Web-based GIS for simulation, and visualisation: EPANET and data-rich shapefiles

Gareth Lewis[1], Brett Snider[1], Lydia Vamvakeridou-Lyroudia[1,2], Albert S Chen[1], Slobodan Djordjević[1] and Dragan A Savić[1,2]

[1] Centre for Water Systems, University of Exeter, Harrison Building, North Park Road, Exeter, EX4 4QF, UK
[2] KWR Water Research Institute, Groningenhaven 7, P.O. Box 1072, 3430 BB Nieuwegein, the Netherlands

g.lewis2@exeter.ac.uk

Abstract. The management of water resources often involves with spatially and temporally varied information collected from various providers with different data formats. Although certain software or applications such as QGIS[1] and EPANET[2] have been developed to support the analyses and decision making, these solutions do have some critical weaknesses: 1) it can take users considerable time to become proficient in using a given application, 2) applications generally assume a level of domain-specific knowledge, 3) applications may require customisation through plug-ins to provide suitable information and 4) applications are often tied to specific hardware / operating system configurations. To address these issues, the aqua3s and Fiware4Water research programmes were developed as 'cloud first' projects, using the cloud/web to deliver functionality. In this work, we developed approaches for integrating and visualising information to support water management, specifically developing a web-based EPANET simulation and visualisation for large water networks (c30,000 EPANET nodes and links), and a web-based visualisation of regional flood data shapefiles for Trieste and surrounding regions. In both cases, data was processed and balanced between client and server to minimise client loading and maximise responsiveness.

1. Introduction

For those working with water systems, QGIS [1] and EPANET [2] provide fundamental visualisation and simulation platforms. QGIS allows users to visualise complex environments using a collection of layers built on top of vector or XYZ tile sets to view data graphically with map and/or satellite image contexts. Likewise, the EPANET application enables users to interactively build and simulate water networks using the EPANET hydrostatic solver to provide accurate simulation results.

Whilst these applications are incredibly useful, they can be criticised for having steep learning curves and, in the case of EPANET, only providing limited platform (Windows) support. For developers of water management systems, it is desirable to take core functionality from QGIS (layered map and data visualisation) and EPANET (water network visualisation and simulation) and present it to users in a way that can: 1) be used without the need for bespoke computers and 2) be used with little training or support.

The aqua3s project had requirements to take typical QGIS and EPANET visualisations and present them within a browser context, requiring visualisation through a suitable web-based tile-map visualisation solution.

2. Web-based tile-map visualisation

Web-based tile-maps describe the browser-based implementation of the vector and XYZ tile approaches of QGIS and describe an approach where geographic maps are split into collections of tiles allowing users to scroll over a map giving the impression of an endless map surface. Map tiles are subdivided into increasing levels of detail, to create the effect of zooming into or out of the map and this is provided for the web with services such as Google Maps, Microsoft Bing and Apple Maps.

For developers of tile-map visualisations there is an array of solutions available, all of which tend to deliver similar functionalities but often different business models. For our initial work, Google Maps[3], Mapbox[4], leaflet.js[5] and Maplibre[6] were considered, and programming was undertaken with all excluding Google Maps, given the credit card requirement of its commercial terms.

The remaining three frameworks were similar in their conceptualisations, which is not surprising given that Maplibre is an open-source drop-in for Mapbox and Leaflet.js also appears to have had some involvement with Mapbox. The frameworks work on the concept of a view being a collection of layers, which are either provided by the map tile server (ground, buildings, roads, labels etc) or provided by users, with five distinct components, Figure 1, which form the basis for user-defined visualisations.

Figure 1: Typical user-defined visualisations: default marker[7], GeoJSON geometries[8], animated images[9], video[10], WMS[11].

The default marker visualisation type can be thought of as the de facto visualisation/interaction style for web-based tile-maps and consists of a ‘pin’ in the map at a given location. Typically, the pin can be clicked by a user to reveal pin-specific information.

The GeoJSON geometry visualisation type is a more open-ended approach to visualisation/interaction and can be considered as a data transport into the tile-map solution. GeoJSON[12] supports a range of geometry types (points, lines, and polygons) giving users the opportunity to create pin-like interactions with point and line types and/or region-based interaction with the polygon.

The animated images visualisation type can be used in ‘weather map’ style visualisations, where animated visualisation data is overlayed onto a map. Care should be taken that the pixel size of animated data may not work well with different map zooms, the Maplibre example[9] is a good case in point. The image of 640x456 pixels is projected onto an area of about 600 km x 600 km, giving a resolution of one pixel per km, which results in a good presentation from distance, but detail is lost when the user zooms into the map.

The video visualisation type provides a similar solution to animated images, but the use of a video format rather than a collection of images allows for more frames of animation for a given data footprint, which may be useful for managing collections of files for fluid animations. Like the animated images, video visualisation is sensitive to map zooming, the Maplibre example[10] applying a lot of detail to a relatively small area (the inverse of the animated images example).

The WMS (Web map service) visualisation types provides an interface for visualising external web maps and requires an external web map server to host content.

For our use cases of visualising shapefiles and visualising EPANET simulations, it would appear that GeoJSON is likely to be the most suitable solution, given our need to work with large amounts of geographic data and our need for ‘close up’ detail.

3. Visualising shapefiles

For the aqua3s project, pilot sites were using shapefiles to store a wide range of visual and quantitative data relating to the operation of their water networks, Figure 2, typically regional flood maps, areas of commercial importance, sites of wells, regional boundaries and so on.

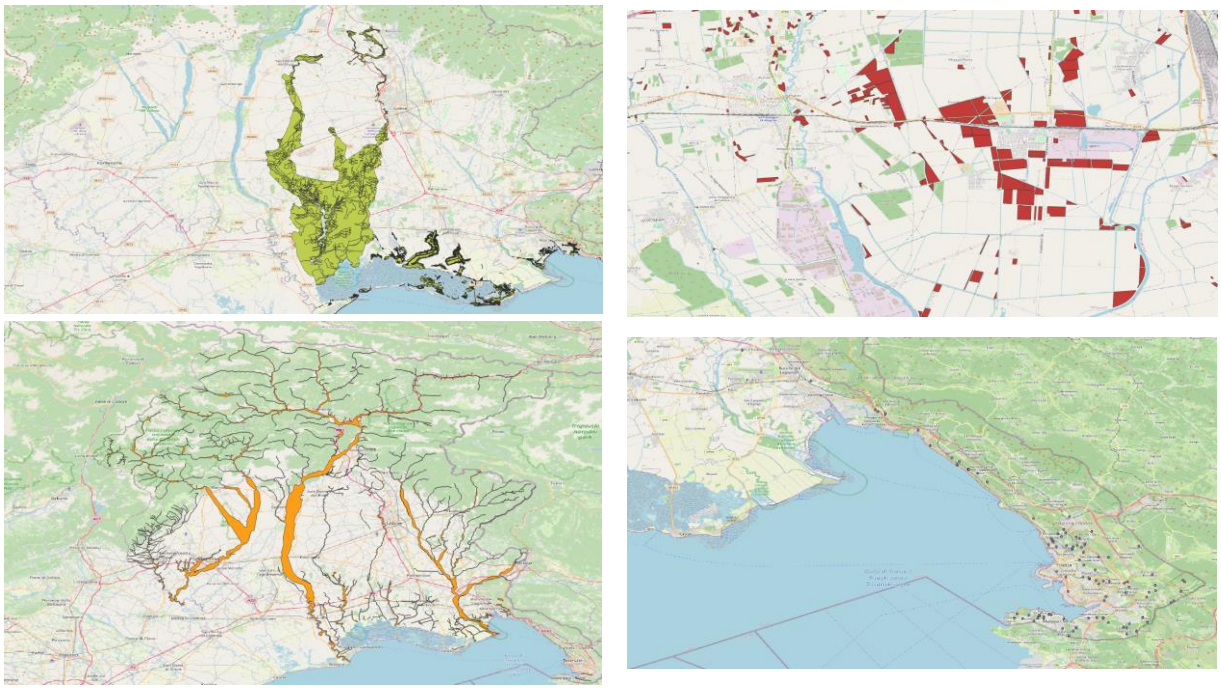


Figure 2: Typical shapefiles. Flood maps (top left), commercial locations (top right), river flows (bottom left) and wells (bottom right).

The shapefiles presented for visualisation tended to be large and complex files, typically with file sizes in the order of 100-200mb and geometry and metadata for up to 150,000 features and associated metadata per file. The shapefiles covered up to an area of 60km x 80km with a resolution of 3m x 3m. Typical operation of the application would see all the shapefiles downloaded to the client, with users having the ability to show and hide different shapefile layers, and to reveal metadata associated with shapefile features.

As stated previously, shapefiles are not a ‘concrete’ data type for MapLibre, so they would need to be converted into GeoJSON, a process which could be undertaken on the client or server, depending on processing requirements. It was expected that the larger shapefiles, c150,000 features, were likely to have a significant conversion load and were likely to create substantial GeoJSON files which may not work well with MapLibre, particularly giving the noticeable delays when working with the shapefiles in QGIS.

4. Approach

Initially, shapefiles were downloaded to the map-tile browser client and converted to GeoJSON in-situ, though this tended to be a time-consuming process and made for a poor user experience. It also proved to be beyond the visualisation capacity of Leaflet.js, resulting in a move to first Mapbox and then Maplibre given its open-source credentials.

An offline approach was taken to convert shapefile data into GeoJSON using geopandas[13] and to transform the projection of the resultant geometry into wgs84 format (the default for Mapbox). This was a very slow process and undertaken infrequently, with the resulting GeoJSON files stored on a file server. Once downloaded to the client, GeoJSON data was loaded into the framework using map.addSource[14] and map.addLayer[15], as per the Maplibre sample.

This resulted in a working 'proof of concept', i.e., it was possible to convert shapefiles into GeoJSON and have large GeoJSON files visualised in MapLibre with metadata available through mouse enter/exit events.

However, the large file size of the GeoJSON files (and number of files downloaded to the client) resulted in a long wait before all the GeoJSON layers were available to the user.

Visual inspection of the GeoJSON files revealed that around half the files was feature metadata with the remainder being feature geometry. The entire metadata was not required, as it was only referenced when the user's mouse entered a region and a pop-up dialog displayed it. Therefore, the metadata could be downloaded on demand.

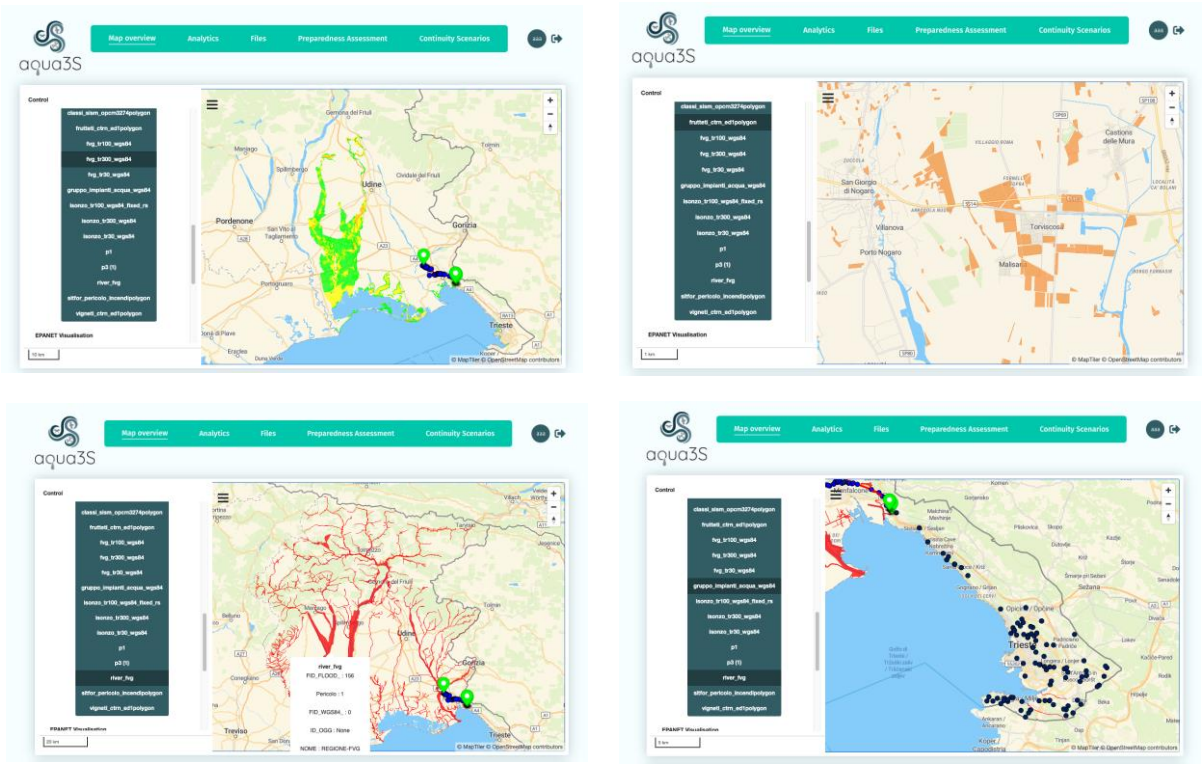


Figure 3: Shapefiles from Figure 2 visualised in aqua3s. Flood maps (top left), commercial locations (top right), river flows (bottom left) and wells (bottom right).

To further reduce the footprint of the GeoJSON data, the resolution of coordinate data was reduced. Geopandas appeared to default to 15dp for WGS84 coordinate data which accounts for most of the resulting file size. Through trial and error, it was found that resolution could be reduced to 6dp without 'noticeable' issues, though this is entirely dependent on the resolution of data that was being used, the 3m x 3m resolution of flood maps. It should also be noted that WGS84 measures in degrees, so the

length of a degree is not a consistent measure, therefore, our acceptable results and 34 north may not be valid for other latitudes.

Finally, the resulting files were zipped, given a footprint reduction from c200mb to around c10Mb. Whilst this data needs to be unpacked on the browser, it's a significantly less intensive operation than building the GeoJSON file on the browser.

5. Results

Figure 3 details the shapefiles from Figure 2 visualised as GeoJSON layers in the aqau3s client application. In general, MapLibre has proved to be a very capable tile-map platform, easily dealing with all the data that was downloaded onto it. One significant advantage in moving from Mapbox to Maplibre was the availability of source code which enabled us to solve a couple of troublesome issues with layer re-initialisation 'style.load' call-back[16].

6. Visualising EPANET simulations

For EPANET simulation, clients of aqua3s wanted to be able to visualise water networks modelled with EPANET and view the results of hydraulic and quality simulations in a manner like the existing EPANET application[17], Figure 4.

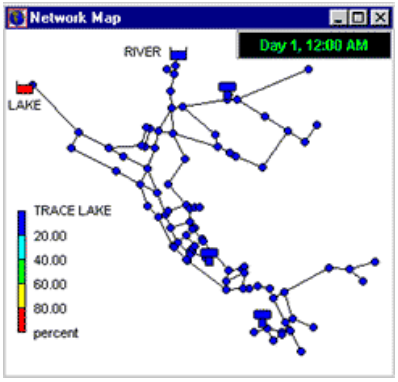


Figure 4: EPANET application water network visualisation.

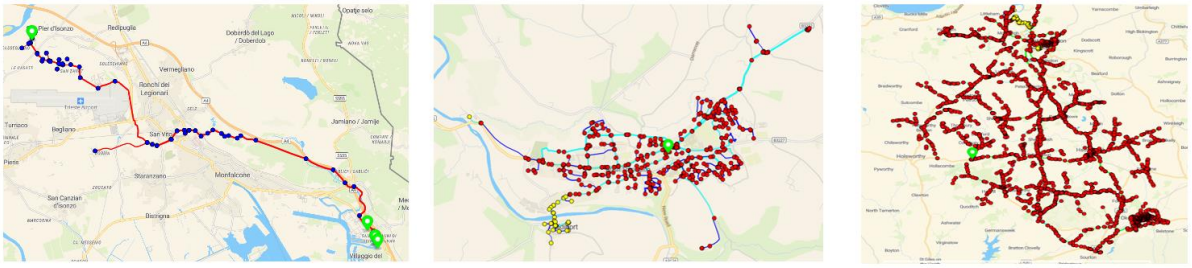
6.1. Visualising Topology

On first inspection, EPANET networks appeared to be similar to shapefiles, in that they contain rich sets of geometry data that could be described as features. However, EPANET simulations give the network a dynamic quality, in the sense that the values of the network will change over time. The EPANET application displays this through colours and provides users with numeric data on demand. So, whilst the topology of the network doesn't change over time, its representation does. Maplibre provides functionality to update layers in the setData() function, so an animated solution will need to build layer data (as GeoJSON), update layer data as the result of simulation data and finally set the layer data to reflect the changes.

Constructing a GeoJSON representation of an EPANET network was a straightforward task, notwithstanding the need to spatially convert elements into WGS84, as required for MapLibre visualisation. The biggest challenge was in determining a suitable structure for the collection of EPANET components which resulted in building a set of Maplibre layers to represent the pipes, pumps, valves, junctions, reservoirs, and tanks. Whilst the Maplibre 'add multiple geometries' sample[8] does show a single GeoJSON with multiple layers, we found that separating the data made it conceptually simpler to work with, particularly when dealing with simulation data and mouse enter/exit calls.

Having worked with large shapefiles of over 100,000 features, the fairly complex (by EPANET standards) water networks, Figure 5, appeared fairly small in comparison and given the relatively

small payloads (c1Mb) there seemed to be little need to compress geometry data in the way that we had for shapefiles.



Junctions	60	Junctions	1000	Junctions	12000
Pipes	72	Pipes	780	Pipes	9500
Valves	2	Valves	250	Valves	2800
Payload	5.5k	Payload	95k	Payload	c1Mb

Figure 5: EPANET water networks of differing complexity.

6.2. Simulation Data

Within the Centre for Water Systems, EPANET simulations are typically run over 7 days with a 15-minute resolution, creating around 700 data points per component per simulation. Simulations can be run in the EPANET app and saved as binary files[18] and/or run through the EPANET libraries, either to completion as per the app, or in a stepwise form. For our use cases, it was decided to run the simulation once and store the results for visualisation. Whilst the simulation was not massively computationally expensive, typically taking less than 2 minutes in the biggest case, there seemed to be little value in re-running the simulation if users were unable to change simulation parameters.

For our large case, the resulting simulation output file was 100Mb. Whilst this could be loaded into memory using Python and converted into Python data types, it was a slow purpose and echoed back to the shapefile metadata, in that not all users were going to require all of the data all the time, so an approach was developed where the binary file was loaded, and data was unpacked on demand using Python's struct.unpack_from[19].

6.3. Simulation Process

EPANET divides components into one of two types, nodes, or links, for simulation and organises its colourisation visualisation around four modes for nodes (supply, head, pressure, and quality) and eight modes for links (flow, velocity, headloss, quality, status, setting, reaction rate and friction). Therefore, the simulation visualisation only needs to deal with a small subset of simulation data at any one time. However, unlike shapefile visualisation where only user selected metadata is downloaded, simulation visualisation requires all the node and link data to be downloaded for the given simulation frame and relevant modes. However, displaying this information visually, rather than quantitatively, requires colour information to be sent to the client which occupies a smaller data footprint, even more so when the visualisation uses EPANET's five colour lookup, Figure 4, effectively just requiring a map of indices to colour values and a colour index for each component. On the client, the simulation data was parsed and used to update the colour value of each feature in each layer of the model. On completion, setData was called and the visualisation is updated, Figure 6.

7. Conclusions

In general, using MapLibre as a tile-map for the aqua3s project has worked exceptionally well. The library is generally easy to use and has a good level of visualisation performance which has provide suitable for the size and complexity of data we have been working with for both the EPANET and shapefile domains. The open-source nature of the library enabled us to look into the underlying code

when things were not working quite as expected and this did allow us to solve a particularly annoying issue with refreshing visualisations after style.load.

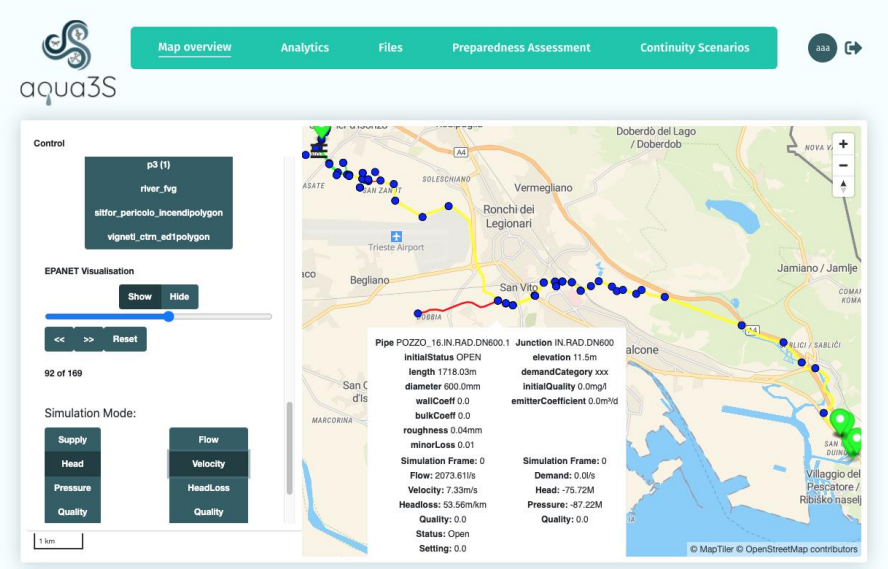


Figure 6: EPANET simulation visualization in aqua3s, showing colourised water network and simulation details.

To a large degree, choosing a suitable technology stack for the project was a bit of a beauty pageant, in that it is a time and resource-limited activity, so there is a tendency to go with whatever works, rather than the absolute best possible solution. There is also the scope to re-engineer existing technologies to meet performance needs, particularly in the case of dismantling GeoJSON data to make it 'fit' our needs and although GeoJSON felt like a 'fat' format, it compresses well and keeps readability, which moving to a binary format would not allow so easily.

Currently, the biggest issue with the overall project is the time taken for the Python-based conversion of Shapefiles to GeoJSON, though this may be addressed by moving a different conversion library.

Acknowledgements
The work presented in this paper was funded by the ongoing EC H2020 aqua3S (GA 832876), Fiware4Water (GA 821036) and LOTUS (GA 820881) projects.

References

[1] QGIS Home Page https://qgis.org/en/site/ (accessed Jul. 12, 2021)
[2] Aqua3s project homepage https://aqua3s.eu/ (accessed Jul. 12, 2021)
[3] Google maps homepage https://cloud.google.com/maps-platform/pricing/sheet (accessed Jul. 12, 2021)
[4] Mapbox Homepage https://www.mapbox.com/ (accessed Jul. 12, 2021)
[5] Leaflet.js Homepage https://leafletjs.com/ (accessed Jul. 12, 2021)
[6] MapLibre Homepage https://maplibre.org/ (accessed Apr. 01, 2022)
[7] Add a default marker https://maplibre.org/maplibre-gl-js-docs/example/add-a-marker/ (accessed Apr. 01, 2022)
[8] Add multiple geometries from one GeoJSON source https://maplibre.org/maplibre-gl-js-docs/example/multiple-geometries/ (accessed Apr. 01, 2022)
[9] Animate a series of images https://maplibre.org/maplibre-gl-js-docs/example/animate-images/ (accessed Apr. 01, 2022)

[10] Add a video https://maplibre.org/maplibre-gl-js-docs/example/video-on-a-map/ (accessed Apr. 01, 2022)
[11] Add a WMS Source https://maplibre.org/maplibre-gl-js-docs/example/wms/ (accessed Apr. 01, 2022)
[12] GeoJSON Homepage https://geojson.org/ (accessed Apr. 01, 2022)
[13] Geopandas Homepage https://geopandas.org/en/stable/ (accessed Apr. 01, 2022)
[14] MapLibre.AddSource https://maplibre.org/maplibre-gl-js-docs/api/map/#map#addsource (accessed Apr. 01, 2022)
[15] Maplibre Map.AddLayer https://maplibre.org/maplibre-gl-js-docs/api/map/#map#addlayer (accessed Apr. 01, 2022)
[16] MapLibre.map.on https://maplibre.org/maplibre-gl-js-docs/api/map/#map#on (accessed Apr. 01, 2022)
[17] EPANET Application https://www.epa.gov/water-research/epanet (accessed Apr. 01, 2022)
[18] OWA EPANET outfile format http://wateranalytics.org/EPANET/_out_file.html (accessed Apr. 01, 2022)
[19] Python struct https://docs.python.org/3/library/struct.html (accessed Apr. 01, 2022).