Modernizing R's web-mapping capabilities

Tim Appelhans

2025-10-01

Executive Summary

When it comes to web-mapping in R, RStudio's (Posit's) `leaflet` package has established itself as the de-facto standard. While the underlying JavaScript web mapping library (also named `leaflet`) is versatile and robust, it lacks certain key aspects of newer web-mapping libraries, such as MapLibre GL JS (see mapgl for a recent R wrapper). This especially includes the lack of native 3D WebGL rendering. As such, `leaflet` struggles with the rendering of large data (i.e. more than 1 million points/vertices). Another part is the slow data transfer from R memory to the browser. Several (de-)serialization steps are needed to transfer R's in-memory data representation to the browser, usually realized using GeoJSON data format. New tools, such as GeoArrow speed up this process tremendously.

Furthermore, the geo-spatial community has adopted cloud native geospatial data formats for storage, as well as for analysing and visualising very large data collections. Enabling direct rendering of such data in the browser will help researchers find and filter relevant data more quickly.

Having support for cloud-native formats and a way to visualise geo-spatial quickly will strongly enhance the user experience across the whole R-spatial community. Packages such as `tmap`, `mapdeck` and also R package `leaflet` will benefit from the implementations described in this proposal.

This application aims at solving current pain points for geo-spatial R users in 3 steps:

1. **Utilising (geo)arrow for fast geo-spatial data transfer**

   Two new R packages will be developed to

   a. enable fast transfer of geo-spatial data from R memory to the browser, using `geoarrow` and
   b. allow fast web map overlays of these data by eliminating the need to (de-)serialise them multiple times. Development of this has already started in `geoarrowDeckgl`.

2. **Direct rendering support for cloud-native data formats**

   A minimum set cloud native data formats that will be supported include GeoParquet, flatgeobuf and Cloud Optimised Geotiff, but can most likely be extended to Cloud Optimised Point Clouds and others. Note that `PMTiles` is already natively supported by R package `mapgl`.

3. **Bindings to and instructions for existing frameworks**

   Even though development of 1. and 2. will concentrate on package `mapgl`, and hence, `maplibre` as the rendering platform, it will not be limited to it. We aim at providing dedicated bindings for other packages including `leaflet`, `tmap`, `mapdeck` and `mapview`.

With these developments, web-mapping capabilities in R will become more user-friendly, as they will enable scalability of geo-spatial data visualisation beyond current limits. Utilising `geoarrow` for transfer of in-memory data will reduce rendering times significantly and support for cloud native data formats will enable visualisation of larger-than-memory data.

To achieve this, the applicant requests funds of **12000€** (see Section Budget & funding plan for details).

Signatories

Supporting authors:

- Kyle Walker - mapgl
- Dewey Dunnington - geoarrow | nanoarrow
- Kira McDonald - mapglview
- Edzer Pebesma - r-spatial
- Martijn Tennekes - r-tmap

Project team

Tim Appelhans

Contributors

- Jakub Nowosad - geocompx
- Edzer Pebesma - r-spatial

Consulted

Josiah Parry

The Problem

Compared to a decade ago, creation of geo-spatial data sets from various domains has led to increasingly large geo-spatial data collections. New, modern, mostly web-based tools have emerged in the geo-spatial data science realm in recent years that enable handling larger data volumes in storage, analysis and visualisation. Removing friction and performance bottlenecks have provided end-users with smooth and performant workflows. From an R perspective, these tools can roughly be broken down into three categories:

1. **Data storage**

   Traditionally, data to be analysed used to reside on the users machine. This has changed significantly in recent years, mainly as a consequence of the sheer amount of data being available.

   The geo-spatial community is certainly not the only community to see this shift to cloud-native data, and just like others, it has developed tailored formats for geo-spatial data that are optimised for the domain. It is clear that Cloud native geospatial data formats have become more and more relevant and are being widely used in the field. While R already has support for some of these formats, e.g. flatgeobuf and Cloud Optimised Geotiff are supported in package `leafem`, other formats, such as GeoParquet or Cloud Optimised Point Clouds are still unsupported, at least for direct visualisation without the need to read into R memory.

2. **Data transfer**

   Transferring in-memory data from a local R process to the web browser for visualisation is generally done through several serialization and de-serialization steps via `GeoJSON` data representation. While this is robust and well established, it can be very slow, especially for large and complicated geospatial data. While it's hard to quantify solidly, currently available solutions will certainly struggle with data beyond 1 million points (vertices in case of line or polygon data), especially when additional attribute data should be visualised.

   New technologies, such as GeoArrow, have emerged and provide unprecendented perfomance/speed enhancements, as data can be transferred directly to the web browser without the need for copies and little to no (de-)serialization. We have already started to implement GeoArrow support in geoArrowDeckgl, an add-on package for `mapgl` based on the works of Kyle Barron's deck.gl-layers, a fast, memory efficient JavaScript library for deck.gl.

3. **WebGL based mapping**

   As mentioned earlier, `leaflet` is a well established and widely used web-mapping library, yet, it has two distinct drawbacks:

   - it is not using WebGL natively, and
   - it is 2D only.

   While extensions exist to enable WebGL rendering (e.g. Leaflet.glify with an R wrapper package leafgl), there is no way of extending leaflet to support 3D rendering. Yet, many geo-spatial sub-domains have a need to visualise 3D data, such as flight tracks, animal movement data, building heights, terrain and hydrological data sets, to name a few.

   Furthermore, some `leaflet` extensions relying on e.g. `leafem` are currently not well maintained or developed. It remains to be seen if and how some of these will be updated to support migration to the pending major, and potentially breaking release of leafletjs 2.0.

   Reliance on `leaflet` dependencies that are not well maintained poses a significant risk for the future of R's web-mapping capabilities.

   MapLibre GL JS is a younger, more performant and actively maintained web-mapping JavaScript library and Kyle Walker has already ported large parts of it to R with his `mapgl` package. MapLibre supports a few notable things out-of-the-box:

   - WebGL rendering,
   - 3D rendering,
   - PMTiles data (for larger-than-RAM data), and
   - Vector tiles for background map layers (`leaflet` uses raster tiles by default) - meaning that styling is much more flexible and performant as it can be done client-side on-the-fly.

   An overview of selected Maplibre capabilities as compared to leaflet can be found in this blog post.

All in all, to adjust to the changing landscape of geo-spatial data types and availability, efforts are needed to support visualisation of growing data amounts and to ensure that R keeps up with continuing developments in the field outside of R.

The proposal

Overview

To overcome the outlined problems with regards to slow data transfer between R and the browser and lack of cloud-native data support, two main implementations will be realised

with this proposal. First, a small package (working title `geoarrowWidget`) will be developed to provide general functionality for quickly transferring geo-spatial data from R memory to the browser via the geoarrow data specification. Second, `geoarrowDeckgl` will be finalised and extended with dedicated functions to both enable overlays for `maplibre` maps, as well as support visualisation of cloud-native data formats directly, i.e. without the need to read data into R memory. These developments will focus on extending web-mapping workflows provided by package `mapgl`, however, in a third step `geoarrowDeckgl` will be expanded with dedicated bindings for other web-mapping R packages, such as `leaflet`, `tmap`, `mapdeck` and `mapview` so that a wide range of R spatial users will benefit from these efforts.

Detail

In more detail, this proposal can be divided into three major milestones:

1. **Fast & efficient geo-spatial data transfer and web-mapping overlays**

Data transfer functionality between R memory and the browser will be implemented in a small dedicated add-on package to htmlwidgets - `geoarrowWidget`. It will provide functions to convert geo-spatial in-memory data formats, such as `sf`, `terra` or `wk`, to `geoarrow` and to attach relevant dependencies (the geoarrow data and JavaScript libs to read it) to existing widgets. The intention is that other, including non-mapping applications can benefit from this as well. This functionality already largely exists, but is currently part of `geoarrowDeckgl` and will be moved to this new package.

`geoarrowDeckgl` will focus on providing highly performant overlays for `maplibre` maps using `deck.gl` layers utilising geoarrow/deck.gl-layers which is also used by the python library lonboard. This library is optimised for fast data processing in multiple ways, for example by copying "*binary buffers directly from an Arrow JS Table object to the GPU*", among other optimisations (see here for a full list of features). `Deck.gl` is a GPU-powered tool for rendering large data amounts in the browser. It has dedicated support for several web-mapping libraries, including e.g. `maplibre` and `leaflet`, to overlay its layers on top base maps. The API of `geoarrowDeckgl` is tightly coupled to that of `Deck.gl`, in order to fully utilise and closely reflect the underlying functionality. This also enables easy adaptability to potential future API changes or extensions of this upstream dependency.

To achieve the above, we rely on packages `geoarrow` and `nanoarrow` for data conversion on the R side and several JavaScript libraries, such as `arrowjs` for client-side (browser) data handling, which will be shipped with the above mentioned newly developed packages (see list of dependencies below).

2. **Direct rendering support for cloud-native geo-spatial data formats**

Most data in cloud-native geo-spatial formats can nowadays be read into R through packages such as `sf` and `terra`, among others. However, for visualising data that resides in the cloud it is preferable to not first materialise these data in R memory before sending it off to the rendering canvas in the browser, especially if only a subset of the data is of interest. Cloud-native data formats support partial reads so that data transfer over the wire is minimised, which makes them ideal for web-mapping applications. Reading the complete file and then rendering only parts of it would ignore the benefits of these data formats.

To overcome this, `geoarrowDeckgl` will provide functionality to allow users to visualise cloud data directly in the browser. The aim is to provide an API similar to that developed in 1., so that e.g. styling instructions can be defined in R, transferred to the browser and then be applied to the incoming data when it arrives. This means that the user is not expected to know any JavaScript. Functionality for cloud optimized geotiffs can be implemented with the `Maplibre GL JS` add-on maplibre-cog-protocol, while both `geoparquet` and `flatgeobuf` are supported by geoarrow/deck.gl-layers.

3. **Bindings for existing mapping packages**

The third aim of this proposal is to scale the developments of 1. and 2. to the wider R spatial community.

Separating development of functionality for data transfer and web map overlays ensures that other, potentially non-mapping applications can benefit from fast data transfer without overhead. Detailed documentation and dedicated vignettes outlining how to use the transfer functionality will enable other developers to adopt it for their own needs. Note that this new package for data transfer will mainly be of interest to developers, not for end users; end users will benefit when it is adopted in downstream packages.

The development of `geoarrowDeckgl` will benefit both R developers and end users, as these developments will be made available for other web-mapping packages beyond `mapgl`, i.e. `leaflet` and `mapdeck` at the very least. `Deck.gl` has a dedicated add-on package for `leaflet` support (deck.gl-community/leaflet), however, as their web page states, this is in minimal maintainence mode, so it remains to be seen how long `leaflet` support can be realised. `mapdeck` is a wrapper around `Deck.gl` and as such it will be easy to provide bindings for it. `tmap` now has a dedicated extension package for using `mapgl` as the base map rendering paltform called `tmap.mapgl` which will also benefit from the developments made in `geoarrowDeckgl`. `mapview` is a higher-level mapping package that supports multiple rendering platforms and, as such, is easily extendable to `mapgl` and can benefit from performant rendering support developed through the efforts of this proposal. In fact, a first bare-bones implementation of `mapgl` as a rendering platform is already implemented in an experimental branch of `mapview`.

Minimum Viable Product

A minimum viable product of the above outlined proposal will include (in order of development):

- a new CRAN package - `geoarrowWidget` - for fast data transfer as an add-on to `htmlwidgets`
- a CRAN release of `geoarrowDeckgl` including:
  - support for the major geo-spatial vector layer types for points, lines and polygons
  - support for direct rendering of `cloud native geotiff`, `geoparquet` and `flatgeobuf` data in the browser
  - methods to support `leaflet` and `mapdeck` based workflows
- detailed documentation, vignettes and blog-posts highlighting these new developments

Blog-posts will most likely be published on r-spatial.org, but other platforms such as the RConsortium blog can also be used, when desired by the ISC.

Architecture

The main delivery of this proposal will be `geoarrowDeckgl`, a new R package for fast and efficient overlays of geo-spatial data on web maps. It will provide dedicated methods for the web-mapping packages `mapgl`, `leaflet` and `mapdeck`. Another small package - `geoarrowWidget` - will be developed for fast data transfer from R memory to the browser, which will be in imported by `geoarrowDeckgl`.

Assumptions

There are no critical assumptions being made for this project.

External dependencies

R dependencies:

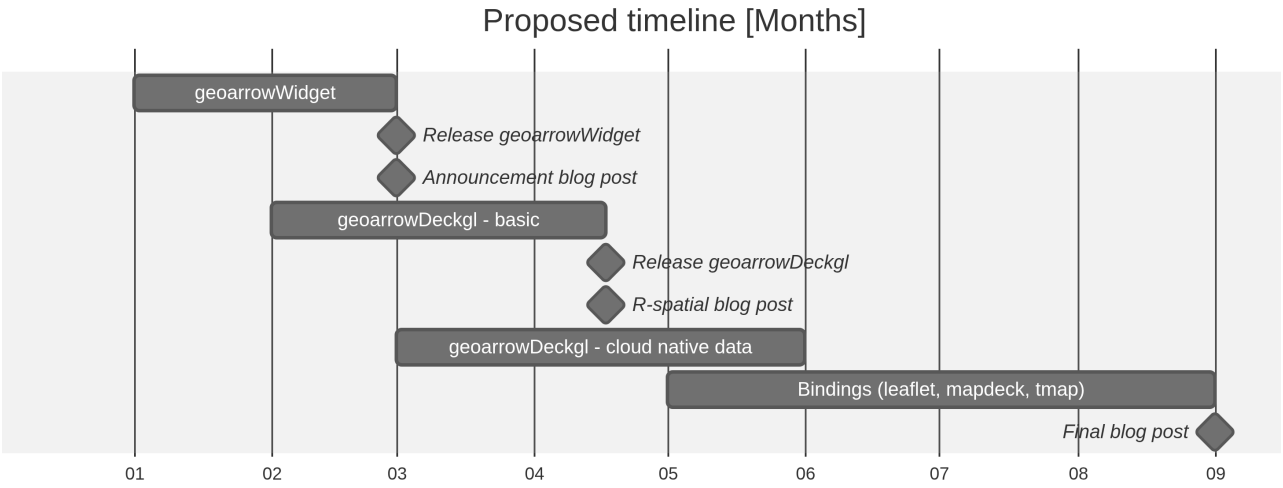
- geoarrow
- nanoarrow

JavaScript dependencies:

- Deck.gl
- geoarrow/deck.gl-layers
- arrowjs

- geoarrowjs
- deck.gl-community/leaflet

Project plan



Proposed timeline; Milestones shown as diamonds.

Start-up phase

A start-up phase is not needed. Work can start right away by separating out the data transfer functionality from `geoarrowDeckgl` to `geoarrowWidget`, so that initial releases of these two packages may even happen within the first 1-2 months of the project.

Technical delivery

All packages will be developed under the umbrella of the r-spatial github organisation and once mature, will be released on CRAN.

Other aspects

A first announcement blog post will be pusblished on the `r-spatial` blog once `geoarrowWidget` is released. A second post is planned for the CRAN release of `geoarrowDeckgl` and a final post will highlight the implementations in other mapping frameworks, such as `leaflet`, `tmap` and

`mapdeck`. Major and minor updates will also be announced via the applicants mastodon profile. Furthermore, a talk at UseR! 2026 in Warsaw is planned. The applicant is also more than happy to give a dedicated RConsortium webinar to highlight the developments and outcomes of this proposal.

Budget & funding plan

As can be seen from the timeline above, there are 3 milestones, whereas the first of those can be achieved rather quickly and can be considered foundational for the others to be delivered in a structured and scalable manner. The remaining two major milestones of this application are marked by the timing of the blog post releases, after about 4.5 months and at the end of the 9 month funding period.

The applicant is currently employed on 75% contract at the Friedrich Alexander University of Erlangen-Nürnberg. The intention of this application is to use the granted funds to extend this contract to 100%, hence a total of 40 hours a month, which equates to 360 hours over a period of 9 months for completing the work outlined in this proposal.

At the applicants current payment rate, 25% of his salary for 9 months amounts to about **12000€**, which is the amount requested by this application. This in turn translates to roughly 34€ per hour worked, before taxes.

Success

Definition of done

This application is considered complete when

- `geoarrowWidget`, `geoarrowDeckgl` are released on CRAN, and
- support for at least `leaflet` and `mapdeck` is implemented.

Full support for `tmap` via `tmap.mapgl` is beyond the scope of this application, and would require support from the `tmap` author. However, the plan is to utilise the implementations of this proposal via dedicated pull requests.

Future work

Future work that is beyond the scope of this application may include

- implementation of further `Deck.gl` layer types (as supported by `geoarrow/deck.gl-layers`)
- transfer of the implementation to other web-mapping R packages, e.g. deckgl