Spatial data in R: simple features and future perspectives

Edzer Pebesma (ifgi, Münster, DE)
Roger Bivand (NHH, Bergen, NO)

What are simple features?

First: what is meant by a *feature*?

- any *thing* in the (real) world
- persons, cars, buildings, rivers, mountains, ...
- but also surfaces, and collections of all of these

*Simple features* refer to:

- a *common architecture for simple feature geometry*
- a *formal standard*: OGC 06-103r4; ISO 19125:
  “OpenGIS Implementation Standard for Geographic information - Simple feature access - Part 1: Common architecture”
- a set of encodings:
  - WKT: “well known text”
  - WKB: “well known binary”
What are simple features?

First: what is meant by a feature?

- any thing in the (real) world
- persons, cars, buildings, rivers, mountains, ...
- but also surfaces, and collections of all of these

*Simple features* refer to:

- a *common architecture for simple feature geometry*
- a *formal standard*: OGC 06-103r4; ISO 19125:
- “OpenGIS Implementation Standard for Geographic information - Simple feature access - Part 1: Common architecture”
- a set of encodings:
  - WKT: “well known text”
  - WKB: “well known binary”
Figure 1: Geometry class hierarchy
How do simple features look like?

Encoded as well-known-text:

POINT(0 0)
LINESTRING(0 0,1 1,1 2)
POLYGON((0 0,4 0,4 4,0 4,0 0),(1 1, 2 1, 2 2, 1 2,1 1))
MULTIPOINT((0 0),(1 2))
MULTILINESTRING((0 0,1 1,1 2),(2 3,3 2,5 4))
MULTIPOLYGON(((0 0,4 0,4 4,0 4,0 0),(1 1,2 1,2 2,1 2,1 1)),
((-1 -1,-1 -2,-2 -2,-2 -1,-1 -1)))
GEOMETRYCOLLECTION(POINT(2 3),LINESTRING(2 3,3 4))

Polygons:

- first polygon: enclosing, counter-clockwise
- second, third, ... polygons: holes, clockwise
2D-only?

No:

POINT Z(0 0 0)
POINT M(0 0 0)
POINT ZM(0 0 0 0)
LINESTRING Z(0 0 1,1 1 1,1 2 3)
POLYGON M((0 0 1,4 0 0,4 4 2,0 4 1,0 0 1))

- Z: third spatial dimension (altitude, height)
- M: “measure”: “A Point value may include an m coordinate value. The m coordinate value allows the application environment to associate some measure with the point values. For example: A stream network may be modeled as multilinestring value with the m coordinate values measuring the distance from the mouth of stream. ”

M cannot be thought of as usual attributes of a polygon or line: an M value is associated with each point of a polygon, line, ...
Current situation in R, w/o SF

- **Current R (CRAN)**
  - `sp`
  - `rgeos`
  - `rgdal`

- **GEOS**
  - Geometry operations; JTS in C++

- **GDAL**
  - Read/write 142 raster, 84 vector formats; many manipulation routines

- **PROJ.4**
  - Coordinate transformation and conversion

**Simple features in R**

**CRAN package**

**github package**

**API (open source)**
Current, with SF pieces

Current R (CRAN)
- sp
- rgeos
- wkb
- rgdal

GEOS: geometry operations; JTS in C++

Simple features in R
- Well-known text (character)
- Well-known binary (raw)
- R native objects
- External pointers to GDAL objects

GDAL: read/write 142 raster, 84 vector formats; many manipulation routines

PROJ.4: coordinate transformation and conversion

CRAN package
github package
API (open source)
Where we want to go

- Current R (CRAN)
  - sp
  - rgeos
  - rgdal

- Simple features in R
  - sf

- GEOS: geometry operations; JTS in C++

- GDAL: read/write 142 raster, 84 vector formats; many manipulation routines

- PROJ.4: coordinate transformation and conversion

- CRAN package
- Github package
- API (open source)
Simple features in R: a proposal

We usually work with sets of simple features, where feature properties (attributes) are in a `data.frame` or similar. Useful constraints will be:

- sets will have a single type (which can, in case of a mix, be `GEOMETRYCOLLECTION`)
- sets will have a single coordinate reference system

Keep it simple:

- feature sets should be a list, and work as a list column in `data.frame` and the like (tidy!)
- use numeric for single point, matrix for a set of points, list for set of sets
- use S3
- of class `sf`, attributes `type` (chr), `epsg` (int) and `proj4string` (chr)
```
> (d = data.frame(a = 1:3, b = I(list(1:2, c(1,3,5), 10:5))))

   a   b
1 1    1, 2
2 2    1, 3, 5
3 3   10, 9, 8, ...

> summary(d)

     a      b
  Min. :1.0  2 -none- numeric
  1st Qu.:1.5  3 -none- numeric
     Median :2.0  6 -none- numeric
       Mean :2.0
    3rd Qu.:2.5
       Max. :3.0

> library(tibble)
> data_frame(a = 1:3, b = list(1:2, c(1,3,5), 10:5))

Source: local data frame [3 x 2]

   a   b
1 1 <int [2]>
2 2 <dbl [3]>
3 3 <int [6]>
```
R implementation: proposal

Although 7 of them are dominant, there are 72 types:

<table>
<thead>
<tr>
<th>XY</th>
<th>XYZ</th>
<th>XYM</th>
<th>XYZM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Geometry</td>
<td>Geometry Z</td>
<td>Geometry M</td>
<td>Geometry ZM</td>
</tr>
<tr>
<td>Point</td>
<td>Point Z</td>
<td>Point M</td>
<td>Point ZM</td>
</tr>
<tr>
<td>LineString</td>
<td>LineString Z</td>
<td>LineString M</td>
<td>LineString ZM</td>
</tr>
<tr>
<td>Polygon</td>
<td>Polygon Z</td>
<td>Polygon M</td>
<td>Polygon ZM</td>
</tr>
<tr>
<td>MultiPoint</td>
<td>MultiPoint Z</td>
<td>MultiPoint M</td>
<td>MultiPoint ZM</td>
</tr>
<tr>
<td>MultiLineString</td>
<td>MultiLineString Z</td>
<td>MultiLineString M</td>
<td>MultiLineString ZM</td>
</tr>
<tr>
<td>MultiPolygon</td>
<td>MultiPolygon Z</td>
<td>MultiPolygon M</td>
<td>MultiPolygon ZM</td>
</tr>
<tr>
<td>GeometryCollection</td>
<td>GeometryCollection Z</td>
<td>GeometryCollection M</td>
<td>GeometryCollection ZM</td>
</tr>
<tr>
<td>CircularString</td>
<td>CircularString Z</td>
<td>CircularString M</td>
<td>CircularString ZM</td>
</tr>
<tr>
<td>CompoundCurve</td>
<td>CompoundCurve Z</td>
<td>CompoundCurve M</td>
<td>CompoundCurve ZM</td>
</tr>
<tr>
<td>CurvePolygon</td>
<td>CurvePolygon Z</td>
<td>CurvePolygon M</td>
<td>CurvePolygon ZM</td>
</tr>
<tr>
<td>MultiCurve</td>
<td>MultiCurve Z</td>
<td>MultiCurve M</td>
<td>MultiCurve ZM</td>
</tr>
<tr>
<td>MultiSurface</td>
<td>MultiSurface Z</td>
<td>MultiSurface M</td>
<td>MultiSurface ZM</td>
</tr>
<tr>
<td>Curve</td>
<td>Curve Z</td>
<td>Curve M</td>
<td>Curve ZM</td>
</tr>
<tr>
<td>Surface</td>
<td>Surface Z</td>
<td>Surface M</td>
<td>Surface ZM</td>
</tr>
<tr>
<td>PolyhedralSurface</td>
<td>PolyhedralSurface Z</td>
<td>PolyhedralSurface M</td>
<td>PolyhedralSurface ZM</td>
</tr>
<tr>
<td>TIN</td>
<td>TIN Z</td>
<td>TIN M</td>
<td>TIN ZM</td>
</tr>
<tr>
<td>Triangle</td>
<td>Triangle Z</td>
<td>Triangle M</td>
<td>Triangle ZM</td>
</tr>
</tbody>
</table>
How does a spatial table look, in PostGIS?

```
edzer@gin-edzer:~$ psql postgis
psql (9.3.13)
Type "help" for help.

postgis=# select * from meuse2 limit 2;
    id |    zinc |            geom
-----+--------+-------------------------------------
     1 |  1022  | 0101000020E61000000000000000008046400000000000804640
     2 |  1141  | 010100002040710000000000000819064100000000D85B1441
(2 rows)

postgis=# select zinc, ST_asText(geom) from meuse2 limit 2;
     zinc |               st_astext
--------+----------------------------------
    1022 | POINT(181072 333611)
    1141 | POINT(181025 333558)
(2 rows)
```

PostGIS keeps in two other tables the information

- that `meuse2` has geometry column `geom`, the CRS ID of it
- what this CRS ID refers to (proj4string, WKT of CRS)
Reading WKT through DBI/RPostgreSQL

```r
> library(RPostgreSQL)
> drv <- dbDriver("PostgreSQL")
> con <- dbConnect(drv,
+  dbname="postgis", user="edzer", password="pw",
+  host="localhost", port='5432'
> query = "select zinc, geom from meuse2 limit 2;"
> (tbl = fetch(dbSendQuery(con, query)))

    zinc        geom
  1 1022 0101000020407100000000000801A064100000000AC5C1441
  2 1141 010100002040710000000000000819064100000000D85B1441

Warning message:
In postgresqlExecStatement(conn, statement, ...) :
  RS-DBI driver warning: (unrecognized PostgreSQL field type geometry (id:16393) in column 1)

> sapply(tbl, class)

           zinc     geom
"numeric" "character"

> query = "select zinc, ST_asText(geom) from meuse2 limit 2;"
> (tbl = fetch(dbSendQuery(con, query)))

    zinc    st_astext
  1 1022  POINT(181072 333611)
  2 1141  POINT(181025 333558)

> sapply(tbl, class)

           zinc    st_astext
"numeric" "character"
```
sf: design considerations (1/2)

- read + write using external libraries (GDAL)
- support PROJ.4 compatible CRS handling
- CRS transformation/conversion through GDAL (= PROJ.4)
- “stick” to S3
- single SF items shall have a class: sfi, or POINT, POLYGON etc
- sets of SF (list column) shall have a class sfc, and have bbox and CRS attributes
- sf table objects with a single sfc shall have a class: sf
- sf shall extend its base class:
  ```r
  > a = data.frame(x = 1:3)
  > (class(a) = c("sf", class(a)))
  [1] "sf" "data.frame"
  ```
- balance simplicity with sp compatibility
- use numeric for single point, matrix for a set of points, list for set of sets
sf: design considerations (1/2)

- start with the low-hanging fruit of the 2D (XY) geometries POINT, multipoint, linestring, polygon, multilinestring, multipolygon, geometrycollection
- keep the path open for all 68 SF types (inherit: XY \Rightarrow XYZ, XYM \Rightarrow XYZM)
- add functions that convert sf objects into the arguments needed by grid::polygonGrob and the like.
- document for each of the non-spatial variables how it relates to the spatial features (constant, aggregate, NA)
Discussion

- it is time for simple features in R; package \texttt{sf} will be doing this
- simple features are standard and ubiquitous (databases, geojson, leaflet, ...)
- we found support by R consortium; positive feedback from ESRI too
- now that \textit{list columns} are tidy, so are we
- \texttt{sf} will focus on I/O, interoperability, and functionality
  - with R plot methods (base, grid)
  - external data sources (GDAL)
  - geometry operations (intersections etc.)
  - migration path, conversion to/from \texttt{sp}