Journal of Statistical Software

November 2020, Volume 96, Issue 1. doi: 10.18637/jss.v096.i01

# colorspace: A Toolbox for Manipulating and Assessing Colors and Palettes

**Achim Zeileis**
Universität Innsbruck

**Jason C. Fisher**
U.S. Geological Survey

**Kurt Hornik**
WU Wirtschaftsuniversität Wien

**Ross Ihaka**
University of Auckland

**Claire D. McWhite**
The University of Texas at Austin

**Paul Murrell**
University of Auckland

**Reto Stauffer**
Universität Innsbruck

**Claus O. Wilke**
The University of Texas at Austin

**Abstract**

The R package **colorspace** provides a flexible toolbox for selecting individual colors or color palettes, manipulating these colors, and employing them in statistical graphics and data visualizations. In particular, the package provides a broad range of color palettes based on the HCL (hue-chroma-luminance) color space. The three HCL dimensions have been shown to match those of the human visual system very well, thus facilitating intuitive selection of color palettes through trajectories in this space. Using the HCL color model, general strategies for three types of palettes are implemented: (1) Qualitative for coding categorical information, i.e., where no particular ordering of categories is available. (2) Sequential for coding ordered/numeric information, i.e., going from high to low (or vice versa). (3) Diverging for coding ordered/numeric information around a central neutral value, i.e., where colors diverge from neutral to two extremes. To aid selection and application of these palettes, the package also contains scales for use with **ggplot2**, **shiny** and **tcltk** apps for interactive exploration, visualizations of palette properties, accompanying manipulation utilities (like desaturation and lighten/darken), and emulation of color vision deficiencies. The **shiny** apps are also hosted online at http://hclwizard.org/.

*Keywords*: color, palette, HCL, RGB, hue, color vision deficiency, R.

## 1. Introduction

Color is an integral element of many statistical graphics and data visualizations. Therefore, colors should be carefully chosen to support all viewers in accessing the information displayed

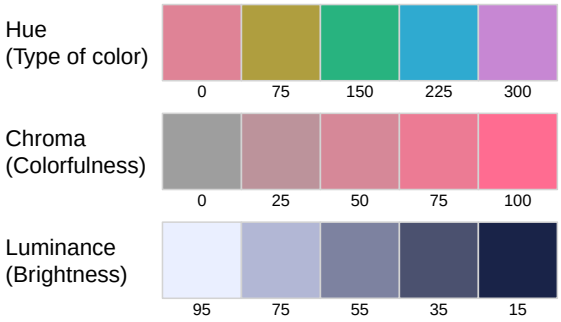


Figure 1: Axes of the HCL color space. Top: Hue $H$ changes from 0 (red) via 75 (yellow), etc. to 300 (purple) with fixed $C = 60$ and $L = 65$. Center: Chroma $C$ changes from 0 (gray) to 100 (colorful) with fixed $H = 0$ (red) and $L = 65$. Bottom: Luminance $L$ changes from 95 (light) to 15 (dark) with fixed $H = 260$ (blue) and $C = 25$ (low, close to gray).

(Tufte 1990; Brewer 1999; Ware 2004; Wilkinson 2005; Wilke 2019). However, until relatively recently many software packages have been using color palettes derived from simple RGB (red-green-blue) color combinations such as the RGB "rainbow" (or "jet") color palette with poor perceptual properties. See Hawkins, McNeall, Stephenson, Williams, and Carlson (2014) and Stauffer, Mayr, Dabernig, and Zeileis (2015) and the references therein for an overview.

To address these problems, many improved color palettes with better perceptual properties have been receiving increasing attention in the literature (Harrower and Brewer 2003; Zeileis, Hornik, and Murrell 2009; Smith and Van der Walt 2015; CARTO 2019; Crameri 2018). Many systems for statistical and scientific computing provide infrastructure for such color palettes. For example, for R (R Core Team 2020) the list of useful packages encompasses **RColorBrewer** (Neuwirth 2014), **viridis** (Garnier 2018), **rcartocolor** (Nowosad 2019), **wesanderson** (Ram and Wickham 2018), and **scico** (Pedersen and Crameri 2020) among many others. Furthermore, packages like **pals** (Wright 2019) and **paletteer** (Hvitfeldt 2020) collect many of the proposed palettes in combination with a unified interface. Most of these palettes, however, are pre-existing palettes, stored as a limited set of colors and interpolated as necessary. And even if specific algorithms have been used in the initial construction of the palettes, these are often not reflected in the software implementations.

The **colorspace** package (Ihaka *et al.* 2020) adopts a somewhat different approach that gives the user direct access to the construction principles underlying its palettes. These are based on simple trajectories in the perceptually-based HCL (hue-chroma-luminance) color space (Wikipedia 2020e) whose axes match those of the human visual system very well: Hue (type of color, dominant wavelength), chroma (colorfulness), luminance (brightness), see Figure 1. Thus, utilizing this color model the **colorspace** package can derive general and adaptable strategies for color palettes; manipulate individual colors and color palettes; and assess and visualize the properties of color palettes (beyond simple color swatches). Specifically, **colorspace** provides three types of palettes based on the HCL model:

- *Qualitative:* Designed for coding categorical information, i.e., where no particular ordering of categories is available and every color should receive the same perceptual weight. Function: `qualitative_hcl()`.
- *Sequential:* Designed for coding ordered/numeric information, i.e., where colors go from high to low (or vice versa). Function: `sequential_hcl()`.

- *Diverging:* Designed for coding ordered/numeric information around a central neutral value, i.e., where colors diverge from neutral to two extremes. Function: `diverging_hcl()`.

A broad collection of prespecified palettes is shipped in the package. In addition, existing palettes can be easily tweaked and new or adapted palettes registered. The prespecified palettes include suitable HCL color choices that closely approximate most palettes from packages **RColorBrewer**, **rcartocolor**, and **viridis** by using only a small set of hue, chroma, and luminance parameters.

To aid choice and application of these palettes the package provides (a) scales for use with **ggplot2** (Wickham 2016), (b) **shiny** (Chang, Cheng, Allaire, Xie, and McPherson 2020) and **tcltk** (R Core Team 2020) apps for interactive exploration, (c) visualizations of palette properties, and (d) accompanying manipulation utilities (like converting to grayscale by desaturation, lighten/darken, and emulation of color vision deficiencies).

The remainder of the paper is organized as follows: Section 2 gives a first overview of the package's "look & feel" and the general workflow. Section 3 summarizes the S4 color space classes and methods in the package. Section 4 introduces the extensible collection of HCL-based palettes along with their construction details. Section 5 presents the toolbox for palette visualization and assessment. Section 6 discusses the implemented techniques for color vision deficiency emulation that help assess the suitability of colors for colorblind viewers. Section 7 briefly highlights the interactive color apps from the package. Some further color manipulation utilities are highlighted in Section 8 before Section 9 concludes the paper.

## 2. A quick tour

The stable release version of **colorspace** is hosted on the Comprehensive R Archive Network (CRAN) at https://CRAN.R-project.org/package=colorspace and the development version is hosted on R-Forge at https://R-Forge.R-project.org/projects/colorspace/.

### 2.1. Choosing HCL-based color palettes

The **colorspace** package ships with a wide range of predefined color palettes, specified through suitable trajectories in the HCL (hue-chroma-luminance) color space. A quick overview can be gained easily with the `hcl_palettes()` function (see Figure 2, some of these are illustrated in more detail later):

```
R> library("colorspace")
R> hcl_palettes(plot = TRUE)
```

A suitable vector of colors can be easily computed by specifying the desired number of colors and the palette name (see Figure 2 for possible palette names), e.g.,

```
R> q4 <- qualitative_hcl(4, palette = "Dark 3")
R> q4

[1] "#E16A86" "#909800" "#00AD9A" "#9183E6"
```


Figure 2: Brief overview of available predefined palettes in **colorspace**.

The functions `sequential_hcl()`, and `diverging_hcl()` work analogously. Additionally, a palette's hue/chroma/luminance parameters can be modified, thus allowing for easy customization of each palette. Moreover, the `choose_palette()`/`hclwizard()` app provides convenient user interfaces to perform palette customization interactively. Finally, even more flexible diverging HCL palettes are provided by `divergingx_hcl()`.

## 2.2. Usage with base graphics

The color vectors returned by the HCL palette functions can usually be passed directly to most base graphics, typically through the `col` argument. Here, the `q4` vector created above is used in a time series display (see the left panel of Figure 3):

```
R> plot(log(EuStockMarkets), plot.type = "single", col = q4, lwd = 2)
R> legend("topleft", colnames(EuStockMarkets), col = q4, lwd = 3, bty = "n")
```

As another example for a sequential palette, we demonstrate how to create a spine plot (see the right panel of Figure 3) displaying the proportion of Titanic passengers that survived per class. The `"Purples 3"` palette is used, which is quite similar to the **ColorBrewer.org** (Harrower and Brewer 2003) palette `"Purples"`. Here, only two colors are employed: a dark purple that is highlighted against a light gray.

```
R> ttnc <- margin.table(Titanic, c(1, 4))
R> spineplot(ttnc, col = sequential_hcl(2, palette = "Purples 3"))
```

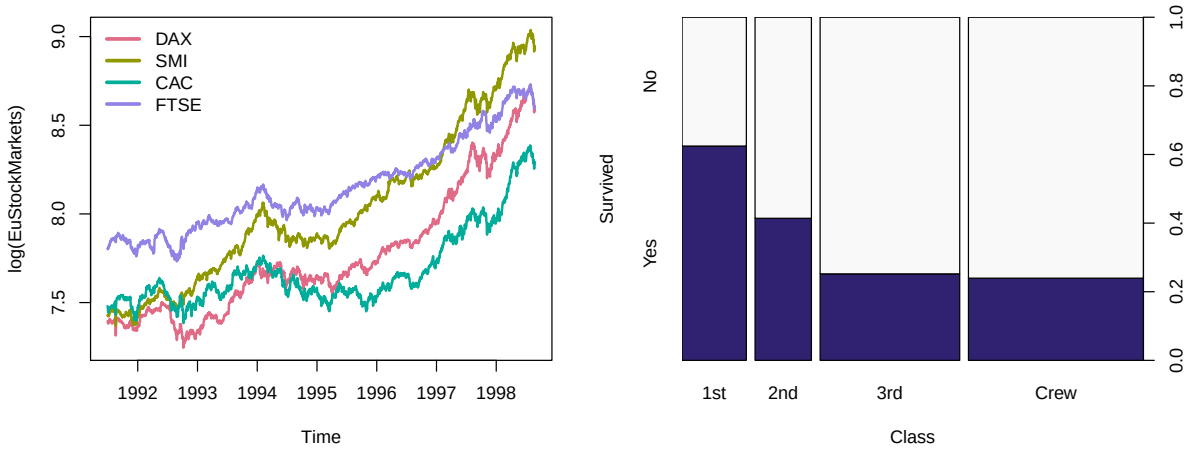


Figure 3: Using **colorspace** with base R graphics. Left: Time series plot of log-prices from `EuStockMarkets` data with `qualitative_hcl(4, "Dark 3")` palette. Right: Spine plot with survival proportions across passenger classes in the `titanic` data with `sequential_hcl(2, "Purples 3")` palette.

### 2.3. Usage with ggplot2

To provide access to the HCL color palettes from within **ggplot2** graphics (Wickham 2016; Wickham *et al.* 2020) suitable discrete, continuous, and binned **ggplot2** color scales are provided. The scales are named via the scheme

```
scale_<aesthetic>_<datatype>_<colorscale>()
```

where

- `<aesthetic>` is the name of the aesthetic (`fill`, `color`, `colour`).
- `<datatype>` is the type of the variable plotted (`discrete`, `continuous`, `binned`).
- `<colorscale>` sets the type of the color scale used (i.e., `qualitative`, `sequential`, `diverging`, `divergingx`).

To illustrate their usage two simple examples are shown using the qualitative `"Dark 3"` and sequential `"Purples 3"` palettes that were also employed above. For the first example, semi-transparent shaded densities of the sepal length from the `iris` data are shown, grouped by species (see the left panel of Figure 4).

```
R> library("ggplot2")
R> ggplot(iris, aes(x = Sepal.Length, fill = Species)) +
+    geom_density(alpha = 0.6) +
+    scale_fill_discrete_qualitative(palette = "Dark 3")
```

And for the second example the sequential palette is used to code the cut levels in a scatter of price by carat in the `diamonds` data (or rather a small subsample thereof, see the right panel of Figure 4). The scale function first generates six colors but then drops the first color because the light gray is too light here. (Alternatively, the chroma and luminance parameters could also be tweaked.)

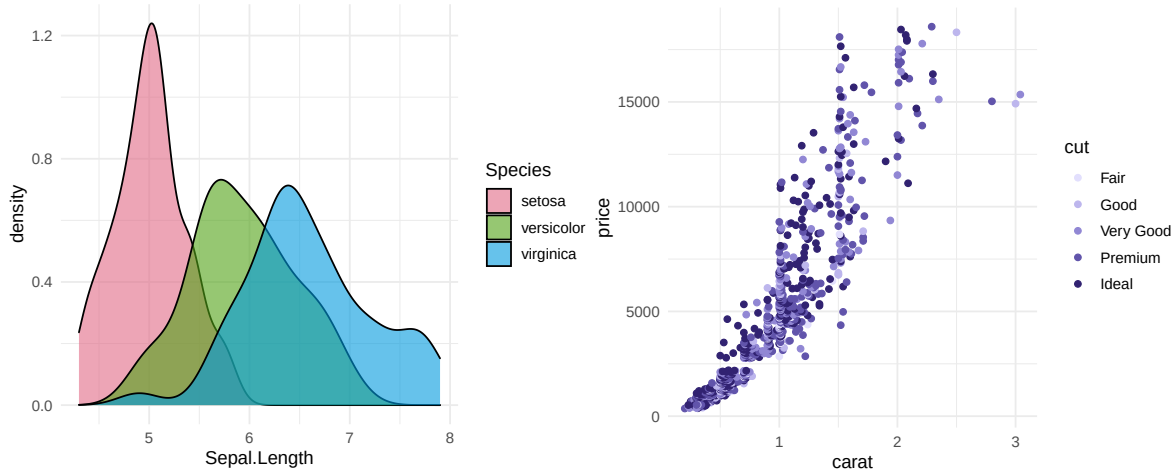


Figure 4: Using **colorspace** with **ggplot2** graphics. Left: Kernel density of sepal length, grouped and shaded by species, in the `iris` data with semi-transparent `scale_fill_discrete_qualitative(palette = "Dark 3")` color scale. Right: Scatter plot of price by carat, shaded by cut levels, in a subsample of the `diamonds` data with the `scale_color_discrete_sequential(palette = "Purples 3", nmax = 6, order = 2:6)` color scale.

```
R> dsamp <- diamonds[1 + 1:1000 * 50, ]
R> ggplot(dsamp, aes(carat, price, color = cut)) + geom_point() +
+    scale_color_discrete_sequential(palette = "Purples 3", nmax = 6,
+      order = 2:6)
```

## 2.4. Palette visualization and assessment

The **colorspace** package also provides a number of functions that aid visualization and assessment of its palettes.

- `demoplot()` can display a palette (with arbitrary number of colors) in a range of typical and somewhat simplified statistical graphics.
- `hclplot()` converts the colors of a palette to the corresponding hue/chroma/luminance coordinates and displays them in HCL space with one dimension collapsed. The collapsed dimension is the luminance for qualitative palettes and the hue for sequential/diverging palettes.
- `specplot()` also converts the colors to hue/chroma/luminance coordinates but draws the resulting spectrum in a line plot.

For the qualitative `"Dark 3"` palette from above the following plots can be obtained (see Figure 5).

```
R> demoplot(q4, "bar")
R> hclplot(q4)
R> specplot(q4, type = "o")
```

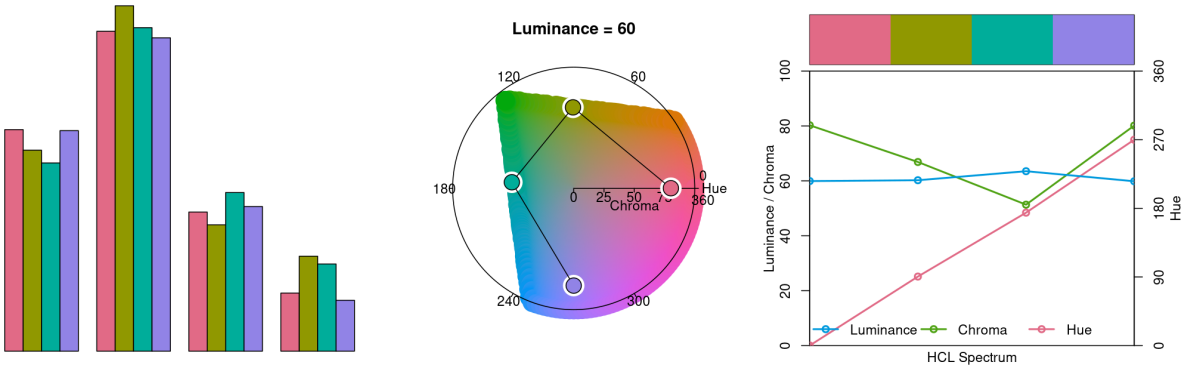


Figure 5: Palette visualization and assessment for the `qualitative_hcl(4, "Dark 3")` palette. Left: Demo bar plot. Center: Hue-chroma plane at fixed $L = 60$ in HCL space. Right: HCL spectrum with linearly changing hue (around color wheel), almost constant chroma, and constant luminance.

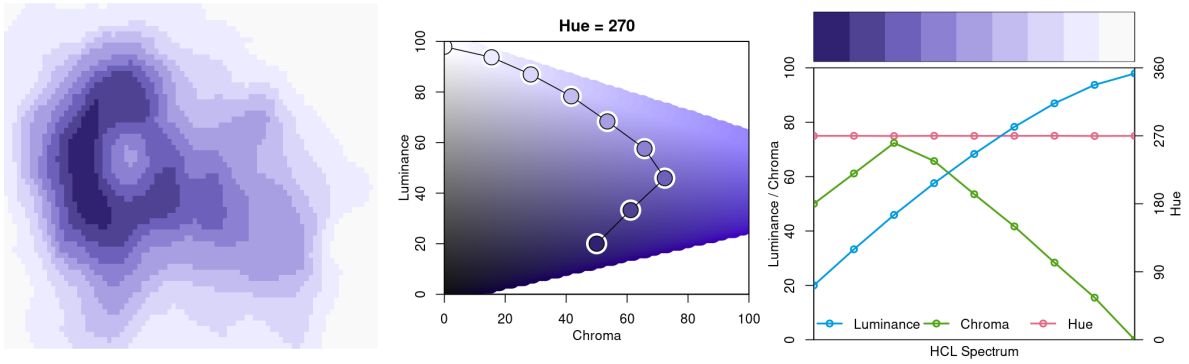


Figure 6: Palette visualization and assessment for the `sequential_hcl(9, "Purples 3")` palette. Left: Demo heatmap. Center: Chroma-luminance plane at fixed $H = 270$ in HCL space. Right: HCL spectrum with constant hue, triangular chroma, and increasing luminance.

The bar plot is used as a typical application for a qualitative palette (in addition to the time series and density plots used above). The other two displays show that luminance is (almost) constant in the palette while the hue changes linearly along the color "wheel" (from degree 0 to 270). Ideally, chroma would have also been constant to completely balance the colors. However, at this luminance the maximum chroma differs across hues so that the palette is fixed up to use less chroma for the yellow and green elements.

Note also that in a bar plot areas are shaded (and not just points or lines) so that lighter colors would be preferable. In the density plot in Figure 4 this was achieved through semi-transparency. Alternatively, luminance could be increased as is done in the `"Pastel 1"` or `"Set 3"` palettes.

Subsequently, the same types of assessment are carried out in Figure 6 for the sequential `"Purples 3"` palette as employed above.

```
R> s9 <- sequential_hcl(9, "Purples 3")
R> demoplot(s9, "heatmap")
R> hclplot(s9)
```

```
R> specplot(s9, type = "o")
```

In Figure 6, a heatmap (based on the well-known Maunga Whau volcano data) is used as a typical application for a sequential palette. The elevation of the volcano is brought out clearly, using dark colors to give emphasis to higher elevations. The other two displays show that hue is constant in the palette while luminance and chroma vary. Luminance increases monotonically from dark to light (as required for a proper sequential palette). Chroma is triangular-shaped which allows the viewer to better distinguish the middle colors in the palette when compared to a monotonic chroma trajectory.

## 3. Color spaces: S4 classes and utilities

At the core of the **colorspace** package are various utilities for computing with color spaces (Wikipedia 2020d), as the name of the package conveys. Thus, the package helps to map various three-dimensional representations of color to each other (Ihaka 2003). A particularly important mapping is the one from the perceptually-based and device-independent color model HCL (hue-chroma-luminance) to standard red-green-blue (sRGB) which is the basis for color specifications in many systems based on the corresponding hexadecimal (or simply hex) codes (Wikipedia 2020i), e.g., in HTML but also in R. For completeness further standard color models are included as well in the package. Their connections are illustrated in Figure 7. Color models that are (or try to be) perceptually-based are displayed with circles and models that are not are displayed with rectangles.

### 3.1. Implemented color spaces

The color spaces, implemented in **colorspace**, along with their corresponding S4 classes and eponymous class constructors, are:

- `RGB()` for the classic red-green-blue color model, which mixes three primary colors with different intensities to obtain a spectrum of colors. The advantage of this color model is (or was) that it corresponded to how computer and TV screens generated colors, hence it was widely adopted and still is the basis for color specifications in many systems. For example, hex color codes are employed in HTML but also in R. However, the RGB model also has some important drawbacks: It does not take into account the output device properties, it is not perceptually uniform (a unit step within RGB does not produce a constant perceptual change in color), and it is unintuitive for humans to specify colors (say brown or pink) in this space. See Wikipedia (2020g) for more details.
- `sRGB()` addresses the issue of device dependency by adopting a so-called gamma correction. Therefore, the gamma-corrected standard RGB (sRGB), as opposed to the linearized RGB above, is a good model for specifying colors in software and for hardware. But it is still unintuitive for humans to work directly with this color space. Therefore, sRGB is a good place to end up in a color space manipulation but it is not a good place to start. See Wikipedia (2020h) for more details.
- `HSV()` is a simple transformation of either the sRGB or the RGB space that tries to capture the perceptual axes: *hue* (dominant wavelength, the type of color), *saturation* (colorfulness), and *value* (brightness, i.e., light vs. dark). Unfortunately, the three axes

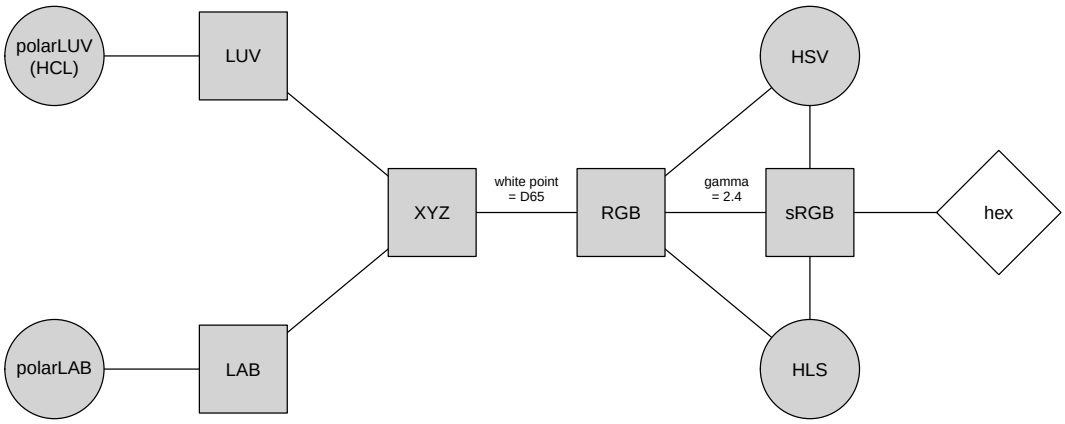


Figure 7: Relationships among three-dimensional color spaces implemented in **colorspace**. Color models that are (or try to be) perceptually-based are displayed with circles, other color models with rectangles.

in the HSV model are confounded so that, e.g., brightness changes dramatically with hue. See Wikipedia (2020f) for more details.

- `HLS()` (hue-lightness-saturation) is another transformation of either sRGB or RGB that tries to capture the perceptual axes. It does a somewhat better job but the dimensions are still strongly confounded. See Wikipedia (2020f) for more details.
- `XYZ()` was established by the CIE (Commission Internationale de l'Eclairage) based on psychophysical experiments with human subjects. It provides a unique triplet of XYZ values, coding the standard observer's perception of the color. It is device-independent but it is not perceptually uniform and the XYZ coordinates have no intuitive meaning. See Wikipedia (2020a) for more details.
- `LUV()` and `LAB()` were therefore proposed by the CIE as perceptually uniform color spaces where the former is typically preferred for emissive technologies (such as screens and monitors) whereas the latter is usually preferred when working with dyes and pigments. The L coordinate in both spaces has the same meaning and captures luminace (light-dark contrasts). Both the U and V coordinates as well as the A and B coordinates measure positions on red/green and yellow/blue axes, respectively, albeit in somewhat different ways. While this corresponds to how human color vision likely evolved (see the next section), these two color models still not correspond to perceptual axes that humans use to describe colors. See Wikipedia (2020c,b) for more details.
- `polarLUV()` and `polarLAB()` take polar coordinates in the UV plane and AB plane, respectively. Specifically, the polar coordinates of the LUV model are known as the HCL (hue-chroma-luminance) model (see Wikipedia 2020e, which points out that the LAB-based polar coordinates are also sometimes referred to as HCL). The HCL model captures the human perceptual axes very well without confounding effects as in the HSV or HLS approaches. (More details follow below.)

All S4 classes for color spaces inherit from a virtual class '`color`' which is internally always represented by matrices with three columns (corresponding to the three dimensions).

Note that since the inception of the color space conversion tools within **colorspace** (in C, Ihaka 2003) other R tools for this purpose became available, notably `grDevices::convertColor()`

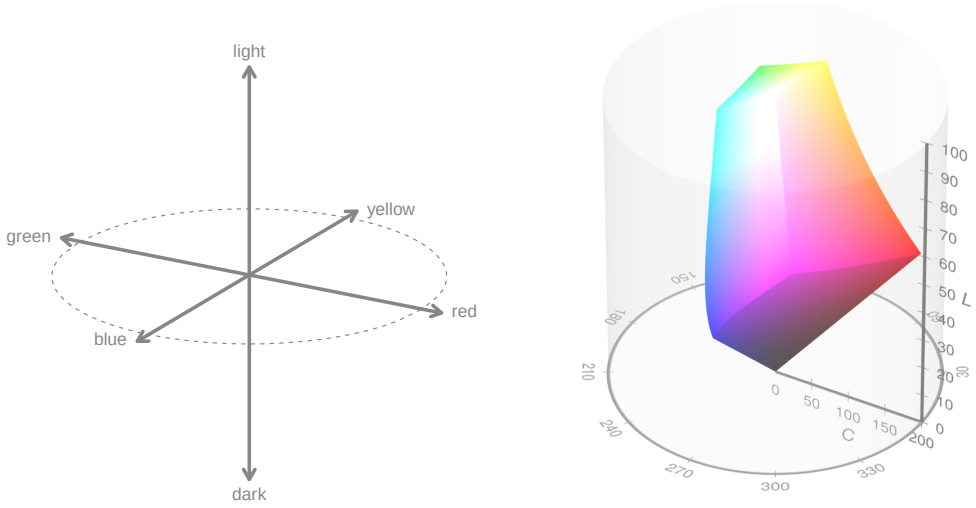


Figure 8: Visualization of axes capturing human color vision (left) and the corresponding HCL color model (right).

(in high-level R, R Core Team 2020) and `farver::convert_colour()` (in C++, Pedersen, Nicolae, and François 2020). For many basic color conversion purposes the **colorspace** package and these alternatives are essentially equally suitable (see the discussion in Zeileis, Gaslam, Murrell, and Pedersen 2018). For more complex conversions, including different chromatic adaptation algorithms, a more comprehensive color science approach is implemented in the R package **colorscience** (Gama and Davis 2018). Finally, base R also provides `grDevices::hcl()` for mapping HCL representations to hex codes.

To make the **colorspace** package self-contained and exactly backward compatible, the C code in **colorspace** is still used as the basis for all color space conversions.

## 3.2. Human color vision and the HCL color model

It has been hypothesized that human color vision has evolved in three distinct stages:

1. Perception of light/dark contrasts (monochrome only).
2. Yellow/blue contrasts (usually associated with our notion of warm/cold colors).
3. Green/red contrasts (helpful for assessing the ripeness of fruit).

See Kaiser and Boynton (1996), Knoblauch (2002), Ihaka (2003), Lumley (2006), Zeileis *et al.* (2009) for more details and references. Thus, colors can be described using a 3-dimensional space as shown in the left panel of Figure 8. However, for describing colors in such a space, it is more natural for humans to employ polar coordinates in the color plane (yellow/blue vs. green/red, visualized by the dashed circle in Figure 8) plus a third light/dark axis. Hence, color models that attempt to capture these perceptual axes are also called perceptually-based color spaces. As already argued above, the HCL model captures these dimensions very well, calling them: *hue*, *chroma*, and *luminance*. The corresponding sRGB gamut, i.e., the HCL colors that can also be represented in sRGB, is visualized in the right panel of Figure 8 (by Horvath and Lipka 2016). An animated version of the same plot is provided online by Horvath and Lipka (2017).

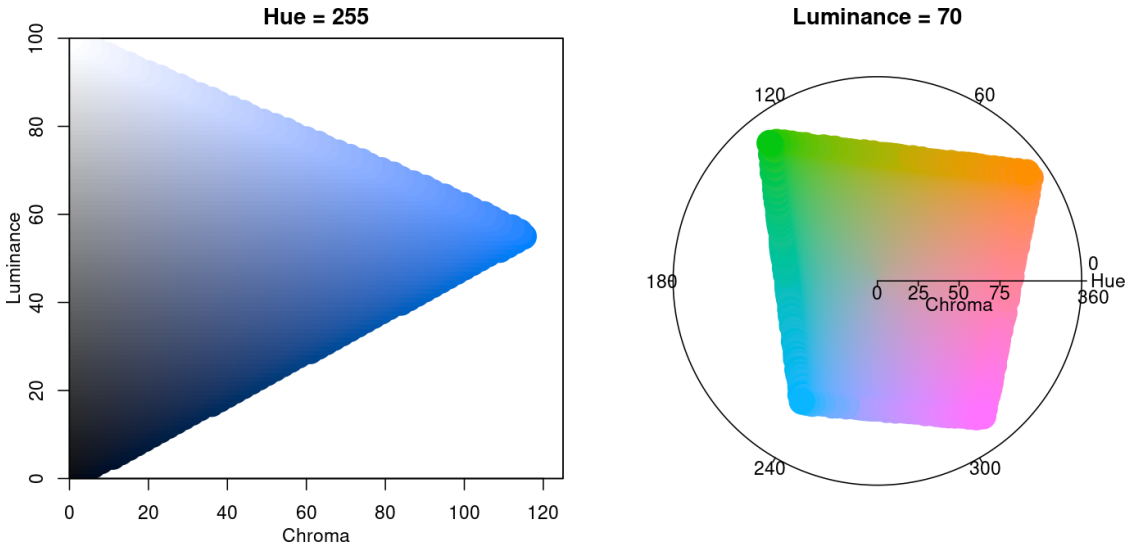


Figure 9: Vertical (left) and horizontal (right) slices of the HCL space yielding a chroma-luminance plane for given hue and a hue-chroma plane for given luminance, respectively.

The shape of the HCL space is a distorted double cone which is seen best by looking at vertical slices, i.e., chroma-luminance planes for given hues. For example, the left panel in Figure 9 depicts the chroma-luminance plane for a certain blue (hue = 255). Along with luminance the colors change from dark to light. With increasing chroma the colors become more colorful, where the highest chroma is possible for intermediate luminance.

As some colors are relatively dark (e.g., blue and red assume their maximum chroma for relatively low luminances) while others are relatively light (e.g., yellow and green), horizontal slices of hue-chroma planes for given hue have somewhat irregular shapes. The right panel in Figure 9 shows such a hue-chroma plane for moderately light colors (luminance = 70). At that luminance, green and orange can become much more colorful compared to blue or red.

### 3.3. Utilities

Several utilities are available for working with the S4 classes implementing the color spaces listed above.

- `as()` method: Convert a '`color`' object to the various color spaces, e.g., `as(x, "sRGB")`.
- `coords()`: Extract the three-dimensional coordinates pertaining to the current '`color`' class.
- `hex()`: Convert a '`color`' object to '`sRGB`' and code in a hex string that can be used within R plotting functions.
- `hex2RGB()`: Convert a given hex color string to an '`sRGB`' color object which can also be coerced to other color spaces.
- `readRGB()` and `readhex()` can read text files into '`color`' objects, either from RGB coordinates or hex color strings.
- `writehex()`: Write hex color strings to a text file.
- `whitepoint()`: Query and change the so-called white point employed in conversions from CIE XYZ to RGB. Defaults to D65 that has been specified by the CIE to approximate daylight (Poynton 2009, FAQ 15).

### 3.4. Illustration of basic colorspace functionality

As an example a vector of colors `x` can be specified in the HCL (or polar LUV) model:

```
R> (x <- polarLUV(L = 70, C = 50, H = c(0, 120, 240)))

      L  C   H
[1,] 70 50   0
[2,] 70 50 120
[3,] 70 50 240
```

The resulting three colors are pastel red (hue = 0), green (hue = 120), and blue (hue = 240) with moderate chroma and luminance. For display in other systems an sRGB representation might be needed:

```
R> (y <- as(x, "sRGB"))

             R         G         B
[1,] 0.8931564 0.5853740 0.6465459
[2,] 0.5266113 0.7224335 0.4590469
[3,] 0.4907804 0.6911937 0.8673877
```

The displayed coordinates can also be extracted as numeric matrices by `coords(x)` or `coords(y)`. We can also, for example, coerce from sRGB to HSV:

```
R> as(y, "HSV")

            H         S         V
[1,] 348.0750 0.3446008 0.8931564
[2,] 104.6087 0.3645825 0.7224335
[3,] 208.0707 0.4341857 0.8673877
```

For display in many systems (including R itself) hex color codes based on the sRGB coordinates can be created:

```
R> hex(x)

[1] "#E495A5" "#86B875" "#7DB0DD"
```

## 4. HCL-based color palettes

As motivated in the previous section, the HCL space is particularly useful for specifying individual colors and color palettes, as its three axes match those of the human visual system very well. Therefore, the **colorspace** package provides three palette functions based on the HCL model: `qualitative_hcl()`, `sequential_hcl()`, and `diverging_hcl()`. Their construction principles are exemplified in Figure 10 and explained in more detail below. The desaturated

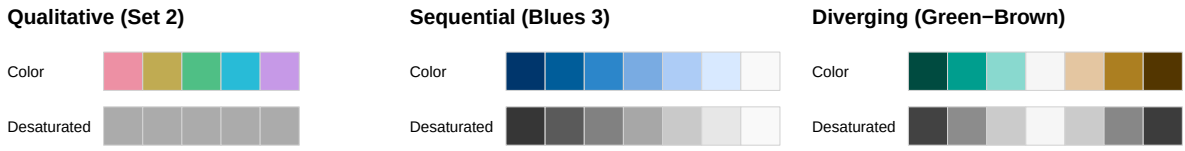


Figure 10: Examples of palette types in **colorspace**. Qualitative palettes are balanced towards the same luminance level while sequential and diverging palettes go from dark to light and/or vice versa, respectively.

palettes in the second row of Figure 10 bring out clearly that luminance differences (light-dark contrasts) are crucial for sequential and diverging palettes while qualitative palettes are balanced at the same luminance.

To facilitate obtaining good sets of colors, HCL parameter combinations that yield useful palettes are accessible by name. These can be listed using the function `hcl_palettes()`:

```
R> hcl_palettes()

HCL palettes

Type:  Qualitative
Names: Pastel 1, Dark 2, Dark 3, Set 2, Set 3, Warm, Cold, Harmonic,
       Dynamic

Type:  Sequential (single-hue)
Names: Grays, Light Grays, Blues 2, Blues 3, Purples 2, Purples 3, Reds 2,
       Reds 3, Greens 2, Greens 3, Oslo

Type:  Sequential (multi-hue)
Names: Purple-Blue, Red-Purple, Red-Blue, Purple-Orange, Purple-Yellow,
       Blue-Yellow, Green-Yellow, Red-Yellow, Heat, Heat 2,
       Terrain, Terrain 2, Viridis, Plasma, Inferno, Dark Mint,
       Mint, BluGrn, Teal, TealGrn, Emrld, BluYl, ag_GrnYl, Peach,
       PinkYl, Burg, BurgYl, RedOr, OrYel, Purp, PurpOr, Sunset,
       Magenta, SunsetDark, ag_Sunset, BrwnYl, YlOrRd, YlOrBr,
       OrRd, Oranges, YlGn, YlGnBu, Reds, RdPu, PuRd, Purples,
       PuBuGn, PuBu, Greens, BuGn, GnBu, BuPu, Blues, Lajolla,
       Turku, Hawaii, Batlow

Type:  Diverging
Names: Blue-Red, Blue-Red 2, Blue-Red 3, Red-Green, Purple-Green,
       Purple-Brown, Green-Brown, Blue-Yellow 2, Blue-Yellow 3,
       Green-Orange, Cyan-Magenta, Tropic, Broc, Cork, Vik, Berlin,
       Lisbon, Tofino
```

To inspect the HCL parameter combinations for a specific palette simply include the `palette` name where upper- vs. lower-case, spaces, etc. are ignored for matching the label, e.g., `"set2"` matches `"Set 2"`:

```
R> hcl_palettes(palette = "set2")

HCL palette
Name: Set 2
Type: Qualitative
Parameter ranges:
 h1 h2 c1 c2 l1 l2 p1 p2 cmax fixup
  0 NA 60 NA 70 NA NA NA   NA  TRUE
```

To compute the actual color hex codes (representing sRGB coordinates) based on these HCL parameters, the functions `qualitative_hcl()`, `sequential_hcl()`, and `diverging_hcl()` can be used which are described in more detail in the following sections. Either all parameters can be specified "by hand" through the HCL parameters, an entire palette can be specified "by name", or the name-based specification can be modified by a few HCL parameters. In case of the HCL parameters, either a vector-based specification such as `h = c(0, 270)` or individual parameters `h1 = 0` and `h2 = 270` can be used.

The first three of the following commands lead to equivalent output. The fourth command yields a modified set of colors (lighter due to a luminance of 80 instead of 70).

```
R> qualitative_hcl(4, h = c(0, 270), c = 60, l = 70)

[1] "#ED90A4" "#ABB150" "#00C1B2" "#ACA2EC"

R> qualitative_hcl(4, h1 = 0, h2 = 270, c1 = 60, l1 = 70)

[1] "#ED90A4" "#ABB150" "#00C1B2" "#ACA2EC"

R> qualitative_hcl(4, palette = "set2")

[1] "#ED90A4" "#ABB150" "#00C1B2" "#ACA2EC"

R> qualitative_hcl(4, palette = "set2", l = 80)

[1] "#FFACBF" "#C6CD70" "#32DDCD" "#C7BEFF"
```

### 4.1. Qualitative palettes

As suggested by Ihaka (2003), `qualitative_hcl()` distinguishes the underlying categories by a sequence of hues while keeping both chroma and luminance constant, to give each color in the resulting palette the same perceptual weight. Thus, `h` should be a pair of hues (or equivalently `h1` and `h2` can be used) with the starting and ending hue of the palette. Then, an equidistant sequence between these hues is employed, by default spanning the full color wheel (i.e., the full 360 degrees). Chroma `c` (or equivalently `c1`) and luminance `l` (or equivalently `l1`) are constants. Finally, `fixup` indicates whether colors with out-of-range coordinates should be corrected (as illustrated in Figure 5).

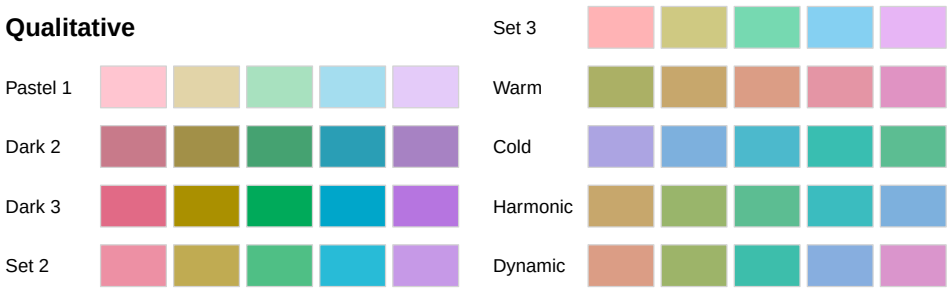


Figure 11: Prespecified qualitative HCL palettes available in `qualitative_hcl()` in **colorspace**.

Figure 11 shows the named palettes available in the `qualitative_hcl()` function. The first five palettes are close to the **ColorBrewer.org** palettes of the same name (Harrower and Brewer 2003). They employ different levels of chroma and luminance and, by default, span the full hue range. The remaining four palettes are taken from Ihaka (2003). They are based on the same chroma (50) and luminance (70) but the hue is restricted to different intervals.

```
R> hcl_palettes("qualitative", plot = TRUE, nrow = 5)
```

When palettes are employed for shading areas in statistical displays (e.g., in bar plots, pie charts, or regions in maps), lighter colors (with moderate chroma and high luminance) such as `"Pastel 1"` or `"Set 3"` are typically less distracting. By contrast, when coloring points or lines, more flashy colors (with high chroma) are often required: On a white background a moderate luminance as in `"Dark 2"` or `"Dark 3"` usually works better while on a black/dark background the luminance should be higher as in `"Set 2"`. Some examples with demo graphics are provided in Section 5.

## 4.2. Sequential palettes (single-hue)

As suggested by Zeileis *et al.* (2009), `sequential_hcl()` codes the underlying numeric values by a monotonic sequence of increasing (or decreasing) luminance. Thus, the function's `l` argument should provide a vector of length 2 with starting and ending luminance (equivalently, `l1` and `l2` can be used). Without chroma (i.e., `c = 0`), this simply corresponds to a grayscale palette like `gray.colors()`, see `"Grays"` and `"Light Grays"` in Figure 12.

For adding chroma, a simple strategy would be to pick a single hue value (via `h` or `h1`) and then decrease chroma from some value (`c` or `c1`) to zero (i.e., gray) along with increasing luminance. This is already very effective for bringing out the extremes (a dark high-chroma color vs. a light gray), see `"Blues 2"`, `"Purples 2"`, `"Reds 2"`, and `"Greens 2"`.

For distinguishing colors in the center of the palette, two strategies can be employed: (a) Hue can be varied as well by specifying an interval of hues in `h` (or beginning hue `h1` and ending hue `h2`). More details are provided in the next section. (b) Instead of a decreasing chroma, a triangular chroma trajectory can be employed from `c1` over `cmax` to `c2` (equivalently specified as a vector `c` of length 3). This yields high-chroma colors in the middle of the palette that are more easily distinguished from the dark and light extremes. See `"Blues 3"`, `"Purples 3"`, `"Reds 3"`, and `"Greens 3"` in Figure 12.

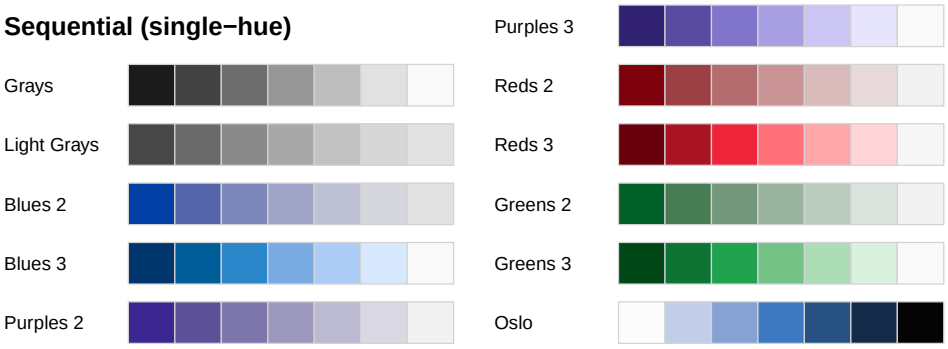


Figure 12: Prespecified sequential single-hue HCL palettes available in `sequential_hcl()` in **colorspace**.

Instead of employing linear trajectories in the chroma or luminance coordinates, some palettes employ a power transformation of the chroma and/or luminance trajectory. Either a vector `power` of length 2 or separate `p1` (for chroma) and `p2` (for luminance) can be specified. If the latter is missing, it defaults to the former.

```
R> hcl_palettes("sequential (single-hue)", n = 7, plot = TRUE, nrow = 6)
```

All except the last are inspired by the **ColorBrewer.org** palettes with the same base name (Harrower and Brewer 2003) but restricted to a single hue only. They are intended for a white/light background. The last palette (`"Oslo"`) is taken from the scientific color maps of Crameri (2018) and is intended for a black/dark background and hence the order is reversed starting from a light blue (not a light gray).

To distinguish many colors in a sequential palette it is important to have a strong contrast on the luminance axis, possibly enhanced by an accompanying pronounced variation in chroma. When only a few colors are needed (e.g., for coding an ordinal categorical variable with few levels) then a lower luminance contrast may suffice.

## 4.3. Sequential palettes (multi-hue)

To not only bring out extreme colors in a sequential palette but also better distinguish middle colors it is a common strategy to employ a sequence of hues. Thus, the basis of such a palette is still a monotonic luminance sequence as above (combined with a monotonic or triangular chroma sequence). But rather than using a single hue, an interval of hues in `h` (or beginning hue `h1` and ending hue `h2`) can be specified.

`sequential_hcl()` allows combined variations in hue (`h` and `h1`/`h2`, respectively), chroma (`c` and `c1`/`c2`/`cmax`, respectively), luminance (`l` and `l1`/`l2`, respectively), and power transformations for the chroma and luminance trajectories (`power` and `p1`/`p2`, respectively). This yields a broad variety of sequential palettes, including many that closely match other well-known color palettes. Figure 13 shows all the named multi-hue sequential palettes in **colorspace**:

```
R> hcl_palettes("sequential (multi-hue)", n = 7, plot = TRUE)
```

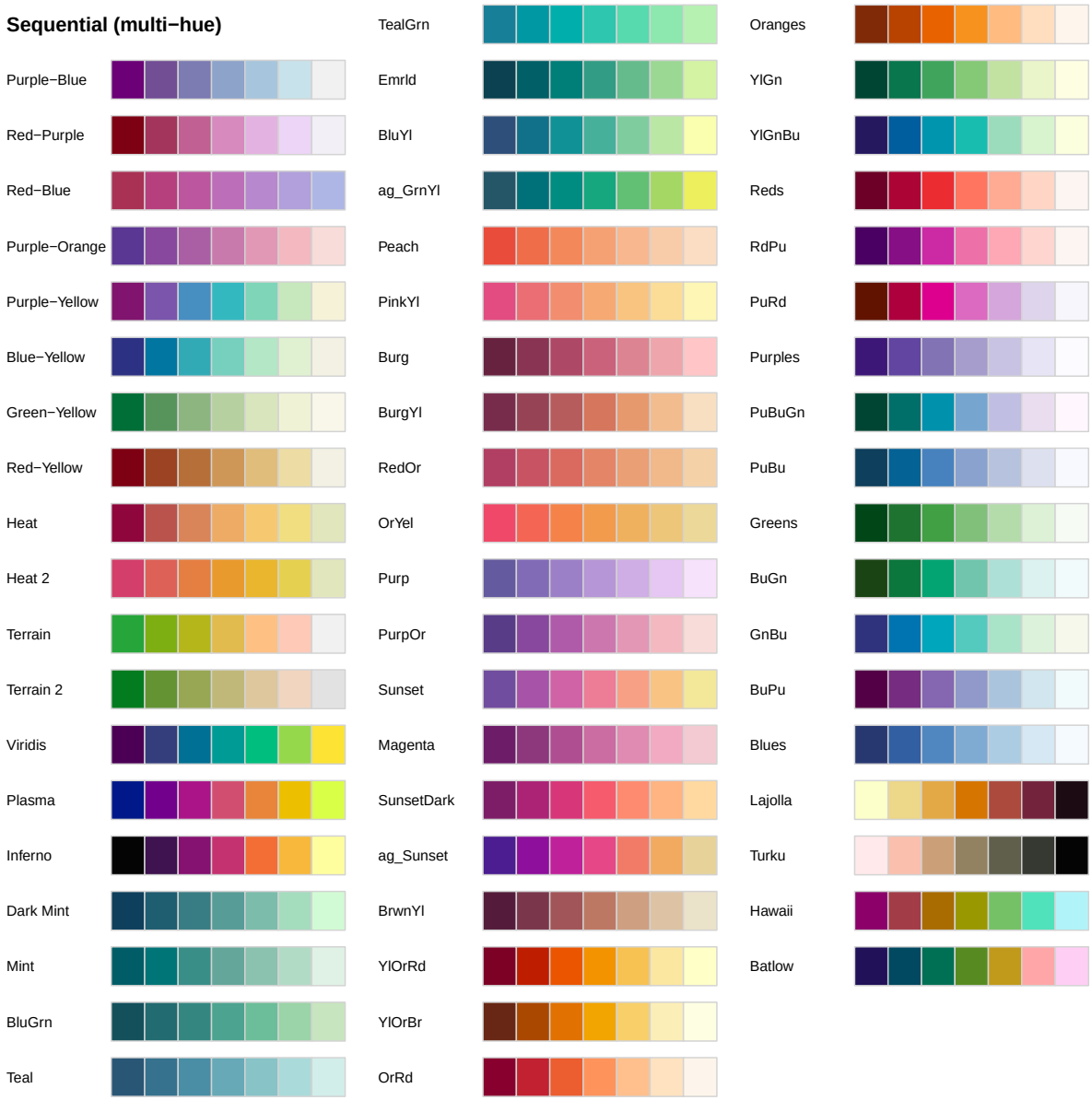


Figure 13: Prespecified sequential multi-hue HCL palettes available in `sequential_hcl()` in **colorspace**.

- `"Purple-Blue"` to `"Terrain 2"` are various palettes created during the development of **colorspace**, e.g., by Zeileis *et al.* (2009) or Stauffer *et al.* (2015) among others.
- `"Viridis"` to `"Inferno"` closely match the palettes that Smith and Van der Walt (2015) developed for **matplotlib** and that gained popularity recently.
- `"Dark Mint"` to `"BrwnYl"` closely match palettes provided in **CARTO** (CARTO 2019).
- `"YlOrRd"` to `"Blues"` closely match **ColorBrewer.org** palettes (Harrower and Brewer 2003).
- `"Lajolla"` to `"Batlow"` closely match the scientific color maps of the same name by Crameri (2018) and the first two of these are intended for a black/dark background.


Figure 14: Prespecified diverging HCL palettes available in `diverging_hcl()` in **colorspace**.

Note that the palettes differ substantially in the amount of chroma and luminance contrasts. For example, many palettes go from a dark high-chroma color to a neutral low-chroma color (e.g., `"Reds"`, `"Purples"`, `"Greens"`, `"Blues"`) or even light gray (e.g., `"Purple-Blue"`). But some palettes also employ relatively high chroma throughout the palette (e.g., the **viridis** and many **CARTO** palettes). To emphasize the extremes the former strategy is typically more suitable while the latter works better if all values along the sequence should receive some more perceptual weight.

## 4.4. Diverging palettes

`diverging_hcl()` codes the underlying numeric values by a triangular luminance sequence with different hues in the left and in the right "arms" of the palette. Thus, it can be seen as a combination of two sequential palettes with some restrictions: (a) a single hue is used for each arm of the palette, (b) chroma and luminance trajectory are balanced between the two arms, (c) the neutral central value has zero chroma. To specify such a palette a vector of two hues `h` (or equivalently `h1` and `h2`), either a single chroma value `c` (or `c1`) or a vector of two chroma values `c` (or `c1` and `cmax`), a vector of two luminances `l` (or `l1` and `l2`), and power parameter(s) `power` (or `p1` and `p2`) are used. For more flexible diverging palettes without the restrictions above (and consequently more parameters) see the `divergingx_hcl()` palettes introduced below.

Figure 14 shows all such diverging palettes that have been named in **colorspace**:

```
R> hcl_palettes("diverging", n = 7, plot = TRUE, nrow = 10)
```

- `"Blue-Red"` to `"Cyan-Magenta"` have been developed for **colorspace** starting from Zeileis *et al.* (2009), taking inspiration from various other palettes, including more balanced and simplified versions of several **ColorBrewer.org** palettes (Harrower and Brewer 2003).
- `"Tropic"` closely matches the palette of the same name from **CARTO** (CARTO 2019).

- `"Broc"` to `"Vik"` and `"Berlin"` to `"Tofino"` closely match the scientific color maps of the same name by Crameri (2018), where the first three are intended for a white/light background and the other three for a black/dark background.

When choosing a particular palette for a display similar considerations apply as for the sequential palettes. Thus, large luminance differences are important when many colors are used while smaller luminance contrasts may suffice for palettes with fewer colors etc.

### 4.5. Construction details

Table 1 summarizes which types of trajectories (*constant*, *linear*, *triangular*) are used for the three HCL coordinates (hue $H$, chroma $C$, luminance $L$) to construct the different types of palettes (*qualitative*, *sequential*, and *diverging*).

As emphasized in Figure 10, luminance is probably the most important property for defining the type of palette. It is constant for qualitative palettes, monotonic for sequential palettes (linear or a power transformation), or uses two monotonic trajectories (linear or a power transformation) diverging from the same neutral value.

Hue trajectories are also rather intuitive and straightforward for the three different types of palettes (constant vs. linear). However, chroma trajectories are probably the most complicated and least obvious from the examples above. Hence, the exact mathematical equations underlying the chroma trajectories are given in the following (i.e., using the parameters `c1`, `c2`, `cmax`, and `p1`, respectively) and are depicted in Figure 15. Analogous equations apply for the other two coordinates.

The trajectories are functions of the *intensity* $i \in [0, 1]$ where 1 corresponds to the full intensity:

$$\text{Constant: } c_1 \tag{1}$$

$$\text{Linear: } c_2 - (c_2 - c_1) \cdot i \tag{2}$$

$$\text{Triangular: } \begin{cases} c_2 - (c_2 - c_{\max}) \cdot \frac{i}{j} & \text{if } i \leq j \\ c_{\max} - (c_{\max} - c_1) \cdot \frac{i-j}{1-j} & \text{if } i > j \end{cases} \tag{3}$$

where $j$ is the intensity at which $c_{\max}$ is assumed. It is constructed such that the slope to the left is the negative of the slope to the right of $j$:

$$j = \left(1 + \frac{|c_{\max} - c_1|}{|c_{\max} - c_2|}\right)^{-1}$$

Instead of using a linear intensity $i$ going from 1 to 0, one can replace $i$ with $i^{p_1}$ in Equations 1–3. This then leads to power-transformed curves that add or remove chroma more slowly or more quickly depending on whether the power parameter $p_1$ is $< 1$ or $> 1$.

The three types of trajectories are also depicted in Figure 15. Note that full intensity $i = 1$ is on the left and zero intensity $i = 0$ is on the right of each panel. The concrete parameters are:

- Constant: `c1 = 80`.
- Linear: `c1 = 80`, `c2 = 10`, `p1 = 1` (black) vs. `p1 = 1.6` (gray).
- Triangular: `c1 = 60`, `cmax = 80`, `c2 = 10`, `p1 = 1` (black) vs. `p1 = 1.6` (gray).

| Type | $H$ | $C$ | $L$ |
|---|---|---|---|
| Qualitative | Linear | Constant | Constant |
| Sequential | Constant (single-hue) *or* Linear (multi-hue) | Linear (+ power) *or* Triangular (+ power) | Linear (+ power) |
| Diverging | Constant (2×) | Linear (+ power) *or* Triangular (+ power) | Linear (+ power) |

Table 1: Types of trajectories used for the HCL coordinates to construct qualitative, sequential, and diverging palettes, see Equations 1–3.

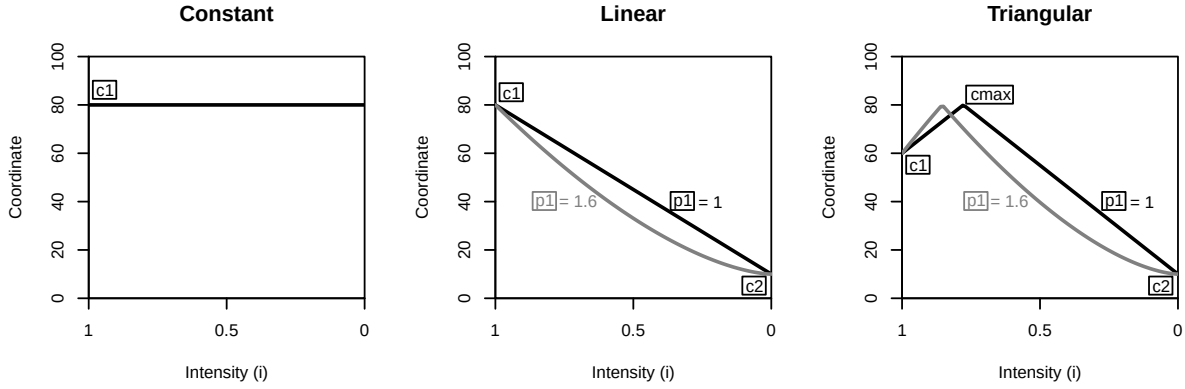


Figure 15: Types of trajectories to construct HCL color palettes, exemplified for the chroma coordinates, see Equations 1–3.

Further discussion of these trajectories and how they can be visualized and assessed for a given color palette is provided in Section 5.

## 4.6. Registering your own palettes

The `hcl_palettes()` already come with a wide range of predefined palettes to which customizations can be easily added. However, it might also be convenient to register a custom palette so that it can subsequently be reused with a new dedicated name. This is supported by adding a `register` argument once to a call to `qualitative_hcl()`, `sequential_hcl()`, or `diverging_hcl()`:

```
R> qualitative_hcl(3, palette = "set2", l = 80, register = "myset")
```

The new palette is then included in `hcl_palettes()`:

```
R> hcl_palettes("Qualitative")

HCL palettes

Type:  Qualitative
Names: Pastel 1, Dark 2, Dark 3, Set 2, Set 3, Warm, Cold, Harmonic,
       Dynamic, myset
```

The palette can be used subsequently in `qualitative_hcl()` as well as the qualitative **ggplot2** color scales (see Section 2.3), e.g.,

```
R> qualitative_hcl(4, palette = "myset")

[1] "#FFACBF" "#C6CD70" "#32DDCD" "#C7BEFF"
```

Remarks:

- The number of colors in the palette that was used during registration is not actually stored and can be modified subsequently. The same holds for arguments `alpha` and `rev`.
- When registering a new palette with a previously-used name, the old palette gets overwritten. We recommend to not overwrite the palettes that are predefined in the package (albeit technically possible).
- The registration of a palette is only stored for the current session. When R is restarted and/or the **colorspace** package reloaded, only the predefined palettes from the package are available. Thus, to make a palette permanently available a registration R code like `colorspace::qualitative_hcl(3, palette = "set2", l = 80, register = "myset")` can be placed in your `.Rprofile` or similar startup scripts.

### 4.7. Flexible diverging palettes

The `divergingx_hcl()` function provides more flexible diverging palettes by simply calling `sequential_hcl()` twice with prespecified sets of hue, chroma, and luminance parameters. Thus, it does not pose any restrictions that the two "arms" of the palette need to be balanced and also may go through a non-gray neutral color (typically light yellow). Consequently, the chroma/luminance paths can be rather unbalanced.

Figure 16 shows all such flexible diverging palettes that have been named in **colorspace**:

```
R> divergingx_palettes(n = 7, plot = TRUE, nrow = 10)
```

- `"ArmyRose"` to `"Tropic"` closely match the palettes of the same name from **CARTO** (CARTO 2019).
- `"PuOr"` to `"Spectral"` closely match the palettes of the same name from **ColorBrewer.org** (Harrower and Brewer 2003).
- `"Zissou 1"` closely matches the palette of the same name from **wesanderson** (Ram and Wickham 2018).
- `"Cividis"` closely matches the palette of the same name from the **viridis** family (Garnier 2018). Note that despite having two "arms" with blue vs. yellow colors and a low-chroma center color, this is probably better classified as a sequential palette due to the monotonic chroma going from dark to light. (See Section 4.8 for more details.)
- `"Roma"` closely matches the palette of the same name by Crameri (2018).

Typically, the more restricted `diverging_hcl()` palettes should be preferred because they are more balanced. However, by being able to go through light yellow as the neutral color warmer diverging palettes are available.

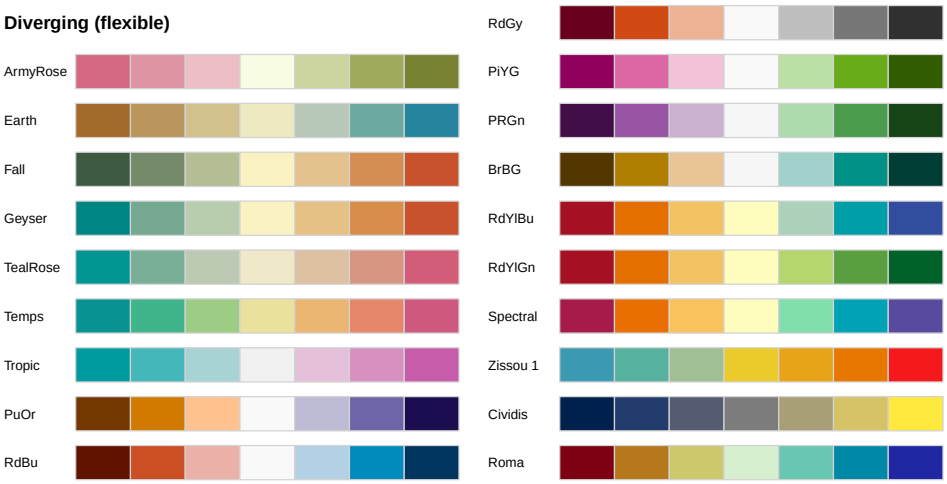


Figure 16: Prespecified flexible diverging HCL palettes available in `divergingx_hcl()` in **colorspace**.

## 4.8. Approximating palettes from other packages

The flexible specification of HCL-based color palettes in **colorspace** allows one to closely approximate color palettes from various other packages:

- **ColorBrewer.org** (Harrower and Brewer 2003) as provided by the R package **RColorBrewer** (Neuwirth 2014). See `demo("brewer", package = "colorspace")`.
- **CARTO** colors (CARTO 2019) as provided by the R package **rcartocolor** (Nowosad 2019). See `demo("carto", package = "colorspace")`.
- The viridis palettes of Smith and Van der Walt (2015) developed for **matplotlib**, as provided by the R package **viridis** (Garnier 2018). See `demo("viridis", package = "colorspace")`.
- The scientific color maps of Crameri (2018) as provided by the R package **scico** (Pedersen and Crameri 2020). See `demo("scico", package = "colorspace")`.

The graphics resulting from the demos can also be viewed online at `http://colorspace.R-Forge.R-project.org/articles/approximations.html`.

Figure 17 shows a selection of such approximations using `specplot()` (see also Section 5.2) for two blue/green/yellow palettes (namely `RColorBrewer::brewer.pal(7, "YlGnBu")` and `viridis::viridis(7)`) and two purple/red/yellow palettes (namely `rcartocolor::carto_pal(7, "ag_Sunset")` and `viridis::plasma(7)`). Each panel compares the hue, chroma, and luminance trajectories of the original palettes (top swatches, solid lines) and their HCL-based approximations (bottom swatches, dashed lines). The palettes are not identical but very close for most colors. Note also that the chroma trajectories from the HCL palettes (green dashed lines) have some kinks which are due to fixing HCL coordinates at the boundaries of admissible RGB colors.

Furthermore, Figure 17 illustrates what sets the viridis palettes apart from other sequential palettes. While the hue and luminance trajectories of `"Viridis"` and `"YlGnBu"` are very similar, the chroma trajectories differ: While lighter colors (with high luminance) have low chroma for `"YlGnBu"`, they have increasing chroma for `"Viridis"`. Similarly, `"ag_Sunset"`

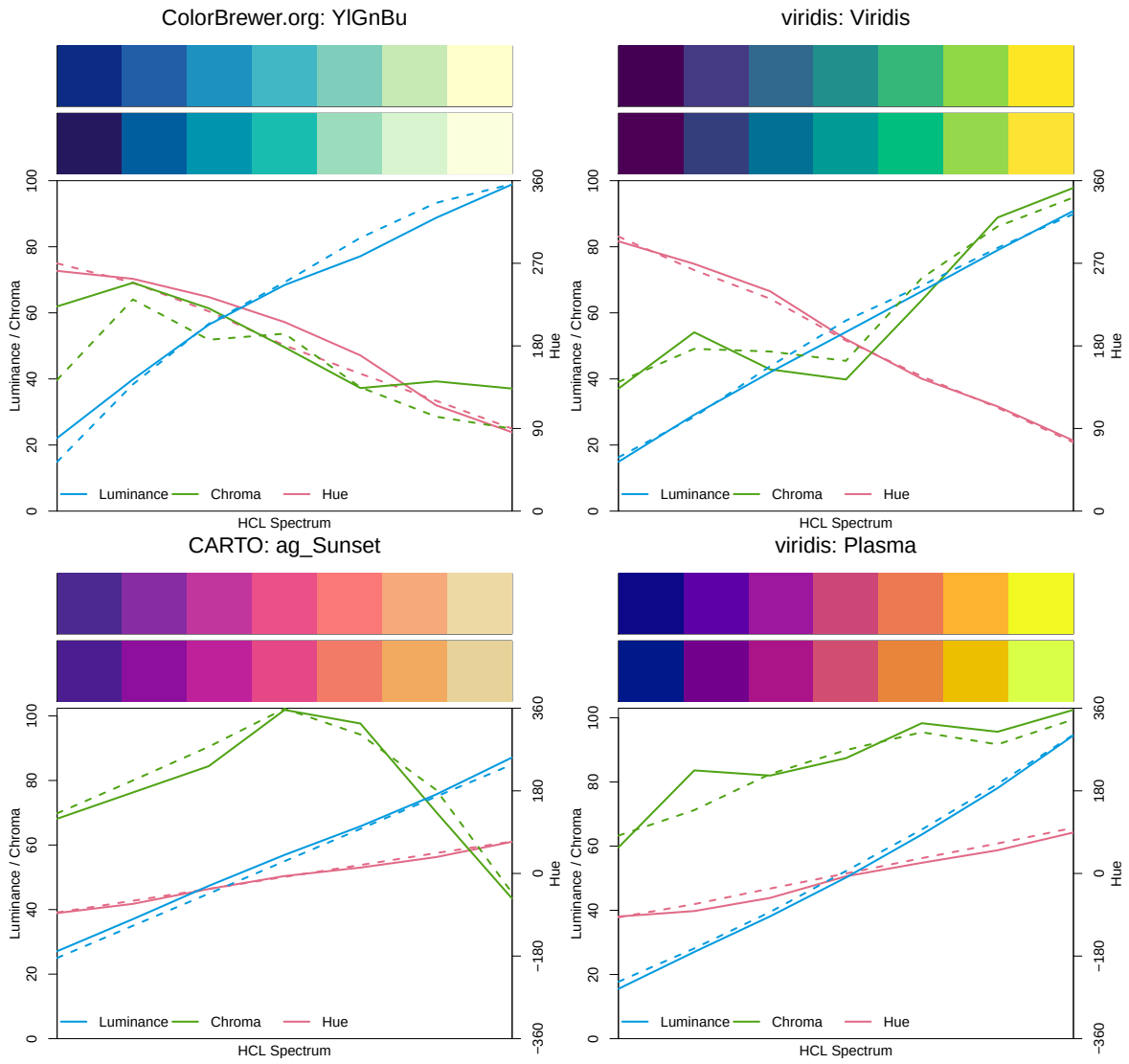


Figure 17: HCL spectrum of four palettes taken from **ColorBrewer.org**, **CARTO**, and **viridis** (top swatches, solid lines) along with their HCL-based approximations (bottom swatches, dashed lines).

and `"Plasma"` have similar hue and luminance trajectories but different chroma trajectories. The result is that the viridis palettes have rather high chroma throughout which does not work as well for sequential palettes on a white/light background as all shaded areas convey high "intensity". However, they work better on a dark/black background (see Figure 28 on page 33). Also, they might be a reasonable alternative for qualitative palettes when grayscale printing should also work.

Another somewhat nonstandard palette from the viridis family is the cividis palette based on blue and yellow hues and hence safe for red-green deficient viewers. Figure 18 shows the corresponding `specplot()` along with an HCL-based approximation. This palette is unusual: The hue and chroma trajectories would suggest a diverging palette, as there are two "arms"

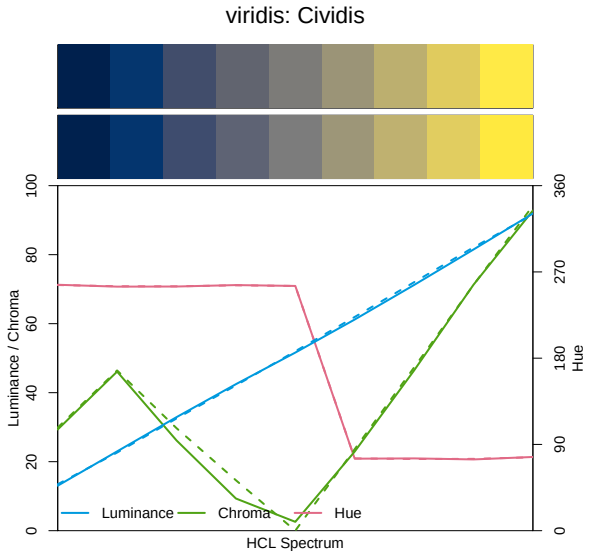


Figure 18: HCL spectrum of `viridis::cividis` (top swatch, solid lines) along with an HCL-based approximation (bottom swatch, dashed lines).

with different hues and a zero-chroma point in the center. However, the luminance trajectory clearly indicates a sequential palette as colors go monotonically from dark to light. Due to this unusual mixture the palette cannot be composed using the trajectories from Table 1.

However, the tools in **colorspace** can still be employed to easily reconstruct the palette. One strategy would be to set up the trajectories manually, using a linear luminance, piecewise linear chroma, and piecewise constant hue:

```
R> cividis_hcl <- function(n) {
+    i <- seq(1, 0, length.out = n)
+    hex(polarLUV(
+      L = 92 - (92 - 13) * i,
+      C = approx(c(1, 0.9, 0.5, 0), c(30, 50, 0, 95), xout = i)$y,
+      H = c(255, 75)[1 + (i < 0.5)]
+    ), fix = TRUE)
+  }
```

Instead of constructing the hex code from the HCL coordinates via `hex(polarLUV(L, C, H))` from **colorspace**, the base R function `hcl(H, C, L)` from **grDevices** could also be used.

In addition to manually setting up a dedicated function `cividis_hcl()`, it is possible to approximate the palette using `divergingx_hcl()` (see Section 4.7), e.g.,

```
R> divergingx_hcl(n,
+    h1 = 255, h2 = NA, h3 = 75,
+    c1 = 30, cmax1 = 47, c2 = 0, c3 = 95,
+    l1 = 13, l2 = 52, l3 = 92,
+    p1 = 1.1, p3 = 1.0
+  )
```

This uses a slight power transformation with `p1 = 1.1` in the blue arm of the palette but otherwise essentially corresponds to what `cividis_hcl()` does. For convenience the above parameters are already preregistered in `divergingx_hcl(n, palette = "Cividis")`.

### 4.9. HCL (and HSV) color palettes corresponding to base R palettes

To facilitate switching from base R palette functions to the HCL-based palettes above, **colorspace** provides a few convenience interfaces:

- `rainbow_hcl()`: Convenience interface to `qualitative_hcl()` for a HCL-based "rainbow" palette to replace the (in)famous `rainbow()` palette.
- `heat_hcl()`: Convenience interface to `sequential_hcl()` with default parameters chosen to generate more balanced heat colors than the basic `heat.colors()` function.
- `terrain_hcl()`: Convenience interface to `sequential_hcl()` with default parameters chosen to generate more balanced terrain colors than the basic `terrain.colors()` function.
- `diverging_hsv()`: Diverging palettes generated in HSV space rather than HCL space as in `diverging_hcl()`. This is provided for didactic purposes to contrast the more balanced HCL palettes with the more flashy and unbalanced HSV palettes.

Meanwhile, base R has also adopted the HCL-based palettes from **colorspace** into the function `hcl.colors()` in **grDevices** (Zeileis and Murrell 2019). This provides all the named palettes introduced in **colorspace** (with the same names, and defaulting to `"Viridis"`) but without the flexibility to modify or adapt existing palettes.

Moreover, the **grDevices** package in base R gained a new function `palette.colors()` (Zeileis, Murrell, Maechler, and Sarkar 2019) that provides various well-established qualitative color palettes that can not be approximated well by `qualitative_hcl()` due to pronounced variations in luminance and chroma. While a qualitative palette with fixed luminance and chroma is more balanced, a certain amount of variations in these properties might be necessary to make more colors distinguishable, especially for viewers with color vision deficiencies.

## 5. Palette visualization and assessment

The **colorspace** package provides several visualization functions for depicting one or more color palettes and their underlying properties. Color palettes can be visualized by:

- `swatchplot()`: Color swatches.
- `specplot()`: Spectrum of HCL and/or RGB trajectories.
- `hclplot()`: Trajectories in 2-dimensional HCL space projections.
- `demoplot()`: Illustrations of typical (and simplified) statistical graphics.

### 5.1. Color swatches

The function `swatchplot()` is a convenience function for displaying collections of palettes that can be specified as lists or matrices of hex color codes. Essentially, it is just a call to the base graphics `rect()` function but with heuristics for choosing default labels, margins, spacings, borders, etc. These heuristics are selected to work well for `hcl_palettes()`

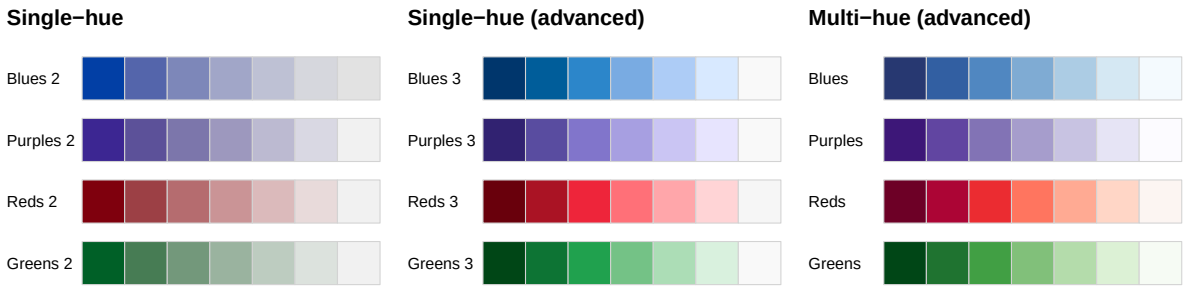


Figure 19: Variations of blue, purple, red, and green palettes with single hue and monotonic chroma (left), single hue and triangular chroma (center), and multiple hues and triangular chroma (right).

and might need further tweaking in future versions of the package. Thus, Figures 1–2 as well as Figures 10–14 all use `swatchplot()` internally. For a simple stand-alone illustration consider: `swatchplot("Palette" = sequential_hcl(5))`. Optionally, swatches emulating color vision deficiencies (see Section 6) can be added by setting `cvd = TRUE`.

Next, we demonstrate a more complex example of a `swatchplot()` with three matrices of sequential color palettes of blues, purples, reds, and greens (see Figure 19).

```
R> bprg <- c("Blues", "Purples", "Reds", "Greens")
R> swatchplot(
+   "Single-hue"            = t(sapply(paste(bprg, 2), sequential_hcl, n = 7)),
+   "Single-hue (advanced)" = t(sapply(paste(bprg, 3), sequential_hcl, n = 7)),
+   "Multi-hue (advanced)"  = t(sapply(bprg,           sequential_hcl, n = 7)),
+   nrow = 5, line = 5)
```

For all palettes, luminance increases monotonically to yield a proper sequential palette. However, the hue and chroma handling is somewhat different to emphasize different parts of the palette.

- *Single-hue:* In each palette the hue is fixed and chroma decreases monotonically (along with increasing luminance). This is typically sufficient to clearly bring out the extreme colors (dark/colorful vs. light gray).
- *Single-hue (advanced):* The hue is fixed (as above) but the chroma trajectory is triangular. Compared to the basic single-hue palette above, this better distinguishes the colors in the middle and not only the extremes.
- *Multi-hue (advanced):* As in the advanced single-hue palette, the chroma trajectory is triangular but additionally the hue varies slightly. This can further enhance the distinction of colors in the middle of the palette.

## 5.2. HCL (and RGB) spectrum

As the properties of a palette in terms of the perceptual dimensions *hue*, *chroma*, and *luminance* are not always clear from looking just at color swatches or (statistical) graphics based on these palettes, the `specplot()` function provides an explicit display for the coordinates of the HCL trajectory associated with a palette. This can bring out clearly various aspects,

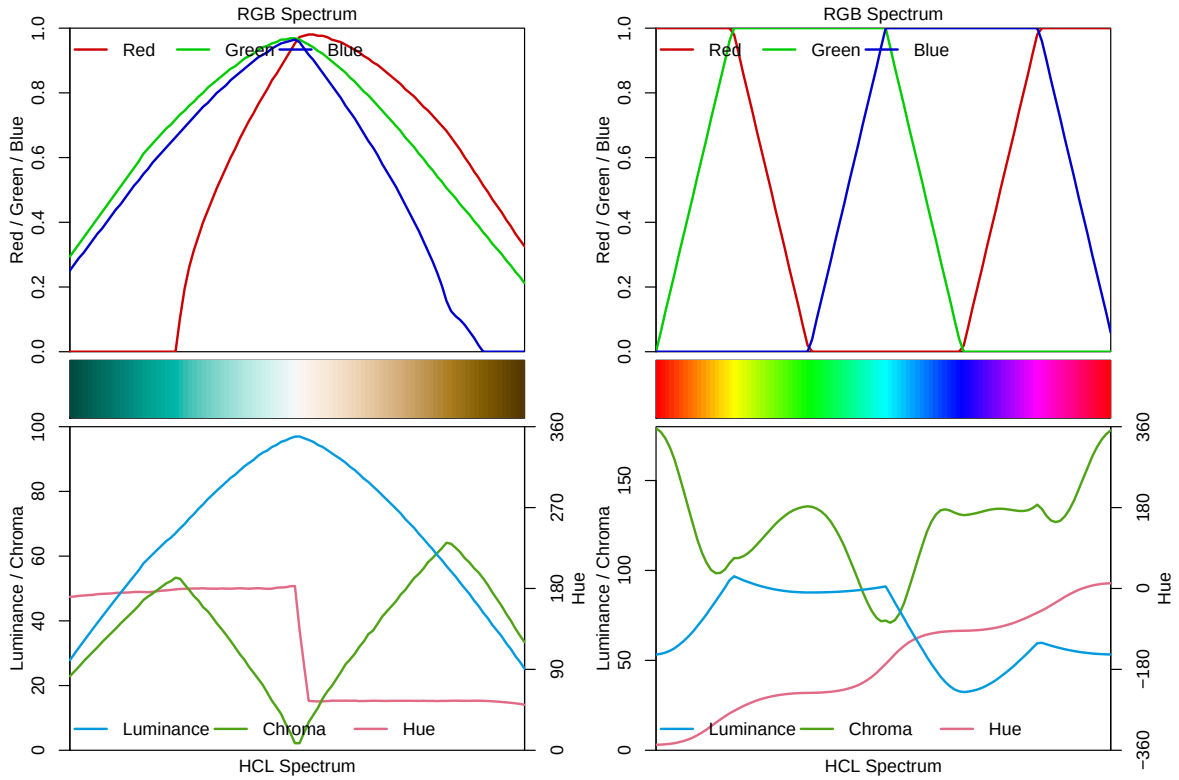


Figure 20: HCL spectrum of the balanced diverging `"Green-Brown"` palette (left panel) and the (in)famous and rather unbalanced `rainbow()` palette (right panel).

e.g., whether hue is constant, whether chroma is monotonic or triangular, and whether luminance is approximately constant (as in many qualitative palettes), monotonic (as in sequential palettes), or diverging.

The function first transforms a given color palette to its HCL (`polarLUV()`) coordinates. As the hues for low-chroma colors are not (or only poorly) identified, they are smoothed by default. Also, to avoid jumps from 0 to 360 or vice versa, the hue coordinates are shifted suitably. By default, the resulting trajectories in the HCL spectrum are visualized by a simple line plot where the $x$-axis gives the ordering of the colors in the palette. The $y$-axis depicts the following information:

- Hue is drawn in red and coordinates are indicated on the axis on the right with range $[0, 360]$ or (if necessary) $[-360, 360]$.
- Chroma is drawn in green with coordinates on the left axis. The range [0, 100] is used unless the palette necessitates higher chroma values.
- Luminance is drawn in blue with coordinates on the left axis in the range [0, 100].

Additionally, a color swatch for the palette is included. Optionally, a second spectrum for the corresponding trajectories of RGB coordinates can be included. However, this is usually just of interest for palettes created in RGB space (or simple transformations of RGB).

As spectrum plots have already been used for illustration in Figures 5 (for a qualitative palette) as well as Figures 6 and 17 (for sequential palettes), this section only provides a

couple of additional illustrations. The diverging `"Green-Brown"` palette is depicted in the left panel of Figure 20. It simply combines a green and a brown/yellow sequential single-hue palette, both with triangular chroma trajectory. Hue is constant in each "arm" of the palette and the chroma/luminance trajectories are rather balanced between both arms. In the center the palette passes through a light gray (with zero chroma) as the neutral value. By including the corresponding RGB spectrum in the top panel, it also becomes apparent that choosing such well-balanced palettes through trajectories in RGB color space is not straightforward. This balanced palette – based on relatively simple HCL trajectories – is contrasted with a poorly-balanced palette – based on simple linear RGB trajectories in the right panel of Figure 20. This depicts the RGB and HCL spectrum of the (in)famous RGB rainbow palette. (See Hawkins *et al.* 2014, for a plea why the RGB rainbow palette should be avoided in almost all scientific graphics.)

```
R> specplot(diverging_hcl(100, "Green-Brown"), rgb = TRUE)
R> specplot(rainbow(100), rgb = TRUE)
```

The RGB spectrum of the rainbow palette shows that the trajectories are quite simple in RGB space but lead to substantial variations in chroma and (more importantly) luminance. This is why this palette is not suitable for encoding underlying data in statistical graphics. See also the related discussion of color vision deficiency in Section 6.

### 5.3. Trajectories in HCL space

While the `specplot()` function above works well for bringing out the HCL coordinates associated with a given palette, it does not show how the palette fits into the HCL space. For example, it is not so clear whether high chroma values are close to the maximum possible for a given hue. Thus, it cannot be easily judged how the parameters of the hue, chroma, and luminance trajectories can be modified to obtain another palette.

Therefore, the `hclplot()` is another visualization of the HCL coordinates associated with a palette. It does so by collapsing over one of the coordinates (either the hue $H$ or the luminance $L$) and displaying a heatmap of colors combining the remaining two dimensions. The coordinates for the given color palette are highlighted to bring out its trajectory. In case the hue is really fixed (as in single-hue sequential palettes) or the luminance is really fixed (as in the qualitative palettes), collapsing is straightforward. However, when the coordinate that is collapsed over is not actually constant in the palette, a simple bivariate linear model is used to capture how the collapsed coordinate varies along with the two displayed coordinates.

The function `hclplot()` has been designed to work well with the `hcl_palettes()` in this package. While it is possible to apply it to other color palettes as well, the results might look weird or confusing if these palettes are constructed very differently (e.g., like the highly saturated base R palettes). To infer the default `type` of projection, `hclplot()` assesses the luminance trajectory and sets the default correspondingly:

- `type = "qualitative"` if luminance is approximately constant.
- `type = "sequential"` if luminance is monotonic.
- `type = "diverging"` if luminance is diverging with two monotonic "arms" in the trajectory.

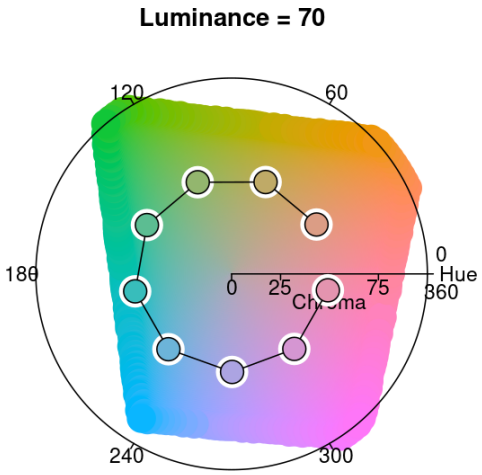


Figure 21: Hue-chroma plane with luminance fixed at $L = 70$ along with the qualitative `"Dynamic"` palette with varying hue $H$ and chroma fixed at $C = 50$.

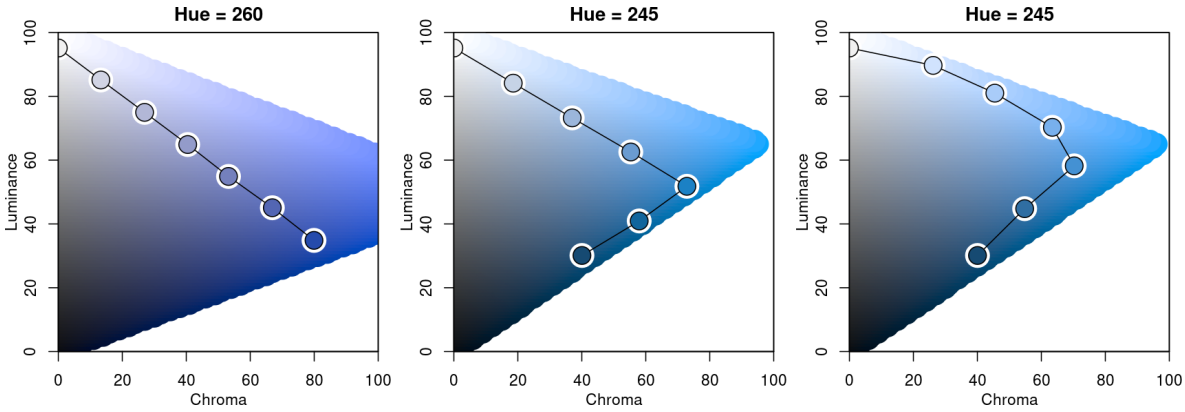


Figure 22: Luminance-chroma planes with variations of blue sequential single-hue palettes (similar to `"Blues 2"` and `"Blues 3"`). Left: Linear chroma for $H = 260$. Center: Triangular chroma for $H = 245$. Right: Power-transformed triangular chroma for $H = 245$.

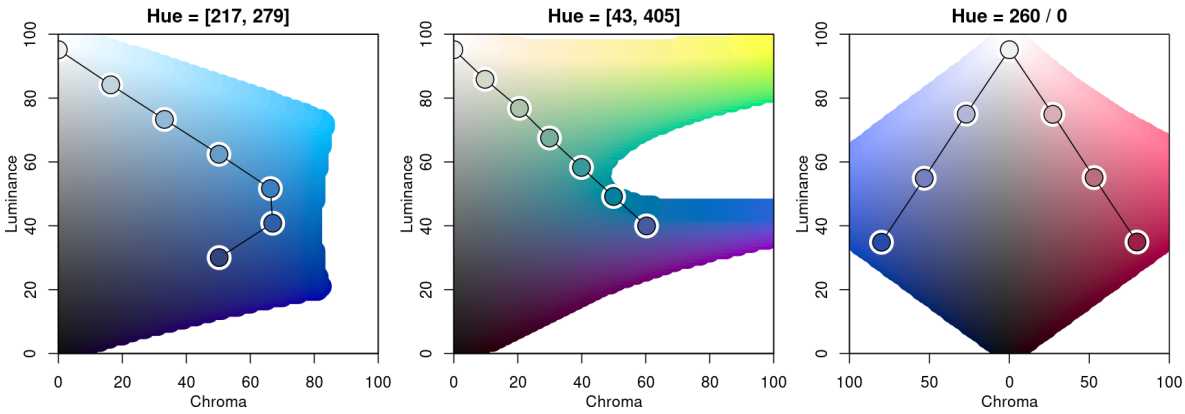


Figure 23: Luminance-chroma planes with blue multi-hue palette and triangular chroma (left), blue-yellow multi-hue palette and linear chroma (center), and diverging blue-red palette with balanced linear chroma.

Thus, for qualitative palettes – where luminance and chroma are fixed – the varying hue is displayed in a projection onto the hue-chroma plane at a given fixed luminance (Figure 21):

```
R> hclplot(qualitative_hcl(9, "Dynamic"))
```

Figure 22 compares three single-hue sequential palettes by projection to the luminance-chroma plane for the given fixed hue. In the left panel the hue 260 is used with a simple linear chroma trajectory. The other two panels employ a triangular chroma trajectory for hue 245, either with a piecewise-linear (center) or power-transformed (right) trajectory.

```
R> par(mfrow = c(1, 3))
R> hclplot(sequential_hcl(7, h = 260, c = 80, l = c(35, 95), power = 1))
R> hclplot(sequential_hcl(7, h = 245, c = c(40, 75, 0), l = c(30, 95),
+    power = 1))
R> hclplot(sequential_hcl(7, h = 245, c = c(40, 75, 0), l = c(30, 95),
+    power = c(0.8, 1.4)))
```

Note that for $H = 260$ it is possible to go to dark colors (low luminance) with high chroma while this is not possible to the same extent for $H = 245$ due to the distorted shape of the HCL space. Hence, chroma has to be decreased when proceeding to the dark low-luminance colors.

Finally, Figure 23 compares two multi-hue sequential palettes along with a diverging palette.

```
R> par(mfrow = c(1, 3))
R> hclplot(sequential_hcl(7, h = c(260, 220), c = c(50, 75, 0),
+    l = c(30, 95), power = 1))
R> hclplot(sequential_hcl(7, h = c(260, 60), c = 60, l = c(40, 95),
+    power = 1))
R> hclplot(diverging_hcl(7, h = c(260, 0), c = 80, l = c(35, 95),
+    power = 1))
```

The multi-hue palette on the left employs a small hue range, resulting in a palette of "blues" just with slightly more distinction of the middle colors in the palette. In contrast, the multi-hue "blue-yellow" palette in the center panel uses a large hue range, resulting in more color contrasts throughout the palette. Finally, the balanced diverging palette in the right panel is constructed from two simple single-hue sequential palettes (for hues 260/blue and 0/red) that are completely balanced between the two "arms" of the palette.

## 5.4. Demonstration of statistical graphics

To demonstrate how different kinds of color palettes work in different kinds of statistical displays, `demoplot()` provides a simple convenience interface to some base graphics with (mostly artificial) data sets. As a first overview, Figure 24 displays all built-in demos with the same sequential heat colors palette: `sequential_hcl(5, "Heat")`. All types of demos can, in principle, deal with arbitrarily many colors from any palette, but the graphics differ in various respects such as:

- Working best for fewer colors (e.g., bar, pie, scatter, lines, ...) vs. many colors (e.g., heatmap, perspective, ...).

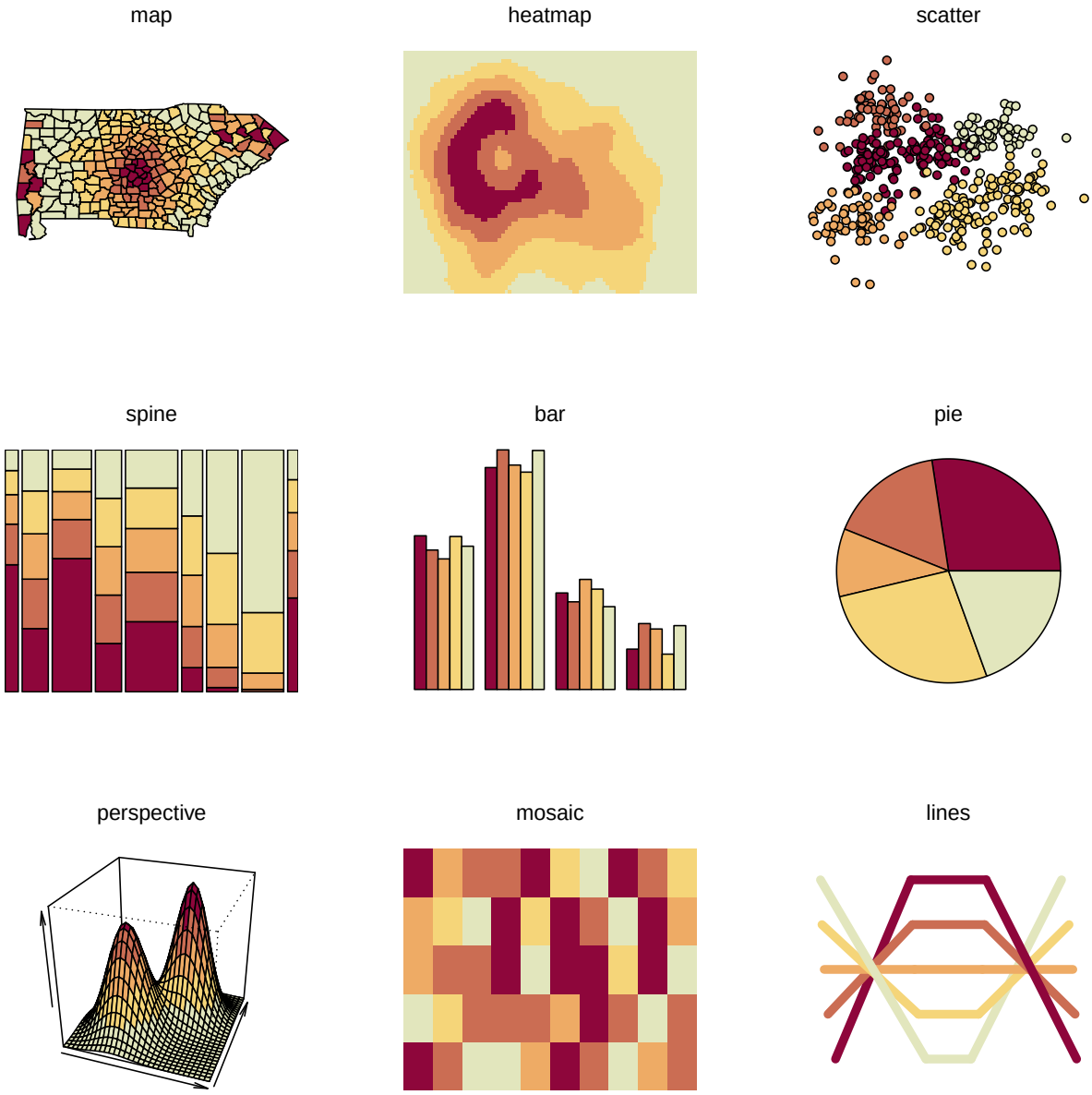


Figure 24: All built-in `demoplot` types with the same `sequential_hcl(5, "Heat")` palette.

- Intended for categorical data (e.g., bar, pie, ...) vs. continuous numeric data (e.g., heatmap, perspective, ...).
- Shading areas (e.g., map, bar, pie, ...) vs. coloring points or lines (scatter, lines).

Hence, in the following Figures 25–27 some further illustrations are organized by type of palette, using suitable demos for the particular palettes.

*Qualitative palettes:* Light pastel colors typically work better for shading areas (pie, left) while darker and more colorful palettes are usually preferred for points (center) or lines (right).

```
R> par(mfrow = c(1, 3))
R> demoplot(qualitative_hcl(4, "Pastel 1"), type = "pie")
```

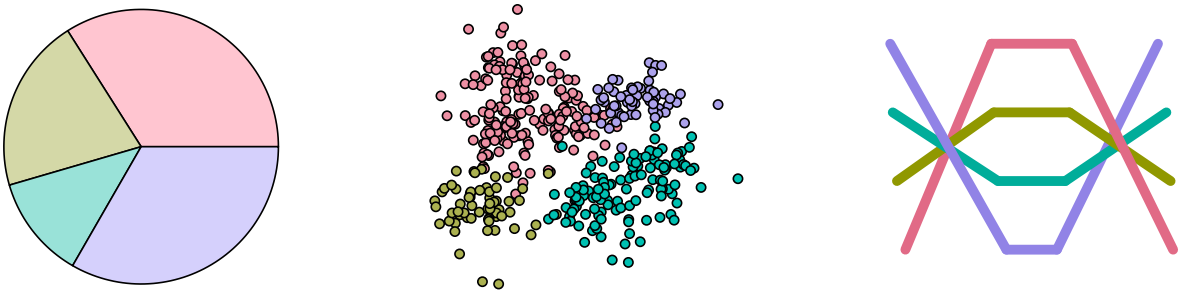

Figure 25: Examples for `demoplot()` with different `qualitative_hcl()` palettes.

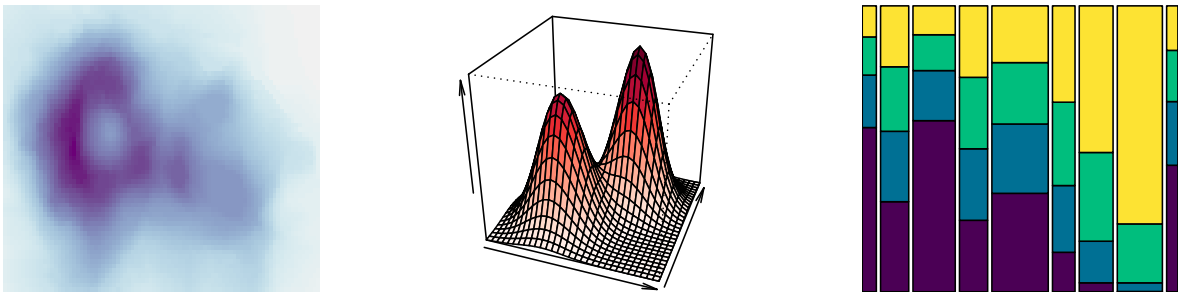

Figure 26: Examples for `demoplot()` with different `sequential_hcl()` palettes.

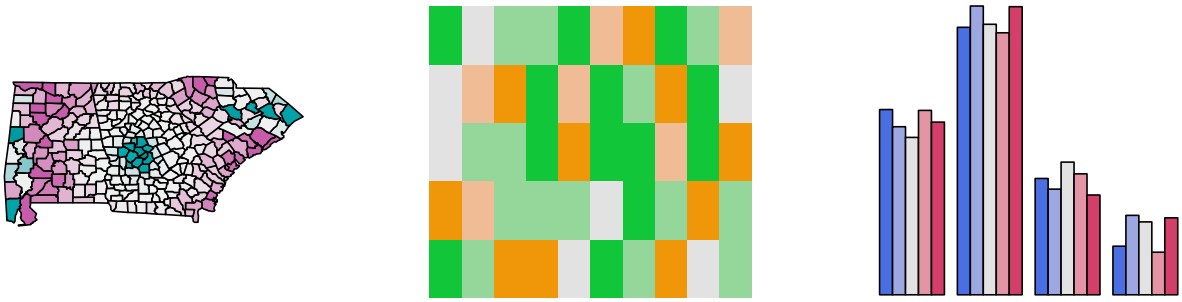

Figure 27: Examples for `demoplot()` with different `diverging_hcl()` palettes.

```
R> demoplot(qualitative_hcl(4, "Set 2"), type = "scatter")
R> demoplot(qualitative_hcl(4, "Dark 3"), type = "lines")
```

*Sequential palettes:* Heatmaps (left) or perspective plots (center) often employ almost continuous gradients with strong luminance contrasts. In contrast, when only a few ordered categories are to be displayed (e.g., in a spine plot, right) more colorful sequential palettes like the viridis palette can be useful.

```
R> par(mfrow = c(1, 3))
R> demoplot(sequential_hcl(99, "Purple-Blue"), type = "heatmap")
R> demoplot(sequential_hcl(99, "Reds"), type = "perspective")
R> demoplot(sequential_hcl( 4, "Viridis"), type = "spine")
```

*Diverging palettes:* In some displays (such as the map, left), it is useful to employ an almost continuous gradient with strong luminance contrast to bring out the extremes. Here, this

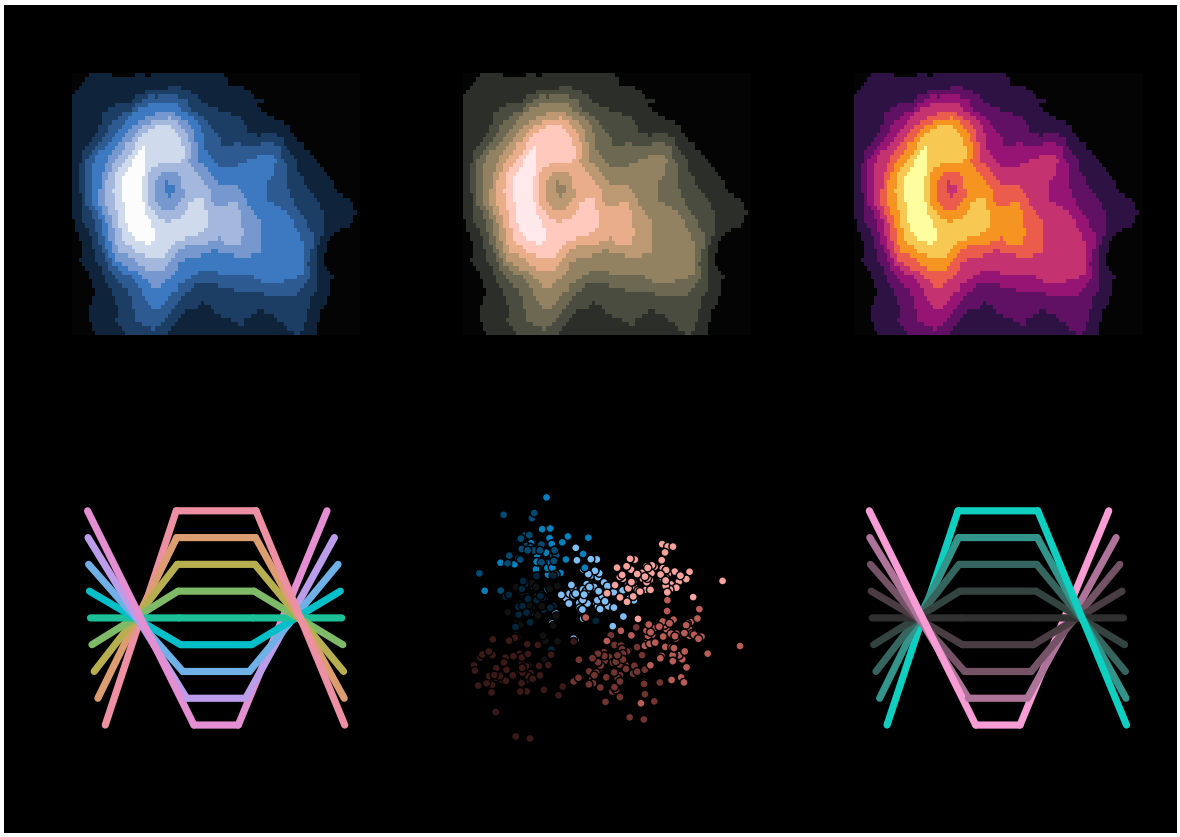

Figure 28: Examples for `demoplot()` with different palettes that work well on a black/dark background.

contrast is amplified by a larger power transformation emphasizing the extremes even further. In contrast, when fewer colors are needed more colorful palettes with lower luminance contrasts can be desired. This is exemplified by a mosaic (center) and bar plot (right).

```
R> par(mfrow = c(1, 3))
R> demoplot(diverging_hcl(99, "Tropic", power = 2.5), type = "map")
R> demoplot(diverging_hcl( 5, "Green-Orange"), type = "mosaic")
R> demoplot(diverging_hcl( 5, "Blue-Red 2"), type = "bar")
```

Figures 25–27 focus on palettes designed for light/white backgrounds. Therefore, to conclude, some palettes are highlighted in Figure 28 that work well on dark/black backgrounds.

```
R> par(mfrow = c(2, 3), bg = "black")
R> demoplot(sequential_hcl(9, "Oslo"), "heatmap")
R> demoplot(sequential_hcl(9, "Turku"), "heatmap")
R> demoplot(sequential_hcl(9, "Inferno", rev = TRUE), "heatmap")
R> demoplot(qualitative_hcl(9, "Set 2"), "lines")
R> demoplot(diverging_hcl(9, "Berlin"), "scatter")
R> demoplot(diverging_hcl(9, "Cyan-Magenta", l2 = 20), "lines")
```

# 6. Color vision deficiency emulation

Different kinds of limitations can be emulated using the physiologically-based model for simulating color vision deficiency (CVD) of Machado, Oliveira, and Fernandes (2009): deuteranomaly (green cone cells defective), protanomaly (red cone cells defective), and tritanomaly (blue cone cells defective). While most other CVD simulations handle only dichromacy, where one of three cones is non-functional, Machado *et al.* (2009) provide a unified model of both dichromacy and anomalous trichromacy, where one cone has shifted spectral sensitivity. As anomalous trichromacy is the most common form of color vision deficiency, it is important to emulate along with the rarer, but more severe dichromacy. Below we briefly describe our R interface to these emulation techniques and show them in practice for a heatmap with sequential palette. Another example with a diverging palette is available at `http://colorspace.R-Forge.R-project.org/articles/color_vision_deficiency.html`. Finally, CVD emulation is particularly useful for bringing out why the RGB rainbow palette is almost always a bad choice in scientific displays. See `http://colorspace.R-Forge.R-project.org/articles/endrainbow.html` for further illustrations.

## 6.1. R functions

The workhorse function to emulate color vision deficiencies is `simulate_cvd()` which can take any vector of valid R colors and transform them according to a certain CVD transformation matrix and transformation equation. The transformation matrices have been established by Machado *et al.* (2009) and are provided in objects `protanomaly_cvd`, `deutanomaly_cvd`, and `tritanomaly_cvd`. The convenience interfaces `deutan()`, `protan()`, and `tritan()` are the high-level functions for simulating the corresponding kind of color blindness with a given `severity` (calling `simulate_cvd()` internally). A severity of `1` corresponds to dichromacy, `0` to normal color vision, and intermediate values to varying severities of anomalous trichromacy. For further guidance on color blindness in relation to statistical graphics see Lumley (2006) which accompanies the R package **dichromat** (Lumley 2013) and is based on earlier emulation techniques (Viénot, Brettel, Ott, M'Barek, and Mollon 1995; Brettel, Viénot, and Mollon 1997; Viénot, Brettel, and Mollon 1999).

## 6.2. Illustration: Heatmap with sequential palette

To illustrate that poor color choices can severely reduce the usefulness of a statistical graphic for readers with color vision deficiencies, we employ the infamous RGB rainbow color palette

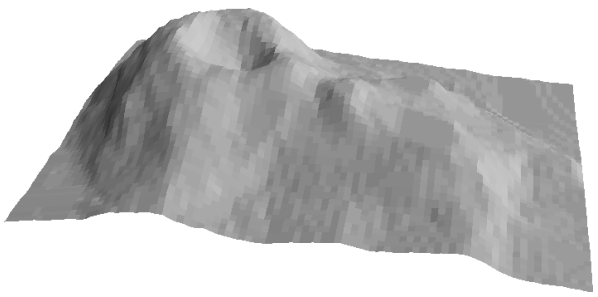

Figure 29: Perspective visualization of Maunga Whau `volcano` data (Mount Eden, Auckland, New Zealand).

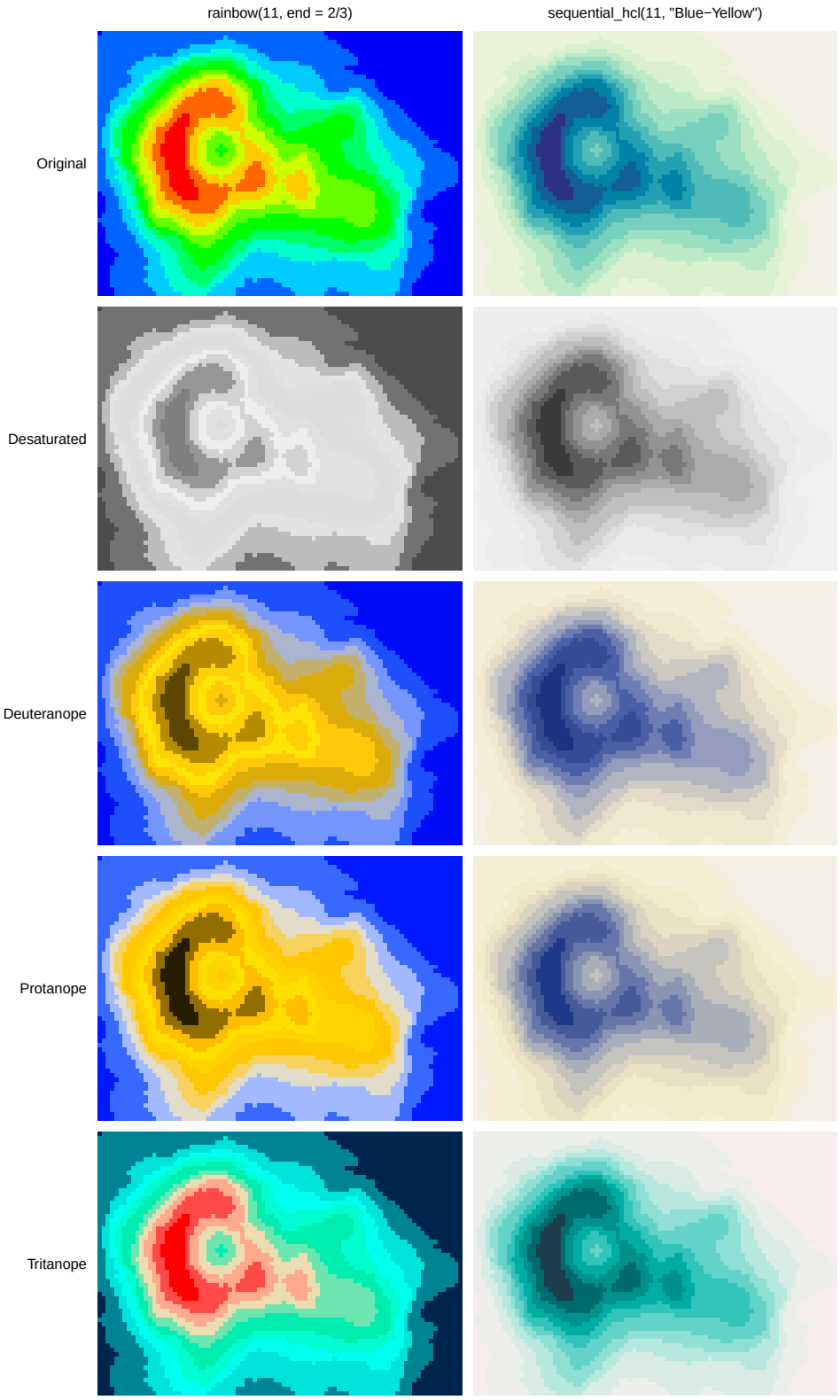


Figure 30: Heatmap of Maunga Whau `volcano` data with RGB rainbow (left) and HCL-based blue-yellow palette (right). The first row shows the original color palettes while subsequent rows emulate various color deficiencies.

in a heatmap. In base R this can be generated by `rainbow(11, end = 2/3)` ranging from red (for high values) to blue (for low values). The poor results for the RGB rainbow palette are contrasted in Figure 30 with a proper sequential palette ranging from dark blue to light yellow: `sequential_hcl(11, "Blue-Yellow")`.

The statistical graphic employed for illustration is a heatmap of the well-known Maunga Whau `volcano` data from base R. This heatmap is easily available as `demoplot(x, "heatmap")` where `x` is the color vector to be used, e.g.,

```
R> rainbow(11, end = 2/3)

 [1] "#FF0000FF" "#FF6600FF" "#FFCC00FF" "#CCFF00FF" "#66FF00FF"
 [6] "#00FF00FF" "#00FF66FF" "#00FFCCFF" "#00CCFFFF" "#0066FFFF"
[11] "#0000FFFF"

R> deutan(rainbow(11, end = 2/3))

 [1] "#5D4700FF" "#B58C01FF" "#FFD005FF" "#FFE408FF" "#FFC809FF"
 [6] "#DBAB0AFF" "#C4B06DFF" "#ACB5D0FF" "#7595FFFF" "#1D50FBFF"
[11] "#000CF7FF"
```

and so on. To aid the interpretation of the heatmap a perspective display using only gray shades is provided in Figure 29, providing another intuitive display of what the terrain around Maunga Whau looks like.

Subsequently, all combinations of palette and color vision deficiency are visualized. Additionally, a grayscale version is created with `desaturate()`. This clearly shows how poorly the RGB rainbow performs, often giving quite misleading impressions of the terrain around Maunga Whau. In contrast, the HCL-based blue-yellow palette works reasonably well in all settings. The most important problem of the RGB rainbow is that it is not monotonic in luminance, making correct interpretation quite hard. Moreover, the red-green contrasts deteriorate substantially in the dichromatic emulations.

## 7. Apps for choosing colors and palettes interactively

To facilitate exploring the package and employing it when working with colors, several graphical user interfaces (GUIs) are provided within the package as **shiny** apps (Chang *et al.* 2020). All of these GUIs/apps can be run locally from within R and are also provided online at http://hclwizard.org/.

- *Palette constructor:* `choose_palette()` or `hclwizard()` or `hcl_wizard()`.
- *Color picker:* `choose_color()` or equivalently `hcl_color_picker()`.
- *Color vision deficiency emulator:* `cvd_emulator()`.

In addition to the **shiny** version, the *palette constructor* app is also available as a Tcl/Tk GUI via R package **tcltk** shipped with base R (R Core Team 2020). The **tcltk** version can only be run locally and is considerably faster while the **shiny** version has a nicer interface with more features and can be run online. The `choose_palette()` function by default starts the **tcltk** version while `hclwizard()`/`hcl_wizard()` by default start the **shiny** version.

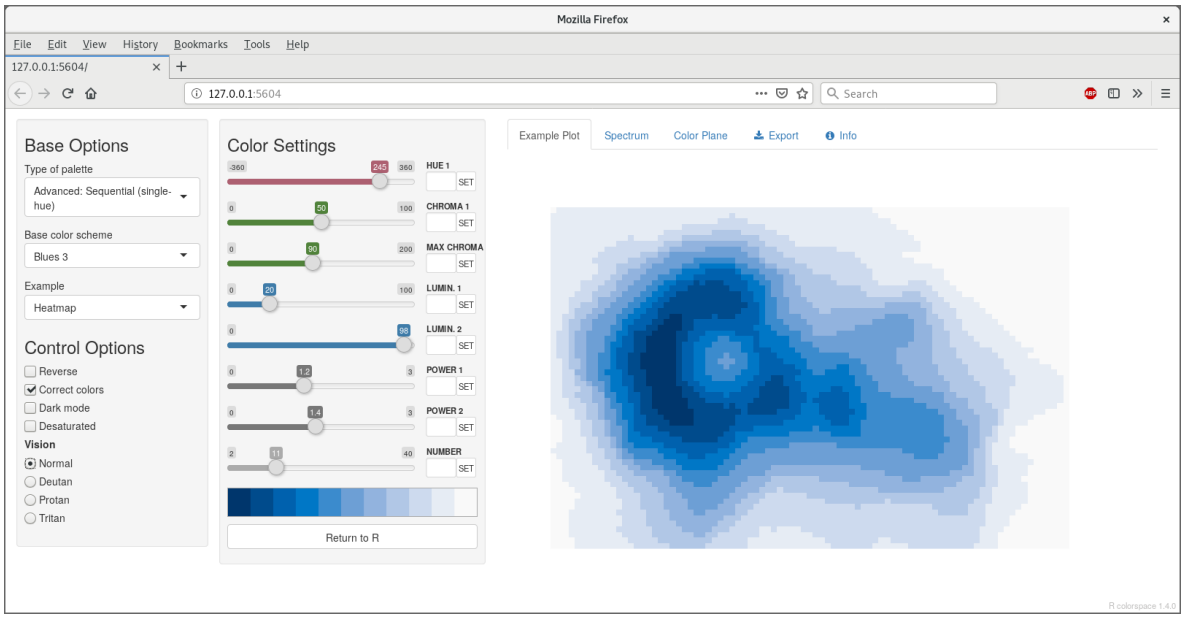


Figure 31: App for interactively choosing HCL-based color palettes: `choose_color()`/`hclwizard()`.

## 7.1. Choose palettes with the HCL color model

The **shiny** version of the palette constructor GUI is shown in Figure 31. It interfaces the `qualitative_hcl()`, `sequential_hcl()`, and `diverging_hcl()` palettes from Section 4. The GUIs allow for interactive modification of the arguments of the respective palette-generating functions, i.e., starting/ending hue, minimal/maximal chroma, minimal/maximal luminance, and power transformations that control how quickly/slowly chroma and/or luminance are changed through the palette. Subsets of the parameters may not be applicable depending on the type of palette chosen.

Optionally, the active palette can be illustrated by using a `specplot()` (see Section 5.2), `hclplot()` (see Section 5.3), or `demoplot()` (see Section 5.4), and assessed using emulation of color vision deficiencies (see Section 6). To facilitate generation of palettes for black/dark backgrounds, a "dark mode" of the GUIs is also available.

The app has been influenced considerably by **ColorBrewer.org** (Harrower and Brewer 2003). Similarities include the selection of a qualitative, sequential, or diverging palette from a list of predefined colors along with an example visualization. However, unlike **ColorBrewer.org** our **shiny** app allows tweaking the HCL parameters underlying each palette. This makes the app much more flexible but also more complex, potentially requiring more thought and experience. Due to the flexibility, our app cannot automatically judge safety regarding color vision deficiencies and printers/photocopiers (as **ColorBrewer.org** does) but instead it allows emulation of color vision deficiencies and desaturation. Finally, **ColorBrewer.org** is geared towards cartography (albeit its palettes are useful much more generally) while our **shiny** app includes a broader range of illustrative displays.

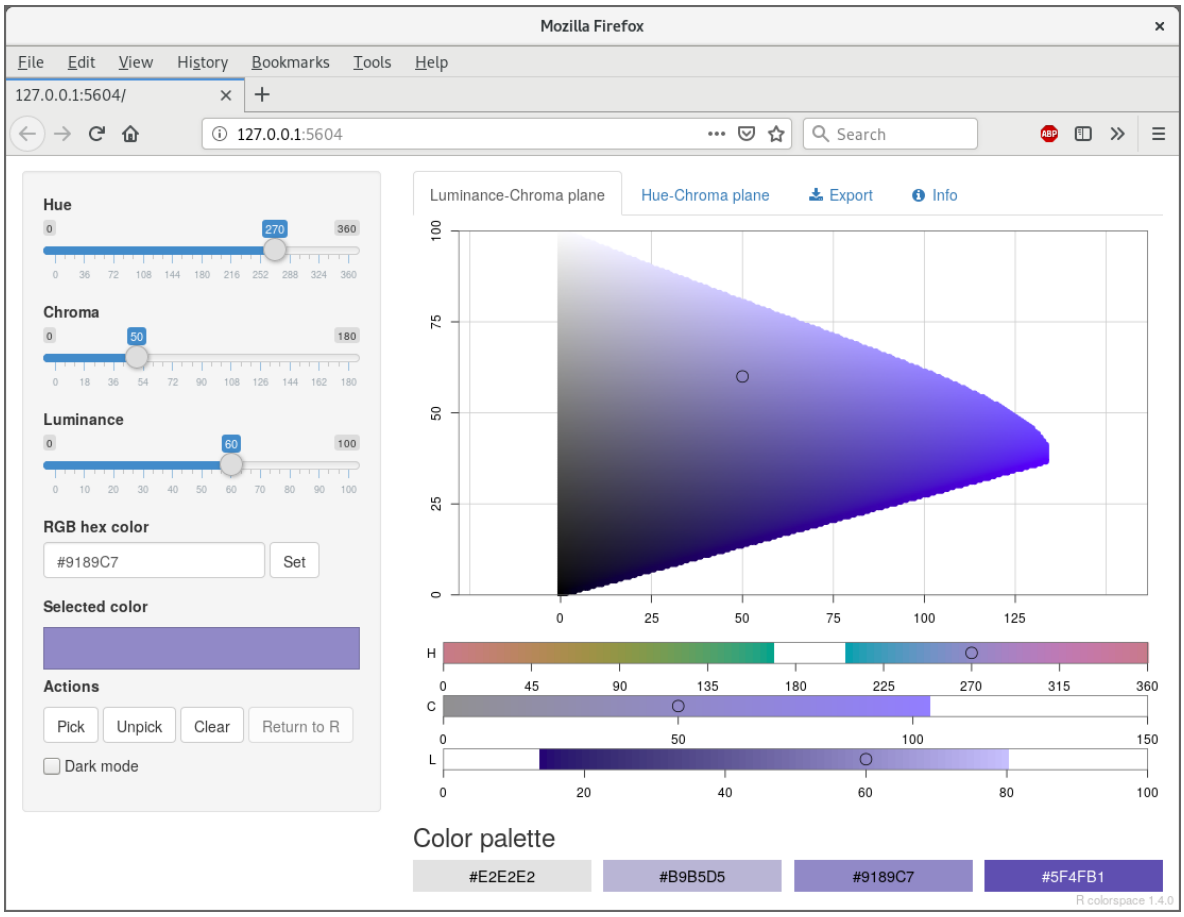


Figure 32: App for interactively choosing individual colors in HCL space: `choose_color()`/`hcl_color_picker()`.

## 7.2. Choose individual colors with the HCL color model

This GUI can be started with either `choose_color()` or equivalently `hcl_color_picker()`. It shows the HCL color space either as a hue-chroma plane for a given luminance value or as a luminance-chroma plane for a given hue. Colors can be entered by:

- Clicking on a color coordinate in the hue-chroma or luminance-chroma plane.
- Specifying the hue/chroma/luminance values via sliders.
- Entering an RGB hex code.

By repeating the selection a palette of colors can be constructed and returned within R for subsequent usage in visualizations.

## 7.3. Emulate color vision deficiencies

This GUI can be started with `cvd_emulator()`. It supports uploading a raster image in JPG or PNG format which is then checked for various kinds of color vision deficiencies at the selected severity. By default the severity is set to 100% and all supported kinds of color vision deficiency are checked for.

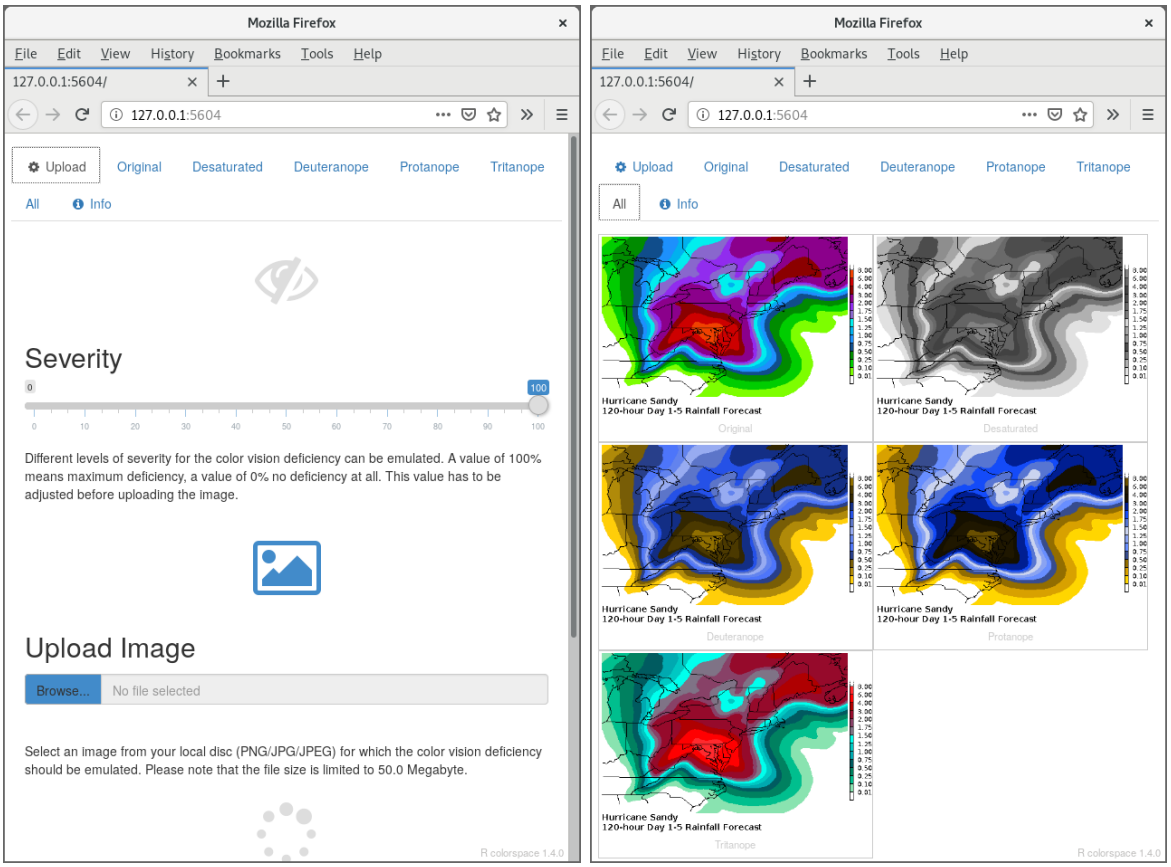


Figure 33: App for emulating color vision deficiencies for uploaded raster images: `cvd_emulator()`.

# 8. Color manipulation and utilities

The **colorspace** package provides several color manipulation utilities that are useful for creating, assessing, or transforming color palettes, namely:

- `desaturate()`: Desaturate colors by chroma removal in HCL space.
- `darken()` and `lighten()`: Algorithmically lighten or darken colors in HCL and/or HLS space.
- `max_chroma()`: Compute maximum chroma for given hue and luminance in HCL space.
- `mixcolor()`: Additively mix two colors by computing their convex combination.

## 8.1. Desaturation in HCL space

Desaturation should map a given color to the gray with the same "brightness". In principle, any perceptually-based color model (HCL, HLS, HSV, ...) could be employed for this but HCL works particularly well because its coordinates capture the perceptual properties better than most other color models.

The `desaturate()` function converts any given hex color code or named R color to the corresponding HCL coordinates and sets the chroma to zero. Thus, only the luminance matters

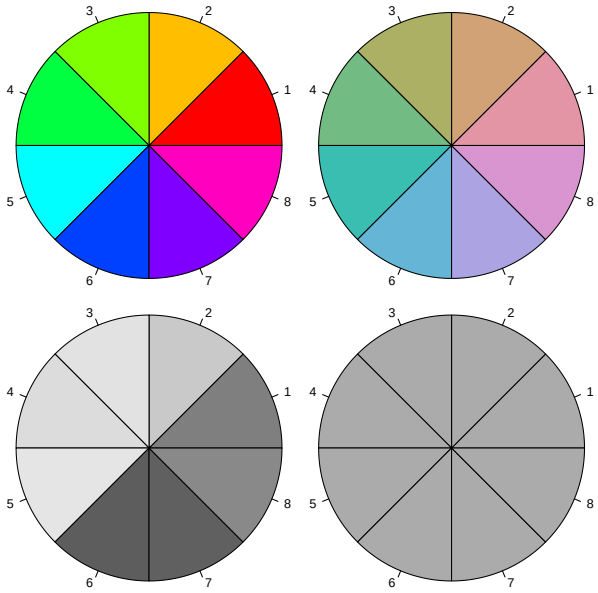

Figure 34: Color wheels in RGB (left) and HCL (right) space in color (top) and desaturated grayscale (bottom).

which captures the "brightness" mentioned above. Finally, the resulting HCL coordinates are transformed back to hex color codes for use in R. First, `desaturate()` is used to desaturate a vector of R color names:

```
R> desaturate(c("white", "orange", "blue", "black"))

[1] "#FFFFFF" "#B8B8B8" "#4C4C4C" "#000000"
```

Notice that the hex codes corresponding to three coordinates in sRGB space are always the same, thus corresponding to gray colors (due to the same amount of red, green, and blue). Analogously, hex color codes can also be transformed – in this case RGB rainbow colors from the base R function `rainbow()`:

```
R> rainbow(3)

[1] "#FF0000FF" "#00FF00FF" "#0000FFFF"

R> desaturate(rainbow(3))

[1] "#7F7F7FFF" "#DCDCDCFF" "#4C4C4CFF"
```

Even this simple example suffices to show that the three RGB rainbow colors have very different grayscale levels. This deficiency is even clearer when using a full color wheel (of colors with hues in [0, 360] degrees). While the RGB `rainbow()` is very unbalanced, the HCL `rainbow_hcl()` (or also `qualitative_hcl()`) is (by design) balanced with respect to luminance.

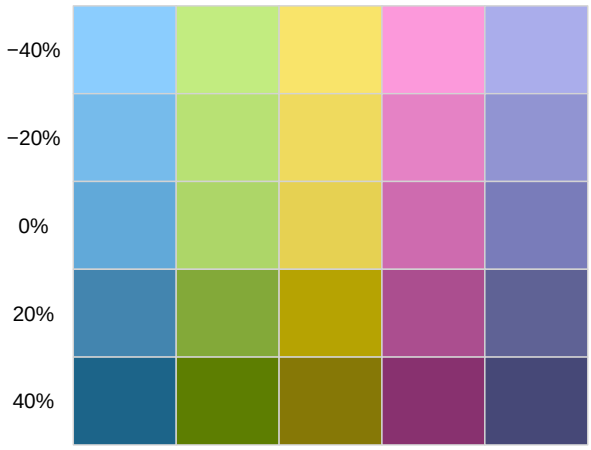


Figure 35: Okabe-Ito palette (0%) along with two levels of both lightening and darkening, respectively.

```
R> wheel <- function(col, radius = 1, ...)
+    pie(rep(1, length(col)), col = col, radius = radius, ...)
R> par(mar = rep(0.5, 4), mfrow = c(2, 2))
R> wheel(rainbow(8))
R> wheel(rainbow_hcl(8))
R> wheel(desaturate(rainbow(8)))
R> wheel(desaturate(rainbow_hcl(8)))
```

## 8.2. Lighten or darken colors

In principle, a similar approach for lightening and darkening colors can be employed as for desaturation above. The colors can simply be transformed to HCL space and then the luminance can either be decreased (turning the color darker) or increased (turning it lighter) while preserving the hue and chroma coordinates. This strategy typically works well for lightening colors, although in some situations the result can be somewhat too colorful. Conversely, when darkening rather light colors with little chroma, this can result in rather gray colors.

In these situations, an alternative might be to apply the analogous strategy in HLS space which is frequently used in HTML style sheets. However, this strategy may also yield colors that are either too gray or too colorful. A compromise that sometimes works well is to adjust the luminance coordinate in HCL space but to take the chroma coordinate corresponding to the HLS transformation.

We have found that typically the HCL-based transformation performs best for lightening colors and this is hence the default in `lighten()`. For darkening colors, the combined strategy often works best and is hence the default in `darken()`. In either case it is recommended to try the other available strategies in case the default yields unexpected results.

Regardless of the chosen color space, the adjustment of the `L` component by a certain `amount` can occur by two methods, relative (the default) or absolute. For example, `L - 100 * amount` is used for absolute darkening, or `L * (1 - amount)` for relative darkening. See `?lighten` and `?darken` for more details.

For illustration the qualitative palette suggested by Okabe and Ito (2008) is transformed by two levels of both lightening and darkening, respectively (see Figure 35).

```
R> oi <- c("#61A9D9", "#ADD668", "#E6D152", "#CE6BAF", "#797CBA")
R> swatchplot( "-40%" = lighten(oi, 0.4), "-20%" = lighten(oi, 0.2),
+    "  0%" = oi, " 20%" = darken(oi, 0.2), " 40%" = darken(oi, 0.4),
+    off = c(0, 0))
```

## 8.3. Adjust transparency of colors

Alpha transparency is useful for making colors semi-transparent, e.g., for overlaying different elements in graphics (Wikipedia 2020i). An alpha value (or alpha channel) of 0 (or 00 in hex strings) corresponds to fully transparent and an alpha value of 1 (or FF in hex strings) corresponds to fully opaque. If a color hex string in R does not provide an explicit alpha transparency, the color is assumed to be fully opaque.

The `adjust_transparency()` function can be used to adjust the alpha transparency of a set of colors. It always returns a hex color specification. This hex color can have the alpha transparency added/removed/modified depending on the specification of the argument `alpha`:

- `alpha = NULL`: Returns a hex vector with alpha transparency only if needed. Thus, it keeps the alpha transparency for the colors (if any) but only if different from opaque.
- `alpha = TRUE`: Returns a hex vector with alpha transparency for all colors, using opaque (FF) as the default if missing.
- `alpha = FALSE`: Returns a hex vector without alpha transparency for all colors (even if the original colors had non-opaque alpha).
- `alpha` numeric: Returns a hex vector with alpha transparency for all colors set to the `alpha` argument (recycled if necessary).

For illustration, the transparency of a single black color is modified to three alpha levels: fully transparent, semi-transparent, and fully opaque, respectively. Black can be equivalently specified by name (`"black"`), hex string (`"#000000"`), or integer position in the palette (`1`).

```
R> adjust_transparency("black",   alpha = c(0, 0.5, 1))

[1] "#00000000" "#00000080" "#000000FF"

R> adjust_transparency("#000000", alpha = c(0, 0.5, 1))

[1] "#00000000" "#00000080" "#000000FF"

R> adjust_transparency(1,         alpha = c(0, 0.5, 1))

[1] "#00000000" "#00000080" "#000000FF"
```

Subsequently, different settings of `alpha` are illustrated for adjusting a vector with three shades of gray, specified by name (`gray`, opaque), opaque hex string (`"#BEBEBE"`), and semi-transparent hex string (`"#BEBEBE80"`). Four types of adjustment are shown: only if necessary (`alpha = NULL`), add (`alpha = TRUE`), remove (`alpha = FALSE`), or modify (`alpha = 0.8`).

```
R> x <- c("gray", "#BEBEBE", "#BEBEBE80")
R> adjust_transparency(x, alpha = NULL)

[1] "#BEBEBE"   "#BEBEBE"   "#BEBEBE80"

R> adjust_transparency(x, alpha = TRUE)

[1] "#BEBEBEFF" "#BEBEBEFF" "#BEBEBE80"

R> adjust_transparency(x, alpha = FALSE)

[1] "#BEBEBE" "#BEBEBE" "#BEBEBE"

R> adjust_transparency(x, alpha = 0.8)

[1] "#BEBEBECC" "#BEBEBECC" "#BEBEBECC"
```

### 8.4. Maximum chroma for given hue and luminance

As the possible combinations of chroma and luminance in HCL space depend on hue, it is not obvious which trajectories through HCL space are possible prior to trying a specific HCL coordinate by calling `polarLUV()`. To avoid having to fix up the color upon conversion to RGB `hex()` color codes, the `max_chroma()` function computes (approximately) the maximum chroma possible. For illustration we show that for given luminance (here: $L = 50$) the maximum chroma varies substantially with hue:

```
R> max_chroma(h = seq(0, 360, by = 60), l = 50)

[1] 137.96  59.99  69.06  39.81  65.45 119.54 137.96
```

Similarly, maximum chroma also varies substantially across luminance values for a given hue (here: $H = 120$, green):

```
R> max_chroma(h = 120, l = seq(0, 100, by = 20))

[1]   0.00  28.04  55.35  82.79 110.28   0.00
```

### 8.5. Additive mixing of two colors

In additive color models like `RGB()` or `XYZ()` it can be useful to combine colors by additive mixing. Below a fully saturated red and green are mixed, yielding a medium brownish yellow.

```
R> R <- RGB(1, 0, 0)
R> G <- RGB(0, 1, 0)
R> Y <- mixcolor(0.5, R, G)
R> Y

       R   G B
[1,] 0.5 0.5 0
```

## 9. Summary and discussion

This paper provides an overview of the broad capabilities of the **colorspace** package for selecting individual colors or color palettes, manipulating these colors, and employing them in various kinds of visualizations.

In particular, the package provides various qualitative, sequential, and diverging palettes derived by relatively simple trajectories in HCL (hue-chroma-luminance) space. In contrast to many other packages providing modern balanced color palettes (such as **ColorBrewer.org**, **CARTO**, **viridis**, or **scico**) special emphasis is given to flexibility of the palettes, which can be adjusted to the particular needs of a given data visualization. The paper also provides various tips and tricks for choosing an effective palette in a given situation. Further useful guidance is provided in many sources, including: Ware (1988), Okabe and Ito (2008), Aigner (2010), Stauffer *et al.* (2015), Zhang (2015), Rost (2018), Wilke (2019), and Ciechanowski (2019), among many others.

There are other R packages that can complement the palettes provided by **colorspace**. **Polychrome** (Coombes and Brock 2020; Coombes, Brock, Abrams, and Abruzzo 2019) implements strategies for qualitative palettes with many "categories". While the qualitative palettes in Section 4 yield only about 6–8 clearly distinguishable colors due to the fixed chroma and luminance, **Polychrome** relaxes this restriction and can thus find a larger number of colors in CIELUV space that are spaced as far apart as possible. Some of these palettes have also been included in the base R function `palette.colors()` in **grDevices** (Zeileis *et al.* 2019) along with other qualitative palettes that provide more distinguishable colors than `qualitative_hcl()`. The palette collection packages **pals** (Wright 2019) and **paletteer** (Hvitfeldt 2020) also provide a wide range of prespecified palettes, including some qualitative schemes with many categories. Note that the palettes are quite diverse, though, and not all of them are equally suitable for coding qualitative information. The visualization functions in **colorspace** from Section 5 may be helpful in assessing their properties. **roloc** (Murrell 2018a,b) also provides color conversions, not between numeric color spaces, but rather from numeric color spaces to English color names.

In addition to the R version of **colorspace**, a Python 2/Python 3 (Van Rossum *et al.* 2011) re-implementation is available at `https://github.com/retostauffer/python-colorspace` which is currently in beta. In the paper we focus on the more mature R implementation replication materials for most examples are also available for Python.

## Computational details

The results in this paper were obtained using R 4.0.3 (R Core Team 2020) with the packages **colorspace** 2.0-0 (Ihaka *et al.* 2020), **ggplot2** 3.3.2 (Wickham *et al.* 2020), **RColorBrewer** 1.1.2 (Neuwirth 2014), **rcartocolor** 2.0.0 (Nowosad 2019), **viridis** 0.5.1 (Garnier 2018), **scico** 1.2.0 (Pedersen and Crameri 2020). R itself and all packages used are available from the Comprehensive R Archive Network (CRAN) at `https://CRAN.R-project.org/`.

## Acknowledgments

The authors would like to thank the journal editors, the associate editor, and two anonymous reviewers for their constructive and helpful feedback that substantially improved the paper.

## References


Aigner W (2010). "Perception and Visualization." URL http://www.ifs.tuwien.ac.at/~silvia/wien/vu-infovis/PDF-Files/02_perception-visualization_1up.pdf.

Brettel H, Viénot F, Mollon JD (1997). "Computerized Simulation of Color Appearance for Dichromats." *Journal of the Optical Society of America A*, **14**, 2647–2655. doi:10.1364/josaa.14.002647.

Brewer CA (1999). "Color Use Guidelines for Data Representation." In *Proceedings of the Section on Statistical Graphics, American Statistical Association*, pp. 55–60. Alexandria. URL http://www.personal.psu.edu/faculty/c/a/cab38/ColorSch/ASApaper.html.

CARTO (2019). "CARTOColors – Data-Driven Color Schemes." URL https://carto.com/carto-colors/.

Chang W, Cheng J, Allaire JJ, Xie Y, McPherson J (2020). *shiny: Web Application Framework for R*. R package version 1.5.0, URL https://CRAN.R-project.org/package=shiny.

Ciechanowski B (2019). "Color Spaces." URL https://ciechanow.ski/color-spaces/.

Coombes KR, Brock G (2020). *Polychrome: Qualitative Palettes with Many Colors*. R package version 1.2.5, URL https://CRAN.R-project.org/package=Polychrome.

Coombes KR, Brock G, Abrams ZB, Abruzzo LV (2019). "**Polychrome**: Creating and Assessing Qualitative Palettes with Many Colors." *Journal of Statistical Software, Code Snippets*, **90**(1), 1–23. doi:10.18637/jss.v090.c01.

Crameri F (2018). "Geodynamic Diagnostics, Scientific Visualisation and **StagLab** 3.0." *Geoscientific Model Development*, **11**(6), 2541–2562. doi:10.5194/gmd-11-2541-2018.

Gama J, Davis G (2018). ***colorscience**: Color Science Methods and Data*. R package version 1.0.5, URL https://CRAN.R-project.org/package=colorscience.

Garnier S (2018). ***viridis**: Default Color Maps from **matplotlib***. R package version 0.5.1, URL https://CRAN.R-project.org/package=viridis.

Harrower MA, Brewer CA (2003). "**ColorBrewer.org**: An Online Tool for Selecting Color Schemes for Maps." *The Cartographic Journal*, **40**(1), 27–37. doi:10.1179/000870403235002042. URL http://ColorBrewer.org/.

Hawkins E, McNeall D, Stephenson D, Williams J, Carlson D (2014). "The End of the Rainbow – An Open Letter to the Climate Science Community." URL http://www.climate-lab-book.ac.uk/2014/end-of-the-rainbow/.

Horvath M, Lipka C (2016). "sRGB Gamut within CIELCHuv Color Space Isosurface." Wikimedia Commons, URL https://commons.wikimedia.org/wiki/File:SRGB_gamut_within_CIELCHuv_color_space_isosurface.png.

Horvath M, Lipka C (2017). "sRGB Gamut within CIELCHuv Color Space Mesh." Wikimedia Commons, URL https://commons.wikimedia.org/wiki/File:SRGB_gamut_within_CIELCHuv_color_space_mesh.webm.

Hvitfeldt E (2020). *paletteer: Comprehensive Collection of Color Palettes*. R package version 1.2.0, URL https://CRAN.R-project.org/package=paletteer.

Ihaka R (2003). "Colour for Presentation Graphics." In K Hornik, F Leisch, A Zeileis (eds.), *Proceedings of the 3rd International Workshop on Distributed Statistical Computing, Vienna, Austria*. URL http://www.ci.tuwien.ac.at/Conferences/DSC-2003/Proceedings/.

Ihaka R, Murrell P, Hornik K, Fisher JC, Stauffer R, Wilke CO, McWhite CD, Zeileis A (2020). **colorspace**: *A Toolbox for Manipulating and Assessing Colors and Palettes*. R package version 2.0-0, URL https://CRAN.R-project.org/package=colorspace.

Kaiser PK, Boynton RM (1996). *Human Color Vision*. 2nd edition. Optical Society of America, Washington.

Knoblauch K (2002). "Color Vision." In S Yantis, H Pashler (eds.), *Steven's Handbook of Experimental Psychology – Sensation and Perception*, volume 1, 3rd edition, pp. 41–75. John Wiley & Sons, New York.

Lumley T (2006). "Color Coding and Color Blindness in Statistical Graphics." *ASA Statistical Computing & Graphics Newsletter*, **17**(2), 4–7. URL http://stat-computing.org/newsletter/issues/scgn-17-2.pdf.

Lumley T (2013). **dichromat**: *Color Schemes for Dichromats*. R package version 2.0-0, URL https://CRAN.R-project.org/package=dichromat.

Machado GM, Oliveira MM, Fernandes LAF (2009). "A Physiologically-Based Model for Simulation of Color Vision Deficiency." *IEEE Transactions on Visualization and Computer Graphics*, **15**(6), 1291–1298. doi:10.1109/tvcg.2009.113. URL http://www.inf.ufrgs.br/~oliveira/pubs_files/CVD_Simulation/CVD_Simulation.html.

Murrell P (2018a). "Generating Colour Names: The **roloc** Package for R." URL https://stattech.wordpress.fos.auckland.ac.nz/2018/01/25/2018-01-generating-colour-names-the-roloc-package-for-r/.

Murrell P (2018b). **roloc**: *Convert Colour Specification to Colour Name*. R package version 0.1-1, URL https://CRAN.R-project.org/package=roloc.

Neuwirth E (2014). **RColorBrewer**: *ColorBrewer Palettes*. R package version 1.1-2, URL https://CRAN.R-project.org/package=RColorBrewer.

Nowosad J (2019). **rcartocolor**: *CARTOColors Palettes*. R package version 2.0.0, URL https://CRAN.R-project.org/package=rcartocolor.

Okabe M, Ito K (2008). "Color Universal Design (CUD): How to Make Figures and Presentations That Are Friendly to Colorblind People." URL http://jfly.iam.u-tokyo.ac.jp/color/.

Pedersen TL, Crameri F (2020). *scico: Colour Palettes Based on the Scientific Colour-Maps*. R package version 1.2.0, URL https://CRAN.R-project.org/package=scico.

Pedersen TL, Nicolae B, François R (2020). *farver: Vectorised Colour Conversion and Comparison*. R package version 2.0.3, URL https://CRAN.R-project.org/package=farver.

Poynton C (2009). "Frequently-Asked Questions about Color." URL http://www.poynton.com/ColorFAQ.html, accessed 2020-11-03.

Ram K, Wickham H (2018). ***wesanderson**: A Wes Anderson Palette Generator*. R package version 0.3.6, URL https://CRAN.R-project.org/package=wesanderson.

R Core Team (2020). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL https://www.R-project.org/.

Rost LC (2018). "What to Consider When Choosing Colors for Data Visualization." URL https://blog.datawrapper.de/colors/.

Smith N, Van der Walt S (2015). "A Better Default Colormap for **matplotlib**." In *SciPy 2015 – Scientific Computing with Python*. Austin. URL https://www.youtube.com/watch?v=xAoljeRJ3lU.

Stauffer R, Mayr GJ, Dabernig M, Zeileis A (2015). "Somewhere over the Rainbow: How to Make Effective Use of Colors in Meteorological Visualizations." *Bulletin of the American Meteorological Society*, **96**(2), 203–216. doi:10.1175/BAMS-D-13-00155.1.

Tufte ER (1990). *Envisioning Information*. Graphics Press, Cheshire.

Van Rossum G, *et al.* (2011). *Python Programming Language*. URL https://www.python.org/.

Viénot F, Brettel H, Mollon JD (1999). "Digital Video Colourmaps for Checking the Legibility of Displays by Dichromats." *Color Research and Application*, **24**(4), 243–252. doi:10.1002/(sici)1520-6378(199908)24:4<243::aid-col5>3.3.co;2-v.

Viénot F, Brettel H, Ott L, M'Barek AB, Mollon JD (1995). "What Do Colour-Blind People See?" *Nature*, **376**, 127–128. doi:10.1038/376127a0.

Ware C (1988). "Color Sequences for Univariate Maps: Theory, Experiments and Principles." *IEEE Computer Graphics and Applications*, **8**(5), 41–49. doi:10.1109/38.7760.

Ware C (2004). "Color." In *Information Visualization: Perception for Design*, chapter 4, pp. 103–149. Morgan Kaufmann Publishers.

Wickham H (2016). ***ggplot2**: Elegant Graphics for Data Analysis*. 2nd edition. Springer-Verlag. doi:10.1007/978-0-387-98141-3.

Wickham H, Chang W, Henry L, Pedersen TL, Takahashi K, Wilke CO, Woo K, Yutani H, Dunnington D (2020). *ggplot2: Create Elegant Data Visualisations Using the Grammar of Graphics*. R package version 3.3.2, URL https://CRAN.R-project.org/package=ggplot2.

Wikipedia (2020a). "CIE 1931 Color Space — Wikipedia, The Free Encyclopedia." URL https://en.wikipedia.org/wiki/CIE_1931_color_space, accessed 2020-11-03.

Wikipedia (2020b). "CIELAB Color Space — Wikipedia, The Free Encyclopedia." URL https://en.wikipedia.org/wiki/CIELAB_color_space, accessed 2020-11-03.

Wikipedia (2020c). "CIELUV — Wikipedia, The Free Encyclopedia." URL https://en.wikipedia.org/wiki/CIELUV, accessed 2020-11-03.

Wikipedia (2020d). "Color Space — Wikipedia, The Free Encyclopedia." URL https://en.wikipedia.org/wiki/Color_space, accessed 2019-03-11.

Wikipedia (2020e). "HCL Color Space — Wikipedia, The Free Encyclopedia." URL https://en.wikipedia.org/wiki/HCL_color_space, accessed 2019-08-23.

Wikipedia (2020f). "HSL and HSV — Wikipedia, The Free Encyclopedia." URL https://en.wikipedia.org/wiki/HSL_and_HSV, accessed 2020-11-03.

Wikipedia (2020g). "RGB Color Space — Wikipedia, The Free Encyclopedia." URL https://en.wikipedia.org/wiki/RGB_color_space, accessed 2020-11-03.

Wikipedia (2020h). "sRGB — Wikipedia, The Free Encyclopedia." URL https://en.wikipedia.org/wiki/sRGB, accessed 2020-11-03.

Wikipedia (2020i). "Web Colors — Wikipedia, The Free Encyclopedia." URL https://en.wikipedia.org/wiki/Web_colors, accessed 2019-03-11.

Wilke CO (2019). *Fundamentals of Data Visualization*. O'Reilly Media. URL https://clauswilke.com/dataviz/color-basics.html.

Wilkinson L (2005). *The Grammar of Graphics*. 2nd edition. Springer-Verlag.

Wright K (2019). *pals: Color Palettes, Colormaps, and Tools to Evaluate Them*. R package version 1.6, URL https://CRAN.R-project.org/package=pals.

Zeileis A, Gaslam B, Murrell P, Pedersen TL (2018). "Benchmarking Color Space Conversions." Twitter discussion, URL https://twitter.com/AchimZeileis/status/1076228936810590208.

Zeileis A, Hornik K, Murrell P (2009). "Escaping RGBland: Selecting Colors for Statistical Graphics." *Computational Statistics & Data Analysis*, **53**, 3259–3270. doi:10.1016/j.csda.2008.11.033.

Zeileis A, Murrell P (2019). "HCL-Based Color Palettes in **grDevices**." The R Blog, URL https://developer.R-project.org/Blog/public/2019/04/01/hcl-based-color-palettes-in-grdevices/.

Zeileis A, Murrell P, Maechler M, Sarkar D (2019). "A New `palette()` for R." The R Blog, URL https://developer.R-project.org/Blog/public/2019/11/21/a-new-palette-for-r/.

Zhang S (2015). "Finding the Right Color Palettes for Data Visualizations." URL https://blog.graphiq.com/finding-the-right-color-palettes-for-data-visualizations-fcd4e707a283.

**Affiliation:**

Achim Zeileis
Universität Innsbruck
Department of Statistics
Faculty of Economics and Statistics
Universitätsstr. 15
6020 Innsbruck, Austria
E-mail: Achim.Zeileis@R-project.org
URL: https://eeecon.uibk.ac.at/~zeileis/

*Journal of Statistical Software* http://www.jstatsoft.org/
published by the Foundation for Open Access Statistics http://www.foastat.org/
November 2020, Volume 96, Issue 1 *Submitted:* 2019-03-18
doi:10.18637/jss.v096.i01 *Accepted:* 2019-12-02