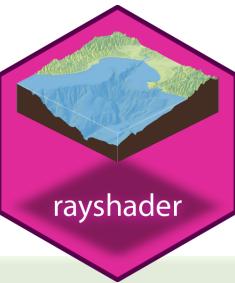


# 2D & 3D Data Visualization with rayshader: : CHEAT SHEET



## Basics

The rayshader package is a simple package with advanced aesthetics. It allows you to create beautiful 2D and 3D visualizations directly from elevation data. This package is specialized in creating map visualizations but is not limited to just that. The features within rayshader allow you to create just about any 3D visualization you want, including something as creative as a GIF. This package does not require you to install different packages to effectively design a map. This cheat sheet will give a brief introduction of the mapping possibilities that can be created with rayshader.

## INSTALLATION

```
install.packages("rayshader")
```

## MAP

Examples in this cheatsheet will be done with rayshaders built-in map, "montereybay" based in the Monterey Canyon, Monterey Bay, California.

```
your_data <- montereybay
```

Try the examples out using YOUR data or follow along using montereybay!

## ADDING LAYERS WITH RAYSHADER:

Layers are added using the pipe operator.

```
your_data %>%  
  <SHADE> %>%  
  <OVERLAY> %>%  
  <RENDER> %>%  
  plot_map()
```

Possible Layers :

- add\_overlay
- add\_shadow
- add\_water

## SPECIAL FEATURES

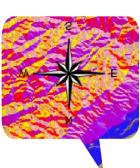
This package is equipped with the ability to transform a **ggplot()** into a 3D visualization.

## Mapping



### sphere\_shade()

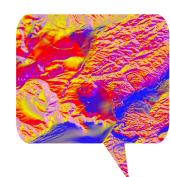
Calculates a color for each point on the surface. Piping in `sphere_shade` alone will **only** give the hillshade, "Realistic View"



### generate\_compass\_overlay()

Give your map a sense of direction, add a compass. Default arguments were used in the making of this compass.

```
your_data %>%
```



### create\_texture()

Allows you to create different pallets of texture.  
(`lightcolor`, `shadowcolor`, `leftcolor`, `rightcolor`, `centercolor`)

```
sphere_shade(texture = create_texture("red", "pink", "yellow",  
  "purple", "blue")) %>%  
(generate_compass_overlay(heightmap = your_data)) %>% plot_map()
```

## ADDING OVERLAY

```
b %>% add_overlay() %>% plot_map()
```

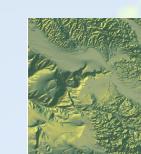


`add_overlay(height_shade(your_data), alphalayer = 0.6)`

Overlays an image with a transparent layer on the current map

## COMBINING SHADOW MAPS

```
b %>% <add_shadow(*)> %>% plot_map()
```



`add_shadow(*ray_shade(your_data,sun altitude=20,zscale=50),max_darken=0.3)`

Layer different shadows onto map and to create desired textured array

## WATER

```
your_data %>% sphere_shade(texture) %>% add_water(detect_water(your_data,  
  cutoff = 0.999),color )
```

\*`detect_water(watermap, color)`

Detects bodies of water within elevation matrix. You will need to specify "cutoff", the lower limit "z" value that you would consider to be water.

`add_water()`

Adds a layer of water to a map



```
your_data %>%  
  sphere_shade(texture =  
  "imhof3") %>%
```

Before and after water addition



```
add_water(detect_water(your_data,  
  cutoff = 0.999), color = "blue") %>%  
  plot_map()
```

## Transformation with ggplot2

The **rayshader** package can be used in combination with **ggplot** to turn a basic 2D visualization into 3D creation.

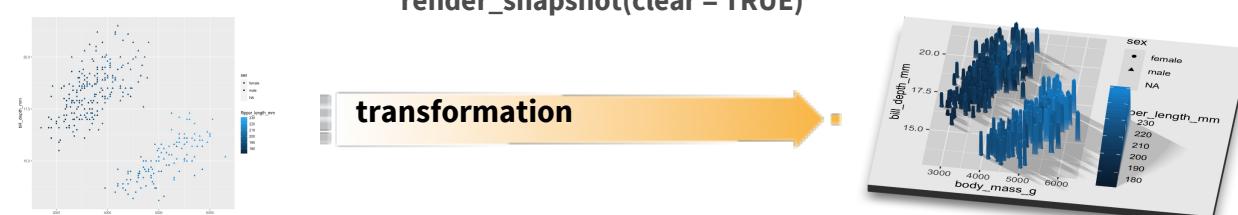
Data used to make these plots was borrowed from the Palmer Penguin package

```
library(palmerpenguins)
```

```
ggplot() + geom_point(data , mapping = aes(x , y , color , shape )) + scale_alpha_continuous()
```

```
plot_gg(..., width, sunangle, windowsize, zoom, theta, phi)
```

```
render_snapshot(clear = TRUE)
```



## More to Explore ~

### SHADE OPTIONS

There are several different shading options within rayshader. Use the pipe operator to directly pipe a shading layer onto the map. If no texture is specified, it will use the default.

\*`add_overlay()` & `add_shadow()` can also be used in combination with the different shading options\*

### ambient\_shade()

Generates a hillshade, darkens areas that have less light from atmosphere

### height\_shade()

Elevation to color shading, with ability to change pallett

### ray\_shade()

raytraces the data, northeast

### texture\_shade()

Allows you to adjust level of detail, contrast and brightness of map

### PLOT

Aside from `plot_map()` you can also plot a 3D interactive visualization that can analyzed

### plot\_3d()

Transforms your data into a 3D visualization. This will open up your map in a new window

### RENDER MAP

After you have created your visualization use the render commands to add or change current aspects of your map!

### render\_camera()

Changes camera settings around scene of map

### render\_highquality()

Give your map a more realistic, highquality feel

### render\_label()

Adjust labels on map

### render\_snapshot()

Captures or saves view of rdl. Use this to capture and view as well

### SAVING YOUR MAP

After plotting your 3D visualization, `save_obj()` will write the object to an obj fil. \*Map must be open at the time of saving\*

# elevatr your own maps: : CHEAT SHEET



elevatr

## Basics

Are you sick of spending unnecessary hours searching for tiles? Are you tired of trying to combine your elevation data layers? Look no more, this package was made for YOU!

The elevatr package takes the hassle out of obtaining elevation data. When used in combination with the sf package you can simplify the entire process down to only 5 steps!

This cheat sheet will take you through the steps of how to find your location coordinates, convert your location into an sf object and then prepare the data to be used in another package called rayshader. rayshader will then take this elevation data of only a few lines and transforms it into a 2D or 3D visualization.

The possibilities are endless. From beginners to more advanced programmers, these packages will give you exactly what you need to create your own elevation map.

## WHERE DOES THIS DATA COME FROM ?

elevatr is equipped with the ability to pull elevation data from AWS Open Data Terrain Tiles and the Open Topography Global datasets API for raster digital elevation models. It is also capable of obtaining point elevation data from the USGS Elevation Point Query Service

## REQUIRED PACKAGES

```
install.packages(elevatr)
install.packages(raster)
install.packages(sf)
install.packages(rayshader)
```

## PACKAGE FUNCTIONS

name	description
get_elev_point	Returns a SpatialPointsDataFrame or sf object.
get_elev_raster	Returns a RasterLayer. *This function will be used in examples*

## CREDITS

<https://cran.r-project.org/web/packages/elevatr/elevatr.pdf> - Developer: Jeffrey Hollister, version 0.4.1, published 2021-07-22

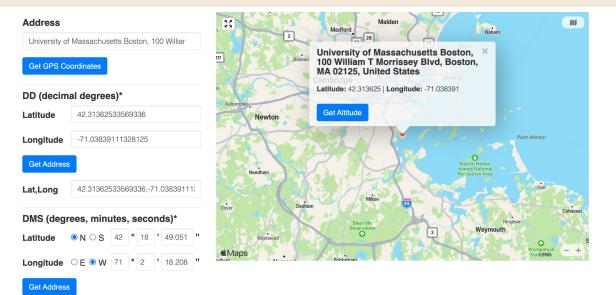
<https://cran.r-project.org/web/packages/rayshader/rayshader.pdf> - Developer: Tyler Morgan-Wall, version 0.24.10, published 2021-04-28

## Location

1. Find a set of coordinates that you are interested in. Go to :

```
https://www.gps-coordinates.net
```

In the address bar, type in your location, and note the latitude and longitude that is given.



2. Next, save that latitude and longitude into a data frame. x = longitude, y = latitude

```
mt_carrigain <- data.frame(x = -71.4474, y = 44.0945)
```

## Ready to Map



### SHADE AND TEXTURE

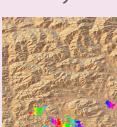
Transform data into a 2D visualization with a preset texture or create your own using `create_texture()`



```
nh_elmat %>%
  sphere_shade(texture = "desert") %>%
```

### WATER

Add water, (set a cutoff - the lower the z-component of the unit normal vector to be classified as water), specify the color you want to use. This can be one that is built into the rayshader package such as 'imhof1','imhof2','imhof3','imhof4', 'desert', 'bw', and 'unicorn'



```
add_water(detect_water(nh_elmat, cutoff = 0.999, zscale = 10),
           color = "unicorn") %>%
```

### 3D PLOT



```
plot_3d(nh_elmat, zscale, fov,
         theta, zoom, phi, windowsize =
         c(800, 800))
```

## Conversion

3. Now that you have your location, you are going to convert it into an sf object using `st_as_sf()`. You will also need to specify your coordinate reference system. (crs)

```
st_as_sf(coords = c("x", "y"), crs = 4326)
```

elevatr 4. Then, use the function `get_elev_raster()` to retrieve your elevation data from the data frame that you created with your location info. This will return a raster object, set a zoom between 1-14.

```
elev <- get_elev_raster(mt_carrigain, z = 12)
```

## GETTING YOUR DATA RAYSHADER READY



5. Last, you are going to convert your raster data into a matrix format using the function `raster_to_matrix`

```
nh_elmat = raster_to_matrix(elev)
```

## THE FULL CODE

The pipe operator is used in both elevatr and rayshader

```
mt_carrigain <- data.frame(x = -71.4474, y = 44.0945) %>%
  st_as_sf(coords = c("x", "y"), crs = 4326)
elev <- get_elev_raster(mt_carrigain, z = 10)
nh_elmat = raster_to_matrix(elev)
```

## Another Example

```
boston_harbor <- data.frame(x = -70.98283, y = 42.34529) %>%
  st_as_sf(coords = c("x", "y"), crs = 4236)
boston_elev <- get_elev_raster(boston_harbor, z = 10)
boston_elmat = raster_to_matrix(boston_elev)
```