5

Further Methods for Handling Spatial Data

This chapter is concerned with a more detailed explanation of some of the methods that are provided for working with the spatial classes described in Chap. 2. We first consider the question of the spatial support of observations, going on to look at overlay and sampling methods for a range of classes of spatial objects. Following this, we cover combining the data stored in the `data` slot of `Spatial*DataFrame` objects with additional data stored as vectors and data frames, as well as the combination of spatial objects. We also apply some of the functions that are available for handling and checking polygon topologies, including the dissolve operation.

5.1 Support

In data analysis in general, the relationship between the abstract constructs we try to measure and the operational procedures used to make the measurements is always important. Very often substantial metadata volumes are generated to document the performance of the instruments used to gather data. Naturally, the same applies to spatial data. Positional data need as much care in documenting their collection as other kinds of data. When approximations are used, they need to be recorded as such. Part of the issue is the correct recording of projection and datum, which are covered in Chap. 4. The time-stamping of observations is typically useful, for example when administrative boundaries change over time.

The recording of position for surveying, for example for a power company, involves the inventorying of cables, pylons, transformer substations, and other infrastructure. Much GIS software was developed to cater for such needs, inventory rather than analysis. For inventory, arbitrary decisions, such as placing the point coordinate locating a building by the right-hand doorpost facing the door from the outside, have no further consequences. When, however, spatial data are to be used for analysis and inference, the differences between arbitrary assumptions made during observation and other possible spatial representations of the phenomena of interest will feed through to the conclusions. The adopted representation is known as its support, and is discussed by Waller and Gotway (2004, pp. 38–39). The point support of a dwelling may be taken as the point location of its right-hand doorpost, a soil sample may have point support of a coordinate surveyed traditionally or by GPS. But the dwelling perhaps should have polygonal support, and in collecting soil samples, most often the point represents a central position in the circle or square used to gather a number of different samples, which are then bagged together for measurement.

An example of the effects of support is the impact of changes in voting district boundaries in election systems, which are not strictly proportional. The underlying voting behaviour is fixed, but different electoral results can be achieved by tallying results in different configurations or aggregations of the voters' dwellings.[1] When carried out to benefit particular candidates or parties, this is known as gerrymandering. The aggregations are arbitrary polygons, because they do not reflect a political entity as such. This is an example of change of support, moving from the position of the dwelling of the voter to some aggregation. Change of support is a significant issue in spatial data analysis, and is introduced in Schabenberger and Gotway (2005, pp. 284–285). A much more thorough treatment is given by Gotway and Young (2002), who show how statistical methods can be used to carry through error associated with change of support to further steps in analysis. In a very similar vein, it can be argued that researchers in particular subject domains should consider involving statisticians from the very beginning of their projects, to allow sources of potential uncertainty to be instrumented if possible. One would seek to control error propagation when trying to infer from the data collected later during the analysis and reporting phase (Guttorp, 2003; Wikle, 2003). An example might be marine biologists and oceanographers not collecting data at the same place and time, and hoping that data from different places and times could be readily combined without introducing systematic error.

One of the consequences of surveying as a profession being overtaken by computers, and of surveying data spreading out to non-surveyor users, is that understanding of the imprecision of positional data has been diluted. Some of the imprecision comes from measurement error, which surveyors know from their training and field experience. But a digital representation of a coordinate looks very crisp and precise, deceptively so. Surveying and cartographic representations are just a summary from available data. Where no data were collected, the actual values are guesswork and can go badly wrong, as users of maritime charts routinely find. Further, support is routinely changed for purposes of visualisation: contours or filled contours representing a grid make the

[1] The CRAN **BARD** package for automated redistricting and heuristic exploration of redistricter revealed preference is an example of the use of R for studying this problem.

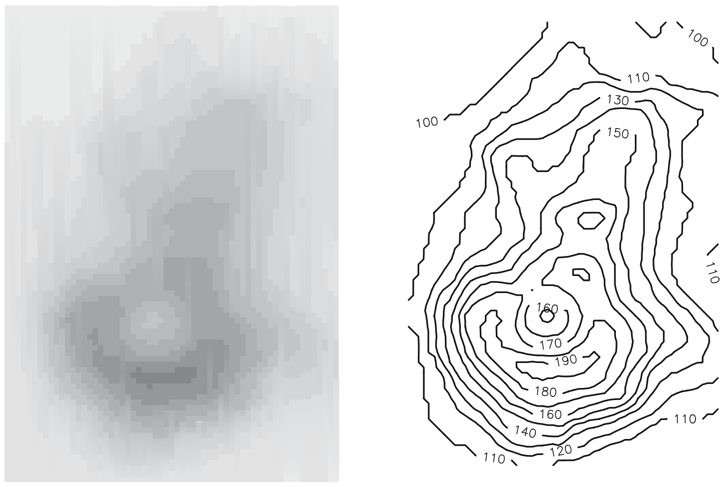

Fig. 5.1. Image plot and contour plot representations of Maunga Whau from the standard R `volcano` data set, for the same elevation class intervals (rotated to put north at the top)

data look much less ‘chunky’ than an image plot of the same data, as Fig. 5.1 shows. In fact, the data were digitised from a paper map by Ross Ihaka, as much other digital elevation data have been, and the paper map was itself a representation of available data, not an exact reproduction of the terrain. Even SRTM data can realistically be used only after cleaning; the 3 arcsec data used in Sect. 2.7 were re-sampled from noisier 1 arcsec data using a specific re-sampling and cleaning algorithm. A different algorithm would yield a slightly different digital elevation model.

While we perhaps expect researchers wanting to use R to analyse spatial data to be applied domain scientists, it is worth noting that geographical information science, the field of study attempting to provide GIS with more consistent foundations, is now actively incorporating error models into position measurement, and into spatial queries based on error in observed values. Say we are modelling crop yield based on soil type and other variables, and our spatial query at point i returns `"sand"`, when in fact the correct value at that location is `"clay"`, our conclusions will be affected. The general application of uncertainty to spatial data in a GIS context is reviewed by Worboys and Duckham (2004, pp. 328–358), and attribute error propagation is discussed by Heuvelink (1998). In an open computing environment like R, it is quite possible to think of ‘uncertain’ versions of the ‘crisp’ classes dealt with so far, in which, for example point position could be represented as a value drawn from a statistical model, allowing the impact of positional uncertainty on other methods of analysis to be assessed (see for example Leung et al., 2004).

5.2 Overlay

Accepting that moving from one spatial representation to another is a typical operation performed with spatial data, `overlay` methods are provided for a number of pairs of spatial data object types. Overlay methods involve combining congruent or non-congruent spatial data objects, and only some are provided directly, chiefly for non-congruent objects. Overlay operations are mentioned by Burrough and McDonnell (1998, pp. 52–53) and covered in much more detail by O'Sullivan and Unwin (2003, pp. 284–314) and Unwin (1996), who show how many of the apparently deterministic inferential problems in overlay are actually statistical in nature, as noted earlier. The basic approach is to query one spatial data object using a second spatial data object of the same or of a different class. The query result will typically be another spatial data object or an object pointing to one of the input objects. Overlaying a `SpatialPoints` object with a `SpatialPolygons` object returns a vector of numbers showing which `Polygons` object each coordinate in the `SpatialPoints` object falls within – this is an extension of the point-in-polygon problem to multiple points and polygons.

To continue the SRTM elevation data example, we can query `auck_el1`, which is a `SpatialGridDataFrame` object, using the transect shown in Fig. 2.7, stored as `SpatialPoints` object `transect_sp`. Using an overlay method, we obtain the elevation values retrieved from querying the grid cells as a `SpatialPointsDataFrame` object.

```
> summary(transect_sp)

Object of class SpatialPoints
Coordinates:
                min       max
coords.x1 174.45800 175.29967
coords.x2 -37.03625 -37.03625
Is projected: FALSE
proj4string : [+proj=longlat +ellps=WGS84]
Number of points: 1011

> transect_el1 <- overlay(auck_el1, transect_sp)
> summary(transect_el1)

Object of class SpatialPointsDataFrame
Coordinates:
                min       max
coords.x1 174.45800 175.29967
coords.x2 -37.03625 -37.03625
Is projected: FALSE
proj4string :
[+proj=longlat +ellps=WGS84 +datum=WGS84 +no_defs]
Number of points: 1011
```

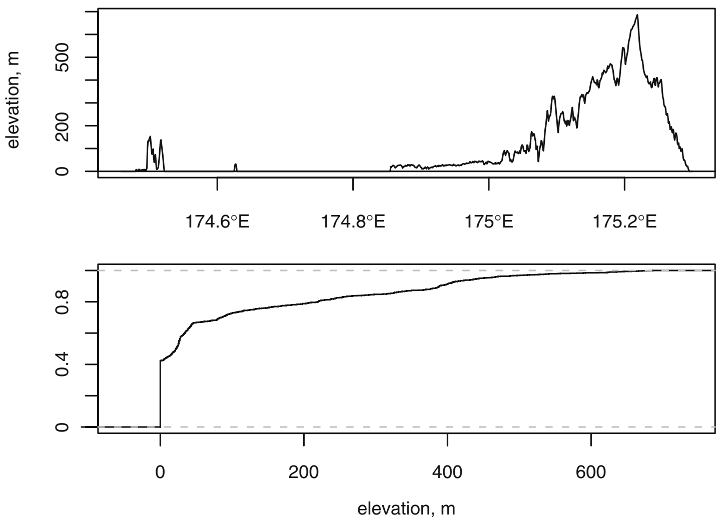

Fig. 5.2. Elevation values along a west–east transect, and a plot of the empirical cumulative distribution function values for elevation on the transect

```
Data attributes:
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
      0       0      22     101     128     686
```

Figure 5.2 shows two views of the resulting data object, first a cross-section of elevation along the transect and below that a plot of the empirical cumulative distribution function for the transect data. This is effectively the same as the diagram termed the hypsometric curve by geomorphologists, the cumulative height frequency curve, with the axes swapped.

Spatial queries of this kind are very common, reading off raster data for sample points of known events or phenomena of interest. Following modelling, the same data are then used to predict the value of the phenomenon of interest for unobserved raster cells. The study reported by Wang and Unwin (1992) on landslide distribution on loess soils is an example, involving the spatial querying of slope and aspect raster layers calculated from a digital elevation model, and lithology classes. As Unwin (1996, pp. 132–136) points out, there are two key issues in the analysis. First, there is considerable error present in the input raster data, and that the only field data collected are for sites at which landslides are known to have taken place. This means that some landslides may not have been included. Second, no matching data are available for other locations at which no landslides have been observed. This context of no control being available on the phenomena of interest is quite common in applied environmental research.

5.3 Spatial Sampling

One way of trying to get control over data in a research setting like the one described might be to sample points from the total study area, to be able to examine whether the observed phenomena seem to be associated with particular ranges of values of the supposed environmental 'drivers'. Sample design is not typically paid much attention in applied spatial data analysis, very often for practical reasons, for example the need to handle incoming data as they flow in, rather than being able to choose which data to use. In the case of veterinary epidemiology, it is not easy to impose clear control because of the time pressure to offer policy advice. Schemes for spatial sampling have been given in the literature, for example by Ripley (1981, pp. 19–27), and they are available in **sp** using generic method `spsample`. Five sampling schemes are available: `"random"`, which places the points at random within the sampling area; `"regular"`, termed a *centric systematic sample* by Ripley and for which the grid offset can be set, and `"stratified"` and `"nonaligned"`, which are implemented as variations on the `"regular"` scheme – `"stratified"` samples one point at random in each cell, and `"nonaligned"` is a systematic masked scheme using combinations of random x and y to yield a single coordinate in each cell. The fifth scheme samples on a hexagonal lattice. The spatial data object passed to the `spsample` method can be simply a `Spatial` object, in which case sampling is carried out within its bounding box. It can be a line object, when samples are taken along the line or lines. More typically, it is a polygon object or a grid object, providing an observation window defining the study area or areas.

Above, we examined SRTM elevation values along a transect crossing the highest point in the region. We can get an impression of which parts of the distribution of elevation values differ from those of the region as a whole by sampling. In the first case, we sample within the GSHHS shoreline polygons shown in Fig. 2.7, using the `"random"` sampling scheme; this scheme drops points within lakes in polygons. The `spsample` methods may return `SpatialPoints` objects with a different number of points than requested. The second and third samples are taken from the `SpatialPixelsDataFrame` object omitting the `NA` values offshore. They differ in using `"random"` and `"regular"` sampling schemes, respectively. The selected points are shown in Fig. 5.3.

```
> set.seed(9876)
> polygon_random <- spsample(auck_gshhs, 1000, type = "random")
> polygon_random_el1 <- overlay(auck_el1, polygon_random)
> grid_random <- spsample(auck_el2, 1000, type = "random")
> grid_random_el1 <- overlay(auck_el1, grid_random)
> grid_regular <- spsample(auck_el2, 1000, type = "regular")
> grid_regular_el1 <- overlay(auck_el1, grid_regular)
```

```
               minimum lower-hinge median upper-hinge maximum    n
transect             0           0     22       128.0     686 1011
```

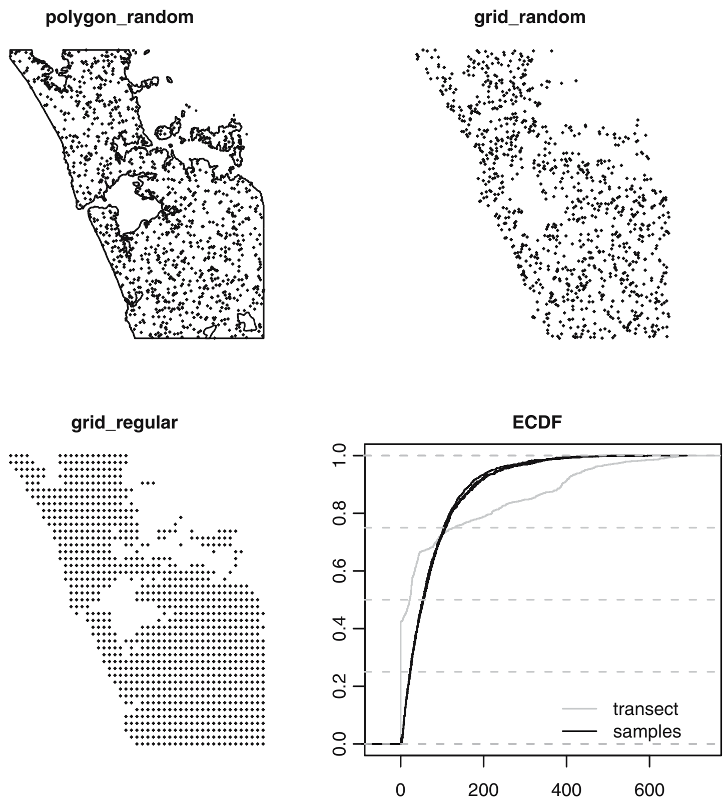

Fig. 5.3. Use of the `spsample` method: three sets of `SpatialPoints` objects and empirical cumulative distribution functions for elevation values for the sample points over-plotted on the transect values shown in Fig. 5.2

```
polygon_random     0         24     56        103.0      488 1000
grid_random        1         22     54        108.0      595  985
grid_regular       2         23     53        108.5      507 1020
```

Once again, we overlay our `SpatialPoints` objects created by sampling on the elevation `SpatialGridDataFrame` object. In this way we read off the elevation values for our sampled points, and can tabulate their five number summaries with that of the transect through the highest point. Both from the output and from the over-plotted empirical cumulative distribution function values shown in Fig. 5.3, we can see that the transect begins to diverge from the region as a whole above the 40th percentile. There is little difference between the three samples.

Alternative sampling schemes are contained in the **spatstat** package, with many point process generating functions for various window objects. The **spsurvey** package, using **sp** classes amongst other data representations, supplements these with some general procedures and the US Environmental Protection Agency Aquatic Resources Monitoring Generalized Random Tessellation Stratified procedure.[2]

5.4 Checking Topologies

In this section and the next, we look at a practical example involving the cleaning of spatial objects originally read into R from shapefiles published by the US Census. We then aggregate them up to metropolitan areas using a text table also from the US Census.

The data in this case are for polygons representing county boundaries in 1990 of North Carolina, South Carolina, and Virginia, as shown in Fig. 5.4. The attribute data for each polygon are the standard polygon identifiers, state and county identifiers, and county names. All the spatial objects have the same number of columns of attribute data of the same types and with the same names. The files are provided without coordinate reference systems as shapefiles; the metadata are used for choosing the `CRS` values.

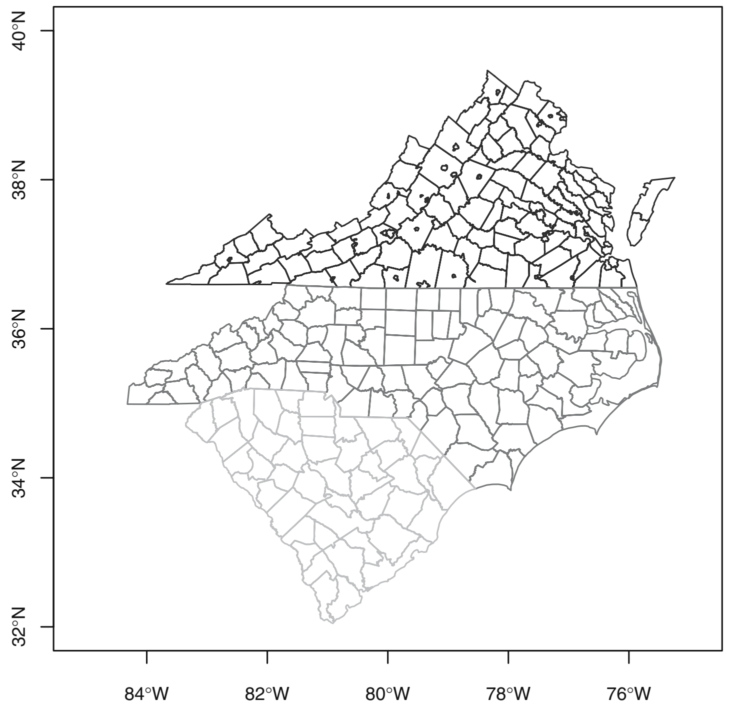

Fig. 5.4. The three states plotted from input spatial objects using different grey colours for county boundaries

[2] `http://www.epa.gov/nheerl/arm/index.htm`.

```
> library(rgdal)
> nc90 <- readOGR(".", "co37_d90")
> proj4string(nc90) <- CRS("+proj=longlat +datum=NAD27")
> sc90 <- readOGR(".", "co45_d90")
> proj4string(sc90) <- CRS("+proj=longlat +datum=NAD27")
> va90 <- readOGR(".", "co51_d90")
> proj4string(va90) <- CRS("+proj=longlat +datum=NAD27")
```

As read in, shapefiles usually have the polygon IDs set to the external file feature sequence number from zero to one less than the number of features. In our case, wanting to combine three states, we need to change the ID values so that they are unique across the study area. We can use the FIPS code (Federal Information Processing Standards Publication 6-4), which is simply the two-digit state FIPS code placed in front of the three-digit within-state FIPS county code, ending up with a five-digit string uniquely identifying each county. We can also drop the first four attribute data columns, two of which (area and perimeter) are misleading for objects in geographical coordinates, and the other two are internal ID values from the software used to generate the shapefiles, replicating the original feature IDs. We can start with the data set of South Carolina (`sc90`):

```
> library(maptools)

> names(sc90)

[1] "AREA"       "PERIMETER"  "CO45_D90_"  "CO45_D90_I" "ST"
[6] "CO"         "NAME"

> sc90a <- spChFIDs(sc90, paste(sc90$ST, sc90$CO, sep = ""))
> sc90a <- sc90a[, -(1:4)]
> names(sc90a)

[1] "ST"   "CO"   "NAME"
```

5.4.1 Dissolving Polygons

When we try the same sequence of commands for North Carolina, we run into difficulties:

```
> names(nc90)

[1] "AREA"       "PERIMETER"  "CO37_D90_"  "CO37_D90_I" "ST"
[6] "CO"         "NAME"

> nc90a <- spChFIDs(nc90, paste(nc90$ST, nc90$CO, sep = ""))

Error in spChFIDs(SP, x) : duplicate IDs
```

Tabulating the frequencies of polygons per unique county ID, we can see that 98 of North Carolina's counties are represented by single polygons, while one has two polygons, and one (on the coast) has four.

```
> table(table(paste(nc90$ST, nc90$CO, sep = "")))

 1  2  4
98  1  1
```

One reason for spatial data being structured in this way is that it is following the OpenGIS®[3] Simple Features Specification, which allows polygons to have one and only one external boundary ring, and an unlimited number of internal boundaries – holes. This means that multiple external boundaries – such as a county made up of several islands – are represented as multiple polygons. In the specification, they are linked to attribute data through a look-up table pointing to the appropriate attribute data row.

We need to restructure the `SpatialPolygons` object such that the `Polygon` objects belonging to each county belong to the same `Polygons` object. To do this, we use a function[4] in the **maptools** package also used for dissolving or merging polygons, but which can be used here to re-package the original features, so that each `Polygons` object corresponds to one and only one county:

```
> nc90a <- unionSpatialPolygons(nc90, IDs = paste(nc90$ST,
+     nc90$CO, sep = ""))
```

The function uses the IDs argument to set the ID slots of the output `SpatialPolygons` object. Having sorted out the polygons, we need to remove the duplicate rows from the data frame and put the pieces back together again:

```
> nc90_df <- as(nc90, "data.frame")[!duplicated(nc90$CO),
+     -(1:4)]
> row.names(nc90_df) <- paste(nc90_df$ST, nc90_df$CO, sep = "")
> nc90b <- SpatialPolygonsDataFrame(nc90a, nc90_df)
```

5.4.2 Checking Hole Status

Looking again at Fig. 5.4, we can see that while neither North Carolina nor South Carolina has included boroughs within counties, these are frequently found in Virginia. While data read from external sources are expected to be structured correctly, with the including polygon having an outer edge and an inner hole, into which the outer edge of the included borough fits, as described in Sect. 2.6.2, we can also check and correct the settings of the hole slot in `Polygon` objects. The `checkPolygonsHoles` function takes a `Polygons` object as its argument, and, if multiple `Polygon` objects belong to it, checks them for hole status using functions from the **gpclib** package:

```
> va90a <- spChFIDs(va90, paste(va90$ST, va90$CO, sep = ""))
> va90a <- va90a[, -(1:4)]
```

[3] See `http://www.opengeospatial.org/`.
[4] This function requires that the **gpclib** package is also installed.

```
> va90_pl <- slot(va90a, "polygons")
> va90_pla <- lapply(va90_pl, checkPolygonsHoles)
> p4sva <- CRS(proj4string(va90a))
> vaSP <- SpatialPolygons(va90_pla, proj4string = p4sva)
> va90b <- SpatialPolygonsDataFrame(vaSP, data = as(va90a,
+     "data.frame"))
```

Here we have changed the `Polygons` ID values as before, and then processed each `Polygons` object in turn for internal consistency, finally re-assembling the cleaned object. So we now have three spatial objects with mutually unique IDs, and with data slots containing data frames with the same numbers and kinds of columns with the same names.

5.5 Combining Spatial Data

It is quite often desirable to combine spatial data of the same kind, in addition to combining positional data of different kinds as discussed earlier in this chapter. There are functions `rbind` and `cbind` in R for combining objects by rows or columns, and `rbind` methods for `SpatialPixels` and `SpatialPixels-DataFrame` objects, as well as a `cbind` method for `SpatialGridDataFrame` objects are included in **sp**. In addition, methods with slightly different names to carry out similar operations are included in the **maptools** package.

5.5.1 Combining Positional Data

The `spRbind` method combines positional data, such as two `SpatialPoints` objects or two `SpatialPointsDataFrame` objects with matching column names and types in their data slots. The method is also implemented for `SpatialLines` and `SpatialPolygons` objects and their `*DataFrame` extensions. The methods do not check for duplication or overlapping of the spatial objects being combined, but do reject attempts to combine objects that would have resulted in non-unique IDs.

Because the methods only take two arguments, combining more than two involves repeating calls to the method:

```
> nc_sc_va90 <- spRbind(spRbind(nc90b, sc90a), va90b)
> FIPS <- sapply(slot(nc_sc_va90, "polygons"), function(x) slot(x,
+     "ID"))
> str(FIPS)

chr [1:282] "37001" "37003" ...

> length(slot(nc_sc_va90, "polygons"))

[1] 282
```

5.5.2 Combining Attribute Data

Here, as very often found in practice, we need to combine data for the same spatial objects from different sources, where one data source includes the geometries and an identifying index variable, and other data sources include the same index variable with additional variables. They often include more observations than our geometries, sometimes have no data for some of our geometries, and not are infrequently sorted in a different order. The data cleaning involved in getting ready for analysis is a little more tedious with spatial data, as we see, but does not differ in principle from steps taken with non-spatial data.

The text file provided by the US Census tabulating which counties belonged to each metropolitan area in 1990 has a header, which has already been omitted, a footer with formatting information, and many blank columns. We remove the footer and the blank columns first, and go on to remove rows with no data – the metropolitan areas are separated in the file by empty lines. The required rows and column numbers were found by inspecting the file before reading it into R:

```
> t1 <- read.fwf("90mfips.txt", skip = 21, widths = c(4,
+     4, 4, 4, 2, 6, 2, 3, 3, 1, 7, 5, 3, 51), colClasses = "character")
```

```
> t2 <- t1[1:2004, c(1, 7, 8, 14)]
> t3 <- t2[complete.cases(t2), ]
> cnty1 <- t3[t3$V7 != "  ", ]
> ma1 <- t3[t3$V7 == "  ", c(1, 4)]
> cnty2 <- cnty1[which(!is.na(match(cnty1$V7, c("37", "45",
+     "51")))), ]
> cnty2$FIPS <- paste(cnty2$V7, cnty2$V8, sep = "")
```

We next break out an object with metro IDs, state and county IDs, and county names (`cnty1`), and an object with metro IDs and metro names (`ma1`). From there, we subset the counties to the three states, and add the FIPS string for each county, to make it possible to combine the new data concerning metro area membership to our combined county map. We create an object (`MA_FIPS`) of county metro IDs by matching the `cnty2` FIPS IDs with those of the counties on the map, and then retrieving the metro area names from `ma1`. These two variables are then made into a data frame, the appropriate row names inserted and combined with the county map, with method `spCbind`. At last we are ready to dissolve the counties belonging to metro areas and to discard those not belonging to metro areas, using `unionSpatialPolygons`:

```
> MA_FIPS <- cnty2$V1[match(FIPS, cnty2$FIPS)]
> MA <- ma1$V14[match(MA_FIPS, ma1$V1)]
> MA_df <- data.frame(MA_FIPS = MA_FIPS, MA = MA, row.names = FIPS)
> nc_sc_va90a <- spCbind(nc_sc_va90, MA_df)
> ncscva_MA <- unionSpatialPolygons(nc_sc_va90a, nc_sc_va90a$MA_FIPS)
```

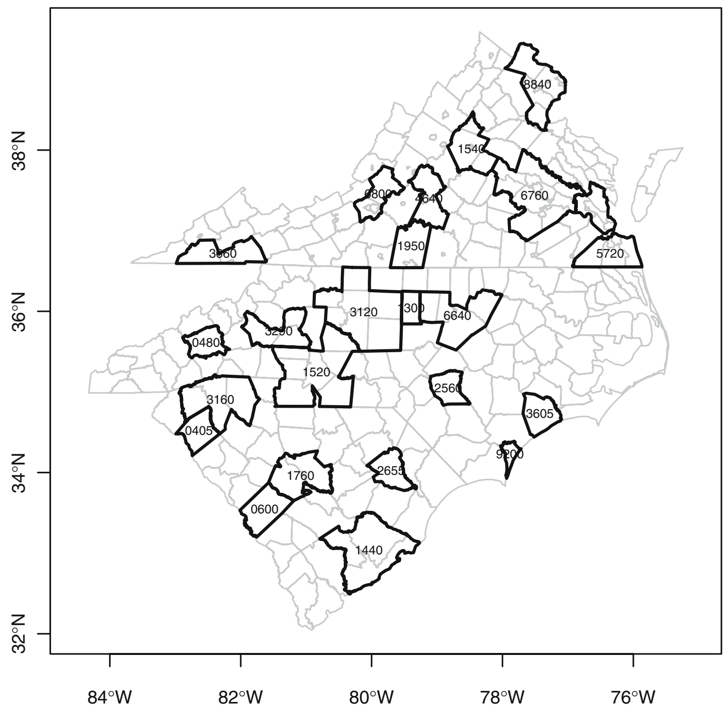

Fig. 5.5. The three states with county boundaries plotted in grey, and Metropolitan area boundaries plotted in black; Metro area standard IDs are shown

Figure 5.5 shows the output object plotted on top of the cleaned input county boundaries. There does appear to be a problem, however, because one of the output boundaries has no name – it is located between 6760 and 5720 in eastern Virginia. If we do some more matching, to extract the names of the metropolitan areas, we can display the name of the area with multiple polygons:

```
> np <- sapply(slot(ncscva_MA, "polygons"), function(x) length(slot(x,
+     "Polygons")))
> table(np)

np
 1  2
22  1

> MA_fips <- sapply(slot(ncscva_MA, "polygons"), function(x) slot(x,
+     "ID"))
> MA_name <- ma1$V14[match(MA_fips, ma1$V1)]
> data.frame(MA_fips, MA_name)[np > 1, ]

   MA_fips                                           MA_name
18    5720  Norfolk-Virginia Beach-Newport News, VA MSA
```

The Norfolk-Virginia Beach-Newport News, VA MSA is located on both sides of Hampton Roads, and the label has been positioned at the centre point of the largest member polygon.

5.6 Auxiliary Functions

New functions and methods are added to **maptools** quite frequently, often following suggestions and discussions on the R-sig-geo mailing list mentioned in Chap. 1. When positions are represented by geographical coordinates, it is often useful to be able to find the azimuth between them. The `gzAzimuth` function is a simple implementation of standard algorithms for this purpose, and gives azimuths calculated on the sphere between a matrix of points and a single point.[5] The `gcDestination` function returns the geographical coordinates of points at a given distance and bearing from given starting points.

A set of methods for matrices or `SpatialPoints` objects in geographical coordinates has been contributed to give timings for sunrise, sunset, and other solar measures for dates given as `POSIXct` objects:

```
> hels <- matrix(c(24.97, 60.17), nrow = 1)
> p4s <- CRS("+proj=longlat +datum=WGS84")
> Hels <- SpatialPoints(hels, proj4string = p4s)
> d041224 <- as.POSIXct("2004-12-24", tz = "EET")
> sunriset(Hels, d041224, direction = "sunrise", POSIXct.out = TRUE)

   day_frac                time
1 0.3924249 2004-12-24 09:25:05
```

Finally, `elide` methods have been provided for translating, rotating, and disguising coordinate positions in **sp** vector classes such as `SpatialPoints`. The geometries can be shifted in two dimensions, scaled such that the longest dimension is scaled $[0, 1]$, flipped, reflected, and rotated, if desired in relation to the bounding box of a different `Spatial` object. The methods can be used for standardising displays, for example in point pattern analysis, or for obfuscating position to meet in part privacy considerations. Since obscuring position was a reason for providing the methods, they have been given a suitably obscure name.

The methods discussed in this chapter are intended to provide ways for manipulating spatial objects to help in structuring analytical approaches to support problems amongst others. These are not the only ways to organise spatial data, do try to make it easier to concentrate on exploring and analysing the data, rather than dealing with the intricacies of particular representations peculiar to specific software or data providers.

[5] The function is based with permission on work by S. Abdali: The Correct Qibla, `http://patriot.net/users/abdali/ftp/qibla.pdf`.