Aggregating and Translating CLUMondo Grids to LPJmL

René Sachse `rene.sachse@uni-potsdam.de`

April 2, 2014

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.

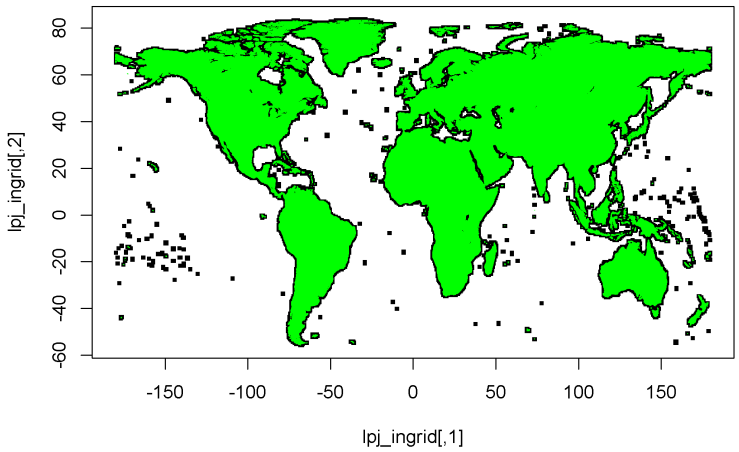


Figure 1: Overlay of LPJmL grid cells (black) and re-projected CLUMondo grid cell centers (green). There is no systematic offset indicating valid re-projection. LPJmL grid cells were plotted with bigger points, so that they can be seen under the CLUMondo points. LPJmL grid cells "pop out" equally in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

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

`transform_asciigrid()` reads CLUMondo output grids (Asselen and Verburg, 2012, 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, verbose=TRUE)
```

The option `verbose=TRUE` returns a list with output. The recent versions of the package return an object of class SpatialPointsDataFrame as output. See section 8.1 for details how to handle data of that format.

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
+ )
```

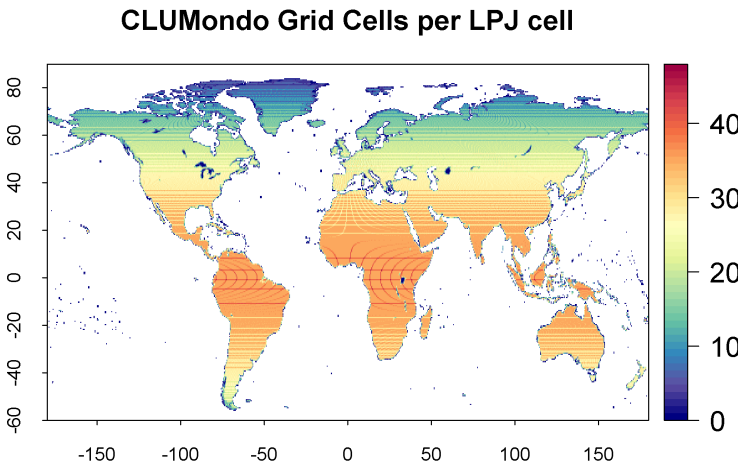


Figure 2: Number of CLUMondo pixels within a LPJmL grid cell.

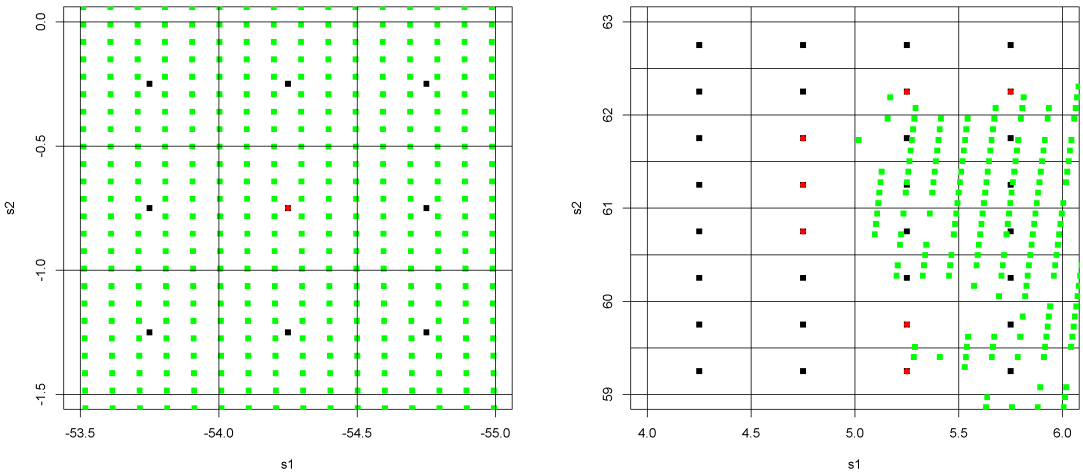
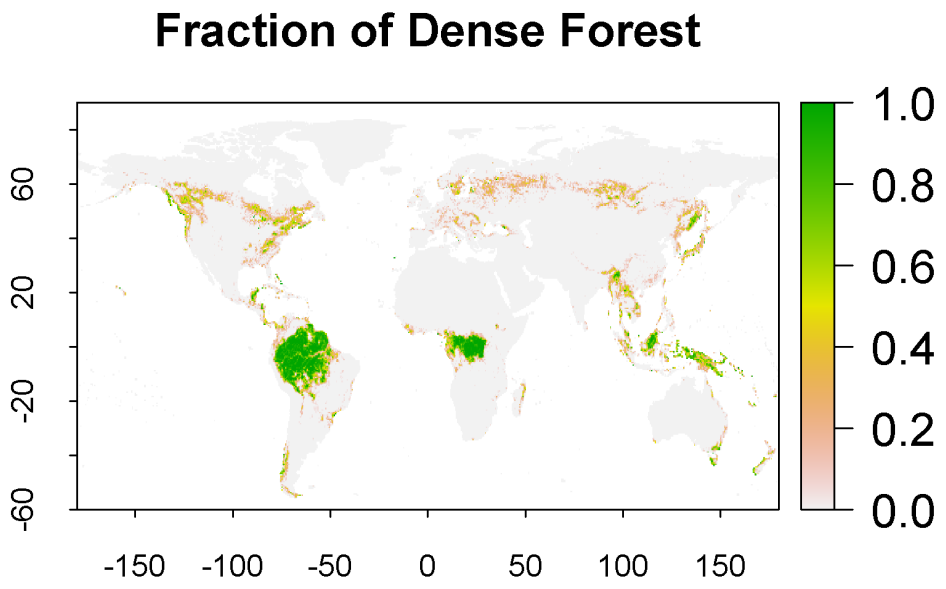


Figure 3: Zoom to the LPJ grid with more than 47 CLUMondo pixel density (left) and less than 5 pixels density (right). Green points: CLUMondo cell centers, Black lines: LPJ grid cells, Red points: LPJ cells with extreme low or high number of CLUMondo pixels.

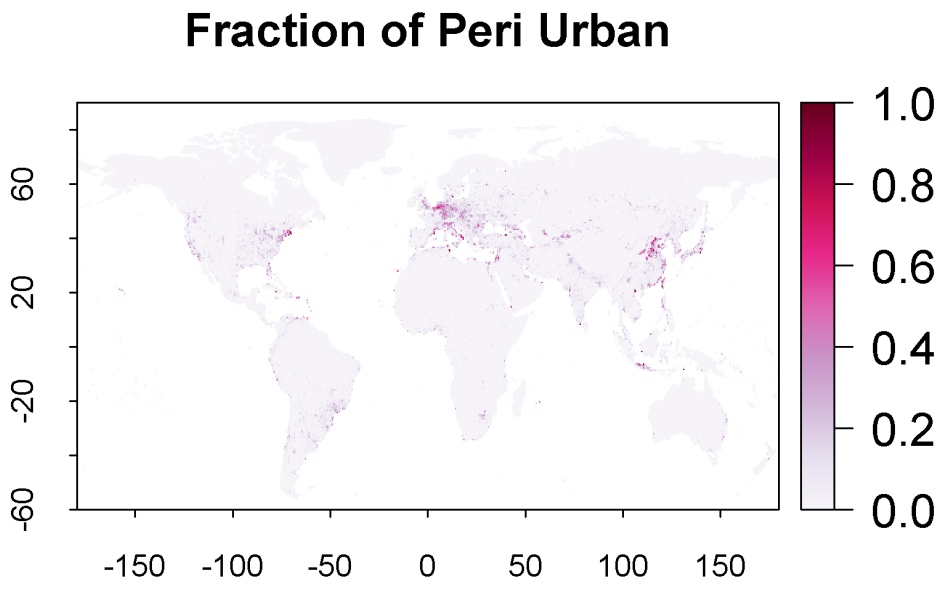
5 Validation of resampling algorithm

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

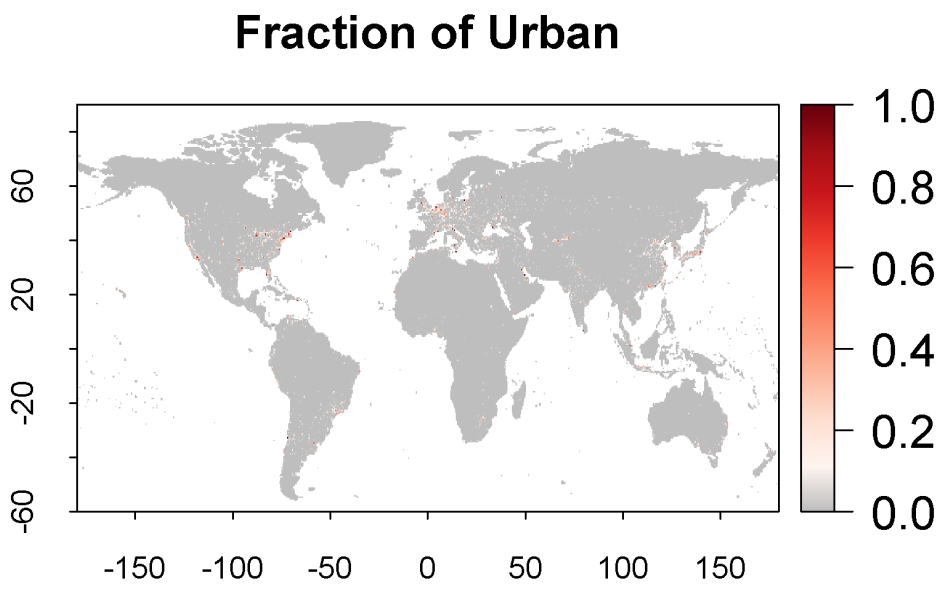
```
> forest  <- CLUtoLPJ2040long$lufrac[19,]
> periurb <- CLUtoLPJ2040long$lufrac[29,]
> urban   <- CLUtoLPJ2040long$lufrac[30,]
> natgrass<- CLUtoLPJ2040long$lufrac[24,]
> cropint <- CLUtoLPJ2040long$lufrac[9,]
> purb     <- colorRampPalette(brewer.pal(9, "PuRd"))
> reds     <- colorRampPalette(c("gray",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



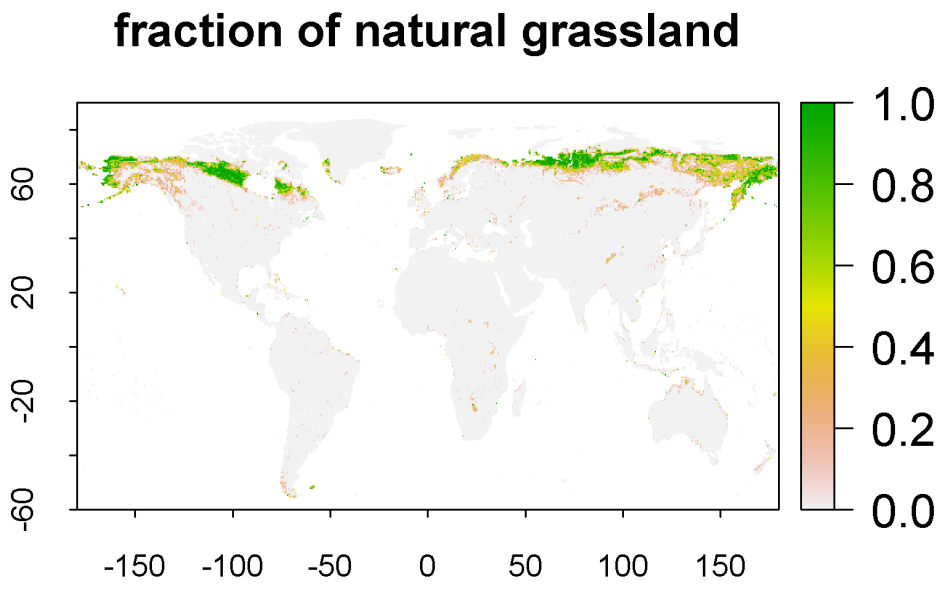

```
> img_periurb <- gridPlot(values=periurb,cex=1.5,
+                         main="Fraction of Peri Urban",
+                         col=purb(1000), mar=c(5,4,4,7))
```



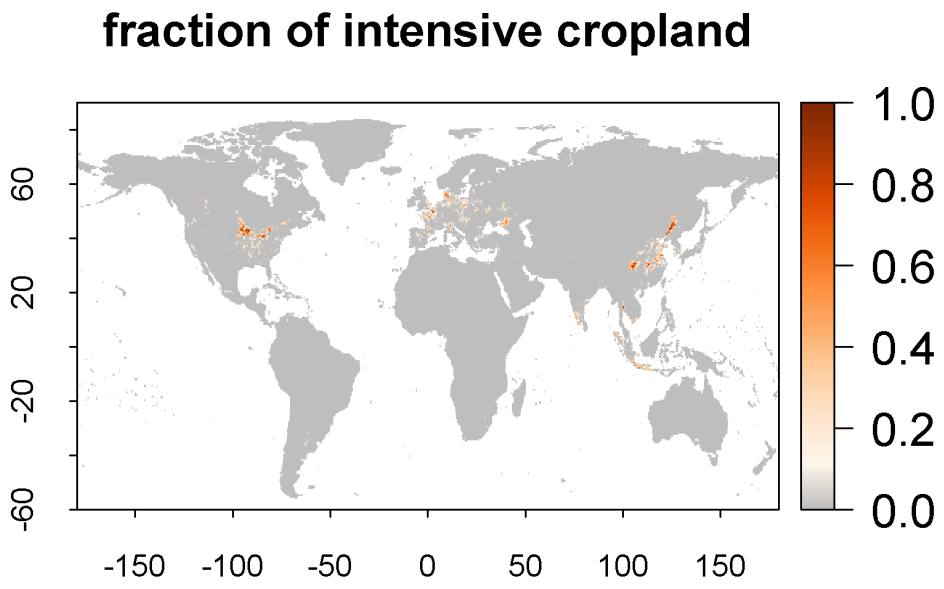

```
> img_urban   <- gridPlot(values=urban, cex=1.5,
+                         main="Fraction of Urban",
+                         col=reds(1000), mar=c(5,4,4,7))
```




```
> 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

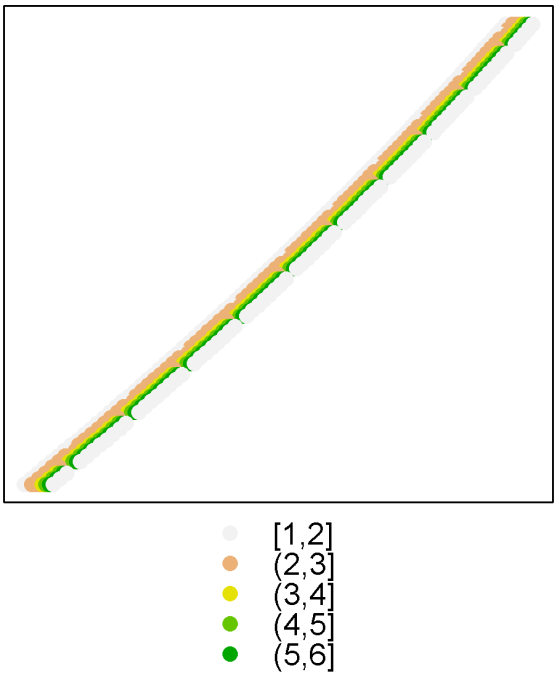
CLUMondo provides 30 different classes of landscape mosaics (Asselen and Verburg, 2013). Each mosaic consists of one of the five land covers: trees, buildt up area, bare area, cropland or pasture. Furthermore, the ClUMondo land use systems contain information about the intensity of the management of the cropland and about the number of livestocks. At the moment intensity and livestock information is not used for the translation.

For LPJmL we sum up trees, built up area and bare area to an area of natural vegetation (LPJmL does not consider urban areas at the moment; however since this area is increasing in the future, this might be an important enhancement). The decision what kind of plant functional types (PFTs) are growing on land, whether it is trees, grass or nothing (bare) is decided by LPJmL on its own on areas reserved for potential natural vegetation. After aggregation for LPJmL we have the three land covers: natveg, cropland and pasture remaining. In principle, natveg and pasture translates directly to LPJmL, while cropland needs to be translated to 13 different crop functional types (CFTs) later. However, CLUMondo pasture was defined differently from the one used in LPJmL standard input data (see extra vignette about translation problems). Maybe we need to add pasture areas from CLUMondo's natural grassland systems without livestock or with only very few livestock to natural vegetation.

To disentangle the underlying land covers per grid cell, the generic function `aggregateMosaics()` or a LPJmL specific function for CLUMondo aggregation `aggregateMosaicsClumondo()` can be used. Since the mosaic composition in CLUMondo is different in each of the 24 world regions, the function `aggregateMosaicsClumondo()` already applies different tables during the aggregation to all world regions.

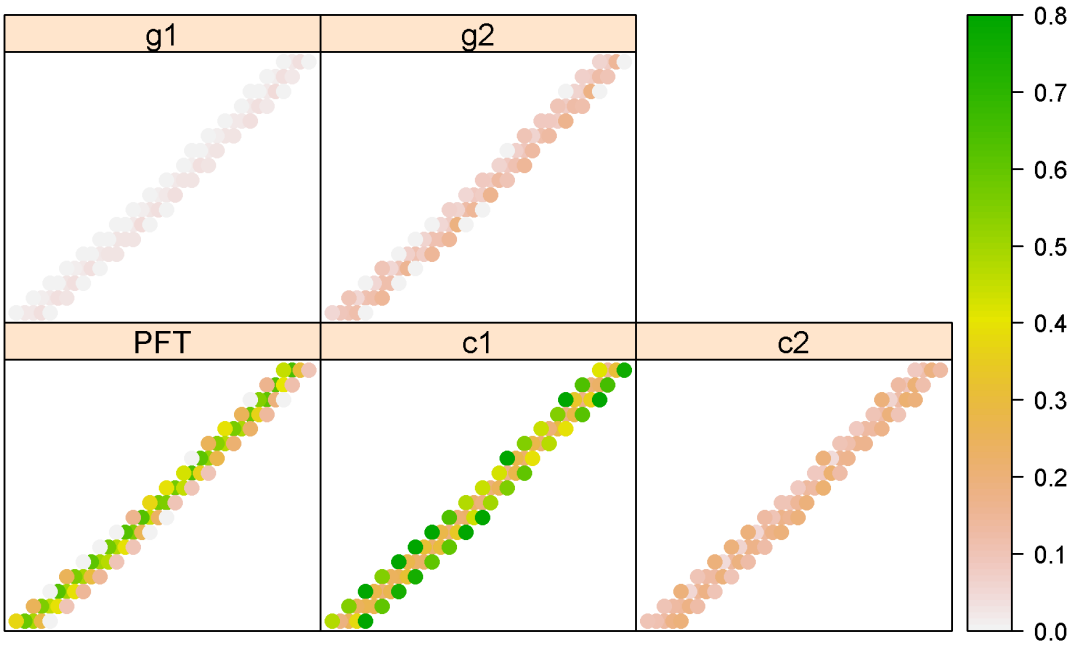
We assume a small artifical map of 6 arbitrary mosaic types:

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



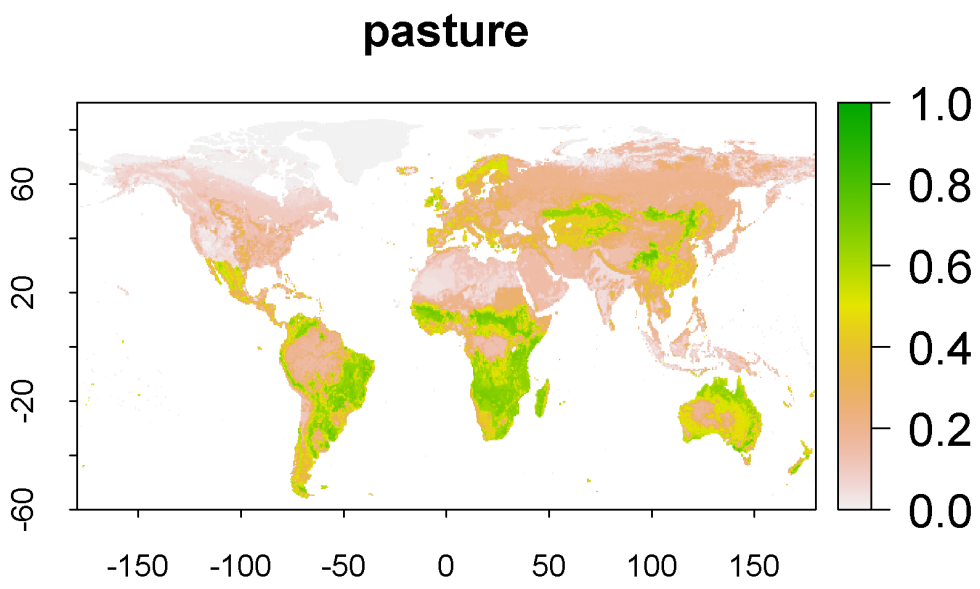
Now we can aggregate these mosaics to some (arbitrary) underlying land use classes:

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

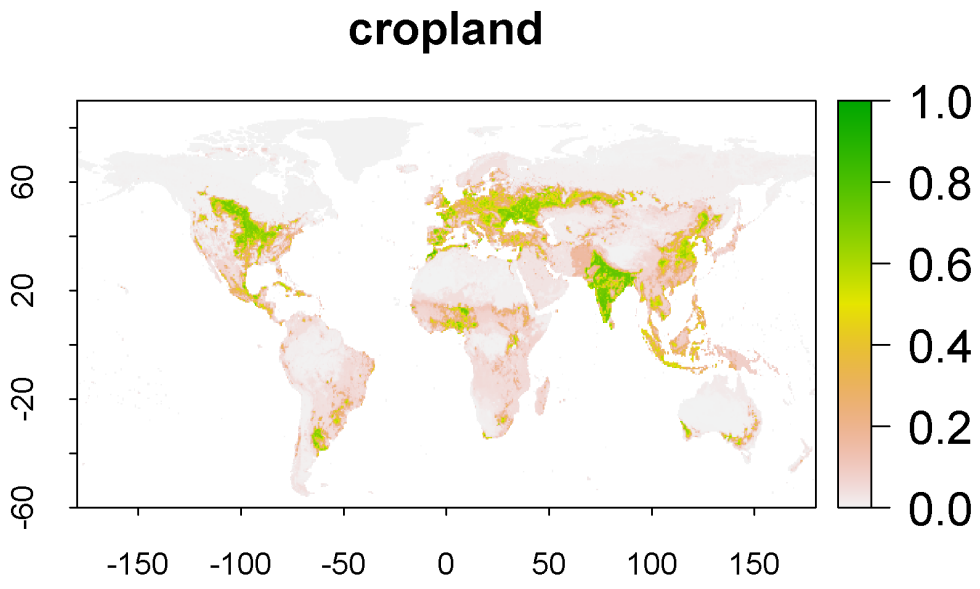


An example for aggregating the CLUMondo Land System map of the year 2000 is as easy as feeding the function `aggregateMosaicsClumondo()` with the output of the resampling-algorithm (`out`), the mosaic composition tables of all world regions (`CLUMosaics`) and with the LPJmL grid containing information for every pixel about the worldregion it belongs to (`lpjGrid`):

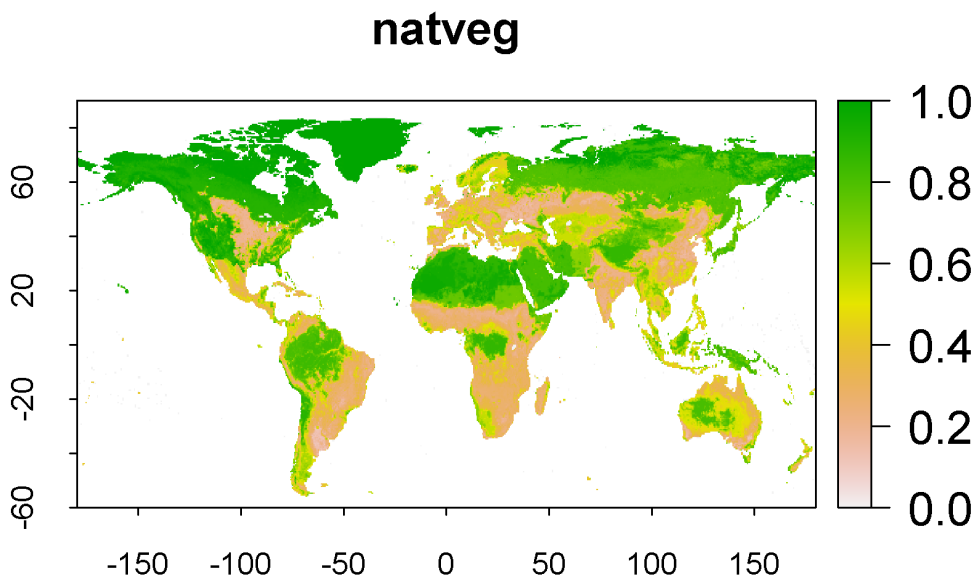
```
> out <- resample_grid(CLUlonglat, generate_grid(), cells=lpj_long_clupos)
> cluAgg <- aggregateMosaicsClumondo(out, CLUMosaics, lpjGrid)

> gridPlot(cluAgg@data[,"pasture"], zlim=c(0,1), main="pasture")
```




```
> gridPlot(cluAgg@data[,"cropland"], zlim=c(0,1), main="cropland")
```



```
> gridPlot(cluAgg@data[,"natveg"], zlim=c(0,1), main="natveg")
```



Please note: At the moment the look up table for the USA is wrong. As a workaround the look up table for Canada is used for the US instead for the time being (see extra tranlastion vignette for an explanation of the problem).

7 Translating CLUMondo Land Use to Crop Funtional Types

7.1 A Method based on an existing and recent land use map

Which crop functional types are growing on cropland can be looked up on an existing map (e.g. a specific year of the standard landuse input data of LPJmL might be used). For that, the existing map is screened in a rectangle around the pixel of interest for crops planted there. The ratio of the different crops on all surrounding pixels than is calculated. In case the screened area of the look up map does not have crops, the mean crop ratios of the corresponding CLUMondo world region is calculated and applied. Using this approach, the preference for cultivating specific crops in a specific area stays constant over time and does not consider scenarios with a change of of cultivation preferences. Accuracy of estimating CFTs from neighbour-cells reaches Nash-Sutcliffe-Efficiencies up to 0.9 for small windows, but diminishes with increasing window sizes (Fig. 5).

A complete CLUMondo to LPJmL translation exercise would look as follows:

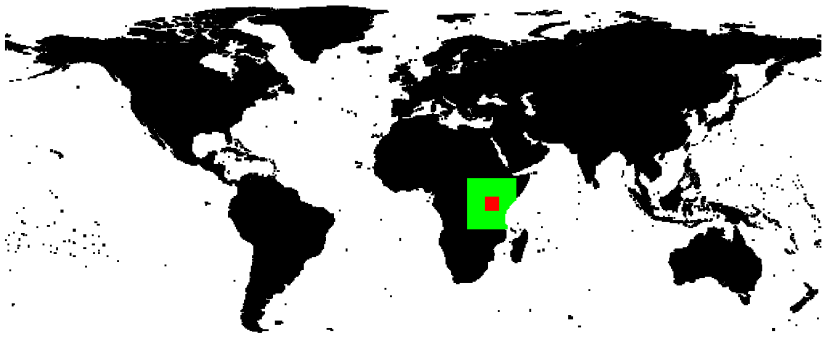


Figure 4: Example of a 10 degree screening window (green) for specific crop functional types around the pixel of interest (red).

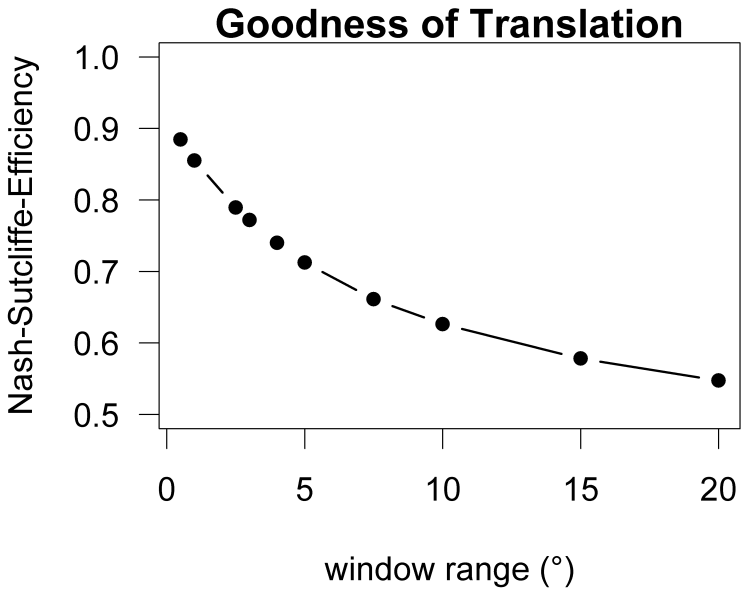


Figure 5: Nash-Sutcliffe-Efficiency of the CFT-assignment to CLUMondo cropland based on the year 2000 LPJmL input map in dependency of the window size. All crops, except pasture were included in the statistical calculation.

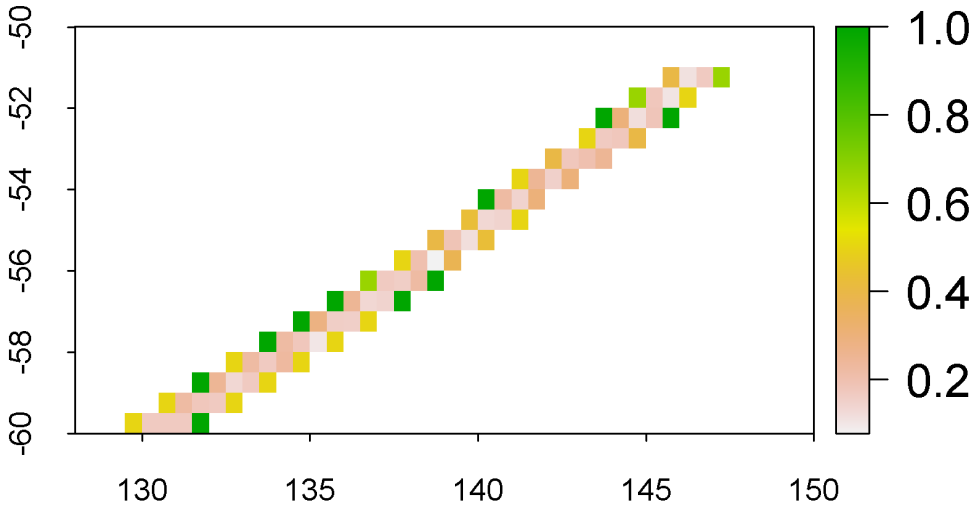
```
> library(luess)
> ### read the LPJmL land use map of the year 2000
> ## (adjust path!, landuse file is not included in this package)
> cftFrac   <- getLPJ( "N:/vmshare/landuse/landuse.bin",
+   2000, 2000, 1700, 32, 2, sizeof_header=43)
> ### re-project the original CLUMondo landuse map for the year 2040
> clu2040     <- transform_asciigrid("../../work/clu2040.txt")
> ### resample CLUMondo land use systems for the LPJmL-grid
> out2040     <- resample_grid(clu2040, generate_grid(), cells=lpj_long_clupos)
> ### aggregate the land covers of the CLUMondo land use systems
> cluAgg2040  <- aggregateMosaicsClumondo(out2040, CLUMosaics, lpjGrid)
> ### translate cropland to specific CFTs
> ## window size = 2.5 degree, takes some minutes time
> out_trans_2_5 <- translateCluToLpj(
+   lpjGrid,
+   range=2.5,
+   landuse=cftFrac,
+   landuseClu=cluAgg2040,
+   scaleFactor=1000
+ )
> ### construct land use data set
> ## use CLUMondo pasture area directly
> out_trans_2_5[1,,14] <- cluAgg2040@data[,"pasture"]*1000
> ### write the land use data set to a file
> ## get an valid LPJmL header
> ## (adjust path, file not included in this package)
> cftHeader <- readLpjHeader("N:/vmshare/landuse/landuse.bin")
> ## update number of years in header (only 1 in this example)
> cftHeader$ny <- 1
> ## write file
> writeLpjLanduse(out_trans_2_5, "landuse-new.bin", cftHeader)
> ### read new file for validation
> cftNew <- getLPJ("landuse-new.bin", 2000,2000,2000,32,2,sizeof_header=43)
> identical(cftNew, out_trans_2_5)
```

8 Changes and new functionality in latest package versions

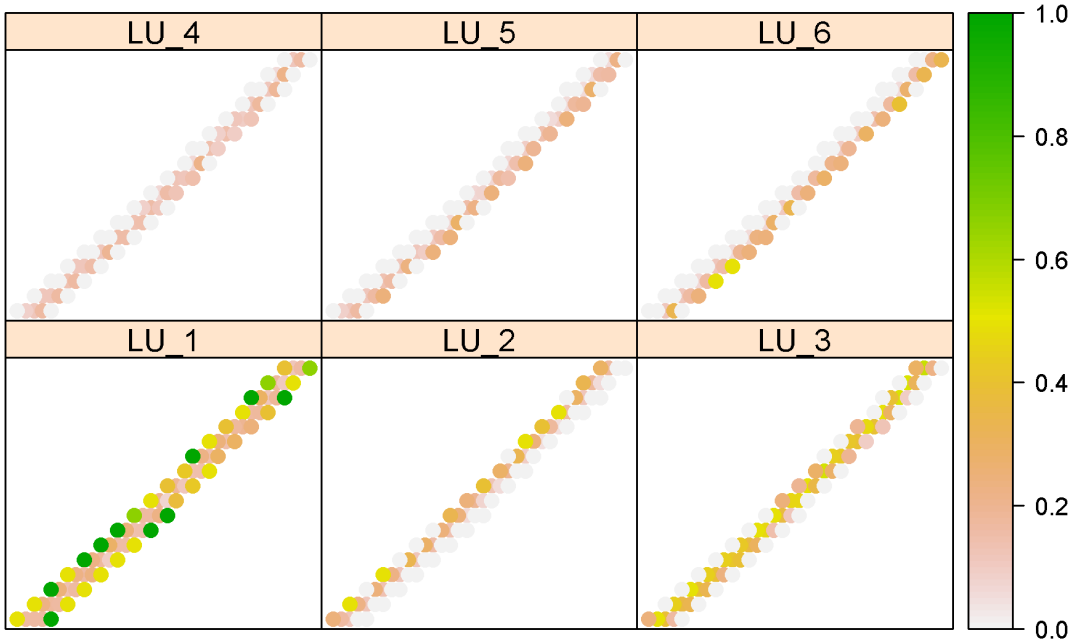
8.1 v0.16.0: resample_grid() output format

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

       LU_1       LU_2      LU_3       LU_4       LU_5      LU_6
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)

       LU_1       LU_2      LU_3       LU_4       LU_5      LU_6
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)
> names(out)

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

8.2 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)
```

8.3 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.

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.

Asselen, S. v. and Verburg, P. H. v. (2013). Land cover change or land use intensification: simulting land system change with a global-scale land change model. *Global Change Biology*, 19(12):3648–3667.