Package ‘farver’

February 28, 2021

Type Package

Title High Performance Colour Space Manipulation

Version 2.1.0

Maintainer Thomas Lin Pedersen <thomasp85@gmail.com>

Description The encoding of colour can be handled in many different ways, using different colour spaces. As different colour spaces have different uses, efficient conversion between these representations are important. The ‘farver’ package provides a set of functions that gives access to very fast colour space conversion and comparisons implemented in C++, and offers speed improvements over the 'convertColor' function in the 'grDevices' package.

License MIT + file LICENSE

Encoding UTF-8

SystemRequirements C++11

RoxygenNote 7.1.1

URL https://farver.data-imaginist.com,
https://github.com/thomasp85/farver

BugReports https://github.com/thomasp85/farver/issues

Suggests testthat (>= 2.1.0), covr

NeedsCompilation yes

Author Thomas Lin Pedersen [cre, aut]
(https://orcid.org/0000-0002-5147-4711),
Berendea Nicolae [aut] (Author of the ColorSpace C++ library),
Romain François [aut] (https://orcid.org/0000-0002-2444-4226)

Repository CRAN

Date/Publication 2021-02-28 19:30:02 UTC
compare_colour

R topics documented:

compare_colour .................................................... 2
convert_colour ..................................................... 3
decode_colour ....................................................... 5
encode_colour ....................................................... 6
manip_channel ....................................................... 7
native-encoding ..................................................... 10

Index 11

---

compare_colour  Calculate the distance between colours

Description

There are many ways to measure the distance between colours. farver provides 5 different algorithms, ranging from simple euclidean distance in RGB space, to different perceptual measures such as CIE2000.

Usage

```r
compare_colour(
  from,
  to = NULL,
  from_space,
  to_space = from_space,
  method = "euclidean",
  white_from = "D65",
  white_to = white_from
)
```

Arguments

from, to  Numeric matrices with colours to compare - the format is the same as that for convert_colour(). If to is not set from will be compared with itself and only the upper triangle will get calculated

from_space, to_space  The colour space of from and to respectively. to_space defaults to be the same as from_space.

method  The method to use for comparison. Either 'euclidean', 'cie1976', 'cie94', 'cie2000', or 'cmc'

white_from, white_to  The white reference of the from and to colour space. Will only have an effect for relative colour spaces such as Lab and luv. Any value accepted by as_white_ref() allowed.
Value

A numeric matrix with the same number of rows as colours in from and the same number of columns as colours in to. If to is not given, only the upper triangle will be returned.

Handling of non-finite and out of bounds values

NA, NaN, -Inf, and Inf are treated as invalid input and will result in NA values for the colour. If a given colourspace has finite bounds in some of their channels, the input will be capped before conversion, and the output will be capped before returning, so that both input and output colours are valid colours in their respective space. This means that converting back and forth between two colourspaces may result in a change in the colour if the gamut of one of the spaces is less than the other.

Examples

```r
r <- decode_colour(rainbow(10))
h <- decode_colour(heat.colors(15))

# Compare two sets of colours
compare_colour(r, h, 'rgb', method = 'cie2000')

# Compare a set of colours with itself
compare_colour(r, from_space = 'rgb', method = 'cmc')

# Compare colours from different colour spaces
h_luv <- convert_colour(h, 'rgb', 'luv')
compare_colour(r, h_luv, 'rgb', 'luv')
```

**convert_colour**  
*Convert between colour spaces*

Description

This function lets you convert between different representations of colours. The API is reminiscent of `grDevices::convertColor()`, but the performance is much better. It is not assured that `grDevices::convertColor()` and `convert_colour()` provide numerically equivalent conversion at 16bit level as the formula used are potentially slightly different. For all intend and purpose, the resulting colours will be equivalent though.

Usage

```r
convert_colour(colour, from, to, white_from = "D65", white_to = white_from)
```
Arguments

colour
A numeric matrix (or an object coercible to one) with colours encoded in the rows and the different colour space values in the columns. For all colours except 'cmyk' this will mean a matrix with three columns - for 'cmyk' it means four columns.

from, to
The input and output colour space. Allowed values are: "cmy", "cmyk", "hsl", "hsb", "hsv", "lab" (CIE L*ab), "hunterlab" (Hunter Lab), "oklab", "lch" (CIE Lch(ab) / polarLAB), "luv", "rgb" (sRGB), "xyz", "yxy" (CIE xyY), "hcl" (CIE Lch(uv) / polarLuv), or "oklch" (Polar form of oklab)

white_from, white_to
The white reference of the from and to colour space. Will only have an effect for relative colour spaces such as Lab and luv. Any value accepted by as_white_ref() allowed.

Value
A numeric matrix with the same number of rows as colour and either 3 or 4 columns depending on the value of to. If colour is given as a data.frame the output will be a data.frame as well

Handling of non-finite and out of bounds values
NA, NaN, -Inf, and Inf are treated as invalid input and will result in NA values for the colour. If a given colourspace has finite bounds in some of their channels, the input will be capped before conversion, and the output will be capped before returning, so that both input and output colours are valid colours in their respective space. This means that converting back and forth between two colourspaces may result in a change in the colour if the gamut of one of the spaces is less than the other.

Note
This function and convertColor() are not numerically equivalent due to rounding errors, but for all intend and purpose they give the same results.

See Also
grDevices::convertColor(), grDevices::col2rgb()

Examples

```r
spectrum <- decode_colour(rainbow(10))
speclab <- convert_colour(spectrum, 'rgb', 'lab')
speclab

# Convert between different white references
convert_colour(speclab, 'lab', 'lab', white_from = 'D65', white_to = 'F10')
```
**decode_colour**

*Decode RGB hex-strings into colour values*

**Description**

This is a version of `grDevices::col2rgb()` that returns the colour values in the standard form expected by `farver` (matrix with a row per colour). As with `encode_colour()` it can do colour conversion on the fly, meaning that you can decode a hex string directly into any of the supported colour spaces.

**Usage**

```r
decode_colour(colour, alpha = FALSE, to = "rgb", white = "D65", na_value = NA)
```

**Arguments**

- `colour`: A character vector of hex-encoded values or a valid colour name as given in `grDevices::colours()`.
- `alpha`: If `TRUE` the alpha channel will be returned as well (scaled between 0 and 1). If no alpha channel exists in the colour it will be assumed 1. If `FALSE` any alpha channel is ignored.
- `to`: The output colour space. Allowed values are: "cmy", "cmyk", "hsl", "hsb", "hsv", "lab" (CIE L*ab), "hunterlab" (Hunter Lab), "oklab", "lch" (CIE Lch(ab) / polarLAB), "luv", "rgb" (sRGB), "xyz", "xyy" (CIE xyY), "hcl" (CIE Lch(uv) / polarLuv), or "oklch" (Polar form of oklab).
- `white`: The white reference of the output colour space. Will only have an effect for relative colour spaces such as Lab and luv. Any value accepted by `as_white_ref()` allowed.
- `na_value`: A valid colour string or `NA` to use when `colour` contains `NA` elements. The general approach in `farver` is to carry `NA` values over, but if you want to mimick `col2rgb()` you should set `na_value = 'transparent'`, i.e. treat `NA` as transparent white.

**Value**

A numeric matrix with a row for each element in `colour` and either 3, 4, or 5 columns depending on the value of `alpha` and `to`.

**Handling of non-finite and out of bounds values**

`NA`, `NaN`, `-Inf`, and `Inf` are treated as invalid input and will result in `NA` values for the colour. If a given colourspace has finite bounds in some of their channels, the input will be capped before conversion, and the output will be capped before returning, so that both input and output colours are valid colours in their respective space. This means that converting back and forth between two colourspace may result in a change in the colour if the gamut of one of the spaces is less than the other.
See Also

Other encoding and decoding functions: encode_colour(), manip_channel

Examples

# basic use
decode_colour(c('#43e1f6', 'steelblue', '#67ce9fe4'))

# Return alpha as well (no alpha value is interpreted as 1)
decode_colour(c('#43e1f6', 'steelblue', '#67ce9fe4'), alpha = TRUE)

# Decode directly into specific colour space
decode_colour(c('#43e1f6', 'steelblue', '#67ce9fe4'), to = 'lch')

--

**encode_colour**

*Encode colours into RGB hex-strings*

Description

This is a version of `grDevices::rgb()` that works with the standard colour format used in farver (matrix or data.frame with colours in rows). It further support taking input from any colour space.

Usage

```r
encode_colour(colour, alpha = NULL, from = "rgb", white = "D65")
```

Arguments

- **colour**: A numeric matrix (or an object coercible to one) with colours encoded in the rows and the different colour space values in the columns. For all colour spaces except 'cmyk' this will mean a matrix with three columns - for 'cmyk' it means four columns.
- **alpha**: A numeric vector between 0 and 1. Will be recycled to the number of rows in `colour`. If NULL or a single NA it will be ignored.
- **from**: The input colour space. Allowed values are: "cmy", "cmyk", "hsl", "hsv", "lab" (CIE L*ab), "hunterlab" (Hunter Lab), "oklab", "lch" (CIE Lch(ab) / polarLAB), "1uv", "rgb" (sRGB), "xyz", "xyy" (CIE xyy), "hcl" (CIE Lch(uv) / polarLuv), or "oklch" (Polar form of oklab).
- **white**: The white reference of the input colour space. Will only have an effect for relative colour spaces such as Lab and luv. Any value accepted by `as_white_ref()` allowed.

Value

A character vector with colours encoded as #RRGGBB(AA)
Handling of non-finite and out of bounds values

NA, NaN, -Inf, and Inf are treated as invalid input and will result in NA values for the colour. If a given colourspace has finite bounds in some of their channels, the input will be capped before conversion, and the output will be capped before returning, so that both input and output colours are valid colours in their respective space. This means that converting back and forth between two colourspaces may result in a change in the colour if the gamut of one of the spaces is less than the other.

Note

The output may differ slightly from that of \texttt{grDevices::rgb()} since \texttt{rgb()} doesn’t round numeric values correctly.

See Also

Other encoding and decoding functions: \texttt{decode_colour()}, \texttt{manip_channel}

Examples

```r
spectrum <- decode_colour(rainbow(10))
encode_colour(spectrum)

# Attach alpha values
encode_colour(spectrum, alpha = c(0.5, 1))

# Encode from a different colour space
spectrum_hcl <- convert_colour(spectrum, 'rgb', 'hcl')
encode_colour(spectrum_hcl, from = 'hcl')
```

manip_channel

Modify colour space channels in hex-encoded colour strings

Description

This set of functions allows you to modify colours as given by strings, without first decoding them. For large vectors of colour values this should provide a considerable speedup.

Usage

```r
set_channel(
  colour,  # colour to modify
  channel,  # channel to modify
  value,  # value to set
  space = "rgb",  # current colour space
  white = "D65",  # white point
  na_value = NA)  # value to use for NA values
```
manip_channel

) add_to_channel(
    colour,
    channel,
    value,
    space = "rgb",
    white = "D65",
    na_value = NA
)

multiply_channel(
    colour,
    channel,
    value,
    space = "rgb",
    white = "D65",
    na_value = NA
)

raise_channel(
    colour,
    channel,
    value,
    space = "rgb",
    white = "D65",
    na_value = NA
)

cap_channel(
    colour,
    channel,
    value,
    space = "rgb",
    white = "D65",
    na_value = NA
)

get_channel(colour, channel, space = "rgb", white = "D65", na_value = NA)

Arguments

colour          A character string giving colours, either as hexadecimal strings or accepted
colour names.
channel         The channel to modify or extract as a single letter, or 'alpha' for the alpha
colour channel.
value           The value to modify with
space           The colour space the channel pertains to. Allowed values are: "cmy", "cmyk",
Manipulate channel values.

Manipulate the channel values for a colour space. This function allows changing
the white reference, adding or multiplying a channel, setting lower and upper
boundaries, and more.

Arguments

- `colour`: A valid colour string or vector.
- `white`: The white reference of the channel colour space. Will only have an
effect for relative colour spaces such as Lab and luv. Any value accepted by
`as_white_ref()` allowed.
- `na_value`: A valid colour string or NA to use when colour contains NA
elements. The general approach in farver is to carry NA values over, but if you
want to mimic `col2rgb()` you should set `na_value = 'transparent'`, i.e. treat
NA as transparent white.
- `value`: A character vector of the same length as colour (or a numeric vector in
the case of `get_channel()`).

Examples

```r
spectrum <- rainbow(10)

# set a specific channel
set_channel(spectrum, 'r', c(10, 50))
set_channel(spectrum, 'l', 50, space = 'lab')
set_channel(spectrum, 'alpha', c(0.5, 1))

# Add value to channel
add_to_channel(spectrum, 'r', c(10, 50))
add_to_channel(spectrum, 'l', 50, space = 'lab')

# Multiply a channel
multiply_channel(spectrum, 'r', c(10, 50))
multiply_channel(spectrum, 'l', 50, space = 'lab')

# set a lower bound on a channel
raise_channel(spectrum, 'r', c(10, 50))
raise_channel(spectrum, 'l', 20, space = 'lab')

# set an upper bound on a channel
cap_channel(spectrum, 'r', c(100, 50))
cap_channel(spectrum, 'l', 20, space = 'lab')
```
native-encoding

Convert to and from the R native colour representation

Description

Colours in R are internally encoded as integers when they are passed around to graphics devices. The encoding splits the 32 bit in the integer between red, green, blue, and alpha, so that each get 8 bit, equivalent to 256 values. It is very seldom that an R user is subjected to this representation, but it is present in the nativeRaster format which can be obtained from e.g. capturing the content of a graphic device (using dev.cap()) or reading in PNG files using png::readPNG(native = TRUE). It is very rare that you might need to convert back and forth between this format, but it is provided here for completeness.

Usage

encode_native(colour, ...)

decode_native(colour)

Arguments

colour For encode_native either a vector of hex-encoded colours/colour names or a matrix encoding colours in any of the supported colour spaces. If the latter, the colours will be encoded to a hex string using encode_colour() first. For decode_native it is a vector of integers.

... Arguments passed on to encode_colour()

Value

encode_native() returns an integer vector and decode_native() returns a character vector, both matching the length of the input.

Examples

# Get native representation of navyblue and #228B22
native_col <- encode_native(c('navyblue', '#228B22'))
native_col

# Convert back
decode_native(native_col)
Index

* encoding and decoding functions
  
  - `decode_colour`, 5
  - `encode_colour`, 6
  - `manip_channel`, 7
  
  - `add_to_channel (manip_channel)`, 7
  - `as_white_ref()`, 2, 4–6, 9
  
  - `cap_channel (manip_channel)`, 7
  - `col2rgb()`, 5, 9
  - `compare_colour`, 2
  - `convert_colour`, 3
  - `convert_colour()`, 2
  - `convertColor()`, 4
  
  - `decode_colour`, 5, 7, 9
  - `decode_native (native-encoding)`, 10
  
  - `encode_colour`, 6, 9
  - `encodecolour()`, 5, 10
  - `encode_native (native-encoding)`, 10
  
  - `get_channel (manip_channel)`, 7
  - `grDevices::col2rgb()`, 4, 5
  - `grDevices::colours()`, 5
  - `grDevices::convertColor()`, 3, 4
  - `grDevices::rgb()`, 6, 7
  
  - `manip_channel`, 6, 7, 7
  - `multiply_channel (manip_channel)`, 7
  
  - `native-encoding`, 10
  
  - `raise_channel (manip_channel)`, 7
  
  - `set_channel (manip_channel)`, 7