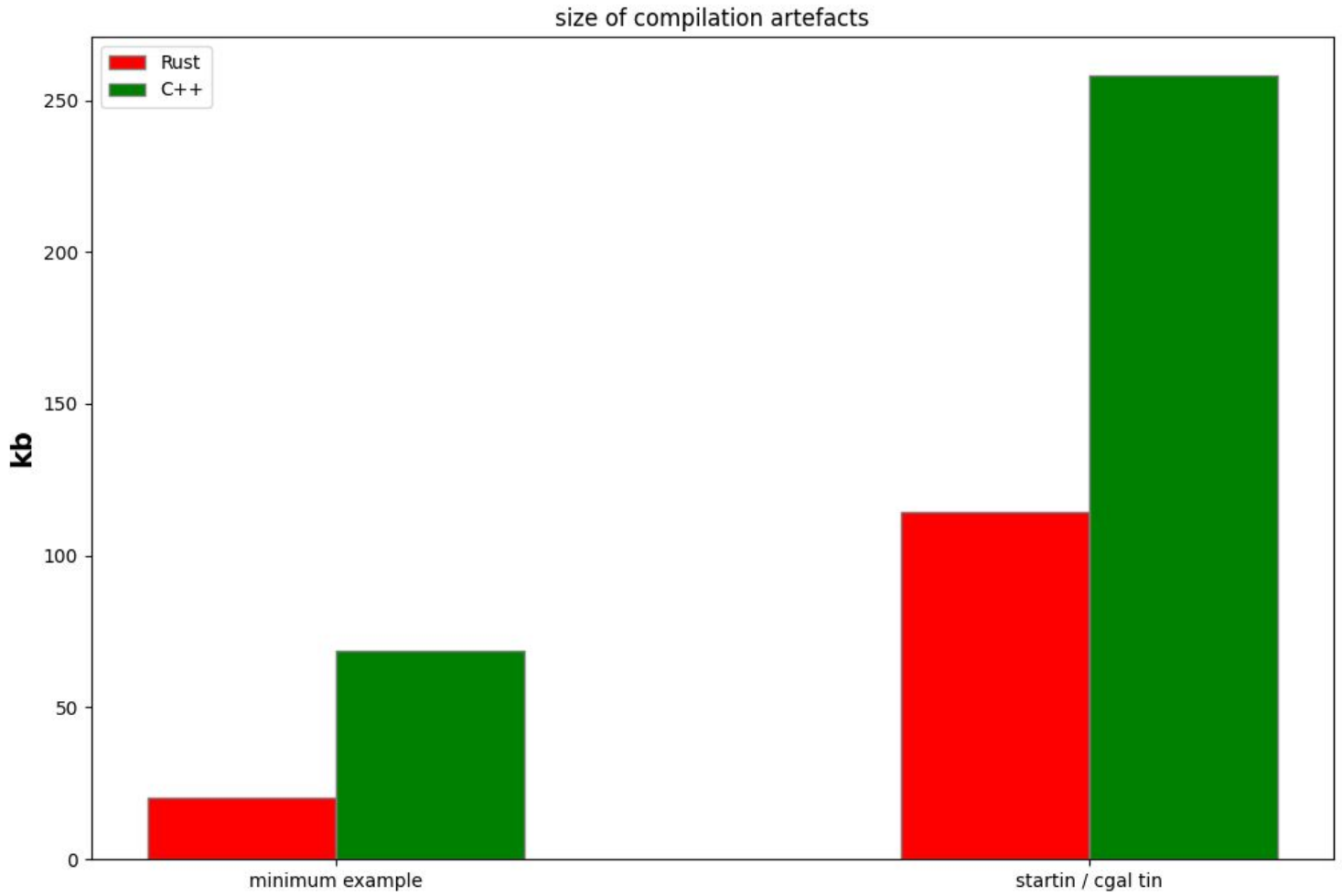
Interfacing this binary with JavaScript was around **six** times as slow compared to the rust equivalent.



TU Delft

Plugin System: Tests

the emscripten compiler produced a binary which requires more than **three** times the size of the same functionality compiled with wasm-pack.



TU Delft

Plugin System: Test Results

Rust

Worked almost immediately for almost any library

- \+ Expressive bindings allow complex data types to be exchanged in a simple manner.
- \- *Still some runtime overhead due to wrappers*

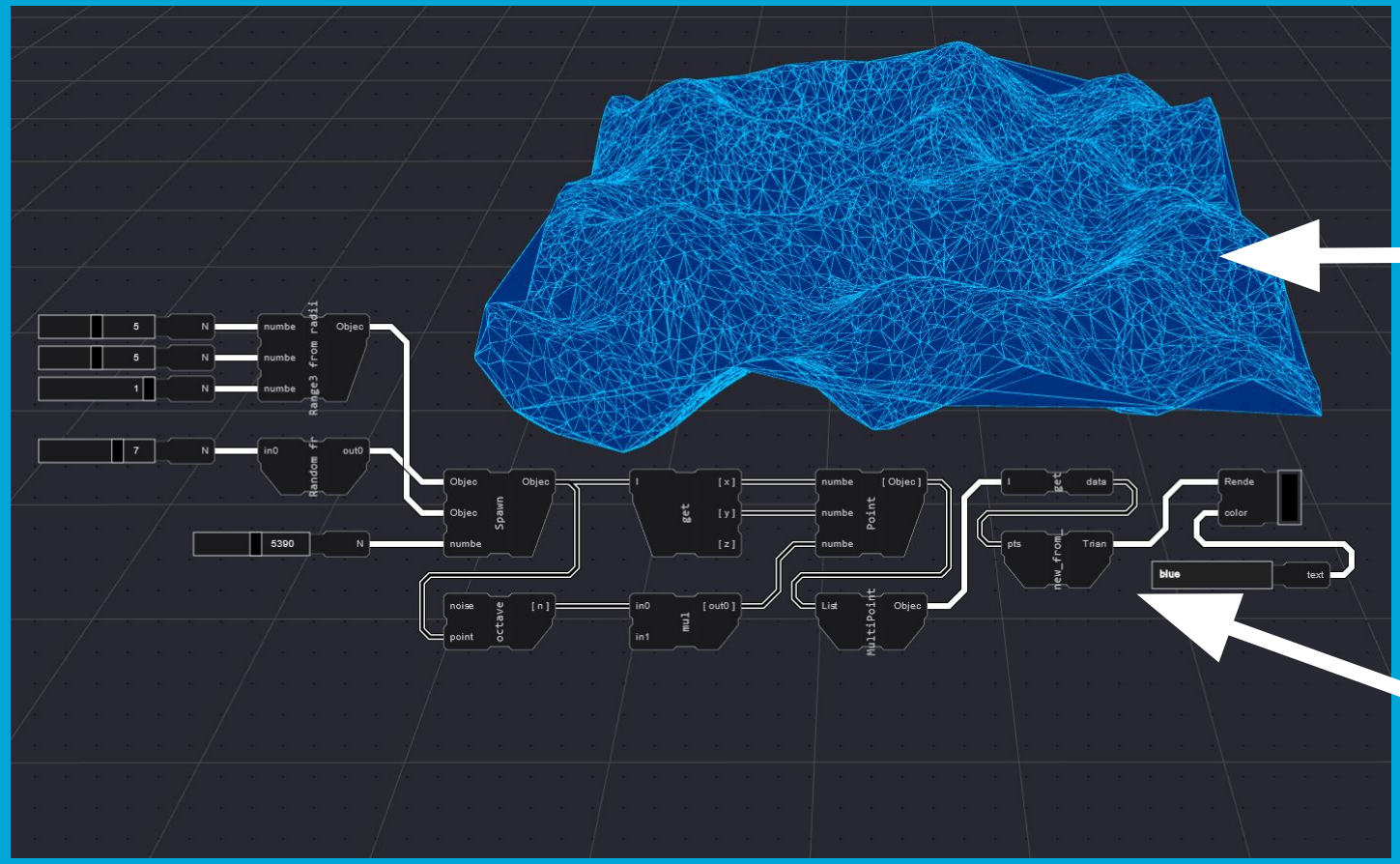
C++

Multiple workarounds eventually allowed some parts of CGAL to be run in geofront, if included in the source code

- \- *Requires many workarounds*
- \- *More wrapper overhead than rust*
- \- *Larger binaries than rust*
- \- *Sub-optimal support for bindings*
  - \+ Interface Types will most likely be added in the future to emscripten

TU Delft

Plugin System: Tests: startin

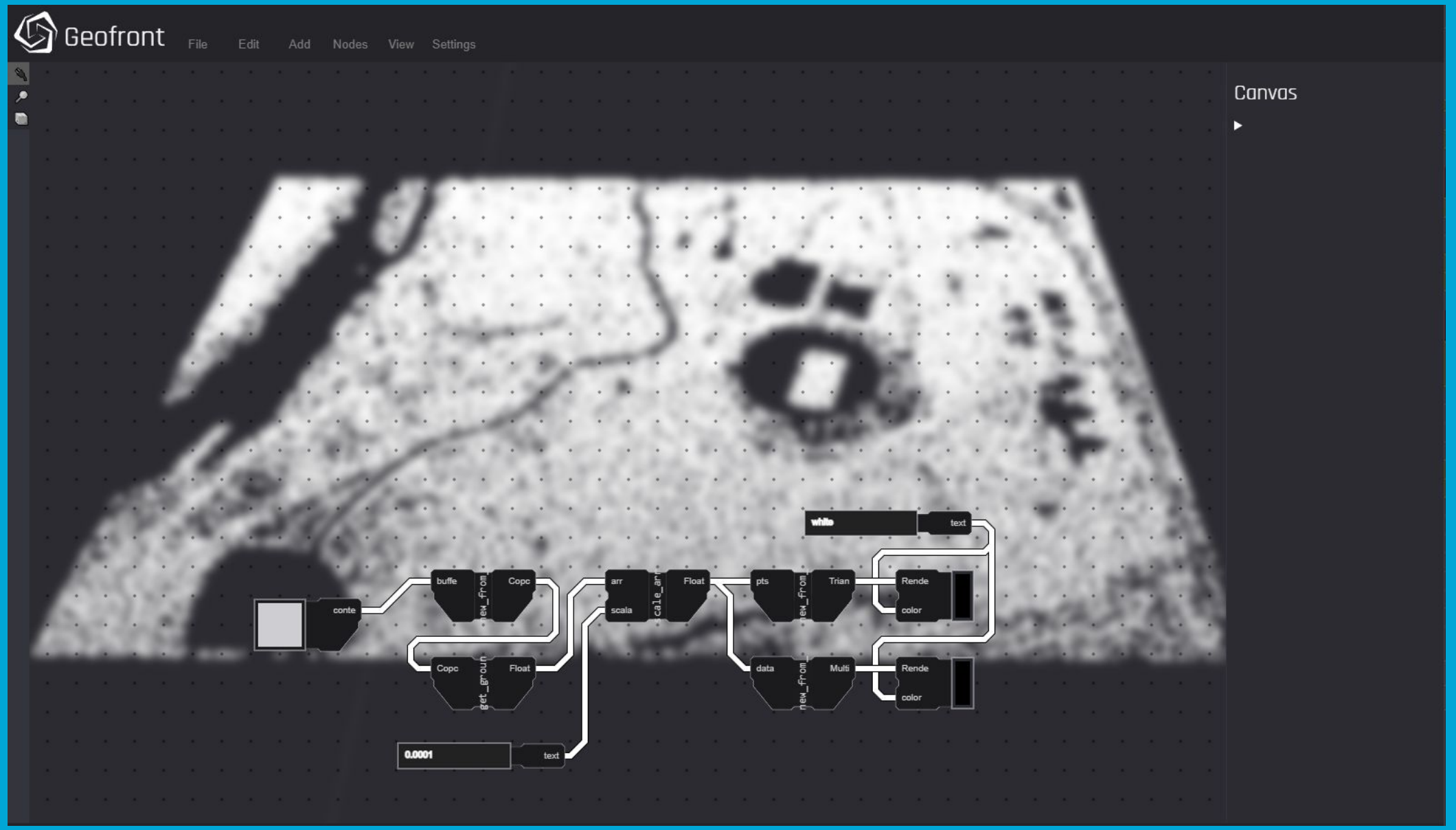



```rust
// impl Renderable for Triangulation
#[wasm_bindgen]
impl Triangulation {

    pub fn gf_has_trait_renderable() -> bool {
        true
    }

    pub fn gf_get_shader_type() -> GeoShaderType {
        GeoShaderType::Mesh
    }

    pub fn gf_get_buffers(&self) -> JsValue {
        let buffer = MeshBuffer {
            verts: self.all_vertices(),
            cells: self.all_triangles(),
        };
        serde_wasm_bindgen::to_value(&buffer).unwrap()
    }
}
```

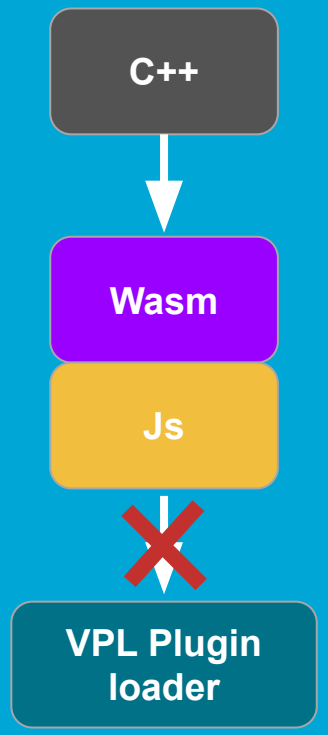
```rust
#[wasm_bindgen]
pub struct Triangulation {
    dt: startin::Triangulation,
}

#[wasm_bindgen]
impl Triangulation {

    pub fn new_from_vec(pts: Vec<f64>) -> Triangulation {
        let mut tri = Triangulation::new();
        tri.insert(pts);
        tri
    }

    pub fn new() -> Triangulation {
        let dt = startin::Triangulation::new();
        Triangulation { dt }
    }
```

TU Delft

Rust Library: copc-rs (Point cloud loader)



C++ Library: CGAL



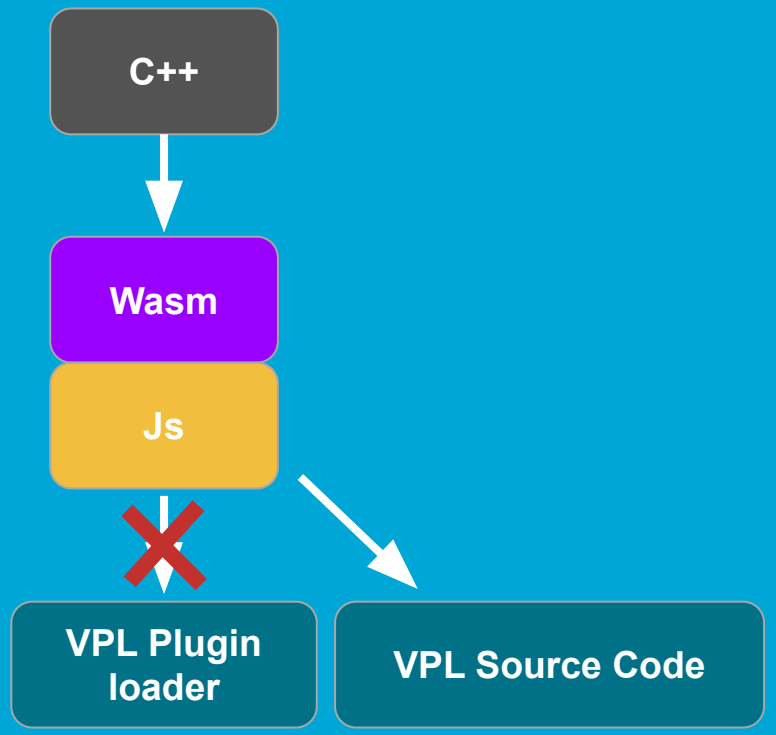
lib.cpp M

plugins > cpp-min-gf > src > lib.cpp

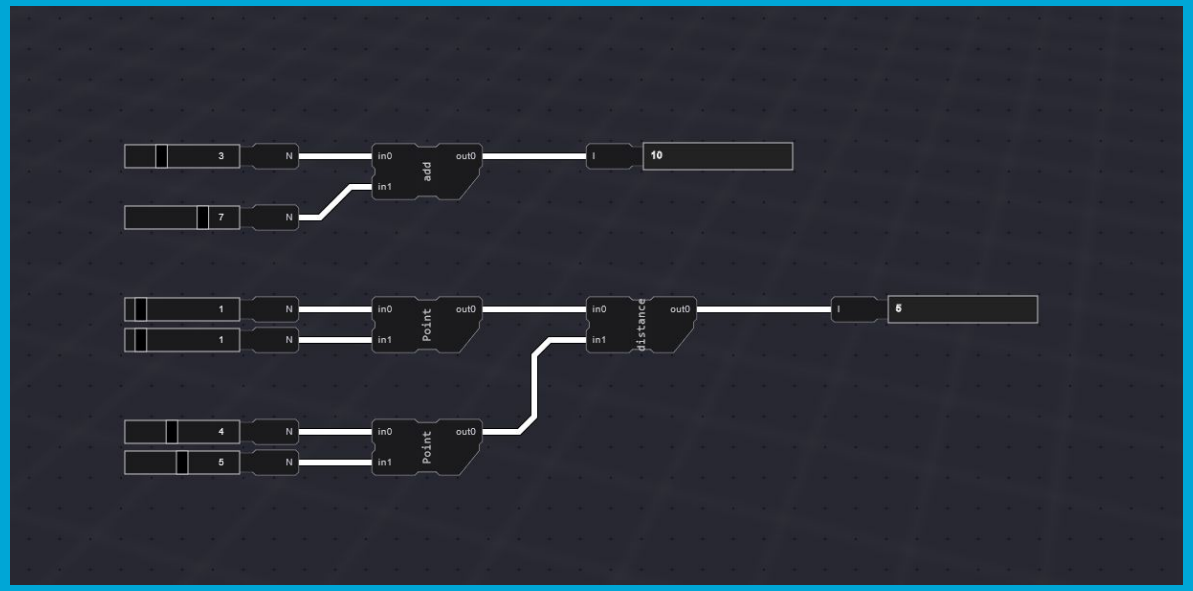
```cpp
// quick_example.cpp
#include <emscripten/bind.h>
#include <cmath>

using namespace emscripten;

float add(float left, float right) {
    return left + right;
}

class Point {
public:
    double x;
    double y;

    Point(double x, double y) :
        x(x),
        y(y) {}

    double distance(Point& other) {
        return std::pow(
            std::pow(x - other.x, 2) + std::pow(y - other.y, 2),
            0.5);
    }
};

EMSCRIPTEN_BINDINGS(cpp_min) {
    function("add", &add);
    class_<Point>("Point")
        .constructor<double, double>()
        .function("distance", &Point::distance)
        .property("x", &Point::x)
        .property("y", &Point::y);
}
```

TU Delft

C++ Library: CGAL

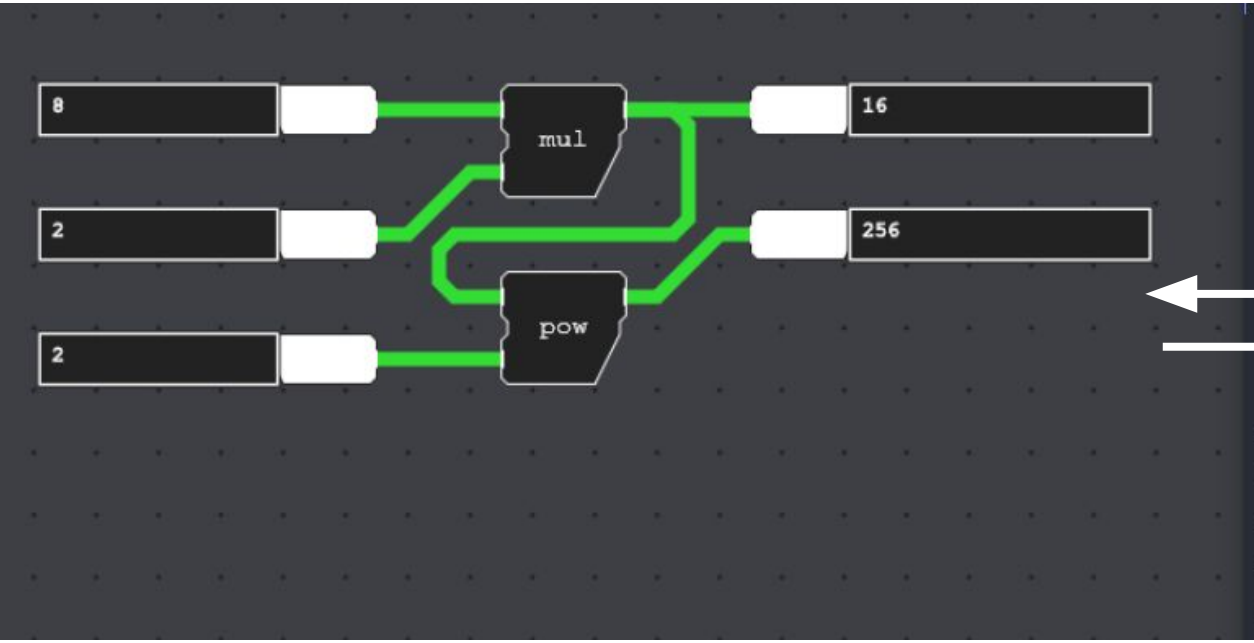



```cpp
// quick_example.cpp
#include <emscripten/bind.h>
#include <cmath>

using namespace emscripten;

float add(float left, float right) {
    return left + right;
}

class Point {
public:
    double x;
    double y;

    Point(double x, double y) :
        x(x),
        y(y) {}

    double distance(Point& other) {
        return std::pow(
            std::pow(x - other.x, 2) + std::pow(y - other.y, 2),
            0.5);
    }
};

EMSCRIPTEN_BINDINGS(cpp_min) {
    function("add", &add);
    class_<Point>("Point")
        .constructor<double, double>()
        .function("distance", &Point::distance)
        .property("x", &Point::x)
        .property("y", &Point::y);
}
```



Plugin System: Zero cost abstraction runtime

Incomplete, but promising



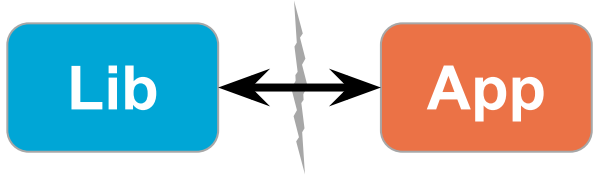

```
function anonymous(
  a /* "widget": "text" | "state": "2" | "x": 2 | "y": -10 */,
  b /* "widget": "text" | "state": "2" | "x": 2 | "y": -12 */,
  c /* "widget": "text" | "state": "8" | "x": 2 | "y": -14 */
) {
    let [d] = math.mul(c, b) /* "x": 6 | "y": -14 */;
    let [e] = math.pow(d, a) /*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

TU Delft

06 Conclusion

TU Delft

sub Q: library & application divide



1. Libraries cannot be directly used, end users are dependent on in-between applications →

+ VPL acts as “a custom GUI for any library”
+ Only dependent on Wasm-bindings
- Exception: C++

2. Applications are not further composable →

+ VPL: Use tools in a composable manner
+ Potree demo: further composable web applications, connect to startin lib

3. Capabilities get lost in in-between steps →

+ Plugin system: Minimum in-between steps
+ Wasm-bindings only limiting factor
- Functional style required

TU Delft

A: All aspects were able to be addressed to an extent

Main research question:

Q: **Is a web based VPL a viable method for directly accessing native GIS libraries with a composable interface?**

Yes

- Provides solutions for app / lib divide
- Unique set of features for GIS lib usage:
  - no-boilerplate plugin system
  - Composable GUI
  - Web-based

No

- More research is required to proof full feasibility:
  - C++ → Interface Types
  - GUI-less runtime → Scalability

A: **Yes**, but with exceptions

TU Delft

Future work

- Improved VPL model:
  - Improved type support
  - Validated dataflow VPL
  - Concurrency
  - Compile to WebAssembly
- Deployment & scalability
  - Cloud-based execution
  - “Deploy as application”
- (Rust as replacement for C++ in GIS or any scientific endeavor)
  - Less error-prone, improved library management, improved wasm support → distribution

Sources:

- Elliott, C. (2007). Tangible functional programming. International Conference on Functional Programming. http://conal.net/papers/Eros/ Accessed 2022-09-27. Related Talk: https://www.youtube.com/watch?v=faJ8N0giqzw.

- 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.

- w3c (2019). World Wide Web Consortium brings a new language to the Web as WebAssembly becomes a W3C Recommendation. https://www.w3.org/2019/12/pressrelease-wasm-rec.html.en

- Clark, L. (2019). WebAssembly Interface Types: Interoperate with All the Things. Mozilla Hacks: the Web developer blog. https://hacks.mozilla.org/2019/08/webassembly-interface-types/

- Kuhail, M. A., Farooq, S., Hammad, R., and Bahja, M. (2021). Characterizing Visual Programming Approaches for End-User Developers: A Systematic Review. IEEE Access, 9:14181–14202.

- Sousa, T. (2012). Dataflow Programming: Concept, Languages and Applications. Unpublished.

TU Delft

Thank you for your attention!

Questions


5
N
numbe
Objec
Range3 from radii
5
N
numbe
0
N
numbe
9
N
in0
out0
Random fr
1000
N
Objec
Objec
Objec
numbe
Spawn
List
Objec
MultiPoint
List
Objec
Delaunay
I
verts
get
faces
Objec
List
iterate
I
[x]
get
[y]
[z]
Objec
Objec
List
Mesh
List
Objec
MultiPoint
numbe
[Objec]
numbe
Point
numbe
Rende
color
text
0.76
N
offse
n
0.46
N
freq
201250
N
amp
1
N
octav
1
N
blend
noise
noise
[n]
point
octave
in0
[out0]
in1
mul
3
N

Discussion:


5
N
numbe
Objec
Range3 from radii
5
N
numbe
0
N
numbe
9
N
in0
out0
Random fr
1000
N
Objec
Objec
Objec
numbe
Spawn
List
Objec
MultiPoint
List
Objec
Delaunay
I
verts
get
faces
Objec
List
iterate
I
[x]
get
[y]
[z]
Objec
Mesh
List
List
Objec
MultiPoint
Rende
color
text
numbe
[Objec]
numbe
Point
numbe
0.76
N
offse
n
0.46
N
freq
201250
N
amp
1
N
octav
1
N
blend
noise
noise
[n]
octave
point
in0
[out0]
mul
in1
3
N

Why can’t an existing, generic VPL's be used? In other words, what is a "GIS VPL" in relationship to 'any' computer graphics VPL?

- In one word: scalability.

- The scale of geodata is an important characteristic of GIS.

- To a large extend, a GIS VPL will require the same features as a VPL for other fields which can be considered 'applied CG'.

    - Construct and view geometry, large collection of transformation components, UI components, etc.

- However, A stark difference can be found in the need for scalability.

    - While development of GIS tools and geo-pipelines often occur on a small scale, the eventual execution of these tools often needs to happen on sizable datasets.

- This means that any VPL which claims to be 'aimed at GIS', will need to at least provide an answer on how it can be scaled to handle massive datasets.

TU Delft

Why does the field of GIS need a(nother) VPL?

- In GIS, VPLs are almost exclusively used as server-side or native ETL tools.

- However, VPLs can serve many other purposes, which are currently not being explored.

- One possible application of a VPL in GIS which has not been explored, is to use a VPL for web-based, serverless geocomputation.

- Such an application would contribute to making geocomputation libraries more directly usable, and web applications more composable.

Is this science?

I think It is valuable to be alert of all ‘trivia’ within scientific research:

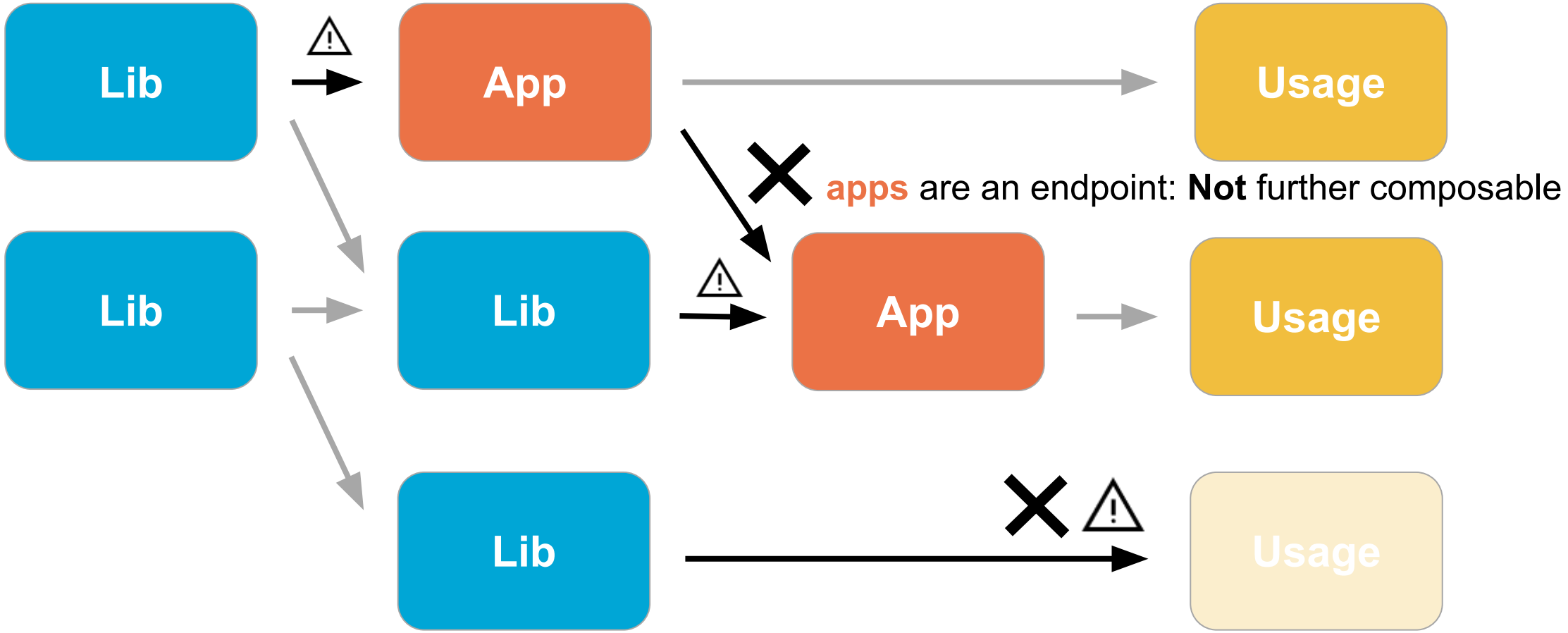
- If trivia becomes overwhelming component of a study, it should be solved
- whether or not that counts as science, I don’t know.
- I do know it is in the interest of science to solve those problems.
- Distribution & Accessibility are the trivial problems I tried to solve, once and for all, for all subsequent research. That was the goal I set out to accomplish at least.

- BME example (lies)

Leftovers:


5
N
numbe
Objec
Range3 from radii
5
N
numbe
0
N
numbe
Objec
Objec
Objec
Spawn
numbe
g
N
in0
out0
Random fr
List
MultiPoint
Objec
1000
N
List
Delaunay
Objec
I
verts
get
faces
Objec
List
iterate
I
[x]
get
[y]
[z]
Objec
Mesh
List
Rende
color
List
MultiPoint
Objec
text
numbe
[Objec]
numbe
Point
numbe
0.76
N
offse
n
0.46
N
freq
201250
N
amp
1
N
octav
1
N
blend
noise
noise
[n]
octave
point
in0
[out0]
mul
in1
3
N

⚠ Some lib capabilities get lost when used in an app



⚠ A lib offers no visualization or GUI.

✕ A lib must be turned into an app before utilization.

TU Delft