Package 'rayimage'

June 12, 2026

Type Package

Title Image Processing for Simulated Cameras

Version 0.26.1

Maintainer Tyler Morgan-Wall <tylermw@gmail.com>

Description Uses convolution-based techniques to generate simulated camera bokeh, depth of field, and other camera effects, using an image and an optional depth map. Accepts both filename inputs and in-memory array representations of images and matrices, including common raster formats such as 'JPEG', 'PNG', 'TIFF', 'TGA', 'BMP', 'PSD', 'GIF', 'HDR', 'PIC', 'PNM', 'DNG', and 'EXR'. Includes functions to perform 2D convolutions, color correction, colorspace conversion, image/matrix reorientation and resizing, image and text overlays, exposure adjustment, camera vignette effects, and image titles.

License GPL-3

LazyData true

Depends R (>= 4.3.0)

Imports Rcpp, png, jpeg, grDevices, grid, tiff, systemfonts, ragg

Suggests magick, testthat (>= 3.0.0), libopenexr (>= 3.4.12-4), cli

LinkingTo Rcpp, RcppArmadillo, progress, tinydng, stbimageheaders

Encoding UTF-8

URL https://www.rayimage.dev, https://github.com/tylermorganwall/rayimage

BugReports https://github.com/tylermorganwall/rayimage/issues

Config/testthat/edition 3

Config/build/compilation-database true

Config/roxygen2/version 8.0.0

NeedsCompilation yes

Author Tyler Morgan-Wall [aut, cph, cre] (ORCID: <https://orcid.org/0000-0002-3131-3814>)

Repository CRAN

Date/Publication 2026-06-12 16:30:02 UTC

Contents

colorspace_descriptors

Prebuilt RGB Working/Display Color Spaces

Description

These objects are **color space descriptors** used by `rayimage` to tag images and to convert between RGB spaces. Each descriptor is a named list with:

- `name`: Character. Human-readable name of the space.
- `primaries`: List with `r`, `g`, `b` entries giving `xy` chromaticities.
- `rgb_to_xyz`: 3x3 numeric matrix (linear RGB -> CIE XYZ, Y=1 white).
- `xyz_to_rgb`: 3x3 numeric matrix (CIE XYZ -> linear RGB).
- `white_xyz`: Length-3 numeric XYZ of the reference white (normalized s.t. Y = 1).
- `white_name`: Character label for the white (e.g., `"D60"`, `"D65"`).

These are used as:

- `attr(img, "colorspace")`: the current working/display space for an image.
- `attr(img, "white_current")`: the current assumed scene/display white (XYZ, Y=1).

Usage

```
CS_ACESCG

CS_SRGB

CS_P3D65

CS_BT2020

CS_ADOBE
```

Details

Prebuilt RGB Working/Display Color Spaces

- **CS_ACESCG**: ACEScg (AP1 primaries, D60, scene-linear). Recommended working space.
- **CS_SRGB**: sRGB/Rec.709 (D65). Typical display space; used for PNG/JPEG/TIFF output.
- **CS_P3D65**: Display P3 (D65). Wide-gamut display space.
- **CS_BT2020**: BT.2020 (D65). Very wide-gamut; common for HDR mastering.
- **CS_ADOBE**: Adobe RGB (D65). Photo workflows.

See Also

- render_convert_colorspace() to convert images between spaces.
- render_white_balance() for Bradford CAT within a working space.
- plot_image(), ray_write_image() for display/output conversion.

Examples

```
# Tag a raw array as sRGB, then convert to ACEScg:
arr = array(runif(64*64*4), dim = c(64, 64, 4))
ri  = ray_read_image(arr, assume_colorspace = CS_SRGB, assume_white = CS_SRGB$white_xyz)
riA = render_convert_colorspace(ri, to_mats = CS_ACESCG)
plot_image(riA) # display-converted to sRGB automatically

# Convert an sRGB JPEG to ACEScg on ingest:
# img = ray_read_image("photo.jpg", normalize = FALSE)
# img_aces = render_convert_colorspace(img, to_mats = CS_ACESCG)
```

---

dragon *Dragon Image*

Description

Dragon Image

Usage

```
dragon
```

Format

An RGB 3-layer HDR rayimg array with 200 rows and 200 columns, generated using the rayrender package, with gamma correction.

---

dragondepth *Dragon Depthmap*

Description

Dragon Depthmap

Usage

```
dragondepth
```

Format

An matrix with 200 rows and 200 columns, representing the depth into the `dragon` image scene. Generated using the rayrender package. Distances range from 847 to 1411.

`generate_2d_disk` *Generate 2D Disk*

Description

Generates a 2D disk with a gradual falloff.

Disk generated using the following formula:

(-22.35 cos(1.68 r^2) + 85.91 sin(1.68 r^2)) exp(-4.89 r^2) + (35.91 cos(4.99 r^2) - 28.87 sin(4.99 r^2)) exp(-4.71 r^2) + (-13.21 cos(8.24 r^2) - 1.57 sin(8.24 r^2)) exp(-4.05 r^2) + (0.50 cos(11.90 r^2) + 1.81 sin(11.90 r^2)) exp(-2.92 r^2) + (0.13 cos(16.11 r^2) - 0.01 sin(16.11 r^2)) exp(-1.51 r^2)

The origin of the coordinate system is the center of the matrix.

Usage

```
generate_2d_disk(dim = c(11, 11), radius = 1, rescale_unity = FALSE)
```

Arguments

`dim`	Default `c(11, 11)`. The dimensions of the matrix.
`radius`	Default `1`. Radius of the disk, compared to the dimensions. Should be less than one.
`rescale_unity`	Default `FALSE`. If `TRUE`, this will rescale the max value to one. Useful if wanting to plot the distribution with `plot_image()`.

Examples

```
image(generate_2d_disk(101), asp=1)
```

generate_2d_exponential *Generate 2D exponential Distribution*

Description

Generates a 2D exponential distribution, with an optional argument to take the exponential to a user-defined power.

Usage

```
generate_2d_exponential(
  falloff = 1,
  dim = c(11, 11),
  width = 3,
  rescale_unity = FALSE
)
```

Arguments

`falloff`	Default `1`. Falloff of the exponential.
`dim`	Default `c(11, 11)`. The dimensions of the matrix.
`width`	Default `3` (`-10` to `10`). The range in which to compute the distribution.
`rescale_unity`	Default `FALSE`. If `TRUE`, this will rescale the max value to one. Useful if wanting to plot the distribution with `plot_image()`.

Examples

```
image(generate_2d_exponential(1,31,3), asp=1)
```

generate_2d_gaussian *Generate 2D Gaussian Distribution*

Description

Generates a 2D gaussian distribution, with an optional argument to take the gaussian to a user-defined power.

Usage

```
generate_2d_gaussian(
  sd = 1,
  power = 1,
  dim = c(11, 11),
  width = 3,
  rescale_unity = FALSE
)
```

Arguments

sd	Default 1. Standard deviation of the normal distribution
power	Default 1. Power to take the distribution. Higher values will result in a sharper peak.
dim	Default c(11, 11). The dimensions of the matrix.
width	Default 3 (-10 to 10). The range in which to compute the distribution.
rescale_unity	Default FALSE. If TRUE, this will rescale the max value to one. Useful if wanting to plot the distribution with plot_image().

Examples

```
image(generate_2d_gaussian(1,1,31), asp=1)
```

---

get_string_dimensions *Get String Dimensions*

Description

Calculates font metrics for a specified font, font size, and style.

Usage

```
get_string_dimensions(string, font = "sans", size = 12, align = "center", ...)
```

Arguments

string	The string to be measured.
font	Default "sans".
size	A numeric value specifying the size of the font in points.
align	Default ‘"left"‘. The string alignment.
...	Other arguments to pass to ‘systemfonts::shape_string()‘

Details

The function renders specific characters ("d" for ascender, "g" for descender, "x" for neutral) using the specified font parameters. It calculates the bounding box of each character to determine the necessary adjustments for accurate text positioning.

Value

A data.frame listing the string dimensions.

Examples

```
# Get height of basic sans font
get_string_dimensions("This is a string", size=24)
```

---

interpolate_array *Matrix/Array Interpolation*

Description

Given a series of X and Y coordinates and an array/matrix, interpolates the Z coordinate using bilinear interpolation.

Usage

```
interpolate_array(image, x, y)
```

Arguments

image	3-layer RGB/4-layer RGBA array, rayimg class, or filename of an image.
x	X indices (or fractional index) to interpolate.
y	Y indices (or fractional index) to interpolate.

Value

Either a vector of values (if image is a matrix) or a list of interpolated values from each layer.

Examples

```
#if(interactive()){
#Interpolate a matrix
interpolate_array(volcano,c(10,10.1,11),c(30,30.5,33))
#Interpolate a 3-layer array (returns list for each channel)
interpolate_array(dragon,c(10,10.1,11),c(30,30.5,33))
#end}
```

plot_image *Plot Image*

Description

Displays the image in the current device.

Usage

```
plot_image(
  image,
  draw_grid = FALSE,
  ignore_alpha = FALSE,
  asp = 1,
  new_page = TRUE,
  return_grob = FALSE,
  gp = grid::gpar(),
  angle = 0,
  show_linear = FALSE
)
```

Arguments

image	3-layer RGB/4-layer RGBA array, rayimg class, or filename of an image.
draw_grid	Default FALSE. If TRUE, this will draw a grid in the background to help disambiguate the actual image from the device (helpful if the image background is the same as the device's background).
ignore_alpha	Default FALSE. Whether to ignore the alpha channel when plotting.
asp	Default 1. Aspect ratio of the pixels in the plot. For example, an aspect ratio of 4/3 will slightly widen the image.
new_page	Default TRUE. Whether to call grid::grid.newpage() before plotting the image.
return_grob	Default FALSE. Whether to return the grob object.
gp	A grid::gpar() object to include for the grid viewport displaying the image.
angle	Default 0. Counter-clockwise rotation (in degrees) applied before plotting.
show_linear	Default FALSE. Most data should be gamma corrected before displaying on a screen. Set to TRUE to turn off this gamma correction.

Examples

```
#Plot the dragon array
plot_image(dragon)
#Make pixels twice as wide as tall
plot_image(dragon, asp = 2)
#Plot non-square images
```

```
plot_image(dragon[1:100,,])
#Make pixels twice as tall as wide
plot_image(dragon[1:100,,], asp = 1/2)
```

`plot_image_grid` *Plot Image Grid*

Description

Displays the image in the current device.

Usage

```
plot_image_grid(
  input_list,
  dim = c(1, 1),
  asp = 1,
  draw_grid = FALSE,
  gp = grid::gpar()
)
```

Arguments

`input_list`	List of 3-layer RGB/4-layer RGBA array, `rayimg` class, or image filenames.
`dim`	Default `c(1,1)`. Width by height of output grid.
`asp`	Default `1`. Aspect ratio of the pixels(s). For example, an aspect ratio of `4/3` will slightly widen the image. This can also be a vector the same length of `input_list` to specify an aspect ratio for each image in the grid.
`draw_grid`	Default `FALSE`. If `TRUE`, this will draw a grid in the background to help disambiguate the actual image from the device (helpful if the image background is the same as the device's background).
`gp`	A `grid::gpar()` object to include for the grid viewport displaying the image.

Examples

```
#Plot the dragon array
plot_image_grid(list(dragon, 1-dragon), dim = c(1,2))
plot_image_grid(list(dragon, 1-dragon), dim = c(2,1))
plot_image_grid(list(dragon, NULL, 1-dragon), dim = c(2,2), asp = c(2,1,1/2))
plot_image_grid(list(dragon, NULL, NULL, dragon), dim = c(2,2), asp = c(2,1,1,1/2))
#Plot alongside the depth matrix
dragon_depth_reoriented = render_reorient(dragondepth,
                                          transpose = TRUE,
                                          flipx = TRUE)/2000
plot_image_grid(list(dragondepth/2000, dragon, dragon, dragondepth/2000),
                dim = c(2,2))
```

print.rayimg *Print method for rayimg*

Description

Displays the working colorspace and non-neutral camera metadata alongside a numeric preview.

Usage

```
## S3 method for class 'rayimg'
print(x = NULL, preview_n = 10, decimals = 3, color = TRUE, ...)
```

Arguments

x	Default NULL. A rayimg object.
preview_n	Default 10. Max rows/cols to display in numeric preview.
decimals	Default 3. Number of decimal places to display in numeric preview.
color	Default TRUE. Colorize headers and R/G/B/A numeric cells when supported.
...	Default ". Ignored.

Value

Invisibly returns x.

ray_read_image *Read Image*

Description

Reads an image from a file/array/matrix. From files, supports JPEG, PNG, TIFF, TGA, BMP, PSD, GIF, HDR, PIC, PNM, EXR, and DNG images.

Usage

```
ray_read_image(
  image,
  convert_to_array = FALSE,
  preview = FALSE,
  source_linear = NA,
  normalize = FALSE,
  dng_normalize = FALSE,
  normalize_to = CS_ACESCG,
  normalize_adapt_white = TRUE,
  assume_colorspace = NULL,
```

```
  assume_white = NULL,
  reset_camera_settings = FALSE,
  ...
)
```

Arguments

image	3-layer RGB/4-layer RGBA array, rayimg class, or filename of an image.
convert_to_array	Default FALSE. Whether to convert 2D B&W images/matrices to RGBA arrays.
preview	Default FALSE. If TRUE, it will display the image in addition to returning it.
source_linear	Default NA, automatically determined based on the image type. Whether the image source is linear data or sRGB. TRUE for matrices, arrays, and floating-point HDR formats (EXR, HDR), and DNG; FALSE for all other file-based formats (e.g., jpeg, png, tiff, tga, bmp, psd, gif, pic, pnm).
normalize	Default FALSE. If TRUE, convert to normalize_to space on read. Note that rayimg inputs will keep their colorspace and ignore this option–use render_convert_colorspace() to change an existing rayimg to a new colorspace.
dng_normalize	Default FALSE. If TRUE, DNG data is normalized to 0..1, using black/white levels; if FALSE, raw values are returned.
normalize_to	Default CS_ACESCG. Target colorspace when normalize=TRUE.
normalize_adapt_white	Default TRUE. If TRUE, Bradford-adapt source white to target white.
assume_colorspace	Default NULL. A colorspace descriptor (e.g., CS_SRGB, CS_ACESCG). If given, the loaded image will be *tagged* with this space rather than the default inferred by file type. No pixel conversion is performed unless normalize=TRUE.
assume_white	Default NULL. Scene/display white for the loaded image. Either a named white ("D60","D65","D50","D55","D75","E") or XYZ with Y=1. If NULL, uses assume_colorspace$white_xyz.
reset_camera_settings	Default FALSE. If TRUE, reset informational exposure/ISO metadata to neutral values (exposure = 0, iso = 100) when wrapping an existing rayimg or attributed array.
...	Arguments to pass to jpeg::readJPEG, png::readPNG, tiff::readTIFF, or libopenexr::read_exr() for supported formats.

Value

A rayimg RGBA array.

Examples

```
#Write as a png
tmparr = tempfile(fileext=".png")
ray_read_image(dragon) |>
 ray_write_image(tmparr)
```

```
ray_read_image(tmparr) |>
 plot_image()
#Write as a JPEG (passing quality arguments via ...)
tmparr = tempfile(fileext=".jpg")
ray_read_image(dragon) |>
 ray_write_image(tmparr, quality = 0.2)
ray_read_image(tmparr) |>
 plot_image()
#Write as a tiff
tmparr = tempfile(fileext=".tiff")
ray_read_image(dragon) |>
 ray_write_image(tmparr)
ray_read_image(tmparr) |>
  plot_image()

#Write as an exr
tmparr = tempfile(fileext=".exr")
ray_read_image(dragon) |>
 ray_write_image(tmparr)
ray_read_image(tmparr) |>
  plot_image()
```

`ray_write_image` *Write Image*

Description

Takes an RGB array/filename and writes it to file.

Usage

```
ray_write_image(image, filename, clamp = FALSE, write_linear = NA, ...)
```

Arguments

`image`	3-layer RGB/4-layer RGBA array, `rayimg` class, or filename of an image.
`filename`	File to write to, with filetype determined by extension. Filetype can be `PNG`, `JPEG`, `TIFF`, `EXR`, or `DNG`.
`clamp`	Default `FALSE`, automatically determined. Whether to clamp the image to 0-1. If the file extension is `PNG` of `JPEG`, this is forced to `TRUE`.
`write_linear`	Default `NA`, automatically determined. By default, images will be gamma corrected (write_linear = FALSE) for all file types except `EXR` and `DNG` (which are linear formats), unless otherwise specified.
`...`	Arguments to pass to either `jpeg::writeJPEG`, `png::writePNG`, `libopenexr::write_exr`, or `tiff::writeTIFF`.

Value

A rayimg RGBA array.

Examples

```
#Write as a png
tmparr = tempfile(fileext=".png")
ray_read_image(dragon) |>
 ray_write_image(tmparr)
ray_read_image(tmparr) |>
 plot_image()
#Write as a JPEG (passing quality arguments via ...)
tmparr = tempfile(fileext=".jpg")
ray_read_image(dragon) |>
 ray_write_image(tmparr, quality = 0.2)
ray_read_image(tmparr) |>
 plot_image()
#Write as a tiff
tmparr = tempfile(fileext=".tiff")
ray_read_image(dragon) |>
 ray_write_image(tmparr)
ray_read_image(tmparr) |>
  plot_image()
```

---

render_alpha_outline *Render Alpha Outline*

Description

Creates a colored RGBA outline or halo from an image alpha channel or a supplied mask. When image is supplied, the original image is composited over the outline by default.

Usage

```
render_alpha_outline(
  image = NULL,
  mask = NULL,
  expand = 0,
  edge_softness = 0.1,
  blur = 0,
  gap_fill = 1,
  gap_fill_alpha_threshold = 0.25,
  color = "black",
  alpha = 1,
  pad = 0,
  composite = TRUE,
  filename = NULL,
```

```
  preview = FALSE
)
```

Arguments

image	Default NULL. Image whose alpha channel will be used as the mask. If mask is supplied, image is only used for metadata.
mask	Default NULL. Optional logical or numeric matrix to use as the outline mask instead of image alpha.
expand	Default 0. Number of pixels to expand the mask.
edge_softness	Default 0.1. Width of the softened halo edge transition, in pixels.
blur	Default 0. Gaussian blur standard deviation applied to the final outline alpha.
gap_fill	Default 1. Maximum alpha gap width, in pixels, to bridge in the expanded outline alpha. This fills border-connected breaks and notches caused by discrete distance quantization. Set to 0 to disable.
gap_fill_alpha_threshold	Default 0.25. Alpha threshold used to classify border gaps. Gap filling only modifies pixels below this threshold that are connected to the image exterior; enclosed interior holes surrounded by pixels at or above this threshold are left unchanged.
color	Default "black". Outline color.
alpha	Default 1. Overall outline alpha multiplier.
pad	Default 0. Padding in pixels before computing the outline. If NULL, padding is computed from expand, edge_softness, and blur.
composite	Default TRUE. If TRUE and image is supplied, composite the outline below the original image. Set to FALSE to return the outline alone.
filename	Default NULL. File to save the image to. If NULL and preview = FALSE, returns the output image.
preview	Default FALSE. If TRUE, display the outline image.

Value

A rayimg RGBA output image with a padding attribute.

Examples

```
txt = render_text_image("\U0001F409",  size = 120, background_alpha = 0)
render_alpha_outline(txt, expand = 2, blur = 1, color = "purple") |>
 plot_image()

#Make a rainbow outline
render_alpha_outline(txt, expand = 2, blur = 0.1, color = "darkgreen") |>
 render_alpha_outline(expand = 2, blur = 0.1, color = "yellow") |>
 render_alpha_outline(expand = 2, blur = 0.1, color = "orange") |>
 render_alpha_outline(expand = 2, blur = 0.1, color = "red") |>
 render_alpha_outline(expand = 2, blur = 0.1, color = "purple") |>
```

```
render_alpha_outline(expand = 2, blur = 0.1, color = "blue") |>
plot_image()
```

---

render_bokeh *Render Bokeh*

Description

Takes an image and a depth map to render the image with depth of field (i.e. similar to "Portrait Mode" in an iPhone). User can specify a custom bokeh shape, or use one of the built-in bokeh types.

Usage

```
render_bokeh(
  image,
  depthmap,
  focus = 0.5,
  focallength = 100,
  fstop = 4,
  filename = NULL,
  preview = TRUE,
  preview_focus = FALSE,
  bokehshape = "circle",
  bokehintensity = 1,
  bokehlimit = 0.8,
  rotation = 0,
  aberration = 0,
  progress = interactive(),
  ...
)
```

Arguments

image	3-layer RGB/4-layer RGBA array, rayimg class, or filename of an image.
depthmap	Depth map filename or 1d array.
focus	Defaults 0.5. Depth in which to blur.
focallength	Default 100. Focal length of the virtual camera.
fstop	Default 4. F-stop of the virtual camera.
filename	Default NULL. The filename of the image to be saved. If this is not given, the image will be plotted instead.
preview	Default TRUE. If FALSE, it will not display the image and just return the RGB array.
preview_focus	Default FALSE. If TRUE, a red line will be drawn across the image showing where the camera will be focused.

bokehshape	Default `circle`. Also built-in: `hex`. The shape of the bokeh. If the user passes in a 2D matrix, that matrix will control the shape of the bokeh.
bokehintensity	Default `1`. Intensity of the bokeh when the pixel intensity is greater than `bokehlimit`.
bokehlimit	Default `0.8`. Limit after which the bokeh intensity is increased by `bokehintensity`.
rotation	Default `0`. Number of degrees to rotate the hexagonal bokeh shape.
aberration	Default `0`. Adds chromatic aberration to the image. Maximum of `1`.
progress	Default `TRUE`. Whether to display a progress bar.
...	Additional arguments to pass to `plot_image()` if `preview = TRUE`.

Value

A `rayimg` RGBA array.

Examples

```
#Plot the dragon
plot_image(dragon)
#Plot the depth map
plot_image(dragondepth/1500)
#Preview the focal plane:
render_bokeh(dragon, dragondepth, focus=950, preview_focus = TRUE)
#Change the focal length:
render_bokeh(dragon, dragondepth, focus=950, focallength=300)
#Add chromatic aberration:
render_bokeh(dragon, dragondepth, focus=950, focallength=300, aberration = 0.5)
#Change the focal distance:
render_bokeh(dragon, dragondepth, focus=600, focallength=300)
render_bokeh(dragon, dragondepth, focus=1300, focallength=300)
#Change the bokeh shape to a hexagon:
render_bokeh(dragon, dragondepth, bokehshape = "hex",
            focallength=300, focus=600)
#Change the bokeh intensity:
render_bokeh(dragon, dragondepth,
            focallength=400, focus=900, bokehintensity = 1)
render_bokeh(dragon, dragondepth,
            focallength=400, focus=900, bokehintensity = 3)
#Rotate the hexagonal shape:
render_bokeh(dragon, dragondepth, bokehshape = "hex", rotation=15, bokehintensity=3,
            focallength=400, focus=800)
```

`render_boolean_distance` *Render Boolean Distance*

Description

Takes a logical matrix and returns the nearest distance to each TRUE value.

Usage

```
render_boolean_distance(boolean, rescale = FALSE)
```

Arguments

boolean	Logical matrix (or matrix of 1s and 0s), where distance will be measured to the TRUE values.
rescale	Default FALSE. Rescales the calculated distance to a range of 0-1. Useful for visualizing the distance matrix.

Value

Matrix of distance values.

Examples

```
#Measure distance to
plot_image(render_boolean_distance(t(volcano) > 150))
plot_image(render_boolean_distance(t(volcano) < 150))
#If we want to rescale this to zero to one (to visualize like an image), set rescale=TRUE
plot_image(render_boolean_distance(t(volcano) > 150,rescale=TRUE))
```

---

render_bw *Render Black and White*

Description

Transforms an image to black and white, preserving luminance.

Usage

```
render_bw(
  image,
  rgb_coef = c(0.2126, 0.7152, 0.0722),
  filename = NULL,
  preview = FALSE
)
```

Arguments

image	3-layer RGB/4-layer RGBA array, rayimg class, or filename of an image.
rgb_coef	Default c(0.2126, 0.7152, 0.0722). Length-3 numeric vector listing coefficients to convert RGB to luminance.
filename	Default NULL. The filename of the image to be saved. If this is not given, the image will be plotted instead.
preview	Default FALSE. Whether to plot the convolved image, or just to return the values.

Value

A rayimg RGBA array.

Examples

```
#Plot the image with a title
dragon |>
 render_title("Dragon", title_offset=c(10,10), title_bar_color="black",
           title_size=20, title_color = "white") |>
 render_bw(preview = TRUE)
```

---

render_clamp *Clamp Image*

Description

Clamps an image to a user-specified range, ignoring the alpha channel.

Usage

```
render_clamp(
  image,
  min_value = 0,
  max_value = 1,
  filename = NA,
  preview = FALSE,
  ...
)
```

Arguments

image	3-layer RGB/4-layer RGBA array, rayimg class, or filename of an image.
min_value	Default 0. Minimum value to clamp the RGB channels in the image to.
max_value	Default 1. Maximum value to clamp the RGB channels in the image to.
filename	Default NA. Filename
preview	Default FALSE. If TRUE, it will display the image in addition to returning it.
...	Arguments to pass to either jpeg::readJPEG, png::readPNG, or tiff::readTIFF.

Value

A rayimg RGBA array.

Examples

```
#The image of the unchanged image
range(dragon)
#Clamp the maximum and minimum values to one and zero
render_clamp(dragon) |>
 range()
```

`render_color_correction` *Render Color Correction Matrix (3x3, linear RGB)*

Description

Apply a technical/look 3x3 matrix to an image

Usage

```
render_color_correction(
  image,
  matrix = diag(3),
  filename = NULL,
  preview = FALSE
)
```

Arguments

`image`	Default `NULL`. 3-layer RGB/4-layer RGBA array, `rayimg`, or filename.
`matrix`	Default `diag(3)`. 3x3 numeric matrix.
`filename`	Default `NULL`. Output path.
`preview`	Default `FALSE`. If `TRUE`, display the image.

Value

A `rayimg` RGBA array.

Examples

```
# We will start with an image that's too warm--we want to correct it to
# match the color of the second as closely as possible.
dragon_D50 = render_white_balance(dragon, target_white = "D50", bake = TRUE)
dragon_D75 = render_white_balance(dragon, target_white = "D75", bake = TRUE)
plot_image(dragon_D50)
plot_image(dragon_D75)

# Fucntion to fit a color correction matrix
fit_cc_matrix = function(src, tgt) {
```

```
stopifnot(ncol(src) == 3, ncol(tgt) == 3, nrow(src) == nrow(tgt))
M_t = apply(tgt, 2, function(y) qr.solve(src, y))
t(M_t)
}

# Sample N RGBs from an image for fit
rand_samples = function(img, n = 5000, seed=1) {
set.seed(seed)
d = dim(img)
ys = sample.int(d[1], n, TRUE); xs = sample.int(d[2], n, TRUE)
cbind(img[cbind(ys,xs,1)], img[cbind(ys,xs,2)], img[cbind(ys,xs,3)])
}

S = rand_samples(dragon_D50)
T = rand_samples(dragon_D75)

M = fit_cc_matrix(S, T)
# Optionally, regularize toward identity to avoid overfitting
#  M = 0.8 * M + 0.2 * diag(3)

updated_source = render_color_correction(dragon_D50, matrix = M)

# Matching the old image (grey) to the target image (orange),
# giving a new image (black).
plot(S,pch=16,cex=0.5,col="grey")
points(T,pch=16,cex=0.5,col="orange")
points(rand_samples(updated_source,seed=2),pch=16,cex=0.5,col="black")

# Plot the images
plot_image(render_title(dragon_D50, "OG Image",
           title_color = "white", title_size = 12))
plot_image(render_title(dragon_D75, "Target Image",
           title_color = "white", title_size = 12))
plot_image(render_title(updated_source, "Corrected OG Image",
           title_color = "white", title_size = 12))
```

`render_convert_colorspace` *Render Convert Colorspace*

Description

Convert between two RGB spaces via XYZ, with optional white adaptation. Operates on **linear** RGB.

Usage

```
render_convert_colorspace(
  image,
  from_mats = NA,
```

```
  to_mats = CS_ACESCG,
  adapt_white = TRUE,
  from_white = NA,
  to_white = NA,
  filename = NULL,
  preview = FALSE
)
```

Arguments

image	3-layer RGB/4-layer RGBA array, rayimg, or filename.
from_mats	Default NA. Source space object; when NA, pulled from attr(src,"colorspace").
to_mats	Default CS_ACESCG. Target space object.
adapt_white	Default TRUE. Apply Bradford CAT from from_white to to_white.
from_white	Default NA. Source white (name or XYZ, Y=1). When NA, uses attr(src,"white_current").
to_white	Default NA. Target white (name or XYZ, Y=1). When NA, uses to_mats$white_xyz.
filename	Default NULL. Output path.
preview	Default FALSE. If TRUE, display the image.

Value

A rayimg RGBA array tagged with the **target** space.

Examples

```
# Read photo, convert to ACEScg with CAT (scene)
photo = ray_read_image(sunset_image, normalize = FALSE)
photo_aces = render_convert_colorspace(
photo,
to_mats = CS_ACESCG,
adapt_white = TRUE
)
tmp_txt = tempfile(fileext = ".png")
render_text_image(
"Sunset",
size = 60,
filename = tmp_txt,
color = "#c300ffff",
background_alpha = 0
)
# Read logo (display-referred), convert primaries only (no CAT)
logo = ray_read_image(tmp_txt, normalize = FALSE) # sRGB/D65
logo_aces = render_convert_colorspace(
logo,
to_mats = CS_ACESCG,
adapt_white = FALSE
)

# Composite in ACEScg, then display (plot_image converts to sRGB/D65 + OETF)
```

```
# Here, we also turn overlay conversion in [render_image_overlay()] off,
 # to show what happens when you don't account for the colorspace difference.
 # By default [render_image_overlay()] will do this for you.
comp1 = render_image_overlay(
photo_aces,
logo_aces,
convert_overlay_colorspace = FALSE
)
comp2 = render_image_overlay(
photo_aces,
logo,
convert_overlay_colorspace = FALSE
)
  #Note the color differences, which arise from the mismatched colorspace on the right.
 plot_image_grid(list(comp1, comp2), dim = c(1, 2))
```

`render_convolution` *Render Convolution*

Description

Takes an image and applies a convolution operation to it, using a user-supplied or built-in kernel. Edges are calculated by limiting the size of the kernel to only that overlapping the actual image (renormalizing the kernel for the edges).

Usage

```
render_convolution(
  image,
  kernel = "gaussian",
  kernel_dim = 11,
  kernel_extent = 3,
  absolute = TRUE,
  min_value = NULL,
  include_alpha = FALSE,
  filename = NULL,
  preview = FALSE,
  progress = FALSE
)
```

Arguments

`image`	3-layer RGB/4-layer RGBA array, `rayimg` class, or filename of an image.
`kernel`	Default `gaussian`. By default, an 11x11 Gaussian kernel with a mean of `0` and a standard deviation of 1, running from `-kernel_extent` to `kernel_extent`. If numeric, this will be the standard deviation of the normal distribution. If a matrix, it will be used directly as the convolution kernel (but resized always to be an odd number of columns and rows).

kernel_dim	Default 11. The dimension of the gaussian kernel. Ignored if user specifies their own kernel.
kernel_extent	Default 3. Extent over which to calculate the kernel.
absolute	Default TRUE. Whether to take the absolute value of the convolution.
min_value	Default NULL. If numeric, specifies the minimum value (for any color channel) for a pixel to have the convolution performed.
include_alpha	Default FALSE. Whether to include the alpha channel in the convolution.
filename	Default NULL. The filename of the image to be saved. If this is not given, the image will be plotted instead.
preview	Default TRUE. Whether to plot the convolved image, or just to return the values.
progress	Default TRUE. Whether to display a progress bar.

Value

A rayimg RGBA array.

Examples

```
#Perform a convolution with the default gaussian kernel
plot_image(dragon)
#Perform a convolution with the default gaussian kernel
render_convolution(dragon, preview = TRUE)
#Increase the width of the kernel
render_convolution(dragon, kernel = 2, kernel_dim=21,kernel_extent=6, preview = TRUE)
#Perform edge detection using an edge detection kernel
edge = matrix(c(-1,-1,-1,-1,8,-1,-1,-1,-1),3,3)
render_convolution(render_bw(dragon), kernel = edge, preview = TRUE, absolute=FALSE)
#Perform edge detection with Sobel matrices
sobel1 = matrix(c(1,2,1,0,0,0,-1,-2,-1),3,3)
sobel2 = matrix(c(1,2,1,0,0,0,-1,-2,-1),3,3,byrow=TRUE)
sob1 = render_convolution(render_bw(dragon), kernel = sobel1)
sob2 = render_convolution(render_bw(dragon), kernel = sobel2)
sob_all = sob1 + sob2
plot_image(sob1)
plot_image(sob2)
plot_image(sob_all)

#Only perform the convolution on bright pixels (bloom)
render_convolution(dragon, kernel = 5, kernel_dim=24, kernel_extent=24,
                   min_value=1, preview = TRUE)
#Use a built-in kernel:
render_convolution(dragon, kernel = generate_2d_exponential(falloff=2, dim=31, width=21),
                   preview = TRUE)
#We can also apply this function to matrices:
volcano |> image()
volcano |>
 render_convolution(kernel=generate_2d_gaussian(sd=1,dim=31)) |>
 image()
#Use a custom kernel (in this case, an X shape):
```

```
custom = diag(10) + (diag(10)[,10:1])
plot_image(custom)
render_convolution(dragon, kernel = custom, preview = TRUE)
```

`render_convolution_fft` *Render Convolution FFT*

Description

Takes an image and applies a convolution operation to it, using a user-supplied or built-in kernel. This function uses a fast Fourier transform and does the convolution in the frequency domain, so it should be faster for much larger kernels.

Usage

```
render_convolution_fft(
  image,
  kernel = "gaussian",
  kernel_dim = c(11, 11),
  kernel_extent = 3,
  absolute = TRUE,
  pad = 50,
  include_alpha = FALSE,
  filename = NULL,
  preview = FALSE
)
```

Arguments

`image`	3-layer RGB/4-layer RGBA array, `rayimg` class, or filename of an image.
`kernel`	Default `gaussian`. By default, an 11x11 Gaussian kernel with a mean of `0` and a standard deviation of `1`, running from `-kernel_extent` to `kernel_extent`. If numeric, this will be the standard deviation of the normal distribution. If a matrix, it will be used directly as the convolution kernel (but resized always to be an odd number of columns and rows).
`kernel_dim`	Default `c(11, 11)`. The dimension of the `gaussian` kernel. Ignored if user specifies their own kernel.
`kernel_extent`	Default `3`. Extent over which to calculate the kernel.
`absolute`	Default `TRUE`. Whether to take the absolute value of the convolution.
`pad`	Default `50`. Amount to pad the image to remove edge effects.
`include_alpha`	Default `FALSE`. Whether to include the alpha channel in the convolution.
`filename`	Default `NULL`. The filename of the image to be saved. If this is not given, the image will be plotted instead.
`preview`	Default `FALSE`. Whether to plot the convolved image, or just to return the values.

Value

A rayimg RGBA array.

Examples

```
#Perform a convolution with the default gaussian kernel
plot_image(dragon)
#Perform a convolution with the default gaussian kernel
render_convolution_fft(dragon, kernel=0.1, preview = TRUE)
#Increase the width of the kernel
render_convolution_fft(dragon, kernel = 2, kernel_dim=21,kernel_extent=6, preview = TRUE)
#Use a built-in kernel:
render_convolution_fft(dragon, kernel = generate_2d_exponential(falloff=2, dim=31, width=21),
                       preview = TRUE)
#Perform edge detection
edge = matrix(c(-1,-1,-1,-1,8,-1,-1,-1,-1),3,3)
render_convolution_fft(render_bw(dragon), kernel = edge, preview = TRUE)
#Perform edge detection with Sobel matrices
sobel1 = matrix(c(1,2,1,0,0,0,-1,-2,-1),3,3)
sobel2 = matrix(c(1,2,1,0,0,0,-1,-2,-1),3,3,byrow=TRUE)
sob1 = render_convolution_fft(render_bw(dragon), kernel = sobel1)
sob2 = render_convolution_fft(render_bw(dragon), kernel = sobel2)
sob_all = sob1 + sob2
plot_image(sob1)
plot_image(sob2)
plot_image(sob_all)
#We can also apply this function to matrices:
volcano |> image()
volcano |>
 render_convolution_fft(kernel=generate_2d_gaussian(sd=1,dim=31)) |>
 image()
# Because this function uses the fast Fourier transform, large kernels will be much faster
# than the same size kernels in [render_convolution()]
render_convolution_fft(dragon, kernel_dim = c(200,200) , preview = TRUE)
#Use a custom kernel (in this case, an X shape):
custom = diag(10) + (diag(10)[,10:1])
#Normalize
custom = custom / 20
plot_image(custom*20)
render_convolution_fft(dragon, kernel = custom, preview = TRUE)
```

---

render_crop *Crop Image*

Description

Crops an image (or matrix) either by specifying axis-aligned limits (width_limits/height_limits) or by specifying a centered box size via center.

Usage

```
render_crop(
  image = NULL,
  width_limits = NULL,
  height_limits = NULL,
  center = NULL,
  filename = NULL,
  preview = FALSE
)
```

Arguments

image	Default NULL. 3-layer RGB/4-layer RGBA array, rayimg class, or filename of an image.
width_limits	Default NULL. Length-2 numeric (xmin, xmax) along the width (x/columns). Values in 0..1 are treated as fractions of image width; otherwise they are pixel indices. Mutually exclusive with center. If provided alone, height_limits defaults to full height.
height_limits	Default NULL. Length-2 numeric (ymin, ymax) along the height (y/rows). Values in 0..1 are treated as fractions of image height; otherwise they are pixel indices. Mutually exclusive with center. If provided alone, width_limits defaults to full width.
center	Default NULL. Length-2 numeric (height, width) for a center-crop. Values in (0,1] are treated as fractions of (image height, image width); otherwise pixels. Mutually exclusive with width_limits/height_limits.
filename	Default NULL. The filename of the image to be saved. If this is not given, the image will be plotted instead.
preview	Default FALSE. Whether to plot the cropped image, or just to return the values.

Value

A rayimg array with the cropped region.

Examples

```
# Left half of the image
render_crop(dragon,
            width_limits = c(0, 0.50),
            height_limits = c(0, 1)) |>
 plot_image()
# Right half of the image
render_crop(dragon,
            width_limits = c(0.50, 1),
            height_limits = c(0, 1)) |>
 plot_image()
# Top middle of the image (25%-50% height, 25%-75% width)
render_crop(dragon,
            height_limits = c(0.25, 0.50),
```

```
           width_limits  = c(0.25, 0.75)) |>
 plot_image()
# Top middle, explicit pixel coords (for a 200x200 image)
render_crop(dragon,
           height_limits = c(50, 100),
           width_limits  = c(50, 150)) |>
 plot_image()
# Center-crop: 50% height and width of the image
render_crop(dragon, center = c(0.5, 0.5)) |>
 plot_image()
# Center-crop: fixed 50x100 pixels
render_crop(dragon, center = c(50, 100)) |>
 plot_image()
```

`render_exposure` *Apply exposure compensation and ISO gain*

Description

Apply exposure compensation and ISO gain

Usage

```
render_exposure(
  image,
  exposure = 0,
  iso = NA,
  auto = FALSE,
  percentile = 0.99,
  verbose = FALSE,
  filename = NA,
  preview = FALSE,
  ...
)
```

Arguments

`image`	3-layer RGB/4-layer RGBA array, `rayimg` class, or filename of an image.
`exposure`	Default `0`. Exposure compensation in stops; applied directly to RGB values.
`iso`	Default `NA`. ISO gain to apply directly to RGB values. `NA` applies no ISO gain and preserves the current informational ISO metadata.
`auto`	Default `FALSE`. If `TRUE`, automatically add the exposure needed to place a luminance percentile at 1.0 for LDR display/output. `exposure` is then treated as an additional compensation value.
`percentile`	Default `0.99`. Luminance percentile used when `auto = TRUE`. Must be greater than 0 and less than or equal to 1. Use 1 for max luminance.

verbose	Default FALSE. If TRUE and auto = TRUE, print the automatically computed exposure adjustment.
filename	Default NA. Output path.
preview	Default FALSE. If TRUE, display the image.
...	Additional args passed to plot_image() (when preview=TRUE) or to ray_write_image() (when filename is given).

Value

A rayimg RGBA array.

Examples

```
# LDR/sRGB (auto): decodes to linear, applies EV, records camera metadata
  render_exposure(dragon, exposure = +1, preview = TRUE)
# Force linear/HDR behavior
  render_exposure(dragon * 2, exposure = -1, preview = TRUE)
```

---

render_gamma_linear *Render Gamma/Linear*

Description

Convert sRGB-encoded color data to linear light (and vice versa).

Usage

```
render_gamma_linear(color, srgb_to_linear = TRUE, as_hex = FALSE)
```

Arguments

color	A rayimg, hex color string (or vector), length-3 numeric RGB vector, or RGB/RGBA matrix.
srgb_to_linear	Default TRUE. If TRUE, converts sRGB-encoded data to linear light. If FALSE, converts linear-light data to sRGB.
as_hex	Default FALSE. When srgb_to_linear = FALSE and input is an RGB/RGBA matrix or array, returns hex: for matrices a length-n vector; for arrays a length-prod(dim(x)[1:2]) vector in column-major order. Alpha (if present) is passed through.

Value

Same shape/type as input unless as_hex = TRUE on matrix/array with srgb_to_linear = FALSE, in which case a character vector of hex codes is returned.

Examples

```
# Numeric RGB vector (sRGB -> linear -> back to sRGB)
srgb_vec = c(0.2, 0.4, 0.6)
linear_vec = render_gamma_linear(srgb_vec)
render_gamma_linear(linear_vec, srgb_to_linear = FALSE)

# Hex input (single color)
linear_from_hex = render_gamma_linear("#336699")
render_gamma_linear(linear_from_hex, srgb_to_linear = FALSE, as_hex = TRUE)

# Greyscale matrix (treated as image plane)
grey_image = matrix(c(0.1, 0.5, 0.9, 0.3), nrow = 2)
linear_grey = render_gamma_linear(grey_image)
render_gamma_linear(linear_grey, srgb_to_linear = FALSE, as_hex = TRUE)

# Array with a single greyscale channel
grey_pixels = array(c(0.2, 0.7), dim = c(1, 2, 1))
linear_grey_arr = render_gamma_linear(grey_pixels)
render_gamma_linear(linear_grey_arr, srgb_to_linear = FALSE, as_hex = TRUE)

# Array with greyscale + alpha
grey_alpha = array(c(0.3, 0.9, 0.6, 0.4), dim = c(1, 2, 2))
linear_grey_alpha = render_gamma_linear(grey_alpha)
render_gamma_linear(linear_grey_alpha, srgb_to_linear = FALSE, as_hex = TRUE)

# RGB array (3 channels)
rgb_pixels = array(
  c(
    0.2, 0.4, 0.6,
    0.7, 0.1, 0.3
  ),
  dim = c(1, 2, 3)
)
linear_rgb = render_gamma_linear(rgb_pixels)
render_gamma_linear(linear_rgb, srgb_to_linear = FALSE, as_hex = TRUE)

# RGBA array (alpha channel preserved)
rgba_pixels = array(
  c(
    0.2, 0.4, 0.6, 0.5,
    0.7, 0.1, 0.3, 0.8
  ),
  dim = c(1, 2, 4)
)
linear_rgba = render_gamma_linear(rgba_pixels)
render_gamma_linear(linear_rgba, srgb_to_linear = FALSE, as_hex = TRUE)

# rayimg input retains metadata and supports hex conversion
linear_img = render_gamma_linear(ray_read_image(rgba_pixels))
render_gamma_linear(linear_img, srgb_to_linear = FALSE, as_hex = TRUE)
```

render_image_overlay *Add Overlay*

Description

Takes an RGB array/filename and composites an image overlay over the full image. For pixel-positioned sprite, label, glyph, or billboard overlays, use `render_sprite_overlay()`.

Usage

```
render_image_overlay(
  image,
  image_overlay = NULL,
  rescale_original = FALSE,
  convert_overlay_colorspace = TRUE,
  alpha = NA,
  filename = NULL,
  preview = FALSE
)
```

Arguments

`image`	3-layer RGB/4-layer RGBA array, `rayimg` class, or filename of an image.
`image_overlay`	Default `NULL`. 3-layer RGB/4-layer RGBA array, `rayimg` class, or filename of an image. This image will be resized to the dimensions of `image` if it does not match exactly, unless `rescale_original = TRUE`.
`rescale_original`	Default `FALSE`. If `TRUE`, function will resize the original image to match the overlay.
`convert_overlay_colorspace`	Default `TRUE`. Whether to convert the overlay's colorspace to match the underlying image.
`alpha`	Default `NA`, using overlay's alpha channel. Otherwise, this sets the alpha transparency by multiplying the existing alpha channel by this value (between 0 and 1).
`filename`	Default `NULL`. File to save the image to. If `NULL` and `preview = FALSE`, returns an RGB array.
`preview`	Default `FALSE`. If `TRUE`, it will display the image in addition to returning it.

Value

A `rayimg` RGBA array.

Examples

```
#Plot the dragon
plot_image(dragon)
#Add an overlay of a red semi-transparent circle:
circlemat = generate_2d_disk(min(dim(dragon)[1:2]))
circlemat = circlemat/max(circlemat)

#Create RGBA image, with a transparency of 0.5
rgba_array = array(1, dim=c(nrow(circlemat),ncol(circlemat),4))
rgba_array[,,1] = circlemat
rgba_array[,,2] = 0
rgba_array[,,3] = 0
dragon_clipped = dragon
dragon_clipped[dragon_clipped > 1] = 1
render_image_overlay(dragon_clipped, image_overlay = rgba_array,
                 alpha=0.5, preview = TRUE)

# Read photo, convert to ACEScg with CAT (scene)
photo = ray_read_image(sunset_image, normalize = FALSE)
photo_aces = render_convert_colorspace(
photo,
to_mats = CS_ACESCG,
adapt_white = TRUE
)
tmp_txt = tempfile(fileext = ".png")
render_text_image(
"Sunset",
size = 60,
filename = tmp_txt,
color = "#c300ff",
background_alpha = 0
)
# Read logo (display-referred), convert primaries only (no CAT)
logo = ray_read_image(tmp_txt, normalize = FALSE) # sRGB/D65
logo_aces = render_convert_colorspace(
logo,
to_mats = CS_ACESCG,
adapt_white = FALSE
)

# Composite in ACEScg, then display (plot_image converts to sRGB/D65 + OETF)
# Here, we also turn overlay conversion in [render_image_overlay()] off,
 # to show what happens when you don't account for the colorspace difference.
 # By default [render_image_overlay()] will do this for you.
comp1 = render_image_overlay(
photo_aces,
logo_aces,
convert_overlay_colorspace = FALSE
) |>
   render_title(title_text = "#c300ff: Match",
                title_bar_color = "white", title_color = "#c300ff", title_bar_alpha=1)
comp2 = render_image_overlay(
```

```
photo_aces,
logo,
convert_overlay_colorspace = FALSE
) |>
 render_title(title_text = "#c300ff: Incorrect",
               title_bar_color = "white", title_color = "#c300ff", title_bar_alpha=1)

plot_image_grid(list(comp1, comp2), dim = c(1, 2))
```

`render_padding` *Add Transparent Padding*

Description

Expands an image or matrix canvas by adding constant-value padding around the original image.

Usage

```
render_padding(
  image,
  pad = 10,
  color = "black",
  alpha = NULL,
  filename = NULL,
  preview = FALSE
)
```

Arguments

`image`	3-layer RGB/4-layer RGBA array, `rayimg` class, matrix, or filename of an image.
`pad`	Default `10`. Padding in pixels. Use a scalar for equal padding on all sides, or `c(top, right, bottom, left)`.
`color`	Default `"black"`. Padding color. Can be a scalar, RGB/RGBA numeric vector, or R color string.
`alpha`	Default `NULL`. Padding alpha for images with an alpha channel. If `NULL`, uses the alpha channel supplied in `color` when present, otherwise defaults to `0` when `color` is omitted and `1` when `color` is supplied. Ignored for RGB and matrix inputs.
`filename`	Default `NULL`. File to save the image to. If `NULL` and `preview = FALSE`, returns the padded image.
`preview`	Default `FALSE`. If `TRUE`, display the padded image.

Value

A padded `rayimg` image or matrix.

Examples

```
render_padding(dragon, pad = 20) |>
 plot_image()
```

`render_reorient` *Reorient Image*

Description

Reorients an image or matrix. Transformations are applied in this order: x, y, and transpose.

Usage

```
render_reorient(
  image,
  flipx = FALSE,
  flipy = FALSE,
  transpose = FALSE,
  filename = NULL,
  preview = FALSE
)
```

Arguments

`image`	3-layer RGB/4-layer RGBA array, `rayimg` class, or filename of an image.
`flipx`	Default `FALSE`. Flip horizontally
`flipy`	Default `FALSE`. Flip vertically.
`transpose`	Default `FALSE`. Transpose image.
`filename`	Default `NULL`. The filename of the image to be saved. If this is not given, the image will be plotted instead.
`preview`	Default `FALSE`. Whether to plot the convolved image, or just to return the values.

Value

3-layer RGB reoriented array or matrix.

Examples

```
#Original orientation
plot_image(dragon)
#Flip the dragon image horizontally
dragon |>
 render_reorient(flipx = TRUE) |>
 plot_image()
#Flip the dragon image vertically
```

```
dragon |>
 render_reorient(flipy = TRUE) |>
 plot_image()
#Transpose the dragon image
dragon |>
 render_reorient(transpose = TRUE) |>
 plot_image()
```

---

render_resized *Resize Image*

Description

Resizes an image or a matrix, using bilinear interpolation.

Usage

```
render_resized(
  image,
  mag = 1,
  dims = NULL,
  filename = NULL,
  preview = FALSE,
  method = "tri"
)
```

Arguments

image	3-layer RGB/4-layer RGBA array, rayimg class, or filename of an image.
mag	Default 1. Amount to magnify the image, preserving aspect ratio. Overridden if dim is not NULL.
dims	Default NULL. Exact resized dimensions.
filename	Default NULL. The filename of the image to be saved. If this is not given, the image will be plotted instead.
preview	Default FALSE. Whether to plot the convolved image, or just to return the values.
method	Default trilinear. Filters to up/downsample the image. Options: bilinear, box, trilinear, catmull, mitchell.

Value

A rayimg RGBA array.

Examples

```
#Plot the image with a title
dragon |>
 render_title("Dragon", title_offset=c(10,10), title_bar_color="black",
           title_size=20, title_color = "white") |>
 plot_image()
#Half of the resolution
render_resized(dragon, mag = 1/2) |>
 render_title("Dragon (half res)", title_offset=c(5,5), title_bar_color="black",
           title_size=10, title_color = "white") |>
 plot_image()
#Double the resolution
render_resized(dragon, mag = 2) |>
 render_title("Dragon (2x res)", title_offset=c(20,20), title_bar_color="black",
           title_size=40, title_color = "white") |>
 plot_image()
#Specify the exact resulting dimensions
render_resized(dragon, dim = c(160,320)) |>
 render_title("Dragon (custom size)", title_offset=c(10,10), title_bar_color="black",
           title_size=20, title_color = "white") |>
 plot_image()
```

---

render_sprite_overlay *Add Sprite Overlay*

Description

Places an image overlay at a pixel location without resizing it to the full destination image. This is intended for sprite-like or billboard-like overlays such as labels, icons, glyphs, and screen-space annotations.

Usage

```
render_sprite_overlay(
  image,
  image_overlay = NULL,
  convert_overlay_colorspace = TRUE,
  alpha = NA,
  overlay_coords = c(1, 1),
  overlay_dims = NULL,
  overlay_anchor = "nw",
  overlay_just = NULL,
  hjust = NULL,
  vjust = NULL,
  preserve_channels = TRUE,
  filename = NULL,
  preview = FALSE
)
```

Arguments

image	3-layer RGB/4-layer RGBA array, rayimg class, or filename of an image.
image_overlay	Default NULL. 3-layer RGB/4-layer RGBA array, rayimg class, or filename of the sprite overlay.
convert_overlay_colorspace	Default TRUE. Whether to convert the overlay's colorspace to match the underlying image.
alpha	Default NA, using overlay's alpha channel. Otherwise, this sets the alpha transparency by multiplying the existing alpha channel by this value (between 0 and 1).
overlay_coords	Default c(1, 1). Pixel coordinate c(x, y) used to position the overlay. x increases from left to right and y from top to bottom, starting at 1.
overlay_dims	Default NULL. Dimensions for the overlay in pixels. If provided, the overlay is resized with render_resized() before compositing.
overlay_anchor	Default "nw". Which corner of the overlay is placed at overlay_coords when no numeric justification is supplied. Options: "nw", "ne", "sw", "se".
overlay_just	Default NULL. Optional numeric c(hjust, vjust) justification used to place the overlay relative to overlay_coords. If set, the overlay's left/top pixel is computed as round(x - hjust * overlay_width) and round(y - vjust * overlay_height).
hjust	Default NULL. Optional horizontal justification. Overrides the first value of overlay_just.
vjust	Default NULL. Optional vertical justification. Overrides the second value of overlay_just.
preserve_channels	Default TRUE. If TRUE, RGB input images return RGB output and RGBA input images return RGBA output.
filename	Default NULL. File to save the image to. If NULL and preview = FALSE, returns an image array.
preview	Default FALSE. If TRUE, it will display the image in addition to returning it.

Value

A rayimg array.

Examples

```
dragon_clipped = dragon
dragon_clipped[dragon_clipped > 1] = 1

# Render a transparent dragon emoji and center it on a pixel with hjust/vjust:
dragon_emoji = render_text_image(
 "\U0001F409",
 size = 80,
 background_alpha = 0,
 use_ragg = TRUE,
```

```
 trim = TRUE,
 trim_padding = 8
)
render_sprite_overlay(
 dragon_clipped,
 image_overlay = dragon_emoji,
 overlay_coords = c(ncol(dragon_clipped) / 2, nrow(dragon_clipped) / 2),
 hjust = 0.5,
 vjust = 0.5,
 preview = TRUE
)

# The same justification can be supplied as overlay_just = c(hjust, vjust):
render_sprite_overlay(
 dragon_clipped,
 image_overlay = dragon_emoji,
 overlay_coords = c(ncol(dragon_clipped), nrow(dragon_clipped)),
 overlay_just = c(1, 1),
 preview = TRUE
)
```

`render_stack` *Stack Images*

Description

Stacks a list of images vertically or horizontally into a single image.

Usage

```
render_stack(
  input_list,
  stack = c("v", "h", "vertical", "horizontal"),
  align = c("center", "start", "end", "left", "right", "top", "bottom"),
  background = c(0, 0, 0, 0),
  convert_colorspace = TRUE,
  filename = NULL,
  preview = FALSE
)
```

Arguments

`input_list`	List of 3-layer RGB/4-layer RGBA array, `rayimg` class, or image filenames.
`stack`	Default `"v"`. Stack direction: `"v"`/`"vertical"` or `"h"`/`"horizontal"`.
`align`	Default `"center"`. Alignment on the non-stacking axis. Options: `"start"`/`"left"`/`"top"`, `"center"`, `"end"`/`"right"`/`"bottom"`.

background Default `c(0,0,0,0)`. Background color for padding, as numeric `c(r,g,b[,a])` in 0..1 or a color string.

convert_colorspace
Default `TRUE`. Whether to convert all images to match the colorspace of the first image in the list.

filename Default `NULL`. File to save the image to. If `NULL` and `preview = FALSE`, returns an RGB array.

preview Default `FALSE`. If `TRUE`, it will display the image in addition to returning it.

Value

A `rayimg` RGBA array.

Examples

```
render_stack(list(dragon, 1 - dragon), stack = "h") |>
 plot_image()
render_stack(list(dragon, render_bw(dragon)), stack = "v") |>
 plot_image()
```

---

render_text_image *Generate Text Image*

Description

Generates an image which tightly fits text.

Usage

```
render_text_image(
  text,
  lineheight = 1,
  color = "black",
  size = 12,
  font = "sans",
  just = "left",
  background_color = "white",
  background_alpha = 1,
  use_ragg = TRUE,
  width = NA,
  height = NA,
  filename = NULL,
  check_text_width = TRUE,
  check_text_height = TRUE,
  trim = FALSE,
  trim_padding = 0,
  preview = FALSE
)
```

Arguments

text	Text to turn into an image.
lineheight	Default 1. Multiplier for the lineheight.
color	Default "black". String specifying the color of the text.
size	Default 12. Numeric value specifying the font size of the text.
font	Default "sans". String specifying the font family for the text. Common options include "sans", "mono", "serif", "Times", "Helvetica", etc.
just	Default "left". Horizontal alignment of the text: "left", "center", or "right".
background_color	Default "white". Color of the background.
background_alpha	Default 1. Transparency of the background. A value between 0 (fully transparent) and 1 (fully opaque).
use_ragg	Default TRUE. Whether to use the ragg package as the graphics device. Required for emojis.
width	Default NA. User-defined textbox width.
height	Default NA. User-defined textbox height.
filename	Default NULL. String specifying the file path to save the resulting image. If NULL and preview = FALSE, the function returns the processed RGB array.
check_text_width	Default TRUE. Whether to manually adjust the bounding box of the resulting image to ensure the string bbox is wide enough for the text. Not all systems provide accurate font sizes: this ensures the string is not cut off at the edges, at the cost of needing to repeatedly render the image internally until a suitable image is found.
check_text_height	Default FALSE. Whether to manually adjust the bounding box of the resulting image to ensure the string bbox is tall enough for the text. This will ensure a tight vertical bounding box on the text.
trim	Default FALSE. If TRUE, post-process the rendered image by cropping it to the tight bounding box of the text or emoji pixels. The bounding box is measured from a transparent text mask, so this can also trim images with opaque backgrounds.
trim_padding	Default 0. Padding in pixels to add after trimming. Use a scalar for equal padding on all sides, or c(top, right, bottom, left). Nonzero padding implies trim = TRUE.
preview	Default FALSE. Boolean indicating whether to display the image after processing. If TRUE, the image is displayed but not saved or returned.

Value

A 3-layer RGB array of the processed image if filename = NULL and preview = FALSE. Otherwise, writes the image to the specified file or displays it if preview = TRUE.

Examples

```
#Generate an image of some text
render_text_image("Some text", preview = TRUE)
#Change the font size
render_text_image("Some text", size = 100, preview = TRUE)
#Change the font color
render_text_image("Some text", size = 100, color="red",preview = TRUE)
#Change the background color and transparency
render_text_image("Some text", size = 50, color="purple",
                  background_color="purple", background_alpha = 0.5,
                  preview = TRUE)
# Plot an emoji with the agg device.
render_text_image("\U0001F600\U0001F680", size = 50, color = "purple", use_ragg = TRUE,
                  background_alpha = 0,
                  preview = TRUE)

# Plot an emoji with the agg device and adjust the height and width (which
# is on by default) to be a tight fit.
render_text_image("\U0001F600\U0001F680", size = 50, color = "purple", use_ragg = TRUE,
                  background_alpha = 0, check_text_width = TRUE,
                  check_text_height = TRUE,
                  preview = TRUE)

# Trim the image to the rendered glyphs and then add transparent padding.
render_text_image("\U0001F409", size = 80, use_ragg = TRUE,
                  background_alpha = 0, trim = TRUE, trim_padding = 8,
                  preview = TRUE)
```

---

render_title *Render a Title on an Image*

Description

Adds a title with optional styling and a title bar to an image. The image can be previewed or saved to a file. Supports both the `grid`-based method and (deprecated) `magick` package for rendering the title.

Usage

```
render_title(
  image,
  title_text = "",
  title_size = 30,
  title_offset = rep(title_size/2, 2),
  title_lineheight = 1,
  title_color = "black",
  title_font = "Arial",
  title_style = "plain",
```

```
  title_bar_color = NA,
  title_bar_alpha = 0.5,
  title_bar_width = NULL,
  title_position = NA,
  title_just = "left",
  use_magick = FALSE,
  filename = NULL,
  preview = FALSE
)
```

Arguments

image	3-layer RGB/4-layer RGBA array, rayimg class, or filename of an image.
title_text	Default "". Text string to be added as the title to the image.
title_size	Default 30. Numeric value specifying the font size of the title text.
title_offset	Default c(15,15). Numeric vector specifying the horizontal and vertical offset of the title text, relative to its anchor position.
title_lineheight	Default 1. Multiplier for the lineheight.
title_color	Default "black". String specifying the color of the title text.
title_font	Default "Arial". String specifying the font family for the title text. Common options include "sans", "mono", "serif", "Times", "Helvetica", etc.
title_style	Default "plain". String specifying the font style, such as "plain", "italic", or "bold".
title_bar_color	Default NULL. Color of the optional title bar. If NULL, no bar is added.
title_bar_alpha	Default 0.5. Transparency level of the title bar. A value between 0 (fully transparent) and 1 (fully opaque).
title_bar_width	Default NULL. Numeric value for the height of the title bar in pixels. If NULL, it is automatically calculated based on the text size and line breaks.
title_position	Default "northwest". String specifying the position of the title text. Only used when use_magick = TRUE. Common options include "northwest", "center", "south", etc.
title_just	Default "left". Horizontal alignment of the title text: "left", "center", or "right".
use_magick	Default FALSE. Boolean indicating whether to use the magick package for rendering titles. This option will be deprecated in future versions.
filename	Default NULL. String specifying the file path to save the resulting image. If NULL and preview = FALSE, the function returns the processed RGB array.
preview	Default FALSE. Boolean indicating whether to display the image after processing. If TRUE, the image is displayed but not saved or returned.

Value

A 3-layer RGB array of the processed image if `filename = NULL` and `preview = FALSE`. Otherwise, writes the image to the specified file or displays it if `preview = TRUE`.

Note

The `use_magick` parameter and all functionality tied to the `magick` package are planned for deprecation. It is recommended to use the `grid`-based method for future compatibility.

Examples

```
#Plot the dragon
render_title(dragon, preview = TRUE, title_text = "Dragon", title_size=20)
#That's hard to see--let's add a title bar:
render_title(dragon, preview = TRUE, title_text = "Dragon", title_size=20,
         title_bar_color="white")
#Change the width of the bar:
render_title(dragon, preview = TRUE, title_text = "Dragon", title_size=20,
         title_bar_color="white", title_offset = c(8,8))
#The width of the bar will also automatically adjust for newlines:
render_title(dragon, preview = TRUE, title_text = "Dragon\n(Blue)", title_size=20,
         title_bar_color="white")
#Change the color and title color:
render_title(dragon, preview = TRUE, title_text = "Dragon", title_size=20,
         title_bar_color="red", title_color = "white")
#Change the transparency:
render_title(dragon, preview = TRUE, title_text = "Dragon",
            title_size=20, title_bar_alpha = 0.8,
            title_bar_color="red", title_color = "white")
#Read directly from a file
temp_image = tempfile(fileext = ".png")
ray_write_image(dragon, temp_image)
render_title(temp_image, preview = TRUE, title_text = "Dragon",
             title_size=20, title_bar_alpha = 0.8,
             title_bar_color="red", title_color = "white")
```

`render_tonemap` *Render Tonemap*

Description

Reinhard / Uncharted(Hable) / HBD operators in linear

Usage

```
render_tonemap(
  image,
  method = c("raw", "reinhard", "uncharted", "hbd"),
```

```
  exposure_bias = 2,
  W = 11.2,
  filename = NULL,
  preview = FALSE
)
```

Arguments

image	Default NULL. 3/4-channel array, rayimg, or filename.
method	Default raw. One of "raw", "reinhard", "uncharted", "hbd". Chooses the global tone-mapping curve. "raw" leaves the image unchanged. "reinhard" applies a classic shoulder that gently compresses highlights. "uncharted" (Hable filmic) gives a film-style toe/shoulder and keeps mid-tones contrasty. "hbd" is a fast filmic-like curve with a soft toe/shoulder. All methods operate on **linear RGB** and return **display-referred linear** (no OETF).
exposure_bias	Default 2. Only for "uncharted". Scalar applied to the linear RGB channels **before** the Uncharted/Hable curve (i.e., a pre-curve exposure gain). Values >1 brighten; <1 darken. exposure_bias=2 is about +1 stop; 0.5 is about -1 stop.
W	Default 11.2. White (linear scene value) that maps to 1.0 **after** the Uncharted/Hable/Reinhard curve. Acts like a "white point" for the shoulder: larger values preserve more highlight detail (later roll-off), smaller values roll off earlier and harder.
filename	Default NULL. Output path.
preview	Default FALSE. Show result.

Value

A rayimg RGBA array.

Examples

```
# Plot unchanged image
render_tonemap(dragon, preview = TRUE)
# Plot Reinhard tonemapped image
render_tonemap(dragon, method = "reinhard", preview = TRUE)
# Plot Uncharted/Hable tonemapped image
render_tonemap(dragon, method = "uncharted",preview = TRUE)
# Plot Uncharted/Hable tonemapped image, white point adjusted
render_tonemap(dragon, method = "uncharted", W=11.4, preview = TRUE)
# Plot Uncharted/Hable tonemapped image, exposure adjusted
render_tonemap(dragon, method = "uncharted", exposure_bias = 1,preview = TRUE)
# Plot Hejl-Burgess-Dawson tonemapped image, exposure adjusted
render_tonemap(dragon, method = "hbd",preview = TRUE)
```

render_to_display *Render To Display*

Description

Convert an image from its working space to a display space, and optionally apply sRGB OETF for viewing.

Usage

```
render_to_display(image, display = CS_SRGB, encode = TRUE, adapt_white = TRUE)
```

Arguments

image	A rayimg, array, or filename.
display	Default CS_SRGB. Target display RGB space.
encode	Default TRUE. If TRUE, apply sRGB OETF (only valid when display is sRGB-like).
adapt_white	Default TRUE. Whether to perform CAT from working white to display white.

Value

A rayimg when encode=FALSE, or a rayimg with sRGB-encoded RGB when encode=TRUE.

render_vibrance *Render Vibrance*

Description

Adjusts image vibrance (adaptive saturation), similar in spirit to the iPhone "Vibrance" control: it boosts muted colors more than already-saturated colors, and reduces saturation more strongly in already-saturated regions when negative. Works on HDR arrays as well (values can exceed 1).

Usage

```
render_vibrance(
  image,
  vibrance = 0,
  protect_luminance = 0.25,
  filename = NULL,
  preview = FALSE
)
```

Arguments

image	Image filename, 3-layer RGB array, 4-layer RGBA array, or matrix. If a filename, it will be read via ray_read_image().
vibrance	Default 0. Numeric scalar in -1..1 (values are clamped). Positive values increase vibrance primarily in low-saturation regions; negative values decrease vibrance primarily in high-saturation regions.
protect_luminance	Default 0.25. Numeric scalar in 0..1. Reduces the vibrance effect in deep shadows and bright highlights to avoid harsh color shifts. Set to 0 to disable.
filename	Default NULL. If not NULL, the processed image will be written to this file.
preview	Default FALSE. Whether to plot the processed image.

Value

4-layer RGBA array (or matrix if passed a matrix and ray_read_image() is not used).

Examples

```
dragon |>
  render_vibrance(vibrance = 0.4, preview = TRUE)
```

---

render_vignette *Add Vignette Effect*

Description

Takes an RGB array/filename and adds a camera vignette effect.

Usage

```
render_vignette(
  image,
  vignette = 0.5,
  color = "#000000",
  radius = 1.1,
  filename = NULL,
  preview = FALSE
)
```

Arguments

image	3-layer RGB/4-layer RGBA array, rayimg class, or filename of an image.
vignette	Default 1. A camera vignetting effect will be added to the image. 1 is the darkest vignetting, while 0 is no vignetting. If vignette is a length-2 vector, the second entry will control the blurriness of the vignette effect (1 is the default, e.g. 2 would double the blurriness but would take much longer to compute).

color	Default "#000000" (black). Color of the vignette.
radius	Default 1.1. Multiplier for the size of the vignette. If 1, the vignette touches the edge of the image.
filename	Default NULL. Filename which to save the image. If NULL and preview = FALSE, returns an RGB array.
preview	Default FALSE. If TRUE, it will display the image in addition to returning it.

Value

A rayimg RGBA array.

Examples

```
#Plot the dragon
plot_image(dragon)
#Add a vignette effect:
render_vignette(dragon, preview = TRUE, vignette = 1, radius=1.1)
#Lighten the vignette effect:
render_vignette(dragon, preview = TRUE, vignette = 0.5)
#Change the radius:
render_vignette(dragon, preview = TRUE, vignette = 1, radius=1.5)
render_vignette(dragon, preview = TRUE, vignette = 0.1, radius=0.8)
#Change the color:
render_vignette(dragon, preview = TRUE, vignette = 1, color="white")
#Increase the width of the blur by 50%:
render_vignette(dragon, preview = TRUE, vignette = c(1,1.5))
```

---

render_white_balance *Render White Balance (Bradford CAT)*

Description

Chromatic-adapt an image from reference_white to target_white **in its working RGB space**.

When bake = FALSE (the default), this should not result in any visible changes to the image when plotted in rayimage via plot_image(), as that function then corrects the white balance for display. Set

Usage

```
render_white_balance(
  image,
  reference_white = NA,
  target_white = "D60",
  bake = FALSE,
  filename = NULL,
  preview = FALSE
)
```

Arguments

image	Default NULL. 3-layer RGB/4-layer RGBA array, rayimg, or filename.
reference_white	Default NA. Source white (XYZ Y=1 or named); when NA, uses attr(img, "white_current").
target_white	Default "D60". Target white (XYZ Y=1 or named).
bake	Default FALSE. If TRUE, this will bake the color change into the image, and return the old tag. This can be used for stylistic adjustments.
filename	Default NULL. Output path.
preview	Default FALSE. If TRUE, display the image.

Examples

```
# Not specifying a target white balance returns an unchanged image
render_white_balance(dragon, preview = TRUE)

# Default white "D60", converting to D50 produces a warmer image
render_white_balance(dragon, preview = TRUE, target_white = "D50", bake = TRUE)

# Default white "D60", converting to "D75" produces a cooler image
render_white_balance(dragon, preview = TRUE, target_white = "D75", bake = TRUE)

# Set reference white to colder "D75", converting to "D50" produces a very warm image
render_white_balance(dragon, preview = TRUE,
                     reference_white= "D75", target_white = "D50", bake = TRUE)

# With a real, non-rendered image:
render_white_balance(sunset_image, preview=TRUE)

# Cooler
render_white_balance(sunset_image, preview=TRUE, target_white = "D75", bake = TRUE)

# Warmer
render_white_balance(sunset_image, preview=TRUE, target_white = "D50", bake = TRUE)
```

---

sunset_image *Sunset Image*

Description

Sunset Image

Usage

```
sunset_image
```

Format

An RGBA 4-layer LDR rayimg array from a JPEG image with 400 rows and 400 columns.

Author(s)

Tyler Morgan-Wall

Index