Package 'rayimage'

July 14, 2024

Type Package

Title Image Processing for Simulated Cameras

Version 0.11.0

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. Includes functions to perform 2D convolutions, reorient and resize images/matrices, add image overlays, generate camera vignette effects, and add titles to images.

License GPL-3

LazyData true

Depends R (>= 4.1.0)

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

Suggests magick, testthat (>= 3.0.0)

LinkingTo Rcpp, RcppArmadillo, progress

RoxygenNote 7.3.1

Encoding UTF-8

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

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

Config/testthat/edition 3

NeedsCompilation yes

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

Repository CRAN

Date/Publication 2024-07-13 22:00:01 UTC

Contents

`add_image_overlay` *Add Overlay*

Description

Takes an RGB array/filename and adds an image overlay.

Usage

```
add_image_overlay(
  image,
  image_overlay = NULL,
  rescale_original = FALSE,
  alpha = NULL,
  filename = NULL,
  preview = FALSE
)
```

Arguments

image	Image filename or 3-layer RGB array.
image_overlay	Default NULL. Either a string indicating the location of a png image to overlay over the image (transparency included), or a 4-layer RGBA array. This image will be resized to the dimension of the image if it does not match exactly.
rescale_original	Default FALSE. If TRUE, function will resize the original image to match the overlay.
alpha	Default NULL, 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

3-layer RGB array of the processed image.

Examples

```
if(run_documentation()){
#Plot the dragon
plot_image(dragon)
}
if(run_documentation()){
#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
add_image_overlay(dragon_clipped, image_overlay = rgba_array,
                  alpha=0.5, preview = TRUE)
}
```

---

add_title *Add Title*

Description

Takes an RGB array/filename and adds a title with an optional titlebar.

Usage

```
add_title(
  image,
  title_text = "",
  title_offset = c(15, 15),
  title_color = "black",
  title_size = 30,
  title_font = "sans",
  title_style = "normal",
  title_bar_color = NULL,
  title_bar_alpha = 0.5,
  title_bar_width = NULL,
  title_position = "northwest",
  filename = NULL,
  preview = FALSE
)
```

Arguments

image	Image filename or 3-layer RGB array.
title_text	Default NULL. Text. Adds a title to the image, using magick::image_annotate().
title_offset	Default c(15,15). Distance from the top-left (default, gravity direction in image_annotate) corner to offset the title.
title_color	Default black. Font color.
title_size	Default 30. Font size in pixels.
title_font	Default sans. String with font family such as "sans", "mono", "serif", "Times", "Helvetica", "Trebuchet", "Georgia", "Palatino" or "Comic Sans".
title_style	Default normal. Font style (e.g. italic).
title_bar_color	Default NULL. If a color, this will create a colored bar under the title.
title_bar_alpha	Default 0.5. Transparency of the title bar.
title_bar_width	Default NULL, automaticly calculated from the size of the text and the number of line breaks. Width of the title bar in pixels.
title_position	Default northwest. Position of the title.
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

3-layer RGB array of the processed image.

Examples

```
if(run_documentation()){
#Plot the dragon
add_title(dragon, preview = TRUE, title_text = "Dragon", title_size=20)
}
if(run_documentation()){
#That's hard to see--let's add a title bar:
add_title(dragon, preview = TRUE, title_text = "Dragon", title_size=20,
         title_bar_color="white")
}
if(run_documentation()){
#Change the width of the bar:
add_title(dragon, preview = TRUE, title_text = "Dragon", title_size=20,
         title_bar_color="white", title_offset = c(8,8))
}
if(run_documentation()){
#The width of the bar will also automatically adjust for newlines:
add_title(dragon, preview = TRUE, title_text = "Dragon\n(Blue)", title_size=20,
         title_bar_color="white")
}
if(run_documentation()){
#Change the color and title color:
add_title(dragon, preview = TRUE, title_text = "Dragon", title_size=20,
         title_bar_color="red", title_color = "white")
}
if(run_documentation()){
#Change the transparency:
add_title(dragon, preview = TRUE, title_text = "Dragon", title_size=20, title_bar_alpha = 0.8,
         title_bar_color="red", title_color = "white")
}
```

---

add_vignette *Add Vignette Effect*

Description

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

Usage

```
add_vignette(
  image,
  vignette = 0.5,
  color = "#000000",
  radius = 1.3,
  filename = NULL,
  preview = FALSE
)
```

Arguments

image	Image filename or 3-layer RGB array.
vignette	Default 0.5. 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 1.3. 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

3-layer RGB array of the processed image.

Examples

```
if(run_documentation()){
#Plot the dragon
plot_image(dragon)
}
if(run_documentation()){
#Add a vignette effect:
add_vignette(dragon, preview = TRUE, vignette = 0.5)
}
if(run_documentation()){
#Darken the vignette effect:
add_vignette(dragon, preview = TRUE, vignette = 1)
}
if(run_documentation()){
#Change the radius:
add_vignette(dragon, preview = TRUE, vignette = 1, radius=1.5)
add_vignette(dragon, preview = TRUE, vignette = 1, radius=0.5)
}
if(run_documentation()){
#Change the color:
add_vignette(dragon, preview = TRUE, vignette = 1, color="white")
}
if(run_documentation()){
#Increase the width of the blur by 50%:
add_vignette(dragon, preview = TRUE, vignette = c(1,1.5))
}
```

dragon *Dragon Image*

Description

Dragon Image

Usage

```
dragon
```

Format

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

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

```
if(run_documentation()){
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

```
if(run_documentation()){
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

```
if(run_documentation()){
image(generate_2d_gaussian(1,1,31), asp=1)
}
```

---

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	Image filename, a matrix, or a 3-layer RGB array.
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,
  rotate = 0,
  draw_grid = FALSE,
  asp = 1,
  new_page = TRUE,
  return_grob = FALSE
)
```

Arguments

image	Image array or filename of an image to be plotted.
rotate	Default 0. Rotates the output. Possible values: 0, 90, 180, 270.
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).
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.

Examples

```
#if(interactive()){
#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)
#end}
```

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)
```

Arguments

input_list	List of array (or matrix) image inputs.
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).

Examples

```
if(run_documentation()){
#Plot the dragon array
plot_image_grid(list(dragon, 1-dragon), dim = c(1,2))
}
if(run_documentation()){
plot_image_grid(list(dragon, 1-dragon), dim = c(2,1))
}
if(run_documentation()){
plot_image_grid(list(dragon, NULL, 1-dragon), dim = c(2,2), asp = c(2,1,1/2))
}
if(run_documentation()){
plot_image_grid(list(dragon, NULL, NULL, dragon), dim = c(2,2), asp = c(2,1,1,1/2))
}
if(run_documentation()){
#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))
}
```

`ray_read_image` *Read Image*

Description

Takes an RGB array/filename and adds an image overlay.

Usage

```
ray_read_image(image, convert_to_array = TRUE, preview = FALSE, ...)
```

Arguments

`image`	Image filename or 3-layer RGB array.
`convert_to_array`	Default `TRUE`. 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.
`...`	Arguments to pass to either `jpeg::readJPEG`, `png::readPNG`, or `tiff::readTIFF`.

Value

3-layer RGB array of the processed image.

Examples

```
if(run_documentation()){
#Write as a png
tmparr = tempfile(fileext=".png")
ray_read_image(dragon) |>
 ray_write_image(tmparr)
ray_read_image(tmparr) |>
 plot_image()
}
if(run_documentation()){
#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()
}
if(run_documentation()){
#Write as a tiff
tmparr = tempfile(fileext=".tiff")
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 = TRUE, ...)
```

Arguments

image	Image filename or 3-layer RGB array.
filename	File to write to, with filetype determined by extension. Filetype can be PNG, JPEG, or TIFF.
clamp	Default TRUE. Whether to clamp the image to 0-1. If the file extension is PNG of JPEG, this is forced to TRUE.
...	Arguments to pass to either jpeg::writeJPEG, png::writePNG, or tiff::writeTIFF.

Value

3-layer RGB array of the processed image.

Examples

```
if(run_documentation()){
#Write as a png
tmparr = tempfile(fileext=".png")
ray_read_image(dragon) |>
 ray_write_image(tmparr)
ray_read_image(tmparr) |>
 plot_image()
}
if(run_documentation()){
#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()
}
if(run_documentation()){
#Write as a tiff
tmparr = tempfile(fileext=".tiff")
ray_read_image(dragon) |>
 ray_write_image(tmparr)
ray_read_image(tmparr) |>
```

```
  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,
  gamma_correction = TRUE,
  progress = interactive(),
  ...
)
```

Arguments

image	Image filename or 3-layer RGB array.
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`.
gamma_correction	Default `TRUE`. Controls gamma correction when adding colors. Default exponent of 2.2.
progress	Default `TRUE`. Whether to display a progress bar.
...	Additional arguments to pass to `plot_image()` if `preview = TRUE`.

Value

3-layer RGB array of the processed image.

Examples

```
if(run_documentation()){
#Plot the dragon
plot_image(dragon)
}
if(run_documentation()){
#Plot the depth map
plot_image(dragondepth/1500)
}
if(run_documentation()){
#Preview the focal plane:
render_bokeh(dragon, dragondepth, focus=950, preview_focus = TRUE)
}
if(run_documentation()){
#Change the focal length:
render_bokeh(dragon, dragondepth, focus=950, focallength=300)
}
if(run_documentation()){
#Add chromatic aberration:
render_bokeh(dragon, dragondepth, focus=950, focallength=300, aberration = 0.5)
}
if(run_documentation()){
#Change the focal distance:
render_bokeh(dragon, dragondepth, focus=600, focallength=300)
render_bokeh(dragon, dragondepth, focus=1300, focallength=300)
}
if(run_documentation()){
#Change the bokeh shape to a hexagon:
render_bokeh(dragon, dragondepth, bokehshape = "hex",
             focallength=300, focus=600)
}
if(run_documentation()){
#Change the bokeh intensity:
```

```
render_bokeh(dragon, dragondepth,
            focallength=400, focus=900, bokehintensity = 1)
render_bokeh(dragon, dragondepth,
            focallength=400, focus=900, bokehintensity = 3)
}
if(run_documentation()){
#Rotate the hexagonal shape:
render_bokeh(dragon, dragondepth, bokehshape = "hex", rotation=15,
            focallength=300, focus=600)
}
```

`render_boolean_distance` *Render Boolean Distance*

Description

Takes an matrix (or and returns the nearest distance to each TRUE.

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

```
if(run_documentation()){
#Measure distance to
plot_image(render_boolean_distance(t(volcano) > 150))
plot_image(render_boolean_distance(t(volcano) < 150))
}
if(run_documentation()){
#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	Image filename, 3-layer RGB array, or matrix.
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

3-layer RGB resized array or matrix.

Examples

```
if(run_documentation()){
#Plot the image with a title
dragon |>
 add_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

Usage

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

Arguments

image	Image filename or 3-layer RGB array.
min_value	Default 0. Minimum value to clamp the image to.
max_value	Default 1. Maximum value to clamp the image to.
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

3-layer RGB array of the processed image.

Examples

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

render_convolution *Render Convolution*

Description

Takes an image and applys 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,
  filename = NULL,
  preview = FALSE,
  gamma_correction = FALSE,
  progress = FALSE
)
```

Arguments

image	Image filename or 3-layer RGB array.
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 he minimum value (for any color channel) for a pixel to have the convolution performed.
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.
gamma_correction	Default TRUE. Controls gamma correction when adding colors. Default exponent of 2.2.
progress	Default TRUE. Whether to display a progress bar.

Value

3-layer RGB array of the processed image.

Examples

```
if(run_documentation()){
#Perform a convolution with the default gaussian kernel
plot_image(dragon)
}
```

```
if(run_documentation()){
#Perform a convolution with the default gaussian kernel
render_convolution(dragon, preview = TRUE)
}
if(run_documentation()){
#Increase the width of the kernel
render_convolution(dragon, kernel = 2, kernel_dim=21,kernel_extent=6, preview = TRUE)
}
if(run_documentation()){
#Perform edge detection using a 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)
}
if(run_documentation()){
#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)
}

if(run_documentation()){
#Only perform the convolution on bright pixels (bloom)
render_convolution(dragon, kernel = 5, kernel_dim=24, kernel_extent=24,
                   min_value=1, preview = TRUE)
}
if(run_documentation()){
#Use a built-in kernel:
render_convolution(dragon, kernel = generate_2d_exponential(falloff=2, dim=31, width=21),
                   preview = TRUE)
}
if(run_documentation()){
#We can also apply this function to matrices:
volcano |> image()
volcano |>
 render_convolution(kernel=generate_2d_gaussian(sd=1,dim=31)) |>
 image()
}
if(run_documentation()){
#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 applys 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,
  filename = NULL,
  preview = FALSE,
  gamma_correction = FALSE
)
```

Arguments

image	Image filename or 3-layer RGB array.
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.
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.
gamma_correction	Default FALSE. Controls gamma correction when adding colors. Default exponent of 2.2.

Value

3-layer RGB array of the processed image.

Examples

```
if(run_documentation()){
#Perform a convolution with the default gaussian kernel
plot_image(dragon)
}
if(run_documentation()){
#Perform a convolution with the default gaussian kernel
render_convolution_fft(dragon, kernel=0.1,preview = TRUE)
}
if(run_documentation()){
#Increase the width of the kernel
render_convolution_fft(dragon, kernel = 2, kernel_dim=21,kernel_extent=6, preview = TRUE)
}
if(run_documentation()){
#Use a built-in kernel:
render_convolution_fft(dragon, kernel = generate_2d_exponential(falloff=2, dim=31, width=21),
                       preview = TRUE)
}
if(run_documentation()){
#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)
}
if(run_documentation()){
#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)
}
if(run_documentation()){
#We can also apply this function to matrices:
volcano |> image()
volcano |>
 render_convolution_fft(kernel=generate_2d_gaussian(sd=1,dim=31)) |>
 image()
}
if(run_documentation()){
# 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)
}
if(run_documentation()){
#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_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`	Image filename, 3-layer RGB array, or matrix.
`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

```
if(run_documentation()){
#Original orientation
plot_image(dragon)
}
if(run_documentation()){
#Flip the dragon image horizontally
dragon |>
 render_reorient(flipx = TRUE) |>
 plot_image()
}
```

```
if(run_documentation()){
#Flip the dragon image vertically
dragon |>
 render_reorient(flipy = TRUE) |>
 plot_image()
}
if(run_documentation()){
#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`	Image filename, 3-layer RGB array, or matrix.
`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

3-layer RGB resized array or matrix.

Examples

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

---

run_documentation *Run Documentation*

Description

This function determines if the examples are being run in pkgdown. It is not meant to be called by the user.

Usage

```
run_documentation()
```

Value

Boolean value.

Examples

```
# See if the documentation should be run.
run_documentation()
```

Index