Aggregating CLUMondo grids

René Sachse rene.sachse@uni-potsdam.de

October 22, 2013

Contents

1 Read and transform CLUMondo grids

First the package `luess` needs to be loaded. It relies a lot on the functionality of the sp package. The luess package contains lots of sample data. To save time we use this packaged data instead of generating it from scratch. Therefore, some commands are out-commented.

```
> library(luess)
> #CLUlonglat <- transform_asciigrid("land_systems.asc")
```

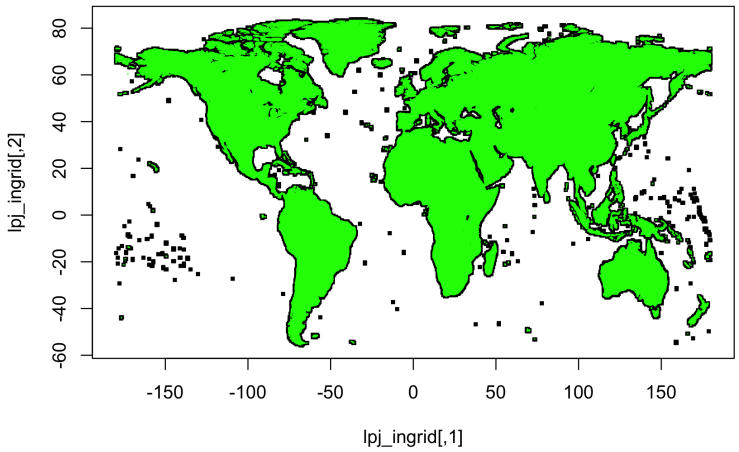


Figure 1: Overlay of LPJmL grid cells (black) and re-projected CLUMondo grid cell centers (green). There is no systematic offset indicating valid re-projection. LPJmL grid cells were plotted with bigger points, so that they can be seen under the CLUMondo points. LPJmL grid cells "pop out" equally in every direction.

`transform_asciigrid()` reads CLUMondo output grids (Asselen and Verburg, 2012; Verburg, 2013) and transforms them from Eckert IV projection (areas equivalent) to longlat format (geographic coordinates, equirectangular projection, plate carrée). The function also could deal with any other projections when specified as argument (for using function arguments, see the function documentation by typing `?transform_asciigrid`).

You can globally inspect the reprojected longlat CLUMondo grid by:

```
> spplot(CLUlonglat, pch=".")
```

or zoom into central Africa:

```
> spplot(CLUlonglat, pch=".",xlim=c(-10,30), ylim=c(-20,20))
```

Because of the transformation, the grid is not regular anymore. The function therefore, returns a data.frame with spatial points (SpatialPointsDataFrame).

The reprojected CLUMondo grid cells perfectly overlap LPJmL grid cells without any systematic offset (Fig. 1).

2 Generate new model grids

LPJmL works with a much lower resolution than CLUMondo. We can generate such an 0.5° grid by using:

```
> cluagg_grid <- generate_grid()
```

3 Match LPJmL grid cells with new grid cells

The data set `lpj_ingrid` contains the coordinates of the cell centers of the 67420 original grid cells used within LPJmL. To figure out, which of the cells of our generated regular grid represent these original cells we can use the function `matchInputGrid`:

```
> lpj_long_clupos <- matchInputGrid(
+   coordinates(cluagg_grid),
+   lpj_ingrid
+ )
```

`lpj_long_clupos` than contains the integer numbers (positions) of the original LPJmL cells within our set up low resolution grid.

4 Resample transformed CLUMondo data for LPJ cells

Resampling is performed using function `over()` internally from package sp. It is determined in which grid cells the cell centers of the CLUMondo cells are located. In case a CLUMondo cell center is located within a grid cell, than that complete CLUMondo pixel is assigned to that cell assuming that the majority of the CLUMondo pixel is belonging to that cell anyway. Furthermore, it is counted how many CLUMondo pixels of each land use system are located within a grid cell. This number divided by the absolute number of CLUMondo pixels within a grid cell provides the fraction of grid cell area this land use system is covering.

```
> CLUtoLPJ2040long <- resample_grid(CLUlonglat, lpj_ingrid,
+                     cells=lpj_long_clupos)
```

Because Eckert IV projection has equivalent areas, but not longlat projection, there are up to 48 CLUMondo pixels within one LPJ grid cell near the equator, while the number of CLUMondo pixels per grid cell is reducing towards the poles or on coasts (Fig. 2 and 3). On coasts LPJmL pixels also cover water surface due to their coarser resolution and therefore contain less CLUMondo pixels in these areas. Stripe patterns evolve when pixels are systematically assigned to one side, leaving the other side with one pixel less. This might look of importance when looking at covered pixels per cell, but later these numbers are used to calculate covered area fractions. After that calculation, errors due to the assignment of pixels to only one grid cell at grid cell boarders should be reduced strongly. Size of this error compared to different resampling methods has not been investigated any further.

```
> nrc  <- sapply(CLUtoLPJ2040long$hrcells, length)
> coor <- cbind(CLUtoLPJ2040long$xcoord, CLUtoLPJ2040long$ycoord)
> mySpectral <- colorRampPalette(c(brewer.pal(11, "Spectral"), "navy"))
> img <- gridPlot(
+   values=nrc,
+   coordinates=coor,
+   main="CLUMondo Grid Cells per LPJ cell",
+   clab="",
+   res=0.5,
+   plot=TRUE,
+   axes=TRUE,
+   col=rev(mySpectral(50)),
+   mar=c(5,4,4,4), cex=1.5
+ )
```

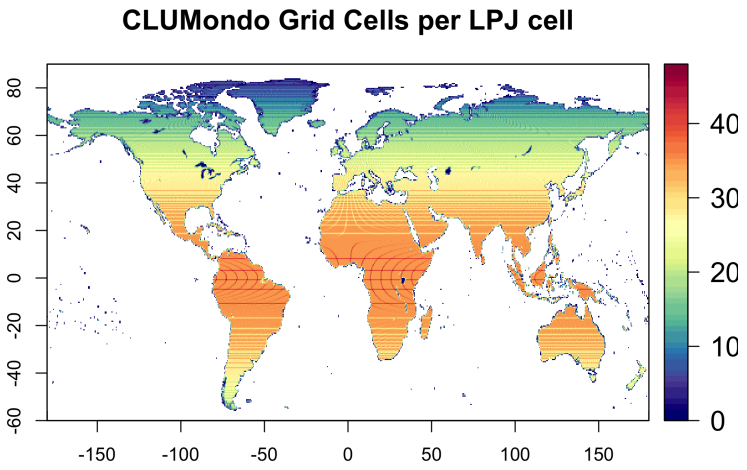


Figure 2: Number of CLUMondo pixels within a LPJmL grid cell.

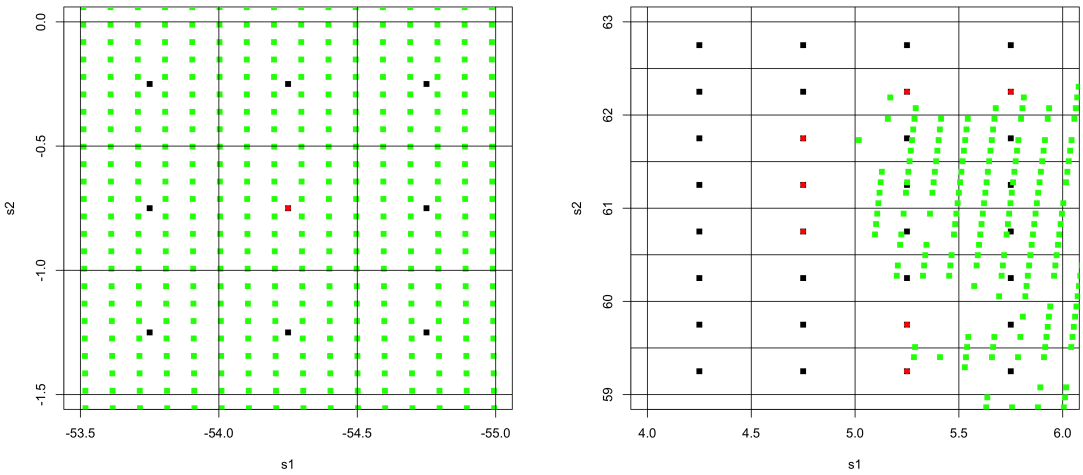
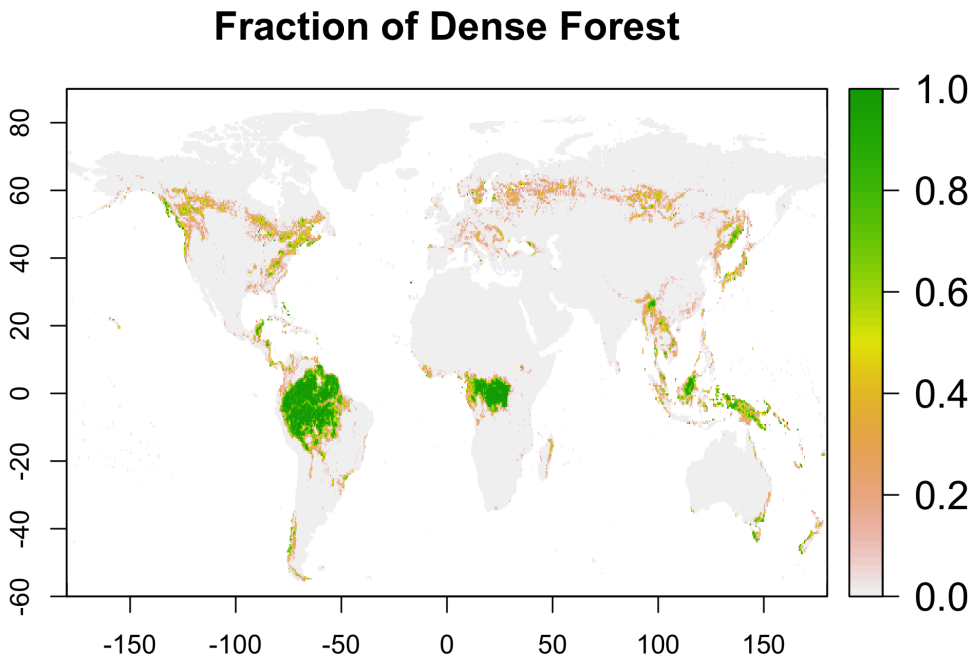


Figure 3: Zoom to the LPJ grid with more than 47 CLUMondo pixel density (left) and less than 5 pixels density (right). Green points: CLUMondo cell centers, Black lines: LPJ grid cells, Red points: LPJ cells with extreme low or high number of CLUMondo pixels.

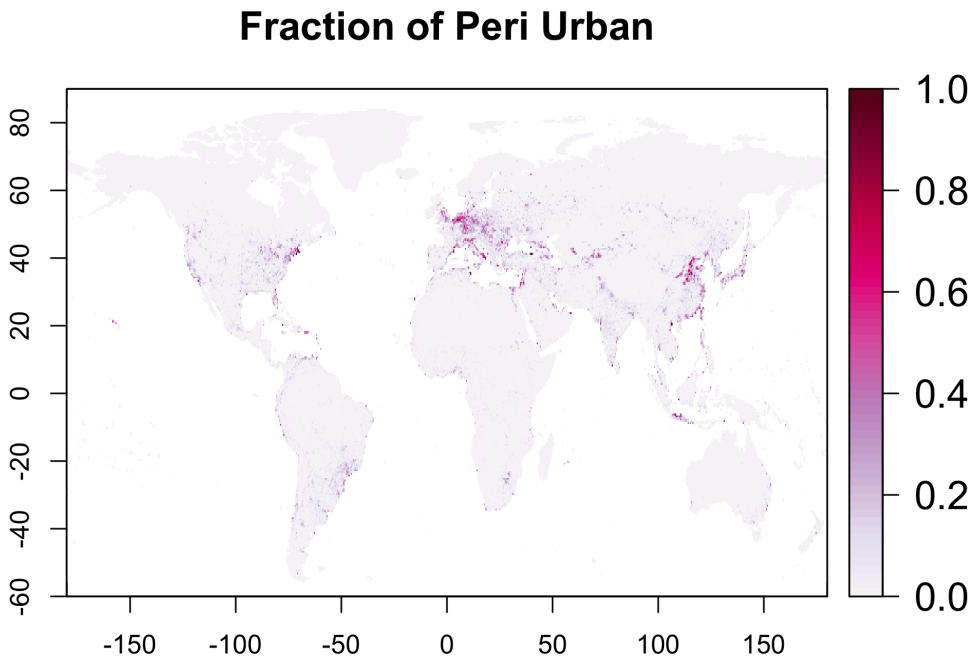
5 Validation of resampling algorithm

Validation can be performed by visualization of the resulting aggregated maps and comparing to the original maps from Asselen and Verburg (2012) and Verburg (2013).

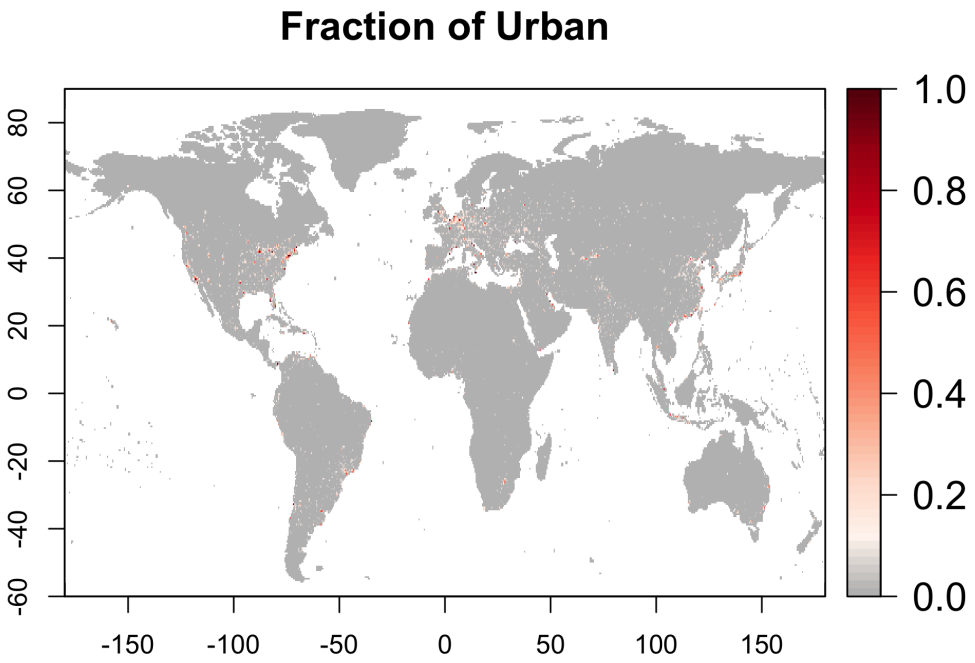
```
> forest  <- CLUtoLPJ2040long$lufrac[19,]
> periurb <- CLUtoLPJ2040long$lufrac[29,]
> urban   <- CLUtoLPJ2040long$lufrac[30,]
> natgrass<- CLUtoLPJ2040long$lufrac[24,]
> cropint <- CLUtoLPJ2040long$lufrac[9,]
> purb     <- colorRampPalette(brewer.pal(9, "PuRd"))
> reds     <- colorRampPalette(c("gray", brewer.pal(9, "Reds")))
> oranges  <- colorRampPalette(c("gray", brewer.pal(9, "Oranges")))

> img_forest  <- gridPlot(values=forest,cex=1.5,
+                         main="Fraction of Dense Forest",
+                         mar=c(5,4,4,7))
```



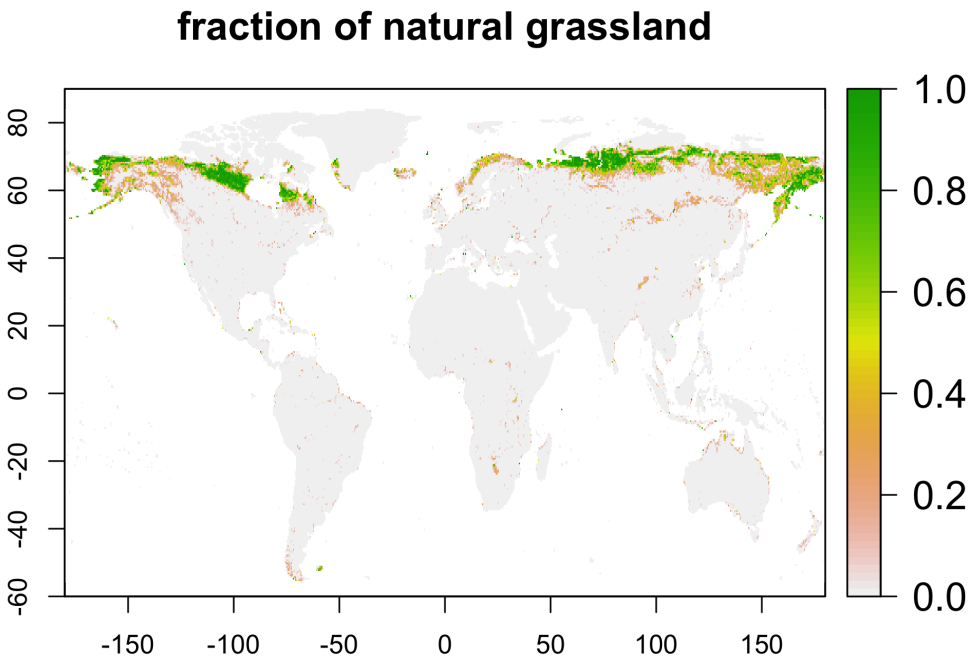

```
> img_periurb <- gridPlot(values=periurb,cex=1.5,
+                         main="Fraction of Peri Urban",
+                         col=purb(1000), mar=c(5,4,4,7))
```



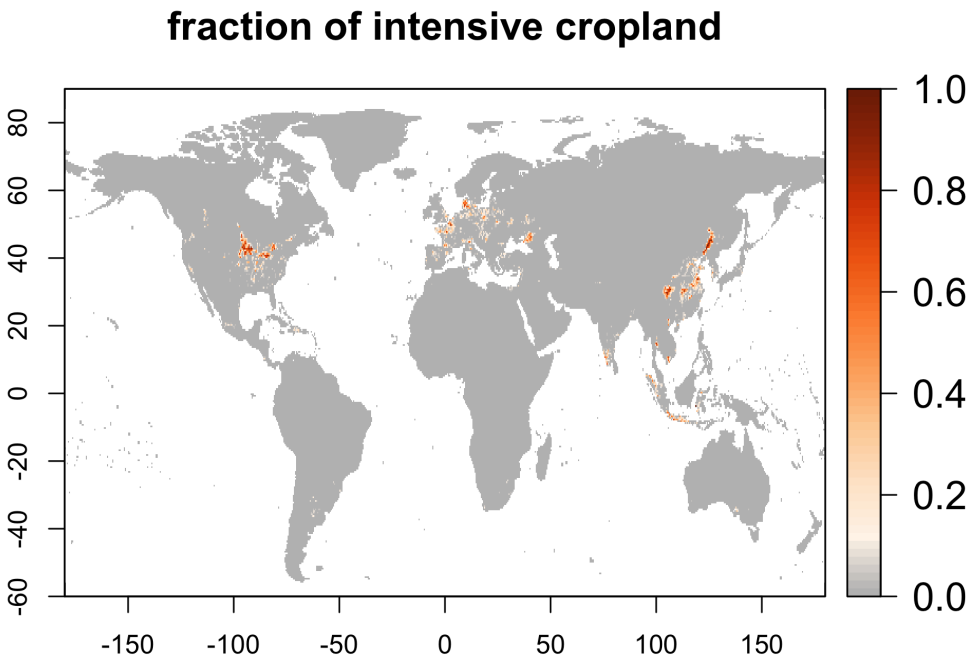

```
> img_urban   <- gridPlot(values=urban, cex=1.5,
+                         main="Fraction of Urban",
+                         col=reds(1000), mar=c(5,4,4,7))
```




```
> img_natgrass<- gridPlot(values=natgrass, cex=1.5,
+                         main="fraction of natural grassland",
+                         mar=c(5,4,4,7))
```




```
> img_cropint <- gridPlot(values=cropint, cex=1.5,
+                         main="fraction of intensive cropland",
+                         col=oranges(1000), mar=c(5,4,4,7))
```

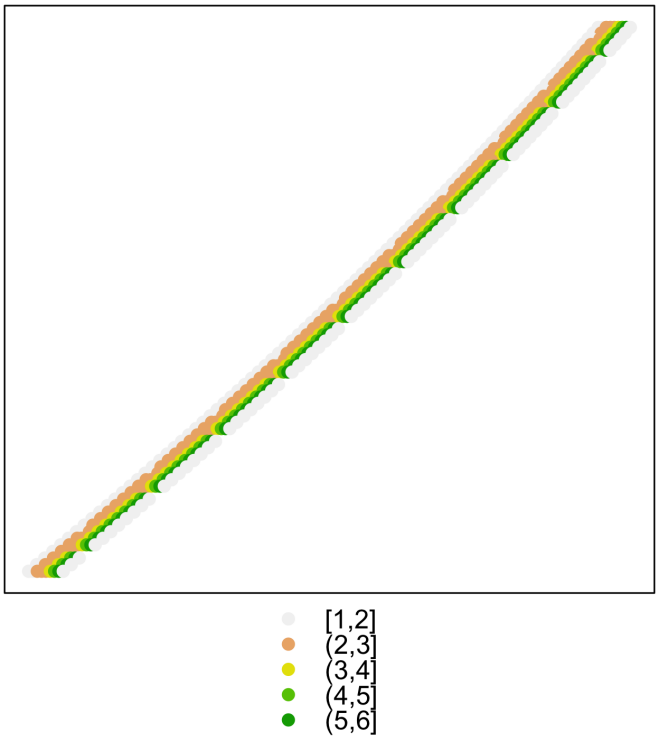


6 Disentangling CLUMondo Mosaics

CLUMondo provides different classes of landscape mosaics. To disentangle the underlying real land use classes per grid cell, the function `aggregateMosaics()` can be used. In the current version no detailed mosaic data for CLUMondo is available (e.g. how much percent of the mosaic types are natural vegetation, agricultural land, etc...). Therefore, the function only is used with arbitrary examples for demonstration purposes at the moment.

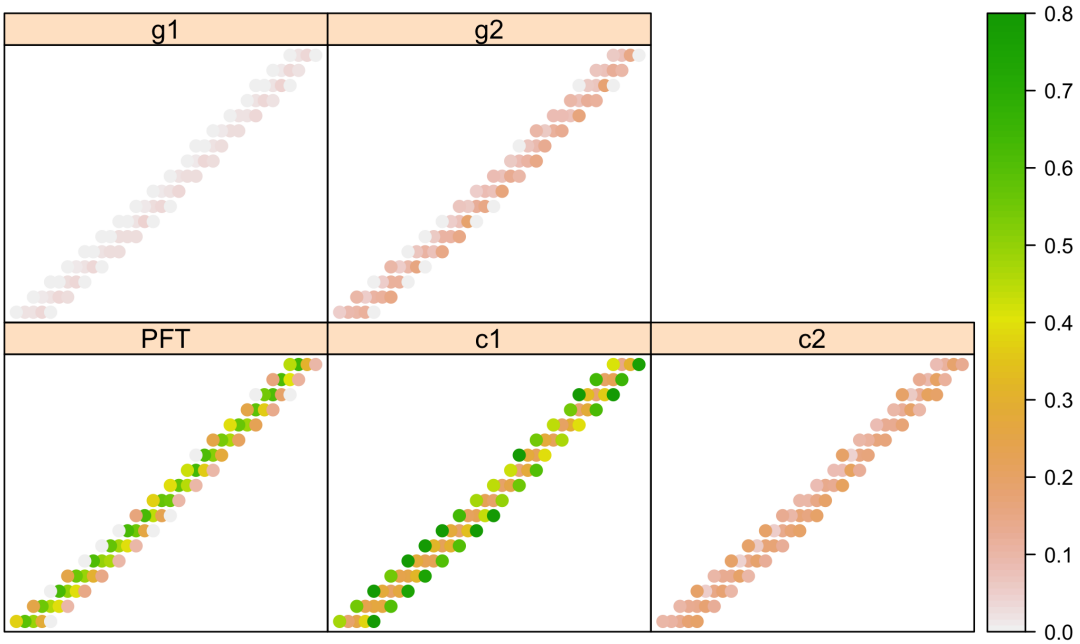
We assume a small artifical map of 6 arbitrary mosaic types:

```
> spplot(smallarea, col.regions=rev(terrain.colors(100)))
```



Now we can aggregate these mosaics to some (arbitrary) underlying land use classes:

```
> mosaicFractions <- data.frame(
+   PFT=c(0,.50,1,0,.10,.30),
+   c1=c(.80,.30,0,0,.10,.70),
+   c2=c(.20,0,0,.80,.10,0),
+   g1=c(0,0,0,.10,.10,0),
+   g2=c(0,.20,0,.10,.60,0)
+ )
> grid_lr <- generate_grid(cellcentre.offset=c(-179.75, -59.75))
> outMosaic <- resample_grid(smallarea, grid_lr)

[1] "Resampling grid."
[1] "Resampling finished."

> outDisentangle <- aggregateMosaics(outMosaic, mosaicFractions)
> spplot(outDisentangle, col.regions=rev(terrain.colors(100)), colorkey=TRUE)
```



7 Changes and new functionality in latest package versions

7.1 v0.17.1: resample_grid() faster without parallelization

Since version v0.17.1 the function `resample_grid()` internally uses `aggregate()` to count original pixel types. This is one magnitude faster than the parallelized loop-variant of the previous versions. Therefore the option `parallel=TRUE` is deprecated as long as only a low number of cores (less than approx. 20 cores) are available.

7.2 v0.17.0: Parallelization for faster grid resampling

Since version v0.17.0 the function `resample_grid()` is capable of using multiple cores for calculation. This speeds up the function significantly. The packages `foreach` and `doParallel` are needed for this. Multiple core usage is invoked by providing the arguments `parallel=TRUE` and a number of `cores` to use:

```
> system.time(
+   out1 <- resample_grid(CLUlonglat, lpjgrid, cells=40000:42000)
+ )
```

```
> system.time(
+   out2 <- resample_grid(CLUlonglat, lpjgrid, cells=40000:42000,
+                         parallel=TRUE, cores=2)
+ )
> identical(out1, out2)
```

7.3 v0.16.0: resample_grid() output format

The function `resample_grid()` returns objects of SpatialPointsDataFrame since version v0.16.0. Pixel type fractions data for this output can be accessed and plotted as follows:

```
> grid_lr <- generate_grid(cellcentre.offset=c(-179.75, -59.75))
> out     <- resample_grid(smallarea, grid_lr)
```

```
[1] "Resampling grid."
[1] "Resampling finished."
```

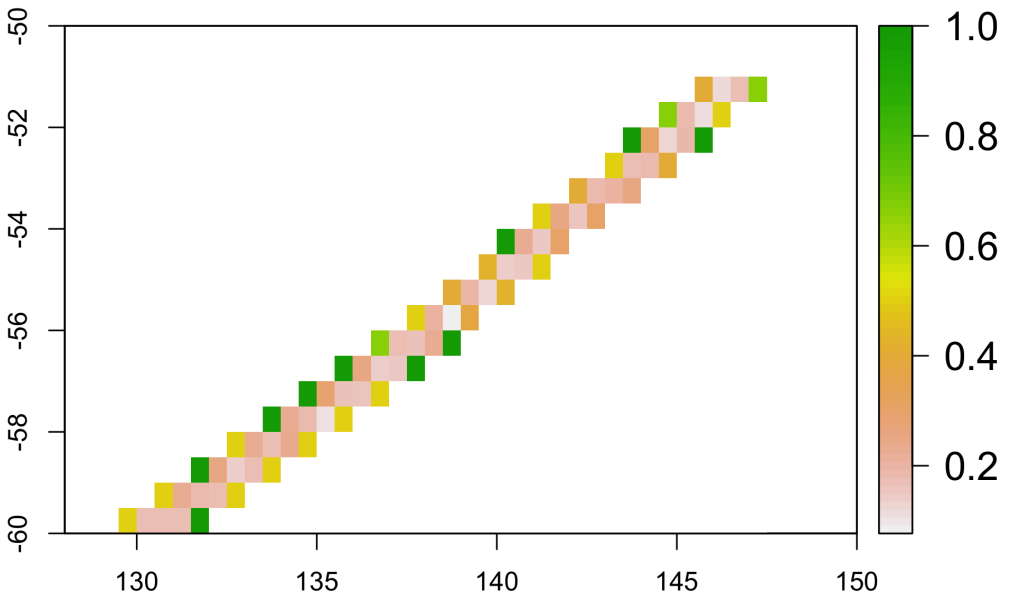
```
> head(attr(out,"data"))
```

```
         V1         V2        V3         V4         V5        V6
1 0.1111111 0.16666667 0.5555556 0.11111111 0.05555556 0.0000000
2 0.1666667 0.00000000 0.2222222 0.16666667 0.22222222 0.2222222
3 0.6666667 0.00000000 0.0000000 0.00000000 0.00000000 0.3333333
4 0.4000000 0.30000000 0.3000000 0.00000000 0.00000000 0.0000000
5 0.1764706 0.17647059 0.5294118 0.05882353 0.05882353 0.0000000
6 0.1052632 0.05263158 0.3157895 0.21052632 0.15789474 0.1578947
```

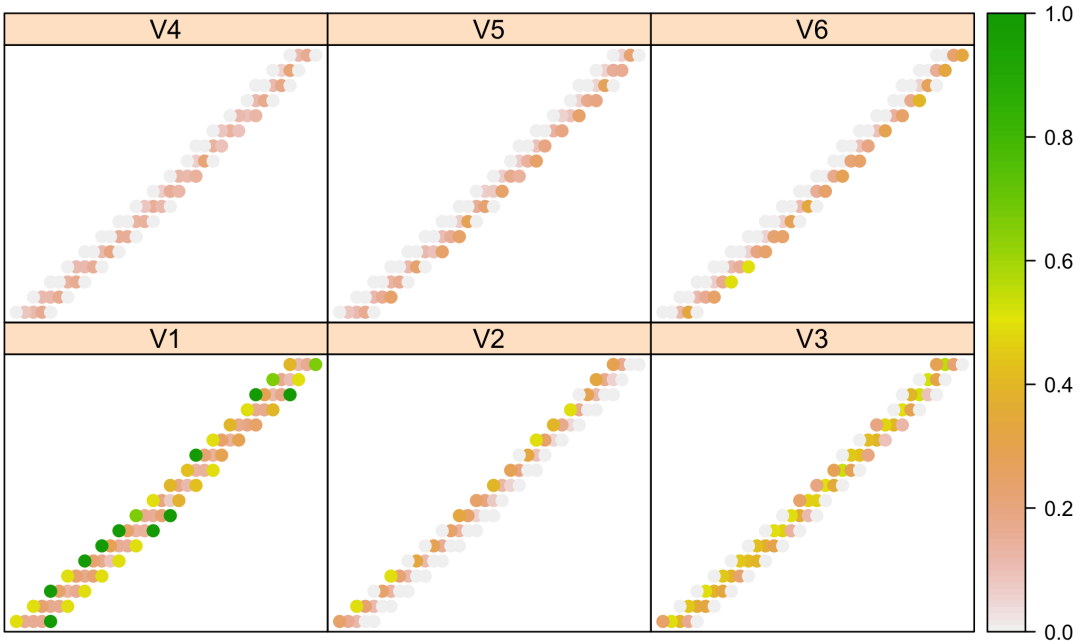
```
> head(out@data)
```

```
         V1         V2        V3         V4         V5        V6
1 0.1111111 0.16666667 0.5555556 0.11111111 0.05555556 0.0000000
2 0.1666667 0.00000000 0.2222222 0.16666667 0.22222222 0.2222222
3 0.6666667 0.00000000 0.0000000 0.00000000 0.00000000 0.3333333
4 0.4000000 0.30000000 0.3000000 0.00000000 0.00000000 0.0000000
5 0.1764706 0.17647059 0.5294118 0.05882353 0.05882353 0.0000000
6 0.1052632 0.05263158 0.3157895 0.21052632 0.15789474 0.1578947
```

```
> img     <- gridPlot(
+              attr(out, "data")[,1],
+              coordinates(out),
+              xlim=c(128,150),
+              ylim=c(-60,-50), cex=1.5
+            )
```




```
> spplot(out, col.regions=rev(terrain.colors(100)), colorkey=TRUE)
```



The old output format of the versions before v0.16.0 can be obtained by specifying `verbose=TRUE`:

```
> out <- resample_grid(smallarea, grid_lr, verbose=TRUE)

[1] "Resampling grid."
[1] "Resampling finished."

> names(out)

[1] "cells"    "xcoord"   "ycoord"   "hrcells"  "hrvalues" "luid"     "lufrac"
```

References

Asselen, S. and Verburg, P. H. (2012). A land system representation for global assessments and land-use modeling. *Global Change Biology*, 18(10):3125–3148.

Verburg, P. H. (2013). Land cover change or land use intensification: simulting land system change with a global-scale land change model. *Global Change Biology*.