Package 'imager'

September 12, 2016

Type Package

Title Image Processing Library Based on 'CImg'

Version 0.31

Date 2016-09-02

Author Simon Barthelme [aut, cre],
Antoine Cecchi [ctb]

Maintainer Simon Barthelme <simon.barthelme@gipsa-lab.fr>

Description Fast image processing for images in up to 4 dimensions (two spatial dimensions, one time/depth dimension, one colour dimension). Provides most traditional image processing tools (filtering, morphology, transformations, etc.) as well as various functions for easily analysing image data using R. The package wraps CImg, <http://cimg.eu>, a simple, modern C++ library for image processing.

License LGPL

Imports Rcpp (>= 0.11.5),methods,stringr,png,jpeg,readbitmap,grDevices,purrr

Depends R (>= 2.10.0),plyr,magrittr

URL http://dahtah.github.io/imager, https://github.com/dahtah/imager

BugReports https://github.com/dahtah/imager/issues

SystemRequirements fftw3

Enhances spatstat

LinkingTo Rcpp

LazyData true

RoxygenNote 5.0.1

Suggests knitr, rmarkdown,ggplot2,dplyr,scales, testthat,OpenMPController

VignetteBuilder knitr

NeedsCompilation yes

Repository CRAN

Date/Publication 2016-09-12 15:08:55

R topics documented:

`add.colour` *Add colour channels to an grayscale image*

Description

Add colour channels to an grayscale image

Usage

```
add.colour(im, simple = TRUE)
```

Arguments

`im`	a grayscale image
`simple`	if TRUE just stack three copies of the grayscale image, if FALSE treat the image as the L channel in an HSL representation. Default TRUE.

Value

an image of class cimg

Author(s)

Simon Barthelme

Examples

```
grayscale(boats) #No more colour channels
add.colour(grayscale(boats)) #Image has depth = 3 (but contains only grays)
```

`as.cimg` *Convert to cimg object*

Description

Imager implements various converters that turn your data into cimg objects. If you convert from a vector (which only has a length, and no dimension), either specify dimensions explicitly or some guesswork will be involved. See examples for clarifications.

Usage

```
as.cimg(obj, ...)

## S3 method for class 'numeric'
as.cimg(obj, ...)

## S3 method for class 'logical'
as.cimg(obj, ...)

## S3 method for class 'double'
as.cimg(obj, ...)

## S3 method for class 'vector'
as.cimg(obj, x = NA, y = NA, z = NA, cc = NA,
  dim = NULL, ...)

## S3 method for class 'matrix'
as.cimg(obj, ...)
```

Arguments

`obj`	an object
`...`	optional arguments
`x`	width
`y`	height
`z`	depth
`cc`	spectrum
`dim`	a vector of dimensions (optional, use instead of xyzcc)

Methods (by class)

- `numeric`: convert numeric
- `logical`: convert logical
- `double`: convert double
- `vector`: convert vector
- `matrix`: Convert to matrix

Author(s)

Simon Barthelme

See Also

as.cimg.array, as.cimg.function, as.cimg.data.frame

Examples

```
as.cimg(1:100,x=10,y=10) #10x10, grayscale image
as.cimg(rep(1:100,3),x=10,y=10,cc=3) #10x10 RGB
as.cimg(1:100,dim=c(10,10,1,1))
as.cimg(1:100) #Guesses dimensions, warning is issued
as.cimg(rep(1:100,3)) #Guesses dimensions, warning is issued
```

---

as.cimg.array *Turn an numeric array into a cimg object*

Description

If the array has two dimensions, we assume it's a grayscale image. If it has three dimensions we assume it's a video, unless the third dimension has a depth of 3, in which case we assume it's a colour image,

Usage

```
## S3 method for class 'array'
as.cimg(obj, ...)
```

Arguments

obj	an array
...	ignored

Examples

```
as.cimg(array(1:9,c(3,3)))
as.cimg(array(1,c(10,10,3))) #Guesses colour image
as.cimg(array(1:9,c(10,10,4))) #Guesses video
```

`as.cimg.data.frame` *Create an image from a data.frame*

Description

This function is meant to be just like as.cimg.data.frame, but in reverse. Each line in the data frame must correspond to a pixel. For example, the data fame can be of the form (x,y,value) or (x,y,z,value), or (x,y,z,cc,value). The coordinates must be valid image coordinates (i.e., positive integers).

Usage

```
## S3 method for class 'data.frame'
as.cimg(obj, v.name = "value", dims, ...)
```

Arguments

`obj`	a data.frame
`v.name`	name of the variable to extract pixel values from (default "value")
`dims`	a vector of length 4 corresponding to image dimensions. If missing, a guess will be made.
`...`	ignored

Value

an object of class cimg

Author(s)

Simon Barthelme

Examples

```
#Create a data.frame with columns x,y and value
df <- expand.grid(x=1:10,y=1:10) %>% mutate(value=x*y)
#Convert to cimg object (2D, grayscale image of size 10*10
as.cimg(df,dims=c(10,10,1,1)) %>% plot
```

`as.cimg.function` *Create an image by sampling a function*

Description

Similar to as.im.function from the spatstat package, but simpler. Creates a grid of pixel coordinates x=1:width,y=1:height and (optional) z=1:depth, and evaluates the input function at these values.

Usage

```
## S3 method for class 'function'
as.cimg(obj, width, height, depth = 1, spectrum = 1,
  standardise = FALSE, dim = NULL, ...)
```

Arguments

`obj`	a function with arguments (x,y), or (x,y,cc), or (x,y,z), etc. Must be vectorised; see examples.
`width`	width of the image (in pixels)
`height`	height of the image (in pixels)
`depth`	depth of the image (in pixels). Default 1.
`spectrum`	number of colour channels. Defaut 1.
`standardise`	coordinates are scaled and centered (see doc for pixel.grid)
`dim`	a vector of image dimensions (can be used instead of width, height, etc.)
`...`	ignored

Value

an object of class cimg

Author(s)

Simon Barthelme

Examples

```
im = as.cimg(function(x,y) cos(sin(x*y/100)),100,100)
plot(im)
#The following is just a rectangle at the center of the image
im = as.cimg(function(x,y) (abs(x) < .1)*(abs(y) < .1) ,100,100,standardise=TRUE)
plot(im)
#Since coordinates are standardised the rectangle scales with the size of the image
im = as.cimg(function(x,y) (abs(x) < .1)*(abs(y) < .1) ,200,200,standardise=TRUE)
plot(im)
#A Gaussian mask around the center
im = as.cimg(function(x,y) dnorm(x,sd=.1)*dnorm(y,sd=.3) ,dim=dim(boats),standardise=TRUE)
```

```
im = im/max(im)

plot(im*boats)
#A Gaussian mask for just the red channel
fun = function(x,y,cc) ifelse(cc==1,dnorm(x,sd=.1)*dnorm(y,sd=.3),0)
im = as.cimg(fun,dim=dim(boats),standardise=TRUE)
plot(im*boats)
```

`as.data.frame.cimg` *Convert a pixel image to a data.frame*

Description

This function combines the output of pixel.grid with the actual values (stored in $value)

Usage

```
## S3 method for class 'cimg'
as.data.frame(x, ..., wide = c(FALSE, "c", "d"))
```

Arguments

`x`	an image of class cimg
`...`	arguments passed to pixel.grid
`wide`	if "c" or "d" return a data.frame that is wide along colour or depth (for example with rgb values along columns). The default is FALSE, with each pixel forming a seperate entry.

Value

a data.frame

Author(s)

Simon Barthelme

Examples

```
#First five pixels
as.data.frame(boats) %>% head(5)
#Wide format along colour axis
as.data.frame(boats,wide="c") %>% head(5)
```

as.data.frame.imlist *Convert image list to data.frame*

Description

Convert image list to data.frame

Usage

```
## S3 method for class 'imlist'
as.data.frame(x, ..., index = "im")
```

Arguments

x	an image list (an imlist object)
...	Passed on to as.data.frame.cimg
index	Name of the colum containing the index (or name) of the image in the list. Default: "im"

Examples

```
#Transform the image gradient into a data.frame
gr <- imgradient(boats,"xy") %>% setNames(c("dx","dy")) %>% as.data.frame
str(gr)
```

as.list.imlist *Convert image list to list*

Description

Convert image list to list

Usage

```
## S3 method for class 'imlist'
as.list(x, ...)
```

Arguments

x	an image list
...	ignored

Value

a list

`as.raster.cimg` *Convert a cimg object to a raster object for plotting*

Description

raster objects are used by R's base graphics for plotting. R wants hexadecimal RGB values for plotting, e.g. gray(0) yields #000000, meaning black. If you want to control precisely how numerical values are turned into colours for plotting, you need to specify a colour scale using the colourscale argument (see examples). Otherwise the default is "gray" for grayscale images, "rgb" for colour. These expect values in [0..1], so the default is to rescale the data to [0..1]. If you wish to over-ride that behaviour, set rescale=FALSE.

Usage

```
## S3 method for class 'cimg'
as.raster(x, frames, rescale = TRUE, colourscale = NULL,
  colorscale = NULL, ...)
```

Arguments

`x`	an image (of class cimg)
`frames`	which frames to extract (in case depth > 1)
`rescale`	rescale so that pixel values are in [0,1]? (subtract min and divide by range). default TRUE
`colourscale`	a function that returns RGB values in hexadecimal
`colorscale`	same as above in American spelling
`...`	ignored

Value

a raster object

Author(s)

Simon Barthelme

See Also

plot.cimg, rasterImage

Examples

```
#A raster is a simple array of RGB values
as.raster(boats) %>% str
#By default as.raster rescales input values, so that:
all.equal(as.raster(boats),as.raster(boats/2)) #TRUE
#Setting rescale to FALSE changes that
try(as.raster(boats,rescale=FALSE))
#The above fails because the pixel values are in the wrong range
boats <- boats/255 #Rescale to 0..1
as.raster(boats,rescale=FALSE) %>% plot
as.raster(boats/2,rescale=FALSE) %>% plot
#For grayscale images, a colourmap should take a single value and
#return  an RGB code
#Example: mapping grayscale value to saturation
cscale <- function(v) hsv(.5,v,1)
grayscale(boats) %>% as.raster(colourscale=cscale) %>% plot
```

---

at *Return or set pixel value at coordinates*

Description

Return or set pixel value at coordinates

Usage

```
at(im, x, y, z = 1, cc = 1)

at(im, x, y, z = 1, cc = 1) <- value

color.at(im, x, y, z = 1)

color.at(im, x, y, z = 1) <- value
```

Arguments

im	an image (cimg object)
x	x coordinate (vector)
y	y coordinate (vector)
z	z coordinate (vector, default 1)
cc	colour coordinate (vector, default 1)
value	replacement

Value

pixel values

Functions

- `at<-`: set value of pixel at a location
- `color.at`: return value of all colour channels at a location
- `color.at<-`: set value of all colour channels at a location

Author(s)

Simon Barthelme

Examples

```
im <- as.cimg(function(x,y) x+y,50,50)
at(im,10,1)
at(im,10:12,1)
at(im,10:12,1:3)
at(im,1,2) <- 10
at(im,1,2)
color.at(boats,x=10,y=10)
im <- boats
color.at(im,x=10,y=10) <- c(255,0,0)
#There should now be a red dot
imsub(im, x %inr% c(1,100), y %inr% c(1,100)) %>% plot
```

---

autocrop *Autocrop image region*

Description

Autocrop image region

Usage

```
autocrop(im, color = c(0, 0, 0), axes = "zyx")
```

Arguments

`im`	an image
`color`	Color used for the crop. If 0, color is guessed.
`axes`	Axes used for the crop.

Examples

```
#Add pointless padding
padded <- pad(boats,30,"xy")
plot(padded)
#Remove padding
autocrop(padded,color=c(0,0,0)) %>% plot
```

`blur_anisotropic` *Blur image anisotropically, in an edge-preserving way.*

Description

Standard blurring removes noise from images, but tends to smooth away edges in the process. This anisotropic filter preserves edges better.

Usage

```
blur_anisotropic(im, amplitude, sharpness = 0.7, anisotropy = 0.6,
  alpha = 0.6, sigma = 1.1, dl = 0.8, da = 30, gauss_prec = 2,
  interpolation_type = 0L, fast_approx = TRUE)
```

Arguments

`im`	an image
`amplitude`	Amplitude of the smoothing.
`sharpness`	Sharpness.
`anisotropy`	Anisotropy.
`alpha`	Standard deviation of the gradient blur.
`sigma`	Standard deviation of the structure tensor blur.
`dl`	Spatial discretization.
`da`	Angular discretization.
`gauss_prec`	Precision of the diffusion process.
`interpolation_type`	Interpolation scheme. Can be 0=nearest-neighbor | 1=linear | 2=Runge-Kutta
`fast_approx`	If true, use fast approximation (default TRUE)

Examples

```
im <- load.image(system.file('extdata/Leonardo_Birds.jpg',package='imager'))
im.noisy <- (im + 80*rnorm(prod(dim(im))))
blur_anisotropic(im.noisy,ampl=1e4,sharp=1) %>% plot
```

boats *Photograph of sailing boats from Kodak set*

Description

This photograph was downloaded from http://r0k.us/graphics/kodak/kodim09.html. Its size was reduced by half to speed up loading and save space.

Usage

```
boats
```

Format

an image of class cimg

Source

http://r0k.us/graphics/kodak/kodim09.html

boxblur *Blur image with a box filter (square window)*

Description

Blur image with a box filter (square window)

Usage

```
boxblur(im, sigma, neumann = TRUE)
```

Arguments

im	an image
sigma	Size of the box window.
neumann	If true, use Neumann boundary conditions, Dirichlet otherwise (default true, Neumann)

See Also

deriche(), vanvliet().

Examples

```
boxblur(boats,5) %>% plot(main="Dirichlet boundary")
boxblur(boats,5,TRUE) %>% plot(main="Neumann boundary")
```

boxblur_xy *Blur image with a box filter.*

Description

This is a recursive algorithm, not depending on the values of the box kernel size.

Usage

```
boxblur_xy(im, sx, sy, neumann = TRUE)
```

Arguments

im	an image
sx	Size of the box window, along the X-axis.
sy	Size of the box window, along the Y-axis.
neumann	If true, use Neumann boundary conditions, Dirichlet otherwise (default true, Neumann)

See Also

blur().

Examples

```
boxblur_xy(boats,20,5) %>% plot(main="Anisotropic blur")
```

bucketfill *Bucket fill*

Description

Bucket fill

Usage

```
bucketfill(im, x, y, z = 1, color, opacity = 1, sigma = 0,
  high_connexity = FALSE)
```

Arguments

im	an image
x	X-coordinate of the starting point of the region to fill.
y	Y-coordinate of the starting point of the region to fill.
z	Z-coordinate of the starting point of the region to fill.
color	Pointer to spectrum() consecutive values, defining the drawing color. If missing, use value at location (x,y,z)
opacity	Opacity of the drawing.
sigma	Tolerance concerning neighborhood values.
high_connexity	Use 8-connexity (only for 2d images, default FALSE).

See Also

bucket_select

Examples

```
#Change the colour of a sail
boats.new <- bucketfill(boats,x=169,y=179,color=c(125,0,125),sigma=20)
layout(t(1:2))
plot(boats,main="Original")
plot(boats.new,main="New sails")
```

---

capture.plot *Capture the current R plot device as a cimg image*

Description

Capture the current R plot device as a cimg image

Usage

```
capture.plot()
```

Value

a cimg image corresponding to the contents of the current plotting window

Author(s)

Simon Barthelme

Examples

```
##interactive only:
##plot(1:10)
###Make a plot of the plot
##capture.plot() %>% plot
```

center.stencil *Center stencil at a location*

Description

Center stencil at a location

Usage

```
center.stencil(stencil, ...)
```

Arguments

stencil	a stencil (data.frame with coordinates dx,dy,dz,dc)
...	centering locations (e.g. x=4,y=2)

Examples

```
stencil <- data.frame(dx=seq(-2,2,1),dy=seq(-2,2,1))
center.stencil(stencil,x=10,y=20)
```

channels *Split a colour image into a list of separate channels*

Description

Split a colour image into a list of separate channels

Usage

```
channels(im, index, drop = FALSE)
```

Arguments

im	an image
index	which channels to extract (default all)
drop	if TRUE drop extra dimensions, returning normal arrays and not cimg objects

Value

a list of channels

See Also

frames

Examples

```
channels(boats)
channels(boats,1:2)
channels(boats,1:2,drop=TRUE) %>% str #A list of 2D arrays
```

`cimg` *Create a cimg object*

Description

cimg is a class for storing image or video/hyperspectral data. It is designed to provide easy interaction with the CImg library, but in order to use it you need to be aware of how CImg wants its image data stored. Images have up to 4 dimensions, labelled x,y,z,c. x and y are the usual spatial dimensions, z is a depth dimension (which would correspond to time in a movie), and c is a colour dimension. Images are stored linearly in that order, starting from the top-left pixel and going along *rows* (scanline order). A colour image is just three R,G,B channels in succession. A sequence of N images is encoded as R1,R2,....,RN,G1,...,GN,B1,...,BN where R_i is the red channel of frame i. The number of pixels along the x,y,z, and c axes is called (in that order), width, height, depth and spectrum. NB: Logical and integer values are automatically converted to type double. NAs are not supported by CImg, so you should manage them on the R end of things.

Usage

```
cimg(X)
```

Arguments

`X`	a four-dimensional numeric array

Value

an object of class cimg

Author(s)

Simon Barthelme

Examples

```
cimg(array(1,c(10,10,5,3)))
```

cimg.dimensions *Image dimensions*

Description

Image dimensions

Usage

```
width(im)

height(im)

spectrum(im)

depth(im)

nPix(im)
```

Arguments

im	an image

Functions

- width: Width of the image (in pixels)
- height: Height of the image (in pixels)
- spectrum: Number of colour channels
- depth: Depth of the image/number of frames in a video
- nPix: Total number of pixels (prod(dim(im)))

cimg.extract *Various shortcuts for extracting colour channels, frames, etc*

Description

Various shortcuts for extracting colour channels, frames, etc

Extract one frame out of a 4D image/video

Usage

```
frame(im, index)

imcol(im, x)

imrow(im, y)

channel(im, ind)

R(im)

G(im)

B(im)
```

Arguments

im	an image
index	frame index
x	x coordinate of the row
y	y coordinate of the row
ind	channel index

Functions

- frame: Extract frame
- imcol: Extract a particular column from an image
- imrow: Extract a particular row from an image
- channel: Extract an image channel
- R: Extract red channel
- G: Extract green channel
- B: Extract blue channel

Author(s)

Simon Barthelme

Examples

```
#Extract the red channel from the boats image, then the first row, plot
rw <- R(boats) %>% imrow(10)
plot(rw,type="l",xlab="x",ylab="Pixel value")
#Note that R(boats) returns an image
R(boats)
#while imrow returns a vector or a list
R(boats) %>% imrow(1) %>% str
imrow(boats,1) %>% str
```

cimg.use.openmp *Control CImg's parallelisation*

Description

On supported architectures CImg can parallelise many operations using OpenMP. Use this function to turn parallelisation on or off.

Usage

```
cimg.use.openmp(mode = "adaptive")
```

Arguments

mode	Either "adaptive","always" or "none". The default is adaptive (parallelisation for large images only).

Value

NULL (function is used for side effects)

Author(s)

Simon Barthelme

Examples

```
cimg.use.openmp("never") #turn off parallelisation
```

cimg2im *Convert cimg to spatstat im object*

Description

The spatstat library uses a different format for images, which have class "im". This utility converts a cimg object to an im object. spatstat im objects are limited to 2D grayscale images, so if the image has depth or spectrum > 1 a list is returned for the separate frames or channels (or both, in which case a list of lists is returned, with frames at the higher level and channels at the lower one).

Usage

```
cimg2im(img, W = NULL)
```

Arguments

img	an image of class cimg
W	a spatial window (see spatstat doc). Default NULL

Value

an object of class im, or a list of objects of class im, or a list of lists of objects of class im

Author(s)

Simon Barthelme

See Also

im, as.im

---

convolve *Convolve image by filter.*

Description

The result res of the convolution of an image img by filter flt is defined to be: $res(x, y, z) = sum_{i,j,k} img(x - i, y - j, z - k) * flt(i, j, k)$

Usage

```
convolve(im, filter, dirichlet = FALSE, normalise = FALSE)
```

Arguments

im	an image
filter	a filter (another image)
dirichlet	boundary condition (FALSE=zero padding, TRUE=dirichlet). Default FALSE
normalise	normalise filter (default FALSE)

See Also

correlate

Examples

```
#Edge filter
filter <- as.cimg(function(x,y) sign(x-5),10,10)
layout(t(1:2))
#Convolution vs. correlation
correlate(boats,filter) %>% plot(main="Correlation")
convolve(boats,filter) %>% plot(main="Convolution")
```

coord.index *Coordinates from pixel index*

Description

Compute (x,y,z,cc) coordinates from linear pixel index.

Usage

```
coord.index(im, index)
```

Arguments

im	an image
index	a vector of indices

Value

a data.frame of coordinate values

Author(s)

Simon Barthelme

See Also

index.coord for the reverse operation

Examples

```
cind <- coord.index(boats,33)
#Returns (x,y,z,c) coordinates of the 33rd pixel in the array
cind
all.equal(boats[33],with(cind,at(boats,x,y,z,cc)))
all.equal(33,index.coord(boats,cind))
```

correlate *Correlation of image by filter*

Description

The correlation of image im by filter flt is defined as: $res(x,y,z) = sum_{i,j,k} im(x+i, y+j, z+k) * flt(i,j,k)$.

Usage

```
correlate(im, filter, dirichlet = FALSE, normalise = FALSE)
```

Arguments

im	an image
filter	the correlation kernel.
dirichlet	boundary condition (FALSE=zero padding, TRUE=dirichlet). Default FALSE
normalise	normalise filter (default FALSE)

Examples

```
#Edge filter
filter <- as.cimg(function(x,y) sign(x-5),10,10)
layout(t(1:2))
#Convolution vs. correlation
correlate(boats,filter) %>% plot(main="Correlation")
convolve(boats,filter) %>% plot(main="Convolution")
```

---

crop.borders *Crop the outer margins of an image*

Description

This function crops pixels on each side of an image. This function is a kind of inverse (centred) padding, and is useful e.g. when you want to get only the valid part of a convolution

Usage

```
crop.borders(im, nx = 0, ny = 0, nz = 0, nPix)
```

Arguments

im	an image
nx	number of pixels to crop along horizontal axis
ny	number of pixels to crop along vertical axis
nz	number of pixels to crop along depth axis
nPix	optional: crop the same number of pixels along all dimensions

Value

an image

Author(s)

Simon Barthelme

Examples

```
#These two versions are equivalent
imfill(10,10) %>% crop.borders(nx=1,ny=1)
imfill(10,10) %>% crop.borders(nPix=1)

#Filter, keep valid part
correlate(boats,imfill(3,3)) %>% crop.borders(nPix=2)
```

`deriche` *Apply recursive Deriche filter.*

Description

The Deriche filter is a fast approximation to a Gaussian filter (order = 0), or Gaussian derivatives (order = 1 or 2).

Usage

```
deriche(im, sigma, order = 0L, axis = "x", neumann = FALSE)
```

Arguments

`im`	an image
`sigma`	Standard deviation of the filter.
`order`	Order of the filter. 0 for a smoothing filter, 1 for first-derivative, 2 for second.
`axis`	Axis along which the filter is computed ('x' , 'y', 'z' or 'c').
`neumann`	If true, use Neumann boundary conditions (default false, Dirichlet)

Examples

```
deriche(boats,sigma=2,order=0) %>% plot("Zeroth-order Deriche along x")
deriche(boats,sigma=2,order=1) %>% plot("First-order Deriche along x")
deriche(boats,sigma=2,order=1) %>% plot("Second-order Deriche along x")
deriche(boats,sigma=2,order=1,axis="y") %>% plot("Second-order Deriche along y")
```

diffusion_tensors *Compute field of diffusion tensors for edge-preserving smoothing.*

Description

Compute field of diffusion tensors for edge-preserving smoothing.

Usage

```
diffusion_tensors(im, sharpness = 0.7, anisotropy = 0.6, alpha = 0.6,
  sigma = 1.1, is_sqrt = FALSE)
```

Arguments

im	an image
sharpness	Sharpness
anisotropy	Anisotropy
alpha	Standard deviation of the gradient blur.
sigma	Standard deviation of the structure tensor blur.
is_sqrt	Tells if the square root of the tensor field is computed instead.

displacement *Estimate displacement field between two images.*

Description

Estimate displacement field between two images.

Usage

```
displacement(sourceIm, destIm, smoothness = 0.1, precision = 5,
  nb_scales = 0L, iteration_max = 10000L, is_backward = FALSE)
```

Arguments

sourceIm	Reference image.
destIm	Reference image.
smoothness	Smoothness of estimated displacement field.
precision	Precision required for algorithm convergence.
nb_scales	Number of scales used to estimate the displacement field.
iteration_max	Maximum number of iterations allowed for one scale.
is_backward	If false, match I2(X + U(X)) = I1(X), else match I2(X) = I1(X - U(X)).

display *Display object using CImg library*

Description

CImg has its own functions for fast, interactive image plotting. Use this if you get frustrated with slow rendering in Rstudio.

Usage

```
display(x, ...)
```

Arguments

x	an image or a list of images
...	ignored

See Also

display.cimg, display.imlist

display.cimg *Display image using CImg library*

Description

Press escape or close the window to exit.

Usage

```
## S3 method for class 'cimg'
display(x, ..., rescale = TRUE)
```

Arguments

x	an image (cimg object)
...	ignored
rescale	if true pixel values are rescaled to 0...255 (default TRUE)

Examples

```
##Not run: interactive only
##display(boats,TRUE) #Normalisation on
##display(boats/2,TRUE) #Normalisation on, so same as above
##display(boats,FALSE) #Normalisation off
##display(boats/2,FALSE) #Normalisation off, so different from above
```

display.list *Display image list using CImg library*

Description

Click on individual images to zoom in.

Usage

```
## S3 method for class 'list'
display(x, ...)
```

Arguments

x	a list of cimg objects
...	ignored

Examples

```
##Not run: interactive only
## imgradient(boats,"xy") %>% display
```

distance_transform *Compute Euclidean distance function to a specified value.*

Description

The distance transform implementation has been submitted by A. Meijster, and implements the article 'W.H. Hesselink, A. Meijster, J.B.T.M. Roerdink, "A general algorithm for computing distance transforms in linear time.", In: Mathematical Morphology and its Applications to Image and Signal Processing, J. Goutsias, L. Vincent, and D.S. Bloomberg (eds.), Kluwer, 2000, pp. 331-340.' The submitted code has then been modified to fit CImg coding style and constraints.

Usage

```
distance_transform(im, value, metric = 2L)
```

Arguments

im	an image
value	Reference value.
metric	Type of metric. Can be <tt> 0=Chebyshev | 1=Manhattan | 2=Euclidean | 3=Squared-euclidean </tt>.

Examples

```
imd <- function(x,y) imdirac(c(100,100,1,1),x,y)
#Image is three white dots
im <- imd(20,20)+imd(40,40)+imd(80,80)
plot(im)
#How far are we from the nearest white dot?
distance_transform(im,1) %>% plot
```

---

erode *Erode/dilate image by a structuring element.*

Description

Erode/dilate image by a structuring element.

Usage

```
erode(im, mask, boundary_conditions = TRUE, normalise = FALSE)

erode_rect(im, sx, sy, sz = 1L)

erode_square(im, size)

dilate(im, mask, boundary_conditions = TRUE, normalise = FALSE)

dilate_rect(im, sx, sy, sz = 1L)

dilate_square(im, size)

mopening(im, mask, boundary_conditions = TRUE, normalise = FALSE)

mopening_square(im, size)

mclosing_square(im, size)

mclosing(im, mask, boundary_conditions = TRUE, normalise = FALSE)
```

Arguments

im	an image
mask	Structuring element.
boundary_conditions	Boundary conditions.
normalise	Determines if the closing is locally normalised (default FALSE)
sx	Width of the structuring element.
sy	Height of the structuring element.

sz	Depth of the structuring element.
size	size of the structuring element.

Functions

- erode_rect: Erode image by a rectangular structuring element of specified size.
- erode_square: Erode image by a square structuring element of specified size.
- dilate: Dilate image by a structuring element.
- dilate_rect: Dilate image by a rectangular structuring element of specified size
- dilate_square: Dilate image by a square structuring element of specified size
- mopening: Morphological opening (erosion followed by dilation)
- mopening_square: Morphological opening by a square element (erosion followed by dilation)
- mclosing_square: Morphological closing by a square element (dilation followed by erosion)
- mclosing: Morphological closing (dilation followed by erosion)

Examples

```
fname <- system.file('extdata/Leonardo_Birds.jpg',package='imager')
im <- load.image(fname) %>% grayscale
outline <- threshold(-im,"95%")
plot(outline)
mask <- imfill(5,10,val=1) #Rectangular mask
plot(erode(outline,mask))
plot(erode_rect(outline,5,10)) #Same thing
plot(erode_square(outline,5))
plot(dilate(outline,mask))
plot(dilate_rect(outline,5,10))
plot(dilate_square(outline,5))
```

---

extract_patches *Extract image patches and return a list*

Description

Patches are rectangular (cubic) image regions centered at cx,cy (cz) with width wx and height wy (opt. depth wz) WARNINGS: - values outside of the image region are considered to be 0. - widths and heights should be odd integers (they're rounded up otherwise).

Usage

```
extract_patches(im, cx, cy, wx, wy)

extract_patches3D(im, cx, cy, cz, wx, wy, wz)
```

Arguments

im	an image
cx	vector of x coordinates for patch centers
cy	vector of y coordinates for patch centers
wx	vector of patch widths (or single value)
wy	vector of patch heights (or single value)
cz	vector of z coordinates for patch centers
wz	vector of coordinates for patch depth

Value

a list of image patches (cimg objects)

Functions

- extract_patches3D: Extract 3D patches

Examples

```
#2 patches of size 5x5 located at (10,10) and (10,20)
extract_patches(boats,c(10,10),c(10,20),5,5)
```

FFT *Compute the Discrete Fourier Transform of an image*

Description

This function is equivalent to R's builtin fft, up to normalisation (R's version is unnormalised, this one is). It calls CImg's implementation. Important note: FFT will compute a multidimensional Fast Fourier Transform, using as many dimensions as you have in the image, meaning that if you have a colour video, it will perform a 4D FFT. If you want to compute separate FFTs across channels, use imsplit.

Usage

```
FFT(im.real, im.imag, inverse = FALSE)
```

Arguments

im.real	The real part of the input (an image)
im.imag	The imaginary part (also an image. If missing, assume the signal is real).
inverse	If true compute the inverse FFT (default: FALSE)

Value

a list with components "real" (an image) and "imag" (an image), corresponding to the real and imaginary parts of the transform

Author(s)

Simon Barthelme

Examples

```
im <- as.cimg(function(x,y) sin(x/5)+cos(x/4)*sin(y/2),128,128)
ff <- FFT(im)
plot(ff$real,main="Real part of the transform")
plot(ff$imag,main="Imaginary part of the transform")
sqrt(ff$real^2+ff$imag^2) %>% plot(main="Power spectrum")
#Check that we do get our image back
check <- FFT(ff$real,ff$imag,inverse=TRUE)$real #Should be the same as original
mean((check-im)^2)
```

---

frames *Split a video into separate frames*

Description

Split a video into separate frames

Usage

```
frames(im, index, drop = FALSE)
```

Arguments

im	an image
index	which channels to extract (default all)
drop	if TRUE drop extra dimensions, returning normal arrays and not cimg objects

Value

a list of frames

See Also

channels

get.locations *Return coordinates of subset of pixels*

Description

Typical use case: you want the coordinates of all pixels with a value above a certain threshold

Usage

```
get.locations(im, condition)
```

Arguments

im	the image
condition	a function that takes scalars and returns logicals

Value

coordinates of all pixels such that condition(pixel) == TRUE

Author(s)

Simon Barthelme

Examples

```
im <- as.cimg(function(x,y) x+y,10,10)
get.locations(im,function(v) v < 4)
get.locations(im,function(v) v^2 + 3*v - 2 < 30)
```

get.stencil *Return pixel values in a neighbourhood defined by a stencil*

Description

A stencil defines a neighbourhood in an image (for example, the four nearest neighbours in a 2d image). This function centers the stencil at a certain pixel and returns the values of the neighbourhing pixels.

Usage

```
get.stencil(im, stencil, ...)
```

Arguments

im	an image
stencil	a data.frame with values dx,dy,[dz],[dcc] defining the neighbourhood
...	where to center, e.g. x = 100,y = 10,z=3,cc=1

Value

pixel values in neighbourhood

Author(s)

Simon Barthelme

Examples

```
#The following stencil defines a neighbourhood that
#includes the next pixel to the left (delta_x = -1) and the next pixel to the right (delta_x = 1)
stencil <- data.frame(dx=c(-1,1),dy=c(0,0))
im <- as.cimg(function(x,y) x+y,w=100,h=100)
get.stencil(im,stencil,x=50,y=50)

#A larger neighbourhood that includes pixels upwards and
#downwards of center (delta_y = -1 and +1)
stencil <- stencil.cross()
im <- as.cimg(function(x,y) x,w=100,h=100)
get.stencil(im,stencil,x=5,y=50)
```

---

get_gradient *Compute image gradient.*

Description

Compute image gradient.

Usage

```
get_gradient(im, axes = "", scheme = 3L)
```

Arguments

im	an image
axes	Axes considered for the gradient computation, as a C-string (e.g "xy").
scheme	= Numerical scheme used for the gradient computation: 1 = Backward finite differences 0 = Centered finite differences 1 = Forward finite differences 2 = Using Sobel masks 3 = Using rotation invariant masks 4 = Using Deriche recursive filter. 5 = Using Van Vliet recursive filter.

Value

a list of images (corresponding to the different directions)

See Also

imgradient

`get_hessian` *Return image hessian.*

Description

Return image hessian.

Usage

```
get_hessian(im, axes = "")
```

Arguments

`im`	an image
`axes`	Axes considered for the hessian computation, as a character string (e.g "xy").

`grab` *Select image regions interactively*

Description

These functions let you select a shape in an image (a point, a line, or a rectangle) They either return the coordinates of the shape (default), or the contents. In case of lines contents are interpolated.

Usage

```
grabLine(im, coord = TRUE)

grabRect(im, coord = TRUE)

grabPoint(im, coord = TRUE)
```

Arguments

`im`	an image
`coord`	if TRUE, return coordinates. if FALSE, content

Value

either a vector of coordinates, or an image

Author(s)

Simon Barthelme

See Also

display

Examples

```
##Not run: interactive only
##grabRect(boats)
##grabRect(boats,TRUE)
```

---

grayscale *Convert an RGB image to grayscale*

Description

This function converts from RGB to grayscale by first converting to HSL and keeping only the L channel

Usage

```
grayscale(im)
```

Arguments

im	an RGB image

Value

a grayscale image (spectrum == 1)

haar *Compute Haar multiscale wavelet transform.*

Description

Compute Haar multiscale wavelet transform.

Usage

```
haar(im, inverse = FALSE, nb_scales = 1L)
```

Arguments

im	an image
inverse	Compute inverse transform (default FALSE)
nb_scales	Number of scales used for the transform.

Examples

```
#Image compression: set small Haar coefficients to 0
hr <- haar(boats,nb=3)
mask.low <- threshold(abs(hr),"75%")
mask.high <- threshold(abs(hr),"95%")
haar(hr*mask.low,inverse=TRUE,nb=3) %>% plot(main="75% compression")
haar(hr*mask.high,inverse=TRUE,nb=3) %>% plot(main="95% compression")
```

idply *Split an image along axis, apply function, return a data.frame*

Description

Shorthand for imsplit followed by ldply

Usage

```
idply(im, axis, fun, ...)
```

Arguments

im	image
axis	axis for the split (e.g "c")
fun	function to apply
...	extra arguments to function fun

Examples

```
idply(boats,"c",mean) #mean luminance per colour channel
```

iiply *Split an image, apply function, recombine the results as an image*

Description

This is just imsplit followed by llply followed by imappend

Usage

```
iiply(im, axis, fun, ...)
```

Arguments

im	image
axis	axis for the split (e.g "c")
fun	function to apply
...	extra arguments to function fun

Examples

```
##' #Normalise colour channels separately, recombine
iiply(boats,"c",function(v) (v-mean(v))/sd(v)) %>% plot
```

ilply *Split an image along axis, apply function, return a list*

Description

Shorthand for imsplit followed by llply

Usage

```
ilply(im, axis, fun, ...)
```

Arguments

im	image
axis	axis for the split (e.g "c")
fun	function to apply
...	extra arguments for function fun

Examples

```
parrots <- load.example("parrots")
ilply(parrots,"c",mean) #mean luminance per colour channel
```

im2cimg *Convert an image in spatstat format to an image in cimg format*

Description

as.cimg.im is an alias for the same function

Usage

```
im2cimg(img)
```

Arguments

img	a spatstat image

Value

a cimg image

Author(s)

Simon Barthelme

imager *imager: an R library for image processing, based on CImg*

Description

CImg by David Tschumperle is a C++ library for image processing. It provides most common functions for image manipulation and filtering, as well as some advanced algorithms. imager makes these functions accessible from R and adds many utilities for accessing and working with image data from R. You should install ImageMagick if you want support for image formats beyond PNG and JPEG, and ffmpeg if you need to work with videos (in which case you probably also want to take a look at experimental package imagerstreams on github). Package documentation is available at http://dahtah.github.io/imager/.

`imager.combine` *Combining images*

Description

These functions take a list of images and combine them by adding, multiplying, taking the parallel min or max, etc. The max. in absolute value of (x1,x2) is defined as x1 if (|x1| > |x2|), x2 otherwise. It's useful for example in getting the most extreme value while keeping the sign.

Usage

```
add(x)

average(x)

mult(x)

parmax(x)

parmax.abs(x)

parmin.abs(x)

parmin(x)

enorm(x)

which.parmax(x)

which.parmin(x)
```

Arguments

`x`	a list of images

Functions

- `add`: Add images
- `average`: Average images
- `mult`: Multiply images (pointwise)
- `parmax`: Parallel max over images
- `parmax.abs`: Parallel max in absolute value over images,
- `parmin.abs`: Parallel max in absolute value over images,
- `parmin`: Parallel min over images
- `enorm`: Euclidean norm (i.e. sqrt(A^2 + B^2 + ...))

- `which.parmax`: index of parallel maxima
- `which.parmin`: index of parallel minima

Author(s)

Simon Barthelme

See Also

imsplit,Reduce

Examples

```
im1 <- as.cimg(function(x,y) x,100,100)
im2 <- as.cimg(function(x,y) y,100,100)
im3 <- as.cimg(function(x,y) cos(x/10),100,100)
l <- list(im1,im2,im3)
add(l) %>% plot #Add the images
average(l) %>% plot #Average the images
mult(l) %>% plot #Multiply
parmax(l) %>% plot #Parallel max
parmin(l) %>% plot #Parallel min
#Edge detection
imgradient(boats,"xy") %>% enorm %>% plot
#Pseudo-artistic effects
llply(seq(1,35,5),function(v) boxblur(boats,v)) %>% parmin %>% plot
llply(seq(1,35,5),function(v) boxblur(boats,v)) %>% average %>% plot

#At each pixel, which colour channel has the maximum value?
imsplit(boats,"c") %>% which.parmax %>% table
```

---

imager.replace *Replace part of an image with another*

Description

These replacement functions let you modify part of an image (for example, only the red channel). Note that cimg objects can also be treated as regular arrays and modified using the usual [] operator.

Usage

```
channel(x, ind) <- value

R(x) <- value

G(x) <- value

B(x) <- value
```

```
frame(x, ind) <- value
```

Arguments

x	an image to be modified
ind	an index
value	the image to insert

Functions

- channel<-: Replace image channel
- R<-: Replace red channel
- G<-: Replace green channel
- B<-: Replace blue channel
- frame<-: Replace image frame

See Also

imdraw

Examples

```
boats.cp <- boats
#Set the green channel in the boats image to 0
G(boats.cp) <- 0
#Same thing, more verbose
channel(boats.cp,2) <- 0
#Replace the red channel with noise
R(boats.cp) <- imnoise(width(boats),height(boats))
#A new image with 5 frames
tmp <- imfill(10,10,5)
#Fill the third frame with noise
frame(tmp,3) <- imnoise(10,10)
```

---

imager.subset *Array subset operator for cimg objects*

Description

Internally cimg objects are 4D arrays (stored in x,y,z,c mode) but often one doesn't need all dimensions. This is the case for instance when working on grayscale images, which use only two. The array subset operator works like the regular array [] operator, but it won't force you to use all dimensions. There are easier ways of accessing image data, for example imsub, channels, R, G, B, and the like.

Arguments

x	an image (cimg object)
drop	if true return an array, otherwise return an image object (default FALSE)
...	subsetting arguments

See Also

imsub, which provides a more convenient interface, autocrop, imdraw

Examples

```
im <- imfill(4,4)
dim(im) #4 dimensional, but the last two ones are singletons
im[,1,,] <- 1:4 #Assignment the standard way
im[,1] <- 1:4 #Shortcut
as.matrix(im)
im[1:2,]
dim(boats)
#Arguments will be recycled, as in normal array operations
boats[1:2,1:3,] <- imnoise(2,3) #The same noise array is replicated over the three channels
```

---

imappend *Combine a list of images into a single image*

Description

All images will be concatenated along the x,y,z, or c axis.

Usage

```
imappend(imlist, axis)
```

Arguments

imlist	a list of images (all elements must be of class cimg)
axis	the axis along which to concatenate (for example 'c')

See Also

imsplit (the reverse operation)

Examples

```
imappend(list(boats,boats),"x") %>% plot
imappend(list(boats,boats),"y") %>% plot
plyr::rlply(3,imnoise(100,100)) %>% imappend("c") %>% plot
boats.gs <- grayscale(boats)
plyr::llply(seq(1,5,l=3),function(v) isoblur(boats.gs,v)) %>% imappend("c") %>% plot
```

imcoord *Coordinates as images*

Description

These functions return pixel coordinates for an image, as an image. All is made clear in the examples (hopefully)

Usage

```
Xc(im)

Yc(im)

Zc(im)

Cc(im)
```

Arguments

im	an image

Value

another image of the same size, containing pixel coordinates

Functions

- Xc: X coordinates
- Yc: Y coordinates
- Zc: Z coordinates
- Cc: C coordinates

See Also

as.cimg.function, pixel.grid

Examples

```
im <- imfill(5,5) #An image
Xc(im) #An image of the same size, containing the x coordinates of each pixel
Xc(im) %>% imrow(1)
Yc(im) %>% imrow(3) #y is constant along rows
Yc(im) %>% imcol(1)
#Mask bits of the boats image:
plot(boats*(Xc(boats) < 100))
plot(boats*(dnorm(Xc(boats),m=100,sd=30))) #Gaussian window
```

imdirac *Generates a "dirac" image, i.e. with all values set to 0 except one.*

Description

This small utility is useful to examine the impulse response of a filter

Usage

```
imdirac(dims, x, y, z = 1, cc = 1)
```

Arguments

dims	a vector of image dimensions, or an image whose dimensions will be used. If dims has length < 4 some guesswork will be used (see examples and ?as.cimg.array)
x	where to put the dirac (x coordinate)
y	y coordinate
z	z coordinate (default 1)
cc	colour coordinate (default 1)

Value

an image

Author(s)

Simon Barthelme

Examples

```
#Explicit settings of all dimensions
imdirac(c(50,50,1,1),20,20)
imdirac(c(50,50),20,20) #Implicit
imdirac(c(50,50,3),20,20,cc=2) #RGB
imdirac(c(50,50,7),20,20,z=2) #50x50 video with 7 frames
#Impulse response of the blur filter
imdirac(c(50,50),20,20) %>% isoblur(sigma=2)  %>% plot
#Impulse response of the first-order Deriche filter
imdirac(c(50,50),20,20) %>% deriche(sigma=2,order=1,axis="x")  %>% plot
##NOT RUN, interactive only
##Impulse response of the blur filter in space-time
##resp <- imdirac(c(50,50,100),x=25,y=25,z=50)  %>%  isoblur(16)
###Normalise to 0...255 and play as video
##renorm(resp) %>% play(normalise=FALSE)
```

imdraw *Draw image on another image*

Description

Draw image on another image

Usage

```
imdraw(im, sprite, x = 1, y = 1, z = 1, opacity = 1)
```

Arguments

im	background image
sprite	sprite to draw on background image
x	location
y	location
z	location
opacity	transparency level (default 1)

Author(s)

Simon Barthelme

See Also

imager.combine, for different ways of combining images

Examples

```
im <- load.example("parrots")
boats.small <- imresize(boats,.5)
#I'm aware the result is somewhat ugly
imdraw(im,boats.small,x=400,y=10,opacity=.7) %>% plot
```

imeval *Pixel-wise evaluation of a CImg expression*

Description

This function provides experimental support for CImg's "math expression parser", a byte-compiled mini-language.

Usage

```
imeval(im, expr)
```

Arguments

im	an image
expr	an expression (as string)

Examples

```
imfill(10,10) %>% imeval('x+y') %>% plot
# Box filter
boxf = "v=0;for(iy=y-3,iy<y+3,iy++,for(ix=x-3,ix< x+3,ix++,v+=i(ix,iy)));v"
imeval(boats,boxf) %>% plot
# Example by D. Tschumperle: Julia set
julia <-  "
   zr = -1.2 + 2.4*x/w;
   zi = -1.2 + 2.4*y/h;
   for (iter = 0, zr^2+zi^2<=4 && iter<256, iter++,
     t = zr^2 - zi^2 + 0.5;
     (zi *= 2*zr) += 0.2;
     zr = t
   );
   iter"
imfill(500,500) %>% imeval(julia) %>% plot
```

imfill *Create an image of custom size by filling in repeated values*

Description

This is a convenience function for quickly creating blank images, or images filled with a specific colour. See examples.

Usage

```
imfill(x = 1, y = 1, z = 1, val = 0, dim = NULL)
```

Arguments

x	width (default 1)
y	height (default 1)
z	depth (default 1)
val	fill-in values. Either a single value (for grayscale), or RGB values for colour
dim	dimension vector (optional, alternative to specifying x,y,z)

Value

an image object (class cimg)

Author(s)

Simon Barthelme

Examples

```
imfill(20,20) %>% plot #Blank image of size 20x20
imfill(20,20,val=c(1,0,0)) %>% plot #All red image
imfill(dim=dim(boats)) #Blank image of the same size as the boats image
```

---

imgradient *Compute image gradient*

Description

Light interface for get_gradient. Refer to get_gradient for details on the computation.

Usage

```
imgradient(im, axes, scheme = 3)
```

Arguments

im	an image of class cimg
axes	direction along which to compute the gradient. Either a single character (e.g. "x"), or multiple characters (e.g. "xyz")
scheme	numerical scheme (default '3')

Value

an image or a list of images, depending on the value of "axes"

Author(s)

Simon Barthelme

Examples

```
grayscale(boats) %>% imgradient("x") %>% plot
imgradient(boats,"xy") #Returns a list
```

`imhessian` *Compute image hessian.*

Description

Compute image hessian.

Usage

```
imhessian(im, axes = c("xx", "xy", "yy"))
```

Arguments

`im`	an image
`axes`	Axes considered for the hessian computation, as a character string (e.g "xy" corresponds to d/(dx*dy)). Can be a list of axes. Default: xx,xy,yy

Value

an image, or a list of images

Examples

```
imhessian(boats,"xy") %>% plot(main="Second-derivative, d/(dx*dy)")
```

`iminfo` *Return information on image file*

Description

This function calls ImageMagick's "identify" utility on an image file to get some information. You need ImageMagick on your path for this to work.

Usage

```
iminfo(fname)
```

Arguments

`fname`	path to a file

Value

a list with fields name, format, width (pix.), height (pix.), size (bytes)

Author(s)

Simon Barthelme

Examples

```
## Not run:
someFiles <- dir("*.png") #Find all PNGs in directory
iminfo(someFiles[1])
#Get info on all files, as a data frame
info <- plyr::ldply(someFiles,function(v) iminfo(v) %>% as.data.frame)

## End(Not run)
```

---

imlist *Image list*

Description

An imlist object is simply a list of images (of class cimg). For convenience, some generic functions are defined that wouldn't work on plain lists, like plot, display and as.data.frame.

Usage

```
imlist(l)

is.imlist(l)

as.imlist.list(l)
```

Arguments

l	a list
...	ignored

See Also

plot.imlist, display.imlist, as.data.frame.imlist

Examples

```
#imsplit returns objects of class "imlist"
imsplit(boats,"c")
list(a=imfill(3,3),b=imfill(10,10)) %>% imlist
imsplit(boats,"x",6) %>% plot
```

`imnoise` *Generate (Gaussian) white-noise image*

Description

A white-noise image is an image where all pixel values are drawn IID from a certain distribution. Here they are drawn from a Gaussian.

Usage

```
imnoise(x = 1, y = 1, z = 1, cc = 1, mean = 0, sd = 1, dim = NULL)
```

Arguments

`x`	width
`y`	height
`z`	depth
`cc`	spectrum
`mean`	mean pixel value (default 0)
`sd`	std. deviation of pixel values (default 1)
`dim`	dimension vector (optional, alternative to specifying x,y,z,cc)

Value

a cimg object

Author(s)

Simon Barthelme

Examples

```
imnoise(100,100,cc=3) %>% plot(main="White noise in RGB")
imnoise(100,100,cc=3) %>% isoblur(5) %>% plot(main="Filtered (non-white) noise")
imnoise(dim=dim(boats)) #Noise image of the same size as the boats image
```

imrotate *Rotate image by an arbitrary angle.*

Description

Most of the time, the size of the image is modified.

Usage

```
imrotate(im, angle, interpolation = 1L, boundary = 0L)
```

Arguments

im	an image
angle	Rotation angle, in degrees.
interpolation	Type of interpolation. Can be <tt> 0=nearest | 1=linear | 2=cubic </tt>.
boundary	Boundary conditions. Can be <tt> 0=dirichlet | 1=neumann | 2=periodic </tt>.

imsharpen *Sharpen image.*

Description

The default sharpening filter is inverse diffusion. The "shock filter" is a non-linear diffusion that has better edge-preserving properties.

Usage

```
imsharpen(im, amplitude, type = "diffusion", edge = 1, alpha = 0,
  sigma = 0)
```

Arguments

im	an image
amplitude	Sharpening amplitude (positive scalar, 0: no filtering).
type	Filtering type. "diffusion" (default) or "shock"
edge	Edge threshold (shock filters only, positive scalar, default 1).
alpha	Window size for initial blur (shock filters only, positive scalar, default 0).
sigma	Window size for diffusion tensor blur (shock filters only, positive scalar, default 0).

Examples

```
layout(t(1:2))
plot(boats,main="Original")
imsharpen(boats,150)  %>% plot(main="Sharpened")
```

imshift *Shift image content.*

Description

Shift image content.

Usage

```
imshift(im, delta_x = 0L, delta_y = 0L, delta_z = 0L, delta_c = 0L,
  boundary_conditions = 0L)
```

Arguments

im	an image
delta_x	Amount of displacement along the X-axis.
delta_y	Amount of displacement along the Y-axis.
delta_z	Amount of displacement along the Z-axis.
delta_c	Amount of displacement along the C-axis.
boundary_conditions	can be: - 0: Zero border condition (Dirichlet). - 1: Nearest neighbors (Neumann). - 2: Repeat Pattern (Fourier style).

Examples

```
imshift(boats,10,50) %>% plot
```

imsplit *Split an image along a certain axis (producing a list)*

Description

Use this if you need to process colour channels separately, or frames separately, or rows separately, etc. You can also use it to chop up an image into blocks. Returns an "imlist" object, which is essentially a souped-up list.

Usage

```
imsplit(im, axis, nb = -1)
```

Arguments

im	an image
axis	the axis along which to split (for example 'c')
nb	number of objects to split into. if nb=-1 (the default) the maximum number of splits is used ie. split(im,"c") produces a list containing all individual colour channels.

See Also

imappend (the reverse operation)

Examples

```
im <- as.cimg(function(x,y,z) x+y+z,10,10,5)
imsplit(im,"z") #Split along the z axis into a list with 5 elements
imsplit(im,"z",2) #Split along the z axis into two groups
imsplit(boats,"x",-200) %>% plot #Blocks of 200 pix. along x
imsplit(im,"z",2) %>% imappend("z") #Split and reshape into a single image
```

---

imsub *Select part of an image*

Description

imsub selects an image part based on coordinates: it allows you to select a subset of rows, columns, frames etc. Refer to the examples to see how it works

Usage

```
imsub(im, ...)

subim(im, ...)
```

Arguments

im	an image
...	various conditions defining a rectangular image region

Details

subim is an alias defined for backward-compatibility.

Value

an image with some parts cut out

Functions

- subim: alias for imsub

Author(s)

Simon Barthelme

Examples

```
parrots <- load.example("parrots")
imsub(parrots,x < 30) #Only the first 30 columns
imsub(parrots,y < 30) #Only the first 30 rows
imsub(parrots,x < 30,y < 30) #First 30 columns and rows
imsub(parrots, sqrt(x) > 8) #Can use arbitrary expressions
imsub(parrots,x > height/2,y > width/2)  #height and width are defined based on the image
#Using the %inr% operator, which is like %in% but for a numerical range
all.equal(imsub(parrots,x %inr% c(1,10)),
  imsub(parrots,x >= 1,x <= 10))
imsub(parrots,cc==1) #Colour axis is "cc" not "c" here because "c" is an important R function
##Not run
##imsub(parrots,x+y==1)
##can't have expressions involving interactions between variables (domain might not be square)
```

---

imwarp *Image warping*

Description

Image warping consists in remapping pixels, ie. you define a function M(x,y,z) -> (x',y',z') that displaces pixel content from (x,y,z) to (x',y',z'). Actual implementations rely on either the forward transformation M, or the backward (inverse) transformation M^-1. In CImg the forward implementation will go through all source (x,y,z) pixels and "paint" the corresponding pixel at (x',y',z'). This will result in unpainted pixels in the output if M is expansive (for example in the case of a scaling M(x,y,z) = 5*(x,y,z)). The backward implementation will go through every pixel in the destination image and look for ancestors in the source, meaning that every pixel will be painted. There are two ways of specifying the map: absolute or relative coordinates. In absolute coordinates you specify M or M^-1 directly. In relative coordinates you specify an offset function D: M(x,y) = (x,y) + D(x,y) (forward) M^-1(x,y) = (x,y) - D(x,y) (backward)

Usage

```
imwarp(im, map, direction = "forward", coordinates = "absolute",
  boundary = "dirichlet", interpolation = "linear")
```

Arguments

im	an image
map	a function that takes (x,y) or (x,y,z) as arguments and returns a named list with members (x,y) or (x,y,z)
direction	"forward" or "backward" (default "forward")
coordinates	"absolute" or "relative" (default "relative")
boundary	boundary conditions: "dirichlet", "neumann", "periodic". Default "dirichlet"
interpolation	"nearest", "linear", "cubic" (default "linear")

Details

Note that 3D warps are possible as well. The mapping should be specified via the "map" argument, see examples.

Value

a warped image

Author(s)

Simon Barthelme

See Also

warp for direct access to the CImg function

Examples

```
im <- load.example("parrots")
#Shift image
map.shift <- function(x,y) list(x=x+10,y=y+30)
imwarp(im,map=map.shift) %>% plot
#Shift image (backward transform)
imwarp(im,map=map.shift,dir="backward") %>% plot

#Shift using relative coordinates
map.rel <- function(x,y) list(x=10+0*x,y=30+0*y)
imwarp(im,map=map.rel,coordinates="relative") %>% plot

#Scaling
map.scaling <- function(x,y) list(x=1.5*x,y=1.5*y)
imwarp(im,map=map.scaling) %>% plot #Note the holes
map.scaling.inv <- function(x,y) list(x=x/1.5,y=y/1.5)
imwarp(im,map=map.scaling.inv,dir="backward") %>% plot #No holes

#Bending
map.bend.rel <- function(x,y) list(x=50*sin(y/10),y=0*y)
imwarp(im,map=map.bend.rel,coord="relative",dir="backward") %>% plot #No holes
```

im_split *Split an image along a certain axis (producing a list)*

Description

Split an image along a certain axis (producing a list)

Usage

```
im_split(im, axis, nb = -1L)
```

Arguments

im	an image
axis	the axis along which to split (for example 'c')
nb	number of objects to split into. if nb=-1 (the default) the maximum number of splits is used ie. split(im,"c") produces a list containing all individual colour channels

See Also

imappend (the reverse operation)

index.coord *Linear index in internal vector from pixel coordinates*

Description

Pixels are stored linearly in (x,y,z,c) order. This function computes the vector index of a pixel given its coordinates

Usage

```
index.coord(im, coords, outside = "stop")
```

Arguments

im	an image
coords	a data.frame with values x,y,z (optional), c (optional)
outside	what to do if some coordinates are outside the image: "stop" issues error, "NA" replaces invalid coordinates with NAs. Default: "stop".

Value

a vector of indices (NA if the indices are invalid)

Author(s)

Simon Barthelme

See Also

coord.index, the reverse operation

Examples

```
im <- as.cimg(function(x,y) x+y,100,100)
px <- index.coord(im,data.frame(x=c(3,3),y=c(1,2)))
im[px] #Values should be 3+1=4, 3+2=5
```

---

interp *Interpolate image values*

Description

This function provides 2D and 3D (linear or cubic) interpolation for pixel values. Locations need to be provided as a data.frame with variables x,y,z, and c (the last two are optional).

Usage

```
interp(im, locations, cubic = FALSE, extrapolate = TRUE)
```

Arguments

im	the image (class cimg)
locations	a data.frame
cubic	if TRUE, use cubic interpolation. If FALSE, use linear (default FALSE)
extrapolate	allow extrapolation (to values outside the image)

Examples

```
loc <- data.frame(x=runif(10,1,width(boats)),y=runif(10,1,height(boats))) #Ten random locations
interp(boats,loc)
```

is.cimg *Checks that an object is a cimg object*

Description

Checks that an object is a cimg object

Usage

```
is.cimg(x)
```

Arguments

x	an object

Value

logical

isoblur *Blur image isotropically.*

Description

Blur image isotropically.

Usage

```
isoblur(im, sigma, neumann = TRUE, gaussian = FALSE)
```

Arguments

im	an image
sigma	Standard deviation of the blur.
neumann	If true, use Neumann boundary conditions, Dirichlet otherwise (default true, Neumann)
gaussian	Use a Gaussian filter (actually vanVliet-Young). Default: 0th-order Deriche filter.

See Also

deriche,vanvliet

medianblur

Examples

```
isoblur(boats,3) %>% plot(main="Isotropic blur, sigma=3")
isoblur(boats,3) %>% plot(main="Isotropic blur, sigma=10")
```

`label` *Label connected components.*

Description

The algorithm of connected components computation has been primarily done by A. Meijster, according to the publication: 'W.H. Hesselink, A. Meijster, C. Bron, "Concurrent Determination of Connected Components.", In: Science of Computer Programming 41 (2001), pp. 173–194'.

Usage

```
label(im, high_connectivity = FALSE, tolerance = 0)
```

Arguments

`im`	an image
`high_connectivity`	4(false)- or 8(true)-connectivity in 2d case, and between 6(false)- or 26(true)-connectivity in 3d case. Default FALSE
`tolerance`	Tolerance used to determine if two neighboring pixels belong to the same region.

Examples

```
imname <- system.file('extdata/parrots.png',package='imager')
im <- load.image(imname) %>% grayscale
#Thresholding yields different discrete regions of high intensity
regions <- isoblur(im,10) %>% threshold("97%")
labels <- label(regions)
layout(t(1:2))
plot(regions,"Regions")
plot(labels,"Labels")
```

liply *Apply function to each element of a list, then combine the result as an image by appending along specified axis*

Description

This is just a shortcut for llply followed by imappend

Usage

```
liply(lst, fun, axis, ...)
```

Arguments

lst	a list
fun	function to apply
axis	which axis to append along (e.g. "c" for colour)
...	further arguments to be passed to fun

Examples

```
build.im <- function(size) as.cimg(function(x,y) (x+y)/size,size,size)
liply(c(10,50,100),build.im,"y") %>% plot
```

load.example *Load example image*

Description

Imager ships with four test pictures and a video. Two (parrots and boats) come from the [Kodak set](http://r0k.us/graphics/kodak/). Another (birds) is a sketch of birds by Leonardo, from Wikimedia. Also from Wikimedia: the Hubble Deep field (hubble). The test video ("tennis") comes from [xiph.org](https://media.xiph.org/video/derf/)'s collection.

Usage

```
load.example(name)
```

Arguments

name	name of the example

Value

an image

Author(s)

Simon Barthelme

Examples

```
load.example("hubble") %>% plot
load.example("birds") %>% plot
load.example("parrots") %>% plot
```

---

load.image *Load image from file or URL*

Description

PNG, JPEG and BMP are supported via the readbitmap package. You'll need to install ImageMagick for other formats. If the image is actually a video, you'll need ffmpeg. If the path is actually a URL, it should start with http(s) or ftp(s).

Usage

```
load.image(file)
```

Arguments

file	path to file or URL

Value

an object of class 'cimg'

Examples

```
#Find path to example file from package
fpath <- system.file('extdata/Leonardo_Birds.jpg',package='imager')
im <- load.image(fpath)
plot(im)
#Load the R logo directly from the CRAN webpage
#load.image("https://cran.r-project.org/Rlogo.jpg") %>% plot
```

map_il *Type-stable map for use with the purrr package*

Description

Works like purrr::map, purrr::map_dbl and the like but ensures that the output is an image list.

Usage

```
map_il(...)
```

Arguments

...	passed to map

Value

an image list

Author(s)

Simon Barthelme

Examples

```
#Returns a list
imsplit(boats,"x",2) %>% purrr::map(~ isoblur(.,3))
#Returns an "imlist" object
imsplit(boats,"x",2) %>% map_il(~ isoblur(.,3))
#Fails if function returns an object that's not an image
try(imsplit(boats,"x",2) %>% map_il(~ . > 2))
```

medianblur *Blur image with the median filter. In a window of size n x n centered at pixel (x,y), compute median pixel value over the window. Optionally, ignore values that are too far from the value at current pixel.*

Description

Blur image with the median filter.

In a window of size n x n centered at pixel (x,y), compute median pixel value over the window. Optionally, ignore values that are too far from the value at current pixel.

Usage

```
medianblur(im, n, threshold = 0)
```

Arguments

im	an image
n	Size of the median filter.
threshold	Threshold used to discard pixels too far from the current pixel value in the median computation. Can be used for edge-preserving smoothing. Default 0 (include all pixels in window).

See Also

isoblur, boxblur

Examples

```
medianblur(boats,5) %>% plot(main="Median blur, 5 pixels")
medianblur(boats,10) %>% plot(main="Median blur, 10 pixels")
medianblur(boats,10,8) %>% plot(main="Median blur, 10 pixels, threshold = 8")
```

---

mirror *Mirror image content along specified axis*

Description

Mirror image content along specified axis

Usage

```
mirror(im, axis)
```

Arguments

im	an image
axis	Mirror axis ("x","y","z","c")

Examples

```
mirror(boats,"x") %>% plot
mirror(boats,"y") %>% plot
```

pad *Pad image with n pixels along specified axis*

Description

Pad image with n pixels along specified axis

Usage

```
pad(im, nPix, axes, pos = 0, val = 0)
```

Arguments

im	the input image
nPix	how many pixels to pad with
axes	which axes to pad along
pos	-1: prepend 0: center 1: append
val	value to fill the padding with (default 0)

Value

a padded image

Author(s)

Simon Barthelme

Examples

```
pad(boats,20,"xy") %>% plot
pad(boats,20,pos=-1,"xy") %>% plot
pad(boats,20,pos=1,"xy") %>% plot
```

patchstat *Return image patch summary*

Description

Patches are rectangular image regions centered at cx,cy with width wx and height wy. This function provides a fast way of extracting a statistic over image patches (for example, their mean). Supported functions: sum,mean,min,max,median,var,sd, or any valid CImg expression. WARNINGS: - values outside of the image region are considered to be 0. - widths and heights should be odd integers (they're rounded up otherwise).

Usage

```
patchstat(im, expr, cx, cy, wx, wy)
```

Arguments

im	an image
expr	statistic to extract. a string, either one of the usual statistics like "mean","median", or a CImg expression.
cx	vector of x coordinates for patch centers
cy	vector of y coordinates for patch centers
wx	vector of patch widths (or single value)
wy	vector of patch heights (or single value)

Value

a numeric vector

See Also

extract_patches

Examples

```
im <- grayscale(boats)
#Mean of an image patch centered at (10,10) of size 3x3
patchstat(im,'mean',10,10,3,3)
#Mean of image patches centered at (10,10) and (20,4) of size 2x2
patchstat(im,'mean',c(10,20),c(10,4),5,5)
#Sample 10 random positions
ptch <- pixel.grid(im) %>% dplyr::sample_n(10)
#Compute median patch value
with(ptch,patchstat(im,'median',x,y,3,3))
```

---

patch_summary_cimg *Extract a numerical summary from image patches, using CImg's mini-language Experimental feature.*

Description

Extract a numerical summary from image patches, using CImg's mini-language Experimental feature.

Usage

```
patch_summary_cimg(im, expr, cx, cy, wx, wy)
```

Arguments

im	an image
expr	a CImg expression (as a string)
cx	vector of x coordinates for patch centers
cy	vector of y coordinates for patch centers
wx	vector of coordinates for patch width
wy	vector of coordinates for patch height

Examples

```
#Example: median filtering using patch_summary_cimg
#Center a patch at each pixel
im <- grayscale(boats)
patches <- pixel.grid(im)  %>% mutate(w=3,h=3)
#Extract patch summary
out <- mutate(patches,med=patch_summary_cimg(im,"ic",x,y,w,h))
as.cimg(out,v.name="med") %>% plot
```

---

periodic.part — *Compute the periodic part of an image, using the periodic/smooth decomposition of Moisan (2009)*

Description

Moisan (2009) defines an additive image decomposition im = periodic + smooth where the periodic part shouldn't be too far from the original image. The periodic part can be used in frequency-domain analyses, to reduce the artifacts induced by non-periodicity.

Usage

```
periodic.part(im)
```

Arguments

im	an image

Value

an image

Author(s)

Simon Barthelme

References

L. Moisan, Periodic plus Smooth Image Decomposition,J. Math. Imaging Vision, vol. 39:2, pp. 161-179, 2011

Examples

```
im <- load.example("parrots") %>% subim(x <= 512)
layout(t(1:3))
plot(im,main="Original image")
periodic.part(im) %>% plot(main="Periodic part")
#The smooth error is the difference between
#the original image and its periodic part
(im-periodic.part(im)) %>% plot(main="Smooth part")
```

---

permute_axes *Permute image axes*

Description

By default images are stored in xyzc order. Use permute_axes to change that order.

Usage

```
permute_axes(im, perm)
```

Arguments

im	an image
perm	a character string, e.g., "zxyc" to have the z-axis come first

Examples

```
im <- array(0,c(10,30,40,3)) %>% as.cimg
permute_axes(im,"zxyc")
```

`pixel.grid` *Return the pixel grid for an image*

Description

The pixel grid for image im gives the (x,y,z,c) coordinates of each successive pixel as a data.frame. The c coordinate has been renamed 'cc' to avoid conflicts with R's c function. NB: coordinates start at (x=1,y=1), corresponding to the top left corner of the image, unless standardise == TRUE, in which case we use the usual Cartesian coordinates with origin at the center of the image and scaled such that x varies between -.5 and .5, and a y arrow pointing up

Usage

```
pixel.grid(im, standardise = FALSE, drop.unused = TRUE, dim = NULL)
```

Arguments

`im`	an image
`standardise`	If TRUE use a centered, scaled coordinate system. If FALSE use standard image coordinates (default FALSE)
`drop.unused`	if TRUE ignore empty dimensions, if FALSE include them anyway (default TRUE)
`dim`	a vector of image dimensions (optional, may be used instead of "im")

Value

a data.frame

Examples

```
im <- as.cimg(array(0,c(10,10))) #A 10x10 image
pixel.grid(im) %>% head
pixel.grid(dim=dim(im)) %>% head #Same as above
pixel.grid(dim=c(10,10,3,2)) %>% head
pixel.grid(im,standardise=TRUE) %>% head
pixel.grid(im,drop.unused=FALSE) %>% head
```

play *Play a video*

Description

A very basic video player. Press the space bar to pause and ESC to close.

Usage

```
play(vid, loop = FALSE, delay = 30L, normalise = TRUE)
```

Arguments

vid	A cimg object, to be played as video
loop	loop the video (default false)
delay	delay between frames, in ms. Default 30.
normalise	if true pixel values are rescaled to 0...255 (default TRUE). The normalisation is based on the *first frame*. If you don't want the default behaviour you can normalise by hand. Default TRUE.

plot.cimg *Display an image using base graphics*

Description

If you want to control precisely how numerical values are turned into colours for plotting, you need to specify a colour scale using the colourscale argument (see examples). Otherwise the default is "gray" for grayscale images, "rgb" for colour. These expect values in [0..1], so the default is to rescale the data to [0..1]. If you wish to over-ride that behaviour, set rescale=FALSE. See examples for an explanation. If the image is one dimensional (i.e., a simple row or column image), then pixel values will be plotted as a line.

Usage

```
## S3 method for class 'cimg'
plot(x, frame, xlim = c(1, width(x)), ylim = c(height(x), 1),
  xlab = "x", ylab = "y", rescale = TRUE, colourscale = NULL,
  colorscale = NULL, interpolate = TRUE, ...)
```

Arguments

x	the image
frame	which frame to display, if the image has depth > 1
xlim	x plot limits (default: 1 to width)
ylim	y plot limits (default: 1 to height)
xlab	x axis label
ylab	y axis label
rescale	rescale pixel values so that their range is [0,1]
colourscale, colorscale	an optional colour scale (default is gray or rgb)
interpolate	should the image be plotted with antialiasing (default TRUE)
...	other parameters to be passed to plot.default (eg "main")

See Also

display, which is much faster, as.raster, which converts images to R raster objects

Examples

```
plot(boats,main="Boats") #extra arguments are passed to default plot function
plot(boats,axes=FALSE,xlab="",ylab="")

#Pixel values are rescaled to 0-1 by default, so that the following two plots are identical
plot(boats)
plot(boats/2,main="Rescaled")
#If you don't want that behaviour, you can set rescale to FALSE, but
#then you need to make sure values are in [0,1]
try(plot(boats,rescale=FALSE)) #Error!
try(plot(boats/255,rescale=FALSE)) #Works
#You can specify a colour scale if you don't want the default one.
#A colour scale is a function that takes pixels values and return an RGB code,
#like R's rgb function,e.g.
rgb(0,1,0)
#Let's switch colour channels
cscale <- function(r,g,b) rgb(b,g,r)
plot(boats/255,rescale=FALSE,colourscale=cscale)
#Display slice of HSV colour space
im <- imfill(255,255,val=1)
im <- list(Xc(im)/255,Yc(im)/255,im) %>% imappend("c")
plot(im,colourscale=hsv,rescale=FALSE,
     xlab="Hue",ylab="Saturation")
#In grayscale images, the colourscale function should take in a single value
#and return an RGB code
boats.gs <- grayscale(boats)
#We use an interpolation function from package scales
cscale <- scales::gradient_n_pal(c("red","purple","lightblue"),c(0,.5,1))
plot(boats.gs,rescale=FALSE,colourscale=cscale)
#Plot a one-dimensional image
```

```
imsub(boats,x==1) %>% plot(main="Image values along first column")
#Plotting with and without anti-aliasing:
boats.small <- imresize(boats,.3)
plot(boats.small,interp=TRUE)
plot(boats.small,interp=FALSE)
```

`plot.imlist` *Plot an image list*

Description

Each image in the list will be plotted separately. The layout argument controls the overall layout of the plot window. The default layout is "rect", which will fit all of your images into a rectangle that's as close to a square as possible.

Usage

```
## S3 method for class 'imlist'
plot(x, layout = "rect", ...)
```

Arguments

`x`	an image list (of type imlist)
`layout`	either a matrix (in the format defined by the layout command) or one of "row","col" or "rect". Default: "rect"
`...`	other parameters, to be passed to the plot command

Author(s)

Simon Barthelme

Examples

```
imsplit(boats,"c") #Returns an image list
```

renorm *Renormalise image*

Description

Pixel data is usually expressed on a 0...255 scale for displaying. This function performs a linear renormalisation to range min...max

Usage

```
renorm(x, min = 0, max = 255)
```

Arguments

x	numeric data
min	min of the range
max	max of the range

Author(s)

Simon Barthelme

Examples

```
renorm(0:10)
renorm(-5:5) #Same as above
```

resize *Resize image to new dimensions. If pd[x,y,z,v]<0, it corresponds to a percentage of the original size (the default value is -100).*

Description

Resize image to new dimensions. If pd[x,y,z,v]<0, it corresponds to a percentage of the original size (the default value is -100).

Usage

```
resize(im, size_x = -100L, size_y = -100L, size_z = -100L,
  size_c = -100L, interpolation_type = 1L, boundary_conditions = 0L,
  centering_x = 0, centering_y = 0, centering_z = 0, centering_c = 0)
```

Arguments

im	an image
size_x	Number of columns (new size along the X-axis).
size_y	Number of rows (new size along the Y-axis).
size_z	Number of slices (new size along the Z-axis).
size_c	Number of vector-channels (new size along the C-axis).
interpolation_type	Method of interpolation: -1 = no interpolation: raw memory resizing. 0 = no interpolation: additional space is filled according to boundary_conditions. 1 = nearest-neighbor interpolation. 2 = moving average interpolation. 3 = linear interpolation. 4 = grid interpolation. 5 = cubic interpolation. 6 = lanczos interpolation.
boundary_conditions	Border condition type.
centering_x	Set centering type (only if interpolation_type=0).
centering_y	Set centering type (only if interpolation_type=0).
centering_z	Set centering type (only if interpolation_type=0).
centering_c	Set centering type (only if interpolation_type=0).

---

resize_doubleXY *Resize image uniformly*

Description

Resize image by a single scale factor. For non-uniform scaling and a wider range of options, see resize.

Usage

```
resize_doubleXY(im)

resize_halfXY(im)

resize_tripleXY(im)

imresize(im, scale = 1)
```

Arguments

im	an image
scale	a scale factor

Value

an image

Functions

- `resize_doubleXY`: Double size
- `resize_halfXY`: Half size
- `resize_tripleXY`: Triple size
- `imresize`: resize by scale factor

Author(s)

Simon Barthelme

References

For double-scale, half-scale, triple-scale, etc. uses an anisotropic scaling algorithm described in: http://scale2x.sourceforge.net/algorithm.html.

See Also

resize

Examples

```
im <- load.example("parrots")
imresize(im,1/4) #Quarter size
liply(2:4,function(ind) imresize(im,1/ind),"x") %>%  plot
```

RGBtoHSL *Colour space conversions in imager*

Description

All functions listed here assume the input image has three colour channels (spectrum(im) == 3)

Usage

```
RGBtoHSL(im)

HSLtoRGB(im)

RGBtoHSV(im)

HSVtoRGB(im)

RGBtoHSI(im)
```

```
HSItoRGB(im)

RGBtosRGB(im)

sRGBtoRGB(im)

RGBtoYCbCr(im)

YCbCrtoRGB(im)

RGBtoYUV(im)

YUVtoRGB(im)
```

Arguments

im	an image

Functions

- RGBtoHSL: RGB to HSL conversion
- HSLtoRGB: HSL to RGB conversion
- RGBtoHSV: RGB to HSV conversion
- HSVtoRGB: HSV to RGB conversion
- RGBtoHSI: RGB to HSI conversion
- HSItoRGB: HSI to RGB conversion
- RGBtosRGB: RGB to sRGB conversion
- sRGBtoRGB: sRGB to RGB conversion
- RGBtoYCbCr: RGB to YCbCr conversion
- YCbCrtoRGB: YCbCr to RGB conversion
- RGBtoYUV: RGB to YUV conversion
- YUVtoRGB: YUV to RGB conversion

---

rotate_xy *Rotate image by an arbitrary angle, around a center point.*

Description

Rotate image by an arbitrary angle, around a center point.

Usage

```
rotate_xy(im, angle, cx, cy, zoom = 1, interpolation = 1L,
  boundary_conditions = 0L)
```

Arguments

im	an image
angle	Rotation angle, in degrees.
cx	X-coordinate of the rotation center.
cy	Y-coordinate of the rotation center.
zoom	Zoom factor.
interpolation	Interpolation type. 0=nearest | 1=linear | 2=cubic
boundary_conditions	Boundary conditions. 0=dirichlet | 1=neumann | 2=periodic

Examples

```
rotate_xy(boats,30,200,400) %>% plot
rotate_xy(boats,30,200,400,boundary=2) %>% plot
```

---

save.image *Save image*

Description

You'll need ImageMagick for formats other than PNG and JPEG. If the image is actually a video, you'll need ffmpeg.

Usage

```
save.image(im, file)
```

Arguments

im	an image (of class cimg)
file	path to file. The format is determined by the file's name

Value

nothing

Examples

```
#Create temporary file
tmpF <- tempfile(fileext=".png")
#Save boats image
save.image(boats,tmpF)
#Read back and display
load.image(tmpF) %>% plot
```

selectSimilar *Select a region of homogeneous colour*

Description

The underlying algorithm is the same as the bucket fill (AKA flood fill). Unlike with the bucket fill, the image isn't changed, the function simply returns a binary mask of the selected region

Usage

```
selectSimilar(im, x, y, z = 1, sigma = 0, high_connexity = FALSE)
```

Arguments

im	an image
x	X-coordinate of the starting point of the region to fill.
y	Y-coordinate of the starting point of the region to fill.
z	Z-coordinate of the starting point of the region to fill.
sigma	Tolerance concerning neighborhood values.
high_connexity	Use 8-connexity (only for 2d images, default FALSE).

See Also

bucketfill

Examples

```
#Select part of a sail
impart <- selectSimilar(boats,x=169,y=179,sigma=20)
layout(t(1:2))
plot(boats,main="Original")
plot(impart,main="Selected region")
```

squeeze *Remove empty dimensions from an array*

Description

Works just like Matlab's squeeze function: if anything in dim(x) equals one the corresponding dimension is removed

Usage

```
squeeze(x)
```

Arguments

x	an array

Examples

```
A <- array(1:9,c(3,1,3)) #3D array with one flat dimension
A %>% squeeze #flat dimension removed
```

---

stencil.cross *A cross-shaped stencil*

Description

Returns a stencil corresponding to all nearest-neighbours of a pixel

Usage

```
stencil.cross(z = FALSE, cc = FALSE, origin = FALSE)
```

Arguments

z	include neighbours along the z axis
cc	include neighbours along the cc axis
origin	include center pixel (default false)

Value

a data.frame defining a stencil

Author(s)

Simon Barthelme

See Also

get.stencil

threshold *Threshold grayscale image*

Description

Thresholding corresponding to setting all values below a threshold to 0, all above to 1. If you call threshold with thr="auto" a threshold will be computed automatically using kmeans (ie., using a variant of Otsu's method). This works well if the pixel values have a clear bimodal distribution. If you call threshold with a string argument of the form "XX%" (e.g., "98%"), the threshold will be set at percentile XX. Computing quantiles or running kmeans is expensive for large images, so if approx == TRUE threshold will skip pixels if the total number of pixels is above 10,000. Note that thresholding a colour image will threshold all the colour channels jointly, which may not be the desired behaviour! Use iiply(im,"c",threshold) to find optimal values for each channel separately.

Usage

```
threshold(im, thr = "auto", approx = TRUE)
```

Arguments

im	the image
thr	a threshold, either numeric, or "auto", or a string for quantiles
approx	Skip pixels when computing quantiles in large images (default TRUE)

Value

a thresholded image

Author(s)

Simon Barthelme

Examples

```
im <- load.example("birds")
im.g <- grayscale(im)
threshold(im.g,"15%") %>% plot
threshold(im.g,"auto") %>% plot
threshold(im.g,.1) %>% plot
```

vanvliet *Young-Van Vliet recursive Gaussian filter.*

Description

The Young-van Vliet filter is a fast approximation to a Gaussian filter (order = 0), or Gaussian derivatives (order = 1 or 2).

Usage

```
vanvliet(im, sigma, order = 0L, axis = "x", neumann = FALSE)
```

Arguments

im	an image
sigma	standard deviation of the Gaussian filter
order	the order of the filter 0,1,2,3
axis	Axis along which the filter is computed. Can be <tt> 'x' | 'y' | 'z' | 'c' </tt>.
neumann	If true, use Neumann boundary conditions (default false, Dirichlet)

References

From: I.T. Young, L.J. van Vliet, M. van Ginkel, Recursive Gabor filtering. IEEE Trans. Sig. Proc., vol. 50, pp. 2799-2805, 2002. (this is an improvement over Young-Van Vliet, Sig. Proc. 44, 1995)

Boundary conditions (only for order 0) using Triggs matrix, from B. Triggs and M. Sdika. Boundary conditions for Young-van Vliet recursive filtering. IEEE Trans. Signal Processing, vol. 54, pp. 2365-2367, 2006.

Examples

```
vanvliet(boats,sigma=2,order=0) %>% plot("Zeroth-order Young-van Vliet along x")
vanvliet(boats,sigma=2,order=1) %>% plot("First-order Young-van Vliet along x")
vanvliet(boats,sigma=2,order=1) %>% plot("Second-order Young-van Vliet along x")
vanvliet(boats,sigma=2,order=1,axis="y") %>% plot("Second-order Young-van Vliet along y")
```

warp *Warp image*

Description

Warp image

Usage

```
warp(im, warpfield, mode = 0L, interpolation = 1L,
  boundary_conditions = 0L)
```

Arguments

im	an image
warpfield	Warping field. The (x,y,z) fields should be stacked along the colour coordinate.
mode	Can be 0=backward-absolute | 1=backward-relative | 2=forward-absolute | 3=forward-relative
interpolation	Can be <tt> 0=nearest | 1=linear | 2=cubic </tt>.
boundary_conditions	Boundary conditions. Can be <tt> 0=dirichlet | 1=neumann | 2=periodic </tt>.

See Also

imwarp for a user-friendly interface

Examples

```
#Shift image via warp
warp.x <- imfill(width(boats),height(boats),val=5)
warp.y <- imfill(width(boats),height(boats),val=20)
warpfield <- list(warp.x,warp.y) %>% imappend("c")
warp(boats,warpfield,mode=1) %>% plot
```

---

watershed *Compute watershed transform.*

Description

The watershed transform is a label propagation algorithm. The value of non-zero pixels will get propagated to their zero-value neighbours. The propagation is controlled by a priority map. See examples.

Usage

```
watershed(im, priority, fill_lines = TRUE)
```

Arguments

im	an image
priority	Priority map.
fill_lines	Sets if watershed lines must be filled or not.

Examples

```
#In our initial image we'll place three seeds
#(non-zero pixels) at various locations, with values 1, 2 and 3.
#We'll use the watershed algorithm to propagate these values
imd <- function(x,y) imdirac(c(100,100,1,1),x,y)
im <- imd(20,20)+2*imd(40,40)+3*imd(80,80)
layout(t(1:3))
plot(im,main="Seed image")
#Now we build an priority map: neighbours of our seeds
#should get high priority.
#We'll use a distance map for that
p <- 1-distance_transform(sign(im),1)
plot(p,main="Priority map")
watershed(im,p) %>% plot(main="Watershed transform")
```

`%inr%` *Check that value is in a range*

Description

A shortcut for x >= a | x <= b.

Usage

```
x %inr% range
```

Arguments

`x`	numeric values
`range`	a vector of length two, of the form c(a,b)

Value

a vector of logicals 1:10

Author(s)

Simon Barthelme

Index