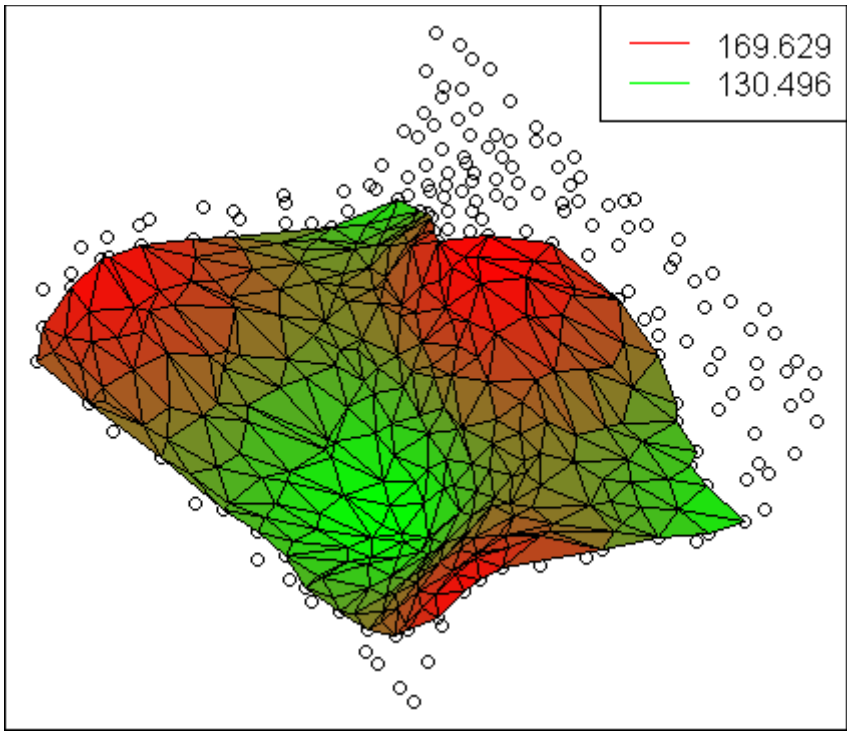
А.Н.Каретин.

# Применение пакета статистических вычислений R 2.13.0 для решения геодезических задач по наблюдению за поверхностями.




Typeset by OpenOffice.org Write

2011г.

*Введение.*

Все программные продукты, разработанные для обработки результатов наблюдений за поверхностями, являются дорогостоящими и закрытыми. Помимо этого, данные программные продукты не предоставляют необходимой деталировки результата, а в большинстве случаев данные программные продукты не позволяют решать простые с геометрической точки зрения задачи. То есть, эти продукты решают определенные задачи, но не позволяют, в силу непрозрачности данных продуктов, варьировать элементы этих задач. Что в свою очередь сужает производительность данных программных продуктов, а в определенных случаях делает их просто неэффективными. Это относится в первую очередь к такому продукту как CREDO_TER. Этот программный продукт выдает настолько утрированный результат, что невозможно убедиться в его надежности или дорабатывать данный результат.

В данной методичке будет показан не только уровень технологий, распространяемых по лицензии GPL, но и их невероятную прозрачность и полноту представляемого результата.

Автор данной методики желает представить читателям программный продукт, распространяющийся по лицензии GPL, и именуемый R 2.13.0. Данный программный продукт является системой статистических вычислений с очень удобным представлением данных в виде списков и таблиц.

---

«И не будет после нас тьмы...»

А.Н.Каретин.

30.06.2011г.

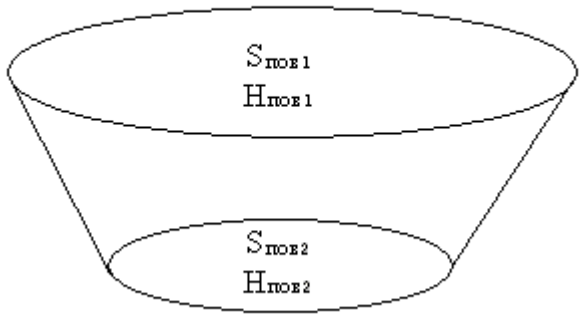
*Теоретические основы задач обработки наблюдений за поверхностями.*

Задачи обработки результатов наблюдений за поверхностями ставят своей целью получение для любых поверхностей, как бы сложны они не были, определенных общих характеристик. К таким характеристикам относится уровень поверхности ($H_{пов}$), уровенная площадь поверхности ($S_{пов}$), элементы среднего уклона поверхности [$i_{пов}$, $a(i_{пов})$], искривление поверхности ($j_{пов}$), максимальный уклон поверхности ($i_{max}$).

Перечисленные выше характеристики поверхностей необходимы только для масштабных задач, относящихся к поверхностям в целом. Эти характеристики оказывают влияние на все задачи меньшего масштаба.

Наиболее часто данные характеристики требуются для расчета объема грунта ($V_{гр}$), заключенного между двумя поверхностями. Поверхности при этом обычно представлены разными наборами точек, наблюдаемыми в разные периоды времени.

Схема расчета в этом случае выглядит так.




$$V_{гр} = \frac{1}{3}\left(S_{пов1} + S_{пов2} + \sqrt{S_{пов1} \cdot S_{пов2}}\right)(H_{пов1} - H_{пов2})$$

Как видно имея общие характеристики поверхностей задача расчета объема грунта тут же решается одной простой формулой и не является сложной.

Сложным здесь является получение общих характеристик для сложных поверхностей. И этому будет посвящена следующая глава.

*Получение общих характеристик поверхностей (Sпов, Hпов).*

Поверхность, заданная наборами точек, может быть также представлена набором непересекающихся треугольников. Построение данного набора треугольников называется триангуляцией. Распространенным и повсеместно применяемым способом триангуляции является триангуляция Делоне. Эта триангуляция соблюдает набор определенных геометрических правил и хорошо отражает триангулируемую поверхность. Рассмотрение в данной статье алгоритма триангуляции производится не будет, упомяну лишь, что мною применяется инкрементальный алгоритм от заданной границы с быстрой сортировкой.

После того, как произведена триангуляция (точки объедены в треугольники), для каждого треугольника можно найти площадь и уровень его центра:

$$S_{тр} = \frac{1}{2}\left|(x_1 + x_2)(y_1 - y_2) + (x_2 + x_3)(y_2 - y_3) + (x_3 + x_1)(y_3 - y_1)\right|$$

$$H_{тр} = \frac{1}{3}(H_1 + H_2 + H_3)$$

Получив площади и уровни всех треугольников, можно получить площадь и уровень всей поверхности:

$$S_{пов} = \sum S_{тр}$$

$$H_{пов} = \frac{\sum (S_{тр} \cdot H_{тр})}{S_{пов}}$$

Если произвести подобные расчеты для двух поверхностей, относящихся к одному объекту, можно спокойно решить задачу поставленную в предыдущей главе.

*Применение пакета R 2.13.0 для получения общих параметров поверхностей.*

Для получения общих параметров поверхностей и их визуализации, автором данной методички был набран набор функций на языке R. Листинги данных функций представлены в приложении данной методички. В данной главе будет объяснено только их назначение. Функции разделены на две библиотеки:

ahull2d_v1.1.r - функции вычислительной геометрии
geo_ter_v1.1.r - функции обработки геоданных

Для работы с этими функциями надо поместить их в какую-нибудь папку ("C:\R\geoter\R", например), запустить программу R и в ее консоли набрать следующие команды:

```
setwd("C:/R/geoter/R")
getwd()
dir()
source("ahull2d_v1.1.r")
source("geo_ter_v1.1.r")
```

Далее устанавливаем рабочей папку с данными, которые могут быть представлены текстовым файлом с числами, разделенными пробелом, либо файлом csv (числа разделяются "точкой с запятой"). Например:

```
setwd("C:/R/geoter/data")
getwd()
dir()
```

В зависимости от того, как представлены данные применяем соответствующую команду:

```
nxyz<-read.table("test.nxyz.txt")
nxyz<-read.csv("test.nxyz.csv")
names(nxyz)<-c("N","X","Y","Z")
nxyz
```

Разделяем таблицу для удобства работы с данными:

```
x<-nxyz$X
y<-nxyz$Y
z<-nxyz$Z
```

Для выпуклой оболочки определяем границу:

```
gr<-convexhull2d(x,y)
```

Для данных с определенной границей точки границы либо располагаем в начале файла данных в порядке обхода против часовой стрелки, тогда:

```
gr<-seq(n)
gr
```

где n – количество точек в границе, либо указываем их напрямую:

```
gr<-c(N1,N2,...,Nk)
```

где N1,N2,...,Nk – номера точек в общем списке.

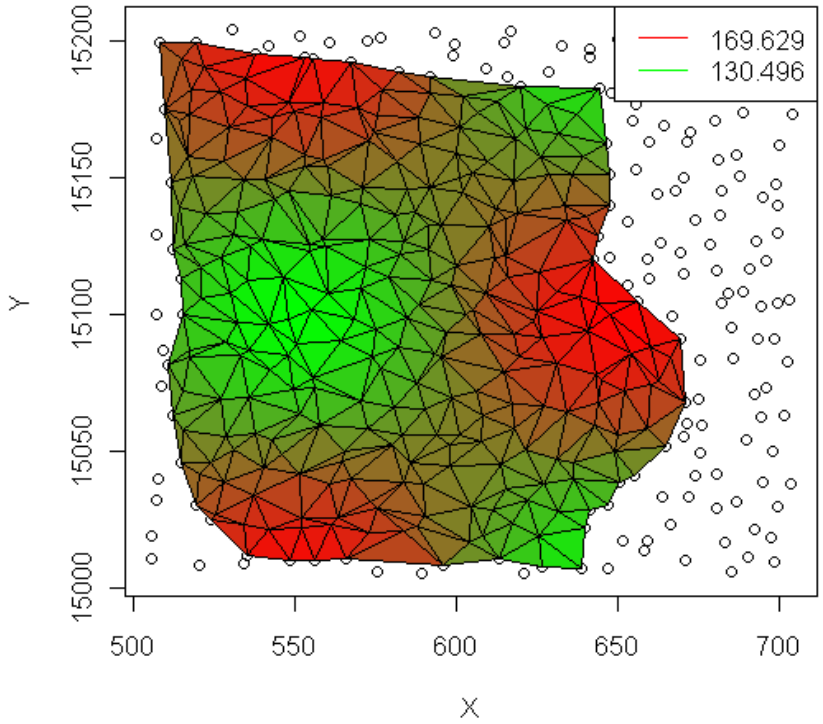
Определившись с границей производим триангуляцию:

```
tri<-delaunay2d(x,y,gr)
tri
```

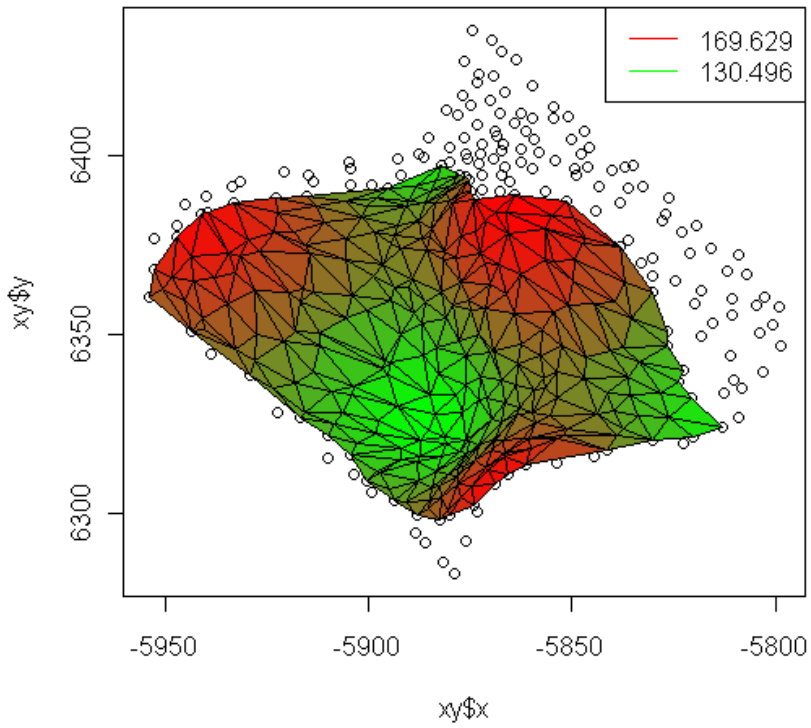
Определившись со структурой поверхности определяем ее характеристики:

```
povSZ<-tri.area.fast(tri,x,y,z)
povSZ
```

Ну и, конечно же, немного графики
`tri.plot.2d(tri,x,y,z)`




`tri.plot.3d(tri,x,y,z)`




Сохраняем результат, составляем отсчет. И все.

ПРИЛОЖЕНИЕ.

Листинги функций R.

ahull2d_v1.1.r:

```
# ahull2d v6.1 - add-ins for SciLab 4.1.2
# made in: <<TerraNoNames [http://mykaralw.narod.ru/]>>
# 29.06.2011.

# Functions:
# function tumb <- tumbling2d(x1,y1,x2,y2,xt,yt)   --- O(N)
# function gr <- convexhull2d(X,Y)                 --- O(N*log(N))
# function tri <- delaunay2d(X,Y,gr)               --- O(N*log(N)~N^2)


tumbling2d<-function(x1,y1,x2,y2,xt,yt)
{
#determination sides tumbling:
#tumb>0 - to the left;
#tumb<0 - to the right.
#EXAMPLE:
#    pright<-(tumbling2d(0,0,10,0,c(2,5,7),c(6,-1,4))<0)
#O(N)
# Copyright Akon&TerraNoNames
      dyt <- yt - y1
      dxt <- xt - x1
      dyp <- y2 - y1
      dxp <- x2 - x1
      tumb <- dxp * dyt - dxt * dyp
      return(tumb)
}

convexhull2d<-function(Xc,Yc)
{
#Searching for of the proturberant shell.
#The algorithm Grehema.
#point - vector point shells.
#EXAMPLE:
#    gr<-convexhull2d(X,Y)
#O(N*log(N)): N=1000points, W=2000GH, Memory=2000Gb, Time=0.3c.
# Copyright Akon&TerraNoNames
      nx <- length(Xc)
      ny <- length(Yc)
      if (nx==ny)
      {
            n<-nx
            if (n < 4)
            {
                  point <- seq(n)
            }
            else
            {
                  xt <- Xc[1]
                  yt <- Yc[1]
                  for (i in 2:n)
                  {
                        if (xt > Xc[i])
                        {
                              xt <- Xc[i]
                              yt <- Yc[i]
```

```
                              zt <- i
                        }
                  }
                  dx <- Xc - xt
                  dy <- Yc - yt
                  dl <- sqrt(dx*dx+dy*dy)
                  tt <- dy/dl
                  tt[zt] <- 0
                  tts <- sort(tt,index.return=TRUE)
                  ttsix <- tts$ix
                  sols<-seq(n)
                  sols[1] <- zt
                  sols[2] <- ttsix[1]
                  sols[3] <- ttsix[2]
                  tis <- 3
                  for (i in 3:n)
                  {
                        tis <- tis + 1
                        sols[tis] <- ttsix[i]
                        while((tumbling2d(Xc[sols[tis - 2]],Yc[sols[tis -
2]],Xc[sols[tis - 1]],Yc[sols[tis - 1]],Xc[sols[tis]],Yc[sols[tis]]) <= 0) &
(tis > 3))
                        {
                              tis <- (tis - 1)
                              sols[tis] = sols[tis + 1]
                        }
                  }
                  point = sols[1:tis]
            }
      }
      else
      {
            tis <- 0
            point <- "incompatible dimensions X and Y"
      }
      return(point)
}

#function [counttr,tri]=delaunay2d(Xr,Yr,grr)
delaunay2d<-function(Xc,Yc,grc)
{
#The triangulation Delone.
#The method of direct searching for.
#Iterration sort.
#tri - table triangle.
#EXAMPLE:
#    gr<-convexhull2d(X,Y)
#    tri<-delaunay2dis(X,Y,gr)
#O(N*log(N)~N^2): N=1000points, W=2000GH, Memory=2000Gb, Time=24c.
# Copyright Akon&TerraNoNames
      nx <- length(Xc);
      ny <- length(Yc);
      ngr <- length(grc);
      grt <- grc
      tric <- c()
      if (nx == ny)
      {
            n <- nx
            nt <- n * 10;
            i <- ngr;
            if (grc[1] != grc[ngr])
            {
                  grt <- c(grc,grc[1])
                  i <- (i+1)
```

```
            }
            counta <- (i-1)
            bega <- 1
            alive1 <- grt[1:counta]
            alive2 <- grt[2:i]
            alive3 <- rep(-1,counta)
            alive4 <- rep(0,counta)
            avec <- matrix(rep(0,n*n),nrow=n,ncol=n)
            counttr <- 0
            tric1<-c()
            tric2<-c()
            tric3<-c()
            for (i in 1:counta)
            {
                  avec[alive1[i],alive2[i]] <- i
                  avec[alive2[i],alive1[i]] <- i
            }
            ap <- rep(1,n)
            while (counta >= bega)
            {
                  i <- alive1[bega]
                  j <- alive2[bega]
                  k <- alive3[bega]
                  compla <- alive4[bega]
                  hn <- (-1)
                  if (compla == 0)
                  {
                        x1 <- Xc[i]
                        y1 <- Yc[i]
                        x2 <- Xc[j]
                        y2 <- Yc[j]
                        x0 <- (x1+x2)/2
                        y0 <- (y1+y2)/2
                        r2t <- (Xc-x0)*(Xc-x0)+(Yc-y0)*(Yc-y0)
                        r2si <- sort(r2t,index.return=TRUE)
                        r2s <- r2si$x
                        nts <- r2si$ix
                        t1 <- x1*x1+y1*y1
                        t2 <- x2*x2+y2*y2
                        t3 <- Xc*Xc+Yc*Yc;
                        sc <- (-x1)*(Yc-y2)+x2*(Yc-y1)+Xc*(y1-y2)
                        sa <- (-t1)*(Yc-y2)+t2*(Yc-y1)+t3*(y1-y2)
                        sb <- (-t1)*(Xc-x2)+t2*(Xc-x1)+t3*(x1-x2)
                        tc <- tumbling2d(x1,y1,x2,y2,Xc,Yc)
                        hi <- 0
                        while ((hi < n) & (hn < 0))
                        {
                              hi <- (hi+1)
                              h <- nts[hi]
                              if ((h != i) & (h != j) & (h != k) & (ap[h] > 0) &
(tc[h] > 0) & (sc[h] != 0))
                              {
                                    xc <- 0.5*sa[h]/sc[h]
                                    yc <- -0.5*sb[h]/sc[h]
                                    r2c <- (x1-xc)*(x1-xc)+(y1-yc)*(y1-yc)
                                    hn <- h
                                    r2 <- (Xc-xc)*(Xc-xc)+(Yc-yc)*(Yc-yc)
                                    for (li in 1:n)
                                    {
                                          l <- nts[li]
                                          if ((l != i) & (l != j) & (l != k) &
(l != h))
                                          {
                                                if ((tc[l] > 0) & (r2[l] < r2c))
```

```
                                        {
                                              hn <- (-1)
                                        }
                                  }
                            }
                      }
                }
                alive4[bega] <- hn
          }
          if (hn > 0)
          {
                k <- 0
                if (avec[i,hn] > 0)
                {
                      k <- avec[i,hn]
                }
                if (k == 0)
                {
                      counta <- (counta+1)
                      alive1 <- c(alive1,i)
                      alive2 <- c(alive2,hn)
                      alive3 <- c(alive3,j)
                      alive4 <- c(alive4,0)
                      avec[i,hn] <- counta
                      avec[hn,i] <- counta
                }
                else
                {
                      alive4[k] <- j
                      ap[i] <- (-1)
                }
                k <- 0
                if (avec[j,hn] > 0)
                {
                      k <- avec[j,hn]
                }
                if (k == 0)
                {
                      counta <- (counta+1)
                      alive1 <- c(alive1,hn)
                      alive2 <- c(alive2,j)
                      alive3 <- c(alive3,i)
                      alive4 <- c(alive4,0)
                      avec[j,hn] <- counta
                      avec[hn,j] <- counta
                }
                else
                {
                      alive4[k] <- i
                      ap[j] <- (-1)
                }
                counttr <- (counttr+1)
                tric1 <- c(tric1,i)
                tric2 <- c(tric2,j)
                tric3 <- c(tric3,hn)
                h <- 1
          }
          bega <- (bega+1)
    }
    tri <- data.frame(A=tric1,B=tric2,C=tric3)
}
else
{
    tri <- "incompatible dimensions X and Y"
```

```
      }
      return(tri)
}

print("Library ahull2d_v1.1 load...")
print(" Functions:")
print("   tumb <- tumbling2d(x1,y1,x2,y2,xt,yt)   --- O(N)")
print("   gr <- convexhull2d(X,Y)                 --- O(N*log(N))")
print("   tri <- delaunay2d(X,Y,gr)               --- O(N*log(N)~N^2)")
```

geo_ter_v1.1.r:

```
# geo_ter v1.1 - add-ins for R 2.13.0
# made in: <<TerraNoNames [http://mykaralw.narod.ru/]>>
# 25.06.2011.

# Functions:
# function [xp,yp,zp,ztri]=triparam(Tri,X,Y,Z)                        --- O(N)
# function [stri,ztri]=triarea(Tri,X,Y,Z)                             --- O(N)
# function [sarea,zarea]=triareafill(stri,ztri)                       --- O(N)
# function [sarea,zarea]=triareafast(Tri,X,Y,Z)                       --- O(N)
# function [X]=invertrows(X0)                                         --- O(N)
# function [cmap]=rbcolormap(n)                                       --- O(N)
# function [cmap]=selectcolormap(csh,n)                               --- O(N)
# function [X,Y,n,minz,maxz,xps,yps]=triplotbase(Tri,X,Y,Z)           --- O(N)
# function []=triplot(Tri,X,Y,Z,csh)                                  --- O(N)
# function [pk,dlt]=pkdelta(X,Y,X0,Y0,sina,cosa)                      --- O(N)
# function [n,dltl,zl,dlts,zps]=triplot3dbase(Tri,X,Y,Z,gmm,tt,kz)    --- O(N)
# function []=triplot3d(Tri,X,Y,Z,csh,gmm,tt,kz)                      --- O(N)


tri.param <- function(Tri,X,Y,Z)
{
#base parameter triangle.
#EXAMPLE:
#    gr=convexhull2d(X,Y)
#    tri=delaunay2d(X,Y,gr)
#    [xp,yp,zp,ztri]=triparam(tri,X,Y,Z);
#O(N)
# Copyright Akon&TerraNoNames
 m <- length(Tri)
 n <- length(Tri$A)
 triA <- Tri$A
 triB <- Tri$B
 triC <- Tri$C
 xpA <- X[triA]
 ypA <- Y[triA]
 zpA <- Z[triA]
 xpB <- X[triB]
 ypB <- Y[triB]
 zpB <- Z[triB]
 xpC <- X[triC]
 ypC <- Y[triC]
 zpC <- Z[triC]
 ztri <- (zpA+zpB+zpC)/3
 triT <- data.frame(xA=xpA,yA=ypA,xB=xpB,yB=ypB,xC=xpC,yC=ypC,zT=ztri)
 return(triT)
}

tri.area <- function(Tri,X,Y,Z)
{
#areas triangle.
#EXAMPLE:
```

```
#    gr <- convexhull2d(X,Y)
#    tri <- delaunay2d(X,Y,gr)
#    [stri,ztri] <- triarea(tri,X,Y,Z)
#O(N)
# Copyright Akon&TerraNoNames
 tri.data <- tri.param(Tri,X,Y,Z)
 xA <- tri.data$xA
 xB <- tri.data$xB
 xC <- tri.data$xC
 yA <- tri.data$yA
 yB <- tri.data$yB
 yC <- tri.data$yC
 stri <- 1/2*abs((xA+xB)*(yA-yB)+(xB+xC)*(yB-yC)+(xC+xA)*(yC-yA))
 ztri <- tri.data$zT
 triS <- data.frame(sT=stri,zT=ztri)
 return(triS)
}

tri.area.full <- function(tridata)
{
#global area triangle.
#EXAMPLE:
#    gr <- convexhull2d(X,Y)
#    tri <- delaunay2d(X,Y,gr)
#    triSZ <- tri.area(tri,X,Y,Z)
#    [sarea,zarea] <- tri.area.fill(triSZ)
#O(N)
# Copyright Akon&TerraNoNames
 sarea <- sum(tridata$sT)
 zarea <- sum(tridata$sT*tridata$zT)/sarea
 triSZ <- data.frame(S=sarea,Z=zarea)
 return(triSZ)
}

tri.area.fast <- function(Tri,X,Y,Z)
{
#global area triangle.
#EXAMPLE:
#    gr <- convexhull2d(X,Y)
#    tri <- delaunay2d(X,Y,gr)
#    [sarea,zarea] <- tri.area.fast(tri,X,Y,Z)
#O(N)
# Copyright Akon&TerraNoNames
 tridata <- tri.area(Tri,X,Y,Z)
 triSZ <- tri.area.full(tridata)
 return(triSZ)
}

hot.colors <- function(n)
{
      m <- trunc(n/3)
      m3 <- n-2*m
      pas <- 0.75

      r <- c(seq(0,pas,len=m),rep(1,n-m))
      g <- c(rep(0,m),seq(0,pas,len=m),rep(1,n-2*m))
      b <- c(rep(0,2*m),seq(0,pas,len=m3))
      rgb(r,g,b)
}

rb.colors <- function(n)
{
      m <- trunc(n)
      r <- seq(0,1,len=m)
```

```
      b <- r[m:1]
      g <- (1-sqrt(r*r+b*b))/2
      rgb(r,g,b)
}

rb.color <- function(a,b,i)
{
      r <- (i-a)/(b-a)
      b <- 1-r
      g <- (1-sqrt(r*r+b*b))/2
      rgb(r,g,b)
}

rg.color <- function(a,b,i)
{
      r <- (i-a)/(b-a)
      g <- 1-r
      b <- (1-sqrt(r*r+g*g))/2
      rgb(r,g,b)
}

gray.color <- function(a,b,i)
{
      r <- (i-a)/(b-a)
      b <- r
      g <- r
      rgb(r,g,b)
}

tri.plot.2d <- function(Tri,X,Y,Z)
{
 m <- length(Tri)
 n <- length(Tri$A)
 triA <- Tri$A
 triB <- Tri$B
 triC <- Tri$C
 xpA <- X[triA]
 ypA <- Y[triA]
 zpA <- Z[triA]
 xpB <- X[triB]
 ypB <- Y[triB]
 zpB <- Z[triB]
 xpC <- X[triC]
 ypC <- Y[triC]
 zpC <- Z[triC]
 ztri <- (zpA+zpB+zpC)/3
 minz <- min(ztri)
 maxz <- max(ztri)
 plot(X,Y)
 for (i in 1:n)
 {
  tr.x <- c(xpA[i],xpB[i],xpC[i],xpA[i])
  tr.y <- c(ypA[i],ypB[i],ypC[i],ypA[i])
  tr.c <- rg.color(minz,maxz,ztri[i])
  polygon(tr.x,tr.y,col=tr.c)
 }
 minz <- trunc(1000*minz+0.5)/1000
 maxz <- trunc(1000*maxz+0.5)/1000
 legend("topright", legend = c(maxz, minz),lty = c(1,1),col =
c(rg.color(minz,maxz,maxz),rg.color(minz,maxz,minz)),bg = 'white')
}

tri.plot.3d <- function(Tri,X,Y,Z,teta=0.5,zeta=0.5)
{
```

```
 tt <- teta
 zt <- zeta
 res <- matrix(c(tt,-tt,-zt,0,tt,tt,zt,0,zt,zt,zt,0,0,0,0,1),nrow=4,ncol=4)
 m <- length(Tri)
 n <- length(Tri$A)
 triA <- Tri$A
 triB <- Tri$B
 triC <- Tri$C
 xpA <- X[triA]
 ypA <- Y[triA]
 zpA <- Z[triA]
 xpB <- X[triB]
 ypB <- Y[triB]
 zpB <- Z[triB]
 xpC <- X[triC]
 ypC <- Y[triC]
 zpC <- Z[triC]
 ztri <- (zpA+zpB+zpC)/3
 minz <- min(ztri)
 maxz <- max(ztri)
 xy <- trans3d(x,y,z,res)
 plot(xy)
 for (i in 1:n)
 {
  tr.x <- c(xpA[i],xpB[i],xpC[i],xpA[i])
  tr.y <- c(ypA[i],ypB[i],ypC[i],ypA[i])
  tr.z <- c(zpA[i],zpB[i],zpC[i],zpA[i])
  tr.c <- rg.color(minz,maxz,ztri[i])
  trs.xy <- trans3d(tr.x,tr.y,tr.z,res)
  trs.x <- trs.xy$x
  trs.y <- trs.xy$y
  polygon(trs.x,trs.y,col=tr.c)
 }
 minz <- trunc(1000*minz+0.5)/1000
 maxz <- trunc(1000*maxz+0.5)/1000
 legend("topright", legend = c(maxz, minz),lty = c(1,1),col =
c(rg.color(minz,maxz,maxz),rg.color(minz,maxz,minz)),bg = 'white')
}

print("Library geo_ter_v1.1 load...")
```

ДЛЯ ЗАМЕТОК.

URL.

| | |
|---|---|
| http://www.R-project.org | – ресурс The R Foundation for Statistical Computing |
| http://www.astonshell.com/rus | – ресурс хорошего текстового редактора Bred. |
| http://www.openoffice.org/ | – ресурс OpenOffice.org. |
| http://www.foxitsoftware.com/ | – ресурс Foxit Software, содержит Foxit Reader, для просмотра и печати PDF. |
| http://www.cs.wisc.edu/~ghost/ | – ресурс Ghostscript, системы для просмотра, конвертирования и печати EPS, EPSI, PS, PDF. |
| http://www.ghostgum.com.au/ | – ресурс Ghostgum Software, содержит GSView, позволяющий работать с системой Ghostscript в визуальном режиме. |
| http://windjview.sourceforge.net | – ресурс WinDjView, программы для просмотра DjVu. |

«И не будет после нас тьмы...»

А.Н.Каретин.

30.06.2011г.