Package ‘EBImage’

September 24, 2012

Version 3.12.0

Title Image processing toolbox for R

Author Oleg Sklyar, Gregoire Pau, Mike Smith, Wolfgang Huber

Maintainer Gregoire Pau <gpau@ebi.ac.uk>

SystemRequirements ImageMagick (>= 6.3.7), GTK+ (> 2.6)

Depends R (>= 2.8.0), methods, graphics, stats, utils, abind

Description EBImage is an R package which provides general purpose functionality for the reading, writing, processing and analysis of images. Furthermore, in the context of microscopy based cellular assays, EBImage offers tools to transform the images, segment cells and extract quantitative cellular descriptors.

License Artistic-2.0

LazyLoad true

biocViews Visualization

R topics documented:

`bwlabel` *Binary segmentation*

Description

Labels connected (connected sets) objects in a binary image.

Usage

```
bwlabel(x)
```

Arguments

`x`	An `Image` object or an array. `x` is considered as a binary image, whose pixels of value 0 are considered as background ones and other pixels as foreground ones.

Details

All pixels for each connected set of foreground (non-zero) pixels in `x` are set to an unique increasing integer, starting from 1. Hence, `max(x)` gives the number of connected objects in `x`.

Value

An `Grayscale Image` object or an array, containing the labelled version of `x`.

Author(s)

Gregoire Pau, 2009

Examples

```
## simple example
x = readImage(system.file('images', 'shapes.png', package='EBImage'))
x = x[110:512,1:130]
if (interactive()) display(x, title='Binary')
y = bwlabel(x)
if (interactive()) display(normalize(y), title='Segmented')

## read nuclei images
x = readImage(system.file('images', 'nuclei.tif', package='EBImage'))
if (interactive()) display(x)

## computes binary mask
y = thresh(x, 10, 10, 0.05)
y = opening(y, makeBrush(5, shape='disc'))
if (interactive()) display(y, title='Cell nuclei binary mask')

## bwlabel
z = bwlabel(y)
if (interactive()) display(normalize(z), title='Cell nuclei')
nbnuclei = apply(z, 3, max)
cat('Number of nuclei=', paste(nbnuclei, collapse=','),'\n')

## recolor nuclei in colors
cols = c('black', sample(rainbow(max(z))))
zrainbow = Image(cols[1+z], dim=dim(z))
if (interactive()) display(zrainbow, title='Cell nuclei (recolored)')
```

`channel` *Color and image color mode conversions*

Description

`channel` handles color space conversions between image modes. `rgbImage` combines `Grayscale` images into a `Color` one.

Usage

```
channel(x, mode)
rgbImage(red, green, blue)
```

Arguments

`x`	An `Image` object or an array.
`mode`	A character value specifying the target mode for conversion. See Details.
`red, green, blue`	`Image` objects in `Grayscale` color mode or arrays of the same dimension. If missing, a black image will be used.

Details

Conversion modes:

`rgb` Converts a `Grayscale` image or an array into a `Color` image, replicating RGB channels.

`gray, grey` Converts a `Color` image into a `Grayscale` image, using uniform 1/3 RGB weights.

`red, green, blue` Extracts the `red`, `green` or `blue` channel from a `Color` image. Returns a `Grayscale` image.

`asred, asgreen, asblue` Converts a `Grayscale` image or an array into a `Color` image of the specified hue.

`channel` changes the pixel intensities, unlike `colorMode` which just changes the way that EBImage should render an image,

Value

An `Image` object or an array.

Author(s)

Oleg Sklyar, <osklyar@ebi.ac.uk>

See Also

colorMode

Examples

```
x = readImage(system.file("images", "shapes.png", package="EBImage"))
if (interactive()) display(x)
y = channel(x, 'asgreen')
if (interactive()) display(y)

## rgbImage
x = readImage(system.file('images', 'nuclei.tif', package='EBImage'))
y = readImage(system.file('images', 'cells.tif', package='EBImage'))
if (interactive()) display(x, title='Cell nuclei')
if (interactive()) display(y, title='Cell bodies')

cells = rgbImage(green=1.5*y, blue=x)
if (interactive()) display(cells, title='Cells')
```

`Combine` *Combining images*

Description

Merges images to create image sequences.

Usage

```
combine(x, ..., along)
```

Arguments

`x`	An `Image` object, an array, or a list of `Image` objects and arrays.
`...`	`Image` objects or arrays.
`along`	an optional numeric. See details.

Details

The function `combine` uses `abind` to merge multi-dimensionnal arrays along the dimension specified by the value `along`.

If `along` is missing, a default value depending on the color mode of `x` is used. If `x` is a `Grayscale` image or an array, `along` is set to 3 and image objects are combined on this dimension. If `x` is a `Color` image, `along` is set to 4 and image objects are combined on this dimension, leaving room on the third dimension for color channels.

Value

An `Image` object or an array.

Author(s)

Gregoire Pau

See Also

`Image`

Examples

```
if (interactive()) {
  ## combination of color images
  lena = readImage(system.file("images", "lena-color.png", package="EBImage"))
  x = combine(lena, flip(lena), flop(lena))
  if (interactive()) display(x)

  ## Blurred lenas
  x = resize(lena, 128, 128)
  xt = list()
  for (t in seq(0.1, 5, len=9)) xt=c(xt, list(blur(x, s=t)))
  xt = combine(xt)
  if (interactive()) display(xt, title='Blurred Lenas')
}
```

`computeFeatures` *Compute object features*

Description

Computes morphological and texture features from image objects.

Usage

```
computeFeatures(x, ref, methods.noref=c("computeFeatures.moment", "computeFeatures.shape"),
  methods.ref=c("computeFeatures.basic", "computeFeatures.moment", "computeFeatures.haralick"),
  xname="x", refnames, properties=FALSE, expandRef=standardExpandRef, ...)
computeFeatures.basic(x, ref, properties=FALSE, basic.quantiles=c(0.01, 0.05, 0.5, 0.95, 0.99),
computeFeatures.shape(x, properties=FALSE, xs, ...)
computeFeatures.moment(x, ref, properties=FALSE, xs, ...)
computeFeatures.haralick(x, ref , properties=FALSE, haralick.nbins=32, haralick.scales=c(1, 2),
standardExpandRef(ref, refnames)
```

Arguments

`x`	An `Image` object or an array containing labelled objects. Labelled objects are pixel sets with the same unique integer value.
`ref`	A matrix or a list of matrices, containing the intensity values of the reference objects.
`methods.noref`	A character vector containing the function names to be called to compute features without reference intensities. Default is `computeFeatures.moment` and `computeFeatures.shape`.
`methods.ref`	A character vector containing the function names to be called to compute features with reference intensities. Default is `computeFeatures.basic`, `computeFeatures.moment` and `computeFeatures.haralick`.
`xname`	A character string naming the object layer. Default is `x`.
`refnames`	A character vector naming the reference intensity layers. Default are the names of `ref`, if present. If not, reference intensity layers are named using lower-case letters.
`properties`	A logical. If `FALSE`, the default, the function returns the feature matrix. If `TRUE`, the function returns feature properties.
`expandRef`	A function used to expand the reference images. Default is `standardExpandRef`. See Details.
`basic.quantiles`	A numerical vector indicating the quantiles to compute.
`haralick.nbins`	An integer indicating the number of bins using to compute the Haralick matrix. See Details.
`haralick.scales`	A integer vector indicating the number of scales to use to compute the Haralick features.
`xs`	An optional temporary object created by `computeFeatures` used for performance considerations.
`...`	Optional arguments passed to the feature computation functions.

Details

Features are named x.y.f, where x is the object layer, y the reference image layer and f the feature name. Examples include `cell.dna.mean`, indicating mean DNA intensity computed in the cell or `nucleus.tubulin.cx`, indicating the x center of mass of tubulin computed in the nucleus region.

The function `computeFeatures` computes a set of features. Features are organized in 4 classes, each computed by a different function. The function `computeFeatures.basic` computes spatial-independent statistics about pixel intensities:

- b.mean: mean intensity
- b.sd: standard deviation intensity
- b.mad: mad intensity
- b.q*: quantile intensity

The function `computeFeatures.shape` computes features that quantify object shape:

- s.area: area size (in pixels)
- s.perimeter: perimeter (in pixels)
- s.radius.mean: mean radius (in pixels)
- s.radius.max: max radius (in pixels)
- s.radius.min: min radius (in pixels)

The function `computeFeatures.moment` computes features related to object moments, with can be computed with or without reference intensities:

- m.cx: center of mass x (in pixels)
- m.cy: center of mass y (in pixels)
- m.majoraxis: elliptical fit major axis (in pixels)
- m.eccentricity: elliptical eccentricity defined by sqrt(1-majoraxis^2/minoraxis^2). Circle eccentricity is 0 and straight line eccentricity is 1.
- m.theta: object angle (in radians)

The function `computeFeatures.haralick` computes features that quantify pixel texture. Features are named according to Haralick's original paper.

Value

If `properties` if `FALSE` (by default), `computeFeatures` returns a matrix of n cells times p features, where p depends of the options given to the function. Returns `NULL` if no object is present.

If `properties` if `TRUE`, `computeFeatures` returns a matrix of p features times 2 properties (translation and rotation invariance). Feature properties are useful to filter out features that may not be needed for specific tasks, e.g. cell position when doing cell classification.

Author(s)

Gregoire Pau, <`gregoire.pau@embl.de`>, 2011

References

R. M. Haralick, K Shanmugam and Its'Hak Deinstein (1979). *Textural Features for Image Classification.* IEEE Transactions on Systems, Man and Cybernetics.

See Also

`bwlabel`, `propagate`

Examples

```
## load and segment nucleus
y = readImage(system.file("images", "nuclei.tif", package="EBImage"))[,,1]
x = thresh(y, 10, 10, 0.05)
x = opening(x, makeBrush(5, shape='disc'))
x = bwlabel(x)
if (interactive()) display(y, title="Cell nuclei")
if (interactive()) display(x, title="Segmented nuclei")

## compute shape features
fts = computeFeatures.shape(x)
fts

## compute features
ft = computeFeatures(x, y, xname="nucleus")
cat("median features are:\n")
apply(ft, 2, median)

## compute feature properties
ftp = computeFeatures(x, y, properties=TRUE, xname="nucleus")
ftp
```

`denoise` *Blurring images*

Description

Blurs an image with ImageMagick functions.

Usage

```
blur(x, r=0, s=0.5)
gblur(x, r=0, s=0.5)
```

Arguments

`x`	An `Image` object or an array.
`r`	A numeric value for the radius of the pixel neighbourhood. The default value 0 enables automatic radius selection.
`s`	The standard deviation of the Gaussian filter used for blurring. For reasonable results, `r` must be larger than `s`.

Details

`blur` uses an unspecified separable kernel. `gblur` uses a Gaussian kernel. The algorithms used by these ImageMagick functions are not well defined and hence, the usage of `filter2` is preferable to `blur` or `gblur`.

Value

An `Image` object or an array, containing the blurred version of `x`.

Author(s)

Oleg Sklyar, <osklyar@ebi.ac.uk>, 2005-2007

References

ImageMagick: http://www.imagemagick.org.

See Also

`filter2`

Examples

```
x = readImage(system.file("images", "lena.gif", package="EBImage"))
if (interactive()) display(x)

y = blur(x, r=3, s=2)
if (interactive()) display(y, title='blur(x, r=3, s=2)')

y = gblur(x, r=3, s=2)
if (interactive()) display(y, title='gblur(x, r=3, s=2)')
```

`display` *Interactive image display*

Description

Display images.

Usage

```
display(x, title=paste(deparse(substitute(x))), useGTK=TRUE)
animate(x)
```

Arguments

`x`	An `Image` object or an array.
`useGTK`	A logical of length 1. See details.
`title`	Window title.

Details

By default (and if available), the `display` function uses GTK to open a window and display the image. Multiple windows can be opened in this way.

If GTK is not available or if `useGTK` is `FALSE`, ImageMagick is used; only one window at a time can be open, and it needs to be closed by the user interactively before the next window can be opened. The ImageMagick display is not available on MS-Windows.

The `animate` function shows an animated sequence of images and uses `ImageMagick`. Similar limitations as for `display` apply (only one window, not on MS-Windows.)

Value

The functions are called for their side effect. Return value is invisible `NULL`.

Author(s)

Oleg Sklyar, `<osklyar@ebi.ac.uk>`

References

ImageMagick: `http://www.imagemagick.org` GTK: `http://www.gtk.org`, on MS-Windows `http://gladewin32.sf.net`

Examples

```
## single image
lena = readImage(system.file("images", "lena-color.png", package="EBImage"))
if (interactive()) display(lena)

## animated threshold
x = readImage(system.file("images", "lena-color.png", package="EBImage"))
x = resize(x, 128, 128)
xt = list()
for (t in seq(0.1, 5, len=9)) xt=c(xt, list(blur(x, s=t)))
xt = combine(xt)
if (interactive()) display(xt, title='Blurred Lenas')
```

`distmap` *Distance map transform*

Description

Computes the distance map transform of a binary image. The distance map is a matrix which contains for each pixel the distance to its nearest background pixel.

Usage

```
distmap(x, metric=c('euclidean', 'manhattan'))
```

Arguments

`x`	An `Image` object or an array. `x` is considered as a binary image, whose pixels of value 0 are considered as background ones and other pixels as foreground ones.
`metric`	A character indicating which metric to use, L1 distance (`manhattan`) or L2 distance (`euclidean`). Default is `euclidean`.

Details

A fast algorithm of complexity O(M*N*log(max(M,N))), where (M,N) are the dimensions of `x`, is used to compute the distance map.

Value

An `Image` object or an array, with pixels containing the distances to the nearest background points.

Author(s)

Gregoire Pau, <gpau@ebi.ac.uk>, 2008

References

M. N. Kolountzakis, K. N. Kutulakos. Fast Computation of the Euclidean Distance Map for Binary Images, Infor. Proc. Letters 43 (1992).

Examples

```
x = readImage(system.file("images", "shapes.png", package="EBImage"))
if (interactive()) display(x)
dx = distmap(x)
if (interactive()) display(dx/10, title='Distance map of x')
```

---

drawCircle *Draw a circle on an image.*

Description

Draw a circle on an image.

Usage

```
drawCircle(img, x, y, radius, col, fill=FALSE, z=1)
```

Arguments

`img`	An `Image` object or an array.
`x, y, radius`	numerics indicating the center and the radius of the circle.
`col`	A numeric or a character string specifying the color of the circle.
`fill`	A logical indicating whether the circle should be filled. Default is `FALSE`.
`z`	A numeric indicating on which frame of the image the circle should be drawn. Default is 1.

Value

An `Image` object or an array, containing the transformed version of `img`.

Author(s)

Gregoire Pau, 2010

Examples

```
## Simple white circle
x = matrix(0, nrow=300, ncol=300)
y = drawCircle(x, 100, 200, 47, col=1)
if (interactive()) display(y)

## Simple filled yellow circle
x = channel(y, 'rgb')
y = drawCircle(x, 200, 140, 57, col='yellow', fill=TRUE)
if (interactive()) display(y)
```

`drawtext` *Draw text on images.*

Description

Draws text on images.

Usage

```
drawtext(img, xy, labels, font, col)

drawfont(family=switch(.Platform$OS.type, windows="Arial", "helvetica"),
         style="n", size=14, weight=200, antialias=TRUE)
```

Arguments

`img`	An `Image` object or an array.
`xy`	Matrix (or a list of matrices if `img` contains multiple frames) of coordinates of labels.
`labels`	A character vector (or a list of vectors if `img` contains multiple frames) containing the labels to be output.
`font`	A font object, returned by `drawfont`. If missing, a default OS-dependent font will be chosen.
`col`	A character vector of font colors.
`family`	A character value indicating the font family to use. Valid examples on Linux/UNIX systems include `helvetica`, `times`, `courier` and `symbol`. Valid examples on Windows machines include TrueType like `Arial` and `Verdana`.
`style`	A character value specifying the font style to use. Supported styles are: `normal` (default), `italic`, and `oblique`.
`size`	Font size in points.
`weight`	A numeric value indicating the font weight (bold font). Supported values range between 100 and 900.
`antialias`	A logical value indicating whether the font should be anti-aliased.

Value

An `Image` object or an array, containing the transformed version of `img`.

Author(s)

Oleg Sklyar, `<osklyar@ebi.ac.uk>`, 2007

Examples

```
lena = readImage(system.file("images", "lena-color.png", package="EBImage"))
font = drawfont(weight=600, size=28)
lena = drawtext(lena, xy=c(250, 450), labels="Lena", font=font, col="white")
if (interactive()) display(lena)
```

`EBImage` *Package overview*

Description

`EBImage` is an image processing and analysis package for R. Its primary goal is to enable automated analysis of large sets of images such as those obtained in high throughput automated microscopy.

The package uses the `ImageMagick` library for image I/O operations and some image processing methods. The `GTK` library is used for displaying images using `display`.

`EBImage` relies on the `Image` object to store and process images but also works on multi-dimensional arrays.

Package content

Image methods

- `Image`
- `as.Image`, `is.Image`, `as.raster.Image`
- `colorMode`, `imageData`
- `getFrame`, `getNumberOfFrames`

Image I/O, display

- `readImage`, `writeImage`
- `display`, `animate`
- `image`

Spatial transform

- `resize`, `flip`, `flop`
- `rotate`, `translate`, `affine`

Image segmentation, objects manipulation

- `thresh`, `bwlabel`
- `watershed`, `propagate`
- `ocontour`
- `paintObjects`, `rmObjects`, `reenumerate`

Image enhancement, filtering

- `normalize`
- `filter2`, `blur`, `gblur`
- `equalize`

Morphological operations

- `makeBrush`
- `erode`, `dilate`, `opening`, `closing`
- `distmap`
- `floodFill`, `fillHull`

Colorspace manipulation

- `rgbImage`, `channel`

Image stacking, combining, tiling

- `stackObjects`
- `combine`
- `tile`, `untile`

Drawing on images

- `drawfont`, `drawtext`, `drawCircle`

Features extraction

- `computeFeatures`
- `computeFeatures.basic`, `computeFeatures.moment`, `computeFeatures.shape`, `computeFeatures.haralick`
- `standardExpandRef`

Obsolete

- `getFeatures`
- `hullFeatures`
- `edgeProfile`, `edgeFeatures`
- `moments`, `cmoments`, `smoments`, `rmoments`
- `haralickFeatures`, `haralickMatrix`
- `zernikeMoments`

Authors

Oleg Sklyar, <`osklyar@ebi.ac.uk`>, Copyright 2005-2007

Gregoire Pau, <`gpau@ebi.ac.uk`>

Wolfgang Huber, <`huber@ebi.ac.uk`>

Mike Smith, <`msmith@ebi.ac.uk`>

```
European Bioinformatics Institute
European Molecular Biology Laboratory
Wellcome Trust Genome Campus
Hinxton
Cambridge CB10 1SD
UK
```

Examples

```
example(readImage)
example(display)
example(rotate)
example(propagate)
```

`EBImage-deprecated` *EBImage deprecated functions*

Description

These following functions are deprecated and will be defunct in the next Bioconductor release.

`equalize` *Histogram equalization*

Description

Equalize the histogram of an image.

Usage

```
equalize(x)
```

Arguments

x	An Image object or an array.

Details

The algorithm used by this ImageMagick function is not well defined.

Value

An Image object or an array, containing the transformed version of x.

Author(s)

Oleg Sklyar, <osklyar@ebi.ac.uk>, 2006-2007

References

ImageMagick: http://www.imagemagick.org.

Examples

```
x = readImage(system.file("images", "lena.gif", package="EBImage"))
if (interactive()) display(x)

y = equalize(x)
if (interactive()) display(y, title='equalize(x)')
```

---

fillHull *Fill holes in objects*

Description

Fill holes in objects.

Usage

```
fillHull(x)
```

Arguments

x	An Image object or an array.

Details

`fillHull` fills holes in the objects defined in `x`, where objects are sets of pixels with the same unique integer value.

Value

An `Image` object or an array, containing the transformed version of `x`.

Author(s)

Gregoire Pau, Oleg Sklyar; 2007

See Also

`bwlabel`

Examples

```
x = readImage(system.file('images', 'nuclei.tif', package='EBImage'))
if (interactive()) display(x)

y = thresh(x, 10, 10, 0.05)
if (interactive()) display(y, title='Cell nuclei')

y = fillHull(y)
if (interactive()) display(y, title='Cell nuclei without holes')
```

`filter2` *2D Convolution Filter*

Description

Filters an image using the fast 2D FFT convolution product.

Usage

```
filter2(x, filter)
```

Arguments

`x`	An `Image` object or an array.
`filter`	An `Image` object or an array, with odd spatial dimensions. Must contain only one frame.

Details

Linear filtering is useful to perform low-pass filtering (to blur images, remove noise...) and high-pass filtering (to detect edges, sharpen images). The function `makeBrush` is useful to generate filters.

Data is reflected around borders.

If `x` contains multiple franes, the filter will be applied one each frame.

Value

An `Image` object or an array, containing the filtered version of `x`.

Author(s)

Gregoire Pau, <`gpau@ebi.ac.uk`>

See Also

`makeBrush`, `convolve`, `fft`, `blur`

Examples

```
x = readImage(system.file("images", "lena-color.png", package="EBImage"))
if (interactive()) display(x, title='Lena')

## Low-pass disc-shaped filter
f = makeBrush(21, shape='disc', step=FALSE)
if (interactive()) display(f, title='Disc filter')
f = f/sum(f)
y = filter2(x, f)
if (interactive()) display(y, title='Filtered lena')

## High-pass Laplacian filter
la = matrix(1, nc=3, nr=3)
la[2,2] = -8
y = filter2(x, la)
if (interactive()) display(y, title='Filtered lena')
```

`floodFill` *Region filling*

Description

Fill regions in images.

Usage

```
floodFill(x, pt, col, tolerance=0)
```

Arguments

`x`	An `Image` object or an array.
`pt`	Coordinates of the start filling point.
`col`	Fill color. This argument should be a numeric for Grayscale images and an R color for Color images.
`tolerance`	Color tolerance used during the fill.

Details

Flood fill is performed using the fast scan line algorithm. Filling starts at `pt` and grows in connected areas where the absolute difference of the pixels intensities (or colors) remains below `tolerance`.

Value

An `Image` object or an array, containing the transformed version of `x`.

Author(s)

Gregoire Pau, Oleg Sklyar; 2007

Examples

```
x = readImage(system.file("images", "shapes.png", package="EBImage"))
y = floodFill(x, c(67, 146), 0.5)
if (interactive()) display(y)

y = channel(y, 'rgb')
y = floodFill(y, c(48, 78), 'red')
y = floodFill(y, c(156, 52), 'orange')
if (interactive()) display(y)

x = readImage(system.file("images", "lena.gif", package="EBImage"))
y = floodFill(x, c(226, 121), 1, tolerance=0.1)
if (interactive()) display(y)
```

`getFeatures` *Extract feature extraction from image objects*

Description

Extracts numerical features from image objects.

Usage

```
getFeatures(x, ref, N=12, R=30, nc=32)
```

Arguments

`x`	An `Image` object or an array containing object masks. Object masks are sets of pixels with the same unique integer value.
`ref`	An `Image` object or an array, containing the intensity values of the objects.
`N`	Passed to `zernikeMoments`. Integer value defining the degree of the Zernike polynomials, which in turn defines the number of features calculated. Defaults to 12.
`R`	Passed to `zernikeMoments`. Defines the radius of the circle around an object centre from which the features are calculated. See details. Defaults to 30.
`nc`	Passed to `haralickFeatures`. A numeric value. Specifies the number of gray levels to bin `ref` into when computing the co-occurrence matrix. Defaults to 32.

Details

Combines and returns the features returned by `hullFeatures`, `moments`, `edgeFeatures`, `haralickFeatures` and `zernikeMoments`.

Value

`getFeatures` returns feature matrices.

Author(s)

Oleg Sklyar, <osklyar@ebi.ac.uk>, 2007

See Also

hullFeatures, moments, edgeFeatures haralickFeatures, zernikeMoments

Examples

```
x = readImage(system.file('images', 'nuclei.tif', package='EBImage'))
x = x[,,1]
if (interactive()) display(x)

## computes object mask
y = thresh(x, 10, 10, 0.05)
y = opening(y, makeBrush(5, shape='disc'))
mask = bwlabel(y)
if (interactive()) display(mask, title='Cell nuclei')

## features
ftrs = getFeatures(mask, x)[[1]]
print(ftrs[1:5,])

## paint nuclei with an eccentricity higher than 0.85
maskb = mask
id = which(ftrs[,'m.ecc']<0.85)
maskb[!is.na(match(maskb, id))] = 0

img = paintObjects(maskb, channel(x, 'rgb'), col='red')
if (interactive()) display(img, title='Nuclei with high eccentricity')
```

Image *Image class*

Description

The package EBImage uses the class Image to store and process images. Images are stored as multi-dimensional arrays containing the pixel intensities. The class Image extends the base class array and uses the colormode slot to store how the color information of the multi-dimensional data is handled.

The colormode slot could be either Grayscale or Color. In both modes, the two first dimensions of the underlying array are understood to be the spatial dimensions of the image. In the Grayscale mode, the remaining dimensions contain other images. In the the Color mode, the third dimension contains the red, green and blue channels of the image and the remaining dimensions contain other images.

All methods of the package EBImage works either with Image objects or multi-dimensional arrays but in the latter case, the color mode is assumed to be Grayscale.

Usage

```
Image(data, dim, colormode)
as.Image(x)
is.Image(x)
as.raster.Image(y)

colorMode(y)
colorMode(y) <- value

imageData(y)
imageData(y) <- value

getFrame(y, i, type='total')
getNumberOfFrames(y, type='total')
```

Arguments

`data`	A vector or array containing the pixel intensities of an image. If missing, a default 1x1 null array is used.
`dim`	A vector containing the final dimensions of an `Image` object. If missing, equals to `dim(data)`.
`colormode`	A numeric or a character string containing the color mode which could be either `Grayscale` or `Color`. If missing, equals to `Grayscale`.
`x`	An R object.
`y`	An `Image` object or an array.
`i`	A numeric.
`value`	For `colorMode`, a numeric or a character string containing the color mode which could be either `Grayscale` or `Color`. For `imageData`, an `Image` object or an array.
`type`	A character string containing `total` or `render`. Default is `total`.

Details

Depending of `type`, `getNumberOfFrames` returns the total number of frames contained in the object `y` or the number of renderable frames. The total number of frames is independent of the color mode and is equal to the product of all the dimensions except the two first ones. The number of renderable frames is equal to the total number of frames in the `Grayscale` color mode and is equal to the product of all the dimensions except the three first ones in the `Color` color mode.

`getFrame` returns the i-th frame contained in the image `y`. If `type` is `total`, the function is unaware of the color mode and returns an xy-plane. If `render`, the function returns the i-th image as shown by the `display` function.

Value

`Image` and `as.Image` return a new `Image` object.

`is.Image` returns TRUE if `x` is an `Image` object and FALSE otherwise.

`as.raster.Image` coerces an Image object to a raster object.

`colorMode` returns the color mode of `y` and `colorMode<-` changes the color mode of `y`.

`imageData` returns the array contained in an `Image` object.

Author(s)

Oleg Sklyar, `<osklyar@ebi.ac.uk>`, 2005-2007

See Also

`readImage`, `display`

Examples

```
s1 = exp(12i*pi*seq(-1, 1, length=300)^2)
y = Image(outer(Im(s1), Re(s1)))
if (interactive()) display(normalize(y))

x = Image(rnorm(300*300*3),dim=c(300,300,3), colormode='Color')
if (interactive()) display(x)

w = matrix(seq(0, 1, len=300), nc=300, nr=300)
m = abind(w, t(w), along=3)
z = Image(m, colormode='Color')
if (interactive()) display(normalize(z))

y = Image(c('red', 'violet', '#ff51a5', 'yellow'), dim=c(71, 71))
if (interactive()) display(y)

## colorMode example
x = readImage(system.file('images', 'nuclei.tif', package='EBImage'))
x = x[,,1:3]
if (interactive()) display(x, title='Cell nuclei')
colorMode(x)=Color
if (interactive()) display(x, title='Cell nuclei in RGB')
```

---

morphology *Perform morphological operations on images*

Description

Functions to perform morphological operations on binary images.

Usage

```
dilate(x, kern)
erode(x, kern)
opening(x, kern)
closing(x, kern)

makeBrush(size, shape=c('box', 'disc', 'diamond', 'gaussian'), step=TRUE, sigma=0.3)
```

Arguments

x	An Image object or an array. x is considered as a binary image, whose pixels of value 0 are considered as background ones and other pixels as foreground ones.
kern	An Image object or an array, containing the structuring element. kern is considered as a binary image, whose pixels of value 0 are considered as background ones and other pixels as foreground ones.
size	A numeric containing the size of the brush, in pixels.
shape	A character vector indicating the shape of the brush. Can be box, disc, diamond or gaussian. Default is box.
step	a logical indicating if the brush is binary. Default is TRUE. The argument is relevant only for the disc and diamond shapes.
sigma	An optional numeric containing the standard deviation of the Gaussian shape. Default is 0.3.

Details

`dilate` applies the mask positioning its centre over every background pixel (0), every pixel which is not covered by the mask is reset to foreground (1).

`erode` applies the mask positioning its centre over every foreground pixel (!=0), every pixel which is not covered by the mask is reset to background (0).

`opening` is an erosion followed by a dilation and `closing` is a dilation followed by an erosion.

`makeBrush` generates brushes of various sizes and shapes that can be used as structuring elements.

Value

`dilate`, `erode`, `opening` and `closing` return the transformed `Image` object or array, after the corresponding morphological operation.

`makeBrush` generates a 2D matrix containing the desired brush.

Author(s)

Oleg Sklyar, <`osklyar@ebi.ac.uk`>, 2006

Examples

```
x = readImage(system.file("images", "shapes.png", package="EBImage"))
if (interactive()) display(x)
kern = makeBrush(5, shape='diamond')
if (interactive()) display(kern, title='Structuring element')
if (interactive()) display(erode(x, kern), title='Erosion of x')
if (interactive()) display(dilate(x, kern), title='Dilatation of x')

## makeBrush
x = makeBrush(100, shape='diamond')
if (interactive()) display(x, title="makeBrush(100, shape='diamond')")
x = makeBrush(100, shape='disc', step=FALSE)
if (interactive()) display(x, title="makeBrush(100, shape='disc', step=FALSE)")
x = makeBrush(100, shape='gaussian', sigma=10)
if (interactive()) display(2000*x, title="makeBrush(100, shape='gaussian', sigma=10)")
```

`normalize` *Intensity values linear scaling*

Description

Linearly scale the intensity values of an image to a specified range.

Usage

```
normalize(x, separate=TRUE, ft=c(0,1))
```

Arguments

x	An Image object or an array.
separate	If TRUE, normalizes each frame separately.
ft	A numeric vector of 2 values, target minimum and maximum intensity values after normalization.

Value

An Image object or an array, containing the transformed version of x.

Author(s)

Oleg Sklyar, <osklyar@ebi.ac.uk>, 2006-2007

Examples

```
x = readImage(system.file('images', 'shapes.png', package='EBImage'))
x = x[110:512,1:130]
y = bwlabel(x)
if (interactive()) display(x, title='Original')

print(range(y))
y = normalize(y)
print(range(y))

if (interactive()) display(y, title='Segmented')
```

---

obsolete *Obsolete feature computation functions*

Description

The following functions to compute object features are obsolete. Please use computeFeatures instead.

Usage

```
moments(x, ref)
cmoments(x, ref)
rmoments(x, ref)
smoments(x, ref, pw=3, what="scale")
edgeFeatures(x, ref)
edgeProfile(x, ref, n=32, fft=TRUE, scale=TRUE, rotate=TRUE)
haralickFeatures(x, ref, nc = 32)
haralickMatrix(x, ref, nc = 32)
hullFeatures(x)
 zernikeMoments(x, ref, N = 12, R = 30)
```

Arguments

`x`	An `Image` object or an array containing object masks. Object masks are sets of pixels with the same unique integer value.
`ref`	An `Image` object or an array, containing the intensity values of the objects.
`pw`	A numeric value specifying the maximum moment order to compute. Default is 3.
`what`	A character string partially matching `central` or `scale`, specifiying what kind of moments to compute. Default is `scale`.
`n`	An integer value giving the number of angle measures. The full circle of `[-pi,pi]` is divided into `n-1` segments, at which edges the profile is approximated.
`fft`	A logical value. If `TRUE`, the resulting profile is the `fft` transformation of the distance profile giving the frequences of angular changes in shape.
`scale`	A logical value. If `TRUE`, the resulting profile is scaled by the effective radius (calcualted as part of `link{hull.features}`) making the profile scale invariant.
`rotate`	A logical value. If `TRUE`, the resulting profile is shifted by the object's roation angle (calculated from the `moments` on the `ref` image, if provided, and on the hull otherwise.
`nc`	A numeric value. Specifies the number of gray levels used to compute the co-occurrence matrix. Default value is 32.
`N`	A numeric. Indicates the maximal order of Zernike polynomials to be computed. Default value is 12.
`R`	A numeric. Defines the radius of the circle in pixels around object centers from which the features are calculated.

Value

Obsolete.

Author(s)

Gregoire Pau, `<gregoire.pau@embl.de>`, 2011

See Also

`computeFeatures`, `computeFeatures.basic`, `computeFeatures.shape`, `computeFeatures.moment`, `computeFeatures.haralick`

Examples

```
example(computeFeatures)
```

ocontour *Oriented contours*

Description

Computes the oriented contour of objects.

Usage

```
ocontour(x)
```

Arguments

x	An Image object or an array, containing objects. Only integer values are considered. Pixels of value 0 constitute the background. Each object is a set of pixels with the same unique integer value. Objets are assumed connected.

Value

A list of matrices, containing the coordinates of object oriented contours.

Author(s)

Gregoire Pau, <gpau@ebi.ac.uk>, 2008

Examples

```
x = readImage(system.file("images", "shapes.png", package="EBImage"))
x = x[1:120,50:120]
if(interactive()) display(x)
oc = ocontour(x)
plot(oc[[1]], type='l')
points(oc[[1]], col=2)
```

paintObjects *Marks objects in images*

Description

This function marks objects in images.

Usage

```
paintObjects(x, tgt, opac=c(1, 1), col=c('red', NA))
```

Arguments

`x`	An `Image` object in `Grayscale` color mode or an array containing object masks. Object masks are sets of pixels with the same unique integer value.
`tgt`	An `Image` object or an array, containing the intensity values of the objects.
`opac`	A numeric vector of two opacity values for drawing object boundaries and object bodies. Opacity ranges from 0 to 1, with 0 being fully transparent and 1 fully opaque.
`col`	A character vector of two R colors for drawing object boundaries and object bodies. By default, object boundaries are painted in `red` while object bodies are not painted.

Value

An `Image` object or an array, containing the painted version of `tgt`.

Author(s)

Oleg Sklyar, <`osklyar@ebi.ac.uk`>, 2006-2007

See Also

`bwlabel`, `watershed`, `link{getFeatures}`

Examples

```
## load images
nuc = readImage(system.file('images', 'nuclei.tif', package='EBImage'))
cel = readImage(system.file('images', 'cells.tif', package='EBImage'))
img = rgbImage(green=cel, blue=nuc)
if (interactive()) display(img, title='Cells')

## segment nuclei
nmask = thresh(nuc, 10, 10, 0.05)
nmask = opening(nmask, makeBrush(5, shape='disc'))
nmask = fillHull(nmask)
nmask = bwlabel(nmask)
if (interactive()) display(normalize(nmask), title='Cell nuclei mask')

## segment cells, using propagate and nuclei as 'seeds'
ctmask = opening(cel>0.1, makeBrush(5, shape='disc'))
cmask = propagate(cel, nmask, ctmask)
if (interactive()) display(normalize(cmask), title='Cell mask')

## using paintObjects to highlight objects
res = paintObjects(cmask, img, col='#ff00ff')
res = paintObjects(nmask, res, col='#ffff00')
if (interactive()) display(res, title='Segmented cells')
```

`propagate`	*Voronoi-based segmentation on image manifolds*

Description

Find boundaries between adjacent regions in an image, where seeds have been already identified in the individual regions to be segmented. The method finds the Voronoi region of each seed on a manifold with a metric controlled by local image properties. The method is motivated by the problem of finding the borders of cells in microscopy images, given a labelling of the nuclei in the images.

Algorithm and implementation are from Jones et al. [1].

Usage

```
propagate(x, seeds, mask=NULL, lambda=1e-4, ext, seed.centers)
```

Arguments

`x`	An `Image` object or an array, containing the image to segment.
`seeds`	An `Image` object or an array, containing the seeding objects of the already identified regions.
`mask`	An optional `Image` object or an array, containing the binary image mask of the regions that can be segmented. If missing, the whole image is segmented.
`lambda`	A numeric value. The regularisation parameter used in the metric, determining the trade-off between the Euclidian distance in the image plane and the contribution of the gradient of `x`. See details.
`ext`	Deprecated.
`seed.centers`	Deprecated.

Details

The method operates by computing a discretized approximation of the Voronoi regions for given seed points on a Riemann manifold with a metric controlled by local image features.

Under this metric, the infinitesimal distance d between points v and v+dv is defined by:

```
d^2 = ( (t(dv)*g)^2 + lambda*t(dv)*dv )/(lambda + 1)
```

, where g is the gradient of image `x` at point v.

`lambda` controls the weight of the Euclidian distance term. When `lambda` tends to infinity, d tends to the Euclidian distance. When `lambda` tends to 0, d tends to the intensity gradient of the image.

The gradient is computed on a neighborhood of 3x3 pixels.

Segmentation of the Voronoi regions in the vicinity of flat areas (having a null gradient) with small values of `lambda` can suffer from artefacts coming from the metric approximation.

Value

An `Image` object or an array, containing the labelled objects.

License

The implementation is based on CellProfiler C++ source code [2, 3]. An LGPL license was granted by Thouis Jones to use this part of CellProfiler's code for the `propagate` function.

Author(s)

The original CellProfiler code is from Anne Carpenter <carpenter@wi.mit.edu>, Thouis Jones <thouis@csail.mit.edu>, In Han Kang <inthek@mit.edu>. Responsible for this implementation: Greg Pau.

References

[1] T. Jones, A. Carpenter and P. Golland, "Voronoi-Based Segmentation of Cells on Image Manifolds", CVBIA05 (535-543), 2005

[2] A. Carpenter, T.R. Jones, M.R. Lamprecht, C. Clarke, I.H. Kang, O. Friman, D. Guertin, J.H. Chang, R.A. Lindquist, J. Moffat, P. Golland and D.M. Sabatini, "CellProfiler: image analysis software for identifying and quantifying cell phenotypes", Genome Biology 2006, 7:R100

[3] CellProfiler: http://www.cellprofiler.org

See Also

`bwlabel`, `watershed`

Examples

```
## a paraboloid mountain in a plane
n = 400
x = (n/4)^2 - matrix(
       (rep(1:n, times=n) - n/2)^2 + (rep(1:n, each=n) - n/2)^2,
       nrow=n, ncol=n)
x = normalize(x)

## 4 seeds
seeds = array(0, dim=c(n,n))
seeds[51:55, 301:305] = 1
seeds[301:305, 101:105] = 2
seeds[201:205, 141:145] = 3
seeds[331:335, 351:355] = 4

lambda = 10^seq(-8, -1, by=1)
segmented = Image(dim=c(dim(x), length(lambda)))

for(i in seq(along=lambda)) {
  prop = propagate(x, seeds, lambda=lambda[i])
  prop = prop/max(prop)
  segmented[,,i] = prop
}

if(interactive()){
  display(x, title='Image')
  display(seeds/max(seeds), title='Seeds')
  display(segmented, title="Voronoi regions")
}
```

`readImage` *Image I/O*

Description

Functions to read and write images from/to files and URL's. The supported image formats depend on the capabilities of ImageMagick.

Usage

```
readImage(files, colormode)
writeImage(x, files, quality = 100)
```

Arguments

`files`	A character vector of file names or URLs. If missing, an interactive file chooser is displayed.
`x`	An `Image` object or an array.
`quality`	A numeric, ranging from 1 to 100. Default is 100.
`colormode`	Deprecated.

Details

When writing images in formats supporting lossy compression (like JPEG), the quality can be specified used a `quality` value in the range `[1,100]`. The best quality is obtained with 100.

The file format is deduced from the file name extension.

`ImageMagick` is used to perform all image I/O operations. Therefore, the package supports all the file types supported by `ImageMagick`.

When reading images, files of different formats can be mixed in any sequence, including mixing single 2D images with TIFF image stacks. The result will contain a stack with all images and stacks, at the size of the first image read. Subsequent images are cropped (if larger) or filled with background (if smaller).

`readImage` returns an `Image` object, containing an array of double values ranging from 0 (black) to 1 (white). Image formats have a limited dynamic range (e.g. JPEG: 8 bit, TIFF: 16 bit) and `writeImage` may cause some loss of accuracy.

Value

`readImage` returns a new `Image` object. `writeImage` returns `invisible(files)`.

Author(s)

Oleg Sklyar, <`osklyar@ebi.ac.uk`>, 2005-2006

References

ImageMagick: `http://www.imagemagick.org`

Examples

```
## Reads and display images
f = system.file("images", "lena-color.png", package="EBImage")
x = readImage(f)
if (interactive()) display(x)

x = readImage(system.file("images", "nuclei.tif", package="EBImage"))
if (interactive()) display(x)

try({
  im = readImage("http://www.google.com/intl/en/images/logo.gif")
  if (interactive()) display(im)
})

## Converts a TIFF file into JPEG
f1 = system.file("images", "lena-color.png", package="EBImage")
x1 = readImage(f1)
f2 = paste(tempfile(), "jpeg", sep=".")
writeImage(x1, f2)
cat("Converted '", f1, "' into '", f2, "'.\n", sep='')
```

`resize` *Spatial linear transformations*

Description

Rotates, mirrors and resizes images.

Usage

```
flip(x)
flop(x)
resize(x, w, h, blur=1, filter="Lanczos")
rotate(x, angle=90)
affine(x, m)
```

Arguments

`x`	An `Image` object or an array.
`w, h`	Width and height of a new image. One of these arguments can be missing to enable proportional resizing.
`blur`	The blur factor, where 1 (`TRUE`) is blurry, 0 (`FALSE`) is sharp.
`filter`	Interpolating sampling filter.
`angle`	Image rotation angle in degrees.
`m`	The affine 3x2 transformation matrix.

Details

flip transforms x in its vertical mirror image by reflecting the pixels around the central x-axis.

flop transforms x in its horizontal mirror image by reflecting the pixels around the central y-axis.

resize scales the image to the desired dimensions using the supplied interpolating filter. Available filters are: Point, Box, Triangle, Hermite, Hanning, Hamming, Blackman, Gaussian, Quadratic, Cubic, Catrom, Mitchell, Lanczos, Bessel and Sinc. The filter Box performs a nearest-neighbor interpolation and is fast but introduces considerable aliasing. The filter Triangle performs a bi-linear interpolation and is a good trade-off between speed adn aliasing. Cubic interpolation with the filter Cubic is also a good trade-off. High-quality and slower interpolation is achieved with the Lanczos filter. The algorithm used by this ImageMagick function is not well defined.

rotate rotates the image counter-clockwise with the specified angle. Rotated images are usually larger than the originals and have empty triangular corners filled in black. The algorithm used by this ImageMagick function is not well defined.

affine returns the affine transformation of the image, where pixels coordinates, denoted by the matrix px, are transformed to cbind(px, 1)%*%m.

Value

An Image object or an array, containing the transformed version of x.

Author(s)

Oleg Sklyar, <osklyar@ebi.ac.uk>, 2006-2007

References

ImageMagick: http://www.imagemagick.org.

See Also

translate

Examples

```
x = readImage(system.file("images", "lena.gif", package="EBImage"))
if (interactive()) display(x)

y = flip(x)
if (interactive()) display(y, title='flip(x)')

y = flop(x)
if (interactive()) display(y, title='flop(x)')

y = resize(x, 128)
if (interactive()) display(y, title='resize(x, 128)')

y = rotate(x, 30)
if (interactive()) display(y, title='rotate(x, 30)')

m = matrix(c(0.6, 0.2, 0, -0.2, 0.3, 300), nrow=3)
if (interactive()) display(affine(x, m), title='affine transform')
```

`rmObjects` *Object removal and reindexation*

Description

The `rmObjects` functions deletes objects from an image by setting their pixel intensity values to 0. `reenumerate` re-enumerates all objects in an image from 0 (background) to the actual number of objects.

Usage

```
rmObjects(x, index)

reenumerate(x)
```

Arguments

`x`	An `Image` object in `Grayscale` color mode or an array containing object masks. Object masks are sets of pixels with the same unique integer value.
`index`	A numeric vector (or a list of vectors if `x` contains multiple frames) containing the indexes of objects to remove in the frame.

Value

An `Image` object or an array, containing the new objects.

Author(s)

Oleg Sklyar, <`osklyar@ebi.ac.uk`>, 2006-2007

See Also

`bwlabel`, `watershed`

Examples

```
## make objects
x = readImage(system.file('images', 'shapes.png', package='EBImage'))
x = x[110:512,1:130]
y = bwlabel(x)
if (interactive()) display(normalize(y), title='Objects')

## remove and reenumerate
y = rmObjects(y, 5)
if (interactive()) display(normalize(y), title='Removal')
y = reenumerate(y)
if (interactive()) display(normalize(y), title='Reenumerated')
```

stackObjects *Places detected objects into an image stack*

Description

Places detected objects into an image stack.

Usage

```
stackObjects(x, ref, index, combine=TRUE, rotate, bg.col='black', ext, centerby, rotateby)
```

Arguments

`x`	An `Image` object or an array containing object masks. Object masks are sets of pixels with the same unique integer value.
`ref`	An `Image` object or an array, containing the intensity values of the objects.
`combine`	If `x` contains multiple images, specifies if the resulting list of image stacks with individual objects should be combined using `combine` into a single image stack.
`bg.col`	Background pixel color.
`ext`	A numeric controlling the size of the output simage. If missing, `ext` is estimated from data. See details.
`index, rotate, centerby, rotateby`	Deprecated.

Details

`stackObjects` creates a set of `nbobj` images of size `(2*ext+1, 2*ext+1)`, where `nbobj` is the number of objects in `x`, and places each object of `x` in this set.

If not specified, `ext` is estimated using the 95% quantile of 2*sqrt(`g.l1`), where `g.l1` is the semi-major axis descriptor extracted from `hullFeatures`, taken over all the objects of the image `x`.

Value

An `Image` object containing the stacked objects contained in `x`. If `x` contains multiple images and if `combine` is `TRUE`, `stackObjects` returns a list of `Image` objects.

Author(s)

Oleg Sklyar, <`osklyar@ebi.ac.uk`>, 2006-2007

See Also

`combine`, `tile`, `hullFeatures`

Examples

```
## simple example
x = readImage(system.file('images', 'shapes.png', package='EBImage'))
x = x[110:512,1:130]
y = bwlabel(x)
if (interactive()) display(normalize(y), title='Objects')
z = stackObjects(y, normalize(y))
if (interactive()) display(z, title='Stacked objects')

## load images
nuc = readImage(system.file('images', 'nuclei.tif', package='EBImage'))
cel = readImage(system.file('images', 'cells.tif', package='EBImage'))
img = rgbImage(green=cel, blue=nuc)
if (interactive()) display(img, title='Cells')

## segment nuclei
nmask = thresh(nuc, 10, 10, 0.05)
nmask = opening(nmask, makeBrush(5, shape='disc'))
nmask = fillHull(bwlabel(nmask))

## segment cells, using propagate and nuclei as 'seeds'
ctmask = opening(cel>0.1, makeBrush(5, shape='disc'))
cmask = propagate(cel, nmask, ctmask)

## using paintObjects to highlight objects
res = paintObjects(cmask, img, col='#ff00ff')
res = paintObjects(nmask, res, col='#ffff00')
if (interactive()) display(res, title='Segmented cells')

## stacked cells
st = stackObjects(cmask, img)
if (interactive()) display(st, title='Stacked objects')
```

`thresh` *Adaptive thresholding*

Description

Thresholds an image using a moving rectangular window.

Usage

```
thresh(x, w=5, h=5, offset=0.01)
```

Arguments

`x`	An `Image` object or an array.
`w, h`	Width and height of the moving rectangular window.
`offset`	Thresholding offset from the averaged value.

Details

This function returns the binary image resulting from the comparison between an image and its filtered version with a rectangular window. It is equivalent of doing `{f = matrix(1, nc=2*w+1, nr=2*h+1) ; f=f/sum`
but slightly faster. The function `filter2` provides hence more flexbility than `thresh`.

Value

An `Image` object or an array, containing the transformed version of `x`.

Author(s)

Oleg Sklyar, <`osklyar@ebi.ac.uk`>, 2005-2007

See Also

`filter2`

Examples

```
x = readImage(system.file('images', 'nuclei.tif', package='EBImage'))
if (interactive()) display(x)
y = thresh(x, 10, 10, 0.05)
if (interactive()) display(y)
```

`tile` *Tiling/untiling images*

Description

Given a sequence of frames, `tile` generates a single image with frames tiled. `untile` is the inverse function and divides an image into a sequence of images.

Usage

```
tile(x, nx=10, lwd=1, fg.col="#E4AF2B", bg.col="gray")
untile(x, nim, lwd=1)
```

Arguments

`x`	An `Image` object, an array or a list of these objects.
`nx`	The number of tiled images in a row.
`lwd`	The width of the grid lines between tiled images, can be 0.
`fg.col`	The color of the grid lines.
`bg.col`	The color of the background for extra tiles.
`nim`	A numeric vector of 2 elements for the number of images in both directions.

Details

After object segmentation, `tile` is a useful addition to `stackObjects` to have an overview of the segmented objects.

Value

An `Image` object or an array, containing the tiled/untiled version of `x`.

Author(s)

Oleg Sklyar, <`osklyar@ebi.ac.uk`>, 2006-2007

See Also

`stackObjects`

Examples

```
## make a set of blurred Lenas
lena = readImage(system.file("images", "lena-color.png", package="EBImage"))
x = resize(lena, 128, 128)
xt = list()
for (t in seq(0.1, 5, len=9)) xt=c(xt, list(blur(x, s=t)))
xt = combine(xt)
if (interactive()) display(xt, title='Blurred Lenas')

## tile
xt = tile(xt, 3)
if (interactive()) display(xt, title='Tiled Lenas')

## untile
xu = untile(lena, c(3, 3))
if (interactive()) display(xu, title='Lena blocks')
```

`translate` *Image translation*

Description

Translates an image.

Usage

```
translate(x, v)
```

Arguments

`x`	An `Image` object or an array.
`v`	The translation vector or a matrix of translation vectors if `x` contains several images.

Details

Borders are repeated during translation.

Value

An `Image` object or an array, containing the translated version of `x`.

Author(s)

Gregoire Pau, <`gpau@ebi.ac.uk`>, 2008

See Also

`resize`, `rotate`

Examples

```
x = readImage(system.file("images", "lena-color.png", package="EBImage"))
y = translate(x, c(20,20))
if (interactive()) {
  display(x, title='Lena')
  display(y, title='Translated lena')
}

## gradient
y = translate(x, c(1,1))
if (interactive()) display(0.5+4*(y-x), title='NE gradient')
```

`watershed` *Watershed transformation and watershed based object detection*

Description

Watershed transformation and watershed based object detection.

Usage

```
watershed(x, tolerance=1, ext=1)
```

Arguments

`x`	An `Image` object or an array.
`tolerance`	The minimum height of the object in the units of image intensity between its highest point (seed) and the point where it contacts another object (checked for every contact pixel). If the height is smaller than the tolerance, the object will be combined with one of its neighbors, which is the highest. Tolerance should be chosen according to the range of `x`. Default value is 1, which is a reasonable value if `x` comes from `distmap`.
`ext`	Radius of the neighborhood in pixels for the detection of neighboring objects. Higher value smoothes out small objects.

Details

The algorithm identifies and separates objects that stand out of the background (zero). After the water fill, the source image is flipped upside down and the resulting valleys (values with higher intensities) are filled in first until another object or background is met. The deepest valleys (pixels with highest intensity) become indexed first, starting from 1.

The function `bwlabel` is a simpler, faster alternative to segment connected objects from binary images.

Value

An `Grayscale` `Image` object or an array, containing the labelled version of `x`.

Author(s)

Oleg Sklyar, <`osklyar@ebi.ac.uk`>, 2007

See Also

`bwlabel`, `propagate`

Examples

```
x = readImage(system.file('images', 'shapes.png', package='EBImage'))
x = x[110:512,1:130]
if (interactive()) display(x, title='Binary')
y = distmap(x)
if (interactive()) display(normalize(y), title='Distance map')
w = watershed(y)
if (interactive()) display(normalize(w), title='Watershed')
```

Index