Package ‘rayshader’

April 28, 2021

Type Package

Title Create Maps and Visualize Data in 2D and 3D

Version 0.24.10

Date 2021-04-25

Author Tyler Morgan-Wall

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

Description Uses a combination of raytracing and multiple hill shading methods to produce 2D and 3D data visualizations and maps. Includes water detection and layering functions, programmable color palette generation, several built-in textures for hill shading, 2D and 3D plotting options, a built-in path tracer, 'Wavefront' OBJ file export, and the ability to save 3D visualizations to a 3D printable format.

License GPL-3

LazyData true

Depends R (>= 3.0.2)

Imports doParallel, foreach, Rcpp, progress, raster, scales, png, magrittr, rgl, grDevices, grid, utils, methods, terrainmeshr, rayimage

Suggests reshape2, viridis, av, magick, ggplot2, sf, rayrender, isoband, maptools, geosphere

LinkingTo Rcpp, progress, RcppArmadillo

RoxygenNote 7.1.1


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

NeedsCompilation yes

Repository CRAN

Date/Publication 2021-04-28 15:30:02 UTC
## R topics documented:

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>add_overlay</td>
<td>3</td>
</tr>
<tr>
<td>add_shadow</td>
<td>4</td>
</tr>
<tr>
<td>add_water</td>
<td>5</td>
</tr>
<tr>
<td>ambient_shade</td>
<td>6</td>
</tr>
<tr>
<td>calculate_normal</td>
<td>8</td>
</tr>
<tr>
<td>create_texture</td>
<td>9</td>
</tr>
<tr>
<td>detect_water</td>
<td>10</td>
</tr>
<tr>
<td>generate_altitude_overlay</td>
<td>11</td>
</tr>
<tr>
<td>generate_compass_overlay</td>
<td>12</td>
</tr>
<tr>
<td>generate_contour_overlay</td>
<td>15</td>
</tr>
<tr>
<td>generate_label_overlay</td>
<td>17</td>
</tr>
<tr>
<td>generate_line_overlay</td>
<td>21</td>
</tr>
<tr>
<td>generate_point_overlay</td>
<td>23</td>
</tr>
<tr>
<td>generate_polygon_overlay</td>
<td>24</td>
</tr>
<tr>
<td>generate_scalebar_overlay</td>
<td>26</td>
</tr>
<tr>
<td>generate_waterline_overlay</td>
<td>31</td>
</tr>
<tr>
<td>height_shade</td>
<td>34</td>
</tr>
<tr>
<td>lamb_shade</td>
<td>35</td>
</tr>
<tr>
<td>montereybay</td>
<td>36</td>
</tr>
<tr>
<td>monterey_counties_sf</td>
<td>37</td>
</tr>
<tr>
<td>monterey_roads_sf</td>
<td>38</td>
</tr>
<tr>
<td>plot_3d</td>
<td>38</td>
</tr>
<tr>
<td>plot_gg</td>
<td>42</td>
</tr>
<tr>
<td>plot_map</td>
<td>46</td>
</tr>
<tr>
<td>raster_to_matrix</td>
<td>48</td>
</tr>
<tr>
<td>ray_shade</td>
<td>49</td>
</tr>
<tr>
<td>reduce_matrix_size</td>
<td>51</td>
</tr>
<tr>
<td>render_camera</td>
<td>51</td>
</tr>
<tr>
<td>render_compass</td>
<td>53</td>
</tr>
<tr>
<td>render_depth</td>
<td>56</td>
</tr>
<tr>
<td>render_highquality</td>
<td>59</td>
</tr>
<tr>
<td>render_label</td>
<td>63</td>
</tr>
<tr>
<td>render_movie</td>
<td>66</td>
</tr>
<tr>
<td>render_path</td>
<td>69</td>
</tr>
<tr>
<td>render_points</td>
<td>71</td>
</tr>
<tr>
<td>render_polygons</td>
<td>74</td>
</tr>
<tr>
<td>render_scalebar</td>
<td>76</td>
</tr>
<tr>
<td>render_snapshot</td>
<td>78</td>
</tr>
<tr>
<td>render_water</td>
<td>81</td>
</tr>
<tr>
<td>resize_matrix</td>
<td>82</td>
</tr>
<tr>
<td>save_3dprint</td>
<td>83</td>
</tr>
<tr>
<td>save_obj</td>
<td>85</td>
</tr>
<tr>
<td>save_png</td>
<td>86</td>
</tr>
<tr>
<td>sphere_shade</td>
<td>88</td>
</tr>
<tr>
<td>texture_shade</td>
<td>89</td>
</tr>
<tr>
<td>%&gt;%</td>
<td>91</td>
</tr>
</tbody>
</table>
### add_overlay

**Index**

<table>
<thead>
<tr>
<th>add_overlay</th>
<th>Add Overlay</th>
</tr>
</thead>
</table>

**Description**

Overlays an image (with a transparency layer) on the current map.

**Usage**

```r
add_overlay(
  hillshade,
  overlay,
  alphacolor = NULL,
  alphamethod = "max",
  alphalayer = 1,
  rescale_original = FALSE
)
```

**Arguments**

- **hillshade**: A three-dimensional RGB array or 2D matrix of shadow intensities.
- **overlay**: A three or four dimensional RGB array, where the 4th dimension represents the alpha (transparency) channel. If the array is 3D, ‘alphacolor’ should also be passed to indicate transparent regions.
- **alphacolor**: Default ‘NULL’. If ‘overlay’ is a 3-layer array, this argument tells which color is interpreted as completely transparent.
- **alphamethod**: Default ‘max’. Method for dealing with pre-existing transparency with ‘layeralpha’. If ‘max’, converts all alpha levels higher than ‘layeralpha’ to the value set in ‘layeralpha’. Otherwise, this just sets all transparency to ‘layeralpha’.
- **alphalayer**: Default ‘1’. Defines minimum transparency of layer. If transparency already exists in ‘overlay’, the way ‘add_overlay’ combines the two is determined in argument ‘alphamethod’.
- **rescale_original**: Default ‘FALSE’. If ‘TRUE’, ‘hillshade’ will be scaled to match the dimensions of ‘overlay’ (instead of the other way around).

**Value**

Hillshade with overlay.
### add_shadow

#### Description

Multiplies a texture array or shadow map by a shadow map.

#### Usage

```
add_shadow(hillshade, shadowmap, max_darken = 0.7, rescale_original = FALSE)
```

#### Arguments

- **hillshade**: A three-dimensional RGB array or 2D matrix of shadow intensities.
- **shadowmap**: A matrix that indicates the intensity of the shadow at that point. 0 is full darkness, 1 is full light.
- **max_darken**: Default `0.7`. The lower limit for how much the image will be darkened. 0 is completely black, 1 means the shadow map will have no effect.
- **rescale_original**: Default `FALSE`. If `TRUE`, `hillshade` will be scaled to match the dimensions of `shadowmap` (instead of the other way around).

#### Value

Shaded texture map.
Examples

#First we plot the sphere_shade() hillshade of `montereybay` with no shadows

```r
montereybay %>%
sphere_shade(colorintensity=0.5) %>%
plot_map()
```

#Raytrace the `montereybay` elevation map and add that shadow to the output of sphere_shade()

```r
montereybay %>%
sphere_shade(colorintensity=0.5) %>%
add_shadow(ray_shade(montereybay,sunaltitude=20,zscale=50),max_darken=0.3) %>%
plot_map()
```

#Increase the intensity of the shadow map with the max_darken argument.

```r
montereybay %>%
sphere_shade(colorintensity=0.5) %>%
add_shadow(ray_shade(montereybay,sunaltitude=20,zscale=50),max_darken=0.1) %>%
plot_map()
```

#Decrease the intensity of the shadow map.

```r
montereybay %>%
sphere_shade(colorintensity=0.5) %>%
add_shadow(ray_shade(montereybay,sunaltitude=20,zscale=50),max_darken=0.7) %>%
plot_map()
```

---

**add_water**

*Add Water*

**Description**

Adds a layer of water to a map.

**Usage**

```
add_water(hillshade, watermap, color = "imhof1")
```

**Arguments**

- **hillshade**  
  A three-dimensional RGB array.

- **watermap**  
  Matrix indicating whether water was detected at that point. 1 indicates water, 0 indicates no water.
ambient_shade

Calculate Ambient Occlusion Map

Description

Calculates Ambient Occlusion Shadow Map

Usage

ambient_shade(
  heightmap,
  anglebreaks = 90 * cospi(seq(5, 85, by = 5)/180),
  sunbreaks = 24,
  maxsearch = 30,
  multicore = FALSE,
  zscale = 1,
  cache_mask = NULL,
  shadow_cache = NULL,
  progbar = interactive(),
  ...
)

Examples

# Here we even out a portion of the volcano dataset to simulate water:
island_volcano = volcano
island_volcano[island_volcano < mean(island_volcano)] = mean(island_volcano)

# Setting a minimum area avoids classifying small flat areas as water:
  sphere_shade(texture="imhof3") %>%
  add_water(detect_water(island_volcano, min_area = 400),color="imhof3") %>%
  plot_map()

# We'll do the same thing with the Monterey Bay dataset to fill in the ocean:

montbay_water = montereybay
montbay_water[montbay_water < 0] = 0

montereybay %>%
  sphere_shade(texture="imhof4") %>%
  add_water(detect_water(montbay_water),color="imhof4") %>%
  plot_map()
ambient_shade

Arguments

heightmap A two-dimensional matrix, where each entry in the matrix is the elevation at that point. All points are assumed to be evenly spaced.

anglebreaks Default ‘90*cospi(seq(5, 85,by =5)/180)’. The angle(s), in degrees, as measured from the horizon from which the light originates.

sunbreaks Default ‘24’. Number of rays to be sent out in a circle, evenly spaced, around the point being tested.

maxsearch Default ‘30’. The maximum horizontal distance that the system should propagate rays to check for surface intersections.

multicore Default FALSE. If TRUE, multiple cores will be used to compute the shadow matrix. By default, this uses all cores available, unless the user has set ‘options("cores")’ in which the multicore option will only use that many cores.

zscale Default 1. The ratio between the x and y spacing (which are assumed to be equal) and the z axis.

cache_mask Default ‘NULL’. A matrix of 1 and 0s, indicating which points on which the raytracer will operate.

shadow_cache Default ‘NULL’. The shadow matrix to be updated at the points defined by the argument ‘cache_mask’.

probar Default ‘TRUE’ if interactive, ‘FALSE’ otherwise. If ‘FALSE’, turns off progress bar.

Additional arguments to pass to the ‘makeCluster’ function when ‘multicore=TRUE’.

Value

Shaded texture map.

Examples

#Here we produce a ambient occlusion map of the `montereybay` elevation map.
## Not run:
plot_map(ambient_shade(heightmap = montereybay))

## End(Not run)

#We can increase the distance to look for surface intersections `maxsearch`
#and the density of rays sent out around the point `sunbreaks`.
## Not run:
plot_map(ambient_shade(montereybay, sunbreaks = 24,maxsearch = 100, multicore=TRUE))

## End(Not run)

#Create the Red Relief Image Map (RRIM) technique using a custom texture and ambient_shade(),
#with an addition lambertian layer added with lamb_shade() to improve topographic clarity.
## Not run:
bigmb = resize_matrix(montereybay, scale=2, method="cubic")
bigmb >%
sphere_shade(zscale=3, texture = create_texture("red","red","red","red","white")) >%
add_shadow(ambient_shade(bigmb, maxsearch = 100, multicore = TRUE,zscale=1),0) >%
## calculate_normal

### Description

Calculates the normal unit vector for every point on the grid.

### Usage

```r
calculate_normal(heightmap, zscale = 1, progbar = FALSE)
```

### Arguments

- **heightmap**: A two-dimensional matrix, where each entry in the matrix is the elevation at that point. All points are assumed to be evenly spaced.
- **zscale**: Default 1.
- **progbar**: Default `FALSE`. If `TRUE`, turns on progress bar.

### Value

Matrix of light intensities at each point.

### Examples

```r
# Here we produce a light intensity map of the `volcano` elevation map.
# Cache the normal vectors of the volcano dataset
volcanocache = calculate_normal(volcano)

# Use the cached vectors to speed up calculation of `sphere_shade()` on a map.
sphere_shade(volcano, normalvectors = volcanocache) %>%
plot_map()
```
### Description
Create a texture map based on 5 user-supplied colors.

### Usage
```r
create_texture(
    lightcolor,
    shadowcolor,
    leftcolor,
    rightcolor,
    centercolor,
    cornercolors = NULL
)
```

### Arguments
- **lightcolor**: The main highlight color. Corresponds to the top center of the texture map.
- **shadowcolor**: The main shadow color. Corresponds to the bottom center of the texture map. This color represents slopes directed directly opposite to the main highlight color.
- **leftcolor**: The left fill color. Corresponds to the left center of the texture map. This color represents slopes directed 90 degrees to the left of the main highlight color.
- **rightcolor**: The right fill color. Corresponds to the right center of the texture map. This color represents slopes directed 90 degrees to the right of the main highlight color.
- **centercolor**: The center color. Corresponds to the center of the texture map. This color represents flat areas.
- **cornercolors**: Default `NULL`. The colors at the corners, in this order: NW, NE, SW, SE. If this vector isn’t present (or all corners are specified), the mid-points will just be interpolated from the main colors.

### Examples
```r
# Here is the `imhof1` palette:
create_texture("fff673","55967a","8fb28a","55967a","cfe0a9") %>%
plot_map()

# Here is the `unicorn` palette:
create_texture("red","green","blue","yellow","white") %>%
plot_map()
```
Description

Detects bodies of water (of a user-defined minimum size) within an elevation matrix.

Usage

detect_water(
  heightmap,
  zscale = 1,
  cutoff = 0.999,
  min_area = length(heightmap)/400,
  max_height = NULL,
  normalvectors = NULL,
  keep_groups = FALSE,
  progbar = FALSE
)

Arguments

heightmap A two-dimensional matrix, where each entry in the matrix is the elevation at that point. All grid points are assumed to be evenly spaced.

zscale Default ‘1’. The ratio between the x and y spacing (which are assumed to be equal) and the z axis. For example, if the elevation levels are in units of 1 meter and the grid values are separated by 10 meters, ‘zscale’ would be 10.

cutoff Default ‘0.999’. The lower limit of the z-component of the unit normal vector to be classified as water.

min_area Default length(heightmap)/400. Minimum area (in units of the height matrix x and y spacing) to be considered a body of water.

max_height Default ‘NULL’. If passed, this number will specify the maximum height a point can be considered to be water.

normalvectors Default ‘NULL’. Pre-computed array of normal vectors from the ‘calculate_normal’ function. Supplying this will speed up water detection.

keep_groups Default ‘FALSE’. If ‘TRUE’, the matrix returned will retain the numbered grouping information.

progbar Default ‘FALSE’. If ‘TRUE’, turns on progress bar.

Value

Matrix indicating whether water was detected at that point. 1 indicates water, 0 indicates no water.
generate_altitude_overlay

Examples

library(magrittr)
# Here we even out a portion of the volcano dataset to simulate water:
island_volcano = volcano
island_volcano[island_volcano < mean(island_volcano)] = mean(island_volcano)

# Setting a minimum area avoids classifying small flat areas as water:
island_volcano %>%
  sphere_shade(texture="imhof3") %>%
  add_water(detect_water(island_volcano, min_area = 400), color="imhof3") %>%
  plot_map()

---

generate_altitude_overlay

Generate Altitude Overlay

Description

Using a hillshade and the height map, generates a semi-transparent hillshade to layer onto an existing map.

Usage

generate_altitude_overlay(
  hillshade,
  heightmap,
  start_transition,
  end_transition = NULL,
  lower = TRUE
)

Arguments

hillshade The hillshade to transition into.
heightmap A two-dimensional matrix, where each entry in the matrix is the elevation at that point. All grid points are assumed to be evenly spaced.
start_transition Elevation above which 'hillshade' is completely transparent.
end_transition Default 'NULL'. Elevation below which 'hillshade' is completely opaque. By default, this is equal to 'start_transition'.
lower Default 'TRUE'. This makes 'hillshade' completely opaque below 'start_transition'. If 'FALSE', the direction will be reversed.

Value

4-layer RGB array representing the semi-transparent hillshade.
Examples

```r
# Create a bathymetric hillshade
# Only run these examples if the 'magick' package is installed.
if (!'magick' %in% rownames(utils::installed.packages())) {

  water_palette = colorRampPalette(c("darkblue", "dodgerblue", "lightblue"))(200)
bathy_hs = height_shade(montereybay, texture = water_palette)
plot_map(bathy_hs)

  # Set everything below 0m to water palette
  montereybay %>%
    sphere_shade(zscale=10) %>%
    add_overlay(generate_altitude_overlay(bathy_hs, montereybay, 0, 0)) %>%
    add_shadow(ray_shade(montereybay,zscale=50),0.3) %>%
    plot_map()

  # Add snow peaks by setting `lower = FALSE`
  snow_palette = "white"
snow_hs = height_shade(montereybay, texture = snow_palette)

  # Set the snow transition region from 500m to 1200m
  montereybay %>%
    sphere_shade(zscale=10, texture = "desert") %>%
    add_overlay(generate_altitude_overlay(bathy_hs, montereybay, 0, 0)) %>%
    add_overlay(generate_altitude_overlay(snow_hs, montereybay, 500, 1200, lower=FALSE)) %>%
    add_shadow(ambient_shade(montereybay,zscale=50,maxsearch=100),0) %>%
    plot_map()
}
```

---

**generate_compass_overlay**

*Generate Compass Overlay*

**Description**

This adds the compass

Based on code from "Auxiliary Cartographic Functions in R: North Arrow, Scale Bar, and Label with a Leader Arrow"

**Usage**

```r
generate_compass_overlay(
x = 0.85,
y = 0.15,
size = 0.075,
text_size = 1,
bearing = 0,
)```
heightmap = NULL,
width = NA,
height = NA,
color1 = "white",
color2 = "black",
text_color = "black",
border_color = "black",
border_width = 1,
halo_color = NA,
halo_expand = 1,
halo_alpha = 1,
halo_offset = c(0, 0),
halo_blur = 1
)

Arguments

x  Default ‘NULL’. The horizontal percentage across the map (measured from the bottom-left corner) where the compass is located.

y  Default ‘NULL’. The vertical percentage across the map (measured from the bottom-left corner) where the compass is located.

size  Default ‘0.05’. Size of the compass, in percentage of the map size.

text_size  Default ‘1’. Text size.

bearing  Default ‘0’. Angle (in degrees) of north.

heightmap  Default ‘NULL’. The original height map. Pass this in to extract the dimensions of the resulting RGB image array automatically.

width  Default ‘NA’. Width of the resulting image array. Default the same dimensions as height map.

height  Default ‘NA’. Width of the resulting image array. Default the same dimensions as height map.

color1  Default ‘white’. Primary color of the compass.

color2  Default ‘black’. Secondary color of the symcompass.

text_color  Default ‘black’. Text color.

border_color  Default ‘black’. Border color of the scale bar.

border_width  Default ‘1’. Width of the scale bar border.

halo_color  Default ‘NA’, no halo. If a color is specified, the compass will be surrounded by a halo of this color.

halo_expand  Default ‘1’. Number of pixels to expand the halo.

halo_alpha  Default ‘1’. Transparency of the halo.

halo_offset  Default ‘c(0,0)’. Horizontal and vertical offset to apply to the halo, in percentage of the image.

halo_blur  Default ‘1’. Amount of blur to apply to the halo. Values greater than ‘30’ won’t result in further blurring.
generate_compass_overlay

Value

Semi-transparent overlay with a compass.

Examples

#Only run these examples if the 'magick' package is installed.
if ("magick" %in% rownames(utils::installed.packages())) {

#Create the water palette
water_palette = colorRampPalette(c("darkblue", "dodgerblue", "lightblue"))(200)
bathy_hs = height_shade(montereybay, texture = water_palette)

#Generate flat water heightmap
mbay = montereybay
mbay[mbay < 0] = 0

base_map = mbay %>%
  height_shade() %>%
  add_overlay(generate_altitude_overlay(bathy_hs, montereybay, 0, 0)) %>%
  add_shadow(lamb_shade(montereybay,zscale=50),0.3)

#Plot a compass
base_map %>%
  add_overlay(generate_compass_overlay(heightmap = montereybay)) %>%
  plot_map()

#Change the position to be over the water
base_map %>%
  add_overlay(generate_compass_overlay(heightmap = montereybay, x = 0.15)) %>%
  plot_map()

#Change the text color for visibility
base_map %>%
  add_overlay(generate_compass_overlay(heightmap = montereybay, x = 0.15, text_color="white")) %>%
  plot_map()

#Alternatively, add a halo color to improve contrast
base_map %>%
  add_overlay(generate_compass_overlay(heightmap = montereybay, x = 0.15, y=0.15, halo_color="white", halo_expand = 1)) %>%
  plot_map()

#Alternatively, add a halo color to improve contrast
base_map %>%
  add_overlay(generate_compass_overlay(heightmap = montereybay, x = 0.15, y=0.15, halo_color="white", halo_expand = 1)) %>%
  plot_map()

#Change the color scheme
base_map %>%
  add_overlay(generate_compass_overlay(heightmap = montereybay, x = 0.15, y=0.15, halo_color="white", halo_expand = 1, color1 = "purple", color2 = "red")) %>%
```r
plot_map()

#Remove the inner border
base_map %>%
  add_overlay(generate_compass_overlay(heightmap = montereybay, x = 0.15, y=0.15,  
                                         border_color=NA,  
                                         halo_color="white", halo_expand = 1,  
                                         color1 = "darkolivegreen4", color2 = "burlywood3")) %>%
  plot_map()

#Change the size of the compass and text
base_map %>%
  add_overlay(generate_compass_overlay(heightmap = montereybay, x = 0.75, y=0.75,  
                                        halo_color="white", halo_expand = 1,  
                                        size=0.075*2, text_size = 1.25)) %>%
  add_overlay(generate_compass_overlay(heightmap = montereybay, x = 0.45, y=0.45,  
                                        halo_color="white", halo_expand = 1,  
                                        size=0.075)) %>%
  add_overlay(generate_compass_overlay(heightmap = montereybay, x = 0.15, y=0.15,  
                                        halo_color="white", halo_expand = 1,  
                                        size=0.075/2, text_size = 0.75)) %>%
  plot_map()

#Change the bearing of the compass
base_map %>%
  add_overlay(generate_compass_overlay(heightmap = montereybay, x = 0.85, y=0.85,  
                                        halo_color="white", halo_expand = 1, bearing=30,  
                                        size=0.075)) %>%
  add_overlay(generate_compass_overlay(heightmap = montereybay, x = 0.5, y=0.5,  
                                        halo_color="white", halo_expand = 1, bearing=15,  
                                        size=0.075)) %>%
  add_overlay(generate_compass_overlay(heightmap = montereybay, x = 0.15, y=0.15,  
                                        halo_color="white", halo_expand = 1, bearing=-45,  
                                        size=0.075)) %>%
  plot_map()

#Create a drop shadow effect
base_map %>%
  add_overlay(generate_compass_overlay(heightmap = montereybay, x = 0.15, y=0.15,  
                                        text_color="white", halo_alpha=0.5, halo_blur=2,  
                                        halo_color="black", halo_expand = 1, halo_offset = c(0.003,-0.003))) %>%
  plot_map()
```

---

**generate_contour_overlay**

*Generate Contour Overlay*
Description

Calculates and returns an overlay of contour lines for the current height map.

Usage

generate_contour_overlay(
  heightmap,
  levels = NA,
  nlevels = NA,
  zscale = 1,
  width = NA,
  height = NA,
  color = "black",
  linewidth = 1
)

Arguments

heightmap  A two-dimensional matrix, where each entry in the matrix is the elevation at that point. All grid points are assumed to be evenly spaced.
levels     Default 'NA'. Automatically generated with 10 levels. This argument specifies the exact height levels of each contour.
nlevels    Default 'NA'. Controls the auto-generation of levels. If levels is length-2, this will automatically generate 'nlevels' breaks between 'levels[1]' and 'levels[2]'..
zscale     Default '1'. The ratio between the x and y spacing (which are assumed to be equal) and the z axis. For example, if the elevation levels are in units of 1 meter and the grid values are separated by 10 meters, 'zscale' would be 10.
width      Default 'NA'. Width of the resulting overlay. Default the same dimensions as heightmap.
height     Default 'NA'. Width of the resulting overlay. Default the same dimensions as heightmap.
color      Default 'black'. Color.
linewidth  Default '1'. Line width.

Value

Semi-transparent overlay with contours.

Examples

#Add contours to the montereybay dataset

#Only run these examples if the 'magick' package is installed.
if ("magick" %in% rownames(utils::installed.packages())) {

  montereybay %>%
    height_shade() %>%
```r
generate_label_overlay

add_overlay(generate_contour_overlay(montereybay)) %>%
add_shadow(ray_shade(montereybay,zscale=50), 0.3) %>%
plot_map()

# Add a different contour color for above and below water, and specify levels manually
water_palette = colorRampPalette(c("darkblue", "dodgerblue", "lightblue"))(200)
bathy_hs = height_shade(montereybay, texture = water_palette)
breaks = seq(range(montereybay)[1], range(montereybay)[2], length.out=50)
water_breaks = breaks[breaks < 0]
land_breaks = breaks[breaks > 0]

montereybay %>%
  height_shade() %>%
  add_overlay(generate_altitude_overlay(bathy_hs, montereybay, 0, 0)) %>%
  add_shadow(ray_shade(montereybay, zscale=50), 0.3) %>%
  add_overlay(generate_contour_overlay(montereybay, levels = water_breaks, color="white")) %>%
  add_overlay(generate_contour_overlay(montereybay, levels = land_breaks, color="black")) %>%
  plot_map()

# Increase the resolution of the contour to improve the appearance of lines
montereybay %>%
  height_shade() %>%
  add_overlay(generate_contour_overlay(montereybay, levels = water_breaks, color="white",
                                        height = nrow(montereybay)*2,
                                        width = ncol(montereybay)*2)) %>%
  add_overlay(generate_contour_overlay(montereybay, levels = land_breaks, color="black",
                                        height = nrow(montereybay)*2,
                                        width = ncol(montereybay)*2)) %>%
  plot_map()

# Increase the number of breaks and the transparency (via add_overlay)
montereybay %>%
  height_shade() %>%
  add_shadow(ray_shade(montereybay, zscale=50), 0.3) %>%
  add_overlay(generate_contour_overlay(montereybay, linewidth=2, nlevels=100,
                                        height = nrow(montereybay)*2,
                                        width = ncol(montereybay)*2, color="black",
                                        alphalayer=0.5)) %>%
  plot_map()

# Manually specify the breaks with levels
montereybay %>%
  height_shade() %>%
  add_overlay(generate_contour_overlay(montereybay, linewidth=2, levels = seq(-2000, 0, 100))) %>%
  add_shadow(ray_shade(montereybay, zscale=50), 0.3) %>%
  plot_map()

}
```
**generate_label_overlay**

*Generate Label Overlay*

**Description**

This uses the `maptools::placeLabel()` function to generate labels for the given scene. Either use an `sf` object or manually specify the x/y coordinates and label.

**Usage**

```r
generate_label_overlay(
  labels,
  extent,
  x = NULL,
  y = NULL,
  heightmap = NULL,
  width = NA,
  height = NA,
  text_size = 1,
  color = "black",
  font = 1,
  pch = 16,
  point_size = 1,
  point_color = NA,
  offset = c(0, 0),
  data_label_column = NULL,
  halo_color = NA,
  halo_expand = 0,
  halo_alpha = 1,
  halo_offset = c(0, 0),
  halo_blur = 1,
  seed = NA
)
```

**Arguments**

- `labels` A character vector of labels, or an `sf` object with `POINT` geometry and a column for labels.
- `extent` A `raster::Extent` object with the bounding box for the height map used to generate the original map.
- `x` Default `NULL`. The x-coordinate, if `labels` is not an `sf` object.
- `y` Default `NULL`. The y-coordinate, if `labels` is not an `sf` object.
- `heightmap` Default `NULL`. The original height map. Pass this in to extract the dimensions of the resulting overlay automatically.
- `width` Default `NA`. Width of the resulting overlay. Default the same dimensions as height map.
generate_label_overlay

height Default 'NA'. Width of the resulting overlay. Default the same dimensions as height map.

text_size Default '1'. Text size.

color Default 'black'. Color of the labels.

font Default '1'. An integer which specifies which font to use for text. If possible, device drivers arrange so that 1 corresponds to plain text (the default), 2 to bold face, 3 to italic and 4 to bold italic.

pch Default '20', solid. Point symbol. '0' = square, '1' = circle, '2' = triangle point up, '3' = plus, '4' = cross, '5' = diamond, '6' = triangle point down, '7' = square cross, '8' = star, '9' = diamond plus, '10' = circle plus, '11' = triangles up and down, '12' = square plus, '13' = circle cross, '14' = square and triangle down, '15' = filled square, '16' = filled circle, '17' = filled triangle point-up, '18' = filled diamond, '19' = solid circle, '20' = bullet (smaller circle), '21' = filled circle blue, '22' = filled square blue, '23' = filled diamond blue, '24' = filled triangle point-up blue, '25' = filled triangle point down blue

point_size Default '0', no points. Point size.

point_color Default 'NA'. Colors of the points. Unless otherwise specified, this defaults to 'color'.

offset Default 'c(0,0)'. Horizontal and vertical offset to apply to the label, in units of 'geometry'.

data_label_column Default 'NULL'. The column in the 'sf' object that contains the labels.

halo_color Default 'NA', no halo. If a color is specified, the text label will be surrounded by a halo of this color.

halo_expand Default '2'. Number of pixels to expand the halo.

halo_alpha Default '1'. Transparency of the halo.

halo_offset Default 'c(0,0)'. Horizontal and vertical offset to apply to the halo, in units of 'geometry'.

halo_blur Default '1'. Amount of blur to apply to the halo. Values greater than '30' won’t result in further blurring.

seed Default 'NA', no seed. Random seed for ensuring the consistent placement of labels around points.

Value

Semi-transparent overlay with labels.

Examples

#Add the included 'sf' object with roads to the montereybay dataset
#Only run these examples if the 'magick' package is installed.
if (!'magick' %in% rownames(utils::installed.packages())) {

#Create the water palette
water_palette = colorRampPalette(c("darkblue", "dodgerblue", "lightblue"))(200)

bathy_hs = height_shade(montereybay, texture = water_palette)

# We're plotting the polygon data here for counties around Monterey Bay. We'll first
# plot the county names at the polygon centroids.
bathy_hs %>%
add_shadow(lamb_shade(montereybay,zscale=50),0.3) %>%
add_overlay(generate_polygon_overlay(monterey_counties_sf, palette = rainbow,
        extent = attr(montereybay,"extent"),
        heightmap = montereybay)) %>%
add_overlay(generate_label_overlay(labels=monterey_counties_sf,
        color="black", point_size = 1, text_size = 1,
        data_label_column = "NAME",
        extent= attr(montereybay,"extent"), heightmap = montereybay,
        seed=1)) %>%
plot_map()

# It's hard to read these values, so we'll add a white halo.
bathy_hs %>%
add_shadow(lamb_shade(montereybay,zscale=50),0.3) %>%
add_overlay(generate_polygon_overlay(monterey_counties_sf, palette = rainbow,
        extent = attr(montereybay,"extent"),
        heightmap = montereybay)) %>%
add_overlay(generate_label_overlay(labels=monterey_counties_sf,
        color="black", point_size = 1, text_size = 1,
        data_label_column = "NAME",
        extent= attr(montereybay,"extent"), heightmap = montereybay,
        halo_color = "white", halo_expand = 3,
        seed=1)) %>%
plot_map()

# Plot the actual town locations, using the manual plotting interface instead of the `sf` object
montereybay %>%
height_shade() %>%
add_overlay(generate_altitude_overlay(bathy_hs, montereybay, 0, 0)) %>%
add_shadow(lamb_shade(montereybay,zscale=50),0.3) %>%
add_overlay(generate_label_overlay(labels=as.character(monterey_counties_sf$NAME),
        x=as.numeric(as.character(monterey_counties_sf$INTPTLON)),
        y=as.numeric(as.character(monterey_counties_sf$INTPTLAT)),
        color="black", point_size = 1, text_size = 1,
        extent= attr(montereybay,"extent"), heightmap = montereybay,
        halo_color = "white", halo_expand = 3,
        seed=1)) %>%
plot_map()

# Adding a softer blurred halo
montereybay %>%
height_shade() %>%
add_overlay(generate_altitude_overlay(bathy_hs, montereybay, 0, 0)) %>%
add_shadow(lamb_shade(montereybay,zscale=50),0.3) %>%
add_overlay(generate_label_overlay(labels=as.character(monterey_counties_sf$NAME),
        x=as.numeric(as.character(monterey_counties_sf$INTPTLON)),
        y=as.numeric(as.character(monterey_counties_sf$INTPTLAT)),
        color="black", point_size = 1, text_size = 1,
        extent= attr(montereybay,"extent"), heightmap = montereybay,
        halo_color = "white", halo_expand = 3,
        seed=1)) %>%
generate_line_overlay

```
color="black", point_size = 1, text_size = 1, 
extent= attr(montereybay,"extent"), heightmap = montereybay, 
halo_color = "white", halo_expand = 3, halo_blur=10, 
seed=1)) %>%

plot_map()

#Changing the seed changes the locations of the labels
montereybay %>%
  height_shade() %>%
  add_overlay(generate_altitude_overlay(bathy_hs, montereybay, 0, 0)) %>%
  add_shadow(lamb_shade(montereybay,zscale=50),0.3) %>%
  add_overlay(generate_label_overlay(labels=as.character(monterey_counties_sf$NAME), 
                                x=as.numeric(as.character(monterey_counties_sf$INTPTLON)), 
                                y=as.numeric(as.character(monterey_counties_sf$INTPTLAT)), 
                                color="black", point_size = 1, text_size = 1, 
                                extent= attr(montereybay,"extent"), heightmap = montereybay, 
                                halo_color = "white", halo_expand = 3, halo_blur=10, 
                                seed=2)) %>%

plot_map()

}
```

---

### generate_line_overlay  Generate Line Overlay

#### Description

Calculates and returns an overlay of contour lines for the current height map.

#### Usage

```
generate_line_overlay(
  geometry, 
  extent, 
  heightmap = NULL, 
  width = NA, 
  height = NA, 
  color = "black", 
  linewidth = 1, 
  lty = 1, 
  data_column_width = NULL, 
  offset = c(0, 0)
)
```

#### Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>geometry</td>
<td>An <code>sf</code> object with LINESTRING geometry.</td>
</tr>
<tr>
<td>extent</td>
<td>A <code>raster::Extent</code> object with the bounding box for the height map used to generate the original map.</td>
</tr>
</tbody>
</table>
**heightmap**
Default 'NULL'. The original height map. Pass this in to extract the dimensions of the resulting overlay automatically.

**width**
Default 'NA'. Width of the resulting overlay. Default the same dimensions as height map.

**height**
Default 'NA'. Width of the resulting overlay. Default the same dimensions as height map.

**color**
Default 'black'. Color of the lines.

**linewidth**
Default '1'. Line width.

**lty**
Default '1'. Line type. ‘1’ is solid, ‘2’ is dashed, ‘3’ is dotted, ‘4’ is dot-dash, ‘5’ is long dash, and ‘6’ is dash-long-dash.

**data_column_width**
Default 'NULL'. The numeric column to map the width to. The maximum width will be the value specified in 'linewidth'.

**offset**
Default 'c(0,0)'. Horizontal and vertical offset to apply to the line, in units of 'geometry'.

**Value**
Semi-transparent overlay with contours.

**Examples**

```
#Add the included 'sf' object with roads to the montereybay dataset
#Only run these examples if the 'magick' package is installed.
if ("magick" %in% rownames(utils::installed.packages())) {

  water_palette = colorRampPalette(c("darkblue", "dodgerblue", "lightblue"))(200)
  bathy_hs = height_shade(montereybay, texture = water_palette)
  montereybay %>%
    height_shade() %>%
    add_overlay(generate_altitude_overlay(bathy_hs, montereybay, 0, 0)) %>%
    add_overlay(generate_line_overlay(monterey_roads_sf, attr(montereybay,"extent"), heightmap = montereybay), alphalayer=0.8) %>%
    add_shadow(ray_shade(montereybay,zscale=50),0.3) %>%
    plot_map()

  #Change the line width, color, and transparency
  montereybay %>%
    height_shade() %>%
    add_overlay(generate_altitude_overlay(bathy_hs, montereybay, 0, 0)) %>%
    add_overlay(generate_line_overlay(monterey_roads_sf, linewidth=3, color="white", attr(montereybay,"extent"), heightmap = montereybay), alphalayer=0.8) %>%
    add_shadow(ray_shade(montereybay,zscale=50),0.3) %>%
    plot_map()

  #Manually specify the width and height to improve visual quality of the lines
  montereybay %>%
    height_shade() %>%
```
add_overlay(generate_altitude_overlay(bathy_hs, montereybay, 0, 0)) %>%
add_shadow(ray_shade(montereybay,zscale=50),0.3) %>%
add_overlay(generate_line_overlay(monterey_roads_sf, linewidth=3, color="white",
attr(montereybay,"extent"), width = 1080, height = 1080),
alphalayer=0.8) %>%
plot_map()

}

generate_point_overlay

Generate Point Overlay

Description

Calculates and returns an overlay of points for the current map.

Usage

generate_point_overlay(
  geometry,
  extent,
  heightmap = NULL,
  width = NA,
  height = NA,
  pch = 20,
  color = "black",
  size = 1,
  offset = c(0, 0),
  data_column_width = NULL
)

Arguments

geometry An ‘sf’ object with POINT geometry.
extent A ‘raster::Extent’ object with the bounding box for the height map used to generate the original map.
heightmap Default ‘NULL’. The original height map. Pass this in to extract the dimensions of the resulting overlay automatically.
width Default ‘NA’. Width of the resulting overlay. Default the same dimensions as height map.
height Default ‘NA’. Width of the resulting overlay. Default the same dimensions as height map.
generate_polygon_overlay

Description

Transforms an input `sf` object into an image overlay for the current height map.

Usage

```r
generate_polygon_overlay(
  geometry,
  extent,
  heightmap = NULL,
  color = "black", color = "black", size = 1, size = 1,
  offset = c(0, 0), offset = c(0, 0),
  data_column_width = NULL, data_column_width = NULL
)
```

Examples

```r
# Add the included `sf` object with roads to the montereybay dataset
if(all(c("sf", "magick") %in% rownames(utils::installed.packages()))) {
  monterey_city = sf::st_sfc(sf::st_point(c(-121.893611, 36.603056)))
  montereybay <-
    monterey_city %>%
    height_shade() %>%
    add_overlay(generate_point_overlay(monterey_city, color="red", size=12,
                                        offset = attr(montereybay, "extent"), heightmap = montereybay)) %>%
    add_shadow(ray_shade(montereybay, zscale=50), 0.3) %>%
    plot_map()
}
```
generate_polygon_overlay

width = NA,
height = NA,
offset = c(0, 0),
data_column_fill = NULL,
linemcolor = "black",
palette = "white",
linewidth = 1
)

Arguments

geometry An ‘sf’ object with POLYGON geometry.
extent A ‘raster::Extent’ object with the bounding box for the height map used to generate the original map.
heightmap Default ‘NULL’. The original height map. Pass this in to extract the dimensions of the resulting overlay automatically.
width Default ‘NA’. Width of the resulting overlay. Default the same dimensions as height map.
height Default ‘NA’. Width of the resulting overlay. Default the same dimensions as height map.
offset Default ‘c(0,0)’. Horizontal and vertical offset to apply to the polygon, in units of ‘geometry’.
data_column_fill Default ‘NULL’. The column to map the polygon fill color to.
linemcolor Default ‘black’. Color of the lines.
palette Default ‘black’. Single color, named vector color palette, or palette function. If this is a named vector and ‘data_column_fill’ is not ‘NULL’, it will map the colors in the vector to the names. If ‘data_column_fill’ is a numeric column, this will give a continuous mapping.
linewdth Default ‘1’. Line width.

Value
Image overlay representing the input polygon data.

Examples

#Plot the counties around Monterey Bay, CA
#Only run these examples if the ‘magick’ package is installed.
if ("magick" %in% rownames(utils::installed.packages())) {

generate_polygon_overlay(monterey_counties_sf, palette = rainbow,
extent = attr(montereybay,"extent"), heightmap = montereybay) %>%
plot_map()

#These counties include the water, so we'll plot bathymetry data over the polygon
data to only include parts of the polygon that fall on land.
generate_scalebar_overlay

Generate Scalebar Overlay

Description

This function creates an overlay with a scale bar of a user-specified length. It uses the coordinates of the map (specified by passing an extent) and then creates a scale bar at a specified x/y proportion across the map. If the map is not projected (i.e. is in lat/long coordinates) this function will use the 'geosphere' package to create a scale bar of the proper length.

Usage

generate_scalebar_overlay(
    extent,
length,
  x = 0.05,
  y = 0.05,
  latlong = FALSE,
  thickness = NA,
  bearing = 90,
  unit = "m",
  flip_ticks = FALSE,
  labels = NA,
  text_size = 1,
  decimals = 0,
  text_offset = 1,
  adj = 0.5,
  heightmap = NULL,
  width = NA,
  height = NA,
  color1 = "white",
  color2 = "black",
  text_color = "black",
  font = 1,
  border_color = "black",
  tick_color = "black",
  border_width = 1,
  tick_width = 1,
  halo_color = NA,
  halo_expand = 1,
  halo_alpha = 1,
  halo_offset = c(0, 0),
  halo_blur = 1
)

Arguments

extent    A 'raster::Extent' object with the bounding box for the height map used to generate the original map. If this is in lat/long coordinates, be sure to set `latlong = TRUE`.

length    The length of the scale bar, in 'units'. This should match the units used on the map, unless `extent` uses lat/long coordinates. In that case, the distance should be in meters.

x          Default '0.05'. The x-coordinate of the bottom-left corner of the scale bar, as a proportion of the full map width.

y          Default '0.05'. The y-coordinate of the bottom-left corner of the scale bar, as a proportion of the full map height.

latlong    Default 'FALSE'. Set to 'TRUE' if the map is in lat/long coordinates to get an accurate scale bar (using distance calculated with the 'geosphere' package).

thickness  Default 'NA'; automatically computed as 1/20th the length of the scale bar. Width of the scale bar.
generate_scalebar_overlay

<table>
<thead>
<tr>
<th>Option</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bearing</td>
<td>‘90’, horizontal.</td>
<td>Direction (measured from north) of the scale bar.</td>
</tr>
<tr>
<td>unit</td>
<td>‘m’</td>
<td>Displayed unit on the scale bar.</td>
</tr>
<tr>
<td>flip_ticks</td>
<td>‘FALSE’</td>
<td>Whether to flip the ticks to the other side of the scale bar.</td>
</tr>
<tr>
<td>labels</td>
<td>‘NA’</td>
<td>Manually specify the three labels with a length-3 character vector. Use this if you want display units other than meters.</td>
</tr>
<tr>
<td>text_size</td>
<td>‘1’</td>
<td>Text size.</td>
</tr>
<tr>
<td>decimals</td>
<td>‘0’</td>
<td>Number of decimal places for scale bar labels.</td>
</tr>
<tr>
<td>text_offset</td>
<td>‘1’</td>
<td>Amount of offset to apply to the text from the scale bar, as a multiple of ‘thickness’.</td>
</tr>
<tr>
<td>adj</td>
<td>‘0.5’, centered.</td>
<td>Text justification. ‘0’ is left-justified, and ‘1’ is right-justified.</td>
</tr>
<tr>
<td>heightmap</td>
<td>‘NULL’</td>
<td>The original height map. Pass this in to extract the dimensions of the resulting RGB image array automatically.</td>
</tr>
<tr>
<td>width</td>
<td>‘NA’</td>
<td>Width of the resulting image array. Default the same dimensions as height map.</td>
</tr>
<tr>
<td>height</td>
<td>‘NA’</td>
<td>Width of the resulting image array. Default the same dimensions as height map.</td>
</tr>
<tr>
<td>color1</td>
<td>‘black’</td>
<td>Primary color of the scale bar.</td>
</tr>
<tr>
<td>color2</td>
<td>‘white’</td>
<td>Secondary color of the scale bar.</td>
</tr>
<tr>
<td>text_color</td>
<td>‘black’</td>
<td>Text color.</td>
</tr>
<tr>
<td>font</td>
<td>‘1’</td>
<td>An integer which specifies which font to use for text. If possible, device drivers arrange so that 1 corresponds to plain text (the default), 2 to bold face, 3 to italic and 4 to bold italic.</td>
</tr>
<tr>
<td>border_color</td>
<td>‘black’</td>
<td>Border color of the scale bar.</td>
</tr>
<tr>
<td>tick_color</td>
<td>‘black’</td>
<td>Tick color of the scale bar.</td>
</tr>
<tr>
<td>border_width</td>
<td>‘1’</td>
<td>Width of the scale bar border.</td>
</tr>
<tr>
<td>tick_width</td>
<td>‘1’</td>
<td>Width of the tick.</td>
</tr>
<tr>
<td>halo_color</td>
<td>‘NA’</td>
<td>No halo. If a color is specified, the text label will be surrounded by a halo of this color.</td>
</tr>
<tr>
<td>halo_expand</td>
<td>‘1’</td>
<td>Number of pixels to expand the halo.</td>
</tr>
<tr>
<td>halo_alpha</td>
<td>‘1’</td>
<td>Transparency of the halo.</td>
</tr>
<tr>
<td>halo_offset</td>
<td>‘c(0,0)’</td>
<td>Horizontal and vertical offset to apply to the halo, as a proportion of the full scene.</td>
</tr>
<tr>
<td>halo_blur</td>
<td>‘1’</td>
<td>Amount of blur to apply to the halo. Values greater than ‘30’ won’t result in further blurring.</td>
</tr>
</tbody>
</table>
Examples

# Only run these examples if the 'magick' package is installed.
if ("magick" %in% rownames(utils::installed.packages())) {

    # Create the water palette
    water_palette = colorRampPalette(c("darkblue", "dodgerblue", "lightblue"))(200)
    bathy_hs = height_shade(montereybay, texture = water_palette)

    # Generate flat water heightmap
    mbay = montereybay
    mbay[mbay < 0] = 0

    base_map = mbay %>%
                height_shade() %>%
                add_overlay(generate_altitude_overlay(bathy_hs, montereybay, 0, 0)) %>%
                add_shadow(lamb_shade(montereybay, zscale=50), 0.3)

    # For convenience, the extent of the montereybay dataset is included as an attribute
    mb_extent = attr(montereybay, "extent")

    # Add a scalebar
    base_map %>%
                add_overlay(generate_scalebar_overlay(extent = mb_extent, length = 40000,
                                                    heightmap = montereybay,
                                                    latlong=TRUE)) %>%
                plot_map()

    # Change the text color
    base_map %>%
                add_overlay(generate_scalebar_overlay(extent = mb_extent, length = 40000,
                                                    text_color = "white",
                                                    heightmap = montereybay,
                                                    latlong=TRUE)) %>%
                plot_map()

    # Change the length
    base_map %>%
                add_overlay(generate_scalebar_overlay(extent = mb_extent, length = 30000,
                                                    text_color = "white",
                                                    heightmap = montereybay,
                                                    latlong=TRUE)) %>%
                plot_map()

    # Change the thickness (default is length/20)
    base_map %>%
                add_overlay(generate_scalebar_overlay(extent = mb_extent, length = 30000,
                                                    text_color = "white", thickness = 30000/10,
                                                    heightmap = montereybay,
                                                    latlong=TRUE)) %>%
                plot_map()
#Change the text offset (given in multiples of thickness)
base_map %>%
  add_overlay(generate_scalebar_overlay(extent = mb_extent, length = 30000,
    text_color = "white", thickness = 30000/10,
    text_offset = 0.75,
    heightmap = montereybay, latlong=TRUE)) %>%
plot_map()

#Change the primary and secondary colors, along with the border and tick color
base_map %>%
  add_overlay(generate_scalebar_overlay(extent = mb_extent, length = 30000,
    text_color = "white", border_color = "white",
    tick_color = "white",
    color1 = "darkolivegreen4", color2 = "burlywood3",
    heightmap = montereybay, latlong=TRUE)) %>%
plot_map()

#Add a halo
base_map %>%
  add_overlay(generate_scalebar_overlay(extent = mb_extent, length = 40000,
    halo_color = "white", halo_expand = 1,
    heightmap = montereybay, latlong=TRUE)) %>%
plot_map()

#Change the orientation, position, text alignment, and flip the ticks to the other side
base_map %>%
  add_overlay(generate_scalebar_overlay(extent = mb_extent, length = 40000, x = 0.07,
    bearing=0, adj = 0, flip_ticks = TRUE,
    halo_color = "white", halo_expand = 1.5,
    heightmap = montereybay, latlong=TRUE)) %>%
plot_map()

#64373.8 meters in 40 miles
#Create custom labels, change font and text size, remove the border/ticks, and change the color
#Here, we specify a width and height to double the resolution of the image (for sharper text)
base_map %>%
  add_overlay(generate_scalebar_overlay(extent = mb_extent, length = 64373.8, x = 0.07,
    labels = c("0", "20", "40 miles"), thickness=2500,
    text_size=3, font = 2, text_offset = 0,
    text_color="white", color2="#bf323b", border_color=NA,
    tick_color="red", tick_width=0,
    bearing=0, adj = 0, flip_ticks = TRUE,
    halo_color="black", haloblur=3, halo_alpha=0.5,
    width = ncol(montereybay)*2,
    height = nrow(montereybay)*2,
    latlong=TRUE), rescale_original=TRUE) %>%
plot_map()
`generate_waterline_overlay`

`generate_waterline_overlay()`

*Generate Waterline Overlay*

**Description**

Using a height map or a boolean matrix, generates a semi-transparent waterline overlay to layer onto an existing map. This uses the method described by P. Felzenszwalb & D. Huttenlocher in "Distance Transforms of Sampled Functions" (Theory of Computing, Vol. 8, No. 19, September 2012) to calculate the distance to the coast. This distance matrix can be returned directly by setting the 'return_distance_matrix' argument to 'TRUE'.

**Usage**

```r
generate_waterline_overlay(
  heightmap,
  color = "white",
  linewidth = 1,
  boolean = FALSE,
  min = 0.001,
  max = 0.2,
  breaks = 9,
  smooth = 0,
  fade = TRUE,
  alpha_dist = max,
  alpha = 1,
  falloff = 1.3,
  evenly_spaced = FALSE,
  zscale = 1,
  cutoff = 0.999,
  min_area = length(heightmap)/400,
  max_height = NULL,
  return_distance_matrix = FALSE
)
```

**Arguments**

- `heightmap`: A two-dimensional matrix, where each entry in the matrix is the elevation at that point. If `boolean = TRUE`, this will instead be interpreted as a logical matrix indicating areas of water.
- `color`: Default ‘white’. Color of the lines.
- `linewidth`: Default ‘1’. Line width.
boolean  Default ‘FALSE’. If ‘TRUE’, this is a boolean matrix (0 and 1) indicating contiguous areas in which the lines are generated (instead of a height matrix, from which the boolean matrix is derived using ‘detect_water()’).

min  Default ‘0.001’. Percent distance (measured from the furthest point from shore) where the waterlines stop.

max  Default ‘0.2’. Percent distance (measured from the furthest point from shore) where the waterlines begin.

breaks  Default ‘9’. Number of water lines.

smooth  Default ‘0’, no smoothing. Increase this to smooth water lines around corners.

fade  Default ‘FALSE’. If ‘FALSE’, lines will not fade with distance from shore.

alpha_dist  Default to the value specified in ‘max’. Percent distance (measured from the furthest point from shore) where the waterlines fade entirely, when ‘fade = TRUE’.

alpha  Default ‘1’. Maximum transparency for waterlines. This scales the transparency for all other levels.

falloff  Default ‘1.3’. Multiplicative decrease in distance between each waterline level.

evenly_spaced  Default ‘FALSE’. If ‘TRUE’, ‘falloff’ will be ignored and the lines will be evenly spaced.

zscale  Default ‘1’. Arguments passed to ‘detect_water()’. Ignored if ‘boolean = TRUE’. The ratio between the x and y spacing (which are assumed to be equal) and the z axis. For example, if the elevation levels are in units of 1 meter and the grid values are separated by 10 meters, ‘zscale’ would be 10.

cutoff  Default ‘0.999’. Arguments passed to ‘detect_water()’. Ignored if ‘boolean = TRUE’. The lower limit of the z-component of the unit normal vector to be classified as water.

min_area  Default ‘length(heightmap)/400’. Arguments passed to ‘detect_water()’. Ignored if ‘boolean = TRUE’. Minimum area (in units of the height matrix x and y spacing) to be considered a body of water.

max_height  Default ‘NULL’. Arguments passed to ‘detect_water()’. Ignored if ‘boolean = TRUE’. If passed, this number will specify the maximum height a point can be considered to be water. ‘FALSE’, the direction will be reversed.

return_distance_matrix  Default ‘FALSE’. If ‘TRUE’, this function will return the boolean distance matrix instead of contour lines.

Value

4-layer RGB array representing the waterline overlay.

Examples

```r
#Only run these examples if the 'magick' package is installed.
if ("magick" %in% rownames(utils::installed.packages())) {

#Create a flat body of water for Monterey Bay
montbay = montereybay
```
montbay[montbay < 0] = 0

# Generate base map with no lines
basemap = montbay %>%
  height_shade() %>%
  add_water(detect_water(montbay), color="dodgerblue") %>%
  add_shadow(texture_shade(montbay, detail=1/3, brightness = 15, contrast = 5),0) %>%
  add_shadow(lamb_shade(montbay, zscale=50),0)
plot_map(basemap)

# Add waterlines
basemap %>%
  add_overlay(generate_waterline_overlay(montbay)) %>%
  plot_map()

# Change minimum line distance:
basemap %>%
  add_overlay(generate_waterline_overlay(montbay, min = 0.02)) %>%
  plot_map()

# Change maximum line distance
basemap %>%
  add_overlay(generate_waterline_overlay(montbay, max = 0.4)) %>%
  plot_map()

# Smooth waterlines
basemap %>%
  add_overlay(generate_waterline_overlay(montbay, max = 0.4, smooth=2)) %>%
  plot_map()

# Increase number of breaks
basemap %>%
  add_overlay(generate_waterline_overlay(montbay, breaks = 20, max=0.4)) %>%
  plot_map()

# Make lines evenly spaced:
basemap %>%
  add_overlay(generate_waterline_overlay(montbay, evenly_spaced = TRUE)) %>%
  plot_map()

# Change variable distance between each line
basemap %>%
  add_overlay(generate_waterline_overlay(montbay, falloff=1.5)) %>%
  plot_map()

# Turn off fading
basemap %>%
  add_overlay(generate_waterline_overlay(montbay, fade=FALSE)) %>%
  plot_map()

# Fill up the entire body of water with lines and make them all 50% transparent
basemap %>%
  add_overlay(generate_waterline_overlay(montbay, max = 0.4, smooth=2, breaks = 20, evenly_spaced = TRUE, falloff=1.5, fade=FALSE)) %>%
  plot_map()
height_shade

```r
add_overlay(generate_waterline_overlay(montbay, fade=FALSE, max=1, alpha = 0.5, color="white", evenly_spaced = TRUE, breaks=50)) %>%
plot_map()
}
```

### height_shade

**Calculate Terrain Color Map**

**Description**

Calculates a color for each point on the surface using a direct elevation-to-color mapping.

**Usage**

```r
height_shade(
  heightmap,
  texture = (grDevices::colorRampPalette(c("#6AA85B", "#D9CC9A", "#FFFFFF")))(256),
  range = NULL,
  keep_user_par = TRUE
)
```

**Arguments**

- `heightmap` A two-dimensional matrix, where each entry in the matrix is the elevation at that point.
- `texture` Default ‘terrain.colors(256)’. A color palette for the plot.
- `range` Default ‘NULL’, the full range of the heightmap. A length-2 vector specifying the maximum and minimum values to map the color palette to.
- `keep_user_par` Default ‘TRUE’. Whether to keep the user’s ‘par()' settings. Set to ‘FALSE’ if you want to set up a multi-pane plot (e.g. set 'par(mfrow)').

**Value**

RGB array of hillshaded texture mappings.

**Examples**

```r
#Create a direct mapping of elevation to color:
montereybay %>%
  height_shade() %>%
  plot_map()

#Add a shadow:
montereybay %>%
  height_shade() %>%
  add_shadow(ray_shade(montereybay,zscale=50),0.3) %>%
```
lamb_shade

plot_map()

# Change the palette:

montereybay %>%
  height_shade(texture = topo.colors(256)) %>%
  add_shadow(ray_shade(montereybay, zscale=50), 0.3) %>%
  plot_map()

# Really change the palette:

montereybay %>%
  height_shade(texture = rainbow(256)) %>%
  add_shadow(ray_shade(montereybay, zscale=50), 0.3) %>%
  plot_map()

---

lamb_shade  Calculate Lambert Shading Map

Description

Calculates local shadow map for a elevation matrix by calculating the dot product between light direction and the surface normal vector at that point. Each point’s intensity is proportional to the cosine of the normal vector.

Usage

lamb_shade(
  heightmap,
  sunaltitude = 45,
  sunangle = 315,
  zscale = 1,
  zero_negative = TRUE
)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>heightmap</td>
<td>A two-dimensional matrix, where each entry in the matrix is the elevation at that point. All points are assumed to be evenly spaced.</td>
</tr>
<tr>
<td>sunaltitude</td>
<td>Default ‘45’. The azimuth angle as measured from the horizon from which the light originates.</td>
</tr>
<tr>
<td>sunangle</td>
<td>Default ‘315’ (NW). The angle around the matrix from which the light originates.</td>
</tr>
<tr>
<td>zscale</td>
<td>Default ‘1’. The ratio between the x and y spacing (which are assumed to be equal) and the z axis.</td>
</tr>
</tbody>
</table>
zero_negative  Default ‘TRUE’. Zeros out all values below 0 (corresponding to surfaces facing away from the light source).

Value

Matrix of light intensities at each point.

Examples

```r
#Generate a basic hillshade
montereybay %>%
  lamb_shade(zscale=200) %>%
  plot_map()

#Increase the intensity by decreasing the zscale
montereybay %>%
  lamb_shade(zscale=50) %>%
  plot_map()

#Change the sun direction
montereybay %>%
  lamb_shade(zscale=200, sunangle=45) %>%
  plot_map()

#Change the sun altitude
montereybay %>%
  lamb_shade(zscale=200, sunaltitude=60) %>%
  plot_map()
```

Description

This dataset is a downsampled version of a combined topographic and bathymetric elevation matrix representing the Monterey Bay, CA region. Original data from the NOAA National Map website.

Usage

montereybay

Format

A matrix with 540 rows and 540 columns. Elevation is in meters, and the spacing between each coordinate is 200 meters (zscale = 200). Water level is 0. Raster extent located in "extent" attribute. CRS located in "CRS" attribute.
monterey_counties_sf

**Source**


**Examples**

```r
# This is the full code (commented out) used to generate this dataset from the original NOAA data:
# raster::raster("monterey_13_navd88_2012.nc")
# bottom_left = c(y=-122.366765, x=36.179392)
# top_right = c(y=-121.366765, x=37.179392)
# extent_latlong = sp::SpatialPoints(rbind(bottom_left, top_right),
#     proj4string=sp::CRS("+proj=longlat +ellps=WGS84 +datum=WGS84"))
# monterey_cropped = raster::crop(montbay,extent_latlong)
# montbay_mat = raster_to_matrix(montbay_cropped)
# montereybay = resize_matrix(montbay_mat,0.05)
# attr(montereybay, "extent") = extent_latlong
# attr(montereybay, "crs") = crs(monterey_cropped)
# attr(montereybay, "crs") = crs(monterey_cropped)
# attr(montereybay, "rayshader_data") = TRUE
```

---

**monterey_counties_sf  California County Data Around Monterey Bay**

**Description**

This dataset is an `sf` object containing polygon data from the U.S. Department of Commerce with selected geographic and cartographic information from the U.S. Census Bureau’s Master Address File / Topologically Integrated Geographic Encoding and Referencing (MAF/TIGER) Database (MTDB). This data has been trimmed to only include 26 features in the extent of the `montereybay` dataset.

**Usage**

```r
monterey_counties_sf
```

**Format**

An `sf` object with MULTIPOLYGON geometry.

**Source**


**Examples**

```r
# This is the full code (commented out) used to generate this dataset from the original data:
# counties = sf::st_read("t1_2016_06_cousub.shp")
# monterey_counties_sf = sf::st_crop(counties, attr(montereybay,"extent"))
```
**monterey_roads_sf**  
*Road Data Around Monterey Bay*

### Description
This dataset is an ‘sf’ object containing line data from the U.S. Department of Commerce with selected roads, TIGER/Line Shapefile, 2015, state, California, Primary and Secondary Roads State-based Shapefile. This data has been trimmed to only include 330 features in the extent of the ‘montereybay’ dataset.

### Usage
```
monterey_roads_sf
```

### Format
An ‘sf’ object with LINESTRING geometry.

### Source
https://www2.census.gov/geo/tiger/TIGER2015/PRISECROADS/tl_2015_06_prisecroads.zip

### Examples
```r
# This is the full code (commented out) used to generate this dataset from the original data:
#counties = sf::st_read("tl_2015_06_prisecroads.shp")
#monterey_roads_sf = sf::st_crop(counties, attr(montereybay,"extent"))
```

---

**plot_3d**  
*Plot 3D*

### Description
Displays the shaded map in 3D with the ‘rgl’ package.

### Usage
```
plot_3d(
    hillshade,
    heightmap,
    zscale = 1,
    baseshape = "rectangle",
    solid = TRUE,
    soliddepth = "auto",
    solidcolor = "grey20",
    solidlinecolor = "grey30",
)```
plot_3d

shadow = TRUE,
shadowdepth = "auto",
shadowcolor = "grey50",
shadowwidth = "auto",
water = FALSE,
waterdepth = 0,
watercolor = "dodgerblue",
wateralpha = 0.5,
waterlinecolor = NULL,
waterlinealpha = 1,
linewidth = 2,
lineantialias = FALSE,
theta = 45,
phi = 45,
fov = 0,
zoom = 1,
background = "white",
windowsize = 600,
precomputed_normals = NULL,
asp = 1,
triangulate = FALSE,
max_error = 0,
max_tri = 0,
verbose = FALSE,
...

Arguments

hillshade                   Hillshade/image to be added to 3D surface map.
heightmap                  A two-dimensional matrix, where each entry in the matrix is the elevation at that point. All points are assumed to be evenly spaced.
zscale                     Default '1'. The ratio between the x and y spacing (which are assumed to be equal) and the z axis. For example, if the elevation levels are in units of 1 meter and the grid values are separated by 10 meters, 'zscale' would be 10. Adjust the zscale down to exaggerate elevation features.
baseshape                  Default 'rectangle'. Shape of the base. Options are c("rectangle","circle","hex").
solid                      Default 'TRUE'. If 'FALSE', just the surface is rendered.
soliddepth                 Default 'auto', which sets it to the lowest elevation in the matrix minus one unit (scaled by zscale). Depth of the solid base.
solidcolor                 Default 'grey20'. Base color.
solidlinecolor             Default 'grey30'. Base edge line color.
shadow                     Default 'TRUE'. If 'FALSE', no shadow is rendered.
shadowdepth                Default 'auto', which sets it to 'soliddepth - soliddepth/10'. Depth of the shadow layer.
shadowcolor                Default 'grey50'. Color of the shadow.
shadowwidth  Default 'auto', which sizes it to 1/10th the smallest dimension of 'heightmap'. Width of the shadow in units of the matrix.
water         Default 'FALSE'. If 'TRUE', a water layer is rendered.
waterdepth    Default '0'. Water level.
watercolor    Default 'lightblue'. Color of the water.
wateralpha    Default '0.5'. Water transparency.
waterlinecolor Default 'NULL'. Color of the lines around the edges of the water layer.
waterlinealpha Default '1'. Water line transparency.
linewidth     Default '2'. Width of the edge lines in the scene.
lineantialias Default 'FALSE'. Whether to anti-alias the lines in the scene.
theta         Default '45'. Rotation around z-axis.
phi            Default '45'. Azimuth angle.
fov            Default '0'–isometric. Field-of-view angle.
zoom          Default '1'. Zoom factor.
background    Default 'grey10'. Color of the background.
windowsize    Default '600'. Position, width, and height of the 'rgl' device displaying the plot. If a single number, viewport will be a square and located in upper left corner. If two numbers, (e.g. 'c(600,800)'), user will specify width and height separately. If four numbers (e.g. 'c(200,0,600,800)'), the first two coordinates specify the location of the x-y coordinates of the bottom-left corner of the viewport on the screen, and the next two (or one, if square) specify the window size. NOTE: The absolute positioning of the window does not currently work on macOS (tested on Mojave), but the size can still be specified.
precomputed_normals Default 'NULL'. Takes the output of 'calculate_normals()' to save computing normals internally.
asp           Default '1'. Aspect ratio of the resulting plot. Use 'asp = 1/cospi(mean_latitude/180)' to rescale lat/long at higher latitudes to the correct the aspect ratio.
triangulate   Default 'FALSE'. Reduce the size of the 3D model by triangulating the height map. Set this to 'TRUE' if generating the model is slow, or moving it is choppy. Will also reduce the size of 3D models saved to disk.
max_error     Default '0.001'. Maximum allowable error when triangulating the height map, when 'triangulate = TRUE'. Increase this if you encounter problems with 3D performance, want to decrease render time with 'render_highquality()', or need to save a smaller 3D OBJ file to disk with 'save_obj()'.
max_tri       Default '0', which turns this setting off and uses 'max_error'. Maximum number of triangles allowed with triangulating the height map, when 'triangulate = TRUE'. Increase this if you encounter problems with 3D performance, want to decrease render time with 'render_highquality()', or need to save a smaller 3D OBJ file to disk with 'save_obj()'.
verbose       Default 'TRUE', if 'interactive()'. Prints information about the mesh triangulation if 'triangulate = TRUE'.
...           Additional arguments to pass to the 'rgl::par3d' function.
Examples

```r
if(interactive()) {
  # Plotting a spherical texture map of the built-in `montereybay` dataset.
  montereybay %>%
    sphere_shade(texture="desert") %>%
    plot_3d(montereybay,zscale=50)
  render_snapshot(clear = TRUE)
}

# With a water layer
montereybay %>%
  sphere_shade(texture="imhof2") %>%
  plot_3d(montereybay, zscale=50, water = TRUE, watercolor="imhof2",
           waterlinecolor="white", waterlinealpha=0.5)
  render_snapshot(clear = TRUE)

# We can also change the base by setting "baseshape" to "hex" or "circle"
montereybay %>%
  sphere_shade(texture="imhof1") %>%
  plot_3d(montereybay, zscale=50, water = TRUE, watercolor="imhof1", theta=-45, zoom=0.7,
           waterlinecolor="white", waterlinealpha=0.5,baseshape="circle")
  render_snapshot(clear = TRUE)

montereybay %>%
  sphere_shade(texture="imhof1") %>%
  plot_3d(montereybay, zscale=50, water = TRUE, watercolor="imhof1", theta=-45, zoom=0.7,
           waterlinecolor="white", waterlinealpha=0.5,baseshape="hex")
  render_snapshot(clear = TRUE)

# Or we can carve out the region of interest ourselves, by setting those entries to NA
# to the elevation map passed into `plot_3d`
# Here, we only include the deep bathymetry data by setting all points greater than -10
# in the copied elevation matrix to NA.
mb_water = montereybay
mb_water[mb_water > -10] = NA

montereybay %>%
  sphere_shade(texture="imhof1") %>%
  plot_3d(mb_water, zscale=50, water = TRUE, watercolor="imhof1", theta=-45,
           waterlinecolor="white", waterlinealpha=0.5)
  render_snapshot(clear = TRUE)
```

Description

Plots a ggplot2 object in 3D by mapping the color or fill aesthetic to elevation.

Currently, this function does not transform lines mapped to color into 3D.

If there are multiple legends/guides due to multiple aesthetics being mapped (e.g. color and shape), the package author recommends that the user pass the order of the guides manually using the ggplot2 function "guides()". Otherwise, the order may change when processing the ggplot2 object and result in a mismatch between the 3D mapping and the underlying plot.

Using the shape aesthetic with more than three groups is not recommended, unless the user passes in custom, solid shapes. By default in ggplot2, only the first three shapes are solid, which is a requirement to be projected into 3D.

Usage

```r
plot_gg(
  ggobj,
  width = 3,
  height = 3,
  height_aes = NULL,
  invert = FALSE,
  shadow_intensity = 0.5,
  units = c("in", "cm", "mm"),
  scale = 150,
  pointcontract = 0.7,
  offset_edges = FALSE,
  preview = FALSE,
  raytrace = TRUE,
  sunangle = 315,
  anglebreaks = seq(30, 40, 0.1),
  multicore = FALSE,
  lambert = TRUE,
  triangulate = TRUE,
  max_error = 0.001,
  max_tri = 0,
  verbose = FALSE,
  reduce_size = NULL,
  save_height_matrix = FALSE,
  save_shadow_matrix = FALSE,
  saved_shadow_matrix = NULL,
  ...
)
```
Arguments

- **ggobj**: ggplot object to projected into 3D.
- **width**: Default ‘3’. Width of ggplot, in ‘units’.
- **height**: Default ‘3’. Height of ggplot, in ‘units’.
- **height_aes**: Default ‘NULL’. Whether the ‘fill’ or ‘color’ aesthetic should be used for height values, which the user can specify by passing either ‘fill’ or ‘color’ to this argument. Automatically detected. If both ‘fill’ and ‘color’ aesthetics are present, then ‘fill’ is default.
- **invert**: Default ‘FALSE’. If ‘TRUE’, the height mapping is inverted.
- **shadow_intensity**: Default ‘0.5’. The intensity of the calculated shadows.
- **units**: Default ‘in’. One of c("in", "cm", "mm").
- **scale**: Default ‘150’. Multiplier for vertical scaling: a higher number increases the height of the 3D transformation.
- **pointcontract**: Default ‘0.7’. This multiplies the size of the points and shrinks them around their center in the 3D surface mapping. Decrease this to reduce color bleed on edges, and set to ‘1’ to turn off entirely. Note: If ‘size’ is passed as an aesthetic to the same geom that is being mapped to elevation, this scaling will not be applied. If ‘alpha’ varies on the variable being mapped, you may want to set this to ‘1’, since the points now have a non-zero width stroke outline (however, mapping ‘alpha’ in the same variable you are projecting to height is probably not a good choice, as the ‘alpha’ variable is ignored when performing the 3D projection).
- **offset_edges**: Default ‘FALSE’. If ‘TRUE’, inserts a small amount of space between polygons for "geom_sf", "geom_tile", "geom_hex", and "geom_polygon" layers. If you pass in a number, the space between polygons will be a line of that width. Note: this feature may end up removing thin polygons from the plot entirely—use with care.
- **preview**: Default ‘FALSE’. If ‘TRUE’, the raytraced 2D ggplot will be displayed on the current device.
- **raytrace**: Default ‘FALSE’. Whether to add a raytraced layer.
- **sunangle**: Default ‘315’ (NW). If raytracing, the angle (in degrees) around the matrix from which the light originates.
- **anglebreaks**: Default ‘seq(30,40,0.1)’. The azimuth angle(s), in degrees, as measured from the horizon from which the light originates.
- **multicore**: Default ‘FALSE’. If raytracing and ‘TRUE’, multiple cores will be used to compute the shadow matrix. By default, this uses all cores available, unless the user has set ‘options("cores")’ in which the multicore option will only use that many cores.
- **lambert**: Default ‘TRUE’. If raytracing, changes the intensity of the light at each point based proportional to the dot product of the ray direction and the surface normal at that point. Zeros out all values directed away from the ray.
plot_gg

triangulate Default ‘FALSE’. Reduce the size of the 3D model by triangulating the height map. Set this to ‘TRUE’ if generating the model is slow, or moving it is choppy. Will also reduce the size of 3D models saved to disk.

max_error Default ‘0.001’. Maximum allowable error when triangulating the height map, when ‘triangulate = TRUE’. Increase this if you encounter problems with 3D performance, want to decrease render time with ‘render_highquality()’, or need to save a smaller 3D OBJ file to disk with ‘save_obj()’.

max_tri Default ‘0’, which turns this setting off and uses ‘max_error’. Maximum number of triangles allowed with triangulating the height map, when ‘triangulate = TRUE’. Increase this if you encounter problems with 3D performance, want to decrease render time with ‘render_highquality()’, or need to save a smaller 3D OBJ file to disk with ‘save_obj()’.

verbose Default ‘TRUE’, if ‘interactive()’. Prints information about the mesh triangulation if ‘triangulate = TRUE’.

reduce_size Default ‘NULL’. A number between ‘0’ and ‘1’ that specifies how much to reduce the resolution of the plot, for faster plotting. By default, this just decreases the size of height map, not the image. If you wish the image to be reduced in resolution as well, pass a numeric vector of size 2.

save_height_matrix Default ‘FALSE’. If ‘TRUE’, the function will return the height matrix used for the ggplot.

save_shadow_matrix Default ‘FALSE’. If ‘TRUE’, the function will return the shadow matrix for use in future updates via the ‘shadow_cache’ argument passed to ‘ray_shade’.

saved_shadow_matrix Default ‘NULL’. A cached shadow matrix (saved by the a previous invocation of ‘plot_gg(..., save_shadow_matrix=TRUE)’ to use instead of raytracing a shadow map each time.

... Additional arguments to be passed to ‘plot_3d()’.

Value

Opens a 3D plot in rgl.

Examples

if(interactive()) {
  library(ggplot2)
  library(viridis)

ggdiamonds = ggplot(diamonds, aes(x, depth)) + 
  stat_density_2d(aes(fill = stat(nlevel)), geom = "polygon", n = 200, bins = 50, contour = TRUE) + 
  facet_wrap(clarity~.) +
  scale_fill_viridis_c(option = "A")
  ## Not run:
  plot_gg(ggdiamonds,multicore = TRUE,width=5,height=5, scale=250, windowsize=c(1400,866),
          zoom = 0.55, phi = 30)
## End(Not run)

# Change the camera angle and take a snapshot:
## Not run:
render_camera(zoom=0.5,theta=-30,phi=30)
render_snapshot(clear = TRUE)
## End(Not run)

# Contours and other lines will automatically be ignored. Here is the volcano dataset:

```r
ggvolcano = volcano %>%
  reshape2::melt() %>%
  ggplot() +
  geom_tile(aes(x=Var1,y=Var2,fill=value)) +
  geom_contour(aes(x=Var1,y=Var2,z=value),color="black") +
  scale_x_continuous("X",expand = c(0,0)) +
  scale_y_continuous("Y",expand = c(0,0)) +
  scale_fill_gradientn("Z",colours = terrain.colors(10)) +
  coord_fixed()

ggvolcano
```
## Not run:
```
plot_gg(ggvolcano, multicore = TRUE, raytrace = TRUE, width = 7, height = 4,
        scale = 300, windowsize = c(1400, 866), zoom = 0.6, phi = 30, theta = 30)
render_snapshot(clear = TRUE)
```
## End(Not run)

# Here, we will create a 3D plot of the mtcars dataset. This automatically detects
# that the user used the `color` aesthetic instead of the `fill`.

```r
mtplot = ggplot(mtcars) +
  geom_point(aes(x=mpg,y=disp,color=cyl)) +
  scale_color_continuous(limits=c(0,8))
```
# Preview how the plot will look by setting `preview = TRUE`: We also adjust the angle of the light.
## Not run:
```
plot_gg(mtplot, width=3.5, sunangle=225, preview = TRUE)
```
## End(Not run)

## Not run:
```
plot_gg(mtplot, width=3.5, multicore = TRUE, windowsize = c(1400,866), sunangle=225,
        zoom = 0.60, phi = 30, theta = 45)
render_snapshot(clear = TRUE)
```
## End(Not run)

# Now let's plot a density plot in 3D.

```r
mtplot_density = ggplot(mtcars) +
  stat_density_2d(aes(x=mpg,y=disp, fill=..density..), geom = "raster", contour = FALSE) +
  scale_x_continuous(expand=c(0,0)) +
  scale_y_continuous(expand=c(0,0)) +
  scale_fill_gradient(low="pink", high="red")
mtplot_density
```
## Not run:
plot_gg(mtplot_density, width = 4, zoom = 0.60, theta = -45, phi = 30, windowsize = c(1400,866))
render_snapshot(clear = TRUE)

## End(Not run)

This also works facetted.
mtplot_density_facet = mtplot_density + facet_wrap(~cyl)

#Preview this plot in 2D:
## Not run:
plot_gg(mtplot_density_facet, preview = TRUE)

## End(Not run)

## Not run:
plot_gg(mtplot_density_facet, windowsize=c(1400,866),
   zoom = 0.55, theta = -10, phi = 25)
render_snapshot(clear = TRUE)

## End(Not run)

That is a little cramped. Specifying a larger width will improve the readability of this plot.
## Not run:
plot_gg(mtplot_density_facet, width = 6, preview = TRUE)

## End(Not run)

That's better. Let's plot it in 3D, and increase the scale.
## Not run:
plot_gg(mtplot_density_facet, width = 6, windowsize=c(1400,866),
   zoom = 0.55, theta = -10, phi = 25, scale=300)
render_snapshot(clear = TRUE)

## End(Not run)

### plot_map

**Plot Map**

**Description**

Displays the map in the current device.

**Usage**

```
plot_map(
  hillshade,
  rotate = 0,
  asp = 1,
  title_text = NA,
  title_offset = c(20, 20),
```

title_color = "black",
title_size = 30,
title_font = "sans",
title_style = "normal",
title_bar_color = NULL,
title_bar_alpha = 0.5,
title_position = "northwest",
keep_user_par = FALSE,
...
)

Arguments

hillshade  Hillshade to be plotted.
asp        Default ‘1’. Aspect ratio of the resulting plot. Use ‘asp = 1/cospi(mean_latitude/180)’
to rescale lat/long at higher latitudes to the correct the aspect ratio.
title_text Default ‘NULL’. Text. Adds a title to the image, using ‘magick::image_annotate()’.
title_offset Default ‘c(20,20)’. 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.
keep_user_par Default ‘TRUE’. Whether to keep the user’s ‘par()’ settings. Set to ‘FALSE’ if
       you want to set up a multi-pane plot (e.g. set ‘par(mfrow)’).

Examples

#Plotting the Monterey Bay dataset with bathymetry data

water_pal = colorRampPalette(c("darkblue", "dodgerblue", "lightblue"))(200)
bathy_hs = height_shade(montereybay, texture = water_pal)

#Set everything below 0m to water palette
montereybay %>%
sphere_shade(zscale=10) %>%
add_overlay(generate_altitude_overlay(bathy_hs, montereybay, 0, 0)) %>%
add_shadow(ray_shade(montereybay,zscale=50),0.3) %>%
plot_map()

#Correcting the aspect ratio for the latitude of Monterey Bay

extent_mb = attr(montereybay,"extent")
mean_latitude = mean(c(extent_mb@ymax,extent_mb@ymin))

montereybay %>%
sphere_shade(zscale=10) %>%
add_overlay(generate_altitude_overlay(bathy_hs, montereybay, 0, 0)) %>%
add_shadow(ray_shade(montereybay,zscale=50),0.3) %>%
plot_map(asp = 1/cospi(mean_latitude/180))

---

raster_to_matrix  Raster to Matrix

**Description**

Turns a raster into a matrix suitable for rayshader.

**Usage**

```r
raster_to_matrix(raster, verbose = interactive())
```

**Arguments**

- `raster`  
The input raster. Either a RasterLayer object, or a filename.
- `verbose`  
Default ‘interactive()’. Will print dimensions of the resulting matrix.

**Examples**

#Save montereybay as a raster and open using the filename.

```r
if("rgdal" %in% rownames(utils::installed.packages())) {
  temp_raster_filename = paste0(tempfile(),".tif")
raster::writeRaster(raster::raster(t(montereybay)),temp_raster_filename)
elmat = raster_to_matrix(temp_raster_filename)
elmat %>%
sphere_shade() %>%
plot_map()
}
```
**Description**

Calculates shadow map for a elevation matrix by propagating rays from each matrix point to the light source(s), lowering the brightness at each point for each ray that intersects the surface.

**Usage**

```r
ray_shade(
  heightmap,  # A two-dimensional matrix, where each entry in the matrix is the elevation at that point. All points are assumed to be evenly spaced.
  sunaltitude = 45,  # Default '45'. The angle, in degrees (as measured from the horizon) from which the light originates. The width of the light is centered on this value and has an angular extent of 0.533 degrees, which is the angular extent of the sun. Use the 'anglebreaks' argument to create a softer (wider) light. This has a hard minimum/maximum of 0/90 degrees.
  sunangle = 315,  # Default '315' (NW). The angle, in degrees, around the matrix from which the light originates. Zero degrees is North, increasing clockwise.
  maxsearch = NULL,  # Defaults to the longest possible shadow given the 'sunaltitude' and 'heightmap'. Otherwise, this argument specifies the maximum distance that the system should propagate rays to check.
  lambert = TRUE,  # Default 'TRUE'. Changes the intensity of the light at each point based proportional to the dot product of the ray direction and the surface normal at that point. Zeros out all values directed away from the ray.
  zscale = 1,  # Default '1'. The ratio between the x and y spacing (which are assumed to be equal) and the z axis. For example, if the elevation is in units of meters and the grid values are separated by 10 meters, 'zscale' would be 10.
  multicore = FALSE,  # If TRUE, uses multiple cores to speed up the computation. If FALSE, uses a single core.
  cache_mask = NULL,  # A mask used to cache the results of ray tracing. If NULL, no caching is used.
  shadow_cache = NULL,  # A cache for storing the shadow values. If NULL, no caching is used.
  progbar = interactive(),  # If TRUE, shows a progress bar during the computation.
  anglebreaks = NULL,  # If not NULL, creates softer (wider) lights by breaking the light into multiple segments.
  ...
)
```

**Arguments**

- **heightmap**: A two-dimensional matrix, where each entry in the matrix is the elevation at that point. All points are assumed to be evenly spaced.
- **sunaltitude**: Default '45'. The angle, in degrees (as measured from the horizon) from which the light originates. The width of the light is centered on this value and has an angular extent of 0.533 degrees, which is the angular extent of the sun. Use the 'anglebreaks' argument to create a softer (wider) light. This has a hard minimum/maximum of 0/90 degrees.
- **sunangle**: Default '315' (NW). The angle, in degrees, around the matrix from which the light originates. Zero degrees is North, increasing clockwise.
- **maxsearch**: Defaults to the longest possible shadow given the 'sunaltitude' and 'heightmap'. Otherwise, this argument specifies the maximum distance that the system should propagate rays to check.
- **lambert**: Default 'TRUE'. Changes the intensity of the light at each point based proportional to the dot product of the ray direction and the surface normal at that point. Zeros out all values directed away from the ray.
- **zscale**: Default '1'. The ratio between the x and y spacing (which are assumed to be equal) and the z axis. For example, if the elevation is in units of meters and the grid values are separated by 10 meters, 'zscale' would be 10.
**multicore**
Default ‘FALSE’. If ‘TRUE’, multiple cores will be used to compute the shadow matrix. By default, this uses all cores available, unless the user has set `options('cores')` in which the multicore option will only use that many cores.

**cache_mask**
Default ‘NULL’. A matrix of 1 and 0s, indicating which points on which the raytracer will operate.

**shadow_cache**
Default ‘NULL’. The shadow matrix to be updated at the points defined by the argument ‘cache_mask’. If present, this will only compute the raytraced shadows for those points with value ‘1’ in the mask.

**progbar**
Default ‘TRUE’ if interactive, ‘FALSE’ otherwise. If ‘FALSE’, turns off progress bar.

**anglebreaks**
Default ‘NULL’. A vector of angle(s) in degrees (as measured from the horizon) specifying from where the light originates. Use this instead of ‘sunaltitude’ to create a softer shadow by specifying a wider light. E.g. ‘anglebreaks = seq(40,50,by=0.5)’ creates a light 10 degrees wide, as opposed to the default ...

Additional arguments to pass to the ‘makeCluster’ function when ‘multicore=TRUE’.

---

**Value**
Matrix of light intensities at each point.

**Examples**

```r
#First we ray trace the Monterey Bay dataset.
#The default angle is from 40-50 degrees azimuth, from the north east.
montereybay %>%
  ray_shade(zscale=50) %>%
  plot_map()

#Change the altitude of the sun to 25 degrees
montereybay %>%
  ray_shade(zscale=50, sunaltitude=25) %>%
  plot_map()

#Remove the lambertian shading to just calculate shadow intensity.
montereybay %>%
  ray_shade(zscale=50, sunaltitude=25, lambert=FALSE) %>%
  plot_map()

#Change the direction of the sun to the South East
montereybay %>%
  ray_shade(zscale=50, sunaltitude=25, sunangle=225) %>%
  plot_map()
```

---
reduce_matrix_size  

Reduce Matrix Size (deprecated)

Description
Reduce Matrix Size (deprecated)

Usage
reduce_matrix_size(...)

Arguments
...  
Arguments to pass to resize_matrix() function.

Value
Reduced matrix.

Examples
#Deprecated lambertian material. Will display a warning.

montbaysmall = reduce_matrix_size(montereybay, scale=0.5)
montbaysmall %>%
sphere_shade() %>%
plot_map()

render_camera  

Render Camera

Description
Changes the position and properties of the camera around the scene. If no values are entered, prints and returns the current values.

Usage
render_camera(theta = NULL, phi = NULL, zoom = NULL, fov = NULL)

Arguments

theta  
Defaults to current value. Rotation angle.

phi  
Defaults to current value. Azimuth angle. Maximum ‘90’.

zoom  
Defaults to current value. Positive value indicating camera magnification.

fov  
Defaults to current value. Field of view of the camera. Maximum ‘180’.
Examples

```r
if(interactive()) {
  # Not run:
  montereybay %>%
    sphere_shade() %>%
    plot_3d(montereybay, zscale = 50, water = TRUE, waterlinecolor = "white")
  render_snapshot()

  # Shift the camera over and add a title
  # Not run:
  render_camera(theta = -45, phi = 45)
  render_snapshot(title_text = "Monterey Bay, CA",
                  title_bar_color = "grey50")

  # Shift to an overhead view (and change the text/title bar color)
  # Not run:
  render_camera(theta = 0, phi = 90, zoom = 0.9)
  render_snapshot(title_text = "Monterey Bay, CA",
                  title_color = "white",
                  title_bar_color = "darkgreen")

  # Shift to a front view and add a vignette effect
  # Not run:
  render_camera(theta = -90, phi = 30, zoom = 0.8)
  render_snapshot(title_text = "Monterey Bay, CA",
                  title_color = "white",
                  title_bar_color = "blue",
                  vignette = TRUE)

  # Change the field of view (fov) and make the title bar opaque.
  # Not run:
  render_camera(theta = -90, phi = 30, zoom = 0.5, fov = 130)
  render_snapshot(title_text = "Monterey Bay, CA",
                  title_color = "black",
                  title_bar_alpha = 1,
                  title_bar_color = "lightblue",
                  vignette = TRUE)

  # Here we render a series of frames to later stitch together into a movie.
  # Not run:
  phivec = 20 + 70 * 1/(1 + exp(seq(-5, 10, length.out = 180)))
}
```
phivecfull = c(phivec, rev(phivec))
thetavec = 270 + 45 * sin(seq(0, 359, length.out = 360) * pi/180)
zoomvechalf = 0.5 + 0.5 * 1/(1 + exp(seq(-5, 10, length.out = 180)))
zoomvec = c(zoomvechalf, rev(zoomvechalf))

for(i in 1:360) {
  render_camera(theta = thetavec[i], phi = phivecfull[i], zoom = zoomvec[i])
  # uncomment the next line to save each frame to the working directory
  # render_snapshot(paste0("frame", i, ".png"))
}

# Run this command in the command line using ffmpeg to stitch together a video:
# ffmpeg -framerate 60 -i frame%d.png -vcodec libx264 raymovie.mp4
# And run this command to convert the video to post to the web:
# ffmpeg -i raymovie.mp4 -pix_fmt yuv420p -profile:v baseline -level 3 -vf scale=-2:-2 rayweb.mp4

# Or we can use render_movie() to do this all automatically with type="custom" (uncomment to run):
# render_movie(filename = tempfile(fileext = ".mp4"), type = "custom",
#  theta = thetavec, phi = phivecfull, zoom = zoomvec, fov=0)
# rgl::rgl.close()

## End(Not run)

---

## render_compass

**Render Compass Symbol**

### Description

Places a compass on the map to specify the North direction.

### Usage

```r
render_compass(
  angle = 0,
  position = "SE",
  x = NULL,
  y = NULL,
  z = NULL,
  compass_radius = NULL,
  scale_distance = 1,
  color_n = "darkred",
  color_arrow = "grey90",
  color_background = "grey60",
  color_bevel = "grey20",
  position_circular = FALSE,
  clear_compass = FALSE
)
```
render_compass

Arguments

angle       Default ‘0’. The direction the arrow should be facing.
position    Default ‘SE’. A string representing a cardinal direction. Ignored if ‘x’, ‘y’, and ‘z’ are manually specified.
x           Default ‘NULL’. X position. If not entered, automatically calculated using ‘position’ argument.
compass_radius Default ‘NULL’. The radius of the compass. If not entered, automatically calculated. Increase or decrease the size of the compass.
scale_distance Default ‘1’. Multiplier that moves the compass away from the center of the map.
color_n     Default ‘darkred’. Color of the letter N.
color_arrow Default ‘grey90’. Color of the arrow.
color_background Default ‘grey20’. Color of the area right under the arrow.
color_bevel Default ‘grey20’. Color of the bevel.
position_circular Default ‘FALSE’. If ‘TRUE’, will place compass at a constant radius away from the map, as opposed to directly next to it. Overridden if user manually specifies position.
clear_compass Default ‘FALSE’. Clears the compass symbol(s) on the map.

Value

Adds compass to map. No return value.

Examples

#Add a North arrow to the map, by default in the bottom right (SE)
if(interactive()) {
  ## Not run:
  montereybay %>%
    sphere_shade() %>%
    plot_3d(montereybay,theta=-45, water=TRUE)
  render_compass()
  render_snapshot()

  #Remove the existing symbol with ‘clear_compass = TRUE’
  render_compass(clear_compass = TRUE)

  #Point the N towards the light, at 315 degrees:
  render_compass(angle = 315)
  render_snapshot()
  render_compass(clear_compass = TRUE)
# We can change the position by specifying a direction (here are three):

```r
render_camera(theta=45,phi=45)
render_compass(position = "NW")
render_compass(position = "E")
render_compass(position = "S")
render_snapshot()
render_compass(clear_compass = TRUE)
```

# We can also change the distance away from the edge by setting the `scale_distance` argument.

```r
render_compass(position = "NW", scale_distance = 1.4)
render_compass(position = "E", scale_distance = 1.4)
render_compass(position = "S", scale_distance = 1.4)
```

# Zoom in slightly:

```r
render_camera(theta=45,phi=45,zoom=0.7)
render_snapshot()
render_compass(clear_compass = TRUE)
```

# We can also specify the radius directly with `compass_radius`:

```r
render_camera(theta=0,phi=45,zoom=1)
render_compass(position = "N", scale_distance = 1.5, compass_radius=200)
render_compass(position = "E", scale_distance = 1.4, compass_radius=50)
render_compass(position = "S", scale_distance = 1.3, compass_radius=25)
render_compass(position = "W", scale_distance = 1.2, compass_radius=10)
render_snapshot()
render_compass(clear_compass = TRUE)
```

# We can also adjust the position manually, by specifying all x, y and z arguments.

```r
render_camera(theta=-45,phi=45,zoom=0.9)
render_compass(x = 150, y = 50, z = 150)
render_snapshot()
```

# Compass support is also included in render_highquality()

```r
render_highquality(clamp_value=10)
render_compass(clear_compass = TRUE)
```

# We can change the colors in the compass, and also set it a constant distance away with `position_circular = TRUE`:

```r
render_camera(theta=0,phi=45,zoom=0.75)
render_compass(position = "N", color_n = "#55967a", color_arrow = "#fff673",
               color_background = "#cfe0a9", color_bevel = "#8fb28a", position_circular = TRUE)
render_compass(position = "NE", color_n = "black", color_arrow = "grey90",
               color_background = "grey50", color_bevel = "grey20", position_circular = TRUE)
render_compass(position = "E", color_n = "red", color_arrow = "blue",
               color_background = "yellow", color_bevel = "purple", position_circular = TRUE)
render_compass(position = "SE", color_n = c(0.7,0.5,0.9), color_arrow = c(0.8,0.8,1),
               color_background = c(0.2,0.2,0.1), color_bevel = c(0.6,0.4,0.6),
               position_circular = TRUE)
render_compass(position = "S", color_n = "#ffe3b3", color_arrow = "#6a463a",
               color_background = "#abaf98", color_bevel = "grey20", position_circular = TRUE)
render_compass(position = "SW", color_n = "#ffe3a3", color_arrow = "#f1c3a9",
```
render_depth

Render Depth of Field

Description

Adds depth of field to the current RGL scene by simulating a synthetic aperture. The size of the circle of confusion is determined by the following formula (\(z_{\text{depth}}\) is from the image’s depth map).

\[
\text{abs}(z_{\text{depth}}-\text{focus}) \times \text{focal_length} \times 2 / (f_{\text{stop}} \times z_{\text{depth}} \times (\text{focus} - \text{focal_length}))
\]

Usage

render_depth(
  focus = 0.5,
  focallength = 100,
  fstop = 4,
  filename = NULL,
  preview_focus = FALSE,
  bokehshape = "circle",
  bokehintensity = 1,
  bokehlimit = 0.8,
  rotation = 0,
  gamma_correction = TRUE,
  aberration = 0,
  transparent_water = FALSE,
  heightmap = NULL,
  zscale = NULL,
  title_text = NULL,
  title_offset = c(20, 20),
  title_color = "black",
  title_size = 30,
  title_font = "sans",
  title_bar_color = NULL,
  title_bar_alpha = 0.5,
  title_position = "northwest",
  image_overlay = NULL,
render_depth

vignette = FALSE,
progbar = interactive(),
instant_capture = interactive(),
clear = FALSE,
bring_to_front = FALSE,
...
)

Arguments

focus Defaults '0.5'. Depth in which to blur. Minimum 0, maximum 1.
focallength Default '1'. Focal length of the virtual camera.
fstop Default '1'. F-stop of the virtual camera.
filename The filename of the image to be saved. If this is not given, the image will be plotted instead.
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.
bokehintensity Default '3'. 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 hexagon bokeh shape.
gamma_correction Default 'TRUE'. Controls gamma correction when adding colors. Default exponent of 2.2.
aberration Default '0'. Adds chromatic aberration to the image. Maximum of '1'.
transparent_water Default 'FALSE'. If 'TRUE', depth is determined without water layer. User will have to re-render the water layer with 'render_water()' if they want to recreate the water layer.
heightmap Default 'NULL'. The height matrix for the scene. Passing this will allow 'render_depth()' to automatically redraw the water layer if 'transparent_water = TRUE'.
zsclae Default 'NULL'. The zscale value for the heightmap. Passing this will allow 'render_depth()' to automatically redraw the water layer if 'transparent_water = TRUE'.
title_text Default 'NULL'. Text. Adds a title to the image, using magick::image_annotate.
title_offset Default 'c(20,20)'. 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".
render_depth

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_position

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.

vignette
Default ‘FALSE’. If ‘TRUE’ or numeric, 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.

progbar
Default ‘TRUE’ if in an interactive session. Displays a progress bar.

instant_capture
Default ‘TRUE’ if interactive, ‘FALSE’ otherwise. If ‘FALSE’, a slight delay is added before taking the snapshot. This can help stop prevent rendering issues when running scripts.

clear
Default ‘FALSE’. If ‘TRUE’, the current ‘rgl’ device will be cleared.

bring_to_front
Default ‘FALSE’. Whether to bring the window to the front when rendering the snapshot.

... Additional parameters to pass to magick::image_annotate.

Value
4-layer RGBA array.

Examples

#Only run these examples if the webshot2
if(interactive()) {
  montreybay %>%
  sphere_shade() %>%
  plot_3d(montreybay,zscale=50, water=TRUE, waterlinecolor="white",
      zoom=0.3,theta=-135,fov=70, phi=20)

  #Preview where the focal plane lies
  render_depth(focus=0.75, preview_focus=TRUE)

  #Render the depth of field effect
  render_depth(focus=0.75, focallength = 100)

  #Add a chromatic aberration effect
  render_depth(focus=0.75, focallength = 100, aberration = 0.3)

  #Render the depth of field effect, ignoring water and re-drawing the waterlayer
  render_depth(focus=0.9, preview_focus=TRUE,
      heightmap = montreybay, zscale=50, transparent_water=TRUE)
render_depth(focus=0.9, heightmap = montereybay, zscale=50, transparent_water=TRUE)
grl::rgl.close()

montereybay %>%
sphere_shade() %>%
plot_3d(montereybay,zscale=50, water=TRUE, waterlinecolor="white",
        zoom=0.7,phi=30,fov=60,theta=-90)

render_camera(theta=45,zoom=0.15,phi=20)

#Change the bokeh shape and intensity
render_depth(focus=0.7, bokehshape = "circle",focallength=300,bokehintensity=30,
        title_text = "Circular Bokeh", title_size = 30, title_color = "white",
        title_bar_color = "black")
render_depth(focus=0.7, bokehshape = "hex",focallength=300,bokehintensity=30,
        title_text = "Hexagonal Bokeh", title_size = 30, title_color = "white",
        title_bar_color = "black")

#Add a title and vignette effect.
render_camera(theta=0,zoom=0.7,phi=30)
render_depth(focus = 0.75,focallength = 100, title_text = "Monterey Bay, CA",
        title_size = 20, title_color = "white", title_bar_color = "black", vignette = TRUE)

#
grl::rgl.close()
width = NULL,
height = NULL,
text_angle = NULL,
text_size = 6,
text_offset = c(0, 0, 0),
line_radius = 0.5,
point_radius = 0.5,
smooth_line = FALSE,
scale_text_angle = NULL,
scale_text_size = 6,
scale_text_offset = c(0, 0, 0),
title_text = NULL,
title_offset = c(20, 20),
title_color = "black",
title_size = 30,
title_font = "sans",
title_bar_color = NULL,
title_bar_alpha = 0.5,
ground_material = rayrender::diffuse(),
ground_size = 1e+05,
scene_elements = NULL,
camera_location = NULL,
camera_lookat = NULL,
camera_interpolate = 1,
clear = FALSE,
print_scene_info = FALSE,
clamp_value = 10,

Arguments

filename Filename of saved image. If missing, will display to current device.
light Default ‘TRUE’. Whether there should be a light in the scene. If not, the scene will be lit with a bluish sky.
lightdirection Default ‘315’. Position of the light angle around the scene. If this is a vector longer than one, multiple lights will be generated (using values from ‘lightaltitude’, ‘lightintensity’, and ‘lightcolor’)
lightaltitude Default ‘45’. Angle above the horizon that the light is located. If this is a vector longer than one, multiple lights will be generated (using values from ‘lightdirection’, ‘lightintensity’, and ‘lightcolor’)
lightsize Default ‘NULL’. Radius of the light(s). Automatically chosen, but can be set here by the user.
lightintensity Default ‘500’. Intensity of the light.
lightcolor Default ‘white’. The color of the light.
obj_material Default ‘rayrender::diffuse()’. The material properties of the object file.
cache_filename  Name of temporary filename to store OBJ file, if the user does not want to rewrite the file each time.
width  Defaults to the width of the rgl window. Width of the rendering.
height  Defaults to the height of the rgl window. Height of the rendering.
text_angle  Default ‘NULL’, which forces the text always to face the camera. If a single angle (degrees), will specify the absolute angle all the labels are facing. If three angles, this will specify all three orientations (relative to the x,y, and z axes) of the text labels.
text_size  Default ‘6’. Height of the text.
text_offset  Default ‘c(0,0,0)’. Offset to be applied to all text labels.
line_radius  Default ‘0.5’. Radius of line/path segments.
point_radius  Default ‘0.5’. Radius of 3D points (rendered with ‘render_points()’).
smooth_line  Default ‘FALSE’. If ‘TRUE’, the line will be rendered with a continuous smooth line, rather than straight segments.
scale_text_angle  Default ‘NULL’. Same as ‘text_angle’, but for the scale bar.
scale_text_size  Default ‘6’. Height of the scale bar text.
scale_text_offset  Default ‘c(0,0,0)’. Offset to be applied to all scale bar text labels.
title_text  Default ‘NULL’. Text. Adds a title to the image, using magick::image_annotate.
title_offset  Default ‘c(20,20)’. 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_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.
ground_material  Default ‘diffuse()’. Material defined by the rayrender material functions.
ground_size  Default ‘100000’. The width of the plane representing the ground.
scene_elements  Default ‘NULL’. Extra scene elements to add to the scene, created with rayrender.
camera_location  Default ‘NULL’. Custom position of the camera. The ‘FOV’, ‘width’, and ‘height’ arguments will still be derived from the rgl window.
camera_lookat  Default ‘NULL’. Custom point at which the camera is directed. The ‘FOV’, ‘width’, and ‘height’ arguments will still be derived from the rgl window.
render_highquality

camera_interpolate
  Default ‘c(0,0)’. Maximum ‘1’, minimum ‘0’. Sets the camera at a point be-
  tween the ‘rgl’ view and the ‘camera_location’ and ‘camera_lookat’ vectors.

clear
  Default ‘FALSE’. If ‘TRUE’, the current ‘rgl’ device will be cleared.

print_scene_info
  Default ‘FALSE’. If ‘TRUE’, it will print the position and lookat point of the
  camera.

clamp_value
  Default ‘10’. See documentation for ‘rayrender::render_scene()’.

... Additional parameters to pass to ‘rayrender::render_scene()’

Examples

#Render the volcano dataset using pathtracing
if(interactive()) {
  volcano %>%
  sphere_shade() %>%
  plot_3d(volcano,zscale = 2)
  render_highquality()

#Change position of light
render_highquality(lightdirection = 45)

#Change vertical position of light
render_highquality(lightdirection = 45, lightaltitude=10)

#Change the ground material
render_highquality(lightdirection = 45, lightaltitude=60, 
  ground_material = rayrender::diffuse(checkerperiod = 30, checkercolor="grey50"))

#Add three different color lights and a title
render_highquality(lightdirection = c(0,120,240), lightaltitude=45, 
  lightcolor=c("red","green","blue"), title_text = "Red, Green, Blue", 
  title_bar_color="white", title_bar_alpha=0.8)

#Change the camera:
render_camera(theta=-45,phi=60,fov=60,zoom=0.8)
render_highquality(lightdirection = c(0),
  title_bar_color="white", title_bar_alpha=0.8)

#Add a shiny metal sphere
render_camera(theta=-45, phi=60, fov=60, zoom=0.8)
render_highquality(lightdirection = c(0,120,240), lightaltitude=45,
  lightcolor=c("red","green","blue"),
  scene_elements = rayrender::sphere(z=-60, y=0, radius=20, material=rayrender::metal()))

# Add a red light to the volcano and change the ambient light to dusk
render_camera(theta=45, phi=45)
render_highquality(lightdirection = c(240), lightaltitude=30,
  lightcolor=c("#5555ff"),
  scene_elements = rayrender::sphere(z=0, y=15, x=-18, radius=5,
    material=rayrender::light(color="red", intensity=10)))

# Manually change the camera location and direction
render_camera(theta=45, phi=45, fov=90)
render_highquality(lightdirection = c(240), lightaltitude=30, lightcolor=c("#5555ff"),
  camera_location = c(50,10,10), camera_lookat = c(0,15,0),
  scene_elements = rayrender::sphere(z=0, y=15, x=-18, radius=5,
    material=rayrender::light(color="red", intensity=10)))
rl::rl.close()

render_label

render_label

Render Label

Description

Adds a marker and label to the current 3D plot

Usage

render_label(
  heightmap, text, lat, long, altitude = NULL, extent = NULL, x = NULL, y = NULL, z = NULL, zscale = 1, relativez = TRUE, offset = 0,
clear_previous = FALSE,
textsize = 1,
dashed = FALSE,
dashlength = "auto",
linewidth = 3,
antialias = FALSE,
alpha = 1,
textalpha = 1,
freetype = TRUE,
adjustvec = NULL,
family = "sans",
fonttype = "standard",
linecolor = "black",
textcolor = "black"
)

Arguments

heightmap A two-dimensional matrix, where each entry in the matrix is the elevation at that point. All points are assumed to be evenly spaced.
text The label text.
lat A latitude for the text. Must provide an ‘raster::extent’ object to argument ‘extent’ for the map.
long A latitude for the text. Must provide an ‘raster::extent’ object to argument ‘extent’ for the map.
altitude Default ‘NULL’. Elevation of the label, in units of the elevation matrix (scaled by zscale). If none is passed, this will default to 10 percent above the maximum altitude in the heightmap.
extent Default ‘NULL’. A ‘raster::Extent’ object with the bounding box of the displayed 3D scene.
x Default ‘NULL’. Directly specify the ‘x’ index in the matrix to place the label.
y Default ‘NULL’. Directly specify the ‘y’ index in the matrix to place the label.
z Default ‘NULL’. Elevation of the label, in units of the elevation matrix (scaled by zscale).
zscale Default ‘1’. The ratio between the x and y spacing (which are assumed to be equal) and the z axis. For example, if the elevation levels are in units relativez Default ‘TRUE’. Whether ‘z’ should be measured in relation to the underlying elevation at that point in the heightmap, or set absolutely (‘FALSE’).
offset Elevation above the surface (at the label point) to start drawing the line.
clear_previous Default ‘FALSE’. If ‘TRUE’, it will clear all existing text and lines rendered with ‘render_label()’. If no other arguments are passed to ‘render_label()’, this will just remove all existing lines.
textsize Default ‘1’. A numeric character expansion value.
dashed Default ‘FALSE’. If ‘TRUE’, the label line is dashed.
dashlength  Default 'auto'. Length, in units of the elevation matrix (scaled by 'zscale') of the dashes if 'dashed = TRUE'.

linewidth  Default '3'. The line width.

antialias  Default 'FALSE'. If 'TRUE', the line with be have anti-aliasing applied. NOTE: anti-aliasing can cause some unpredictable behavior with transparent surfaces.

alpha  Default '1'. Transparency of the label line.

textalpha  Default '1'. Transparency of the label text.

freetype  Default 'TRUE'. Set to 'FALSE' if freetype is not installed (freetype enables anti-aliased fonts). NOTE: There are occasionally transparency issues when positioning FreeType fonts in front and behind a transparent surface.

adjustvec  Default 'c(0.5,-0.5)'. The horizontal and vertical offset for the text. If 'freetype = FALSE' and on macOS/Linux, this is adjusted to 'c(0.33,-0.5)' to keep the type centered.

family  Default '"sans"'. Font family. Choices are 'c("serif", "sans", "mono", "symbol")'.

fonttype  Default '"standard"'. The font type. Choices are 'c("standard", "bold", "italic", "bolditalic")'. NOTE: These require FreeType fonts, which may not be installed on your system. See the documentation for rgl::text3d() for more information.

linecolor  Default 'black'. Color of the line.

textcolor  Default 'black'. Color of the text.

Examples

if(interactive()) {
  ## Not run:
  montereybay %>%
  sphere_shade() %>%
  plot_3d(montereybay,zscale=50,water=TRUE, watercolor="#233aa1")
  render_snapshot()

  ## End(Not run)

  santa_cruz = c(36.962957, -122.021033)
  # We want to add a label to Santa Cruz, so we use the x and y matrix coordinate (x=220 and y=330)
  ## Not run:
  render_label(montereybay, lat = santa_cruz[1], long = santa_cruz[2],
               extent = attr(montereybay, "extent"),
               altitude=12000, zscale=50, text = "Santa Cruz")
  render_snapshot()

  ## End(Not run)

  monterey = c(36.603053, -121.892933)
  # We can also change the linetype to dashed by setting 'dashed = TRUE' (additional options allow the user to control the dash length). You can clear the existing lines by setting '
clear_previous = TRUE'.
  ## Not run:
  render_label(montereybay, lat = monterey[1], long = monterey[2], altitude = 10000,
               clear_previous = TRUE)

  ## End(Not run)
}
canyon = c(36.621049, -122.333912)
# By default, z specifies the altitude above that point on the elevation matrix. We can also specify
# an absolute height by setting `relativez=FALSE`.
## Not run:
render_label(montereybay, lat = canyon[1], long = canyon[2], altitude = 2000,
            extent = attr(montereybay, "extent"),
            zscale = 50, text = "Monterey Canyon", relativez = FALSE)
render_snapshot()
## End(Not run)

# We can also render labels in high quality with `render_highquality()`, specifying a custom
# line radius. By default, the labels point towards the camera, but you can fix their angle with
# argument "text_angle".
## Not run:
render_camera(theta = 35, phi = 35, zoom = 0.80, fov = 60)
render_label(montereybay, lat = monterey[1], long = monterey[2], altitude = 10000,
            extent = attr(montereybay, "extent"),
            zscale = 50, text = "Monterey", textcolor = "white", linecolor = "darkred",
            dashed = TRUE, clear_previous = TRUE)
render_label(montereybay, lat = canyon[1], long = canyon[2], altitude = 2000, zscale = 50,
            text = "Monterey Canyon", relativez = FALSE)
render_highquality(samples = 200, text_size = 24, line_radius = 2, text_offset = c(0, 20, 0),
                    lightdirection = 180, clamp_value = 10)
# Fixed text angle
render_highquality(samples = 200, text_size = 24, line_radius = 2, text_offset = c(0, 20, 0),
                    lightdirection = 180, text_angle = 0, clamp_value = 10)
## End(Not run)
# We can remove all existing labels by calling `render_label(clear_previous = TRUE)`
## Not run:
render_label(clear_previous = TRUE)
render_snapshot()
rgl::rgl.close()
## End(Not run)
render_movie

Description

Renders a movie using the av package. Moves the camera around a 3D visualization using either a standard orbit, or accepts vectors listing user-defined values for each camera parameter. If the latter, the values must be equal in length to ‘frames’ (or of length ‘1’, in which the value will be fixed).

Usage

render_movie(
  filename,
  type = "orbit",
  frames = 360,
  fps = 30,
  phi = 30,
  theta = 0,
  zoom = NULL,
  fov = NULL,
  title_text = NULL,
  title_offset = c(20, 20),
  title_color = "black",
  title_size = 30,
  title_font = "sans",
  title_bar_color = NULL,
  title_bar_alpha = 0.5,
  image_overlay = NULL,
  vignette = FALSE,
  title_position = "northwest",
  audio = NULL,
  progresbar = interactive(),
  ...
)

Arguments

filename  Filename. If not appended with ‘.mp4’, it will be appended automatically.
type  Default ‘orbit’, which orbits the 3D object at the user-set camera settings ‘phi’, ‘zoom’, and ‘fov’. Other options are ‘oscillate’ (sine wave around ‘theta’ value, covering 90 degrees), or ‘custom’ (which uses the values from the ‘theta’, ‘phi’, ‘zoom’, and ‘fov’ vectors passed in by the user).
frames  Default ‘360’. Number of frames to render.
fps  Default ‘30’. Frames per second. Recommend either 30 or 60 for web.
phi  Defaults to current view. Azimuth values, in degrees.
theta  Default to current view. Theta values, in degrees.
zoom  Defaults to the current view. Zoom value, between ‘0’ and ‘1’.
fov  Defaults to the current view. Field of view values, in degrees.
title_text  Default ‘NULL’. Text. Adds a title to the movie, using magick::image_annotate.
title_offset Default 'c(20,20)'. 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_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.

image_overlay Default 'NULL'. Either a string indicating the location of a png image to overlay over the whole movie (transparency included), or a 4-layer RGBA array. This image will be resized to the dimension of the movie if it does not match exactly.

vignette Default 'FALSE'. If 'TRUE' or numeric, 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.

title_position Default 'northwest'. Position of the title.

audio Default 'NULL'. Optional file with audio to add to the video.

progressbar Default 'TRUE' if interactive, 'FALSE' otherwise. If 'FALSE', turns off progress bar. Will display a progress bar when adding an overlay or title.

... Additional parameters to pass to magick::image_annotate.

Examples

if(interactive()) {
  filename_movie = tempfile()

  #By default, the function produces a 12 second orbit at 30 frames per second, at 30 degrees azimuth.
  montereybay %>%
    sphere_shade(texture="imhof1") %>%
    plot_3d(montereybay, zscale=50, water = TRUE, watercolor="imhof1",
    waterlinecolor="white", waterlinealpha=0.5)
  #Un-comment the following to run:
  #render_movie(filename = filename_movie)
  filename_movie = tempfile()

  #You can change to an oscillating orbit. The magnification is increased and azimuth angle set to 30. A title has also been added using the title_text argument.
  montereybay %>%
    sphere_shade(texture="imhof1") %>%
    plot_3d(montereybay, zscale=50, water = TRUE, watercolor="imhof1",
    waterlinecolor="white", waterlinealpha=0.5)
  #Un-comment the following to run:
  #render_movie(filename = filename_movie, type = "oscillate",}
# frames = 60, phi = 30, zoom = 0.8, theta = -90,
# title_text = "Monterey Bay: Oscillating"

filename_movie = tempfile()

# Finally, you can pass your own set of values to the 
# camera parameters as a vector with type = "custom".
phivechalf = 30 + 60 * 1/(1 + exp(seq(-7, 20, length.out = 180)/2))
phivecfull = c(phivechalf, rev(phivechalf))
thetavec = -90 + 45 * sin(seq(0,359,length.out = 360) * pi/180)
zoomvec = 0.45 + 0.2 * 1/(1 + exp(seq(-5, 20, length.out = 180)))
zoomvecfull = c(zoomvec, rev(zoomvec))

montereybay %>%
  sphere_shade(texture="imhof1") %>%
  plot_3d(montereybay, zscale=50, water = TRUE, watercolor="imhof1",
          waterlinecolor="white", waterlinealpha=0.5)
  
  # Un-comment the following to run
  # render_movie(filename = filename_movie, type = "custom",
  # frames = 360, phi = phivecfull, zoom = zoomvecfull, theta = thetavec)
  
  rgl::rgl.close()

---

render_path

**Render Path**

**Description**

Adds a 3D path to the current scene, using latitude/longitude or coordinates in the reference system defined by the extent object. If no altitude is provided, the path will be elevated a constant offset above the heightmap. If the path goes off the edge, the nearest height on the heightmap will be used.

**Usage**

```r
render_path(
  extent = NULL,
  lat,
  long = NULL,
  altitude = NULL,
  zscale = 1,
  heightmap = NULL,
  linewidth = 3,
  color = "black",
  antialias = FALSE,
  offset = 5,
  clear_previous = FALSE
)
```
Arguments

extent  A `raster::Extent` object with the bounding box of the displayed 3D scene.
lat  Vector of latitudes (or other coordinate in the same coordinate reference system
as extent). Can also be an `sf` or `SpatialLineDataFrame` object.
long  Default `NULL`. Vector of longitudes (or other coordinate in the same coordinate reference system as extent). Ignored if lat is an `sf` or `SpatialLineDataFrame` object.
altitude  Default `NULL`. Elevation of each point, in units of the elevation matrix (scaled by zscl). If left `NULL`, this will be just the elevation value at the surface, offset by `offset`.
zscl  Default `1`. The ratio between the x and y spacing (which are assumed to be equal) and the z axis in the original heightmap.
heightmap  Default `NULL`. Automatically extracted from the rgl window—only use if auto-extraction of matrix extent isn’t working. A two-dimensional matrix, where each entry in the matrix is the elevation at that point. All points are assumed to be evenly spaced.
linewidth  Default `3`. The line width.
color  Default `black`. Color of the line.
antialias  Default `FALSE`. If `TRUE`, the line will have anti-aliasing applied. NOTE: anti-aliasing can cause some unpredictable behavior with transparent surfaces.
offset  Default `5`. Offset of the track from the surface, if `altitude = NULL`.
clear_previous  Default `FALSE`. If `TRUE`, it will clear all existing paths.

Examples

if(interactive()) {

#Starting at Moss Landing in Monterey Bay, we are going to simulate a flight of a bird going
#out to sea and diving for food.

#First, create simulated lat/long data
set.seed(2009)
moss_landing_coord = c(36.806807, -121.793332)
x_vel_out = -0.001 + rnorm(1000)[1:300]/1000
y_vel_out = rnorm(1000)[1:300]/200
z_out = c(seq(0,2000,length.out = 180), seq(2000,0,length.out=10),
          seq(0,2000,length.out = 100), seq(2000,0,length.out=10))

bird_track_lat = list()
bird_track_long = list()
bird_track_lat[[1]] = moss_landing_coord[1]
bird_track_long[[1]] = moss_landing_coord[2]
for(i in 2:300) {
bird_track_lat[[i]] = bird_track_lat[[i-1]] + y_vel_out[i]
bird_track_long[[i]] = bird_track_long[[i-1]] + x_vel_out[i]
}
#Render the 3D map
montereybay %>%
sphere_shade() %>%
plot_3d(montereybay, zscale=50, water=TRUE,
       shadowcolor="#40310a", watercolor="#233aa1", background = "tan",
       theta=210, phi=22, zoom=0.20, fov=55)

#Pass in the extent of the underlying raster (stored in an attribute for the montereybay
#dataset) and the latitudes, longitudes, and altitudes of the track.
render_path(extent = attr(montereybay, "extent"),
            lat = unlist(bird_track_lat), long = unlist(bird_track_long),
            altitude = z_out, zscale=50, color="white", antialias=TRUE)
render_snapshot()

#We'll set the altitude to right above the water to give the tracks a "shadow".
render_path(extent = attr(montereybay, "extent"),
            lat = unlist(bird_track_lat), long = unlist(bird_track_long),
            altitude = 10, zscale=50, color="black", antialias=TRUE)
render_camera(theta=30, phi=35, zoom=0.45, fov=70)
render_snapshot()
#Remove the path:
render_path(clear_previous=TRUE)

#Finally, we can also plot just GPS coordinates offset from the surface by leaving altitude `NULL`
# Here we plot a spiral of values surrounding Moss Landing. This requires the original heightmap.
t = seq(0, 2*pi, length.out=1000)
circle_coords_lat = moss_landing_coord[1] + 0.5 * t/8 * sin(t*6)
circle_coords_long = moss_landing_coord[2] + 0.5 * t/8 * cos(t*6)
render_path(extent = attr(montereybay, "extent"), heightmap = montereybay,
            lat = unlist(circle_coords_lat), long = unlist(circle_coords_long),
            zscale=50, color="red", antialias=TRUE, offset=100, linewidth=5)
render_camera(theta = 160, phi=33, zoom=0.4, fov=55)
render_snapshot()

#And all of these work with `render_highquality()`
render_highquality(clamp_value=10, line_radius=3)
rgl::rgl.close()

render_points

Render Points

Description

Adds 3D datapoints to the current scene, using latitude/longitude or coordinates in the reference
system defined by the extent object. If no altitude is provided, the points will be elevated a constant
offset above the heightmap. If the points goes off the edge, the nearest height on the heightmap will
be used.
Usage

render_points(
  extent = NULL,
  lat = NULL,
  long = NULL,
  altitude = NULL,
  zscale = 1,
  heightmap = NULL,
  size = 3,
  color = "black",
  offset = 5,
  clear_previous = FALSE
)

Arguments

extent A ‘raster::Extent’ object with the bounding box of the displayed 3D scene.
lat Vector of latitudes (or other coordinate in the same coordinate reference system as extent).
long Vector of longitudes (or other coordinate in the same coordinate reference system as extent).
altitude Elevation of each point, in units of the elevation matrix (scaled by zscale).
zscale Default ‘1’. The ratio between the x and y spacing (which are assumed to be equal) and the z axis in the original heightmap.
heightmap Default ‘NULL’. Automatically extracted from the rgl window–only use if auto-extraction of matrix extent isn’t working. A two-dimensional matrix, where each entry in the matrix is the elevation at that point. All points are assumed to be evenly spaced.
size Default ‘3’. The point size.
color Default ‘black’. Color of the point.
offset Default ‘5’. Offset of the track from the surface, if ‘altitude = NULL’.
clear_previous Default ‘FALSE’. If ‘TRUE’, it will clear all existing points.

Examples

if(interactive()) {

  #Starting at Moss Landing in Monterey Bay, we are going to simulate a flight of a bird going
  #out to sea and diving for food.

  #First, create simulated lat/long data
  set.seed(2009)
  moss_landing_coord = c(36.806807, -121.793332)
  x_vel_out = -0.001 + rnorm(1000)[1:300]/1000
  y_vel_out = rnorm(1000)[1:300]/200
  z_out = c(seq(0,2000,length.out = 180), seq(2000,0,length.out=10),
            seq(0,2000,length.out = 100), seq(2000,0,length.out=10))

  render_points(extent = moss_landing_coord, lat = x_vel_out, long = y_vel_out, altitude = z_out, color = "red")
}


bird_track_lat = list()
bird_track_long = list()
bird_track_lat[[1]] = moss_landing_coord[1]
bird_track_long[[1]] = moss_landing_coord[2]
for(i in 2:300) {
bird_track_lat[[i]] = bird_track_lat[[i-1]] + y_vel_out[i]
bird_track_long[[i]] = bird_track_long[[i-1]] + x_vel_out[i]
}

# Render the 3D map
montereybay %>%
sphere_shade() %>%
plot_3d(montereybay,zscale=50,water=TRUE,
  shadowcolor="#40310a", background = "tan",
  theta=210, phi=22, zoom=0.20, fov=55)

# Pass in the extent of the underlying raster (stored in an attribute for the montereybay
dataset) and the latitudes, longitudes, and altitudes of the track.
render_points(extent = attr(montereybay,"extent"),
  lat = unlist(bird_track_lat), long = unlist(bird_track_long),
  altitude = z_out, zscale=50,color="white")
render_snapshot()

# We'll set the altitude to zero to give the tracks a "shadow" over the water.
render_points(extent = attr(montereybay,"extent"),
  lat = unlist(bird_track_lat), long = unlist(bird_track_long),
  altitude = 0, zscale=50, color="black")
render_camera(theta=30,phi=35,zoom=0.45,fov=70)
render_snapshot()

# Remove the points:
render_points(clear_previous=TRUE)

# Finally, we can also plot just GPS coordinates offset from the surface by leaving altitude 'NULL'
# Here we plot a circle of values surrounding Moss Landing. This requires the original heightmap.
t = seq(0,2*pi,length.out=100)
circle_coords_lat = moss_landing_coord[1] + 0.3 * sin(t)
circle_coords_long = moss_landing_coord[2] + 0.3 * cos(t)
render_points(extent = attr(montereybay,"extent"), heightmap = montereybay,
  lat = unlist(circle_coords_lat), long = unlist(circle_coords_long),
  zscale=50, color="red", offset=100, size=5)
render_camera(theta = 160, phi=33, zoom=0.4, fov=55)
render_snapshot()

# And all of these work with 'render_highquality()'
render_highquality(point_radius = 3, clamp_value=10)
rgl::rgl.close()
render_polygons  Render Polygons

Description

Adds 3D polygons to the current scene, using latitude/longitude or coordinates in the reference system defined by the extent object.

Usage

render_polygons(
  polygon,
  extent,
  color = "red",
  top = 1,
  bottom = NA,
  data_column_top = NULL,
  data_column_bottom = NULL,
  heightmap = NULL,
  scale_data = 1,
  parallel = FALSE,
  holes = 0,
  lit = TRUE,
  light_altitude = c(45, 30),
  light_direction = c(315, 135),
  light_intensity = 0.3,
  clear_previous = FALSE
)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>polygon</td>
<td>‘sf’ object, &quot;SpatialPolygon&quot; ‘sp’ object, or xy coordinates of polygon represented in a way that can be processed by ‘xy.coords()’. If xy-coordinate based polygons are open, they will be closed by adding an edge from the last point to the first.</td>
</tr>
<tr>
<td>extent</td>
<td>A ‘raster::Extent’ object with the bounding box for the height map used to generate the original map.</td>
</tr>
<tr>
<td>color</td>
<td>Default ‘black’. Color of the polygon.</td>
</tr>
<tr>
<td>top</td>
<td>Default ‘1’. Extruded top distance. If this equals ‘bottom’, the polygon will not be extruded and just the one side will be rendered.</td>
</tr>
<tr>
<td>bottom</td>
<td>Default ‘0’. Extruded bottom distance. If this equals ‘top’, the polygon will not be extruded and just the one side will be rendered.</td>
</tr>
<tr>
<td>data_column_top</td>
<td>Default ‘NULL’. A string indicating the column in the ‘sf’ object to use to specify the top of the extruded polygon.</td>
</tr>
</tbody>
</table>
render_polygons

data_column_bottom
Default ‘NULL’. A string indicating the column in the ‘sf’ object to use to specify the bottom of the extruded polygon.

heightmap
Default ‘NULL’. Automatically extracted from the rgl window—only use if auto-extraction of matrix extent isn’t working. A two-dimensional matrix, where each entry in the matrix is the elevation at that point. All points are assumed to be evenly spaced.

scale_data
Default ‘1’. If specifying ‘data_column_top’ or ‘data_column_bottom’, how much to scale that value when rendering.

parallel
Default ‘FALSE’. If ‘TRUE’, polygons will be extruded in parallel, which may be faster (depending on how many geometries are in ‘polygon’).

holes
Default ‘0’. If passing in a polygon directly, this specifies which index represents the holes in the polygon. See the ‘earcut’ function in the ‘decido’ package for more information.

lit
Default ‘TRUE’. Whether to light the polygons.

light_altitude
Default ‘c(45, 60)’. Degree(s) from the horizon from which to light the polygons.

light_direction
Default ‘c(45, 60)’. Degree(s) from north from which to light the polygons.

light_intensity
Default ‘0.3’. Intensity of the specular highlight on the polygons.

clear_previous
Default ‘FALSE’. If ‘TRUE’, it will clear all existing polygons.

Examples

if(interactive()) {

# Render the county borders as polygons in Monterey Bay
montereybay %>%
  sphere_shade(texture = "desert") %>%
  add_shadow(ray_shade(montereybay, zscale=50)) %>%
  plot_3d(montereybay, water=TRUE, windowsize=800, watercolor="dodgerblue")
render_camera(theta=140, phi=55, zoom = 0.85, fov=30)

# We will apply a negative buffer to create space between adjacent polygons:
mont_county_buff = sf::st_simplify(sf::st_buffer(monterey_counties_sf, -0.003), dTolerance=0.001)
render_polygons(mont_county_buff,
  extent = attr(montereybay,"extent"), top=10,
  parallel=TRUE)
render_snapshot()

# We can specify the bottom of the polygons as well. Here I float the polygons above the surface
# by specifying the bottom argument. We clear the previous polygons with ‘clear_previous = TRUE’.
render_camera(theta=-60, phi=20, zoom = 0.85, fov=0)
render_polygons(mont_county_buff,
  extent = attr(montereybay,"extent"), bottom = 190, top=200,
  parallel=TRUE, clear_previous=TRUE)
}
render_snapshot()

# We can set the height of the data to a column in the sf object: we'll use the land area.
# We'll have to scale this value because it's max value is 2.6 billion:
render_camera(theta=-60, phi=60, zoom = 0.85, fov=30)
render_polygons(mont_county_buff,
    extent = attr(montereybay,"extent"), data_column_top = "ALAND",
    scale_data = 300/(2.6E9), color="chartreuse4",
    parallel=TRUE,clear_previous=TRUE)
render_snapshot()

# This function also works with `render_highquality()`
render_highquality(samples=400, clamp_value=10)
rgl::rgl.close()

render_scalebar

Description
Places a compass on the map to specify the North direction.

Usage
render_scalebar(
    limits,
    position = "W",
    y = NULL,
    segments = 10,
    scale_length = 1,
    label_unit = "",
    offset = NULL,
    radius = NULL,
    color_first = "darkred",
    color_second = "grey80",
    color_text = "black",
    text_switch_side = FALSE,
    text_x_offset = 0,
    text_y_offset = 0,
    text_z_offset = 0,
    clear_scalebar = FALSE)

Arguments
limits The distance represented by the scale bar. If a numeric vector greater than length
1, this will specify the breaks along the scale bar to place labels, with the maximum
value in limits assumed to be the last label. Must be non-negative.
render_scalebar

position Default 'W'. A string representing a direction. Can be 'N', 'E', 'S', and 'W'.
y Default 'NULL'. The height of the scale bar, automatically calculated if 'NULL'.
segments Default '10'. Number of colored segments in the scalebar.
scale_length Default '1'. Length of the scale bar, relative to the side of the map specified in 'position'. If a length-2 vector, the first number specifies the start and stop points along the side.
lable_unit Default 'NULL'. The distance unit for the label.
offset Default 'NULL'. The distance away from the edge to place the scale bar. If 'NULL', automatically calculated.
radius Default 'NULL'. The radius of the cylinder representing the scale bar. If 'NULL', automatically calculated.
color_first Default 'darkred'. Primary color in the scale bar.
color_second Default 'grey90'. Secondary color in the scale bar.
color_text Default 'black'. Color of the text.
text_switch_side Default 'FALSE'. Switches the order of the text.
text_x_offset Default '0'. Distance offset for text in the x direction.
text_y_offset Default '0'. Distance offset for text in the y direction.
text_z_offset Default '0'. Distance offset for text in the z direction.
clear_scalebar Default 'FALSE'. Clears the scale bar(s) on the map.

Value

Displays snapshot of current rgl plot (or saves to disk).

Examples

```r
if(interactive()) {
  #Add a scale bar to the montereybay dataset, here representing about 80km
  # Not run:
  montereybay %>%
  sphere_shade() %>%
  plot_3d(montereybay, theta=45, water=TRUE)
  render_scalebar(limits=c(0, 80), label_unit = "km")
  render_snapshot()

  #This function works with `render_highquality()`
  render_highquality(lightdirection=250, lightaltitude=40, scale_text_size=24, clamp_value=10)
  render_scalebar(clear_scalebar = TRUE)

  #We can change the position by specifying a cardinal direction to 'position', and the
color by setting 'color_first' and 'color_second'
  render_scalebar(limits=c(0,80), label_unit = "km", position = "N",
               color_first = "darkgreen", color_second = "lightgreen")
```

render_snapshot()
render_scalebar(clear_scalebar = TRUE)

# And switch the orientation by setting 'text_switch_side = TRUE'
render_scalebar(limits=c(0,80), label_unit = "km", position = "N", text_switch_side = TRUE,
color_first = "darkgreen", color_second = "lightgreen")
render_snapshot()
render_scalebar(clear_scalebar = TRUE)

# We can add additional breaks by specifying additional distances in 'limits'
render_scalebar(limits=c(0,40,80), label_unit = "km")
render_snapshot()
render_scalebar(clear_scalebar = TRUE)

# We can also manually specify the height by setting the 'y' argument:
render_scalebar(limits=c(0,40,80), y=-70, label_unit = "km")
render_snapshot()
render_scalebar(clear_scalebar = TRUE)

# Here we change the total size by specifying a start and end point along the side,
# and set the number of colored 'segments':
render_scalebar(limits=c(0,20, 40), segments = 4, scale_length = c(0.5,1), label_unit = "km")
render_scalebar(limits=c(0,20, 40), segments = 4, position = "N", text_switch_side = TRUE,
scale_length = c(0.25,0.75), label_unit = "km")
render_snapshot()
render_scalebar(clear_scalebar = TRUE)

# Change the radius of the scale bar with 'radius'. Here, the autopositioning doesn't work well with
# the labels, so we provide additional offsets with 'text_y_offset' and 'text_x_offset' to fix it.
render_scalebar(limits=c(0,20, 40), segments = 4, scale_length = c(0.5,1),
           label_unit = "km", radius=10,text_y_offset=-20,text_x_offset=20)
render_snapshot(clear=TRUE)

## End(Not run)
}

---

render_snapshot Render Snapshot of 3D Visualization

Description

Either captures the current rgl view and displays, or saves the current view to disk.

Usage

render_snapshot()
Arguments

filename
Filename of snapshot. If missing, will display to current device.

clear
Default 'FALSE'. If 'TRUE', the current 'rgl' device will be cleared.

title_text
Default 'NULL'. Text. Adds a title to the image, using magick::image_annotate.

title_offset
Default 'c(20, 20)'. 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_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_position
Default 'northwest'. Position of the title.

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.

vignette
Default 'FALSE'. If 'TRUE' or numeric, 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.
instant_capture  Default 'TRUE' if interactive, 'FALSE' otherwise. If 'FALSE', a slight delay is added before taking the snapshot. This can help stop prevent rendering issues when running scripts.

bring_to_front  Default 'FALSE'. Whether to bring the window to the front when taking the snapshot.

keep_user_par  Default 'TRUE'. Whether to keep the user’s 'par()' settings. Set to 'FALSE' if you want to set up a multi-pane plot (e.g. set 'par(mfrow)').

webshot  Default 'FALSE'. Set to 'TRUE' to have rgl use the 'webshot2' package to take images, which can be used when 'rgl.useNULL = TRUE'.

width  Default 'NULL'. Optional argument to pass to 'rgl::snapshot3d()' to specify the width when 'webshot = TRUE'.

height  Default 'NULL'. Optional argument to pass to 'rgl::snapshot3d()' to specify the height when 'webshot = TRUE'.

...  Additional parameters to pass to magick::image_annotate.

Value

Displays snapshot of current rgl plot (or saves to disk).

Examples

```r
if(interactive()) {

  montereybay %>%
    sphere_shade() %>%
    plot_3d(montereybay,zscale=50,zoom=0.6,theta=-90,phi=30)

  render_snapshot()

  render_snapshot(title_text = "Monterey Bay, California",
                  title_color = "white", title_bar_color = "black",
                  title_font = "Helvetica", gravity = "North")

  render_snapshot(title_text = "Monterey Bay, California",
                  title_color = "white", title_bar_color = "darkgreen",
                  vignette = TRUE,
                  title_font = "Helvetica", gravity = "North")

  rgl::rgl.close()
}
```
render_water

**Render Water Layer**

**Description**

Adds water layer to the scene, removing the previous water layer if desired.

**Usage**

```r
render_water(
  heightmap,
  waterdepth = 0,
  watercolor = "lightblue",
  zscale = 1,
  wateralpha = 0.5,
  waterlinecolor = NULL,
  waterlinealpha = 1,
  linewidth = 2,
  remove_water = TRUE
)
```

**Arguments**

- `heightmap`: A two-dimensional matrix, where each entry in the matrix is the elevation at that point. All points are assumed to be evenly spaced.
- `waterdepth`: Default '0'.
- `watercolor`: Default 'lightblue'.
- `zscale`: Default '1'. The ratio between the x and y spacing (which are assumed to be equal) and the z axis. For example, if the elevation levels are in units of 1 meter and the grid values are separated by 10 meters, 'zscale' would be 10.
- `wateralpha`: Default '0.5'. Water transparency.
- `waterlinecolor`: Default 'NULL'. Color of the lines around the edges of the water layer.
- `waterlinealpha`: Default '1'. Water line transparency.
- `linewidth`: Default '2'. Width of the edge lines in the scene.
- `remove_water`: Default 'TRUE'. If 'TRUE', will remove existing water layer and replace it with new layer.

**Examples**

```r
if(interactive()) {
  # Not run:
  montereybay %>%
    sphere_shade() %>%
    plot_3d(montereybay,zscale=50)
  render_snapshot()
}
## End(Not run)

## We want to add a layer of water after the initial render.
## Not run:
render_water(montereybay,zscale=50)
render_snapshot()
## End(Not run)

## Call it again to change the water depth
## Not run:
render_water(montereybay,zscale=50,waterdepth=-1000)
render_snapshot()
## End(Not run)

## Add waterlines
## Not run:
render_camera(theta=-45)
render_water(montereybay,zscale=50,waterlinecolor="white")
render_snapshot(clear = TRUE)
rgl::rgl.close()
## End(Not run)

}

---

**resize_matrix**  
*Resize Matrix*

### Description

Resizes a matrix (preserving contents) by specifying the desired output dimensions or a scaling factor.

### Usage

```r
resize_matrix(
  heightmap,
  scale = 1,
  width = NULL,
  height = NULL,
  method = "bilinear"
)
```

### Arguments

- **heightmap**  
The elevation matrix.
scale  Default ‘0.5’. The amount to scale down the matrix. Scales using bilinear interpolation.
width  Default ‘NULL’. Alternative to ‘scale’ argument. The desired output width. If ‘width’ is less than 1, it will be interpreted as a scaling factor—e.g. 0.5 would halve the resolution for the width.
height Default ‘NULL’. Alternative to ‘scale’ argument. The desired output width. If ‘height’ is less than 1, it will be interpreted as a scaling factor—e.g. 0.5 would halve the resolution for the height.
method Default ‘bilinear’. Method of interpolation. Alternatively ‘cubic’, which is slightly smoother, although current implementation slightly scales the image.

Examples

#Reduce the size of the monterey bay dataset by half

montbaysmall = resize_matrix(montereybay, scale=0.5)
montbaysmall %>%
sphere_shade() %>
plot_map()

#Reduce the size of the monterey bay dataset from 540x540 to 100x100
montbaysmall = resize_matrix(montereybay, width = 100, height = 100)
montbaysmall %>%
sphere_shade() %>
plot_map()

#Increase the size of the volcano dataset 3x
volcanobig = resize_matrix(volcano, scale=3)
volcanobig %>
sphere_shade() %>
plot_map()

#Increase the size of the volcano dataset 2x, using cubic interpolation
volcanobig = resize_matrix(volcano, scale=3, method="cubic")
volcanobig %>
sphere_shade() %>
plot_map()

save_3dprint

Description

Writes a stereolithography (STL) file that can be used in 3D printing.
Usage

```r
save_3dprint(
  filename,
  maxwidth = 125,
  unit = "mm",
  rotate = TRUE,
  remove_extras = TRUE,
  clear = FALSE
)
```

Arguments

- `filename` String with the filename. If `.stl` is not at the end of the string, it will be appended automatically.
- `maxwidth` Default `125`. Desired maximum width of the 3D print in millimeters. Uses the units set in `unit` argument. Can also pass in a string, "125mm" or "5in".
- `unit` Default `mm`. Units of the `maxwidth` argument. Can also be set to inches with `in`.
- `rotate` Default `TRUE`. If `FALSE`, the map will be printing on its side. This may improve resolution for some 3D printing types.
- `remove_extras` Default `TRUE`. Removes non-topographic features from base: lines, water, labels, and the shadow.
- `clear` Default `FALSE`. If `TRUE`, the current `rgl` device will be cleared.

Value

Writes an STL file to `filename`. Regardless of the unit displayed, the output STL is in millimeters.

Examples

```r
if(interactive()) {
  filename_stl = tempfile()
  # Save the STL file into `filename_stl`
  volcano %>%
    sphere_shade() %>%
    plot_3d(volcano, zscale=3)
    render_snapshot()
  save_3dprint(filename_stl, clear=TRUE)

  # Save the STL file into `filename_stl`, setting maximum width to 100 mm
  volcano %>%
    sphere_shade() %>%
    plot_3d(volcano, zscale=3)
    render_snapshot()
  save_3dprint(filename_stl, maxwidth = 100, clear=TRUE)
}
```
save_obj

Save OBJ

Description

Writes the textured 3D rayshader visualization to an OBJ file.

Usage

```r
save_obj(
  filename,
  save_texture = TRUE,
  water_index_refraction = 1,
  manifold_geometry = FALSE,
  all_face_fields = FALSE
)
```

Arguments

- `filename` String with the filename. If `.obj` is not at the end of the string, it will be appended automatically.
- `save_texture` Default `TRUE`. If the texture should be saved along with the geometry.
- `water_index_refraction` Default `1`. The index of refraction for the rendered water.
- `manifold_geometry` Default `FALSE`. If `TRUE`, this will take the additional step of making the mesh manifold.
- `all_face_fields` Default `FALSE`. If `TRUE`, all OBJ face fields (v/vn/vt) will always be written.
Examples

```r
if(interactive()) {
  filename_obj = tempfile(fileext = "obj")

  # Save model of volcano
  volcano %>%
    sphere_shade() %>%
    plot_3d(volcano, zscale = 2)
  save_obj(filename_obj)

  # Save model of volcano without texture
  save_obj(filename_obj, save_texture = FALSE)
  rgl::rgl.close()

  # Make water have realistic index of refraction
  montereybay %>%
    sphere_shade() %>%
    plot_3d(montereybay, zscale = 50)
  save_obj(filename_obj, water_index_refraction = 1.5)
  rgl::rgl.close()
}
```

Description

Writes the hillshaded map to file.

Usage

```r
save_png(
  hillshade,
  filename,
  title_text = NA,
  title_offset = c(20, 20),
  title_color = "black",
  title_size = 30,
  title_font = "sans",
  title_style = "normal",
  title_bar_color = NULL,
```
save_png

title_bar_alpha = 0.5,
title_position = "northwest",
rotate = 0,
asp = 1
)

Arguments

hillshade Array (or matrix) of hillshade to be written.
filename String with the filename. If '.png' is not at the end of the string, it will be
 appended automatically.
title_text Default ‘NULL’. Text. Adds a title to the image, using ‘magick::image_annotate()’. 
title_offset Default ‘c(20,20)’. 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.
rotate Default 0. Rotates the output. Possible values: 0, 90, 180, 270.
asp Default ‘1’. Aspect ratio of the resulting plot. Use ‘asp = 1/cos(pi(mean_latitude/180))’
to rescale lat/long at higher latitudes to the correct aspect ratio.

Examples

filename_map = tempfile()

#Save the map into ‘filename_map’
montereybay %>%
sphere_shade() %>%
save_png(filename_map)

#Rotate the map 180 degrees:

montereybay %>%
sphere_shade() %>%
save_png(filename_map, rotate=180)
**sphere_shade**  
*Calculate Surface Color Map*

**Description**
Calculates a color for each point on the surface using the surface normals and hemispherical UV mapping. This uses either a texture map provided by the user (as an RGB array), or a built-in color texture.

**Usage**
```
sphere_shade(
heightmap,  
sunangle = 315,  
texture = "imhof1",  
normalvectors = NULL,  
colorintensity = 1,  
zscale = 1,  
progbar = interactive()
)
```

**Arguments**
- `heightmap`: A two-dimensional matrix, where each entry in the matrix is the elevation at that point. All points are assumed to be evenly spaced.
- `sunangle`: Default `315` (NW). The direction of the main highlight color (derived from the built-in palettes or the `create_texture` function).
- `texture`: Default `"imhof1"`. Either a square matrix indicating the spherical texture mapping, or a string indicating one of the built-in palettes (`"imhof1","imhof2","imhof3","imhof4","desert","bw", and "unicorn"").
- `normalvectors`: Default `NULL`. Cache of the normal vectors (from `calculate_normal` function). Supply this to speed up texture mapping.
- `colorintensity`: Default `1`. The intensity of the color mapping. Higher values will increase the intensity of the color mapping.
- `zscale`: Default `1/colorintensity`. The ratio between the x and y spacing (which are assumed to be equal) and the z axis. Ignored unless `colorintensity` missing.
- `progbar`: Default `TRUE` if interactive, `FALSE` otherwise. If `FALSE`, turns off progress bar.

**Value**
RGB array of hillshaded texture mappings.
texture_shade

Examples

# Basic example:
montereybay %>%
  sphere_shade() %>%
  plot_map()

# Decrease the color intensity:
montereybay %>%
  sphere_shade(colorintensity=0.1) %>%
  plot_map()

# Change to a built-in color texture:
montereybay %>%
  sphere_shade(texture="desert") %>%
  plot_map()

# Change the highlight angle:
montereybay %>%
  sphere_shade(texture="desert", sunangle = 45) %>%
  plot_map()

# Create our own texture using the `create_texture` function:
montereybay %>%
  sphere_shade(texture=create_texture("springgreen","darkgreen",
                                 "turquoise","steelblue3","white")) %>%
  plot_map()

---

**texture_shade**  
*Calculate Texture Shading Map*

**Description**

Calculates a shadow for each point on the surface using the method described by Leland Brown in "Texture Shading: A New Technique for Depicting Terrain Relief."

**Usage**

```r
texture_shade(
  heightmap,
  detail = 0.5,
  contrast = 1,
  brightness = 0,
  transform = TRUE,
  dx = 1,
  dy = 1,
  pad = 50
)
```
Arguments

heightmap A two-dimensional matrix, where each entry in the matrix is the elevation at that point.
detail Default ‘0.5’. Amount of detail in texture shading algorithm. ‘0’ is the least detail, while ‘1’ is the most.
contrast Default ‘1’. Standard brightness. Amount of contrast in the texture shading. This transforms the resulting darkness using the formula ‘tanh(input * contrast + brightness)’.
brightness Default ‘0’, standard brightness. Higher values will brighten the texture hillshade, while lower values will darken it.
transform Default ‘TRUE’. Whether to apply the ‘tanh(input * contrast + brightness)’ transformation. This transforms the resulting darkness using the formula ‘tanh(input * contrast + brightness)’.
dx Default ‘1’. The distance between each row of data (compared to the height axis).
dy Default ‘1’. The distance between each column of data (compared to the height axis).
pad Default ‘50’. The amount to pad the heightmap so edge effects don’t appear from the fourier transform. Only increase this if you encounter boundary effects.

Value

2D matrix of hillshade values.

Examples

# Create a direct mapping of elevation to color:

# Plut using default values
montereybay %>%
texture_shade() %>%
plot_map()

# Increase the level of detail
montereybay %>%
texture_shade(detail=1) %>%
plot_map()

# Decrease the level of detail
montereybay %>%
texture_shade(detail=0) %>%
plot_map()

# Increase the level of contrast
montereybay %>%
texture_shade(contrast=3) %>%
plot_map()
#Increase the brightness for this level of contrast
montereybay %>%
  texture_shade(contrast=5, brightness = 2) %>%
  plot_map()

#Add a texture_shade() layer into a map
montbay = montereybay
montbay[montbay < 0] = 0

montbay %>%
  height_shade() %>%
  add_water(detect_water(montbay), color="dodgerblue") %>%
  add_shadow(texture_shade(montbay, detail=1/3, contrast = 5, brightness = 6),0) %>%
  add_shadow(lamb_shade(montbay,zscale=50),0) %>%
  plot_map()

## Description
re-export magrittr pipe operator
Index

* datasets
  - monterey_counties_sf, 37
  - monterey_roads_sf, 38
  - montereybay, 36
  >, 91

  add_overlay, 3
  add_shadow, 4
  add_water, 5
  ambient_shade, 6

  calculate_normal, 8
  create_texture, 9

  detect_water, 10
  generate_altitude_overlay, 11
  generate_compass_overlay, 12
  generate_contour_overlay, 15
  generate_label_overlay, 17
  generate_line_overlay, 21
  generate_point_overlay, 23
  generate_polygon_overlay, 24
  generate_scalebar_overlay, 26
  generate_waterline_overlay, 31

  height_shade, 34
  lamb_shade, 35

  monterey_counties_sf, 37
  monterey_roads_sf, 38
  montereybay, 36

  plot_3d, 38
  plot_gg, 42
  plot_map, 46

  raster_to_matrix, 48
  ray_shade, 49
  reduce_matrix_size, 51

  render_camera, 51
  render_compass, 53
  render_depth, 56
  render_highquality, 59
  render_label, 63
  render_movie, 66
  render_path, 69
  render_points, 71
  render_polygons, 74
  render_scalebar, 76
  render_snapshot, 78
  render_water, 81
  resize_matrix, 82

  save_3dprint, 83
  save_obj, 85
  save_png, 86
  sphere_shade, 88

  texture_shade, 89

92