Package ‘rcaiman’

July 23, 2025

Type Package

Title CAnopy IMage ANalysis

Version 1.2.2

Date 2023-11-14

Description Classify hemispherical
photographs of the plant canopy with algorithms specially developed for
such a task and well documented in
Díaz and Lencinas (2015) <doi:10.1109/lgrs.2015.2425931> and
Díaz and Lencinas (2018) <doi:10.1139/cjfr-2018-0006>. It supports
non-circular hemispherical photography, such as those acquired with
15mm lenses or with auxiliary fish-eye lenses attached to mobile devices.
For smartphone-based hemispherical photography see
Díaz (2023) <doi:10.1111/2041-210x.14059>. Most of the functions also
support restricted view photography.

License GPL-3

BugReports `https://github.com/GastonMauroDiaz/rcaiman/issues`

Encoding UTF-8

RoxygenNote 7.2.3

Depends filenamer, magrittr, colorspace, terra

Imports methods, testthat, pracma, stats, utils, Rdpack, spatial,
lidR, tcltk

Suggests autothresholdr, conicfit, EBImage, bbmle, imager, reticulate

RdMacros Rdpack

NeedsCompilation no

Author Gastón Mauro Díaz [aut, cre] (ORCID:
<`https://orcid.org/0000-0002-0362-8616`>)

Maintainer Gastón Mauro Díaz <`gastonmaurodiaz@gmail.com`>

Repository CRAN

Date/Publication 2023-11-14 19:20:02 UTC

Contents

`apply_thr` *Apply threshold*

Description

Global or local thresholding of images.

Usage

```
apply_thr(r, thr)
```

Arguments

`r`	SpatRaster. A greyscale image.
`thr`	Numeric vector of length one or a single-layer raster from the class SpatRaster. Threshold.

Details

It is a wrapper function around the operator > from the `terra` package. If a single threshold value is provided as the `thr` argument, it is applied to every pixel of the object `r`. If instead a SpatRaster is provided, a particular threshold is applied to each particular pixel.

Value

An object of class SpatRaster with values `0` and `1`.

See Also

Other Binarization Functions: `obia()`, `ootb_mblt()`, `ootb_obia()`, `regional_thresholding()`, `thr_isodata()`, `thr_mblt()`

Examples

```
r <- read_caim()
bin <- apply_thr(r$Blue, thr_isodata(r$Blue[]))
plot(bin)
## Not run:
# This function is useful in combination with the ‘autothresholdr’
# package. For example:
require(autothresholdr)
thr <- auto_thresh(r$Blue[], "IsoData")[1]
bin <- apply_thr(r$Blue, thr)
plot(bin)

## End(Not run)
```

azimuth_image *Build azimuth image*

Description

Build a single-layer image with azimuth angles as pixel values, assuming upwards-looking hemispherical photography with the optical axis vertically aligned.

Usage

```
azimuth_image(z, orientation = 0)
```

Arguments

`z`	SpatRaster built with `zenith_image()`.
`orientation`	Azimuth angle (degrees) at which the top of the image was pointing at the moment of taking the picture. This design decision was made because the usual field protocol is recording the angle at which the top of the camera points.

Value

An object of class SpatRaster with azimuth angles in degrees. If the `orientation` argument is zero, North (0º) is pointing up as in maps, but East (90º) and West (270º) are flipped regarding to maps. To understand why, take two flash-card size pieces of paper; put one on a table in front of you and draw on it a compass rose; take the other and hold it with your arms extended over your head and, following the directions of the compass rose in front of you, draw another one in the paper side that is facing down—it will be an awkward position, like if you were taking an upward-looking photo with a mobile device while looking at the screen—; finally, put it down and compare both compass roses.

See Also

Other Lens Functions: `calc_diameter()`, `calc_relative_radius()`, `calc_zenith_colrow()`, `calibrate_lens()`, `crosscalibrate_lens()`, `expand_noncircular()`, `extract_radiometry()`, `fisheye_to_equidistant()`, `fisheye_to_pano()`, `lens()`, `test_lens_coef()`, `zenith_image()`

Examples

```
z <- zenith_image(600, lens("Nikon_FCE9"))
a <- azimuth_image(z)
plot(a)
## Not run:
a <- azimuth_image(z, 45)
plot(a)

## End(Not run)
```

`calc_co` *Calculate canopy openness*

Description

Calculate canopy openness

Usage

```
calc_co(bin, m = NULL, z, a, angle_width)
```

Arguments

`bin`	SpatRaster. Binarized hemispherical canopy image.
`m`	SpatRaster. A mask. For hemispherical photographs, check `mask_hs()`.
`z`	SpatRaster built with `zenith_image()`.
`a`	SpatRaster built with `azimuth_image()`.
`angle_width`	Numeric vector of length one. It should be `30, 15, 10, 7.5, 6, 5, 3.75, 3, 2.5, 1.875, 1` or `0.5` degrees. This constrain is rooted in the requirement of a value able to divide both the `0` to `360` and `0` to `90` ranges into a whole number of segments.

Details

Canopy openness calculated as in the equation from Gonsamo et al. (2011):

$CO = \sum_{i=1}^{N} GF(\phi_i, \theta_i) \cdot [(cos(\theta_1) - cos(\theta_2))/n]$,

where $GF(\phi_i, \theta_i)$ is the gap fraction of the cell i, θ_1 and θ_2 are the minimum and maximum zenith angle of the cell i, n is the number of cells on the ring delimited by θ_1 and θ_2, and N is the total number of cells.

When a mask is provided through the `m` argument, the equation is modified as follow:

$\frac{CO=\sum_{i=1}^{N} GF(\phi_i,\theta_i)\cdot[(cos(\theta_1)-cos(\theta_2))/n]}{\sum_{i=1}^{N}(cos(\theta_1)-cos(\theta_2))/n}$.

This allows the masking of any individual cell.

Value

Numeric vector of length one.

References

Gonsamo A, Walter JN, Pellikka P (2011). "CIMES: A package of programs for determining canopy geometry and solar radiation regimes through hemispherical photographs." *Computers and Electronics in Agriculture*, **79**(2), 207–215. doi:10.1016/j.compag.2011.10.001.

Examples

```
caim <- read_caim()
z <- zenith_image(ncol(caim), lens())
a <- azimuth_image(z)
m <- mask_hs(z, 0, 70)
bin <- apply_thr(caim$Blue, thr_isodata(caim$Blue[m]))
plot(bin)
calc_co(bin, m, z, a, 10)
```

`calc_diameter` *Calculate diameter*

Description

Calculate the diameter in pixels of a 180º fisheye image.

Usage

```
calc_diameter(lens_coef, radius, angle)
```

Arguments

`lens_coef`	Numeric vector. Polynomial coefficients of the lens projection function. See `calibrate_lens()`.
`radius`	Numeric vector. Distance in pixels from the zenith.
`angle`	Numeric vector. Zenith angle in degrees.

Details

This function helps handle devices with a field of view different than 180º. Given a lens projection function and data points consisting of radii (pixels) and their correspondent zenith angle (θ), it returns the horizon radius (i.e., the radius for θ equal to 90º).

When working with non-circular hemispherical photography, this function will help to find the diameter that a circular image would have if the equipment would record the whole hemisphere.

The required data (radius-angle data pairs) can be obtained following the instructions given in the user manual of Hemisfer software. The following is a slightly simpler alternative:

- Find a vertical wall and a leveled floor, both well-constructed.
- Draw a triangle of $5 \times 4 \times 3$ meters on the floor, with the 4-meter side over the wall.
- Locate the camera over the vertex that is 3 meters away from the wall. Place it at a given height above the floor, 1.3 meters for instance.
- Make a mark on the wall at the chosen height over the wall-vertex nearest to the camera-vertex. Make four more marks with one meter of spacing and following a horizontal line. This will create marks for 0º, 18º, 34º, 45º, and 54º θ.
- Before taking the photograph, do not forget to align the zenith coordinates with the 0º θ mark and check if the optical axis is leveled.

The line selection tool of ImageJ can be used to measure the distance in pixels between points on the image. Draw a line, and use the dropdown menu Analyze>Measure to obtain its length.

For obtaining the projection of a new lens, refer to `calibrate_lens()`.

Value

Numeric vector of length one. Diameter adjusted to a whole number (see `zenith_image()` for details about that constrain).

See Also

Other Lens Functions: `azimuth_image()`, `calc_relative_radius()`, `calc_zenith_colrow()`, `calibrate_lens()`, `crosscalibrate_lens()`, `expand_noncircular()`, `extract_radiometry()`, `fisheye_to_equidistant()`, `fisheye_to_pano()`, `lens()`, `test_lens_coef()`, `zenith_image()`

Examples

```
# Nikon D50 and Fisheye Nikkor 10.5mm lens
calc_diameter(lens("Nikkor_10.5mm"), 1202, 54)
```

`calc_relative_radius` *Calculate relative radius*

Description

Calculate the relative radius given a zenith angle and lens function. This is known as the projection function.

Usage

```
calc_relative_radius(angle, lens_coef)
```

Arguments

`angle`	Numeric vector. Zenith angles in degrees.
`lens_coef`	Numeric vector. Polynomial coefficients of the lens projection function. See `calibrate_lens()`.

See Also

Other Lens Functions: `azimuth_image()`, `calc_diameter()`, `calc_zenith_colrow()`, `calibrate_lens()`, `crosscalibrate_lens()`, `expand_noncircular()`, `extract_radiometry()`, `fisheye_to_equidistant()`, `fisheye_to_pano()`, `lens()`, `test_lens_coef()`, `zenith_image()`

`calc_zenith_colrow` *Calculate zenith raster coordinates*

Description

Calculate zenith raster coordinates from points digitized with the open-source software package ‘ImageJ’. The zenith is the point on the image that represents the zenith when upward-looking photographs are taken with the optical axis vertically aligned.

Usage

```
calc_zenith_colrow(path_to_csv)
```

Arguments

`path_to_csv`	Character vector. Path to a CSV file created with the point selection tool of ‘ImageJ’ software.

Details

The technique described under the headline 'Optical center characterization' of the user manual of the software Can-Eye can be used to acquire the data for determining the zenith coordinates. This technique was used by Pekin and Macfarlane (2009), among others. Briefly, it consists in drilling a small hole in the cap of the fisheye lens (it must be away from the center of the cap), and taking about ten photographs without removing the cap. The cap must be rotated about 30º before taking each photograph.(**NOTE:** The method implemented here does not support multiple holes).

The point selection tool of 'ImageJ' software should be used to manually digitize the white dots and create a CSV file to feed this function. After digitizing the points on the image, use the dropdown menu Analyze>Measure to open the Results window. To obtain the CSV file, use File>Save As...

Another method–only valid when enough of the circle perimeter is depicted in the image– is taking a very bright picture (for example, a picture of the corner of a room with walls painted in light colors) with the lens completely free (do not use any mount). Then, digitize points over the circle perimeter. This was the method used for producing the example file (see Examples). It is worth noting that the perimeter of the circle depicted in a circular hemispherical photograph is not necessarily the horizon.

Value

Numeric vector of length two. Raster coordinates of the zenith, assuming a lens facing up with its optical axis parallel to the vertical line. It is important to note the difference between the raster coordinates and the Cartesian coordinates. In the latter, the vertical axis value decreases downward, but the opposite is true for the raster coordinates, which works like a spreadsheet.

References

Pekin B, Macfarlane C (2009). "Measurement of crown cover and leaf area index using digital cover photography and its application to remote sensing." *Remote Sensing*, **1**(4), 1298–1320. doi:10.3390/rs1041298.

See Also

Other Lens Functions: `azimuth_image()`, `calc_diameter()`, `calc_relative_radius()`, `calibrate_lens()`, `crosscalibrate_lens()`, `expand_noncircular()`, `extract_radiometry()`, `fisheye_to_equidistant()`, `fisheye_to_pano()`, `lens()`, `test_lens_coef()`, `zenith_image()`

Examples

```
## Not run:
path <- system.file("external/points_over_perimeter.csv",
                    package = "rcaiman")
calc_zenith_colrow(path)

## End(Not run)
```

`calibrate_lens` *Calibrate lens*

Description

Calibrate a fisheye lens

Usage

```
calibrate_lens(path_to_csv, degree = 3)
```

Arguments

`path_to_csv`	Character vector. Path to a CSV file created with the point selection tool of 'ImageJ' software.
`degree`	Numeric vector of length one. Polynomial model degree.

Details

Fisheye lenses have a wide field of view and the same distortion in all directions running orthogonally to the optical axis. The latter property allows fitting a precise mathematical relationship between distances to the zenith on the image space and zenith angles on the hemispherical space (assuming upward-looking hemispherical photography with the optical axis vertically aligned).

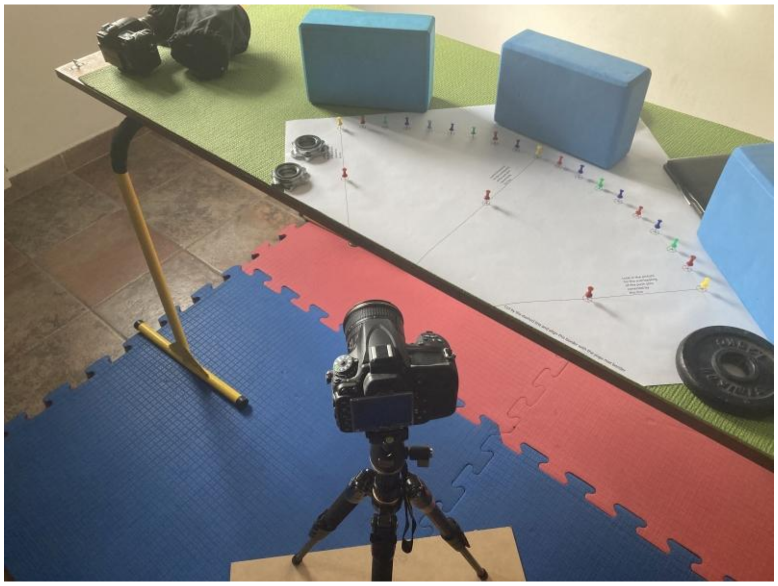
The method outlined here, known as the simple method, is explained in details in Díaz and et al. (2024). Next explanation might serve mostly as a handbook.

Step-by-step guide for producing a CSV file to feed this function:

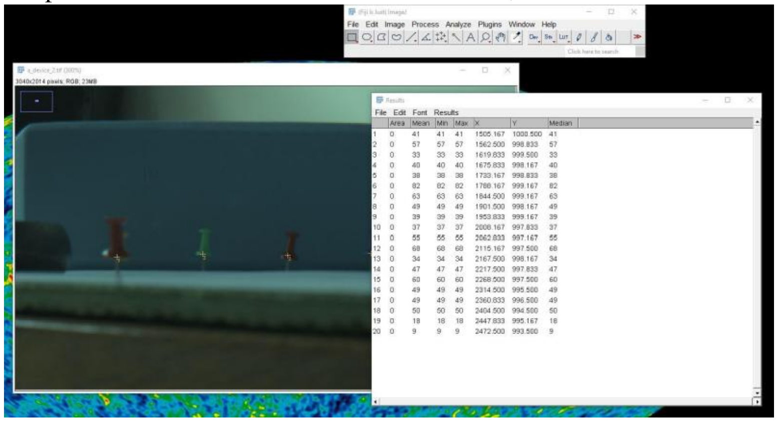
Materials:

- this package and ImageJ
- camera and lens
- tripod
- standard yoga mat
- table at least as wide as the yoga mat width
- twenty two push pins of different colors
- one print of this sheet (A1 size, almost like a research poster).
- scissors
- some patience

Instructions:
Cut the sheet by the dashed line. Place the yoga mat extended on top of the table. Place the sheet on top of the yoga mat. Align the dashed line with the yoga mat border closest to you. Place push pins on each cross. If you are gentle, the yoga mat will allow you to do that without damaging the table. Of course, other materials could be used to obtain the same result, such as cardboard, foam, nails, etc.



Place the camera on the tripod. Align its optical axis with the table while looking for getting an image showing the overlapping of the three pairs of push pins, as instructed in the print. In order to take care of the line of pins at 90º relative to the optical axis, it may be of help to use the naked eye to align the entrance pupil of the lens with the pins. The alignment of the push pins only guarantees the position of the lens entrance pupil, the leveling should be cheeked with an instrument, and the alignment between the optical axis and the radius of the zenith push pin should be taken into account. In practice, the latter is achieved by aligning the camera body with the orthogonal frame made by the quarter circle.
Take a photo and transfer it to the computer, open it with ImageJ, and use the point selection tool to digitize the push pins, starting from the zenith push pin and not skipping any shown push pin. End with an additional point where the image meets the surrounding black (or the last pixel in case there is not blackness because it is not a circular hemispherical image. There is no need to follow the line formed by the push pins). Then, use the dropdown menu Analyze>Measure to open the window Results. To obtain the CSV, use File>Save As...



	Area	Mean	Min	Max	X	Y	Median
1	0	41	41	41	1505.167	1000.500	41
2	0	57	57	57	1562.500	998.833	57
3	0	33	33	33	1619.833	999.500	33
4	0	40	40	40	1675.833	998.167	40
5	0	38	38	38	1733.167	998.833	38
6	0	82	82	82	1788.167	999.167	82
7	0	63	63	63	1844.500	999.167	63
8	0	49	49	49	1901.500	998.167	49
9	0	39	39	39	1953.833	999.167	39
10	0	37	37	37	2008.167	997.833	37
11	0	55	55	55	2062.833	997.167	55
12	0	68	68	68	2115.167	997.500	68
13	0	34	34	34	2167.500	998.167	34
14	0	47	47	47	2217.500	997.833	47
15	0	60	60	60	2268.500	997.500	60
16	0	49	49	49	2314.500	995.500	49
17	0	49	49	49	2360.833	996.500	49
18	0	50	50	50	2404.500	994.500	50
19	0	18	18	18	2447.833	995.167	18
20	0	9	9	9	2472.500	993.500	9

Use `test_lens_coef()` to test if coefficients are OK.

Value

An object of class *list* with named elements. *ds* is the dataset used to fit the model, *model* is the fitted model (class `lm`, see `stats::lm()`), *horizon_radius* is the radius at 90º, *lens_coef* is a numeric vector of length equal to the `degree` argument containing the polynomial model coefficients for predicting relative radius (`coefficients(model)/horizon_radius`), *zenith_colrow* are the raster coordinates of the zenith push pin, *max_theta* is the maximum zenith angle in degrees, and *max_theta_px* is the distance in pixels between the zenith and the maximum zenith angle in pixels units.

Note

If we imagine the fisheye image as an analog clock, it is possible to calibrate 3 o'clock by attaching the camera to the tripod in landscape mode while leaving the quarter-circle at the lens's right side. To calibrate 9 o'clock, it will be necessary to rotate the camera to put the quarter-circle at the lens's left side. To calibrate 12 and 6 o'clock, it will be necessary to do the same but with the camera in portrait mode. If several directions are sampled with this procedure, a character vector of length greater than one in which each element is a path to a CSV files could be provided as the `path_to_csv` argument.

References

Díaz GM, et al. (2024). "Simple calibration of fisheye lenses for hemipherical photography of the forest canopy." *Manuscript in preparation*.

See Also

Other Lens Functions: `azimuth_image()`, `calc_diameter()`, `calc_relative_radius()`, `calc_zenith_colrow()`, `crosscalibrate_lens()`, `expand_noncircular()`, `extract_radiometry()`, `fisheye_to_equidistant()`, `fisheye_to_pano()`, `lens()`, `test_lens_coef()`, `zenith_image()`

Examples

```
path <- system.file("external/Results_calibration.csv", package = "rcaiman")
calibration <- calibrate_lens(path)
coefficients(calibration$model)
calibration$lens_coef %>% signif(3)
calibration$horizon_radius

## Not run:
test_lens_coef(calibration$lens_coef) #MacOS and Windows tend to differ here
test_lens_coef(c(0.628, 0.0399, -0.0217))

## End(Not run)

.fp <- function(theta, lens_coef) {
  x <- lens_coef[1:5]
  x[is.na(x)] <- 0
  for (i in 1:5) assign(letters[i], x[i])
  a * theta + b * theta^2 + c * theta^3 + d * theta^4 + e * theta^5
}
```

```
plot(calibration$ds)
theta <- seq(0, pi/2, pi/180)
lines(theta, .fp(theta, coefficients(calibration$model)))
```

`chessboard` *Do chessboard segmentation*

Description

Do chessboard segmentation

Usage

```
chessboard(r, size)
```

Arguments

`r`	SpatRaster.
`size`	Numeric vector of length one. Size of the square segments.

Value

A single layer image of the class SpatRaster with integer values.

See Also

Other Segmentation Functions: `mask_hs()`, `mask_sunlit_canopy()`, `polar_qtree()`, `qtree()`, `rings_segmentation()`, `sectors_segmentation()`, `sky_grid_segmentation()`

Examples

```
caim <- read_caim()
seg <- chessboard(caim, 20)
plot(caim$Blue)
plot(extract_feature(caim$Blue, seg))
```

`cie_sky_model_raster` *CIE sky model raster*

Description

CIE sky model raster

Usage

```
cie_sky_model_raster(z, a, sun_coord, sky_coef)
```

Arguments

`z`	SpatRaster built with `zenith_image()`.
`a`	SpatRaster built with `azimuth_image()`.
`sun_coord`	Numeric vector of length two. The solar disk center represented with zenith and azimuth angles in degrees.
`sky_coef`	Numeric vector of length five. Parameters of the sky model.

See Also

Other Sky Reconstruction Functions: `fit_cie_sky_model()`, `fit_coneshaped_model()`, `fit_trend_surface()`, `fix_reconstructed_sky()`, `interpolate_sky_points()`, `ootb_sky_reconstruction()`

Examples

```
z <- zenith_image(50, lens())
a <- azimuth_image(z)
path <- system.file("external", package = "rcaiman")
skies <- read.csv(file.path(path, "15_CIE_standard_skies.csv"))
# parameters are from http://dx.doi.org/10.1016/j.energy.2016.02.054
sky_coef <- skies[4,1:5]
sun_coord <- c(45, 0)
plot(cie_sky_model_raster(z, a, sun_coord, sky_coef))
```

`colorfulness` *Quantify colorfulness*

Description

Quantify the colorfulness of an image

Usage

```
colorfulness(caim, m = NULL)
```

Arguments

`caim`	SpatRaster. The return of a call to `read_caim()` or `read_caim_raw()`.
`m`	SpatRaster. A mask. For hemispherical photographs, check `mask_hs()`. Default (`NULL`) is the equivalent to enter `!is.na(caim$Red)`.

Details

Quantify the colorfulness of an sRGB image using a bidimensional space formed by the green/red and the blue/yellow axes of the *CIE LAB* space, symbolized with *A* and *B*, respectively. The colorfulness index (CI) is defined as

$$CI = \frac{A_o}{A_p} \cdot 100,$$

where A_o and A_p are the observed and potential area of the *AB* plane. A_o refers to the colors from the image while A_p to the colors from the whole sRGB cube.

Value

A numeric vector of length one.

Note

An early version of this function was used in Martin et al. (2020).

References

Martin DA, Wurz A, Osen K, Grass I, Hölscher D, Rabemanantsoa T, Tscharntke T, Kreft H (2020). “Shade-Tree Rehabilitation in Vanilla Agroforests is Yield Neutral and May Translate into Landscape-Scale Canopy Cover Gains.” *Ecosystems*, **24**(5), 1253–1267. doi:10.1007/s10021020-005865.

See Also

Other Tool Functions: `correct_vignetting()`, `defuzzify()`, `extract_dn()`, `extract_feature()`, `extract_rl()`, `extract_sky_points_simple()`, `extract_sky_points()`, `extract_sun_coord()`, `find_sky_pixels_nonnull()`, `find_sky_pixels()`, `masking()`, `optim_normalize()`, `percentage_of_clipped_highli` `read_bin()`, `read_caim_raw()`, `read_caim()`, `write_bin()`, `write_caim()`

Examples

```
caim <- read_caim() %>% normalize()
colorfulness(caim)
```

correct_vignetting *Correct vignetting effect*

Description

Correct vignetting effect

Usage

```
correct_vignetting(r, z, lens_coef_v)
```

Arguments

r	SpatRaster. A fish-eye image.
z	SpatRaster built with `zenith_image()`.
lens_coef_v	Numeric vector. Coefficients of a vignetting function (f_v) of the type $f_v = 1 + a \cdot \theta + b \cdot \theta^2 + ... + m \cdot \theta^n$, where θ is the zenith angle, a, b, c and m are the coefficients. The maximum polynomial degree supported is fifth. See `extract_radiometry()` for additional details.

Value

The argument `r` but with corrected values.

See Also

Other Tool Functions: `colorfulness()`, `defuzzify()`, `extract_dn()`, `extract_feature()`, `extract_rl()`, `extract_sky_points_simple()`, `extract_sky_points()`, `extract_sun_coord()`, `find_sky_pixels_nonnull()`, `find_sky_pixels()`, `masking()`, `optim_normalize()`, `percentage_of_clipped_highlights()`, `read_bin()`, `read_caim_raw()`, `read_caim()`, `write_bin()`, `write_caim()`

Examples

```
path <- system.file("external/DSCN4500.JPG", package = "rcaiman")
caim <- read_caim(path, c(1250, 1020) - 745, 745 * 2, 745 * 2)
z <- zenith_image(ncol(caim), lens("Nikon_FCE9"))
r <- gbc(caim$Blue)
r
r <- correct_vignetting(r, z, c(0.0638, -0.101))
r
```

`crop_caim` *Crop a canopy image from a file*

Description

Function that complements `read_caim()` and `read_caim_raw()`

Usage

```
crop_caim(r, upper_left = NULL, width = NULL, height = NULL)
```

Arguments

`r`	SpatRaster
`upper_left`	An integer vector of length two. The pixels coordinates of the upper left corner of a region of interest (ROI). These coordinates should be in the raster coordinates system. This system works like a spreadsheet, i.e, when going down through the vertical axis, the *row* number increases (**IMPORTANT**: column and row must be provided instead of row and column, as is the norm for objects from the class *data.frame* and others alike)
`width, height`	An integer vector of length one. The size of the boxy ROI whose upper left corner is the `upper_left` argument.

Value

SpatRaster

Examples

```
caim <- read_caim()
ncell(caim)
caim <- crop_caim(caim, c(231,334), 15, 10)
ncell(caim)
```

`crosscalibrate_lens` *Cross-calibrate lens*

Description

Cross-calibrate lens

Usage

```
crosscalibrate_lens(
  path_to_csv_uncal,
  path_to_csv_cal,
  zenith_colrow_uncal,
  zenith_colrow_cal,
  diameter_cal,
  lens_coef,
  degree = 3
)
```

Arguments

`path_to_csv_uncal, path_to_csv_cal`	Character vector of length one. Path to a CSV file created with the point selection tool of 'ImageJ' software (*cal* and *uncal* stand for calibrated and uncalibrated, respectively).
`zenith_colrow_uncal, zenith_colrow_cal`	Numeric vector of length two. Raster coordinates of the zenith. See `calc_zenith_colrow()` (*cal* and *uncal* stand for calibrated and uncalibrated, respectively).
`diameter_cal`	Numeric vector of length one. Diameter in pixels of the image taken with the calibrated camera.
`lens_coef`	numeric
`degree`	Numeric vector of length one. Polynomial model degree.

Details

Read the help page of `calibrate_lens()` for understanding the theory being this function.

This function is intended to be used when a camera calibrated with a method of higher accuracy than the one proposed in `calibrate_lens()` is available or there is a main camera to which all other devices should be adjusted.

It requires two photographs taken from the exact same location with the calibrated and uncalibrated camera. This means that the lens entrance pupils should match and the optical axes should be aligned.

Points should be digitized in tandem with ImageJ and saved as CSV files.

Value

An object of class *list* with named elements. *ds* is the dataset used to fit the model, *model* is the fitted model (class `lm`, see `stats::lm()`), *horizon_radius* is the radius at 90º, *lens_coef* is a numeric vector of length equal to the degree argument containing the polynomial model coefficients for predicting relative radius (`coefficients(model)/horizon_radius`).

See Also

Other Lens Functions: `azimuth_image()`, `calc_diameter()`, `calc_relative_radius()`, `calc_zenith_colrow()`, `calibrate_lens()`, `expand_noncircular()`, `extract_radiometry()`, `fisheye_to_equidistant()`, `fisheye_to_pano()`, `lens()`, `test_lens_coef()`, `zenith_image()`

defuzzify *Defuzzify a fuzzy classification*

Description

This function translates degree of membership into Boolean logic using a regional approach. The result will ensure that the fuzzy and Boolean version will agree at the chosen level of aggregation (controlled by the argument `segmentation`). This method makes perfect sense to translate a subpixel classification of gap fraction (or a linear ratio) into a binary product.

Usage

```
defuzzify(mem, segmentation)
```

Arguments

`mem`	An object of the class SpatRaster. Degree of membership.
`segmentation`	An object of the class SpatRaster such as the result of a call to `sky_grid_segmentation()`.

Value

An object of the class SpatRaster containing binary information.

Note

This method is also available in the HSP software package (Lang et al. 2013).

References

Lang M, Kodar A, Arumäe T (2013). “Restoration of above canopy reference hemispherical image from below canopy measurements for plant area index estimation in forests.” *Forestry Studies*, **59**(1), 13–27. doi:10.2478/fsmu20130008.

See Also

Other Tool Functions: `colorfulness()`, `correct_vignetting()`, `extract_dn()`, `extract_feature()`, `extract_rl()`, `extract_sky_points_simple()`, `extract_sky_points()`, `extract_sun_coord()`, `find_sky_pixels_nonnull()`, `find_sky_pixels()`, `masking()`, `optim_normalize()`, `percentage_of_clipped_highli`
`read_bin()`, `read_caim_raw()`, `read_caim()`, `write_bin()`, `write_caim()`

Examples

```
## Not run:
path <- system.file("external/DSCN4500.JPG", package = "rcaiman")
caim <- read_caim(path, c(1250, 1020) - 745, 745 * 2, 745 * 2)
z <- zenith_image(ncol(caim), lens("Nikon_FCE9"))
a <- azimuth_image(z)
r <- gbc(caim$Blue)
```

```
r <- correct_vignetting(r, z, c(0.0638, -0.101)) %>% normalize()
bin <- find_sky_pixels(r, z, a)
bin <- ootb_mblt(r, z, a, bin)
plot(bin$bin)
ratio <- r / bin$sky_s
ratio <- normalize(ratio, 0, 1, TRUE)
plot(ratio)
g <- sky_grid_segmentation(z, a, 10)
bin2 <- defuzzify(ratio, g)
plot(bin2)
plot(abs(bin$bin - bin2))

## End(Not run)
```

---

enhance_caim *Enhance canopy image*

Description

This function was first proposed in Díaz and Lencinas (2015). It uses the color perceptual attributes (hue, lightness, and chroma) to enhance the contrast between the sky and plants through fuzzy classification. It applies the next classification rules (here expressed in natural language): clear sky is blue and clouds decrease its chroma; if clouds are highly dense, then the sky is achromatic, and, in such cases, it can be light or dark; everything that does not match this description is not sky. These linguistic rules were translated to math language by means of fuzzy logic. This translation was thoughtfully explained in the aforementioned article.

Usage

```
enhance_caim(
  caim,
  m = NULL,
  sky_blue = NULL,
  sigma = NULL,
  w_red = 0,
  thr = NULL,
  fuzziness = NULL,
  gamma = NULL
)
```

Arguments

`caim`	SpatRaster. The return of a call to `read_caim()` or `read_caim_raw()`.
`m`	SpatRaster. A mask. For hemispherical photographs, check `mask_hs()`.
`sky_blue`	color. Is the `target_color` argument to be passed to `membership_to_color()`. Default (`NULL`) is the equivalent to enter `sRGB(0.1, 0.4, 0.8)`–the HEX color code is #1A66CC, it can be entered into a search engine (such as Mozilla Firefox) to see a color swatch.

`sigma`	Numeric vector of length one. Use `NULL` (default) to estimate it automatically as the euclidean distance between `target_color` and grey in the *CIE LAB* color space.
`w_red`	Numeric vector of length one. Weight of the red channel. A single layer image is calculated as a weighted average of the blue and red channels. This layer is used as lightness information. The weight of the blue is the complement of `w_red`.
`thr`	Numeric vector of length one. Location parameter of the logistic membership function. Use `NULL` to estimate it automatically with `thr_isodata()`.
`fuzziness`	Numeric vector of length one. This number is a constant value that scales `mem`. Use `NULL` to estimate it automatically as the midpoint between the maximum and minimum values of `lightness`.
`gamma`	Numeric vector of length one. This is for applying a gamma back correction to the lightness information (see Details and argument `w_red`).

Details

This is a pixel-wise methodology that evaluates the possibility for a pixel to be member of the class *Gap*. High score could mean either high membership to `sky_blue` or, in the case of achromatic pixels, a high membership to values above `thr`. The algorithm internally uses `membership_to_color()` and `local_fuzzy_thresholding()`. The argument `sky_blue` is the `target_color` of the former function and its output is the argument `mem` of the latter function.

The argument `sky_blue` can be obtained from a photograph that clearly shows the sky. Then, it can be used to process all the others photograph taken with the same equipment, configuration, and protocol.

Via the `gamma` argument, `gbc()` can be internally used to back-correct the values passed to `local_fuzzy_thresholding()`.

Value

An object of class SpatRaster (with same pixel dimensions than `caim`) that should show more contrast between the sky and plant pixels than any of the individual bands from `caim`; if not, different parameters should be tested.

Note

If you use this function in your research, please cite Díaz and Lencinas (2015) in addition to this package (`citation("rcaiman")`).

The default value of argument `m` is the equivalent to enter `!is.na(caim$Red)`. See the Details section in `local_fuzzy_thresholding()` to understand how this argument can modify the output.

References

Díaz GM, Lencinas JD (2015). "Enhanced gap fraction extraction from hemispherical photography." *IEEE Geoscience and Remote Sensing Letters*, **12**(8), 1785–1789. doi:10.1109/lgrs.2015.2425931.

See Also

Other Pre-processing Functions: `gbc()`, `local_fuzzy_thresholding()`, `membership_to_color()`, `normalize()`

Examples

```
## Not run:
caim <- read_caim()
z <- zenith_image(ncol(caim), lens())
a <- azimuth_image(z)
m <- !is.na(z)

mn <- quantile(caim$Blue[m], 0.01)
mx <- quantile(caim$Blue[m], 0.99)
r <- normalize(caim$Blue, mn, mx, TRUE)

bin <- find_sky_pixels(r, z, a)
mblt <- ootb_mblt(r, z, a, bin)

mx <- optim_normalize(caim, mblt$bin)
mn <- min(caim[m])

sky_blue_sample <- crop_caim(caim, c(327, 239), 41, 89)
plotRGB(normalize(sky_blue_sample, mn, mx, TRUE)*255)
sky_blue <- apply(sky_blue_sample[], 2, median) %>%
  normalize(., mn, mx) %>%
  as.numeric() %>%
  matrix(., ncol = 3) %>%
  sRGB()
hex(sky_blue)
# Use hex() to obtain the HEX color code. To see a color swatch, enter the
# HEX code into a search engine (such as Mozilla Firefox).
# NOTE: see extract_dn() for an alternative method to obtain sky_blue

as(sky_blue, "HSV") #saturatio (S) is low
# To obtain same hue (H) but greater saturation
sky_blue <- HSV(239, 0.85, 0.5) %>% as(., "sRGB") %>% as(., "LAB")
hex(sky_blue)

caim <- normalize(caim, mx = mx, force_range = TRUE)
ecaim <- enhance_caim(caim, m, sky_blue = sky_blue)
plot(ecaim)
plot(caim$Blue)

## to compare
plot(apply_thr(ecaim, thr_isodata(ecaim[m])))
plot(apply_thr(caim$Blue, thr_isodata(caim$Blue[m])))

## End(Not run)
```

`expand_noncircular` *Expand non-circular*

Description

Expand a non-circular hemispherical photograph.

Usage

```
expand_noncircular(caim, z, zenith_colrow)
```

Arguments

`caim`	SpatRaster. The return of a call to `read_caim()` or `read_caim_raw()`.
`z`	SpatRaster built with `zenith_image()`.
`zenith_colrow`	Numeric vector of length two. Raster coordinates of the zenith. See `calc_zenith_colrow()`.

Value

An object of class SpatRaster that is the result of adding margins (`NA` pixels) to `caim`. The zenith point depicted in the picture should be in the center of the image or very close to it.

See Also

Other Lens Functions: `azimuth_image()`, `calc_diameter()`, `calc_relative_radius()`, `calc_zenith_colrow()`, `calibrate_lens()`, `crosscalibrate_lens()`, `extract_radiometry()`, `fisheye_to_equidistant()`, `fisheye_to_pano()`, `lens()`, `test_lens_coef()`, `zenith_image()`

Examples

```
## Not run:

 # ====================================================================
 # Non-circular Fisheye images from a Smartphone with an Auxiliary Lens
 # (Also applicable to Non-circular images from DSLR cameras)
 # ====================================================================

 path <- system.file("external/APC_0581.jpg", package = "rcaiman")
 caim <- read_caim(path)
 z <- zenith_image(2132/2,  c(0.7836, 0.1512, -0.1558))
 a <- azimuth_image(z)
 zenith_colrow <- c(1063, 771)/2
 caim <- expand_noncircular(caim, z, zenith_colrow)
 plot(caim$Blue, col = seq(0,1,1/255) %>% grey())
 m <- !is.na(caim$Red) & !is.na(z)
 plot(m, add = TRUE, alpha = 0.3, legend = FALSE)
```

```
# ===========================
# Restricted View Canopy Photo
# ===========================

path <- system.file("external/APC_0020.jpg", package = "rcaiman")
caim <- read_caim(path)
plot(caim)
caim <- normalize(caim)
diameter <- calc_diameter(lens(), sqrt(nrow(caim)^2 + ncol(caim)^2)/2, 90)
z <- zenith_image(diameter, lens())
caim <- expand_noncircular(caim, z, c(ncol(caim)/2, nrow(caim)/2))
m <- !is.na(caim$Red)
a <- azimuth_image(z)
caim[!m] <- 0
z <- normalize(z, 0, 90) * 30.15 #60.3º diagonal FOV according to metadata
plot(caim$Blue, col = seq(0,1,1/255) %>% grey())
m <- !is.na(caim$Red) & !is.na(z)
plot(m, add = TRUE, alpha = 0.3, legend = FALSE)

## End(Not run)
```

extract_dn *Extract digital numbers*

Description

Wrapper function around `terra::extract()`.

Usage

```
extract_dn(r, img_points, use_window = TRUE, fun = NULL)
```

Arguments

`r`	SpatRaster. A fish-eye image.
`img_points`	The result of a call to `extract_sky_points()`, or an object of the same class and structure.
`use_window`	Logical vector of length one. If `TRUE`, a 3×3 window will be used to extract the digital number from `r`.
`fun`	A `function` that takes a vector as input and returns a one-length numeric or logical vector as output (e.g. mean).

Value

An object of the class *data.frame*. It is the argument `img_points` with an added column per each layer from `r`. The layer names are used to name the new columns. If a function is provided as the `fun` argument, the result will be summarized per column using the provided function, and the *row* and *col* information will be omitted. Moreover, if `r` is an RGB image, a color will be returned instead of a *data.frame*. The latter feature is useful for obtaining the `sky_blue` argument for `enhance_caim()`.

Note

The point selection tool of 'ImageJ' software can be used to manually digitize points and create a CSV file from which to read coordinates (see Examples). After digitizing the points on the image, use the dropdown menu Analyze>Measure to open the Results window. To obtain the CSV file, use File>Save As...

See Also

Other Tool Functions: `colorfulness()`, `correct_vignetting()`, `defuzzify()`, `extract_feature()`, `extract_rl()`, `extract_sky_points_simple()`, `extract_sky_points()`, `extract_sun_coord()`, `find_sky_pixels_nonnull()`, `find_sky_pixels()`, `masking()`, `optim_normalize()`, `percentage_of_clipped_highli` `read_bin()`, `read_caim_raw()`, `read_caim()`, `write_bin()`, `write_caim()`

Examples

```
## Not run:
caim <- read_caim()
r <- caim$Blue
bin <- apply_thr(r, thr_isodata(r[]))
z <- zenith_image(ncol(caim), lens())
a <- azimuth_image(z)
g <- sky_grid_segmentation(z, a, 10)
sky_points <- extract_sky_points(r, bin, g)
sky_points <- extract_dn(caim, sky_points)
head(sky_points)

# ImageJ can be used to digitize points
path <- system.file("external/sky_points.csv",
                    package = "rcaiman")
sky_points <- read.csv(path)
sky_points <- sky_points[c("Y", "X")]
colnames(sky_points) <- c("row", "col")
head(sky_points)
plot(bin)
points(sky_points$col, nrow(caim) - sky_points$row, col = 2, pch = 10)
extract_dn(caim, sky_points, fun = median)

## End(Not run)
```

`extract_feature` *Extract feature*

Description

Extract features from raster images.

Usage

```
extract_feature(
  r,
  segmentation,
  fun = mean,
  return_raster = TRUE,
  ignore_label_0 = TRUE
)
```

Arguments

`r`	SpatRaster. Single layer raster.
`segmentation`	SpatRaster. The segmentation of `r`.
`fun`	A `function` that takes a vector as input and returns a one-length numeric or logical vector as output (e.g. mean).
`return_raster`	Logical vector of length one, see details.
`ignore_label_0`	Logical vector of length one. If this is `TRUE`, the segment labeled with `0` will be ignored.

Details

Given a single-layer raster, a segmentation, and a function, `extract_features` will return a numeric vector or a SpatRaster depending on whether the parameter `return_raster` is `TRUE` or `FALSE`. For the first case, each pixel of each segment will adopt the respective extracted feature value. For the second case, the return will be a vector of length equal to the total number of segments. Each value will be obtained by processing all pixels that belong to a segment with the provided function.

Value

If `return_raster` is set to `TRUE`, then an object of class SpatRaster with the same pixel dimensions than `r` will be returned. Otherwise, the return is a numeric vector of length equal to the number of segments found in `segmentation`.

See Also

Other Tool Functions: `colorfulness()`, `correct_vignetting()`, `defuzzify()`, `extract_dn()`, `extract_rl()`, `extract_sky_points_simple()`, `extract_sky_points()`, `extract_sun_coord()`, `find_sky_pixels_nonnull()`, `find_sky_pixels()`, `masking()`, `optim_normalize()`, `percentage_of_clipped_highli`
`read_bin()`, `read_caim_raw()`, `read_caim()`, `write_bin()`, `write_caim()`

Examples

```
r <- read_caim()
z <- zenith_image(ncol(r),lens())
a <- azimuth_image(z)
g <- sky_grid_segmentation(z, a, 10)
print(extract_feature(r$Blue, g, return_raster = FALSE))
# plot(extract_feature(r$Blue, g, return_raster = TRUE))
```

`extract_radiometry` *Extract radiometry data*

Description

Extract radiometry from images taken with the aid of a portable light source and the calibration board detailed in `calibrate_lens()`. The end goal is to obtain the data required to model the vignetting effect.

Usage

```
extract_radiometry(l, size_px = NULL)
```

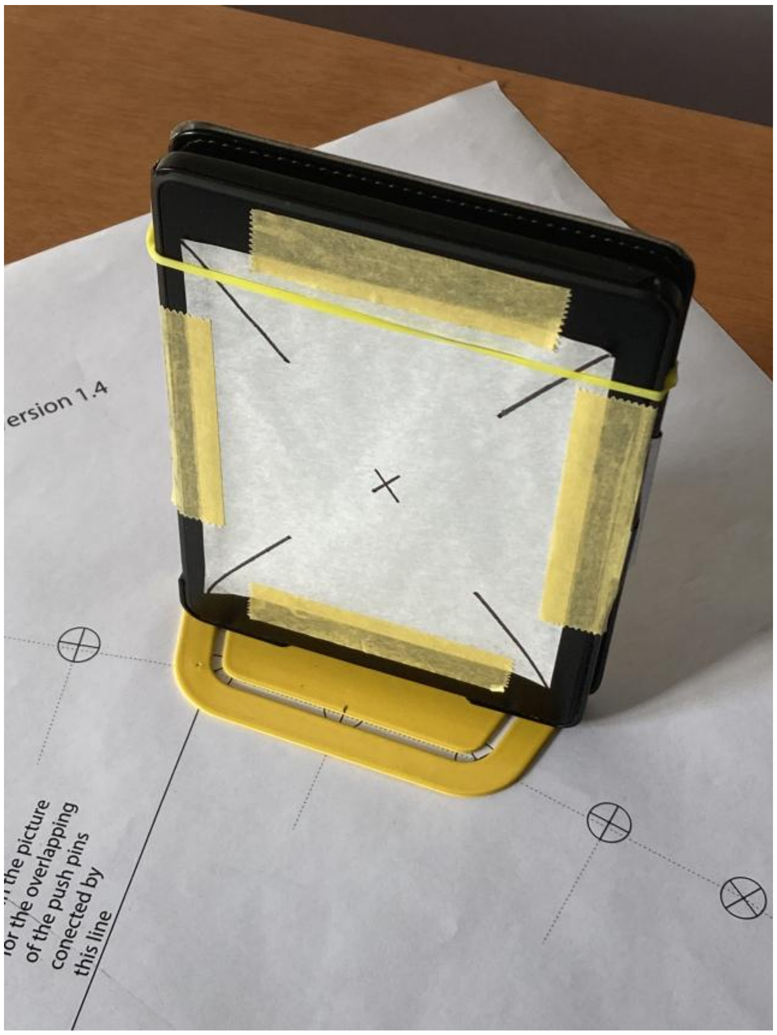
Arguments

`l`	List of prepossessed images (SpatRaster) for radiometry sampling. These images must comply with the equidistant projection.
`size_px`	Numeric vector of length one. Diameter in pixels of the circular sampling area at the image center. This area is modified considering the equidistant projection distortion. Therefore, it will be visualized as an ellipse at any other place but the image center.

Details

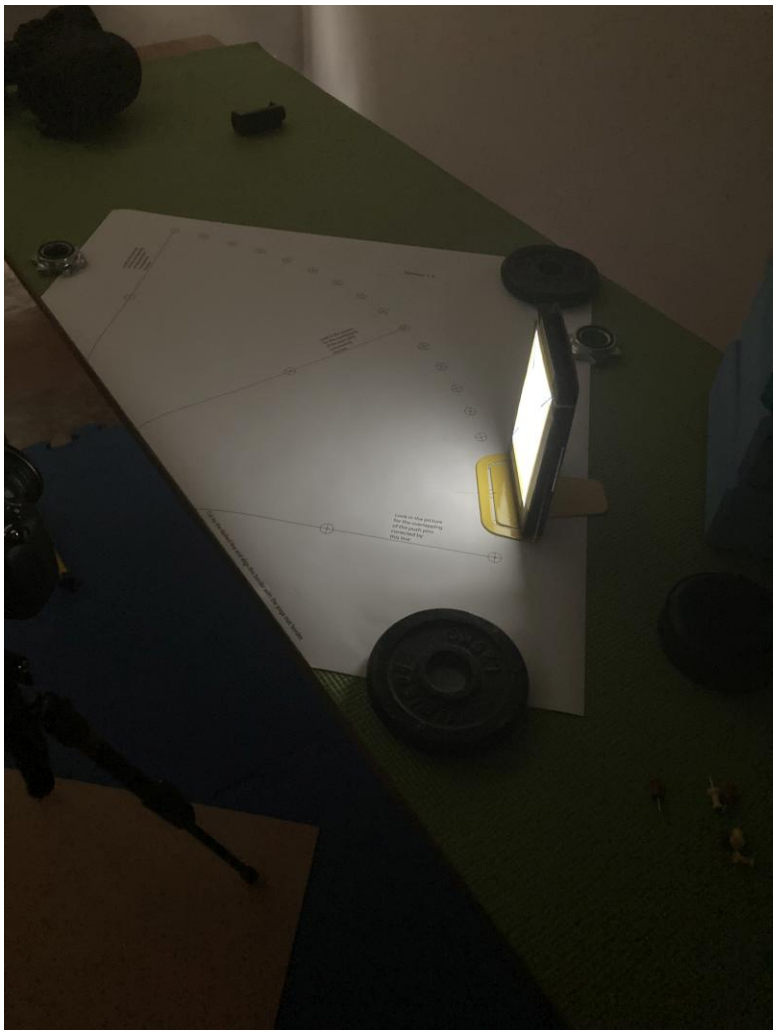
Lenses have the inconvenient property of increasingly attenuating light in the direction orthogonal to the optical axis. This phenomenon is known as the vignetting effect and varies with lens model and aperture setting. The method outlined here, known as the simple method, is explained in details in Díaz and et al. (2024). Next explanation might serve mostly as a quick recap of it.

The development of the simple method was done with a Kindle Paperwhite eBooks reader of 6" with built-in light. However, an iPhone 6 plus was also tested in the early stages of development and no substantial differences were observed. A metal bookends desk book holder was used to fasten the eBook reader upright and a semi-transparent paper to favor a Lambertian light distribution. In addition, the latter was used to draw on in order to guide pixel sampling. The book holder also facilitated the alignment of the screen with the dotted lines of the printed quarter-circle.



As a general guideline, a wide variety of mobile devices could be used as light sources, but if scattered data points are obtained with it, then other light sources should be tested in order to double check that the light quality is not the reason for points scattering.

With the room only illuminated by the portable light source, nine photographs should be taken with the light source located in the equivalent to 0, 10, 20, 30, 40, 50, 60, 70, and 80 degrees of zenith angle, respectively. Camera configuration should be in manual mode and set with the aperture (f/number) for which a vignetting function is required. The shutter speed should be regulated to obtain light-source pixels with middle grey values. The nine photographs should be taken **without changing the camera configuration or the light conditions**.



This code exemplify how to use the function to obtain the data and base R functions to obtain the vignetting function (f_v).

```
.read_raw <- function(path_to_raw_file) {
  r <- read_caim_raw(path_to_raw_file, z, a, zenith_colrow,
                     radius = 500, only_blue = TRUE)
  r
}

l <- Map(.read_raw, dir("raw/up/", full.names = TRUE))
up_data <- extract_radiometry(l)
l <- Map(.read_raw, dir("raw/down/", full.names = TRUE))
down_data <- extract_radiometry(l)
l <- Map(.read_raw, dir("raw/right/", full.names = TRUE))
right_data <- extract_radiometry(l)
l <- Map(.read_raw, dir("raw/left/", full.names = TRUE))
```

```
left_data <- extract_radiometry(l)

ds <- rbind(up_data, down_data, right_data, left_data)

plot(ds, xlim = c(0, pi/2), ylim= c(0.5,1.05),
      col = c(rep(1,9),rep(2,9),rep(3,9),rep(4,9)))
legend("bottomleft", legend = c("up", "down", "right", "left"),
       col = 1:4, pch = 1)

fit <- lm((1 - ds$radiometry) ~ poly(ds$theta, 3, raw = TRUE) - 1)
summary(fit)
coef <- -fit$coefficients #did you notice the minus sign?
.fv <- function(x) 1 + coef[1] * x + coef[2] * x^2 + coef[3] * x^3
curve(.fv, add = TRUE, col = 2)
coef
```

Once one of the aperture settings is calibrated, it can be used to calibrate all the rest. To do so, the equipment should be used to take photographs in all desired exposition and without moving the camera, including the aperture already calibrated and preferably under an open sky in stable diffuse light conditions. The same procedure, which minor adaptations, is applicable to cross-camera calibration. Below code could be used as a template.

```
zenith_colrow <- c(1500, 997)*2
diameter <- 947*4
z <- zenith_image(diameter, c(0.689, 0.0131, -0.0295))
a <- azimuth_image(z)

files <- dir("raw/", full.names = TRUE)
l <- list()
for (i in seq_along(files)) {
  if (i == 1) {
    # because the first aperture was the one already calibrated
    lens_coef_v <- c(0.0302, -0.320, 0.0908)
  } else {
    lens_coef_v <- NULL
  }
  l[[i]] <- read_caim_raw(files[i], z, a, zenith_colrow,
                          radius = 500,
                          only_blue = TRUE,
                          lens_coef_v = lens_coef_v)
}

ref <- l[[1]]
rings <- rings_segmentation(zenith_image(ncol(ref), lens()), 3)
theta <- seq(1.5, 90 - 1.5, 3) * pi/180

.fun <- function(r) {
  r <- extract_feature(r, rings, return_raster = FALSE)
  r/r[1]
}

l <- Map(.fun, l)

.fun <- function(x) {
  x / l[[1]][] # because the first is the one already calibrated
}
radiometry <- Map(.fun, l)

l <- list()
for (i in 2:length(radiometry)) {
  l[[i-1]] <- data.frame(theta = theta, radiometry = radiometry[[i]][])
}
ds <- l[[1]]
head(ds)
# The result is one dataset (ds) for each file. This is all what it is needed
# before using base R functions to fit a vignetting function
```

Value

An object from the class `data.frame` with columns *theta* (zenith angle in radians) and *radiometry* (digital number (DN) or relative digital number (RDN), depending on argument `z_thr`.

References

Díaz GM, et al. (2024). "Simple calibration of fisheye lenses for hemipherical photography of the forest canopy." *Manuscript in preparation*.

See Also

Other Lens Functions: `azimuth_image()`, `calc_diameter()`, `calc_relative_radius()`, `calc_zenith_colrow()`, `calibrate_lens()`, `crosscalibrate_lens()`, `expand_noncircular()`, `fisheye_to_equidistant()`, `fisheye_to_pano()`, `lens()`, `test_lens_coef()`, `zenith_image()`

`extract_rl` *Extract relative luminance*

Description

Extract the luminance relative to the zenith digital number

Usage

```
extract_rl(r, z, a, sky_points, no_of_points = 3, z_thr = 5, use_window = TRUE)
```

Arguments

`r`	SpatRaster. A normalized greyscale image. Typically, the blue channel extracted from a canopy photograph. Please see `read_caim()` and `normalize()`.
`z`	SpatRaster built with `zenith_image()`.
`a`	SpatRaster built with `azimuth_image()`.
`sky_points`	An object of class *data.frame*. The result of a call to `extract_sky_points()`. As an alternative, both ImageJ and HSP software package (Lang et al. 2013) can be used to manually digitize points. See `extract_dn()` and `read_manual_input()` for details.
`no_of_points`	Numeric vector of length one. The number of near-zenith points required for the estimation of the zenith DN.
`z_thr`	Numeric vector on length one. The starting maximum zenith angle used to search for near-zenith points.
`use_window`	Logical vector of length one. If `TRUE`, a 3×3 window will be used to extract the digital number from `r`.

Details

The search for near-zenith points starts in the region ranged between `0` and `z_thr`. If the number of near-zenith points is less than `no_of_points`, the region increases by steps of `2` degrees of zenith angle till the required number of points is reached.

Value

A list of three objects, *zenith_dn* and *max_zenith_angle* from the class *numeric*, and *sky_points* from the class *data.frame*; *zenith_dn* is the estimated zenith digital number, *max_zenith_angle* is the maximum zenith angle reached in the search for near-zenith sky points, and *sky_points* is the input argument `sky_points` with the additional columns: *a*, *z*, *dn*, and *rl*, which stand for azimuth and zenith angle in degrees, digital number, and relative luminance, respectively. If `NULL` is provided as `no_of_points`, then *zenith_dn* is forced to one and, therefore, *dn* and *rl* will be identical.

Note

The point selection tool of 'ImageJ' software can be used to manually digitize points and create a CSV file from which to read coordinates (see Examples). After digitizing the points on the image, use the dropdown menu Analyze>Measure to open the Results window. To obtain the CSV file, use File>Save As...

References

Lang M, Kodar A, Arumäe T (2013). "Restoration of above canopy reference hemispherical image from below canopy measurements for plant area index estimation in forests." *Forestry Studies*, **59**(1), 13–27. doi:10.2478/fsmu20130008.

See Also

Other Tool Functions: colorfulness(), correct_vignetting(), defuzzify(), extract_dn(), extract_feature(), extract_sky_points_simple(), extract_sky_points(), extract_sun_coord(), find_sky_pixels_nonnull(), find_sky_pixels(), masking(), optim_normalize(), percentage_of_clipped_highli
read_bin(), read_caim_raw(), read_caim(), write_bin(), write_caim()

Examples

```
## Not run:
caim <- read_caim() %>% normalize(., 0, 20847)
z <- zenith_image(ncol(caim), lens())
a <- azimuth_image(z)
plotRGB(caim*255)

path <- system.file("external/sky_points.csv",
                    package = "rcaiman")
sky_points <- read.csv(path)
sky_points <- sky_points[c("Y", "X")]
colnames(sky_points) <- c("row", "col")
head(sky_points)
points(sky_points$col, nrow(caim) - sky_points$row, col = 2, pch = 10)
extract_rl(caim$Blue, z, a, sky_points, 1)

## End(Not run)
```

---

extract_sky_points   *Extract sky points*

Description

Extract sky points for model fitting

Usage

```
extract_sky_points(r, bin, g, dist_to_plant = 3, min_raster_dist = 3)
```

Arguments

`r`	SpatRaster. A normalized greyscale image. Typically, the blue channel extracted from a canopy photograph. Please see `read_caim()` and `normalize()`.
`bin`	SpatRaster. This should be a preliminary binarization of `r` useful for masking pixels that are very likely pure sky pixels.
`g`	SpatRaster built with `sky_grid_segmentation()` or `chessboard()`.
`dist_to_plant, min_raster_dist`	Numeric vector of length one or `NULL`.

Details

This function will automatically sample sky pixels from the sky regions delimited by bin. The density and distribution of the sampling points is controlled by the arguments g, dist_to_plant, and min_raster_dist.

As the first step, sky pixels from r are evaluated to find the pixel with maximum digital value (local maximum) per cell of the g argument. The dist_to_plant argument allows users to establish a buffer zone for bin, meaning a size reduction of the original sky regions.

The final step is filtering these local maximum values by evaluating the Euclidean distances between them on the raster space. Any new point with a distance from existing points minor than min_raster_dist is discarded. Cell labels determine the order in which the points are evaluated.

To skip a given filtering step, use code NULL as argument input. For instance, min_raster_dist = NULL will return points omitting the final step.

Value

An object of the class *data.frame* with two columns named *row* and *col*.

See Also

fit_cie_sky_model()

Other Tool Functions: colorfulness(), correct_vignetting(), defuzzify(), extract_dn(), extract_feature(), extract_rl(), extract_sky_points_simple(), extract_sun_coord(), find_sky_pixels_nonnull(), find_sky_pixels(), masking(), optim_normalize(), percentage_of_clipped_highli
read_bin(), read_caim_raw(), read_caim(), write_bin(), write_caim()

Examples

```
## Not run:
caim <- read_caim()
r <- caim$Blue
caim <- normalize(caim, 0, 20847, TRUE)
z <- zenith_image(ncol(caim), lens())
a <- azimuth_image(z)
plotRGB(caim*255)
bin <- ootb_obia(caim, z, a, HSV(239, 0.85, 0.5), gamma = NULL)
g <- sky_grid_segmentation(z, a, 10)
sky_points <- extract_sky_points(r, bin, g,
                                 dist_to_plant = 3,
                                 min_raster_dist = 10)
plot(bin)
points(sky_points$col, nrow(caim) - sky_points$row, col = 2, pch = 10)

## End(Not run)
```

`extract_sky_points_simple` *Extract sky points*

Description

Extract sky points for model fitting

Usage

```
extract_sky_points_simple(r, z, a)
```

Arguments

`r`	SpatRaster. A normalized greyscale image. Typically, the blue channel extracted from a canopy photograph. Please see `read_caim()` and `normalize()`.
`z`	SpatRaster built with `zenith_image()`.
`a`	SpatRaster built with `azimuth_image()`.

Details

This function will automatically sample sky pixels following this simple strategy:

- mask out the region of `r` above 15 and below 75 degrees of zenith angle,
- divide the hemisphere into sectors of 15 degrees each (see `sectors_segmentation()`),
- search for the maximum digital value in each sector (n = 24),
- divide the hemisphere into rings of 5 degrees each (see `rings_segmentation()`,
- search for the maximum digital value in each ring (n = 12)
- combine these local maxima (n = 36).

Value

An object of the class *data.frame* with two columns named *col* and *row*.

See Also

Other Tool Functions: `colorfulness()`, `correct_vignetting()`, `defuzzify()`, `extract_dn()`, `extract_feature()`, `extract_rl()`, `extract_sky_points()`, `extract_sun_coord()`, `find_sky_pixels_nonnull()`, `find_sky_pixels()`, `masking()`, `optim_normalize()`, `percentage_of_clipped_highlights()`, `read_bin()`, `read_caim_raw()`, `read_caim()`, `write_bin()`, `write_caim()`

Examples

```
## Not run:
path <- system.file("external/DSCN4500.JPG", package = "rcaiman")
caim <- read_caim(path, c(1250, 1020) - 745, 745 * 2, 745 * 2)
z <- zenith_image(ncol(caim), lens("Nikon_FCE9"))
a <- azimuth_image(z)
r <- gbc(caim$Blue)
sky_points <- extract_sky_points_simple(r, z, a)
plot(r)
points(sky_points$col, nrow(caim) - sky_points$row, col = 2, pch = 10)

## End(Not run)
```

`extract_sun_coord` *Extract sun coordinates*

Description

Extract the sun coordinates for CIE sky model fitting.

Usage

```
extract_sun_coord(r, z, a, bin, g, max_angular_dist = 30)
```

Arguments

`r`	SpatRaster. A normalized greyscale image. Typically, the blue channel extracted from a canopy photograph. Please see `read_caim()` and `normalize()`.
`z`	SpatRaster built with `zenith_image()`.
`a`	SpatRaster built with `azimuth_image()`.
`bin`	SpatRaster. This should be a preliminary binarization of `r` useful for masking pixels that are very likely pure sky pixels.
`g`	SpatRaster built with `sky_grid_segmentation()` or `chessboard()`.
`max_angular_dist`	Numeric vector of length one. Angle in degrees to control the potential maximum size of the solar corona.

Details

This function uses an object-based image analyze framework. The segmentation is given by `g` and `bin`. For every cell of `g`, the pixels from `r` that are equal to one on `bin` are selected and its maximum value is calculated. Then, the 95th percentile of these maximum values is computed and used to filter out cells below that threshold; i.e, only the cells with at least one extremely bright sky pixel is selected.

The selected cells are grouped based on adjacency, and new bigger segments are created from these groups. The degree of membership to the class *Sun* is calculated for every new segment by computing the number of cells that constitute the segment and its mean digital number (values taken from `r`). In other words, the largest and brightest segments are the ones that score higher. The one with the highest score is selected as the *sun seed*.

The angular distance from the sun seed to every other segments are computed, and only the segments not farther than `max_angular_dist` are classified as part of the sun corona. A multi-part segment is created by merging the sun-corona segments and, finally, the center of its bounding box is considered as the sun location.

Value

Object of class *list* with two numeric vectors of length two named *row_col* and *zenith_azimuth*. The former is the raster coordinates of the solar disk (row and column), and the other is the angular coordinates (zenith and azimuth angles in degrees).

See Also

Other Tool Functions: `colorfulness()`, `correct_vignetting()`, `defuzzify()`, `extract_dn()`, `extract_feature()`, `extract_rl()`, `extract_sky_points_simple()`, `extract_sky_points()`, `find_sky_pixels_nonnull()`, `find_sky_pixels()`, `masking()`, `optim_normalize()`, `percentage_of_clipped_highli`
`read_bin()`, `read_caim_raw()`, `read_caim()`, `write_bin()`, `write_caim()`

Examples

```
## Not run:
caim <- read_caim()
r <- caim$Blue
caim <- normalize(caim, 0, 20847, TRUE)
z <- zenith_image(ncol(caim), lens())
a <- azimuth_image(z)
m <- !is.na(z)
plotRGB(caim*255)
bin <- ootb_obia(caim, z, a, m, HSV(239, 0.85, 0.5), gamma = NULL)
g <- sky_grid_segmentation(z, a, 10)
sun_coord <- extract_sun_coord(r, z, a, bin, g, max_angular_dist = 30)
points(sun_coord$row_col[2], nrow(caim) - sun_coord$row_col[1],
        col = 3, pch = 10)

## End(Not run)
```

`find_sky_pixels` *Find sky pixels*

Description

Find sky pixels automatically.

Usage

```
find_sky_pixels(r, z, a, sample_size_pct = 30)
```

Arguments

`r`	SpatRaster. A normalized greyscale image. Typically, the blue channel extracted from a canopy photograph. Please see `read_caim()` and `normalize()`.
`z`	SpatRaster built with `zenith_image()`.
`a`	SpatRaster built with `azimuth_image()`.
`sample_size_pct`	Numeric vector of length one. Minimum percentage of cells to sample. The population is comprised of 1296 cells of 5×5 degrees.

Details

This function assumes that:

- there is at least one pure sky pixel at the level of cells of 30×30 degrees, and
- sky pixels have a digital number (DN) greater than canopy pixels have.

For each 30×30 cell, this method computes a quantile value and uses it as a threshold to select the pure sky pixels from the given cell. As a result, a binarized image is produced in a regional binarization fashion (`regional_thresholding()`). This process starts with a quantile probability of 0.99. After producing the binarized image, this function uses a search grid with cells of 5×5 degrees to count how many of these cells have at least one sky pixel (pixels equal to one in the binarized image). If the percentage of cells with sky pixels does not reach argument `sample_size_pct`, it goes back to the binarization step but decreasing the probability by 0.01 points.

If probability reach 0.9 and the `sample_size_pct` criterion were not yet satisfied, the `sample_size_pct` is decreased one percent and the process starts all over again.

Value

An object of class SpatRaster with values `0` and `1`. This layer masks pixels that are very likely pure sky pixels.

See Also

Other Tool Functions: `colorfulness()`, `correct_vignetting()`, `defuzzify()`, `extract_dn()`, `extract_feature()`, `extract_rl()`, `extract_sky_points_simple()`, `extract_sky_points()`, `extract_sun_coord()`, `find_sky_pixels_nonnull()`, `masking()`, `optim_normalize()`, `percentage_of_clipped_high`
`read_bin()`, `read_caim_raw()`, `read_caim()`, `write_bin()`, `write_caim()`

Examples

```
## Not run:
caim <- read_caim() %>% normalize(., 0, 20847)
z <- zenith_image(ncol(caim), lens())
a <- azimuth_image(z)
r <- caim$Blue
```

```
r[is.na(r)] <- 0
bin <- find_sky_pixels(r, z, a)
plot(bin)

## End(Not run)
```

---

find_sky_pixels_nonnull

Find sky pixels following the non-null criteria

Description

Cells without sky pixels are the so-called null cells. This type of cells are mathematically intractable by models typically used to obtain canopy metrics. This function find sky pixels using increase in number of null cells as the stopping criteria.

Usage

```
find_sky_pixels_nonnull(r, sky, g, intercept = 0, slope = 1, w = 0.5)
```

Arguments

`r`	SpatRaster. A normalized greyscale image. Typically, the blue channel extracted from a canopy photograph. Please see `read_caim()` and `normalize()`.
`sky`	An object of class SpatRaster produced with `fit_coneshaped_model()`, `fit_trend_surface()`, `fit_cie_sky_model()`, or `ootb_sky_reconstruction()`. It also support a numeric vector of length one. For instance, it could be a value obtained with a combination of `extract_sky_points()` and `extract_dn()`. The latter can be understood as modelling the sky with a plane.
`g`	SpatRaster built with `sky_grid_segmentation()` or `chessboard()`.
`intercept, slope`	Numeric vector of length one. These are linear function coefficients.
`w`	Numeric vector of length one. Weighting parameter from Díaz and Lencinas (2018)'s Equation 1. See `thr_mblt()`

Details

The arguments `sky`, `intercept`, `slope`, and `w` are passed to `thr_mblt()` whose output is in turn passed to `apply_thr()` along with `r`. As a result, `r` is binarized and used along with `g` to compute the number of null cells. The process is repeated but increasing `w` in steps of 0.05 as long as the number of null cells remains constant.

Value

An object of class SpatRaster with values `0` and `1`.

See Also

Other Tool Functions: colorfulness(), correct_vignetting(), defuzzify(), extract_dn(), extract_feature(), extract_rl(), extract_sky_points_simple(), extract_sky_points(), extract_sun_coord(), find_sky_pixels(), masking(), optim_normalize(), percentage_of_clipped_highlights(
read_bin(), read_caim_raw(), read_caim(), write_bin(), write_caim()

Examples

```
## Not run:
caim <- read_caim()
r <- caim$Blue %>% normalize()
caim <- normalize(caim, 0, 20847, TRUE)
z <- zenith_image(ncol(caim), lens())
a <- azimuth_image(z)
m <- !is.na(z)
bin <- ootb_obia(caim, z, a, m, HSV(239, 0.85, 0.5), gamma = NULL)
g <- sky_grid_segmentation(z, a, 3)
sky_points <- extract_sky_points(r, bin, g,
                                 dist_to_plant = 5,
                                 min_raster_dist = 5)
rl <- extract_rl(r, z, a, sky_points)
model <- fit_coneshaped_model(rl$sky_points)
summary(model$model)

sky <- model$fun(z, a)
sky <- fit_trend_surface(sky, z, a, !is.na(z))$image
plot(r/sky)

x <- predict(model$model)
y <- predict(model$model) + model$model$residuals
mblt <- coefficients(lm(x~y))

g <- sky_grid_segmentation(z, a, 10)
bin <- find_sky_pixels_nonnull(r, sky, g, mblt[1], mblt[2], w = 0.1)
plot(bin)

## End(Not run)
```

---

fisheye_to_equidistant
Fisheye to equidistant

Description

Fisheye to equidistant projection (also known as polar projection).

Usage

```
fisheye_to_equidistant(
  r,
  z,
  a,
  m = NULL,
  radius = NULL,
  k = NULL,
  p = NULL,
  rmax = 100
)
```

Arguments

`r`	SpatRaster. A fish-eye image.
`z`	SpatRaster built with `zenith_image()`.
`a`	SpatRaster built with `azimuth_image()`.
`m`	SpatRaster. A mask. For hemispherical photographs, check `mask_hs()`.
`radius`	Numeric integer of length one. Radius of the reprojected hemispherical image (i.e., the output).
`k`	Numeric vector of length one. Number of k-nearest neighbors.
`p`	Numeric vector of length one. Power for inverse-distance weighting.
`rmax`	Numeric vector of length one. Maximum radius where to search for *knn*. Increase this value if pixels with value `0` or `FALSE` appears where other values are expected.

Details

The pixel values and their image coordinates are treated as points to be reprojected and interpolated. To that end, this function use `lidR::knnidw()` as workhorse function, so arguments `k`, `p`, and `rmax` are passed to it. If the user does not input values to these arguments, both `k` and `p` are automatically defined by default as follow: when a binarized image is provided as argument `r`, both parameters are set to `1`; otherwise, they are set to `9` and `2`, respectively.

Note

Default value for the `radius` argument is equivalent to input the radius of the `r` argument.

See Also

Other Lens Functions: `azimuth_image()`, `calc_diameter()`, `calc_relative_radius()`, `calc_zenith_colrow()`, `calibrate_lens()`, `crosscalibrate_lens()`, `expand_noncircular()`, `extract_radiometry()`, `fisheye_to_pano()`, `lens()`, `test_lens_coef()`, `zenith_image()`

Examples

```
## Not run:
path <- system.file("external/DSCN4500.JPG", package = "rcaiman")
caim <- read_caim(path, c(1250, 1020) - 745, 745 * 2, 745 * 2)
z <- zenith_image(ncol(caim), lens("Nikon_FCE9"))
a <- azimuth_image(z)
r <- gbc(caim$Blue)
r <- correct_vignetting(r, z, c(0.0638, -0.101)) %>% normalize()
bin <- ootb_mblt(r, z, a)$bin
bin_equi <- fisheye_to_equidistant(bin, z, a)
plot(bin)
plot(bin_equi)
# Use write_bin(bin, "path/file_name") to have a file ready
# to calcute LAI with CIMES, GLA, CAN-EYE, etc.

# It can be used to reproject RGB photographs
plotRGB(caim)
caim <- fisheye_to_equidistant(caim, z, a)
plotRGB(caim)

## End(Not run)
```

`fisheye_to_pano` *Fisheye to panoramic*

Description

Fisheye to panoramic (cylindrical projection)

Usage

```
fisheye_to_pano(r, z, a, fun = mean, angle_width = 1)
```

Arguments

`r`	SpatRaster. A fish-eye image.
`z`	SpatRaster built with `zenith_image()`.
`a`	SpatRaster built with `azimuth_image()`.
`fun`	A `function` that takes a vector as input and returns a one-length numeric or logical vector as output (e.g. mean).
`angle_width`	Numeric vector of length one. It should be `30, 15, 10, 7.5, 6, 5, 3.75, 3, 2.5, 1.875, 1` or `0.5` degrees. This constrain is rooted in the requirement of a value able to divide both the `0` to `360` and `0` to `90` ranges into a whole number of segments.

Details

An early version of this function was used in Díaz et al. (2021).

References

Díaz GM, Negri PA, Lencinas JD (2021). "Toward making canopy hemispherical photography independent of illumination conditions: A deep-learning-based approach." *Agricultural and Forest Meteorology*, **296**, 108234. doi:10.1016/j.agrformet.2020.108234.

See Also

Other Lens Functions: `azimuth_image()`, `calc_diameter()`, `calc_relative_radius()`, `calc_zenith_colrow()`, `calibrate_lens()`, `crosscalibrate_lens()`, `expand_noncircular()`, `extract_radiometry()`, `fisheye_to_equidistant()`, `lens()`, `test_lens_coef()`, `zenith_image()`

Examples

```
## Not run:
caim <- read_caim()
z <- zenith_image(ncol(caim), lens())
a <- azimuth_image(z)
pano <- fisheye_to_pano(caim, z, a)
plotRGB(pano %>% normalize() %>% multiply_by(255))

## End(Not run)
```

`fit_cie_sky_model` *Fit CIE sky model*

Description

Use maximum likelihood to estimate the coefficients of the CIE sky model that best fit to data sampled from a canopy photograph.

Usage

```
fit_cie_sky_model(
  r,
  z,
  a,
  sky_points,
  zenith_dn,
  sun_coord,
  custom_sky_coef = NULL,
  std_sky_no = NULL,
  general_sky_type = NULL,
  twilight = TRUE,
  rmse = FALSE,
  method = "BFGS"
)
```

Arguments

`r`	SpatRaster. A normalized greyscale image. Typically, the blue channel extracted from a canopy photograph. Please see `read_caim()` and `normalize()`.
`z`	SpatRaster built with `zenith_image()`.
`a`	SpatRaster built with `azimuth_image()`.
`sky_points`	The *data.frame* returned by `extract_rl()` or a *data.frame* with same structure and names.
`zenith_dn`	Numeric vector of length one. Zenith digital number, see `extract_rl()` for how to obtain it.
`sun_coord`	An object of class *list*. The result of a call to `extract_sun_coord()` or an object with same structure and names. See also `row_col_from_zenith_azimuth()` in case you want to provide values based on date and time of acquisition and the `suncalc` package.
`custom_sky_coef`	Numeric vector of length five. Custom starting coefficients of the sky model. By default, they are drawn from standard skies.
`std_sky_no`	Numeric vector. Standard sky number from Li et al. (2016)'s Table 1.
`general_sky_type`	Character vector of length one. It could be any of these: "Overcast", "Clear", or "Partly cloudy". See Table 1 from Li et al. (2016) for additional details.
`twilight`	Logical vector of length one. If it is `TRUE` and the initial standard parameters belong to the "Clear" general sky type, sun zenith angles from 90 to 96 degrees will be tested (civic twilight). This is necessary since `extract_sun_coord()` would mistakenly recognize the center of what can be seen of the solar corona as the solar disk.
`rmse`	Logical vector of length one. If it is `TRUE`, the criteria for selecting the best sky model is to choose the one with the lowest **root mean square error (RMSE)** calculated by using the `sky_points` argument as the source of reference values. Otherwise, the criteria is to evaluate the whole image by calculating the **out-of-range index** as $\sum_{i=1}^{N}(r_i/sky_i)^2$, where r is the `r` argument, sky is the raster obtained from the fitted model with `cie_sky_model_raster()` and `zenith_dn`, i is the index that represents the position of a given pixel on the raster grid, and N is the total number of pixels that satisfy either of these inequalities: $r_i/sky_i < 0$ and $r_i/sky_i > 1$.
`method`	Optimization method to use. See `optim`.

Details

This function is based on Lang et al. (2010). In theory, the best result would be obtained with data showing a linear relation between digital numbers and the amount of light reaching the sensor. See `extract_radiometry()` and `read_caim_raw()` for further details. As a compromise solution, `gbc()` can be used.

The following code exemplifies how this package can be used to compare the manually-guided fitting provided by HSP (Lang et al. 2013) against the automatic fitting provided by this package. The code assumes that the user is working within an RStudio project located in the HSP project folder.

```
r <- read_caim("manipulate/IMG_1013.pgm") %>% normalize()
z <- zenith_image(ncol(r), lens())
a <- azimuth_image(z)
manual_input <- read_manual_input(".", "IMG_1013" )
sun_coord <- manual_input$sun_coord$row_col
sun_coord <- zenith_azimuth_from_row_col(z, sun_coord, lens())
sky_points <- manual_input$sky_points
rl <- extract_rl(r, z, a, sky_points)
model <- fit_cie_sky_model(r, z, a, rl$sky_points, rl$zenith_dn, sun_coord)
cie_sky <- model$relative_luminance * model$zenith_dn
plot(r/cie_sky)

r <- read_caim("manipulate/IMG_1013.pgm")
sky_coef <- read_opt_sky_coef(".", "IMG_1013")
cie_sky_manual <- cie_sky_model_raster(z, a, sun_coord$zenith_azimuth, sky_coef)
cie_sky_manual <- cie_sky_manual * manual_input$zenith_dn
plot(r/cie_sky_manual)
```

Value

object from the class *list*. The result includes the following: (1) the output produced by `bbmle::mle2()`, (2) the 5 coefficients, (3 and 4) observed and predicted values, (5) the digital number at the zenith, (6) the sun coordinates –zenith and azimuth angle in degrees–, and (7) the description of the standard sky from which the initial coefficients were drawn. See Li et al. (2016) to know more about these coefficients.

Note

If you use this function in your research, please cite Lang et al. (2010) in addition to this package (`citation("rcaiman")`).

References

Lang M, Kodar A, Arumäe T (2013). "Restoration of above canopy reference hemispherical image from below canopy measurements for plant area index estimation in forests." *Forestry Studies*, **59**(1), 13–27. doi:10.2478/fsmu20130008.

Lang M, Kuusk A, M~ottus M, Rautiainen M, Nilson T (2010). "Canopy gap fraction estimation from digital hemispherical images using sky radiance models and a linear conversion method." *Agricultural and Forest Meteorology*, **150**(1), 20–29. doi:10.1016/j.agrformet.2009.08.001.

Li DH, Lou S, Lam JC, Wu RH (2016). "Determining solar irradiance on inclined planes from classified CIE (International Commission on Illumination) standard skies." *Energy*, **101**, 462–470. doi:10.1016/j.energy.2016.02.054.

See Also

Other Sky Reconstruction Functions: `cie_sky_model_raster()`, `fit_coneshaped_model()`, `fit_trend_surface()`, `fix_reconstructed_sky()`, `interpolate_sky_points()`, `ootb_sky_reconstruction()`

Examples

```
## Not run:
caim <- read_caim() %>% normalize()
z <- zenith_image(ncol(caim), lens())
a <- azimuth_image(z)

# Manual method after Lang et al. (2010)
# ImageJ can be used to digitize points
path <- system.file("external/sky_points.csv",
                    package = "rcaiman")
sky_points <- read.csv(path)
sky_points <- sky_points[c("Y", "X")]
colnames(sky_points) <- c("row", "col")
head(sky_points)
plot(caim$Blue)
points(sky_points$col, nrow(caim) - sky_points$row, col = 2, pch = 10)

xy <- c(210, 451) #originally captured with click() after x11()
sun_coord <- zenith_azimuth_from_row_col(z, z, c(nrow(z) - xy[2],xy[1]))
points(sun_coord$row_col[2], nrow(caim) - sun_coord$row_col[1],
       col = 3, pch = 1)

rl <- extract_rl(caim$Blue, z, a, sky_points)

set.seed(7)
model <- fit_cie_sky_model(caim$Blue, z, a, rl$sky_points,
                           rl$zenith_dn, sun_coord,
                           general_sky_type = "Clear",
                           rmse = FALSE,
                           twilight = FALSE,
                           method = "SANN")
summary(model$mle2_output)
plot(model$obs, model$pred)
abline(0,1)
r2 <- lm(model$pred~model$obs) %>% summary(.) %>% .$r.squared
r2
sky_cie <- cie_sky_model_raster(z, a,
                                model$sun_coord$zenith_azimuth,
                                model$coef) * model$zenith_dn
plot(sky_cie)
plot(caim$Blue/sky_cie)

# A quick demonstration of how to use interpolation to improve sky modelling
# after Lang et al. (2010)
sky <- interpolate_sky_points(rl$sky_points, caim$Blue, rmax = ncol(caim)/7)
plot(sky)
sky <- sky * rl$zenith_dn * (1 - r2) + sky_cie * r2
sky <- terra::cover(sky, sky_cie)
plot(sky)
plot(caim$Blue/sky)

# how to provide a custom starting coefficient
```

```
model <- fit_cie_sky_model(caim$Blue, z, a, rl$sky_points,
                           rl$zenith_dn, sun_coord,
                           custom_sky_coef = model$coef,
                           method = "SANN")
plot(model$obs, model$pred, ylim = range(model$obs))
abline(0,1)

## End(Not run)
```

`fit_coneshaped_model` *Fit cone-shaped model*

Description

Statistical modeling for predicting digital numbers from spherical coordinates.

Usage

```
fit_coneshaped_model(sky_points, use_azimuth_angle = TRUE)
```

Arguments

`sky_points`	The *data.frame* returned by `extract_rl()` or a *data.frame* with same structure and names.
`use_azimuth_angle`	Logical vector of length one. If `TRUE`, the Equation 4 from Díaz and Lencinas (2018)) is used: $sDN = a + b \cdot \theta + c \cdot \theta^2 + d \cdot sin(\phi) + e \cdot cos(\phi)$, where sDN is sky digital number, a, b, c, d and e are coefficients, θ is zenith angle, and ϕ is azimuth angle. If `FALSE`, the next simplified version based on Wagner (2001) is used: $sDN = a + b \cdot \theta + c \cdot \theta^2$.

Details

This method was presented in Díaz and Lencinas (2018), under the heading *Estimation of the sky DN as a previous step for our method*. If you use this function in your research, please cite that paper in addition to this package (`citation("rcaiman"`).

Value

A list of two objects, one of class `function` and the other of class `lm` (see `stats::lm()`). If the fitting fails, it returns `NULL`. The function requires two arguments–zenith and azimuth in degrees–to return relative luminance.

References

Díaz GM, Lencinas JD (2018). "Model-based local thresholding for canopy hemispherical photography." *Canadian Journal of Forest Research*, **48**(10), 1204–1216. doi:10.1139/cjfr20180006.

Wagner S (2001). "Relative radiance measurements and zenith angle dependent segmentation in hemispherical photography." *Agricultural and Forest Meteorology*, **107**(2), 103–115. doi:10.1016/s01681923(00)00232x.

See Also

`thr_mblt()`

Other Sky Reconstruction Functions: `cie_sky_model_raster()`, `fit_cie_sky_model()`, `fit_trend_surface()`, `fix_reconstructed_sky()`, `interpolate_sky_points()`, `ootb_sky_reconstruction()`

Examples

```
## Not run:
path <- system.file("external/DSCN4500.JPG", package = "rcaiman")
caim <- read_caim(path, c(1250, 1020) - 745, 745 * 2, 745 * 2)
z <- zenith_image(ncol(caim), lens("Nikon_FCE9"))
a <- azimuth_image(z)
r <- gbc(caim$Blue)

sky_points <- extract_rl(r, z, a, extract_sky_points_simple(r, z, a),
                         NULL,
                         use_window = FALSE)#this is important when
                                             #extract_sky_points_simple()
                                             #is used
model <- fit_coneshaped_model(sky_points$sky_points)
summary(model$model)
sky_cs <- model$fun(z, a)
plot(r/sky_cs)
plot(sky_cs)

z <- zenith_image(50, lens())
a <- azimuth_image(z)
sky_cs <- model$fun(z, a)
persp(sky_cs, theta = 90, phi = 20)

## End(Not run)
```

`fit_trend_surface` *Fit a trend surface to sky digital numbers*

Description

Fit a trend surface using `spatial::surf.ls()` as workhorse function.

Usage

```
fit_trend_surface(r, z, a, bin, filling_source = NULL, np = 6)
```

Arguments

`r`	SpatRaster. A normalized greyscale image. Typically, the blue channel extracted from a canopy photograph. Please see `read_caim()` and `normalize()`.
`z`	SpatRaster built with `zenith_image()`.
`a`	SpatRaster built with `azimuth_image()`.
`bin`	SpatRaster. This should be a preliminary binarization of `r` useful for masking pixels that are very likely pure sky pixels.
`filling_source`	SpatRaster. An actual or reconstructed above-canopy image to complement the sky pixels detected through the gaps of `r`. A photograph taken immediately after or before taking `r` under the open sky with the same equipment and configuration is a very good option but not recommended under fleeting clouds. The orientation relative to the North must be the same as for `r`. If it is set to `NULL` (default), only sky pixels from `r` will be used as input.
`np`	degree of polynomial surface

Details

This function is meant to be used after `fit_coneshaped_model()`.

This method was presented in Díaz and Lencinas (2018), under the heading *Estimation of the sky DN as a previous step for our method*. If you use this function in your research, please cite that paper in addition to this package (`citation("rcaiman")`).

Value

A list with an object of class SpatRaster and of class `trls` (see `spatial::surf.ls()`).

Note

If an incomplete above-canopy image is available as filling source, non-sky pixels should be turned `NA` or they will be erroneously considered as sky pixels.

References

Díaz GM, Lencinas JD (2018). "Model-based local thresholding for canopy hemispherical photography." *Canadian Journal of Forest Research*, **48**(10), 1204–1216. doi:10.1139/cjfr20180006.

See Also

`thr_mblt()`

Other Sky Reconstruction Functions: `cie_sky_model_raster()`, `fit_cie_sky_model()`, `fit_coneshaped_model()`, `fix_reconstructed_sky()`, `interpolate_sky_points()`, `ootb_sky_reconstruction()`

Examples

```
## Not run:
caim <- read_caim()
r <- caim$Blue
caim <- normalize(caim, 0, 20847, TRUE)
z <- zenith_image(ncol(caim), lens())
a <- azimuth_image(z)
m <- !is.na(z)

bin <- ootb_obia(caim, z, a, m, HSV(239, 0.85, 0.5), gamma = NULL)

g <- sky_grid_segmentation(z, a, 10)
sky_points <- extract_sky_points(r, bin, g, dist_to_plant = 5)
plot(bin)
points(sky_points$col, nrow(caim) - sky_points$row, col = 2, pch = 10)
rl <- extract_rl(r, z, a, sky_points)

model <- fit_coneshaped_model(rl$sky_points)
summary(model$model)
sky_cs <- model$fun(z, a)
persp(terra::aggregate(sky_cs, 10), theta = 90, phi = 45)

sky_s <- fit_trend_surface(r, z, a, bin, sky_cs)
persp(terra::aggregate(sky_s$image, 10), theta = 90, phi = 45)

## End(Not run)
```

`fix_reconstructed_sky` *Fix reconstructed sky*

Description

Automatically edit a raster image of sky digital numbers (DNs) reconstructed with functions such as `fit_coneshaped_model()` and `fit_trend_surface()`.

Usage

```
fix_reconstructed_sky(sky, z, r, bin)
```

Arguments

`sky`	SpatRaster. Sky DNs predicted with functions such as `fit_coneshaped_model()` and `fit_trend_surface()`.
`z`	SpatRaster built with `zenith_image()`.
`r`	SpatRaster. The source of the sky DNs used to build sky (the data source).
`bin`	SpatRaster. The binarization of `r` used to select the sky DNs for building the sky argument.

Details

The predicted sky DNs are usually erroneous near the horizon because either they are a misleading extrapolation or are based on corrupted data (non-pure sky DNs).

The proposed automatic edition consists of:

- flattening the values below the minimum value from the data source defined by `r` and `bin`and
- forcing the values toward the horizon to become gradually the median value from the data source.

The latter is achieved by calculating the weighted average of the median value and the predicted sky DNs, using the ratio of `z` to `90` to determine the weights.

Value

An object of class SpatRaster. The argument `sky` with dimensions unchanged but values edited.

See Also

Other Sky Reconstruction Functions: `cie_sky_model_raster()`, `fit_cie_sky_model()`, `fit_coneshaped_model()`, `fit_trend_surface()`, `interpolate_sky_points()`, `ootb_sky_reconstruction()`

Examples

```
## Not run:
caim <- read_caim()
r <- caim$Blue
caim <- normalize(caim, 0, 20847, TRUE)
z <- zenith_image(ncol(caim), lens())
a <- azimuth_image(z)
bin <- find_sky_pixels(r, z, a)
sky <- fit_trend_surface(r, z, a, bin)$image
sky <- fix_reconstructed_sky(sky, z, r, bin)
plot(sky)

## End(Not run)
```

`gbc` *Gamma back correction*

Description

Gamma back correction of JPEG images

Usage

```
gbc(DN_from_JPEG, gamma = 2.2)
```

Arguments

`DN_from_JPEG`	Numeric vector or object of the SpatRaster class. Digital numbers from a JPEG file (0 to 255, i.e., the standard 8-bit encoded).
`gamma`	Numeric vector of length one. Gamma value. Please see Díaz and Lencinas (2018) for details.

Details

Digital cameras usually use sRGB as color space. It is a standard developed to ensure accurate color and tone management. The transfer function of sRGB, known as gamma correction, is very close to a power function with the exponent 1/2.2. This is why a DN of a born-digital photograph that was encoded in sRGB has a non-linear relationship with luminance despite having the sensor a linear response.

Value

The same class as `DN_from_JPEG`, with dimension unchanged but values rescaled between `0` and `1` in a non-linear fashion.

References

Díaz GM, Lencinas JD (2018). "Model-based local thresholding for canopy hemispherical photography." *Canadian Journal of Forest Research*, **48**(10), 1204–1216. doi:10.1139/cjfr20180006.

See Also

Other Pre-processing Functions: `enhance_caim()`, `local_fuzzy_thresholding()`, `membership_to_color()`, `normalize()`

Examples

```
path <- system.file("external/DSCN4500.JPG", package = "rcaiman")
r <- read_caim(path, c(1250, 1020) - 745, 745 * 2, 745 * 2)
r
gbc(r)
```

`interpolate_sky_points` *Interpolate sky points*

Description

Interpolate values from canopy photographs.

Usage

```
interpolate_sky_points(sky_points, r, k = 3, p = 2, rmax = 200, col_id = "rl")
```

Arguments

`sky_points`	An object of class *data.frame*. The data.frame returned by `extract_rl()` or `extract_dn()`, or a *data.frame* with same basic structure and names.
`r`	SpatRaster. The image from which `sky_points` was obtained.
`k`	Numeric vector of length one. Number of k-nearest neighbors.
`p`	Numeric vector of length one. Power for inverse-distance weighting.
`rmax`	Numeric vector of length one. Maximum radius where to search for *knn*.
`col_id`	Numeric vector of length one. ID of the column with the values to interpolate.

Details

This function use `lidR::knnidw()` as workhorse function, so arguments `k`, `p`, and `rmax` are passed to it.

This function is based on Lang et al. (2010). In theory, the best result would be obtained with data showing a linear relation between digital numbers and the amount of light reaching the sensor. See `extract_radiometry()` and `read_caim_raw()` for further details. As a compromise solution, `gbc()` can be used.

Default parameters are the ones used by Lang et al. (2010). The argument `rmax` should account for between 15 to 20 degrees, but it is expressed in pixels units. So, image resolution and lens projections should be taken into account to set this argument properly.

Value

An object of class SpatRaster.

References

Lang M, Kuusk A, M~ottus M, Rautiainen M, Nilson T (2010). "Canopy gap fraction estimation from digital hemispherical images using sky radiance models and a linear conversion method." *Agricultural and Forest Meteorology*, **150**(1), 20–29. doi:10.1016/j.agrformet.2009.08.001.

See Also

Other Sky Reconstruction Functions: `cie_sky_model_raster()`, `fit_cie_sky_model()`, `fit_coneshaped_model()`, `fit_trend_surface()`, `fix_reconstructed_sky()`, `ootb_sky_reconstruction()`

Examples

```
## Not run:
caim <- read_caim()
r <- caim$Blue
caim <- normalize(caim, 0, 20847, TRUE)
z <- zenith_image(ncol(caim), lens())
a <- azimuth_image(z)
m <- !is.na(z)

bin <- ootb_obia(caim, z, a, m, HSV(239, 0.85, 0.5), gamma = NULL)
```

```
g <- sky_grid_segmentation(z, a, 10)
sky_points <- extract_sky_points(r, bin, g, dist_to_plant = 3)
plot(bin)
points(sky_points$col, nrow(caim) - sky_points$row, col = 2, pch = 10)
sky_points <- extract_dn(r, sky_points)

sky <- interpolate_sky_points(sky_points, r, col_id = 3)
plot(sky)
plot(r/sky)

# A quick demonstration of how to use trend surface fitting to smooth the
# interpolation
persp(terra::aggregate(sky, 10), theta = 45, phi = 30)
sky_s <- fit_trend_surface(sky, z, a, !is.na(z))
persp(terra::aggregate(sky_s$image, 10), theta = 45, phi = 30)
plot(sky_s$image)
plot(r)
plot(r/sky_s$image)
plot(apply_thr(r/sky_s$image, 0.5))

## End(Not run)
```

lens *Access the lens database*

Description

Database of lens projection functions and field of views.

Usage

```
lens(type = "equidistant", max_fov = FALSE)
```

Arguments

type	Character vector of length one. The name of the lens.
max_fov	Logical vector of length one. Use TRUE to return the maximum field of view in degrees.

Details

In upward-looking leveled hemispherical photography, the zenith is the center of a circle whose perimeter is the horizon. This is true only if the lens field of view is 180º. The relative radius is the radius of concentric circles expressed as a fraction of the radius that belongs to the circle that has the horizon as perimeter. The equidistant model, also called polar, is the most widely used as a standard reference. Real lenses can approximate the projection models, but they always have some kind of distortion. In the equidistant model, the relation between zenith angle and relative radius is modeled with a straight line. Following Hemisfer software, this package uses a polynomial curve to model lens distortion. A third-order polynomial is sufficient in most cases (Frazer et al. 2001). Equations should be fitted with angles in radians.

Eventually, this will be a large database, but only the following lenses are available at the moment:

- equidistant: standard equidistant projection (Schneider et al. 2009).
- Nikkor_10.5mm: AF DX Fisheye Nikkor 10.5mm f/2.8G ED (Pekin and Macfarlane 2009)
- Nikon_FCE9: Nikon FC-E9 converter (Díaz and et al. 2024)
- Olloclip: Auxiliary lens for mobile devices made by Olloclip (Díaz and et al. 2024)
- Nikkor_8mm: AF–S Fisheye Nikkor 8–15mm f/3.5–4.5E ED (Díaz and et al. 2024)

Value

If `max_fov` is set to `TRUE`, it returns a numeric vector of length one, which is the lens maximum field of view in degrees. Otherwise, it returns a numeric vector with the coefficients of the lens function.

References

Díaz GM, et al. (2024). "Simple calibration of fisheye lenses for hemipherical photography of the forest canopy." *Manuscript in preparation*.

Frazer GW, Fournier RA, Trofymow JA, Hall RJ (2001). "A comparison of digital and film fisheye photography for analysis of forest canopy structure and gap light transmission." *Agricultural and Forest Meteorology*, **109**(4), 249–263. doi:10.1016/s01681923(01)00274x.

Pekin B, Macfarlane C (2009). "Measurement of crown cover and leaf area index using digital cover photography and its application to remote sensing." *Remote Sensing*, **1**(4), 1298–1320. doi:10.3390/rs1041298.

Schneider D, Schwalbe E, Maas H (2009). "Validation of geometric models for fisheye lenses." *ISPRS Journal of Photogrammetry and Remote Sensing*, **64**(3), 259–266. doi:10.1016/j.isprsjprs.2009.01.001.

See Also

Other Lens Functions: `azimuth_image()`, `calc_diameter()`, `calc_relative_radius()`, `calc_zenith_colrow()`, `calibrate_lens()`, `crosscalibrate_lens()`, `expand_noncircular()`, `extract_radiometry()`, `fisheye_to_equidistant()`, `fisheye_to_pano()`, `test_lens_coef()`, `zenith_image()`

Examples

```
lens("Nikon_FCE9")
lens("Nikon_FCE9", max_fov = TRUE)

.fp <- function(theta, lens_coef) {
  x <- lens_coef[1:5]
  x[is.na(x)] <- 0
  for (i in 1:5) assign(letters[i], x[i])
  a * theta + b * theta^2 + c * theta^3 + d * theta^4 + e * theta^5
}
```

```
theta <- seq(0, pi/2, pi/180)
plot(theta, .fp(theta, lens()), type = "l", lty = 2,
      ylab = "relative radius")
lines(theta, .fp(theta, lens("Nikon_FCE9")))
```

local_fuzzy_thresholding

Local fuzzy thresholding

Description

This function was first presented in Díaz and Lencinas (2015). It uses a threshold value as the location parameter of a logistic membership function whose scale parameter depends on a variable, here named `mem`. This dependence can be explained as follows: if the variable is equal to `1`, then the membership function is same as a threshold function because the scale parameter is `0`; lowering the variable increases the scale parameter, thus blurring the threshold because it decreases the steepness of the curve. Since the variable is defined pixel by pixel, this should be considered as a **local** fuzzy thresholding method.

Usage

```
local_fuzzy_thresholding(lightness, m, mem, thr = NULL, fuzziness = NULL)
```

Arguments

`lightness`	SpatRaster. A normalized greyscale image (see `normalize()`).
`m`	SpatRaster. A mask. For hemispherical photographs, check `mask_hs()`.
`mem`	SpatRaster. It is the scale parameter of the logistic membership function. Typically it is obtained with `membership_to_color()`.
`thr`	Numeric vector of length one. Location parameter of the logistic membership function. Use `NULL` to estimate it automatically with `thr_isodata()`.
`fuzziness`	Numeric vector of length one. This number is a constant value that scales `mem`. Use `NULL` to estimate it automatically as the midpoint between the maximum and minimum values of `lightness`.

Details

Argument `m` can be used to affect the automatic estimation of `thr` and `fuzziness`.

If you use this function in your research, please cite Díaz and Lencinas (2015) in addition to this package (`citation("rcaiman"`).

Value

An object of class SpatRaster with same pixel dimensions than `caim`. Depending on `mem`, changes could be subtle.

References

Díaz GM, Lencinas JD (2015). "Enhanced gap fraction extraction from hemispherical photography." *IEEE Geoscience and Remote Sensing Letters*, **12**(8), 1785–1789. doi:10.1109/lgrs.2015.2425931.

See Also

Other Pre-processing Functions: `enhance_caim()`, `gbc()`, `membership_to_color()`, `normalize()`

Examples

```
## Not run:
caim <- read_caim()
z <- zenith_image(ncol(caim), lens())
a <- azimuth_image(z)

caim <- normalize(caim)

# ImageJ can be used to digitize points
path <- system.file("external/sky_points.csv",
                    package = "rcaiman")
img_points <- read.csv(path)
img_points <- img_points[c("Y", "X")]
colnames(img_points) <- c("row", "col")
head(img_points)
target_color <- extract_dn(caim, img_points, fun = median)
as(target_color, "HSV")
target_color <- HSV(240, 0.85, 0.5) #to increase saturation

mem <- membership_to_color(caim, target_color)
mem_thr <- local_fuzzy_thresholding(mean(caim), m,  mem$membership_to_grey)
plot(mem_thr)

## End(Not run)
```

`masking` *Image masking*

Description

Image masking

Usage

```
masking(r, m, RGB = c(1, 0, 0))
```

Arguments

`r`	SpatRaster. The image. Values should be normalized, see `normalize()`. Only methods for images with one or three layers have been implemented.
`m`	SpatRaster. A mask. For hemispherical photographs, check `mask_hs()`.
`RGB`	Numeric vector of length three. RGB color code. Red is the default color.

Value

An object of class SpatRaster that essentially is `r` with areas where `m` is equal to zero painted in a solid color. If `r` is a single layer image, then the layer is triplicated to allow the use of color.

See Also

`mask_hs()`

Other Tool Functions: `colorfulness()`, `correct_vignetting()`, `defuzzify()`, `extract_dn()`, `extract_feature()`, `extract_rl()`, `extract_sky_points_simple()`, `extract_sky_points()`, `extract_sun_coord()`, `find_sky_pixels_nonnull()`, `find_sky_pixels()`, `optim_normalize()`, `percentage_of_clipped_highlights()`, `read_bin()`, `read_caim_raw()`, `read_caim()`, `write_bin()`, `write_caim()`

Examples

```
## Not run:
 r <- read_caim()
 z <- zenith_image(ncol(r), lens())
 a <- azimuth_image(z)
 m <- mask_hs(z, 20, 70) & mask_hs(a, 90, 180)

 masked_caim <-  masking(normalize(r), m)
 plotRGB(masked_caim * 255)

 masked_bin <- masking(apply_thr(r$Blue, 125), m)
 plotRGB(masked_bin * 255)

## End(Not run)
```

`mask_hs` *Mask hemisphere*

Description

Given a zenith or azimuth image and angle restrictions, this function produces a mask.

Usage

```
mask_hs(r, from, to)
```

Arguments

`r`	SpatRaster built with `zenith_image()` or `azimuth_image()`.
`from, to`	angle in degrees, inclusive limits.

Value

An object of class SpatRaster with values `0` and `1`.

See Also

`masking()`

Other Segmentation Functions: `chessboard()`, `mask_sunlit_canopy()`, `polar_qtree()`, `qtree()`, `rings_segmentation()`, `sectors_segmentation()`, `sky_grid_segmentation()`

Examples

```
## Not run:
z <- zenith_image(1000, lens())
a <- azimuth_image(z)
m1 <- mask_hs(z, 20, 70)
plot(m1)
m2 <- mask_hs(a, 330,360)
plot(m2)
plot(m1 & m2)
plot(m1 | m2)

# 15 degrees at each side of 0
m1 <- mask_hs(a, 0, 15)
m2 <- mask_hs(a, 345, 360)
plot(m1 | m2)

# better use this
plot(!is.na(z))
# instead of this
plot(mask_hs(z, 0, 90))

## End(Not run)
```

`mask_sunlit_canopy` *Mask sunlit canopy*

Description

It is a wrapper function around `membership_to_color()`. It was developed with images in sRGB color space (Díaz 2023).

Usage

```
mask_sunlit_canopy(caim, m = NULL)
```

Arguments

caim	SpatRaster. The return of a call to `read_caim()` or `read_caim_raw()`.
m	SpatRaster. A mask. For hemispherical photographs, check `mask_hs()`.

Value

An object of class SpatRaster with values `0` and `1`.

References

Díaz GM (2023). "Optimizing forest canopy structure retrieval from smartphone-based hemispherical photography." *Methods in Ecology and Evolution*, **14**(3), 875–884. doi:10.1111/2041-210x.14059.

See Also

Other Segmentation Functions: `chessboard()`, `mask_hs()`, `polar_qtree()`, `qtree()`, `rings_segmentation()`, `sectors_segmentation()`, `sky_grid_segmentation()`

Examples

```
## Not run:
path <- system.file("external/APC_0020.jpg", package = "rcaiman")
caim <- read_caim(path)
plotRGB(caim)
caim <- normalize(caim)
m <- mask_sunlit_canopy(caim)
plot(m)

## End(Not run)
```

---

membership_to_color *Compute the membership to a target color*

Description

This function was first presented in Díaz and Lencinas (2015). It computes the degree of membership to a color using two Gaussian membership functions and the axes *A* and *B* from the *CIE LAB* color space. The lightness dimension is not considered in the calculations.

Usage

```
membership_to_color(caim, target_color, sigma = NULL)
```

Arguments

`caim`	SpatRaster. The return of a call to `read_caim()` or `read_caim_raw()`.
`target_color`	color.
`sigma`	Numeric vector of length one. Use `NULL` (default) to estimate it automatically as the euclidean distance between `target_color` and grey in the *CIE LAB* color space.

Details

If you use this function in your research, please cite Díaz and Lencinas (2015) in addition to this package (`citation("rcaiman")`).

Value

It returns an object from the class SpatRaster. First layer is the membership to the target color. Second layer is the membership to grey. Both memberships are calculated with same `sigma`.

References

Díaz GM, Lencinas JD (2015). "Enhanced gap fraction extraction from hemispherical photography." *IEEE Geoscience and Remote Sensing Letters*, **12**(8), 1785–1789. doi:10.1109/lgrs.2015.2425931.

See Also

Other Pre-processing Functions: `enhance_caim()`, `gbc()`, `local_fuzzy_thresholding()`, `normalize()`

Examples

```
## Not run:
caim <- read_caim() %>% normalize
z <- zenith_image(ncol(caim), lens())
a <- azimuth_image(z)
m <- !is.na(z)

sky_blue <- HSV(239, 0.85, 0.5)
mem <- membership_to_color(caim, sky_blue)
plot(mem)

## End(Not run)
```

`normalize` *Normalize data*

Description

Normalize numeric and raster data.

Usage

```
normalize(r, mn = NULL, mx = NULL, force_range = FALSE)
```

Arguments

`r`	SpatRaster or numeric vector.
`mn`	Numeric vector of length one. Minimum expected value. Default is equivalent to enter the minimum value from `r`.
`mx`	Numeric vector of length one. Maximum expected value. Default is equivalent to enter the maximum value from `r`.
`force_range`	Logical vector of length one. If it is `TRUE`, the range is forced to be between `0` and `1` by flattening values found below and above those limits.

Details

Normalize data laying between `mn` and `mx` to the range `0` to `1`. Data greater than `mx` get values greater than `1` in a proportional fashion. Conversely, data less than `mn` get values less than `0`.This function can be used for linear stretching of the histogram.

Value

An object from the same class as `r` with values from `r` linearly rescaled to make `mn` equal to zero and `mx` equal to one. Therefore, if `mn` and `mx` do not match the actual minimum and maximum from `r`, then the output will not cover the 0-to-1 range and may be outside that range if `force_range` is set to `FALSE`.

See Also

Other Pre-processing Functions: `enhance_caim()`, `gbc()`, `local_fuzzy_thresholding()`, `membership_to_color()`

Examples

```
normalize(read_caim())
```

`obia` *Do object-based image analysis of canopy photographs*

Description

Object-based image analysis targeting the canopy silhouette.

Usage

```
obia(r, z = NULL, a = NULL, bin, segmentation, gf_mn = 0.2, gf_mx = 0.95)
```

Arguments

r	SpatRaster. A normalized greyscale image. Typically, the blue channel extracted from a canopy photograph. Please see `read_caim()` and `normalize()`.
z	SpatRaster built with `zenith_image()`.
a	SpatRaster built with `azimuth_image()`.
bin	SpatRaster. This should be a working binarization of `r` without gross errors.
segmentation	SpatRaster built with `polar_qtree()` or `qtree()`.
gf_mn, gf_mx	Numeric vector of length one. The minimum/maximum gap fraction that a segment should comply with to be considered as one containing foliage.

Details

This method was first presented in Díaz and Lencinas (2015). This version is simpler since it relies on a better working binarized image. The version from 2015 uses an automatic selection of samples followed by a *knn* classification of segments containing foliage. This version uses de gap fraction extracted from `bin` to classify *foliage* by defining upper and lower limits through the arguments `gf_mx` and `gf_mn`.

This method produces a synthetic layer by computing the ratio of `r` to the maximum value of `r` at the segment level. This process is carried out only on the pixels covered by the classes *foliage* and *sky*. The latter is defined by `bin` equal to one. To avoid spurious values, the quantile `0.9` is computed instead of the maximum. Pixels not belonging to the class *foliage* return as `NA`.

Default values of `z` and `a` allows the processing of restricted view photographs.

If you use this function in your research, please cite Díaz and Lencinas (2015) in addition to this package (`citation("rcaiman"`).

Value

SpatRaster.

References

Díaz GM, Lencinas JD (2015). "Enhanced gap fraction extraction from hemispherical photography." *IEEE Geoscience and Remote Sensing Letters*, **12**(8), 1785–1789. doi:10.1109/lgrs.2015.2425931.

See Also

Other Binarization Functions: `apply_thr()`, `ootb_mblt()`, `ootb_obia()`, `regional_thresholding()`, `thr_isodata()`, `thr_mblt()`

Examples

```
## Not run:
caim <- read_caim() %>% normalize()
z <- zenith_image(ncol(caim), lens())
a <- azimuth_image(z)
m <- !is.na(z)
ecaim <- enhance_caim(caim, m)
```

```
bin <- apply_thr(ecaim, thr_isodata(ecaim[m]))
plot(bin)

seg <- polar_qtree(caim, z, a)
synth <- obia(caim$Blue, z, a, bin, seg)
plot(synth)
foliage <- !is.na(synth)
hist(synth[foliage])
synth <- terra::cover(synth, bin)
plot(synth)
hist(synth[foliage])

## End(Not run)
```

`ootb_mblt` *Out-of-the-box model-based local thresholding*

Description

Out-of-the-box version of the model-based local thresholding (MBLT) algorithm

Usage

```
ootb_mblt(r, z, a, bin = NULL, fix_cs_sky = FALSE, w = 0.5)
```

Arguments

`r`	SpatRaster. A normalized greyscale image. Typically, the blue channel extracted from a canopy photograph. Please see `read_caim()` and `normalize()`.
`z`	SpatRaster built with `zenith_image()`.
`a`	SpatRaster built with `azimuth_image()`.
`bin`	SpatRaster. This should be a preliminary binarization of `r` useful for masking pixels that are very likely pure sky pixels.
`fix_cs_sky`	Logical vector of length one. If it is `TRUE`, `fix_reconstructed_sky()` is used to fix the cone-shaped sky.
`w`	Numeric vector of length one. Weighting parameter from Díaz and Lencinas (2018)'s Equation 1. See `thr_mblt()`

Details

This function is a hard-coded version of a MBLT pipeline. The MBLT approach proposes a linear relationship between background value and optimal threshold value. This function uses statistical models for sky reconstruction that are able to explain smooth changes in sky brightness, so this function works best under clear skies or overcast conditions. After the reconstruction, local thresholds are linearly predicted from sky brightness values (see `thr_mblt()`).

As a high-level summary, the function starts producing a working binarized image and ends with a refined binarized image.

The pipeline combines these main functions `extract_sky_points_simple()` or `extract_sky_points()`, `fit_coneshaped_model()`, and `fit_trend_surface()`. The code can be easily inspected by calling `ootb_mblt` without parenthesis. Advanced users can use that code as a template.

The MBLT algorithm was first presented in Díaz and Lencinas (2018). The version presented here differs from the original in the following main aspects:

- The original version used a global thresholding method to provide sky points to the cone-shaped model. This one uses `extract_sky_points_simple()`. Nevertheless, a binarized image can be provided through the `bin` argument, triggering the use of `extract_sky_points()` instead of `extract_sky_points_simple()`.
- `intercept` and `slope` are automatically obtained using data from sky points and a linear model for accuracy evaluation after Piñeiro et al. (2008). This approach handles inaccuracies in background reconstruction (see `thr_mblt()` for additional details).
- This version does not use asynchronous acquisition under the open sky, as the original method did. The cone-shaped model (`fit_coneshaped_model()`) run without a filling source and the cone-shaped sky is used as filling source for trend surface fitting (`fit_trend_surface()`).

This function searches for black objects against a light background. When regular canopy hemispherical images are provided as input, the algorithm will find dark canopy elements against a bright sky almost everywhere in the picture and, therefore, the result will fit user expectations. However, if a hemispherical photograph taken under the open sky is provided, this algorithm would be still searching black objects against a light background, so the darker portions of the sky will be taken as objects, i.e., canopy. As a consequence, this will not fit users expectations since they are looking for the classes *Gap* and *No-gap*, no matter if one of those are not in the picture itself. This kind of error could happen with photographs of open forests for the same working principle.

If you use this function in your research, please cite Díaz and Lencinas (2018) in addition to this package (`citation("rcaiman"`).

Value

Object from class list containing the binarized image (named *bin*) and the reconstructed skies (named *sky_cs* and *sky_s*).

Note

If `NULL` is provided as the `w` argument, the weight is calculated as the coefficient of determination (R^2) of linear model for accuracy evaluation (Piñeiro et al. 2008).

References

Díaz GM, Lencinas JD (2018). "Model-based local thresholding for canopy hemispherical photography." *Canadian Journal of Forest Research*, **48**(10), 1204–1216. doi:10.1139/cjfr20180006.

Piñeiro G, Perelman S, Guerschman JP, Paruelo JM (2008). "How to evaluate models: Observed vs. predicted or predicted vs. observed?" *Ecological Modelling*, **216**(3-4), 316–322. doi:10.1016/j.ecolmodel.2008.05.006.

See Also

Other Binarization Functions: apply_thr(), obia(), ootb_obia(), regional_thresholding(), thr_isodata(), thr_mblt()

Examples

```
## Not run:
path <- system.file("external/DSCN4500.JPG", package = "rcaiman")
caim <- read_caim(path, c(1250, 1020) - 745, 745 * 2, 745 * 2)
z <- zenith_image(ncol(caim), lens("Nikon_FCE9"))
a <- azimuth_image(z)
r <- gbc(caim$Blue)
r <- correct_vignetting(r, z, c(0.0638, -0.101)) %>% normalize()
bin <- find_sky_pixels(r, z, a)
bin <- ootb_mblt(r, z, a, bin)
plot(bin$bin)


## End(Not run)
```

---

ootb_obia *Out-of-the-box object-based image analysis of canopy photographs*

Description

Out-of-the-box version of methods first presented in Díaz and Lencinas (2015).

Usage

```
ootb_obia(
  caim,
  z = NULL,
  a = NULL,
  m = NULL,
  sky_blue = NULL,
  w_red = 0,
  gamma = 2.2
)
```

Arguments

caim	SpatRaster. The return of a call to `read_caim()` or `read_caim_raw()`.
z	SpatRaster built with `zenith_image()`.
a	SpatRaster built with `azimuth_image()`.
m	SpatRaster. Default (`NULL`) is the equivalent to enter `!is.na(z)` for hemispherical photography, or enter `!is.na(caim$Red)` for restricted view photography.

`sky_blue`	color. Is the `target_color` argument to be passed to `membership_to_color()`. Default (`NULL`) is the equivalent to enter `sRGB(0.1, 0.4, 0.8)`–the HEX color code is #1A66CC, it can be entered into a search engine (such as Mozilla Firefox) to see a color swatch.
`w_red`	Numeric vector of length one. Weight of the red channel. A single layer image is calculated as a weighted average of the blue and red channels. This layer is used as lightness information. The weight of the blue is the complement of `w_red`.
`gamma`	Numeric vector of length one. This is for applying a gamma back correction to the lightness information (see Details and argument `w_red`).

Details

This function is a hard-coded version of a pipeline that combines these main functions `mask_sunlit_canopy()`, `enhance_caim()`, `polar_qtree()`/`qtree()`, and `obia()`. The code can be easily inspected by calling `ootb_obia` –no parenthesis. Advanced users can use that code as a template.

Pixels from the synthetic layer returned by `obia()` that lay between `0` and `1` are assigned to the class *plant* only if they comply with the following conditions:

- Their values are equal to `0` after `defuzzify()` with a sky grid segmentation of `10` degrees.
- Their values are equal to `0` after `apply_thr()` with a threshold computed with `thr_isodata()`.
- They are not exclusively surrounded by sky pixels.

Use the default values of `z` and `a` to process restricted view photographs.

If you use this function in your research, please cite Díaz and Lencinas (2015) or Díaz (2023) in addition to this package (`citation("rcaiman"`).

Value

An object of class SpatRaster with values `0` and `1`.

References

Díaz GM (2023). "Optimizing forest canopy structure retrieval from smartphone-based hemispherical photography." *Methods in Ecology and Evolution*, **14**(3), 875–884. doi:10.1111/2041-210x.14059.

Díaz GM, Lencinas JD (2015). "Enhanced gap fraction extraction from hemispherical photography." *IEEE Geoscience and Remote Sensing Letters*, **12**(8), 1785–1789. doi:10.1109/lgrs.2015.2425931.

See Also

Other Binarization Functions: `apply_thr()`, `obia()`, `ootb_mblt()`, `regional_thresholding()`, `thr_isodata()`, `thr_mblt()`

Examples

```
## Not run:
# ==============================================
# Circular Hemispherical Photo (from a raw file)
# ==============================================

caim <- read_caim()
z <- zenith_image(ncol(caim), lens())
a <- azimuth_image(z)
m <- !is.na(z)

mn <- quantile(caim$Blue[m], 0.01)
mx <- quantile(caim$Blue[m], 0.99)
r <- normalize(caim$Blue, mn, mx, TRUE)

bin <- find_sky_pixels(r, z, a)
mblt <- ootb_mblt(r, z, a, bin)
plot(mblt$bin)

mx <- optim_normalize(caim, mblt$bin)
ncaim <- normalize(caim, mx = mx, force_range = TRUE)
plotRGB(ncaim*255)
plotRGB(normalize(caim)*255)
percentage_of_clipped_highlights(ncaim$Blue, m)

bin2 <- ootb_obia(ncaim, z, a, gamma = NULL)
plot(bin2)

# =====================================
# Hemispherical Photo from a Smartphone
# =====================================

path <- system.file("external/APC_0581.jpg", package = "rcaiman")
caim <- read_caim(path) %>% normalize()
z <- zenith_image(2132/2, c(0.7836, 0.1512, -0.1558))
a <- azimuth_image(z)
zenith_colrow <- c(1063, 771)/2
caim <- expand_noncircular(caim, z, zenith_colrow) %>% normalize()
m <- !is.na(caim$Red) & !is.na(z)
caim[!m] <- 0

bin <- ootb_obia(caim, z, a)
plot(bin)

# ============================
# Restricted View Canopy Photo
# ============================

path <- system.file("external/APC_0020.jpg", package = "rcaiman")
caim <- read_caim(path) %>% normalize()

bin <- ootb_obia(caim)
```

```
plot(bin)

## End(Not run)
```

`ootb_sky_reconstruction` *Out-of-the-box sky reconstruction*

Description

Build an above canopy image from a single below canopy image

Usage

```
ootb_sky_reconstruction(
  r,
  z,
  a,
  bin,
  filling_source = NULL,
  dist_to_plant = 3,
  sun_coord = NULL,
  general_sky_type = NULL,
  twilight = TRUE,
  rmse = TRUE,
  method = "BFGS",
  try_grids = TRUE,
  thin_points = TRUE,
  refine_sun_coord = TRUE,
  try_optims = TRUE,
  force_sampling = TRUE,
  interpolate = TRUE
)
```

Arguments

`r`	SpatRaster. A normalized greyscale image. Typically, the blue channel extracted from a canopy photograph. Please see `read_caim()` and `normalize()`.
`z`	SpatRaster built with `zenith_image()`.
`a`	SpatRaster built with `azimuth_image()`.
`bin`	SpatRaster. This should be a preliminary binarization of `r` useful for masking pixels that are very likely pure sky pixels.
`filling_source`	SpatRaster. An actual or reconstructed above-canopy image to complement the sky pixels detected through the gaps of `r`. A photograph taken immediately after or before taking `r` under the open sky with the same equipment and configuration is a very good option but not recommended under fleeting clouds. The

	orientation relative to the North must be the same as for `r`. If it is set to `NULL` (default), only sky pixels from `r` will be used as input.
`dist_to_plant`	Numeric vector of length one or `NULL`. See `extract_sky_points()`.
`sun_coord`	An object of class *list*. The result of a call to `extract_sun_coord()` or an object with same structure and names. See also `row_col_from_zenith_azimuth()` in case you want to provide values based on date and time of acquisition and the `suncalc` package.
`general_sky_type`	Character vector of length one. It could be any of these: "Overcast", "Clear", or "Partly cloudy". See Table 1 from Li et al. (2016) for additional details.
`twilight`	Logical vector of length one. If it is `TRUE` and the initial standard parameters belong to the "Clear" general sky type, sun zenith angles from 90 to 96 degrees will be tested (civic twilight). This is necessary since `extract_sun_coord()` would mistakenly recognize the center of what can be seen of the solar corona as the solar disk.
`rmse`	Logical vector of length one. If it is `TRUE`, the criteria for selecting the best sky model is to choose the one with the lowest **root mean square error (RMSE)** calculated by using the `sky_points` argument as the source of reference values. Otherwise, the criteria is to evaluate the whole image by calculating the **out-of-range index** as $\sum_{i=1}^{N}(r_i/sky_i)^2$, where r is the `r` argument, sky is the raster obtained from the fitted model with `cie_sky_model_raster()` and `zenith_dn`, i is the index that represents the position of a given pixel on the raster grid, and N is the total number of pixels that satisfy either of these inequalities: $r_i/sky_i < 0$ and $r_i/sky_i > 1$.
`method`	Optimization method to use. See `optim`.
`try_grids`	Logical vector of length one.
`thin_points`	Logical vector of length one.
`refine_sun_coord`	Logical vector of length one.
`try_optims`	Logical vector of length one.
`force_sampling`	Logical vector of length one.
`interpolate`	Logical vector of length one. If `TRUE`, `interpolate_sky_points()` will be used.

Details

This function is a hard-coded version of a pipeline that uses these main functions `fit_cie_sky_model()` and `interpolate_sky_points()`.

The pipeline is an automatic version of the Lang et al. (2010) method.

Providing a `filling source` triggers an alternative pipeline in which the sky is fully reconstructed with `interpolate_sky_points()` after a dense sampling (1×1 degree cells), which is supported by the fact that sky digital numbers will be available for every pixel, either from `r` gaps or from the filling source.

Value

If a filling source is not provided, the result is an object from the class *list* that includes the following: (1) the reconstructed sky (SpatRaster), (2) the output produced by `fit_cie_sky_model()`, (3) the out-of-range index (see `fit_cie_sky_model()`), (4) sky points that were not involved in obtaining (2), (5) an object from the class `lm` (see `stats::lm()`) that is the result of validating (1) with (4) and the method recommended by Piñeiro et al. (2008), and (6) a binarized image produced with (1), the coefficients from (4) and `thr_mblt()` with `apply_thr()`, using 'w=0.95'. If a filling source is provided, only a reconstructed sky (SpatRaster) is returned.

References

Lang M, Kuusk A, M~ottus M, Rautiainen M, Nilson T (2010). "Canopy gap fraction estimation from digital hemispherical images using sky radiance models and a linear conversion method." *Agricultural and Forest Meteorology*, **150**(1), 20–29. doi:10.1016/j.agrformet.2009.08.001.

Li DH, Lou S, Lam JC, Wu RH (2016). "Determining solar irradiance on inclined planes from classified CIE (International Commission on Illumination) standard skies." *Energy*, **101**, 462–470. doi:10.1016/j.energy.2016.02.054.

Piñeiro G, Perelman S, Guerschman JP, Paruelo JM (2008). "How to evaluate models: Observed vs. predicted or predicted vs. observed?" *Ecological Modelling*, **216**(3-4), 316–322. doi:10.1016/j.ecolmodel.2008.05.006.

See Also

Other Sky Reconstruction Functions: `cie_sky_model_raster()`, `fit_cie_sky_model()`, `fit_coneshaped_model()`, `fit_trend_surface()`, `fix_reconstructed_sky()`, `interpolate_sky_points()`

Examples

```
## Not run:
caim <- read_caim()
z <- zenith_image(ncol(caim), lens())
a <- azimuth_image(z)
m <- !is.na(z)

m <- !is.na(z)
mn <- quantile(caim$Blue[m], 0.01)
mx <- quantile(caim$Blue[m], 0.99)
r <- normalize(caim$Blue, mn, mx, TRUE)

bin <- find_sky_pixels(r, z, a)
bin <- ootb_mblt(r, z, a, bin)
plot(bin$bin)

mx <- optim_normalize(caim, m)

r <- normalize(caim$Blue)
caim <- normalize(caim, mx = mx, force_range = TRUE)
```

```
bin <- ootb_obia(caim, z, a, m, HSV(239, 0.85, 0.5), gamma = NULL)
plot(bin)
bin <- ootb_mblt(r, z, a, bin)$bin
plot(bin)

set.seed(7)
sky <- ootb_sky_reconstruction(r, z, a, bin)

sky$sky
sky$validation %>% summary()
plot(sky$sky)
plot(r/sky$sky)
hist(r/sky$sky, xlim = c(0, 2), breaks = 255)
hist((r/sky$sky)[bin], xlim = c(0, 2), breaks = 255)
plot((r/sky$sky)>1.1)

plot(sky$bin)

sky2 <- ootb_sky_reconstruction(r, z, a, sky$bin, sky$sky)
plot(sky2)
plot(r/sky2)
hist(r/sky2, xlim = c(0, 2), breaks = 255)
hist((r/sky2)[sky$bin], xlim = c(0, 2), breaks = 255)
plot((r/sky2)>1.1)

## End(Not run)
```

optim_normalize *Optimize normalize parameters*

Description

Wrapper function for `bbmle::mle2()`. Optimize normalize parameters by maximizing `colorfulness()` and minimizing saturation.

Usage

```
optim_normalize(caim, bin, method = "BFGS")
```

Arguments

`caim`	SpatRaster. The return of a call to `read_caim()` or `read_caim_raw()`.
`bin`	SpatRaster. This should be a preliminary binarization of `r` useful for masking pixels that are very likely pure sky pixels.
`method`	Optimization method to use. See `optim`.

Value

Numeric vector of length one. The values for using as `mx` argument with `normalize()`.

See Also

Other Tool Functions: colorfulness(), correct_vignetting(), defuzzify(), extract_dn(), extract_feature(), extract_rl(), extract_sky_points_simple(), extract_sky_points(), extract_sun_coord(), find_sky_pixels_nonnull(), find_sky_pixels(), masking(), percentage_of_clipped_high
read_bin(), read_caim_raw(), read_caim(), write_bin(), write_caim()

Examples

```
## Not run:
caim <- read_caim()
z <- zenith_image(ncol(caim), lens())
a <- azimuth_image(z)
m <- !is.na(z)

mn <- quantile(caim$Blue[m], 0.01)
mx <- quantile(caim$Blue[m], 0.99)
r <- normalize(caim$Blue, mn, mx, TRUE)

bin <- find_sky_pixels(r, z, a)
mblt <- ootb_mblt(r, z, a, bin)
plot(mblt$bin)

mx <- optim_normalize(caim, mblt$bin)
ncaim <- normalize(caim, mx = mx, force_range = TRUE)
plotRGB(ncaim*255)
plotRGB(normalize(caim)*255)
percentage_of_clipped_highlights(ncaim$Blue, m)

## End(Not run)
```

---

percentage_of_clipped_highlights

Percentage of clipped highlights

Description

Wrapper function for terra::freq()

Usage

```
percentage_of_clipped_highlights(r, m)
```

Arguments

r	Single-layer object from the SpatRaster.
m	SpatRaster. A mask. For hemispherical photographs, check mask_hs().

Value

Numeric vector of lenght one.

See Also

Other Tool Functions: `colorfulness()`, `correct_vignetting()`, `defuzzify()`, `extract_dn()`, `extract_feature()`, `extract_rl()`, `extract_sky_points_simple()`, `extract_sky_points()`, `extract_sun_coord()`, `find_sky_pixels_nonnull()`, `find_sky_pixels()`, `masking()`, `optim_normalize()`, `read_bin()`, `read_caim_raw()`, `read_caim()`, `write_bin()`, `write_caim()`

Examples

```
r <- read_caim()$Blue
z <- zenith_image(ncol(r), lens())
m <- !is.na(z)
percentage_of_clipped_highlights(r, m)
r <- normalize(r, 0, 1000, TRUE)
percentage_of_clipped_highlights(r, m)
```

`polar_qtree` *Do quad-tree segmentation in the polar space*

Description

The quad-tree segmentation algorithm is a top-down process that makes recursive divisions in four equal parts until a condition is satisfied and stops locally. The usual implementation of the quad-tree algorithm is based on the raster structure and this is why the result are squares of different sizes. This method implements the quad-tree segmentation in a polar space, so the segments are shaped like windshields, though some of them will look elongated in height. The pattern is two opposite and converging straight sides and two opposite and parallel curvy sides.

Usage

```
polar_qtree(r, z, a, scale_parameter = 0.2)
```

Arguments

`r`	SpatRaster.
`z`	SpatRaster built with `zenith_image()`.
`a`	SpatRaster built with `azimuth_image()`.
`scale_parameter`	Numeric vector of length one. Quad-tree is a top-down method. This parameter controls the stopping condition. Therefore, it allows controlling the size of the resulting segments. Ultimately, segments sizes will depend on both this parameter and the heterogeneity of `r`.

Details

The algorithm splits segments of 30 degrees resolution into four sub-segments and calculates the standard deviation of the pixels from `r` delimited by each of those segments. The splitting process stops locally if the sum of the standard deviation of the sub-segments minus the standard deviation of the parent segment (named *delta*) is less or equal than the `scale_parameter`. If `r` has more than one layer, *delta* is calculated separately and *delta* mean is used to evaluate the stopping condition.

Value

A single layer image of the class SpatRaster with integer values.

See Also

Other Segmentation Functions: `chessboard()`, `mask_hs()`, `mask_sunlit_canopy()`, `qtree()`, `rings_segmentation()`, `sectors_segmentation()`, `sky_grid_segmentation()`

Examples

```
## Not run:
caim <- read_caim() %>% normalize()
z <- zenith_image(ncol(caim), lens())
a <- azimuth_image(z)
seg <- polar_qtree(caim, z, a)
plot(seg)
plot(extract_feature(caim$Blue, seg))

## End(Not run)
```

`qtree` *Do quad-tree segmentation*

Description

The quad-tree segmentation algorithm is a top-down process that makes recursive divisions in four equal parts until a condition is satisfied and stops locally. This is the usual implementation of the quad-tree algorithm, so it produces squared segments of different sizes. This particular implementation allows up to five sizes.

Usage

```
qtree(r, scale_parameter = 0.2)
```

Arguments

`r`	SpatRaster.

`scale_parameter`	Numeric vector of length one. Quad-tree is a top-down method. This parameter controls the stopping condition. Therefore, it allows controlling the size of the resulting segments. Ultimately, segments sizes will depend on both this parameter and the heterogeneity of `r`.

Details

The algorithm starts splitting the entire image into large squared segments. Depending on the aspect ratio, starting grids will going from 4×4 to 1×4 or 4×1. Then, it splits each segment into four sub-segments and calculates the standard deviation of the pixels from `r` delimited by each of those sub-segments and segment. The splitting process stops locally if *delta*, the sum of the standard deviation of the sub-segments minus the standard deviation of the parent segment, is less or equal than the `scale_parameter`. If `r` has more than one layer, *delta* is calculated separately and *delta* mean is used to evaluate the stopping condition.

Value

A single layer image of the class SpatRaster with integer values.

See Also

Other Segmentation Functions: `chessboard()`, `mask_hs()`, `mask_sunlit_canopy()`, `polar_qtree()`, `rings_segmentation()`, `sectors_segmentation()`, `sky_grid_segmentation()`

Examples

```
## Not run:
caim <- read_caim() %>% normalize()
seg <- qtree(caim, scale_parameter = 0.05)
plot(caim$Blue)
plot(extract_feature(caim$Blue, seg))
plot(extract_feature(seg, seg, length))

## End(Not run)
```

`read_bin` *Read binarized images*

Description

Wrapper functions for `terra::rast()`.

Usage

```
read_bin(path)
```

Arguments

`path`	Character vector of length one. Path to a binarized image.

Value

An object from class SpatRaster.

See Also

Other Tool Functions: `colorfulness()`, `correct_vignetting()`, `defuzzify()`, `extract_dn()`, `extract_feature()`, `extract_rl()`, `extract_sky_points_simple()`, `extract_sky_points()`, `extract_sun_coord()`, `find_sky_pixels_nonnull()`, `find_sky_pixels()`, `masking()`, `optim_normalize()`, `percentage_of_clipped_highlights()`, `read_caim_raw()`, `read_caim()`, `write_bin()`, `write_caim()`

Examples

```
## Not run:
z <- zenith_image(1000, lens())
m <- !is.na(z)
my_file <- file.path(tempdir(), "mask.tif")
write_bin(m, my_file)
m_from_disk <- read_bin(my_file)
plot(m - m_from_disk)

## End(Not run)
```

`read_caim` *Read a canopy image from a file*

Description

Wrapper function for `terra::rast()`.

Usage

```
read_caim(path = NULL, upper_left = NULL, width = NULL, height = NULL)
```

Arguments

`path`	Character vector of length one. Path to an image, including file extension. The function will return a data example if no arguments are provided.
`upper_left`	An integer vector of length two. The pixels coordinates of the upper left corner of a region of interest (ROI). These coordinates should be in the raster coordinates system. This system works like a spreadsheet, i.e, when going down through the vertical axis, the *row* number increases (**IMPORTANT**: column and row must be provided instead of row and column, as is the norm for objects from the class *data.frame* and others alike)
`width, height`	An integer vector of length one. The size of the boxy ROI whose upper left corner is the `upper_left` argument.

Details

Run `read_caim()` to obtain an example of a hemispherical photo taken in non-diffuse light conditions in a *Nothofagus pumilio* forest with a FC-E9 auxiliary lens attached to a Nikon Coolpix 5700.

Since this function aims to read born-digital color photographs, RGB-JPEG and RGB-TIFF are the expected input. However, since this function is a wrapper for `terra::rast()`, format compatibility is heritages from it.

Use `upper_left`, `width`, and `height` to read a particular region from the file. Although any image editor can be used to obtain those parameters, this function was tested with 'ImageJ', so it might be wise to use it.

TIP: For obtaining `upper_left`, `width`, and `height`, open the image on the Fiji distro of ImageJ, draw a rectangular selection, and go to Edit>Selection>Specify. The same workflow may work with other distros.

Value

An object from class SpatRaster with its layers named *Red*, *Green*, and *Blue* when a born-digital color photographs is provided as input.

Note

The example image was obtained with this code:

```
zenith_colrow <- c(1290, 988)
z <- zenith_image(745*2, lens("Nikon_FCE9"))
a <- azimuth_image(z)
r <- read_caim_raw("DSCN4606.NEF", z, a, zenith_colrow, radius = 300)
z <- zenith_image(ncol(r), lens())
r <- correct_vignetting(r, z, c(0.0638, -0.101))
r <- c(mean(r$Y, r$M), r$G, r$C)
r <- normalize(r, -1)
write_caim(r*2^16-2, "example.tif", 16)
```

See Also

Other Tool Functions: `colorfulness()`, `correct_vignetting()`, `defuzzify()`, `extract_dn()`, `extract_feature()`, `extract_rl()`, `extract_sky_points_simple()`, `extract_sky_points()`, `extract_sun_coord()`, `find_sky_pixels_nonnull()`, `find_sky_pixels()`, `masking()`, `optim_normalize()`, `percentage_of_clipped_highlights()`, `read_bin()`, `read_caim_raw()`, `write_bin()`, `write_caim()`

Examples

```
path <- system.file("external/DSCN4500.JPG", package = "rcaiman")
zenith_colrow <- c(1250, 1020)
diameter <- 745*2
caim <- read_caim(path, zenith_colrow - diameter/2, diameter, diameter)
plot(caim$Blue)
```

read_caim_raw *Read a canopy image from a raw file*

Description

Function that complements read_caim()

Usage

```
read_caim_raw(
  path = NULL,
  z = NULL,
  a = NULL,
  zenith_colrow = NULL,
  radius = 700,
  rmax = 100,
  k = 1,
  p = 1,
  only_blue = FALSE,
  offset_value = NULL
)
```

Arguments

path	Character vector of length one.Path to a raw file, including file extension.
z	SpatRaster built with `zenith_image()`.
a	SpatRaster built with `azimuth_image()`.
zenith_colrow	Numeric vector of length two. Raster coordinates of the zenith. See `calc_zenith_colrow()`.
radius	Numeric integer of length one. Radius of the reprojected hemispherical image (i.e., the output).
rmax	Numeric vector of length one. Maximum radius where to search for *knn*. Increase this value if pixels with value `0` or `FALSE` appears where other values are expected.
k	Numeric vector of length one. Number of k-nearest neighbors.
p	Numeric vector of length one. Power for inverse-distance weighting.
only_blue	Logical vector of length one. If `TRUE`, only values from the blue or cyan wavelength will be processed.
offset_value	numeric vector. This values will replace the `black_level_per_channel` metadata obtained with `rawpy`.

Details

This function facilitates the integration of the `rawpy` Python package into the R environment via the `reticulate` package. This integration allows `rcaiman` to access and pre-process raw data.

Here is a step-by-step guide to assist users in setting up the environment for efficient processing:

Check Python Accessibility::
To ensure that R can access a Python installation, run the following test:

```
reticulate::py_eval("1+1")
```

If R can access Python successfully, you will see `2` in the console. If not, you will receive instructions on how to install Python.

Create a Virtual Environment::
After passing the Python accessibility test, create a virtual environment using the following command:

```
reticulate::virtualenv_create()
```

Install `rawpy`::
Install the rawpy package within the virtual environment:

```
reticulate::py_install("rawpy")
```

For RStudio Users::
If you are an RStudio user who works with projects, you will need a *.Renviron* file in the root of each project. To create a *.Renviron* file, follow these steps:

- Create a "New Blank File" named ".Renviron" (without an extension) in the project's root directory.
- Run bellow code:

```
path <- file.path(reticulate::virtualenv_root(),
reticulate::virtualenv_list(), "Scripts", "python.exe")
paste("RETICULATE_PYTHON =", path)
```

- Copy/paste the line from the console (the string between the quotes) into the .Renviron file. This is an example `RETICULATE_PYTHON = ~/.virtualenvs/r-reticulate/Scripts/python.exe`
- Do not forget to save the changes

By following these steps, users can easily set up their environment to access raw data efficiently, but it is not the only way of doing it.

Value

An object from class SpatRaster. Single-layer raster if `only_blue` is equal to `TRUE`. Otherwise, a raster with as many layers as there are distinct colors in the Color Filter Array. Layer names are taken from the color description metadata.

See Also

Other Tool Functions: `colorfulness()`, `correct_vignetting()`, `defuzzify()`, `extract_dn()`, `extract_feature()`, `extract_rl()`, `extract_sky_points_simple()`, `extract_sky_points()`, `extract_sun_coord()`, `find_sky_pixels_nonnull()`, `find_sky_pixels()`, `masking()`, `optim_normalize()`, `percentage_of_clipped_highlights()`, `read_bin()`, `read_caim()`, `write_bin()`, `write_caim()`

`read_manual_input` *Read manual input*

Description

Read manual input stored in an HSP project.

Usage

```
read_manual_input(path_to_HSP_project, img_name)
```

Arguments

`path_to_HSP_project`	Character vector of length one. Path to the HSP project folder. For instance, "C:/Users/johndoe/Documents/HSP/Projects/my_prj/".
`img_name`	Character vector of length one. For instance, "DSCN6342.pgm" or "DSCN6342". See details.

Details

Refer to the Details section of function `write_sky_points()`.

Value

A list of numeric vectors named *weight*, *max_points*, *angle*, *point_radius*, *sun_coord*, *sky_points* and *zenith_dn*.

See Also

Other HSP Functions: `read_opt_sky_coef()`, `row_col_from_zenith_azimuth()`, `write_sky_points()`, `write_sun_coord()`, `zenith_azimuth_from_row_col()`

`read_opt_sky_coef` *Read optimized sky coefficients*

Description

Read optimized CIE sky coefficients stored in an HSP project.

Usage

```
read_opt_sky_coef(path_to_HSP_project, img_name)
```

Arguments

`path_to_HSP_project`	Character vector of length one. Path to the HSP project folder. For instance, "C:/Users/johndoe/Documents/HSP/Projects/my_prj/".
`img_name`	Character vector of length one. For instance, "DSCN6342.pgm" or "DSCN6342". See details.

Details

Refer to the Details section of function `write_sky_points()`.

Value

Numeric vector of length five.

See Also

`cie_sky_model_raster()`

Other HSP Functions: `read_manual_input()`, `row_col_from_zenith_azimuth()`, `write_sky_points()`, `write_sun_coord()`, `zenith_azimuth_from_row_col()`

`regional_thresholding` *Regional thresholding*

Description

Regional thresholding of greyscale images.

Usage

```
regional_thresholding(
  r,
  segmentation,
  method,
  intercept = NULL,
  slope = NULL,
  prob = NULL
)
```

Arguments

`r`	SpatRaster. A normalized greyscale image. Typically, the blue channel extracted from a canopy photograph. Please see `read_caim()` and `normalize()`.
`segmentation`	SpatRaster. The result of segmenting `r`. Arguably, the result of a call to `rings_segmentation()` will be the preferred choice for fisheye images.
`method`	Character vector of length one. See details for current options.
`intercept, slope`	Numeric vector of length one. These are linear function coefficients.
`prob`	Numeric vector of length one. Probability for `stats::quantile()` calculation.

Details

Methods currently implemented are:

- **Diaz2018**: method presented in Díaz and Lencinas (2018) applied regionally. If this method is selected, the arguments `intercept`, `slope`, and `prob` should be provided. It works segment-wise extracting the digital numbers per segment and passing them to `stats::quantile()` along with `prob`, which aggregated result is in turn passed to `thr_mblt()` along with `intercept` and `slope`. Finally, this threshold image is applied to obtain a binarized image.
- **Methods from autothresholdr package**: this function can call methods from `autothresholdr::auto_thresh()`. For instance, use `"IsoData"` to use the algorithm by Ridler and Calvard (1978), which was recommended by Jonckheere et al. (2005).
- **Method isodata from this package**: Use `"thr_isodata"` to use `thr_isodata()`.

Value

An object of class SpatRaster with values `0` and `1`.

References

Díaz GM, Lencinas JD (2018). "Model-based local thresholding for canopy hemispherical photography." *Canadian Journal of Forest Research*, **48**(10), 1204–1216. doi:10.1139/cjfr20180006.

Jonckheere I, Nackaerts K, Muys B, Coppin P (2005). "Assessment of automatic gap fraction estimation of forests from digital hemispherical photography." *Agricultural and Forest Meteorology*, **132**(1-2), 96–114. doi:10.1016/j.agrformet.2005.06.003.

Ridler TW, Calvard S (1978). "Picture thresholding using an iterative selection method." *IEEE Transactions on Systems, Man, and Cybernetics*, **8**(8), 630–632. doi:10.1109/tsmc.1978.4310039.

See Also

Other Binarization Functions: `apply_thr()`, `obia()`, `ootb_mblt()`, `ootb_obia()`, `thr_isodata()`, `thr_mblt()`

Examples

```
## Not run:
path <- system.file("external/DSCN4500.JPG", package = "rcaiman")
caim <- read_caim(path, c(1250, 1020) - 745, 745 * 2, 745 * 2)
z <- zenith_image(ncol(caim), lens("Nikon_FCE9"))
r <- gbc(caim$Blue)
r <- correct_vignetting(r, z, c(0.0638, -0.101)) %>% normalize()
rings <- rings_segmentation(z, 15)
bin <- regional_thresholding(r, rings, "Diaz2018", -7.8, 0.95 * 0.5, 0.99)
plot(bin)
bin <- regional_thresholding(r, rings, "thr_isodata")
plot(bin)
#'
## End(Not run)
```

`rings_segmentation` *Do rings segmentation*

Description

Segment an hemispherical view by slicing the zenith angle from zero to 90º in equals intervals.

Usage

```
rings_segmentation(z, angle_width, return_angle = FALSE)
```

Arguments

`z`	SpatRaster built with `zenith_image()`.
`angle_width`	Numeric vector of length one. Angle in degrees able to divide the angle range into a whole number of segments.
`return_angle`	Logical vector of length one. If it is `FALSE`, all the pixels that belong to a segment are labeled with an ID number. Otherwise, the angle mean of the segment is assigned to the pixels.

Value

An object from the class SpatRaster with segments shaped like concentric rings.

See Also

Other Segmentation Functions: `chessboard()`, `mask_hs()`, `mask_sunlit_canopy()`, `polar_qtree()`, `qtree()`, `sectors_segmentation()`, `sky_grid_segmentation()`

Examples

```
z <- zenith_image(600, lens())
rings <- rings_segmentation(z, 15)
plot(rings == 1)
```

`row_col_from_zenith_azimuth`

Obtain row and col numbers from zenith and azimuth angles

Description

Obtain row and col numbers from zenith and azimuth angles

Usage

```
row_col_from_zenith_azimuth(z, a, za, lens_coef = NULL)
```

Arguments

`z`	SpatRaster built with `zenith_image()`.
`a`	SpatRaster built with `azimuth_image()`.
`za`	Numeric vector of length two. Zenith and azimuth angles in degrees.
`lens_coef`	Numeric vector. Polynomial coefficients of the lens projection function. See `calibrate_lens()`.

Value

Numeric vector of length two.

Note

Use the `lens_coef` argument to calculate coordinates below the horizon.

See Also

Other HSP Functions: `read_manual_input()`, `read_opt_sky_coef()`, `write_sky_points()`, `write_sun_coord()`, `zenith_azimuth_from_row_col()`

Examples

```
z <- zenith_image(1000, lens())
a <- azimuth_image(z)
row_col_from_zenith_azimuth(z, a, c(45, 270))
```

`sectors_segmentation` *Do sectors segmentation*

Description

Segment a hemispherical view by slicing the azimuth angle from zero to 360º in equals intervals.

Usage

```
sectors_segmentation(a, angle_width, return_angle = FALSE)
```

Arguments

`a`	SpatRaster built with `azimuth_image()`.
`angle_width`	Numeric vector of length one. Angle in degrees able to divide the angle range into a whole number of segments.
`return_angle`	Logical vector of length one. If it is `FALSE`, all the pixels that belong to a segment are labeled with an ID number. Otherwise, the angle mean of the segment is assigned to the pixels.

Value

An object from the class SpatRaster with segments shaped like pizza slices.

See Also

Other Segmentation Functions: `chessboard()`, `mask_hs()`, `mask_sunlit_canopy()`, `polar_qtree()`, `qtree()`, `rings_segmentation()`, `sky_grid_segmentation()`

Examples

```
z <- zenith_image(600, lens())
a <- azimuth_image(z)
sectors <- sectors_segmentation(a, 15)
plot(sectors == 1)
```

`sky_grid_segmentation` *Do sky grid segmentation*

Description

Segment the hemisphere view into segments of equal angular resolution for both zenith and azimuth angles.

Usage

```
sky_grid_segmentation(z, a, angle_width, sequential = FALSE)
```

Arguments

`z`	SpatRaster built with `zenith_image()`.
`a`	SpatRaster built with `azimuth_image()`.
`angle_width`	Numeric vector of length one. It should be `30, 15, 10, 7.5, 6, 5, 3.75, 3, 2.5, 1.875, 1` or `0.5` degrees. This constrain is rooted in the requirement of a value able to divide both the `0` to `360` and `0` to `90` ranges into a whole number of segments.
`sequential`	Logical vector of length one. If it is `TRUE`, the segments are labeled with sequential numbers. By default (`FALSE`), labeling numbers are not sequential (see Details).

Details

Intersecting rings with sectors makes a grid in which each cell is a portion of the hemisphere. Each pixel of the grid is labeled with an ID that codify both ring and sector IDs. For example, a grid with a regular interval of one degree has segment from `1001` to `360090`. This numbers are calculated with: $sectorID \times 1000 + ringID$, where $sectorID$ is the ID number of the sector and $ringID$ is the ID number of the ring.

Value

An object from the class SpatRaster with segments shaped like windshields, though some of them will look elongated in height. The pattern is two opposite and converging straight sides and two opposite and parallel curvy sides.

See Also

Other Segmentation Functions: `chessboard()`, `mask_hs()`, `mask_sunlit_canopy()`, `polar_qtree()`, `qtree()`, `rings_segmentation()`, `sectors_segmentation()`

Examples

```
z <- zenith_image(600, lens())
a <- azimuth_image(z)
g <- sky_grid_segmentation(z, a, 15)
plot(g == 24005)
## Not run:
g <- sky_grid_segmentation(z, a, 15, sequential = TRUE)
col <- terra::unique(g) %>% nrow() %>% rainbow() %>% sample()
plot(g, col = col)

## End(Not run)
```

test_lens_coef *Test lens projection functions*

Description

Test if a lens projection function will work between the 0-to-1 range.

Usage

```
test_lens_coef(lens_coef)
```

Arguments

`lens_coef`	Numeric vector. Polynomial coefficients of the lens projection function. See `calibrate_lens()`.

Details

The package tolerate a number very close to 1 but not exactly 1 as long as it is greater than 1. Therefore, when the test fails at this *"Test that lens projection function does not predict values barely below one"*, the best practice is to manually edit the last coefficient. For instance, changing it from -0.0296 to -0.0295. See `testthat::expect_equal()` for further details.

If it fails in *"Test that lens projection function works between the 0-to-1 range"*, collecting data again might be necessary.

Value

Returns `invisible(TRUE)` and print "Test passed" if all tests pass, otherwise throws an error.

See Also

Other Lens Functions: `azimuth_image()`, `calc_diameter()`, `calc_relative_radius()`, `calc_zenith_colrow()`, `calibrate_lens()`, `crosscalibrate_lens()`, `expand_noncircular()`, `extract_radiometry()`, `fisheye_to_equidistant()`, `fisheye_to_pano()`, `lens()`, `zenith_image()`

Examples

```
test_lens_coef(lens("Nikon_FCE9"))
test_lens_coef(2/pi)
```

thr_isodata — *Calculate a threshold with the isodata method*

Description

Threshold calculated with the algorithm by Ridler and Calvard (1978), which was recommended by Jonckheere et al. (2005).

Usage

```
thr_isodata(x)
```

Arguments

`x`	Numeric vector or a single-column *matrix* or *data.frame* able to be coerced to numeric.

Details

The implementation is based on the IsoData method of Auto Threshold ImageJ plugin by Gabriel Landini, which is now available in the `autothresholdr` package (`autothresholdr::auto_thresh()`). However, I found this implementarion more versatile since it is not restricted to an 8-bit input.

Value

Numeric vector of length one.

References

Jonckheere I, Nackaerts K, Muys B, Coppin P (2005). "Assessment of automatic gap fraction estimation of forests from digital hemispherical photography." *Agricultural and Forest Meteorology*, **132**(1-2), 96–114. doi:10.1016/j.agrformet.2005.06.003.

Ridler TW, Calvard S (1978). "Picture thresholding using an iterative selection method." *IEEE Transactions on Systems, Man, and Cybernetics*, **8**(8), 630–632. doi:10.1109/tsmc.1978.4310039.

See Also

Other Binarization Functions: `apply_thr()`, `obia()`, `ootb_mblt()`, `ootb_obia()`, `regional_thresholding()`, `thr_mblt()`

Examples

```
caim <- read_caim()
r <- gbc(caim$Blue)
thr <- thr_isodata(values(r))
bin <- apply_thr(r, thr)
plot(bin)
```

thr_mblt *Calculate thresholds with the model-based method*

Description

Transform background digital number into threshold values

Usage

```
thr_mblt(dn, intercept, slope)
```

Arguments

`dn`	Numeric vector or SpatRaster. Digital number of the background. These values should be normalized and, if they are extracted from a JPEG image, gamma back corrected.
`intercept, slope`	Numeric vector of length one. These are linear function coefficients.

Details

This function transforms background digital numbers into threshold values by means of the Equation 1 from Díaz and Lencinas (2018), which is a linear function with the slope modified by a weighting parameter. This simple function was found by studying canopy models, also known as targets, which are perforated surfaces made of a rigid and dark material . These models were backlighted with homogeneous lighting, photographed with a Nikon Coolpix 5700 set to acquire in JPEG format, and those images were gamma back corrected with a default gamma value equal to 2.2 (see `gbc()`). Results shown that the optimal threshold value was linearly related with the background digital number (see Figure 1 and Figure 7 from Díaz and Lencinas (2018)). This shifted the aim from finding the optimal threshold, following Song et al. (2014) method, to obtaining the background DN as if the canopy was not there, as Lang et al. (2010) proposed.

Working principle:

Díaz and Lencinas (2018) observed the following linear relationship between the background value, usually the sky digital number (SDN), and the optimal threshold value (OTV):

$$IV = a + b \cdot SDN \qquad \text{(Equation 1a)}$$

$$OTV = a + b \cdot w \cdot SDN \qquad \text{(Equation 1b)}$$

were IV is the initial value (Wagner 2001), which is the boundary between SDN and the mixed pixels, i.e, the pixels that are neither *Gap* or *Non-gap* (Macfarlane 2011), a and b are the intercept and slope coefficients, respectively, and w is a weighting parameter that takes into account that OTV is always lower than IV. If SDN is calculated at the pixel level, a local thresholding method can be applied by evaluating, pixel by pixel, if the below canopy digital number (CDN) is greater than the OTV. Formally, If $CDN > OTV$, then assign *Gap* class, else assign *Non-gap* class.

This conclusion drawn from an image processing point of view matches with previous findings drawn from a radiometric measurement paradigm, which are introduced next.

Cescatti (2007) posed that cameras can be used as a radiation measurement device if they are properly calibrated. This method, denominated by the author as LinearRatio, seeks to obtain the transmittance (T) as the ratio of below to above canopy radiation:

$$T = CDN/SDN \qquad \text{(Equation 2)}$$

were CDN is below canopy digital number (DN), i.e., the DN extracted from a canopy hemispherical photograph.

The LinearRatio method uses T as a proxy for gap fraction. It requires twin cameras, one below and the other above the canopy. In contrast, Lang et al. (2010) proposed to obtain SDN by manually selecting pure sky pixels from canopy hemispherical photographs and reconstructing the whole sky by subsequent modeling and interpolating—this method is often referred to as LinearRatio single camera or LinearRatioSC.

Equation 2 can be seen as a standardization of the distance between CDN and SDN. With that in mind, it is useful to rewrite Equation 1b as an inequality that can be evaluated to return a logical statement that is directly translated into the desired binary classification:

$$CDN > a + b \cdot w \cdot SDN \qquad \text{(Equation 3)}$$

Then, combining Equation 2 and 3, we find that Díaz and Lencinas (2018) parameters can be applied to T:

$$CDN/SDN > a + b \cdot w \cdot SDN/SDN \qquad \text{(Equation 4a)}$$

$$T > a + b \cdot w \qquad \text{(Equation 4b)}$$

From Equation 2 it is evident that any bias introduced by the camera optical and electronic system will be canceled during the calculation of T as long as only one camera is involved. Therefore, After examining Equation 4b, we can conclude that intercept 0 and slope 1 are theoretically correct.In addition, the w parameter can be used to filter out mixed pixels. The greater w, the greater the possibility of selecting pure sky pixels.

Value

An object of the same class and dimensions than `dn`.

Note

It is worth noting that Equation 1 was developed with 8-bit images, so calibration of new coefficient should be done in the 0 to 255 domain since that is what `thr_mblt()` expect, although the `dn` argument should be normalized. The latter, in spite of sounding counter intuitive, was a design decision aiming to harmonize the whole package.

Nevertheless, new empirical calibration on JPEG files may be unnecessary since the values -7.8 `intercept` and 0.95 `slope` that had been observed with back-gamma corrected JPEG files produced with the Nikon Coolpix 5700 camera are sufficiently close to the theoretical values that it sounds reasonable to interpret them as a confirmation of the theory.

Users are encouraged to adopt raw file acquisition (`read_caim_raw()`).

To apply the weighting parameter (w) from Equation 1, just provide the argument `slope` as $slope \times w$.

References

Cescatti A (2007). "Indirect estimates of canopy gap fraction based on the linear conversion of hemispherical photographs." *Agricultural and Forest Meteorology*, **143**(1-2), 1–12. doi:10.1016/j.agrformet.2006.04.009.

Díaz GM, Lencinas JD (2018). "Model-based local thresholding for canopy hemispherical photography." *Canadian Journal of Forest Research*, **48**(10), 1204–1216. doi:10.1139/cjfr20180006.

Lang M, Kuusk A, M~ottus M, Rautiainen M, Nilson T (2010). "Canopy gap fraction estimation from digital hemispherical images using sky radiance models and a linear conversion method." *Agricultural and Forest Meteorology*, **150**(1), 20–29. doi:10.1016/j.agrformet.2009.08.001.

Macfarlane C (2011). "Classification method of mixed pixels does not affect canopy metrics from digital images of forest overstorey." *Agricultural and Forest Meteorology*, **151**(7), 833–840. doi:10.1016/j.agrformet.2011.01.019.

Song GM, Doley D, Yates D, Chao K, Hsieh C (2014). "Improving accuracy of canopy hemispherical photography by a constant threshold value derived from an unobscured overcast sky." *Canadian Journal of Forest Research*, **44**(1), 17–27. doi:10.1139/cjfr20130082.

Wagner S (2001). "Relative radiance measurements and zenith angle dependent segmentation in hemispherical photography." *Agricultural and Forest Meteorology*, **107**(2), 103–115. doi:10.1016/s01681923(00)00232x.

See Also

`normalize()`, `gbc()`, `apply_thr()` and `regional_thresholding()`.

Other Binarization Functions: `apply_thr()`, `obia()`, `ootb_mblt()`, `ootb_obia()`, `regional_thresholding()`, `thr_isodata()`

Examples

```
thr_mblt(gbc(125), -7.8, 0.95 * 0.5)
```

write_bin *Write binarized images*

Description

Wrapper functions for `terra::writeRaster()`.

Usage

```
write_bin(bin, path)
```

Arguments

`bin`	SpatRaster.
`path`	Character vector of length one. Path for writing the image.

Value

No return value. Called for side effects.

See Also

Other Tool Functions: `colorfulness()`, `correct_vignetting()`, `defuzzify()`, `extract_dn()`, `extract_feature()`, `extract_rl()`, `extract_sky_points_simple()`, `extract_sky_points()`, `extract_sun_coord()`, `find_sky_pixels_nonnull()`, `find_sky_pixels()`, `masking()`, `optim_normalize()`, `percentage_of_clipped_highlights()`, `read_bin()`, `read_caim_raw()`, `read_caim()`, `write_caim()`

Examples

```
## Not run:
z <- zenith_image(1000, lens())
m <- !is.na(z)
my_file <- file.path(tempdir(), "mask")
write_bin(m, my_file)
my_file <- as.filename(my_file) %>%
  insert(., ext = "tif", replace = TRUE) %>%
  as.character()
m_from_disk <- read_bin(my_file)
plot(m - m_from_disk)

## End(Not run)
```

write_caim *Write canopy image*

Description

Wrapper function for terra::writeRaster().

Usage

```
write_caim(caim, path, bit_depth)
```

Arguments

caim	SpatRaster.
path	Character vector of length one. Path for writing the image.
bit_depth	Numeric vector of length one.

Value

No return value. Called for side effects.

See Also

Other Tool Functions: colorfulness(), correct_vignetting(), defuzzify(), extract_dn(), extract_feature(), extract_rl(), extract_sky_points_simple(), extract_sky_points(), extract_sun_coord(), find_sky_pixels_nonnull(), find_sky_pixels(), masking(), optim_normalize(), percentage_of_clipped_highlights(), read_bin(), read_caim_raw(), read_caim(), write_bin()

Examples

```
## Not run:
caim <- read_caim() %>% normalize(., 0, 255)
write_caim(caim * 2^8-2, file.path(tempdir(), "test_8bit"), 8)
write_caim(caim * 2^16-2, file.path(tempdir(), "test_16bit"), 16)
# Note: the normalized values are scaled by multiplying by 2^bit_depth-2
# to avoid storing in the maximum bin because those values will be
# interpreted as NA by read_caim(), and that is undesired.

## End(Not run)
```

write_sky_points *Write sky points*

Description

Create a special file to interface with the HSP software.

Usage

```
write_sky_points(sky_points, path_to_HSP_project, img_name)
```

Arguments

`sky_points`	An object of the class *data.frame*. The result of a calling to `extract_sky_points()`.
`path_to_HSP_project`	Character vector of length one. Path to the HSP project folder. For instance, "C:/Users/johndoe/Documents/HSP/Projects/my_prj/".
`img_name`	Character vector of length one. For instance, "DSCN6342.pgm" or "DSCN6342". See details.

Details

This function is part of a workflow that connects this package with the HSP software package (Lang et al. 2013).

This function was designed to be called after `extract_sky_points()`. The `r` argument provided to `extract_sky_points()` should be an image pre-processed by the HSP software. Those images are stored as PGM files in the subfolder "manipulate" of the project folder (which will be in turn a subfolder of the "projects" folder). Those PGM files can be read with `read_caim()`.

The `img_name` argument of `write_sky_points()` should be the name of the file associated to the aforementioned `r` argument.

The following code exemplifies how this package can be used in conjunction with the HSP software. The code assumes that the user is working within an RStudio project located in the HSP project folder.

```
r <- read_caim("manipulate/IMG_1014.pgm")
plot(r)
z <- zenith_image(ncol(r), lens())
a <- azimuth_image(z)
g <- sky_grid_segmentation(z, a, 10)
mblt <- ootb_mblt(r, z, a)$bin
bin <- mask_hs(z, 0, 85) & bin

sun_coord <- extract_sun_coord(r, z, a, bin, g)
write_sun_coord(sun_coord$row_col, ".", "IMG_1014")

sky_points <- extract_sky_points(r, bin, g)
write_sky_points(sky_points, ".", "IMG_1014")
```

Value

None. A file will be written in the HSP project folder.

References

Lang M, Kodar A, Arumäe T (2013). "Restoration of above canopy reference hemispherical image from below canopy measurements for plant area index estimation in forests." *Forestry Studies*, **59**(1), 13–27. doi:10.2478/fsmu20130008.

See Also

Other HSP Functions: `read_manual_input()`, `read_opt_sky_coef()`, `row_col_from_zenith_azimuth()`, `write_sun_coord()`, `zenith_azimuth_from_row_col()`

`write_sun_coord` *Write sun coordinates*

Description

Create a special file to interface with the HSP software.

Usage

```
write_sun_coord(sun_coord, path_to_HSP_project, img_name)
```

Arguments

`sun_coord`	Numeric vector of length two. Raster coordinates of the solar disk that can be obtained by calling to `extract_sun_coord()`. **TIP**: if the output of `extrac_sun_coord()` is `sun_coord`, then you should provide here this: `sun_coord$row_col`. See also `row_col_from_zenith_azimuth()` in case you want to provide values based on date and time of acquisition and the `suncalc` package.
`path_to_HSP_project`	Character vector of length one. Path to the HSP project folder. For instance, "C:/Users/johndoe/Documents/HSP/Projects/my_prj/".
`img_name`	Character vector of length one. For instance, "DSCN6342.pgm" or "DSCN6342". See details.

Details

Refer to the Details section of function `write_sky_points()`.

Value

None. A file will be written in the HSP project folder.

See Also

Other HSP Functions: `read_manual_input()`, `read_opt_sky_coef()`, `row_col_from_zenith_azimuth()`, `write_sky_points()`, `zenith_azimuth_from_row_col()`

`zenith_azimuth_from_row_col` *Obtain zenith and azimuth angles from row and col numbers*

Description

Obtain zenith and azimuth angles from row and col numbers

Usage

```
zenith_azimuth_from_row_col(z, a, row_col, lens_coef = NULL)
```

Arguments

`z`	SpatRaster built with `zenith_image()`.
`a`	SpatRaster built with `azimuth_image()`.
`row_col`	Numeric vector of length two. Row and col numbers.
`lens_coef`	Numeric vector. Polynomial coefficients of the lens projection function. See `calibrate_lens()`.

Value

Numeric vector of length two.

Note

Use the `lens_coef` argument to calculate coordinates below the horizon.

See Also

Other HSP Functions: `read_manual_input()`, `read_opt_sky_coef()`, `row_col_from_zenith_azimuth()`, `write_sky_points()`, `write_sun_coord()`

Examples

```
z <- zenith_image(1000, lens())
a <- azimuth_image(z)
zenith_azimuth_from_row_col(z, a, c(501, 750))
```

`zenith_image` *Build Zenith image*

Description

Build a single layer-image with zenith angle values, assuming upwards-looking hemispherical photography with the optical axis vertically aligned.

Usage

```
zenith_image(diameter, lens_coef)
```

Arguments

`diameter`	Numeric vector of length one. Diameter in pixels expressed as an even integer. The latter is to simplify calculations by having the zenith point located between pixels. Snapping the zenith point between pixels does not affect accuracy because half-pixel is less than the uncertainty in localizing the circle within the picture.
`lens_coef`	Numeric vector. Polynomial coefficients of the lens projection function. See `calibrate_lens()`.

Value

An object of class SpatRaster of zenith angles in degrees, showing a complete hemispherical view with the zenith on the center.

See Also

Other Lens Functions: `azimuth_image()`, `calc_diameter()`, `calc_relative_radius()`, `calc_zenith_colrow()`, `calibrate_lens()`, `crosscalibrate_lens()`, `expand_noncircular()`, `extract_radiometry()`, `fisheye_to_equidistant()`, `fisheye_to_pano()`, `lens()`, `test_lens_coef()`

Examples

```
z <- zenith_image(600, lens("Nikon_FCE9"))
plot(z)
```

Index