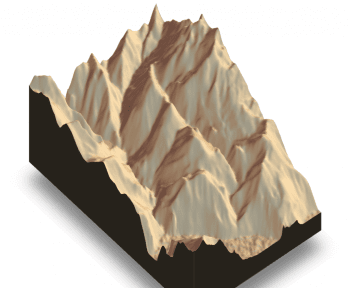
**Using the amazing package rayshader I wanted to render a video of my tour to Alpe d’Huez. Now I created an R package that can use any GPX file and return a 3D video animation from it.**

In July 2019 my friend Tjark and I went to France to cycle the 21 hairpin bends to Alpe d’Huez. The climb has a distance to the summit (at 1,860 m (6,102 ft)) of 13.8 km (8.6 mi), with an average gradient of 8.1% and a maximum gradient of 13%. Two years before we climbed up the amazing Mont Ventoux in Provence, and now we wanted to do Alpe d’Huez. Due to a lack of Hotels, we stayed up at the village itself. We started our tour facing down the Col de Sarenne, via Mizoën and through the valley up the 21 hairpins. It was hot, it was steep, it was exhausting.

I was not at my best fitness. But I am a data scientist and I wanted to know how “slow” was I? There are several ways to find out. I could look at my average speed, a line chart or do a **video**. Even Strava, the app I used for tracking, has a built-in app to make a video called FlyBy. This tool is just two- dimensional. But on twitter and github I found the amazing package rayshader. It allows creating a 3D landscape out of any elevation data. Moreover, you can overlay your landscape with different maps or light conditions. So I thought, why don’t I put my GPS data on top of it? Said and done:

In the next chapters, I will explain how you can create your own video from any .gpx file with package rayshaderanimate.

**Downloading Elevation data**



rayshader 3D graphic of the landscape around Alpe d’Huez

My first problem was getting the elevation data around Alpe d’Huez. Thus I found the great package raster. You can download elevation data for a boundary box all over the world by calling:

raw\_data <- raster::getData("SRTM", lon = bbox[1, 1], lat = bbox[2, 2]))

data\_cropped <- raster::crop(raw\_data, raster::extent(bbox))

I wrapped this function inside package rayshaderanimate. It would allow you to download the data. It then filters the data for the boundary box (not done by raster). Afterward, you can transform it into a rayshader readable format:

library(rayshader)

library(rayshaderanimation)

bbox <- data.frame(min = c(6.022108, 45.012030),

max = c(6.179801, 45.11066))

el\_mat <- get\_elevdata\_from\_bbox()

elmat\_rayshade <- el\_mat %>% unlabel\_elevdata()

elmat\_rayshade %>%

sphere\_shade(texture = "desert") %>%

plot\_3d(elmat\_rayshade, zscale = 22,

fov = 0, theta = 135, zoom = 0.75,

phi = 45, windowsize = c(1000, 800))

Please note that the SRTM data has a resolution of about 250m. If you would like a higher resolution you would need to register at the EU-DEM portal at: <https://land.copernicus.eu/imagery-in-situ/eu-dem/eu-dem-v1.1?tab=downloada> .

There you can download large GEOTiff data. I also wrapped loading this data inside my package:

el\_mat\_eudem <- get\_elevdata\_from\_bbox(bbox = bbox,

type ="EUDEM",

file = "eu\_dem\_v11\_E40N20/eu\_dem\_v11\_E40N20.TIF")

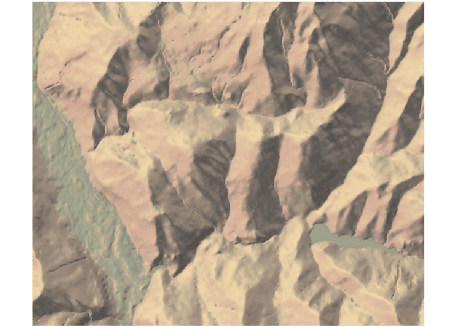
The difference between EU-DEM and SRTM data can be visualized using this code:

plot\_2d\_elevdata(el\_mat %>% unlabel\_elevdata)

plot\_2d\_elevdata(el\_mat\_eudem %>% unlabel\_elevdata)



SRTM elevation profile around Alpe d’Huez



EU-DEM elevation profile around Alpe d’Huez

**Reading in GPX files**

The next task for me was to read in my gps data. The plotKML package has a function for that. I wrapped it inside my package. It outputs my GPX file as a table with longitudinal, latitudinal coordinates and a time vector. The table gets stored in the gpx\_table variable and the boundary box gets stored inside the bbox variable.

gpx\_file\_loc <- system.file("Alpe\_d\_Huez.gpx", package="rayshaderanimate")

gpx\_table <- get\_table\_from\_gpx(gpx\_file\_loc) %>%

get\_enriched\_gpx\_table()

bbox <- get\_bbox\_from\_gpx\_table(gpx\_table)

**Making a ggplot video from elevation data and GPS data**

Animating the line of the GPS data means painting it on top of the landscape. I came to the conclusion that I need to paint every video scene image by image. Meaning if the line should move for two seconds, I would need 48 images to get a frame-rate of 24 images per second.

I did not find any better way than creating a 2D graphic of the elevation data and rendering it as a 3D ggplot. Meaning each step of the video has to be rendered as a ggplot. Afterward, using rayshader I make a 3D image out of this. I take a snapshot and continue the process with a longer line of my GPS data.

**The trouble with GPS data** is that it does not get captured in equally distributed time frames. Sometimes my phone would capture my position every second, sometimes every 30 seconds. So first I needed to create a function that equally distributes the time from my GPX table. The function is inside my package:

video\_indeces <- get\_video\_indeces(time\_data = gpx\_table$time\_right,

number\_of\_screens = number\_of\_screens)

where *number\_of\_screens* is the number of frames going into the video. In my case, I wanted to capture ~300 screens to make it a ten second video.   
For each screen, I needed to paint a ggplot. Ggplot needs the elevation data in a long format. This call will transform the elevation data to a ggplot format:

elmat\_long <- get\_elevdata\_long(el\_mat)

From the elevation data, the gpx table and the video indeces I can create every snapshot by rendering a ggplot:

my\_plot <- ggplot() +

geom\_tile(

data = elevdata\_long,

aes\_string(

"as.numeric(as.character(variable))",

"deg\_elmat\_lat",

fill = "value"),

alpha = 0.75) +

scale\_x\_continuous(

paste0("Longitude | ",

gpx\_table$time[video\_indx]),

expand = c(0,0)) +

scale\_y\_continuous("Latitude", expand = c(0,0)) +

scale\_fill\_gradientn("Elevation",

colours = terrain.colors(10)) +

coord\_fixed() +

geom\_path(

data = gpx\_table[1:video\_indx, ],

aes\_string(x = "lon", y = "lat",

color = "-rel\_speed"),

shape = 15, size = 1, stroke = 0) +

scale\_color\_viridis\_c(option = "A") +

guides(colour=FALSE)

As you can see I inserted a column rel\_speed inside the gpx table to make faster pieces of the track darker. The x-axis label will show the real time of each image being captured.

To render this plot as a 3D graphic rayshader provides the function *plot\_gg*. The image changes by tweeking angles and zooms. To take a picture from the 3D graphic I used the *render\_snapshot* function.

plot\_gg(my\_plot, shadow\_intensity = 0.7, width = 5, height = 5,

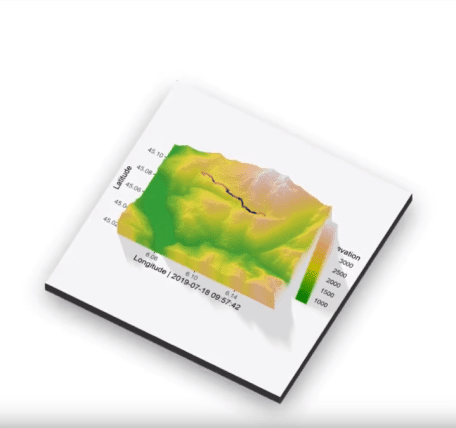
multicore = TRUE, scale = 350, raytrace = TRUE)

render\_snapshot(filename = file.path(tempdir(),

paste0("video", video\_indx, ".png")),

clear = TRUE)

rayshader image of a 3D landscape with a GPS data overlay and time label on the x-axis

The difficult part is rendering all 300 images into a video. [ffmpeg](https://www.ffmpeg.org/) provides a simple API under Windows for this task. The links to all images get stored inside a txt file. From this text file ffmpeg can render the video as an mp4.

ffmpeg -y -f concat -r 24 -safe 0 -i "video\_path.txt" -vf "fps=24,format=yuv420p" output.mp4

I also added functionality to render videos as gifs inside my package. Although I do not recommend rendering them as a gif. The gif files can become rather large. For more details on creating the video, please read the package vignette of rayshaderanimate package.

**How about a storytelling video?**

The video rendered up-to-now does not look like the video I showed at the top. I wanted to use the rayshader sphere shade with a map overlay for the video, too. While trying an image overlay with my data from Alpe d’Huez I noticed, that just the EUDEM data has a resolution that is high enough to render a sphere shade.

But there were certain tweeks needed. I’ll not describe them in detail, but the major problem was rayshader beeing programmed for having data from the US, meaning west of Greenich Meridian. My data points are located east of Greenich Meridian. The function *get\_elevdata\_list* to overcome this problem.

My function *get\_image\_overlay* can be used to derive an overlay image for a certain area. Adding the overlay image can simply be done by using the functionalities of rayshader, meaning *add\_overlay* and *plot\_3d*.

# Format elevation data for west of Greenich

elevation\_data\_list <- get\_elevdata\_list(el\_mat\_eudem)

elevation\_matrix <- elevation\_data\_list$elevation\_matrix

# Calculate shadow and water

elevation\_matrix %>%

sphere\_shade(texture = "desert") %>%

add\_water(detect\_water(elevation\_matrix), color = "desert") %>%

add\_shadow(ray\_shade(elevation\_matrix,

zscale = 3, maxsearch = 300), 0.5)

# Receive Overlay Image from arcgis with a specific boundary box

bbox <- get\_bbox\_from\_gpx\_table(gpx\_table, arcgis = TRUE)

overlay\_img <- get\_image\_overlay(bbox\_arcgis)

# Plot 3D with overlay image

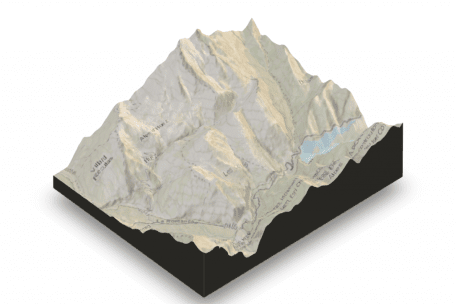
elev\_elem <- elev\_elem %>% add\_overlay(overlay\_img, alphalayer = 0.5)

elev\_elem %>%

plot\_3d(elevation\_matrix, zscale = 15, fov = 1, theta = 280,

zoom = 1.5, phi = 60, windowsize = c(1200, 800))

3d rayshader landscape with image overlay in the French Alps



Now the only things missing are the place markers and the route. Both can be added from the *gpx\_table*. To do this I needed to map the latitude and longitude of the *gpx\_table* to my elevation matrix resulting in *lat\_idx* and *lon\_idx*. Additionally, I added a label to some places, as you can see in this case it is place number 100.

for (i in 1:100) {

render\_label(elevation\_matrix,

x = gpx\_table[1, "lon\_idx"],

y = gpx\_table[1, "lat\_idx"],

z = 100, zscale = 15,

text = NULL, textsize = 15,

linewidth = 6, freetype = FALSE,

color = "#0f9ad1")

}

render\_label(elevation\_matrix,

x = gpx\_table[100, "lon\_idx"],

y = gpx\_table[i, "lat\_idx"],

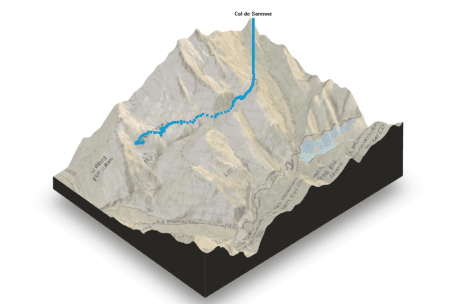
z = 2200, zscale = 15,

text = gpx\_table[100, "label"],

textsize = 1, linewidth = 7, freetype = FALSE,

color = "#0f9ad1", family = "mono", antialias = TRUE)

3d rayshader landscape with route and label in the French Alps



I perform this process for each single point of the *gpx\_table.* At each point I take a snapshot by *rayshader::render\_snapshot.* All snapshots will be stored and converted to a video by *ffmpeg.* I added some additional features as an elevation profile or a title image. Those were added to the snapshots using *magick::image\_append.*

**Why the video?**

Two-dimensional animations cannot really describe the loss of speed uphill. The human perception of mountains is not represented by elevation lines. But elevation lines are the only way to visualize them in two-dimensional plots. The rayshader package allows a way better impression height and steepness. This is why I wanted to use it to visualize my climb to Alpe d’Huez.

Hairpin bends at the mid range of the Alpe d’Huez climb

Now please enjoy watching the video of my cycling climb. You can truly see at the end of the video how I was suffering from the heat at the 21 hairpins. The last passage of the video describes me being slow. The color of the GPS line is bright and it takes long until it reaches the mountain top. Just to give you an impression I added a photo of the lowest 8 hairpins.