Simple Features for R: Standardized Support for Spatial Vector Data

by Edzer Pebesma

Abstract Simple features are a standardized way of encoding spatial vector data (points, lines, polygons) in computers. The **sf** package implements simple features in R, and has roughly the same capacity for spatial vector data as packages **sp**, **rgeos**, and **rgdal**. We describe the need for this package, its place in the R package ecosystem, and its potential to connect R to other computer systems. We illustrate this with examples of its use.

What are simple features?

Features can be thought of as "things" or objects that have a spatial location or extent; they may be physical objects like a building, or social conventions like a political state. *Feature geometry* refers to the spatial properties (location or extent) of a feature, and can be described by a point, a point set, a linestring, a set of linestrings, a polygon, a set of polygons, or a combination of these. The *simple* adjective of *simple features* refers to the property that linestrings and polygons are built from points connected by *straight* line segments. Features typically also have other properties (temporal properties, color, name, measured quantity), which are called *feature attributes*. Not all spatial phenomena are easy to represent by "things or objects:" continuous phenoma such as water temperature or elevation are better represented as *functions* mapping from continuous or sampled space (and time) to values (Scheider et al., 2016), and are often represented by *raster* data rather than vector (points, lines, polygons) data.

Simple feature access (Herring, 2011) is an international standard for representing and encoding spatial data, dominantly represented by point, line, and polygon geometries (ISO, 2004). It is widely used e.g. by spatial databases (Herring, 2010), GeoJSON (Butler et al., 2016), GeoSPARQL (Perry and Herring, 2012), and open source libraries that empower the open source geospatial software landscape including GDAL (Warmerdam, 2008), GEOS (GEOS Development Team, 2017), and liblwgeom (a PostGIS component, Obe and Hsu (2015)).

The need for a new package

The **sf** (Pebesma, 2018) package is an R package for reading, writing, handling, and manipulating simple features in R, reimplementing the vector (points, lines, polygons) data handling functionality of packages **sp** (Pebesma and Bivand, 2005; Bivand et al., 2013), **rgdal** (Bivand et al., 2017) and **rgeos** (Bivand and Rundel, 2017). However, **sp** has some 400 direct reverse dependencies, and a few thousand indirect ones. Why was there a need to write a package with the potential to replace it?

First of all, at the time of writing **sp** (2003) there was no standard for simple features, and the ESRI shapefile was by far the dominant file format for exchanging vector data. The lack of a clear (open) standard for shapefiles, the omnipresence of "bad" or malformed shapefiles, and the many limitations of the ways it can represent spatial data adversely affected **sp**, for instance in the way it represents holes in polygons, and a lack of discipline to register holes with their enclosing outer ring. Such ambiguities could influence plotting of data, or communication with other systems or libraries.

The *simple feature access* standard is now widely adopted, but the **sp** package family has to make assumptions and do conversions to load them into R. This means that you cannot round-trip data, e.g., loading data in R, manipulating them, exporting them and getting the same geometries back. With **sf**, this is no longer a problem.

A second reason was that external libraries heavily used by R packages for reading and writing spatial data (GDAL) and for geometrical operations (GEOS) have developed stronger support for the simple feature standard.

A third reason was that the package cluster now known as the **tidyverse** (Wickham, 2017, 2014), which includes popular packages such as **dplyr** (Wickham et al., 2017) and **ggplot2** (Wickham, 2016), does not work well with the spatial classes of **sp**:

- **tidyverse** packages assume objects not only *behave* like `data.frames` (which **sp** objects do by providing methods), but *are* `data.frames` in the sense of being a list with equally sized column vectors, which **sp** does not do.

- attempts to "tidy" polygon objects for plotting with **ggplot2** ("fortify") by creating `data.frame` objects with records for each polygon node (vertex) were neither robust nor efficient.

A simple (S3) way to store geometries in `data.frame` or similar objects is to put them in a geometry list-column, where each list element contains the geometry object of the corresponding record, or `data.frame` "row"; this works well with the **tidyverse** package family.

Conventions

Classes

The main classes introduced by package **sf** are

`"sf"`: a `data.frame` (or `tbl_df`) with one or more geometry list-columns, and an attribute `sf_column` indicating the *active* geometry list-column of class `sfc`,

`"sfc"`: a list-column with a set of feature geometries

`"sfg"`: element in a geometry list-column, a feature geometry

`"crs"`: a coordinate reference system, stored as attribute of an `"sfc"`

Except for `"sfg"`, all these classes are implemented as lists. Objects of class `"sfg"` are subtyped according to their class, classes have the following storage form:

`POINT`: numeric vector with a single point

`MULTIPOINT`: numeric matrix with zero or more points in rows

`LINESTRING`: numeric matrix with zero or more points in rows

`POLYGON`: list with zero or more numeric matrices (points as rows); polygon outer ring is followed by zero or more inner rings (holes)

`MULTILINESTRING`: list with zero or more numeric matrices, points in rows

`MULTIPOLYGON`: list of lists following the `POLYGON` structures

`GEOMETRYCOLLECTION`: list of zero or more of the (classed) structures above

All geometries have an empty form, indicating the missing (or `NA`) equivalent for a geometry.

Functions and methods

Category	Functions
binary predicates	`st_contains`, `st_contains_properly`, `st_covered_by`, `st_covers`, `st_crosses`, `st_disjoint`, `st_equals`, `st_equals_exact`, `st_intersects`, `st_is_within_distance`, `st_within`, `st_touches`, `st_overlaps`
binary operations	`st_relate`, `st_distance`
unary operations	`st_dimension`, `st_area`, `st_length`, `st_is_longlat`, `st_is_simple`, `st_is_valid`, `st_jitter`, `st_geohash`, `st_geometry_type`
miscellaneous	`st_sample`, `st_line_sample`, `st_join`, `st_interpolate_aw`, `st_make_grid`, `st_graticule`, `sf_extSoftVersion`, `rawToHex`, `st_proj_info`
setters	`st_set_agr`, `st_set_crs`
constructors	`st_sfc`, `st_sf`, `st_as_sf`, `st_as_sfc`, `st_point`, `st_multipoint`, `st_linestring`, `st_multilinestring`, `st_polygon`, `st_multipolygon`, `st_geometrycollection`, `st_combine`, `st_bind_cols`
in- & output	`st_read`, `st_read_db`, `st_write`, `st_write_db`, `read_sf`, `write_sf`, `st_drivers`, `st_layers`
plotting	`st_viewport`, `st_wrap_dateline`, `sf.colors`

Table 1: Functions provided by package **sf**, arranged by functional category.

Functions are listed in Table 1. Some functions or methods operate on both attributes and geometries, e.g. `aggregate` and `summarise` compute grouped statistics and group (union) corresponding

class	methods
sfg	as.matrix, c, coerce, format, head, Ops, plot, print, st_as_binary, st_as_grob, st_as_text, st_transform, st_coordinates, st_geometry, st_boundary, st_buffer, st_centroid, st_convex_hull, st_difference, st_intersection, st_line_merge, st_make_valid, st_node, st_point_on_surface, st_polygonize, st_segmentize, st_simplify, st_split, st_sym_difference, st_triangulate, st_union, st_voronoi, st_cast, st_collection_extract, st_is, st_zm
sfc	[, [<-, as.data.frame, c, coerce, format, Ops, print, rep, st_as_binary, st_as_text, st_bbox, st_coordinates, st_crs, st_crs<-, st_geometry, st_precision, st_set_precision, str, summary, st_transform, st_boundary, st_buffer, st_centroid, st_convex_hull, st_difference, st_intersection, st_line_merge, st_make_valid, st_node, st_point_on_surface, st_polygonize, st_segmentize, st_simplify, st_split, st_sym_difference, st_triangulate, st_union, st_voronoi, st_cast, st_collection_extract, st_is, st_zm, obj_sum, type_sum
sf	[, [[<-, $<-, aggregate, cbind, coerce, merge, plot, print, rbind, st_agr, st_agr<-, st_bbox, st_coordinates, st_crs, st_crs<-, st_geometry, st_geometry<-, st_precision, st_set_precision, st_transform, st_boundary, st_buffer, st_centroid, st_convex_hull, st_difference, st_intersection, st_line_merge, st_make_valid, st_node, st_point_on_surface, st_polygonize, st_segmentize, st_simplify, st_split, st_sym_difference, st_triangulate, st_union, st_voronoi, st_cast, st_collection_extract, st_is, st_zm, anti_join, arrange, distinct, filter, full_join, gather, group_by, inner_join, left_join, nest, mutate, rename, right_join, sample_frac, sample_n, select, semi_join, separate, slice, spread, summarise, transmute, ungroup, unite
crs	$, is.na, Ops, print, st_as_text, st_crs

Table 2: Methods for **sf** classes: colours indicate geometry operation, class manipulation, tidyverse; and Ops refers to (a subset of) arithmetical operations.

geometries, and `st_interpolate_aw` carries out area-weighted interpolation (Do et al., 2015). The function `st_join` joins pairs of tables based on a geometrical predicate such as `st_intersects`.

Generic methods for `sf` objects are listed in Table 2. Many of them are for creation, extraction, and conversion, and many of them are not needed for every-day work. Where possible, methods act either on a geometry (`sfg`), a geometry set (`sfc`), or a geometry set with attributes (`sf`), Methods return an object of identical class. Coordinate reference systems (CRS) carry through all operations, except for `st_transform`, which transforms coordinates from one reference system into another, and hence, the CRS changes.

Serialisations

The simple feature access defines two serialisation standards: well-known-text (WKT) and well-known-binary (WKB). Well-known text is the default print form and `sfc` columns can be read from WKT `character` vectors, using `st_as_sfc`:

```
> library(sf)
Linking to GEOS 3.5.1, GDAL 2.1.2, proj.4 4.9.3
> (pt <- st_point(c(0,1)))
POINT (0 1)
> (pol <- st_polygon(list(rbind(c(0,0), c(1,0), c(1,1), c(0,1), c(0,0)))))
POLYGON ((0 0, 1 0, 1 1, 0 1, 0 0))
> st_as_sfc("POINT(0 1)") # returns sfc:
Geometry set for 1 feature
geometry type:  POINT
dimension:      XY
bbox:           xmin: 0 ymin: 1 xmax: 0 ymax: 1
epsg (SRID):    NA
proj4string:    NA
POINT (0 1)
```

R native simple feature geometries can be written to WKB using `st_as_binary`:

```
> st_as_binary(st_point(c(0,1)))
 [1] 01 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 f0 3f
```

```
> st_as_binary(st_polygon(list(rbind(c(0,0), c(1,0), c(1,1), c(0,1), c(0,0)))))
 [1] 01 03 00 00 00 01 00 00 00 05 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
[26] 00 00 00 00 00 00 00 00 00 f0 3f 00 00 00 00 00 00 00 00 00 00 00 00 00 00
[51] 00 f0 3f 00 00 00 00 00 00 f0 3f 00 00 00 00 00 00 00 00 00 00 00 00 00 00
[76] f0 3f 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Similarly, binary encoded geometries can be read back using `st_as_sfc`.

All communication to and from the underlying libraries GDAL, GEOS and liblwgeom, as well as direct reading and writing of geometry BLOBs in spatial databases, uses binary serialisation and deserialisation, written in C++. This makes code not only fast but also robust: for all possible geometry classes, a single interface is used to communicate to a variety of endpoints.

Spherical geometry

The GEOS library provides a large set of operations for data in a two-dimensional space. For unprojected, geographic data the coordinates are longitude and latitude, and describe points on a sphere (or ellipsoid), not on a plane. The **sf** package allows such data to be passed to all geometric operations, but will emit a message if this happens through GEOS, assuming a flat Earth. For the functions `st_area`, `st_length`, `st_distance`, `st_is_within_distance`, and `st_segmentize` specialized spherical functions, taken from **lwgeom** (Pebesma), are used. The advantage of this package e.g. over **geosphere** (Hijmans, 2016a) is that it supports simple features for distance calculations, where geosphere only computes distances between points. Function st_sample has been modified to work for spherical coordinates when sampling points on an area over a sphere.

It would be nice to get a (more) complete set of functions working for spherical geometry. Potential candidate libraries to be used for this include **s2** (Rubak and Ooms, 2017), liblwgeom (part of PostGIS), CGAL (Fabri and Pion, 2009), and boost.Geometry.

Tidy tools

During the development of **sf**, considerable effort was put into making the new data structures work with the tidyverse. This was done by providing methods for **dplyr** verbs (Table 2), and by helping develop a **ggplot2** `geom` function (next section) that plots maps well.

The tidy tools manifesto prescribes four principles, which we will comment on:

1. **Reuse existing data structures.** We use the simplest R structures (numeric vector for point, matrix for point set, list for any other set), and fully support two standardized serializations (WKT, WKB)
2. **Compose simple functions with the pipe.** functions and methods were designed such that they can be used easily in pipe-based workflows; replacement functions like `st_crs<-` were augmented by `st_set_crs` to make this look better.
3. **Embrace functional programming.** Functions were kept type-safe, empty geometries and empty lists are supported, and operation overloading was done creatively e.g. by providing `Ops` for scaling and shifting a polygon:

   ```
   > pol * 2 + pt
   POLYGON ((0 1, 2 1, 2 3, 0 3, 0 1))
   ```

   Functions like `st_join` for a spatial join allow the user to pass a join function that is *compatible* with `st_intersects`, making the spatial predicate applied for the join completely customisable.
4. **Design for humans.** with the experience of having (co-)written and maintained **sp** for a decade, we have tried to keep **sf** simple and lean. Methods were used as much as possible to keep the namespace small. All functions and methods start with `st_` (for "spacetime", following PostGIS convention) to keep them recognizable, and searchable using tab-completion.

Plotting

Figure 1 (left) shows the default plot for an `"sf"` object with more than one attribute: no color keys are given, default colours depend on whether the variable is numeric (top) or a factor (bottom). Figure 1 was obtained by:

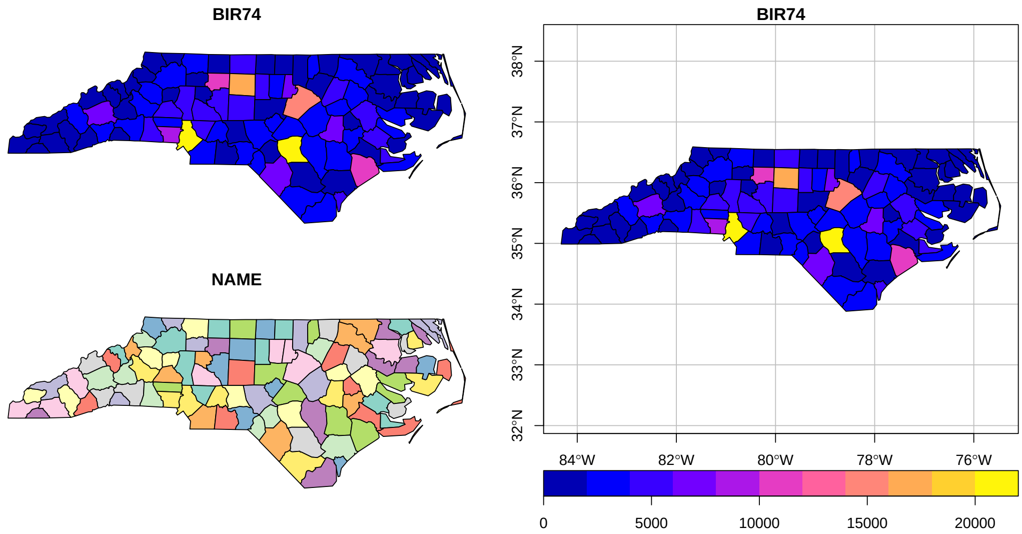

Figure 1: At left: default plot for `sf` object with two attributes; on right: plot for a single attribute with color key, axes and graticule.

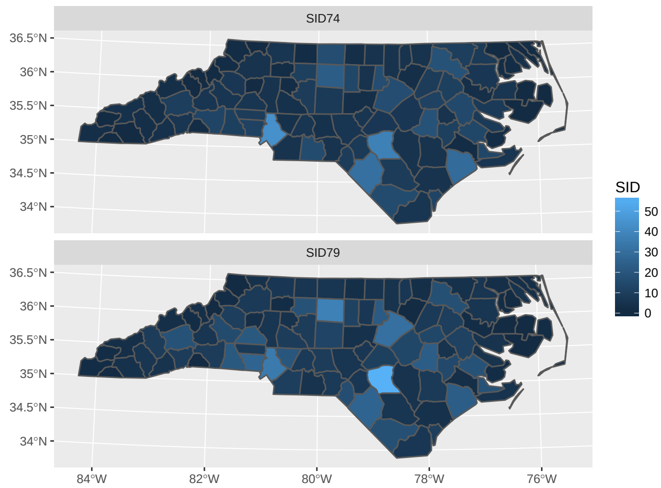

Figure 2: Plot generated with `ggplot2::geom_sf`, the now curved graticules follow constant long/lat lines.

```
> library(sf)
> nc = read_sf(system.file("gpkg/nc.gpkg", package="sf"))
> plot(nc[, c(9,5)])
```

When we plot a single attribute, a color key is default (unless `key.pos=NULL`). The following command

```
> plot(nc[, 9], key.pos = 1, axes = TRUE, graticule = TRUE)
```

adds axes and a graticule (longitude/latitude grid lines) on the right side of Figure 1.

Figure 2 shows a plot generated by `ggplot2` (version 2.2.1 or later):

```
> library(ggplot2)
> library(tidyr)
> library(dplyr)
> nc2 <- nc %>% st_transform(32119) %>% select(SID74, SID79, geom) %>%
+   gather(VAR, SID, -geom)
> ggplot() + geom_sf(data = nc2, aes(fill = SID)) + facet_wrap( ~ VAR, ncol = 1)
```

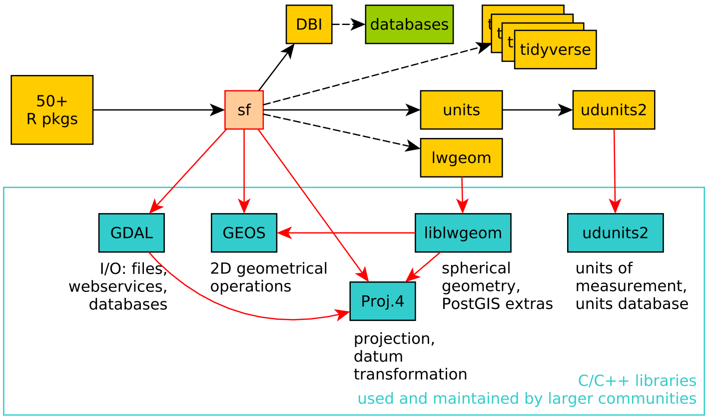

Figure 3: Dependencies of **sf** on other R packages and external system libraries.

Rasters, time series, and units

For some users, starting with **sf** feels like closing an old book (**sp**), and opening a new one. But it is not as if this new book has a similar content, or size. It is unsure when, or even whether at all, the hundreds of packages that use **sp** classes will be modified to use the `sf` classes.

The most heard question is where raster data are in this new book: **sp** provides simple classes for gridded data, **raster** (Hijmans, 2016b) provides heavy duty classes and a massive number of methods to work with them, tightly integrated with the **sp** vector classes. The current version of **raster** accepts **sf** objects in some of its functions by converting them to (the smaller set of) **sp** objects. At the time of writing this, we can only say that this is an area of active discussion, exploration and development, and we will be happy to point interested readers to where the public components of this discussion are taking place.

Besides raster data, time series for spatial features (e.g. for monitoring stations) are hard to map onto `sf` objects: one would either have to put time slices in columns, or add a time column and repeat the feature geometry for each observation. Raster data, spatial time series, and raster time series are the focus of the stars project.

A new aspect of the package is the ability to retrieve spatial measures and to set e.g. distance parameters with explicit measurement units (Pebesma et al., 2016):

```
> st_area(st_transform(nc[1, ], 2264)) # NC state plane, US foot

12244955726 US_survey_foot^2

> st_crs(2264)$units

[1] "us-ft"

> st_area(st_transform(nc[1, ], 2264)) %>% units::set_units(km^2) # convert:

1137.598 km^2
```

which might first confuse, but has the potential to prevent a whole category of scientific errors.

Connections to other computer systems and scalability

In many cases, analysing spatial data with R starts with importing data, or ends with exporting data, from or to a file or database. The ability to do this is primarily given by the well-known text (WKT) and well-known binary (WKB) serialisations that are part of the simple feature standard, and that are supported by **sf**. Communication with the GDAL, GEOS, and liblwgeom libraries uses WKB both ways. GDAL currently has drivers for 93 different spatial vector data connections (file formats, data bases, web services). Figure 3 shows the dependencies of **sf** on other R packages and system libraries. A reason to build upon these libraries is that they are used and maintained by, and hence reflect concensus of, the large community of spatial data experts outside R.

Besides using GDAL, **sf** can directly read and write from and to spatial databases. This currently works with PostGIS using **RPostgreSQL**; making this work with **RPostgres** and in general with spatial databases using **DBI** is under active development. Initial experiments indicate that working with massive, out-of-memory spatial databases in R is possible using the **dbplyr** framework. This not only removes the memory limits of R, but also benefits from the persistent spatial indexes of these databases.

For planar data, **sf** builds its spatial indexes on the fly for spatial binary predicates (`st_intersects`, `st_contains` etc.) and its binary operations (`st_intersection`, `st_difference` etc). A blog post about the spatial indexes in **sf** describes how using indexes makes these operations feasible for larger in-memory datasets. For spherical data, indexes e.g. provided by liblwgeom or by s2 still need to be explored.

Summary and further reading

We present a new package, **sf**, for simple features in R, as a modern alternative for parts of the **sp**-family of packages. It provides new foundational classes to handle spatial vector data in R, and has been received with considerable enthusiasm and uptake. While implementing **sf**, several well-proven concepts have been maintained (separation of geometries and attributes, libraries used), new links have been made (**dplyr**, **ggplot2**, spatial databases), and new concepts have been explored and implemented (units, spatial indexes).

For further reading into the full capabilities of **sf** and its rationale, the reader is refered to the six vignettes that come with the package.

Acknowledgments

Writing **sf** would not have been possible without all the prior work and continuous help of Roger Bivand. Package contributers are Ian Cook, Tim Keitt, Michael Sumner, Robin Lovelace, Hadley Wickham, Jeroen Ooms, and Etienne Racine. All contributors to GitHub issues are also acknowledged. Special thanks go to Dirk Eddelbuettel for developing **Rcpp** (Eddelbuettel et al., 2011; Eddelbuettel, 2013).

Support from the R Consortium has been very important for the development, visibility and fast adoption of **sf**, and is gratefully acknowledged. Anonymous reviewers are acknowledged for helpful comments.

Bibliography

R. Bivand and C. Rundel. *rgeos: Interface to Geometry Engine - Open Source ('GEOS')*, 2017. URL https://CRAN.R-project.org/package=rgeos. R package version 0.3-25. [p439]

R. Bivand, T. Keitt, and B. Rowlingson. *rgdal: Bindings for the 'Geospatial' Data Abstraction Library*, 2017. URL https://CRAN.R-project.org/package=rgdal. R package version 1.2-15. [p439]

R. S. Bivand, E. Pebesma, and V. Gomez-Rubio. *Applied Spatial Data Analysis with R, Second Edition*. Springer-Verlag, 2013. URL http://www.asdar-book.org/. [p439]

H. Butler, M. Daly, A. Doyl, S. Gillies, S. Hagen, and T. Schaub. The GeoJSON format, 2016. ISSN 2070-1721. URL https://tools.ietf.org/html/rfc7946. [p439]

V. H. Do, C. Thomas-Agnan, and A. Vanhems. Accuracy of areal interpolation methods for count data. *Spatial Statistics*, 14:412 – 438, 2015. URL https://doi.org/10.1016/j.spasta.2015.07.005. [p441]

D. Eddelbuettel. *Seamless R and C++ Integration with Rcpp*. Springer-Verlag, New York, 2013. [p445]

D. Eddelbuettel, R. François, J. Allaire, K. Ushey, Q. Kou, N. Russel, J. Chambers, and D. Bates. Rcpp: Seamless R and C++ integration. *Journal of Statistical Software*, 40(8):1–18, 2011. [p445]

A. Fabri and S. Pion. CGAL: The computational geometry algorithms library. In *Proceedings of the 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 538–539. ACM, 2009. [p442]

GEOS Development Team. *GEOS - Geometry Engine, Open Source*. Open Source Geospatial Foundation, 2017. URL https://trac.osgeo.org/geos/. [p439]

J. R. Herring. OpenGIS implementation standard for geographic information-simple feature access-part 2: SQL option. *Open Geospatial Consortium Inc*, 2010. URL http://portal.opengeospatial.org/files/?artifact_id=25354. [p439]

J. R. Herring. OpenGIS implementation standard for geographic information-simple feature access-part 1: Common architecture. *Open Geospatial Consortium Inc*, page 111, 2011. URL http://portal.opengeospatial.org/files/?artifact_id=25355. [p439]

R. J. Hijmans. *geosphere: Spherical Trigonometry*, 2016a. URL https://CRAN.R-project.org/package=geosphere. R package version 1.5-5. [p442]

R. J. Hijmans. *Raster: Geographic Data Analysis and Modeling*, 2016b. URL https://CRAN.R-project.org/package=raster. R package version 2.5-8. [p444]

ISO. *Geographic Information – Simple Feature Access – Part 1: Common Architecture*, 2004. URL https://www.iso.org/standard/40114.html. ISO 19125-1:2004. [p439]

R. O. Obe and L. S. Hsu. *PostGIS in Action*. Manning Publications Co., 2015. [p439]

E. Pebesma. *lwgeom: Bindings to Selected 'liblwgeom' Functions for Simple Features*. URL https://CRAN.R-project.org/package=lwgeom. R package version 0.1-5. [p442]

E. Pebesma. *sf: Simple Features for R*, 2018. URL https://CRAN.R-project.org/package=sf. R package version 0.6-1. [p439]

E. Pebesma, T. Mailund, and J. Hiebert. Measurement units in R. *The R Journal*, 8(2):486–494, 2016. URL https://journal.r-project.org/archive/2016-2/pebesma-mailund-hiebert.pdf. [p444]

E. J. Pebesma and R. S. Bivand. Classes and methods for spatial data in R. *R News*, 5(2):9–13, 2005. URL https://CRAN.R-project.org/doc/Rnews/. [p439]

M. Perry and J. Herring. OGC GeoSPARQL-a geographic query language for RDF data. *OGC Implementation Standard, ref: OGC*, 2012. [p439]

E. Rubak and J. Ooms. *S2: Google's S2 Library for Geometry on the Sphere*, 2017. URL https://CRAN.R-project.org/package=s2. R package version 0.1-1. [p442]

S. Scheider, B. Gräler, E. Pebesma, and C. Stasch. Modeling spatiotemporal information generation. *International Journal of Geographical Information Science*, 30(10):1980–2008, 2016. URL https://doi.org/10.1080/13658816.2016.1151520. [p439]

F. Warmerdam. The geospatial data abstraction library. In *Open Source Approaches in Spatial Data Handling*, pages 87–104. Springer-Verlag, 2008. [p439]

H. Wickham. Tidy data. *Journal of Statistical Software, Articles*, 59(10):1–23, 2014. ISSN 1548-7660. URL https://doi.org/10.18637/jss.v059.i10. [p439]

H. Wickham. *Ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag, 2016. [p439]

H. Wickham. *Tidyverse: Easily Install and Load the 'Tidyverse'*, 2017. URL https://CRAN.R-project.org/package=tidyverse. R package version 1.1.1. [p439]

H. Wickham, R. Francois, L. Henry, and K. Müller. *dplyr: A Grammar of Data Manipulation*, 2017. URL https://CRAN.R-project.org/package=dplyr. R package version 0.7.4. [p439]

Edzer Pebesma
Institute for Geoinformatics
Heißenbergstraße 2
Münster, Germany
ORCiD: 0000-0001-8049-7069
edzer.pebesma@uni-muenster.de