Fully saturated RGB rainbow colors are still widely used in scientific visualizations despite their widely-recognized disadvantages. A recent wild-caught example is presented, showing its limitations along with a better HCL-based alternative palette.

**#endrainbow**

The go-to palette in many software packages is – or used to be until rather recently – the so-called rainbow: a palette created by changing the hue in highly-saturated RGB colors. This has been widely recognized as having a number of disadvantages including: abrupt shifts in brightness, misleading for viewers with color vision deficiencies, too flashy to look at for a longer time.

Here, we present the most recent addition to that example collection, a map of influenza severity in Germany. Along with the original map and its poor choice of colors we:

* highlight its problems by desaturation to grayscale and by emulating color vision deficiencies,
* suggest a proper sequential HCL-based palette, and
* provide the R code that can extract and replace the color palette in a PNG graphics file.

**Influenza in Germany**

The original color palette (left) is the classic rainbow ranging from “normal” (blue) to “strongly increased” (red). As all colors in the palette are very flashy and highly-saturated it is hard to grasp intuitively which areas are most affected by influenza. Also, the least interesting “normal” areas stand out as blue is the darkest color in the palette.

As an alternative, a proper multi-hue sequential HCL palette is used on the right. This has smooth gradients and the overall message can be grasped quickly, giving focus to the high-risk regions depicted with dark/colorful colors. However, the extremely sharp transitions between “normal” and “strongly increased” areas (e.g., in the North and the East) might indicate some overfitting in the underlying smoothing for the map.

[influenza-rainbow](https://i0.wp.com/eeecon.uibk.ac.at/~zeileis/assets/posts/2019-03-07-endrainbow/influenza-rainbow.png?ssl=1)  
[influenza-purpleyellow](https://i1.wp.com/eeecon.uibk.ac.at/~zeileis/assets/posts/2019-03-07-endrainbow/influenza-purpleyellow.png?ssl=1)

Converting all colors to grayscale brings out even more clearly why the overall picture is so hard to grasp with the original palette: The gradients are discontinuous switching several times between bright and dark. Thus, it is hard to identify the high-risk regions while this is more natural and straightforward with the HCL-based sequential palette.

[influenza-rainbow-gray](https://i0.wp.com/eeecon.uibk.ac.at/~zeileis/assets/posts/2019-03-07-endrainbow/influenza-rainbow-gray.png?ssl=1)  
[influenza-purpleyellow-gray](https://i2.wp.com/eeecon.uibk.ac.at/~zeileis/assets/posts/2019-03-07-endrainbow/influenza-purpleyellow-gray.png?ssl=1)

Emulating green-deficient vision (deuteranopia) emphasizes the same problems as the desaturated version above but shows even more problems with the original palette: The wrong areas in the map “pop out”, making the map extremely hard to use for viewers with red-green deficiency. The HCL-based palette on the other hand is equally accessible for color-deficient viewers as for those with full color vision.

[influenza-rainbow-deutan](https://i0.wp.com/eeecon.uibk.ac.at/~zeileis/assets/posts/2019-03-07-endrainbow/influenza-rainbow-deutan.png?ssl=1)  
[influenza-purpleyellow-deutan](https://i2.wp.com/eeecon.uibk.ac.at/~zeileis/assets/posts/2019-03-07-endrainbow/influenza-purpleyellow-deutan.png?ssl=1)

**Replication in R**

The desaturated and deuteranope version of the original image (a screenshot of the RKI web page) are relatively easy to produce using the colorspace function cvd\_emulator("influenza-rainbow.png"). Internally, this reads the RGB colors for all pixels in the PNG, converts them with the colorspace functions desaturate() and deutan(), respectively, and saves the PNG again. Below we also do this “by hand”.

What is more complicated is the replacement of the original rainbow palette with a properly balanced HCL palette (without access to the underlying data). Luckily the image contains a legend from which the original palette can be extracted. Subsequently, it is possibly to index all colors in the image, replace them, and write out the PNG again.

As a first step we read the original PNG image using the R package png, returning a height x width x 4 array containing the three RGB (red/green/blue) channels plus a channel for alpha transparency. Then, this is turned into a height x width matrix containing color hex codes using the base rgb() function:

img <- png::readPNG("influenza-rainbow.png")

img <- matrix(

rgb(img[,,1], img[,,2], img[,,3]),

nrow = nrow(img), ncol = ncol(img)

)

Using a manual search we find a column of pixels from the palette legend (column 630) and thin it to obtain only 99 colors:

pal\_rain <- img[96:699, 630]

pal\_rain <- pal\_rain[seq(1, length(pal\_rain), length.out = 99)]

For replacement we use a slightly adapted sequential\_hcl().The "Purple-Yellow" palette is currently only in version 1.4-1 of the package on R-Forge but other sequential HCL palettes could also be used here.

library("colorspace")

pal\_hcl <- sequential\_hcl(99, "Purple-Yellow", p1 = 1.3, c2 = 20)

Now for replacing the RGB rainbow colors with the sequential colors, the following approach is taken: The original image is indexed by matching the color of each pixel to the closest of the 99 colors from the rainbow palette. Furthermore, to preserve the black borders and the gray shadows, 50 shades of gray are also offered for the indexing. To match pixel colors to palette colors a simple Manhattan distance (sum of absolute distances) is used in the CIELUV color space:

# 50 shades of gray

pal\_gray <- gray(0:50/50)

## HCL coordinates for image and palette

img\_luv <- coords(as(hex2RGB(as.vector(img)), "LUV"))

pal\_luv <- coords(as(hex2RGB(c(pal\_rain, pal\_gray)), "LUV"))

## Manhattan distance matrix

dm <- matrix(NA, nrow = nrow(img\_luv), ncol = nrow(pal\_luv))

for(i in 1:nrow(pal\_luv)) dm[, i] <- rowSums(abs(t(t(img\_luv) - pal\_luv[i,])))

idx <- apply(dm, 1, which.min)

Now each element of the img hex color matrix can be easily replaced by indexing a new palette with 99 colors (plus 50 shades of gray) using the idx vector. This is what the pal\_to\_png() function below does, writing the resulting matrix to a PNG file. The function is somewhat quick and dirty, makes no sanity checks, and assumes img and idx are in the calling environment.

pal\_to\_png <- function(pal = pal\_hcl, file = "influenza.png", rev = FALSE) {

ret <- img

pal <- if(rev) c(rev(pal), rev(pal\_gray)) else c(pal, pal\_gray)

ret[] <- pal[idx]

ret <- coords(hex2RGB(ret))

dim(ret) <- c(dim(img), 3)

png::writePNG(ret, target = file)

}

With this function, we can easily produce the PNG graphic with the desaturated palette and the deuteranope version”

pal\_to\_png(desaturate(pal\_rain), "influenza-rainbow-gray.png")

pal\_to\_png( deutan(pal\_rain), "influenza-rainbow-deutan.png")

The analogous graphics for the HCL-based "Purple-Yellow" palette are generated by:

pal\_to\_png( pal\_hcl, "influenza-purpleyellow.png")

pal\_to\_png(desaturate(pal\_hcl), "influenza-purpleyellow-gray.png")

pal\_to\_png( deutan(pal\_hcl), "influenza-purpleyellow-deutan.png")

**Further remarks**

Given that we have now extracted the pal\_rain palette and set up the pal\_hcl alternative we can also use the colorspace function specplot() to understand how the perceptual properties of the colors change across the two palettes. For the HCL-based palette the hue/chroma/luminance changes smoothly from dark/colorful purple to a light yellow. In contrast, in the original RGB rainbow chroma and, more importantly, luminance change non-monotonically and rather abruptly:

specplot(pal\_rain)

specplot(pal\_hcl)

[influenza-rainbow-spectrum](https://i1.wp.com/eeecon.uibk.ac.at/~zeileis/assets/posts/2019-03-07-endrainbow/influenza-rainbow-spectrum.png?ssl=1)  
[influenza-purpleyellow-spectrum](https://i0.wp.com/eeecon.uibk.ac.at/~zeileis/assets/posts/2019-03-07-endrainbow/influenza-purpleyellow-spectrum.png?ssl=1)

Given that the colors in the image are indexed now and the gray shades are in a separate subvector, we can now easily rev-erse the order in both subvectors. This yields a black background with white letters and we can use the "Inferno" palette that works well on dark backgrounds:

pal\_to\_png(sequential\_hcl(99, "Inferno"), "influenza-inferno.png", rev = TRUE)

[influenza-inferno](https://i2.wp.com/eeecon.uibk.ac.at/~zeileis/assets/posts/2019-03-07-endrainbow/influenza-inferno.png?ssl=1)