Handling shapefiles in the spatstat package

Adrian Baddeley, Rolf Turner and Ege Rubak

2026-11-27
spatstat version 3.6-4

This vignette explains how to read data into the spatstat package from files in the popular 'shapefile' format.

This vignette is part of the documentation included in spatstat version 3.6-4. The information applies to spatstat versions 1.36-0 and above.

1 Shapefiles

A shapefile represents a list of spatial objects — a list of points, a list of lines, or a list of polygonal regions — and each object in the list may have additional variables attached to it.

A dataset stored in shapefile format is actually stored in a collection of text files, for example

```
mydata.shp
mydata.prj
mydata.sbn
mydata.dbf
```

which all have the same base name mydata but different file extensions. To refer to this collection you will always use the filename with the extension shp, for example mydata.shp.

2 Helper package

We'll use the sf package to handle shapefile data. Previously the now defunct package maptools was used in this vignette together with the older (still functional) sp.

Both sf and sp support a standard set of spatial data types in R. The sp package uses S4 classes and sf uses S3 classes. These standard data types can be handled by many other packages, so it is useful to convert your spatial data into one of the data types supported by sf or sp. With the retirement of maptools there are no direct conversion tools between spatstat data types and sp data types and for this reason we recommend using sf if possible. However, if you are already using sp formats you can convert data from sp format to sf format and then to spatstat format.

To read and write files in shapefile format sf uses the system library GDAL and newer versions of sp uses sf under the hood.

3 Caveat about longitude-latitude coordinates

The shapefile format supports geographical coordinates, usually longitude-latitude coordinates, which specify locations on the curved surface of the Earth. However, spatstat deals only with spatial data on a flat two-dimensional plane.

If you follow the recommendation to read in shapefile (or other formats) data with `sf` and then converting to `spatstat` format you should encounter an error if you try to convert data in longitude-latitude coordinates directly to `spatstat` format. However, if you manage to read in longitude-latitude data and convert them into `spatstat` objects without `sf`, longitude and latitude coordinates will most likely be treated as x and y coordinates, so that the Earth's surface is effectively mapped to a rectangle. This mapping distorts distances and areas.

If your study region is a *small* region of the Earth's surface (about 3 degrees, 180 nautical miles, 200 statute miles, 320 km across) then a reasonable approach is to use the latitude and longitude as x and y coordinates, after multiplying the longitude coordinates by the cosine of the latitude of the centre of the region. This will approximately preserve areas and distances. This calculation is a simple example of a *geographical projection* and there are some much better projections available. It may be wise to use `st_transform()` in `sf` to perform the appropriate projection for you, and then to convert the projected data into `spatstat` objects.

If your study region is a large part of the sphere, then your data may not be amenable to the techniques provided by `spatstat` because the geometry is fundamentally different. Please consider the extension package `spatstat.sphere`.

4 How to read shapefiles into `spatstat`

To read shapefile data into `spatstat`, you follow two steps:

1. using the facilities of `sf`, read the shapefiles and store the data in one of the standard formats supported by `sf`.
2. convert the `sf` data type into one of the data types supported by `spatstat`.

4.1 Read shapefiles using `sf`

Here's how to read shapefile data.

1. ensure that the package `sf` is installed.
2. start R and load the package:

   ```
   > library(sf)
   ```

3. read the shapefile into an object in the `sf` package using `st_read`, for example

   ```
   > x <- st_read(system.file("shape/nc.shp", package="sf"))
   ```

4. This will read in the data as an object of class `sf`. This is basically a `data.frame` with a designated geometry column (of class `sfc`). All other columns contain information/features (marks in `spatstat` terminology) related to the geometries. For example the geometry column could be a list of polygonal boundaries of the counties of a state and the other columns could contain the name of the county and other registered values of interest. To find out what kind of spatial objects are represented by the dataset, inspect the class of the geometry column:

   ```
   > st_geometry_type(x, by_geometry = FALSE)
   ```

   There are many possible classes but the ones of main interest here are:

   - `POINT` (or `MULTIPOINT`) indicating each row refers to a point (or several points)

- `LINESTRING` (or `MULTILINESTRING`) indicating each row refers to a collection of sequentially connected straight line segments (or several of these)
- `POLYGON` (or `MULTIPOLYGON`) indicating each row refers to a polygon which may have holes inside (or several of these)

4.2 Convert data to `spatstat` format

To convert the dataset to an object in the `spatstat` package, the procedure depends on the type of the geometry column, as explained below.

Both packages `sf` and `spatstat` must be **loaded** in order to convert the data.

4.2.1 Geometries of class `POINT`/`MULTIPOINT`

A `sf` object `x` with geometry column of class `POINT` represents a spatial point pattern. Use `as.ppp(x)` to convert it to a spatial point pattern in `spatstat`:

```
> X <- as.ppp(x)
```

(The conversion is performed by `as.ppp.sf`, a function in `sf`.)

The window for the point pattern will be taken from the bounding box of the points. You will probably wish to change this window, usually by taking another dataset to provide the window information. Use `[.ppp` to change the window: if `X` is a point pattern object of class `"ppp"` and `W` is a window object of class `"owin"`, type

```
> X <- X[W]
```

If the `sf` object `x` contains other columns than the geometry column these are additional variables ('marks') attached to each point.

At the time of writing `as.ppp(x)` will unfortunately only use the first column of additional data as the `marks` of the point pattern `X`. In that case you can extract the data frame of auxiliary data by `df <- st_drop_geometry(x)` and manually assign them as marks in `spatstat`:

```
> df <- st_drop_geometry(x)
> X <- as.ppp(x)
> marks(X) <- df
```

If the class of the geometry column is `MULTIPOINT` you need to first cast to `POINT` and then convert as described above:

```
> x_point <- st_cast(x, "POINT")
> X <- as.ppp(x_point)
```

If you have a combination of `POINT` and `MULTIPOINT` you may first need an intermediate cast to `MULTIPOINT` before casting to `POINT`:

```
> x_multipoint <- st_cast(x, "MULTIPOINT")
> x_point <- st_cast(x, "POINT")
> X <- as.ppp(x_point)
```

4.2.2 Geometries of class LINESTRING or MULTILINESTRING

A "line segment" is the straight line between two points in the plane.

In the spatstat package, an object of class psp ("planar segment pattern") represents a pattern of line segments, which may or may not be connected to each other (like matches which have fallen at random on the ground).

In the sf package, a geometry column of class LINESTRING represents a **list** of **connected curves**, each curve consisting of a sequence of straight line segments that are joined together (like several pieces of a broken bicycle chain.)

For a geometry column of class MULTILINESTRING each element of the top list (the geometry column) is it self a list of connected curves.

list of lists

So the spatstat and sf data types do not correspond exactly.

The list of connected curves in a LINESTRING column may be useful when representing a single river system where each branch of the river may be in its own element of the list (row of the column).

The list-of-lists hierarchy in a MULTILINESTRING column is useful when representing river of a continent where each element of the primary list (row in the column) would correspond to a river system and each of these would be a list of the branches of the river system.

For example, if Africa is an object of class sf with geometry column of class MULTILINESTRING representing the main rivers of Africa, then Africa will have hundreds of rows representing each of the rivers. The geometry column geo <- st_geometry(Africa) is then a list, where geo[[i]] might represent the total river system the i-th river (e.g. the Nile). The branches of each river system consist of several different curved lines. Thus geo[[i]][[j]] would represent the jth branch of the i-th river and would be of class LINESTRING.

For an object x of class sf with geometry column of class MULTILINESTRING or LINESTRING, there are several things that you might want to do:

1. collect together all the line segments (all the segments that make up all the connected curves) and store them as a single object of class psp.

   To do this, use as.psp(x) to convert it to a spatial line segment pattern.

   Note: Any auxilliary information stored in other columns is automatically attached as marks to the line segments of the spatstat psp object.

2. convert each connected curve to an object of class psp, keeping different connected curves separate.

   To do this, type something like the following:

   ```
   > out <- lapply(geo, function(z) { lapply(z, as.psp) })
   ```

   (The conversion is performed by as.psp.MULTILINESTRING, a function in sf. So the sf and spatstat packages must be loaded in order for this to work.)

   Any auxiliary data in other columns can be attached as marks by this command:

   ```
   > dat <- st_drop_geometry(Africa)
   > for(i in seq(nrow(dat))){
   +   out[[i]] <- lapply(out[[i]], "marks<-", value=dat[i, , drop=FALSE])
   + }
   ```

The result will be a **list of lists** of objects of class `psp`. Each one of these objects represents a connected curve, although the `spatstat` package does not know that. The list structure will reflect the list structure of the original `MULTILINESTRING` object `x`. If that's not what you want, then use `curvelist <- do.call("c", out)` or

```
> curvegroup <- lapply(out, function(z) { do.call("superimpose", z)})
```

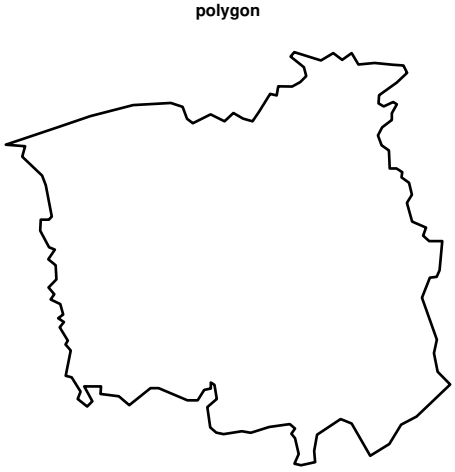
to collapse the list-of-lists-of-`psp`'s into a list-of-`psp`'s. In the first case, `curvelist[[i]]` is a `psp` object representing the `i`-th connected curve. In the second case, `curvegroup[[i]]` is a `psp` object containing all the line segments in the `i`-th group of connected curves (for example the `i`-th river system – the Nile – in the `Africa` example).

The window for the spatial line segment pattern can be specified as an argument `window` to the function `as.psp`.

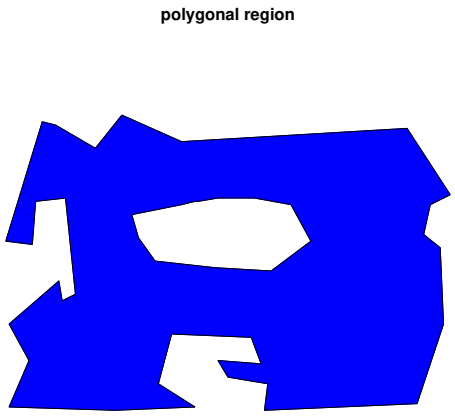
In the `spatstat` package, an object of class `psp` (representing a collection of line segments) may have a data frame of marks. Note that each *line segment* in a `psp` object may have different mark values. For the converted data the mark variables attached to a particular *group of connected lines* in the `sf` object, will be duplicated and attached to each *line segment* in the resulting `psp` object.

4.2.3 Geometries of class `POLYGON` or `MULTIPOLYGON`

First, so that we don't go completely crazy, let's introduce some terminology. A *polygon* is a closed curve that is composed of straight line segments. You can draw a polygon without lifting your pen from the paper. This is called a `POLYGON` in `sf` terminology.



A *polygonal region* is a region in space whose boundary is composed of straight line segments. A polygonal region may consist of several unconnected pieces, and each piece may have holes. The boundary of a polygonal region consists of one or more polygons. To draw the boundary of a polygonal region, you may need to lift and drop the pen several times. This is called a `MULTIPOLYGON` in `sf` terminology.



An object of class `owin` in `spatstat` represents a polygonal region. It is a region of space that is delimited by boundaries made of lines.

An object `x` with geometry column of class `MULTIPOLYGON` represents a **list of polygonal regions**. For example, a single geometry column of class `MULTIPOLYGON` could store information about every State in the United States of America (or the United States of Malaysia). Each State would be a separate polygonal region (and it might contain holes such as lakes).

There are two things that you might want to do with a geometry column of class `MULTIPOLYGON`:

1. combine all the polygonal regions together into a single polygonal region, and convert this to a single object of class `owin`.

   For example, you could combine all the States of the USA together and obtain a single object that represents the territory of the USA.

   To do this, use `as.owin(x)`. The result is a single window (object of class `"owin"`) in the `spatstat` package.

2. keep the different polygonal regions separate; convert each one of the polygonal regions to an object of class `owin`.

   For example, you could keep the States of the USA separate, and convert each State to an object of class `owin`.

   To do this, type the following:

   ```
   > geo <- st_geometry(x)
   > windows <- lapply(geo, as.owin)
   ```

   The result is a list of objects of class `owin`. Often it would make sense to convert this to a tessellation object, by typing

   ```
   > te <- tess(tiles=windows)
   ```

(The conversion is performed by `as.owin.MULTIPOLYGON`, a function in `sf`. So the `sf` and `spatstat` packages must be loaded in order for this to work.)

The following is different from what happened in previous versions of `spatstat` (prior to version `1.36-0`.)

During the conversion process, the geometry of the polygons will be automatically "repaired" if needed. Polygon data from shapefiles often contain geometrical inconsistencies such as self-intersecting boundaries and overlapping pieces. For example, these can arise from small errors in curve-tracing. Geometrical inconsistencies are tolerated in an object with geometry column of class `MULTIPOLYGON` which is a list of lists of polygonal curves. However, they are not tolerated in an object of class `owin`, because an `owin` must specify a well-defined region of space. These data inconsistencies must be repaired to prevent technical problems. In `spatstat` polygon-clipping code is used to automatically convert polygonal lines into valid polygon boundaries. The repair process changes the number of vertices in each polygon, and the number of polygons (if you chose option 1). To disable the repair process, set `spatstat.options(fixpolygons=FALSE)`.

4.2.4 Auxiliary information

Typically an object `x` of class `sf` with geometry column of type (`MULTI`)`POLYGON` has other columns with additional variables attached to each polygon. The data frame of auxiliary data is extracted by `df <- st_drop_geometry(x)`.

There is currently no facility in `spatstat` for attaching marks to an `owin` object directly, but if you have collected the separate regions in a tessellation you can attach marks to the tessellation, so the entire workflow becomes:

```
> geo <- st_geometry(x)
> df <- st_drop_geometry(x)
> windows <- lapply(geo, as.owin)
> te <- tess(tiles=windows)
> marks(te) <- df
```

However, if the regions are kept as a list of `owin` objects it is possible to take advantage of `spatstat`'s support of objects called **hyperframes**, which are like data frames except that the entries can be any type of object. Thus we can represent the data in `spatstat` as follows:

```
> h <- hyperframe(window=windows)
> h <- cbind.hyperframe(h, df)
```

Then `h` is a hyperframe containing a column of `owin` objects followed by the columns of auxiliary data.