EBSpat an R package devoted to simulation and estimation around nearest-neighbour type Gibbs point processes

R. Drouilhet

LJK Grenoble

Plan

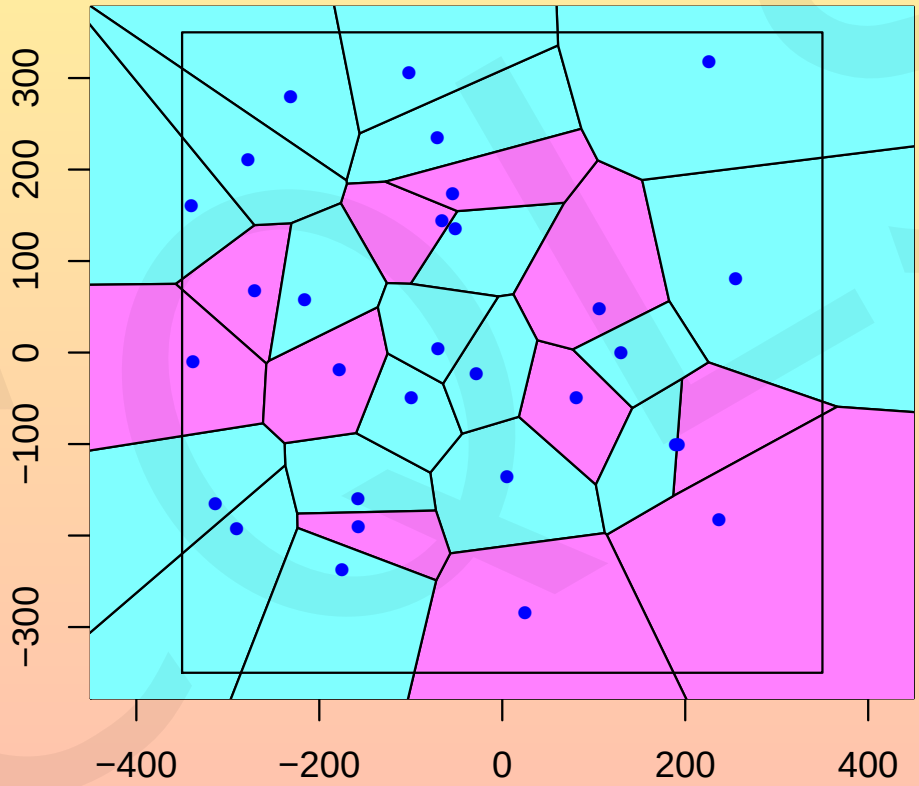
Motivation

- After a long period of theoretical research on **nearest-neighbour Gibbs point processes** around the main topics:
  - Existence of stationary Gibbs states, Phase transition, Percolation
  - Statistical properties of the pseudo-likelihood and Takacs-Fiksel estimators

  with as main collaborators (in chronological order):
  - **E**tienne **B**ertin (as in **EB**Spat) and Jean-Michel Billiot
  - Jean-François Coeurjolly
  - David Dereudre and Hans-Otto Georgii
  - Frederic Lavancier
- the need to make our results available for **practical applications**!

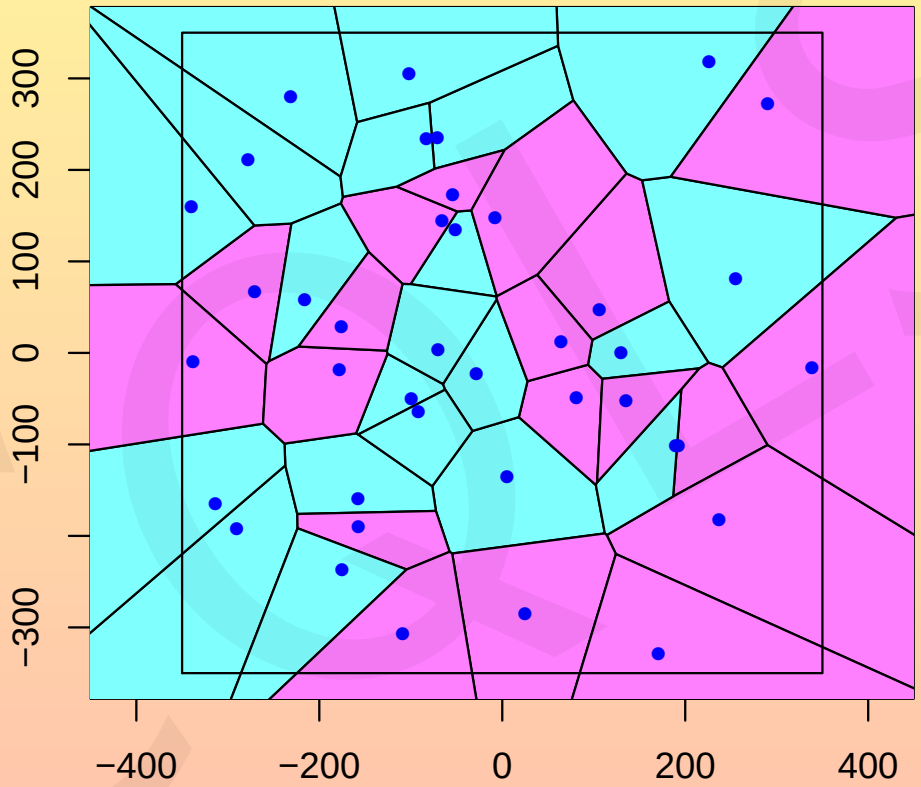
Plan

```
> vor <- EBVor(marks=EBMarks(m=int(1,1:2)))
> print(c(vor$center,vor$size))
[1]   0   0 700 700
> insert(vor,runif(60,-350,350),m=sample(1:2,30,rep=T))
> plot(vor,vcCol=m,dvCex=.8)
```



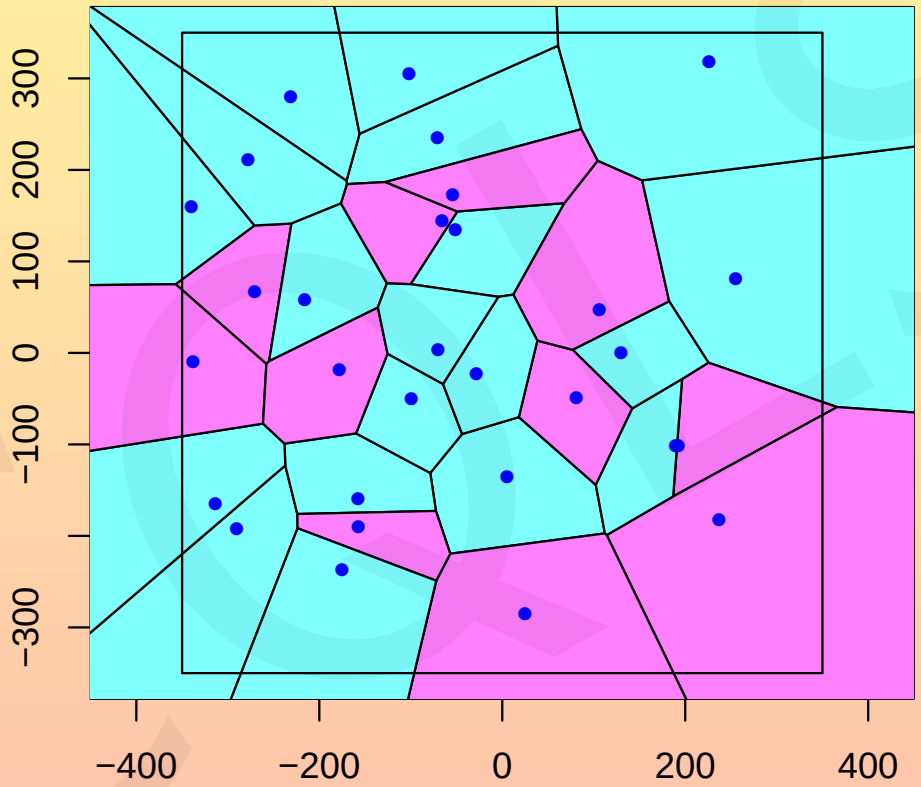

```
> length(vor)
[1] 30
```

```
> #10 new points inserted => it is incremental!
> insert(vor,runif(20,-350,350),m=sample(1:2,10,rep=T))
> plot(vor)
```



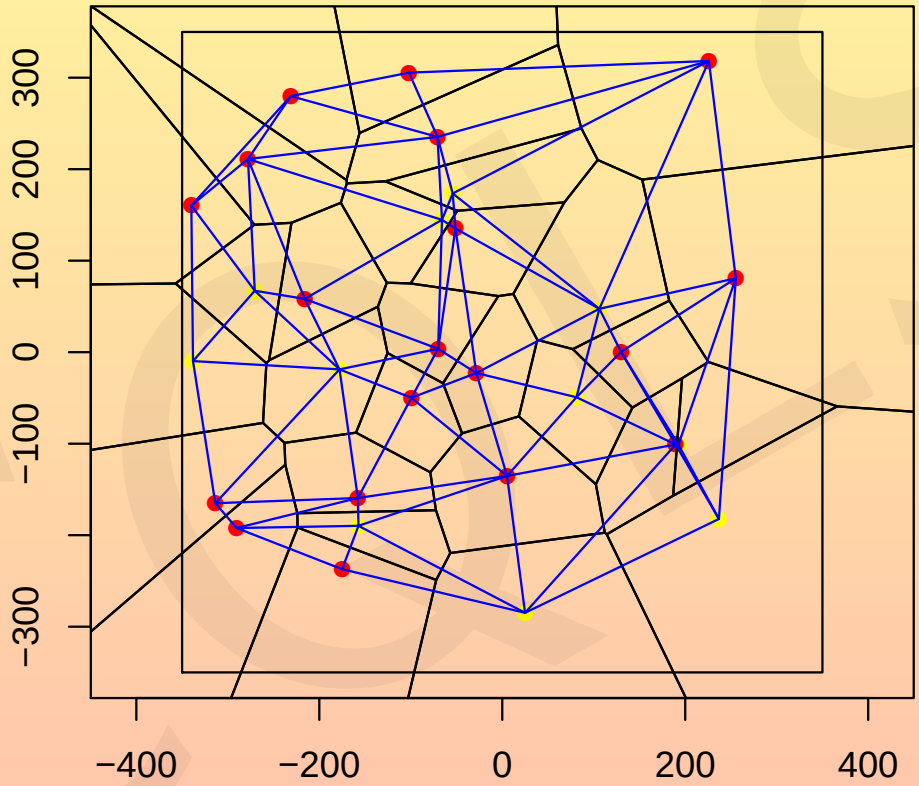

```
> length(vor)
[1] 40
```

```
> #back to the initial configuration by deleting the last 10 points!
> delete(vor,31:40)
> plot(vor)
```



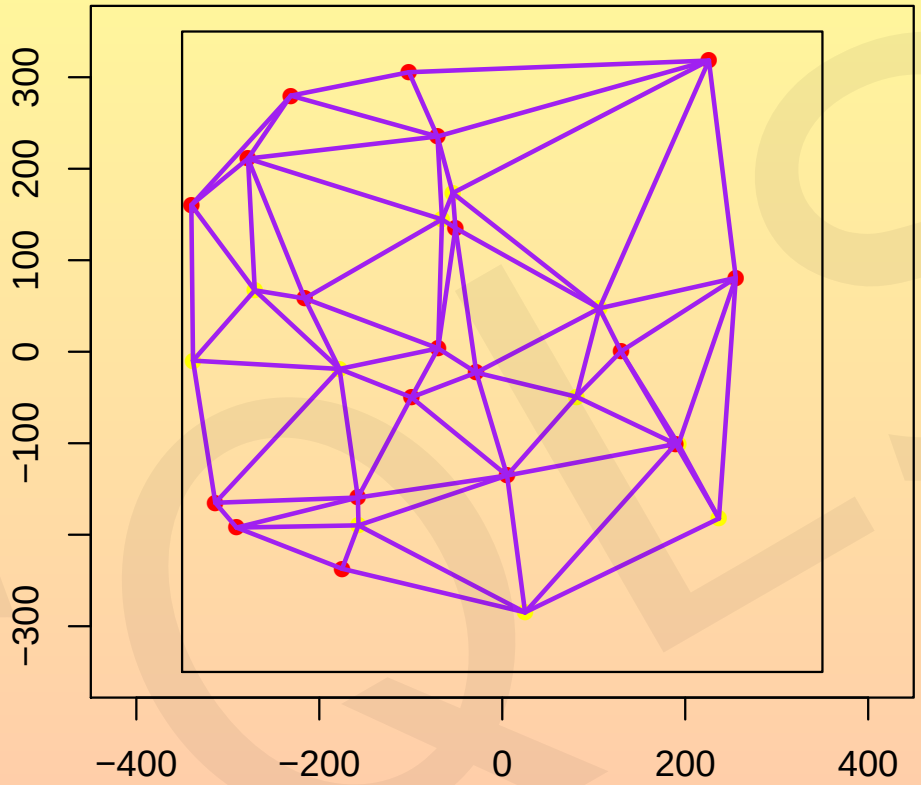

```
> length(vor)
[1] 30
```

```
> plot(vor,0) #plot history
Available choices: (=> stands for the current)
1:  initial default plot (only delaunay vertices, i.e type='dv')
2:=>plot.EBVor(vor, vcCol = m, dvCex = 0.8)
> plot(vor,dvCol=m,type=c("dv","vc","de"))
```



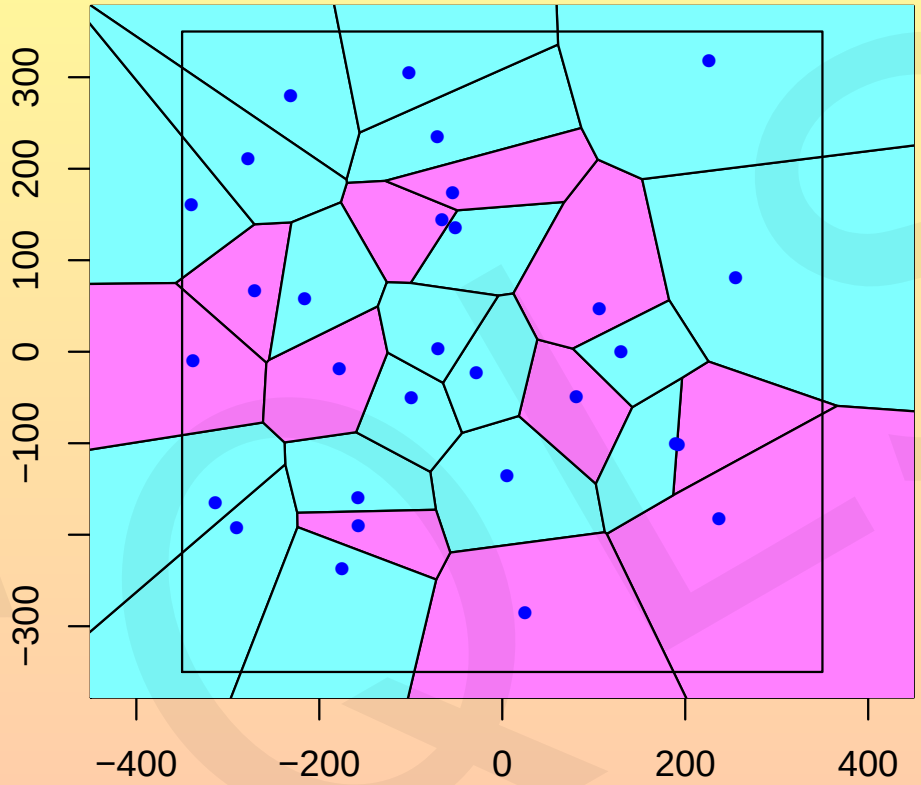

```
> plot(vor,0)
Available choices: (=> stands for the current)
1:  initial default plot (only delaunay vertices, i.e type='dv')
2:  plot.EBVor(vor, vcCol = m, dvCex = 0.8)
3:=>plot.EBVor(vor, dvCol = m, type = c("dv", "vc", "de"))
```

```
> plot(vor,dvCol=m,deArgs=list(lwd=2,col="purple"))
```



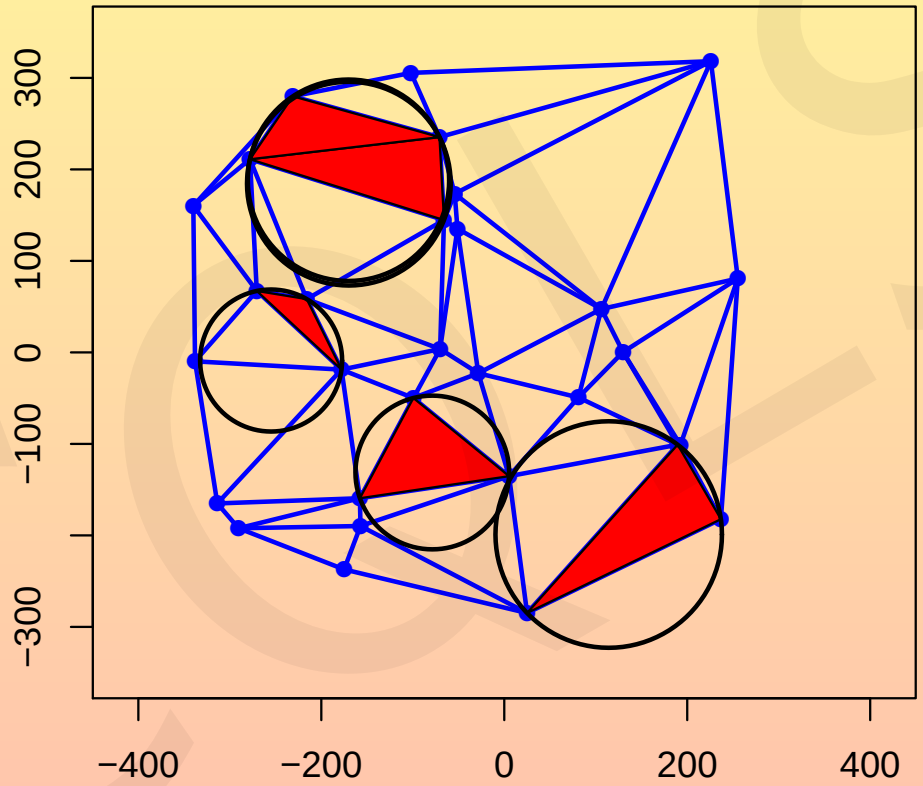

```
> plot(vor,0)
Available choices: (=> stands for the current)
1:  initial default plot (only delaunay vertices, i.e type='dv')
2:  plot.EBVor(vor, vcCol = m, dvCex = 0.8)
3:  plot.EBVor(vor, dvCol = m, type = c("dv", "vc", "de"))
4:=>plot.EBVor(vor, dvCol = m, deArgs = list(lwd = 2, col = "purple"))
```

```
> plot(vor,2) #first user-defined plot
```

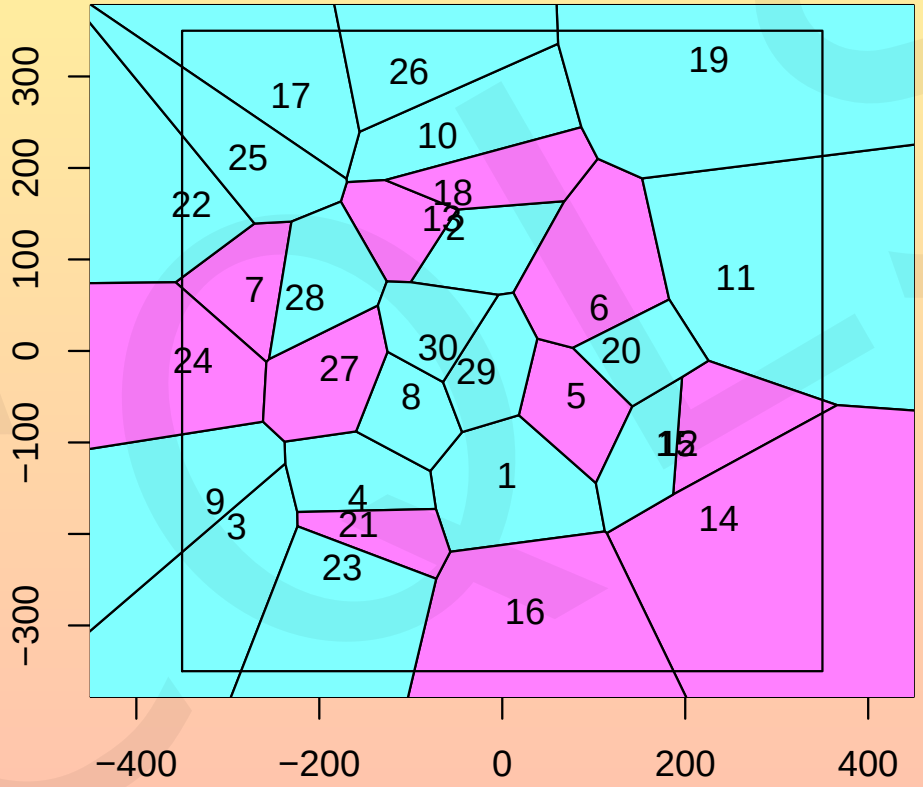



```
> plot(vor,0)
Available choices: (=> stands for the current)
1:  initial default plot (only delaunay vertices, i.e type='dv')
2:=>plot.EBVor(vor, vcCol = m, dvCex = 0.8)
3:  plot.EBVor(vor, dvCol = m, type = c("dv", "vc", "de"))
4:  plot.EBVor(vor, dvCol = m, deArgs = list(lwd = 2, col = "purple"))
```

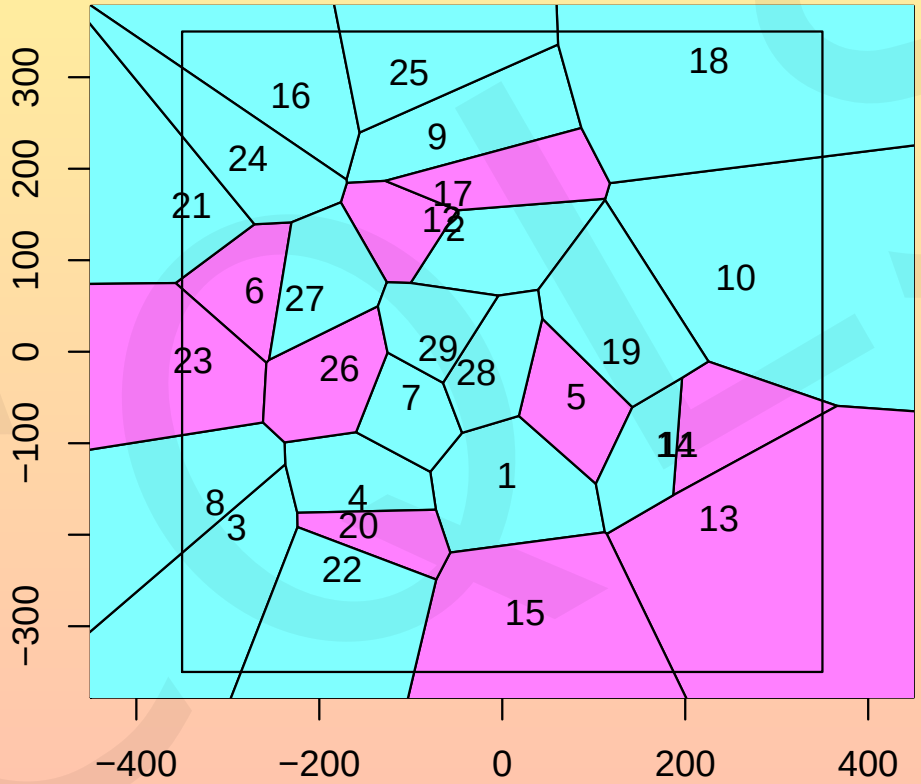
```
> circles(vor,sample(1:30,5))
```



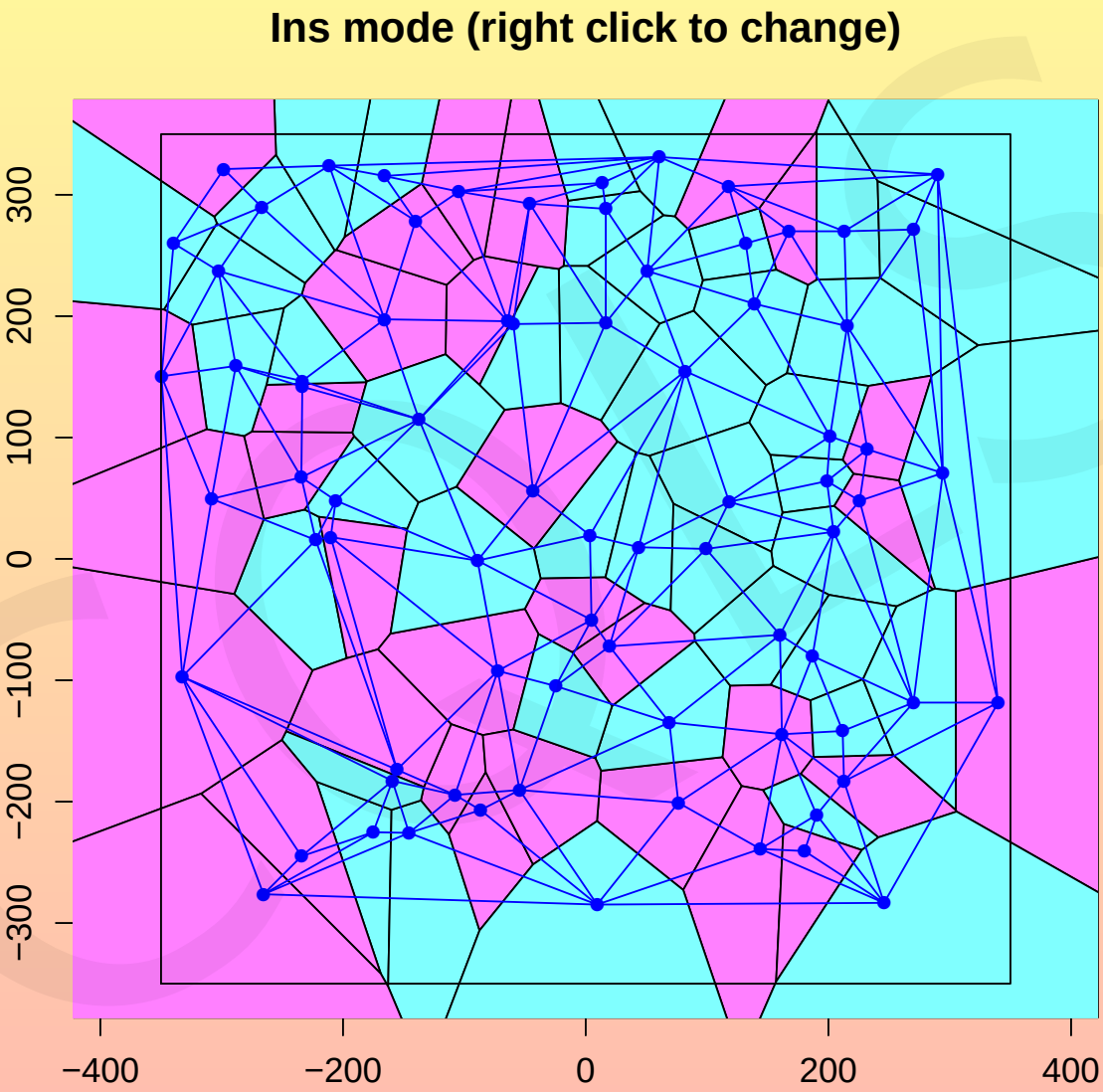
```
> plot(vor,vcCol=m,type=c("de","dv"))
> labels(vor)
```



```
> delete(vor,6)
> plot(vor);labels(vor) #current and last plot
```

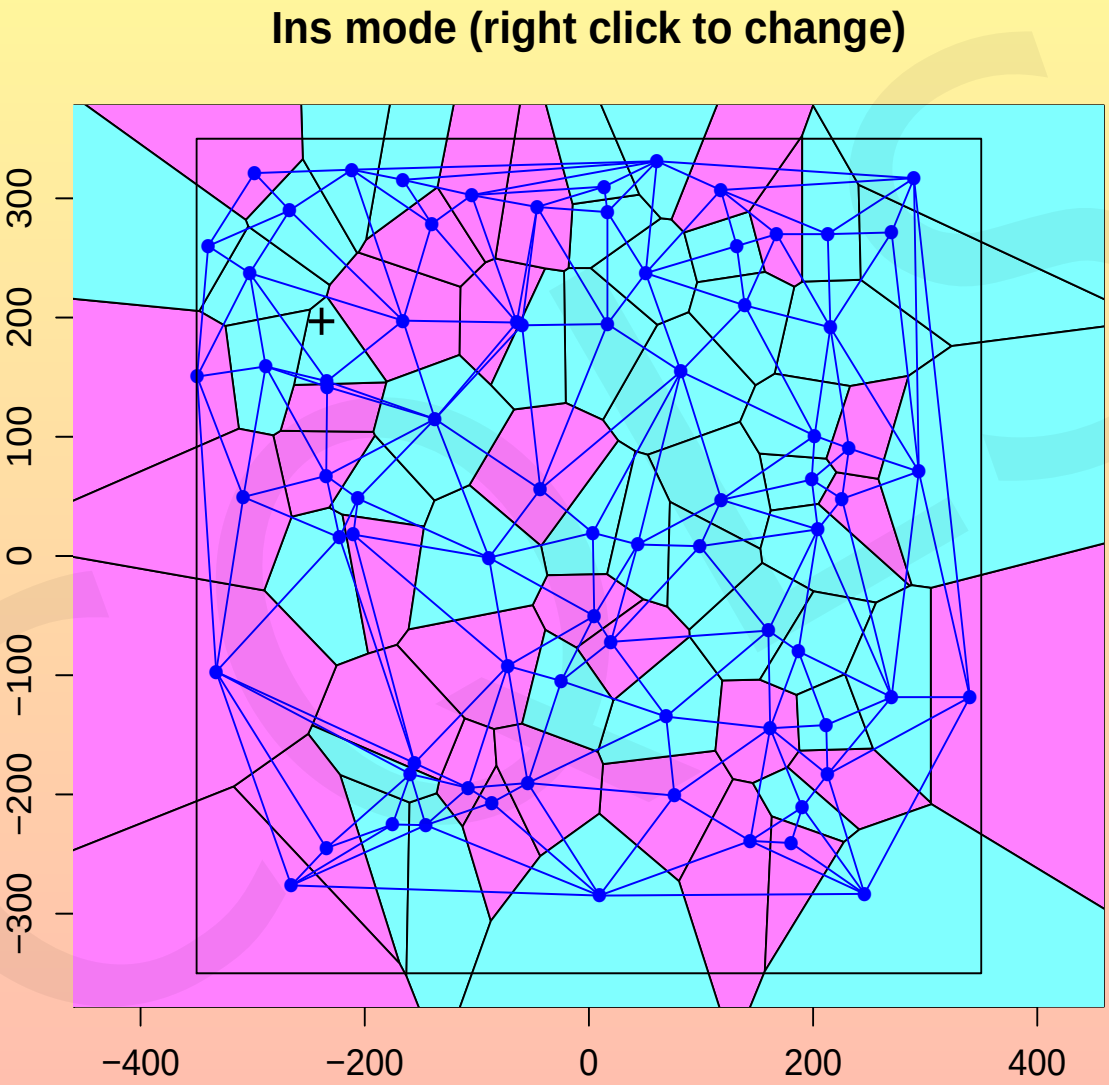


```
> run(vor) #an exploratory tool or a toy!
```



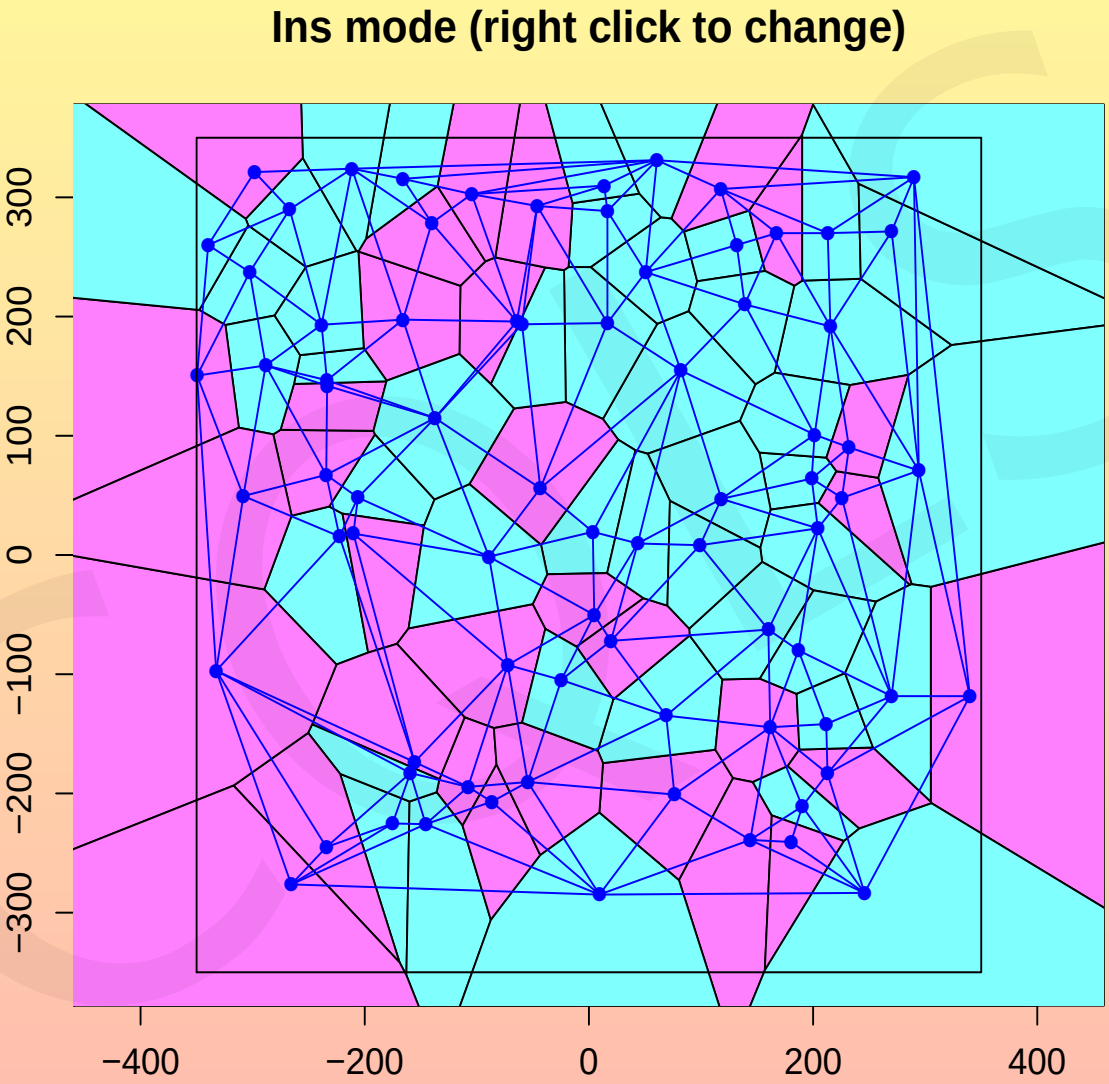
⇒ Soon: use of tcltk tools!

```
> run(vor) #an exploratory tool or a toy!
```



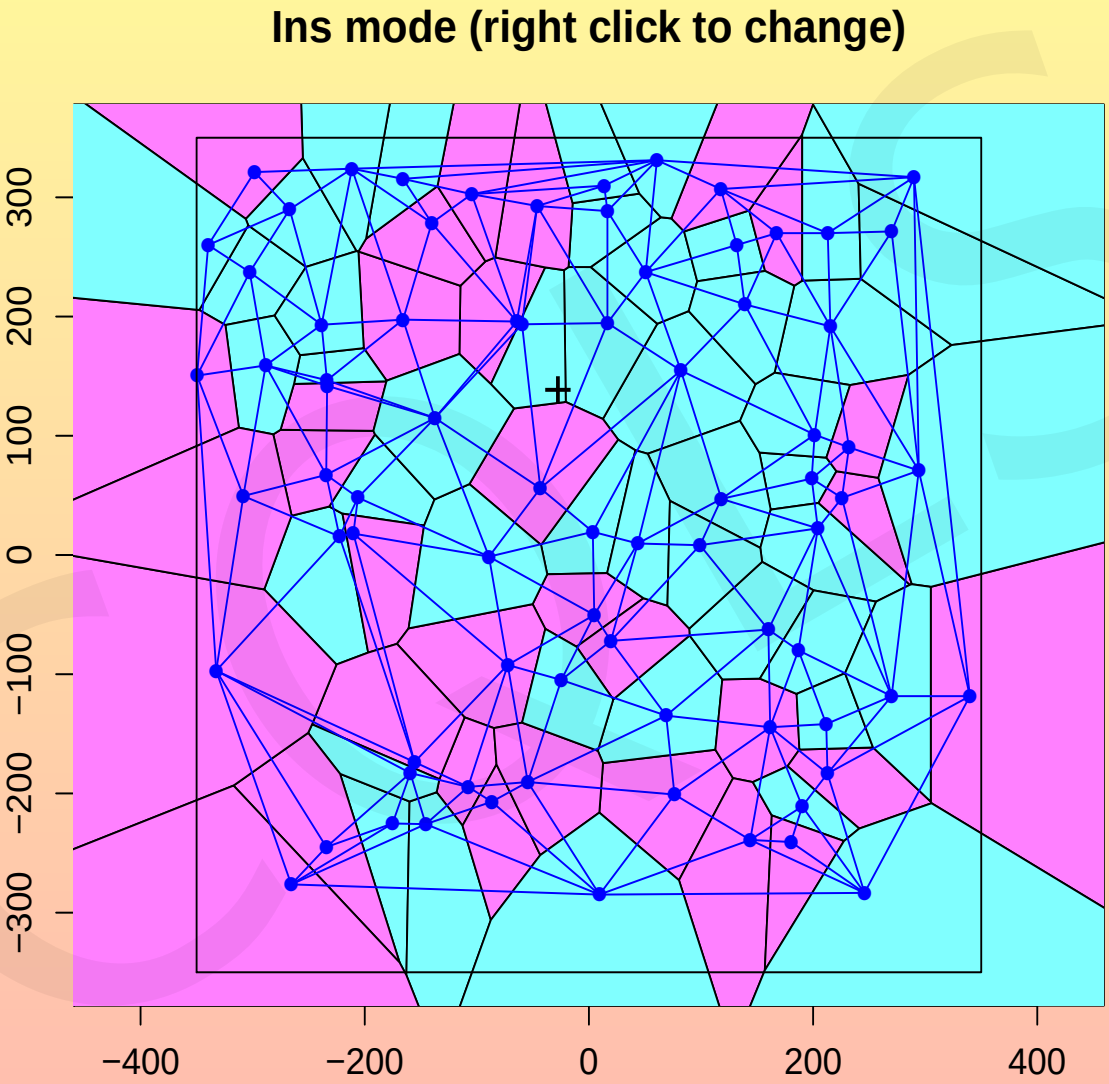
⇒ Soon: use of tcltk tools!

```
> run(vor) #an exploratory tool or a toy!
```



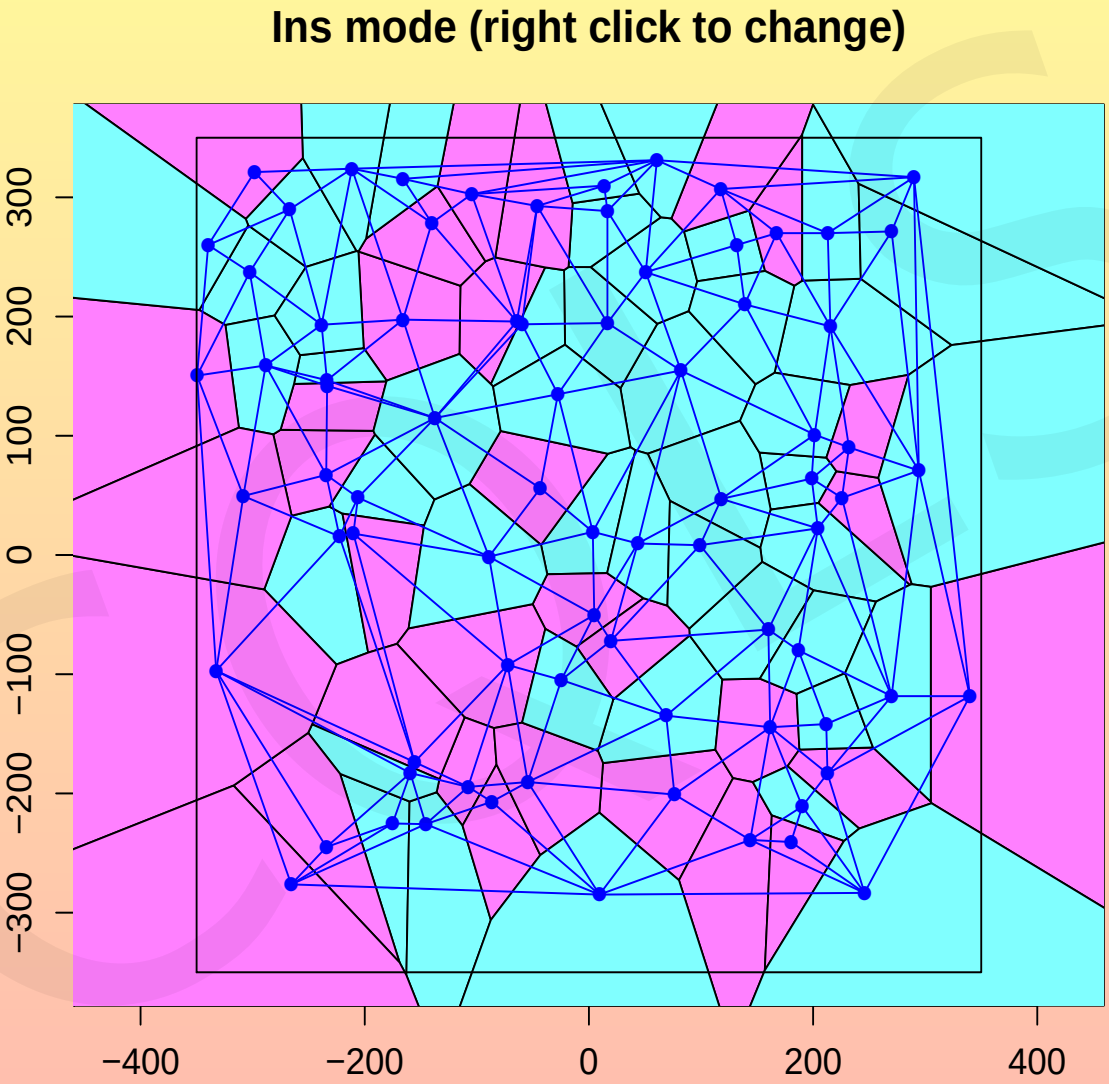
⇒ Soon: use of tcltk tools!

```
> run(vor) #an exploratory tool or a toy!
```



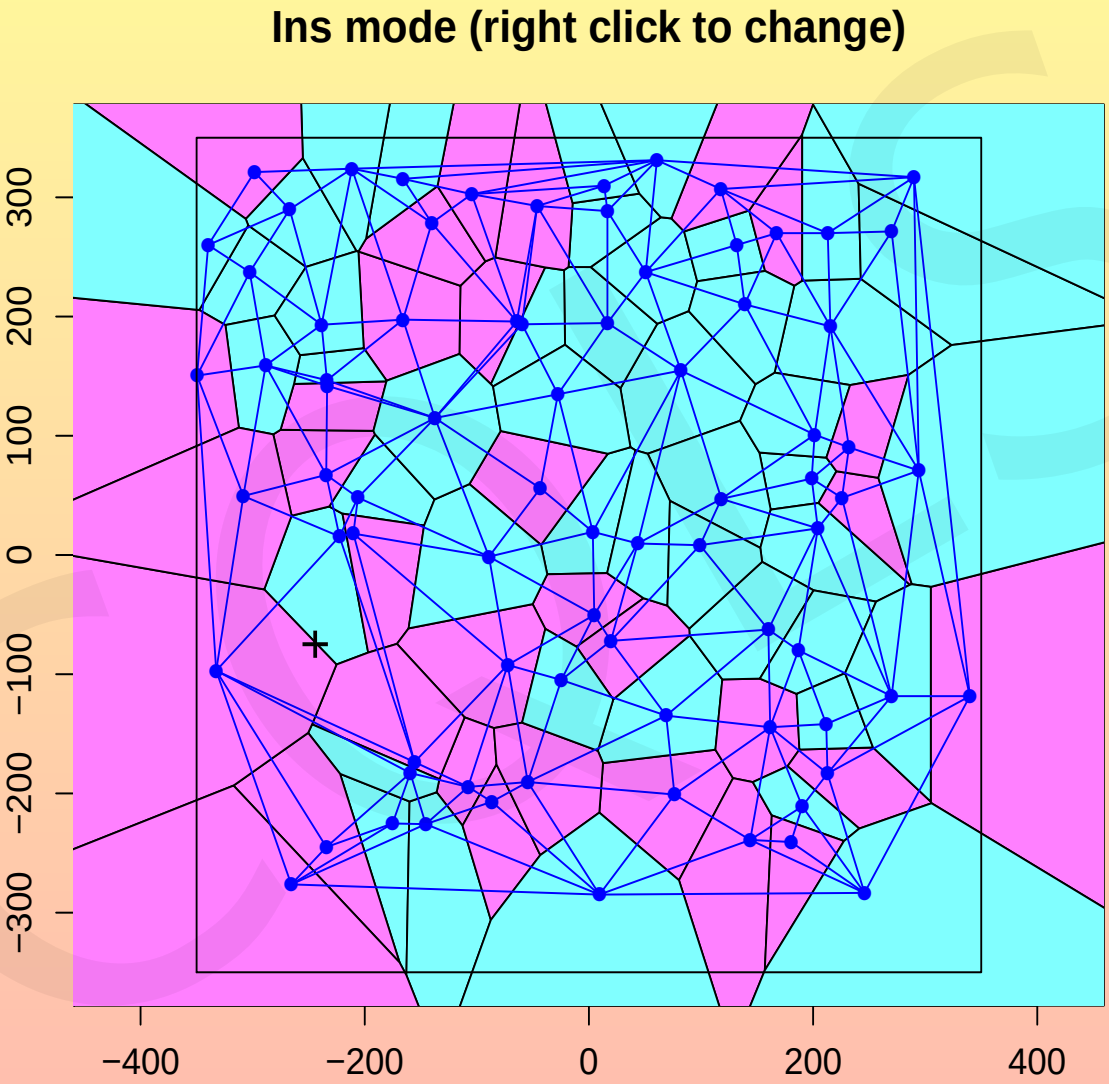
⇒ Soon: use of tcltk tools!

```
> run(vor) #an exploratory tool or a toy!
```



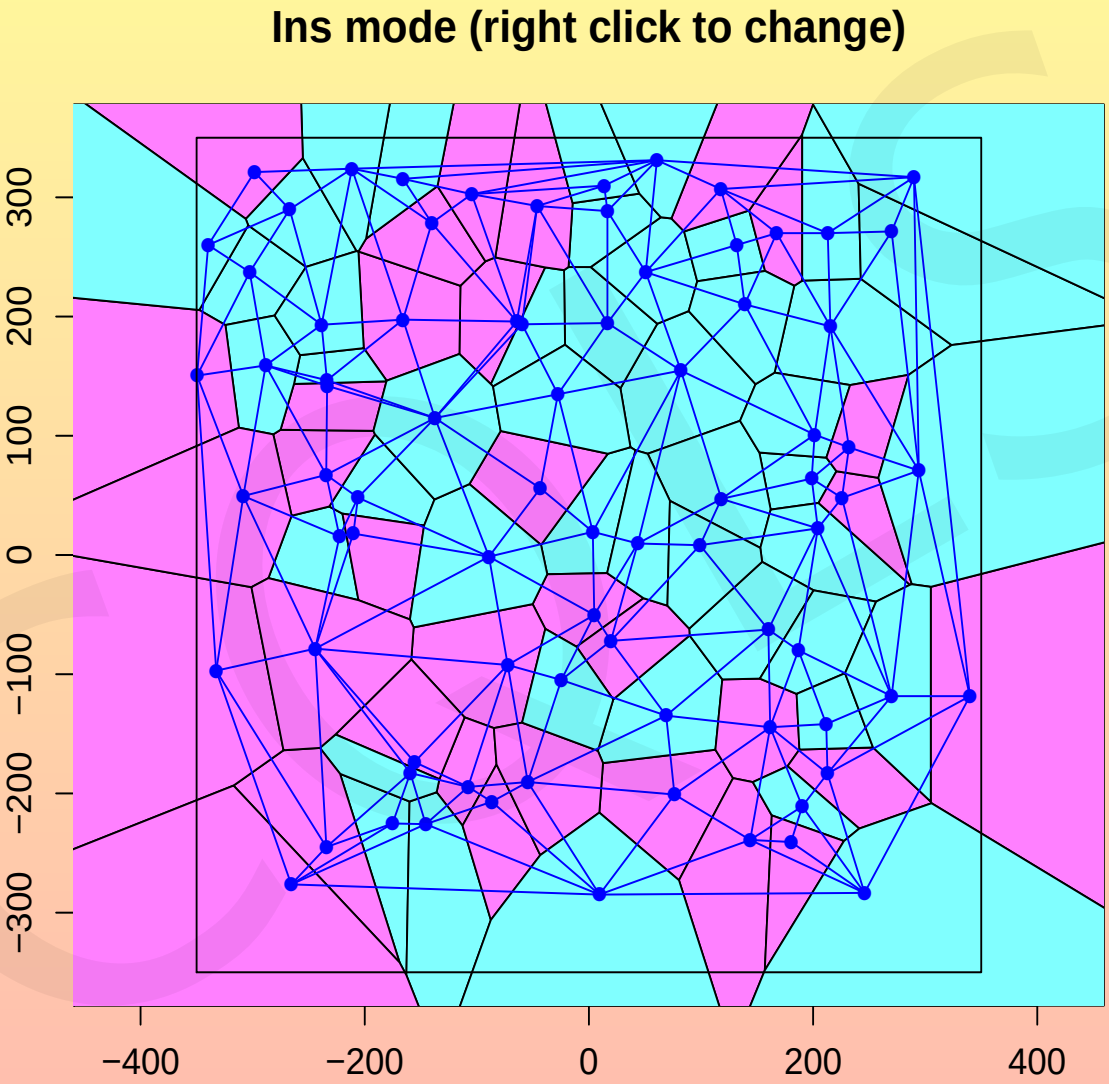
⇒ Soon: use of tcltk tools!

```
> run(vor) #an exploratory tool or a toy!
```



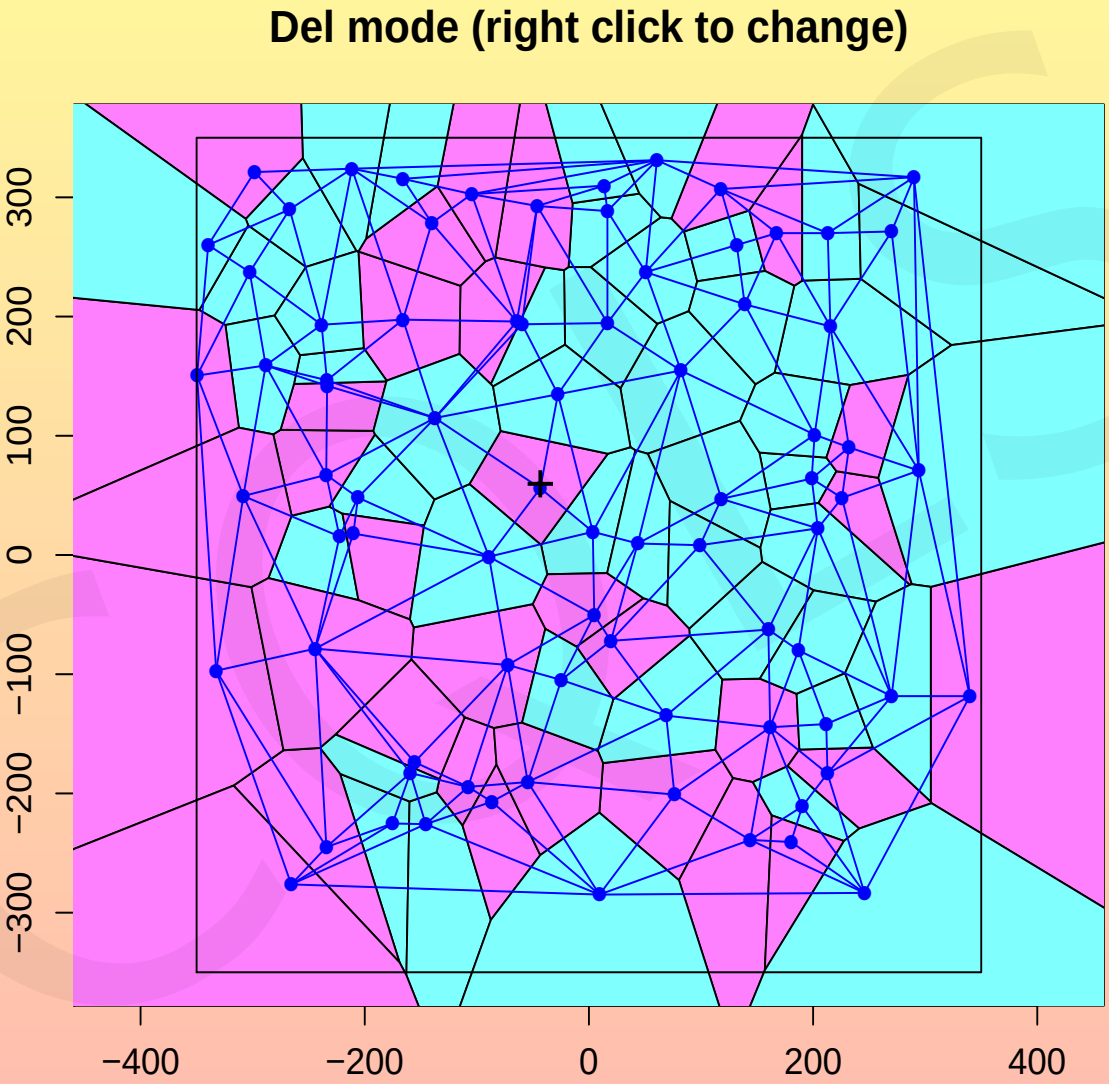
⇒ Soon: use of tcltk tools!

```
> run(vor) #an exploratory tool or a toy!
```



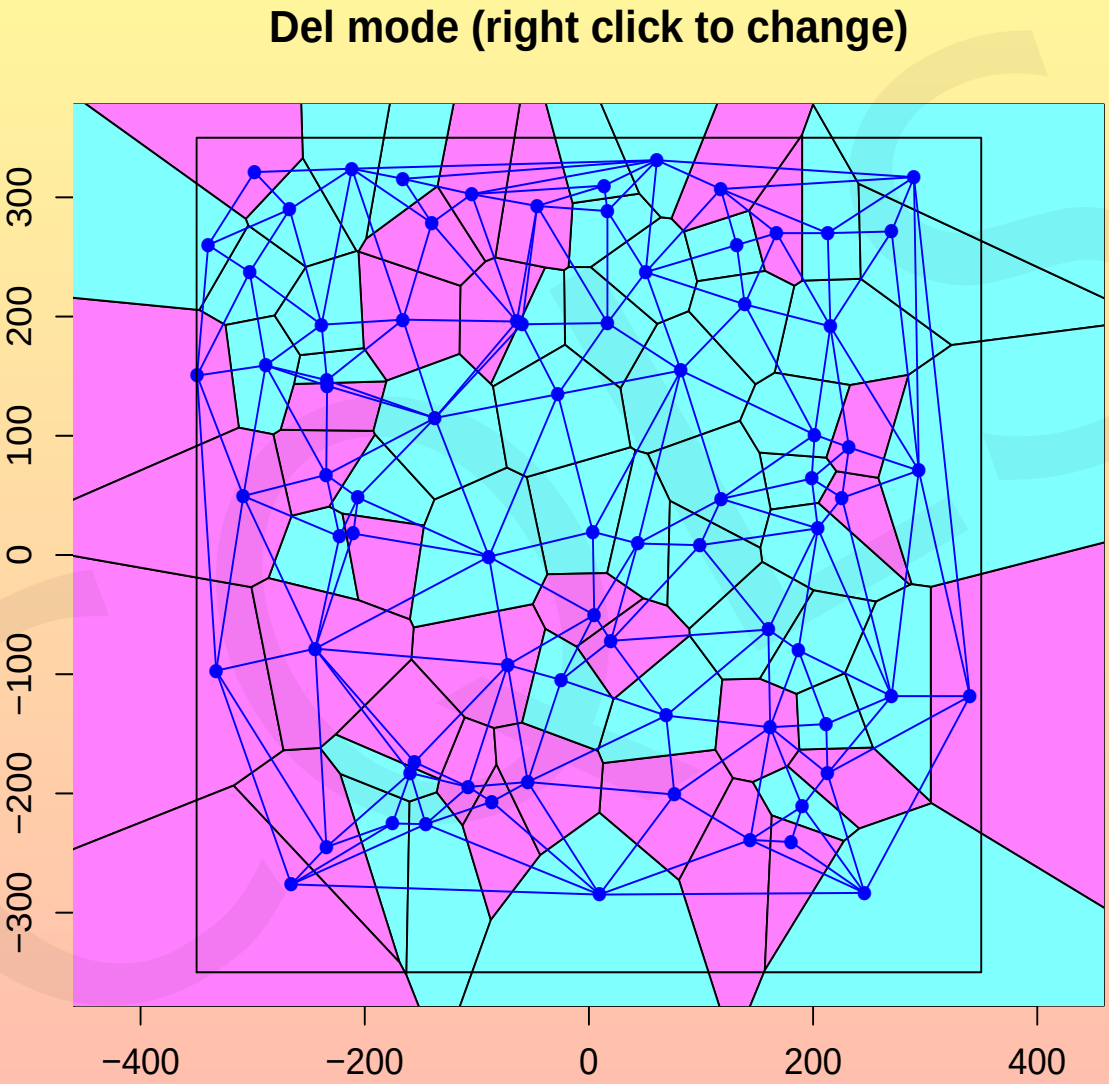
⇒ Soon: use of tcltk tools!

```
> run(vor) #an exploratory tool or a toy!
```



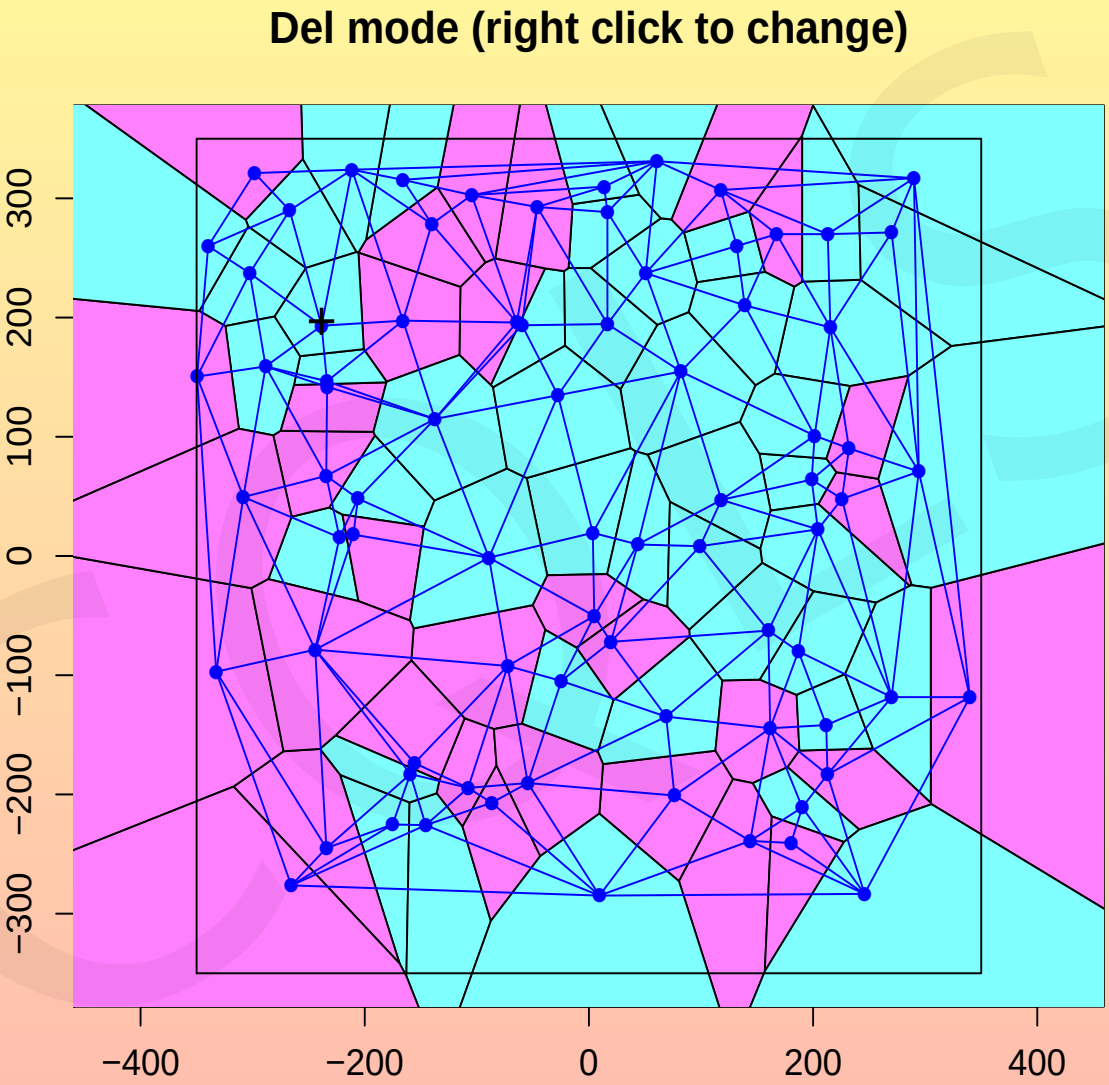
⇒ Soon: use of tcltk tools!

```
> run(vor) #an exploratory tool or a toy!
```



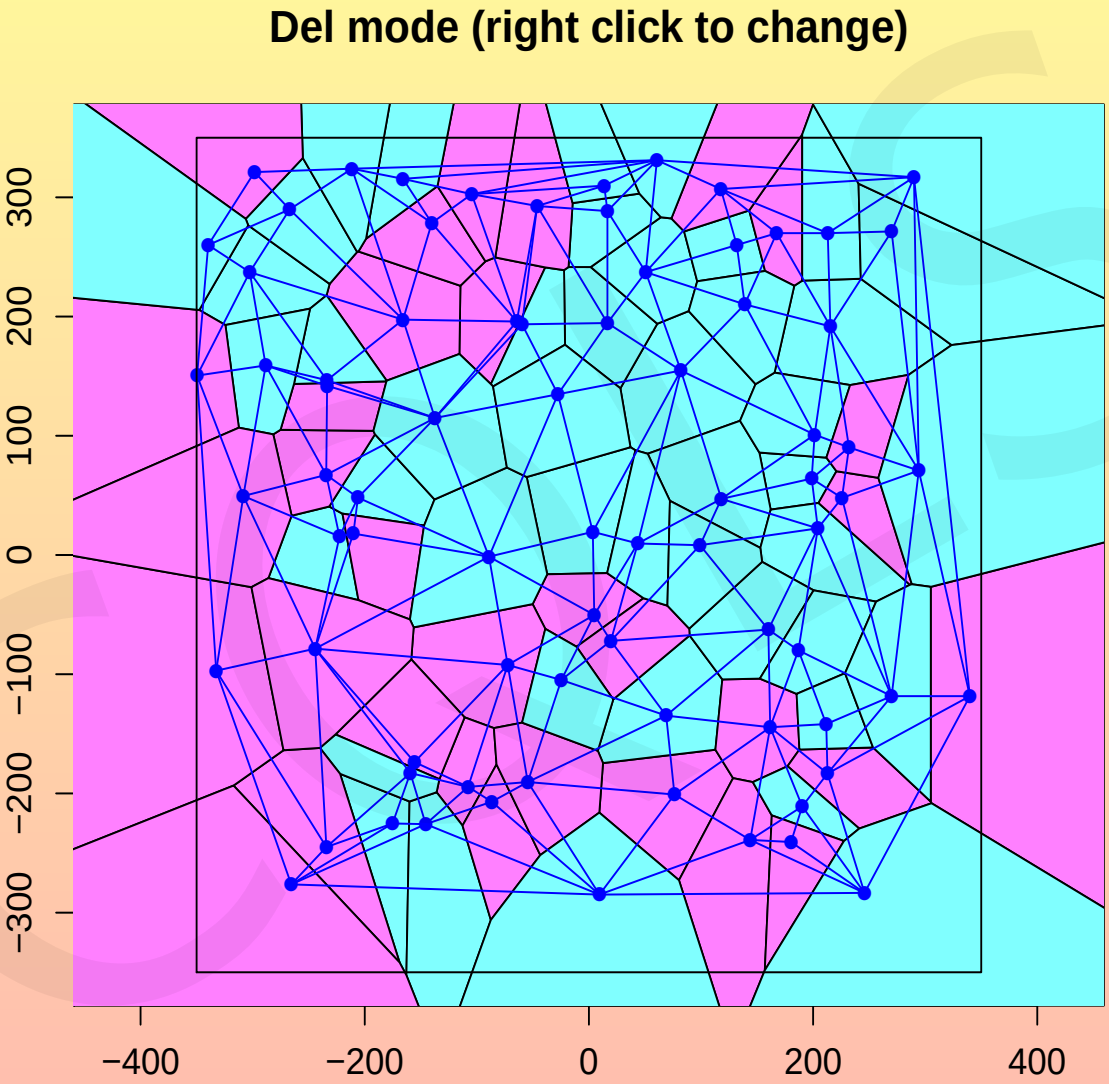
⇒ Soon: use of tcltk tools!

```
> run(vor) #an exploratory tool or a toy!
```



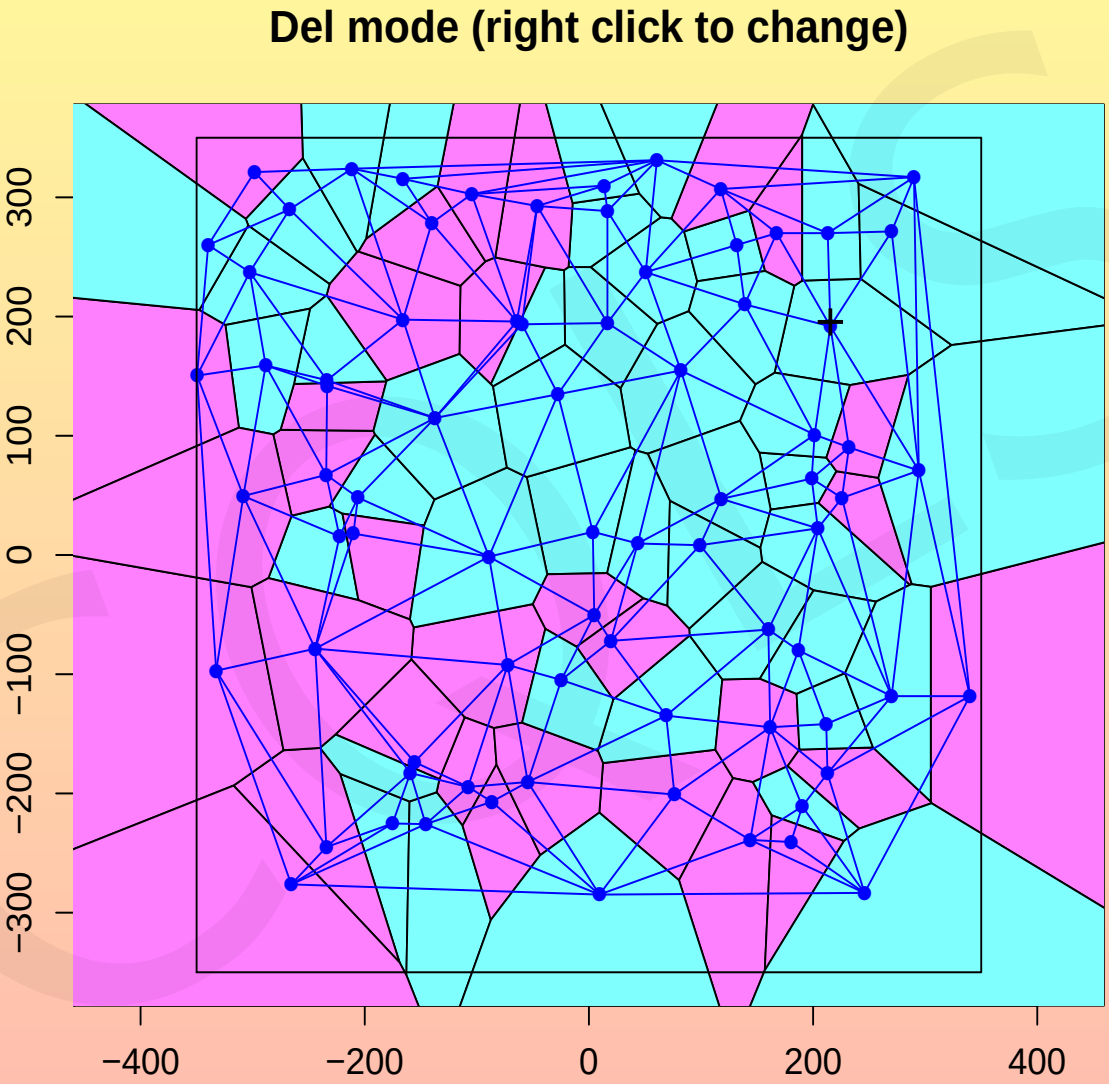
⇒ Soon: use of tcltk tools!

```
> run(vor) #an exploratory tool or a toy!
```



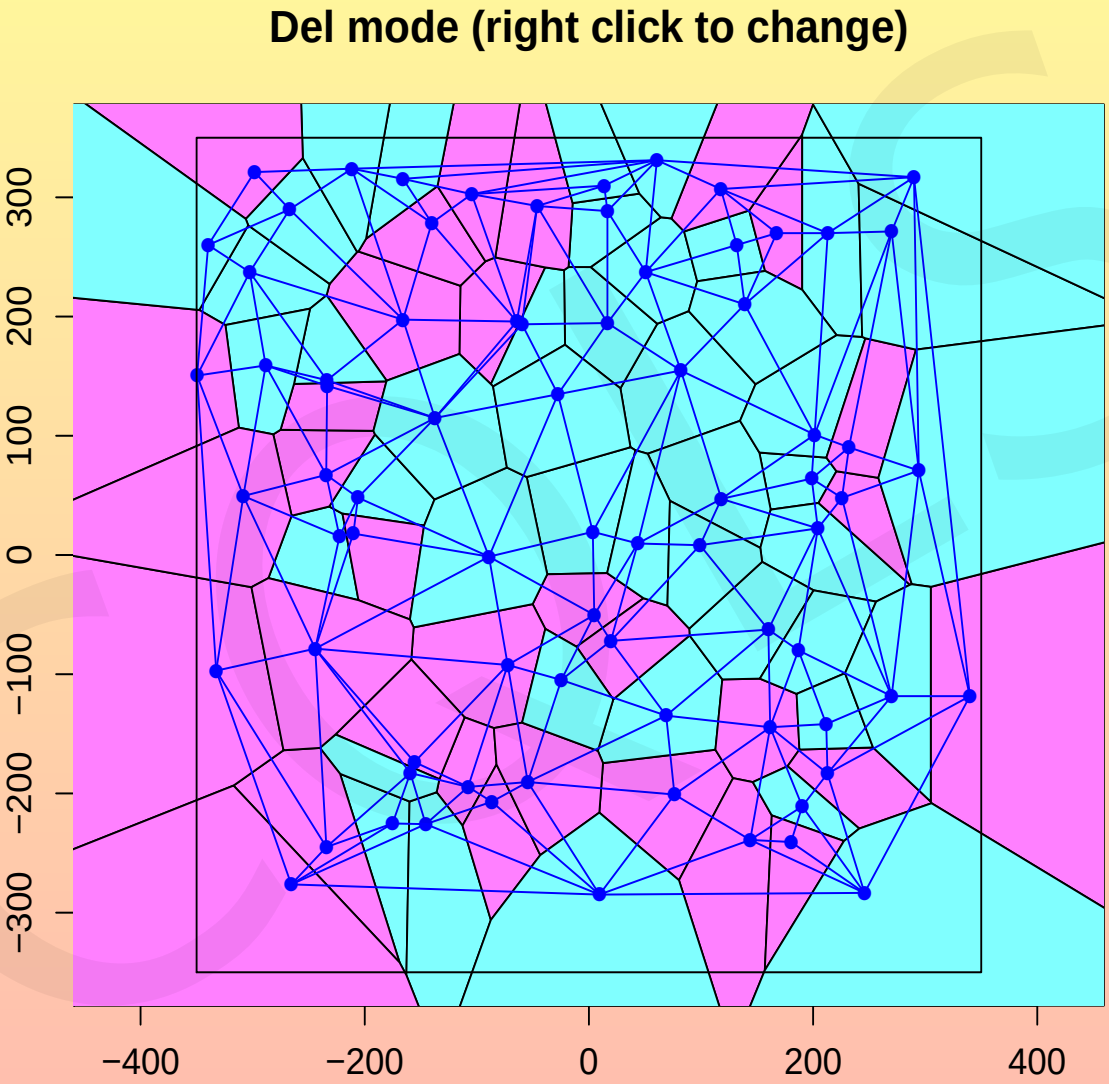
⇒ Soon: use of tcltk tools!

```
> run(vor) #an exploratory tool or a toy!
```



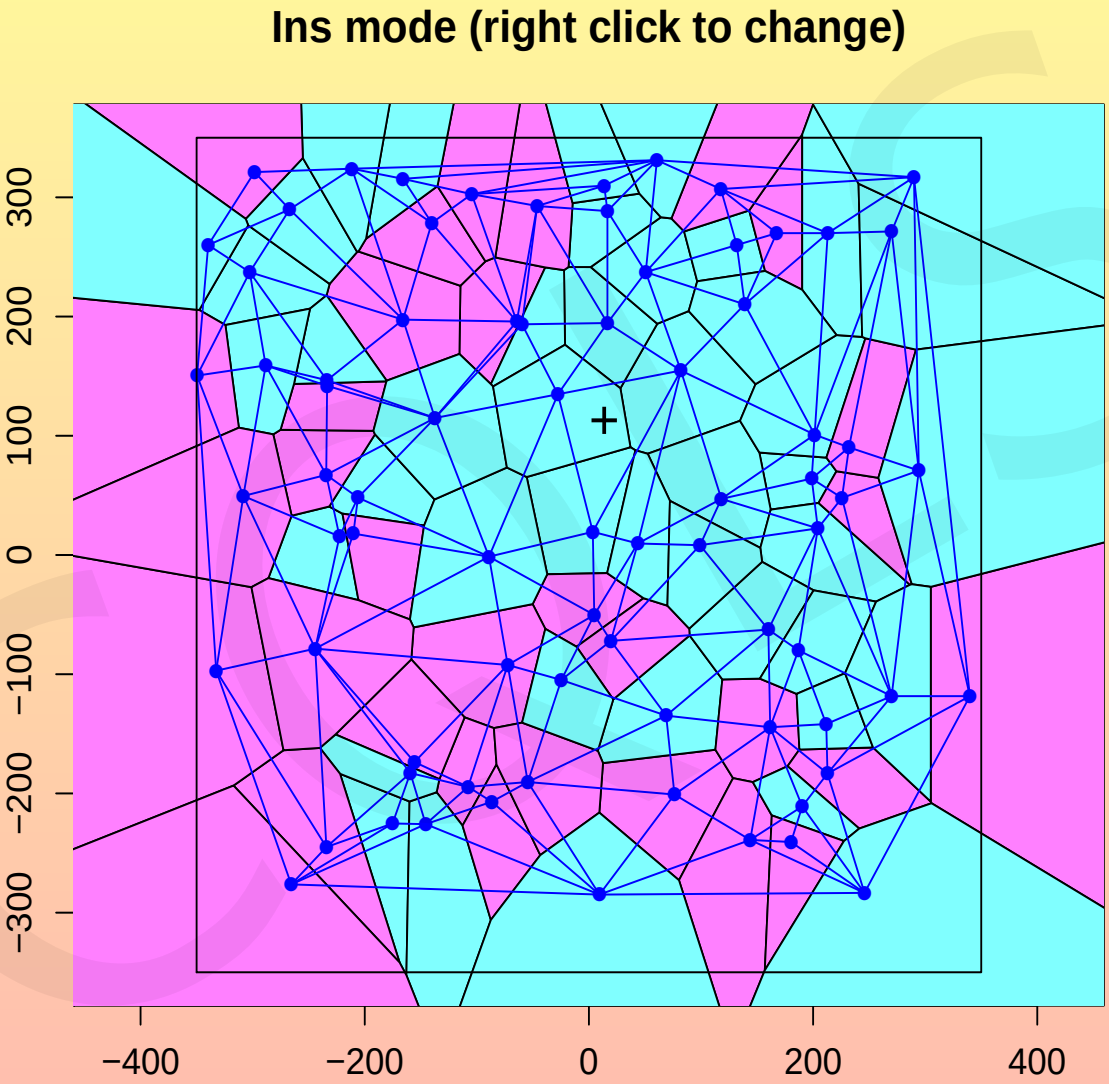
⇒ Soon: use of tcltk tools!

```
> run(vor) #an exploratory tool or a toy!
```



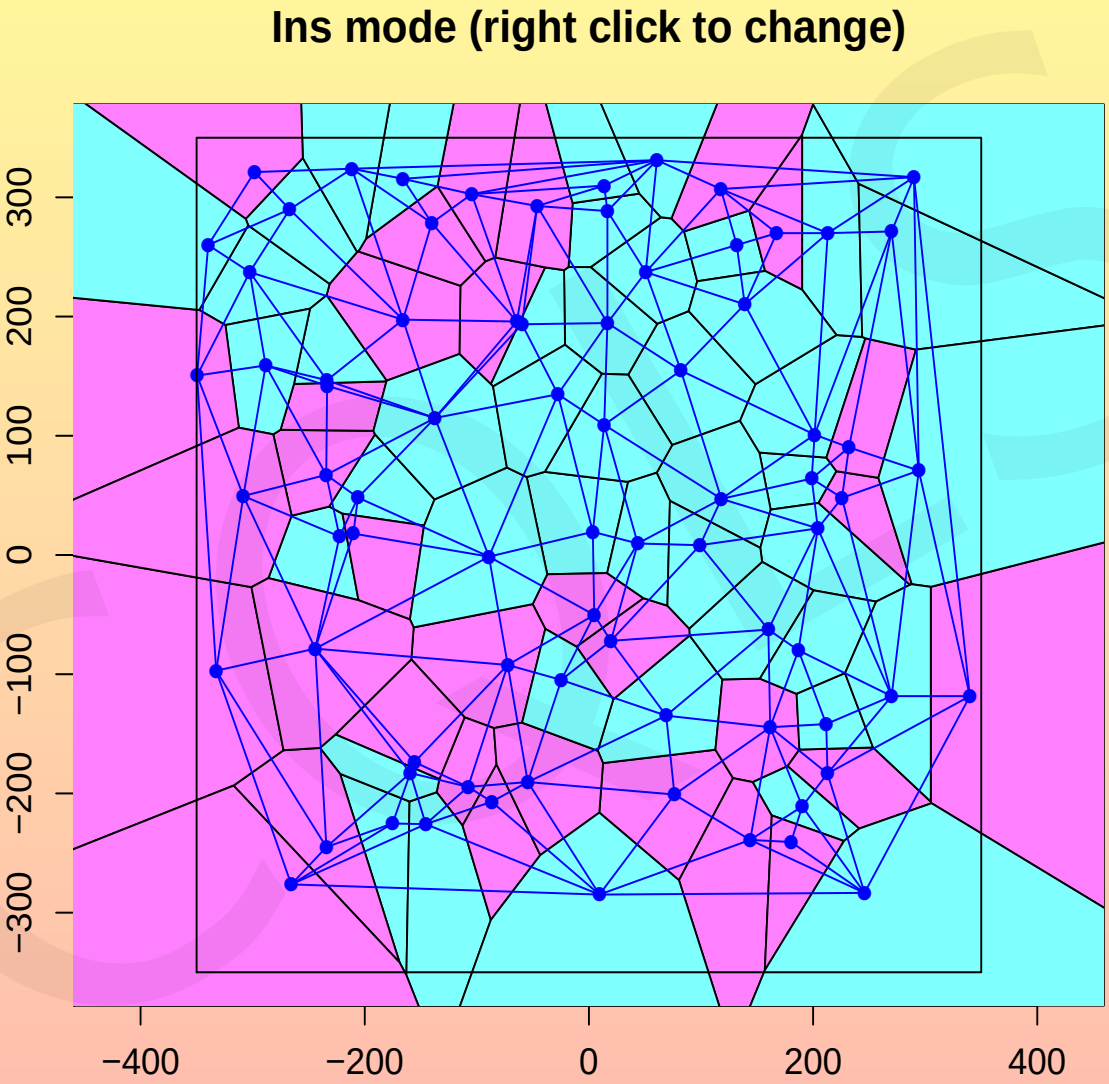
⇒ Soon: use of tcltk tools!

```
> run(vor) #an exploratory tool or a toy!
```



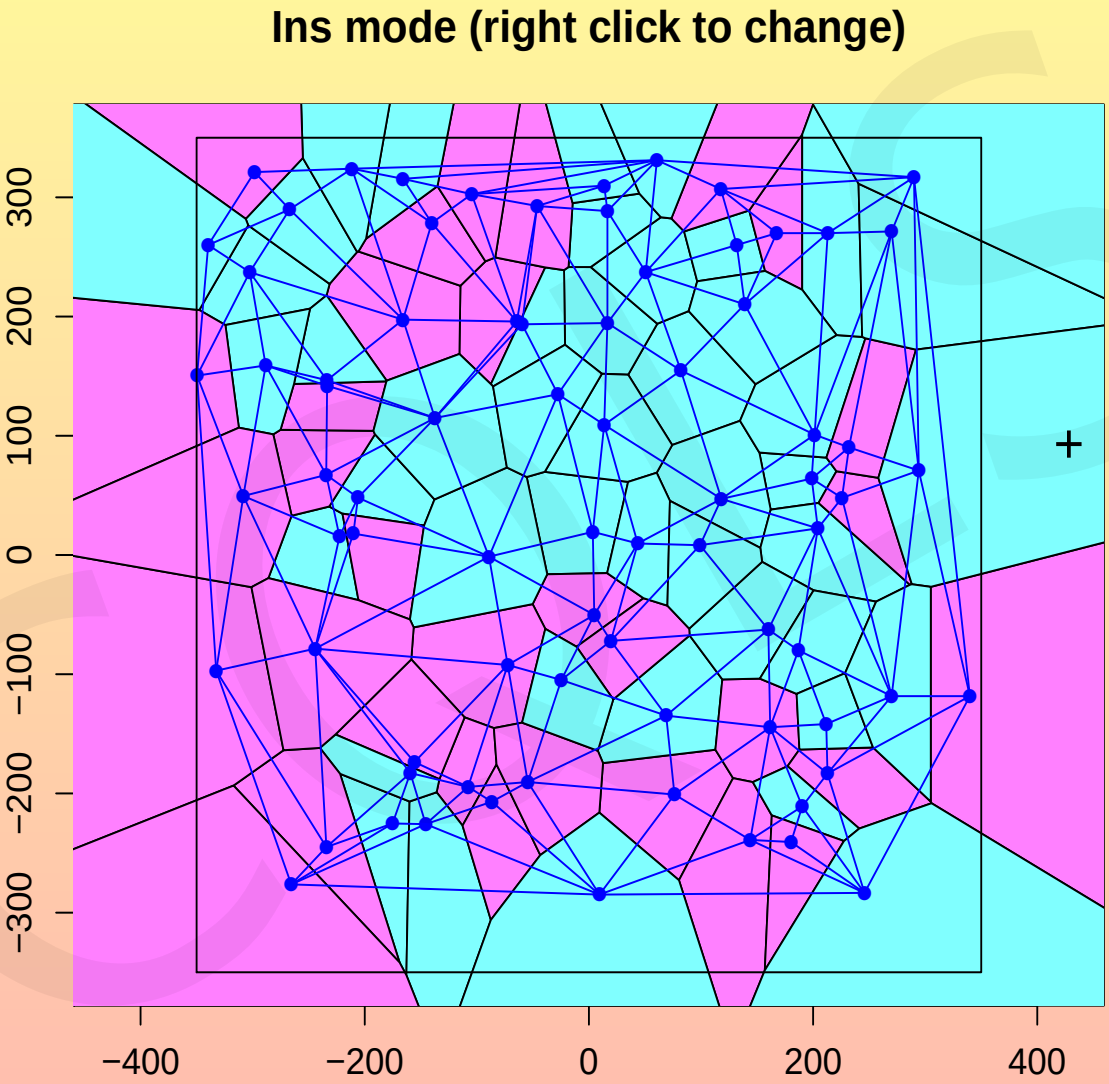
⇒ Soon: use of tcltk tools!

```
> run(vor) #an exploratory tool or a toy!
```



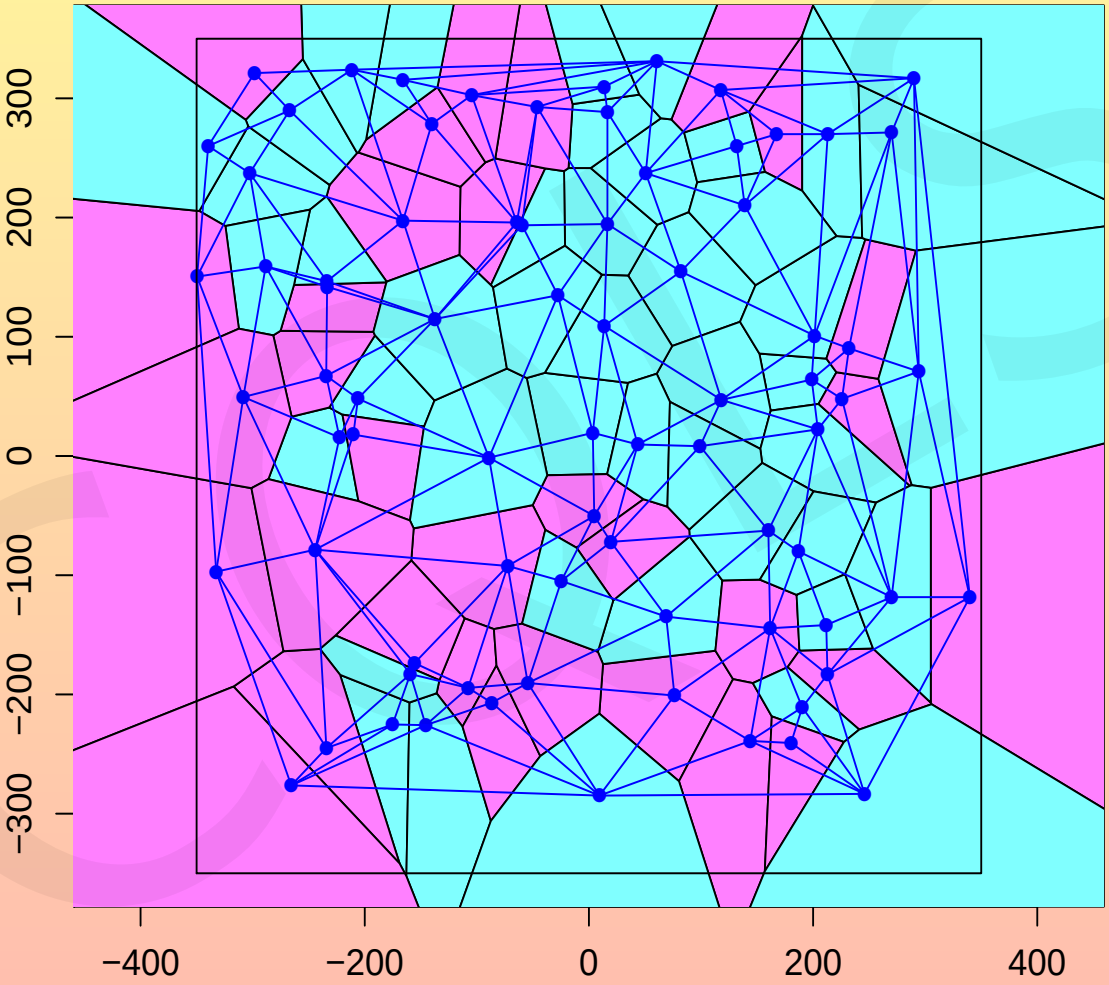
⇒ Soon: use of tcltk tools!

```
> run(vor) #an exploratory tool or a toy!
```



⇒ Soon: use of tcltk tools!

```
> run(vor) #an exploratory tool or a toy!
```



⇒ Soon: use of tcltk tools!

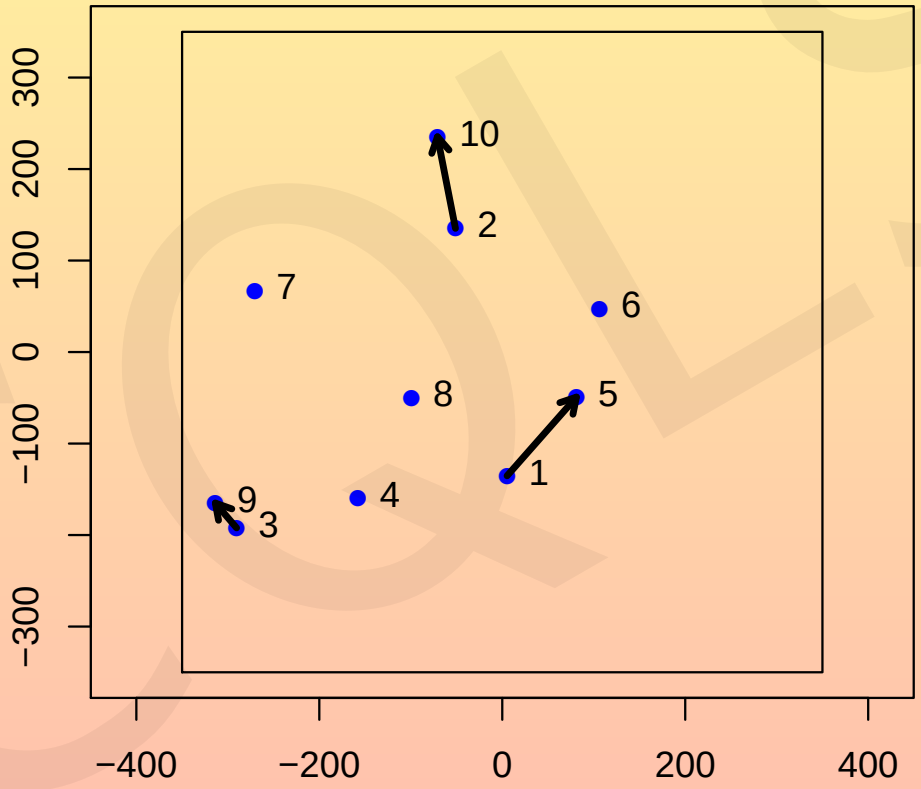
```
> Del1(vor)
   id          x1           x2 v_m          a
1  29  -70.117211    3.6999174   1   9094.165
2  28  -28.368767  -22.7961664   1  10506.977
...
28  1  -51.164633  135.1714566   1  13961.699
29  0    5.234742 -135.2620458   1  19765.036
> Del1(vor)$a
 [1]    9094.165   10506.977   12802.405   13412.075  256087.869
...
[26]   11336.908  292759.253   13961.699   19765.036
> Del2(vor)[1:3,]
  id1 id2       x11       x12       x21       x22 v1_m v2_m
1  15  20  24.91793 -284.8383 -157.1613 -189.76724    2    2
2  15  22  24.91793 -284.8383 -175.0837 -237.17171    2    1
3  12  27 -65.90502  144.6528 -215.9074   58.25538    2    1
          a1          a2       l2       l1         ol2         ol
1 1407878.376   7545.971 42191.35 205.4053    1104.533   33.23451
2 1407878.376 326729.907 42272.74 205.6034 2206562.965 1485.45042
3    8542.216  12802.405 29965.23 173.1047   10162.307  100.80827
          da
1 1400332.406
2 1081148.469
3    4260.188
```

```
> Del3(vor)[1:3,]
  id1 id2 id3         x11         x12         x21         x22         x31
1   0  15  20    5.234742 -135.2620    24.91793 -284.83831 -157.16130
2  12  27  29  -65.905020  144.6528 -215.90739   58.25538  -70.11721
3  15  20  22   24.917926 -284.8383 -157.16130 -189.76724 -175.08367
          x32 v1_m v2_m v3_m           a1           a2          a3
1 -189.767239    1    2    2   19765.036 1407878.376   7545.971
2    3.699917    2    1    1    8542.216   12802.405   9094.165
3 -237.171706    2    2    1 1407878.376    7545.971 326729.907
         ta       tp          c1          c2          r2          r
1 12681.714 527.5700  -56.83287 -219.51292 10950.599 104.64511
2 10389.675 469.7839 -126.19644   75.91517  8359.923  91.43261
3  5167.634 461.6880  -72.21534 -248.97328 10721.171 103.54309
         sa       ga
1 0.8050206 1.377820
2 0.8806397 1.242597
3 0.2472369 1.451056
```

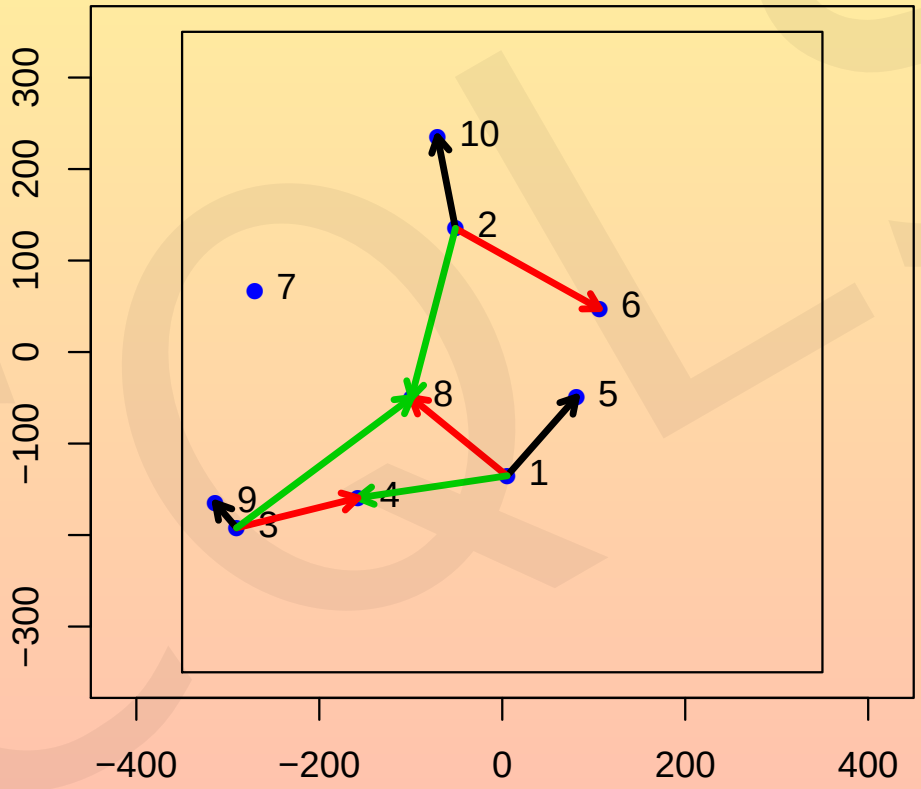
```
> vor <- EBVor(marks=EBMarks(m=int(1,1:2)))
> insert(vor,runif(20,-350,350))
> nearestNeighbours(vor)      # all
Neighbours (1-NNG) of 1: 5
Neighbours (1-NNG) of 2: 10
...
Neighbours (1-NNG) of 9: 3
Neighbours (1-NNG) of 10: 2
> nearestNeighbours(vor,1)    # first
Neighbours (1-NNG) of 1: 5
> nearestNeighbours(vor,3:1) # a subset
Neighbours (1-NNG) of 3: 9
Neighbours (1-NNG) of 2: 10
Neighbours (1-NNG) of 1: 5
> nearestNeighbours(vor,order=2)
Neighbours (2-NNG) of 1: 5,8
Neighbours (2-NNG) of 2: 10,6
...
Neighbours (2-NNG) of 9: 3,4
Neighbours (2-NNG) of 10: 2,6
```

```
> summary(nearestNeighbours(vor))          # all
1 [5.234742,-135.262] --(114.6914)--> 5 [81.08052,-49.22993]
2 [-51.16463,135.1715] --(102.0386)--> 10 [-70.84649,235.2939]
...
9 [-313.6677,-165.0756] --(35.72935)--> 3 [-290.4048,-192.1944]
10 [-70.84649,235.2939] --(102.0386)--> 2 [-51.16463,135.1715]
> summary(nearestNeighbours(vor,1))       # first
1 [5.234742,-135.262] --(114.6914)--> 5 [81.08052,-49.22993]
> summary(nearestNeighbours(vor,3:1))     # a subset
3 [-290.4048,-192.1944] --(35.72935)--> 9 [-313.6677,-165.0756]
2 [-51.16463,135.1715] --(102.0386)--> 10 [-70.84649,235.2939]
1 [5.234742,-135.262] --(114.6914)--> 5 [81.08052,-49.22993]
> summary(nearestNeighbours(vor,order=2)) # 2-nng
1 [5.234742,-135.262] --(114.6914)--> 5 [81.08052,-49.22993]
1 [5.234742,-135.262] --(135.0524)--> 8 [-99.36502,-49.83341]
2 [-51.16463,135.1715] --(102.0386)--> 10 [-70.84649,235.2939]
2 [-51.16463,135.1715] --(180.1434)--> 6 [106.159,47.41643]
...
10 [-70.84649,235.2939] --(102.0386)--> 2 [-51.16463,135.1715]
10 [-70.84649,235.2939] --(258.1257)--> 6 [106.159,47.41643]
```

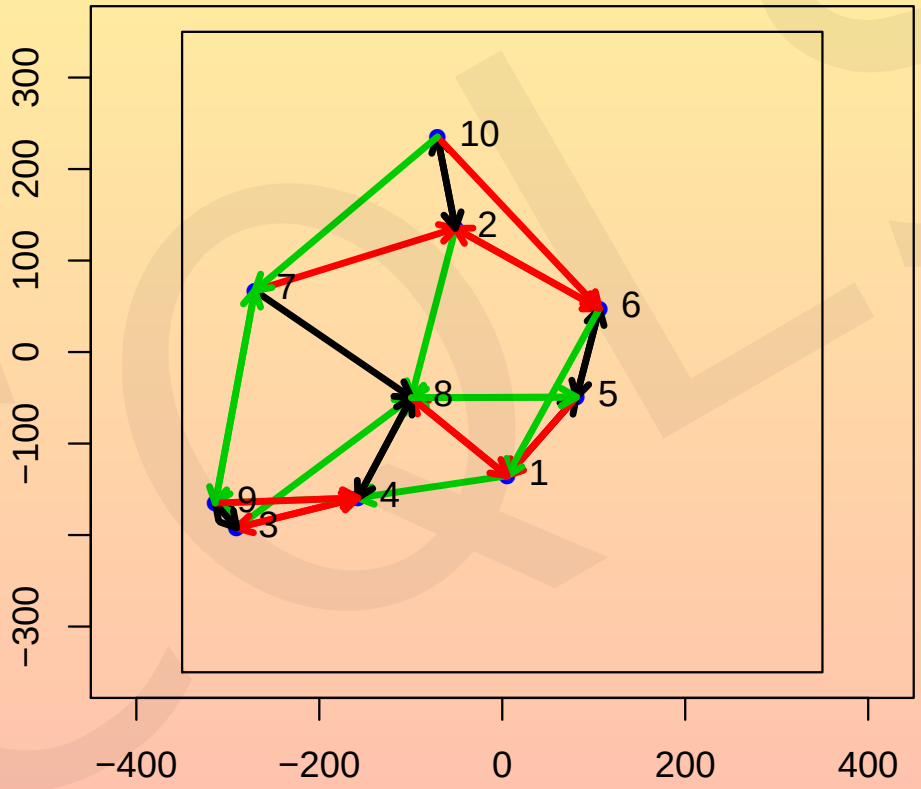
```
> plot(vor)
> labels(vor,pos=4)
> plot(nearestNeighbours(vor,1:3),lwd=3) # need a main plot initialized
```



```
> plot(vor)
> labels(vor,pos=4)
> plot(nearestNeighbours(vor,1:3,order=3),lwd=3,col=1:3)
```



```
> # a global plot is also available!
> plot(vor,type=c("dv","3-nng"),nngCol=1:3,nngArgs=list(lwd=3))
> labels(vor,pos=4)
```



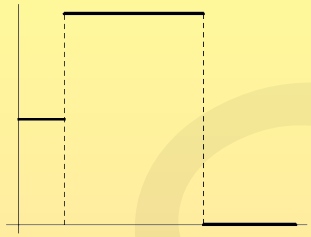
Plan

Gibbs Distribution in Λ

$$P_\Lambda(F) = Z_\Lambda^{-1} \oint_\Lambda d\varphi \mathbb{1}_F(\varphi) e^{-V(\varphi)}$$

$$V(\varphi) = \theta_1 |\varphi| + \sum_{\xi \in G_2(\varphi)} g_2(\xi).$$

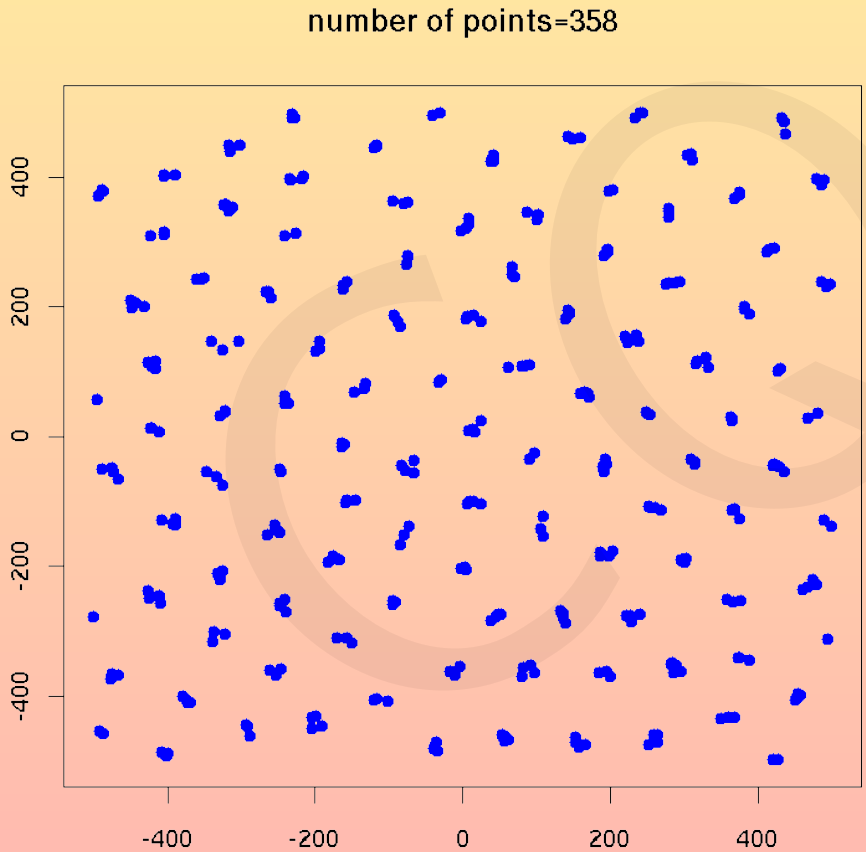
$$g_2(\xi) = \theta_2 \mathbb{1}_{[d_1, d_2[}(\|\xi\|) + \theta_3 \mathbb{1}_{[d_2, d_3[}(\|\xi\|)$$



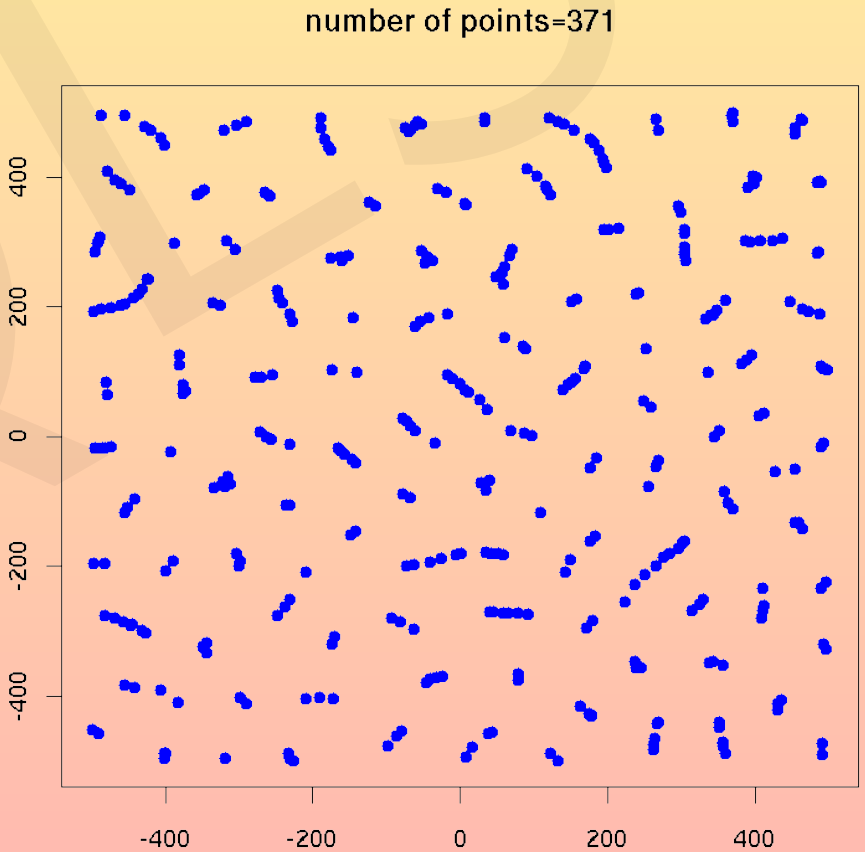
with

$\theta_2 = 2, \theta_3 = 4$

$\mathbf{d} = (0, 20, 80)$



$G_2(\varphi) = \mathcal{P}_2(\varphi)$ and $\theta_1 = -2$



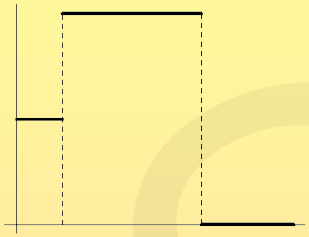
$G_2(\varphi) = Del_2(\varphi)$ and $\theta_1 = 2$

Gibbs Distribution in Λ

$$P_\Lambda(F) = Z_\Lambda^{-1} \oint_\Lambda d\varphi \mathbb{1}_F(\varphi) e^{-V(\varphi)}$$

$$V(\varphi) = \theta_1 |\varphi| + \sum_{\xi \in G_2(\varphi)} g_2(\xi).$$

$$g_2(\xi) = \theta_2 \mathbb{1}_{[d_1, d_2[}(\|\xi\|) + \theta_3 \mathbb{1}_{[d_2, d_3[}(\|\xi\|)$$

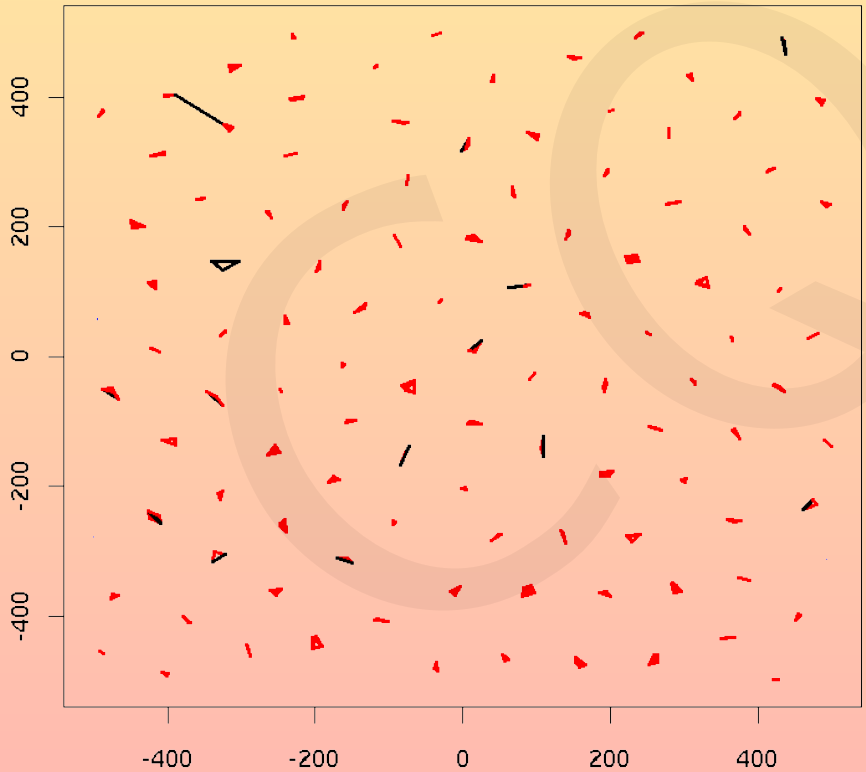


with

$\theta_2 = 2, \theta_3 = 4$

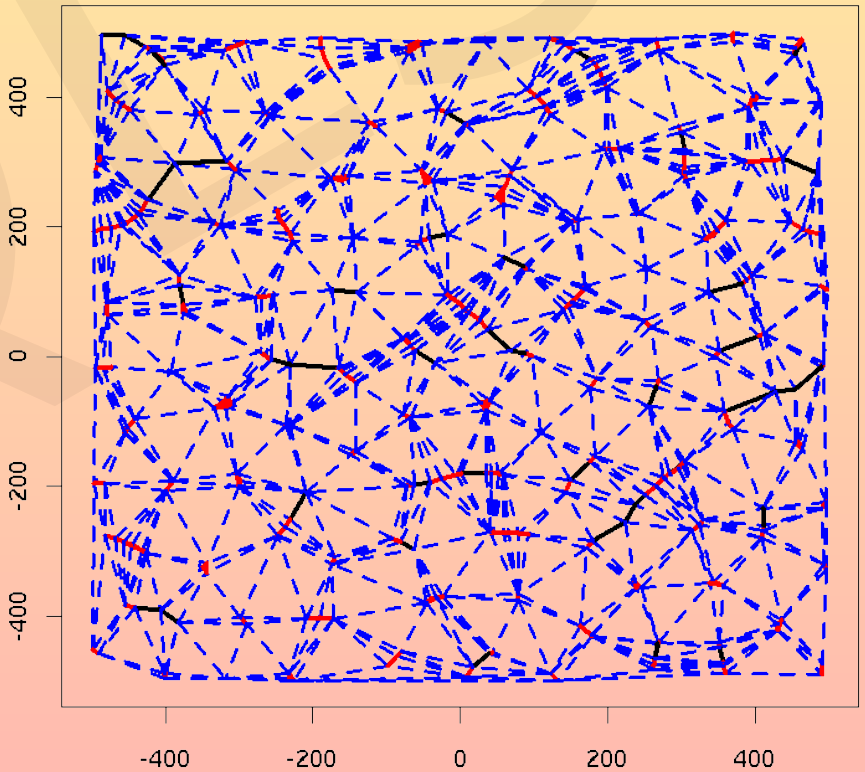
$\mathbf{d} = (0, 20, 80)$

Small 425 (0.7%), Medium 19 (0%), Large 63459 (99.3%)



$G_2(\varphi) = \mathcal{P}_2(\varphi)$ and $\theta_1 = -2$

Small 280 (26.1%), Medium 41 (3.8%), Large 750 (70%)



$G_2(\varphi) = Del_2(\varphi)$ and $\theta_1 = 2$

The corresponding R instructions

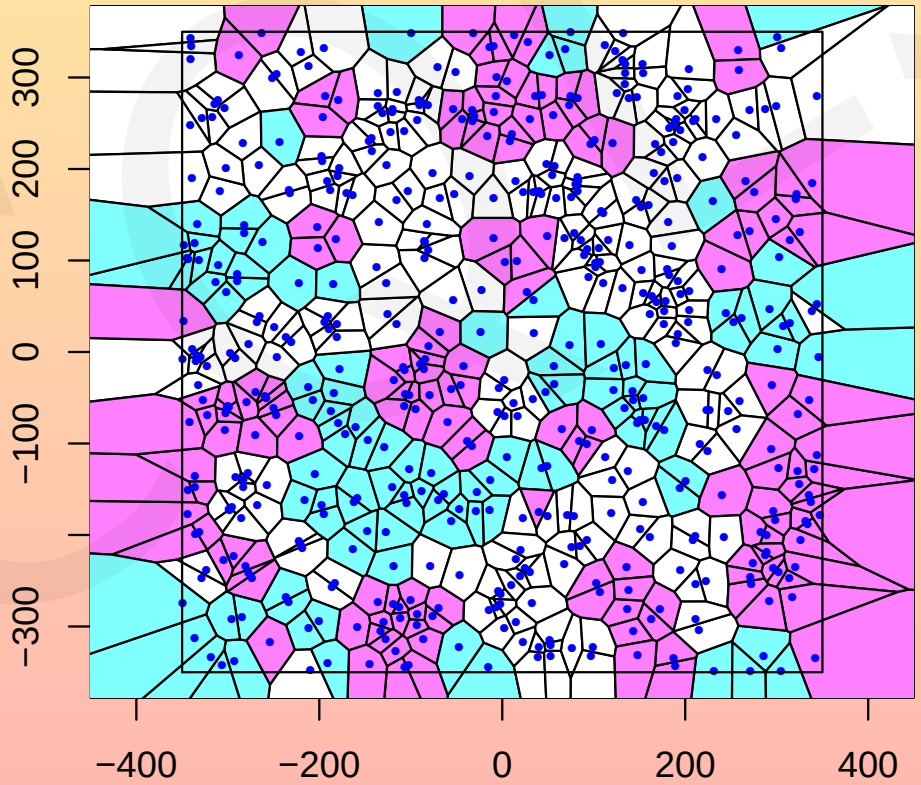
$$V(\varphi) = -2|\varphi| + \sum_{\xi \in \mathcal{P}_2(\varphi)} 2 \times \mathbb{1}_{[0,20[}(\|\xi\|) + 4 \times \mathbb{1}_{[20,80[}(\|\xi\|)$$

```
> ga <- EBGibbs(~(-2)+All2(sum(th*c(l<=20,20<l)),th=c(2,4),range=80))
> # notice that range=80 really fastens the simulation in comparison with
> # ga <- EBGibbs(~(-2)+All2(sum(th*c(l<=20,20<l<=80)),th=c(2,4))
> run(ga)
```

$$V(\varphi) = 2|\varphi| + \sum_{\xi \in \mathcal{D}el_2(\varphi)} 2 \times \mathbb{1}_{[0,20[}(\|\xi\|) + 4 \times \mathbb{1}_{[20,80[}(\|\xi\|)$$

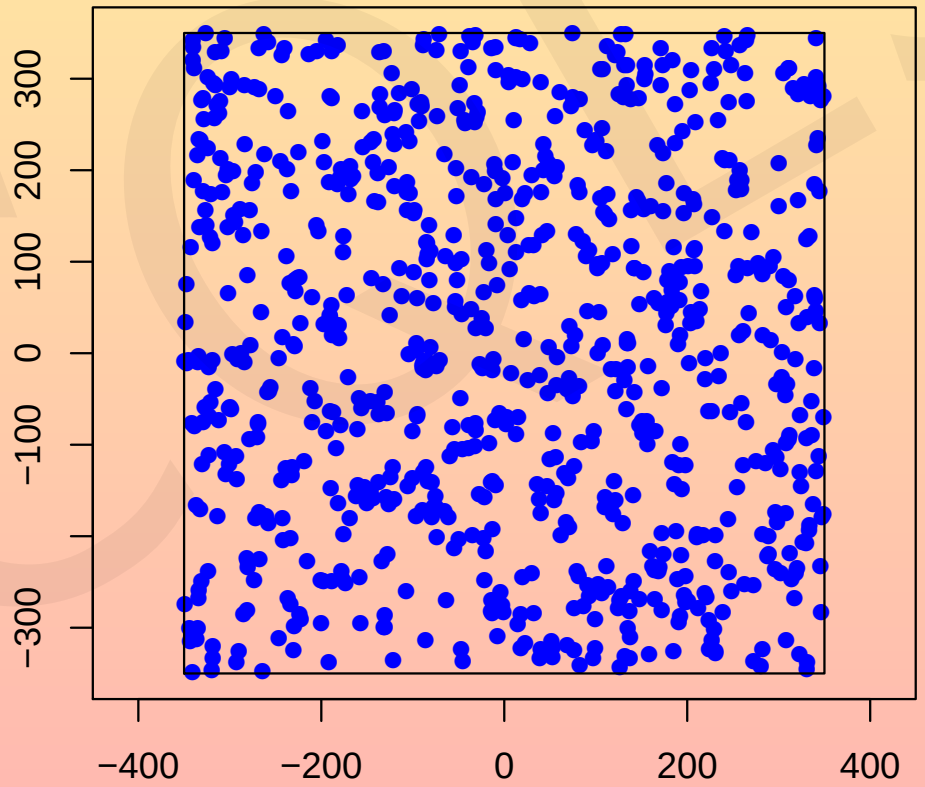
```
> # Same interaction function but restricted to the Delaunay graph!
> gd <- EBGibbs(~ 2 + Del2(th[1]*(l<=20)+th[2]*(20<l & l<=80),th=c(2,4)))
> # which is equivalent to:
> # gd <- EBGibbs(~ 2 + Del2(sum(th*c(l<=20,20<l & l<=80)),th=c(2,4)))
> # No need here to add range=80 because of nearest-neighbours property
> run(gd)
```

```
> gdm<-EBGibbs(~5+Del2(theta*(l2<1600)*abs(v[[1]]$m-v[[2]]$m),theta=2),
+                          marks=EBMarks(m=int(1,1:3)))
> param(gdm)  # parameters could be updated via this method
$Single
[1] 5
$theta
[1] 2
> run(gdm,m=10000) #Rmk: m stands here for the number of iterations!
> plot(gdm, vcCol = m, dvCex = 0.5)
```

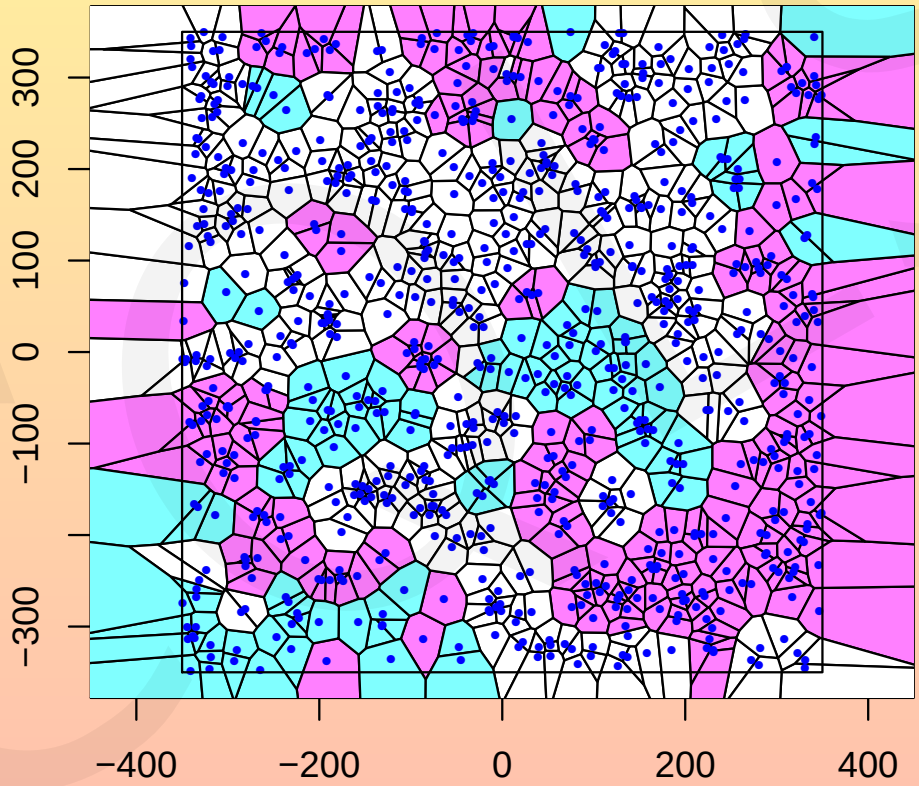


```
> # same result as before but with theta initialized in 2 steps
> gdm2<-EBGibbs(~5+Del2(theta*(l2<1600)*abs(v[[1]]$m-v[[2]]$m)),
+                          marks=EBMarks(m=int(1,1:3)))
> param(gdm2)
$Single
[1] 5
$theta
[1] "Need to be initialized!"
> run(gdm2)
Message d'avis :
In run.EBGibbs(gdm2) :
  theta needs to be initialized first via param method!
> param(gdm2,theta=2)
$theta
[1] 2
> run(gdm2,m=10000) #same result than before with "gdm"
```

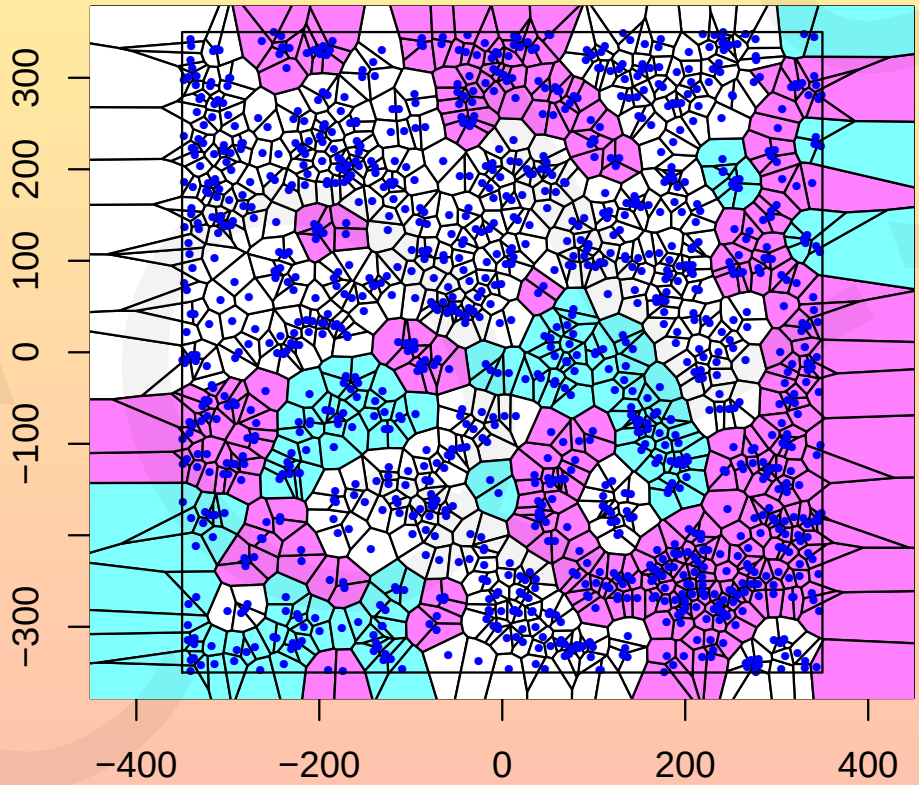
```
> # Possibility to update the parameters
> param(gdm2,theta=3,Single=4)
$Single
[1] 4
$theta
[1] 3
> empty(gdm2) #all the points removed
> run(gdm2) #no plot already done => default one used!
nbPoints: in=413 (500.000000x500.000000),out=835 (700.000000x700.000000)
```



```
> gdm2$sim$m  #number of iterations (default value)
[1] 10000
> plot(gdm2, vcCol = m, dvCex = 0.5)
```

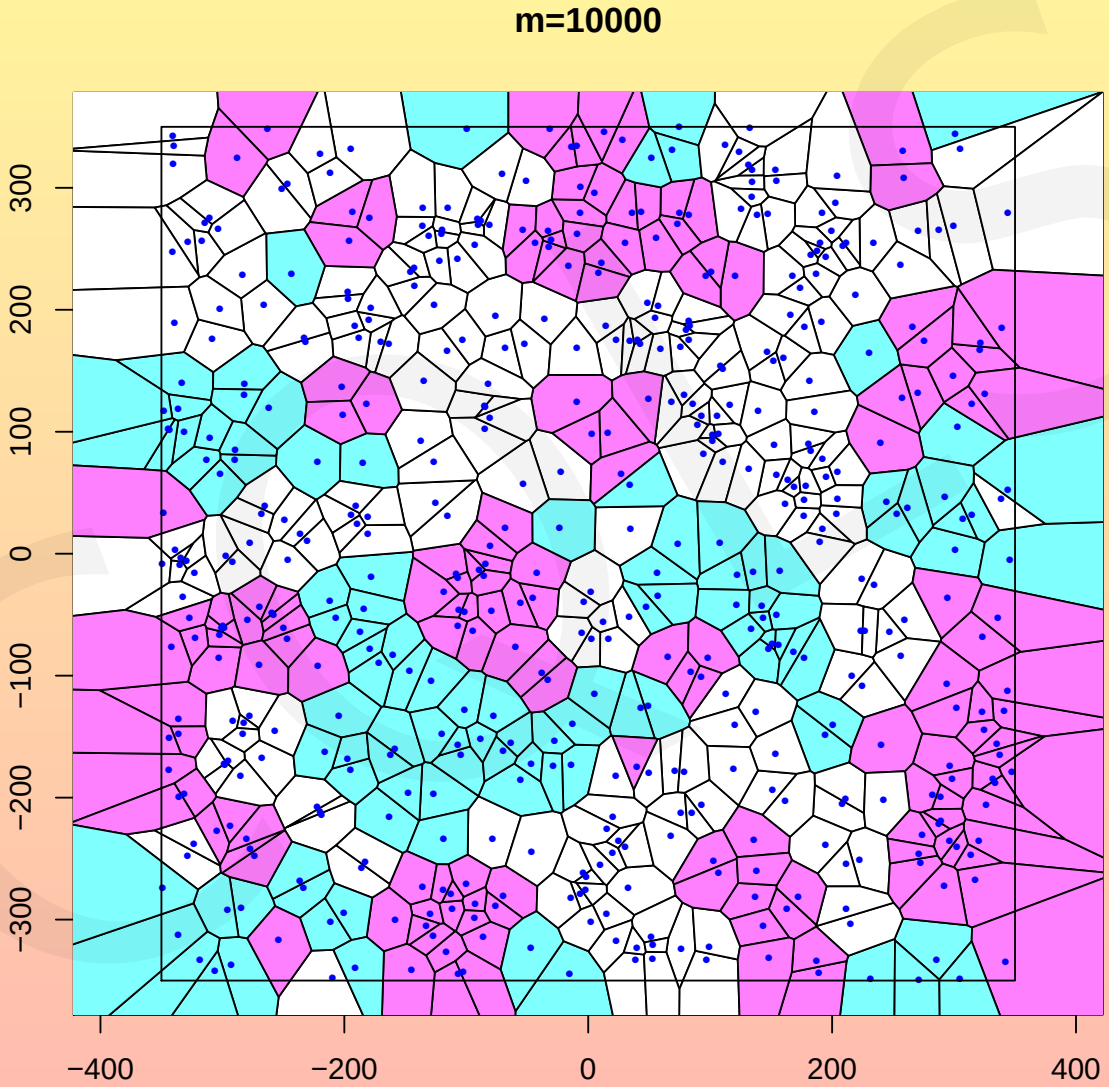


```
> # Interactively continue the simulation (m can be specified)
> run(gdm2)
nbPoints: in=603 (500.000000x500.000000),out=1266 (700.000000x700.000000)
```



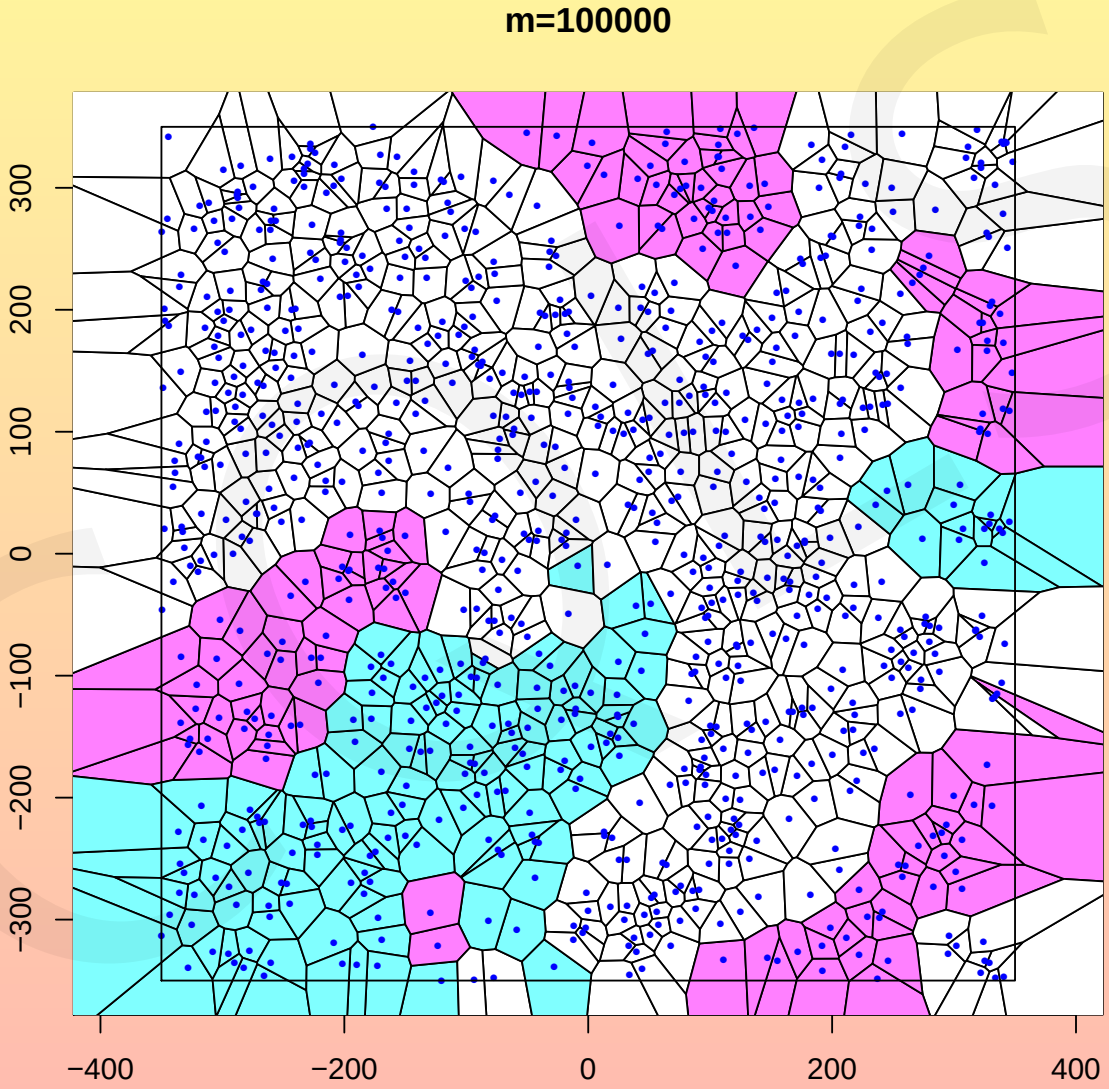
Phase transition detection

```
> run(gdm,m=10000)
```



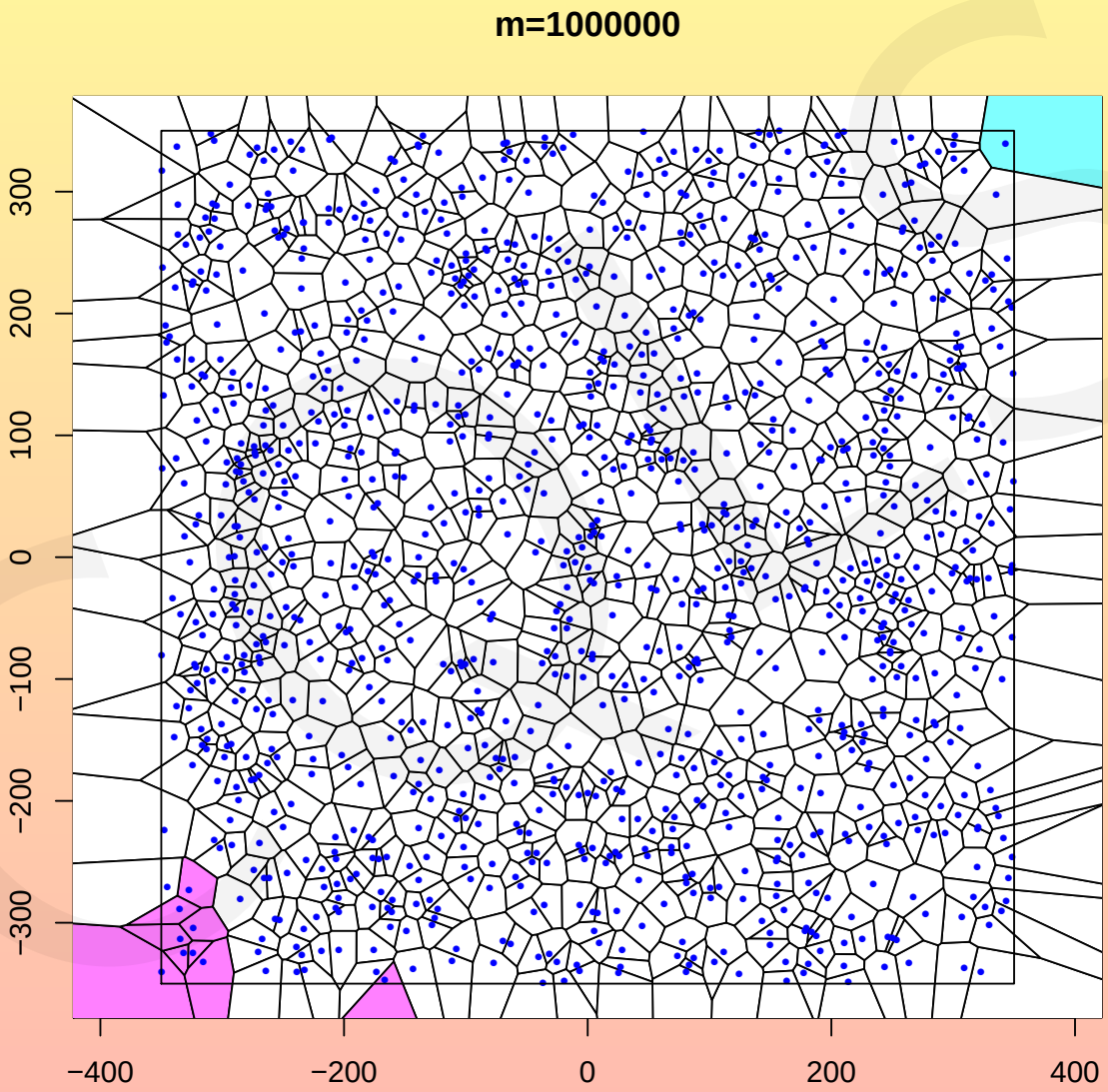
Phase transition detection

```
> run(gdm,m=90000)
```



Phase transition detection

```
> run(gdm,m=900000)
```



Global energy: $V(\varphi) = \sum_{\xi} g(\xi)$

```
> energy(gdm)
[1] 2745
```

Local (pointwise) energy: $V(x|\psi) = V(\psi \cup \{x\}) - V(\psi)$

```
> energy(gdm,1)
[1] 5
```

Local energy: $V(\varphi|\psi) = V(\varphi \cup \psi) - V(\psi)$

```
> energy(gdm,c(1,6,4))
[1] 17
```

Repartition of the (pointwise) local energies:

```
> nrj <- sapply(seq(gdm),function(i) energy(gdm,i))
> table(nrj)
nrj
  3   5   7   9  11
  4 497  27   8   1
```

which point requires a local energy equal to 11 to be inserted?

```
> which(nrj==11)
[1] 261
> energy(gdm,261)
[1] 11
```

Delaunay edges contributing in `energy(gdm,261)`:

```
> infos <- Del2(gdm,261,l2,v)
> infos
$new
           l2 v1_m v2_m
1   4520.6008    3    1
2   3403.7782    2    1
3   4578.8378    1    1
...
7   3942.2716    1    1
8   4811.2488    2    3
9   4270.3575    2    1
10   130.7817    3    1
$old
         l2 v1_m v2_m
1 5914.129    2    1
2 4811.249    2    3
3 5509.102    1    3
4 1650.777    1    2
5 2144.791    3    2
6 4520.601    3    1
7 3403.778    2    1
```

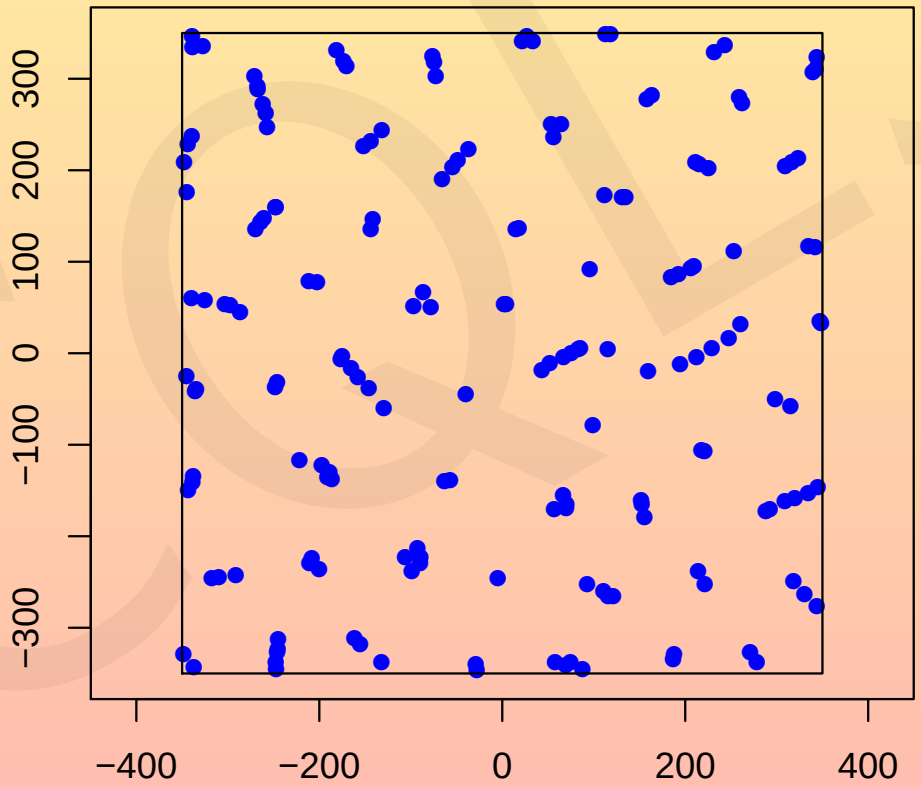
Detailed computation of `energy(gdm,261)`:

```
> sum((infos$new$l2<1600)*abs(infos$new$v1_m - infos$new$v2_m))
[1] 3
> sum((infos$old$l2<1600)*abs(infos$old$v1_m - infos$old$v2_m))
[1] 0
> param(gdm)
$Single
[1] 5
$theta
[1] 2
> 5+(3-0)*2 # Yes!!! since
[1] 11
> energy(gdm,261) #which is computed faster!
[1] 11
```

Plan

This part is **unfortunately** in its early stage since the results are not very stable (**possible explanations**: theory or programming error or not a proper realization ...)

```
> gd <- EBGibbs(~ 2 + Del2(th[1]*(l<=20)+th[2]*(20<l & l<=80),th=c(2,4)))
> run(gd)
```



MPL-Estimation inside the domain $[-250, 250]^2$:

```
> pld <- EBPseudoExpo(gd~Del2(l<=20,20<l & l<=80),weight=TRUE)
> param(gd)
$Single
[1] 2
$th
[1] 2 4
> run(pld,c(0,0,0))
[1] 0 0 0
$par
[1] 2.053993 2.003846 4.276364
$value
[1] 0.00172411
$counts
function gradient
     859      101
$convergence
[1] 1
$message
NULL
[1] 2.053993 2.003846 4.276364
```

Innovation and residual $[-250, 250]^2$:

```
> resd <- EBResid( # interaction first
+                  gd~Del2(Th[1]*(l<=20)+Th[2]*(20<l & l<=80)),
+                  1,                    #first functional
+                  del2(l<=20),          #second one
+                  del2(20<l & l<=80) #third one
+         )
> run(resd,Single=0,Th=c(0,0))                          # innovation
[1] 0.999676 0.376956 4.174736
> run(resd,Single=2,Th=c(2,4))                          # innovation
[1] -6.009023e-06 -1.011641e-05  1.654284e-05
> sum(run(resd,Single=2,Th=c(2,4))^2)                   # Takacs-Fiksel
[1] 4.121156e-10
> run(resd,Single=pld$par[1],Th=pld$par[-1])            # residual
[1] -1.669128e-05 -1.202242e-05 -1.084119e-06
> sum(run(resd,Single=pld$par[1],Th=pld$par[-1])^2) # Takacs-Fiksel
[1] 4.243127e-10
```

Takacs-Fiksel estimation in $[-250, 250]^2$:

```
> tkd <- EBTakacsFiksel( # interaction first
+                   gd~Del2(Th[1]*(l<=20)+Th[2]*(20<l & l<=80)),
+                   1,                    #first functional
+                   del2(l<=20),          #second one
+                   del2(20<l & l<=80) #third one
+         )
> param(tkd,Single=0,Th=c(0,0))          # need initialization
> run(tkd)                               # pretty slow!!!!
...
> run(tkd)                               # run (several times)
$par
  Single      Th1      Th2
1.824091 2.080801 5.141669
$value
[1] 6.933614e-17
$counts
function gradient
       1        1
$convergence
[1] 0
$message
NULL
```

Plan

Todo

A lot of stuff has to be done:

- Better compatibility with the huge `spatstat` `R` package
- The package is still experimental and needs a lot of stabilization
- More interaction type based on the k-nearest neighbours and Gabriel graphs
- Towards to 3D (and higher dimension)
- **Final step:** `R` documentation
- ...