Package ‘imager’

April 26, 2024

Type Package

Title Image Processing Library Based on ‘CImg’

Version 1.0.1

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-3

Imports Rcpp (>= 1.0.0), methods, stringr, png, jpeg, readbitmap, grDevices, purrr, downloader, igraph

Depends R (>= 4.0.0), magrittr


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

SystemRequirements fftw3, libtiff, X11

LinkingTo Rcpp

LazyData true

RoxxygenNote 7.2.3

Suggests knitr, rmarkdown, ggplot2, dplyr, scales, testthat, raster, spatstat.geom, magick, Cairo

VignetteBuilder knitr

Encoding UTF-8

NeedsCompilation yes

Author Simon Barthelme [aut],
  David Tschumperle [ctb],
  Jan Wijffels [ctb],
  Haz Edine Assemal [ctb],
  Shota Ochi [ctb],
  Aaron Robotham [cre],
  Rodrigo Tobar [ctb]
Maintainer Aaron Robotham <aaron.robotham@uwa.edu.au>
Repository CRAN
Date/Publication 2024-04-26 10:40:03 UTC

R topics documented:

add.colour .......................................................... 5
as.cimg ............................................................... 6
as.cimg.array ..................................................... 7
as.cimg.data.frame ............................................. 8
as.cimg.function ................................................. 9
as.cimg.im ........................................................ 10
as.cimg.raster .................................................... 11
as.data.frame.cimg ............................................. 11
as.data.frame.imlist ........................................... 12
as.data.frame.pixset ........................................... 13
as.igraph.cimg .................................................. 13
as.igraph.pixset ................................................. 15
as.imlist.list .................................................... 16
as.pixset .......................................................... 17
as.raster.cimg .................................................. 18
at ................................................................. 19
autocrop ............................................................ 20
bbox ............................................................... 21
blur_anisotropic ................................................ 22
boats ............................................................... 23
boundary ........................................................... 23
boxblur ............................................................. 24
boxblur_xy .......................................................... 25
bucketfill .......................................................... 25
cannyEdges ........................................................ 26
capture.plot ....................................................... 27
center.stencil ..................................................... 28
channels ........................................................... 28
ci ................................................................. 29
cimg ............................................................... 30
cimg.dimensions ................................................ 30
cimg.extract ..................................................... 31
cimg.openmp ...................................................... 32
cimg2im ............................................................ 33
circles .............................................................. 34
clean ............................................................... 35
colorise ............................................................. 36
common_pixsets .................................................. 37
contours ........................................................... 38
coord.index ........................................................ 39
correlate .......................................................... 40
crop.borders ................................................................. 41
deriche ................................................................. 42
diffusion_tensors ......................................................... 43
displacement .............................................................. 43
display ................................................................. 44
display.cimg ............................................................. 44
display.list ............................................................. 45
distance_transform ......................................................... 46
draw_circle ................................................................. 46
draw_rect ................................................................. 47
draw_text ................................................................. 48
erode ................................................................. 49
extract_patches ....................................................... 51
FFT ................................................................. 52
flatten.alpha ......................................................... 53
frames ................................................................. 54
get.locations ............................................................. 54
get.stencil .............................................................. 55
get_gradient ............................................................ 56
get_hessian ............................................................. 56
grab ................................................................. 57
grayscale ............................................................... 58
grow ................................................................. 58
gsdim ................................................................. 59
haar ................................................................. 60
highlight ............................................................... 61
hough_circle ............................................................. 61
hough_line .............................................................. 62
idply ................................................................. 63
iilply ................................................................. 64
ilply ................................................................. 64
imager ................................................................. 65
imager.combine .......................................................... 66
imager.replace .......................................................... 69
imager.subset .......................................................... 70
imappend ............................................................... 71
imchange ............................................................... 71
imcoord ............................................................... 72
imdirc ............................................................... 74
imdraw .............................................................. 75
imeval ............................................................... 76
imfill ............................................................... 78
imgradient ............................................................ 79
imhessian ............................................................. 79
iminfo ............................................................... 80
imlap ............................................................... 81
imlist ............................................................... 81
imnoise ............................................................. 82
R topics documented:

implot ......................................................... 83
imrep ....................................................... 84
imrotate .................................................... 84
imsharpen ................................................... 85
imshift ..................................................... 86
imsplit ..................................................... 86
imsup ....................................................... 87
imwarp ..................................................... 88
im_split ................................................... 90
index.coord ............................................... 90
inpaint .................................................... 91
interact ................................................... 92
interp ...................................................... 93
is.cimg .................................................... 94
is.imlist ................................................... 94
is.pixset .................................................. 95
isoblr ...................................................... 95
label ....................................................... 96
liply ......................................................... 97
load.dir ................................................... 97
load.example ............................................ 98
load.image ............................................... 99
load.video ............................................... 99
magick ..................................................... 101
make.video ............................................... 102
map_il ..................................................... 103
medianblur ............................................... 104
mirror ..................................................... 105
mutate_plyr ............................................... 105
nline ....................................................... 106
pad ........................................................ 107
patchstat .................................................. 108
patch_summary_cimg ..................................... 109
periodic.part ............................................ 109
permute_axes ............................................. 110
pixel.grid ................................................. 111
pixset ..................................................... 112
play ......................................................... 112
plot.cimg ............................................... 113
plot.imlist ............................................... 115
px.flood .................................................. 116
px.na ..................................................... 117
px.remove_outer ......................................... 117
RasterPackage ............................................ 118
renorm .................................................... 119
resize ..................................................... 119
resize_doubleXY ......................................... 120
RGBtoHSL .................................................. 122
**add.colour**

Add colour channels to a grayscale image or pixel set

**Description**

Add colour channels to a grayscale image or pixel set

**Usage**

```r
add.colour(im, simple = TRUE)
add.color(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. For pixel sets this option makes no sense and is ignored.

**Value**

an image of class cimg

**Functions**

- `add.color()`: Alias for `add.colour`

**Author(s)**

Simon Barthelme
Examples

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

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

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

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

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

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

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

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

```r
## 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)

- `as.cimg(numeric)`: convert numeric
- `as.cimg(logical)`: convert logical
- `as.cimg(double)`: convert double
- `as.cimg(cimg)`: return object
- `as.cimg(vector)`: convert vector
- `as.cimg(matrix)`: Convert to matrix

Author(s)

Simon Barthelme

See Also

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

Examples

```r
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
```

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

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

Arguments

- `obj`: an array
- `...`: ignored
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 frame 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) %>% dplyr::mutate(value=x*y)
#Convert to cimg object (2D, grayscale image of size 10*10
as.cimg(df, dims=c(c(10,10,1,1))) %>% plot

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.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. Default 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

```r
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.cimg.im

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

#### Description

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

#### Usage

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

#### Arguments

- `obj`: a spatstat image
- `...`: optional arguments

#### Value

A cimg image

#### Author(s)

Simon Barthelme
as.cimg.raster  Convert a raster object to a cimg object

Description
R’s native object for representing images is a "raster". This function converts raster objects to cimg objects.

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

Arguments
obj  a raster object
...
ignored

Value
a cimg object

Author(s)
Simon Barthelme

Examples
rst <- as.raster(matrix((1:4)/4,2,2))
as.cimg(rst) %>% plot(int=FALSE)
all.equal(rst,as.raster(as.cimg(rst)))

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 separate entry.

Value

- a data.frame

Author(s)

Simon Barthelme

Examples

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

Description

Convert image list to data.frame

Usage

```r
## 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 column containing the index (or name) of the image in the list. Default: "im"

Examples

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

Usage

## S3 method for class 'pixset'
as.data.frame(x, ..., drop = FALSE)

Arguments

x                      pixset to convert
...                    ignored
drop                   drop flat dimensions

See Also

where

Examples

px <- boats > 250
# Convert to array of logicals
as.logical(px) %>% dim
# Convert to data.frame: gives all pixel locations in the set
as.data.frame(px) %>% head
# Drop flat dimensions
as.data.frame(px,drop=TRUE) %>% head

Form a graph from an image

Description

In this graph representation, every pixel is a vertex connected to its neighbours. The image values
along edges are stored as graph attributes (see examples).

Usage

## S3 method for class 'cimg'
as.igraph(x, mask = px.all(channel(im, 1)), ...)

Arguments

\section*{Arguments}

\begin{itemize}
\item \textit{x} \hspace{1cm} an image (must be 2D, 3D not implemented yet)
\item \textit{mask} \hspace{1cm} optional: a pixset. if provided, pixels are only connected if they are both in the pixset.
\item \ldots \hspace{1cm} ignored
\end{itemize}

Value

\vspace{0.5cm}

\section*{Value}

a graph (igraph format) with attributes value.from, value.to and dist

Author(s)

\vspace{0.5cm}

\section*{Author(s)}

Simon Barthelme

See Also

\vspace{0.5cm}

\section*{See Also}

as.igraph.pixset

Examples

\vspace{0.5cm}

\section*{Examples}

\begin{verbatim}
library(igraph)
im <- imfill(5,5)
G <- as.igraph(im)
plot(G)
#Shortest-path distance from pixel 1 to all other pixels
d <- igraph::distances(G,1) %>% as.vector
as.cimg(d,dim=gsdim(im)) %>% plot(interpolate=FALSE)
#Notice that moving along the diagonal has the same cost
#as moving along the cardinal directions, whereas the Euclidean distance
#is actually sqrt(2) and not 1.
#Modify weight attribute, to change the way distance is computed
igraph::E(G)$weight <- G$dist
d2 <- igraph::distances(G,1) %>% as.vector
as.cimg(d2,dim=gsdim(im)) %>% plot(interpolate=FALSE)
#More interesting example
im <- grayscale(boats)
G <- as.igraph(im)
#value.from holds the value of the source pixel, value.to the sink's
#set w_{ij} = (|v_i - v_j|)/d_{ij}
igraph::E(G)$weight <- (abs(G$value.from - G$value.to))/G$dist
igraph::distances(G,5000) %>% as.vector %>%
as.cimg(dim=gsdim(im)) %>% plot
\end{verbatim}
Form an adjacency graph from a pixset

Description

Return a graph where nodes are pixels, and two nodes are connected if and only if both nodes are in the pixset, and the pixels are adjacent. Optionally, add weights corresponding to distance (either 1 or \sqrt{2}, depending on the orientation of the edge). The graph is represented as an igraph "graph" object.

Usage

## S3 method for class 'pixset'
as.igraph(x, weighted = TRUE, ...)

Arguments

x a pixset
weighted add weight for distance (default TRUE)
... ignored

Value

an igraph "graph" object

See Also

as.igraph.cimg

Examples

library(igraph)
# Simple 3x3 lattice
px <- px.all(imfill(3,3))
as.igraph(px) %>% plot
# Disconnect central pixel
px[5] <- FALSE
as.igraph(px) %>% plot
# Form graph from thresholded image
im <- load.example("coins")
px <- threshold(im) %>% fill(5)
G <- as.igraph(px)
# Label connected components
v <- (igraph::clusters(G)$membership)
as.cimg(v,dim=dim(px)) %>% plot
# Find a path across the image that avoids all the coins
G <- as.igraph(!px)
start <- index.coord(im, data.frame(x=1, y=100))
end <- index.coord(im, data.frame(x=384, y=300))
sp <- igraph::shortest_paths(G, start, end, output="vpath")
path <- sp$vpath[[1]] %>% as.integer %>% coord.index(im, .)

---

**as.imlist.list**

Convert various objects to image lists

**Description**

Convert various objects to image lists

**Usage**

```r
## S3 method for class 'list'
as.imlist(obj, ...)
as.imlist(obj, ...)
## S3 method for class 'imlist'
as.imlist(obj, ...)
## S3 method for class 'cimg'
as.imlist(obj, ...)
```

**Arguments**

- `obj` an image list
- `...` ignored

**Value**

a list

**Methods (by class)**

- `as.imlist(list)`: convert from list
- `as.imlist(imlist)`: Convert from imlist (identity)
- `as.imlist(cimg)`: Convert from image

**Examples**

```r
list(a=boats, b=boats*2) %>% as.imlist
```
as.pixset

Methods to convert various objects to pixsets

Description

Methods to convert various objects to pixsets

Usage

as.pixset(x, ...)

## S3 method for class 'cimg'
as.pixset(x, ...)

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

Arguments

x          object to convert to pixset
...
obj        pixset to convert

Methods (by class)

• as.pixset(cimg): convert cimg to pixset

Functions

• as.cimg(pixset): convert pixset to cimg

Examples

# When converting an image to a pixset, the default is to include all pixels with non-zero value
as.pixset(boats)
# The above is equivalent to:
boats!=0
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, colourscale = NULL, col.na = rgb(0, 0, 0, 0), ...
)

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
colourscale same as above in American spelling
col.na which colour to use for NA values, as R rgb code. The default is "rgb(0,0,0,0)", which corresponds to a fully transparent colour.
... ignored

Value

a raster object

Author(s)

Simon Barthelme
at

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
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(im, x, y, z = 1, cc = 1) <- value: set value of pixel at a location
• color.at(): return value of all colour channels at a location
• color.at(im, x, y, z = 1) <- value: set value of all colour channels at a location

Author(s)

Simon Bartheleme

Examples

```r
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 %in% c(1,100), y %in% c(1,100)) %>% plot
```

autocrop

Autocrop image region

Description

Autocrop image region

Usage

```r
autocrop(im, color = color.at(im, 1, 1), axes = "zyx")
```

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>im</td>
<td>an image</td>
</tr>
<tr>
<td>color</td>
<td>Colour used for the crop. If missing, the colour is taken from the top-left pixel. Can also be a colour name (e.g. &quot;red&quot;, or &quot;black&quot;)</td>
</tr>
<tr>
<td>axes</td>
<td>Axes used for the crop.</td>
</tr>
</tbody>
</table>
Examples

```r
# Add pointless padding
padded <- pad(boats, 30, "xy")
plot(padded)

# Remove padding
autocrop(padded) %>% plot

# You can specify the colour if needs be
autocrop(padded, "black") %>% plot

# Autocrop has a zero-tolerance policy: if a pixel value is slightly different from the one you gave
# the pixel won't get cropped. A fix is to do a bucket fill first
padded <- isoblur(padded, 10)
autocrop(padded) %>% plot

padded2 <- bucketfill(padded, 1, 1, col = c(0, 0, 0), sigma = 0.1)
autocrop(padded2) %>% plot
```

bbox

*Compute the bounding box of a pixset*

Description

This function returns the bounding box of a pixset as another pixset. If the image has more than one frame, a bounding cube is returned. If the image has several colour channels, the bounding box is computed separately in each channel. crop.bbox crops an image using the bounding box of a pixset.

Usage

```r
bbox(px)
crop.bbox(im, px)
```

Arguments

- `px` a pixset
- `im` an image

Value

A pixset object

Functions

- `crop.bbox()`: crop image using the bounding box of pixset px

Author(s)

Simon Barthelme
Examples

```r
im <- grayscale(boats)
px <- im > .85
plot(im)
highlight(bbox(px))
highlight(px, col="green")
crop.bbox(im, px) %>% 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

```r
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

```r
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

---

**boundary**

*Find the boundary of a shape in a pixel set*

**Description**

Find the boundary of a shape in a pixel set.

**Usage**

`boundary(px, depth = 1, high_connexity = FALSE)`

**Arguments**

- `px`: pixel set
- `depth`: boundary depth (default 1)
- `high_connexity`: if FALSE, use 4-point neighbourhood. If TRUE, use 8-point. (default FALSE)
Examples

```
px.diamond(10,30,30) %>% boundary %>% plot
px.square(10,30,30) %>% boundary %>% plot
px.square(10,30,30) %>% boundary(depth=3) %>% plot
px <- (px.square(10,30,30) | px.circle(12,30,30))
boundary(px,high=TRUE) %>% plot(int=TRUE,main="8-point neighbourhood")
boundary(px,high=TRUE) %>% plot(int=FALSE,main="4-point neighbourhood")
```

boxblur

Blur image with a box filter (square window)

Description

Blur image with a box filter (square window)

Usage

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

Arguments

- **im**: an image
- **boxsize**: Size of the box window (can be subpixel).
- **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: a vector of values (of length spectrum(im)), or a colour name (e.g. "red"). If missing, use the colour at location (x,y,z).
opacity: opacity. If the opacity is below 1, paint with transparency.
sigma: Tolerance for neighborhood values: spread to neighbours if difference is less than sigma (for grayscale). If there are several channels, the sum of squared differences is used: if it below sigma^2, the colour spreads.

high_connexity: Use 8-connexity (only for 2d images, default FALSE).

See Also

px.flood

Examples

# Change the colour of a sail
boats.new <- bucketfill(boats, x=169, y=179, color="pink", sigma=.2)
plot(boats, main="Original")
plot(boats.new, main="New sails")

# More spreading, lower opacity, colour specified as vector
ugly <- bucketfill(boats, x=169, y=179, color=c(0,1,0), sigma=.6, opacity=.5)
plot(ugly)

cannyEdges Canny edge detector

Description

If the threshold parameters are missing, they are determined automatically using a k-means heuristic. Use the alpha parameter to adjust the automatic thresholds up or down. The thresholds are returned as attributes. The edge detection is based on a smoothed image gradient with a degree of smoothing set by the sigma parameter.

Usage

cannyEdges(im, t1, t2, alpha = 1, sigma = 2)
**capture.plot**

Capture the current R plot device as a cimg image

---

**Arguments**

- **im**: input image
- **t1**: threshold for weak edges (if missing, both thresholds are determined automatically)
- **t2**: threshold for strong edges
- **alpha**: threshold adjustment factor (default 1)
- **sigma**: smoothing

**Author(s)**

Simon Barthelme

**Examples**

```r
  cannyEdges(boats) %>% plot #Make thresholds less strict
cannyEdges(boats, alpha = .4) %>% plot #Make thresholds more strict
cannyEdges(boats, alpha = 1.4) %>% plot
```

---

**Description**

Capture the current R plot device as a cimg image

**Usage**

```r
  capture.plot()
```

**Value**

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

**Author(s)**

Simon Barthelme

**Examples**

```r
  ##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
```r
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

---

**ci**

*Concatenation for image lists*

**Description**

Allows you to concatenate image lists together, or images with image lists. Doesn’t quite work like R’s “c” primitive: image lists are always *flat*, not nested, meaning each element of an image list is an image.

**Usage**

\[ \text{ci}(\ldots) \]

**Arguments**

\[ \ldots \quad \text{objects to concatenate} \]

**Value**

an image list

**Author(s)**

Simon Barthelme

**Examples**

\[
\begin{align*}
l1 & \leftarrow \text{imlist}(\text{boats,grayscale}(\text{boats})) \\
l2 & \leftarrow \text{imgradient}(\text{boats},"xy") \\
\text{ci}(l1,l2) & \# \text{List + list} \\
\text{ci}(l1,\text{imfill}(3,3)) & \# \text{List + image} \\
\text{ci}(\text{imfill}(3,3),l1,l2) & \# \text{Three elements, etc.}
\end{align*}
\]
cimg is a class for storing image or video/hyperspectral data. It is designed to provide easy inter-
action 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)))

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

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>im</td>
<td>an image</td>
</tr>
<tr>
<td>index</td>
<td>frame index</td>
</tr>
<tr>
<td>x</td>
<td>x coordinate of the row</td>
</tr>
<tr>
<td>y</td>
<td>y coordinate of the row</td>
</tr>
<tr>
<td>ind</td>
<td>channel index</td>
</tr>
</tbody>
</table>

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

```r
# 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
```

Description

On supported architectures CImg can parallelise many operations using OpenMP (e.g. `imager.combine`). Use this function to turn parallelisation on or off.
Usage

cimg.use.openmp(mode = "adaptive", nthreads = 1, verbose = FALSE)
cimg.limit.openmp()

Arguments

mode Either "adaptive", "always" or "none". The default is adaptive (parallelisation for large images only).
nthreads The number of OpenMP threads that imager should use. The default is 1. Set to 0 to get no more than 2, based on OpenMP environment variables.
verbose Whether to output information about the threads being set.

Details

You need to be careful that ‘nthreads’ is not higher than the value in the system environment variable OMP_THREAD_LIMIT (this can be checked with Sys.getenv('OMP_THREAD_LIMIT')). The OMP_THREAD_LIMIT thread limit usually needs to be correctly set before launching R, so using Sys.setenv once a session has started is not certain to work.

Value

NULL (function is used for side effects)

Functions

• cimg.limit.openmp(): Limit OpenMP thread count to no more than 2, based on OpenMP environment variables.

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

circles

Add circles to plot

Description

  Base R has a function for plotting circles called "symbols". Unfortunately, the size of the circles is
  inconsistent across devices. This function plots circles whose radius is specified in used coordinates.

Usage

circles(x, y, radius, bg = NULL, fg = "white", ...)

Arguments

  x       centers (x coordinate)
  y       centers (y coordinate)
  radius  radius (in user coordinates)
  bg      background colour
  fg      foreground colour
  ...     passed to polygon, e.g. lwd

Value

  none, used for side effect
**clean**

*Clean up and fill in pixel sets (morphological opening and closing)*

**Description**

Cleaning up a pixel set here means removing small isolated elements (speckle). Filling in means removing holes. Cleaning up can be achieved by shrinking the set (removing speckle), followed by growing it back up. Filling in can be achieved by growing the set (removing holes), and shrinking it again.

**Usage**

```r
clean(px, ...)
```

```r
fill(px, ...)
```

**Arguments**

- `px`: a pixset
- `...`: parameters that define the structuring element to use, passed on to "grow" and "shrink"

**Functions**

- `fill()`: Fill in holes using morphological closing

**Author(s)**

Simon Barthelme

**Examples**

```r
im <- load.example("birds") %>% grayscale
sub <- imsub(-im,y>380) %>% threshold("85%")
plot(sub)

# Turn into a pixel set
px <- sub==1

layout(t(1:2))
plot(px,main="Before clean-up")
clean(px,3) %>% plot(main="After clean-up")
```
# Fill in the holes
px <- clean(px,3)
plot(px,main="Before filling-in")
fill(px,28) %>% plot(main="After filling-in")

---

**colorise**

*Fill in a colour in an area given by a pixset*

## Description

Paint all pixels in pixset px with the same colour

## Usage

```
colorise(im, px, col, alpha = 1)
```

## Arguments

- **im**: an image
- **px**: either a pixset or a formula, as in imeval.
- **col**: colour to fill in. either a vector of numeric values or a string (e.g. "red")
- **alpha**: transparency (default 1, no transparency)

## Value

an image

## Author(s)

Simon Bartheleme

## Examples

```
im <- load.example("coins")
colorise(im,Xc(im) < 50,"blue") %>% plot
# Same thing with the formula interface
colorise(im,- x < 50,"blue") %>% plot
# Add transparency
colorise(im,- x < 50,"blue",alpha=.5) %>% plot
# Highlight pixels with low luminance values
colorise(im,- . < 0.3,"blue",alpha=.2) %>% plot
```
Various useful pixsets

Description
These functions define some commonly used pixsets. px.left gives the left-most pixels of an image, px.right the right-most, etc. px.circle returns an (approximately) circular pixset of radius r, embedded in an image of width x and height y. Mathematically speaking, the set of all pixels whose L2 distance to the center equals r or less. px.diamond is similar but returns a diamond (L1 distance less than r). px.square is also similar but returns a square (Linf distance less than r).

Usage
px.circle(r, x = 2 * r + 1, y = 2 * r + 1)
px.diamond(r, x = 2 * r + 1, y = 2 * r + 1)
px.square(r, x = 2 * r + 1, y = 2 * r + 1)
px.left(im, n = 1)
px.top(im, n = 1)
px.bottom(im, n = 1)
px.right(im, n = 1)
px.borders(im, n = 1)
px.all(im)
px.none(im)

Arguments
- r: radius (in pixels)
- x: width (default 2*r+1)
- y: height (default 2*r+1)
- im: an image
- n: number of pixels to include

Value
a pixset
Functions

- `px.circle()`: A circular-shaped pixset
- `px.diamond()`: A diamond-shaped pixset
- `px.square()`: A square-shaped pixset
- `px.left(n)`: n left-most pixels (left-hand border)
- `px.top(n)`: n top-most pixels
- `px.bottom(n)`: n bottom-most pixels
- `px.right(n)`: n right-most pixels
- `px.borders(n)`: image borders (to depth n)
- `px.all()`: all pixels in image
- `px.none()`: no pixel in image

Author(s)

Simon Bartheleme

Examples

```r
px.circle(20,350,350) %>% plot(interp=FALSE)
pixel.circle(3) %>% plot(interp=FALSE)
r <- 5
layout(t(1:3))
plot(px.circle(r,20,20))
plot(px.square(r,20,20))
plot(px.diamond(r,20,20))
# These pixsets are useful as structuring elements
px <- grayscale(boats) > .8
grow(px,px.circle(5)) %>% plot
# The following functions select pixels on the left, right, bottom, top of the image
im <- imfill(10,10)
px.left(im,3) %>% plot(int=FALSE)
pixel.right(im,1) %>% plot(int=FALSE)
pixel.top(im,4) %>% plot(int=FALSE)
pixel.bottom(im,2) %>% plot(int=FALSE)
# All of the above
px.borders(im,1) %>% plot(int=FALSE)
```

<table>
<thead>
<tr>
<th>contours</th>
<th>Return contours of image/pixset</th>
</tr>
</thead>
</table>

Description

This is just a light interface over contourLines. See help for contourLines for details. If the image has more than one colour channel, return a list with the contour lines in each channel. Does not work on 3D images.
Usage

countours(x, nlevels, ...)

Arguments

x an image or pixset
nlevels number of contour levels. For pixsets this can only equal two.
... extra parameters passed to contourLines

Value

a list of contours

Author(s)

Simon Barthelme

See Also

highlight

Examples

boats.gs <- grayscale(boats)
c <- contour(boats.gs,nlevels=3)
plot(boats.gs)
#Add contour lines
purrr::walk(ct, function(v) lines(v$x,v$y,col="red"))
#Contours of a pixel set
px <- boats.gs > .8
plot(boats.gs)
c <- contour(px)
#Highlight pixset
purrr::walk(ct, function(v) lines(v$x,v$y,col="red"))

description

Coordinates from pixel index

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

Usage

table

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

```r
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/convolution of image by filter**

Description

The correlation of image `im` by filter `flt` is defined as:

\[
res(x, y, z) = \sum_{i,j,k} im(x + i, y + j, z + k) \times flt(i, j, k)
\]

The convolution of an image `img` by filter `flt` is defined to be:

\[
res(x, y, z) = \sum_{i,j,k} img(x - i, y - j, z - k) \times flt(i, j, k)
\]

Usage

```r
correlate(im, filter, dirichlet = TRUE, normalise = FALSE)
convolve(im, filter, dirichlet = TRUE, normalise = FALSE)
```

Arguments

- `im`: an image
- `filter`: the correlation kernel.
- `dirichlet`: boundary condition. Dirichlet if true, Neumann if false (default TRUE, Dirichlet)
- `normalise`: compute a normalised correlation (ie. local cosine similarity)
Functions

• convolve(): convolve image with filter

Examples

```r
# 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

```r
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
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**

```r
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**

```r
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)).

Description

CImg has its own functions for fast, interactive image plotting. Use this if you get frustrated with slow rendering in RStudio. Note that you need X11 library to use this function.

Usage

display(x, ...)

Arguments

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

See Also

display.cimg, display.imlist

Description

Press escape or close the window to exit. Note that you need X11 library to use this function.

Usage

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

x an image (cimg object)
...
rescale if true pixel values are rescaled to [0-1] (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
...

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 0=Chebyshev | 1=Manhattan | 2=Euclidean | 3=Squared-euclidean.

Examples

```r
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
```

draw_circle  
Draw circle on image

Description

Add circle or circles to an image. Like other native CImg drawing functions, this is meant to be basic but fast. Use implot for flexible drawing.

Usage

draw_circle(im, x, y, radius, color = "white", opacity = 1, filled = TRUE)
**Arguments**

- **im**: an image
- **x**: x coordinates
- **y**: y coordinates
- **radius**: radius (either a single value or a vector of length equal to length(x))
- **color**: either a string ("red"), a character vector of length equal to x, or a matrix of dimension length(x) times spectrum(im)
- **opacity**: scalar or vector of length equal to length(x). 0: transparent 1: opaque.
- **filled**: fill circle (default TRUE)

**Value**

an image

**Author(s)**

Simon Barthelme

**See Also**

implot

**Examples**

draw_circle(boats,c(50,100),c(150,200),30, "darkgreen") %>% plot
draw_circle(boats,125,60, radius=30, col=c(0,1,0), opacity=.2, filled=TRUE) %>% plot

draw_rect

**Draw rectangle on image**

**Description**

Add a rectangle to an image. Like other native CIImg drawing functions, this is meant to be basic but fast. Use implot for flexible drawing.

**Usage**

draw_rect(im, x0, y0, x1, y1, color = "white", opacity = 1, filled = TRUE)
Arguments

- **im**: an image
- **x0**: x coordinate of the bottom-left corner
- **y0**: y coordinate of the bottom-left corner
- **x1**: x coordinate of the top-right corner
- **y1**: y coordinate of the top-right corner
- **color**: either a vector, or a string (e.g. "blue")
- **opacity**: 0: transparent 1: opaque.
- **filled**: fill rectangle (default TRUE)

Value

- an image

Author(s)

Simon Barthelme

See Also

implot, draw_circle

Examples

```r
draw_rect(boats,1,1,50,50,"darkgreen") %>% plot
```

draw_text

**Description**

Like other native CImg drawing functions, this is meant to be basic but fast. Use implot for flexible drawing.

**Usage**

```r
draw_text(im, x, y, text, color, opacity = 1, fsize = 20)
```
Arguments

- `im` an image
- `x` x coord.
- `y` y coord.
- `text` text to draw (a string)
- `color` either a vector or a string (e.g. "red")
- `opacity` 0: transparent 1: opaque.
- `fsize` font size (in pix., default 20)

Value

an image

Author(s)

Simon Barthelme

See Also

`implot, draw_circle, draw_rect`

Examples

```r
draw_text(boats, 100, 100, "Some text", col="black") %>% plot
```

Erode/dilate image by a structuring element.

Description

Erode/dilate image by a structuring element.

Usage

```r
erode(im, mask, boundary_conditions = TRUE, real_mode = FALSE)
erode_rect(im, sx, sy, sz = 1L)
erode_square(im, size)
dilate(im, mask, boundary_conditions = TRUE, real_mode = FALSE)
dilate_rect(im, sx, sy, sz = 1L)
dilate_square(im, size)
```
mopening(im, mask, boundary_conditions = TRUE, real_mode = FALSE)

mopening_square(im, size)

mclosing_square(im, size)

mclosing(im, mask, boundary_conditions = TRUE, real_mode = FALSE)

Arguments

im an image
mask Structuring element.
boundary_conditions Boundary conditions. If FALSE, pixels beyond image boundaries are considered to be 0, if TRUE one. Default: TRUE.
real_mode If TRUE, perform erosion as defined on the reals. If FALSE, perform binary erosion (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
extract_patches

extract_patches(boats,c(10,10),c(10,20),5,5)

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 subject to boundary conditions. The default is to set them to 0 (Dirichlet), other boundary conditions are listed below. - widths and heights should be odd integers (they’re rounded up otherwise).

Usage

extract_patches(im, cx, cy, wx, wy, boundary_conditions = 0L)

extract_patches3D(im, cx, cy, cz, wx, wy, wz, boundary_conditions = 0L)

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)

boundary_conditions : integer. Can be 0 (Dirichlet, default), 1 (Neumann) 2 (Periodic) 3 (mirror).

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**

```r
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**

```r
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)
```
**flatten.alpha**

Flatten alpha channel

### Description

Flatten alpha channel

### Usage

```
flatten.alpha(im, bg = "white")
```

### Arguments

- `im`: an image (with 4 RGBA colour channels)
- `bg`: background: either an RGB image, or a vector of colour values, or a string (e.g. "blue"). Default: white background.

### Value

a blended image

### Author(s)

Simon Barthelme

### See Also

rm.alpha

### Examples

```
# Add alpha channel
alpha <- Xc(grayscale(boats))/width(boats)
boats.a <- imlist(boats, alpha) %>% imappend("c")
flatten.alpha(boats.a) %>% plot
flatten.alpha(boats.a, "darkgreen") %>% plot
```
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
get.stencil

Author(s)
Simon Barthelme

Examples

```r
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 neighbouring pixels.

Usage

```r
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

```r
#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_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").

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_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. Note that grabLine does not support the "pixset" return type. Note that you need X11 library to use these functions.

Usage

```r
grabLine(im, output = "coord")
grabRect(im, output = "coord")
grabPoint(im, output = "coord")
```

Arguments

- `im` an image
- `output` one of "im", "pixset", "coord", "value". Default "coord"

Value

Depending on the value of the output parameter. Either a vector of coordinates (output = "coord"), an image (output = "im"), a pixset (output = "pixset"), or a vector of values (output = "value"). grabLine and grabPoint support the "value" output mode and not the "im" output.

Author(s)

Simon Barthelme

See Also

display

Examples

```r
## Not run: interactive only
## grabRect(boats)
## grabRect(boats, TRUE)
```
grayscale  
*Convert an RGB image to grayscale*

**Description**

This function converts from RGB images to grayscale

**Usage**

```r
grayscale(im, method = "Luma", drop = TRUE)
```

**Arguments**

- `im`: an RGB image
- `method`: either "Luma", in which case a linear approximation to luminance is used, or "XYZ", in which case the image is assumed to be in sRGB color space and CIE luminance is used.
- `drop`: if TRUE returns an image with a single channel, otherwise keep the three channels (default TRUE)

**Value**

a grayscale image (spectrum == 1)

**Examples**

```r
gray_function(boats) %>% plot
# In many pictures, the difference between Luma and XYZ conversion is subtle
gray_function(boats, method="XYZ") %>% plot
gray_function(boats, method="XYZ", drop=FALSE) %>% dim
```

---

grow  
*Grow/shrink a pixel set*

**Description**

Grow/shrink a pixel set through morphological dilation/erosion. The default is to use square or rectangular structuring elements, but an arbitrary structuring element can be given as input. A structuring element is a pattern to be moved over the image: for example a 3x3 square. In "shrink" mode, a element of the pixset is retained only if and only the structuring element fits entirely within the pixset. In "grow" mode, the structuring element acts like a neighbourhood: all pixels that are in the original pixset *or* in the neighbourhood defined by the structuring element belong the new pixset.
Usage

grow(px, x, y = x, z = x, boundary = TRUE)

shrink(px, x, y = x, z = x, boundary = TRUE)

Arguments

px a pixset
x either an integer value, or an image/pixel set.
y width of the rectangular structuring element (if x is an integer value)
z depth of the rectangular structuring element (if x is an integer value)
boundary are pixels beyond the boundary considered to have value TRUE or FALSE (default TRUE)

Functions

• shrink(): shrink pixset using erosion

Examples

#A pixel set:
a <- grayscale(boats) > .8
plot(a)
#Grow by a 8x8 square
grow(a,8) %>% plot
#Grow by a 8x2 rectangle
grow(a,8,2) %>% plot
#Custom structuring element
el <- matrix(1,2,2) %>% as.cimg
all.equal(grow(a,el),grow(a,2))
#Circular structuring element
px.circle(5) %>% grow(a,.) %>% plot

#Sometimes boundary conditions matter
im <- imfill(10,10)
px <- px.all(im)
shrink(px,3,bound=TRUE) %>% plot(main="Boundary conditions: TRUE")
shrink(px,3,bound=FALSE) %>% plot(main="Boundary conditions: FALSE")


gsdim
Grayscale dimensions of image

Description

Shortcut, returns the dimensions of an image if it had only one colour channel
Usage

gsdim(im)

Arguments

im an image

Value

returns c(dim(im)[1:3],1)

Author(s)

Simon Barthelme

Examples

imnoise(dim=gsdim(boats))

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")
highlight

Highlight pixel set on image

Description

Overlay an image plot with the contours of a pixel set. Note that this function doesn’t do the image plotting, just the highlighting.

Usage

highlight(px, col = "red", ...)

Arguments

- **px**: a pixel set
- **col**: color of the contours
- **...**: passed to the “lines” function

Author(s)

Simon Barthelme

See Also

colorise, another way of highlighting stuff

Examples

```r
#Select similar pixels around point (180,200)
px <- px.flood(boats,180,200,sigma=.08)
plot(boats)
#Highlight selected set
highlight(px)
px.flood(boats,18,50,sigma=.08) %>% highlight(col="white",lwd=3)
```

hough_circle

Circle detection using Hough transform

Description

Detects circles of known radius in a pixset. The output is an image where the pixel value at (x,y) represents the amount of evidence for the presence of a circle of radius r at position (x,y). NB: in the current implementation, does not detect circles centred outside the limits of the pixset.
Usage

hough_circle(px, radius)

Arguments

px a pixset (e.g., the output of a Canny detector)
radius radius of circle

Value

a histogram of Hough scores, with the same dimension as the original image.

Author(s)

Simon Bartheleme

Examples

im <- load.example('coins')
px <- cannyEdges(im)
# Find circles of radius 20
hc <- hough_circle(px, 20)
plot(hc)
# Clean up, run non-maxima suppression
nms <- function(im,sigma) { im[dilate_square(im,sigma) != im] <- 0; im}
hc.clean <- isoblur(hc, 3) %>% nms(50)
# Top ten matches
df <- as.data.frame(hc.clean) %>%
dplyr::arrange(desc(value)) %>% head(10)
with(df,circles(x,y,20,fg="red",lwd=3))

Description

Two algorithms are used, depending on the input: if the input is a pixset then the classical Hough transform is used. If the input is an image, then a faster gradient-based heuristic is used. The method returns either an image (the votes), or a data.frame. In both cases the parameterisation used is the Hesse normal form (theta, rho), where a line is represented as the set of values such that cos(theta)*x + sin(theta)*y = rho. Here theta is an angle and rho is a distance. The image form returns a histogram of scores in (rho, theta) space, where good candidates for lines have high scores. The data.frame form may be more convenient for further processing in R: each line represents a pair (rho, theta) along with its score. If the 'shift' argument is true, then the image is assumed to start at x=1, y=1 (more convenient for plotting in R). If false, the image begins at x=0, y=0 and in both cases the origin is at the top left.
**Usage**

```r
hough_line(im, ntheta = 100, data.frame = FALSE, shift = TRUE)
```

**Arguments**

- `im` : an image or pixset
- `ntheta` : number of bins along theta (default 100)
- `data.frame` : return a data.frame? (default FALSE)
- `shift` : if TRUE, image is considered to begin at (x=1,y=1).

**Value**

either an image or a data.frame

**Author(s)**

Simon Barthelme

**Examples**

```r
# Find the lines along the boundary of a square
px <- px.square(30,80,80) %>% boundary
plot(px)
# Hough transform
hough_line(px,ntheta=200) %>% plot

df <- hough_line(px,ntheta=800,data.frame=TRUE)
# Plot lines with the highest score
plot(px)
with(subset(df,score > quantile(score,.9995)),nfline(theta,rho,col="red"))
plot(boats)
df <- hough_line(boats,ntheta=800,data=TRUE)
```

---

**idply**

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

**Description**

Shorthand for `imsplit` followed by `purrr::map_df`

**Usage**

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

Arguments

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

Examples

iiply(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 purrr::map followed by imappend

Usage

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

Arguments

i   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 purrr::map

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

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://asgr.github.io/imager/.

Author(s)
Maintainer: Aaron Robotham <aaron.robotham@uwa.edu.au>
Authors:
• Simon Barthelme <simon.barthelme@gipsa-lab.fr>

Other contributors:
imager.combine

• David Tschumperle [contributor]
• Jan Wijffels [contributor]
• Haz Edine Assemllal [contributor]
• Shota Ochi <shotaochi1990@gmail.com> [contributor]
• Rodrigo Tobar [contributor]

See Also

Useful links:

• https://asgr.github.io/imager/
• https://github.com/asgr/imager/
• Report bugs at https://github.com/asgr/imager/issues

---

**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. "parsort","parrank" and "parorder" aren’t really reductions because they return a list of the same size. They perform a pixel-wise sort (resp. order and rank) across the list.

**Usage**

```r
add(x, na.rm = FALSE)
wsum(x, w, na.rm = FALSE)
average(x, na.rm = FALSE)
mult(x, na.rm = FALSE)
parmax(x, na.rm = FALSE)
parmax.abs(x)
parmin.abs(x)
parmin(x, na.rm = FALSE)
enorm(x)
parmed(x, na.rm = FALSE)
```
parquan(x, prob = 0.5, na.rm = FALSE)
parvar(x, na.rm = FALSE)
parsd(x, na.rm = FALSE)
parall(x)
parany(x)
equal(x)
which.parmax(x)
which.parmin(x)
parsort(x, increasing = TRUE)
parorder(x, increasing = TRUE)
parrank(x, increasing = TRUE)

Arguments

x a list of images
na.rm ignore NAs (default FALSE)
w weights (must be the same length as the list)
prob probability level for parquan, default of 0.5 returns the median
increasing if TRUE, sort in increasing order (default TRUE)

Details

parvar returns an unbiased estimate of the variance (as in the base var function). parsd returns the square root of parvar. parquan returns the specified quantile, and defines this in the same manner as the default R quantile function (type = 7). Using parmed and parquan with quan = 0.5 will return the same result, but parmed will be slightly faster (but only a few percent).

To correctly use multiple threads users should set ‘nthreads’ in cimg.use.openmp. You also need to be careful that this is not higher than the value in the system environment variable OMP_THREAD_LIMIT (this can be checked with Sys.getenv("OMP_THREAD_LIMIT")). The OMP_THREAD_LIMIT thread limit usually needs to be correctly set before launching R, so using Sys.setenv once a session has started is not certain to work.

Functions

• add(): Add images
  • wsum(): Weighted sum of 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 min in absolute value over images,
• `parmin()`: Parallel min over images
• `enorm()`: Euclidean norm (i.e. sqrt(A^2 + B^2 + ...))
• `parmed()`: Parallel Median over images
• `parquan()`: Parallel Quantile over images
• `parvar()`: Variance
• `parsd()`: Std. deviation
• `parall()`: Parallel all (for pixsets)
• `parany()`: Parallel any (for pixsets)
• `equal()`: Test equality
• `which.parmax()`: index of parallel maxima
• `which.parmin()`: index of parallel minima
• `parsort()`: pixel-wise sort
• `parorder()`: pixel-wise order
• `parrank()`: pixel-wise rank

**Author(s)**
Simon Barthelme

**See Also**
imsplit, Reduce

**Examples**

```r
im1 <- as.cimg(function(x,y) x,50,50)
im2 <- as.cimg(function(x,y) y,50,50)
im3 <- as.cimg(function(x,y) cos(x/10),50,50)
l <- imlist(im1,im2,im3)
add(l) %>% plot #Add the images
average(l) %>% plot #Average the images
mult(l) %>% plot #Multiply
wsum(l,c(.1,8,.1)) %>% plot #Weighted sum
parmax(l) %>% plot #Parallel max
parmin(l) %>% plot #Parallel min
parmed(l) %>% plot #Parallel median
parsd(l) %>% plot #Parallel std. dev
#parsort can also be used to produce parallel max. and min
(parsort(l)[[1]]) %>% plot("Parallel min")
```
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(x, ind) <- value: Replace image channel
• R(x) <- value: Replace red channel
• G(x) <- value: Replace green channel
• B(x) <- value: Replace blue channel
• frame(x, ind) <- value: Replace image frame
See Also

imdrow

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**

```r
imappend(list(boats,boats),"x") %>% plot
imappend(list(boats,boats),"y") %>% plot
purrr::map(1:3, ~imnoise(100,100)) %>% imappend("c") %>% plot
boats.gs <- grayscale(boats)
purrr::map(seq(1,5,l=3),function(v) isoblur(boats.gs,v)) %>% imappend("c") %>% plot
#imappend also works on pixsets
imsplit(boats > .5,"c") %>% imappend("x") %>% plot
```

---

**imchange**

**Modify parts of an image**

---

**Description**

A shortcut for modifying parts of an image, using imeval syntax. See doc for imeval first. As part of a pipe, avoids the creating of intermediate variables.

**Usage**

imchange(obj, where, fo, env = parent.frame())
Arguments

obj an image or imlist
where where to modify. a pixset, or a formula (in imeval syntax) that evaluates to a pixset.
fo a formula (in imeval syntax) used to modify the image part
env evaluation environment (see imeval)

Value

a modified image

Author(s)

Simon Barthelme

See Also

imeval

Examples

#Set border to 0:
imchange(boats, px.borders(boats, 10), ~ 0) %>% plot
#Eq. to
im <- boats
im[px.borders(im, 10)] <- 0
#Using formula syntax
imchange(boats, ~ px.borders(., 10), ~ 0)
#Replace with grayscale ramp
imchange(boats, ~ px.borders(., 10), ~ xs) %>% plot
#Kill red channel in image
imchange(boats, ~ c==1, ~ 0) %>% plot
#Shift hue by an amount depending on eccentricity
load.example("parrots") %>%
    RGBtoHSL %>%
imchange(~ c==1, ~ .+80*exp(-(rho/550)^2) ) %>%
    HSLtoRGB %>%
    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
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**

```r
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**

```r
# 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**

\[
\text{imdraw}(\text{im}, \text{sprite}, x = 1, y = 1, z = 1, \text{opacity} = 1)
\]

**Arguments**

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

**Author(s)**

Simon Barthelme

**See Also**

imager.combine, for different ways of combining images

**Examples**

```r
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

Evaluation in an image context

Description

imeval does for images what "with" does for data.frames, namely contextual evaluation. It provides various shortcuts for pixel-wise operations. imdo runs imeval, and reshapes the output as an image of the same dimensions as the input (useful for functions that return vectors). imeval takes inspiration from purrr::map in using formulas for defining anonymous functions using the "." argument. Usage is made clear (hopefully) in the examples. The old version of imeval used CImg's internal math parser, but has been retired.

Usage

```r
imeval(obj, ..., env = parent.frame())
```

```r
imdo(obj, form)
```

Arguments

- **obj**: an image, pixset or imlist
- **...**: one or more formula objects, defining anonymous functions that will be evaluated with the image as first argument (with extra contextual variables added to the evaluation context)
- **env**: additional variables (defaults to the calling environment)
- **form**: a single formula

Functions

- imdo(): run imeval and reshape

Author(s)

Simon Barthelme

See Also

imchange, which modifies specific parts of an image

Examples

```r
## Computing mean absolute deviation
imeval(boats, ~ mean(abs(.-median(.))))
```

```r
##Equivalent to:
mean(abs(boats-median(boats)))
```

```r
##Two statistics
```
imeval(boats, mad = ~ mean(abs(-median(.))), sd = ~ sd(.))

## imeval can precompute certain quantities, like the x or y coord. of each pixel
imeval(boats, ~ x) %>% plot
## same as Xc(boats) %>% plot
## Other predefined quantities:
## w is width, h is height
imeval(boats, ~ x/w) %>% range
## It defines certain transformed coordinate systems:
## Scaled x,y,z
## xs = x/w
## ys = y/h
## Select upper-left quadrant (returns a pixset)
imeval(boats, ~ xs < .5 & ys < .5) %>% plot
## Fade effect
imeval(boats, ~ xs*.%) %>% plot
## xc and yc are another set of transformed coordinates
## where xc=0, yc=0 is the image center
imeval(boats, ~ (abs(xc)/w)*.) %>% plot

## rho, theta: circular coordinates. rho is distance to center (in pix.), theta angle
## Gaussian mask with sd 10 pix.
blank <- imfill(30, 30)
imeval(blank, ~ dnorm(rho, sd=w/3)) %>% plot(int=FALSE)
imeval(blank, ~ theta) %>% plot
## imeval is made for interactive use, meaning it accesses the environment it got called from, e.g. this works:
f <- function()
{
  im1 <- imfill(3, 3, val=1)
im2 <- imfill(3, 3, val=3)

  imeval(im1, ~ .+im2)
}
f()
## imeval accepts lists as well
map_il(1:3, ~ isoblur(boats, .)) %>%
  imeval(~ xs*.%) %>%
  plot

## imeval is useful for defining pixsets:
## here, all central pixels that have value under the median
greyscale(boats) %>%
  imeval(~ (. > median(.)) & rho < 150) %>%
  plot
## Other abbreviations are defined:
## s for imshift, b for isoblur, rot for imrotate.
## e.g.
imeval(boats, ~ .*s(.,3)) %>% plot

# The rank function outputs a vector
greyscale(boats) %>% rank %>% class
# Auto-reshape into an image
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. If val is a logical value, creates a pixset instead.

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, or a character string for a colour (e.g. "blue")
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(20,20,val="red") %>% plot #Same, using R colour name
imfill(3,3,val=FALSE) #Pixset
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 = "xy", 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"). Default: "xy"

scheme
numerical scheme (default '3', rotation invariant)

Value

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

Author(s)

Simon Bartheleme

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 <- purrr::map_df(someFiles,function(v) iminfo(v) %>% as.data.frame)

## End(Not run)
**imlap**

Compute image Laplacian

---

**Description**

The Laplacian is the sum of second derivatives, approximated here using finite differences.

**Usage**

imlap(im)

**Arguments**

- im: an image

**Examples**

imlap(boats) %>% plot

---

**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 DEPRECATION NOTE: in v0.30 of imager, the original behaviour of the "imlist" function was to take a list and turn it into an image list. This behaviour has now been changed to make "imlist" be more like "list". If you wish to turn a list into an image list, use as.imlist.

**Usage**

imlist(...)

**Arguments**

... images to be included in the image list

**See Also**

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

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
**implot**

Plot objects on image using base graphics

**Description**

This function lets you use an image as a canvas for base graphics, meaning you can use R functions like "text" and "points" to plot things on an image. The function takes as argument an image and an expression, executes the expression with the image as canvas, and outputs the result as an image (of the same size).

**Usage**

```r
implot(im, expr, ...)
```

**Arguments**

- `im`: an image (class cimg)
- `expr`: an expression (graphics code to execute)
- `...`: passed on to `plot.cimg`, to control the initial rendering of the image (for example the colorscale)

**Value**

an image

**Author(s)**

Simon Bartheleme

**See Also**

plot, capture.plot

**Examples**

```r
## Not run:

b.new <- implot(boats, text(150, 50, "Boats!!!", cex=3))
plot(b.new)

# Draw a line on a white background
bg <- imfill(150, 150, val=1)
implot(bg, lines(c(50, 50), c(50, 100), col="red", lwd=4))%>%plot

# You can change the rendering of the initial image
im <- grayscale(boats)
draw.fun <- function() text(150, 50, "Boats!!!", cex=3)
out <- implot(im, draw.fun(), colorscale=function(v) rgb(0, v, v), rescale=FALSE)
plot(out)

## End(Not run)
```
imrep

Replicate images

Description

Kinda like rep, for images. Copy image n times and (optionally), append.

Usage

imrep(x, n = 1, axis = NULL)

Arguments

x         an image
n         number of replications
axis      axis to append along (one of NULL, "x","y","z","c"). Default: NULL

Value

either an image or an image list

Author(s)

Simon Barthelme

Examples

#Result is a list
imrep(boats,3) %>% %>% plot
#Result is an image
imrep(boats,3,"x") %>% plot
#Make an animation by repeating each frame 10x
#map_il(1:5,. isoblu(boats,.)) %>% imrep(10,"z") %>%
#     imappend("z") %>% play

imrotate

Rotate an image along the XY plane.

Description

If cx and cy aren’t given, the default is to centre the rotation in the middle of the image. When cx and cy are given, the algorithm used is different, and does not change the size of the image.

Usage

imrotate(im, angle, cx, cy, interpolation = 1L, boundary = 0L)
Arguments

- **im**: an image
- **angle**: Rotation angle, in degrees.
- **cx**: Center of rotation along x (default, image centre)
- **cy**: Center of rotation along y (default, image centre)
- **interpolation**: Type of interpolation. One of 0=nearest,1=linear,2=cubic.
- **boundary**: Boundary conditions. One of 0=dirichlet, 1=neumann, 2=periodic

See Also

- `imwarp`, for flexible image warping, which includes rotations as a special case

Examples

```r
imrotate(boats,30) %>% plot
#Shift centre to (20,20)
imrotate(boats,30,cx=20,cy=20) %>% plot
```

---

**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

```r
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

```r
layout(t(1:2))
plot(boats,main="Original")
imsharpen(boats,150) %>% plot(main="Sharpened")
```
**imshift**  
*Shift image content.*

**Description**

Shift image content.

**Usage**

```r
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**

```r
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.
### imsplit

#### Usage

```r
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, i.e. `split(im,"c")` produces a list containing all individual colour channels.

#### See Also

- `imappend` (the reverse operation)

#### Examples

```r
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(boats,"z",2) %>% imappend("z") #Split and reshape into a single image
#You can also split pixsets
imsplit(boats > .5,"c") %>% plot
```

### imsub

#### 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

```r
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
all.equal(imsub(parrots, x %in% 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)
Usage

```r
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

```r
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**

```r
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**

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

Arguments

im an image
cords 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

cord.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

 inpaint Fill-in NA values in an image

Description

Fill in NA values (inpainting) using a Gaussian filter, i.e. replace missing pixel values with a weighted average of the neighbours.

Usage

inpaint(im, sigma)

Arguments

im input image
sigma std. deviation of the Gaussian (size of neighbourhood)

Value

an image with missing values filled-in.
Author(s)

Simon Barthelme

Examples

```r
im <- boats
im[sample(nPix(im), 1e4)] <- NA
inpaint(im, 1) %>% imlist(im, .) %>%
  setNames(c("before", "after")) %>% plot(layout = "row")
```

Description

To explore the effect of certain image manipulations, filter settings, etc., it’s useful to have a basic interaction mechanism. You can use shiny for that, but imager provides a lightweight alternative. The user writes a function that gets called every time a user event happens (a click, a keypress, etc.). The role of the function is to process the event and output an image, which will then be displayed. You can exit the interface at any time by pressing Esc. See examples for more. This feature is experimental!!! Note that you need X11 library to use this function.

Usage

```r
interact(fun, title = ", init)
```

Arguments

- `fun`: a function that takes a single argument (a list of user events) and returns an image to be plotted. The image won’t be rescaled before plotting, so make sure RGB values are in [0,1].
- `title`: a title for the window (default ", none)
- `init`: initial image to display (optional)

Value

an image, specifically the last image displayed

Author(s)

Simon Barthelme
Examples

```
#Implement a basic image gallery:
#press "right" and "left" to view each image in a list
gallery <- function(iml)
{
    ind <- 1
    f <- function(state)
    {
        if (state$key=="arrowleft")
        {
            ind <<- max(ind-1,1)
        }
        if (state$key=="arrowright")
        {
            ind <<- min(ind+1,length(iml))
        }
        iml[[ind]]
    }
    interact(f)
}
##Not run (interactive only)
##map_il(1:10,~ isoblur(boats,.)) %>% gallery
```

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

is.imlist  Check that an object is an imlist object

Description

Check that an object is an imlist object

Usage

is.imlist(x)

Arguments

x  an object

Value

logical
is.pixset

Check that an object is a pixset object

Description

Check that an object is a pixset object

Usage

is.pixset(x)

Arguments

x

an object

Value

logical

isoblur

Blur image isotropically.

Description

Blur image isotropically.

Usage

isoblur(im, sigma, neumann = TRUE, gaussian = TRUE, na.rm = FALSE)

Arguments

im

an image

sigma

Standard deviation of the blur (positive)

neumann

If true, use Neumann boundary conditions, Dirichlet otherwise (default true, Neumann)

gaussian

Use a Gaussian filter (actually van Vliet-Young). Default: 0th-order Deriche filter.

na.rm

if TRUE, ignore NA values. Default FALSE, in which case the whole image is NA if one of the values is NA (following the definition of the Gaussian filter)

See Also

deriche, vanvliet, inpaint, medianblur
Examples

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

label

*Label connected components.*

Description


Usage

```r
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

```r
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")
```
lippy

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 purrr::map followed by imappend

Usage
lippy(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)
lippy(c(10,50,100),build.im,"y") %>% plot

load.dir

Load all images in a directory

Description
Load all images in a directory and return them as an image list.

Usage
load.dir(path, pattern = NULL, quiet = FALSE)

Arguments
path directory to load from
pattern optional: file pattern (ex. *.jpg). Default NULL, in which case we look for file extensions png,jpeg,jpg,tif,bmp.
quiet if TRUE, loading errors are quiet. If FALSE, they are displayed. Default FALSE
Value

an image list

Author(s)

Simon Barthelme

Examples

```r
path <- system.file(package="imager") %>% paste0("/extdata")
load.dir(path)
```

Description

Imager ships with five 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. The "coins" image comes from scikit-image. The Hubble Deep field (hubble) is from Wikimedia. The test video ("tennis") comes from [xiph.org](https://media.xiph.org/video/derf/)’s collection.

Usage

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

Arguments

name name of the example

Value

an image

Author(s)

Simon Barthelme

Examples

```r
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 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

load.video

Load a video using ffmpeg

Description

You need to have ffmpeg on your path for this to work. This function uses ffmpeg to split the video into individual frames, which are then loaded as images and recombined. Videos are memory-intensive, and load.video performs a safety check before loading a video that would be larger than maxSize in memory (default 1GB)
load.video

Usage

load.video(
  fname,
  maxSize = 1,
  skip.to = 0,
  frames = NULL,
  fps = NULL,
  extra.args = "",
  verbose = FALSE
)

Arguments

fname  file to load
maxSize max. allowed size in memory, in GB (default max 1GB).
skip.to skip to a certain point in time (in sec., or "hh:mm:ss" format)
frames number of frames to load (default NULL, all)
fps frames per second (default NULL, determined automatically)
extra.args extra arguments to be passed to ffmpeg (default "", none)
verbose if TRUE, show ffmpeg output (default FALSE)

Value

an image with the extracted frames along the "z" coordinates

Author(s)

Simon Barthelme

See Also

save.video, make.video

Examples

fname <- system.file('extdata/tennis_sif.mpeg',package='imager')
## Not run
## load.video(fname) %>% play
## load.video(fname,fps=10) %>% play
## load.video(fname,skip=2) %>% play
Convert a magick image to a cimg image or image list and vice versa

Description

The magick library package stores its data as "magick-image" object, which may in fact contain several images or an animation. These functions convert magick objects into imager objects or imager objects into magick objects. Note that cimg2magick function requires magick package.

Usage

```r
magick2imlist(obj, alpha = "rm", ...)  
magick2cimg(obj, alpha = "rm", ...)  
cimg2magick(im, rotate = TRUE)
```

Arguments

- `obj` an object of class "magick-image"
- `alpha` what do to with the alpha channel ("rm": remove and store as attribute, "flatten": flatten, "keep": keep). Default: "rm"
- `...` ignored
- `im` an image of class cimg
- `rotate` determine if rotate image to adjust orientation of image

Value

an object of class cimg or imlist  
an object of class "magick-image"

Author(s)

Jan Wijffels, Simon Barthelme  
Shota Ochi

See Also

flatten.alpha, rm.alpha
Description

You need to have ffmpeg on your path for this to work. This function uses ffmpeg to combine individual frames into a video. save.video can be called directly with an image or image list as input. make.video takes as argument a directory that contains a sequence of images representing individual frames to be combined into a video.

Usage

```r
make.video(
  dname,
  fname,
  pattern = "image-%d.png",
  fps = 25,
  extra.args = "",
  verbose = FALSE
)
```

```r
save.video(im, fname, ...)
```

Arguments

dname: name of a directory containing individual files
fname: name of the output file. The format is determined automatically from the name (example "a.mpeg" will have MPEG format)
pattern: pattern of filename for frames (the default matches "image-1.png", "image-2.png", etc.. See ffmpeg documentation for more).
fps: frames per second (default 25)
extra.args: extra arguments to be passed to ffmpeg (default "", none)
verbose: if TRUE, show ffmpeg output (default FALSE)
im: an image or image list
...: extra arguments to save.video, passed on to make.video

Functions

- `save.video()`: Save a video using ffmpeg

Author(s)

Simon Barthelme
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(...)

map2_il(...)

pmap_il(...)

Arguments

... passed to map

Value

an image list

Functions

- map2_il(): Parallel map (two values)
- pmap_il(): Parallel map (multiple values)

Examples

```r
## Not run
## iml <- map_il(seq(0,20,l=60),~ isoblur(boats,.))
## f <- tempfile(fileext=".avi")
## save.video(iml,f)
## load.video(f) %>% play
## #Making a video from a directory
## dd <- tempdir()
## for (i in 1:length(iml)) {
##   png(sprintf("%s/image-%i.png",dd,i));
##   plot(iml[[i]]); dev.off() }
## make.video(dd,f)
## load.video(f) %>% play
```
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.

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

mutate_plyr
Mutate a data frame by adding new or replacing existing columns.

Description
This function copied directly from plyr, and modified to use a different name to avoid namespace collisions with dplyr/tidyverse functions.

Usage
mutate_plyr(.data, ...)

Arguments
.data the data frame to transform
... named parameters giving definitions of new columns.

Details
This function is very similar to transform but it executes the transformations iteratively so that later transformations can use the columns created by earlier transformations. Like transform, unnamed components are silently dropped.
Mutate seems to be considerably faster than transform for large data frames.
**nfline**  
*Plot a line, Hesse normal form parameterisation*

**Description**

This is a simple interface over abline meant to be used along with the Hough transform. In the Hesse normal form \((\theta, \rho)\), a line is represented as the set of values \((x, y)\) such that \(\cos(\theta) x + \sin(\theta) y = \rho\). Here \(\theta\) is an angle and \(\rho\) is a distance. See the documentation for `hough_lines`.

**Usage**

```r
nfline(theta, rho, col, ...)
```

**Arguments**

- `theta`: angle (radians)
- `rho`: distance
- `col`: colour
- `...`: other graphical parameters, passed along to `abline`

**Value**

nothing

**Author(s)**

Simon Barthelme

**Examples**

```r
# Boring example, see ?hough_lines
plot(boats)
nfline(theta=0, rho=10, col="red")
```
Description

Pad image with n pixels along specified axis

Usage

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

Arguments

- `im`: the input image
- `nPip`: how many pixels to pad with
- `axes`: which axes to pad along
- `pos`: -1: prepend 0: center 1: append
- `val`: colour of the padded pixels (default 0 in all channels). Can be a string for colour images, e.g. "red", or "black".

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
pad(boats,20, pos=1, "xy", val="red") %>% 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

```r
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) %>% dplyr::mutate(w=3,h=3)
# Extract patch summary
out <- dplyr::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 (2011)

Description

Moisan (2011) defines an additive image decomposition $im = \text{periodic} + \text{smooth}$ 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.
permute_axes

Usage
   periodic.part(im)

Arguments
   im             an image

Value
   an image

Author(s)
   Simon Barthelme

References

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., "zyc" to have the z-axis come first
Examples

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

**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**

```r
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**

```r
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
```
Description

Pixel sets represent sets of pixels in images (ROIs, foreground, etc.). From an implementation point of view, they’re just a thin layer over arrays of logical values, just like the cimg class is a layer over arrays of numeric values. Pixsets can be turned back into logical arrays, but they come with a number of generic functions that should make your life easier. They are created automatically whenever you run a test on an image (for example `im > 0` returns a pixset).

Usage

`pixset(x)`

Arguments

- `x` an array of logical values

Examples

```r
# A test on an image returns a pixset
boats > 250
# Pixsets can be combined using the usual Boolean operators
(boats > 230) & (Xc(boats) < width(boats)/2)
# Subset an image using a pixset
boats[boats > 250]
# Turn a pixset into an image
as.cimg(boats > 250)
# Equivalently:
(boats > 250) + 0
```

Description

A very basic video player. Press the space bar to pause and ESC to close. Note that you need X11 library to use this function.

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,
  axes = TRUE,
  main = "",
  xaxs = "i",
  yaxs = "i",
  asp = 1,
  col.na = rgb(0, 0, 0, 0),
  ...
)
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`: an optional colour scale (default is gray or rgb)
- `interpolate`: should the image be plotted with antialiasing (default TRUE)
- `axes`: Whether to draw axes (default TRUE)
- `main`: Main title
- `xaxs`: The style of axis interval calculation to be used for the x-axis. See ?par
- `yaxs`: The style of axis interval calculation to be used for the y-axis. See ?par
- `asp`: aspect ratio. The default value (1) means that the aspect ratio of the image will be kept regardless of the dimensions of the plot. A numeric value other than one changes the aspect ratio, but it will be kept the same regardless of dimensions. Setting `asp="varying"` means the aspect ratio will depend on plot dimensions (this used to be the default in versions of imager < 0.40)
- `col.na`: which colour to use for NA values, as R rgb code. The default is "rgb(0,0,0)", which corresponds to a fully transparent colour.
- `...`: other parameters to be passed to plot.default (e.g. "main")

See Also

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

Examples

```r
plot(boats, main="Boats")
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

```r
imsplit(boats,"c") # Returns an image list
imsplit(boats,"c") %>% plot
imsplit(boats,"c") %>% plot(layout="row")
imsplit(boats,"c") %>% plot(layout="col")
imsplit(boats,"x",5) %>% plot(layout="rect")
```

---

**px.flood**

*Select a region of homogeneous colour*

Description

Select pixels that are similar to a seed pixel. 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 pixel set containing the selected pixels.

Usage

```r
px.flood(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 flood
- `y`: Y-coordinate of the starting point of the region to flood
- `z`: Z-coordinate of the starting point of the region to flood
- `sigma`: Tolerance concerning neighborhood values.
- `high_connexity`: Use 8-connexity (only for 2d images, default FALSE).

Details

Old name: selectSimilar (deprecated)

See Also

bucketfill

Examples

```r
# Select part of a sail
px <- px.flood(boats,x=169,y=179,sigma=.2)
plot(boats)
highlight(px)
```
px.na

Description
A pixset containing all NA pixels

Usage
px.na(im)

Arguments
im an image

Value
a pixset

Examples
im <- boats
im[1] <- NA
px.na(im)

px.remove_outer

Description
Remove all connected regions that touch image boundaries

Usage
px.remove_outer(px)

Arguments
px a pixset

Value
a pixset
Author(s)
Simon Barthelme

Examples

```r
im <- draw_circle(imfill(100,100),c(0,50,100),c(50,50,50),radius=10,color=1)
plot(im)
as.pixset(im) %>% px.remove_outer %>% plot
```

Description

The raster library stores its data as "RasterLayer" and "RasterBrick" objects. The raster package can store its data out-of-RAM, so in order not to load too much data the "maxpixels" argument sets a limit on how many pixels are loaded.

Usage

```r
## S3 method for class 'RasterLayer'
as.cimg(obj, maxpixels = 1e+07, ...)

## S3 method for class 'RasterStackBrick'
as.imlist(obj, maxpixels = 1e+07, ...)
```

Arguments

- `obj` an object of class "RasterLayer"
- `maxpixels` max. number of pixels to load (default 1e7)
- `...` ignored

Author(s)
Simon Barthelme, adapted from the image method for RasterLayer by Robert J Hijmans
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

Description
If the dimension arguments are negative, they are interpreted as a proportion of the original image.

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).

See Also

See imresize for an easier interface.

| 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, interpolation = 3)
Arguments

- **im**: an image
- **scale**: a scale factor
- **interpolation**: interpolation method to use (see doc for `resize`). Default 3, linear. Set to 5 for cubic, 6 for Lanczos (higher quality).

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, triple-scale, etc. uses an anisotropic scaling algorithm described in: [http://www.scale2x.it/algorithm.html](http://www.scale2x.it/algorithm.html). For half-scaling uses what the CImg doc describes as an "optimised filter", see `resize_halfXY` in CImg.h.

See Also

- `resize`

Examples

```r
im <- load.example("parrots")
imresize(im,1/4) #Quarter size
map_iil(2:4,~ imresize(im,1/.) ) %>% imappend("x") %>% plot
```
Description

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

Usage

RGBtoHSL(im)
RGBtoXYZ(im)
XYZtoRGB(im)
HSLtoRGB(im)
RGBtoHSV(im)
HSVtoRGB(im)
RGBtoHSI(im)
HSItoRGB(im)
RGBtosRGB(im)
sRGBtoRGB(im)
RGBtoYCbCr(im)
YCbCrtoRGB(im)
RGBtoYUV(im)
YUVtoRGB(im)
LabtoRGB(im)
RGBtoLab(im)
LabtoXYZ(im)
XYZtoLab(im)
LabtosRGB(im)
sRGBtoLab(im)

Arguments

im an image

Functions

- RGBtoHSL(): RGB to HSL conversion
- RGBtoXYZ(): CIE RGB to CIE XYZ (1931) conversion, D65 white point
- XYZtoRGB(): CIE XYZ to CIE RGB (1931) conversion, D65 white point
- 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
- LabtoRGB(): Lab to RGB (linear)
- RGBtoLab(): RGB (linear) to Lab
- LabtoXYZ(): Lab to XYZ
- XYZtoLab(): XYZ to Lab
- LabtosRGB(): Lab to sRGB
- sRGBtoLab(): sRGB to Lab

---

rm.alpha Remove alpha channel and store as attribute

Description

Remove alpha channel and store as attribute

Usage

rm.alpha(im)

Arguments

im an image with 4 RGBA colour channels
rotate_xy

Value

an image with only three RGB channels and the alpha channel as attribute

Author(s)

Simon Barthelme

See Also

flatten.alpha

Examples

#An image with 4 colour channels (RGBA)
im <- imfill(2,2,val=c(0,0,0,0))
#Remove fourth channel
rm.alpha(im)
attr(rm.alpha(im),"alpha")

---

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, 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.
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.

Usage

save.image(im, file, quality = 0.7)

Arguments

im
an image (of class cimg)
file
path to file. The format is determined by the file’s name
quality
(JPEG only) default 0.7. Higher quality means less compression.

Value

nothing

See Also

save.video

Examples

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

split_connected

Split pixset into connected components

Description

Compute connected components (using “label”), then split into as many sets as there are compo- nents. Useful for segmentation

Usage

split_connected(px, ...)


**Arguments**

- **px**: a pixset
- **...**: further arguments passed to label

**Value**

- a list of pixsets

**Author(s)**

- Simon Barthelme

**See Also**

- label

**Examples**

```r
px <- isoblur(grayscale(boats),5) > .75
plot(px)
spl <- split_connected(px)
plot(spl[[1]])
px <- isoblur(grayscale(boats),5) > .75
plot(px)
spl <- split_connected(px)
plot(spl[[1]])
```

---

**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**

```r
squeeze(x)
```

**Arguments**

- **x**: an array

**Examples**

```r
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**

\[
\text{stencil.cross}(z = \text{FALSE}, \ cc = \text{FALSE}, \ \text{origin} = \text{FALSE})
\]

**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 \(\text{thr} = \text{"auto"}\) a threshold will be computed automatically using kmeans (i.e., 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 \(\text{approx} = \text{TRUE}\) 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 iapply(im,"c",threshold) to find optimal values for each channel separately.

**Usage**

\[
\text{threshold}(\text{im}, \ \text{thr} = \text{"auto"}, \ \text{approx} = \text{TRUE}, \ \text{adjust} = 1)
\]
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)
- **adjust**: use to adjust the automatic threshold: if the auto-threshold is at k, effective threshold will be at adjust\*k (default 1)

Value

a pixset with the selected pixels

Author(s)

Simon Barthelme

Examples

```r
im <- load.example("birds")
im.g <- grayscale(im)
threshold(im.g,"15%") %>% plot
threshold(im.g,"auto") %>% plot
threshold(im.g,.1) %>% plot
#if auto-threshold is too high, adjust downwards or upwards
#using "adjust"
threshold(im,adjust=.5) %>% plot
threshold(im,adjust=1.3) %>% plot
```

Description

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

Usage

```r
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. One of 'x', 'y', 'z', 'c'
- **neumann**: If true, use Neumann boundary conditions (default false, Dirichlet)
References

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 0=nearest | 1=linear | 2=cubic.
- **boundary_conditions**: Boundary conditions. Can be 0=dirichlet | 1=neumann | 2=periodic.

**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
```
where

Return locations in pixel set

**Description**

Return locations in pixel set

**Usage**

where(x)

---

**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**

```r
# 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")
```
Arguments

\( x \)  
a pixset

Examples

#All pixel locations with value greater than .99
where(boats > .99)

%inr%

Check that value is in a range

Description

A shortcut for \( x \geq a \) \( \text{or} \) \( x \leq b \).

Usage

\( x \ %inr\ % range \)

Arguments

\( x \)  
numeric values

\( \text{range} \)  
a vector of length two, of the form \( c(a,b) \)

Value

a vector of logicals 1:10

Author(s)

Simon Barthelme
Index

* datasets
  boats, 23
  %inr%, 131
  add (imager.combine), 66
  add.color (add.colour), 5
  add.colour, 5
  as.cimg, 6
  as.cimg.array, 7
  as.cimg.data.frame, 8
  as.cimg.function, 9
  as.cimg.im, 10
  as.cimg.pixset (as.pixset), 17
  as.cimg.raster, 11
  as.cimg.RasterLayer (RasterPackage), 118
  as.data.frame.cimg, 11
  as.data.frame.imlist, 12
  as.data.frame.pixset, 13
  as.igraph.cimg, 13
  as.igraph.pixset, 15
  as.imlist (as.imlist.list), 16
  as.imlist.list, 16
  as.imlist.RasterStackBrick (RasterPackage), 118
  as.pixset, 17
  as.raster.cimg, 18
  at, 19
  at<- (at), 19
  autocrop, 20
  average (imager.combine), 66
  B (cimg.extract), 31
  B<- (imager.replace), 69
  bbox, 21
  blur_anisotropic, 22
  boats, 23
  boundary, 23
  boxblur, 24
  boxblur_xy, 25
  bucketfill, 25
  cannyEdges, 26
  capture.plot, 27
  Cc (imcoord), 72
  center.stencil, 28
  channel (cimg.extract), 31
  channel<- (imager.replace), 69
  channels, 28
  ci, 29
  cimg, 30
  cimg dimensions, 30
  cimg.extract, 31
  cimg.limit.openmp (cimg.openmp), 32
  cimg.openmp, 32
  cimg.use.openmp, 67
  cimg.use.openmp (cimg.openmp), 32
  cimg2im, 33
  cimg2magick (magick), 101
  circles, 34
  clean, 35
  color.at (at), 19
  color.at<- (at), 19
  colorise, 36
  common_pixsets, 37
  contours, 38
  convert_pixset (as.data.frame.pixset), 13
  convolve (correlate), 40
  coord.index, 39
  correlate, 40
  crop.bbox (bbox), 21
  crop.borders, 41
  depth (cimg.dimensions), 30
  deriche, 42
  diffusion_tensors, 43
  dilate (erode), 49
  dilate_rect (erode), 49
  dilate_square (erode), 49
  displacement, 43
  display, 44
display.cimg, 44
display.list, 45
distance_transform, 46
draw_circle, 46
draw_rect, 47
draw_text, 48

enorm (imager.combine), 66
equal (imager.combine), 66
erode, 49
erode_rect (erode), 49
erode_square (erode), 49
extract_patches, 51
extract_patches3D (extract_patches), 51

FFT, 52
fill (clean), 35
flatten.alpha, 53
frame (cimg.extract), 31
frame<~ (imager.replace), 69
frames, 54

G (cimg.extract), 31
G<~ (imager.replace), 69
get.locations, 54
get.stencil, 55
getgradient, 56
getheessian, 56
grab, 57
grabLine (grab), 57
grabPoint (grab), 57
grabRect (grab), 57
grayscale, 58
grow, 58
gsdim, 59

haar, 60
height (cimg.dimensions), 30
highlight, 61
hough_circle, 61
hough_line, 62
HSIToRGB (RGBtoHSL), 122
HSLToRGB (RGBtoHSL), 122
HSVtoRGB (RGBtoHSL), 122

idply, 63
iiply, 64
iilply, 65
im_split, 90

imager, 65
imager-package (imager), 65
imager.colourspaces (RGBtoHSL), 122
imager.combine, 32, 66
imager.replace, 69
imager.subset, 70
imappend, 71
imchange, 71
imcol (cimg.extract), 31
imcoord, 72
imdirc, 74
imdo (imeval), 76
imdraw, 75
imeval, 76
imfill, 78
imgradient, 79
imhessian, 79
iminfo, 80
imlap, 81
imlist, 81
imnoise, 82
implot, 83
imrep, 84
imresize (resize_doubleXY), 120
imrotate, 84
imrow (cimg.extract), 31
imsharpen, 85
imshift, 86
imsplit, 86
imsup, 87
imwarp, 88
index.coord, 90
inpaint, 91
interact, 92
interp, 93
is.cimg, 94
is.imlist, 94
is.pixset, 95
isoblur, 95

label, 96
LabtoRGB (RGBtoHSL), 122
LabtossRGB (RGBtoHSL), 122
LabtoXYZ (RGBtoHSL), 122
liply, 97
load.dir, 97
load.example, 98
load.image, 99
load.video, 99
INDEX

magick, 101
magick2cimg (magick), 101
magick2imlist (magick), 101
make.video, 102
map2 Il (map Il), 103
map Il, 103
mclosing (erode), 49
mclosing_square (erode), 49
medianblur, 104
mirror, 105
mopening (erode), 49
mopening_square (erode), 49
mult (imager.combine), 66
mutate_plyr, 105
nfline, 106
nPix (cimg.dimensions), 30
pad, 107
parall (imager.combine), 66
parany (imager.combine), 66
parmax (imager.combine), 66
parmed (imager.combine), 66
parmin (imager.combine), 66
parorder (imager.combine), 66
parquann (imager.combine), 66
parquann (imager.combine), 66
parranks (imager.combine), 66
patch_summary_cimg, 109
patchstat, 108
periodic.part, 109
permute_axes, 110
pixel.grid, 111
pixset, 112
play, 112
plot.cimg, 113
plot.imlist, 115
pmap Il (map Il), 103
px.all (common_pixsets), 37
px.borders (common_pixsets), 37
px.bottom (common_pixsets), 37
px.circle (common_pixsets), 37
px.diamond (common_pixsets), 37
px.flood, 116
px.left (common_pixsets), 37
px.na, 117
px.none (common_pixsets), 37
px.remove_outer, 117
px.right (common_pixsets), 37
px.square (common_pixsets), 37
px.top (common_pixsets), 37
R (cimg.extract), 31
R<- (imager.replace), 69
RasterPackage, 118
renorm, 119
resize, 119
resize_doubleXY, 120
resize_halfXY (resize_doubleXY), 120
resize_tripleXY (resize_doubleXY), 120
resize_uniform (resize_doubleXY), 120
RGBtoHSI (RGBtoHSL), 122
RGBtoHSL, 122
RGBtoHSV (RGBtoHSL), 122
RGBtoLab (RGBtoHSL), 122
RGBtosRGB (RGBtoHSL), 122
RGBtoXYZ (RGBtoHSL), 122
RGBtoYCbCr (RGBtoHSL), 122
RGBtoYUV (RGBtoHSL), 122
rm.alpha, 123
rotate_xy, 124
save.image, 125
save.video (make.video), 102
shrink (grow), 58
spectrum (cimg.dimensions), 30
split_connected, 125
squeeze, 126
sRGBtoLab (RGBtoHSL), 122
sRGBtoRGB (RGBtoHSL), 122
stencil.cross, 127
subim (imsub), 87
threshold, 127
transform, 105
vanvllet, 128
warp, 129
watershed, 130
where, 130
which.parmax (imager.combine), 66
which.parmin (imager.combine), 66
width (cimg.dimensions), 30
wsum (imager.combine), 66
Xc (imcoord), 72
INDEX

XYZtoLab (RGBtoHSL), 122
XYZtoRGB (RGBtoHSL), 122

Yc (imcoord), 72
YCbCrtoRGB (RGBtoHSL), 122
YUVtoRGB (RGBtoHSL), 122

Zc (imcoord), 72