Rain filtering

2015 Aug 10: code changed to assess number of blips based on existing filtering, not just unfiltered patch count.

This page details the current approach for automatically filtering rain from blipmovies. Users should be aware that the use of this use of code will eliminate scans that contain more than the specified number of blips. So, applying the code during data collection isn't recommended (radR is not 'smart' enough to know whether the targets are actually rain or not; if a scan contains enough blips to be considered rain, yet all of the targets are birds, the scan will be eliminated anyway), and careful selection of how many blips are enough to indicate rain is encouraged.

Note that this is a crude approach, but will eliminate much of the worst parts of blipmovies. Eliminating these scans will have a huge impact on how long it takes to build tracks for a whole blipmovie.

We use a PATCH_STATS hook function. The PATCH_STATS hook allows a user function to decide exactly which patches are to be treated as blips. The PATCH_STATS hook functions are called after all patch characteristic (number of samples, area, angular span, etc.) have been tabulated in the data frame RSS$patches. The hook function receives a single parameter, which is a logical vector with one element per patch. TRUE elements indicate patches which have "passed" all other filters (e.g. num_samples > 100) and are considered blips. FALSE elements indicate patches which have "failed" at least one filter. A user patch stats function can return NULL, to indicate it is not changing the status of any patches (i.e. including or excluding any as blips); or it must return a logical vector of the same length, indicating which patches are blips. This returned vector can simply be the parameter it receives, which has the effect of not changing any blips. Here is an example of how to drop "rain" scans:

```
num.blips.cutoff <- 500

drop.crowded.scans <- function(x) {
   if (sum(x) > num.blips.cutoff) ## if there are at least num.blips.cuto
     return (rep(FALSE, length(x))) ## return a FALSE for each patch, fil
   else
     return(NULL) ## otherwise, don't do any further filtering; could als
}
```

```
## now add the function to the PATCH_STATS hook, which calls it every sca
## right before anything is done with the blips
rss.add.hook("PATCH_STATS", "norain", drop.crowded.scans, read.only=FALS
```

We set a cutoff which determines which scans will be dropped (namely those with more patches than the cutoff).

The function `drop.crowded.scans()` examines the number of blips (namely `sum(x)`), and if this is larger than the cutoff, the function returns an all FALSE vector, which means no patches are accepted as blips. Otherwise, the function returns NULL to indicate it does not want to change the status of any blips.

The call to `rss.add.hook()` is what "installs" the hook function, so that radR knows to call it at the appropriate stage in the processing of each scan. We give it the name "norain" so it won't overwrite any PATCH_STATS hook function defined by a plugin or by the use at the console. (each hook can have as many hook functions as desired, but they must all have distinct names).

The code listed above can be added to `the file radR/main/startup.windows.R` or `radR/main/startup.unix.R` as appropriate for your computer. A version which logs the timestamps of dropped scans would look like this:

```
num.blips.cutoff <- 500

## increase default printing precision so timestamps are output with a si
options(digits=11)

## add a hook function to open a rainscans file when the blipmovie is ope
rss.add.hook("START_SOURCE", function(...)  {
    rain.scans.file <<- file(paste(RSS$source$file.basename, ".rainscans
    cat ("timestamp,ts.string\n", file=rain.scans.file)
    })

plot.title.date.format <- "%Y %b %d %H:%M:%OS1" ## copy of GUI$plot.title

drop.crowded.scans <- function(x) {
   if (sum(x) > num.blips.cutoff) {
     ## if we're really processing this scan (not just browsing through t
     ## then write timestamp in two formats
     if (RSS$play.state >= RSS$PS$PLAYING && !RSS$previewing)
        cat (RSS$scan.info$timestamp, ",", format(structure(RSS$scan.info
     return (rep(FALSE, length(x)))
   } else {
     return(NULL)
   }
}

rss.add.hook("PATCH_STATS", "norain", drop.crowded.scans, read.only=FALS
```

Note that this version writes the timestamp to a file in two formats:
raw numeric, and formatted as in the radR plot window titlebar.
Quitting radR will close the file.