Package ‘imager’

June 5, 2015

Type  Package
Title  Image processing library based on CImg
Version  0.1
Date  2015-05-07
Author  Simon Barthelmé <simon.barthelme@gipsa-lab.fr>
Maintainer  Simon Barthelmé <simon.barthelme@gipsa-lab.fr>
Description  imager provides fast image processing functions for images in up to 4 dimensions.
License  CeCILL-C (V1)
Imports  Rcpp (>= 0.11.5)
Depends  plyr, magrittr, stringr, grDevices
LinkingTo  Rcpp

R topics documented:

  add.colour  ......................................................... 3
  as.cimg.array  .................................................. 4
  as.cimg.data.frame  ............................................. 4
  as.cimg.function  ............................................... 5
  as.data.frame.cimg  ............................................. 6
  as.im.cimg  ..................................................... 6
  as.raster.cimg  ................................................ 7
  at  ................................................................. 8
  autocrop  ......................................................... 8
  B  ................................................................. 9
  blur_anisotropic  ............................................... 9
  boxblur  .......................................................... 10
  boxblur_xy  ........................................................ 10
  bucket_fill  ...................................................... 11
  bucket_select  .................................................. 11
  capture.plot  .................................................... 12
  center.stencil  .................................................. 12
  channel  ........................................................ 12
<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>channels</td>
<td>13</td>
</tr>
<tr>
<td>cimg</td>
<td>13</td>
</tr>
<tr>
<td>convolve</td>
<td>14</td>
</tr>
<tr>
<td>correlate</td>
<td>15</td>
</tr>
<tr>
<td>deriche</td>
<td>15</td>
</tr>
<tr>
<td>diffusion_tensors</td>
<td>16</td>
</tr>
<tr>
<td>dilate</td>
<td>16</td>
</tr>
<tr>
<td>dilate_rect</td>
<td>17</td>
</tr>
<tr>
<td>dilate_square</td>
<td>17</td>
</tr>
<tr>
<td>displacement</td>
<td>18</td>
</tr>
<tr>
<td>display</td>
<td>18</td>
</tr>
<tr>
<td>display_list</td>
<td>19</td>
</tr>
<tr>
<td>distance_transform</td>
<td>19</td>
</tr>
<tr>
<td>erode</td>
<td>20</td>
</tr>
<tr>
<td>erode_rect</td>
<td>20</td>
</tr>
<tr>
<td>erode_square</td>
<td>21</td>
</tr>
<tr>
<td>frames</td>
<td>21</td>
</tr>
<tr>
<td>G</td>
<td>22</td>
</tr>
<tr>
<td>get.locations</td>
<td>22</td>
</tr>
<tr>
<td>get.stencil</td>
<td>23</td>
</tr>
<tr>
<td>get_gradient</td>
<td>24</td>
</tr>
<tr>
<td>get_hessian</td>
<td>24</td>
</tr>
<tr>
<td>haar</td>
<td>25</td>
</tr>
<tr>
<td>imager</td>
<td>25</td>
</tr>
<tr>
<td>imappend</td>
<td>25</td>
</tr>
<tr>
<td>imdirac</td>
<td>26</td>
</tr>
<tr>
<td>imsplit</td>
<td>26</td>
</tr>
<tr>
<td>interp</td>
<td>27</td>
</tr>
<tr>
<td>isobblur</td>
<td>28</td>
</tr>
<tr>
<td>label</td>
<td>28</td>
</tr>
<tr>
<td>load.image</td>
<td>29</td>
</tr>
<tr>
<td>mclosing</td>
<td>29</td>
</tr>
<tr>
<td>mclosing_square</td>
<td>30</td>
</tr>
<tr>
<td>medianblur</td>
<td>30</td>
</tr>
<tr>
<td>mirror</td>
<td>30</td>
</tr>
<tr>
<td>mopening</td>
<td>31</td>
</tr>
<tr>
<td>mopening_square</td>
<td>31</td>
</tr>
<tr>
<td>pad</td>
<td>32</td>
</tr>
<tr>
<td>pixel.grid</td>
<td>32</td>
</tr>
<tr>
<td>pixel.index</td>
<td>33</td>
</tr>
<tr>
<td>play</td>
<td>33</td>
</tr>
<tr>
<td>plot.cimg</td>
<td>34</td>
</tr>
<tr>
<td>R</td>
<td>34</td>
</tr>
<tr>
<td>rcpp_hello_world</td>
<td>35</td>
</tr>
<tr>
<td>resize</td>
<td>35</td>
</tr>
<tr>
<td>resize_doubleXY</td>
<td>36</td>
</tr>
<tr>
<td>resize_halfXY</td>
<td>36</td>
</tr>
<tr>
<td>resize_tripleXY</td>
<td>36</td>
</tr>
</tbody>
</table>
**add.colour**

Add colour channels to a grayscale image

**Description**

Add colour channels to a grayscale image

**Usage**

```r
add.colour(im)
```

**Arguments**

- `im`

**Value**

an image of class cimg

**Author(s)**

Simon Barthelme
\texttt{as.cimg.data.frame}  \hspace{1cm} \textit{Create an image from a data.frame}

\textbf{Description}

The data frame must 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).

\textbf{Usage}

\begin{verbatim}
## S3 method for class 'data.frame'
as.cimg(df, v.name = "value", dims)
\end{verbatim}

\textbf{Arguments}

- \texttt{df} \hspace{1cm} a data.frame
- \texttt{v.name} \hspace{1cm} name of the variable to extract pixel values from (default "value")
- \texttt{dims} \hspace{1cm} a vector of length 4 corresponding to image dimensions. If missing, a guess will be made.

\textbf{Value}

an object of class cimg

\textbf{Author(s)}

Simon Barthelme
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 \(x=1:\text{width}, y=1:\text{height}\) and (optional) \(z=1:\text{depth}\), and evaluate the input function at these values.

Usage

```
## S3 method for class 'function'
as.cimg(fun, width, height, depth = 1,
        normalise.coord = FALSE)
```

Arguments

- `fun`: a function with arguments \((x,y)\) or \((x,y,z)\). Must be vectorised.
- `width`
- `height`
- `depth`
- `normalise.coord`: coordinates are normalised so that \(x,y,z\) are in \((0,1)\) (default FALSE)

Value

an object of class cimg

Author(s)

Simon Barthelmé

Examples

```
im = as.cimg(function(x,y) cos(sin(x*y/100)),100,100)
plot(im)
im = as.cimg(function(x,y) cos(sin(x*y/100)),100,100,normalise.coord=TRUE)
plot(im)
```
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**

```r
## S3 method for class 'cimg'
as.data.frame(im)
```

**Arguments**

- `im` an image of class `cimg`

**Value**

a data.frame

**Author(s)**

Simon Barthelme

---

as.im.cimg  
**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**

```r
as.im.cimg(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`
as.raster.cimg

Author(s)
Simon Barthelme

See Also
im, as.im

---

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

Description
raster objects are used by R’s base graphics for plotting

Usage

```r
## S3 method for class 'cimg'
as.raster(im, frames, rescale.color = TRUE)
```

Arguments

- `im`: a cimg object
- `frames`: which frames to extract (in case depth > 1)
- `rescale.color`: rescale so that pixel values are in [0,1]? (subtract min and divide by range). default TRUE

Value

a raster object

Author(s)
Simon Barthelme

See Also
plot.cimg, rasterImage
at

*Return pixel value at coordinates*

**Description**

Return pixel value at coordinates

**Usage**

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

**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**

pixel values

**Author(s)**

Simon Barthelme

**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)
```

---

**autocrop**

*Autocrop image region*

**Description**

Autocrop image region

**Usage**

```r
autocrop(im, color, axes = "zyx")
```
Arguments

color  Color used for the crop. If 0, color is guessed.
axes  Axes used for the crop.


B

Extract blue channel

Description

Extract blue channel

Usage

B(im)


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

Description

Blur image anisotropically, in an edge-preserving way.

Usage

blur_anisotropic(inp, amplitude = 0.7, anisotropy = 0.6,
alpha = 0.6, sigma = 1.1, dl = 0.8, da = 30, gauss_prec = 2,
interpolation_type = 0L, is_fast_approx = TRUE)

Arguments

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
is_fast_approx  Determines if a fast approximation of the gaussian function is used or not.
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,amp=1e4,sharp=1) # plot
```

---

**boxblur**

*Blur image with a box filter.*

**Description**

Blur image with a box filter.

**Usage**

```r
boxblur(inp, sigma, boundary_conditions = TRUE)
```

**Arguments**

- `sigma` Size of the box window.
- `boundary_conditions` Boundary conditions. Can be 0=dirichlet | 1=neumann.

**See Also**

`deriche()`, `vanvliet()`.

---

**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**

```r
boxblur_xy(inp, sx, sy, boundary_conditions = TRUE)
```

**Arguments**

- `boundary_conditions` Boundary conditions. Can be false=dirichlet | true=neumann.
- `sigma_x` Size of the box window, along the X-axis.
- `sigma_y` Size of the box window, along the Y-axis.
- `sigma_z` Size of the box window, along the Z-axis.

**See Also**

`blur()`.
**bucket_fill**

**Bucket fill**

**Description**

Bucket fill

**Usage**

```python
bucket_fill(im, x, y, z, color, opacity = 1, sigma = 0,
            is_high_connexity = FALSE)
```

**Arguments**

- **x**: X-coordinate of the starting point of the region to fill.
- **y**: Y-coordinate of the starting point of the region to fill.
- **z**: Z-coordinate of the starting point of the region to fill.
- **color**: Pointer to spectrum() consecutive values, defining the drawing color.
- **opacity**: Opacity of the drawing.
- **sigma**: Tolerance concerning neighborhood values.
- **is_high_connexity**: Use 8-connexity (only for 2d images).

**bucket_select**

**Select a region of homogeneous colour**

**Description**

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

**Usage**

```python
bucket_select(im, x, y, z, sigma = 0, is_high_connexity = FALSE)
```

**Arguments**

- **x**: X-coordinate of the starting point of the region to fill.
- **y**: Y-coordinate of the starting point of the region to fill.
- **z**: Z-coordinate of the starting point of the region to fill.
- **sigma**: Tolerance concerning neighborhood values.
- **is_high_connexity**: Use 8-connexity (only for 2d images).
**capture.plot**  
*Capture the current R plot device as a cimg image*

**Description**  
Capture the current R plot device as a cimg image

**Usage**  
capture.plot()

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

**Author(s)**  
Simon Barthelme

**Examples**  
```r
plot(1:10)
capture.plot() #> plot #A plot of the plot
```

---

**center.stencil**  
*Center stencil at a location*

**Description**  
Center stencil at a location

**Usage**  
center.stencil(stencil, ...)

---

**channel**  
*Extract an image channel*

**Description**  
Extract an image channel

**Usage**  
channel(im, ind)
channels

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

cimg

Create a cimg object

Description

Create a cimg object

Usage

cimg(x)

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

Arguments

x a four-dimensional numeric array
Details

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.

Value

an object of class cimg

Methods (by generic)

• as.matrix:

Author(s)

Simon Barthelme

convolve

Convolve image by a mask.

Description

The result res of the convolution of an image img by a mask mask is defined to be: res(x,y,z) =
sum_i,j,k img(x-i,y-j,z-k)*mask(i,j,k)

Usage

convolve(im, filter, boundary_conditions = TRUE, is_normalised = FALSE)

Arguments

boundary_conditions
    = the border condition type (0=zero, 1=dirichlet)
mask
    = the correlation kernel.
is_normalised
    = enable local normalization.
**correlate**

**Correlate image by a mask.**

**Description**

The correlation of the image instance this by the mask mask is defined to be: \( \text{res}(x,y,z) = \sum_{i,j,k} (*\text{this})(x + i, y + j, z + k) * \text{mask}(i, j, k) \).

**Usage**

```plaintext
correlate(im, filter, boundary_conditions = TRUE, is_normalised = FALSE)
```

**Arguments**

- `boundary_conditions` = the border condition type (0=zero, 1=dirichlet)
- `mask` = the correlation kernel.
- `is_normalised` = enable local normalization.

---

**deriche**

**Apply recursive Deriche filter.**

**Description**

Apply recursive Deriche filter.

**Usage**

```plaintext
deriche(inp, sigma, order = 0L, axis = "x", boundary_conditions = 0L)
```

**Arguments**

- `sigma` = Standard deviation of the filter.
- `order` = Order of the filter. Can be <tt> 0=smooth-filter | 1=1st-derivative | 2=2nd-derivative </tt>.
- `axis` = Axis along which the filter is computed. Can be <tt> 'x' | 'y' | 'z' | 'c' </tt>.
- `boundary_conditions` = Boundary conditions. Can be <tt> 0=dirichlet | 1=neumann </tt>. 
diffusion_tensors  
*Compute field of diffusion tensors for edge-preserving smoothing.*

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

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

**Arguments**
- **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.

dilate  
*Dilate image by a structuring element.*

**Description**
Dilate image by a structuring element.

**Usage**
```
dilate(im, mask, boundary_conditions = TRUE, is_normalised = FALSE)
```

**Arguments**
- **mask**: Structuring element.
- **boundary_conditions**: Boundary conditions.
- **is_normalized**: Sets if the erosion is locally normalized.
dilate_rect

Dilate image by a rectangular structuring element of specified size.

Description
Dilate image by a rectangular structuring element of specified size.

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

Arguments
- **sx**: Width of the structuring element.
- **sy**: Height of the structuring element.
- **sz**: Depth of the structuring element.

---

dilate_square

Dilate image by a square structuring element of specified size.

Description
Dilate image by a square structuring element of specified size.

Usage
```
dilate_square(im, size)
```

Arguments
- **size**: Size of the structuring element.
displacement

*Estimate displacement field between two images.*

**Description**

Estimate displacement field between two images.

**Usage**

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

**Arguments**

- **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)).
- **source**: Reference image.

---

display

*Display image using CImg library*

**Description**

Display image using CImg library

**Usage**

```cpp
display(im)
```

**Arguments**

- **im**: an image (cimg object)
**display_list**

*Display image list using CImg library*

**Description**

Display image list using CImg library

**Usage**

display_list(imlist)

**Arguments**

- **imlist** a list of cimg objects

---

**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**

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

_Erode image by a structuring element._

**Description**

Erode image by a structuring element.

**Usage**

```r
erode(im, mask, boundary_conditions = TRUE, is_normalized = FALSE)
```

**Arguments**

- **mask**: Structuring element.
- **boundary_conditions**: Boundary conditions.
- **is_normalized**: Sets if the erosion is locally normalized.


erode_rect

_Erode image by a rectangular structuring element of specified size._

**Description**

Erode image by a rectangular structuring element of specified size.

**Usage**

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

**Arguments**

- **sx**: Width of the structuring element.
- **sy**: Height of the structuring element.
- **sz**: Depth of the structuring element.
**erode_square**

Erode image by a square structuring element of specified size.

**Description**

Erode image by a square structuring element of specified size.

**Usage**

`erode_square(im, size)`

**Arguments**

- `size`: size of the structuring element.

---

**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
G  

Extract green channel

Description  
Extract green channel

Usage  
G(im)

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

Arguments  
  im  
  the image

  condition  
  a function that takes scalars and returns logicals

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

Author(s)  
Simon Barthelme

Examples
  im <- as.cimg(function(x,y) x+y,10,10)
  get.locations(im,function(v) v < 4)
  get.locations(im,function(v) v^2 + 3*v - 2 < 30)
**get.stencil** | Return pixel values in a neighbourhood defined by a stencil

**Description**

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

**Usage**

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

**Arguments**

- `im`
- `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_gradient  Compute image gradient.

Description
Compute image gradient.

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

Arguments
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)

get_hessian  Return image hessian.

Description
Return image hessian.

Usage
get_hessian(im, axes = "")

Arguments
axes Axes considered for the hessian computation, as a character string (e.g "xy").
**haar**

*Compute Haar multiscale wavelet transform.*

**Description**
Compute Haar multiscale wavelet transform.

**Usage**

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

**Arguments**

- `nb_scales`: Number of scales used for the transform.
- `axis`: Axis considered for the transform.
- `invert`: Set inverse of direct transform.

---

**imager**

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

**Description**
CImg by David Tschumperlé 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 some basic plotting and subsetting. You should install ImageMagick if you want support for common image formats (png, jpg, etc.).

---

**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**

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

**Arguments**

- `axis`: the axis along which to split (for example 'c')
- `im`: an image

**See Also**
imsplit (the reverse operation)
**imdirac**

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

**Description**

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

**Usage**

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

**Arguments**

- **dims**: a vector of image dimensions, or an image whose dimensions will be used.
- **x**: where to put the dirac.
- **y**
- **z**: (default 1)
- **cc**: (default 1)

**Value**

an image

**Author(s)**

Simon Barthelme

**Examples**

```plaintext
# Impulse response of the blur filter
imdirac(c(50,50,1,1),20,20) %>% isoblu(sigma=2) %>% plot
# Impulse response of the first-order Deriche filter
imdirac(c(50,50,1,1),20,20) %>% deriche(sigma=2,order=1, axis="x") %>% plot
```

---

**imsplit**

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

**Description**

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

**Usage**

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

See Also

imappend (the reverse operation)

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)

Arguments

im the image (class cimg)
locations a data.frame
cubic if TRUE, use cubic interpolation. If FALSE, use linear (default FALSE)

Examples

im <- load.image(system.file('extdata/parrots.png',package='imager'))
loc <- data.frame(x=runif(10,1,width(im)),y=runif(10,1,height(im))) #Ten random locations
interp(im,loc)
**isoblur**  
*Blur image isotropically.*

**Description**

Blur image isotropically.

**Usage**

```
isoblur(inp, sigma, boundary_conditions = TRUE, is_gaussian = FALSE)
```

**Arguments**

- `sigma`: Standard deviation of the blur.
- `boundary_conditions`: Boundary conditions. Can be `<tt> 0=dirichlet | 1=neumann</tt>`

**See Also**

deriche(), vanvliet().

---

**label**  
*Label connected components.*

**Description**

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

**Usage**

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

**Arguments**

- `is_high_connectivity`: Boolean that choose between 4(false)- or 8(true)-connectivity in 2d case, and between 6(false)- or 26(true)-connectivity in 3d case.
- `tolerance`: Tolerance used to determine if two neighboring pixels belong to the same region.
**load.image**

*Load image from file*

**Description**

You’ll need ImageMagick for some formats.

**Usage**

`load.image(file)`

**Arguments**

- `file` path to file

**Value**

an object of class `cimg`

---

**mclosing**

*Morphological closing (dilation followed by erosion)*

**Description**

Morphological closing (dilation followed by erosion)

**Usage**

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

**Arguments**

- `mask` Structuring element.
- `boundary_conditions` Boundary conditions.
- `is_normalized` Determines if the closing is locally normalized.
mclosing_square

Morphological closing by a square element (dilation followed by erosion)

Description
Morphological closing by a square element (dilation followed by erosion)

Usage
mclosing_square(im, size)

Arguments
size size of the square element

medianblur
Blur image with the median filter.

Description
Blur image with the median filter.

Usage
medianblur(inp, n, threshold)

Arguments
n Size of the median filter.
threshold Threshold used to discard pixels too far from the current pixel value in the median computation.

mirror
Mirror image content along specified axis

Description
Mirror image content along specified axis

Usage
mirror(im, axis)

Arguments
axis Mirror axis ("x","y","z","c")
### mopening

**Morphological opening (erosion followed by dilation)**

**Description**

Morphological opening (erosion followed by dilation)

**Usage**

```plaintext
mopening(im, mask, boundary_conditions = TRUE, is_normalised = FALSE)
```

**Arguments**

- `mask`: Structuring element.
- `boundary_conditions`: Boundary conditions.
- `is_normalised`: Determines if the opening is locally normalized.

### mopening_square

**Morphological opening by a square element (erosion followed by dilation)**

**Description**

Morphological opening by a square element (erosion followed by dilation)

**Usage**

```plaintext
mopening_square(im, size)
```

**Arguments**

- `size`: Size of the square element
pad  

Pad image with n pixels along specified axis

**Description**

Pad image with n pixels along specified axis

**Usage**

```r
pad(im, nPix, axis, pos = 0, val = 0)
```

**Arguments**

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

**Value**

a padded image

**Author(s)**

Simon Barthelme

---

pixel.grid  

Returns the pixel grid for an image

**Description**

The pixel grid for image im gives the (x,y,z,c) coordinates of each successive pixel as a data.frame. The c coordinate has been renamed 'cc' to avoid conflicts with R’s `c` function. NB: coordinates start at (x=1,y=1), corresponding to the top left corner of the image

**Usage**

```r
pixel.grid(im)
```

**Arguments**

- `im`

**Value**

a data.frame
**pixel.index**  
*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**

```
pixel.index(im, coords)
```

**Arguments**

- **im**: an image
- **coords**: a data.frame with values x,y,z (optional), c (optional)

**Value**

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

**Author(s)**

Simon Barthelmé

**Examples**

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

---

**play**

*Play a video*

**Description**

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

**Usage**

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

**Arguments**

- **vid**: A cimg object, to be played as video
- **loop**: loop the video (default false)
- **delay**: delay between frames, in ms. Default 30.
plot.cimg  
*Display an image using base graphics*

Description

Display an image using base graphics

Usage

```r
## S3 method for class 'cimg'
plot(im, frame, rescale.color = TRUE, ...)
```

Arguments

- `im`: the image
- `frame`: which frame to display, if the image has depth > 1
- `rescale.color`: rescale channels so that the values are in [0,1]
- `...`: other parameters to be passed to plot.default (eg "main")

See Also

`display`, which is much faster

---

R  
*Extract red channel*

Description

Extract red channel

Usage

```r
R(im)
```
**rcpp_hello_world**

**Simple function using Rcpp**

**Description**

Simple function using Rcpp

**Usage**

```r
rcpp_hello_world()
```

**Examples**

```r
## Not run:
rcpp_hello_world()

## End(Not run)
```

**resize**

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

**Description**

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

**Usage**

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

**Arguments**

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

### centering_x
- Set centering type (only if interpolation_type=0).

### centering_y
- Set centering type (only if interpolation_type=0).

### centering_z
- Set centering type (only if interpolation_type=0).

### centering_c
- Set centering type (only if interpolation_type=0).

---

### resize_doubleXY
*Resize image to double-size, using the Scale2X algorithm.*

**Description**

Use anisotropic upscaling algorithm [described here](http://scale2x.sourceforge.net/algorithm.html).

**Usage**

```
resize_doubleXY(im)
```

---

### resize_halfXY
*Resize image to half-size, using an optimized filter*

**Description**

Use anisotropic upscaling algorithm [described here](http://scale2x.sourceforge.net/algorithm.html).

**Usage**

```
resize_halfXY(im)
```

---

### resize_tripleXY
*Resize image to triple-size, using the Scale2X algorithm.*

**Description**

Use anisotropic upscaling algorithm [described here](http://scale2x.sourceforge.net/algorithm.html).

**Usage**

```
resize_tripleXY(im)
```
**rotate**

Rotate image by an arbitrary angle.

**Description**

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

**Usage**

rotate(im, angle, interpolation = 1L, boundary = 0L)

**Arguments**

- **angle** Rotation angle, in degrees.
- **interpolation** Type of interpolation. Can be `0=nearest | 1=linear | 2=cubic`.
- **boundary** Boundary conditions. Can be `0=dirichlet | 1=neumann | 2=periodic`.

**rotate_xy**

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

**Description**

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

**Usage**

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

**Arguments**

- **angle** Rotation angle, in degrees.
- **cx** X-coordinate of the rotation center.
- **cy** Y-coordinate of the rotation center.
- **zoom** Zoom factor.
- **boundary_conditions** Boundary conditions. Can be `0=dirichlet | 1=neumann | 2=periodic`.
- **interpolation_type** Type of interpolation. Can be `0=nearest | 1=linear | 2=cubic`.
save.image  

**Description**

You’ll need ImageMagick for some formats.

**Usage**

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

**Arguments**

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

**Value**

nothing

select_patches  

**Description**

Return image patches centered at cx, cy with width wx and height wy

**Usage**

```r
select_patches(im, cx, cy, wx, wy)
```

**Arguments**

- `cx, cy`: vector of coordinates for patch centers
- `wx, wy`: vector of coordinates for patch width and height
**sharpen**

*Sharpen image.*

**Description**

Sharpen image.

**Usage**

```python
sharpen(im, amplitude, sharpen_type = FALSE, edge = 1, alpha = 0, sigma = 0)
```

**Arguments**

- **amplitude**: Sharpening amplitude
- **sharpen_type**: Select sharpening method. Can be `false=inverse diffusion | true=shock filters`.
- **edge**: Edge threshold (shock filters only).
- **alpha**: Gradient smoothness (shock filters only).
- **sigma**: Tensor smoothness (shock filters only).

---

**shift**

*Shift image content.*

**Description**

Shift image content.

**Usage**

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

**Arguments**

- **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).
**squeeze**

Remove empty dimensions from an array

**Description**

Works just like Matlab’s squeeze function: if anything in \( \text{dim}(x) \) equals one the corresponding dimension is removed.

**Usage**

```r
squeeze(x)
```

**Arguments**

- `x`: an array

---

**stencil.cross**

A cross-shaped stencil

**Description**

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

**Usage**

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

**Arguments**

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

**Value**

a data.frame defining a stencil

**Author(s)**

Simon Barthelme

**See Also**

get.stencil
subim

Select part of an image

Description

subim 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

subim(im, ...)

Arguments

im

...

Value

an image with some parts cut out

Author(s)

Simon Barthenme

Examples

parrots <- load.image(system.file('extdata/parrots.png', package='imager'))
subim(parrots, x < 30) #Only the first 30 columns
subim(parrots, y < 30) #Only the first 30 rows
subim(parrots, x < 30, y < 30) #First 30 columns and rows
subim(parrots, sqrt(x) > 8) #Can use arbitrary expressions
subim(parrots, x > height/2, y > width/2) #height and width are defined based on the image
subim(parrots, cc==1) #Colour axis is "cc" not "c" here because "c" is an important R function
##Not run
##subim(parrots,x+y==1)
##can't have expressions involving interactions between variables (domain might not be square)
threshold

**Threshold grayscale image**

**Usage**

```
threshold(im, thr)
```

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>im</code></td>
<td>the image</td>
</tr>
<tr>
<td><code>thr</code></td>
<td>a threshold, either numeric, or a string with format &quot;XX</td>
</tr>
</tbody>
</table>

A thresholded image

Thresholding corresponding to setting all values below a threshold to 0, all above to 1.

```
im <- load.image(system.file('extdata/Leonardo_Birds.jpg',package='imager'))
grayscale(im) %>% threshold("15\%") %>% plot
```

Simon Barthelme

---

vanvliet

**Van Vliet recursive Gaussian filter.**

**Description**


**Usage**

```
vannvliet(inp, sigma, order = 0L, axis = "x", boundary_conditions = 0L)
```

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>sigma</code></td>
<td>standard deviation of the Gaussian filter</td>
</tr>
<tr>
<td><code>order</code></td>
<td>the order of the filter 0,1,2,3</td>
</tr>
</tbody>
</table>
| `axis`  | Axis along which the filter is computed. Can be `<tt> 'x' | 'y' | 'z' | 'c' </tt>`.
| `boundary_conditions` | Boundary conditions. Can be `<tt> 0=dirichlet | 1=neumann </tt>` (Dirichlet boundary condition has a strange behavior)

**Details**

### watershed

**Compute watershed transform.**

**Description**

Non-zero values are propagated to zero-valued ones according to the priority map.

**Usage**

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

**Arguments**

- **priority**: Priority map.
- **fill_lines**: Sets if watershed lines must be filled or not.

---

### [.cimg

**Array subset operator for cimg objects**

**Description**

Works mostly just like the regular array version of `x[...]`, the only difference being that it returns cimg objects when it makes sense to do so. For example `im[,1]` is just like `as.array(im)[,1]` except it returns a cimg object (containing only the first colour channel).

**Usage**

```r
## S3 method for class 'cimg'
x[...]
```

**Arguments**

- **x**
- **...**

**See Also**

`imsub`, which provides a more convenient interface, crop
Index

[.cimg, 43
add.colour, 3
as.cimg.array, 4
as.cimg.data.frame, 4
as.cimg.function, 5
as.data.frame.cimg, 6
as.im.cimg, 6
as.matrix.cimg (cimg), 13
as.raster.cimg, 7
at, 8
autocrop, 8
B, 9
blur_anisotropic, 9
boxblur, 10
boxblur_xy, 10
bucket_fill, 11
bucket_select, 11
capture.plot, 12
center.stencil, 12
channel, 12
channels, 13
cimg, 13
convolve, 14
correlate, 15
deriche, 15
diffusion_tensors, 16
dilate, 16
dilate_rect, 17
dilate_square, 17
displacement, 18
display, 18
display_list, 19
distance_transform, 19
erode, 20
erode_rect, 20
erode_square, 21
frames, 21
G, 22
get.locations, 22
get.stencil, 23
get.gradient, 24
get_hessian, 24
haar, 25
imager, 25
imager-package (imager), 25
imappend, 25
imdirac, 26
imsplit, 26
interp, 27
isoblu, 28
label, 28
load.image, 29
mclosing, 29
mclosing_square, 30
medianblur, 30
mirror, 30
mopening, 31
mopening_square, 31
pad, 32
pixel.grid, 32
pixel.index, 33
play, 33
plot.cimg, 34
R, 34
rcpp_hello_world, 35
resize, 35
resize_doubleXY, 36
resize_halfXY, 36
resize_tripleXY, 36
rotate, 37
rotate_xy, 37
save.image, 38
select_patches, 38
sharpen, 39
shift, 39
squeeze, 40
stencil.cross, 40
subim, 41
threshold, 42
vanvliet, 42
watershed, 43