8. Image Processing - Color Image Processing

Institute for Biomedical Imaging, Hamburg University of Technology

- Lecture: Prof. Dr.-Ing. Tobias Knopp
- Exercise: Konrad Scheffler, M.Sc.

Table of Contents

8.1 Overview

Until now we only handled images with a single value per pixel. In this lecture we discuss how this translates to color images.

A color image can be interpreted as *multi-channel image* with three different color channels. The function f representing the image is a mapping

$$f : \Omega \rightarrow [0,1]^3.$$

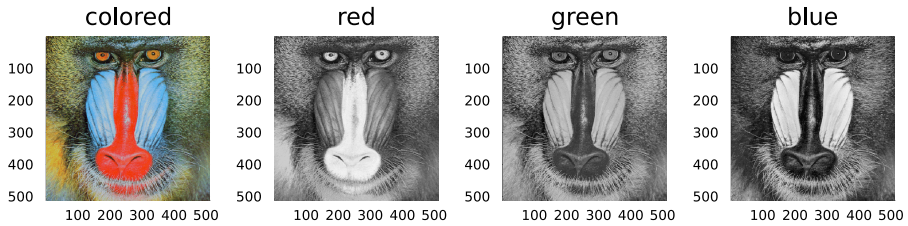
Each pixel value is thus a vector

$$f(\mathbf{r}) = \begin{pmatrix} f_\mathrm{r}(\mathbf{r}) \\ f_\mathrm{g}(\mathbf{r}) \\ f_\mathrm{b}(\mathbf{r}) \end{pmatrix}$$

where $f_\mathrm{r}(\mathbf{r})$, $f_\mathrm{g}(\mathbf{r})$, and $f_\mathrm{b}(\mathbf{r})$ are the scalar channels representing the colors red, green, and blue.

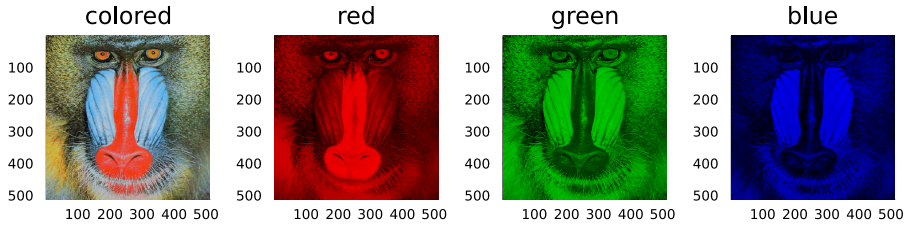
The space covered by the three colors is called the color space. RGB is the default color space used in digital image processing most of the time.

Let us load an image and look at the individual channels:



RGB is an additive color space where a value of 1 means that the color is fully present while a value of 0 means that the color is not present at all. In the image of Lena we can see that the red channel is much brighter then the other ones.

We have shown the individual color channels as grayscale images such that the intensity of the different channels can be directly compared. Instead we can also visualize them using the base colors:



True Color and False Color

One can group colored image into two groups:

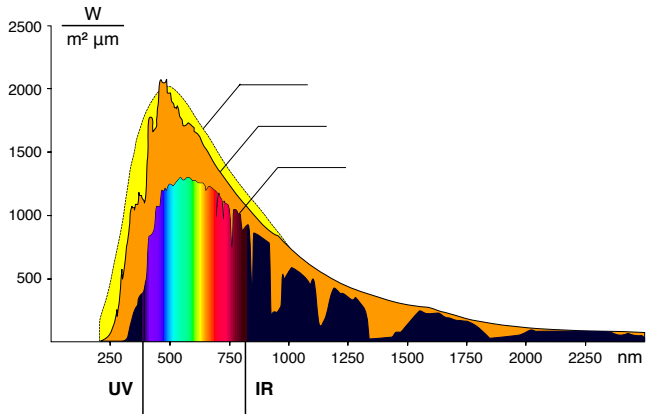
- **True color** images are those representing real objects that reflect with a certain wavelength.
- **False color** or **pseudocolor** images are those representing some abstract quantity using the color to transport the data in a way that the information is well received by a human observer.

An example of the first is the image of Lena. An example of the later would be a height map of a landscape.

8.2 Color Fundamentals

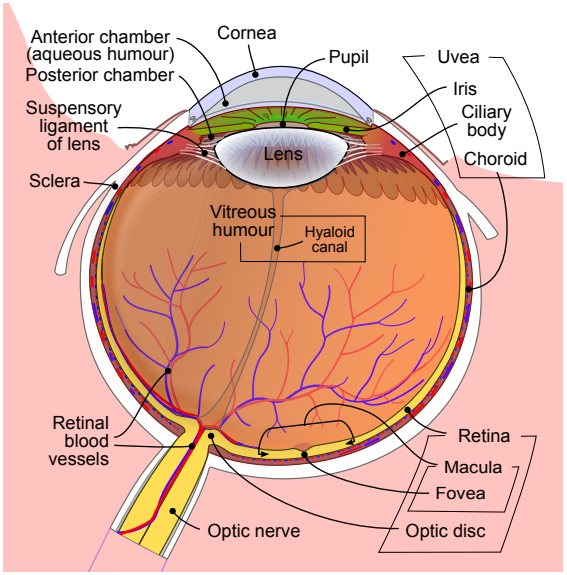
Color is actually not a direct property of physical objects. Instead objects reflect light that appears as color in the visual perception system of humans. When using colors in digital devices we are trying to emulate this procedure.

The light traveling through space is the superposition of waves with different wavelengths. Sunlight, for instance, has a broad wavelength range being emitted, as is shown in the following picture:



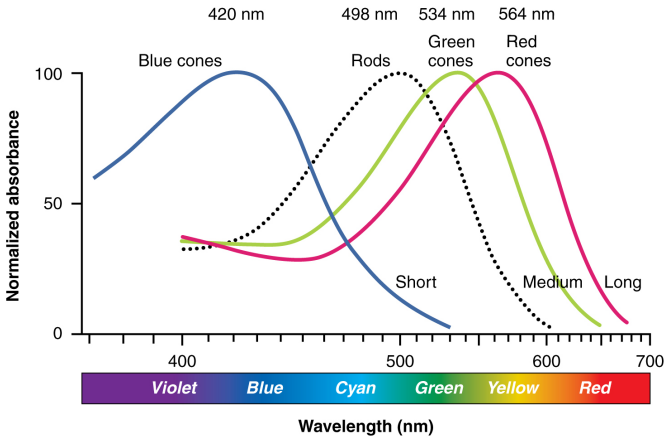
The human eye can only detect a certain range of wavelengths (between 400 nm und 780 nm).

In the following image a human eye is shown. In the retina, the photoreceptor cells are detecting the light intensity. This information is then transported to the human brain via the optical nerve.



There are two types of photoreceptor cells:

- Rods are responsible for seeing in the dark. They only have one color channel and are perceived as gray values.
- Cones are responsible for seeing colors and have three cone types (S,M,L). The S cone is perceived as blue, the M cone is perceived as green and the L cone is perceived as red. They react on the following wavelengths



What becomes apparent now is that the cones are generating a three-channel signal for a each wavelength and that our brain then combines them into one perceived color. The shown colors appear if the light has a single wavelength. In case of multiple wavelengths that are superimposing, one perceives a mixture.

Definition

The response to a stimulus with spectral composition Φ is given by the three integrated responses

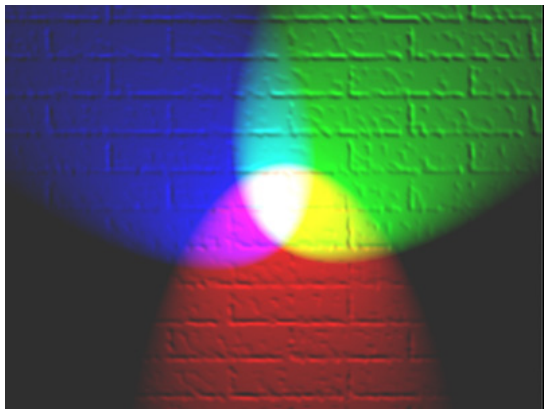
$$L = \int_0^\infty \Phi(\lambda)\bar{L}(\lambda)\mathrm{d}\lambda$$
$$M = \int_0^\infty \Phi(\lambda)\bar{M}(\lambda)\mathrm{d}\lambda$$
$$S = \int_0^\infty \Phi(\lambda)\bar{S}(\lambda)\mathrm{d}\lambda,$$

where $\bar{L}$, $\bar{M}$, and $\bar{S}$ (sometimes without bar) are the cone response functions. We refer to these responses as *tristimulus values*.

Note

The cones are on purpose not named RGB but SML (small, medium, long). This is because they are not related to a single color. For instance, when looking at a purely green wavelength, all three cones are receiving a signal. The signal is very small in the S cone, much larger in the M cone and a little bit lower in the L cone.

Mixing is also shown in the following image where three different light sources (RGB) are used to create mixtures on a reflecting wall.



One important thing now is that one can generate the same color in different ways. For instance:

- We can generate yellow by using a monochromatic light source with 580 nm wavelength
- We can take two light sources (540 nm (green) and 700 nm (red)) and mix them together.

This effect is called metamerism and is the basis of digital output devices.

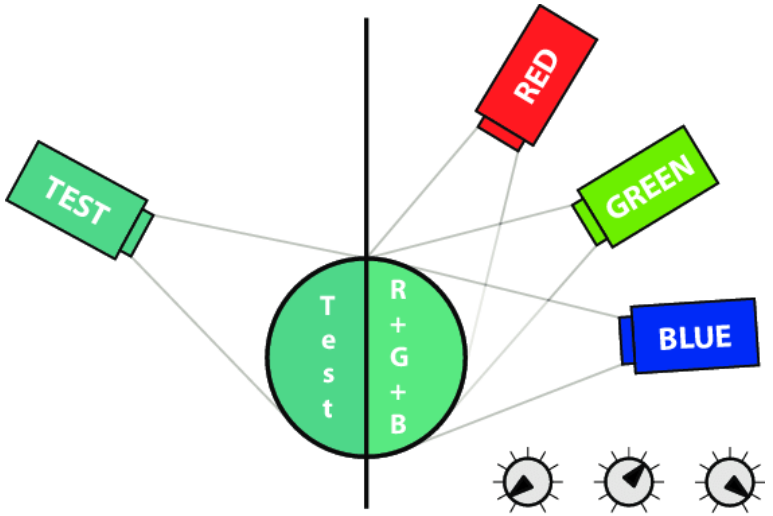
Definition

Grassmann's Laws describe empirically the perception of mixtures of colored lights.
→ **Symmetry law:** If color stimulus A matches color stimulus B, then B matches A
→ **Transitive law:** If A matches B and B matches C, then A matches C
→ **Proportionality law:** If A matches B, then αA matches αB, $\alpha \geq 0$
→ **Additivity law:** If A matches B, C matches D, and $A + C$ matches $B + D$, then $A + D$ matches $B + C$

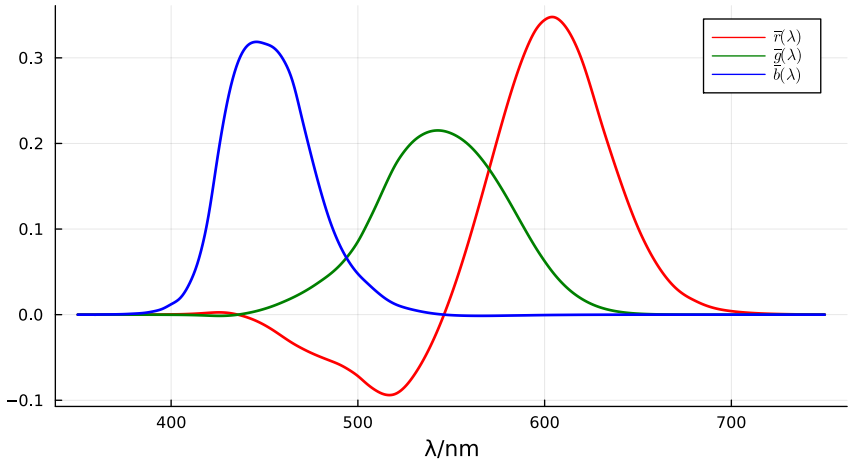
8.2.1 Color Matching

Since color is made in the human brain it is difficult to measure a perceived color. The connection between the wavelength spectrum and the perceived color was developed by the International Commission on Illumination, see CIE1931color_space.

The idea is to take three different monochromatic light sources at standardized wavelengths of 700 nm (red), 546.1 nm (green) and 435.8 nm (blue) and try to match each color in the visible spectrum (generated by a monochromatic light source) by adding red, green and blue. The experimental setting looks like this:



The resulting color matching functions $\bar{r}(\lambda)$, $\bar{g}(\lambda)$, $\bar{b}(\lambda)$ look like this:



The test pattern was swept from 380 nm to 780 nm and a set of people has adjusted the nobs to match the colors.

What is quite interesting are the negative values. These are colors on the spectrum that cannot be created using RGB. Instead one has added the amount of red to the test pattern to match the color.

Note

You should now realize that our display technology cannot visualize all possible colors since they are also based on mixing RGB values and diodes cannot make negative light

Definition

Similar to the LMS tristimulus values we can define RGB tristimulus values as

$$R = \int_0^\infty \Phi(\lambda)\bar{r}(\lambda)\mathrm{d}\lambda$$
$$G = \int_0^\infty \Phi(\lambda)\bar{g}(\lambda)\mathrm{d}\lambda$$
$$B = \int_0^\infty \Phi(\lambda)\bar{b}(\lambda)\mathrm{d}\lambda,$$

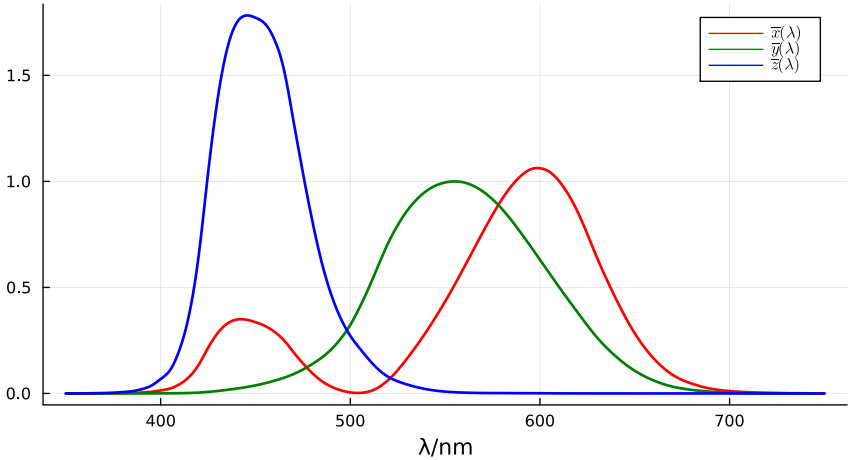
If we use the monochromatic light source with wavelength λ we have $\Psi(\lambda) = \delta(\lambda)$, which means we determine $\bar{r}(\lambda)$, $\bar{g}(\lambda)$, and $\bar{b}(\lambda)$ by putting Dirac functions into this equation and looking at the RGB tristimulus values.

8.2.2 XYZ Color Space

Since negative values are difficult to handle, the international commission has introduced virtual values XYZ that can be transformed back and forth to RGB using the transformation:

$$\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = \frac{1}{0.176\,97} \begin{pmatrix} 0.490\,00 & 0.310\,00 & 0.200\,00 \\ 0.176\,97 & 0.812\,40 & 0.010\,63 \\ 0.000\,00 & 0.010\,00 & 0.990\,00 \end{pmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix}$$

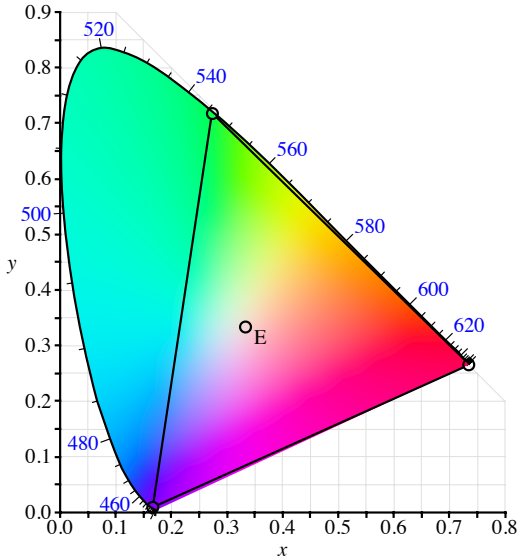
If we take the RGB stimulus functions $\bar{r}(\lambda)$, $\bar{g}(\lambda)$, $\bar{b}(\lambda)$ and calculate the associated $\bar{x}(\lambda)$, $\bar{y}(\lambda)$, $\bar{z}(\lambda)$ we obtain:



We can now derive normalized values

$$x = \frac{X}{X+Y+Z}$$
$$y = \frac{Y}{X+Y+Z}$$
$$z = \frac{Z}{X+Y+Z} = 1 - x - y$$

which allows us to express the color by just the two values xy which can be used to draw a chromaticity diagram:



This is named the color gamut and it looks like a horse shoe. The three dots are the three base colors RGB and within the triangle one has all the colors that can be created using (positive) RGB mixing. On the upper left are the colors that cannot be created using RGB. With XYZ we can now represent all possible colors in the specified wavelength range.

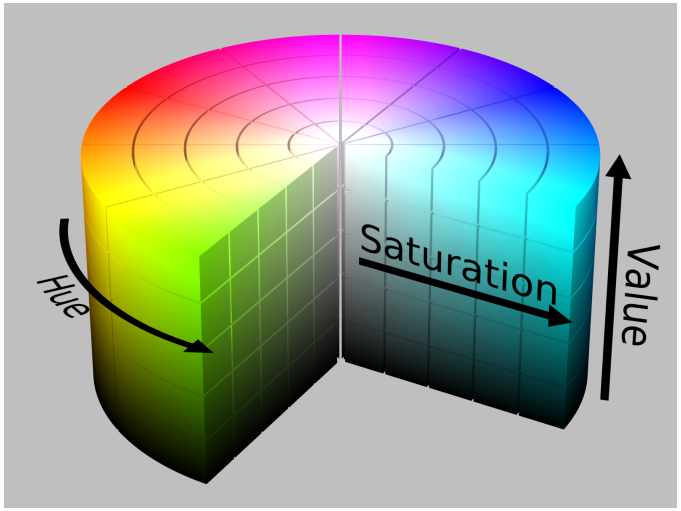
> **Note**
>
> One can also normalize the RGB values leading to rgb values and derive a color gamut for RGB colors.

8.2.3 HSB/HSV Color Space

Instead of RGB and XYZ we can also characterize a color by the three terms

- **brightness**: This describes how dark a color is.
- **hue**: This describes the dominant wavelength in the color.
- **saturation**: This describes the colorfulness.

This is named the HSB or HSV color space.



Lets play around with this:

H = 0.0

S = 0.0

B = 0.0

```
1 HSB(H,S,B)
```

Basically what is happening here is that you select a base color and then either make it darker by putting black to it (brightness change) or you put white to it (saturation change).

The brightness of an image is related to the luminance which is the total amount of light being transmitted.

Hue and saturation can be combined to be the chromaticity of a color, which is the value chosen in the color gamut before.

8.2.4 RGB Color Space

We already introduced the RGB color space and how it historically arose. RGB is a device dependent color space since the location of the R, G, and B primitives in the xy chromacity diagram is not uniquely defined, i.e. there exist different definitions and standards. This is why in practice it requires proper color management with color profiles that characterize a color input or output device.

When using RGB in the computer, the values are normalized to the range $[0, 1]$, sometimes also [0,255] when considering 8-bit values. Furthermore the values are often non-linearly transformed using a gamma transformation to account for the fact that the sensitivity of the human eye for bright colors is different than for dark colors.

Let us have a look at the sRGB profile, which is the one being regularly used (e.g. in the web).

To convert sRGB values R_{srgb}, G_{srgb}, B_{srgb} to CIE XYZ we first need to apply the non-linear transformation

$$C_{\text{linear}} = \begin{cases} \dfrac{C_{\text{srgb}}}{12.92}, & C_{\text{srgb}} \leq 0.04045 \\ \left(\dfrac{C_{\text{srgb}} + 0.055}{1.055}\right)^{2.4}, & C_{\text{srgb}} > 0.04045 \end{cases}$$

where C is R, G, or B.

These gamma-expanded values (sometimes called *linear values* or *linear-light values*) are multiplied by the following matrix to obtain CIE XYZ:

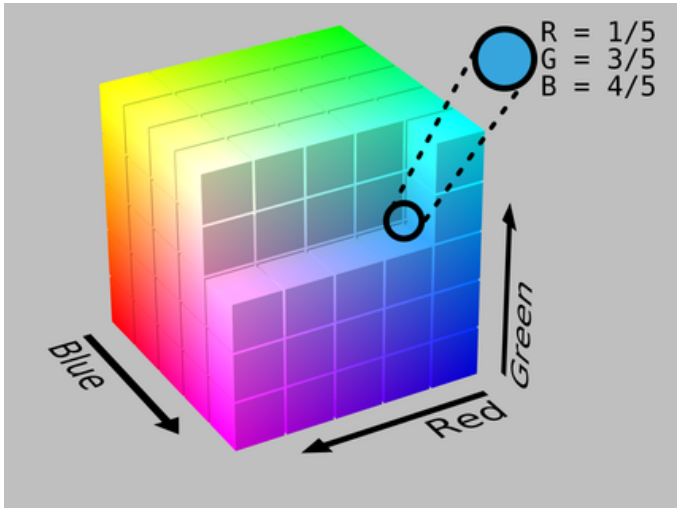
$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} 0.4124 & 0.3576 & 0.1805 \\ 0.2126 & 0.7152 & 0.0722 \\ 0.0193 & 0.1192 & 0.9505 \end{bmatrix} \begin{bmatrix} R_{\text{linear}} \\ G_{\text{linear}} \\ B_{\text{linear}} \end{bmatrix}$$

This matrix can be derived from the table

Chromaticity	Red	Green	Blue	White point
x	0.6400	0.3000	0.1500	0.3127
y	0.3300	0.6000	0.0600	0.3290
Y	0.2126	0.7152	0.0722	1.0000

that specifies where in the xy diagram the RGB primitives lay. Since xy are normalized values it is also necessary to report Y which can be used to recover XYZ.

Geometrically the RGB colors are usually shown as a cube:



And of course you can also play around with RGB in this notebook:

R = 0.0

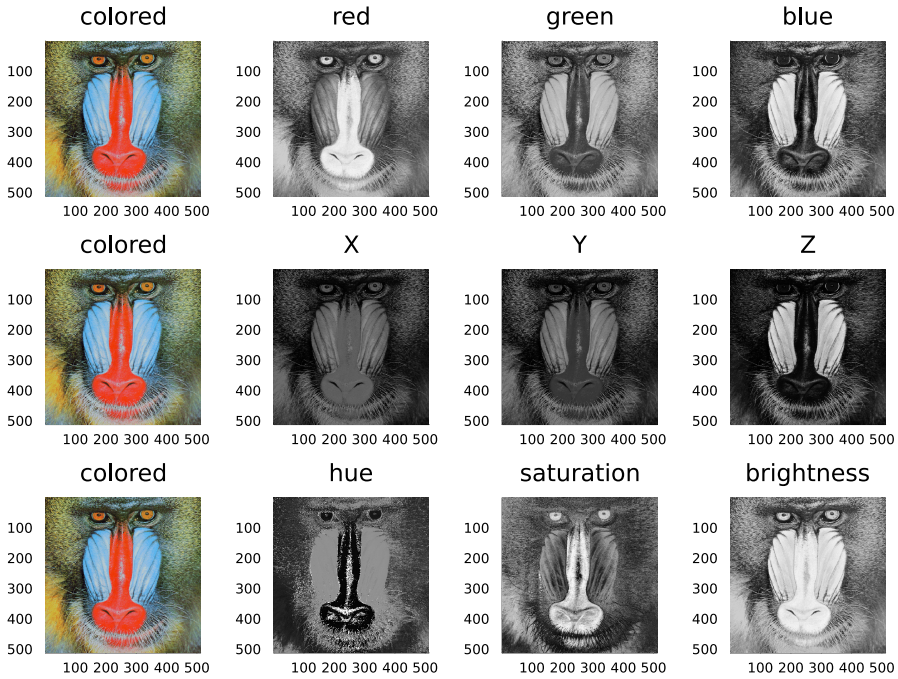
G = 0.0

B = 0.0

```
1 RGB(R_,G_,B_)
```

8.2.5 Example

We have now learned that colors can be represented in different color spaces. Let us apply this to the Lena image and look at the individual channels:



Note

You might wonder why we do not discuss CMY or CMYK, which are well known subtractive color spaces. The reason is that CMYK is only important for printers that are based on this color model (i.e. laser printers).

8.3 Color Processing

One important question is how all the image processing algorithms developed in the last lectures can be applied to colored images.

The answer is simple: In the majority of cases one can apply the algorithms channel-wise. In a well-designed image-processing framework, image values are treated as small vectors for which the operations

- multiplication with a scalar
- addition

are supported. If the image processing algorithm just needs these two operations, then the algorithm will work both with scalars and with colors.

Let us try this out:

```
RGB(0.8,0.0,0.0) + RGB(0.0,0.8,0.0)
```

```
3*RGB(0.2,0.0,0.0)
```

```
MethodError: no method matching *(::ColorTypes.RGB{Float64}, ::ColorTypes.RGB{Float64})
Math on colors is deliberately undefined in ColorTypes, but see the ColorVectorSpace package.
You may also need `⋅`, `⊙`, or `⊗`.
Closest candidates are:
*(::Any, ::Any, !Matched::Any, !Matched::Any...)
@ Base operators.jl:578
*(::Union{ColorTypes.TransparentColor{C, T}, C} where {T, C<:Union{ColorTypes.AbstractRGB{T}, ColorTypes.AbstractGray{T}}}, !Matched::Real)
@ ColorVectorSpace ~/.julia/packages/ColorVectorSpace/tLy1N/src/ColorVectorSpace.jl:247
*(!Matched::Real, ::Union{ColorTypes.TransparentColor{C, T}, C} where {T, C<:Union{ColorTypes.AbstractRGB{T}, ColorTypes.AbstractGray{T}}})
@ ColorVectorSpace ~/.julia/packages/ColorVectorSpace/tLy1N/src/ColorVectorSpace.jl:246
...

  1. top-level scope @ Local: 1 [inlined]
```

```
RGB(0.8,0.0,0.0) * RGB(0.0,0.8,0.0)
```

So you can see that only those operations are supported, which actually make sense.

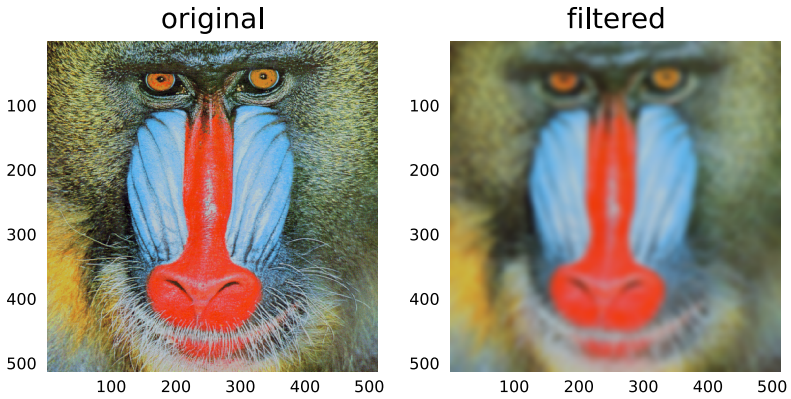
In some algorithms the amplitude of a color is required. In this case one can work channel-wise, or use the brightness value as the amplitude. Julia uses the first approach such that:

```
true
```

```
abs(RGB(0.8,0.0,0.0)) == RGB(0.8,0.0,0.0)
```

8.3.1 Example

Let us have a look at a simple Gaussian filter example:




```
let
  plot(
    heatmap(mandril, title="original"),
    heatmap( imfilter(mandril, Kernel.gaussian(5)), title="filtered" ),
    size = (600,300)
  )
end
```

As you can see, the code just works since a convolution requires only additions an scalar multiplications.

8.3.2 Color Data Types

Gray images are stored on disk using integer values ranging from 8 bit to 64 bit. When doing image processing they are usually converted to floating point numbers (32 bit or 64 bit).

When switching to colored images we need to store the three values RGB, which are most of the time put next to each other in main memory (or file memory).

```
RGB{N0f8}(0.0,0.749,1.0)
```

```
true
```

```
isbits(RGB{N0f8}(0.0,0.749,1.0))
```

```
3
```

```
sizeof(RGB{N0f8}(0.0,0.749,1.0))
```

Quite often, in addition to RGB an additional transparency channel is stored. This channel is also named the alpha channel and thus we store the four values as RGBA or ARGB. In the first case, the alpha channel is at the last position, in the second case, it is at the first position.

```
RGBA{N0f8}(0.0,0.749,1.0,0.5)
```

We here used the data type `N0f8` which is an 8 bit integer normalized to the range of [0,1]. In other programming languages this will simply be an 8 bit integer. The type `RGB{N0f8}` thus has 24 bits while `RGBA{N0f8}` has 32 bits.

Since it is beneficial for modern CPUs to have 32 bit alignment, the alpha channel is often used in non-transparent cases as well.

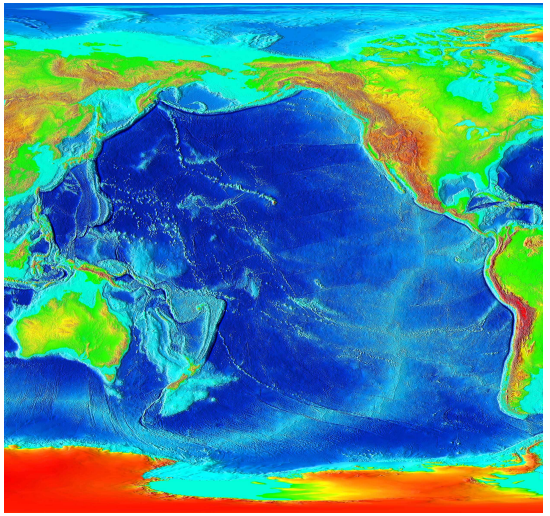
8.4 Pseudocolors

Images are not only used for displaying data that we can detect optically but the are also used to visualize general information. Some examples are:

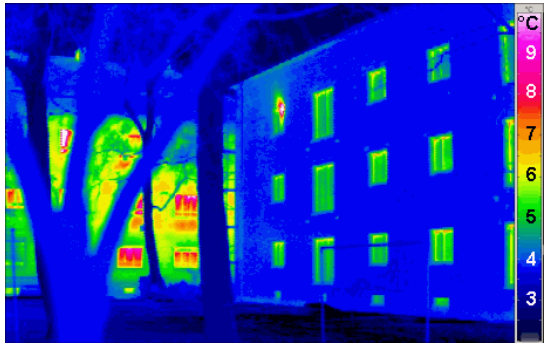
- height map of a landscape
- thermal imaging
- medical imaging

Here are some example images:

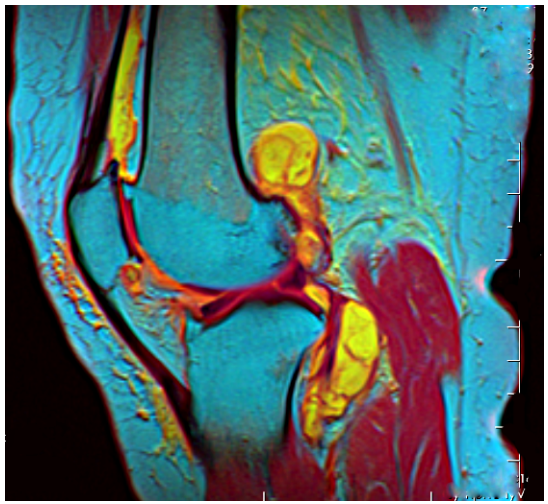
Height Map:



Thermal imaging:



Magnetic Resonance Image of a Knee:



```
LocalResource("img/Knee_MRI_113035_rgbcb.png",:width=>400)
```

We name such use of color false color or pseudocolor.

Why use color?

One might question why to use color at all in these applications. The reasons are:

- better perception
- encode multiple things simultaneously

Both are two different use cases and it is important to understand that color has to be used in an adequate fashion to reach these goals.

Note

Using color just to make an image *look nicer* is not an adequate use of color. In fact it can imply that the visual perception is actually degraded. We see an example of this later.

8.4.1 Color Mapping

Gray Colormap

A gray color c is represented in number form as an element of $[0, 1]$ where 0 is the color black and 1 is the color white. In-between all shades of gray are defined.

General Colormap

A general color c is represented as an RGB tuple $c = (r, g, b) \in [0, 1]^3$. A colormap $\kappa : [0, 1] \to [0, 1]^3$ maps an input value between 0 and 1 to an output color.

The colormap is defined on the domain of real numbers in the interval $[0, 1]$. In practice colormaps are build using a set of discrete colors. Using linear interpolation it is possible to define a continuous function based on the discrete values.

Let $c_k \in [0, 1]^3$ for $k = 1, \ldots, K$ be K colors. Then we can define

$$\kappa(\alpha) = \begin{cases} c_\beta & \text{if } \beta \text{ is an integer} \\ (1 - w)c_{\lfloor\beta\rfloor} + wc_{\lfloor\beta\rfloor+1} & \text{otherwise} \end{cases}$$

to be the linearly interpolated colormap with $\beta = \alpha(K - 1) + 1$, which is α scaled to $[1, K]$, and weighting $w = \beta - \lfloor\beta\rfloor$.

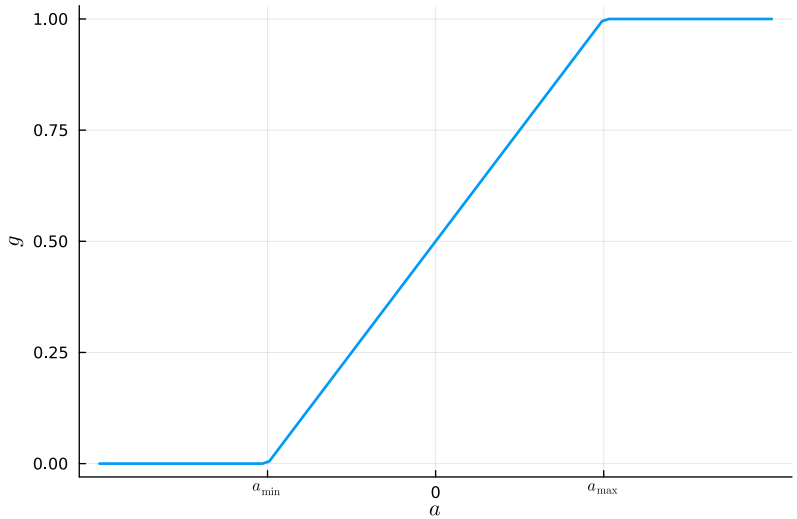
What do we need for Color Mapping?

- a colormap $\kappa(\alpha)$
- a minimal value $a_{\min}$ that maps to the darkest color $\kappa(0)$
- a maximal value $a_{\max}$ that maps to the brightest color $\kappa(1)$

Windowing

The mapping between a real valued quantity a and the input to the colormap α is called windowing and can be expressed by a function

$$g(a) = \begin{cases} 0, & \text{for } a \le a_{\min} \\ \frac{a - a_{\min}}{a_{\max} - a_{\min}}, & \text{for } a_{\min} < a < a_{\max} \\ 1, & \text{for } a \ge a_{\max} \end{cases}$$



Instead of $a_{\min}$ and $a_{\max}$ it is also common to consider

- $\mathrm{WW} = a_{\max} - a_{\min}$ (Window Width or Contrast)
- $\mathrm{WL} = (a_{\max} + a_{\min})/2$ (Window Level or Brightness)

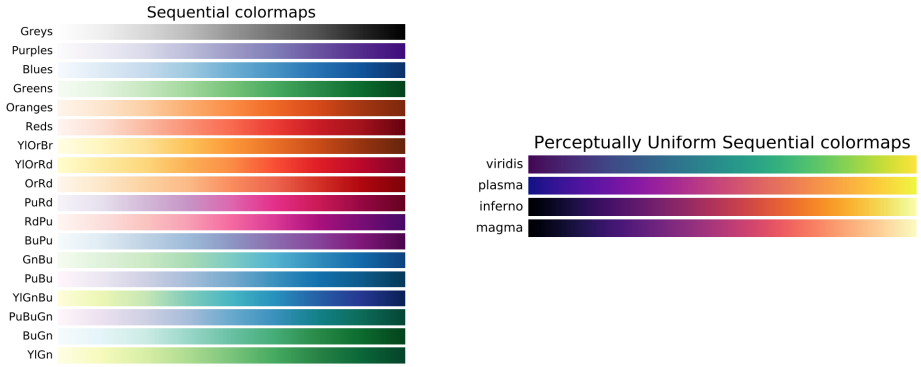
Lets combine everything

To colorize a function $f(x, y)$ we apply for each position x,y:

$$f_{\text{colorized}}(x, y) = \kappa(g(f(x, y)))$$

8.4.2 Colormaps

The following shows some colormaps that have been developed for the 2D plotting library Matplotlib:

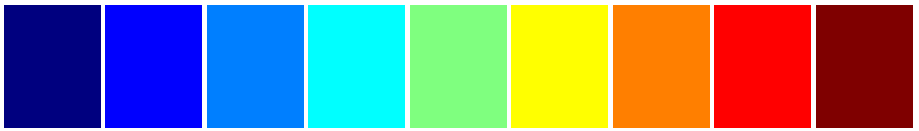


One should take the following into account when using a colormap:

- What do I actually want to encode in the color?
- Small value changes should result in small color changes.
- What happens if you convert the colormap to grayscale? Is it still sequential (monotonic)?
- What about color blindness

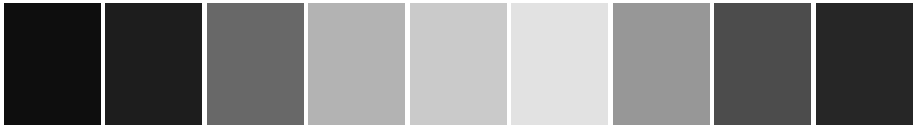
Abuse of Color

An example where these guidelines are not taken into account is the jet colormap, which looks like this:

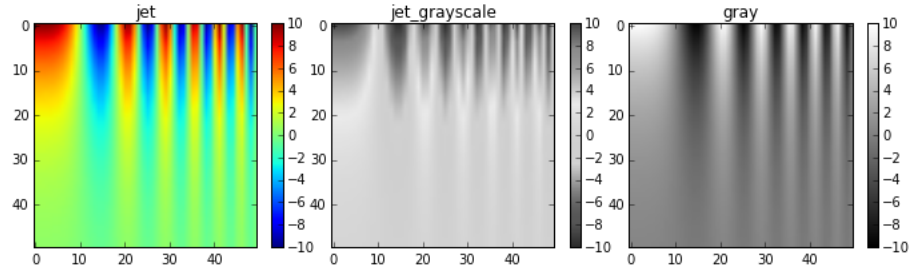


Obviously, the idea of jet is to use as many colors as possible. It is basically a sampling of the light spectrum (i.e. a rainbow pattern).

But lets convert this to gray:



The issue now is that blue and red are mapped to similar gray values and cannot be distinguished. What this implies can be seen in the following example where a wave-like function is illustrated:



What you can see is that the yellow regions are highlighted in the jet image while there is actually no reason for that when looking at the gray colorized image at the right. Consequently, a false impression is generated.

Note

In practice the best is to just use a perceptually uniform sequential colormap such as viridis. They have been designed by color experts and are a good default choice in most cases.

8.4.3 Complex Coloring

As a more advanced topic we consider how to visualize complex numbers. We already know the following ways:

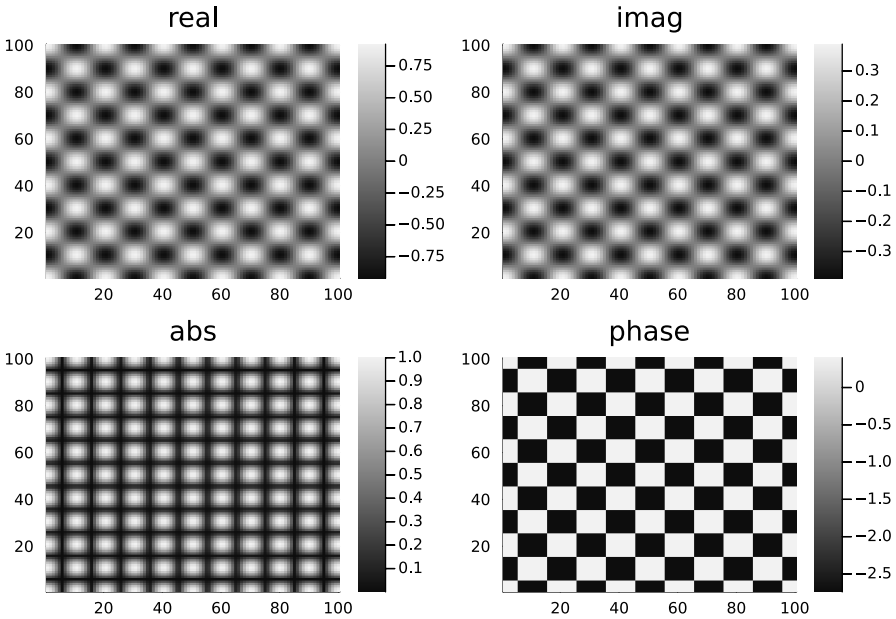
- display real and imaginary part separately
- display amplitude and phase separately

Both have their pros and cons depending on what you want to focus on. One issue of separate images is, however, that you cannot associate directly the different pixel informations.

Let us look at the following function

$$f(x, y) = \cos(2\pi 5x)\cos(2\pi 5y)\ \exp(0.5\mathrm{i})$$

which is one cosine transform basis function multiplied by a phase to make it complex (for illustration purpose):

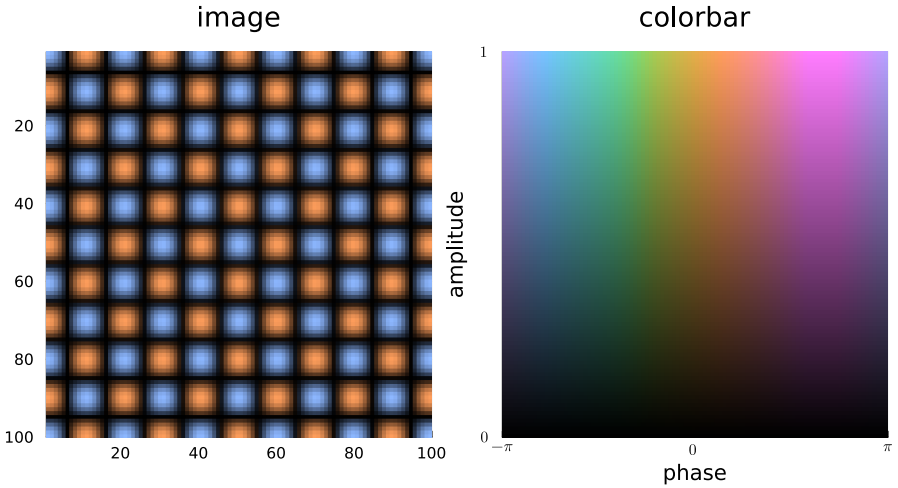


Observations

- real/imag and abs/phase carry the same information but are differently perceived by the viewer.
- Whether the waves have maxima or minima is clear in the real/imag image but not in the abs image. One needs to additionally take the phase information into account.

In order to get the best of both worlds it is possible to combine the abs and the phase image into a single image by exploiting colors.

The idea is to encode the absolute value in the brightness and encode the phase as a color:



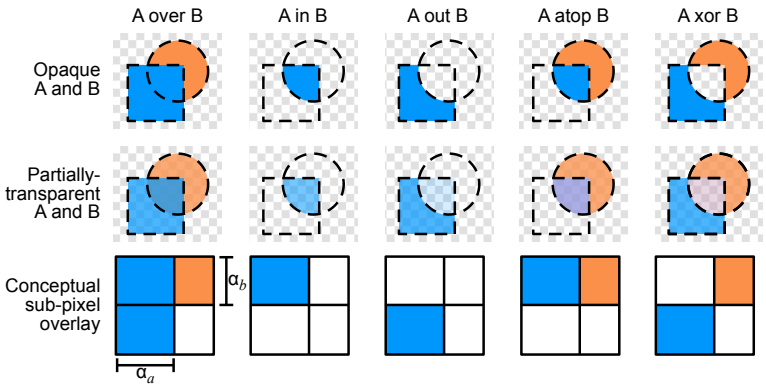
That is much nicer. The only downside is that it makes the colorbar a little bit more complex. In particular, the colormap is now 2D instead of 1D. In this notebook we build the colorbar on our own since Plots.jl has no build-in support for 2D colormaps.

> **Note**
>
> The shown colormap is periodic since the phase is also periodic.

8.5 Alpha Blending

In computer graphics, alpha compositing or alpha blending is the process of combining one image with a background to create the appearance of partial or full transparency. It is often useful to render pixels in separate passes or layers and then combine the resulting 2D images into a single, final image called the composite.

Given two image $\boldsymbol{A}$ and $\boldsymbol{B}$ there are various ways to combine the two images:

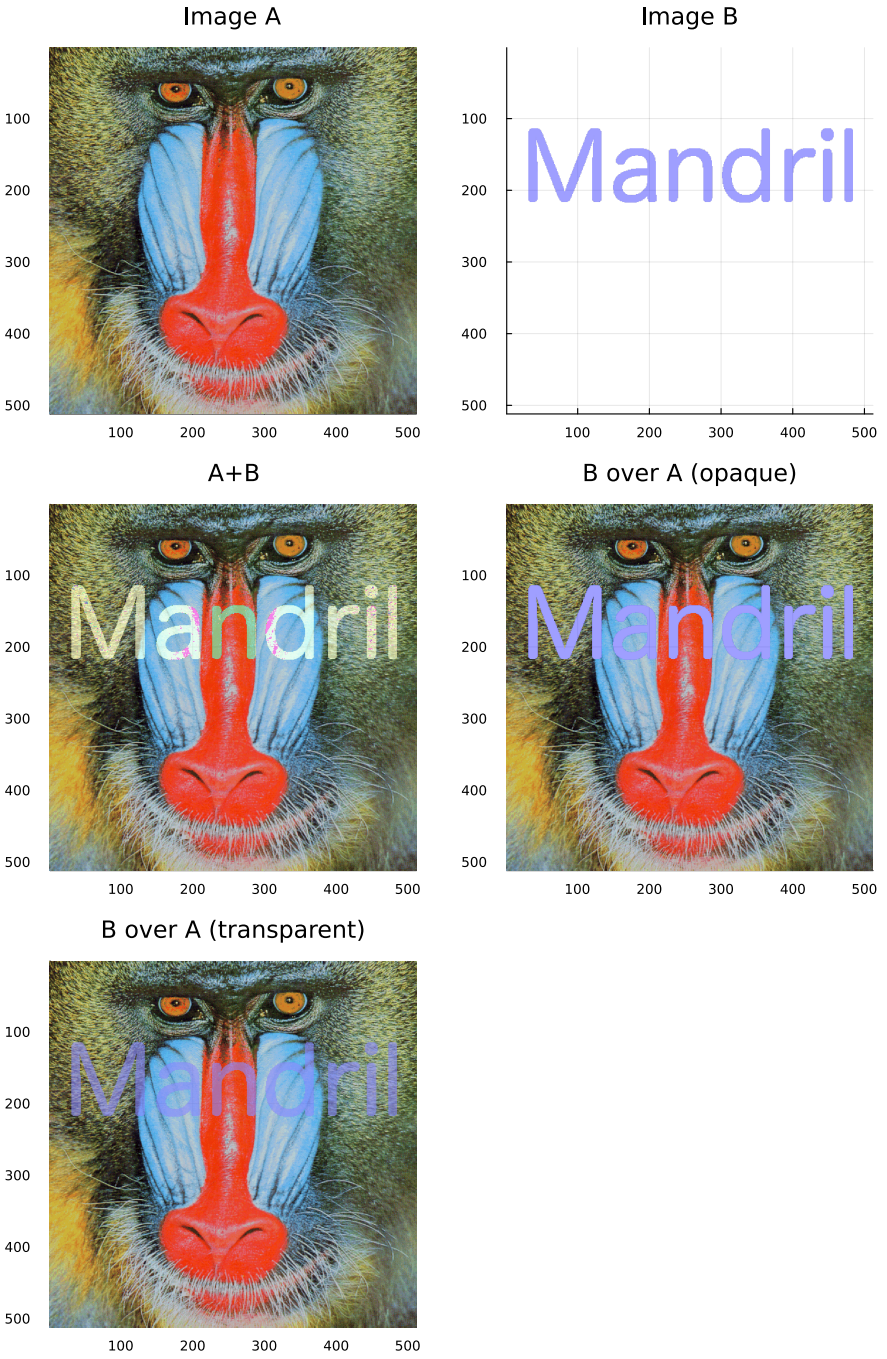


We are here primarily discussing $\boldsymbol{A}$ over $\boldsymbol{B}$, which is the most important one.

As you can see in the image there are two ways to do the blending:

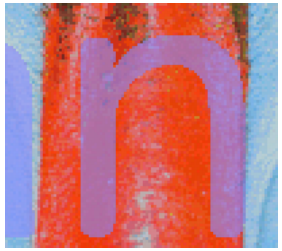
- opaque: this means that the color of the object laying on top is shown.
- partially-transparent: this means that the colors are mixed in the overlaying region.

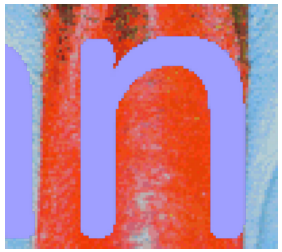
Let us apply both in a typical scenario where you want to overlay some text on an image:



You can see that the simple addition is wrong and leads to an integer overflow that simply wraps to a wrong color.

Both over operations accomplish the goal of generating a joint image. In the transparent version you can still look through the text. To see this better we can zoom in a little bit:





Implementation

The opaque over operation is implement like this

$$C_o = \begin{cases} C_a & \text{if } \alpha_a = 1 \\ C_b & \text{else} \end{cases}$$

Here C_o, C_a and C_b stand for the color components of the pixels in the output image, image A and image B respectively, applied to each color channel (red/green/blue) individually. α_a is the alpha value of the image A, which controls, whether the pixel of image A or B is shown.

The tranparent over operator involves a convex combination of the colors:

$$\alpha_o = \alpha_a + \alpha_b(1 - \alpha_a)$$
$$C_o = \frac{C_a\alpha_a + C_b\alpha_b(1 - \alpha_a)}{\alpha_o}$$

whereas $\alpha_o$and α_b are the alpha values of the output image and image B pixels.

8.6 Wrapup

In this lecture you learned:

- what color is.
- that there are different types of color images (true color / false color).
- how to perform image processing on colored images.
- how to colorize images.
- how to do alpha blending.