The Mirage Method in sampSurf*

J. H. Gove

USDA Forest Service, Northern Research Station, 271 Mast Road, Durham, NH 03824 USA

(603) 868-7667; e-mail: jgove@fs.fed.us

Thursday 18th September, 2014 3:38pm

Contents

1 Introduction

This document covers some of the implementation of the "mirage" method (MM) in `sampSurf`, and presumes a general familiarity with the workings of the package (see the vignettes on the web page in the footnote below for more details). The mirage method is a clever way to correct for issues of boundary slopover (a.k.a. boundary overlap or edge effect) when sampling near the forest edge (Gregoire and Valentine, 2008, p. 223–224). The essential problem is that stems lying near the boundary of the tract can have a portion of their inclusion zones (IZs) falling partly outside of the tract. If one does not correct for this external inclusion zone slopover, then the trees so affected will not have their full probability of being selected into the sample when sample points are restricted to landing within the tract boundaries. This problem was recognized early on in forest sampling by various authors. A simple correction was also suggested quite early by allowing sample points to fall a little ways outside the tract into a "buffer" region (Masuyama, 1953). The buffer must be large enough to encompass the entire portion of any inclusion slopover. Later, Schmid-Haas (1969) developed the mirage method, as a boundary correction method based on a simple, but elegant geometric reflection transformation. Both methods are simple to use and provide unbiased corrections.

An extension to Schmid-Haas's method has recently been developed (Lynch and Gove, 2014) to

*R `sampSurf` package vignette series paper: `http://sampsurf.r-forge.r-project.org/`.

handle Monte Carlo-based sampling methods that produce spatially referenced estimates resulting in predictable surface geometry within individual stem inclusion zones.[1] This extension is termed the "generalized mirage method," and is the method used in `sampSurf`.

The way `sampSurf` is designed, the type of "Tract" object that is used for a simulation conveys the information to the `sampSurf` constructor function on how one would like to treat boundary overlap, if any. For example, if a simple "Tract" object is passed, no boundary overlap correction will be done and the estimates could be biased in the presence of slopover. When a "bufferedTract" object is used, it is assumed that all of the IZs fall inside the overall tract, which includes the buffer area, and what amounts to Masuyama's method is used.[2] Similarly, when a "mirageTract" object is used for the simulation, the MM will be used to correct for any slopover. In this way, one can use the same population of "Stem" objects and their associated "InclusionZone" objects with each of the different correction methods.

Of course, when making comparisons of the above correction methods, one must be careful to either adjust for different size tracts, or use tracts of the same extents. We show later in § 4 one method for rectifying estimates (more details can be found in Gove, 2014).

1.1 Preliminaries

Prior to actually illustrating some mirage examples, we will demonstrate the importance of making sure that all "Stems" (or portions thereof) in the population lie inside the tract. This is especially important for "downLog" objects, which can extend across the tract boundary. In the § 2 we will introduce a function that will facilitate both checking for these circumstances and correcting them if need be.

The following tract and logs will be used to demonstrate some of the concepts of the mirage method within `sampSurf`.

```
R> ptr = Tract(c(x = 20, y = 20), cellSize = 1)
R> dls.1 = vector('list',2)
R> dls.1[[1]] = downLog(buttDiam = 20, topDiam = 5, logLen = 8,
+                       logAngle = pi/8, centerOffset = c(x = 22, y = 18))
R> dls.1[[2]] = downLog(buttDiam = 20, topDiam = 5, logLen = 10,
+                       logAngle = 2.2*pi/4, centerOffset = c(x = 10, y = 4))
R> dls.1 = downLogs(dls.1)
```

[1] Examples include critical height sampling and the Monte Carlo variant of distance limited sampling.

[2] Please note that it is up to the user to insure that all inclusion zones fall within the buffer, if they do not, then a similar bias will result as if a regular tract is used. The `sampSurf` constructor does warn if any inclusion zones slop outside the tract.

2 Clipping Stems

When using the buffer method, it is assumed that the internal buffer will be chosen to be large enough by the user that all stems will fall inside the tract, and that and their inclusion zones will ultimately fall inside the buffer region. This presupposes a knowledge of the population (which we possess) and of the factors influencing the size of the inclusion zones (which we possess), and is not an unreasonable requirement. When we speak of using a "bufferedTract" object within `sampSurf` in accordance with Masuyama's method, a little clarification might be helpful. Masuyama's method adds an external buffer surrounding the tract, within which, sample points may fall and select stems that are *inside* the tract. In `sampSurf`, an internal buffer is used to create a "bufferedTract" object. One first creates a "Tract" object with overall extents of the entire area, including the buffer. A "bufferedTract" is then created from this object by specifying the size of the buffer (in feet or meters, depending on the units) to be placed inside, or internal to, this overall tract area. Thus, the actual tract where stems reside will be this internal portion of the overall "Tract" object which is surrounded by the buffer. In other words to create a tract of a certain size, with a buffer of say, 10 m, first create a "Tract" object that is 10 m larger on each side than the desired tract. Then create a "bufferedTract" object from this with a 10 m buffer. The portion internal to the buffer is our real tract where stems lie, and the buffer contains (fully) any inclusion zones that slopover.

The mirage method, on the other hand, folds the portions of the inclusion zones that extend externally to the tract back into the tract about the boundary—much like origami. For this method to work, no sections of stems should fall outside the tract, or if they do, only the section lying inside the tract should be used in the construction of the inclusion zone for the object. This is illustrated in Figure 1a. Note that the log in the northeast section of the tract is actually not in the tract. This is because the center of the log lies outside the tract, and it is this point that is used by the system to determine whether a log is inside the boundary. So technically, this log, while part of the population, is not within the tract and will cause an error when building the sampling surface. In addition, with both logs under sausage sampling (Gove and Van Deusen, 2011), some portion of the inclusion area outside the tract will fold back correctly, but will result in "phantom" points with no tally inside the tract, which must be miraged regardless.

In order to generate Figure 1a, we first generate the inclusion zones for the logs under sausage sampling and PDS (Williams and Gove, 2003) (with probability proportional to coverage area—see the `pdsType` constructor argument).

```
R> dls.1.saus = downLogIZs(dls.1, iZone = 'sausageIZ', plotRadius = 3)
R> plot(dls.1.saus, axes=TRUE, xlim=c(0,30))
R> (pds.1 = perpendicularDistance(30))
```

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

```
------------------------------------------------------------

ArealSampling...
  units of measurement:  metric

perpendicularDistance...
  kPDS factor = 30 per meter [or dimensionless] for volume [surface/coverage area]
  volume [surface/coverage area] factor = 166.66667 cubic meters [square meters] per hectare
```

```
R> dls.1.pds = downLogIZs(dls.1, iZone = 'perpendicularDistanceIZ',
+                         pds = pds.1, pdsType = 'coverageArea')
R> plot(dls.1.pds, add=TRUE, izColor=NA)
R> plot(perimeter(ptr), add=TRUE, lty='dashed')
```

One simple answer to this problem is to truncate the logs so that only that portion of the log falling within the tract is counted as part of the population. This also resolves the question of whether to include a log in the population only if its base lies inside the tract (it fell outwards—the top log in Figure 1), and not when the log falls into the tract (the bottom log). It also makes sense to tally only those portions of the logs within the tract on an inventory, unless perhaps one is tracking dead wood over time. The following uses the `clipStemsToTract` method to cut the logs at the tract boundary and retain only those portions internal to the tract.[3] Note that we must recreate the inclusion zones for these "new" clipped logs, since their dimensions have changed.

```
R> dls.2 = clipStemsToTract(dls.1, ptr)$stems
R> dls.2.saus = downLogIZs(dls.2, iZone = 'sausageIZ', plotRadius = 3)
R> plot(dls.2.saus, axes=TRUE, xlim=c(0,30))
R> dls.2.pds = downLogIZs(dls.2, iZone = 'perpendicularDistanceIZ',
+                         pds = pds.1, pdsType = 'coverageArea')
R> plot(dls.2.pds, add=TRUE, izColor=NA)
R> plot(perimeter(ptr), add=TRUE, lty='dashed')
```

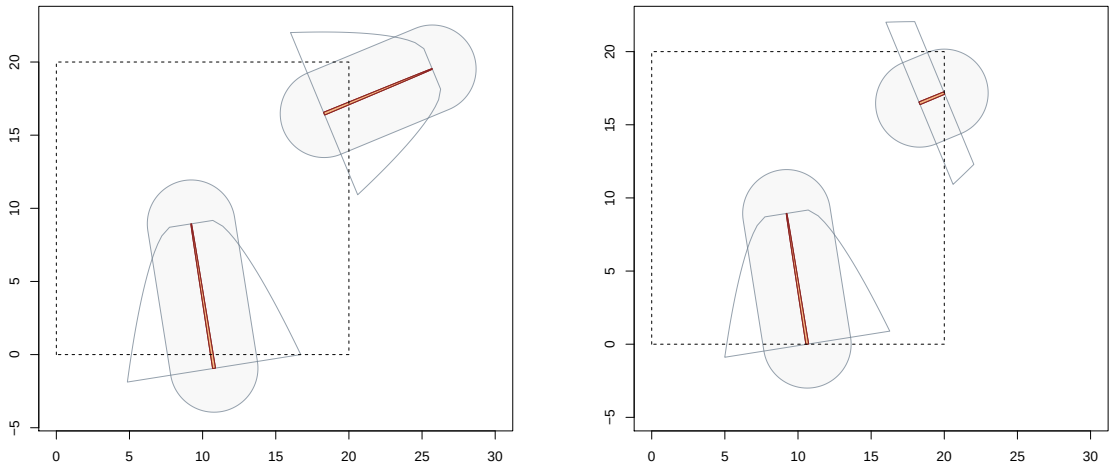
2.1 Return structure

The `clipStemsToTract` function returns a list with the following structure...

- *df*: A data frame noting which stems are in, out or intersect (for logs) the boundary, and where they intersect (N, S, E, W border).[4] Note especially that this data frame always has as

[3] Please see `?clipStemsToTract` for more details on this method and the `list` object it returns.

[4] A log can intersect more than one boundary.



(a) Full

(b) Clipped

Figure 1: Example showing (a) external portions of logs and (b) clipped logs under sausage and PDS (with probability proportional to coverage area) sampling.

many rows as the *original* input stem container has stems. In addition, if all stems lie inside, this will be `NA`.

- *status*: A vector with the total number of stems, the number falling inside the tract, outside the tract, and for logs, intersecting.
- *stems*: If `checkOnly=TRUE` was passed, the original collection; if `!checkOnly`, the new collection with any stems outside removed. For logs, any that intersected are now new log objects clipped to the boundary. In addition, if all stems lie inside the tract, this will be `NA`.

3 Mirage Method

The mirage method is implemented within `sampSurf` when building the "InclusionZoneGrid" object for each stem.[5] At this point we are determining the actual values of the surface *for the individual stem* at each grid cell. Therefore, it is the logical place to re-pile attribute density (e.g., Gregoire and Valentine, 2008, p. 93) within the tract when there is slopover from any portion of the inclusion zone lying outside the boundary.

[5] This step is automatically done as part of building a sampling surface with the `sampSurf` constructor.

In order to facilitate this, some new classes and constructors have been created. First, a new "mirageTract" subclass is required Gove (2013a), the new methods written for constructing "InclusionZoneGrid" subclass objects (see below) will work with a "mirageTract" object and know how to reflect the external portions of the inclusion zone (if any) back into the tract. The `sampSurf` constructors key on this "Tract" subclass and will choose the correct methods for implementing the mirage method on the inclusion zone.

Secondly, the extra information required to reflect inclusion zones about "mirageTract" boundaries is handled using a new "mirageInclusionZoneGrid" class with associated `izGridMirage` constructor method (Gove, 2013b). As long as logs have been clipped to the boundary, the constructor is capable of working on either "downLog" or "standingTree" objects.

In the following, we first create a "mirageTract" object; notice that it is created from an existing "Tract" object. Then the `izGridMirage` constructor method is applied, creating an object of class "mirageInclusionZoneGrid".

```
R> mtr = mirageTract(ptr)
R> mpds.cl1 = izGridMirage(dls.2.pds@iZones[[1]], mtr,
+                          truncateOverlap = FALSE)
R> class(mpds.cl1)
```

```
[1] "mirageInclusionZoneGrid"
attr(,"package")
[1] "sampSurf"
```

The class, and thus the object generated from the constructor has some extra information in it beyond what is in the superclass that is detailed in Gove (2013b). Suffice it to say that one slot, `izGrid.extended`, contains an "InclusionZoneGrid" object that keeps the "extended" tract grid cells used to encompass the external portion of the inclusion zones (slopover). If one calls `izGridMirage` with the argument `truncateOverlap = FALSE` (default), then the cells external to the tract boundary, but within the slopover portion of the inclusion zone, will retain their estimates. It is these estimates that are are folded back onto the matching internal cells[6]. Alternatively, if `truncateOverlap = TRUE`, then these external cells are set to zero. The default is used below so that we can visually compare the external values with the internal surface.

To recap, the actual "mirageInclusionZoneGrid" object is the "miraged" version with no external data retained—everything is clipped to the tract boundary; but one slot contains the full "InclusionZoneGrid" object that records the extended tract with external grid information retained for comparison if desired. Please note that the object in the `izGrid.extended` slot will have too much total attribute density in it because it contains the external (if `truncateOverlap = FALSE`) grid

[6] That is, the estimates associated with these external cells are that portion of the overall attribute density that gets "re-piled" back into the tract, as mentioned above.

cell estimates too: it is not meant to be used for anything other than instruction or comparison. The actual "mirageInclusionZoneGrid" object is what contains the correct folded, reflected attribute density and will give the correct estimates for all grid cells internal to the tract, this is the version that will always be used in creating a sampling surface.

The code below plots the actual "mirageInclusionZoneGrid" object under PDS for this example (Figure 2a). Note that no grid cells external to the tract have been retained as is the normal case in `sampSurf`.

```
R> plot(mpds.cl1, gridCenters=TRUE)
R> plot(perimeter(mtr), add=TRUE, border='red')
```

The following shows how to plot the `izGrid.extended` slot, illustrating what the extended surface looks like (Figure 2b). Note that the attribute density (volume by default) has been folded back correctly into the tract under mirage. Both the external portions and their respective reflections are also highlighted with the polygons contained within the "mirageInclusionZoneGrid" object. (See Gove (2013b) or the system help for more details on the slots.)

```
R> plot(mpds.cl1, tract = mtr, showExtended = TRUE,
+      showReflectedSlivers = TRUE, gridCenters = TRUE)
```

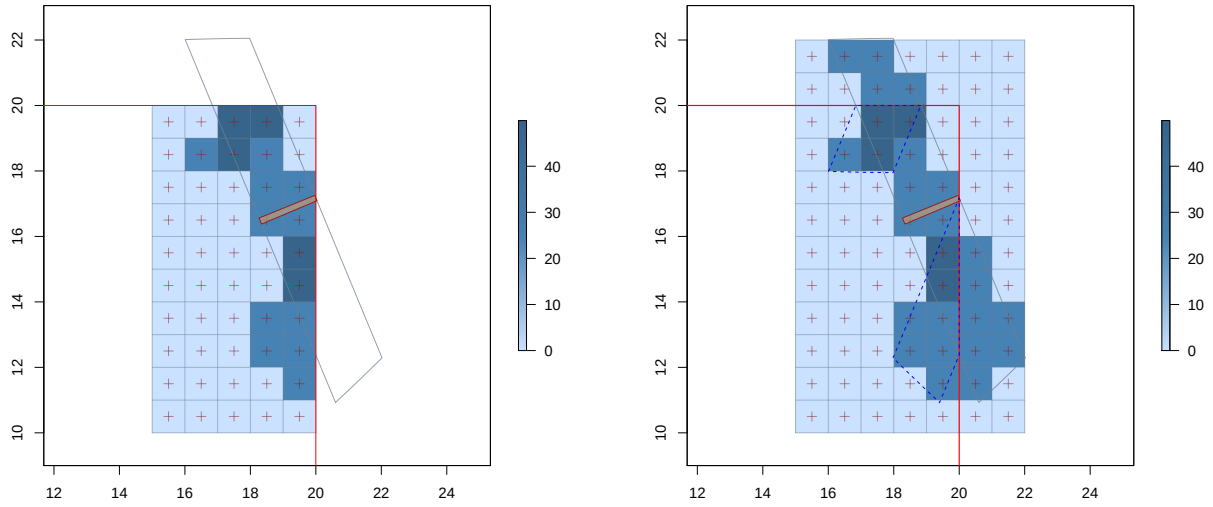
In Figure 2 we can see that the miraged portions of the inclusion zones do not fold completely back onto the internal portion of zone, creating "phantom" areas within the tract where a cruiser would need to mirage even zero-tally points. These phantom areas were first noted in the case of point relascope sampling (Gove et al., 1999), where a version of the 'walkthrough' method, known as boundary reflection, was proposed as a solution. However, as will be discussed elsewhere, walkthrough fails on this example, so even with the phanton zones, mirage is a better correction in this kind of situation.

4 Mirage Sampling Surfaces

The background information has been discussed, now we can compute a sampling surface using mirage on the log population and tract that we have been using previously, under PDS.

```
R> ss.m = sampSurf(dls.2.pds, mtr)
```

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



(a) Clipped (b) Extended

Figure 2: An example with PDS (with probability proportional to coverage area) showing both (a) the actual clipped "mirageInclusionZoneGrid" object and (b) the extended object in the `izGrid.extended` slot with external portions retained and reflected zones illustrated (dashed).

```
R> summary(ss.m)


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

Inclusion zone objects: perpendicularDistanceIZ (with PP to: coverageArea)
Measurement units = metric
Number of logs = 2
True log volume = 0.1899435 cubic meters
True log length = 10.885076 meters
True log surface area = 4.9136449 square meters
True log coverage area = 1.5640183 square meters
True log biomass = NA
True log carbon = NA

Estimate attribute: volume
```

```
Surface statistics...
  mean = 0.18648547
  bias = -0.00345803
  bias percent = -1.8205572
  sum = 74.594188
  var = 0.14980461
  st. dev. = 0.387046
  cv % = 207.54754
  surface max = 1.9963533
  total # grid cells = 400
  grid cell resolution (x & y) = 1 meters
  # of background cells (zero) = 315
  # of inclusion zone cells = 85
```

Now, as a comparison, make a tract that is just large enough to contain the inclusion zones without any slopover. Then compute the sampling surface for the case where no boundary correction is required.

```
R> bbox(dls.2.pds)
```

```
          min       max
x  4.99359789 22.021477
y -0.89327872 22.051850
```

```
R> ltr = Tract(matrix(c(0,-1,23,23), nrow = 2, dimnames = list(c('x','y'),
+             c('min','max'))), cellSize = 1)
R> ss.1 = sampSurf(dls.2.pds, ltr)
```

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

```
R> summary(ss.1)
```

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

Inclusion zone objects: perpendicularDistanceIZ (with PP to: coverageArea)
```

```
Measurement units = metric
Number of logs = 2
True log volume = 0.1899435 cubic meters
True log length = 10.885076 meters
True log surface area = 4.9136449 square meters
True log coverage area = 1.5640183 square meters
True log biomass = NA
True log carbon = NA

Estimate attribute: volume
Surface statistics...
  mean = 0.18648547
  bias = -0.00345803
  bias percent = -1.8205572
  sum = 102.93998
  var = 0.17750157
  st. dev. = 0.42130936
  cv % = 225.92074
  surface max = 1.3774838
  total # grid cells = 552
  grid cell resolution (x & y) = 1 meters
  # of background cells (zero) = 460
  # of inclusion zone cells = 92
```

Notice that the areas of the two tracts are different. Because we compare total estimates (not per unit area) in `sampSurf`, we must re-scale the statistics such as the surface maximum and variance. The mean is always comparable without correction because individual cell estimates take into account the total number of grid cells (the details are found in Gove, 2014).

```
R> mtr
```

```
------------------------------------------------------------
object of class Tract
------------------------------------------------------------
Measurement units = metric
Area in square meters = 400 (0.04 hectares)

class       : mirageTract
dimensions  : 20, 20, 400  (nrow, ncol, ncell)
resolution  : 1, 1  (x, y)
extent      : 0, 20, 0, 20  (xmin, xmax, ymin, ymax)
coord. ref. : NA
```

```
data source : in memory
names       : surf
values      : 0, 0  (min, max)
```

```
R> ltr
```

```
------------------------------------------------------------
object of class Tract
------------------------------------------------------------
Measurement units = metric
Area in square meters = 552 (0.0552 hectares)

class       : Tract
dimensions  : 24, 23, 552  (nrow, ncol, ncell)
resolution  : 1, 1  (x, y)
extent      : 0, 23, -1, 23  (xmin, xmax, ymin, ymax)
coord. ref. : NA
data source : in memory
names       : surf
values      : 0, 0  (min, max)
```

The tract area returned via the `area` function is in units of square meters, corresponding to the tract resolution: there are 400 and 552 grid cells in `mtr` and `ltr`, respectively, and each is 1m^2. So we need to divide each by the number of hectares in the tract to get things on a comparable footing (i.e., put everything on a per-hectare or per-acre basis[7]).

Here is the adjustment for the surface maximum volume estimate, note how in this case, increasing the tract size from smaller than a hectare to a hectare (i.e., effectively allowing for all the extra zero-valued background cells) increases the maximum volume estimate for the tract (this is because we need to "integrate" over a larger number of background cells, so the estimates within inclusion zones must inflate accordingly such that taking the mean works out correctly)...

```
R> c(area(mtr), area(ltr))            #in m^2
```

```
[1] 400 552
```

```
R> (smph = .StemEnv$smpHectare)       #built-in package constant
```

[7] Or we could just bring the smaller tract to the resolution of the larger for comparison as in Gove (2014).

```
[1] 10000
```

```
R> ss.m@surfStats$max * smph/area(mtr)
```

```
[1] 49.908832
```

```
R> ss.l@surfStats$max * smph/area(ltr)
```

```
[1] 24.954416
```

```
R> ss.l@surfStats$max * smph/area(ltr) * 2
```

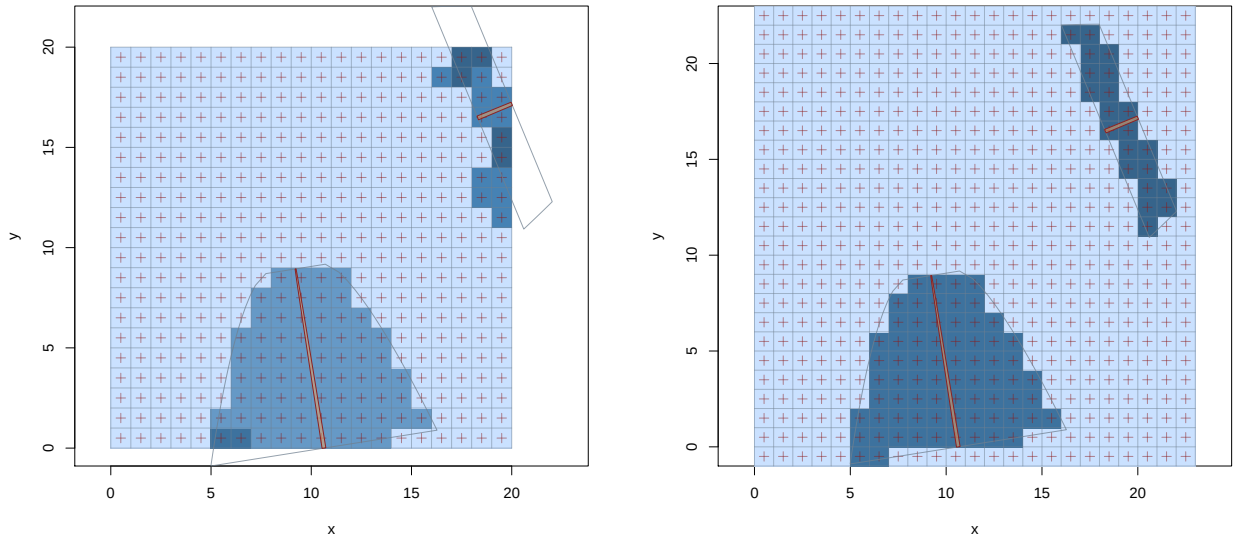
```
[1] 49.908832
```

In the final line of code, we check that the maximum for the mirage is twice that for the non-miraged tract—this, of course conforms to the fact that those cells in the miraged estimate are doubled in the reflection zone. So the scaling correction has indeed allowed us to make comparisons between the tracts of different sizes. Similar adjustments will be necessary when comparing to buffered tracts, that is, Masuyama's method; this is illustrated in Gove (2014).

The two sampling surfaces can be compared visually in Figure 3. Notice the difference in the tract extents, and how the volume attribute density has been heaped back into the tract under mirage. In addition, the shading will be different between the two figures because of the different surface maxima, this can be demonstrated by plotting them with the argument `useImage=FALSE`. The following code demonstrates how we can enlarge the plot area to show the portions of the inclusion zones that extend beyond the tract in order to produce Figure 3a. It is done by finding the overall `bbox` for the tract plus the inclusion zone collection.

```
R> bbox.iz = bbox(ss.m@izContainer)
R> bbox.tr = bbox(ss.m@tract)
R> bbs = array(dim=c(2,2,2))      #array of bboxes for both IZ and tract
R> bbs[,,1] = bbox.iz
R> bbs[,,2] = bbox.tr
R> dimnames(bbs) = dimnames(bbox.tr)
R> ext.bb = bboxSum(bbs)          #overall extent
R> xlim = c(ext.bb['x','min'], ext.bb['x','max'])
R> ylim = c(ext.bb['y','min'], ext.bb['y','max'])
R> plot(ss.m, axes=TRUE, gridLines=TRUE, gridCenters=TRUE,
+      xlim=xlim, ylim=ylim)
```

```
R> plot(ss.l, axes=TRUE, gridLines=TRUE, gridCenters=TRUE)
```



(a) Mirage

(b) Internal

Figure 3: Sampling surfaces with PDS (with probability proportional to coverage area) showing both (a) the mirage method surface, and (b) the surface with inclusion zones completely contained internally to the tract.

5 Gory Details

These are notes for programmers, if you want to read on, feel free to, but the information is not necessary for using the mirage method in `sampSurf` and may only serve to confuse the casual user.

5.1 "mirageInclusionZoneGrid"

There are several points to be made with respect to the "mirageInclusionZoneGrid" objects.

5.1.1 Small slivers

It can happen that the inclusion zones for a stem will overlap the boundary in such a way that no grid cell centers outside the tract are included within it.[8] This, of course, depends on the size of the sliver polygon and the grid cell resolution. If it happens a slight bias can be recorded for an individual stem because no sample points will be reflected back into the tract, so that area will be unaccounted for in the estimate. The bias is usually on the order of a couple percent. For a full surface with many stems, it will be insignificant.

The "mirageInclusionZoneGrid" object will reflect the outcome of the mirage by the information that is stored in its slots, depending on the cardinal direction. The following three possibilities are allowed (we will use east and an example below)...

1. No slopover in direction `east`: `slopOver['east']` will be `FALSE`. The `east.polygon` and `east.grid` slots will be `NULL`.
2. Slopover is present, but in a small sliver that does not intersect any grid cells: `slopOver['east']` will be `TRUE`. The `east.polygon` will have a valid sliver polygon object. The `east.grid` slot will be `NULL` because there is no grid that the polygon sliver covers.
3. Slopover is present and at least one grid cell is covered: `slopOver['east']` will be `TRUE`. The `east.polygon` will have a valid sliver polygon object. The `east.grid` will have a valid "RasterLayer" class object.

It follows that we should be able to distinguish any situation as needed from tests on these three slots, again, for any given cardinal direction.

5.1.2 Sample depth

The sample `depth` column in the `izObject@data` slot will have `depth=1` for all cells that are covered by estimates. This can mean one of two things. First, if mirage reflects all cells outside the tract back to within the inclusion zone that is internal to the tract, then it is business as usual. However, a second possibility is that a portion of the external inclusion zone is reflected back into a "phantom" zone that falls outside the original inclusion zone internal to the tract. These cells that are part of the phantom reflection also receive depth of one, even though they are outside the inclusion zone. We need this to happen if we are to correctly calculate covariances via "lapSurf"; i.e., so we can identify estimates for a stem regardless of whether they fall within or external to the inclusion zone.

It is important to note that `depth` does *not* record the number of reflections that occur under mirage for any given grid cell.

[8] Recall that an extended grid is created that adds these external cells to the tract for the purpose of mirage.

5.1.3 Background cell values

The above phantom areas raise another point that one needs to be aware of. In the "InclusionZoneGrid" class and its mirage subclass, the grid values stored in the `izObject@grid@data@values` slot of the "RasterLayer" object[9] will have zeros for cells covered by the internal inclusion zone, and `NA` for background cells. However, now we have the situation where phantom cells have values in some of what still are background cells (i.e., external to the inclusion zone). Therefore the values in the `izObject@data` slot will not exactly align with those in `izObject@grid@data@values`. Here is an illustration from our previous PDS example...

```
R> gvals = mpds.cl1@grid@data@values
R> dvals = mpds.cl1@data[c('volume','depth')]
R> head(cbind(gvals, dvals), 8)
```

```
  gvals    volume depth
1    NA  0.000000     0
2    NA  0.000000     0
3     0 49.908832     1
4     0 49.908832     1
5    NA  0.000000     0
6    NA  0.000000     0
7    NA 24.954416     1
8     0 49.908832     1
```

```
R> table(ifelse(is.na(gvals), 'bg','iz'))
```

```
bg iz
40 10
```

```
R> table(ifelse(dvals$volume>0, 'iz','bg'))
```

```
bg iz
34 16
```

In the results from the first line of code we see that one of the background cells (`gvals==NA`) has a volume due to the phantom reflection. The tabulation of these two slots in the following lines of

[9] This also all applies to the `izGrid.extended` slot grid.

code illustrates that there are several more (phantom) background cells with values as well (`bg` is background, `iz` is inclusion zone).[10]

This should not present a problem unless one is using the result of `getValues` to mask cells. Please see the code in `heapIZ`, which does not do this, but rather uses `setValues` to exchange values directly for establishing the sampling surface, thereby circumventing this potential problem.

References

J. H. Gove. *The "Tract" Class*, 2013a. URL `http://CRAN.R-project.org/package=sampSurf`. `sampSurf` package vignette. 6

J. H. Gove. *The "InclusionZoneGrid" Class*, 2013b. URL `http://CRAN.R-project.org/package=sampSurf`. `sampSurf` package vignette. 6, 7

J. H. Gove. *Comparing Surface Statistics in* ***sampSurf***, 2014. URL `http://sampsurf.r-forge.r-project.org/`. `sampSurf` vignette, only available on R-Forge. 2, 10, 11, 12

J. H. Gove and P. C. Van Deusen. On fixed-area plot sampling for downed coarse woody debris. *Forestry*, 84(2):109–117, 2011. 3

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. 7

T. G. Gregoire and H. T. Valentine. *Sampling strategies for natural resources and the environment*. Applied environmental statistics. Chapman & Hall/CRC, N.Y., 2008. 1, 5

T. B. Lynch and J. H. Gove. The unbiasedness of a generalized mirage boundary correction method for Monte Carlo integration estimators of volume. *Canadian Journal of Forest Research*, 44(7): 810–819, 2014. 1

M. Masuyama. A rapid method for estimating basal area in a timber survey—an applicaion of integral geometry to areal sampling problems. *Sankhya*, 12, 1953. 1, 2, 3, 12

P. Schmid-Haas. Stichprobenam waldrand (sampling at the edge of the forest). *Mitt Schweiz Anst Forstl Versuchswes*, 45(3):234–303, 1969. 1

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. 3

[10] Note, in general, that the values in the `izObject@grid@data@values` slot are normally retrieved using the `raster getValues` function on the object; i.e., `getValues(izObject@grid)`.