R as a GIS: overlay/aggregate, rgeos, osmar, networks

Edzer Pebesma

ifgi
Institute for Geoinformatics
University of Münster

edzer.pebesma@uni-muenster.de

Sept 5, 2012

Organisation

9:00–10:30 Getting your data in R: rgdal, osmar

11:00-12:30 Vector GIS: rgeos

13:30-15:00 Raster GIS: raster

15:30-17:00 Challenges: trajectories, routing

inbetween: breaks; blocks: 45 mins lecture, 45 mins discussion / exercises.

Poll: your favourite GIS software?

Your favorite GIS software (for programming spatial data analysis)?

GRASS GIS	14%
SAGA GIS	3%
QGIS	2%
ArcGIS	19%
R	34%
Combination of those	25%
None of the above	3%
Total votes:	59

(`http://geostat-course.org/`, Aug 28, 2012)

R as a GIS: overlay/aggregate, rgeos, osmar, networks

Why this seminar?

- I believe R can be seen as a GIS (and hence, as sth beyond a GIS), especially since
  - spatial packages today largely adopt `sp` classes
  - `rgdal` does vector and raster I/O
  - `rgeos` does geometry operations
  - `raster` does (big data) raster analysis
  - these packages are all pretty robust
  - several interfaces exist to open data (e.g. open street maps: `osmar`)
- R might be superior (to what?) in doing overlay and aggregation,
- networks, and network analysis (spatial and non-spatial) are a challenging topic, and can use more exposure.

R

- what is R?
- what is an R package?
- what is CRAN?
- where is the spatial task view?
- how do I find in package *x* how to do task *y*?
- how do I find out how to do ... with R?

- what is R?
- what is an R package?
- what is CRAN?
- where is the spatial task view?
- how do I find in package *x* how to do task *y*?
- how do I find out how to do ... with R?

```
> library(fortunes)
> fortune("only how")

Evelyn Hall: I would like to know how (if) I can extract some
of the information from the summary of my nlme.

Simon Blomberg: This is R. There is no if. Only how.
   -- Evelyn Hall and Simon 'Yoda' Blomberg
      R-help (April 2005)
```

I will assume you understand this:

```
> a = data.frame(varA = c(1,1.5,2),
+  varB = c("a", "a", "b"))
> a[1,]

  varA varB
1    1    a

> a[1, drop=FALSE]

  varA
1  1.0
2  1.5
3  2.0

> a[,1]

[1] 1.0 1.5 2.0

> a[1]

  varA
1  1.0
2  1.5
3  2.0

> a[[1]]

[1] 1.0 1.5 2.0
```

```
> a["varA"]

  varA
1  1.0
2  1.5
3  2.0

> a[c("varA", "varB")]

  varA varB
1  1.0    a
2  1.5    a
3  2.0    b

> a$varA

[1] 1.0 1.5 2.0

> a$varA <- 3:1
> a

  varA varB
1    3    a
2    2    a
3    1    b
```

Spatial data

Spatial data refresher:

- points, lines, polygons, grids
- storage: shapefiles, grid files, in- or out-of-memory
- data bases (e.g. PostGIS): geometry + attributes
- topology representation of polygons
- spatial indexes
- projected data, or long/lat?

What makes a GIS a GIS?

What makes a GIS a GIS?

- store, retrieve spatial data
- visualize spatial data
- manipulate spatial data
- analyze, model spatial data
  - analyze attributes, as in a data base
  - analyze geometries, or attributes depending on geometry

"A geographic information system is a system designed to capture, store, manipulate, analyze, manage, and present all types of geographical data" (wikipedia, from esri.com)
"In the simplest terms, GIS is the merging of cartography, statistical analysis, and database technology." (wikipedia)

Where to get our data?

- data that come with R: see Spatial Task View
- R packages: maps, mapdata, maptools (GSHHS shoreline files),
- Landsat, MODIS, SRTM, corine, ...
- gadm.org: .RData files!
- read, search `r-sig-geo`
- NUTS: eurostat
- search, talk, ask, ...

How to get our data into R?

Simple answer: using `rgdal` (`readGDAL` or `readOGR`).

How to get our data into R?

Simple answer: using `rgdal` (`readGDAL` or `readOGR`). More complete:

- `readGDAL` or `readOGR` read the whole data set from disk into R, that is, into the computers main or working memory ("RAM").
- for grids, there are low-level routines: `GDAL.open` opens a file, and `getRasterData` (or `getRasterTable`) to read *portions* of data; but also (little known!):

```
> library(rgdal)
> x = GDAL.open("NDV_19980401_Gambia__the_Extract.tif")
> class(x)
```

```
[1] "GDALReadOnlyDataset"
attr(,"package")
[1] "rgdal"
```

```
> image(x[1:100, 1:100])
> class(x[1:100, 1:100])
```

```
[1] "SpatialGridDataFrame"
attr(,"package")
[1] "sp"
```

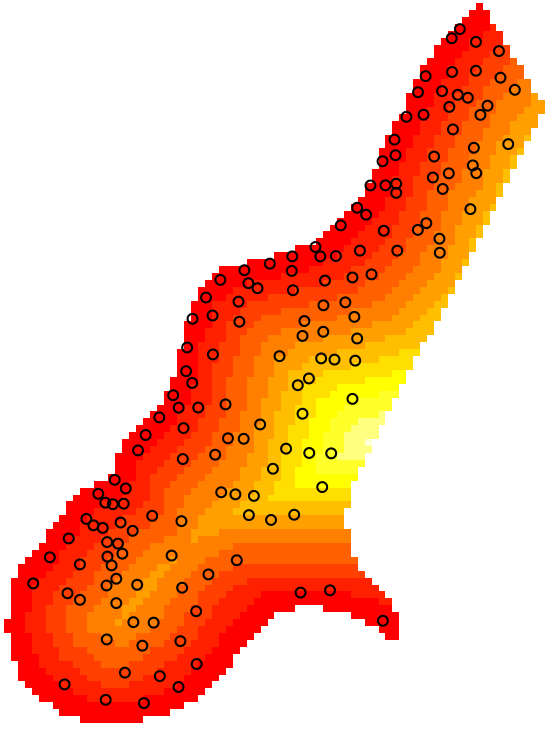
reads only the portion requested into memory

overlay: visual

base plot: plotting sequentially, e.g.
lattice (spplot):
note: transparency is a colour attribute

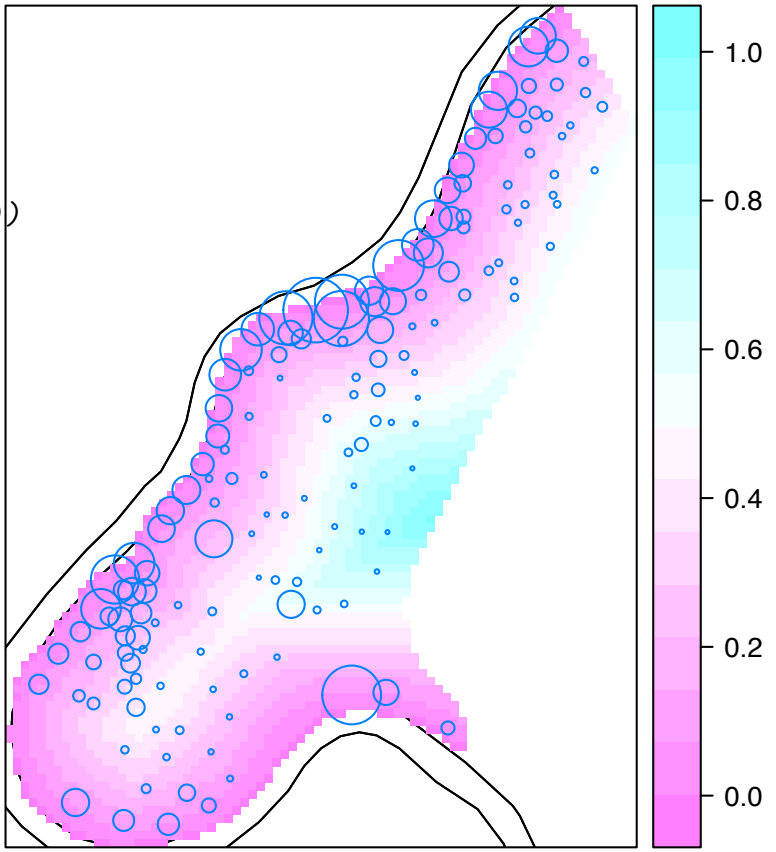
Overlay: visual - 1. by incrementally plotting

```
> library(sp)
> loadMeuse()
> image(meuse.grid["dist"])
> points(meuse)
> # add lines, legend, text, ...
```



Overlay: visual - 2. using compound plot functions

```
> library(sp)
> loadMeuse(river = TRUE)
> size = meuse$zinc / mean(meuse$zinc)
> pts = list("sp.points", meuse, pch = 1, cex = size)
> riv = list("sp.polygons", meuse.riv)
> plt = spplot(meuse.grid["dist"], sp.layout = list(pts, riv))
> class(plt)

[1] "trellis"

> print(plt)
```



- a plot object is created, which contains everything
- this object can be manipulated, but the most used option is to print (i.e., show) it.

Cartography?

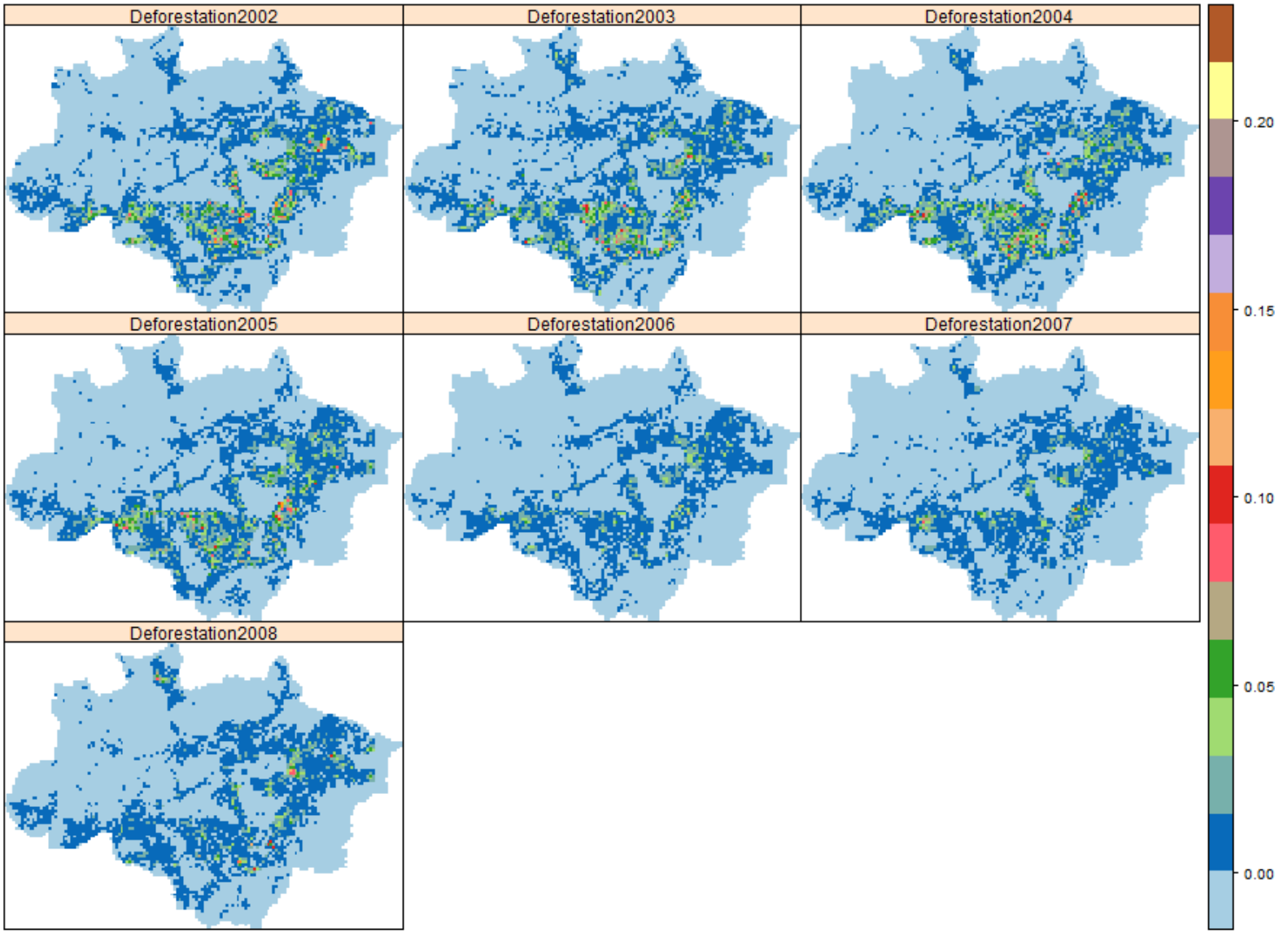
- A map is a plot with longitude and latitude, and a controlled aspect ratio; any plotting software can "do" maps, however
- reference comes from coast lines, rivers, lakes, topography, political boundaries, cities, land use etc.
- reference grid lines (parallels, meridians) may be required, and be non-straigh
- axes tics usually show little, but some information
- custom elements are often present (arrow, scale bar, multi-type legend)
- label placement is challenging (but see: rgeos::polyLabel)

What is R good at?

- simple, repetitive graphs:
  - many, similar graphs, over different pages
  - many graphs combined in a lattice (grid: lattice, ggplot)
- non-interactive, reproducible use
- control of all details
- richness of graphics devices,
- portability, cross-platform, options for deployment

What is R bad at?

- interactive use: zoom, pan, edit graph element etc.
- control is not trivial
- incompatible plotting systems: base, lattice, ggplot, ...



G. Espindola, 2012: Spatiotemporal trends of land use change in the Brazilian Amazon; INPE

Two work horses: rgdal, rgeos

```
> library(rgdal)
Geospatial Data Abstraction Library extensions to R successfully loaded
Loaded GDAL runtime: GDAL 1.9.1, released 2012/05/15
Path to GDAL shared files: /usr/share/gdal/1.9
Loaded PROJ.4 runtime: Rel. 4.7.1, 23 September 2009, [PJ_VERSION: 470]
Path to PROJ.4 shared files: (autodetected)
>
> library(rgeos)
Loading required package: stringr
Loading required package: plyr
rgeos: (SVN revision (unknown))
GEOS runtime version: 3.3.3-CAPI-1.7.4
Polygon checking: TRUE
```

- `rgdal` links to the GDAL (raster) and OGR (vector) data I/O library, as well as `PROJ.4` for CRS (coordinate reference systems) (re)projections
- `rgeos` links to the GEOS (Geometry Open Source) library, which powers PostGIS: does the "usual" geometry operations for features

What is numerical overlay?

Method `over(x,y)` provides: consistent spatial overlay for points, grids, lines and polygons: at the spatial locations of object x retrieve the indexes or attributes from spatial object y and `NA` in case of no match (index `vector` if y has only geometry, attribute `data.frame` if it has attributes too).

```
> library(sp)
> loadMeuse()
> over(meuse, geometry(meuse.grid))[1:10]

 [1]   9  24  28  41  93 128  75  71
 [9] 138 161

> over(meuse, meuse.grid)[1:3,]

  part.a part.b       dist soil
1      1      0 0.00135803    1
2      1      0 0.01222430    1
3      1      0 0.10302900    1
  ffreq
1     1
2     1
3     1
```

In SQL, this resembles a left outer join of two tables

What if there are no, or multiple matches?

No match:

```
> library(sp)
> loadMeuse()
> m2 = meuse[-1,] # remove first record
> over(meuse, geometry(m2))[1:5]

[1] NA  1  2  3  4

> over(meuse, m2)[1:3, 1:4]

  cadmium copper lead zinc
1      NA     NA   NA   NA
2     8.6     81  277 1141
3     6.5     68  199  640
```

Multiple matches:

```
> library(sp)
> loadMeuse()
> m2 = meuse[c(1,1:155),] # duplicate first record
> over(meuse, geometry(m2))[1:5]

[1] 2 3 4 5 6

> over(meuse, m2)[1:3, 1:4]

  cadmium copper lead zinc
1    11.7     85  299 1022
2     8.6     81  277 1141
3     6.5     68  199  640
```

So, by default, all multiple maches are ignored.

What if we want multiple matches?

```
> library(sp)
> loadMeuse()
> m2 = meuse[c(1,1:155),] # duplicate first record
> over(meuse, geometry(m2), returnList = TRUE)[1:3]

[[1]]
[1] 1 2

[[2]]
[1] 3

[[3]]
[1] 4

> over(meuse, m2[1:4], returnList = TRUE)[1:2]

[[1]]
    cadmium copper lead zinc
1      11.7     85  299 1022
1.1    11.7     85  299 1022

[[2]]
  cadmium copper lead zinc
2     8.6     81  277 1141
```

What if we want to compute over multiple matches?

```
> library(sp)
> loadMeuse()
> m2 = meuse[c(1,1:155),] # duplicate first record
> over(meuse, m2[1:4], returnList = FALSE, fn = max)[1:2,1:4]

  cadmium copper lead zinc
1    11.7     85  299 1022
2     8.6     81  277 1141
```

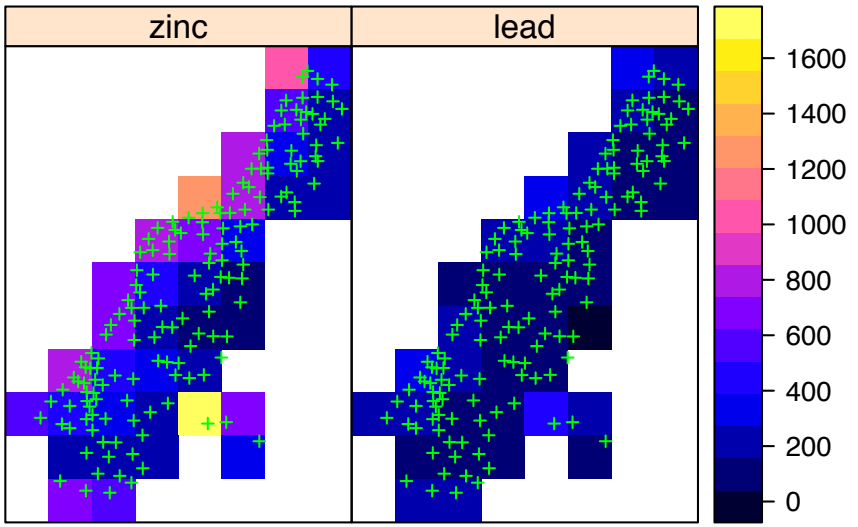
Although this is the same as

```
> over(meuse, meuse)[1:2,1:4]

  cadmium copper lead zinc
1    11.7     85  299 1022
2     8.6     81  277 1141
```

in the first case, actually the maximum is computed (`fn = max`) over the multiple matched records, and returned, as record values. In fact, this process is called *aggregation*.

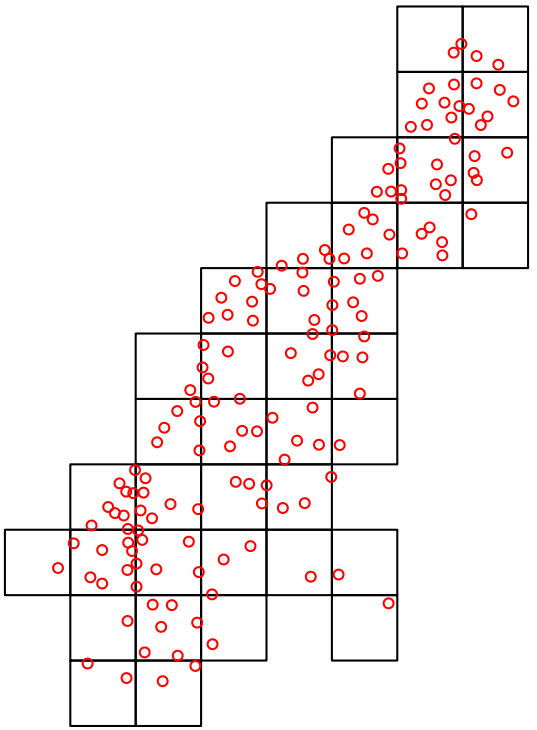
Aggregation, the R way

```
> library(sp)
> # for a data.frame, based on a table column:
> m = as(meuse, "data.frame")[c("zinc", "lead")]
> aggregate(m, by = list(ffreq = meuse$ffreq), mean)

  ffreq     zinc     lead
1     1 625.7500 197.9762
2     2 273.2083  99.3750
3     3 309.9565 103.0870

> # create a coarse grid:
> off = gridparameters(meuse.grid)$cellcentre.offset + 20
> gt = GridTopology(off, c(400,400), c(8,11))
> SG = SpatialGrid(gt)
> proj4string(SG) = proj4string(meuse.grid)
> # for a Spatial object, based on another Spatial object:
> agg = aggregate(meuse[c("zinc","lead")], SG, FUN = mean)
> spplot(agg, sp.layout = pts)
```



Which pixels are covered by points? Selection with `over`

```
> SP = as(SG, "SpatialPolygons")
> over(SP, geometry(meuse))

 [1]  NA  NA  NA  NA  NA  NA   1   3
 [9]  NA  NA  NA  NA  NA  NA   8   5
[17]  NA  NA  NA  NA  NA  19  12  24
[25]  NA  NA  NA  NA  52  37  32  31
[33]  NA  NA  NA  55  58  42  NA  NA
[41]  NA  NA  64  62  49 106  NA  NA
[49]  NA  NA  65  86 104 107  NA  NA
[57]  NA  72  70  68 103  NA  NA  NA
[65]  92  77  76  94  82 118  NA  NA
[73]  NA 150  95 143  NA 155  NA  NA
[81]  NA 147 144  NA  NA  NA  NA  NA

> length(SP[!is.na(over(SP, geometry(meuse)))])

[1] 38

> length(SP[meuse]) # equivalent!

[1] 38

> plot(as(SP[meuse], "SpatialPolygons"))
> points(meuse, col = 'red')
```



Possible over methods

Spatial data (see vignette in sp):

	y: points	y: lines	y: polygons	y: pixels/grids
x: points	sp	rgeos	sp	sp
x: lines	rgeos	rgeos	rgeos	rgeos
x: polygons	sp	rgeos	rgeos	sp
x: pixels/grids	sp	rgeos	sp	sp

Table: `over` methods implemented for different `x` and `y` arguments.

Spatio-temporal data: see `spacetime` vignette

Things we cannot do simply, yet

- polygon-polygon aggregate with areal weights (e.g. dominant land use class, areal weighted yields; approximate solution would be: to rasterize, and do it cell / point-wise)
- treating grid cells as cells (in all cases), instead of points centered at mid of cell

Suggestions for next week's development workshop?

- add a `cellIsArea` logical flag to `GridTopology`, and make it effective in all `over` methods
- develop a polygon-polygon overlay routine, following Colin Rundel's initial sketch on `r-sig-geo`.

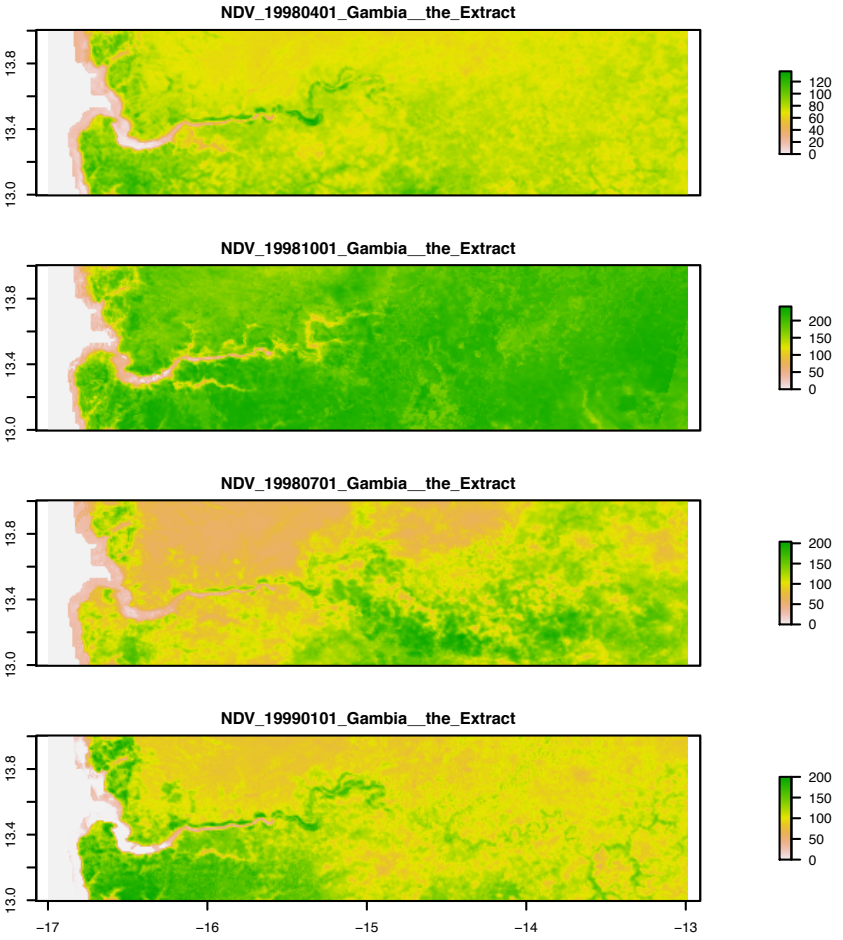
Nine-intersection model

The nine-intersection model is a comprehensive model, from which most relations (touches, overlaps, intersects) can be derived – see http://en.wikipedia.org/wiki/DE-9IM

```
> library(rgeos)
> gRelate(meuse.riv, meuse.riv)

[1] "2FFF1FFF2"

> gRelate(meuse.riv, SP)

[1] "212101212"

> gRelate(meuse.riv, SP, byid = TRUE)[1:5]

[1] "FF2FF1212" "FF2FF1212"
[3] "FF2FF1212" "FF2FF1212"
[5] "FF2FF1212"
```

Raster analysis with `raster`

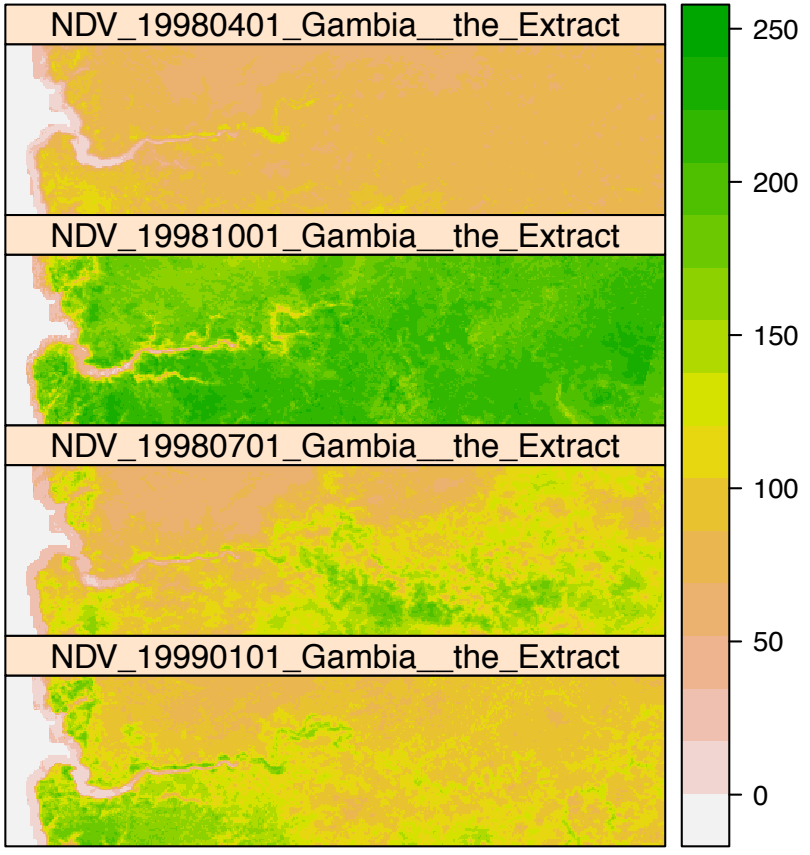
```
> nm = c("NDV_19980401_Gambia__the_Extract.tif",
+   "NDV_19981001_Gambia__the_Extract.tif",
+   "NDV_19980701_Gambia__the_Extract.tif",
+   "NDV_19990101_Gambia__the_Extract.tif")
> library(raster)
> r = stack(nm)
> plot(r, nc=1)
```



Raster – compatible with sp

```
> spplot(as(r, "SpatialGridDataFrame"),
+    col.regions=rev(terrain.colors(255)))
```

note that this plot will take values in memory, and plot *all* pixels!
(runtime, file size of e.g. pdf)



Raster – elementary idea

Package `raster` has a number of vignettes.
Rasters are single-layer, and can be made from scratch or files:

```
> r <- raster(nrows=10, ncols=10)
> r <- setValues(r, 1:ncell(r))
> # or
> r = raster("NDV_19990101_Gambia__the_Extract.tif")
```

groups of rasters can form a stack:

```
> nm

[1] "NDV_19980401_Gambia__the_Extract.tif"
[2] "NDV_19981001_Gambia__the_Extract.tif"
[3] "NDV_19980701_Gambia__the_Extract.tif"
[4] "NDV_19990101_Gambia__the_Extract.tif"

> st <- stack(nm)
```

We can do algebra on raster layers, e.g. cell-wise:

```
> r1 = raster("NDV_19990101_Gambia__the_Extract.tif")
> r2 = raster("NDV_19981001_Gambia__the_Extract.tif")
> r0 = (3 * r1 + 7 * r2)/10
```

Raster – higher level functions

- change extent: shift, expand, trim, merge, crop
- change resolution: aggregate, disaggregate, resample
- change CRS: projectRaster
- combine layers: overlay, mask, cover
- reclassify: reclass, cut
- apply user-defined function: calc
- filter-type, focal functions: focal, focalStack, focalNA
- distance, clump, area, edge
- interpolate, predict
- vector-to-raster: rasterize, rasterToPoints, rasterToPolygons
- stats, summary, plotting
- low-level access/manipulation functions

everything in-memory, or (as far as rasters are involved) on-disk!

Raster stacks, bricks

Rasters can be stacked, or bricked, by

```
> st = stack(r1, r2)
> br = brick(r1, r2)
```

The `RasterStack` and `RasterBrick` are largely similar, but as a file proxy / handle, bricks only work with *single*, multi-banded files (e.g., NetCDF, GeoTIFF etc).

Raster analysis and R

Raster solves the problem of large imagery that needs to be processed out of RAM, but still

- the images are files (per scene, date, theme, sensor)
- the images need to be on a local disk

This scales up to the extent that one can download the data needed.
For larger data (e.g. global coverage 20 years 1 km MODIS data), one would need another solution, e.g. a raster data base, preferably integrated with R.

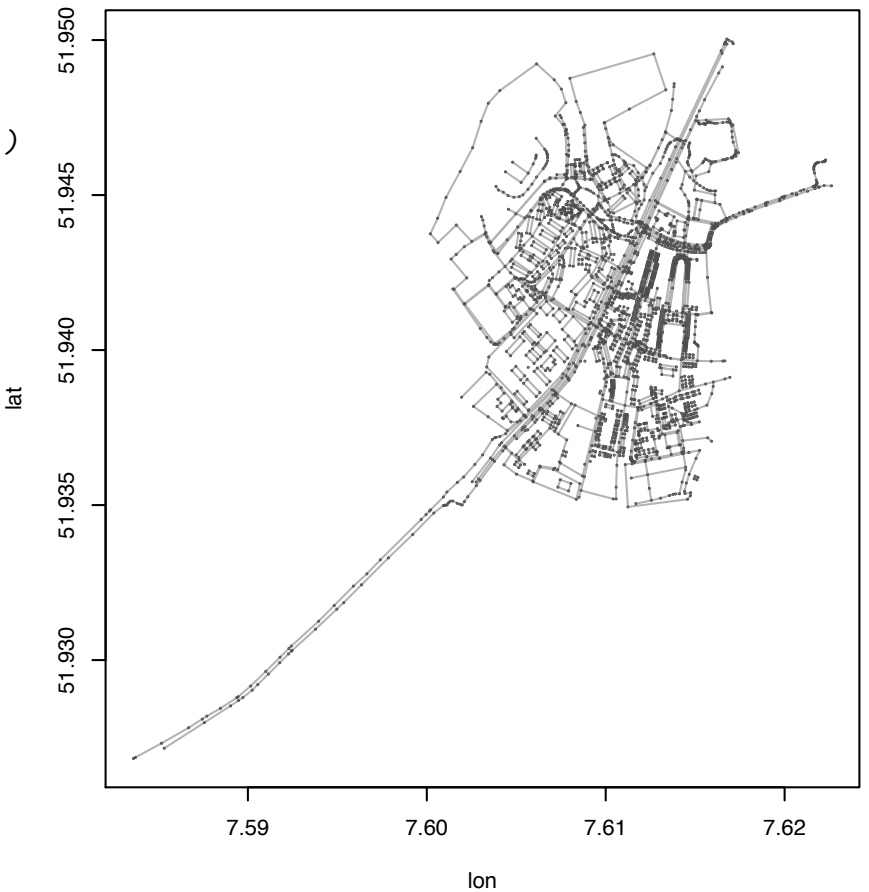
Open street maps: OpenStreetMap

```
> library(OpenStreetMap)
> m <- c(25.7738889,-80.1938889)
> j <- c(58.3019444,-134.4197222)
> map <- openmap(j,m,4)
> plot(map)
```

seems to retrieve bitmaps of OSM, as backdrop, to draw (other) statistical maps on top of

Open street maps: osmar

```
> library(osmar)
> api = osmsource_api()
> pt =  c(7.609983,51.940812) # ifgi
> size = 0.005
> bb = do.call(corner_bbox, as.list(c(pt - size, pt + size)))
> ifgi = get_osm(bb, source = api)
> plot(ifgi)
> dx = c(-1, 1, 1, -1, -1)
> dy = c(-1, -1, 1, 1, -1)
> lines(pt[1] + dx * size, pt[2] + dy * size, col = 'red')
> lapply(as_sp(ifgi), class)

$points
[1] "SpatialPointsDataFrame"
attr(,"package")
[1] "sp"

$lines
[1] "SpatialLinesDataFrame"
attr(,"package")
[1] "sp"

$polygons
[1] "SpatialPolygonsDataFrame"
attr(,"package")
[1] "sp"

> class(as_igraph(ifgi))

[1] "igraph"
```



seems to give access to all vector data, in sp or igraph0 format!

Networks, or graphs

Quoated from WikiPedia:
"A graph data structure consists of a finite (and possibly mutable) set of ordered pairs, called **edges** or **arcs**, of certain entities called **vertices** or **nodes**. As in mathematics, an `edge(x,y)` is said to point or go from x to y. The nodes may be part of the graph structure, or may be external entities represented by integer indices or references.
A graph data structure may also associate to each edge some edge value, such as a symbolic label or a numeric attribute (cost, capacity, length, etc.)."

What we still need

is a `SpatialNetwork`, which could be as simple as

```
> library(igraph0)
> setClass("SpatialNetwork",
+  contains = "SpatialLines",
+  representation(network = "igraph"),
+  validity = function(object) {
+  stopifnot(length(object@Lines) != length(E(network)))
+  # check that line segments indeed connect at nodes
+  TRUE
+  }
+ )
> SpatialNetwork = function(sl, nw) {
+    new("SpatialNetwork", sl, network = nw)
+ }
> # sn = SpatialNetwork(as_sp(ifgi)$lines, as_igraph(ifgi))
```

but clearly, this was thought too simple.