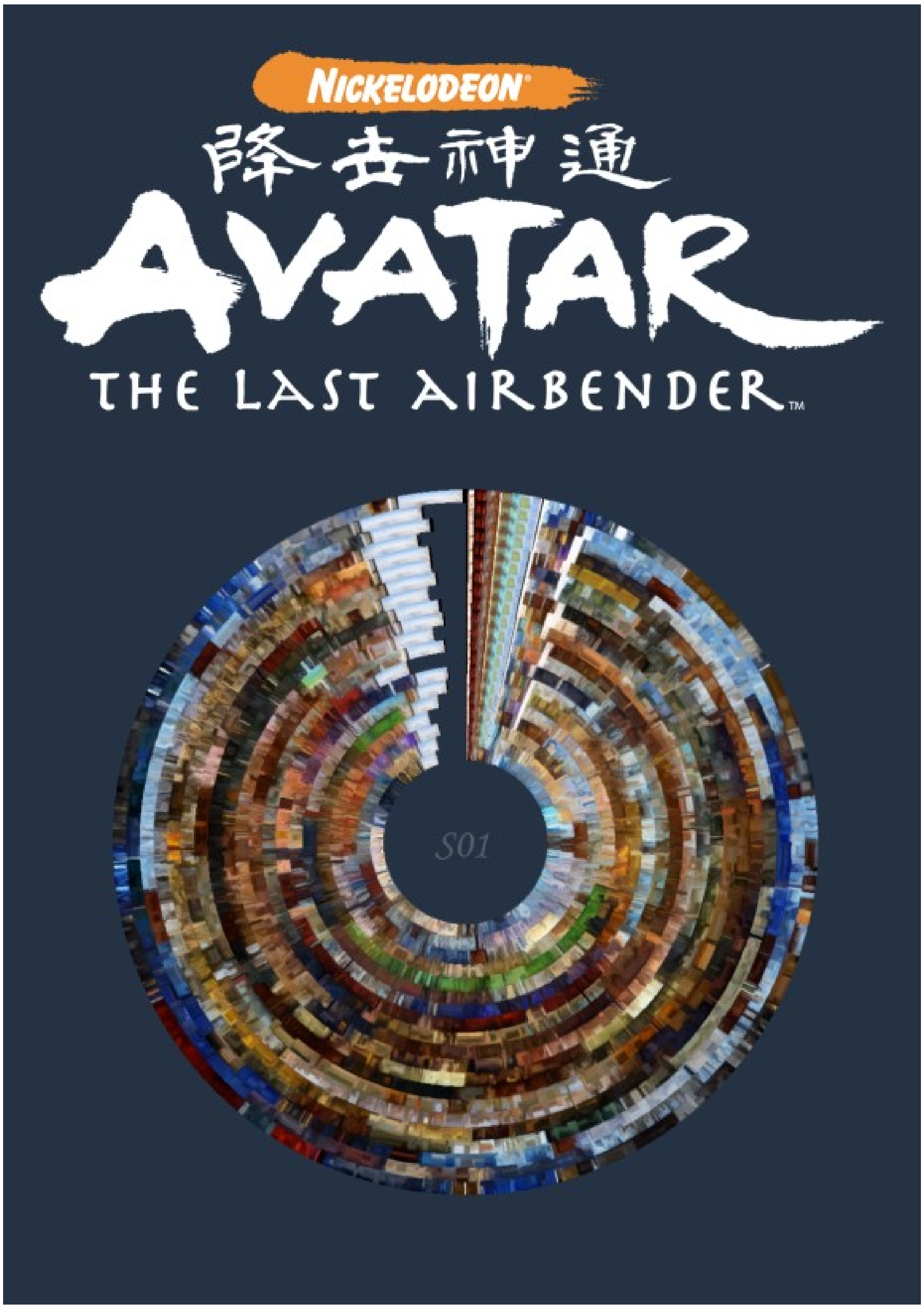
This post is going to be a little different from my usual, as it relies heavily on calls to `ffmpeg`. This should work on Linux/OSX just fine, but the system call will need to be modified for Windows.



Project Setup

The first part of this will rely on you having the video files on your computer. You could recreate this by playing the video and taking screen captures at 1 second intervals, but having the video files will be a lot faster.

For this tutorial, I used the path `./import/avatar/s01` for the video files. The name of the video files shouldn't matter as long as when sorted alphabetically, they are in the correct order.

Required Libraries

The next step is to load the reqired libraries. Glue just makes the readability much easier.

```
library(tidyverse)
library(glue)
library(ggthemes)
library(png)
```

You will also need `ffmpeg` installed. You can do this on OSX with `brew install ffmpeg` or on Linux with `sudo apt-get install ffmpeg`. This will be needed for the bulk of the work.

Create the Frames Directory

We will be writing the frames to `./export/frames/s01` for this example. I was initially going to save all 3 seasons for the tutorial, but ran out of disk space.

```
dir.create("./export")
dir.create("./export/frames")
dir.create("./export/frames/s01")
dir.create("./export/1px")
```

The `1px` exports won't be as useful as the the `frames` exports, otherwise I would recommend seperating into seasons as well. The `1px` folder can just be dumped upon completion or when rerunning with different scaling parameters.

Creating Frames

Now it is time to put `ffmpeg` to work! We will be reading in all of the files in the `s01` directory, extracting the season and episode pattern (ex: `S01E01`), and creating a frame capture for every second. We are storing the frames in the season and episode directory (ex: `S01E01`) using a 4 digit numerical string (ex: `img0001.png`). If you create captures more frequently than every second, or the videos are longer, you may want to increase the digits. For `S01E01` the last frame captured for me was `img1272.png`.

```
list.files("./import/avatar/s01") %>% lapply(function(file_name) {
  episode <- file_name %>% str_extract("S[0-9]{2}E[0-9]{2}")
  episode %>% print
    # Create directory for episode
  dir.create(glue('./export/frames/s01/{episode}'))
    # Generate a frame every second of the episode
  system(glue('ffmpeg -i "import/avatar/s01/{file_name}" -vf fps=1 ./export
/frames/s01/{episode}/img%04d.png'))
})
```

Resizing Frames to 1px Width

Now that we have the frames all ready, you might want to look at your disk space. We just added tens thousands of png files. If you are fine on disk space, we can generate even more images!

The code below reads in all of the frames from each episode and generates a 1×20 pixel image. There is likely a faster way of doing this.

```
list.files("./export/frames/s01") %>% lapply(function(episode){
  list.files(glue("./export/frames/s01/{episode}")) %>%
lapply(function(file_name){
    dir.create(glue('./export/1px/{episode}'))
        # Generates a 1px width by 20px height image
    system(glue("ffmpeg -i ./export/frames/s01/{episode}/{file_name} -vf
scale=1:20 ./export/1px/{episode}/{file_name}"))
  })
})
```

Reading in the Data

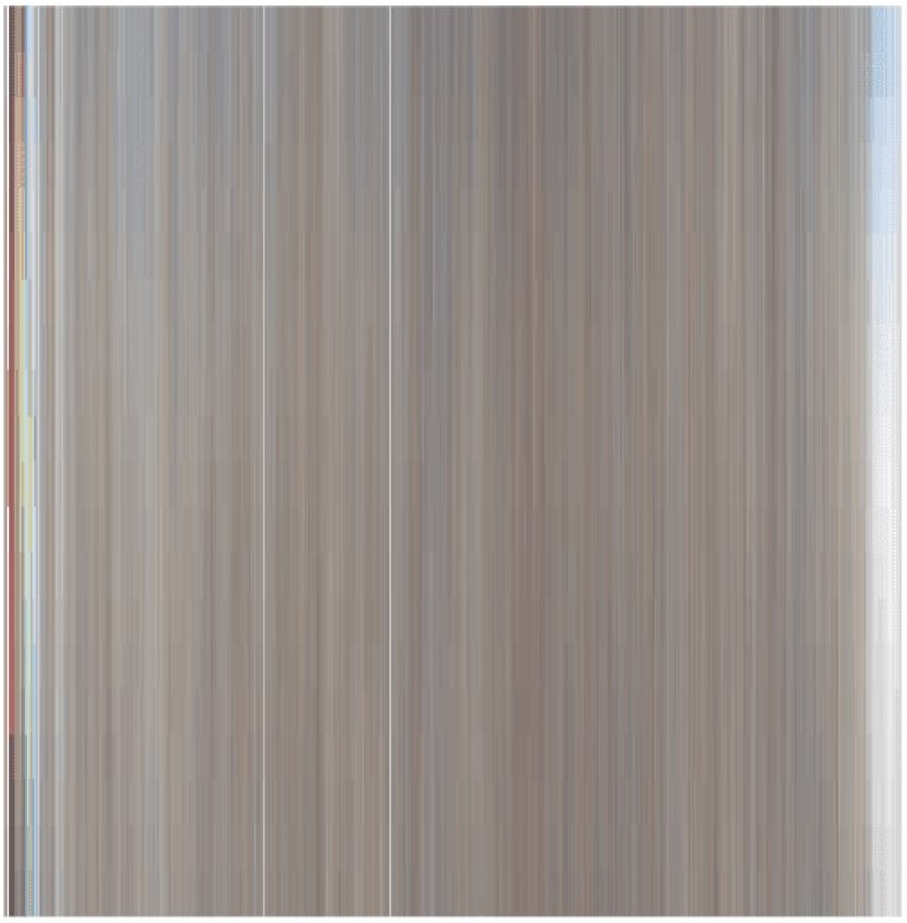
Now to the core of the visualization. I found readPNG works best for me. Esentially we are looping through the multidimentional array provided by readPNG, where the last dimention is an array of RGB values for the `x` and `y` coordinates of the image.

```
barcodes <- list.files("./1px") %>% lapply(function(episode){
  columns <- list.files(glue("./1px/{episode}")) %>% lapply(function(file_name){
    pixel_file <- readPNG(glue("./1px/{episode}/{file_name}"))
    hex <- c()
    for(i in 1:length(pixel_file[,1,1])) {
      pixel <- pixel_file[i,1,]
      hex <- append(hex,rgb(pixel[1], pixel[2], pixel[3]))
    }
    return(data.frame(column = file_name %>% str_extract("[0-9]+") %>%
as.integer(),
                      row = length(hex):1,
                      values = hex))
  })
  barcode <- columns %>% bind_rows()
  barcode$episode <- episode
  return(barcode)
})
```

This will bind the rows in a long format, resulting in a dataframe that will look like this.

column	row	values
1	20	#020003
1	19	#010002
1	18	#020002
1	17	#020002
1	16	#010002
1	15	#020002
1	14	#010002
1	13	#020002
1	12	#020002
1	11	#010002
1	10	#010002
1	9	#020002
1	8	#010002
1	7	#010002
1	6	#020002
1	5	#010002
1	4	#010001
1	3	#020002
1	2	#010002
1	1	#010002

The first column (`column`) is is defines the x position, and the column `row` defines the y position. Right now, you would get a lot of pixels overlapping if you tried to plot the data.



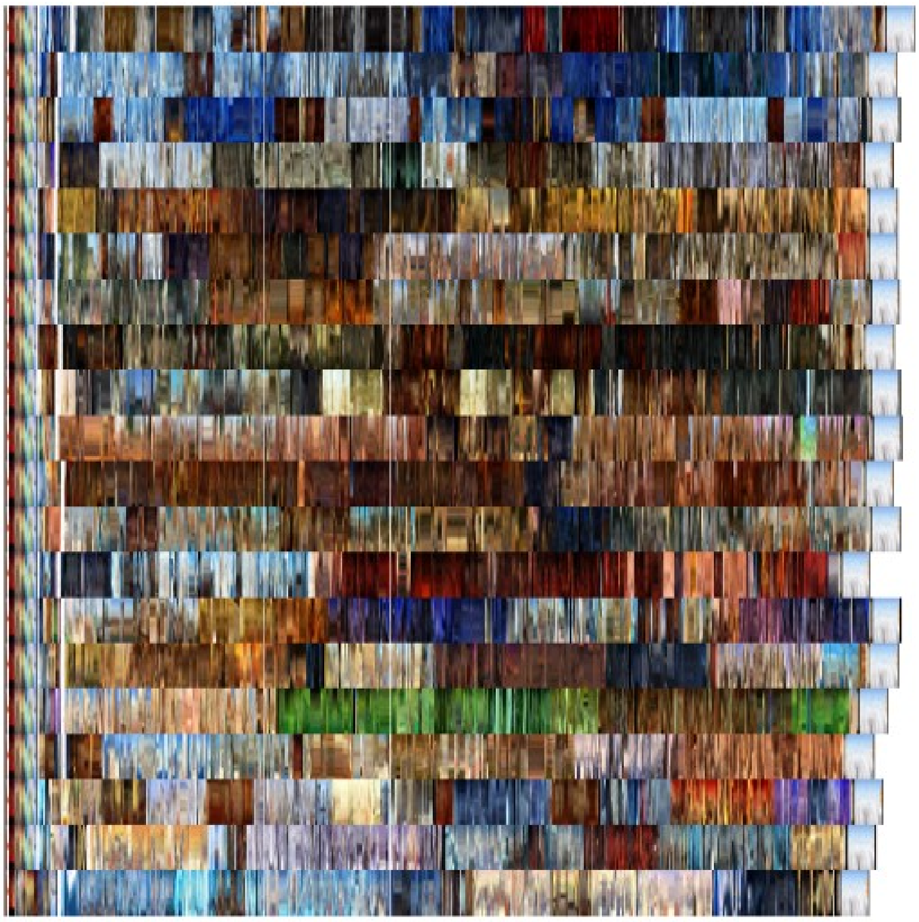
While this is pretty cool, it isn’t exactly what we are going for. If we want the episodes to stack vertically, we will need to create an offset for row y (`row`) field based on the episode number. Since we made the 1px wide images 20px tall, we will offset by 20.

```
barcodes$offset <- barcodes$episode %>%
  str_extract_all('[0-9]+$') %>%
  as.numeric()
barcodes$real_row <- (20 * barcodes$offset) + barcodes$row
```

Ready to Graph

Alright, we made it! We are ready to graph! We are using `theme_map()` just to hide all of the regular plot options, like axis, tick marks, and titles.

```
p1 <- ggplot(barcodes, aes(x=column,y=real_row,fill=values)) +
  geom_tile() +
  scale_fill_manual(values = unique(barcodes$values), limits =
unique(barcodes$values)) +
  theme_map() + theme(legend.position = "none")
p1
```



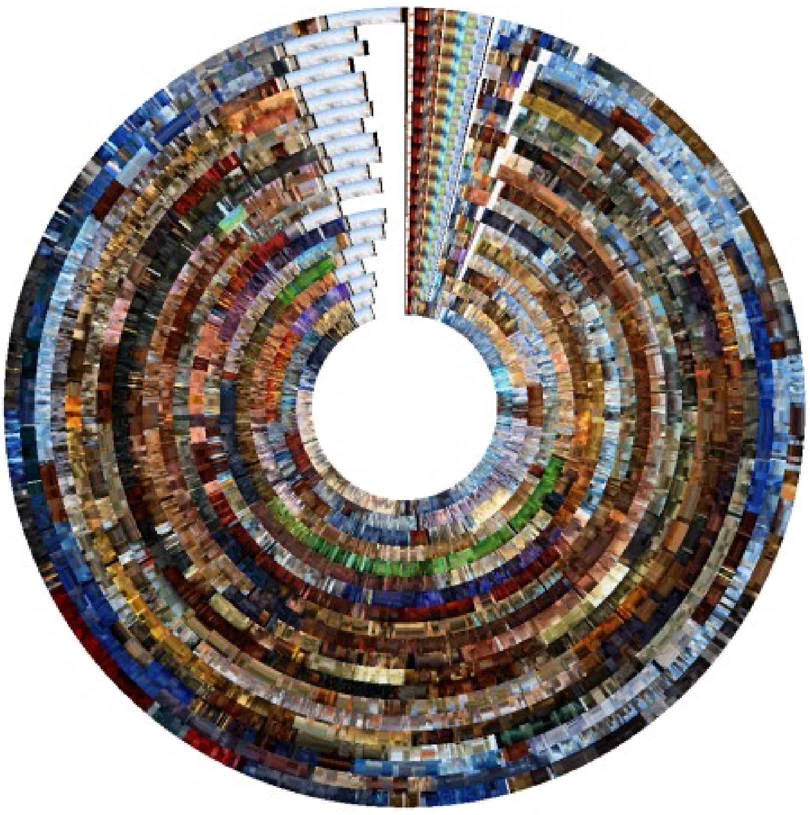
Now this is interesting. The episodes aren't all of the same length. We can fix this by just sampling an equal number of frames and ordering sequentially between the opening and ending credits. That can be a little bonus for you to implement.

Polar Plunge

Polar coordinates are slow. They are soooooo slow. This is due to resizing all of those little rectangluar tiles based on the position. There are faster ways of doing this, but not in R (that I know of).

We also want to expand the limit of the y-axis for the nice donut effect. The closer to the center of the image, the more warped the frames get. This will also give you some room to label the center. Expanding the y-axis to roughly -25% of the max `real_row` value is a good start.

```
p1 <- ggplot(barcodes, aes(x=column,y=real_row,fill=values)) +
  geom_tile() +
  coord_polar() + expand_limits(y = -100) +
  scale_fill_manual(values = unique(barcodes$values), limits =
unique(barcodes$values)) +
  theme_map() + theme(legend.position = "none")
p1
```



Awesome

Now add some style to the graph, throw on some text, or maybe take it upon yourself to nest all 3 seasons inside of each other. This is a great way to break TV and movies into halves or quarters, and nest an average pallete in rings between the episodes as well. Have fun!