Package ‘magick’

May 11, 2018

Type Package

Title Advanced Graphics and Image-Processing in R

Version 1.9

Description Bindings to 'ImageMagick': the most comprehensive open-source image processing library available. Supports many common formats (png, jpeg, tiff, pdf, etc) and manipulations (rotate, scale, crop, trim, flip, blur, etc). All operations are vectorized via the Magick++ STL meaning they operate either on a single frame or a series of frames for working with layers, collages, or animation. In RStudio images are automatically previewed when printed to the console, resulting in an interactive editing environment. The latest version of the package includes a native graphics device for creating in-memory graphics or drawing onto images using pixel coordinates.

License MIT + file LICENSE

URL https://github.com/ropensci/magick#readme

BugReports https://github.com/ropensci/magick/issues

SystemRequirements ImageMagick++: ImageMagick-c++-devel (rpm) or libmagick++-dev (deb)

VignetteBuilder knitr

Imports Rcpp (>= 0.12.12), magrittr, knitr, curl

LinkingTo Rcpp

Suggests spelling, jsonlite, methods, rmarkdown, rsvg, png, pdftools, ggplot2, raster, rgdal, gapminder, IRdisplay, tesseract (>= 2.0)

Encoding UTF-8

RoxygenNote 6.0.1.9000

Language en-US

NeedsCompilation yes

Author Jeroen Ooms [aut, cre] (<https://orcid.org/0000-0002-4035-0289>)

Maintainer Jeroen Ooms <jeroen@berkeley.edu>

Repository CRAN

Date/Publication 2018-05-11 20:09:05 UTC
R topics documented:

- analysis ................................................................. 2
- animation .............................................................. 3
- as_EBI mage ............................................................. 5
- attributes ............................................................. 6
- autoviewer ............................................................. 6
- coder_info ............................................................. 7
- color ................................................................. 8
- composite ............................................................ 10
- device ............................................................... 12
- editing .............................................................. 13
- effects ............................................................... 16
- geometry ............................................................ 17
- image_g gplot ......................................................... 19
- ocr ................................................................. 20
- options ............................................................. 21
- painting ............................................................. 22
- transform ........................................................... 23
- wizard ............................................................... 25
- _index_ .............................................................. 26

Index 27

---

**Description**

Functions for image calculations and analysis. This part of the package needs more work.

**Usage**

```r
image_compare(image, reference_image, metric = "", fuzz = 0)

image_compare_dist(image, reference_image, metric = "", fuzz = 0)

image_fft(image)
```

**Arguments**

- `image` magick image object returned by `image_read()` or `image_graph()`
- `reference_image` another image to compare to
- `metric` string with a metric from `metric_types()` such as "AE" or "phash"
- `fuzz` Fuzz percentage: value between 0 and 100. Relative distance between colors to be considered similar in the filling algorithm.
Details

For details see Image++ documentation. Short descriptions:

- `image_compare` calculates a metric by comparing image with a reference image.
- `image_fft` returns Discrete Fourier Transform (DFT) of the image as a magnitude / phase image pair. I wish I knew what this means.

Here `image_compare()` is vectorized over the first argument and returns the diff image with the calculated distortion value as an attribute.

See Also

Other image: _index_, animation, attributes, color, composite, device, editing, effects, geometry, ocr, options, painting, transform

Examples

```r
out1 <- image_blur(logo, 3)
out2 <- image_oilpaint(logo, 3)
input <- c(logo, out1, out2, logo)
if(magick_config()$version >= "6.8.7"){
  diff_img <- image_compare(input, logo, metric = "AE")
  attributes(diff_img)
}
```

---

**animation**

**Image Frames and Animation**

Description

Operations to manipulate or combine multiple frames of an image. Details below.

Usage

```r
image_animate(image, fps = 10, loop = 0, dispose = c("background", "previous", "none"))

image_morph(image, frames = 8)

image_mosaic(image, operator = NULL)

image_montage(image)

image_flatten(image, operator = NULL)

image_average(image)

image_append(image, stack = FALSE)
```
image_apply(image, FUN, ...)

Arguments

- **image**: magick image object returned by `image_read()` or `image_graph()`
- **fps**: frames per second
- **loop**: how many times to repeat the animation. Default is infinite.
- **dispose**: a frame disposal method from `dispose_types()`
- **frames**: number of frames to use in output animation
- **operator**: string with a composite operator from `compose_types()`
- **stack**: place images top-to-bottom (TRUE) or left-to-right (FALSE)
- **FUN**: a function to be called on each frame in the image
- **...**: additional parameters for FUN

Details

For details see Magick++ STL documentation. Short descriptions:

- `image_animate` coalesces frames by playing the sequence and converting to gif format.
- `image_morph` expands number of frames by interpolating intermediate frames to blend into each other when played as an animation.
- `image_mosaic` inlays images to form a single coherent picture.
- `image_montage` creates a composite image by combining frames.
- `image_flatten` merges frames as layers into a single frame using a given operator.
- `image_average` averages frames into single frame.
- `image_append` stack images left-to-right (default) or top-to-bottom.
- `image_apply` applies a function to each frame

The `image_apply` function calls an image function to each frame and joins results back into a single image. Because most operations are already vectorized this is often not needed. Note that FUN() should return an image. To apply other kinds of functions to image frames simply use `lapply`, `vapply`, etc.

See Also

Other image: `_index_`, `analysis`, `attributes`, `color`, `composite`, `device`, `editing`, `effects`, `geometry`, `ocr`, `options`, `painting`, `transform`
Examples

# Combine images
logo <- image_read("https://jeroen.github.io/images/Rlogo.png")
oldlogo <- image_read("https://developer.r-project.org/logo/Rlogo-3.png")

# Create morphing animation
both <- image_scale(c(oldlogo, logo), "400")
image_average(image_crop(both))
image_animate(image_morph(both, 10))

# Create thumbnails from GIF
banana <- image_read("https://jeroen.github.io/images/banana.gif")
length(banana)
image_average(banana)
image_flatten(banana)
image_append(banana)
image_append(banana, stack = TRUE)

# Append images together
wizard <- image_read("wizard:"
image_append(image_scale(c(image_append(banana[c(1,3)], stack = TRUE), wizard)))

image_composite(banana, image_scale(logo, "300"))

# Break down and combine frames
front <- image_scale(banana, "300")
background <- image_background(image_scale(logo, "400"), 'white')
frames <- image_apply(front, function(x){image_composite(background, x, offset = "+70+30")})
image_animate(frames, fps = 10)

as_EBImage

Convert to EBImage

Description

Convert a Magck image to EBImage class. Note that EBImage only supports multi-frame images in greyscale.

Usage

as_EBImage(image)

Arguments

image magick image object returned by image_read() or image_graph()
**attributes**  
*Image Attributes*

**Description**

Attributes are properties of the image that might be present on some images and might affect image manipulation methods.

**Usage**

```r
image_comment(image, comment = NULL)
image_info(image)
```

**Arguments**

- `image`: magick image object returned by `image_read()` or `image_graph()`
- `comment`: string to set an image comment

**Details**

Each attribute can be get and set with the same function. The `image_info()` function returns a data frame with some commonly used attributes.

**See Also**

Other image: `_index_`, `analysis`, `animation`, `color`, `composite`, `device`, `editing`, `effects`, `geometry`, `ocr`, `options`, `painting`, `transform`

---

**autoviewer**  
*RStudio AutoViewer*

**Description**

This enables a `addTaskCallback` that automatically updates the viewer after the state of a magick graphics device has changed. This is enabled by default in RStudio.

**Usage**

```r
autoviewer_enable()
autoviewer_disable()
```
Examples

`## Only has effect in RStudio (or other GUI with a viewer):
autoviewer_enable()

```r
img <- magick::image_graph()
plot(1)
abline(0, 1, col = "blue", lwd = 2, lty = "solid")
abline(0.1, 1, col = "red", lwd = 3, lty = "dotted")
```

`autoviewer_disable()
```
```
abline(0.2, 1, col = "green", lwd = 4, lty = "twodash")
abline(0.3, 1, col = "black", lwd = 5, lty = "dotdash")
```
```
autoviewer_enable()
```
```
abline(0.4, 1, col = "purple", lwd = 6, lty = "dashed")
abline(0.5, 1, col = "yellow", lwd = 7, lty = "longdash")
```

---

**coder_info**

**Magick Configuration**

**Description**

ImageMagick can be configured to support various additional tool and formats via external libraries. These functions show which features ImageMagick supports on your system.

**Usage**

`coder_info(format)`

`magick_config()`

**Arguments**

format image format such as png, tiff or pdf.

**Details**

Note that `coder_info` raises an error for unsupported formats.

**References**

[https://www.imagemagick.org/Magick++/CoderInfo.html](https://www.imagemagick.org/Magick++/CoderInfo.html)

**Examples**

`coder_info("png")`
`coder_info("jpg")`
`coder_info("pdf")`
`coder_info("tiff")`
`coder_info("gif")`
<table>
<thead>
<tr>
<th>color</th>
<th>Image Color</th>
</tr>
</thead>
</table>

**Description**

Functions to adjust contrast, brightness, colors of the image. Details below.

**Usage**

```plaintext
image_modulate(image, brightness = 100, saturation = 100, hue = 100)
```

```plaintext
image_quantize(image, max = 256, colorspace = "rgb", dither = NULL, treedepth = NULL)
```

```plaintext
image_map(image, map, dither = FALSE)
```

```plaintext
image_channel(image, channel = "lightness")
```

```plaintext
image_transparent(image, color, fuzz = 0)
```

```plaintext
image_background(image, color, flatten = TRUE)
```

```plaintext
image_colorize(image, opacity, color)
```

```plaintext
image_contrast(image, sharpen = 1)
```

```plaintext
image_normalize(image)
```

```plaintext
image_enhance(image)
```

```plaintext
image_equalize(image)
```

```plaintext
image_median(image, radius = 1)
```

**Arguments**

- `image`: magick image object returned by `image_read()` or `image_graph()`.
- `brightness`: modulation of brightness as percentage of the current value (100 for no change).
- `saturation`: modulation of saturation as percentage of the current value (100 for no change).
- `hue`: modulation of hue is an absolute rotation of -180 degrees to +180 degrees from the current position corresponding to an argument range of 0 to 200 (100 for no change).
- `max`: preferred number of colors in the image. The actual number of colors in the image may be less than your request, but never more.
- `colorspace`: string with a `colorspace` from `colorspace_types` for example "gray", "rgb" or "cmyk".
dither  apply Floyd/Steinberg error diffusion to the image: averages intensities of several neighboring pixels

treedepth  depth of the quantization color classification tree. Values of 0 or 1 allow selection of the optimal tree depth for the color reduction algorithm. Values between 2 and 8 may be used to manually adjust the tree depth.

map  reference image to map colors from

channel  a string with a channel from channel_types for example "alpha" or "hue" or "cyan"

color  a valid color string such as "navyblue" or "#000080"

fuzz  Fuzz percentage: value between 0 and 100. Relative distance between colors to be considered similar in the filling algorithm.

flatten  should image be flattened before writing? This also replaces transparency with background color.

opacity  percentage of opacity used for coloring

sharpen  enhance intensity differences in image

radius  replace each pixel with the median color in a circular neighborhood

Details

For details see Magick++ STL documentation. Short descriptions:

- **image_modulate** adjusts brightness, saturation and hue of image relative to current.
- **image_quantize** reduces number of unique colors in the image.
- **image_map** replaces colors of image with the closest color from a reference image.
- **image_channel** extracts a single channel from an image and returns as grayscale.
- **image_transparent** sets pixels approximately matching given color to transparent.
- **image_background** sets background color. When image is flattened, transparent pixels get background color.
- **image_colorize** overlays a solid color frame using specified opacity.
- **image_contrast** enhances intensity differences in image
- **image_normalize** increases contrast by normalizing the pixel values to span the full range of colors
- **image_enhance** tries to minimize noise
- **image_equalize** equalizes using histogram equalization
- **image_median** replaces each pixel with the median color in a circular neighborhood

Note that colors are also determined by image properties imagetype and colorspace which can be modified via image_convert().

See Also

Other image: __index__, analysis, animation, attributes, composite, device, editing, effects, geometry, ocr, options, painting, transform
Examples

```r
# manually adjust colors
logo <- image_read("logo:"
image_modulate(logo, brightness = 200)
image_modulate(logo, saturation = 150)
image_modulate(logo, hue = 200)

# Reduce image to 10 different colors using various spaces
image_quantize(logo, max = 10, colorspace = 'gray')
image_quantize(logo, max = 10, colorspace = 'rgb')
image_quantize(logo, max = 10, colorspace = 'cmyk')

# Change background color
translogo <- image_transparent(logo, 'white')
image_background(translogo, "pink", flatten = TRUE)

# Compare to flood-fill method:
image_fill(logo, "pink", fuzz = 20)

# Other color tweaks
image_colorize(logo, 50, "red")
image_contrast(logo)
image_normalize(logo)
image_enhance(logo)
image_equalize(logo)
image_median(logo)

# Alternate way to convert into black-white
image_convert(logo, type = 'grayscale')
```

---

**Image Composite**

**Description**

Similar to the ImageMagick composite utility: compose an image on top of another one using a CompositeOperator.

**Usage**

```r
image_composite(image, composite_image, operator = "atop", offset = "+0+0",
    compose_args = "")

image_border(image, color = "lightgray", geometry = "10x10",
    operator = "copy")

image_frame(image, color = "lightgray", geometry = "25x25+6+6")
```
composite

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>image</td>
<td>magick image object returned by <code>image_read()</code> or <code>image_graph()</code></td>
</tr>
<tr>
<td>composite_image</td>
<td>composition image</td>
</tr>
<tr>
<td>operator</td>
<td>string with a composite operator from <code>compose_types()</code></td>
</tr>
<tr>
<td>offset</td>
<td>a geometry_point string to set x/y offset of top image</td>
</tr>
<tr>
<td>compose_args</td>
<td>additional arguments needed for some composite operations</td>
</tr>
<tr>
<td>color</td>
<td>a valid color string such as &quot;navyblue&quot; or &quot;#000080&quot;</td>
</tr>
<tr>
<td>geometry</td>
<td>a geometry string to set height and width of the border, e.g. &quot;10x8&quot;. In addition <code>image_frame</code> allows for adding shadow by setting an offset e.g. &quot;20x10+7+2&quot;.</td>
</tr>
</tbody>
</table>

Details

The `image_compose` function is vectorized over both image arguments: if the first image has $n$ frames and the second $m$ frames, the output image will contain $n \times m$ frames.

The `image_border` function creates a slightly larger solid color frame and then composes the original frame on top. The `image_frame` function is similar but has an additional feature to create a shadow effect on the border (which is really ugly).

See Also

Other image: \_index\_, analysis, animation, attributes, color, device, editing, effects, geometry, ocr, options, painting, transform

Examples

```r
# Compose images using one of many operators
imlogo <- image_scale(image_read("logo:"), "x275")
rllogo <- image_read("https://developer.r-project.org/Logo/Rlogo-3.png")

# Standard is atop
image_composite(imlogo, rllogo)

# Same as 'blend 50' in the command line
image_composite(imlogo, rllogo, operator = "blend", compose_args="50")

# Add a border frame around the image
image_border(imlogo, "red", "10x10")
image_frame(imlogo)
```
device  

Magick Graphics Device

Description

Graphics device that produces a Magick image. Can either be used like a regular device for making plots, or alternatively via `image_draw` to open a device which draws onto an existing image using pixel coordinates. The latter is vectorized, i.e. drawing operations are applied to each frame in the image.

Usage

```r
image_graph(width = 800, height = 600, bg = "white", pointsize = 12,
            res = 72, clip = TRUE, antialias = TRUE)

image_draw(image, pointsize = 12, res = 72, antialias = TRUE, ...)

image_capture()
```

Arguments

- **width**  
in pixels
- **height**  
in pixels
- **bg**  
background color
- **pointsize**  
size of fonts
- **res**  
resolution in pixels
- **clip**  
enable clipping in the device. Because clipping can slow things down a lot, you can disable it if you don’t need it.
- **antialias**  
TRUE/FALSE: enables anti-aliasing for text and strokes
- **image**  
an existing image on which to start drawing
- **...**  
additional device parameters passed to `plot.window` such as `xlim`, `ylim`, or `mar`.

Details

The device is a relatively recent feature of the package. It should support all operations but there might still be small inaccuracies. Also it is a bit slower than some of the other devices, in particular for rendering text and clipping. Hopefully this can be optimized in the next version.

By default `image_draw` sets all margins to 0 and uses graphics coordinates to match image size in pixels (width x height) where (0, 0) is the top left corner. Note that this means the y axis increases from top to bottom which is the opposite of typical graphics coordinates. You can override all this by passing custom `xlim`, `ylim` or `mar` values to `image_draw`.

The `image_capture` function returns the current device as an image. This only works if the current device is a magick device or supports `dev.capture`.
See Also

Other image: _index_, analysis, animation, attributes, color, composite, editing, effects, geometry, ocr, options, painting, transform

Examples

```r
# Regular image
frink <- image_read("https://jeroen.github.io/images/frink.png")

# Produce image using graphics device
fig <- image_graph(res = 96)
ggplot2::qplot(mpg, wt, data = mtcars, colour = cyl)
dev.off()

# Combine
out <- image_composite(fig, frink, offset = "+70+30")
print(out)

# Or paint over an existing image
img <- image_draw(frink)
rect(20, 20, 200, 100, border = "red", lty = "dashed", lwd = 5)
abline(h = 300, col = "blue", lwd = '10', lty = "dotted")
text(10, 250, "Hoiven-Glaven", family = "monospace", cex = 4, srt = 90)
palette(rainbow(11, end = 0.9))
symbols(rep(200, 11), seq(0, 400, 40), circles = runif(11, 5, 35),
   bg = 1:11, inches = FALSE, add = TRUE)
dev.off()
print(img)

# Vectorized example with custom coordinates
earth <- image_read("https://jeroen.github.io/images/earth.gif")
img <- image_draw(earth, xlim = c(0,1), ylim = c(0,1))
rect(.1, .1, .9, .9, border = "red", lty = "dashed", lwd = 5)
text(.5, .9, "Our planet", cex = 3, col = "white")
dev.off()
print(img)
```

Description

Read, write and join or combine images. All image functions are vectorized, meaning they operate either on a single frame or a series of frames (e.g. a collage, video, or animation). Besides paths and URLs, `image_read()` supports commonly used bitmap and raster object types.
Usage

```r
image_read(path, density = NULL, depth = NULL, strip = FALSE)

image_read_svg(path, width = NULL, height = NULL)

image_read_pdf(path, pages = NULL, density = 300, password = "")

image_write(image, path = NULL, format = NULL, quality = NULL, depth = NULL, density = NULL, comment = NULL, flatten = FALSE)

image_convert(image, format = NULL, type = NULL, colorspace = NULL, depth = NULL, antialias = NULL)

image_data(image, channels = NULL, frame = 1)

image_raster(image, frame = 1, tidy = TRUE)

image_display(image, animate = TRUE)

image_browse(image, browser = getOption("browser"))

image_strip(image)

image_blank(width, height, color = "none", pseudo_image = "")

image_join(...)

image_attributes(image)
```

Arguments

- **path**: a file, url, or raster object or bitmap array
- **density**: resolution to render pdf or svg
- **depth**: color depth (either 8 or 16)
- **strip**: drop image comments and metadata
- **width**: in pixels
- **height**: in pixels
- **pages**: integer vector with page numbers. Defaults to all pages.
- **password**: user `password` to open protected pdf files
- **image**: magick image object returned by `image_read()` or `image_graph()`
- **format**: output format such as "png", "jpeg", "gif", "rgb" or "rgba".
- **quality**: number between 0 and 100 for jpeg quality. Defaults to 75.
- **comment**: text string added to the image metadata for supported formats
- **flatten**: should image be flattened before writing? This also replaces transparency with background color.
type string with imagetype value from image_types for example grayscale to convert into black/white

colorspace string with a colorspace from colorspace_types for example "gray", "rgb" or "cmyk"

antialias enable anti-aliasing for text and strokes

channels string with image channel(s) for example "rgb", "rgba", "cmyk","gray", or "ycbcr". Default is either "gray", "rgb" or "rgba" depending on the image

frame integer setting which frame to extract from the image

tidy converts raster data to long form for use with geom_raster. If FALSE output is the same as as.raster().

animate support animations in the X11 display

browser argument passed to browseURL

color a valid color string such as "navyblue" or "#000080"

pseudo_image string with pseudo image specification for example "radial-gradient:purple-yellow"

... several images or lists of images to be combined

Details

All standard base vector methods such as [[].c(), as.list(), as.raster(), rev(), length(), and print() can be used to work with magick image objects. Use the standard img[i] syntax to extract a subset of the frames from an image. The img[[i]] method is an alias for image_data() which extracts a single frame as a raw bitmap matrix with pixel values.

For reading svg or pdf it is recommended to use image_read_svg() and image_read_pdf() if the rsvg and pdftools R packages are available. These functions provide more rendering options and better quality than built-in svg/pdf rendering delegates from imagemagick itself.

X11 is required for image_display() which is only works on some platforms. A more portable method is image_browse() which opens the image in a browser. RStudio has an embedded viewer that does this automatically which is quite nice.

See Also

Other image: _index_, analysis, animation, attributes, color, composite, device, effects, geometry, ocr, options, painting, transform

Examples

# Download image from the web
frink <- image_read("https://jeroen.github.io/images/frink.png")
worldcup_frink <- image_fill(frink, "orange", "+100+200", 20)
image_write(worldcup_frink, "output.png")

# extract raw bitmap array
bitmap <- frink[[1]]

# replace pixels with #FF69B4 ('hot pink') and convert back to image
bitmap[,50:100, 50:100] <- as.raw(c(0xff, 0x69, 0xb4, 0xff))
image_read(bitmap)

# Plot to graphics device via legacy raster format
raster <- as.raster(frink)
par(ask=FALSE)
plot(raster)

# Read bitmap arrays from from other image packages
if(require(webp)) image_read(webp::read_webp("example.webp"))
if(require(svg))
tiger <- image_read_svg("http://jeroen.github.io/images/tiger.svg")
if(require(pdftools))
image_read_pdf(file.path(R.home('doc'), 'NEWS.pdf'), pages = 1, density = 100)
# create a solid canvas
image_blank(600, 400, "green")
image_blank(600, 400, pseudo_image = "radial-gradient:purple-yellow")

---

## Image Effects

**Description**

High level effects applied to an entire image. These are mostly just for fun.

**Usage**

```r
image_despeckle(image, times = 1L)
image_reducenoise(image, radius = 1L)
image_noise(image, noisetype = "gaussian")
image_blur(image, radius = 1, sigma = 0.5)
image_charcoal(image, radius = 1, sigma = 0.5)
image_edge(image, radius = 1)
image_oilpaint(image, radius = 1)
image_emboss(image, radius = 1, sigma = 0.5)
image_implode(image, factor = 0.5)
image_negate(image)
image_convolve(image, kernel = "Gaussian", iterations = 1, scaling = NULL, bias = NULL)
```
Arguments

- `image`: magick image object returned by `image_read()` or `image_graph()
- `times`: number of times to repeat the despeckle operation
- `radius`: radius, in pixels, for various transformations
- `noisetype`: string with a `noisetype` value from `noise_types`
- `sigma`: the standard deviation of the Laplacian, in pixels.
- `factor`: image implode factor (special effect)
- `kernel`: either a matrix or a string with parameterized `kerneltype` such as: "DoG:0,0,2" or "Diamond"
- `iterations`: number of iterations
- `scaling`: string with kernel scaling. The special flag "!" automatically scales to full dynamic range, for example: "50%!"
- `bias`: output bias string, for example "50%"

See Also

Other image: `_index_, analysis, animation, attributes, color, composite, device, editing, geometry, ocr, options, painting, transform

Examples

```r
logo <- image_read("logo:"
image_despeckle(log)
image_reducenoise(log)
image_noise(log)
image_blur(log, 10, 10)
image_charcoal(log)
image_edge(log)
image_oilpaint(log, radius = 3)
image_emboss(log)
image_implode(log)
image_negate(log)
if(magick_config()$version > "6.8.8")
image_convolve(log)
```

---

**Description**

ImageMagick uses a handy geometry syntax to specify coordinates and shapes for use in image transformations. You can either specify these manually as strings or use the helper functions below.
Usage

geometry_point(x, y)

geometry_area(width = NULL, height = NULL, x_off = 0, y_off = 0)

geometry_size_pixels(width = NULL, height = NULL, preserve_aspect = TRUE)

geometry_size_percent(width = 100, height = NULL)

Arguments

x
left offset in pixels

y
top offset in pixels

width
in pixels

height
in pixels

x_off
offset in pixels on x axis

y_off
offset in pixels on y axis

preserve_aspect
if FALSE, resize to width and height exactly, losing original aspect ratio. Only one of percent and preserve_aspect may be TRUE.

Details

See ImageMagick Manual for details about the syntax specification. Examples of geometry strings:

- "500x300" – Resize image keeping aspect ratio, such that width does not exceed 500 and the height does not exceed 300.
- "500x300!" – Resize image to 500 by 300, ignoring aspect ratio
- "500x" – Resize width to 500 keep aspect ratio
- "x300" – Resize height to 300 keep aspect ratio
- "50%x20%" – Resize width to 50 percent and height to 20 percent of original
- "500x300+10+20" – Crop image to 500 by 300 at position 10,20

See Also

Other image: _index_. analysis, animation, attributes, color, composite, device, editing, effects, ocr, options, painting, transform

Examples

# Specify a point
logo <- image_read("logo:")
image_annotate(logo, "Some text", location = geometry_point(100, 200), size = 24)

# Specify image area
image_crop(logo, geometry_area(300, 300), repage = FALSE)
image_ggplot

Description

Converts image to raster using `image_raster()` and then plots it with ggplot2 `geom_raster`. See examples for other ways to use magick images in ggplot2.

Usage

```r
image_ggplot(image)
```

Arguments

- `image` magick image object returned by `image_read()` or `image_graph()`

Examples

```r
# Plot with base R
plot(logo)

# Plot image with ggplot2
library(ggplot2)
myplot <- image_ggplot(logo)
myplot + ggtitle("Test plot")

# Or add to plot as annotation
image <- image_fill(logo, 'none')
raster <- as.raster(image)
myplot <- qplot(mpg, wt, data = mtcars)
myplot + annotation_raster(raster, 25, 35, 3, 5)

# Or overplot image using grid
library(grid)
qplot(speed, dist, data = cars, geom = c("point", "smooth"))
grid.raster(image)
```
Description

Extract text from an image using the tesseract package.

Usage

image_ocr(image, language = "eng", ...)

image_ocr_data(image, language = "eng", ...)

Arguments

image magick image object returned by image_read() or image_graph()

language passed to tesseract. To install additional languages see instructions in tesseract_download().

... additional parameters passed to tesseract

Details

To use this function you need to tesseract first:

install.packages("tesseract")

Best results are obtained if you set the correct language in tesseract. To install additional languages see instructions in tesseract_download().

See Also

Other image: _index_, analysis, animation, attributes, color, composite, device, editing, effects, geometry, options, painting, transform

Examples

if(require("tesseract")){
  img <- image_read("http://jeroen.github.io/images/testocr.png")
  image_ocr(img)
  image_ocr_data(img)
}
**Magick Options**

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>List option types and values supported in your version of ImageMagick. For descriptions see ImageMagick Enumerations.</td>
</tr>
</tbody>
</table>

**Usage**

```plaintext
magick_options()
option_types()
filter_types()
metric_types()
dispose_types()
compose_types()
colorspace_types()
channel_types()
image_types()
kernel_types()
noise_types()
gravity_types()
orientation_types()
```

**References**

ImageMagick Manual: Enumerations

**See Also**

Other image: _index_, analysis, animation, attributes, color, composite, device, editing, effects, geometry, ocr, painting, transform
Description

The `image_fill()` function performs flood-fill by painting starting point and all neighboring pixels of approximately the same color. Annotate simply prints some text on the image.

Usage

```plaintext
image_fill(image, color, point = "1x1", fuzz = 0)

image_annotate(image, text, gravity = "northwest", location = "+0+0",
    degrees = 0, size = 10, font = ",", color = NULL, strokecolor = NULL,
    boxcolor = NULL)
```

Arguments

- **image**: magick image object returned by `image_read()` or `image_graph()`.
- **color**: a valid color string such as "navyblue" or "#000080".
- **point**: a geometry_point string indicating the starting point of the flood-fill.
- **fuzz**: Fuzz percentage: value between 0 and 100. Relative distance between colors to be considered similar in the filling algorithm.
- **text**: annotation text.
- **gravity**: string with gravity value from `gravity_types`.
- **location**: geometry string with location relative to gravity.
- **degrees**: rotates text around center point.
- **size**: font-size in pixels.
- **font**: string with font family such as "sans", "mono", "serif", "Times", "Helvetica", "Trebuchet", "Georgia", "Palatino" or "Comic Sans".
- **strokecolor**: a color string adds a stroke (border around the text).
- **boxcolor**: a color string for background color that annotation text is rendered on.

Details

Note that more sophisticated drawing mechanisms are available via the graphics device using `image_draw`.

See Also

Other image: `_index_`, `analysis`, `animation`, `attributes.color`, `composite`, `device`, `editing`, `effects`, `geometry`, `ocr`, `options`, `transform`
Examples

```r
logo <- image_read("logo:\")
logo <- image_background(logo, 'white')
image_fill(logo, "pink", point = "+450+400")
image_fill(logo, "pink", point = "+450+400", fuzz = 25)
# Add some text to an image
image_annotate(logo, "This is a test")
image_annotate(logo, "CONFIDENTIAL", size = 50, color = "red", boxcolor = "pink",
               degrees = 30, location = "+100+100")

# Setting fonts requires fontconfig support (and that you have the font)
image_annotate(logo, "The quick brown fox", font = "monospace", size = 50)
```

<table>
<thead>
<tr>
<th>transform</th>
<th>Image Transform</th>
</tr>
</thead>
<tbody>
<tr>
<td>-------------</td>
<td>-----------------</td>
</tr>
</tbody>
</table>

Description

Basic transformations like rotate, resize, crop and flip. The geometry syntax is used to specify sizes and areas.

Usage

```r
image_trim(image, fuzz = 0)
image_chop(image, geometry)
image_rotate(image, degrees)
image_resize(image, geometry = NULL, filter = NULL)
image_scale(image, geometry = NULL)
image_sample(image, geometry = NULL)
image_crop(image, geometry = NULL, repage = TRUE)
image_flip(image)
image_flop(image)
image_deskew(image, threshold = 40)
image_page(image, pagesize = NULL, density = NULL)
image_repage(image)
image_orient(image, orientation = NULL)
```
Arguments

- **image**: magick image object returned by `image_read()` or `image_graph()`
- **fuzz**: Fuzz percentage: value between 0 and 100. Relative distance between colors to be considered similar in the filling algorithm.
- **geometry**: a `geometry` string specifying area (for cropping) or size (for resizing).
- **degrees**: value between 0 and 360 for how many degrees to rotate
- **filter**: string with `filter` type from: `filter_types`
- **repage**: resize the canvas to the cropped area
- **threshold**: straightens an image. A threshold of 40 works for most images.
- **pagesize**: geometry string with preferred size and location of an image canvas
- **density**: geometry string with vertical and horizontal resolution in pixels of the image. Specifies an image density when decoding a Postscript or PDF.
- **orientation**: string to set image orientation one of the `orientation_types`. If NULL it applies auto-orientation which tries to infer the correct orientation from the Exif data.

Details

For details see Magick++ STL documentation. Short descriptions:

- **image_trim** removes edges that are the background color from the image.
- **image_chop** removes vertical or horizontal subregion of image.
- **image_crop** cuts out a subregion of original image
- **image_rotate** rotates and increases size of canvas to fit rotated image.
- **image_deskew** auto rotate to correct skewed images
- **image_resize** resizes using custom `filterType`
- **image_scale** and **image_sample** resize using simple ratio and pixel sampling algorithm.
- **image_flip** and **image_flop** invert image vertically and horizontally

The most powerful resize function is **image_resize** which allows for setting a custom resize filter. Output of `image_scale` is similar to `image_resize(img, filter = "point")`.

For resize operations it holds that if no geometry is specified, all frames are rescaled to match the top frame.

See Also

Other image: _index_, analysis, animation, attributes, color, composite, device, editing, effects, geometry, ocr, options, painting
Examples

```haskell
logo <- image_read("logo:")
logo <- image_scale(logo, "400")
image_trim(logo)
image_chop(logo, "100x20")
image_rotate(logo, 45)
# Small image
rose <- image_convert(image_read("rose:"), "png")

# Resize to 400 width or height:
image_resize(rose, "400x")
image_resize(rose, "x400")

# Resize keeping ratio
image_resize(rose, "400x400")

# Resize, force size losing ratio
image_resize(rose, "400x400!")

# Different filters
image_resize(rose, "400x", filter = "Triangle")
image_resize(rose, "400x", filter = "Point")
# Simple pixel resize
image_scale(rose, "400x")
image_sample(rose, "400x")
image_crop(logo, "400x400+200+200")
image_flip(logo)
image_flop(logo)
if(magick_config().version > "6.8.6")
    image_orient(logo)
```

---

### Description

Example images included with ImageMagick:

### Details

- **logo**: ImageMagick Logo, 640x480
- **wizard**: ImageMagick Wizard, 480x640
- **rose**: Picture of a rose, 70x46
- **granite**: Granite texture pattern, 128x128
Description

The magick package for graphics and image processing in R. Important resources:

- R introduction vignette: getting started
- Magick++ API and Magick++ STL detailed descriptions of methods and parameters

Details

Documentation is split into the following pages:

- analysis - metrics and calculations: compare, fft
- animation - manipulate or combine multiple frames: animate, morph, mosaic, montage, average, append, apply
- attributes - image properties: comment, info
- color - contrast, brightness, colors: modulate, quantize, map, transparent, background, colorize, contrast, normalize, enhance, equalize, median
- composite - advanced joining: composite, border, frame
- device - creating graphics and drawing on images
- editing - basic image IO: read, write, convert, join, display, brose
- effects - fun effects: despecle, reducenoise, noise, blur, charcoal, edge, oilpaint, emboss, implode, negate
- geometry - specify points, areas and sizes using geometry syntax
- ocr - extract text from image using tesseract package
- options - list option types and values supported in your version of ImageMagick
- painting - flood fill and annotating text
- transform - shape operations: trim, chop, rotate, resize, scale, sample crop, flip, flop, deskew, page

See Also

Other image: analysis, animation, attributes, color, composite, device, editing, effects, geometry, ocr, options, painting, transform
Index

addTaskCallback, 6
analysis, 2, 4, 6, 9, 11, 13, 15, 17, 18, 20–22, 24, 26
animation, 3, 3, 6, 9, 11, 13, 15, 17, 18, 20–22, 24, 26
as.list(), 15
as.raster(), 15
as_EBImage, 5
attributes, 3, 4, 6, 9, 11, 13, 15, 17, 18, 20–22, 24, 26
autoviewer, 6
autoviewer_disable (autoviewer), 6
autoviewer_enable (autoviewer), 6
browseURL, 15
c(), 15
channel_types, 9
channel_types (options), 21
coder_info, 7
color, 3, 4, 6, 8, 11, 13, 15, 17, 18, 20–22, 24, 26
colorspace_types, 8, 15
colorspace_types (options), 21
compose_types (options), 21
compose_types(), 4, 11
composite, 3, 4, 6, 9, 10, 13, 15, 17, 18, 20–22, 24, 26
dev.capture, 12
device, 3, 4, 6, 9, 11, 12, 15, 17, 18, 20–22, 24, 26
dispose_types (options), 21
dispose_types(), 4 editing, 3, 4, 6, 9, 11, 13, 15, 17, 18, 20–22, 24, 26
effects, 3, 4, 6, 9, 11, 13, 15, 16, 18, 20–22, 24, 26
filter_types, 24
filter_types (options), 21
gemmetry_area (geometry), 17
gemmetry_point, 11, 22
gemmetry_point (geometry), 17
gemmetry_size_percent (geometry), 17
gemmetry_size_pixels (geometry), 17
gemmetry (wizard), 25
gemmetry_types, 22
gemmetry_types (options), 21
image_animate, 4
image_animate (animation), 3
image_annotate (painting), 22
image_append, 4
image_append (animation), 3
image_apply, 4
image_apply (animation), 3
image_attributes (editing), 13
image_average, 4
image_average (animation), 3
image_background, 9
image_background (color), 8
image_blank (editing), 13
image_blur (effects), 16
image_border, 11
image_border (composite), 10
image_browse (editing), 13
image_capture (device), 12
image_channel, 9
image_channel (color), 8
image_charcoal (effects), 16
image_chop, 24
image_chop (transform), 23
image_coalesce (animation), 3
image_colorize, 9
image_colorize (color), 8
image_comment (attributes), 6
image_compare, 3
image_compare (analysis), 2
image_compare_dist (analysis), 2
image_composite (composite), 10
image_contrast, 9
image_contrast (color), 8
image_convert (editing), 13
image_convert(), 9
image_convolve (effects), 16
image_crop, 24
image_crop (transform), 23
image_data (editing), 13
image_data(), 15
image_deskew, 24
image_deskew (transform), 23
image_despeckle (effects), 16
image_device (device), 12
image_display (editing), 13
image_draw, 22
image_draw (device), 12
image_edge (effects), 16
image_emboss (effects), 16
image_enhance, 9
image_enhance (color), 8
image_equalize, 9
image_equalize (color), 8
image.fft, 3
image.fft (analysis), 2
image_fill (painting), 22
image_fill(), 22
image_flatten, 4
image_flatten (animation), 3
image_flip, 24
image_flip (transform), 23
image_flop, 24
image_flop (transform), 23
image_frame, 11
image_frame (composite), 10
image_ggplot, 19
image_graph (device), 12
image_graph(), 2, 4–6, 8, 11, 14, 17, 19, 20, 22, 24
image_implode (effects), 16
image_info (attributes), 6
image_info(), 6
image_join (editing), 13
image_map, 9
image_map (color), 8
image_median, 9
image_median (color), 8
image_modulate, 9
image_modulate (color), 8
image_montage, 4
image_montage (animation), 3
image_morph, 4
image_morph (animation), 3
image_mosaic, 4
image_mosaic (animation), 3
image_negate (effects), 16
image_noise (effects), 16
image_normalize, 9
image_normalize (color), 8
image_ocr (ocr), 20
image_ocr_data (ocr), 20
image_oilpaint (effects), 16
image_orient (transform), 23
image_page (transform), 23
image_quantize, 9
image_quantize (color), 8
image_raster (editing), 13
image_raster(), 19
image_read (editing), 13
image_read(), 2, 4–6, 8, 11, 13, 14, 17, 19, 20, 22, 24
image_read_pdf (editing), 13
image_read_svg (editing), 13
image_reducenoise (effects), 16
image_repage (transform), 23
image_resize, 24
image_resize (transform), 23
image_rotate, 24
image_rotate (transform), 23
image_sample, 24
image_sample (transform), 23
image_scale, 24
image_scale (transform), 23
image_strip (editing), 13
image_transparent, 9
image_transparent (color), 8
INDEX

image_trim, 24
image_trim (transform), 23
image_types, 15
image_types (options), 21
image_write (editing), 13
imagemagick (_index_), 26
kernel_types (options), 21
kerneltype, 17
lapply, 4
length(), 15
logo (wizard), 25
magick (_index_), 26
magick-package (_index_), 26
magick_config (coder_info), 7
magick_options (options), 21
metric_types (options), 21
metric_types(), 2
noise_types, 17
noise_types (options), 21
ocr, 3, 4, 6, 9, 11, 13, 15, 17, 18, 20, 21, 22, 24, 26
option_types (options), 21
options, 3, 4, 6, 9, 11, 13, 15, 17, 18, 20, 21, 22, 24, 26
orientation_types, 24
orientation_types (options), 21
painting, 3, 4, 6, 9, 11, 13, 15, 17, 18, 20, 21, 22, 24, 26
password, 14
pdftools, 15
plot.window, 12
print(), 15
rev(), 15
rose (wizard), 25
rsvg, 15
tesseract, 20, 26
tesseract_download(), 20
transform, 3, 4, 6, 9, 11, 13, 15, 17, 18, 20–22, 23, 26
vapply, 4
wizard, 25