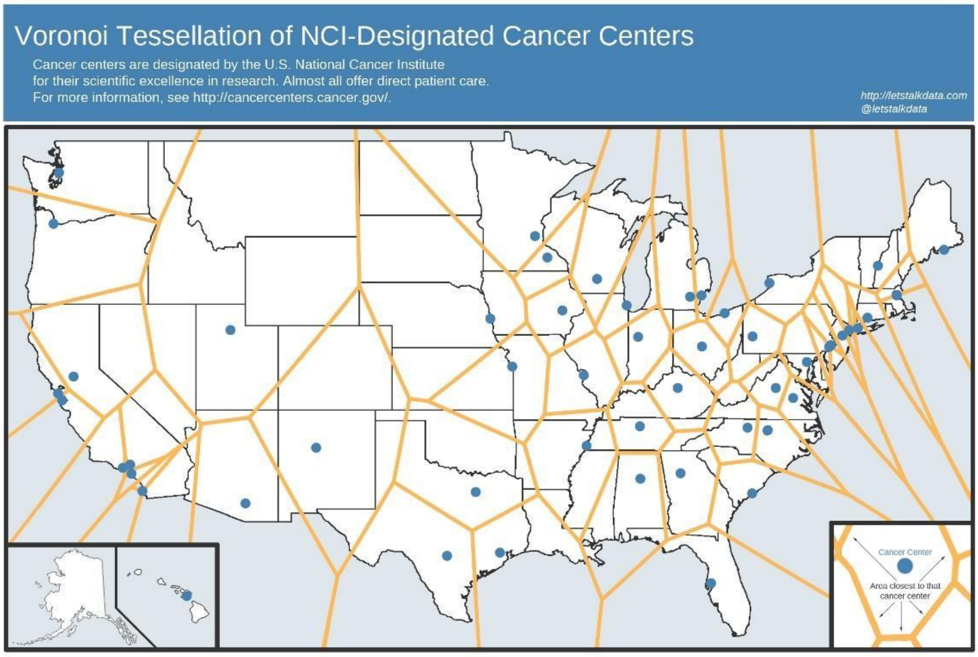
VORONOI DIAGRAMS

April 2016

Johan Blomme
Leenstraat 11 – 8340 Damme-Sijsele
URL : www.johanblomme.net
Email : j.blomme@telenet.be

A Voronoi diagram is a partitioning of a plane into regions based on distance to points in a specific subset of the plane. That set of points (called seeds, sites, or generators) is specified beforehand, and for each seed there is a corresponding region consisting of all points closer to that seed than to any other. These regions are called Voronoi cells.

Example :


Voronoi Tessellation of NCI-Designated Cancer Centers
Cancer centers are designated by the U.S. National Cancer Institute
for their scientific excellence in research. Almost all offer direct patient care.
For more information, see http://cancercenters.cancer.gov/.
http://letstalkdata.com
@letstalkdata
Cancer Center
Area closest to that cancer center

① Introduction

Example 1 : read in a point shapefile to be converted to a Voronoi diagram

```
library(rgdal)

dsn <- system.file("vectors", package = "rgdal")[1]
cities <- readOGR(dsn=dsn, layer="cities")
str(cities)
```

```
> str(cities)
Formal class 'SpatialPointsDataFrame' [package "sp"] with 5 slots
  ..@ data       :'data.frame':       606 obs. of  4 variables:
  .. ..$ NAME      : Factor w/ 602 levels "Abidjan","Abu Zaby",..: 368 34 472 322 419 593 390 
194 263 115 ...
  .. ..$ COUNTRY   : Factor w/ 165 levels "Afghanistan",..: 124 124 124 124 124 124 124 153 12
4 124 ...
  .. ..$ POPULATION: num [1:606] 468000 416000 5825000 152000 1160000 ...
  .. ..$ CAPITAL   : Factor w/ 2 levels "N","Y": 1 1 1 1 1 1 1 1 1 1 ...
  ..@ coords.nrs : num(0) 
  ..@ coords     : num [1:606, 1:2] 33.1 40.6 30.5 150.8 56.2 ...
  .. ..- attr(*, "dimnames")=List of 2
  .. .. ..$ : NULL
  .. .. ..$ : chr [1:2] "coords.x1" "coords.x2"
  ..@ bbox       : num [1:2, 1:2] -165.3 -53.2 177.1 78.2
  .. ..- attr(*, "dimnames")=List of 2
  .. .. ..$ : chr [1:2] "coords.x1" "coords.x2"
  .. .. ..$ : chr [1:2] "min" "max"
  ..@ proj4string:Formal class 'CRS' [package "sp"] with 1 slot
  .. .. ..@ projargs: chr "+proj=longlat +datum=WGS84 +no_defs +ellps=WGS84 +towgs84=0,0,0"
```

Function to create voronoi polygons (originally developed by Carson Farmer) : this (modified) function accepts vectors of coordinates (expected in order x, y) as well as a SPDF (see *http://stackoverflow.com/questions/12156475*)

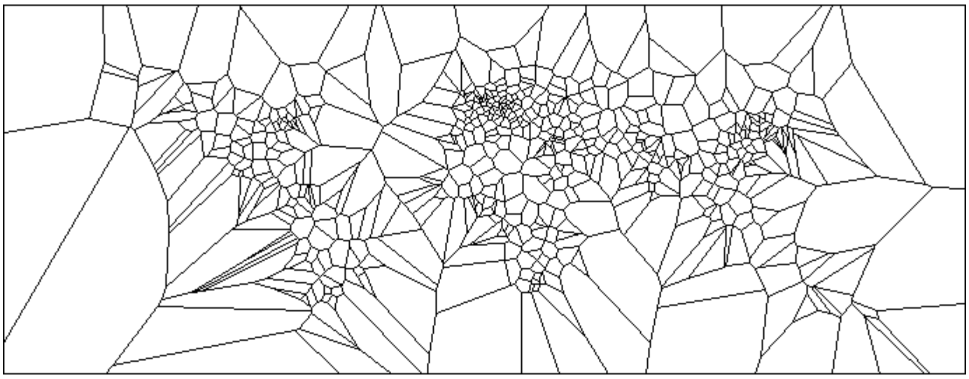
```
# Carson's Voronoi polygons function
library(deldir)
voronoipolygons <- function(x) {
  require(deldir)
  require(sp)
  if (.hasSlot(x, 'coords')) {
    crds <- x@coords  
  } else crds <- x
  z <- deldir(crds[,1], crds[,2])
  w <- tile.list(z)
  polys <- vector(mode='list', length=length(w))
  for (i in seq(along=polys)) {
    pcrds <- cbind(w[[i]]$x, w[[i]]$y)
    pcrds <- rbind(pcrds, pcrds[1,])
    polys[[i]] <- Polygons(list(Polygon(pcrds)), ID=as.character(i))
  }
  SP <- SpatialPolygons(polys)
  voronoi <- SpatialPolygonsDataFrame(SP, data=data.frame(x=crds[,1], 
                                                           y=crds[,2], 
                      row.names=sapply(slot(SP, 'polygons'), 
                      function(x) slot(x, 'ID'))))
}


v <- voronoipolygons(cities)
class(v)
```

```
> class(v)
[1] "SpatialPolygonsDataFrame"
attr(,"package")
[1] "sp"
```

```
str(v@data)
```

```
> str(v@data)
'data.frame':       606 obs. of  2 variables:
 $ x: num  33.1 40.6 30.5 150.8 56.2 ...
 $ y: num  69 64.5 60 59.6 58 ...
```

```
plot(v)
```



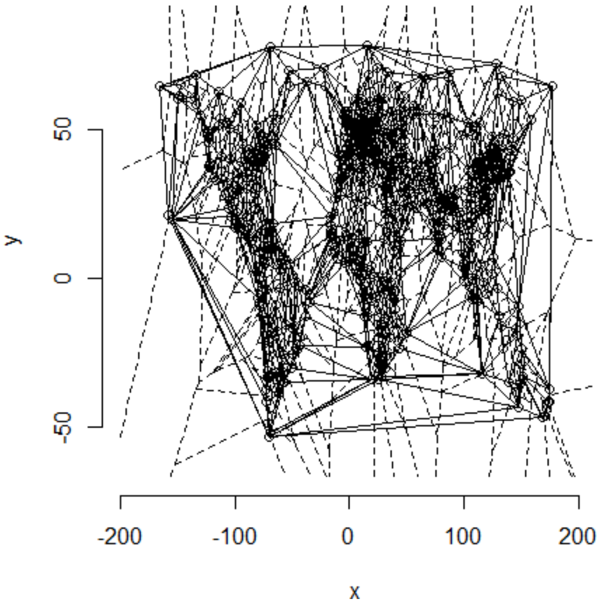
Alternative way to create Voronoi diagram with library deldir.

```
library(deldir)
v_ <- deldir(cities$coords.x1,cities$coords.x2)
class(v_)
```

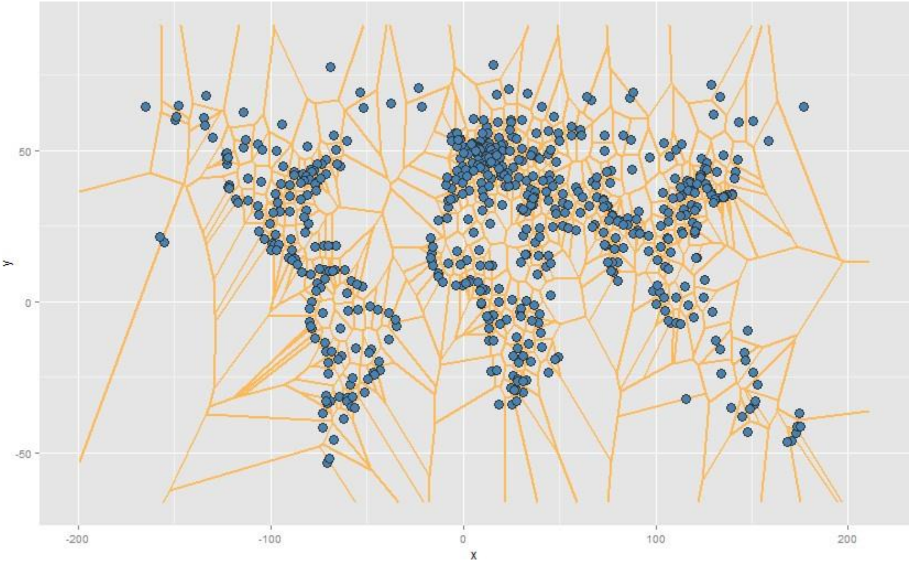
```
> class(v_)
[1] "deldir"
```

```
> str(v_)
List of 9
 $ delsgs  :'data.frame':        1805 obs. of  6 variables:
  ..$ x1  : num [1:1805] 40.6 30.5 30.5 60.6 43.9 ...
  ..$ y1  : num [1:1805] 64.5 60 60 56.8 56.3 ...
  ..$ x2  : num [1:1805] 33.1 33.1 40.6 56.2 40.6 ...
  ..$ y2  : num [1:1805] 69 69 64.5 58 64.5 ...
  ..$ ind1: num [1:1805] 2 3 3 6 7 9 9 10 11 11 ...
  ..$ ind2: num [1:1805] 1 1 2 5 2 5 7 6 6 10 ...
 $ dirsgs  :'data.frame':        1782 obs. of  8 variables:
  ..$ x1  : num [1:1782] 49.4 35 36 59.9 42.6 ...
  ..$ y1  : num [1:1782] 88.1 63.5 61.3 63.2 60.5 ...
  ..$ x2  : num [1:1782] 35 28 35 58.1 40.6 ...
  ..$ y2  : num [1:1782] 63.5 65.6 63.5 56.3 59.7 ...
  ..$ ind1: num [1:1782] 2 3 3 6 7 9 9 10 11 11 ...
  ..$ ind2: num [1:1782] 1 1 2 5 2 5 7 6 6 10 ...
  ..$ bp1 : logi [1:1782] FALSE FALSE FALSE FALSE FALSE FALSE ...
  ..$ bp2 : logi [1:1782] FALSE FALSE FALSE FALSE FALSE FALSE ...
 $ summary :'data.frame':        606 obs. of  9 variables:
  ..$ x       : num [1:606] 33.1 40.6 30.5 150.8 56.2 ...
  ..$ y       : num [1:606] 69 64.5 60 59.6 58 ...
  ..$ n.tri   : num [1:606] 6 6 7 5 5 6 7 6 6 7 ...
  ..$ del.area: num [1:606] 113.4 79.8 49.9 135 41.3 ...
  ..$ del.wts : num [1:606] 0.00295 0.00208 0.0013 0.00351 0.00107 ...
  ..$ n.tside : num [1:606] 6 6 7 5 5 6 7 6 6 7 ...
  ..$ nbpt    : num [1:606] 2 0 0 2 0 0 0 0 0 0 ...
  ..$ dir.area: num [1:606] 358.5 188.2 58.5 424.7 61.9 ...
  ..$ dir.wts : num [1:606] 0.005535 0.002906 0.000903 0.006558 0.000955 ...
 $ n.data  : int 606
 $ n.dum   : num 0
 $ del.area: num 38462
 $ dir.area: num 64763
 $ rw      : num [1:4] -199.5 211.4 -66.3 91.3
 $ ind.orig: int [1:606] 1 2 3 4 5 6 7 8 9 10 ...
 - attr(*, "class")= chr "deldir"
```

```
plot(v_)
```



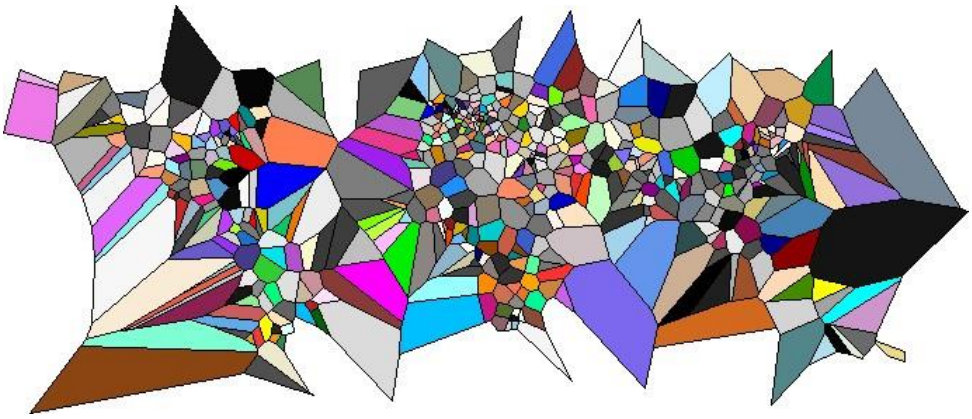
```
# make a plot
#  see http://letstalkdata.com/2014/05/creating-voronoi-diagrams-with-ggplot

library(ggplot2)
ggplot(data=v_$summary, aes(x=x,y=y)) +
  # plot the voronoi lines
  geom_segment(data=v_$dirsgs,aes(x = x1, y = y1, xend = x2, yend = y2),
                size = 1,
                linetype = 1,
                color= "#FFB958") +
  # plot the points
  geom_point(fill=rgb(70,130,180,255,maxColorValue=255),
              pch=21,
              size = 4,
              color="#333333")
```



```
# plot the polygons of a Voronoi tesselation instead of segments
# convert line segments to spatialpolygons objects
# see http://stackoverflow.com/questions/24311304
# function to convert line segments in $dirsgs to spatialpolygons

voronoi <- v_
voronoi$dirsgs <- v_$dirsgs
library(rgeos)

ll <- apply(voronoi$dirsgs, 1, FUN=function(X) {
  readWKT(sprintf("LINESTRING(%s %s, %s %s)", X[1], X[2], X[3], X[4]))
})


ll <- apply(v_$dirsgs, 1, FUN=function(X) {
  readWKT(sprintf("LINESTRING(%s %s, %s %s)", X[1], X[2], X[3], X[4]))
})


# convert SpatialLines list to SpatialPolygons object
pp <- gPolygonize(ll)
class(pp)
```

```
> class(pp)
[1] "SpatialPolygons"
attr(,"package")
[1] "sp"
```

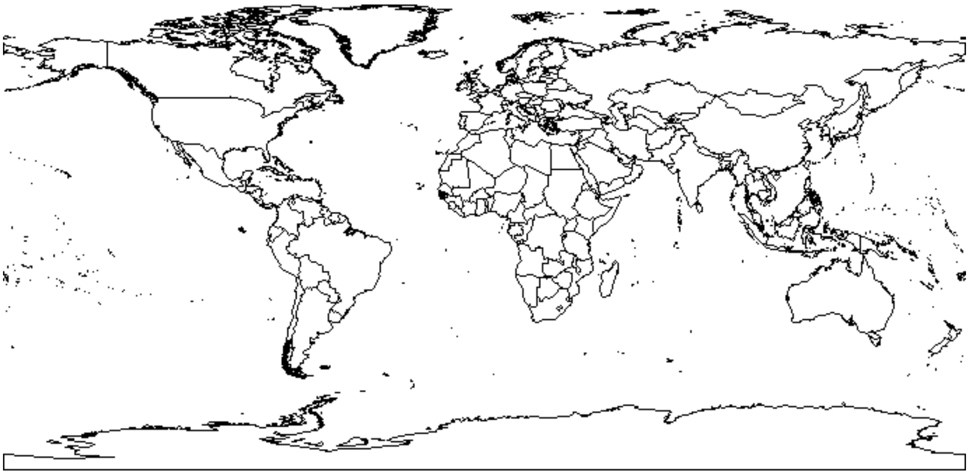
```
# plot
set.seed=11
plot(pp, col=sample(colors(), length(pp)))
```



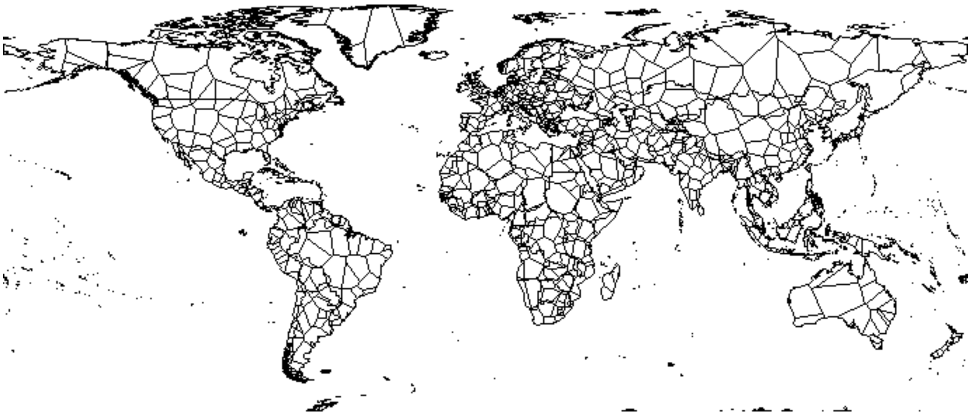
```
# read shapefile of the world
setwd("c:/R/Rdata")
world <- readOGR(".","TM_WORLD_BORDERS_SIMPL-0.3")
class(world)
```

```
> class(world)
[1] "SpatialPolygonsDataFrame"
attr(,"package")
[1] "sp"
```

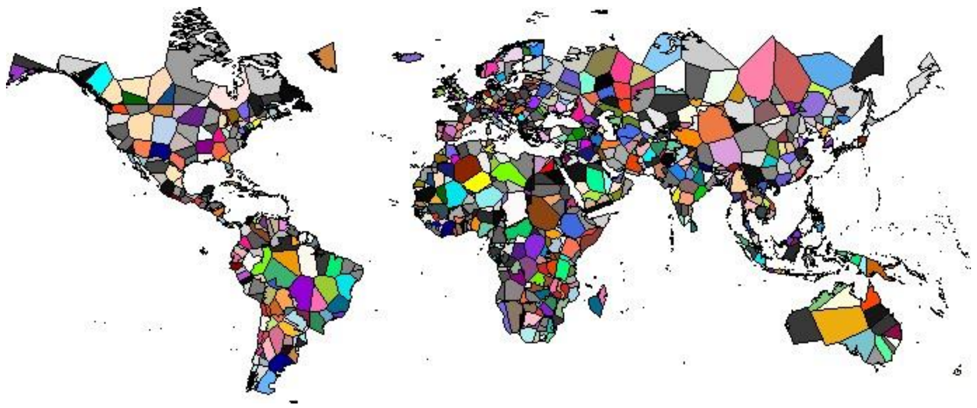
```
plot(world)
```



```
final <- gIntersection(world,v,byid=TRUE)
plot(final)
```



```
final2 <- gIntersection(world,pp,byid=TRUE)
plot(final2,col=sample(colors(), length(pp)))
```

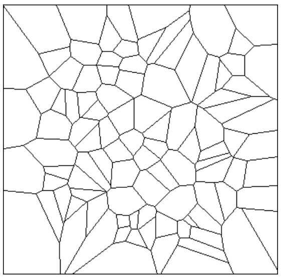


Example 2 : input is vectors of x, y coordinates

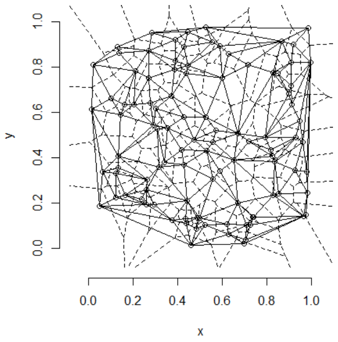
```
dat <- data.frame(x=runif(100), y=runif(100))
str(dat)
```

```
> str(dat)
'data.frame':	100 obs. of  2 variables:
 $ x: num  0.284 0.521 0.818 0.303 0.309 ...
 $ y: num  0.95 0.106 0.436 0.618 0.528 ...
```

```
v2 <- voronoipolygons(dat)
plot(v2)
```



```
library(deldir)
v3 <- deldir(dat$x,dat$y)
plot(v3)
```



② Working with voronoi polygons

```
R script : "c:/R/Rdata/voronoi.R"


# hospitals West Flanders : estimate the population served by hospitals
# note : remove sites with duplicate lat/long, which would break the
  voronoi algorithm !
# e.g. sites <- sites[!duplicated(sites$latitude), ]

# input of x, y coordinates in csv-file
setwd("c:/R/Rdata")
hospitals <- read.csv("ziekenhuizen_wvl_geocoded.csv",header=T,sep=",")
str(hospitals)
```

```
> str(hospitals)
'data.frame':	37 obs. of  4 variables:
 $ address : Factor w/ 37 levels "AZ Alma Campus Sijsele Gentsesteenweg 132 8340 Sijsele",..:
1 2 3 5 4 6 7 8 9 12 ...
 $ lat     : num  51.2 51.2 51.2 50.8 50.8 ...
 $ long    : num  3.32 2.93 2.91 3.26 3.27 ...
 $ accuracy: Factor w/ 4 levels "hospital","locality",..: 4 2 1 3 3 4 3 1 3 1 ...
```

```
head(hospitals)
```

```
> head(hospitals)
                                                                 address      lat     long            accuracy
1                 AZ Alma Campus Sijsele Gentsesteenweg 132 8340 Sijsele 51.20156 3.321246 sublocality_level_1
2            AZ Damiaan Campus H. Hart Gauwelozestraat 100 8400 Oostende 51.21543 2.928656            locality
3     AZ Damiaan Campus Sint-Jozef Nieuwpoortsesteenweg 57 8400 Oostende 51.22322 2.906011            hospital
4                AZ Groeninge Campus O.-L Vrouw Reepkaai 4 8500 Kortrijk 50.82971 3.260921      street_address
5 AZ Groeninge Campus Maria's Voorzienigheid Loofstraat 43 8500 Kortrijk 50.81991 3.267864      street_address
6  AZ Groeninge Campus Sint-Maarten Burg. Vercruysselaan 5 8500 Kortrijk 50.81949 3.257708 sublocality_level_1
```

```
coords <- as.data.frame(cbind(hospitals$long,hospitals$lat))
class(coords)
```

```
> class(coords)
[1] "data.frame"
```

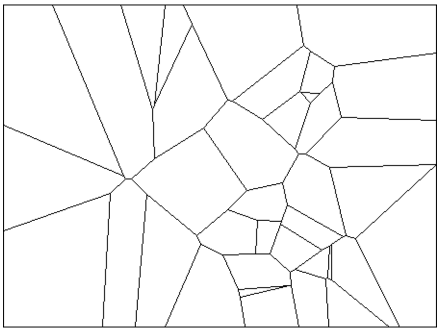
```
str(coords)
```

```
> str(coords)
'data.frame':	37 obs. of  2 variables:
 $ V1: num  3.32 2.93 2.91 3.26 3.27 ...
 $ V2: num  51.2 51.2 51.2 50.8 50.8 ...
```

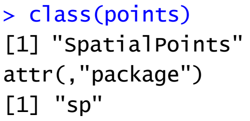
```
names(coords) <- c("long","lat")
vp <- voronoipolygons(coords)
class(vp)
```

```
> class(vp)
[1] "SpatialPolygonsDataFrame"
attr(,"package")
[1] "sp"
```

```
plot(vp)
```

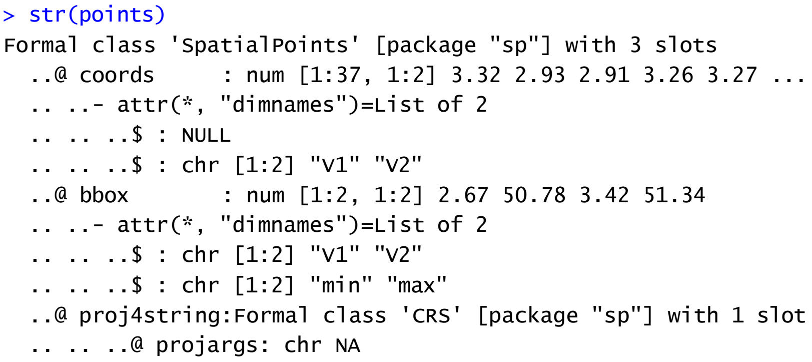


```
# convert coords into SpatialPoints
points <- SpatialPoints(coords)
class(points)
```



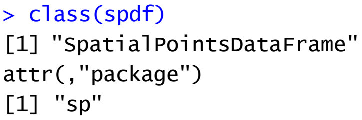
```
> class(points)
[1] "SpatialPoints"
attr(,"package")
[1] "sp"
```

```
str(points)
```



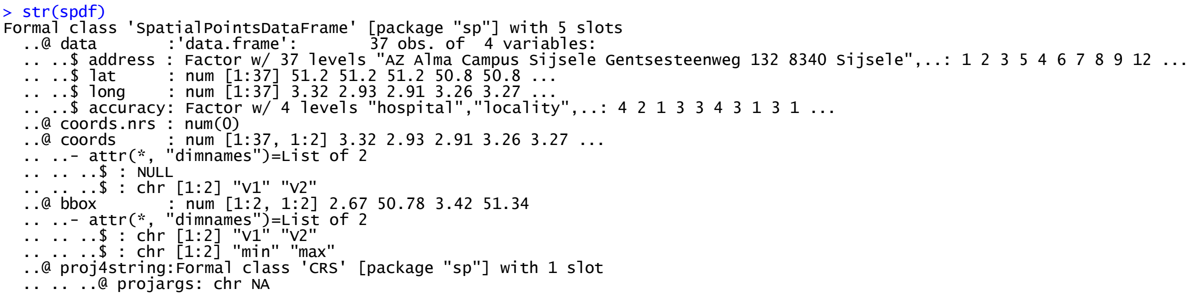
```
> str(points)
Formal class 'SpatialPoints' [package "sp"] with 3 slots
  ..@ coords     : num [1:37, 1:2] 3.32 2.93 2.91 3.26 3.27 ...
  .. ..- attr(*, "dimnames")=List of 2
  .. .. ..$ : NULL
  .. .. ..$ : chr [1:2] "V1" "V2"
  ..@ bbox       : num [1:2, 1:2] 2.67 50.78 3.42 51.34
  .. ..- attr(*, "dimnames")=List of 2
  .. .. ..$ : chr [1:2] "V1" "V2"
  .. .. ..$ : chr [1:2] "min" "max"
  ..@ proj4string:Formal class 'CRS' [package "sp"] with 1 slot
  .. .. ..@ projargs: chr NA
```

```
# create SpatialPointsDataFrame of points
spdf <- SpatialPointsDataFrame(points,hospitals)
class(spdf)
```



```
> class(spdf)
[1] "SpatialPointsDataFrame"
attr(,"package")
[1] "sp"
```

```
str(spdf)
```

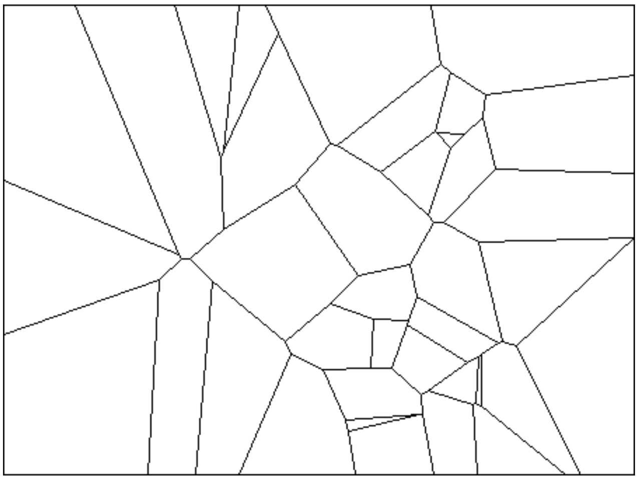


```
> str(spdf)
Formal class 'SpatialPointsDataFrame' [package "sp"] with 5 slots
  ..@ data       :'data.frame':       37 obs. of  4 variables:
  .. ..$ address : Factor w/ 37 levels "AZ Alma Campus Sijsele Gentsesteenweg 132 8340 Sijsele",..: 1 2 3 5 4 6 7 8 9 12 ...
  .. ..$ lat     : num [1:37] 51.2 51.2 51.2 50.8 50.8 ...
  .. ..$ long    : num [1:37] 3.32 2.93 2.91 3.26 3.27 ...
  .. ..$ accuracy: Factor w/ 4 levels "hospital","locality",..: 4 2 1 3 3 4 3 1 3 1 ...
  ..@ coords.nrs : num(0)
  ..@ coords     : num [1:37, 1:2] 3.32 2.93 2.91 3.26 3.27 ...
  .. ..- attr(*, "dimnames")=List of 2
  .. .. ..$ : NULL
  .. .. ..$ : chr [1:2] "V1" "V2"
  ..@ bbox       : num [1:2, 1:2] 2.67 50.78 3.42 51.34
  .. ..- attr(*, "dimnames")=List of 2
  .. .. ..$ : chr [1:2] "V1" "V2"
  .. .. ..$ : chr [1:2] "min" "max"
  ..@ proj4string:Formal class 'CRS' [package "sp"] with 1 slot
  .. .. ..@ projargs: chr NA
```

```
voronoiPolys <- voronoipolygons(spdf)
class(voronoiPolys)
```

```
> class(voronoiPolys)
[1] "SpatialPolygonsDataFrame"
attr(,"package")
[1] "sp"
```

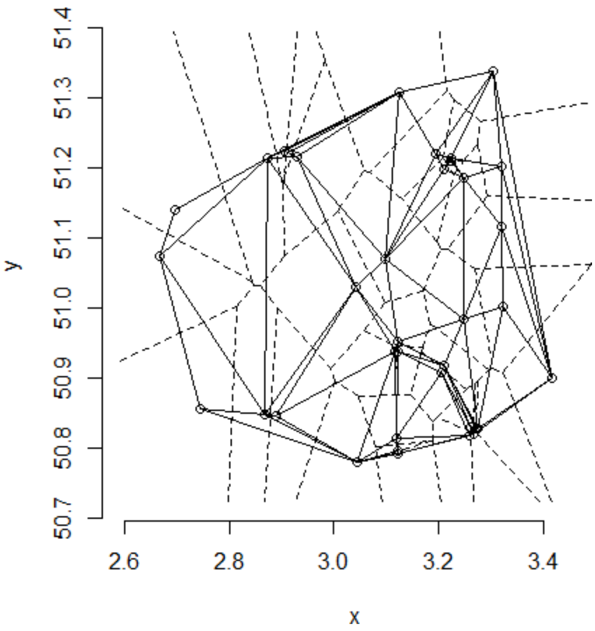
```
plot(voronoiPolys)
```



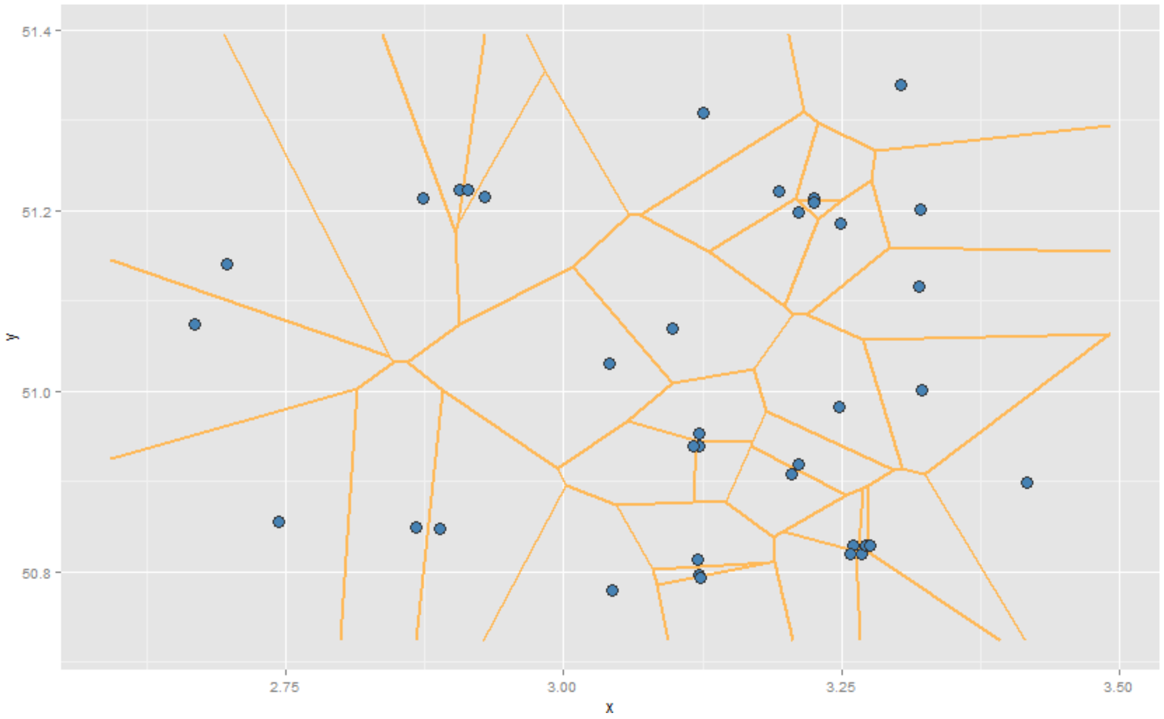
```
library(deldir)
v_ <- deldir(spdf@data$long,spdf@data$lat)
class(v_)
```

```
> class(v_)
[1] "deldir"
```

```
plot(v_)
```

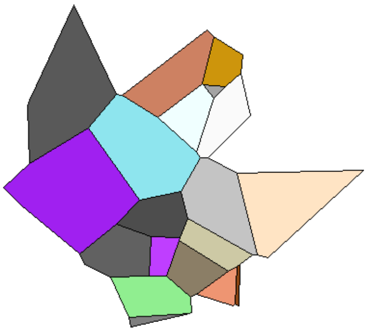


```
# make a plot
#  see  http://letstalkdata.com/2014/05/creating-voronoi-diagrams-with-ggplot

library(ggplot2)
ggplot(data=v_$summary, aes(x=x,y=y)) +
  # plot the voronoi lines
  geom_segment(data=v_$dirsgs,aes(x = x1, y = y1, xend = x2, yend = y2),
               size = 1,
               linetype = 1,
               color= "#FFB958") +
  # plot the points
  geom_point(fill=rgb(70,130,180,255,maxColorValue=255),
             pch=21,
             size = 4,
             color="#333333")
```

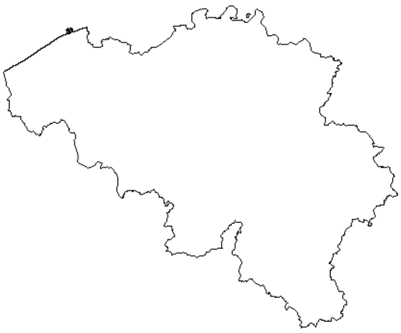


```
# plot the polygons of a Voronoi tesselation instead of segments
# convert line segments to spatialpolygons objects

voronoi <- v_
voronoi$dirsgs <- v_$dirsgs
library(rgeos)

ll <- apply(voronoi$dirsgs, 1, FUN=function(X) {
  readWKT(sprintf("LINESTRING(%s %s, %s %s)", X[1], X[2], X[3], X[4]))
})

# convert SpatialLines list to SpatialPolygons object
pp <- gPolygonize(ll)
```

```
# plot
set.seed=11
plot(pp, col=sample(colors(), length(pp)))
```



```
# plot voronoi polygons (vor) and hospitals (coords)

# read shapefile of Belgium
library(rgdal)
belgium <- readOGR(".","BEL_adm0")
plot(belgium)
```



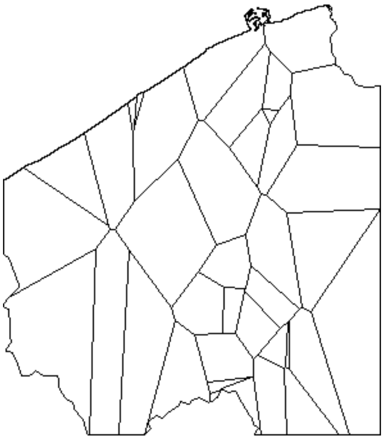
```
class(belgium)
```

```
> class(belgium)
[1] "SpatialPolygonsDataFrame"
attr(,"package")
[1] "sp"
```

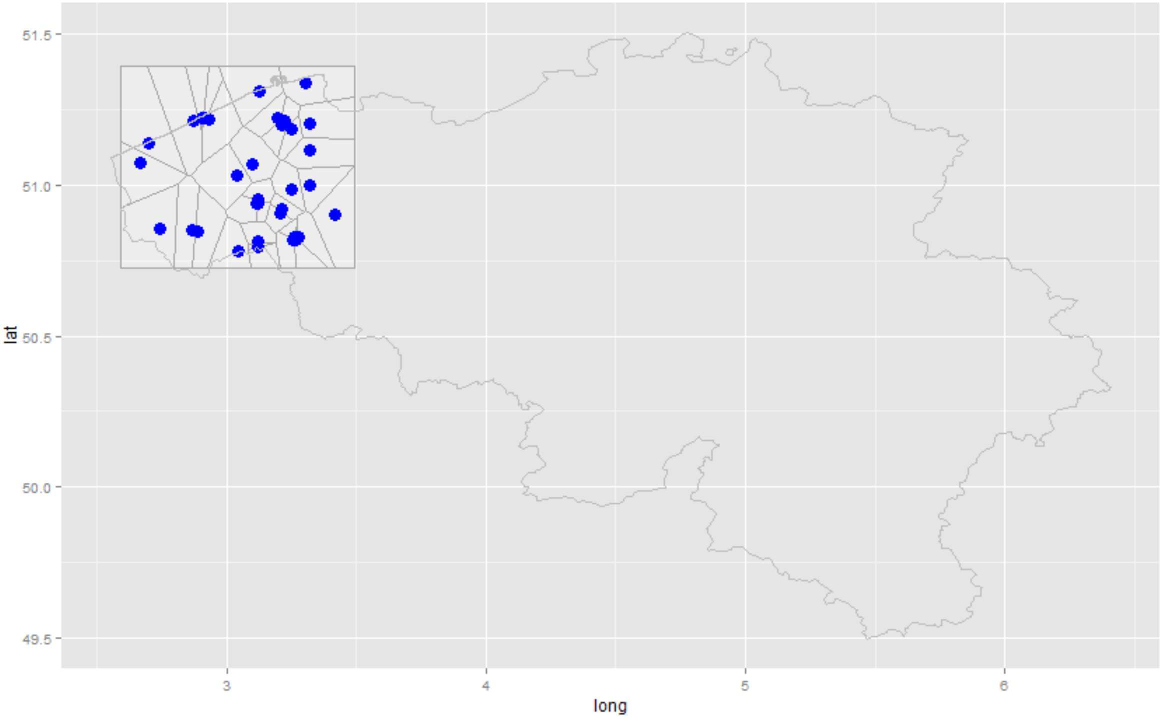
```
final <- gIntersection(belgium,voronoiPolys,byid=TRUE)
class(final)
```

```
> class(final)
[1] "SpatialPolygons"
attr(,"package")
[1] "sp"
```

```
plot(final)
```



```
# convert voronoiPolys and shapefile Belgium into data frames to plot with ggplot2
vor <- fortify(voronoiPolys)
bel <- fortify(belgium)

# plot voronoi polygons (vor) and hospitals (coords)
HospAndVor <- ggplot(data=vor,aes(x=long,y=lat)) +
              geom_polygon(aes(group=group),colour="grey65",size=0.4,fill="white",alpha=0.3) +
              geom_point(data=coords,aes(x=long,y=lat),colour="blue",size=4) +
              geom_polygon(data=bel,aes(x=long,y=lat,group=group),color="grey",fill="NA")
HospAndVor
```



```
# plot voronoi diagram and population dots
# shapefile België mét veld voor populatie
library(rgdal)
library(maptools)
setwd("c:/R/Rdata")
belgie <- readOGR(".","newbelgie")
plot(belgie)
```

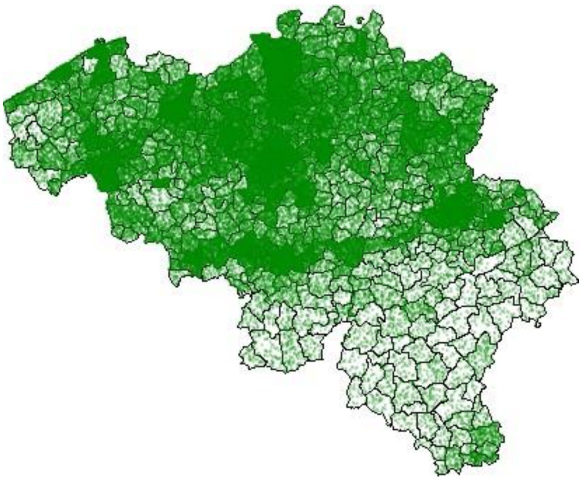
```
str(belgie@data)
```

```
> str(belgie@data)
'data.frame':            589 obs. of  21 variables:
 $ ID_4      : int  3 4 5 11 12 15 16 17 21 27 ...
 $ ID_0      : int  22 22 22 22 22 22 22 22 22 22 ...
 $ ISO       : Factor w/ 1 level "BEL": 1 1 1 1 1 1 1 1 1 1 ...
 $ NAME_0    : Factor w/ 1 level "Belgium": 1 1 1 1 1 1 1 1 1 1 ...
 $ ID_1      : int  1 1 1 1 1 1 1 1 1 1 ...
 $ NAME_1    : Factor w/ 3 levels "Bruxelles","Vlaanderen",..: 2 2 2 2 2 2 2 2 2 2 ...
 $ ID_2      : int  1 1 1 1 1 1 1 1 1 1 ...
 $ NAME_2    : Factor w/ 11 levels "Antwerpen","Brabant Wallon",..: 1 1 1 1 1 1 1 1 1 1 ...
 $ ID_3      : int  1 1 1 1 1 1 1 1 1 1 ...
 $ NAME_3    : Factor w/ 43 levels "Aalst","Antwerpen",..: 2 2 2 2 2 2 2 2 2 2 ...
 $ NAME_4    : Factor w/ 588 levels "Ã‰caussinnes",..: 152 267 568 19 85 270 493 84 344 466 ...
 $ VARNAME_4: Factor w/ 41 levels "'s-Gravenbrakel",..: NA NA NA NA NA NA NA NA NA NA ...
 $ TYPE_4    : Factor w/ 2 levels "Commune","Gemeente": 2 2 2 2 2 2 2 2 2 2 ...
 $ ENGTYPE_4: Factor w/ 1 level "Commune": 1 1 1 1 1 1 1 1 1 1 ...
 $ row       : num  0 1 2 3 4 5 6 7 8 9 ...
 $ niscode   : int  11016 11022 11053 11002 11009 11023 11044 11008 11057 11040 ...
 $ twtot     : int  4248 5123 4258 265777 7323 7360 3123 12434 7981 11095 ...
 $ TOTAALBEV: int  18075 18126 19547 502604 27906 26546 18028 37301 14763 33766 ...
 $ DENSITEIT: num  381 305 219 2458 307 ...
 $ diff      : num  -76.5 -71.7 -78.2 -47.1 -73.8 -72.3 -82.7 -66.7 -45.9 -67.1 ...
 $ aandeel   : num  23.5 28.3 21.8 52.9 26.2 27.7 17.3 33.3 54.1 32.9 ...
```

```
# turn shapefile into dotmap
dots.pop <- dotsInPolys(belgie,belgie@data$TOTAALBEV/100)
class(dots.pop)
```

```
> class(dots.pop)
[1] "SpatialPointsDataFrame"
attr(,"package")
[1] "sp"
```

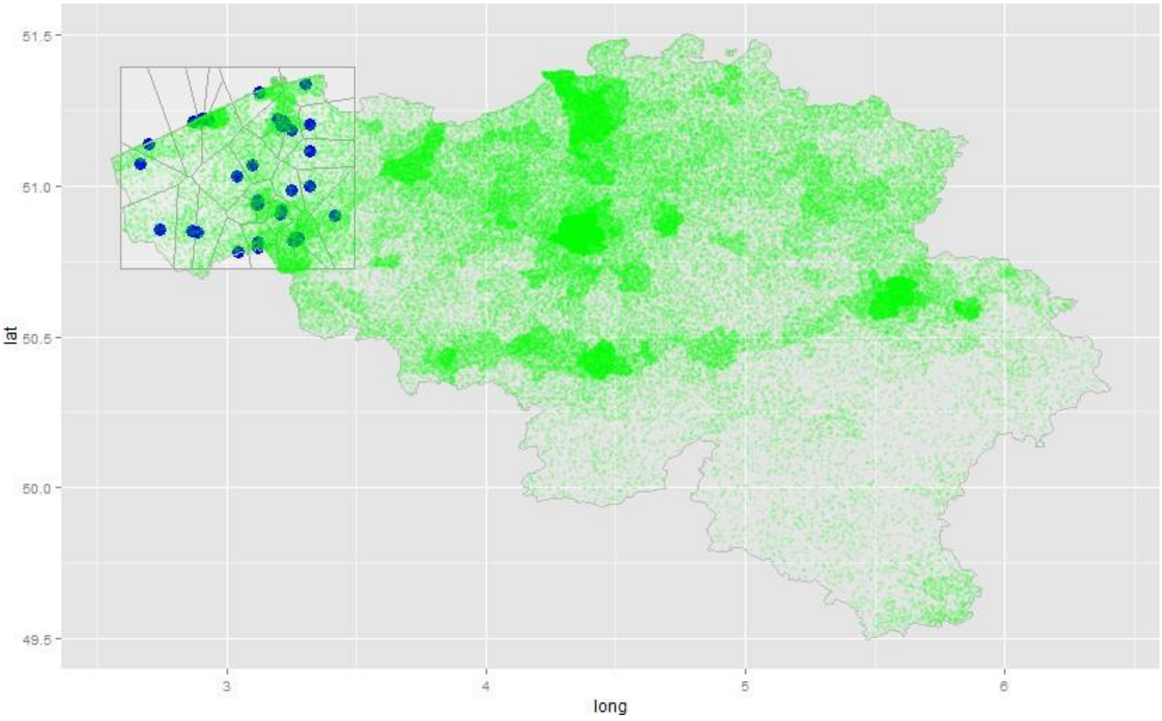
```
plot(belgie,lwd=0.05)
plot(dots.pop,add=TRUE,pch=19,cex=0.1,col="#00880030")
```



```
# extract dataframe from population dots
pop.df <- data.frame(coordinates(dots.pop[,1:2])
str(pop.df)
```

```
> str(pop.df)
'data.frame':	110063 obs. of  2 variables:
 $ x: num  4.49 4.46 4.52 4.5 4.45 ...
 $ y: num  51.4 51.4 51.5 51.5 51.5 ...
```

```
VorAndDots <- HospAndVor +
geom_point(data=pop.df,aes(x=x,y=y),size=0.5,color="green",alpha=0.15)
VorAndDots
```



```
# estimate the total population served by each clinic
# pop.df (data frame) and voronoiPolys (SpatialPolygonsDataFrame) must have the
  same CRS

proj4string(voronoiPolys)

> proj4string(voronoiPolys)
[1] "+proj=longlat +datum=WGS84 +no_defs +ellps=WGS84 +towgs84=0,0,0"

coordinates(pop.df) <- ~ x + y
class(pop.df)

> class(pop.df)
[1] "SpatialPoints"
attr(,"package")
[1] "sp"

proj4string(pop.df) <- proj

# count the number of dots in each polygon in the voronoi diagram

count <- sapply(over(voronoiPolys,SpatialPoints(pop.df),returnList=TRUE),length)
count

> count
   1    2    3    4    5    6    7    8    9   10   11   12   13   14   15   16   17   18   19   20   21   22   23   24   25 
26   27   28 
 372  871   85  175  445  618   36  475  485  227  235  373  356   58  637  178  341  230  238   46 1168  273  251  415  260 
291   25  355 
  29   30   31   32   33   34   35   36   37 
 330  303  385  321  283  317  555  310  461 

class(voronoiPolys)

> class(voronoiPolys)
[1] "SpatialPolygonsDataFrame"
attr(,"package")
[1] "sp"

class(pop.df)

> class(pop.df)
[1] "SpatialPoints"
attr(,"package")
[1] "sp"


# add the results of the count to the polygons
voronoiPolys <- spCbind(voronoiPolys, as.data.frame(count))
str(voronoiPolys@data)

> str(voronoiPolys@data)
'data.frame':	37 obs. of  3 variables:
 $ x    : num  3.32 2.93 2.91 3.26 3.27 ...
 $ y    : num  51.2 51.2 51.2 50.8 50.8 ...
 $ count: int  372 871 85 175 445 618 36 475 485 227 ...

# convert voronoiPolys (SPDF) into a data frame to plot with ggplot2
voronoiPolys@data$id <- rownames(voronoiPolys@data)
str(voronoiPolys@data)

> str(voronoiPolys@data)
'data.frame':	37 obs. of  4 variables:
 $ x    : num  3.32 2.93 2.91 3.26 3.27 ...
 $ y    : num  51.2 51.2 51.2 50.8 50.8 ...
 $ count: int  372 871 85 175 445 618 36 475 485 227 ...
 $ id   : chr  "1" "2" "3" "4" ...
```

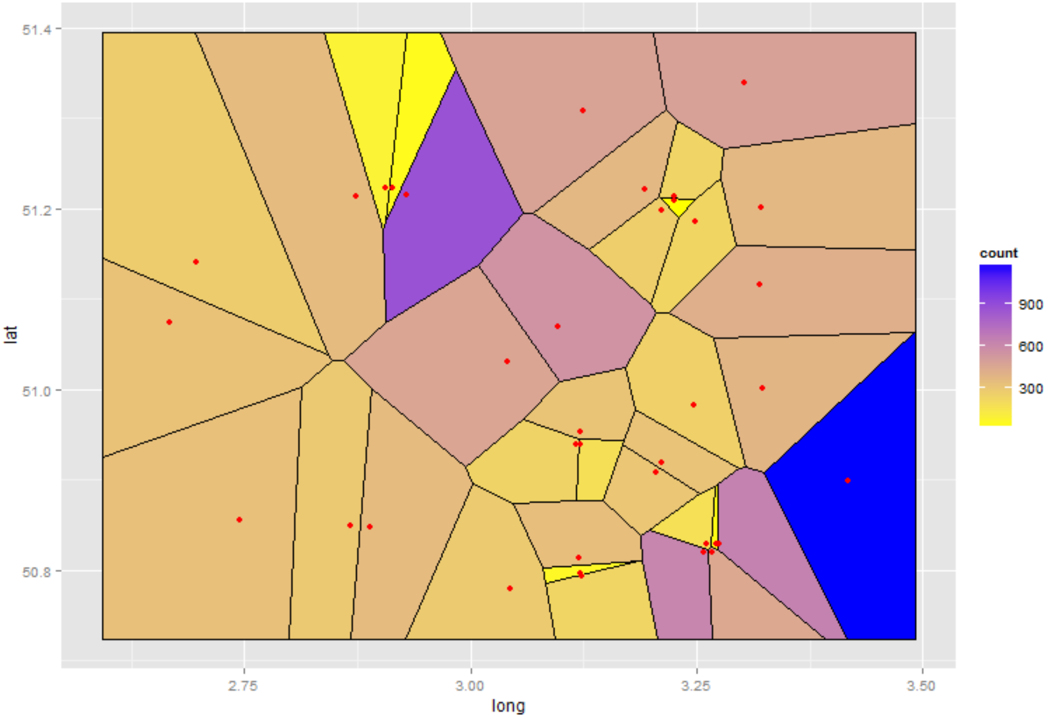
```
voronoiPolys.df <- fortify(voronoiPolys,region="id")
str(voronoiPolys.df)
```

```
> str(voronoiPolys.df)
'data.frame':	239 obs. of  7 variables:
 $ long : num  3.49 3.49 3.29 3.28 3.28 ...
 $ lat  : num  51.3 51.2 51.2 51.2 51.3 ...
 $ order: int  1 2 3 4 5 6 7 8 9 10 ...
 $ hole : logi  FALSE FALSE FALSE FALSE FALSE FALSE ...
 $ piece: Factor w/ 1 level "1": 1 1 1 1 1 1 1 1 1 1 ...
 $ group: Factor w/ 37 levels "1.1","10.1","11.1",..: 1 1 1 1 1 1 2 2 2 2 ...
 $ id   : chr  "1" "1" "1" "1" ...
```

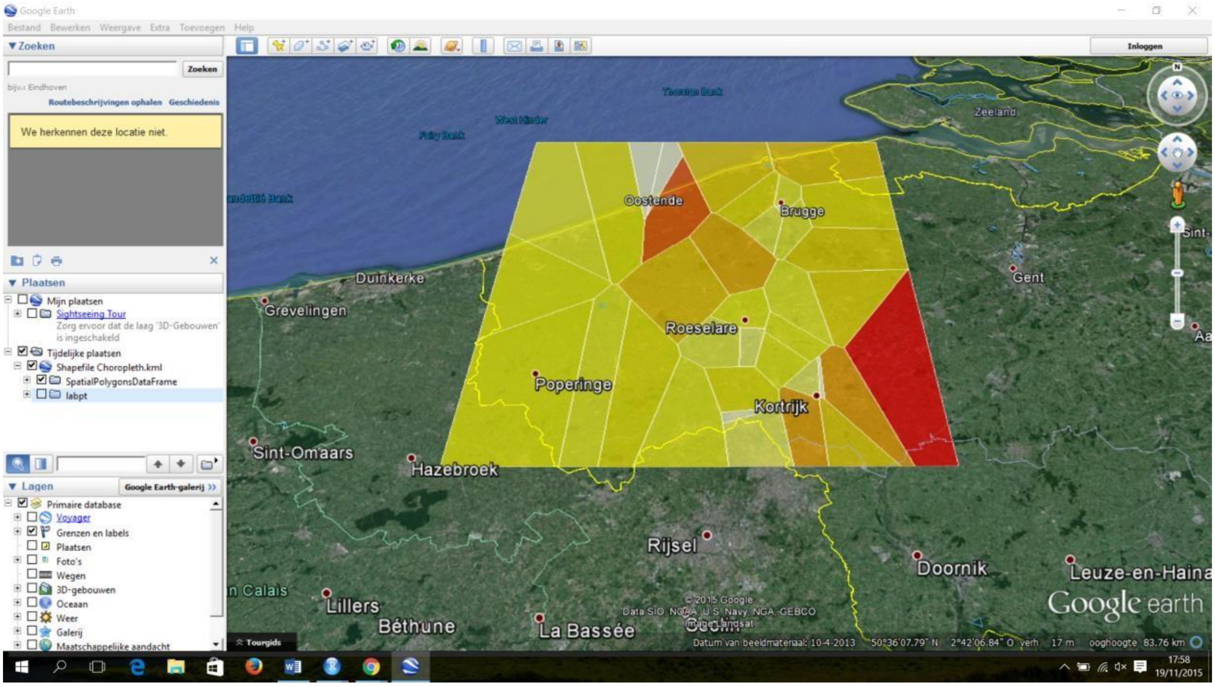
```
voronoiPolys.df <- merge(voronoiPolys.df,voronoiPolys@data,by="id")
str(voronoiPolys.df)
```

```
> str(voronoiPolys.df)
'data.frame':	239 obs. of  10 variables:
 $ id   : chr  "1" "1" "1" "1" ...
 $ long : num  3.49 3.49 3.29 3.28 3.28 ...
 $ lat  : num  51.3 51.2 51.2 51.2 51.3 ...
 $ order: int  1 2 3 4 5 6 7 8 9 10 ...
 $ hole : logi  FALSE FALSE FALSE FALSE FALSE FALSE ...
 $ piece: Factor w/ 1 level "1": 1 1 1 1 1 1 1 1 1 1 ...
 $ group: Factor w/ 37 levels "1.1","10.1","11.1",..: 1 1 1 1 1 1 2 2 2 2 ...
 $ x    : num  3.32 3.32 3.32 3.32 3.32 ...
 $ y    : num  51.2 51.2 51.2 51.2 51.2 ...
 $ count: int  372 372 372 372 372 372 227 227 227 227 ...
```

```
map <- ggplot(voronoiPolys.df,aes(x=long,y=lat)) +
        geom_polygon(aes(group=group,fill=count),colour="black",size=0.1) +
        scale_fill_continuous(low="yellow",high="blue") +
        geom_point(data=coords,aes(x=long,y=lat),colour="red",size=2) +
        coord_equal()
map
```



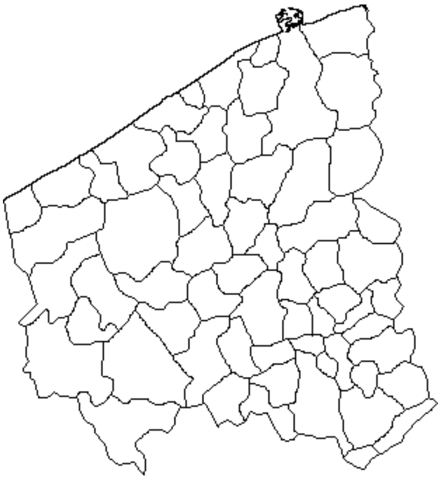
```
# plot voronoiPolys (SpatialPolygonsDataFrame) with Google Earth (library plotKML)

library(plotKML)
voronoiPolys <- spTransform(voronoiPolys,CRS("+proj=longlat +datum=WGS84"))
kml(obj=voronoiPolys, folder.name="", file.name="Shapefile Choropleth.kml",
     kmz=FALSE,colour=count,colour_scale=R_pal[["heat_colors"]],alpha=0.75,
     altitude=0,plot.labpt=TRUE,labels=LABEL,LabelScale=0.5)
```



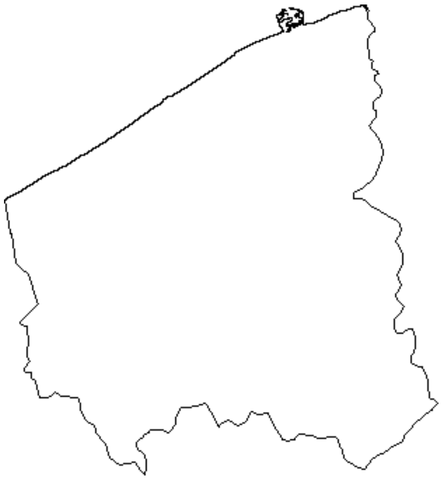
One of the weaknesses of voronoi estimators is their high variability, which makes them very sensitive to noise. We need to clip the voronoi diagram polygons within the geographic borders of the area we want to cover (rather than the arbitrarily large polygons at the edges of the diagram).

To do this, we import a shapefile of Belgian municipalities, select the province of West Flanders and collapse the polygons of that region into a single polygon. Then we repeat the steps of the previous exercise.

```
belgie <- readOGR(".","BEL_adm4")
wvl <- belgie[belgie@data$NAME_2 == "West-Vlaanderen", ]
plot(wvl)
```



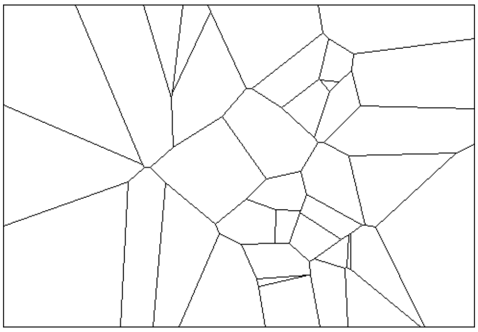
```
library(maptools)
regs <- c("West-Vlaanderen")
sp2 <- wvl@data$NAME_2
sp2[which(!wvl@data$NAME_2 %in% regs)] <- NA
wvlP <- unionSpatialPolygons(wvl,sp2)
plot(wvlP)
```



```
# function to clip voronoi polygons to a given extent
voronoipolygons <- function(x, poly) {
  require(deldir)
  if (.hasSlot(x, 'coords')) {
    crds <- x@coords
  } else crds <- x
  bb = bbox(poly)
  rw = as.numeric(t(bbox(poly)))
  z <- deldir(crds[,1], crds[,2],rw=rw)
  w <- tile.list(z)
  polys <- vector(mode='list', length=length(w))
  require(sp)
  for (i in seq(along=polys)) {
    pcrds <- cbind(w[[i]]$x, w[[i]]$y)
    pcrds <- rbind(pcrds, pcrds[1,])
    polys[[i]] <- Polygons(list(Polygon(pcrds)), ID=as.character(i))
  }
  SP <- SpatialPolygons(polys)

  SpatialPolygonsDataFrame(
    SP, data.frame(x=crds[,1], y=crds[,2],
                   row.names=sapply(slot(SP, 'polygons'),
                                    function(x) slot(x, 'ID'))))
}

V <- voronoipolygons(coords,wvlP)
class(V)
```

```
> class(V)
[1] "SpatialPolygonsDataFrame"
attr(,"package")
[1] "sp"
```

```
plot(V)
```



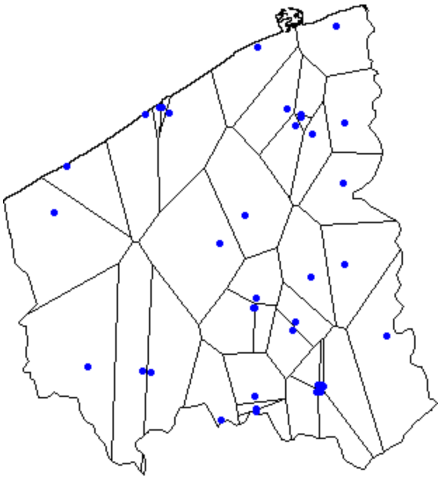
```
proj4string(wvlP)
```

```
> proj4string(wvlP)
[1] "+proj=longlat +datum=WGS84 +no_defs +ellps=WGS84 +towgs84=0,0,0"
```

```
proj4string(V) <- proj4string(wvlP)

final <- gIntersection(wvlP,V,byid=TRUE)
class(final)
```

```
> class(final)
[1] "SpatialPolygons"
attr(,"package")
[1] "sp"
```

```
plot(final)
points(coords,pch=20,col="blue”)
```



```
# convert voronoiPolys and shapefile Belgium into data frames to plot with ggplot2
Vf <- fortify(V)
class(Vf)
```

```
> class(Vf)
[1] "data.frame"
```

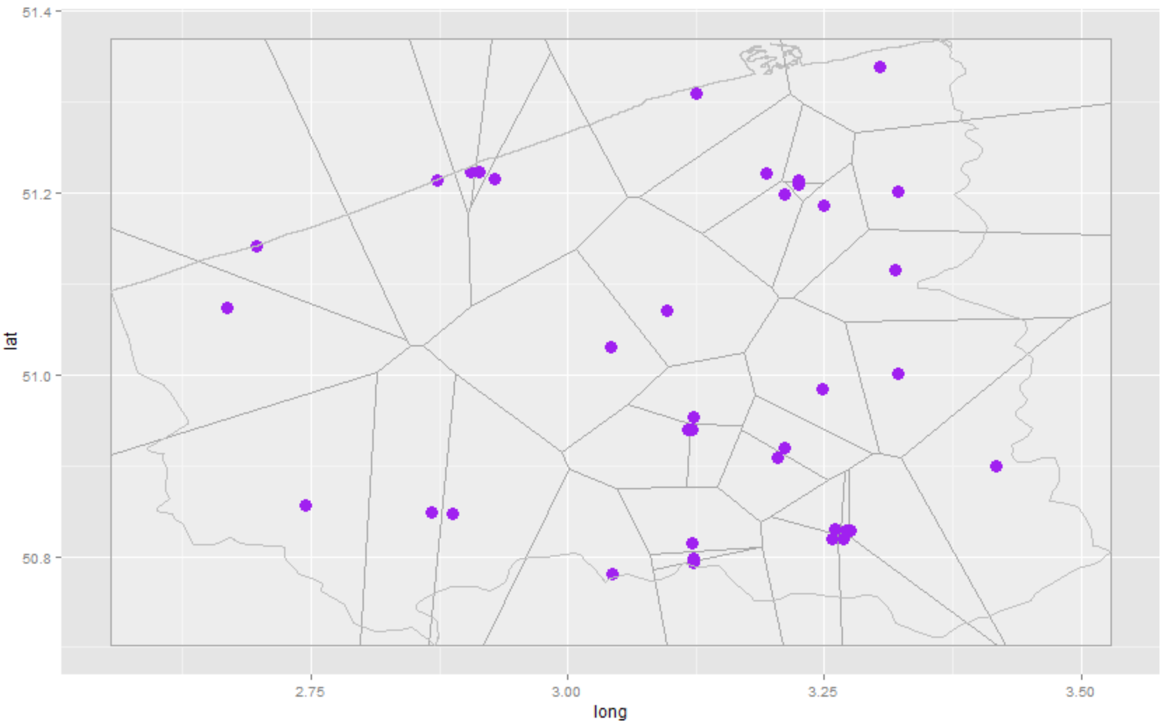
```
str(Vf)
```

```
> str(Vf)
'data.frame':       239 obs. of  7 variables:
 $ long : num  3.53 3.53 3.29 3.28 3.28 ...
 $ lat  : num  51.3 51.2 51.2 51.2 51.3 ...
 $ order: int  1 2 3 4 5 6 1 2 3 4 ...
 $ hole : logi  FALSE FALSE FALSE FALSE FALSE FALSE ...
 $ piece: Factor w/ 1 level "1": 1 1 1 1 1 1 1 1 1 1 ...
 $ group: Factor w/ 37 levels "1.1","2.1","3.1",..: 1 1 1 1 1 1 2 2 2 2 ...
 $ id   : chr  "1" "1" "1" "1" ...
```

```
wvlf <- fortify(wvlP)
str(wvlf)
```

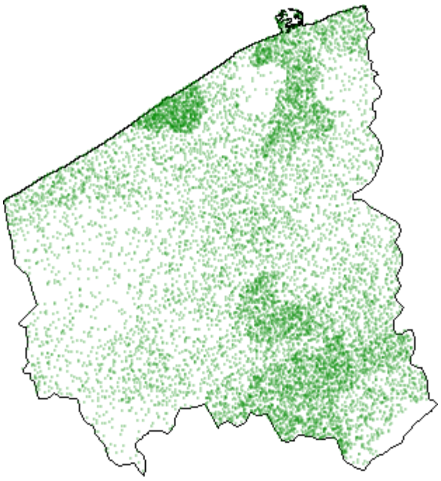
```
> str(wvlf)
'data.frame':       3115 obs. of  7 variables:
 $ long : num  2.81 2.79 2.79 2.78 2.76 ...
 $ lat  : num  50.7 50.7 50.7 50.7 50.8 ...
 $ order: int  1 2 3 4 5 6 7 8 9 10 ...
 $ hole : logi  FALSE FALSE FALSE FALSE FALSE FALSE ...
 $ piece: Factor w/ 2 levels "1","2": 1 1 1 1 1 1 1 1 1 1 ...
 $ group: Factor w/ 2 levels "West-Vlaanderen.1",..: 1 1 1 1 1 1 1 1 1 1 ...
 $ id   : chr  "West-Vlaanderen" "West-Vlaanderen" "West-Vlaanderen" "West-Vlaanderen" ...
```

```
# plot voronoi polygons (Vf) and hospitals (coords)
HospAndVor <- ggplot(data=Vf,aes(x=long,y=lat)) +
  geom_polygon(aes(group=group),colour="grey65",size=0.4,fill="white",alpha=0.3) +
  geom_point(data=coords,aes(x=long,y=lat),colour="purple",size=4) +
  geom_polygon(data=wvlf,aes(x=long,y=lat,group=group),color="grey",fill="NA")
HospAndVor
```



Next, we create a dot density map of the population in West Flanders.

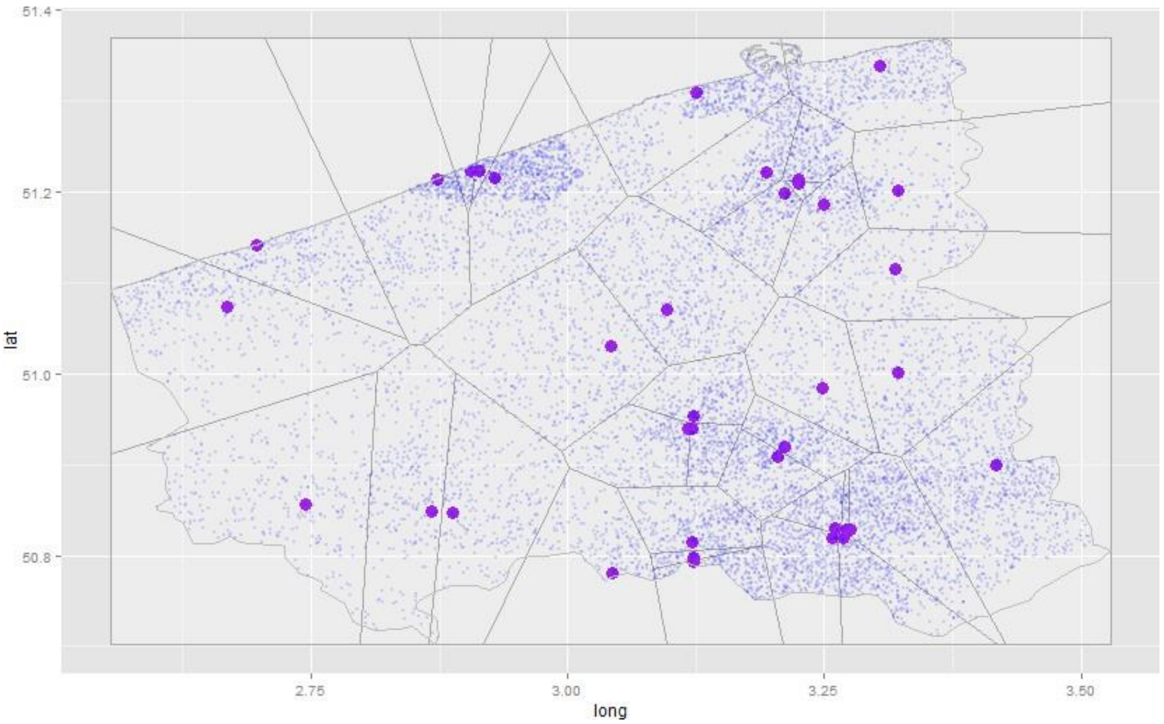
```
belgie <- readOGR(".","newbelgie")
wvl_pop <- belgie[belgie@data$NAME_2=="West-Vlaanderen", ]
# turn shapefile into dotmap
dots.popWVL <- dotsInPolys(wvl_pop,wvl_pop@data$TOTAALBEV/100)
plot(wvlP,lwd=0.05)
plot(dots.popWVL,add=TRUE,pch=19,cex=0.1,col="#00880030")
```



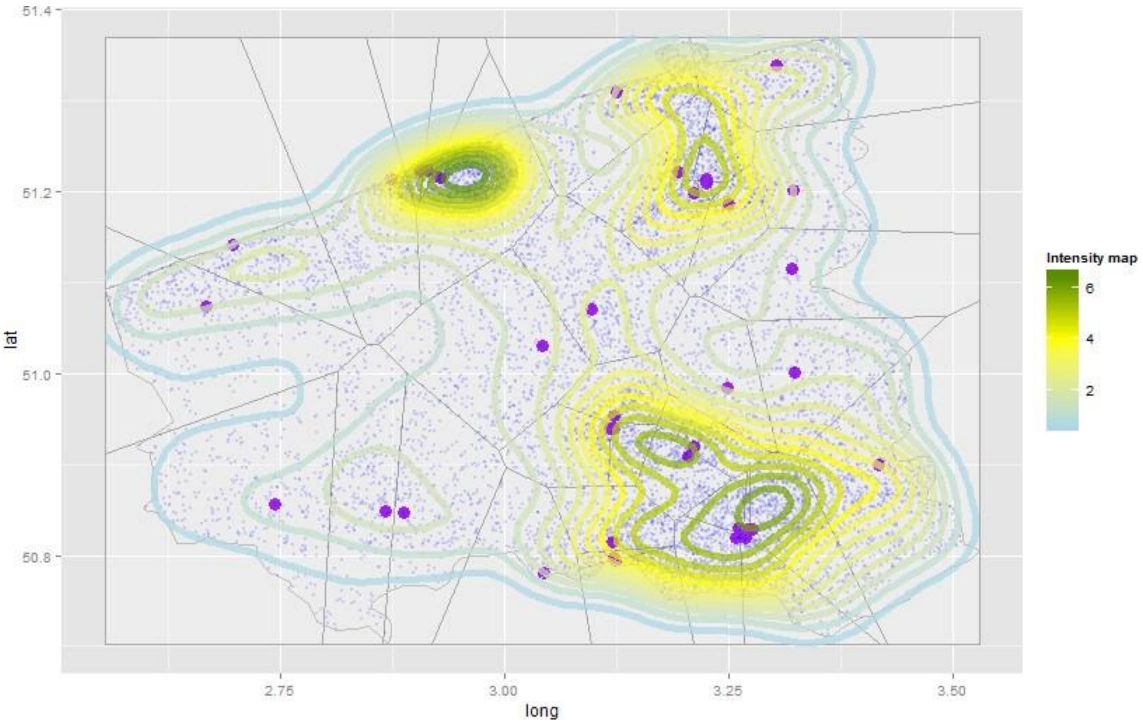
```
# extract dataframe from population dots
popWVL.df <- data.frame(coordinates(dots.popWVL)[,1:2])
str(popWVL.df)
```

```
> str(popWVL.df)
'data.frame':	11560 obs. of  2 variables:
 $ x: num  3.37 3.25 3.32 3.32 3.3 ...
 $ y: num  51.3 51.3 51.3 51.3 51.3 ...
```

```
VorAndDots <- HospAndVor + geom_point(data=popWVL.df,aes(x=x,y=y),
                                       size=0.5,color="blue",alpha=0.15)
VorAndDots
```



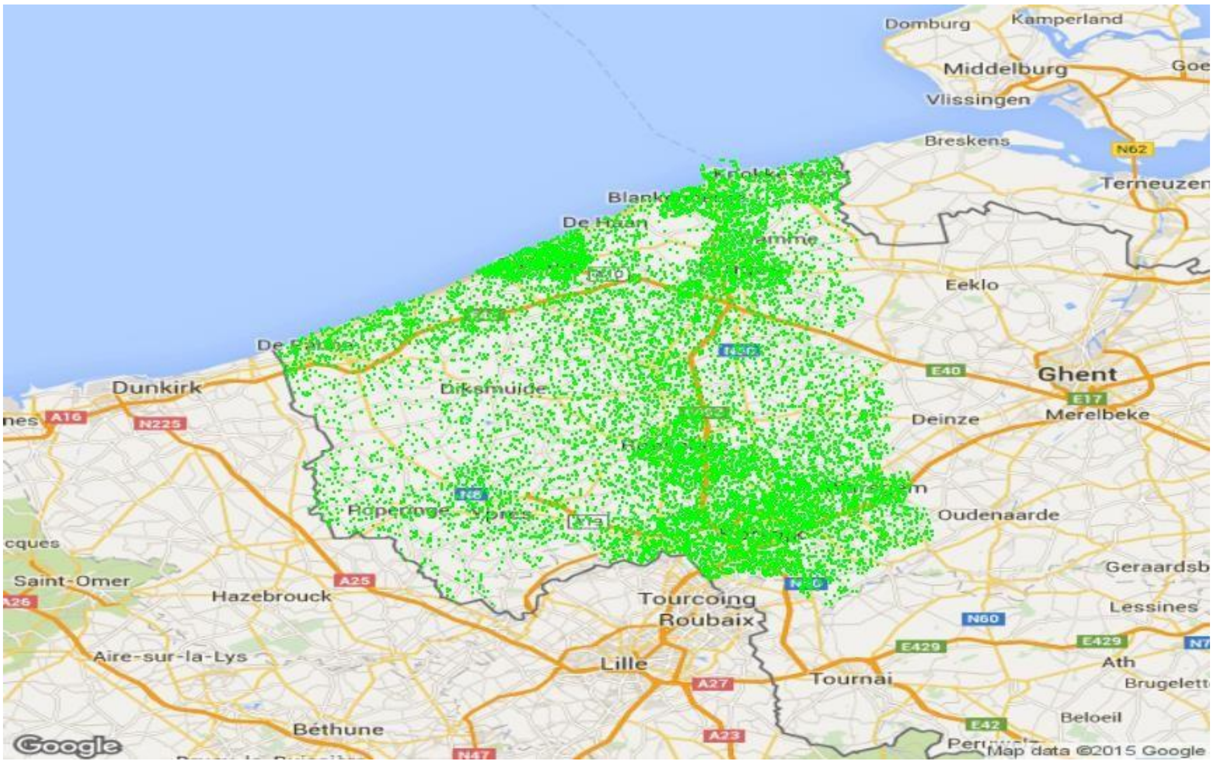
```
# contour plot of population density
contour <- VorAndDots +
            geom_density2d(aes(x=x,y=y,colour = ..level..),
                             size=2,alpha=0.7,data=popWVL.df) +
            scale_colour_gradient2("Intensity map",low="lightblue",
                                     mid="yellow",high="darkgreen",midpoint=4)
contour
```



```
# plotting population data on a Google maps background (library RgoogleMaps)
library(RgoogleMaps)
bb <- qbbox(lat=popWVL.df$y, lon=popWVL.df$x)
str(bb)
```

```
> str(bb)
List of 2
 $ latR: num [1:2] 50.7 51.4
 $ lonR: num [1:2] 2.55 3.51
```

```
map <- GetMap.bbox(bb$lonR, bb$latR)
PlotOnStaticMap(MyMap=map)
PlotOnStaticMap(map,lat=popWVL.df$y,lon=popWVL.df$x,zoom=NULL,cex=.3,
                 pch=19,col="green")
```

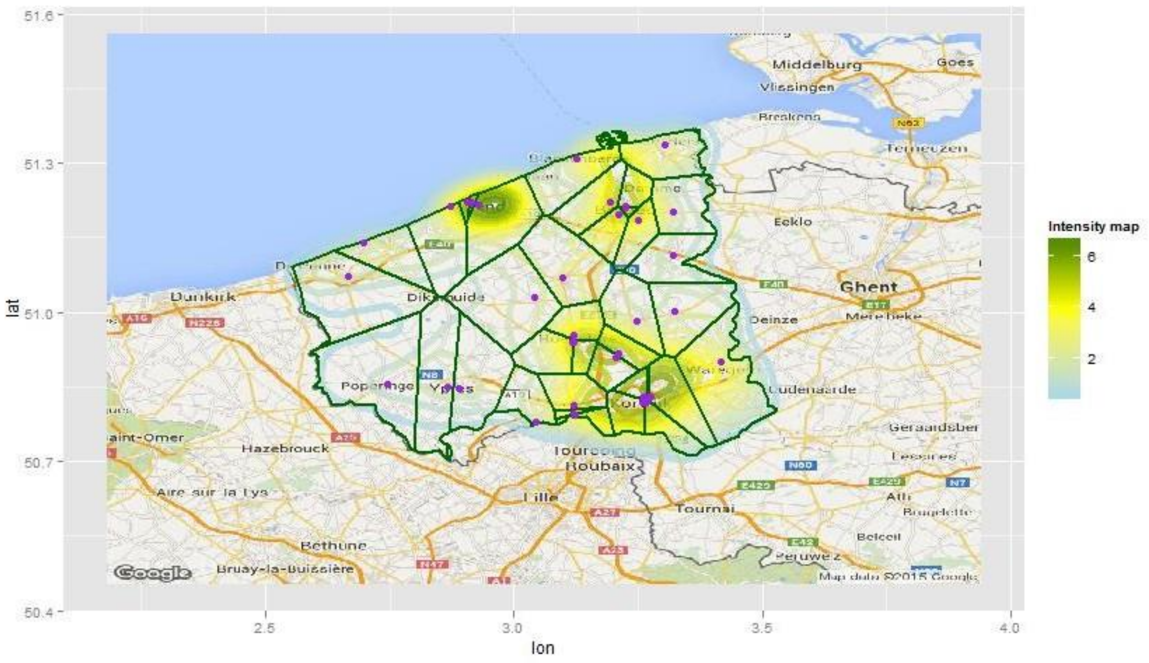


```
# plotting contour map on a Google maps background
centroids <- coordinates(wvlP)
centroids
```

```
> centroids
                    [,1]     [,2]
West-Vlaanderen 3.062247 51.01051
```

```
library(RgoogleMaps)
library(plyr)
library(jpeg)

LOCATION              <- 'west-vlaanderen'
CoordinateCenter      <- c(lat = 51.01051, lon = 3.062247)
COLOR_TYPE            <- c('color','bw')[1]
RGBCoefficients       <- c(0, 1, 0)
ZOOM_LEVEL            <- 9
MAP_TYPE              <- 'osm'
GOOGLE_MAP            <- 'GoogleMap.jpg'
NUMBER_OF_PIXELS      <- 640
```

```
GetMap(center   = CoordinateCenter[c('lat','lon')],
       size     = c(NUMBER_OF_PIXELS, NUMBER_OF_PIXELS),
       zoom     = ZOOM_LEVEL,
       format   = 'jpg',
       maptype  = MAP_TYPE,
       destfile = "test.jpg")


# load map
map <- readJPEG("test.jpg")
GoogleMapColorMatrix <- readJPEG("test.jpg")
GoogleMapColorMatrix <- apply(GoogleMapColorMatrix, 1:2, function(v) rgb(v[1],
v[2], v[3]))
GoogleMapInformationList <- list(lat= CoordinateCenter['lat'],
                                 lon= CoordinateCenter['lon'],
                                 zoom   = ZOOM_LEVEL,
                                 GoogleMapColorMatrix)
str(GoogleMapInformationList)
class(GoogleMapInformationList)
CoordinateIndex <- (-NUMBER_OF_PIXELS/2) : (NUMBER_OF_PIXELS/2 - 1)
CreateLatitudeValuesFromIndex <- function(x) XY2LatLon(GoogleMapInformationList, -
NUMBER_OF_PIXELS/2, x)[1]
CreateLongitudeValuesFromIndex <- function(y) XY2LatLon(GoogleMapInformationList,
y, -NUMBER_OF_PIXELS/2)[2]
Latitudes <- apply(data.frame(CoordinateIndex), 1, CreateLatitudeValuesFromIndex)
Longitudes <- apply(data.frame(CoordinateIndex), 1, CreateLongitudeValuesFromIndex)
Latitudes <- seq(range(Latitudes)[1], range(Latitudes)[2],
length.out=length(Latitudes))
Longitudes <- seq(range(Longitudes)[1], range(Longitudes)[2],
length.out=length(Longitudes))
library(reshape)
GoogleMapColorDataFrame        <- melt(GoogleMapColorMatrix)
names(GoogleMapColorDataFrame) <- c('x','y','fill')
GoogleMapColorDataFrame <- within(GoogleMapColorDataFrame,{
  x <- x - NUMBER_OF_PIXELS/2 - 1
  y <- y - NUMBER_OF_PIXELS/2 - 1
})
XYCoordinates                 <- expand.grid(x = CoordinateIndex, y =
CoordinateIndex)
LatitudesAndLongitudes        <- expand.grid(lat = rev(Latitudes), lon =
Longitudes)
PlotData.Map  <- data.frame(XYCoordinates, LatitudesAndLongitudes)
PlotData.Map  <- suppressMessages(join(PlotData.Map, GoogleMapColorDataFrame, type
= 'right'))
PlotData.Map          <- PlotData.Map[,c('lon','lat','fill')]
LatitudeRange         <- range(PlotData.Map$lat)
LongitudeRange        <- range(PlotData.Map$lon)
str(PlotData.Map)
theme_nothing <- function (base_size = 12){
  structure(list(axis.line            = theme_blank(),
                 axis.text.x          = theme_blank(), axis.text.y = theme_blank(),
                 axis.ticks           = theme_blank(),
                 axis.title.x         = theme_blank(), axis.title.y =
theme_blank(),
                 axis.ticks.length    = unit(0, "lines"), axis.ticks.margin =
unit(0, "lines"),
                 legend.position      = "none",
                 panel.background     = theme_rect(fill = 'white'),
                 panel.border         = theme_blank(),
                 panel.grid.major     = theme_blank(), panel.grid.minor =
theme_blank(),
                 panel.margin         = unit(0, "lines"),
                 plot.background      = theme_rect(colour = 'white'),
                 plot.title           = theme_text(size = base_size * 1.2),
                 plot.margin          = unit(c(-1, -1, -1.5, -1.5), "lines")),
            class = "options")
}
vplayout <- function(x, y) {
  viewport(layout.pos.row = x, layout.pos.col = y)
}

# plot
# convert shapefile of voronoi polygons in West Flanders to a data frame

ff <- fortify(final)

ggplot() +
  geom_tile(aes(x=lon,y=lat,fill=fill),data=PlotData.Map) +
  geom_density2d(aes(x=x,y=y,colour = ..level..),size=2,alpha=0.7,data=popWVL.df) +
  scale_colour_gradient2("Intensity map",low="lightblue",mid="yellow",
                        high="darkgreen",midpoint=4) +
                        scale_fill_identity() +
  geom_polygon(data=ff,aes(x=long,y=lat,group=group),fill="grey",
               size=1,color="darkgreen",alpha=0) +
  geom_point(data=coords,aes(x=long,y=lat),color="purple",cex=2.5) +
  coord_equal()
```



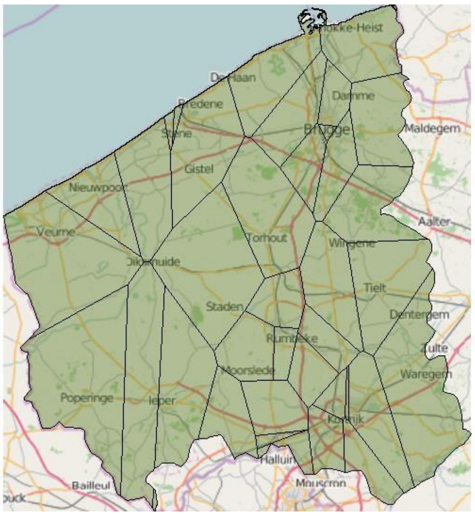
```
# plot voronoi polygons and hospital locations with osm

library(OpenStreetMap)
map.osm <- openmap(c(max(wvlf$lat,na.rm=TRUE),min(wvlf$lon,na.rm=TRUE)),
                   c(min(wvlf$lat,na.rm=TRUE),max(wvlf$lon,na.rm=TRUE)),type="osm")
plot(map.osm)
```

Knokke-Heist
De Haan
Damme
Bredene
Brugge
Maldegem
Stene
Gistel
Nieuwpoort
Aalter
Veurne
Torhout
Wingene
Diksmuide
Tielt
Staden
Dentergem
Rumbeke
Zulte
Moorslede
Waregem
Poperinge
Ieper
Kortrijk
Halluin
Bailleul
Mouscron
puck

```
final <- spTransform(final,osm())
class(final)
```

```
> class(final)
[1] "SpatialPolygons"
attr(,"package")
[1] "sp"
```

```
#alpha definition
add.alpha <- function(col, alpha=1){
  if(missing(col))
    stop("Please provide a vector of colours.")
  apply(sapply(col, col2rgb)/255, 2,
        function(x)
          rgb(x[1], x[2], x[3], alpha=alpha))
}
mycol=add.alpha("#507415",alpha=.4)
plot(final,add=T,col=mycol)
```



```
# add the points (hospital locations to the map)
class(points)
```

```
> class(points)
[1] "SpatialPoints"
attr(,"package")
[1] "sp"
```

```
proj4string(points)
```

```
> proj4string(points)
[1] NA
```

```
proj4string(points) <- CRS("+proj=longlat +ellps=WGS84")
points <- spTransform(points,CRS("+proj=merc +a=6378137 +b=6378137 +lat_ts=0.0
                       +lon_0=0.0 +x_0=0.0 +y_0=0 +k=1.0 +units=
                       m +nadgrids=@null +no_defs"))
```

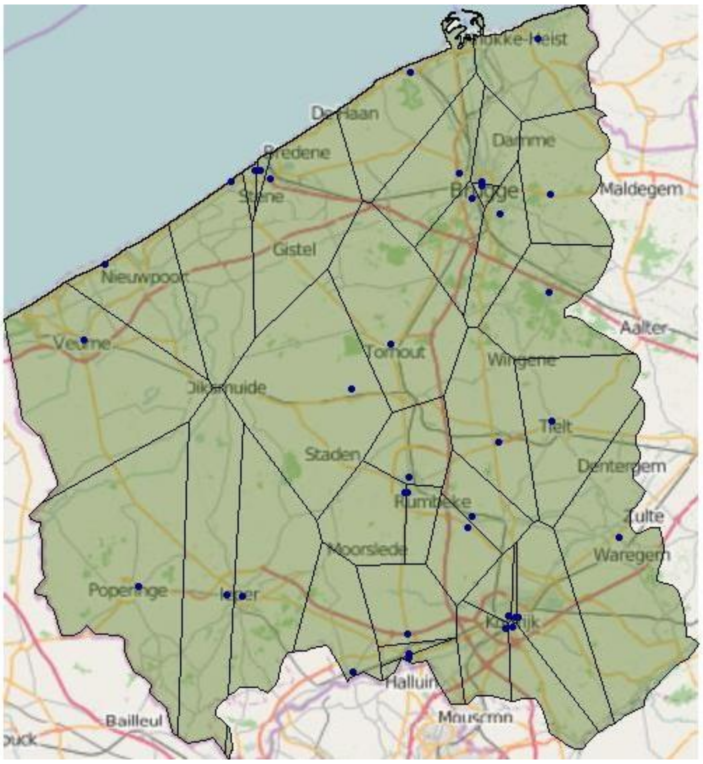
```
proj4string(points)
```

```
> proj4string(points)
[1] "+proj=merc +a=6378137 +b=6378137 +lat_ts=0.0 +lon_0=0.0 +x_0=0.0 +y_0=0 +k=1.0 +units=m +
nadgrids=@null +no_defs"
```

```
proj4string(final)
```

```
> proj4string(final)
[1] "+proj=merc +a=6378137 +b=6378137 +lat_ts=0.0 +lon_0=0.0 +x_0=0.0 +y_0=0 +k=1.0 +units=m +
nadgrids=@null +no_defs"
```

```
points(points@coords,col="darkblue",pch=19,cex=0.7)
```



```
# aggregate dot density data
# count the number of dots within each of the voronoi polygons for West Flanders

pop.wf <- data.frame(coordinates(dots.popWVL)[,1:2])
str(pop.wf)
```

```
> str(pop.wf)
'data.frame':	11560 obs. of  2 variables:
 $ x: num  3.29 3.3 3.37 3.37 3.34 ...
 $ y: num  51.3 51.3 51.3 51.3 51.3 ...
```

```
coordinates(pop.wf) <- ~ x + y
class(pop.wf)
```

```
> class(pop.wf)
[1] "SpatialPoints"
attr(,"package")
[1] "sp"
```

```
proj4string(pop.wf) <- proj
proj4string(pop.wf)
```

```
> proj4string(pop.wf)
[1] "+proj=longlat +datum=WGS84 +no_defs +ellps=WGS84 +towgs84=0,0,0"
```

```
proj4string(V) <- proj
proj4string(V)
```

```
> proj4string(V)
[1] "+proj=longlat +datum=WGS84 +no_defs +ellps=WGS84 +towgs84=0,0,0"
```

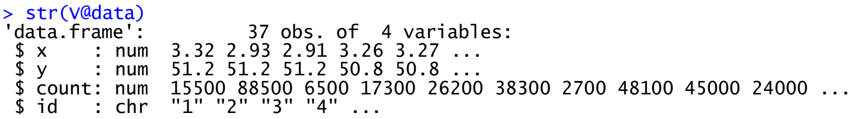
```
# count the number of dots in each polygon in the voronoi diagram
count <- sapply(over(V,pop.wf,returnList=TRUE),length)
count
```

```
> count
  1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27
28  29  30  31
 197 859  79 167 287 360  32 477 454 230 264 384 369  63 624 188 370 199 251  49 899 268 246 224 267 295  21
294 322 315 370
 32  33  34  35  36  37
367 177 303 546 272 471
```

```
# add the results of the count to the voronoi polygons
V <- spCbind(V, as.data.frame(count))
str(V@data)
```

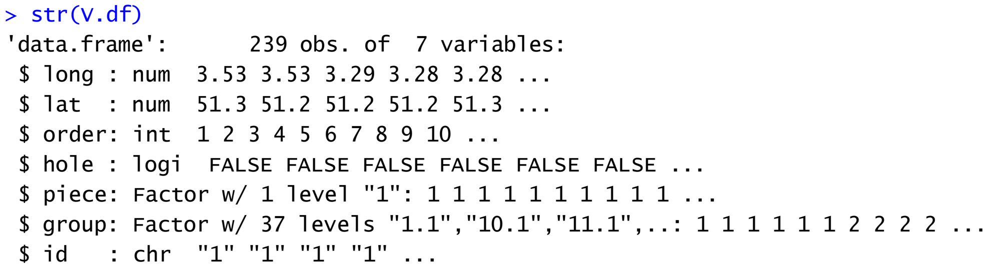
```
> str(V@data)
'data.frame':	37 obs. of  3 variables:
 $ x    : num  3.32 2.93 2.91 3.26 3.27 ...
 $ y    : num  51.2 51.2 51.2 50.8 50.8 ...
 $ count: int  155 885 65 173 262 383 27 481 450 240 ...
```

```
V@data$count <- V@data$count * 100

# convert V (SPDF) into a data frame to plot with ggplot2
V@data$id <- rownames(V@data)
str(V@data)
```



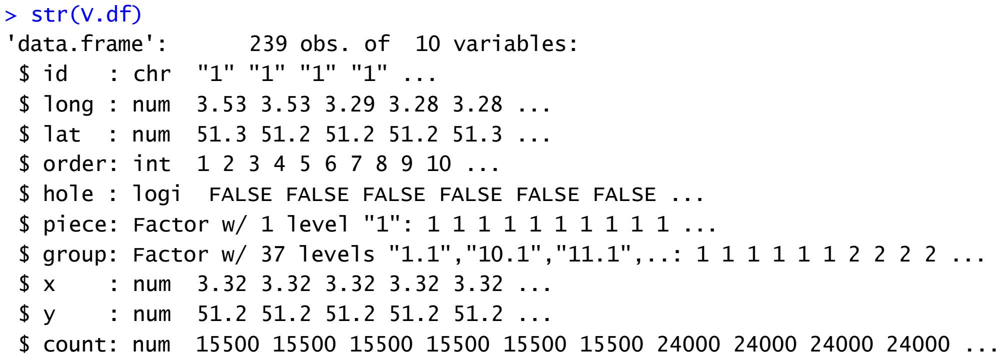
```
> str(V@data)
'data.frame':	37 obs. of  4 variables:
 $ x    : num  3.32 2.93 2.91 3.26 3.27 ...
 $ y    : num  51.2 51.2 51.2 50.8 50.8 ...
 $ count: num  15500 88500 6500 17300 26200 38300 2700 48100 45000 24000 ...
 $ id   : chr  "1" "2" "3" "4" ...
```

```
V.df <- fortify(V,region="id")
str(V.df)
```



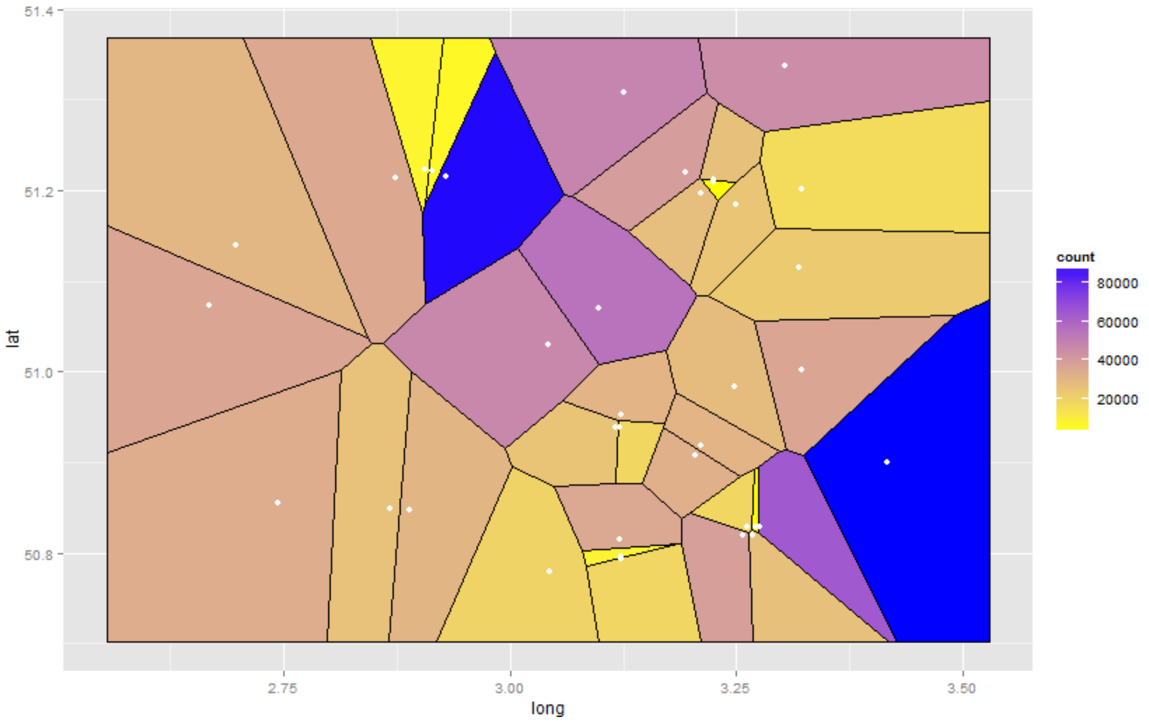
```
> str(V.df)
'data.frame':	239 obs. of  7 variables:
 $ long : num  3.53 3.53 3.29 3.28 3.28 ...
 $ lat  : num  51.3 51.2 51.2 51.2 51.3 ...
 $ order: int  1 2 3 4 5 6 7 8 9 10 ...
 $ hole : logi  FALSE FALSE FALSE FALSE FALSE FALSE ...
 $ piece: Factor w/ 1 level "1": 1 1 1 1 1 1 1 1 1 1 ...
 $ group: Factor w/ 37 levels "1.1","10.1","11.1",..: 1 1 1 1 1 1 2 2 2 2 ...
 $ id   : chr  "1" "1" "1" "1" ...
```

```
V.df <- merge(V.df,V@data,by="id")
str(V.df)
```

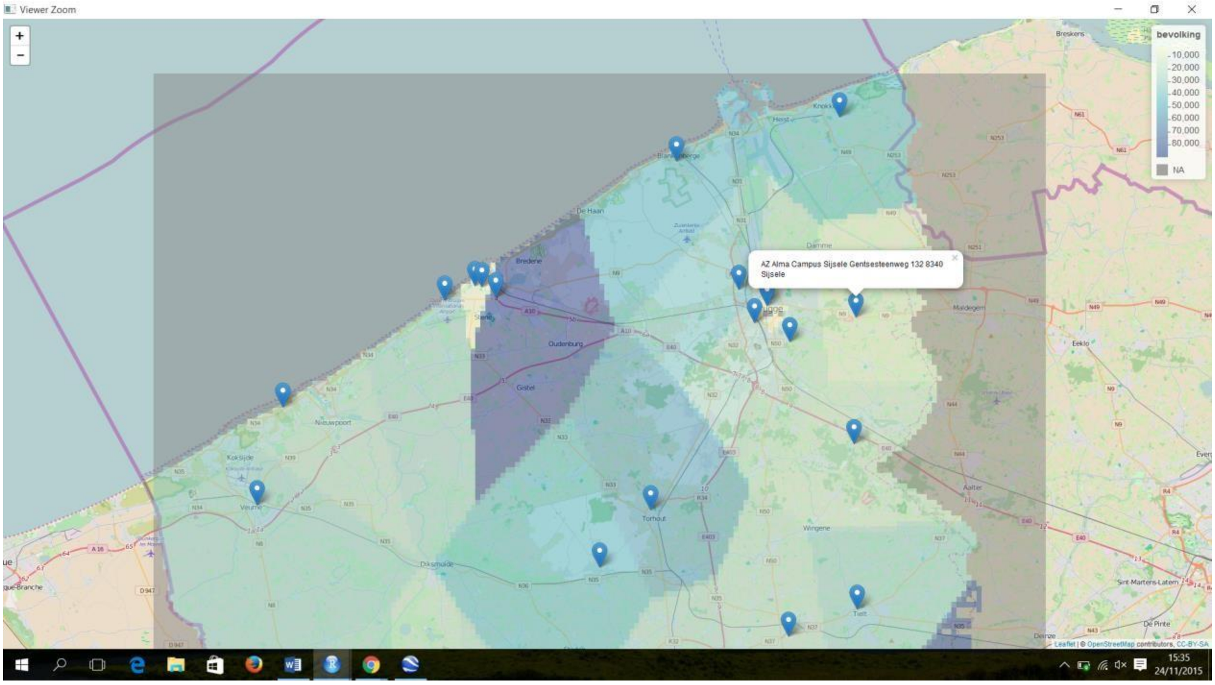


```
> str(V.df)
'data.frame':	239 obs. of  10 variables:
 $ id   : chr  "1" "1" "1" "1" ...
 $ long : num  3.53 3.53 3.29 3.28 3.28 ...
 $ lat  : num  51.3 51.2 51.2 51.2 51.3 ...
 $ order: int  1 2 3 4 5 6 7 8 9 10 ...
 $ hole : logi  FALSE FALSE FALSE FALSE FALSE FALSE ...
 $ piece: Factor w/ 1 level "1": 1 1 1 1 1 1 1 1 1 1 ...
 $ group: Factor w/ 37 levels "1.1","10.1","11.1",..: 1 1 1 1 1 1 2 2 2 2 ...
 $ x    : num  3.32 3.32 3.32 3.32 3.32 ...
 $ y    : num  51.2 51.2 51.2 51.2 51.2 ...
 $ count: num  15500 15500 15500 15500 15500 15500 24000 24000 24000 24000 ...
```

```
map <- ggplot(V.df,aes(x=long,y=lat)) +
  geom_polygon(aes(group=group,fill=count),colour="black",size=0.1) +
  scale_fill_continuous(low="yellow",high="blue") +
  geom_point(data=coords,aes(x=long,y=lat),colour="white",size=2) +
  coord_equal()
map
```



```
# mapping SpatialPolygonsDataFrame "V" with library leaflet
# convert “V” to raster(r3), cropped tot he extent of West Flanders

pal <- colorNumeric(c("#FFFFCC","#41B6C4","#0C2C84"), values(r3))
leaflet() %>% addTiles() %>%
  addRasterImage(r3, colors = pal, opacity = 0.45) %>%
  addMarkers(data=hospitals,~long, ~lat, popup = ~address) %>%
  addLegend(pal=pal, values = values(r3),
            title = "bevolking")
```



③ Voronoi intensity mapping

Source : “Voronoi intensity mapping with R and d3 : Part 1”,
(*http://www.jonzelner.net/2015/02/26/voronoi*)

R script : “c:/R/Rdata/voronoi.R”

In the case of the visualization of crime, for example, we can rely on markers to highlight the location of crime types. This can lead to overplotting in areas that have experienced more than their fair share of crime. A Voronoi intensity map might be the right way to do justice to these data. The basic idea behind a Voronoi diagram is that it is a set of polygons representing the set of points closest to a set of pre-defined points. Applied to crime, each Voronoi cell represents the set of points closer to one of the crimes in the data set than any other. The intuition behind Voronoi intensity mapping is that the closer points are together in a given area, the smaller the Voronoi cells in that area will be.

For this exercise, we use a dataset of the location of homicides in Philadelphia, US.

```
setwd("c:/R/Rdata")
data <- read.csv("PPD_Crime_Incidents_2012-2014.csv",header=T,sep=",")
str(data)
```

```
> str(data)
'data.frame':            569426 obs. of  8 variables:
 $ ï..District           : int  23 19 16 6 24 18 26 22 25 3 ...
 $ PSA                   : Factor w/ 6 levels "1","2","3","4",..: 2 2 1 1 2 1 6 2 2 2 ...
 $ Dispatch.Date.Time    : Factor w/ 446678 levels "01/01/2012 01:03:00 AM",..: 41045 11217 179381 40897 124388 59363 411631
51509 35601 51513 ...
 $ DC.Number             : num  2.01e+11 2.01e+11 2.01e+11 2.01e+11 2.01e+11 ...
 $ Location.Block        : Factor w/ 55578 levels "/C BAGGAGE","` 59TH ST / CATHARINE ST",..: 13219 3728 20498 10718 12778 37
376 12209 9614 15974 16709 ...
 $ UCR.Code              : int  1800 1800 2600 600 1800 800 200 1700 200 200 ...
 $ General.Crime.Category: Factor w/ 33 levels "Aggravated Assault Firearm",..: 18 18 3 30 18 20 24 21 24 24 ...
 $ Coordinates           : Factor w/ 235950 levels "","(0, 0)","(39.8741308038022, -75.2179316185477)",..: 88478 79215 68389
45503 109872 41876 98752 101461 117221 1648 ...
```

```
data$Coordinates <- as.character(data$Coordinates)
data$Y <- sapply(strsplit(as.character(data$Coordinates), ","), "[", 1)
data$X <- sapply(strsplit(as.character(data$Coordinates), ","), "[", 2)
data$Y <- sapply(strsplit(data$Y, split='(', fixed=TRUE), function(x) (x[2]))
data$X <- sapply(strsplit(data$X, split=')', fixed=TRUE), function(x) (x[1]))
data$Y <- as.numeric(data$Y)
data$X <- as.numeric(data$X)
data$datetime <- as.POSIXct(data$Dispatch.Date.Time, format="%m/%d/%Y %I:%M:%S %p")
library(lubridate)
data$year <- year(data$datetime)
data <- data[(data$General.Crime.Category == "Homicide - Criminal" |
              data$General.Crime.Category == "Homicide -  Gross Negligence" |
              data$General.Crime.Category == "Homicide - Justifiable"), ]
str(data)
```

```
> str(data)
'data.frame':            871 obs. of  12 variables:
 $ ï..District           : int  1 1 1 1 1 1 1 1 1 1 ...
 $ PSA                   : Factor w/ 6 levels "1","2","3","4",..: 1 1 1 1 2 1 1 1 1 1 ...
 $ Dispatch.Date.Time    : Factor w/ 446678 levels "01/01/2012 01:03:00 AM",..: 9804 87561 172960 184898 240742 260831 288019
309077 363245 413184 ...
 $ DC.Number             : num  2.01e+11 2.01e+11 2.01e+11 2.01e+11 2.01e+11 ...
 $ Location.Block        : Factor w/ 55578 levels "/C BAGGAGE","` 59TH ST / CATHARINE ST",..: 50703 8930 13172 15011 20201 50
928 13204 16104 9573 11643 ...
 $ UCR.Code              : int  100 100 100 100 100 100 100 100 100 100 ...
 $ General.Crime.Category: Factor w/ 33 levels "Aggravated Assault Firearm",..: 13 13 13 13 13 13 13 13 13 13 ...
 $ Coordinates           : chr  "(39.9280558081946, -75.1787079375114)" "(39.9292819308307, -75.1819903034061)" "(39.92383269
53744, -75.1842793805316)" "(39.9217252579188, -75.1942981211138)" ...
 $ Y                     : num  39.9 39.9 39.9 39.9 39.9 ...
 $ X                     : num  -75.2 -75.2 -75.2 -75.2 -75.2 ...
 $ datetime              : POSIXct, format: "2012-01-10 16:00:00" "2012-03-20 01:30:00" "2012-05-27 05:54:00" "2012-06-05 11:
21:00" ...

 $ year                  : num  2012 2012 2012 2012 2012 ...
```

```
data <- subset(data,select=c(X,Y))
data <- na.omit(data)
names(data) <- c("lon","lat")
str(data)
```

```
> str(data)
'data.frame':	869 obs. of  2 variables:
 $ lon: num  -75.2 -75.2 -75.2 -75.2 -75.2 ...
 $ lat: num  39.9 39.9 39.9 39.9 39.9 ...
 - attr(*, "na.action")=Class 'omit'  Named int [1:2] 38 389
  .. ..- attr(*, "names")= chr [1:2] "35715" "273937"
```

```
head(data,20)
```

```
> head(data,20)
            lon      lat
120   -75.17871 39.92806
919   -75.18199 39.92928
1782  -75.18428 39.92383
1896  -75.19430 39.92173
2527  -75.19158 39.90670
2706  -75.18388 39.92872
2968  -75.18341 39.92611
3167  -75.19167 39.92361
3677  -75.17777 39.92798
4115  -75.18310 39.92696
4331  -75.18672 39.91879
6515  -75.10037 40.04559
7573  -75.08769 40.04680
9863  -75.11230 40.04186
10376 -75.08505 40.04370
11513 -75.07637 40.04328
13413 -75.10086 40.03982
15299 -75.07881 40.05525
15688 -75.16724 39.93156
15690 -75.16724 39.93156
```

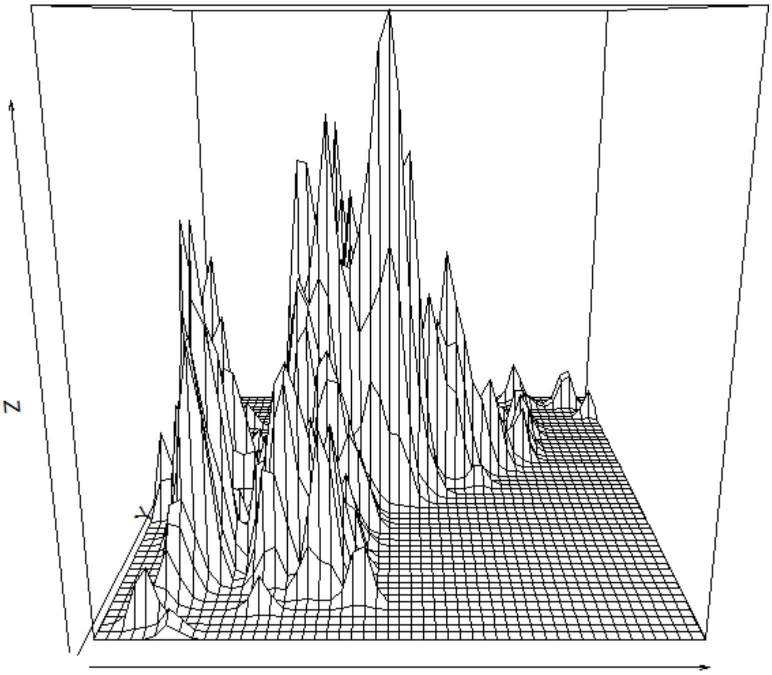
```
# density estimation with package KernSmooth

library(KernSmooth)
# compute density with cross-validation techniques to get optimal bandwidths
kde2d <- bkde2D(data, bandwidth=c(bw.ucv(data[,1]),bw.ucv(data[,2])))
class(kde2d)
```

```
> class(kde2d)
[1] "list"
```

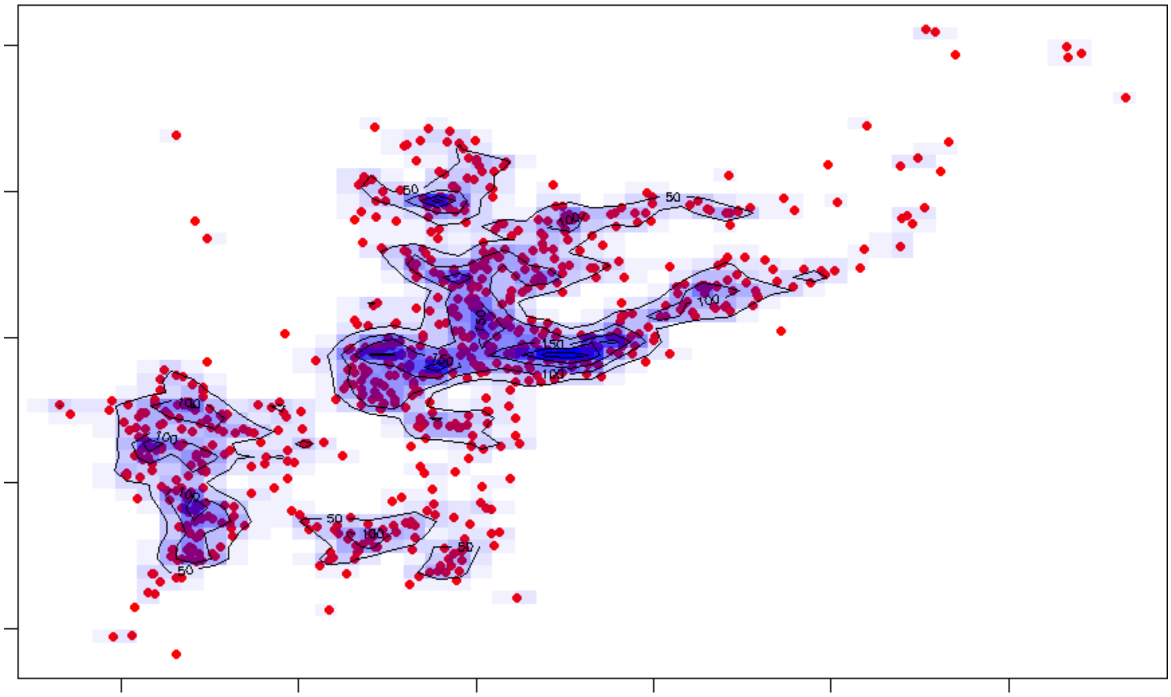
```
str(kde2d)
```

```
> str(kde2d)
List of 3
 $ x1  : num [1:51] -75.3 -75.3 -75.3 -75.3 -75.2 ...
 $ x2  : num [1:51] 39.9 39.9 39.9 39.9 39.9 ...
 $ fhat: num [1:51, 1:51] 0 0.000117 0.003742 0.013732 0.083049 ...
```

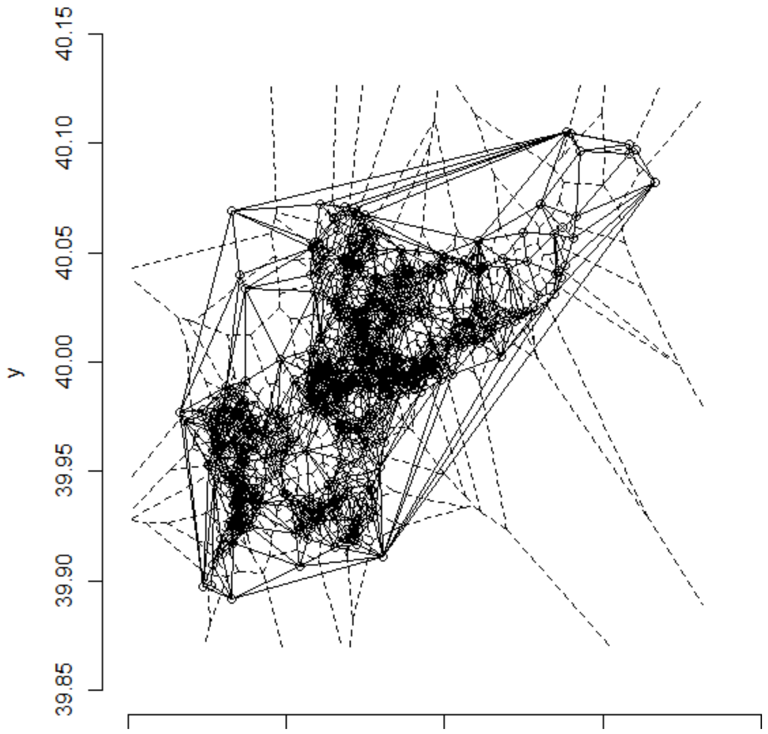
```
z <- kde2d$fhat
persp(z)
```



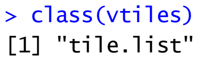
```
clrs <- colorRampPalette(c(rgb(0,0,1,0),rgb(0,0,1,1)),alpha=TRUE)(20)

plot(data,col="red",pch=19)
image(x=kde2d$x1,y=kde2d$x2,z=kde2d$fhat,col=clrs,add=TRUE)
contour(x=kde2d$x1,y=kde2d$x2,z=kde2d$fhat,add=TRUE)
```



```
library(deldir)
voronoi <- deldir(data$lon,data$lat)
plot(voronoi,xlim=c(-75.3,-74.9),ylim=c(39.85,40.14))
```

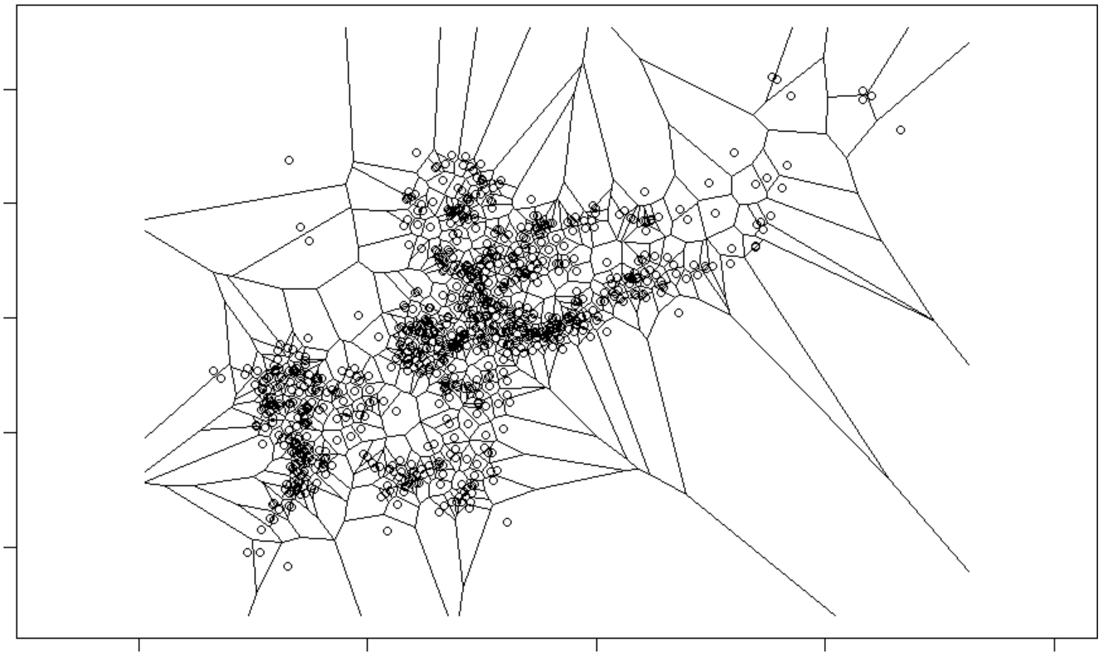


```
vtiles <- tile.list(voronoi)
class(vtiles)
```



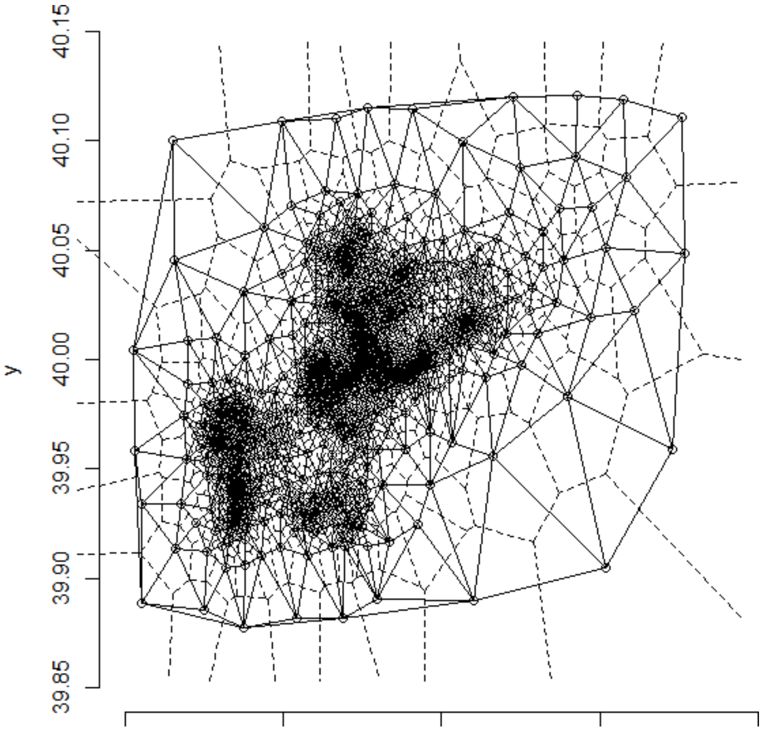
```
> class(vtiles)
[1] "tile.list"
```

```
plot(vtiles)
```

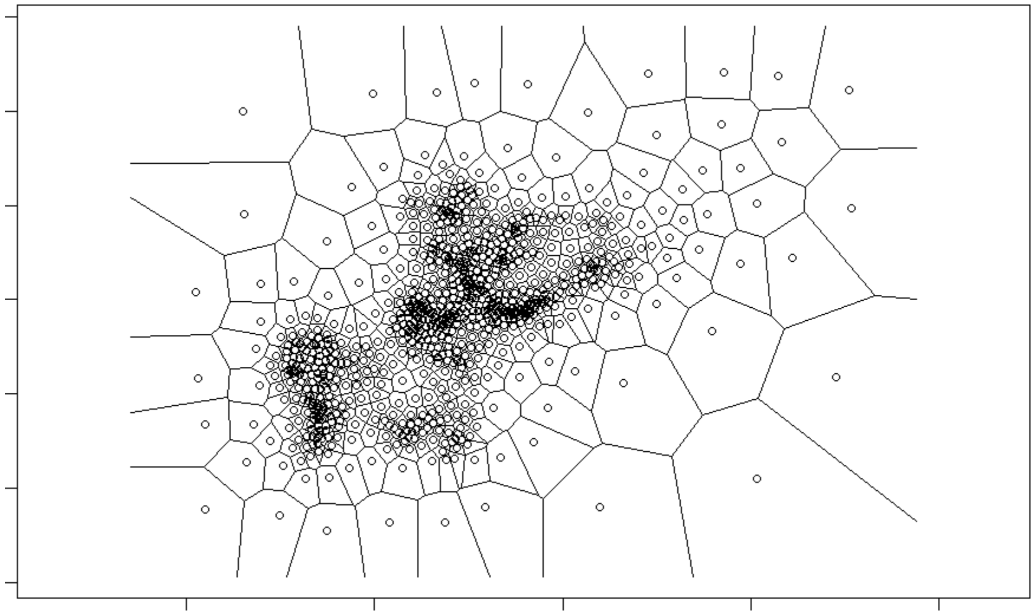


There are some very small polygons in the highest density areas and very irregular ones in the low-density areas. One way to deal with this is to do a little bit of regularization using centroidal voronoi estimation in which we take the output of a single iteration of the voronoi tessalation and make the centroids of the voronoi cells the input sites to the next iteration.

```
for (i in 1:2) {
  voronoi <- deldir(tile.centroids(vtiles))
  vtiles <- tile.list(voronoi)
}
plot(voronoi,xlim=c(-75.3,-74.9),ylim=c(39.85,40.14))
```



```
plot(vtiles)
```



This gives us little less of the extreme variability in polygon size, while still maintaining the large differences in density across space.

What we further need to do is to clip the polygons tot he borders of the city. This will ensure that the areas of the polygons in our intensity map match the portions of the city that they're meant to cover, rather than the arbitrarily large polygons at the edges of the diagram.

We can enclose the voronoi polygons within the geographic borders of Philadelphia with the help of the ***gIntersection function*** (***package rgeos***).

To use this function we need to have both the voronoi polygons and the geographic borders as SpatialPolyDataFrame objects.

First, we convert the neighborhoods of the Philadelphia shapefile into a single polygon.

```
library(rgdal)
library(maptools)
phila <- readOGR(".","Neighborhoods")
class(phila)
```

```
> class(phila)
[1] "SpatialPolygonsDataFrame"
attr(,"package")
[1] "sp"
```

```
plot(phila)
```

```
regs <- c("Philadelphia")
phila@data$NAME_2 <- "Philadelphia"
sp2 <- phila@data$NAME_2
sp2[which(!phila@data$NAME_2 %in% regs)] <- NA
PH <- unionSpatialPolygons(phila,sp2)
plot(PH)
```

To convert the deldir object to a SpatialPolgygonsDataFrame we use the following function :

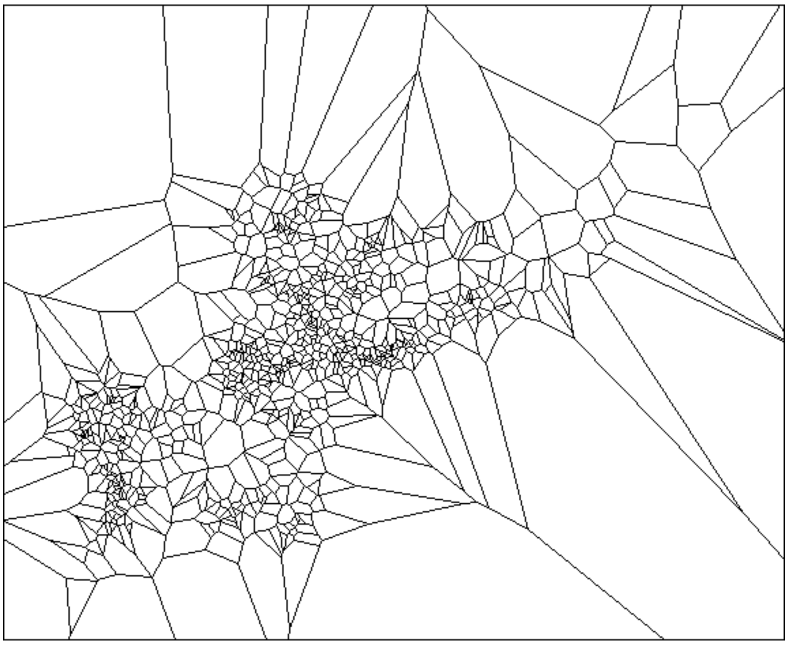
```
voronoipolygons <- function(x, poly) {
  require(deldir)
  if (.hasSlot(x, 'coords')) {
    crds <- x@coords
  } else crds <- x
  bb = bbox(poly)
  rw = as.numeric(t(bbox(poly)))
  z <- deldir(crds[,1], crds[,2],rw=rw)
  w <- tile.list(z)
  polys <- vector(mode='list', length=length(w))
  require(sp)
  for (i in seq(along=polys)) {
    pcrds <- cbind(w[[i]]$x, w[[i]]$y)
    pcrds <- rbind(pcrds, pcrds[1,])
    polys[[i]] <- Polygons(list(Polygon(pcrds)), ID=as.character(i))
  }
  SP <- SpatialPolygons(polys)

  SpatialPolygonsDataFrame(
    SP, data.frame(x=crds[,1], y=crds[,2],
                   row.names=sapply(slot(SP, 'polygons'),
                                    function(x) slot(x, 'ID'))))
}

q <- voronoipolygons(data,PH)
```

```
Error in data.frame(x = crds[, 1], y = crds[, 2], row.names = sapply(slot(SP, : row names
supplied are of the wrong length
```

This is error is due to duplicate coordinates in the data frame, so we need to deduplicate :

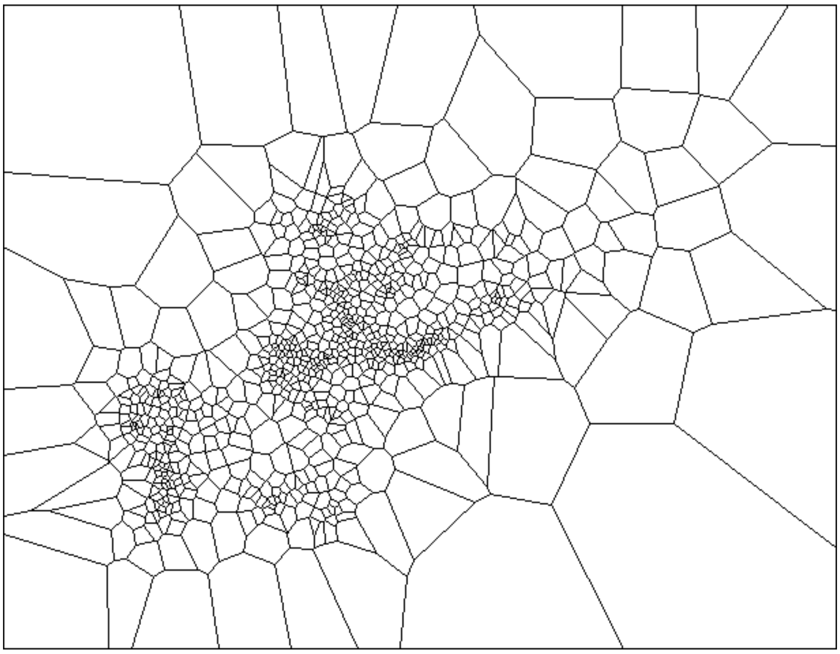
```
data <- data[!duplicated(data$lat), ]
data <- data[!duplicated(data$lon), ]
```

```
q <- voronoipolygons(data,PH)
plot(q)
```



```
# adjusted function to perform centroidal voronoi estimatrion
voronoipolygons2 <- function(x, poly) {
  require(deldir)
  if (.hasSlot(x, 'coords')) {
    crds <- x@coords
  } else crds <- x
  bb = bbox(poly)
  rw = as.numeric(t(bbox(poly)))
  z <- deldir(crds[,1], crds[,2],rw=rw)
  w <- tile.list(z)
    for (i in 1 : 2) {
    voronoi <- deldir(tile.centroids(w))
    vtiles <- tile.list(voronoi)
  }
  polys <- vector(mode='list', length=length(vtiles))
  require(sp)
  for (i in seq(along=polys)) {
    pcrds <- cbind(vtiles[[i]]$x, vtiles[[i]]$y)
    pcrds <- rbind(pcrds, pcrds[1,])
    polys[[i]] <- Polygons(list(Polygon(pcrds)), ID=as.character(i))
  }
  SP <- SpatialPolygons(polys)

  SpatialPolygonsDataFrame(
    SP, data.frame(x=crds[,1], y=crds[,2],
                   row.names=sapply(slot(SP, 'polygons'),
                                    function(x) slot(x, 'ID'))))
}
```

```
q2 <- voronoipolygons2(data,PH)
plot(q2)
```



```
proj4string(PH)
```

```
> proj4string(PH)
[1] "+proj=longlat +datum=WGS84 +no_defs +ellps=WGS84 +towgs84=0,0,0"
```

```
proj4string(q2)
```

```
> proj4string(q2)
[1] NA
```

```
proj4string(q2) <-proj4string(PH)

library(rgeos)
final <- gIntersection(PH, q2, byid=TRUE)
class(final)
```

```
> class(final)
[1] "SpatialPolygons"
attr(,"package")
[1] "sp"
```

```
plot(final)
image(x=kde2d$x1,y=kde2d$x2,z=kde2d$fhat,col=clrs,add=TRUE)
```