Package 'magick'

October 16, 2020

Type Package

Title Advanced Graphics and Image-Processing in R

Version 2.5.0

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://docs.ropensci.org/magick/ (website)
https://github.com/ropensci/magick (devel)

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, curl

LinkingTo Rcpp

Suggests av (>= 0.3), spelling, jsonlite, methods, knitr, rmarkdown, rsvg, webp, pdftools, ggplot2, gapminder, IRdisplay, tesseract (>= 2.0), gifski

Encoding UTF-8

RoxygenNote 7.1.1

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 2020-10-16 06:20:03 UTC

R topics documented:

`analysis` *Image Analysis*

Description

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

Usage

```
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	relative color distance (value between 0 and 100) 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, defines, device, edges, editing, effects(), fx, geometry, morphology, ocr, options(), painting, segmentation, transform(), video

Examples

```
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

```
image_animate(
  image,
  fps = 10,
  delay = NULL,
  loop = 0,
  dispose = c("background", "previous", "none"),
  optimize = FALSE
)

image_morph(image, frames = 8)

image_mosaic(image, operator = NULL)

image_flatten(image, operator = NULL)

image_average(image)

image_append(image, stack = FALSE)

image_apply(image, FUN, ...)

image_montage(
  image,
  geometry = NULL,
  tile = NULL,
  gravity = "Center",
  bg = "white",
  shadow = FALSE
)
```

Arguments

image	magick image object returned by image_read() or image_graph()
fps	frames per second. Ignored if delay is not NULL.
delay	delay after each frame, in 1/100 seconds. Must be length 1, or number of frames. If specified, then fps is ignored.
loop	how many times to repeat the animation. Default is infinite.
dispose	a frame disposal method from dispose_types()
optimize	optimize the gif animation by storing only the differences between frames. Input images must be exactly the same size.
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
geometry	a geometry string that defines the size the individual thumbnail images, and the spacing between them.
tile	a geometry string for example "4x5 with limits on how the tiled images are to be laid out on the final result.
gravity	a gravity direction, if the image is smaller than the frame, where in the frame is the image to be placed.
bg	a background color string
shadow	enable shadows between images

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, defines, device, edges, editing, effects(), fx, geometry, morphology, ocr, options(), painting, segmentation, transform(), video

Examples

```
# Combine images
logo <- image_read("https://jeroen.github.io/images/Rlogo.png")
oldlogo <- image_read("https://jeroen.github.io/images/Rlogo-old.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)
# Simple 4x3 montage
input <- rep(logo, 12)
image_montage(input, geometry = 'x100+10+10', tile = '4x3', bg = 'pink', shadow = TRUE)

# With varying frame size
input <- c(wizard, wizard, logo, logo)
image_montage(input, tile = '2x2', bg = 'pink', gravity = 'southwest')
```

`as_EBImage` *Convert to EBImage*

Description

Convert a Magick 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

```
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, defines, device, edges, editing, effects(), fx, geometry, morphology, ocr, options(), painting, segmentation, transform(), video

autoviewer *RStudio Graphics 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

```
autoviewer_enable()

autoviewer_disable()
```

Examples

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

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

Examples

```
coder_info("png")
coder_info("jpg")
coder_info("pdf")
coder_info("tiff")
coder_info("gif")
```

color *Image Color*

Description

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

Usage

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

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

image_map(image, map, dither = FALSE)

image_channel(image, channel = "lightness")

image_separate(image, channel = "default")

image_combine(image, colorspace = "sRGB", channel = "default")

image_transparent(image, color, fuzz = 0)

image_background(image, color, flatten = TRUE)

image_colorize(image, opacity, color)

image_contrast(image, sharpen = 1)

image_normalize(image)

image_enhance(image)

image_equalize(image)

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	a boolean (defaults to TRUE) specifying whether to 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". Use "none" for transparency.
fuzz	relative color distance (value between 0 and 100) 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`, `defines`, `device`, `edges`, `editing`, `effects()`, `fx`, `geometry`, `morphology`, `ocr`, `options()`, `painting`, `segmentation`, `transform()`, `video`

Examples

```
# 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')
```

`composite` *Image Composite*

Description

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

Usage

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

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

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

image_shadow_mask(image, geometry = "50x10+30+30")

image_shadow(
  image,
  color = "black",
  bg = "white",
  geometry = "50x10+30+30",
  operator = "atop",
  offset = "+20+20"
)
```

Arguments

image	magick image object returned by image_read() or image_graph()
composite_image	composition image
operator	string with a composite operator from compose_types()
offset	string with either a gravity_type or a geometry_point to set position of top image.
gravity	string with gravity value from gravity_types.
compose_args	additional arguments needed for some composite operations
color	a valid color string such as "navyblue" or "#000080". Use "none" for transparency.
geometry	a geometry string to set height and width of the border, e.g. "10x8". In addition image_frame allows for adding shadow by setting an offset e.g. "20x10+7+2".
bg	background color

Details

The image_composite 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 * 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, defines, device, edges, editing, effects(), fx, geometry, morphology, ocr, options(), painting, segmentation, transform(), video

Examples

```
# Compose images using one of many operators
imlogo <- image_scale(image_read("logo:"), "x275")
rlogo <- image_read("https://jeroen.github.io/images/Rlogo-old.png")

# Standard is atop
image_composite(imlogo, rlogo)

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

# Offset can be geometry or gravity
image_composite(logo, rose, offset = "+100+100")
image_composite(logo, rose, gravity = "East")

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

---

defines *Set encoder defines*

Description

So called 'defines' are properties that are passed along to external filters and libraries. Usually defines are used in image_read or image_write to control the image encoder/decoder, but you can also set these manually on the image object.

Usage

```
image_set_defines(image, defines)
```

Arguments

image	magick image object returned by image_read() or image_graph()
defines	a named character vector with extra options to control reading. These are the -define key{=value} settings in the command line tool. Use an empty string for value-less defines, and NA to unset a define.

Details

The defines values must be a character string, where the names contain the defines keys. Each name must be of the format "enc:key" where the first part is the encoder or filter to which the key is passed. For example `"png:....."` defines can control the encoding and decoding of png images.

The image_set_defines function does not make a copy of the image, so the defined values remain in the image object until they are overwritten or unset.

See Also

Other image: _index_, analysis, animation, attributes(), color, composite, device, edges, editing, effects(), fx, geometry, morphology, ocr, options(), painting, segmentation, transform(), video

Examples

```
# Write an image
x <- image_read("https://jeroen.github.io/images/frink.png")
image_write(x, "frink.png")

# Pass some properties to PNG encoder
defines <- c("png:compression-filter" = "1", "png:compression-level" = "0")
image_set_defines(x, defines)
image_write(x, "frink-uncompressed.png")

# Unset properties
defines[1:2] = NA
image_set_defines(x, defines)
image_write(x, "frink-final.png")

# Compare size and cleanup
file.info(c("frink.png", "frink-uncompressed.png", "frink-final.png"))
unlink(c("frink.png", "frink-uncompressed.png", "frink-final.png"))
```

---

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

```
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, defines, edges, editing, effects(), fx, geometry, morphology, ocr, options(), painting, segmentation, transform(), video

Examples

```
# 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)
```

---

edges *Edge / Line Detection*

Description

Best results are obtained by finding edges with `image_canny()` and then performing Hough-line detection on the edge image.

Usage

```
image_edge(image, radius = 1)

image_canny(image, geometry = "0x1+10%+30%")

image_hough_draw(
  image,
  geometry = NULL,
  color = "red",
  bg = "transparent",
  size = 3,
```

```
  overlay = FALSE
)

image_hough_txt(image, geometry = NULL, format = c("mvg", "svg"))
```

Arguments

image	magick image object returned by image_read() or image_graph()
radius	edge size in pixels
geometry	geometry string, see details.
color	a valid color string such as "navyblue" or "#000080". Use "none" for transparency.
bg	background color
size	size in points to draw the line
overlay	composite the drawing atop the input image. Only for bg = 'transparent'.
format	output format of the text, either svg or mvg

Details

For Hough-line detection, the geometry format is {W}x{H}+{threshold} defining the size and threshold of the filter used to find 'peaks' in the intermediate search image. For canny edge detection the format is {radius}x{sigma}+{lower%}+{upper%}. More details and examples are available at the imagemagick website.

See Also

Other image: _index_, analysis, animation, attributes(), color, composite, defines, device, editing, effects(), fx, geometry, morphology, ocr, options(), painting, segmentation, transform(), video

Examples

```
if(magick_config()$version > "6.8.9"){
shape <- demo_image("shape_rectangle.gif")
rectangle <- image_canny(shape)
rectangle %>% image_hough_draw('5x5+20')
rectangle %>% image_hough_txt(format = 'svg') %>% cat()
}
```

editing *Image Editing*

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

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

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

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

image_read_video(path, fps = 1, format = "png")

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

image_convert(
  image,
  format = NULL,
  type = NULL,
  colorspace = NULL,
  depth = NULL,
  antialias = NULL,
  matte = NULL,
  interlace = 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 = "", defines = NULL)

image_destroy(image)

image_join(...)

image_attributes(image)

image_get_artifact(image, artifact = "")

demo_image(path)
```

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
defines	a named character vector with extra options to control reading. These are the -define key{=value} settings in the command line tool. Use an empty string for value-less defines, and NA to unset a define.
width	in pixels
height	in pixels
pages	integer vector with page numbers. Defaults to all pages.
password	user password to open protected pdf files
fps	how many images to capture per second of video. Set to NULL to get all frames from the input video.
format	output format such as "png", "jpeg", "gif", "rgb" or "rgba".
image	magick image object returned by image_read() or image_graph()
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

matte	set to TRUE or FALSE to enable or disable transparency
interlace	string with interlace
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". Use "none" for transparency.
pseudo_image	string with pseudo image specification for example "radial-gradient:purple-yellow"
...	several images or lists of images to be combined
artifact	string with name of the artifact to extract, see the image_deskew for an example.

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 (including rendering of literal svg) 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.

Image objects are automatically released by the garbage collector when they are no longer reachable. Because the GC only runs once in a while, you can also call image_destroy() explicitly to release the memory immediately. This is usually only needed if you create a lot of images in a short period of time, and you might run out of memory.

See Also

Other image: _index_, analysis, animation, attributes(), color, composite, defines, device, edges, effects(), fx, geometry, morphology, ocr, options(), painting, segmentation, transform(), video

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
curl::curl_download("https://jeroen.github.io/images/example.webp", "example.webp")
if(require(webp)) image_read(webp::read_webp("example.webp"))
unlink(c("example.webp", "output.png"))
if(require(rsvg)){
tiger <- image_read_svg("http://jeroen.github.io/images/tiger.svg")
svgtxt <- '<?xml version="1.0" encoding="UTF-8"?>
<svg width="400" height="400" viewBox="0 0 400 400" fill="none">
 <circle fill="steelblue" cx="200" cy="200" r="100" />
 <circle fill="yellow" cx="200" cy="200" r="90" />
</svg>'
circles <- image_read_svg(svgtxt)
}
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_blank(200, 200, pseudo_image = "gradient:#3498db-#db3a34",
  defines = c('gradient:direction' = 'east'))
```

`effects` *Image Effects*

Description

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

Usage

```
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_oilpaint(image, radius = 1)

image_emboss(image, radius = 1, sigma = 0.5)

image_implode(image, factor = 0.5)

image_negate(image)
```

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)

See Also

Other image: _index_, analysis, animation, attributes(), color, composite, defines, device, edges, editing, fx, geometry, morphology, ocr, options(), painting, segmentation, transform(), video

Examples

```
logo <- image_read("logo:")
image_despeckle(logo)
image_reducenoise(logo)
image_noise(logo)
image_blur(logo, 10, 10)
image_charcoal(logo)
image_oilpaint(logo, radius = 3)
image_emboss(logo)
image_implode(logo)
image_negate(logo)
```

---

fx *Image FX*

Description

Apply a custom an fx expression to the image.

Usage

```
image_fx(image, expression = "p", channel = NULL)

image_fx_sequence(image, expression = "p")
```

Arguments

image	magick image object returned by image_read() or image_graph()
expression	string with an fx expression
channel	a value of channel_types() specifying which channel(s) to set

Details

There are two different interfaces. The image_fx function simply applies the same fx to each frame in the input image. The image_fx_sequence function on the other hand treats the entire input vector as a sequence, allowing you to apply an expression with multiple input images. See examples.

See Also

Other image: _index_, analysis, animation, attributes(), color, composite, defines, device, edges, editing, effects(), geometry, morphology, ocr, options(), painting, segmentation, transform(), video

Examples

```
# Show image_fx() expression
img <- image_convert(logo, colorspace = "Gray")
gradient_x <- image_convolve(img, kernel = "Prewitt")
gradient_y <- image_convolve(img, kernel = "Prewitt:90")
gradient <- c(image_fx(gradient_x, expression = "p^2"),
              image_fx(gradient_y, expression = "p^2"))
gradient <- image_flatten(gradient, operator = "Plus")
#gradient <- image_fx(gradient, expression = "sqrt(p)")
gradient


image_fx(img, expression = "pow(p, 0.5)")
image_fx(img, expression = "rand()")

# Use multiple source images
input <- c(logo, image_flop(logo))
image_fx_sequence(input, "(u+v)/2")
```

`geometry` *Geometry Helpers*

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, loosing 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, defines, device, edges, editing, effects(), fx, morphology, ocr, options(), painting, segmentation, transform(), video

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_crop(logo, geometry_area(300, 300, 100, 100), repage = FALSE)

# Specify image size
image_resize(logo, geometry_size_pixels(300))
image_resize(logo, geometry_size_pixels(height = 300))
image_resize(logo, geometry_size_pixels(300, 300, preserve_aspect = FALSE))

# resize relative to current size
image_resize(logo, geometry_size_percent(50))
image_resize(logo, geometry_size_percent(50, 20))
```

---

image_ggplot *Image to ggplot*

Description

Create a ggplot with axes set to pixel coordinates and plot the raster image on it using ggplot2::annotation_raster. See examples for how to plot an image onto an existing ggplot.

Usage

```
image_ggplot(image, interpolate = FALSE)
```

Arguments

image	magick image object returned by image_read() or image_graph()
interpolate	passed to ggplot2::annotation_raster

Examples

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

# Plot image with ggplot2
library(ggplot2)
```

```
myplot <- image_ggplot(logo)
myplot + ggtitle("Test plot")

# Show that coordinates are reversed:
myplot + theme_classic()

# 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)
```

`morphology` *Morphology*

Description

Apply a morphology method. This is a very flexible function which can be used to apply any morphology method with custom parameters. See imagemagick website for examples.

Usage

```
image_morphology(
  image,
  method = "convolve",
  kernel = "Gaussian",
  iterations = 1,
  opts = list()
)

image_convolve(
  image,
  kernel = "Gaussian",
  iterations = 1,
  scaling = NULL,
  bias = NULL
)
```

Arguments

`image`	magick image object returned by `image_read()` or `image_graph()`
`method`	a string with a valid method from `morphology_types()`

kernel	either a square matrix or a string. The string can either be a parameterized kerneltype such as: "DoG:0,0,2" or "Diamond" or it can contain a custom matrix (see examples)
iterations	number of iterations
opts	a named list or character vector with custom attributes
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, defines, device, edges, editing, effects(), fx, geometry, ocr, options(), painting, segmentation, transform(), video

Examples

```
#example from IM website:
if(magick_config()$version > "6.8.8"){
pixel <- image_blank(1, 1, 'white') %>% image_border('black', '5x5')

# See the effect of Dilate method
pixel %>% image_scale('800%')
pixel %>% image_morphology('Dilate', "Diamond") %>% image_scale('800%')

# These produce the same output:
pixel %>% image_morphology('Dilate', "Diamond", iter = 3) %>% image_scale('800%')
pixel %>% image_morphology('Dilate', "Diamond:3") %>% image_scale('800%')

# Plus example
pixel %>% image_morphology('Dilate', "Plus", iterations = 2) %>% image_scale('800%')

# Rose examples
rose %>% image_morphology('ErodeI', 'Octagon', iter = 3)
rose %>% image_morphology('DilateI', 'Octagon', iter = 3)
rose %>% image_morphology('OpenI', 'Octagon', iter = 3)
rose %>% image_morphology('CloseI', 'Octagon', iter = 3)

# Edge detection
man <- demo_image('man.gif')
man %>% image_morphology('EdgeIn', 'Octagon')
man %>% image_morphology('EdgeOut', 'Octagon')
man %>% image_morphology('Edge', 'Octagon')

# Octagonal Convex Hull
 man %>%
   image_morphology('Close', 'Diamond') %>%
   image_morphology('Thicken', 'ConvexHull', iterations = 1)

# Thinning down to a Skeleton
man %>% image_morphology('Thinning', 'Skeleton', iterations = 1)
```

```
# Specify custom kernel matrix usingn a string:
img <- demo_image("test_mag.gif")
i <- image_convolve(img, kernel = '4x5:
       0 -1  0  0
      -1 +1 -1  0
      -1 +1 -1  0
      -1 +1 +1 -1
       0 -1 -1  0 ', bias = "50%")
}
```

---

ocr *Image Text OCR*

Description

Extract text from an image using the tesseract package.

Usage

```
image_ocr(image, language = "eng", HOCR = FALSE, ...)

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().
HOCR	if TRUE return results as HOCR xml instead of plain text
...	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, defines, device, edges, editing, effects(), fx, geometry, morphology, options(), painting, segmentation, transform(), video

Examples

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

`options` *Magick Options*

Description

List option types and values supported in your version of ImageMagick. For descriptions see ImageMagick Enumerations.

Usage

```
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()

morphology_types()

style_types()
```

```
decoration_types()

distort_types()
```

References

ImageMagick Manual: Enumerations

See Also

Other image: _index_, analysis, animation, attributes(), color, composite, defines, device, edges, editing, effects(), fx, geometry, morphology, ocr, painting, segmentation, transform(), video

---

painting *Image Painting*

Description

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

Usage

```
image_fill(image, color, point = "+1+1", fuzz = 0, refcolor = NULL)

image_annotate(
  image,
  text,
  gravity = "northwest",
  location = "+0+0",
  degrees = 0,
  size = 10,
  font = "",
  style = "normal",
  weight = 400,
  kerning = 0,
  decoration = NULL,
  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". Use "none" for transparency.
point	a geometry_point string indicating the starting point of the flood-fill
fuzz	relative color distance (value between 0 and 100) to be considered similar in the filling algorithm
refcolor	if set, fuzz color distance will be measured against this color, not the color of the starting point. Any color (within fuzz color distance of the given refcolor), connected to starting point will be replaced with the color. If the pixel at the starting point does not itself match the given refcolor (according to fuzz) then no action will be taken.
text	character vector of length equal to 'image' or length 1
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".
style	value of style_types for example "italic"
weight	thickness of the font, 400 is normal and 700 is bold.
kerning	increases or decreases whitespace between letters
decoration	value of decoration_types for example "underline"
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.

Setting a font, weight, style only works if your imagemagick is compiled with fontconfig support.

See Also

Other image: _index_, analysis, animation, attributes(), color, composite, defines, device, edges, editing, effects(), fx, geometry, morphology, ocr, options(), segmentation, transform(), video

Examples

```
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)
```

`segmentation` *Image Segmentation*

Description

Basic image segmentation like connected components labelling, blob extraction and fuzzy c-means

Usage

```
image_connect(image, connectivity = 4)

image_split(image, keep_color = TRUE)

image_fuzzycmeans(image, min_pixels = 1, smoothing = 1.5)
```

Arguments

`image`	magick image object returned by `image_read()` or `image_graph()`
`connectivity`	number neighbor colors which are considered part of a unique object
`keep_color`	if TRUE the output images retain the color of the input pixel. If FALSE all matching pixels are set black to retain only the image mask.
`min_pixels`	the minimum number of pixels contained in a hexahedra before it can be considered valid (expressed as a percentage)
`smoothing`	the smoothing threshold which eliminates noise in the second derivative of the histogram (higher values gives smoother second derivative)

Details

- image_connect Connect adjacent pixels with the same pixel intensities to do blob extraction
- image_split Splits the image according to pixel intensities
- image_fuzzycmeans Fuzzy c-means segmentation of the histogram of color components

image_connect performs blob extraction by scanning the image, pixel-by-pixel from top-left to bottom-right where regions of adjacent pixels which share the same set of intensity values get combined.

See Also

Other image: _index_, analysis, animation, attributes(), color, composite, defines, device, edges, editing, effects(), fx, geometry, morphology, ocr, options(), painting, transform(), video

Examples

```
# Split an image by color
img <- image_quantize(logo, 4)
layers <- image_split(img)
layers

# This returns the original image
image_flatten(layers)

# From the IM website
objects <- image_convert(demo_image("objects.gif"), colorspace = "Gray")
objects

# Split image in blobs of connected pixel levels
if(magick_config()$version > "6.9.0"){
objects %>%
  image_connect(connectivity = 4) %>%
  image_split()

# Fuzzy c-means
image_fuzzycmeans(logo)

logo %>%
  image_convert(colorspace = "HCL") %>%
  image_fuzzycmeans(smoothing = 5)
}
```

---

thresholding *Image thresholding*

Description

Thresholding an image can be used for simple and straightforward image segmentation. The function image_threshold() allows to do black and white thresholding whereas image_lat() performs local adaptive thresholding.

Usage

```
image_threshold(
  image,
  type = c("black", "white"),
  threshold = "50%",
  channel = NULL
```

```
)

image_lat(image, geometry = "10x10+5%")
```

Arguments

image	magick image object returned by image_read() or image_graph()
type	type of thresholding, either one of lat, black or white (see details below)
threshold	pixel intensity threshold percentage for black or white thresholding
channel	a value of channel_types() specifying which channel(s) to set
geometry	pixel window plus offset for LAT algorithm

Details

- image_threshold(type = "black"): Forces all pixels below the threshold into black while leaving all pixels at or above the threshold unchanged
- image_threshold(type = "white"): Forces all pixels above the threshold into white while leaving all pixels at or below the threshold unchanged
- image_lat(): Local Adaptive Thresholding. Looks in a box (width x height) around the pixel neighborhood if the pixel value is bigger than the average minus an offset.

Examples

```
test <- image_convert(logo, colorspace = "Gray")
image_threshold(test, type = "black", threshold = "50%")
image_threshold(test, type = "white", threshold = "50%")

# Turn image into BW
test %>%
  image_threshold(type = "white", threshold = "50%") %>%
  image_threshold(type = "black", threshold = "50%")

# adaptive thresholding
image_lat(test, geometry = '10x10+5%')
```

---

transform *Image Transform*

Description

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

Usage

```
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, gravity = NULL, repage = TRUE)

image_extent(image, geometry, gravity = "center", color = "none")

image_flip(image)

image_flop(image)

image_deskew(image, threshold = 40)

image_deskew_angle(image, threshold = 40)

image_page(image, pagesize = NULL, density = NULL)

image_repage(image)

image_orient(image, orientation = NULL)

image_shear(image, geometry = "10x10", color = "none")

image_distort(image, distortion = "perspective", coordinates, bestfit = FALSE)
```

Arguments

image	magick image object returned by image_read() or image_graph()
fuzz	relative color distance (value between 0 and 100) 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
gravity	string with gravity value from gravity_types.
repage	resize the canvas to the cropped area
color	a valid color string such as "navyblue" or "#000080". Use "none" for transparency.

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.
distortion	string to set image orientation one of the distort_types.
coordinates	numeric vector (typically of length 12) with distortion coordinates
bestfit	if set to TRUE the size of the output image can be different from input

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`, `defines`, `device`, `edges`, `editing`, `effects()`, `fx`, `geometry`, `morphology`, `ocr`, `options()`, `painting`, `segmentation`, `video`

Examples

```
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_extent(rose, '200x200', color = 'pink')
image_flip(logo)
image_flop(logo)
skewed <- image_rotate(logo, 5)
deskewed <- image_deskew(skewed)
attr(deskewed, 'angle')
if(magick_config()$version > "6.8.6")
  image_orient(logo)
image_shear(logo, "10x10")
building <- demo_image('building.jpg')
image_distort(building, 'perspective', c(7,40,4,30,4,124,4,123,85,122,100,123,85,2,100,30))
```

`video` *Write Video*

Description

High quality video / gif exporter based on external packages gifski and av.

Usage

```
image_write_video(image, path = NULL, framerate = 10, ...)

image_write_gif(image, path = NULL, delay = 1/10, ...)
```

Arguments

`image`	magick image object returned by `image_read()` or `image_graph()`
`path`	filename of the output gif or video. This is also the return value.
`framerate`	frames per second, passed to av_encode_video
`...`	additional parameters passed to av_encode_video and gifski.
`delay`	duration of each frame in seconds (inverse of framerate)

Details

This requires an image with multiple frames. The GIF exporter accomplishes the same thing as image_animate but much faster and with better quality.

See Also

Other image: _index_, analysis, animation, attributes(), color, composite, defines, device, edges, editing, effects(), fx, geometry, morphology, ocr, options(), painting, segmentation, transform()

---

wizard *Example Images*

Description

Example images included with ImageMagick:

Usage

```
logo
```

Format

An object of class magick-image of length 1.

Details

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

---

index *Magick Image Processing*

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`, `defines`, `device`, `edges`, `editing`, `effects()`, `fx`, `geometry`, `morphology`, `ocr`, `options()`, `painting`, `segmentation`, `transform()`, `video`

Index