Package: rayshader (via r-universe)

July 22, 2026

Type Package

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

Version 0.41.2

Date 2026-07-19

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 (>= 4.1)

Imports doParallel, foreach, Rcpp, progress, raster, scales, png, jpeg, rgl (>= 1.3.16), grDevices, grid, utils, methods, terrainmeshr, rayimage (>= 0.24.1), rayvertex (>= 0.13.1), rayrender (>= 0.40.0), magrittr

Suggests reshape2, viridis, av, magick, ggplot2 (>= 4.0.0), sf, isoband, car (>= 3.1-1), geosphere, gifski, ambient, terra, elevatr, gridExtra, testthat (>= 3.0.0), osmdata, raybevel (>= 0.2.1), ragg, skymodelr

LinkingTo Rcpp, progress, RcppArmadillo

Encoding UTF-8

URL https://www.rayshader.com, https://github.com/tylermorganwall/rayshader

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

Config/testthat/edition 3

Config/roxygen2/version 8.0.0

NeedsCompilation yes

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

Repository https://cran.r-universe.dev

Date/Publication 2026-07-20 04:11:05 UTC

RemoteUrl https://github.com/cran/rayshader

RemoteRef HEAD

RemoteSha 5e2ef2e629b53d5667c0dfa3947f7a25a22de5cb

Contents

`add_overlay` *Add Overlay*

Description

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

Usage

```
add_overlay(
  hillshade = NULL,
  overlay = NULL,
  alphalayer = 1,
  alphacolor = NULL,
  alphamethod = "max",
```

```
  color_epsilon = 0.001,
  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.
`alphalayer`	Default `1`. Defines minimum tranparaency of layer. If transparency already exists in `overlay`, the way `add_overlay()` combines the two is determined in argument `alphamethod`.
`alphacolor`	Default `NULL`. If `overlay` is a 3-layer array, this argument tells which color is interpretted 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`.
`color_epsilon`	Default `1e-3`. Tolerance for equality for alphacolor to determine transparency.
`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.

Examples

```
#Combining base R plotting with rayshader's spherical color mapping and raytracing:
if(run_documentation()) {
montereybay |>
  sphere_shade() |>
  add_overlay(height_shade(montereybay),alphalayer = 0.6)  |>
  add_shadow(ray_shade(montereybay,zscale=50)) |>
  plot_map()
}

if(run_documentation()) {
#Add contours with `generate_contour_overlay()`
montereybay |>
  height_shade() |>
  add_overlay(generate_contour_overlay(montereybay))  |>
  add_shadow(ray_shade(montereybay,zscale=50)) |>
  plot_map()
}
```

add_shadow *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 incidates 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

if(run_documentation()) {
montereybay |>
 sphere_shade(colorintensity=0.5) |>
 plot_map()
}

#Raytrace the `montereybay` elevation map and add that shadow to the output of sphere_shade()
if(run_documentation()) {
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.
if(run_documentation()) {
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.
if(run_documentation()) {
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.
`color`	Default `imhof1`. The water fill color. A hexcode or recognized color string. Also includes built-in colors to match the palettes included in `sphere_shade()`: (`imhof1`,`imhof2`,`imhof3`,`imhof4`, `desert`, `bw`, and `unicorn`).

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:
if(run_documentation()) {
island_volcano |>
 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

if(run_documentation()) {
montereybay |>
 sphere_shade(texture="imhof4") |>
 add_water(detect_water(montbay_water),color="imhof4") |>
 plot_map()
}
```

`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(),
  ...
)
```

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 propogate 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 ray-tracer will operate.
shadow_cache	Default NULL. The shadow matrix to be updated at the points defined by the argument cache_mask.
progbar	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.
if(run_documentation()) {
plot_map(ambient_shade(heightmap = montereybay))
}

#We can increase the distance to look for surface intersections `maxsearch`
#and the density of rays sent out around the point `sunbreaks`.
if(run_documentation()) {
plot_map(ambient_shade(montereybay, sunbreaks = 24,maxsearch = 100, multicore=TRUE))
}
#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.
if(run_documentation()) {
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) |>
 add_shadow(lamb_shade(bigmb),0.5) |>
 plot_map()
}
```

---

calculate_normal *Calculate Normal*

Description

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

Usage

```
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

```
#Here we produce a light intensity map of the `volcano` elevation map.

#Cache the normal vectors of the volcano dataset
if(run_documentation()) {
volcanocache = calculate_normal(volcano)
}

#Use the cached vectors to speed up calculation of `sphere_shade()` on a map.
if(run_documentation()) {
sphere_shade(volcano,normalvectors = volcanocache) |>
 plot_map()
}
```

---

cloud_shade *Cloud Shade*

Description

Render shadows from the 3D floating cloud layer on the ground. Use this function to add shadows to the map with the add_shadow() function.

For realistic results, argument should match those passed to render_clouds(). The exception to this is attenuation_coef, which can be used to adjust the darkness of the resulting shadows.

Usage

```
cloud_shade(
  heightmap,
  start_altitude = 1000,
  end_altitude = 2000,
  sun_altitude = 90,
  sun_angle = 315,
  time = 0,
  cloud_cover = 0.5,
  layers = 10,
  offset_x = 0,
```

```
  offset_y = 0,
  scale_x = 1,
  scale_y = 1,
  scale_z = 1,
  frequency = 0.005,
  fractal_levels = 16,
  attenuation_coef = 1,
  seed = 1,
  zscale = 1
)
```

Arguments

`heightmap`	A two-dimensional matrix, where each entry in the matrix is the elevation at that point. This is used by `render_clouds()` to calculate the regions the clouds should be rendered in.
`start_altitude`	Default `1000`. The bottom of the cloud layer.
`end_altitude`	Default `2000`. The top of the cloud layer.
`sun_altitude`	Default `10`. The angle, in degrees (as measured from the horizon) from which the light originates.
`sun_angle`	Default `315` (NW). The angle, in degrees, around the matrix from which the light originates. Zero degrees is North, increasing clockwise
`time`	Default `0`. Advance this to make the clouds evolve and change in shape.
`cloud_cover`	Default `0.5`. The percentage of cloud cover.
`layers`	Default `90`. The number of layers to render the cloud layer.
`offset_x`	Default `0`. Change this to move the cloud layer sideways.
`offset_y`	Default `0`. Change this to move the cloud layer backwards and forward
`scale_x`	Default `1`. Scale the fractal pattern in the x direction.
`scale_y`	Default `1`. Scale the fractal pattern in the y direction.
`scale_z`	Default `1`. Scale the fractal pattern in the z (altitude) direction. (automatically calculated). Scale the fractal pattern in the z (vertical) direction. s.
`frequency`	Default `0.005`. The base frequency of the noise used to calculate the fractal cloud structure.
`fractal_levels`	Default `16`. The fractal dimension used to calculate the noise. Higher values give more fine structure, but take longer to calculate.
`attenuation_coef`	Default `1`. Amount of attenuation in the cloud (higher numbers give darker shadows). This value is automatically scaled to account for increasing the number of layers.
`seed`	Default `1`. Random seed used to generate clouds.
`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.

Value

A 2D shadow matrix.

Examples

```
if(run_documentation()) {
#Render clouds with cloud shadows on the ground
montereybay  |>
 sphere_shade()  |>
 add_shadow(cloud_shade(montereybay,zscale=50), 0.0) |>
 plot_3d(montereybay,background="darkred",zscale=50)
render_camera(theta=-65, phi = 25, zoom = 0.45, fov = 80)
render_clouds(montereybay, zscale=50)
render_snapshot()
}
if(run_documentation()) {
#Adjust the light direction for shadows and increase the attenuation for darker clouds
montereybay  |>
 sphere_shade()  |>
 add_shadow(cloud_shade(montereybay,zscale=50, sun_altitude=20, attenuation_coef = 3), 0.0) |>
 plot_3d(montereybay,background="darkred",zscale=50)
render_camera(theta=-65, phi = 25, zoom = 0.45, fov = 80)
render_clouds(montereybay, zscale=50)
render_snapshot()
}
```

---

constant_shade *Calculate Constant Color Map*

Description

Generates a constant color layer.

Usage

```
constant_shade(heightmap, color = "white", alpha = 1)
```

Arguments

heightmap	A two-dimensional matrix, where each entry in the matrix is the elevation at that point.
color	Default "white". Color for the constant layer.
alpha	Default 1, the alpha transparency.

Value

RGB array of a single color layer.

Examples

```
if(run_documentation()) {
#Shade a red map
montereybay |>
 constant_shade("red") |>
 add_shadow(lamb_shade(montereybay),0) |>
 plot_map()
}
if(run_documentation()) {
#Shade a green map
montereybay |>
 constant_shade("green") |>
 add_shadow(lamb_shade(montereybay),0) |>
 plot_map()
}
if(run_documentation()) {
#Add a blue tint
montereybay |>
 height_shade() |>
 add_overlay(constant_shade(montereybay, "dodgerblue", alpha=0.25)) |>
 add_shadow(lamb_shade(montereybay,zscale=50),0) |>
 plot_map()
}
if(run_documentation()) {
#Use a blank map on which to draw other data
montereybay |>
 constant_shade() |>
 add_overlay(generate_line_overlay(monterey_roads_sf, linewidth=5, color="black",
                                   attr(montereybay,"extent"), width = 1080, height = 1080),
                                      alphalayer=0.8)  |>
 add_water(detect_water(montereybay < 0), "dodgerblue") |>
 plot_map()
}
```

---

convert_path_to_animation_coords

Calculate Animation Coordinates from Path

Description

Transforms latitude/longitude/altitude coordinates to the reference system used in `render_highquality()`, so they can be used to create high quality pathtraced animations by passing the output to the `animation_camera_coords` argument in `render_highquality()`.

This function converts the path values to rayshader coordinates (by setting `return_coords = TRUE` in `render_path()`) and then subtracts out the rgl y-offset, which can be obtained by calling the internal function `rayshader:::get_scene_depth()`.

Usage

```
convert_path_to_animation_coords(
  lat,
  long = NULL,
  altitude = NULL,
  extent = NULL,
  frames = 360,
  reorder = FALSE,
  reorder_first_index = 1,
  reorder_duplicate_tolerance = 0.1,
  reorder_merge_tolerance = 1,
  simplify_tolerance = 0,
  zscale = 1,
  heightmap = NULL,
  offset = 5,
  type = "bezier",
  offset_lookat = 1,
  constant_step = TRUE,
  curvature_adjust = "none",
  curvature_scale = 30,
  follow_camera = FALSE,
  follow_distance = 100,
  follow_angle = 45,
  follow_rotations = 0,
  follow_fixed = FALSE,
  follow_fixed_offset = c(10, 10, 10),
  damp_motion = FALSE,
  damp_magnitude = 0.1,
  resample_path_evenly = TRUE,
  ...
)
```

Arguments

`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). If left `NULL`, this will be just the elevation value at ths surface, offset by `offset`. If a single value, all data will be rendered at that altitude.
`extent`	Either an object representing the spatial extent of the scene (either from the `raster`, `terra`, `sf`, or `sp` packages), a length-4 numeric vector specifying `c("xmin", "xmax", "ymin", "ymax")`, or the spatial object (from the previously aforementioned packages) which will be automatically converted to an extent object.
`frames`	Default `360`. Total number of animation frames.

`reorder` Default `TRUE`. If `TRUE`, this will attempt to re-order the rows within an `sf` object with multiple paths to be one continuous, end-to-end path. This happens in two steps: merging duplicate paths that have end points that match with another object (within `reorder_duplicate_tolerance` distance), and then merges them (within `reorder_merge_tolerance` distance) to form a continuous path.

`reorder_first_index`
Default `1`. The index (row) of the `sf` object in which to begin the reordering process. This merges and reorders paths within `reorder_merge_tolerance` distance until it cannot merge any more, and then repeats the process in the opposite direction.

`reorder_duplicate_tolerance`
Default `0.1`. Lines that have start and end points (does not matter which) within this tolerance that match a line already processed (order determined by `reorder_first_index`) will be discarded.

`reorder_merge_tolerance`
Default `1`. Lines that have start points that are within this distance to a previously processed line's end point (order determined by `reorder_first_index`) will be reordered within the `sf` object to form a continuous, end-to-end path.

`simplify_tolerance`
Default `0` (no simplification). If greater than zero, simplifies the path to the tolerance specified. This happens after the data has been merged if `reorder = TRUE`. If the input data is specified with long-lat coordinates and `sf_use_s2()` returns `TRUE`, then the value of simplify_tolerance must be specified in meters.

`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.

`offset` Default `5`. Offset of the track from the surface, if `altitude = NULL`.

`type` Default `cubic`. Type of transition between keyframes. Other options are `linear`, `quad`, `bezier`, `exp`, and `manual`. `manual` just returns the values passed in, properly formatted to be passed to `render_animation()`.

`offset_lookat` Default `0`. Amount to offset the lookat position, either along the path (if `constant_step = TRUE`) or towards the derivative of the Bezier curve.

`constant_step` Default `TRUE`. This will make the camera travel at a constant speed.

`curvature_adjust`
Default `none`. Other options are `position`, `lookat`, and `both`. Whether to slow down the camera at areas of high curvature to prevent fast swings. Only used for curve `type = bezier`. This does not preserve key frame positions. Note: This feature will likely result in the `lookat` and `position` diverging if they do not have similar curvatures at each point. This feature is best used when passing the same set of points to `positions` and `lookats` and providing an `offset_lookat` value, which ensures the curvature will be the same.

curvature_scale
: Default 30. Constant dividing factor for curvature. Higher values will subdivide the path more, potentially finding a smoother path, but increasing the calculation time. Only used for curve type = bezier. Increasing this value after a certain point will not increase the quality of the path, but it is scene-dependent.

follow_camera
: Default FALSE. If TRUE, this generates a 3rd person view that follows the path specified in lat, long, and altitude. The distance to the camera is specified by follow_distance, and the angle (off the ground) is specified by follow_angle. Make the camera rotate around the point as it moves by setting follow_rotations to a non-zero number. The camera points in the direction of the You can also set the camera to be a fixed distance and angle above the by settings follow_fixed = TRUE and specifying the distance in follow_fixed_offset.

follow_distance
: Default 100. Distance for the camera to follow the point when follow_camera = TRUE.

follow_angle
: Default 45. Angle (off the ground) of the camera when follow_camera = TRUE.

follow_rotations
: Default 0. Number of rotations around the point when follow_camera = TRUE.

follow_fixed
: Default FALSE. If TRUE, the camera doesn't look in the direction of the path, but rather sits at a fixed relative location to the path.

follow_fixed_offset
: Default c(10,10,10). If follow_fixed = TRUE, the offset from the path to place the camera.

damp_motion
: Default FALSE. Whether the suppress quick, jerky movements of the camera by linearly interpolating between the current camera position and the goal position. Amount of linear interpolation set in damp_magnitude.

damp_magnitude
: Default 0.1. Amount of linear interpolation if damp_motion = TRUE.

resample_path_evenly
: Default TRUE. This re-samples points along the path so that the camera moves at a constant speed along the path. This also allows paths with large numbers of points to be used with a smaller number of frames, and improves computation time of the animation path in those instances.

...
: Other arguments to pass to rayrender::generate_camera_motion()

Examples

```
#Generate a circle in Monterey Bay and fly around on top of it
## Not run:
tmp = tempdir()

# Recreate your original example but compact, with fewer frames & smaller size.
moss_landing_coord = c(36.806807, -121.793332)
t = seq(0, 2 * pi, length.out = 360) # fewer frames for speed
circle_coords_lat = moss_landing_coord[1] + 0.25 * sin(t)
circle_coords_long = moss_landing_coord[2] + 0.25 * cos(t)

# Minimal 3D render -> frames
```

```
extent_mb = attr(montereybay, "extent")
sphere_shade(montereybay) |>
plot_3d(
 montereybay,
 zscale = 50,
 water = TRUE,
 shadowcolor = "#40310a",
 background = "tan",
 theta = 210,
 phi = 22,
 zoom = 0.40,
 fov = 55
)

render_path(
extent = extent_mb,
heightmap = montereybay,
lat = circle_coords_lat,
long = circle_coords_long,
zscale = 50,
color = "red",
antialias = TRUE,
offset = 500,
linewidth = 2
)

cam = convert_path_to_animation_coords(
extent = extent_mb,
heightmap = montereybay,
lat = circle_coords_lat,
long = circle_coords_long,
type = "bezier",
damp_motion = TRUE,
fovs = 80,
zscale = 50,
offset = 1000,
frames = length(t)
)

render_highquality(
samples = 4,
animation_camera_coords = cam,
width = 200,
height = 200,
preview = FALSE,
filename = file.path(tmp, "frame"),
use_extruded_paths = TRUE
)

# Assemble frames -> GIF (pkgdown copies from man/figures)
pngs = sprintf("%s/frame%d.png", tmp, seq_along(t))
av::av_encode_video(
pngs,
```

```
framerate = 24
)

## End(Not run)

#Now we use a "follow camera" for a third person view:
## Not run:
follow_cam = convert_path_to_animation_coords(
extent = extent_mb,
heightmap = montereybay,
lat = circle_coords_lat,
long = circle_coords_long,
type = "bezier",
damp_motion = TRUE,
fovs = 80,
zscale = 50,
follow_camera = TRUE,
offset = 1000,
frames = length(t)
)

render_highquality(
samples = 4,
animation_camera_coords = follow_cam,
width = 200,
height = 200,
preview = FALSE,
filename = file.path(tmp, "frame"),
use_extruded_paths = TRUE
)

# See description for video (if on the documentation website)
pngs = sprintf("%s/frame%d.png", tmp, seq_along(t))
av::av_encode_video(
pngs,
framerate = 24
)

## End(Not run)
```

`convert_rgl_to_raymesh` *Convert rayshader RGL scene to ray_mesh object*

Description

Converts the current RGL rayshader scene to a `ray_mesh` object (see `rayvertex` package for more information)

Usage

```
convert_rgl_to_raymesh(
  save_shadow = TRUE,
  water_attenuation = 0,
  water_surface_color = TRUE,
  water_ior = 1
)
```

Arguments

save_shadow	Default FALSE. If TRUE, this saves a plane with the shadow texture below the model.
water_attenuation	Default 0, no attenuation. Amount that light should be attenuated when traveling through water.
water_surface_color	Default TRUE. Whether the water should have a colored surface or not. This is in contrast to setting a non-zero water attenuation, where the color comes from the attenuation of light in the water itself.
water_ior	Default 1. Water index of refraction.

Value

A ray_mesh object

Examples

```
filename_obj = tempfile(fileext = ".obj")
#Save model of volcano
if(run_documentation()) {
volcano |>
 sphere_shade() |>
 plot_3d(volcano, zscale = 2)

rm_obj = convert_rgl_to_raymesh()
}
```

---

create_texture *Create Texture*

Description

Creates a texture map based on 5 user-supplied colors.

Usage

```
create_texture(
  lightcolor,
  shadowcolor,
  leftcolor,
  rightcolor,
  centercolor,
  cornercolors = NULL,
  old_method = FALSE,
  darken = 1
)
```

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.
old_method	Default FALSE. Whether to use the original method rayshader used for computing textures, which had non-standard color handling.
darken	Default 1. Amount to darken the colors.

Examples

```
#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()
```

`detect_water` *Detect water*

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. Alternatively, if heightmap is a logical matrix, each entry specifies whether that point is water or not.
`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.

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:
island_volcano |>
 sphere_shade(texture="imhof3") |>
 add_water(detect_water(island_volcano, min_area = 400),color="imhof3") |>
 plot_map()
```

---

flag_banner_obj *Flag Banner 3D Model*

Description

3D obj model of a flag (sans pole), to be used with `render_obj()`. Use `flag_full_obj()` to get the complete pole, and `flag_banner_obj()` and `flag_pole_obj()` to style them separately.

Usage

```
flag_banner_obj()
```

Value

File location of the included flag OBJ file (saved with a .txt extension)

Examples

```
#Print the location of the flag file
flag_banner_obj()
```

---

flag_full_obj *Flag 3D Model*

Description

3D obj model of a flag, to be used with `render_obj()`. Use `flag_full_obj()` to get the complete pole, and `flag_banner_obj()` and `flag_pole_obj()` to style them separately.

Usage

```
flag_full_obj()
```

Value

File location of the included flag OBJ file (saved with a .txt extension)

Examples

```
#Print the location of the flag file
flag_full_obj()
```

`flag_pole_obj` *Flag Pole 3D Model*

Description

3D obj model of a flag pole, to be used with `render_obj()`. Use `flag_full_obj()` to get the complete pole, and `flag_banner_obj()` and `flag_pole_obj()` to style them separately.

Usage

```
flag_pole_obj()
```

Value

File location of the included flag OBJ file (saved with a .txt extension)

Examples

```
#Print the location of the flag file
flag_pole_obj()
```

`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

```
#Create a bathymetric hillshade
if(run_documentation()) {
water_palette = colorRampPalette(c("darkblue", "dodgerblue", "lightblue"))(200)
bathy_hs = height_shade(montereybay, texture = water_palette)
plot_map(bathy_hs)
}

if(run_documentation()) {
#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)

if(run_documentation()) {
#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

```
generate_compass_overlay(
  x = 0.85,
  y = 0.15,
  size = 0.05,
  text_size = 1,
  bearing = 0,
  heightmap = NULL,
  width = NA,
  height = NA,
  resolution_multiply = 1,
  color1 = NA,
  color2 = NA,
  text_color = NA,
  border_color = "black",
  border_width = 1,
 compass_type = c("classic", "split_arrow", "triangle_circle", "split_arrow_ring"),
  halo_color = NA,
  halo_expand = 2,
  halo_alpha = 1,
  halo_offset = c(0, 0),
  halo_blur = 0,
  halo_edge_softness = 0.1
)
```

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.
resolution_multiply	Default 1. If passing in heightmap instead of width/height, amount to increase the resolution of the overlay, which should make lines/polygons finer. Should be combined with add_overlay() with rescale_original = TRUE to ensure those added details are captured in the final map.
color1	Default NA, white except for compass_type == "triangle_circle" . Primary color of the compass.
color2	Default NA, 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.
compass_type	Default "classic". Set to "split_arrow", "triangle_circle", or "split_arrow_ring" to draw simple polygon north arrows that match the styles shown above.
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.
halo_edge_softness	Default 0.1. Width of the softened halo edge transition, in pixels.

Value

Semi-transparent overlay with a compass.

Examples

```
if(run_documentation()) {
#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()
}

if(run_documentation()) {
#Change the position to be over the water
base_map |>
 add_overlay(generate_compass_overlay(heightmap = montereybay, x = 0.15)) |>
 plot_map()
}
if(run_documentation()) {
#Change the type
base_map |>
 add_overlay(generate_compass_overlay(heightmap = montereybay, x = 0.15,
                                      compass_type = "split_arrow")) |>
 plot_map()
}
if(run_documentation()) {
#Change the type
base_map |>
 add_overlay(generate_compass_overlay(heightmap = montereybay, x = 0.15,
                                      compass_type = "split_arrow_ring")) |>
 plot_map()
}
if(run_documentation()) {
#Change the type
base_map |>
 add_overlay(generate_compass_overlay(heightmap = montereybay, x = 0.15,
                                      compass_type = "triangle_circle")) |>
 plot_map()
}
if(run_documentation()) {
#Change the text color for visibility
base_map |>
 add_overlay(generate_compass_overlay(heightmap = montereybay, compass_type = "split_arrow",
x = 0.15, text_color="white")) |>
 plot_map()
}
if(run_documentation()) {
#Alternatively, add a halo color to improve contrast
base_map |>
 add_overlay(generate_compass_overlay(heightmap = montereybay, x = 0.15, y=0.15,
             compass_type = "split_arrow", halo_color="white", halo_expand = 2)) |>
 plot_map()
}
if(run_documentation()) {
```

```
#Change the color scheme
base_map |>
 add_overlay(generate_compass_overlay(heightmap = montereybay, x = 0.15, y=0.15,
             compass_type = "split_arrow", halo_color="white",
             halo_expand = 2, color1 = "purple", color2 = "red")) |>
 plot_map()
}
if(run_documentation()) {
#Remove the inner border
base_map |>
 add_overlay(generate_compass_overlay(heightmap = montereybay, x = 0.15, y=0.15,
             border_color=NA,compass_type = "split_arrow",
             halo_color="white", halo_expand = 2,
             color1 = "darkolivegreen4", color2 = "burlywood3")) |>
 plot_map()
}
if(run_documentation()) {
#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 = 2,compass_type = "classic",
             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 = 2,compass_type = "split_arrow_ring",
             size=0.075)) |>
 add_overlay(generate_compass_overlay(heightmap = montereybay, x = 0.15, y=0.15,
             halo_color="white", halo_expand = 2,compass_type = "split_arrow",
             size=0.075/2, text_size = 0.75)) |>
 plot_map()
}
if(run_documentation()) {
#Change the bearing of the compass
base_map |>
 add_overlay(generate_compass_overlay(heightmap = montereybay, x = 0.15, y=0.15,
             halo_color="white", halo_expand = 2, bearing=30, compass_type = "classic",
             size=0.075)) |>
 add_overlay(generate_compass_overlay(heightmap = montereybay, x = 0.35, y=0.15,
        halo_color="white", halo_expand = 2, bearing=15, compass_type = "triangle_circle",
             size=0.075)) |>
 add_overlay(generate_compass_overlay(heightmap = montereybay, x = 0.15, y=0.35,
        halo_color="white", halo_expand = 2, bearing=-45, compass_type = "split_arrow_ring",
             size=0.075)) |>
 plot_map()
}
if(run_documentation()) {
#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=0.7, halo_blur=3,
             halo_color="black", halo_expand = 2, halo_offset = c(0.002,-0.002))) |>
add_overlay(generate_compass_overlay(heightmap = montereybay, x = 0.35, y=0.15,
             text_color="white", halo_alpha=0.7, halo_blur=3,
             compass_type = "split_arrow",
```

```
             halo_color="black", halo_expand = 2, halo_offset = c(0.002,-0.002))) |>
add_overlay(generate_compass_overlay(heightmap = montereybay, x = 0.15, y=0.35,
             text_color="white", halo_alpha=0.2, halo_blur=8,
             compass_type = "split_arrow_ring",
             halo_color="white", halo_expand = 2)) |>
 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,
  resolution_multiply = 1,
  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.

`resolution_multiply`
: Default 1. If passing in `heightmap` instead of width/height, amount to increase the resolution of the overlay, which should make lines/polygons finer. Should be combined with `add_overlay()` with `rescale_original = TRUE` to ensure those added details are captured in the final map.

`color`
: Default `black`. Color.

`linewidth`
: Default 1. Line width.

Value

Semi-transparent overlay with contours.

Examples

```
#Add contours to the montereybay dataset
if(run_documentation()) {
montereybay |>
 height_shade() |>
 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]

if(run_documentation()) {
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()
}
if(run_documentation()) {
#Increase the resolution of the contour to improve the appearance of 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_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()
```

```
}
if(run_documentation()) {
#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, color="black",
                                      width  = ncol(montereybay)*2), alphalayer=0.5) |>
 plot_map()
}
if(run_documentation()) {
#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 `car::placeLabel()` function to generate labels for the given scene. Either use an `sf` object or manually specify the x/y coordinates and label.

Usage

```
generate_label_overlay(
  labels,
  extent,
  x = NULL,
  y = NULL,
  heightmap = NULL,
  width = NA,
  height = NA,
  resolution_multiply = 1,
  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 = 0,
  halo_edge_softness = 0.1,
  seed = NA
)
```

Arguments

labels	A character vector of labels, or an sf object with POINT geometry and a column for labels.
extent	Either an object representing the spatial extent of the scene (either from the raster, terra, sf, or sp packages), a length-4 numeric vector specifying c("xmin", "xmax", "ymin", "ymax"), or the spatial object (from the previously aforementioned packages) which will be automatically converted to an extent object.
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.
height	Default NA. Width of the resulting overlay. Default the same dimensions as height map.
resolution_multiply	Default 1. If passing in heightmap instead of width/height, amount to increase the resolution of the overlay, which should make lines/polygons/text finer. Should be combined with add_overlay() with rescale_original = TRUE to ensure those added details are captured in the final 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 0. Amount of blur to apply to the halo. Values greater than 30 won't result in further blurring.

halo_edge_softness
Default 0.1. Width of the softened halo edge transition, in pixels.

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
if(run_documentation()) {
#Create the water palette
water_palette = colorRampPalette(c("darkblue", "dodgerblue", "lightblue"))(200)
bathy_hs = height_shade(montereybay, texture = water_palette)
#Set label font
par(family = "Arial")

#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()
}
if(run_documentation()) {
```

```
#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()
}
if(run_documentation()) {
#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$INTPTLAT)),
                                    color="black", point_size = 1, text_size = 1,
                               extent= attr(montereybay,"extent"), heightmap = montereybay,
                                    halo_color = "white", halo_expand = 3,
                                    seed=1))  |>
 plot_map()
}
if(run_documentation()) {
#Adding a softer blurred halo
montereybay |>
 height_shade() |>
 add_overlay(generate_altitude_overlay(bathy_hs, montereybay, 0, 0)) |>
 add_shadow(lamb_shade(montereybay,zscale=50),0.3) |>
 add_overlay(generate_label_overlay(labels=as.character(monterey_counties_sf$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, halo_blur=10,
                                    seed=1))  |>
 plot_map()
}
if(run_documentation()) {
#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 lines for the current height map.

Usage

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

Arguments

`geometry`	An `sf` object with LINESTRING geometry.
`extent`	Either an object representing the spatial extent of the scene (either from the `raster`, `terra`, `sf`, or sp packages), a length-4 numeric vector specifying `c("xmin", "xmax", "ymin", "ymax")`, or the spatial object (from the previously aforementioned packages) which will be automatically converted to an extent 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.
`height`	Default `NA`. Width of the resulting overlay. Default the same dimensions as height map.
`resolution_multiply`	Default `1`. If passing in `heightmap` instead of width/height, amount to increase the resolution of the overlay, which should make lines/polygons/text finer. Should be combined with `add_overlay()` with `rescale_original = TRUE` to ensure those added details are captured in the final 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
if(run_documentation()) {
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))  |>
 add_shadow(ray_shade(montereybay,zscale=50),0.3) |>
 plot_map()
}
if(run_documentation()) {
#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()
}
if(run_documentation()) {
#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,
  resolution_multiply = 1,
  pch = 16,
  color = "black",
  size = 1,
  offset = c(0, 0),
  data_column_width = NULL
)
```

Arguments

geometry	An sf object with POINT geometry.
extent	Either an object representing the spatial extent of the scene (either from the raster, terra, sf, or sp packages), a length-4 numeric vector specifying c("xmin", "xmax", "ymin", "ymax"), or the spatial object (from the previously aforementioned packages) which will be automatically converted to an extent 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.
height	Default NA. Width of the resulting overlay. Default the same dimensions as height map.
resolution_multiply	Default 1. If passing in heightmap instead of width/height, amount to increase the resolution of the overlay, which should make lines/polygons/points finer. Should be combined with add_overlay() with rescale_original = TRUE to ensure those added details are captured in the final map.
pch	Default 16, 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

`color` Default `black`. Color of the points.

`size` Default `1`. Point size.

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

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

Value

Semi-transparent overlay with contours.

Examples

```
#Add the included `sf` object with roads to the montereybay dataset
if(run_documentation()) {
 monterey_city = sf::st_sfc(sf::st_point(c(-121.893611, 36.603056)))
 montereybay |>
   height_shade() |>
   add_overlay(generate_point_overlay(monterey_city, color="red", size=2,
                                attr(montereybay,"extent"), heightmap = montereybay))  |>
   add_shadow(ray_shade(montereybay,zscale=50),0.3) |>
   plot_map()
}
```

---

generate_polygon_overlay

Generate Polygon Overlay

Description

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

Usage

```
generate_polygon_overlay(
  geometry,
  extent,
  heightmap = NULL,
  width = NA,
  height = NA,
  resolution_multiply = 1,
```

```
  offset = c(0, 0),
  data_column_fill = NULL,
  linecolor = "black",
  palette = "white",
  linewidth = 1
)
```

Arguments

geometry	An sf object with POLYGON geometry.
extent	Either an object representing the spatial extent of the scene (either from the raster, terra, sf, or sp packages), a length-4 numeric vector specifying c("xmin", "xmax","ymin","ymax"), or the spatial object (from the previously aforementioned packages) which will be automatically converted to an extent 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.
height	Default NA. Width of the resulting overlay. Default the same dimensions as height map.
resolution_multiply	Default 1. If passing in heightmap instead of width/height, amount to increase the resolution of the overlay, which should make lines/polygons/text finer. Should be combined with add_overlay() with rescale_original = TRUE to ensure those added details are captured in the final 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.
linecolor	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.
linewidth	Default 1. Line width.

Value

Image overlay representing the input polygon data.

Examples

```
#Plot the counties around Monterey Bay, CA
if(run_documentation()) {
generate_polygon_overlay(monterey_counties_sf, palette = rainbow,
                         extent = attr(montereybay,"extent"), heightmap = montereybay) |>
 plot_map()
```

```
}
if(run_documentation()) {
#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.
water_palette = colorRampPalette(c("darkblue", "dodgerblue", "lightblue"))(200)
bathy_hs = height_shade(montereybay, texture = water_palette)

generate_polygon_overlay(monterey_counties_sf, palette = rainbow,
                         extent = attr(montereybay,"extent"), heightmap = montereybay) |>
 add_overlay(generate_altitude_overlay(bathy_hs, montereybay, start_transition = 0)) |>
 plot_map()
}
if(run_documentation()) {
#Add a semi-transparent hillshade and change the palette, and remove the polygon lines
montereybay |>
 sphere_shade(texture = "bw") |>
 add_overlay(generate_polygon_overlay(monterey_counties_sf,
                         palette = terrain.colors, linewidth=NA,
                         extent = attr(montereybay,"extent"), heightmap = montereybay),
                         alphalayer=0.7) |>
 add_overlay(generate_altitude_overlay(bathy_hs, montereybay, start_transition = 0)) |>
 add_shadow(ray_shade(montereybay,zscale=50),0) |>
 plot_map()
}
if(run_documentation()) {
#Map one of the variables in the sf object and use an explicitly defined color palette
county_palette = c("087" = "red",    "053" = "blue",   "081" = "green",
                   "069" = "yellow", "085" = "orange", "099" = "purple")
montereybay |>
 sphere_shade(texture = "bw") |>
 add_shadow(ray_shade(montereybay,zscale=50),0) |>
 add_overlay(generate_polygon_overlay(monterey_counties_sf, linecolor="white", linewidth=3,
                         palette = county_palette, data_column_fill = "COUNTYFP",
                         extent = attr(montereybay,"extent"), heightmap = montereybay),
                         alphalayer=0.7) |>
 add_overlay(generate_altitude_overlay(bathy_hs, montereybay, start_transition = 0)) |>
 add_shadow(ray_shade(montereybay,zscale=50),0.5) |>
 plot_map()
}
```

---

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,
  resolution_multiply = 1,
  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 = 0,
  halo_edge_softness = 0.1
)
```

Arguments

extent	Either an object representing the spatial extent of the scene (either from the raster, terra, sf, or sp packages), a length-4 numeric vector specifying c("xmin", "xmax","ymin","ymax"), or the spatial object (from the previously aforementioned packages) which will be automatically converted to an extent object. 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.
bearing	Default 90, horizontal. Direction (measured from north) of the scale bar.
unit	Default m. Displayed unit on the scale bar.
flip_ticks	Default FALSE. Whether to flip the ticks to the other side of the scale bar.
labels	Default NA. Manually specify the three labels with a length-3 character vector. Use this if you want display units other than meters.
text_size	Default 1. Text size.
decimals	Default 0. Number of decimal places for scale bar labels.
text_offset	Default 1. Amount of offset to apply to the text from the scale bar, as a multiple of thickness.
adj	Default 0.5, centered. Text justification. 0 is left-justified, and 1 is right-justified.
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.
resolution_multiply	Default 1. If passing in heightmap instead of width/height, amount to increase the resolution of the overlay, which should make lines/polygons/text finer. Should be combined with add_overlay() with rescale_original = TRUE to ensure those added details are captured in the final map.
color1	Default black. Primary color of the scale bar.
color2	Default white. Secondary color of the scale bar.
text_color	Default black. Text color.
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.
border_color	Default black. Border color of the scale bar.
tick_color	Default black. Tick color of the scale bar.
border_width	Default 1. Width of the scale bar border.
tick_width	Default 1. Width of the tick.
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 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, as a proportion of the full scene.

`halo_blur` Default `0`. Amount of blur to apply to the halo. Values greater than `30` won't result in further blurring.

`halo_edge_softness` Default `0.1`. Width of the softened halo edge transition, in pixels.

Value

Semi-transparent overlay with a scale bar.

Examples

```
if(run_documentation()) {
#Create the water palette
water_palette = colorRampPalette(c("darkblue", "dodgerblue", "lightblue"))(200)
bathy_hs = height_shade(montereybay, texture = water_palette)
#Set scalebar font
par(family = "Arial")

#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()
}
if(run_documentation()) {
#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()
}
if(run_documentation()) {
#Change the length
```

```
base_map |>
 add_overlay(generate_scalebar_overlay(extent = mb_extent, length = 30000,
                                       text_color = "white",
                                       heightmap = montereybay,
                                       latlong=TRUE)) |>
 plot_map()
}
if(run_documentation()) {
#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()
}
if(run_documentation()) {
#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()
}
if(run_documentation()) {
#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()
}
if(run_documentation()) {
#Add a halo
base_map |>
 add_overlay(generate_scalebar_overlay(extent = mb_extent, length = 40000,
                                       halo_color = "white", halo_expand = 1,
                                       heightmap = montereybay, font = 2,
                                       latlong=TRUE)) |>
 plot_map()
}
if(run_documentation()) {
#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, font = 2,
```

```
                                        latlong=TRUE)) |>
 plot_map()
}
if(run_documentation()) {
#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", halo_blur=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*

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

```
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,
  width = NA,
  height = NA,
  resolution_multiply = 1,
  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 TRUE. 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.
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.

`resolution_multiply`
Default `1`. If passing in `heightmap` instead of width/height, amount to increase the resolution of the overlay, which should make lines/polygons/text finer. Should be combined with `add_overlay()` with `rescale_original = TRUE` to ensure those added details are captured in the final map.

`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

```
if(run_documentation()) {
#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)
}
if(run_documentation()) {
#Add waterlines
basemap |>
  add_overlay(generate_waterline_overlay(montbay)) |>
  plot_map()
}
if(run_documentation()) {
#Change minimum line distance:
basemap |>
  add_overlay(generate_waterline_overlay(montbay, min = 0.02)) |>
  plot_map()
}
if(run_documentation()) {
```

```
#Change maximum line distance
basemap |>
  add_overlay(generate_waterline_overlay(montbay, max = 0.4)) |>
  plot_map()
}
if(run_documentation()) {
#Smooth waterlines
basemap |>
  add_overlay(generate_waterline_overlay(montbay, max = 0.4, smooth=2)) |>
  plot_map()
}
if(run_documentation()) {
#Increase number of breaks
basemap |>
  add_overlay(generate_waterline_overlay(montbay, breaks = 20, max=0.4)) |>
  plot_map()
}
if(run_documentation()) {
#Make lines evenly spaced:
basemap |>
  add_overlay(generate_waterline_overlay(montbay, evenly_spaced = TRUE)) |>
  plot_map()
}
if(run_documentation()) {
#Change variable distance between each line
basemap |>
  add_overlay(generate_waterline_overlay(montbay, falloff=1.5)) |>
  plot_map()
}
if(run_documentation()) {
#Turn off fading
basemap |>
  add_overlay(generate_waterline_overlay(montbay, fade=FALSE)) |>
  plot_map()
}
if(run_documentation()) {
#Fill up the entire body of water with lines and make them all 50% transparent
basemap |>
 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

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

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.

Value

RGB array of hillshaded texture mappings.

Examples

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

#Add a shadow:
if(run_documentation()) {
montereybay |>
 height_shade() |>
 add_shadow(ray_shade(montereybay,zscale=50),0.1) |>
 plot_map()
}

#Change the palette:
if(run_documentation()) {
montereybay |>
 height_shade(texture = topo.colors(256)) |>
 add_shadow(ray_shade(montereybay,zscale=50),0.1) |>
 plot_map()
}

#Really change the palette (warning: gratuitous use of rainbow palette):
if(run_documentation()) {
montereybay |>
 height_shade(texture = rainbow(256)) |>
 add_shadow(ray_shade(montereybay,zscale=50),0.1) |>
 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

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 azimuth angle as measured from the horizon from which the light originates.
sunangle	Default 315 (NW). The angle around the matrix from which the light originates.
zscale	Default 1. The ratio between the x and y spacing (which are assumed to be equal) and the z axis.
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

```
if(run_documentation()) {
#Generate a basic hillshade
montereybay |>
 lamb_shade(zscale=200) |>
 plot_map()
}
if(run_documentation()) {
#Increase the intensity by decreasing the zscale
montereybay |>
 lamb_shade(zscale=50) |>
```

```
 plot_map()
}
if(run_documentation()) {
#Change the sun direction
montereybay |>
 lamb_shade(zscale=200, sunangle=45) |>
 plot_map()
}
if(run_documentation()) {
#Change the sun altitude
montereybay |>
 lamb_shade(zscale=200, sunaltitude=60) |>
 plot_map()
}
```

`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

```
monterey_counties_sf
```

Format

An `sf` object with MULTIPOLYGON geometry.

Source

https://catalog.data.gov/dataset/tiger-line-shapefile-2016-state-california-current-county-subdivision-state-based

Examples

```
# This is the full code (commented out) used to generate this dataset from the original data:
#counties = sf::st_read("tl_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

```
# 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"))
```

montereybay *Monterey Bay combined topographic and bathymetric elevation matrix.*

Description

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

Source

https://www.ncei.noaa.gov/metadata/geoportal/rest/metadata/item/gov.noaa.ngdc.mgg.dem:3544/html

Examples

```
# 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
```

---

plot_3d *Plot 3D*

Description

Displays the shaded map in 3D with the `rgl` package.

Note: Calling `plot_3d()` resets the scene cache for the `render_snapshot()`, `render_depth()`, and `render_highquality()`

Usage

```
plot_3d(
  hillshade,
  heightmap,
  zscale = 1,
  baseshape = "rectangle",
  solid = TRUE,
  soliddepth = "auto",
  solidcolor = "grey20",
  solidlinecolor = "grey30",
  shadow = TRUE,
  shadowdepth = "auto",
  shadowcolor = "auto",
  shadow_darkness = 0.5,
  shadowwidth = "auto",
  water = FALSE,
  waterdepth = 0,
  watercolor = "dodgerblue",
```

```
  wateralpha = 0.5,
  waterlinecolor = NULL,
  waterlinealpha = 1,
  linewidth = 2,
  lineantialias = FALSE,
  soil = FALSE,
  soil_freq = 0.1,
  soil_levels = 16,
  soil_color_light = "#b39474",
  soil_color_dark = "#8a623b",
  soil_gradient = 2,
  soil_gradient_darken = 4,
  theta = 45,
  phi = 45,
  fov = 0,
  zoom = 1,
  background = "white",
  windowsize = 600,
  precomputed_normals = NULL,
  triangulate = FALSE,
  max_error = 0,
  max_tri = 0,
  verbose = FALSE,
  plot_new = TRUE,
  close_previous = TRUE,
  clear_previous = TRUE
)
```

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. If heightmap is uniform and set on `auto`, this is automatically set to a slightly lower level than the uniform elevation.
`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 auto. Color of the shadow, automatically computed as shadow_darkness the luminance of the background color in the CIELuv colorspace if not specified.

shadow_darkness
Default 0.5. Darkness of the shadow, if shadowcolor = "auto".

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 tranparency.

linewidth Default 2. Width of the edge lines in the scene.

lineantialias Default FALSE. Whether to anti-alias the lines in the scene.

soil Default FALSE. Whether to draw the solid base with a textured soil layer.

soil_freq Default 0.1. Frequency of soil clumps. Higher frequency values give smaller soil clumps.

soil_levels Default 16. Fractal level of the soil.

soil_color_light
Default "#b39474". Light tint of soil.

soil_color_dark
Default "#8a623b". Dark tint of soil.

soil_gradient Default 2. Sharpness of the soil darkening gradient. 0 turns off the gradient entirely.

soil_gradient_darken
Default 4. Amount to darken the soil_color_dark value for the deepest soil layers. Higher numbers increase the darkening effect.

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.

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.

plot_new
: Default TRUE, opens new window with each plot_3d() call. If FALSE, the data will be plotted in the same window.

close_previous
: Default TRUE. Closes any previously open rgl window. If FALSE, old windows will be kept open.

clear_previous
: Default TRUE. Clears the previously open rgl window if plot_new = FALSE.

Examples

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

#With a water layer
if(run_documentation()) {
montereybay |>
 sphere_shade(texture="imhof2") |>
 plot_3d(montereybay, zscale=50, water = TRUE, watercolor="imhof2",
         waterlinecolor="white", waterlinealpha=0.5)
render_snapshot()
}

#With a soil texture to the base
if(run_documentation()) {
montereybay |>
 sphere_shade(texture="imhof3") |>
 plot_3d(montereybay, zscale=50, water = TRUE,  watercolor="imhof4",
         waterlinecolor="white", waterlinealpha=0.5, soil=TRUE)
render_camera(theta=225, phi=7, zoom=0.5, fov=67)
```

```
render_snapshot()
}

#We can also change the base by setting "baseshape" to "hex" or "circle"
if(run_documentation()) {
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()
}

if(run_documentation()) {
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()
}



#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

if(run_documentation()) {
montereybay |>
 sphere_shade(texture="imhof1") |>
 plot_3d(mb_water, zscale=50, water = TRUE, watercolor="imhof1", theta=-45,
         waterlinecolor="white", waterlinealpha=0.5)
render_snapshot()
}
```

---

plot_gg *Transform ggplot2 objects into 3D*

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

```
plot_gg(
  ggobj,
  ggobj_height = NULL,
  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,
  flat_plot_render = FALSE,
  flat_distance = "auto",
  flat_transparent_bg = FALSE,
  flat_direction = "-z",
  shadow = TRUE,
  shadowdepth = "auto",
  shadowcolor = "auto",
  shadow_darkness = 0.5,
  background = "white",
  preview = FALSE,
  raytrace = TRUE,
  sunangle = 315,
  anglebreaks = seq(30, 40, 0.1),
  multicore = FALSE,
  lambert = TRUE,
  triangulate = FALSE,
  max_error = 0.001,
  max_tri = 0,
  verbose = FALSE,
  emboss_text = 0,
  emboss_grid = 0,
  reduce_size = NULL,
  save_height_matrix = FALSE,
  save_shadow_matrix = FALSE,
  saved_shadow_matrix = NULL,
  monitor_gamma = 1.8,
  plot = TRUE,
  ...
)
```

Arguments

ggobj	ggplot object to projected into 3D.
ggobj_height	Default NULL. A ggplot object that can be used to specify the 3D extrusion separately from the ggobj.
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. You can also specify a number to control the thickness of the offset. Note: this feature may end up removing thin polygons from the plot entirely–use with care.
flat_plot_render	Default FALSE. Whether to render a flat version of the ggplot above (or alongside) the 3D version.
flat_distance	Default "auto". Distance to render the flat version of the plot from the 3D version.
flat_transparent_bg	Default FALSE. Whether to set the background of the flat version of the ggplot to transparent.
flat_direction	Default "-z". Direction to render the flat copy of the plot, if flat_plot_render = TRUE. Other options c("z", "x", "-x", "y", "-y").
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 auto. Color of the shadow, automatically computed as shadow_darkness the luminance of the background color in the CIELab colorspace if not specified.

shadow_darkness
Default 0.5. Darkness of the shadow, if shadowcolor = "auto".

background Default "white". Background color.

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.

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.

emboss_text Default 0, max 1. Amount to emboss the text, where 1 is the tallest feature in the scene.

emboss_grid Default 0, max 1. Amount to emboss the grid lines, where 1 is the tallest feature in the scene. By default, the minor grid lines will be half the size of the major lines. Pass a length-2 vector to specify them seperately (second value is the minor grid height).

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() with save_shadow_matrix = TRUE) to use instead of raytracing a shadow map each time.

monitor_gamma
: Default 1.8. Undo the gamma correction applied by the png device. Ignored if ragg is installed or the cairo PNG device is available.

plot
: Default TRUE. Whether to plot the image when preview = TRUE, or just return the RGBA rayimg.

...
: Additional arguments to be passed to plot_3d().

Value

Opens a 3D plot in rgl.

Examples

```
library(ggplot2)
library(viridis)


ggdiamonds = ggplot(diamonds, aes(x, depth)) +
 stat_density_2d(aes(fill = after_stat(nlevel), color = after_stat(nlevel)),
                 geom = "polygon",
                 n = 200, bins = 50,contour = TRUE) +
 facet_wrap(clarity~.) +
 scale_fill_viridis_c(option = "A") +
 scale_color_viridis_c(option = "A")
if(run_documentation()) {
plot_gg(ggdiamonds,multicore = TRUE,width=5,height=5,scale=250,windowsize=c(1400,866),
       zoom = 0.55, phi = 30)
render_snapshot()
}
#Change the camera angle and take a snapshot:
if(run_documentation()) {
render_camera(zoom=0.5,theta=-30,phi=30)
render_snapshot()
}

#Contours and other lines will automatically be ignored. Here is the volcano dataset:
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() +
 theme(legend.position = "none")
ggvolcano

if(run_documentation()) {
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()
}

if(run_documentation()) {
#You can specify the color and height separately using the `ggobj_height()` argument.
ggvolcano_surface = volcano |>
reshape2::melt() |>
 ggplot() +
 geom_contour(aes(x=Var1,y=Var2,z=value),color="black") +
 geom_contour_filled(aes(x=Var1,y=Var2,z=value))+
 scale_x_continuous("X",expand = c(0,0)) +
 scale_y_continuous("Y",expand = c(0,0)) +
 coord_fixed() +
 theme(legend.position = "none")

plot_gg(ggvolcano_surface, ggobj_height = ggvolcano,
      multicore = TRUE, raytrace = TRUE, width = 7, height = 4,
      scale = 300, windowsize = c(1400, 866), zoom = 0.6, phi = 30, theta = 30)
render_snapshot()
}
#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`.
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.
if(run_documentation()) {
plot_gg(mtplot, width=3.5, sunangle=225, preview = TRUE)
}
if(run_documentation()) {
plot_gg(mtplot, width=3.5, multicore = TRUE, windowsize = c(1400,866), sunangle=225,
       zoom = 0.60, phi = 30, theta = 45)
render_snapshot()
}
#Now let's plot a density plot in 3D.
mtplot_density = ggplot(mtcars) +
 stat_density_2d(aes(x=mpg,y=disp, fill=after_stat(!!str2lang("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
```

```
if(run_documentation()) {
plot_gg(mtplot_density, width = 4,zoom = 0.60, theta = -45, phi = 30,
       windowsize = c(1400,866))
render_snapshot()
}
#This also works facetted.
mtplot_density_facet = mtplot_density + facet_wrap(~cyl)

#Preview this plot in 2D:
if(run_documentation()) {
plot_gg(mtplot_density_facet, preview = TRUE)
}
if(run_documentation()) {
plot_gg(mtplot_density_facet, windowsize=c(1400,866),
       zoom = 0.55, theta = -10, phi = 25)
render_snapshot()
}
#That is a little cramped. Specifying a larger width will improve the readability of this plot.
if(run_documentation()) {
plot_gg(mtplot_density_facet, width = 6, preview = TRUE)
}

#That's better. Let's plot it in 3D, and increase the scale.
if(run_documentation()) {
plot_gg(mtplot_density_facet, width = 6, windowsize=c(1400,866),
       zoom = 0.55, theta = -10, phi = 25, scale=300)
render_snapshot()
}

#We can also render a flat version of the plot alongside (or above/below) the 3D version.
if(run_documentation()) {
plot_gg(mtplot_density_facet, width = 6, windowsize=c(1400,866),
       zoom = 0.65, theta = -25, phi = 35, scale=300, flat_plot_render=TRUE,
       flat_direction = "x")
render_snapshot()
}
```

---

plot_map *Plot Map*

Description

Displays the map in the current device.

Usage

```
plot_map(
  hillshade,
  title_text = NA,
```

```
  title_offset = c(20, 20),
  title_color = "black",
  title_size = 30,
  title_font = "sans",
  title_style = "normal",
  title_bar_color = NA,
  title_bar_alpha = 0.5,
  title_just = "left",
  ...
)
```

Arguments

hillshade	Hillshade to be plotted.
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 NA. If a color, this will create a colored bar under the title.
title_bar_alpha	Default 0.5. Transparency of the title bar.
title_just	Default left. Justification of the title.
...	Additional arguments to pass to the raster::plotRGB function that displays the map.

Examples

```
#Plotting the Monterey Bay dataset with bathymetry data
if(run_documentation()) {
water_palette = colorRampPalette(c("darkblue", "dodgerblue", "lightblue"))(200)
bathy_hs = height_shade(montereybay, texture = water_palette)
#For compass text
par(family = "Arial")

#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()
}
```

raster_to_matrix *Raster to Matrix*

Description

Turns a raster into a matrix suitable for rayshader.

Usage

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

Arguments

raster	The input raster. Either a RasterLayer object, a terra SpatRaster 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.
if(run_documentation()) {
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()
}
```

ray_shade *Calculate Raytraced Shadow Map*

Description

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

Usage

```
ray_shade(
  heightmap,
  sunaltitude = 45,
  sunangle = 315,
  maxsearch = NULL,
  lambert = TRUE,
  zscale = 1,
```

```
  multicore = FALSE,
  cache_mask = NULL,
  shadow_cache = NULL,
  progbar = interactive(),
  anglebreaks = NULL,
  ...
)
```

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

```
#First we ray trace the Monterey Bay dataset.
#The default angle is from 40-50 degrees azimuth, from the north east.
if(run_documentation()) {
montereybay |>
 ray_shade(zscale=50) |>
 plot_map()
}
#Change the altitude of the sun to 25 degrees
if(run_documentation()) {
montereybay |>
 ray_shade(zscale=50, sunaltitude=25) |>
 plot_map()
}
#Remove the lambertian shading to just calculate shadow intensity.
if(run_documentation()) {
montereybay |>
 ray_shade(zscale=50, sunaltitude=25, lambert=FALSE) |>
 plot_map()
}

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

---

reduce_matrix_size *Reduce Matrix Size (defunct)*

Description

Reduce Matrix Size (defunct)

Usage

```
reduce_matrix_size(...)
```

Arguments

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

Value

Reduced matrix.

Examples

```
if(run_documentation()) {
montbaysmall = resize_matrix(montereybay, scale=0.5)
montbaysmall |>
 sphere_shade() |>
 plot_map()
}
```

`render_beveled_polygons` *Render Beveled Polygons*

Description

Adds beveled polygon to the scene using the `raybevel` package. See the `raybevel::generate_beveled_polygon()` function for more information.

Usage

```
render_beveled_polygons(
  polygon,
  extent,
  material = "grey",
  bevel_material = NA,
  angle = 45,
  bevel_width = 5,
  width_raw_units = FALSE,
  bevel = NA,
  zscale = 1,
  bevel_height = 1,
  base_height = 0,
  raw_heights = FALSE,
  raw_offsets = FALSE,
  heights_relative_to_centroid = TRUE,
  set_max_height = FALSE,
  max_height = 10,
  scale_all_max = TRUE,
  data_column_top = NULL,
  data_column_bottom = NULL,
  heightmap = NULL,
  scale_data = 1,
  holes = 0,
  alpha = 1,
  lit = TRUE,
  flat_shading = FALSE,
  light_altitude = c(45, 30),
  light_direction = c(315, 225),
```

```
  light_intensity = 1,
  light_relative = FALSE,
  clear_previous = FALSE,
  ...
)
```

Arguments

polygon	sf object, "SpatialPolygon" 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.
extent	Either an object representing the spatial extent of the 3D scene (either from the raster, terra, sf, or sp packages), a length-4 numeric vector specifying c("xmin", "xmax", "ymin", "ymax"), or the spatial object (from the previously aforementioned packages) which will be automatically converted to an extent object.
material	Default "grey80". If a color string, this will specify the color of the sides/base of the polygon. Alternatively (for more customization), this can be a rayvertex::material_list() object to specify the full color/appearance/material options for the resulting ray_mesh mesh.
bevel_material	Default NA, defaults to the material specified in material. If a color string, this will specify the color of the polygon bevel. Alternatively (for more customization), this can be a rayvertex::material_list() object to specify the full color/appearance/material options for the resulting ray_mesh mesh.
angle	Default 45. Angle of the bevel.
bevel_width	Default 5. Width of the bevel.
width_raw_units	Default FALSE. Whether the bevel width should be measured in raw display units, or the actual units of the map.
bevel	Default NULL. A list with x/y components that specify a bevel profile. See raybevel::generate_bevel()
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.
bevel_height	Default 1. Height from the base of the polygon to the start of the beveled top.
base_height	Default 0. Height of the base of the polygon.
raw_heights	Default FALSE. A logical flag indicating whether the bevel_heights are already in raw format and do not need to be multiplied by the maximum time of the skeleton. See the documentation for raybevel::generate_beveled_polygon() for more info.
raw_offsets	Default FALSE. A logical flag indicating whether the bevel_offsets are already in raw format and do not need to be multiplied by the maximum time of the skeleton. See the documentation for raybevel::generate_beveled_polygon() for more info.

heights_relative_to_centroid
: Default FALSE. Whether the heights should be measured in absolute terms, or relative to the centroid of the polygon.

set_max_height
: Default FALSE. A logical flag that controls whether to set the max height of the roof based on the max_height argument.

max_height
: Default 1. The maximum height of the polygon.

scale_all_max
: Default FALSE. If passing in a list of multiple skeletons with polygons, whether to scale each polygon to the overall max height, or whether to scale each max height to the maximum internal distance in the polygon.

data_column_top
: Default NULL. A string indicating the column in the sf object to use to specify the top of the beveled polygon.

data_column_bottom
: Default NULL. A string indicating the column in the sf object to use to specify the bottom of the beveled 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.

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.

alpha
: Default 1. Transparency of the polygons.

lit
: Default TRUE. Whether to light the polygons.

flat_shading
: Default FALSE. Set to TRUE to have nicer shading on the 3D polygons. This comes with the slight penalty of increasing the memory use of the scene due to vertex duplication. This will not affect software or high quality renders.

light_altitude
: Default c(45, 30). Degree(s) from the horizon from which to light the polygons.

light_direction
: Default c(315, 225). Degree(s) from north from which to light the polygons.

light_intensity
: Default 1. Intensity of the specular highlight on the polygons.

light_relative
: Default FALSE. Whether the light direction should be taken relative to the camera, or absolute.

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

...
: Additional arguments to pass to rgl::triangles3d().

Examples

```
# This function can also create fake "terrain" from polygons by visualizing the distance
# to the nearest edge.
```

```
if(run_documentation()) {
#Render the county borders as polygons in Monterey Bay as terrain
montereybay |>
  sphere_shade(texture = "desert") |>
  add_shadow(ray_shade(montereybay,zscale = 50)) |>
  plot_3d(montereybay, water = TRUE, windowsize = 800, watercolor = "dodgerblue",
          background = "pink")

#We will apply a negative buffer to create space between adjacent polygons. You may
#have to call `sf::sf_use_s2(FALSE)` before running this code to get it to run.
sf::sf_use_s2(FALSE)
mont_county_buff = sf::st_simplify(sf::st_buffer(monterey_counties_sf,-0.003), dTolerance=0.001)

render_beveled_polygons(mont_county_buff,  flat_shading  = TRUE, angle = 45 ,
                        heightmap = montereybay, bevel_width=2000,
                        material = "red",
                        extent = attr(montereybay,"extent"),
                        bevel_height = 5000, base_height=0,
                        zscale=200)
render_camera(theta = 0,  phi = 90, zoom = 0.65, fov = 0)
render_snapshot()
render_camera(theta=194, phi= 35,   zoom = 0.5, fov= 80)
render_snapshot()
}

# Changing the color of the beveled top:
if(run_documentation()) {
render_beveled_polygons(mont_county_buff,  flat_shading  = TRUE, angle = 45 ,
                        heightmap = montereybay, bevel_width=2000,
                        material = "tan", bevel_material = "darkgreen",
                        extent = attr(montereybay,"extent"), clear_previous=TRUE,
                        bevel_height = 5000, base_height=0,
                        zscale=200)
}
# We can create a nice curved surface by passing in a bevel generated with the
# `raybevel::generate_bevel()` function.
if(run_documentation()) {
render_beveled_polygons(mont_county_buff, flat_shading  = TRUE, heightmap = montereybay,
                        bevel = raybevel::generate_bevel("exp",bevel_end = 0.4),
                        #max_height = 10, scale_all_max = TRUE, set_max_height = TRUE,
                        material = rayvertex::material_list(diffuse="red",
                                                            ambient = "darkred",
                                                            diffuse_intensity = 0.2,
                                                            ambient_intensity = 0.1),
                        light_intensity = 1, light_relative = FALSE,
                        extent = attr(montereybay,"extent"), bevel_height = 5000,
                        base_height=0, clear_previous = TRUE,
                        zscale=200)
render_snapshot()
}

# While the bevels all start at the same point in the above example,
# they rise to different levels due to being scaled by the maximum internal distance
```

```
# in the polygon. Setting `scale_all_max = TRUE` ensures the bevels are all scaled to the
# same maximum height (in this case, 3000m above the 5000m bevel start height).
if(run_documentation()) {
render_beveled_polygons(mont_county_buff, flat_shading  = TRUE, heightmap = montereybay,
                bevel = raybevel::generate_bevel("exp",bevel_end = 0.4),
                max_height = 3000, scale_all_max = TRUE, set_max_height = TRUE,
                material = rayvertex::material_list(diffuse="red",
                                                    ambient = "darkred",
                                                    diffuse_intensity = 0.2,
                                                    ambient_intensity = 0.1),
                light_intensity = 1, light_relative = FALSE,
                extent = attr(montereybay,"extent"), bevel_height = 5000,
                base_height=0, clear_previous = TRUE,
                zscale=200)
render_snapshot()
}

# Rendering the polygons with `render_highquality()`
if(run_documentation()) {
  render_highquality()
}

# We can scale the size of the polygon to a column in the `sf` object as well:
# raybevel::generate_bevel() function. We can scale this data down using the `scale_data`
# argument. Note that this is applied as well as the `zscale` argument, and that you
# must think carefully about your scales and values if trying to represent a meaningful
# data visualization with this object.
if(run_documentation()) {
render_beveled_polygons(mont_county_buff, flat_shading = TRUE, angle = 45, bevel_width=1000,
                data_column_top = "ALAND", scale_data = 1e-5, heightmap = montereybay,
                #max_height = 1000, scale_all_max = TRUE, set_max_height = TRUE,
                material = rayvertex::material_list(diffuse="red"),
                light_intensity = 1, light_relative = FALSE,
                extent = attr(montereybay,"extent"), clear_previous = TRUE,
                zscale=200)
render_snapshot()
}
```

`render_buildings` *Render Buildings*

Description

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

Usage

```
render_buildings(
  polygon,
```

```
  extent,
  material = "grey",
  roof_material = NA,
  angle = 45,
  zscale = 1,
  scale_data = 1,
  relative_heights = TRUE,
  heights_relative_to_centroid = FALSE,
  roof_height = 1,
  base_height = 0,
  data_column_top = NULL,
  data_column_bottom = NULL,
  heightmap = NULL,
  holes = 0,
  alpha = 1,
  lit = TRUE,
  flat_shading = FALSE,
  light_altitude = c(45, 30),
  light_direction = c(315, 225),
  light_intensity = 1,
  light_relative = FALSE,
  clear_previous = FALSE,
  ...
)
```

Arguments

polygon	sf object, "SpatialPolygon" 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.
extent	Either an object representing the spatial extent of the 3D scene (either from the raster, terra, sf, or sp packages), a length-4 numeric vector specifying `c("xmin", "xmax", "ymin", "ymax")`, or the spatial object (from the previously aforementioned packages) which will be automatically converted to an extent object.
material	Default `"grey80"`. If a color string, this will specify the color of the sides/base of the building Alternatively (for more customization), this can be a `rayvertex::material_list()` object to specify the full color/appearance/material options for the resulting `ray_mesh` mesh.
roof_material	Default `NA`, defaults to the material specified in `material`. If a color string, this will specify the color of the roof of the building. Alternatively (for more customization), this can be a `rayvertex::material_list()` object to specify the full color/appearance/material options for the resulting `ray_mesh` mesh.
angle	Default `45`. Angle of the roof.
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.

scale_data Default 1. How much to scale the top/bottom value when rendering. Use zscale to adjust the data to account for x/y grid spacing, and this argument to scale the data for visualization.

relative_heights
Default TRUE. Whether the heights specified in roof_height and base_height should be measured relative to the underlying heightmap.

heights_relative_to_centroid
Default FALSE. Whether the heights should be measured in absolute terms, or relative to the centroid of the polygon.

roof_height Default 1. Height from the base of the building to the start of the roof.

base_height Default 0. Height of the base of the roof.

data_column_top
Default NULL. A string indicating the column in the sf object to use to specify the top of the extruded polygon.

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.

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.

alpha Default 1. Transparency of the polygons.

lit Default TRUE. Whether to light the polygons.

flat_shading Default FALSE. Set to TRUE to have nicer shading on the 3D polygons. This comes with the slight penalty of increasing the memory use of the scene due to vertex duplication. This will not affect software or high quality renders.

light_altitude Default c(45, 30). Degree(s) from the horizon from which to light the polygons.

light_direction
Default c(315, 225). Degree(s) from north from which to light the polygons.

light_intensity
Default 1. Intensity of the specular highlight on the polygons.

light_relative Default FALSE. Whether the light direction should be taken relative to the camera, or absolute.

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

... Additional arguments to pass to rgl::triangles3d().

Examples

```
if(run_documentation()) {
# Load and visualize building footprints from Open Street Map
```

```
library(osmdata)
library(sf)
library(raster)

osm_bbox = c(-121.9472, 36.6019, -121.9179, 36.6385)

#Get buildings from OpenStreetMap
opq(osm_bbox) |>
  add_osm_feature("building") |>
  osmdata_sf() ->
osm_data

#Get roads from OpenStreetMap
opq(osm_bbox) |>
  add_osm_feature("highway") |>
  osmdata_sf() ->
osm_road

#Get extent
building_polys = osm_data$osm_polygons
osm_dem = elevatr::get_elev_raster(building_polys, z = 11, clip = "bbox")
e = extent(building_polys)

# Crop DEM, but note that the cropped DEM will have an extent slightly different than what's
# specified in `e`. Save that new extent to `new_e`.
osm_dem |>
  crop(e) |>
  extent() ->
new_e

osm_dem |>
  crop(e) |>
  raster_to_matrix() ->
osm_mat

#Visualize areas less than one meter as water (approximate tidal range)
osm_mat[osm_mat <= 1] = -2

osm_mat |>
  rayimage::render_resized(mag=4) |>
  sphere_shade(texture = "desert") |>
  add_overlay(generate_polygon_overlay(building_polys, extent = new_e,
                                       heightmap = osm_mat,
                                       linewidth = 6,
                                    resolution_multiply = 50), rescale_original = TRUE) |>
  add_overlay(generate_line_overlay(osm_road$osm_lines, extent = new_e,
                                    heightmap = osm_mat,
                                    linewidth = 6,
                                   resolution_multiply = 50), rescale_original = TRUE) |>
  plot_3d(osm_mat, water = TRUE, windowsize = 800, watercolor = "dodgerblue",
          zscale = 10,
          background = "pink")
```

```
#Render buildings
render_buildings(building_polys,  flat_shading  = TRUE,
                 angle = 30 , heightmap = osm_mat,
                 material = "white", roof_material = "white",
                 extent = new_e, roof_height = 3, base_height = 0,
                 zscale=10)
render_camera(theta=220, phi=22, zoom=0.45, fov=0)
render_snapshot()
}

if(run_documentation()) {
#Zoom in to show roof details and render with render_highquality()
render_camera(fov=110)
render_highquality(camera_location = c(18.22, 0.57, -50.83),
                   camera_lookat = c(20.88, -2.83, -38.87),
                   focal_distance = 13, samples = 16,
                   lightdirection = 45)

}
```

`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,
  shift_vertical = 0
)
```

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`.
`shift_vertical`	Default `0`. Amount to shift the viewpoint.

Examples

```
if(run_documentation()) {
montereybay |>
 sphere_shade() |>
 plot_3d(montereybay,zscale = 50, water = TRUE, waterlinecolor="white")
render_snapshot()
}

#Shift the camera over and add a title
if(run_documentation()) {
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)
if(run_documentation()) {
render_camera(theta = 0, phi = 89.9, zoom = 0.9)
render_snapshot(title_text = "Monterey Bay, CA",
                title_color = "white",
                title_bar_color = "darkgreen")
}

#Shift to an front view and add a vignette effect
if(run_documentation()) {
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.
if(run_documentation()) {
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.

if(run_documentation()) {
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)
}
```

`render_clouds` *Render Clouds*

Description

Render a 3D floating cloud layer of the map.

Note: Underlying layers with transparency can cause rendering issues in rgl.

Usage

```
render_clouds(
  heightmap,
  start_altitude = 1000,
  end_altitude = 2000,
  sun_altitude = 10,
  sun_angle = 315,
  time = 0,
  cloud_cover = 0.5,
  layers = 10,
  offset_x = 0,
  offset_y = 0,
  scale_x = 1,
  scale_y = 1,
  scale_z = 1,
  frequency = 0.005,
  fractal_levels = 16,
  attenuation_coef = 1,
  seed = 1,
  zscale = 1,
  baseshape = "rectangle",
  clear_clouds = FALSE
)
```

Arguments

heightmap	A two-dimensional matrix, where each entry in the matrix is the elevation at that point. This is used by render_clouds() to calculate the regions the clouds should be rendered in.
start_altitude	Default 1000. The bottom of the cloud layer.
end_altitude	Default 2000. The top of the cloud layer.
sun_altitude	Default 90. The angle, in degrees (as measured from the horizon) from which the light originates.
sun_angle	Default 315 (NW). The angle, in degrees, around the matrix from which the light originates. Zero degrees is North, increasing clockwise
time	Default 0. Advance this to make the clouds evolve and change in shape.
cloud_cover	Default 0.5. The percentage of cloud cover.
layers	Default 10. The number of layers to render the cloud layer. The default is layers/(start_altitude - end_altitude).
offset_x	Default 0. Change this to move the cloud layer sideways.
offset_y	Default 0. Change this to move the cloud layer backwards and forwards.
scale_x	Default 1. Scale the fractal pattern in the x direction.
scale_y	Default 1. Scale the fractal pattern in the y direction.
scale_z	Default 1. Scale the fractal pattern in the z (vertical) direction. (automatically calculated). Scale the fractal pattern in the z (vertical) direction.
frequency	Default 0.005. The base frequency of the noise used to calculate the fractal cloud structure.
fractal_levels	Default 16. The fractal dimension used to calculate the noise. Higher values give more fine structure, but take longer to calculate.
attenuation_coef	Default 1. Amount of attenuation in the cloud (higher numbers give darker shadows). This value is automatically scaled to account for increasing the number of layers.
seed	Default 1. Random seed used to generate clouds.
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.
baseshape	Default rectangle. Shape of the base. Options are c("rectangle","circle","hex").
clear_clouds	Default FALSE. Clears all existing floating layers on the visualization.

Value

Adds a 3D floating cloud layer to the map. No return value.

Examples

```
if(run_documentation()) {
#Render a cloud layer over Monterey Bay
montereybay  |>
 sphere_shade()  |>
 plot_3d(montereybay,background="brown",zscale=50)

#Render some clouds
render_clouds(montereybay, zscale=50)
render_snapshot()
}
if(run_documentation()) {
#Change the seed for a different set of clouds and add cloud shadows on the ground
montereybay  |>
 sphere_shade()  |>
 add_shadow(cloud_shade(montereybay,zscale=50, seed = 2), 0.0) |>
 plot_3d(montereybay,background="brown",zscale=50)
render_camera(theta=-65, phi = 25, zoom = 0.45, fov = 80)
render_clouds(montereybay, zscale=50, seed=2, clear_clouds = T)
render_snapshot()
}

if(run_documentation()) {
montereybay  |>
 sphere_shade()  |>
 plot_3d(montereybay,background="brown",zscale=50)

#Lower the frequency for larger, smoother clouds
render_clouds(montereybay, zscale=50, frequency = 0.001, clear_clouds = T)
render_snapshot()
}
if(run_documentation()) {
#Increase the frequency for more broken clouds
render_clouds(montereybay, zscale=50, frequency = 0.05, clear_clouds = T)
render_snapshot()
}
if(run_documentation()) {
#Increase the fractal level for fluffier, bumpier clouds
render_clouds(montereybay, zscale=50, fractal_levels = 32, clear_clouds = T)
render_snapshot()
}
if(run_documentation()) {
#Decrease the fractal level for more smoother, continuous clouds
render_clouds(montereybay, zscale=50, fractal_levels = 4, clear_clouds = T)
render_snapshot()
}
if(run_documentation()) {
#Increase the cloud cover
render_clouds(montereybay, zscale=50, cloud_cover=0.8, clear_clouds = T)
render_snapshot()
}
if(run_documentation()) {
```

```
#Decrease the cloud cover
render_clouds(montereybay, zscale=50, cloud_cover=0.2, clear_clouds = T)
render_snapshot()
}
if(run_documentation()) {
#Change the altitude range of the clouds
render_clouds(montereybay,zscale=50,start_altitude=2000,end_altitude = 4000, clear_clouds = T)
render_snapshot()
}
if(run_documentation()) {
#Increase the number of layers
render_clouds(montereybay, zscale=50,start_altitude=2000,end_altitude = 4000, layers = 20,
             clear_clouds = T)
render_snapshot()
}
if(run_documentation()) {
#Change the sun angle and altitude, and increase the attenuation for darker clouds
render_clouds(montereybay,zscale=50,sun_angle=45, sun_altitude= 5, attenuation_coef = 5,
             clear_clouds = T)
render_snapshot()
}
if(run_documentation()) {
#Render the scene with a different baseshape
montereybay  |>
 sphere_shade()  |>
 plot_3d(montereybay,background="darkred",zscale=50, baseshape="hex")
render_clouds(montereybay,zscale=50, seed=3, baseshape="hex", clear_clouds = T)
render_camera(zoom=0.65)
render_snapshot()
}
```

`render_compass` *Render Compass Symbol*

Description

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

Usage

```
render_compass(
  angle = 0,
  position = "SE",
  altitude = NULL,
  zscale = 1,
  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
)
```

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.
`altitude`	Default `NULL`. Altitude of the compass, defaults to maximum height in the map.
`zscale`	Default `1`. The ratio between the x and y spacing (which are assumed to be equal) and the z axis. Only used in combination with `altitude`.
`x`	Default `NULL`. X position. If not entered, automatically calculated using `position` argument.
`y`	Default `NULL`. Y position. If not entered, automatically calculated using `position` argument.
`z`	Default `NULL`. Z 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(run_documentation()) {
montereybay |>
 sphere_shade() |>
 plot_3d(montereybay,theta=-45, water=TRUE)
render_compass()
render_snapshot()
}
if(run_documentation()) {
#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()
}
if(run_documentation()) {
render_compass(clear_compass = TRUE)

#We can change the position by specifying a direction (here are three):
render_camera(theta=45,phi=45)
render_compass(position = "NW")
render_compass(position = "E")
render_compass(position = "S")
render_snapshot()
}
if(run_documentation()) {
render_compass(clear_compass = TRUE)

#We can also change the distance away from the edge by setting the `scale_distance` argument.
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:
render_camera(theta=45,phi=45,zoom=0.7)
render_snapshot()
}
if(run_documentation()) {
render_compass(clear_compass = TRUE)

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

```
if(run_documentation()) {
#We can also adjust the position manually, be specifying all x, y and z arguments.
render_camera(theta=-45,phi=45,zoom=0.9)
render_compass(x = 150, y = 50, z = 150)
render_snapshot()
}
if(run_documentation()) {
# Compass support is also included in render_highquality()
render_highquality(min_variance = 0, samples = 16)
}
if(run_documentation()) {
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`:

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(0.2,0.2,1), 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",
         color_background = "#abaf98", color_bevel = "#66615e", position_circular = TRUE)
render_compass(position = "W", color_n = "#e9e671", color_arrow = "#cbb387",
         color_background = "#7c9695", color_bevel = "#cbb387", position_circular = TRUE)
render_compass(position = "NW", color_n = c(0.7,0,0), color_arrow = c(0.3,0,0),
        color_background = c(0.7,0.5,0.5), color_bevel = c(0.2,0,0), position_circular = TRUE)
render_snapshot()
}
```

---

render_contours *Render Contours*

Description

Adds 3D contours to the current scene, using the heightmap of the 3D surface.

Usage

```
render_contours(
  heightmap = NULL,
  zscale = 1,
  levels = NA,
```

```
  nlevels = NA,
  linewidth = 1,
  color = "black",
  palette = NULL,
  antialias = FALSE,
  offset = 0,
  clear_previous = 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.
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].
linewidth	Default 3. The line width.
color	Default black. Color of the line.
palette	Default NULL. Overrides color. Either a function that returns a color palette of n colors, or a character vector with colors that specifies each color manually.
antialias	Default FALSE. If TRUE, the line with be 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

```
#Add contours to the montereybay dataset
if(run_documentation()) {
montereybay |>
 height_shade() |>
 add_shadow(ray_shade(montereybay,zscale=50),0.3) |>
 plot_3d(montereybay, theta = -45, zscale=50, zoom=0.9, windowsize=800)
render_contours(montereybay, zscale = 50, offset = 100)
render_snapshot()
}

if(run_documentation()) {
#Specify the number of levels
render_contours(montereybay, zscale = 50, offset = 100, nlevels = 30,
               clear_previous = TRUE)
render_snapshot()
}
```

```
if(run_documentation()) {
#Manually specify the breaks with levels
render_contours(montereybay, linewidth = 2,  offset = 100, zscale = 50,
               levels = seq(-2000, 0, 100), clear_previous = TRUE)
render_snapshot()
}

if(run_documentation()) {
#Use a color palette for the contours
volcano |>
 constant_shade() |>
 plot_3d(volcano, zscale = 2, solid = FALSE, zoom = 0.8)
palette = grDevices::colorRampPalette(c("red", "purple", "pink"))
render_contours(volcano, offset = 1, palette = palette, zscale = 2, nlevels = 20)
render_snapshot()
}

if(run_documentation()) {
#Render using `render_highquality()` for a neon light effect
render_highquality(light = FALSE, samples = 16,
                   line_radius = 0.1,
                   path_material = rayrender::light, ground_size = 0,
                   path_material_args = list(importance_sample = FALSE,
                                             color = "purple", intensity = 2))
}
```

`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_depth is from the image's depth map).

`abs(z_depth-focus)*focal_length^2/(f_stop*z_depth*(focus - focal_length))`

Usage

```
render_depth(
  focus = NULL,
  focallength = 100,
  fstop = 4,
  filename = NULL,
  preview_focus = FALSE,
  bokehshape = "circle",
  bokehintensity = 1,
  bokehlimit = 0.8,
  rotation = 0,
```

```
  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 = NA,
  title_bar_alpha = 0.5,
  title_just = "left",
  image_overlay = NULL,
  vignette = FALSE,
  vignette_color = "black",
  vignette_radius = 1.3,
  progbar = interactive(),
  software_render = FALSE,
  width = NULL,
  height = NULL,
  camera_location = NULL,
  camera_lookat = c(0, 0, 0),
  background = "white",
  text_angle = NULL,
  text_size = 10,
  text_offset = c(0, 0, 0),
  point_radius = 0.5,
  line_offset = 1e-07,
  cache_scene = FALSE,
  reset_scene_cache = FALSE,
  print_scene_info = FALSE,
  instant_capture = interactive(),
  clear = FALSE,
  bring_to_front = FALSE,
  ...
)
```

Arguments

focus	Focal point. Defaults to the center of the bounding box. Depth in which to blur, in distance to the camera plane.
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.

`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`.

`zscale` 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".

`title_bar_color`
Default `NA`. If a color, this will create a colored bar under the title.

`title_bar_alpha`
Default `0.5`. Transparency of the title bar.

`title_just` Default `left`. Justification 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.

`vignette_color` Default `"black"`. Color of the vignette.

`vignette_radius`
Default `1.3`. Radius of the vignette, as a porportion of the image dimensions.

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

`software_render`
Default `FALSE`. If `TRUE`, rayshader will use the rayvertex package to render the snapshot, which is not constrained by the screen size or requires OpenGL.

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

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

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.

background
: Default "white". Background color when software_render = TRUE.

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 10. Height of the text.

text_offset
: Default c(0,0,0). Offset to be applied to all text labels.

point_radius
: Default 0.5. Radius of 3D points (rendered with render_points().

line_offset
: Default 1e-7. Small number indicating the offset in the scene to apply to lines if using software rendering. Increase this if your lines aren't showing up, or decrease it if lines are appearing through solid objects.

cache_scene
: Default FALSE. Whether to cache the current scene to memory so it does not have to be converted to a raymesh object each time render_snapshot() is called. If TRUE and a scene has been cached, it will be used when rendering.

reset_scene_cache
: Default FALSE. Resets the scene cache before rendering.

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

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 rayvertex::rasterize_scene().

Value

4-layer RGBA array.

Examples

```
if(run_documentation()) {
montereybay |>
 sphere_shade() |>
 plot_3d(montereybay,zscale=50, water=TRUE, waterlinecolor="white",
         zoom=0.3,theta=-135,fov=70, phi=20)

#Preview where the focal plane lies
render_depth(preview_focus=TRUE)
```

```
}
if(run_documentation()) {
#Render the depth of field effect
render_depth(focallength = 300)
}
if(run_documentation()) {
#Add a chromatic aberration effect
render_depth(focallength = 300, aberration = 0.3)
}
if(run_documentation()) {
#Render the depth of field effect, ignoring water and re-drawing the waterlayer
render_depth(preview_focus=TRUE,
            heightmap = montereybay, zscale=50, focallength=300, transparent_water=TRUE)
render_depth(heightmap = montereybay, zscale=50, focallength=300, transparent_water=TRUE)
render_camera(theta=45,zoom=0.15,phi=20)
}

if(run_documentation()) {
#Change the bokeh shape and intensity
render_depth(focus=900, bokehshape = "circle",focallength=500,bokehintensity=30,
             title_text = "Circular Bokeh", title_size = 30, title_color = "white",
             title_bar_color = "black")
render_depth(focus=900, bokehshape = "hex",focallength=500,bokehintensity=30,
             title_text = "Hexagonal Bokeh", title_size = 30, title_color = "white",
             title_bar_color = "black")
}

if(run_documentation()) {
#Add a title and vignette effect.
render_camera(theta=0,zoom=0.7,phi=30)
render_depth(focallength = 250, title_text = "Monterey Bay, CA",
        title_size = 20, title_color = "white", title_bar_color = "black", vignette = TRUE)
}
```

`render_floating_overlay` *Render Floating overlay*

Description

Render a 2D floating overlay over the map.

Note: Multiple layers with transparency can cause rendering issues in rgl.

Usage

```
render_floating_overlay(
  overlay = NULL,
  altitude = NULL,
  heightmap = NULL,
```

```
  zscale = 1,
  alpha = 1,
  baseshape = "rectangle",
  remove_na = TRUE,
  reorient = TRUE,
  clear_layers = FALSE,
  horizontal_offset = c(0, 0),
  ...
)
```

Arguments

overlay	Overlay (4D RGBA array) to be rendered on the 3D map.
altitude	Altitude to place the overlay.
heightmap	The underlying surface. A two-dimensional matrix, where each entry in the matrix is the elevation at that point.
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.
alpha	Default 1. Multiplies the layer's transparency by this factor. 0 is completely transparent.
baseshape	Default `rectangle`. Shape of the overlay. Options are `c("rectangle", "circle", "hex")`.
remove_na	Default `TRUE`. Whether to make the overlay transparent above empty spaces (represented by `NA` values) in the underlying elevation matrix.
reorient	Default `TRUE`. Whether to reorient the image array to match the 3D plot.
clear_layers	Default `FALSE`. Clears all existing floating layers on the visualization.
horizontal_offset	Default `c(0,0)`. Distance (in 3D space) to offset the floating offset in the x/y directions.
...	Additional arguments to pass to `rgl::triangles3d()`.

Value

Adds a 3D floating layer to the map. No return value.

Examples

```
if(run_documentation()) {
#Render the road network as a floating overlay layer, along with a label annotation and a floating
#point annotation
if(all(length(find.package("sf", quiet = TRUE)) > 0,
      length(find.package("magick", quiet = TRUE)) > 0)) {
 monterey = c(-121.892933,36.603053)
 monterey_city = sf::st_sfc(sf::st_point(monterey))
```

```
#Generate Overlays
road_overlay = generate_line_overlay(monterey_roads_sf, attr(montereybay,"extent"),
                                     heightmap = montereybay)
point_overlay = generate_point_overlay(monterey_city, color="red", size=1,
                                    attr(montereybay,"extent"), heightmap = montereybay)

#Create 3D plot (water transparency set to 1 because multiple transparency layers can interfere)
montereybay |>
  height_shade() |>
  add_shadow(ray_shade(montereybay,zscale=50),0.3) |>
 plot_3d(montereybay, water = T, wateralpha = 1, windowsize = 800, watercolor = "lightblue")
render_camera(theta=-55,phi=45,zoom=0.8)

#Render label
render_label(montereybay, lat = monterey[2], long = monterey[1], altitude = 9900,
             extent = attr(montereybay, "extent"),
             zscale = 50, text = "Monterey", textcolor = "black", linecolor="darkred")

#Render Floating Overlays
render_floating_overlay(road_overlay, altitude = 10000,zscale = 50)
render_floating_overlay(point_overlay, altitude = 100,zscale = 50)
render_snapshot()
}
}
```

`render_highquality` *Render High Quality*

Description

Renders a raytraced version of the displayed rgl scene, using the `rayrender` package. User can specify the light direction, intensity, and color, as well as specify the material of the ground and add additional scene elements.

This function can also generate frames for an animation by passing camera animation information from either `convert_path_to_animation_coords()` or `rayrender::generate_camera_motion()` functions.

Usage

```
render_highquality(
  filename = NA,
  samples = 128,
  sample_method = "sobol_blue",
  min_variance = 1e-07,
  light = TRUE,
  lat = NA,
  long = NA,
  datetime = NA,
  sky_args = list(),
```

```
lightdirection = 315,
lightaltitude = 45,
lightsize = NULL,
lightintensity = 500,
lightcolor = "white",
material = rayrender::diffuse(),
water_attenuation = 0,
water_surface_color = TRUE,
water_ior = 1,
override_material = FALSE,
cache_scene = FALSE,
reset_scene_cache = FALSE,
width = NULL,
height = NULL,
text_angle = NULL,
text_size = 12,
text_offset = c(0, text_size/2, 0),
line_radius = 0.5,
point_radius = 0.5,
smooth_line = FALSE,
use_extruded_paths = FALSE,
scale_text_angle = NULL,
scale_text_size = 12,
scale_text_offset = c(0, scale_text_size/2, 0),
title_text = NULL,
title_offset = c(20, 20),
title_color = "black",
title_size = 30,
title_font = "sans",
title_just = "left",
title_bar_color = NA,
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,
return_scene = FALSE,
print_scene_info = FALSE,
clamp_value = NA,
calculate_consistent_normals = FALSE,
load_normals = TRUE,
point_material = rayrender::diffuse,
point_material_args = list(),
path_material = rayrender::diffuse,
path_material_args = list(),
```

```
  animation_camera_coords = NULL,
  plot = is.na(filename),
  ...
)
```

Arguments

filename	Default NA. Filename of saved image. If missing, will display to current device.
samples	Default 128. The maximum number of samples for each pixel. Increase this to increase the quality of the rendering.
sample_method	Default "sobol_blue", unless samples > 256, in which it defaults to "sobol". The type of sampling method used to generate random numbers. The other options are random (worst quality but fastest), sobol_blue (best option for sample counts below 256), and sobol (slowest but best quality, better than sobol_blue for sample counts greater than 256).
min_variance	Default 1e-6. Minimum acceptable variance for a block of pixels for the adaptive sampler. Smaller numbers give higher quality images, at the expense of longer rendering times. If this is set to zero, the adaptive sampler will be turned off and the renderer will use the maximum number of samples everywhere.
light	Default TRUE. Whether there should be a light in the scene. If not, the scene will be lit with a bluish sky.
lat	Default NA. Latitude (degrees) for automatic sky generation. When lat, long, datetime, or sky_args are supplied, render_highquality() uses skymodelr::generate_sky_latlon to create an EXR environment map, disables the default light, and sets environment_light for the render.
long	Default NA. Longitude (degrees; west < 0) for automatic sky generation.
datetime	Default NA. POSIXct or character date-time used to position the sun.
sky_args	Default empty list(). Additional arguments passed to skymodelr::generate_sky_latlong() (except filename, which is managed internally). The EXR is cached in tempdir() using a filename derived from these inputs.
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.
material	Default rayrender::diffuse(). The material properties of the object file. Only used if override_material = TRUE
water_attenuation	Default 0, no attenuation. Amount that light should be attenuated when traveling through water. This calculates 1-color

water_surface_color
: Default TRUE. Whether the water should have a colored surface or not. This is in contrast to setting a non-zero water attenuation, where the color comes from the attenuation of light in the water itself.

water_ior
: Default 1. Water index of refraction.

override_material
: Default FALSE. Whether to override the default diffuse material with that in argument material.

cache_scene
: Default FALSE. Whether to cache the current scene to memory so it does not have to be converted to a raymesh object each time render_snapshot() is called. If TRUE and a scene has been cached, it will be used when rendering.

reset_scene_cache
: Default FALSE. Resets the scene cache before rendering.

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 1. Radius of 3D points (rendered with render_points()). This scales the existing value of size specified in render_points().

smooth_line
: Default FALSE. If TRUE, the line will be rendered with a continuous smooth line, rather than straight segments.

use_extruded_paths
: Default TRUE. If FALSE, paths will be generated with the rayrender::path() object, instead of rayrender::extruded_path().

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_just
: Default left. Justification of the title.

`title_bar_color`
: Default `NA`. 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.

`camera_interpolate`
: Default `c(0,0)`. Maximum `1`, minimum `0`. Sets the camera at a point between the `rgl` view and the `camera_location` and `camera_lookat` vectors.

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

`return_scene`
: Default `FALSE`. If `TRUE`, this will return the rayrender scene (instead of rendering the image).

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

`clamp_value`
: Default `NA`. If `NA`, uses `100` when the OIDN denoiser is unavailable and `10000` when it is available (via `rayrender::has_denoiser()`).

`calculate_consistent_normals`
: Default `FALSE`. Whether to calculate consistent vertex normals to prevent energy loss at edges.

`load_normals`
: Default `TRUE`. Whether to load the vertex normals if they exist in the OBJ file.

`point_material`
: Default `rayrender::diffuse`. The rayrender material function to be applied to point data.

`point_material_args`
: Default empty `list()`. The function arguments to `point_material`. The argument `color` will be automatically extracted from the rgl scene, but all other arguments can be specified here.

`path_material`
: Default `rayrender::diffuse`. The rayrender material function to be applied to path data.

`path_material_args`
: Default empty `list()`. The function arguments to `path_material`. The argument `color` will be automatically extracted from the rgl scene, but all other arguments can be specified here.

`animation_camera_coords`
: Default `NULL`. Expects camera animation output from either `convert_path_to_animation_coords()` or `rayrender::generate_camera_motion()` functions.

`plot`
: Default `is.na(filename)`. Whether to plot the scene, or just return the RGBA array.

`...`
: Additional parameters to pass to `rayrender::render_scene()`

Examples

```
#Render the volcano dataset using pathtracing
if(run_documentation()) {
volcano %>%
 sphere_shade() %>%
 plot_3d(volcano,zscale = 2)
render_highquality(min_variance = 0, sample_method = "sobol_blue")
}

#Change position of light
if(run_documentation()) {
render_highquality(lightdirection = 45, min_variance = 0, sample_method = "sobol_blue")
}

#Change vertical position of light
if(run_documentation()) {
render_highquality(lightdirection = 45, lightaltitude = 10,
                   min_variance = 0, samples = 16)
}

#Change the ground material
if(run_documentation()) {
render_highquality(lightdirection = 45, lightaltitude=60,
            ground_material = rayrender::diffuse(checkerperiod = 30, checkercolor="grey50"),
                  min_variance = 0, samples = 16)
}

#Add three different color lights and a title
if(run_documentation()) {
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,
                   min_variance = 0, samples = 16)
}

#Change the camera:
if(run_documentation()) {
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,
                   min_variance = 0, samples = 16)
}
#Add a shiny metal sphere
if(run_documentation()) {
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()),
                   min_variance = 0)
}
```

```
#Add a red light to the volcano and change the ambient light to dusk
if(run_documentation()) {
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)),
                   min_variance = 0, samples = 16)
}
#Manually change the camera location and direction
if(run_documentation()) {
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)),
                   min_variance = 0, samples = 16)
}
# Render the shadow of the Washington Monument with a realistic sky at that datetime
# using the `skymodelr` package.
run_examples = length(find.package("sf", quiet = TRUE)) &&
               length(find.package("elevatr", quiet = TRUE)) &&
               length(find.package("raster", quiet = TRUE)) &&
               run_documentation()
if(run_examples) {
library(sf)
#Set location of washington monument
washington_monument_location =  st_point(c(-77.035249, 38.889462))
wm_point = washington_monument_location |>
 st_point() |>
 st_buffer(0.01) |>
 st_sfc(crs = 4326) |>
 st_transform(st_crs(washington_monument_multipolygonz))

elevation_data = elevatr::get_elev_raster(locations = wm_point, z = 14)

scene_bbox = st_bbox(st_buffer(wm_point,300))
cropped_data = raster::crop(elevation_data, scene_bbox)

#Use rayshader to convert that raster data to a matrix
dc_elevation_matrix = raster_to_matrix(cropped_data)

#Remove negative elevation data
dc_elevation_matrix[dc_elevation_matrix < 0] = 0

#Plot a 3D map of the national mall
dc_elevation_matrix |>
 height_shade() |>
 add_shadow(lamb_shade(dc_elevation_matrix), 0) |>
 plot_3d(dc_elevation_matrix, zscale=3.7, water = TRUE, waterdepth = 1,
         soliddepth=-50, windowsize = 800)
#Zoom in on the monument
render_camera(theta=45,  phi=0, zoom= 0.03, fov=130)
```

```
#Render the national monument at solar noon on the solstice
rgl::par3d(ignoreExtent = TRUE)
render_multipolygonz(washington_monument_multipolygonz,
                     extent = raster::extent(cropped_data),
                     zscale = 4, color = "grey80",
                     heightmap = dc_elevation_matrix)
#Render using the built-in (but less accurate) Hosek model.
# Here's it's more yellow than it should be, but it's accurate enough for most renders.
render_highquality(
min_variance = 0,
samples = 16,
long = -77.035249,
lat = 38.889462,
iso = 8,
clamp_value = 10000,
datetime = as.POSIXct("2025-12-21 16:00:00", tz = "EST")
)

# Render the more-accurate Praguq model (that requires
# supplemental data that will be downloaded on the first call) and
# specify a higher resolution environment map via `sky_args`
render_highquality(
min_variance = 0,
samples = 16,
long = -77.035249,
lat = 38.889462,
sky_args = list(hosek = FALSE,resolution=4000),
iso = 8,
clamp_value = 10000,
datetime = as.POSIXct("2025-12-21 16:00:00", tz = "EST")
)
}
```

`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	Either an object representing the spatial extent of the scene (either from the `raster`, `terra`, `sf`, or `sp` packages), a length-4 numeric vector specifying `c("xmin", "xmax","ymin","ymax")`, or the spatial object (from the previously aforementioned packages) which will be automatically converted to an extent object.
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(run_documentation()) {
montereybay |>
 sphere_shade() |>
 plot_3d(montereybay,zscale=50,water=TRUE, watercolor="#233aa1",
         zoom=0.9, windowsize = 800)
render_snapshot()
}

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)
if(run_documentation()) {
render_label(montereybay,lat = santa_cruz[1], long = santa_cruz[2],
             extent = attr(montereybay, "extent"), textsize = 2,
             altitude=12000, zscale=50, text = "Santa Cruz")
render_snapshot()
}

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`.
if(run_documentation()) {
render_label(montereybay, lat = monterey[1], long = monterey[2], altitude = 10000,
            extent = attr(montereybay, "extent"), textsize = 2,
            zscale = 50, text = "Monterey", textcolor = "white", linecolor="darkred",
            dashed = TRUE, clear_previous = TRUE)
render_snapshot()
}

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`.
if(run_documentation()) {
render_label(montereybay,lat=canyon[1], long = canyon[2], altitude = 2000,
            extent = attr(montereybay,"extent"), textsize = 2,
            zscale=50,text = "Monterey Canyon", relativez=FALSE)
render_snapshot()
}

#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`.
if(run_documentation()) {
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"), textsize = 2,
            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, textsize = 2,
            extent = attr(montereybay,"extent"), textcolor = "white", linecolor="white",
            text = "Monterey Canyon", relativez=FALSE)

render_highquality(samples = 16,text_size = 64, line_radius = 3, text_offset = c(0, 20, 0),
                   lightdirection = 180, min_variance = 0)
}
if(run_documentation()) {
#Fixed text angle
render_highquality(samples = 16,text_size = 64, line_radius = 3, text_offset = c(0, 20, 0),
                   lightdirection = 180, text_angle = 0, min_variance = 0)
}
#We can remove all existing labels by calling `render_label(clear_previous = TRUE)`
if(run_documentation()) {
render_label(clear_previous = TRUE)
render_snapshot()
}
```

---

render_movie *Render Movie*

Description

Renders a movie using the **av** or **gifski** packages. 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).

Additional arguments are forwarded to `render_snapshot()` for additional customization arguments (like adding titles).

Usage

```
render_movie(
  filename,
  type = "orbit",
  frames = 360,
  fps = 30,
  phi = 30,
  theta = 0,
  zoom = NULL,
  fov = NULL,
  width = NULL,
  height = NULL,
  audio = NULL,
  progbar = interactive(),
  ...
)
```

Arguments

`filename`	Filename. If not appended with `.mp4`, it will be appended automatically. If the file extension is `gif`, the **gifski** package will be used to generate the animation.
`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. Recommmend 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.
`width`	Default `NULL`, uses the window size by default. Width of the movie. Note that the frames will still be captured at the resolution (and aspect ratio) of the rgl window.
`height`	Default `NULL`, uses the window size by default. Height of the movie. Note that the frames will still be captured at the resolution (and aspect ratio) of the rgl window.

audio	Default NULL. Optional file with audio to add to the video.
progbar	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 render_snapshot().

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.

#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))

#Un-comment the following to run
#render_movie(filename = filename_movie, type = "custom",
#             frames = 360,  phi = phivecfull, zoom = zoomvecfull, theta = thetavec)

}
```

---

render_multipolygonz *Render MULTIPOLYGON Z Geometry*

Description

Adds MULTIPOLYGONZ will be plotted in the coordinate system set by the user-specified `extent` argument as-is.

You can also use `save_multipolygonz_to_obj()` manually to convert sf objects

Usage

```
render_multipolygonz(
  sfobj,
  extent = NULL,
  zscale = 1,
  heightmap = NULL,
  color = "grey50",
  offset = 0,
  obj_zscale = TRUE,
  swap_yz = TRUE,
  clear_previous = FALSE,
  baseshape = "rectangle",
  rgl_tag = "_multipolygon",
  ...
)
```

Arguments

`sfobj`	An sf object with MULTIPOLYGON Z geometry.
`extent`	Either an object representing the spatial extent of the scene (either from the `raster`, `terra`, `sf`, or `sp` packages), a length-4 numeric vector specifying `c("xmin", "xmax", "ymin", "ymax")`, or the spatial object (from the previously aforementioned packages) which will be automatically converted to an extent object.
`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.
`color`	Default `black`. Color of the 3D model, if `load_material = FALSE`.
`offset`	Default `5`. Offset of the track from the surface, if `altitude = NULL`.
`obj_zscale`	Default `TRUE`. Whether to scale the size of the OBJ by zscale to have it match the size of the map. If zscale is very big, this will make the model very small.
`swap_yz`	Default `TRUE`. Whether to swap and Y and Z axes. (Y axis is vertical in rayshader coordinates, but data is often provided with Z being vertical).
`clear_previous`	Default `FALSE`. If `TRUE`, it will clear all existing points.
`baseshape`	Default `rectangle`. Shape of the base. Options are `c("rectangle","circle","hex")`.
`rgl_tag`	Default `""`. Tag to add to the rgl scene id, will be prefixed by `"obj"`
`...`	Additional arguments to pass to `rgl::triangles3d()`.

Examples

```
run_examples = length(find.package("sf", quiet = TRUE)) &&
              length(find.package("elevatr", quiet = TRUE)) &&
              length(find.package("raster", quiet = TRUE)) &&
              run_documentation()
if(run_examples) {
library(sf)
#Set location of washington monument
washington_monument_location =  st_point(c(-77.035249, 38.889462))
wm_point = washington_monument_location |>
 st_point() |>
 st_sfc(crs = 4326) |>
 st_transform(st_crs(washington_monument_multipolygonz))

elevation_data = elevatr::get_elev_raster(locations = wm_point, z = 14)

scene_bbox = st_bbox(st_buffer(wm_point,300))
cropped_data = raster::crop(elevation_data, scene_bbox)

#Use rayshader to convert that raster data to a matrix
dc_elevation_matrix = raster_to_matrix(cropped_data)

#Remove negative elevation data
dc_elevation_matrix[dc_elevation_matrix < 0] = 0

#Plot a 3D map of the national mall
dc_elevation_matrix |>
 height_shade() |>
 add_shadow(lamb_shade(dc_elevation_matrix), 0) |>
 plot_3d(dc_elevation_matrix, zscale=3.7, water = TRUE, waterdepth = 1,
         soliddepth=-50, windowsize = 800)
render_snapshot()
}
if(run_examples) {
#Zoom in on the monument
render_camera(theta=150,  phi=35, zoom= 0.55, fov=70)
#Render the national monument
rgl::par3d(ignoreExtent = TRUE)
render_multipolygonz(washington_monument_multipolygonz,
                     extent = raster::extent(cropped_data),
                     zscale = 4, color = "grey80",
                     heightmap = dc_elevation_matrix)
render_snapshot()
}
if(run_examples) {
#This works with `render_highquality()`
render_highquality(min_variance = 0, samples = 16)
}
```

---

render_obj *Render Obj*

Description

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

If no latitudes or longitudes are passed in, the OBJ will be plotted in the coordinate system set by the user-specified `extent` argument as-is. Use this alongside `save_multipolygonz_to_obj()` to plot 3D polygons imported from geospatial sources in the proper location (but for ease of use, use `render_multipolygonz()` to plot this data directly).

Usage

```
render_obj(
  filename,
  extent = NULL,
  lat = NULL,
  long = NULL,
  altitude = NULL,
  xyz = NULL,
  zscale = 1,
  heightmap = NULL,
  load_material = FALSE,
  load_normals = TRUE,
  color = "grey50",
  offset = 0,
  obj_zscale = FALSE,
  swap_yz = NULL,
  angle = c(0, 0, 0),
  scale = c(1, 1, 1),
  clear_previous = FALSE,
  baseshape = "rectangle",
  lit = FALSE,
  light_altitude = c(45, 30),
  light_direction = c(315, 135),
  light_intensity = 0.3,
  light_relative = FALSE,
  rgl_tag = "",
  ...
)
```

Arguments

`filename`	Filename for the OBJ file.
`extent`	Either an object representing the spatial extent of the scene (either from the `raster`, `terra`, `sf`, or `sp` packages), a length-4 numeric vector specifying `c("xmin", "xmax", "ymin", "ymax")`, or the spatial object (from the previously aforementioned packages) which will be automatically converted to an extent object.
`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	Default NULL. Elevation of each point, in units of the elevation matrix (scaled by zscale). If left NULL, this will be just the elevation value at ths surface, offset by offset. If a single value, the OBJ will be rendered at that altitude.
xyz	Default NULL, ignored. A 3 column numeric matrix, with each row specifying the x/y/z coordinates of the OBJ model(s). Overrides lat/long/altitude and ignores extent to plot the OBJ in raw rgl coordinates.
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.
load_material	Default TRUE. Whether to load the accompanying MTL file to load materials for the 3D model.
load_normals	Default TRUE. Whether to load normals for the 3D model.
color	Default black. Color of the 3D model, if load_material = FALSE.
offset	Default 5. Offset of the model from the surface, if altitude = NULL.
obj_zscale	Default FALSE. Whether to scale the size of the OBJ by zscale to have it match the size of the map. If zscale is very big, this will make the model very small.
swap_yz	Default NULL, defaults to FALSE unless plotting raw coordinates (no lat or long passed). Whether to swap and Y and Z axes. (Y axis is vertical in rayshader coordinates, but data is often provided with Z being vertical).
angle	Default c(0,0,0). Angle of rotation around the x, y, and z axes. If this is a matrix or list, each row (or list entry) specifies the rotation of the nth model specified (number of rows/length of list must equal the length of lat/long).
scale	Default c(1,1,1). Amount to scale the 3D model in the x, y, and z axes. If this is a matrix or list, each row (or list entry) specifies the scale of the nth model specified (number of rows/length of list must equal the length of lat/long).
clear_previous	Default FALSE. If TRUE, it will clear all existing points.
baseshape	Default rectangle. Shape of the base. Options are c("rectangle","circle","hex").
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.
light_relative	Default FALSE. Whether the light direction should be taken relative to the camera, or absolute.
rgl_tag	Default "". Tag to add to the rgl scene id, will be prefixed by "obj"
...	Additional arguments to pass to rgl::triangles3d().

Examples

```
if(run_documentation()) {
#Render the 3D map
moss_landing_coord = c(36.806807, -121.793332)
montereybay |>
 sphere_shade() |>
 plot_3d(montereybay,zscale=50,water=TRUE,
         shadowcolor="#40310a", background = "tan",
         theta=210,  phi=22, zoom=0.20, fov=55)

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)

#Create a rainbow spectrum of flags
render_obj(flag_full_obj(), extent = attr(montereybay,"extent"), heightmap = montereybay,
          lat = unlist(circle_coords_lat), long = unlist(circle_coords_long),
          scale=c(2,2,2), angle=c(0,45,0),
          zscale=50, color=rainbow(100), smooth = FALSE, clear_previous = TRUE)
render_snapshot()
}
if(run_documentation()) {
#Rotate the flag to follow the circle
render_obj(flag_full_obj(), extent = attr(montereybay,"extent"), heightmap = montereybay,
          lat = unlist(circle_coords_lat), long = unlist(circle_coords_long),
          scale=c(2,2,2),
          angle=matrix(c(rep(0,100), seq(0,-360,length.out=101)[-1],rep(0,100)),ncol=3),
          zscale=50, color=rainbow(100), smooth = FALSE, clear_previous = TRUE)
render_snapshot()
}
if(run_documentation()) {
#Style the pole with a different color
render_obj(flag_pole_obj(), extent = attr(montereybay,"extent"), heightmap = montereybay,
          lat = unlist(circle_coords_lat), long = unlist(circle_coords_long),
          scale=c(2,2,2),
          angle=matrix(c(rep(0,100), seq(0,-360,length.out=101)[-1],rep(0,100)),ncol=3),
          zscale=50, color="grey20", smooth = FALSE, clear_previous = TRUE)
render_obj(flag_banner_obj(), extent = attr(montereybay,"extent"), heightmap = montereybay,
          lat = unlist(circle_coords_lat), long = unlist(circle_coords_long),
          scale=c(2,2,2),
          angle=matrix(c(rep(0,100), seq(0,-360,length.out=101)[-1],rep(0,100)),ncol=3),
          zscale=50, color=rainbow(100), smooth = FALSE)

#And all of these work with `render_highquality()`
render_highquality(samples = 16)
}
```

---

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

```
render_path(
  lat,
  long = NULL,
  altitude = NULL,
  groups = NULL,
  extent = NULL,
  zscale = 1,
  heightmap = NULL,
  resample_evenly = FALSE,
  resample_n = 360,
  reorder = FALSE,
  reorder_first_index = 1,
  reorder_duplicate_tolerance = 0.1,
  reorder_merge_tolerance = 1,
  simplify_tolerance = 0,
  linewidth = 0.5,
  color = "black",
  antialias = FALSE,
  offset = 5,
  clear_previous = FALSE,
  return_coords = FALSE,
  tag = "path3d"
)
```

Arguments

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 zscale). If left NULL, this will be just the elevation value at ths surface, offset by offset. If a single value, all data will be rendered at that altitude.
groups	Default NULL. Integer vector specifying the grouping of each lat/long path segment, if lat/long are specified as numeric vectors (as opposed to sf or SpatialLineDataFrame objects, where this information is built-in to the object).
extent	Either an object representing the spatial extent of the 3D scene (either from the raster, terra, sf, or sp packages), a length-4 numeric vector specifying c("xmin", "xmax","ymin","ymax"), or the spatial object (from the previously

aforementioned packages) which will be automatically converted to an extent object.

`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`. Pass this if not including an `altitude` argument, or if no extent passed. A two-dimensional matrix, where each entry in the matrix is the elevation at that point. All points are assumed to be evenly spaced.

`resample_evenly`
Default `FALSE`. If `TRUE`, this will re-sample the path evenly from beginning to end, which can help vastly reduce the number of points used to draw it (which can improve the performance of `render_highquality()` and `render_snapshot()` when using `software_render = TRUE`). This function works only if `reorder = TRUE`, or if the sf object is already ordered from beginning to end.

`resample_n` Default `360`. Number of breaks in which to evenly resample the line if `resample_evenly = TRUE`.

`reorder` Default `FALSE`. If `TRUE`, this will attempt to re-order the rows within an `sf` object with multiple paths to be one continuous, end-to-end path. This happens in two steps: merging duplicate paths that have end points that match with another object (within `reorder_duplicate_tolerance` distance), and then merges them (within `reorder_merge_tolerance` distance) to form a continuous path.

`reorder_first_index`
Default `1`. The index (row) of the `sf` object in which to begin the reordering process. This merges and reorders paths within `reorder_merge_tolerance` distance until it cannot merge any more, and then repeats the process in the opposite direction.

`reorder_duplicate_tolerance`
Default `0.1`. Lines that have start and end points (does not matter which) within this tolerance that match a line already processed (order determined by `reorder_first_index`) will be discarded.

`reorder_merge_tolerance`
Default `1`. Lines that have start points that are within this distance to a previously processed line's end point (order determined by `reorder_first_index`) will be reordered within the `sf` object to form a continuous, end-to-end path.

`simplify_tolerance`
Default `0` (no simplification). If greater than zero, simplifies the path to the tolerance specified. This happens after the data has been merged if `reorder = TRUE`. If the input data is specified with long-lat coordinates and `sf_use_s2()` returns `TRUE`, then the value of simplify_tolerance must be specified in meters.

`linewidth` Default `3`. The line width.

`color` Default `black`. Color of the line.

`antialias` Default `FALSE`. If `TRUE`, the line with be 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.

return_coords	Default FALSE. If TRUE, this will return the internal rayshader coordinates of the path, instead of plotting the line.
tag	Default "path3d". The rgl tag to use when adding the path to the scene.

Examples

```
if(run_documentation()) {
#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()
}
if(run_documentation()) {
#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()
}

if(run_documentation()) {
```

```
#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()
}

if(run_documentation()) {
#And all of these work with `render_highquality()`. Here, I set `use_extruded_paths = TRUE`
#to get thick continuous paths.
render_highquality(line_radius=1, min_variance = 0,
                   use_extruded_paths = TRUE, samples = 16)
}
if(run_documentation()) {
#We can also change the material of the objects by setting the `point_material` and
#`point_material_args` arguments in `render_highquality()`
render_highquality(line_radius=1, min_variance = 0, samples = 16,
                   path_material = rayrender::glossy,  use_extruded_paths = TRUE,
                   path_material_args = list(gloss = 0.5, reflectance = 0.2))
}

if(run_documentation()) {
#For transmissive materials (like `dielectric`), we should specify that the path
#should be rendered with an extruded path. We'll use the `attenuation` argument in
#the `dielectric` function to specify a realistic glass color.
render_path(extent = attr(montereybay,"extent"), heightmap = montereybay, clear_previous = TRUE,
           lat = unlist(circle_coords_lat), long = unlist(circle_coords_long),
           zscale=50, color="white", offset=200, linewidth=5)
render_highquality(line_radius=1, min_variance = 0, samples = 16,
                   lightsize = 2000, lightintensity = 10,
                   path_material = rayrender::dielectric, use_extruded_paths = TRUE,
               path_material_args = list(refraction = 1.5, attenuation = c(0.05,0.2,0.2)))
}
```

`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 (unless that value is NA, in which the point will be removed).

Usage

```
render_points(
  lat = NULL,
  long = NULL,
  altitude = NULL,
  extent = NULL,
  zscale = 1,
  heightmap = NULL,
  size = 0.5,
  color = "black",
  offset = 5,
  clear_previous = FALSE
)
```

Arguments

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	Default NULL. Elevation of each point, in units of the elevation matrix (scaled by zscale). If a single value, all data will be rendered at that altitude.
extent	Either an object representing the spatial extent of the 3D scene (either from the raster, terra, sf, or sp packages), a length-4 numeric vector specifying c("xmin", "xmax","ymin","ymax"), or the spatial object (from the previously aforementioned packages) which will be automatically converted to an extent object.
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. This can be a vector (the same length as lat and long) specifying a size for each point.
color	Default black. Color of the point. This can also be a vector specifying the color of each 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(run_documentation()) {
#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", 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()
}
if(run_documentation()) {
#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),
             offset = 0, zscale=50, color="black")
render_camera(theta=30,phi=35,zoom=0.45,fov=70)
render_snapshot()
}
if(run_documentation()) {
#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()
}
if(run_documentation()) {
#And all of these work with `render_highquality()`
render_highquality(point_radius = 1, min_variance = 0, samples = 16)
}

if(run_documentation()) {
#We can also change the material of the objects by setting the `point_material` and
#`point_material_args` arguments in `render_highquality()`
render_highquality(point_radius = 1, min_variance = 0, samples = 16,
                   point_material = rayrender::glossy,
                   point_material_args = list(gloss = 0.5, reflectance = 0.2))
}
```

`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,
  alpha = 1,
  lit = TRUE,
  light_altitude = c(45, 30),
  light_direction = c(315, 135),
  light_intensity = 0.3,
```

```
  light_relative = FALSE,
  clear_previous = FALSE
)
```

Arguments

polygon	sf object, "SpatialPolygon" 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.
extent	Either an object representing the spatial extent of the 3D scene (either from the raster, terra, sf, or sp packages), a length-4 numeric vector specifying c("xmin", "xmax", "ymin", "ymax"), or the spatial object (from the previously aforementioned packages) which will be automatically converted to an extent object.
color	Default black. Color of the polygon.
top	Default 1. Extruded top distance. If this equals bottom, the polygon will not be extruded and just the one side will be rendered.
bottom	Default 0. Extruded bottom distance. If this equals top, the polygon will not be extruded and just the one side will be rendered.
data_column_top	Default NULL. A string indicating the column in the sf object to use to specify the top of the extruded polygon.
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.
alpha	Default 1. Transparency of the polygons.
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.

`light_relative` Default `FALSE`. Whether the light direction should be taken relative to the camera, or absolute.

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

Examples

```
if(run_documentation()) {
#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. You may
#have to call `sf::sf_use_s2(FALSE)` before running this code to get it to run.
sf::sf_use_s2(FALSE)
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 = FALSE)
render_snapshot()
}
if(run_documentation()) {
#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=FALSE,clear_previous=TRUE)
render_snapshot()
}
if(run_documentation()) {
#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 its 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",
                clear_previous = TRUE)
render_snapshot()
}
if(run_documentation()) {
#This function also works with `render_highquality()`
render_highquality(samples = 16, min_variance = 0)
}
```

`render_raymesh` *Render Raymesh*

Description

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

If no latitudes or longitudes are passed in, the raymesh will be plotted in the coordinate system set by the user-specified `extent` argument as-is. Use this alongside `save_multipolygonz_to_obj()` to plot 3D polygons imported from geospatial sources in the proper location (but for ease of use, use `render_multipolygonz()` to plot this data directly).

Usage

```
render_raymesh(
  raymesh,
  extent = NULL,
  lat = NULL,
  long = NULL,
  altitude = NULL,
  xyz = NULL,
  zscale = 1,
  heightmap = NULL,
  load_normals = TRUE,
  change_material = TRUE,
  color = "grey50",
  offset = 0,
  obj_zscale = FALSE,
  swap_yz = NULL,
  angle = c(0, 0, 0),
  scale = c(1, 1, 1),
  clear_previous = FALSE,
  baseshape = "rectangle",
  flat_shading = FALSE,
  lit = FALSE,
  light_altitude = c(45, 30),
  light_direction = c(315, 135),
  light_intensity = 1,
  light_relative = FALSE,
  rgl_tag = "",
  ...
)
```

Arguments

`raymesh`	`raymesh` object (see the rayvertex package for a description)
`extent`	Either an object representing the spatial extent of the scene (either from the `raster`, `terra`, `sf`, or `sp` packages), a length-4 numeric vector specifying `c("xmin", "xmax", "ymin", "ymax")`, or the spatial object (from the previously aforementioned packages) which will be automatically converted to an extent object.

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	Default NULL. Elevation of each point, in units of the elevation matrix (scaled by zscale). If left NULL, this will be just the elevation value at ths surface, offset by offset. If a single value, the OBJ will be rendered at that altitude.
xyz	Default NULL, ignored. A 3 column numeric matrix, with each row specifying the x/y/z coordinates of the OBJ model(s). Overrides lat/long/altitude and ignores extent to plot the OBJ in raw rgl coordinates.
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.
load_normals	Default TRUE. Whether to load normals for the 3D model.
change_material	Default TRUE. Whether to change the raymesh material (to customize the color).
color	Default black. Color of the 3D model, if load_material = FALSE.
offset	Default 5. Offset of the track from the surface, if altitude = NULL.
obj_zscale	Default FALSE. Whether to scale the size of the OBJ by zscale to have it match the size of the map. If zscale is very big, this will make the model very small.
swap_yz	Default NULL, defaults to FALSE unless plotting raw coordinates (no lat or long passed). Whether to swap and Y and Z axes. (Y axis is vertical in rayshader coordinates, but data is often provided with Z being vertical).
angle	Default c(0,0,0). Angle of rotation around the x, y, and z axes. If this is a matrix or list, each row (or list entry) specifies the rotation of the nth model specified (number of rows/length of list must equal the length of lat/long).
scale	Default c(1,1,1). Amount to scale the 3D model in the x, y, and z axes. If this is a matrix or list, each row (or list entry) specifies the scale of the nth model specified (number of rows/length of list must equal the length of lat/long).
clear_previous	Default FALSE. If TRUE, it will clear all existing points.
baseshape	Default rectangle. Shape of the base. Options are c("rectangle","circle","hex").
flat_shading	Default FALSE. If TRUE, this will use rgl's flat shading.
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.

light_relative	Default FALSE. Whether the light direction should be taken relative to the camera, or absolute.
rgl_tag	Default "". Tag to add to the rgl scene id, will be prefixed by "objraymsh"
...	Additional arguments to pass to rgl::triangles3d().

Examples

```
if(run_documentation()) {
}
```

---

render_resize_window *Resize the rgl Window*

Description

Resize the rgl Window

Usage

```
render_resize_window(width = NULL, height = NULL)
```

Arguments

width	Default NULL, no change to the current value. New window width.
height	Default NULL, no change to the current value. New window height

Value

None

Examples

```
#Resize the rgl window to various sizes
if(run_documentation()) {
montereybay |>
 sphere_shade() |>
 plot_3d(montereybay,zscale=50,zoom=0.6,theta=-90,phi=30)
render_resize_window(width = 800, height = 800)
render_snapshot()
}

if(run_documentation()) {
render_resize_window(width = 200, height = 200)
render_snapshot()
}
if(run_documentation()) {
render_resize_window(width = 800, height = 400)
render_snapshot()
}
```

render_scalebar *Render Scale Bar*

Description

Places a scale bar on the map in 3D.

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.
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.
label_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. Seconary 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

```
#Add a scale bar to the montereybay dataset, here representing about 80km
if(run_documentation()) {
montereybay |>
 sphere_shade() |>
 plot_3d(montereybay,theta=45, water=TRUE)
render_scalebar(limits=c(0, 80), label_unit = "km")
render_snapshot()
}
if(run_documentation()) {
#This function works with `render_highquality()`
render_highquality(lightdirection = 250, lightaltitude = 40,
                   scale_text_size = 36, samples = 16)
render_scalebar(clear_scalebar = TRUE)
}
if(run_documentation()) {
#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)
}
if(run_documentation()) {
#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)
}
if(run_documentation()) {
#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)
}
if(run_documentation()) {
#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)
}
if(run_documentation()) {
#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)
}
if(run_documentation()) {
#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()
}
```

---

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(
  filename,
  clear = FALSE,
  title_text = NULL,
  title_offset = c(20, 20),
  title_color = "black",
  title_size = 30,
  title_font = "sans",
  title_bar_color = NA,
  title_bar_alpha = 0.5,
  title_just = "left",
  image_overlay = NULL,
```

```
  vignette = FALSE,
  vignette_color = "black",
  vignette_radius = 1.3,
  instant_capture = interactive(),
  bring_to_front = FALSE,
  webshot = FALSE,
  width = NULL,
  height = NULL,
  software_render = FALSE,
  camera_location = NULL,
  camera_lookat = c(0, 0, 0),
  background = NULL,
  text_angle = NULL,
  text_size = 30,
  text_offset = c(0, 0, 0),
  point_radius = 0.5,
  line_offset = 1e-07,
  thick_lines = TRUE,
  line_radius = 0.25,
  cache_scene = FALSE,
  reset_scene_cache = FALSE,
  new_page = TRUE,
  print_scene_info = FALSE,
  fsaa = 1,
  rayvertex_lighting = FALSE,
  rayvertex_lights = NULL,
  rayvertex_shadow_map = FALSE,
  plot = TRUE,
  ...
)
```

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 NA. If a color, this will create a colored bar under the title.
title_bar_alpha	Default 0.5. Transparency of the title bar.

`title_just` Default `left`. Justification 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.

`vignette_color` Default `"black"`. Color of the vignette.

`vignette_radius`
Default `1.3`. Radius of the vignette, as a porportion of the image dimensions.

`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.

`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 `software_render = TRUE`..

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

`software_render`
Default `FALSE`. If `TRUE`, rayshader will use the rayvertex package to render the snapshot, which is not constrained by the screen size or requires OpenGL.

`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.

`background` Default `NULL`, defaults to device background. Background color when `software_render = TRUE`.

`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 `30`. Height of the text.

`text_offset` Default `c(0,0,0)`. Offset to be applied to all text labels.

`point_radius` Default `0.5`. Radius of 3D points (rendered with `render_points()`). This scales the existing value of size specified in `render_points()`.

`line_offset` Default `1e-7`. Small number indicating the offset in the scene to apply to lines if using software rendering. Increase this if your lines aren't showing up, or decrease it if lines are appearing through solid objects.

thick_lines — Default TRUE. If software_render = TRUE, this will render path segments as thick cylinders. Otherwise, it will render the lines using a single-pixel anti-aliased line algorithm.

line_radius — Default 0.25. The radius of the thick cylinders if thick_lines = TRUE and software_render = TRUE.

cache_scene — Default FALSE. Whether to cache the current scene to memory so it does not have to be converted to a raymesh object each time render_snapshot() is called. If TRUE and a scene has been cached, it will be used when rendering.

reset_scene_cache — Default FALSE. Resets the scene cache before rendering.

new_page — Default TRUE. Whether to call grid::grid.newpage() before plotting the image.

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

fsaa — Default 1. Integer specifying the amount of anti-aliasing applied software_render = TRUE.

rayvertex_lighting — Default FALSE. If TRUE and software_render = TRUE, the scene will use rayvertex lighting when rendering the scene, using the lights specified in rayvertex_lights. If no lights are specified there, they will be pulled from light objects in the rgl scene.

rayvertex_lights — Default NULL. Use rayvertex::directional_light() and rayvertex::point_light() along with the rayvertex::add_light() function to specify lighting for your scene when rayvertex_lighting = TRUE.

rayvertex_shadow_map — Default FALSE. If TRUE and rayvertex_lighting = TRUE along with software_render = TRUE, shadow mapping will also be applied to the rendered scene.

plot — Default TRUE. Whether to plot the snapshot.

... — Additional parameters to pass to rayvertex::rasterize_scene().

Value

Displays snapshot of current rgl plot (or saves to disk), as well as invisibly returns an RGBA rayimg array.

Examples

```
if(run_documentation()) {
montereybay |>
 sphere_shade() |>
 plot_3d(montereybay,zscale=50,zoom=0.6,theta=-90,phi=30)
}

if(run_documentation()) {
render_snapshot()
}
```

```
#Create a title
if(run_documentation()) {
render_snapshot(title_text = "Monterey Bay, California", title_offset=c(0,20),
                title_color = "white", title_bar_color = "black",
                title_font = "Helvetica", title_position = "north")

#Add a vignette effect
render_camera(zoom=0.8)
render_snapshot(title_text = "Monterey Bay, California",
                title_color = "white", title_bar_color = "darkgreen",
                vignette = TRUE, title_offset=c(0,20),
                title_font = "Helvetica", title_position = "north")
}
#Use software rendering to render a scene with shadow mapping
if(run_documentation()) {
montereybay |>
 height_shade() |>
 plot_3d(montereybay, shadow=FALSE, solidlinecolor = NULL)
#No shadows
render_snapshot(software_render = TRUE)
}
if(run_documentation()) {
#Now with shadow mapped shadows, calculated in rayvertex
render_snapshot(rayvertex_lighting = TRUE,
                rayvertex_lights = rayvertex::directional_light(intensity = 1.2,
                                                                direction = c(-1, 1, -1)),
                rayvertex_shadow_map = TRUE, software_render = TRUE)
}
```

`render_tree` *Render Tree*

Description

Adds a 3D representation of trees to an existing 3D scene generated with rayshader. Users can specify the trees' geographical positions using latitude and longitude or the same coordinate reference system as `extent`. Different types of tree models can be used, including a basic and a cone-shaped tree. Users can also use their own custom tree model in OBJ format. The function allows customization of various aspects of the tree, including the color of the crown and the trunk, the size of the crown (the leafy part of the tree) and the trunk, the overall scale of the tree, and the rotation angle around the x, y, and z axes. Users can also specify the minimum and maximum height of the trees to be rendered.

Usage

```
render_tree(
  lat = NULL,
  long = NULL,
```

```
  extent = NULL,
  type = "basic",
  custom_obj_tree = NULL,
  custom_obj_crown = NULL,
  custom_obj_trunk = NULL,
  crown_color = "#22aa22",
  trunk_color = "#964B00",
  absolute_height = FALSE,
  tree_height = NULL,
  trunk_height_ratio = NULL,
  crown_width_ratio = NULL,
  crown_width = NULL,
  trunk_radius = NULL,
  tree_zscale = TRUE,
  min_height = 0,
  max_height = Inf,
  zscale = 1,
  lit = TRUE,
  heightmap = NULL,
  baseshape = "rectangle",
  angle = c(0, 0, 0),
  clear_previous = FALSE,
  ...
)
```

Arguments

`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).
`extent`	Either an object representing the spatial extent of the 3D scene (either from the `raster`, `terra`, `sf`, or `sp` packages), a length-4 numeric vector specifying `c("xmin", "xmax", "ymin", "ymax")`, or the spatial object (from the previously aforementioned packages) which will be automatically converted to an extent object.
`type`	Default `"basic"`. Type of tree. Other built-in option: `"cone"`.
`custom_obj_tree`	Default `NULL`. Instead of using the built-in types, users can also load a custom tree model in OBJ format. This function loads and manipulates the model, assuming the tree model's trunk begins at the origin. Color and specific trunk/crown proportions will be fixed to the model specified, although the overall scale can be changed per-tree via `crown_height`.
`custom_obj_crown`	Default `NULL`. Instead of using the built-in types, users can also load a custom crown model in OBJ format. This function loads a crown model and allows you to control the crown and trunk proportions separately.

custom_obj_trunk
: Default NULL. Instead of using the built-in types, users can also load a custom trunk model in OBJ format. This function loads a trunk model and allows you to control the crown and trunk proportions separately.

crown_color
: Default "darkgreen". Color(s) of the crown.

trunk_color
: Default "#964B00" (brown). Color(s) of the trunk,

absolute_height
: Default FALSE. Default is specifying the tree height directly, relative to the underlying height map. If TRUE, crown_height will specified by the actual altitude of the top of the tree. Total tree height will be crown_height + trunk_height.

tree_height
: Default NULL. Height of the tree, automatically set to 10 if not specified. If absolute_height = TRUE, then this is interpreted as the altitude of the top of the tree in the coordinate reference system used. If absolute_height = FALSE, then this is interpreted as the height of the tree relative to the underlying heightmap.

trunk_height_ratio
: Default NULL. The ratio of the height of the trunk to the total height of the tree. Default is 1/3rd the crown height if type = "basic", and 1/6th the crown height if type = "cone".

crown_width_ratio
: Default NULL. Ratio of the crown width to the crown height. A value of 1 is spherical.

crown_width
: Default NULL. As an alternative to specifying the ratio, you can use this argument to specify the crown width directly.

trunk_radius
: Default NULL, automatically computed. Default is 1/5rd the trunk height if type = "basic", and 1/10th the trunk height if type = "cone".

tree_zscale
: Default TRUE. Whether to scale the size of the tree by zscale to have it match the size of the map. If zscale is very big, this will make the trees very small.

min_height
: Default NULL. Minimum height of a tree. Set to a positive number to filter out trees below that height.

max_height
: Default NA. Maximum height of a tree. Set to a positive number to filter out trees above that height.

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.

lit
: Default TRUE. Whether to apply lighting to the tree.

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.

baseshape
: Default rectangle. Shape of the base. Options are c("rectangle","circle","hex").

angle
: Default c(0,0,0). Angle of rotation around the x, y, and z axes. If this is a matrix or list, each row (or list entry) specifies the rotation of the nth tree specified (number of rows/length of list must equal the length of lat/long).

clear_previous
: Default FALSE. If TRUE, it will clear all existing trees.

...
: Additional arguments to pass to rgl::triangles3d().

Examples

```
if(run_documentation()) {
#Let's first start by drawing some trees in a circle around Monterey Bay
#We won't scale these to a realistic size (yet)
moss_landing_coord = c(36.806807, -121.793332)
montereybay |>
 sphere_shade() |>
 plot_3d(montereybay,zscale=50,water=TRUE,
         shadowcolor="#40310a", background = "tan",
         theta=210,  phi=22, zoom=0.20, fov=55)

t = seq(0,2*pi,length.out=20)
circle_coords_lat = moss_landing_coord[1] + 0.3 * sin(t)
circle_coords_long = moss_landing_coord[2] + 0.3 * cos(t)

render_tree(extent = attr(montereybay,"extent"), heightmap = montereybay,
           tree_zscale = FALSE, tree_height = 30,  lit = TRUE,
          lat = unlist(circle_coords_lat), long = unlist(circle_coords_long), zscale=50)
render_snapshot()
}
if(run_documentation()) {
#Change the crown width ratio (compared to the height)
render_tree(extent = attr(montereybay,"extent"), heightmap = montereybay,
           tree_zscale = FALSE, tree_height = 60, crown_width_ratio = 0.5,
           clear_previous = TRUE,
          lat = unlist(circle_coords_lat), long = unlist(circle_coords_long), zscale=50)
render_snapshot()
}
if(run_documentation()) {
#Change the trunk height and width
render_tree(extent = attr(montereybay,"extent"), heightmap = montereybay,
           tree_zscale = FALSE, tree_height = 40, crown_width_ratio = 2,
           clear_previous = TRUE, trunk_height_ratio=1/2, trunk_radius = 1.5,
          lat = unlist(circle_coords_lat), long = unlist(circle_coords_long), zscale=50)
render_snapshot()
}
if(run_documentation()) {
#Change the tree type
render_tree(extent = attr(montereybay,"extent"), heightmap = montereybay,
           tree_zscale = FALSE, tree_height = 30,
           clear_previous = TRUE, type = "cone",trunk_height_ratio = 1/6,
          lat = unlist(circle_coords_lat), long = unlist(circle_coords_long), zscale=50)
render_snapshot()
}
if(run_documentation()) {
#Change the crown color:
render_camera(theta = 150,  phi = 38, zoom = 0.4, fov = 55)
render_tree(extent = attr(montereybay,"extent"), heightmap = montereybay,
           tree_zscale = FALSE, tree_height = 30, crown_width_ratio = 0.5 + runif(20),
           crown_color = rainbow(20),  clear_previous = TRUE,
          lat = unlist(circle_coords_lat), long = unlist(circle_coords_long), zscale=50)
render_snapshot()
```

```
}
```

`render_water` *Render Water Layer*

Description

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

Usage

```
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 tranparency.
`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

```
if(run_documentation()) {
montereybay |>
 sphere_shade() |>
 plot_3d(montereybay,zscale=50)
render_snapshot()
}

#We want to add a layer of water after the initial render.
if(run_documentation()) {
render_water(montereybay,zscale=50)
render_snapshot()
}

#Call it again to change the water depth
if(run_documentation()) {
render_water(montereybay,zscale=50,waterdepth=-1000)
render_snapshot()
}

#Add waterlines
if(run_documentation()) {
render_camera(theta=-45)
render_water(montereybay,zscale=50,waterlinecolor="white")
render_snapshot()
}
```

---

resize_matrix *Resize Matrix*

Description

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

Usage

```
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. Alteratively cubic, which is slightly smoother, although current implementation slightly scales the image.

Examples

```
#Reduce the size of the monterey bay dataset by half

if(run_documentation()) {
montbaysmall = resize_matrix(montereybay, scale=0.5)
montbaysmall |>
 sphere_shade() |>
 plot_map()
}
if(run_documentation()) {
#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()
}
if(run_documentation()) {
#Increase the size of the volcano dataset 3x
volcanobig = resize_matrix(volcano, scale=3)
volcanobig |>
 sphere_shade() |>
 plot_map()
}
if(run_documentation()) {
#Increase the size of the volcano dataset 2x, using cubic interpolation
volcanobig = resize_matrix(volcano, scale=3, method="cubic")
volcanobig |>
 sphere_shade() |>
 plot_map()
}
```

---

run_documentation *Run Documentation*

Description

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

Usage

```
run_documentation()
```

Value

Boolean value.

Examples

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

`save_3dprint` *Save 3D Print*

Description

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

Usage

```
save_3dprint(filename, maxwidth = 125, unit = "mm", rotate = 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.

Value

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

Examples

```
filename_stl = tempfile()

#Save the STL file into `filename_stl`
if(run_documentation()) {
volcano |>
 sphere_shade() |>
 plot_3d(volcano,zscale=3)
render_snapshot()
```

```
save_3dprint(filename_stl)
}

#Save the STL file into `filename_stl`, setting maximum width to 100 mm
if(run_documentation()) {
volcano |>
 sphere_shade() |>
 plot_3d(volcano,zscale=3)
render_snapshot()
save_3dprint(filename_stl, maxwidth = 100)
}

#'#Save the STL file into `filename_stl`, setting maximum width to 4 inches
if(run_documentation()) {
volcano |>
 sphere_shade() |>
 plot_3d(volcano,zscale=3)
render_snapshot()
save_3dprint(filename_stl, maxwidth = 4, unit = "in")
}
#'#'#Save the STL file into `filename_stl`, setting maximum width (character) to 120mm
if(run_documentation()) {
volcano |>
 sphere_shade() |>
 plot_3d(volcano,zscale=3)
render_snapshot()
save_3dprint(filename_stl, maxwidth = "120mm")
}
```

`save_multipolygonz_to_obj` *Save MULTIPOLYGON Z sf data to OBJ file*

Description

Converts MULTIPOLYGON Z features into a 3D OBJ model

Usage

```
save_multipolygonz_to_obj(sfobj, filename, swap_yz = FALSE)
```

Arguments

`sfobj`	sf object with MULTIPOLYGON Z geometry,
`filename`	Filename of the OBJ to save the 3D model to.
`swap_yz`	Default `TRUE`., Whether to swap and Y and Z axes. (Y axis is vertical in rayshader coordinates, but data is often provided with Z being vertical).

Examples

```
#Convert the built-in Washington Monument MULTIPOLYGON Z data to an OBJ file
obj_temp = tempfile(fileext=".obj")
save_multipolygonz_to_obj(washington_monument_multipolygonz, obj_temp, swap_yz=TRUE)
#Render with rgl
rgl::open3d()
render_obj(filename=obj_temp, xyz=matrix(c(0,0,0),ncol=3), color="red")
render_camera(theta=30,phi=40)
```

---

save_obj *Save OBJ*

Description

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

Usage

```
save_obj(
  filename,
  save_texture = TRUE,
  water_index_refraction = 1,
  manifold_geometry = FALSE,
  all_face_fields = FALSE,
  save_shadow = 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.
`save_shadow`	Default `FALSE`. If `TRUE`, this saves a plane with the shadow texture below the model.

Examples

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

#Save model of volcano
if(run_documentation()) {
volcano |>
 sphere_shade() |>
 plot_3d(volcano, zscale = 2)

save_obj(filename_obj)
}

#Save model of volcano without texture
if(run_documentation()) {
save_obj(filename_obj, save_texture = FALSE)
}

#Make water have realistic index of refraction
if(run_documentation()) {
montereybay |>
 sphere_shade() |>
 plot_3d(montereybay, zscale = 50)

save_obj(filename_obj, water_index_refraction = 1.5)
}
}
```

---

save_png *Save PNG*

Description

A wrapper around `rayimage::ray_write_image()` to write an image to file.

Usage

```
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 = NA,
  title_bar_alpha = 0.5,
  title_just = "left"
)
```

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 NA. If a color, this will create a colored bar under the title.
title_bar_alpha	Default 0.5. Transparency of the title bar.
title_just	Default left. Position of the title.

Examples

```
filename_map = tempfile()

#Save the map into `filename_map`
montereybay |>
 sphere_shade() |>
 save_png(filename_map)
```

---

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.

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(zscale=10,texture=create_texture("#E9C68D","#AF7F38",
                                               "#674F30","#494D30",
                                               "#B3BEA3")) |>
 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

```
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:
if(run_documentation()) {

#Plut using default values
montereybay |>
  texture_shade() |>
  plot_map()
}
if(run_documentation()) {
#Increase the level of detail
montereybay |>
  texture_shade(detail=1) |>
  plot_map()
}
if(run_documentation()) {
#Decrease the level of detail
montereybay |>
  texture_shade(detail=0) |>
  plot_map()
}
if(run_documentation()) {
#Increase the level of contrast
montereybay |>
  texture_shade(contrast=3) |>
  plot_map()
}
if(run_documentation()) {
#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
if(run_documentation()) {
montbay |>
  height_shade() |>
  add_water(detect_water(montbay), color="dodgerblue") |>
  add_shadow(texture_shade(montbay, detail=1/6, contrast = 4,
                           brightness = 3),0.1) |>
  add_shadow(lamb_shade(montbay,zscale=50),0) |>
  plot_map()
}
```

`washington_monument_multipolygonz`

Washington Monument 3D Model as Multipolygon Z Data

Description

This dataset is an `sf` object containing MULTIPOLYGON Z 3D data of the Washington Monument in Washington, DC.

Usage

```
washington_monument_multipolygonz
```

Format

An `sf` object with MULTIPOLYGONZ geometry.

Source

https://opendata.dc.gov/documents/DCGIS::buildings-in-3d/

Examples

```
# See the `render_multipolygonz()` documentation for examples of using this data.
```

Index