The "sampSurf" Class

Jeffrey H. Gove*
Research Forester
USDA Forest Service
Northern Research Station
271 Mast Road
Durham, New Hampshire 03824 USA
e-mail: jgove@fs.fed.us or e-mail: jhgove@unh.edu

Wednesday 15th February, 2012
1:40pm

Contents

1 Introduction

The sampling surface is the culmination of the different classes and methods that have been developed and discussed in other vignettes. It can be created in any one of several ways (i.e., via different constructors); here we illustrate the overall process of creating a "sampSurf" object using one of the constructors (others are detailed below). Please refer to the individual vignette documentation for more details on each of the methods used below.

First we need a "Tract" object (in this case, a "bufferedTract" object) within which we can create the sampling surface...

```
R> tra = Tract(c(x=100, y=100), cellSize = 0.5, units = 'metric',
```

*Phone: (603) 868-7667; Fax: (603) 868-7604.

```
+             description = 'a 1-hectare tract')
R> (btr = bufferedTract(10, tra))


------------------------------------------------------------
a 1-hectare tract
------------------------------------------------------------
Measurement units = metric
Area in square meters = 10000 (1 hectares)

class       : bufferedTract
dimensions  : 200, 200, 40000  (nrow, ncol, ncell)
resolution  : 0.5, 0.5  (x, y)
extent      : 0, 100, 0, 100  (xmin, xmax, ymin, ymax)
coord. ref. : NA
values      : in memory
min value   : 0
max value   : 0

Buffer width =  10
```

Here we have created a 200×200 cell raster grid with resolution of 0.5 meters (i.e., 1 hectare), with origin at $(0, 0)$ meters. Then we create a buffered tract object with a 10-meter buffer internal to the tract. Both versions have all data values as zero by default to begin so we can build a sampling surface up.

Next, just create a few down logs whose centers lie within the buffer and place them in a "downLogs" container, then make sausage sampling inclusion zones from these logs with plot radius of five meters, and store them in a "downLogIZs" container. The final step is trivial in concept, just accumulate all of the inclusion zones with the `sampSurf` constructor...

```
R> dlogs = downLogs(5, btr@bufferRect)
R> izsSA = downLogIZs(lapply(dlogs@logs, 'sausageIZ', plotRadius=5))
R> ssSA = sampSurf(izsSA, btr)


Number of logs in collection = 5
Heaping log: 1,2,3,4,5,


R> summary(ssSA)


Object of class: sampSurf
------------------------------------------------------------
```

```
sampling surface object
------------------------------------------------------------

Inclusion zone objects: sausageIZ
Measurement units = metric
Number of logs = 5
True log volume = 1.1409335 cubic meters
True log length = 32.05 meters
True log surface area = 19.858703 square meters
True log coverage area = 6.3178041 square meters
True log biomass = NA
True log carbon = NA

Estimate attribute: volume
Surface statistics...
  mean = 1.1417537
  bias = 0.00082015978
  bias percent = 0.071884975
  sum = 45670.148
  var = 24.217512
  st. dev. = 4.9211292
  cv % = 431.01496
  surface max = 37.683285
  total # grid cells = 40000
  grid cell resolution (x & y) = 0.5 meters
  # of background cells (zero) = 37212
  # of inclusion zone cells = 2788
```

We can see from the summary statistics by comparing the mean of the surface with the true volume for all logs, that the method is unbiased[1]. Finally, we are ready to plot the surface...

```
R> plot(ssSA, useImage=FALSE)
```

The result is shown in Figure 1.

[1]There will always be a small non-zero amount listed in the "bias" component, but this does not mean there is a bias in a particular method. This has to do with the finite approximation we are making to the surface. The smaller the grid cell size, the smaller the "bias" will be. This is true of any simulation method and should not be erroneously perceived as a weakness of this particular method.

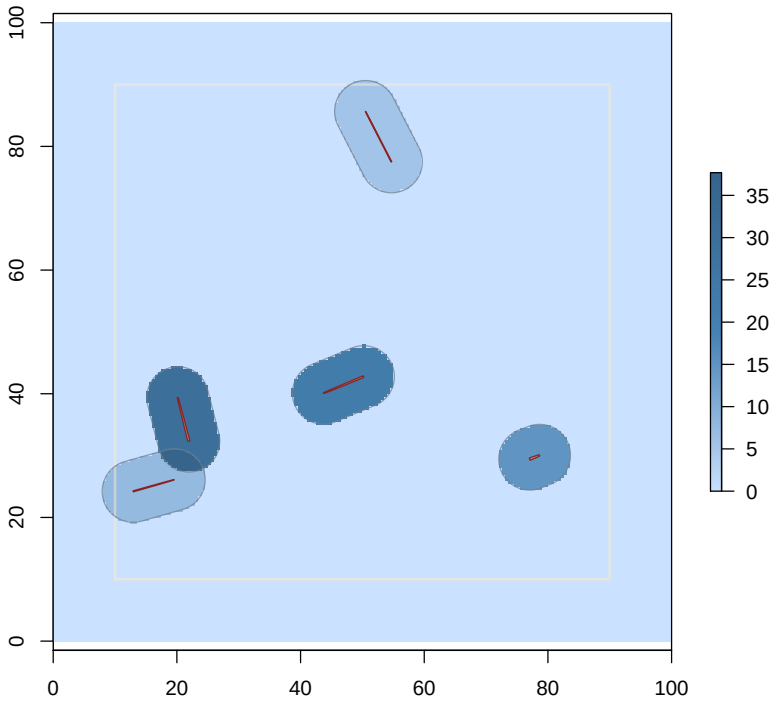

Figure 1: Simple generic "sampSurf" object with some random logs.

2 The "sampSurf" Class

Now that we have shown the basic individual steps to creating a sampling surface we formally introduce the class itself and its constructors.

The base class is defined with the slots...

```
R> showClass('sampSurf')


Class "sampSurf" [package "sampSurf"]

Slots:

Name:  description izContainer       tract    estimate   surfStats
```

```
Class:   character izContainer       Tract   character        list
```

2.1 Class slots

- *description*: Some descriptive text about this class.
- *izContainer*: An "izContainer" subclass object, which is a collection of "InclusionZone" subclass objects such as "downLogIZs" (see vignette entitled: *"The InclusionZone Class"* for examples).
- *tract*: The underlying "Tract" (or subclass) object which will hold the sampling surface.
- *estimate*: The attribute estimate for the tract; i.e., volume ("volume"), or number of stems ("Density"). The full set of attributes can be viewed by issuing the command: `.StemEnv$puaEstimates` within R.
- *surfStats*: A `list` object with the summary statistics for the sampling surface for attribute in the `estimate` slot.

2.2 More Details

There are just a few things that go on behind the scenes within the construction of the sampling surface object that are good to know (without having to look at the code). The object constructor has two main steps...

1. Create "InclusionZoneGrid" objects from each of the individual sausage "InclusionZone" objects within the "downLogIZs" container.
2. Use `heapIZ` to accumulate the individual "InclusionZoneGrid" objects from the first step into the tract.

The above is applied within a simple loop, one object at a time. Then when the "heaping" has been completed, the "sampSurf" object is constructed. It may take some time to construct the object because there is a fair bit of computation going on within these two steps.

3 Object Creation: Other Constructors

The first constructor is shown in the example above. It takes as the signature both an "izContainer" subclass object and a "Tract" object. The second constructor for sampling surface objects generates

the stems (down logs or standing trees), inclusion zones, and surface from scratch. It takes the number of stems and a "Tract" object, along with the type of inclusion zone to generate and simulates the surface from that. This method is simpler if no collection of stem objects is available. An example follows for the sausage and standUp sampling protocols...

```
R> ssSA = sampSurf(2, btr, iZone = 'sausageIZ', plotRadius=5,
+                  buttDiams=c(30,50), startSeed=102)


Number of logs in collection = 2
Heaping log: 1,2,


R> summary(ssSA)


Object of class: sampSurf
------------------------------------------------------------
sampling surface object
------------------------------------------------------------

Inclusion zone objects: sausageIZ
Measurement units = metric
Number of logs = 2
True log volume = 1.3466069 cubic meters
True log length = 11.58 meters
True log surface area = 13.675338 square meters
True log coverage area = 4.3518839 square meters
True log biomass = NA
True log carbon = NA

Estimate attribute: volume
Surface statistics...
  mean = 1.3515718
  bias = 0.0049648839
  bias percent = 0.36869585
  sum = 54062.872
  var = 75.190628
  st. dev. = 8.6712529
  cv % = 641.56806
  surface max = 68.194591
  total # grid cells = 40000
  grid cell resolution (x & y) = 0.5 meters
  # of background cells (zero) = 38906
  # of inclusion zone cells = 1094
```

```
R> ssSU = sampSurf(2, btr, iZone = 'standUpIZ', plotRadius=5,
+                  buttDiams=c(30,50), startSeed=102)


Number of logs in collection = 2
Heaping log: 1,2,


R> summary(ssSU)


Object of class: sampSurf
------------------------------------------------------------
sampling surface object
------------------------------------------------------------

Inclusion zone objects: standUpIZ
Measurement units = metric
Number of logs = 2
True log volume = 1.3466069 cubic meters
True log length = 11.58 meters
True log surface area = 13.675338 square meters
True log coverage area = 4.3518839 square meters
True log biomass = NA
True log carbon = NA

Estimate attribute: volume
Surface statistics...
  mean = 1.3446755
  bias = -0.0019313799
  bias percent = -0.14342567
  sum = 53787.022
  var = 133.6143
  st. dev. = 11.559165
  cv % = 859.62485
  surface max = 121.50701
  total # grid cells = 40000
  grid cell resolution (x & y) = 0.5 meters
  # of background cells (zero) = 39373
  # of inclusion zone cells = 627
```

The examples above generate the same set of logs for each example so the we can directly compare the bias and precision of the methods. This is accomplished by passing `startSeed` with the same random number seed in each call (it gets passed on to the `downLogs` constructor). Note especially

that we must pass along any arguments that internal constructors require. For example, within this version of `sampSurf`, we know from the above example (and some knowledge of how the class objects are constructed) that we will be creating "downLogs" and "downLogIZs" objects within this routine. Therefore, we must, for example, pass the `plotRadius` argument so that the `sausageIZ` constructor will know what size circular plot is required—there is no default for this argument.

3.1 The Chainsaw method

This protocol is somewhat of an anomaly because the inclusion zone is a point as explained in Gove and Van Deusen (2011). However, in sampling surface simulations, all points (grid cells) within the sausage-shaped ("sausageIZ") inclusion zone will intersect the log, and this is the whole log inclusion zone that we want to simulate for each log. So when generating a "sampSurf" object for the chainsaw method, we must have a set of "sausageIZ" inclusion zone objects available for the log population first. In the constructor with `signature(object = 'izContainer', tract='Tract')`, the "downLogIZs" object (as the first signature argument) must contain sausage inclusion zones; but in addition, we must tell the constructor that we want to use the chainsaw method to generate the sampling surface within these zones. This is done by specifying `wantChainSaw = TRUE` in the `sampSurf` method call. This is also done for the constructor with `signature(object = 'numeric', tract='Tract')`. Additionally, in this second case, we must also specify `iZone = 'sausageIZ'` in the method call. Two examples, that are not run because of the length of time to construct even one full chainsaw inclusion zone, are shown below for these respective signatures...

```
R> ss.ch1 = sampSurf(izsSA, btr, wantChainSaw=TRUE)
R> ss.ch2 = sampSurf(2, btr, iZone='sausageIZ', wantChainSaw=TRUE,
+                    plotRadius=5, buttDiams=c(40,50))
```

Note especially that we are able to pass along other arguments as needed. For example, in the second invocation, we pass an argument for the determination of the fixed-area plot radius used to generate the sausage inclusion zone, as well as an argument used in sample log generation for `downLogs`.

It can not be stress enough that the chainsaw method is an anomoly with regard to the fixed-area plot methods in that it must visit every cell within the inclusion zone, decide whether there is a plot-log intersection, and then calculate the volume, etc., of the intersected section. This generates a non-constant surface within the inclusion zone and takes a long time to calculate if the resolution is high or the log and plot radius are large, encompassing a large number of grid cells. Other areal sampling methods also have uneven surface heights within the inclusion zone, but are more straightforward in general; these include, e.g., omnibus perpendicular distance sampling (Ducey et al., 2008) and critical point relascope method[2] (Gove et al., 2005), as two examples.

[2]Not yet implemented.

Finally, based on the above, it is an error to specify `iZone='chainSawIZ'` in the second constructor.[3] The reason is that this method is only defined for one point, which is the inclusion zone. When the "sausageIZ" method is specified, the "chainSawIZ" method is applied by the constructor at the center of each grid point within the sausage-shaped inclusion zones. Therefore, it does not make sense to use the "chainSawIZ" method in the call, as it would only apply to the *single* point that it was defined for, not the entire "Tract". Please see Gove and Van Deusen (2011) and *The "InclusionZone" Class* vignette for more details.

4 Plotting With "rgl"

A method has been added to the `raster` package generic `plot3D` for visualization with `rgl`. To display a surface, one must first have the `rgl` package installed. Then simply issue the command on the "sampSurf" class object to display it, and the `rgl` commands to save it to a file as desired...

```
R> require(rgl)
R> plot3D(ssSA)
R> rgl.postscript(paste(getwd(),'/figures/ss3D.ps',sep=''))
R> rgl.close()
```

Figure 2 shows a sampling surface with a population of 25 logs. The sausage method was used to sample the logs on a one hectare tract. This example shows one view of the surface, which may be rotated and magnified as desired under the `plot3D` function used to interface with `rgl`.

5 Other Sampling Methods

Some methods may require different arguments to the "sampSurf" constructor with first signature argument "numeric". In general, the help documentation is the best place to go for more information. Here we show a couple more examples of creating sampling surfaces for different sampling methods. The list is not exahustive compared to what is implemented within the `sampSurf` package. Please see *The "InclusionZone" Class* vignette for the different methods available in the package.

5.1 Point relascope sampling

Here we present an illustration using the point realscope sampling (PRS) method (Gove et al., 1999). It is instructive to compare this constructor to the previous examples...

[3] Neither does it make sense to try to build a collection of these point-zones for the first constructor.

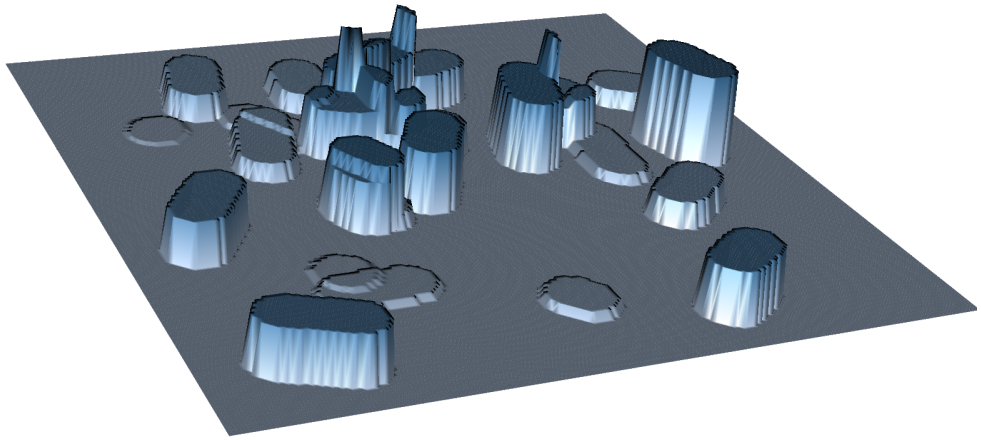

Figure 2: One view of a sampling surface using `rgl`.

```
R> etract = Tract(c(x=100,y=100), cellSize=0.5, units='English')
R> ebuffTr = bufferedTract(15, etract)
R> (angle = .StemEnv$rad2Deg(2*atan(1/2)))

[1] 53.130102

R> prs.as = pointRelascope(angle, units='English')
R> prs.ss = sampSurf(5, ebuffTr, iZone='pointRelascopeIZ', units='English',
+                    prs=prs.as, buttDiams=c(8,16), logLens=c(6,16))

Number of logs in collection = 5
Heaping log: 1,2,3,4,5,

R> summary(prs.ss)

Object of class: sampSurf
-------------------------------------------------------------
```

```
sampling surface object
------------------------------------------------------------

Inclusion zone objects: pointRelascopeIZ
Measurement units = English
Number of logs = 5
True log volume = 41.07627 cubic feet
True log length = 52.68 feet
True log surface area = 161.77659 square feet
True log coverage area = 51.360613 square feet
True log biomass = NA
True log carbon = NA

Estimate attribute: volume
Surface statistics...
  mean = 40.988786
  bias = -0.08748417
  bias percent = -0.21297983
  sum = 1639551.4
  var = 16229.541
  st. dev. = 127.39521
  cv % = 310.80505
  surface max = 723.6507
  total # grid cells = 40000
  grid cell resolution (x & y) = 0.5 feet
  # of background cells (zero) = 35842
  # of inclusion zone cells = 4158
```

The point of the above example is that the sampling surface constructor mentioned required two additional bits of information to be passed on to other routines. The `units` specifies, for example, that the down logs to be created are in "English" units, to go along with the correct units in the "bufferedTract" and "pointRelascope" objects. More importantly, the "pointRelascopeIZ" constructor requires an argument `prs` which is of type "pointRelascope", in order to define the relascope information for the inclusion zones which will be constructed internally within this function call. There is nothing else really new here, and nothing mysterious going on. We are just sending the appropriate information along as required by the constructors for different objects. A plot of this sampling surface is shown in Figure 3. More information for the inclusion zone constructor can be found in the help: `methods?pointRelascopeIZ`.

```
R> plot(prs.ss, axes=TRUE, useImage=FALSE)
```

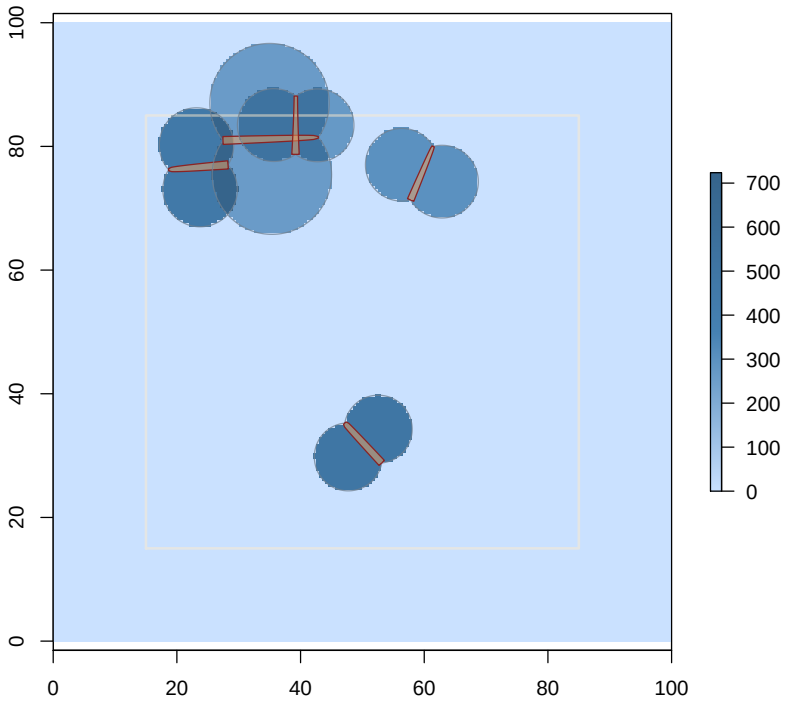

Figure 3: "sampSurf" surface with a few logs under point relascope sampling.

5.2 Perpendicular distance sampling

There are several variants of perpendicular distance sampling (PDS) that are supported in the `sampSurf` pacakge. Here we illustrate only the "canonical" version first described by Williams and Gove (2003). Again, please compare the constructor with that of the previous examples...

```
R> (epds = perpendicularDistance(3, units='English'))
```

```
Object of class: perpendicularDistance
------------------------------------------------------------
perpendicular distance method
------------------------------------------------------------

ArealSampling...
```

```
  units of measurement:  English

perpendicularDistance...
  kPDS factor = 3 per foot [dimensionless] for volume [surface/coverage area]
  volume [surface/coverage area] factor = 7260 cubic feet [square feet] per acre
```

```
R> pds.ss = sampSurf(5, ebuffTr, iZone='perpendicularDistanceIZ', units='English',
+                    pds=epds, buttDiams=c(8,16), logLens=c(6,16))
```

```
Number of logs in collection = 5
Heaping log: 1,2,3,4,5,
```

```
R> summary(pds.ss)
```

```
Object of class: sampSurf
------------------------------------------------------------
sampling surface object
------------------------------------------------------------

Inclusion zone objects: perpendicularDistanceIZ (with PP to: volume)
Measurement units = English
Number of logs = 5
True log volume = 29.517932 cubic feet
True log length = 50.36 feet
True log surface area = 129.27472 square feet
True log coverage area = 41.051313 square feet
True log biomass = NA
True log carbon = NA

Estimate attribute: volume
Surface statistics...
  mean = 29.375
  bias = -0.14293174
  bias percent = -0.48422005
  sum = 1175000
  var = 48096.645
  st. dev. = 219.30947
  cv % = 746.58544
  surface max = 1666.6667
  total # grid cells = 40000
  grid cell resolution (x & y) = 0.5 feet
```

```
# of background cells (zero) = 39295
# of inclusion zone cells = 705
```

Again, because we are using a different sampling method, PDS, the constructor requires a different type of argument passed in the signature. In this case, it is an "ArealSampling" subclass object of class "perpendicularDistance". The result is shown in Figure 4. More information for the inclusion zone constructor can be found in the help: `methods?perpendicularDistanceIZ`.

```
R> plot(pds.ss, axes=TRUE, useImage=FALSE)
```

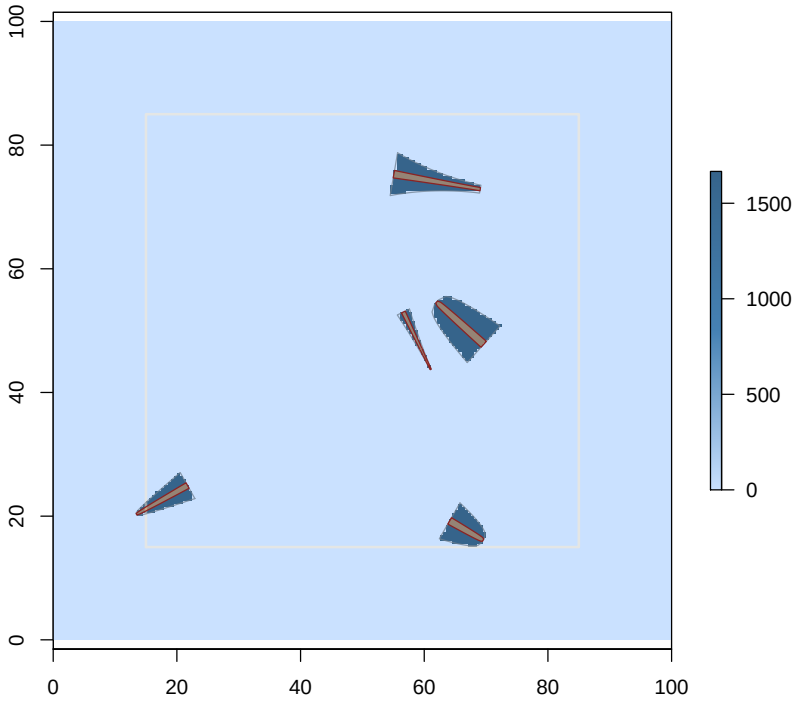

Figure 4: "sampSurf" surface with a few logs under perpendicular distance sampling.

5.3 Horizontal point sampling

Here we present an example using one of the sampling methods that have been implemented for standing trees. Again, we use the simple constructor...

```
R> etract = Tract(c(x=209,y=209), cellSize=0.5, units='English')
R> ebuffTr = bufferedTract(30, etract)
R> aGauge = angleGauge(baf=20, units='English')
R> hps.sse = sampSurf(10, ebuffTr, iZone='horizontalPointIZ', units='English',
+                     angleGauge=aGauge, dbhs=c(6,16), heights=c(16,45))

Number of trees in collection = 10
Heaping tree: 1,2,3,4,5,6,7,8,9,10,

R> summary(hps.sse)

Object of class: sampSurf
------------------------------------------------------------
sampling surface object
------------------------------------------------------------

Inclusion zone objects: horizontalPointIZ
Measurement units = English
Number of trees = 10
True tree volume = 202.37309 cubic feet
True tree basal area = 8.8745073 square feet
True tree surface area = 830.40324 square feet
True tree biomass = NA
True tree carbon = NA

Estimate attribute: volume
Surface statistics...
  mean = 202.28231
  bias = -0.090785392
  bias percent = -0.044860406
  sum = 35343574
  var = 98608.386
  st. dev. = 314.01972
  cv % = 155.23835
  surface max = 1427.2389
  total # grid cells = 174724
  grid cell resolution (x & y) = 0.5 feet
  # of background cells (zero) = 110199
  # of inclusion zone cells = 64525

R> plot(hps.sse, axes=TRUE, useImage=FALSE)
```

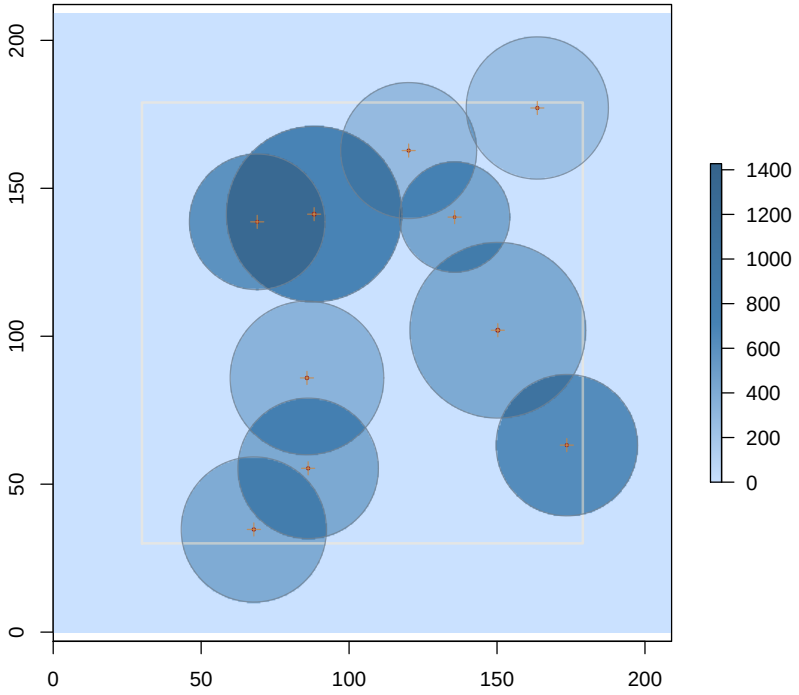

Figure 5: "sampSurf" surface with a small tree population for horizontal point sampling.

6 Summary

Hopefully at this point the idea of how to generate sampling surfaces, and the nuances of signature requirements for the "simple" constructor has been demonstrated. If there is any question, look at the methods help page for the "InclusionZone" subclass you want to use, and see what is required in its signature. The extra argument(s) must be passed to the `sampSurf` constructor in this case.

References

M. J. Ducey, M. S. Williams, J. H. Gove, and H. T. Valentine. Simultaneous unbiased estimates of multiple downed wood attributes in perpendicular distance sampling. *Canadian Journal of Forest Research*, 38:2044–2051, 2008. 8

J. H. Gove and P. C. Van Deusen. On fixed-area plot sampling for downed coarse woody debris. *Forestry*, 84(2):109–117, 2011. 8, 9

J. H. Gove, A. Ringvall, G. Ståhl, and M. J. Ducey. Point relascope sampling of downed coarse woody debris. *Canadian Journal of Forest Research*, 29(11):1718–1726, 1999. 9

J. H. Gove, M. S. Williams, G. Ståhl, and M. J. Ducey. Critical point relascope sampling for unbiased volume estimation of downed coarse woody debris. *Forestry*, 78:417–431, 2005. 8

M. S. Williams and J. H. Gove. Perpendicular distance sampling: an alternative method for sampling downed coarse woody debris. *Canadian Journal of Forest Research*, 33:1564–1579, 2003. 12