Farbreich mit Perl

Herbert „Lichtkind“ Breunung

Graphics::Toolkit::Color

Graphics::Toolkit::Color

- Irgendwas mit Farbe ?

Graphics::Toolkit::Color

- Irgendwas mit Farbe ?
- Im Bereich Computergraphik

Graphics::Toolkit::Color

- Irgendwas mit Farbe ?
- Im Bereich Computergraphik
- Werkzeugkasten:
  - Viele ineinander greifende Funktionen
  - nützliche Konstanten

Graphics::Toolkit::Color

- Was ist Farbe ?

Was ist Farbe

- Physik (Wellen)
- Biologie (Auge)
- Chemie (Rezeptoren)
- Gehirn (Neurologie)
- Kultur (Software)
- Technik

Warum so viele Farbräume

- Physik (Wellen)
- Biologie (Auge)
- Chemie (Rezeptoren)
- Gehirn (Neurologie)
- Kultur (Software)
- Technik

Vortrag

- Was ist Farbe ? →
- Warum Farbräume ? →
- Was machen Übersetzer ? →
- GTC API (mehr und warum) →
- GTC Aufbau (Abstraktinen, OO)

Vortrag

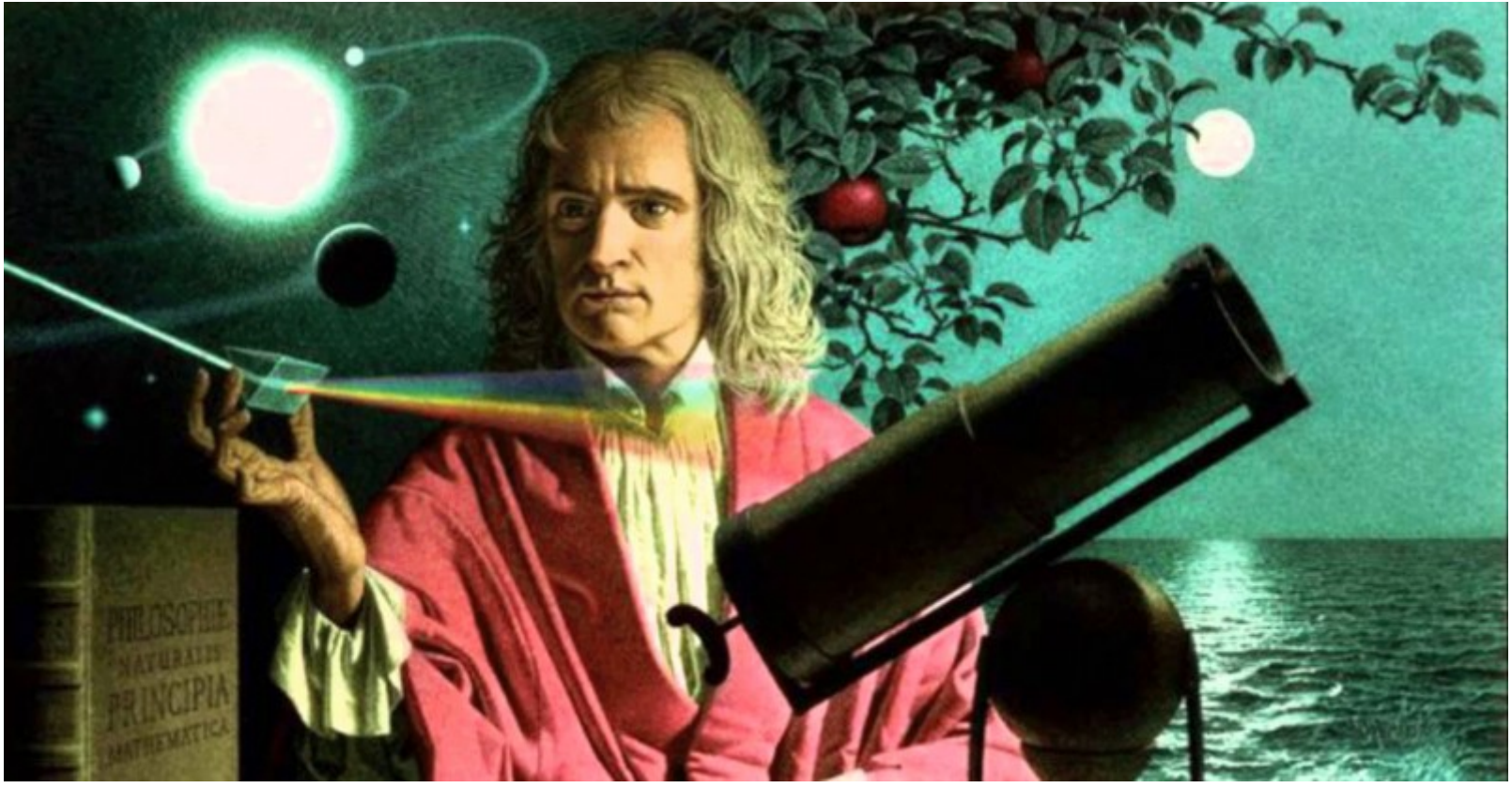
- 1. Was ist Farbe ?
  - Wie gehen Menschen damit um?
- 2. API, features
  - Was kann ich mit GTC tun?
- 3. Softwaredesign
  - GTC interna

Teil 1 - Farben

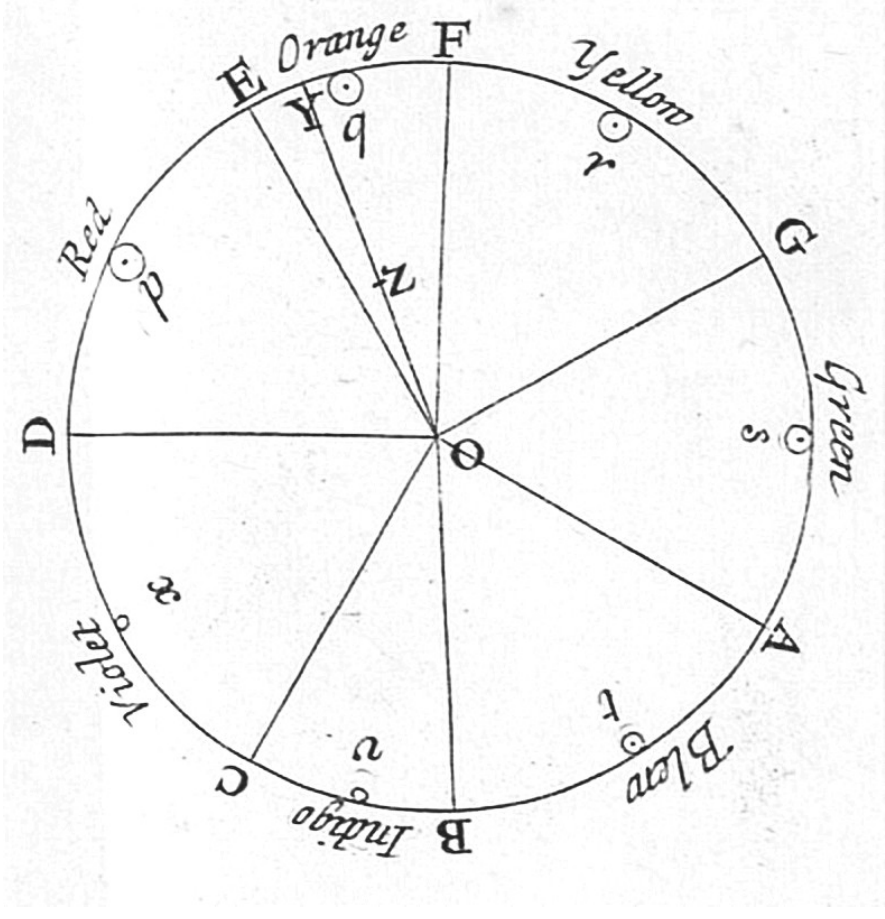
ein paar schöne Details

kein Vorlesen der Doku

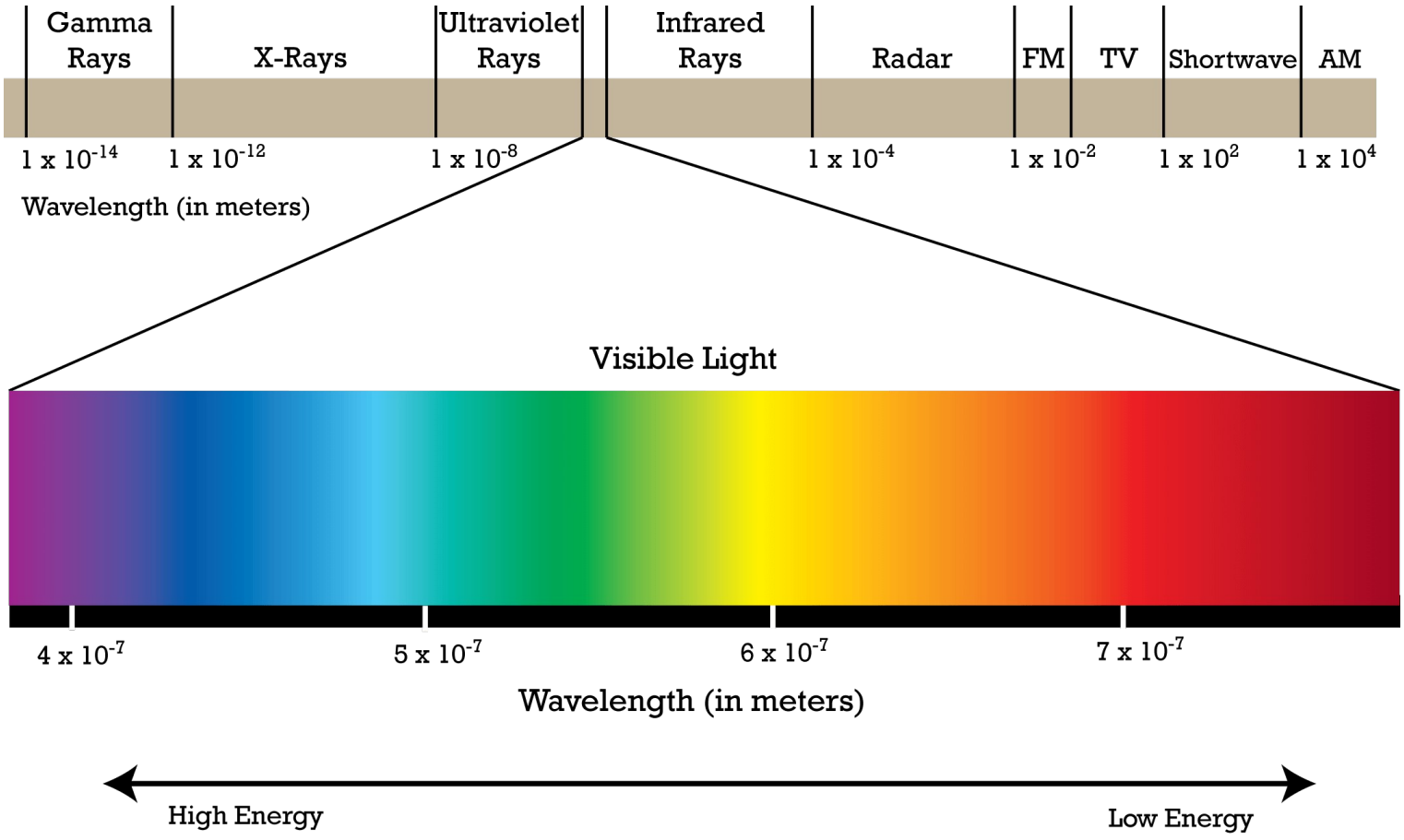
Isaac Newton



Farbkreis, Opticks 1704 → QED

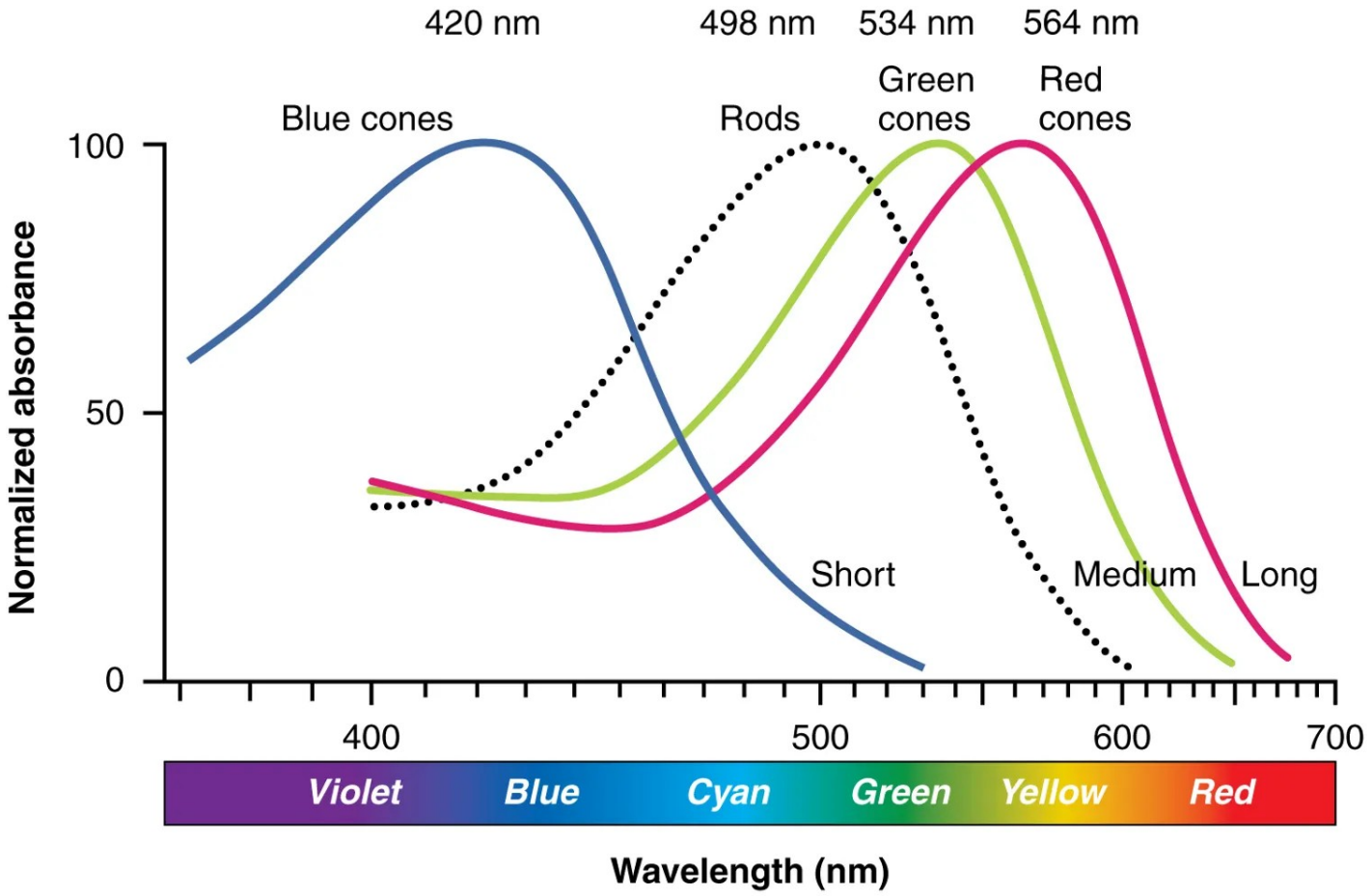


Elektromegnetisches Spektrum

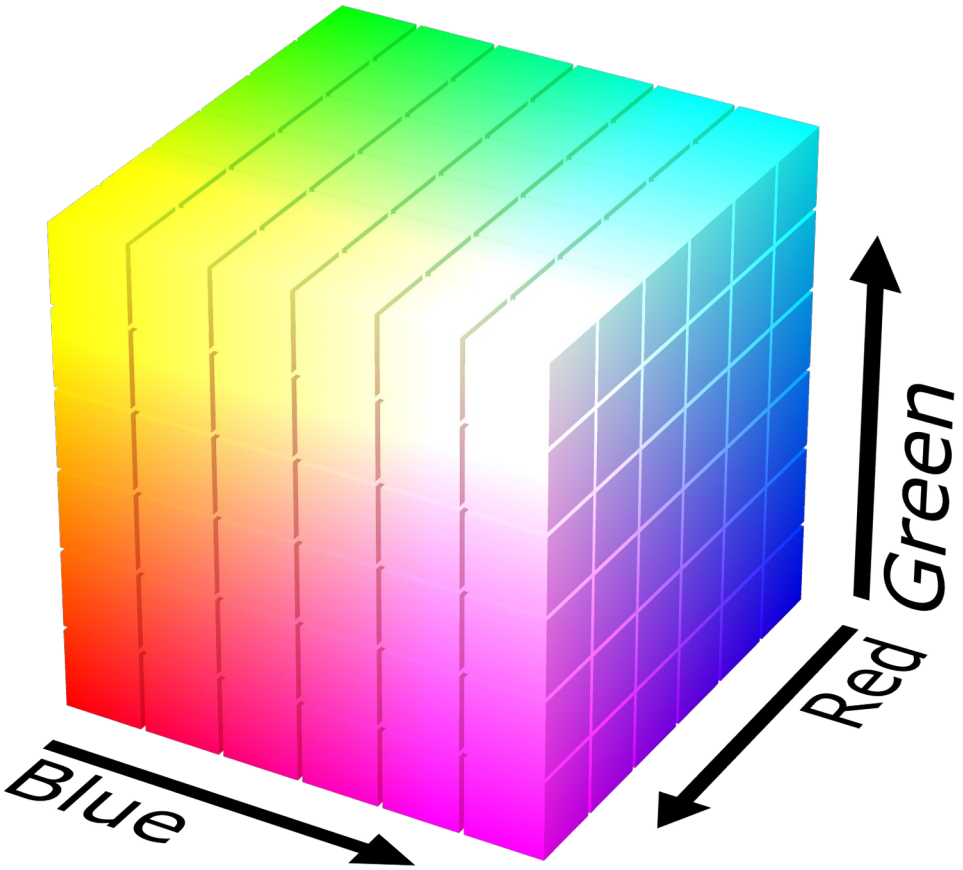


Hermann von Helmholtz

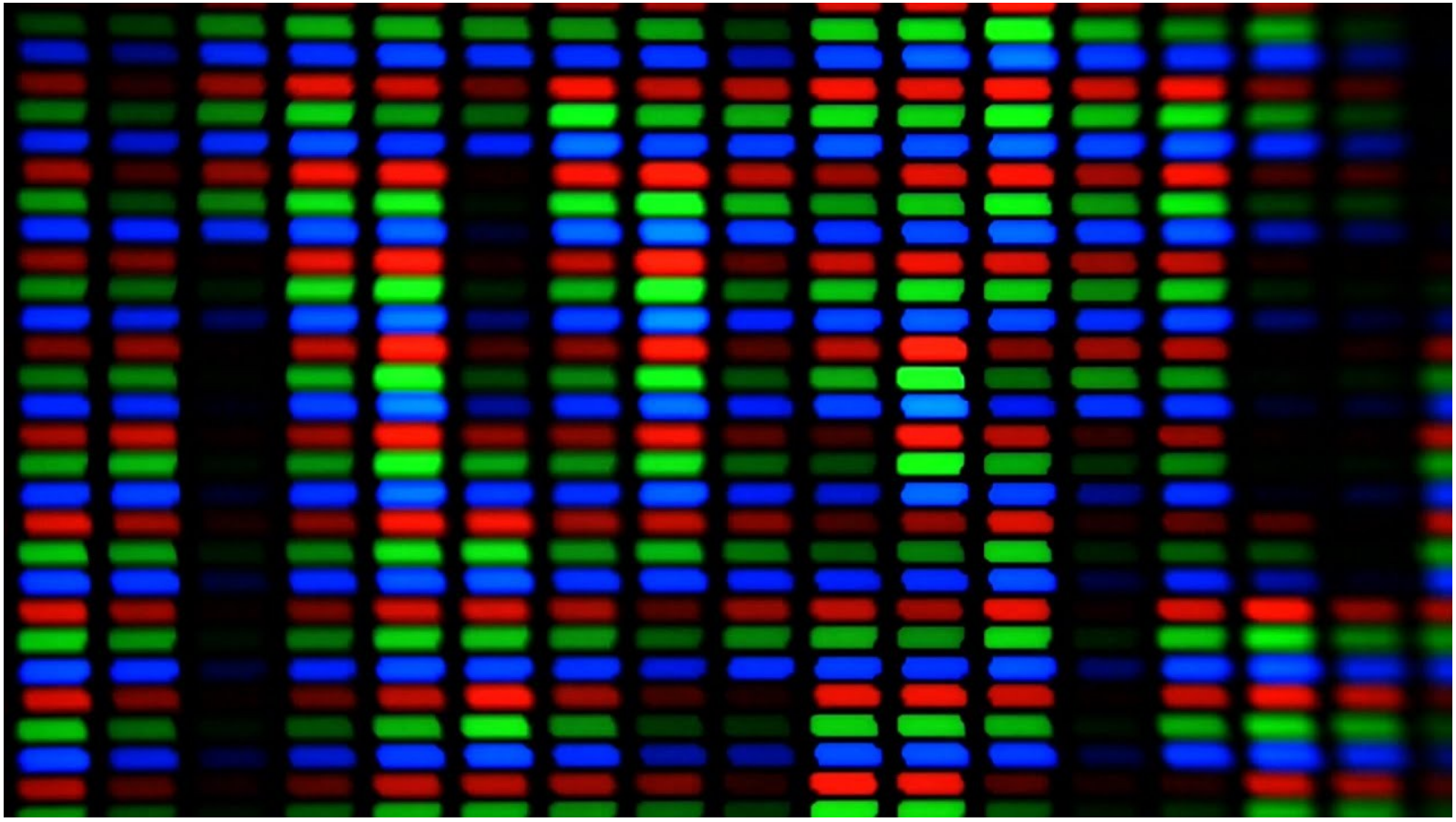
Spektrum der RGB Rezeptoren



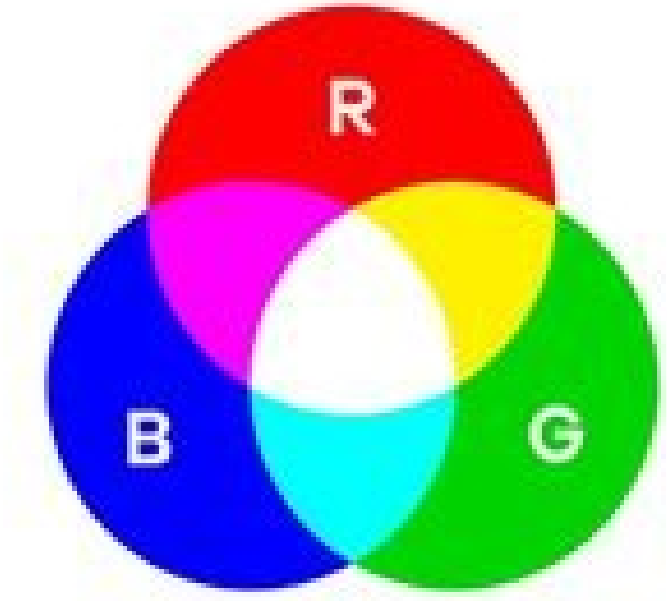
RGB Farbraum



Partitive Farbmischung

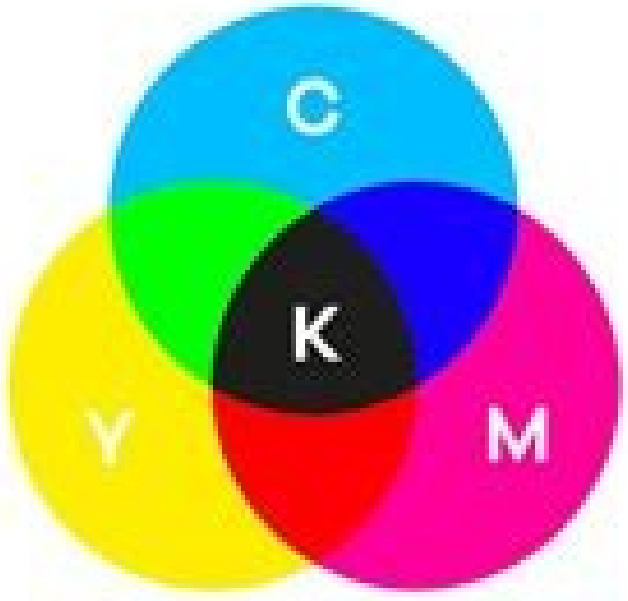


additive Lichter, subtraktiv gedruckt



RGB

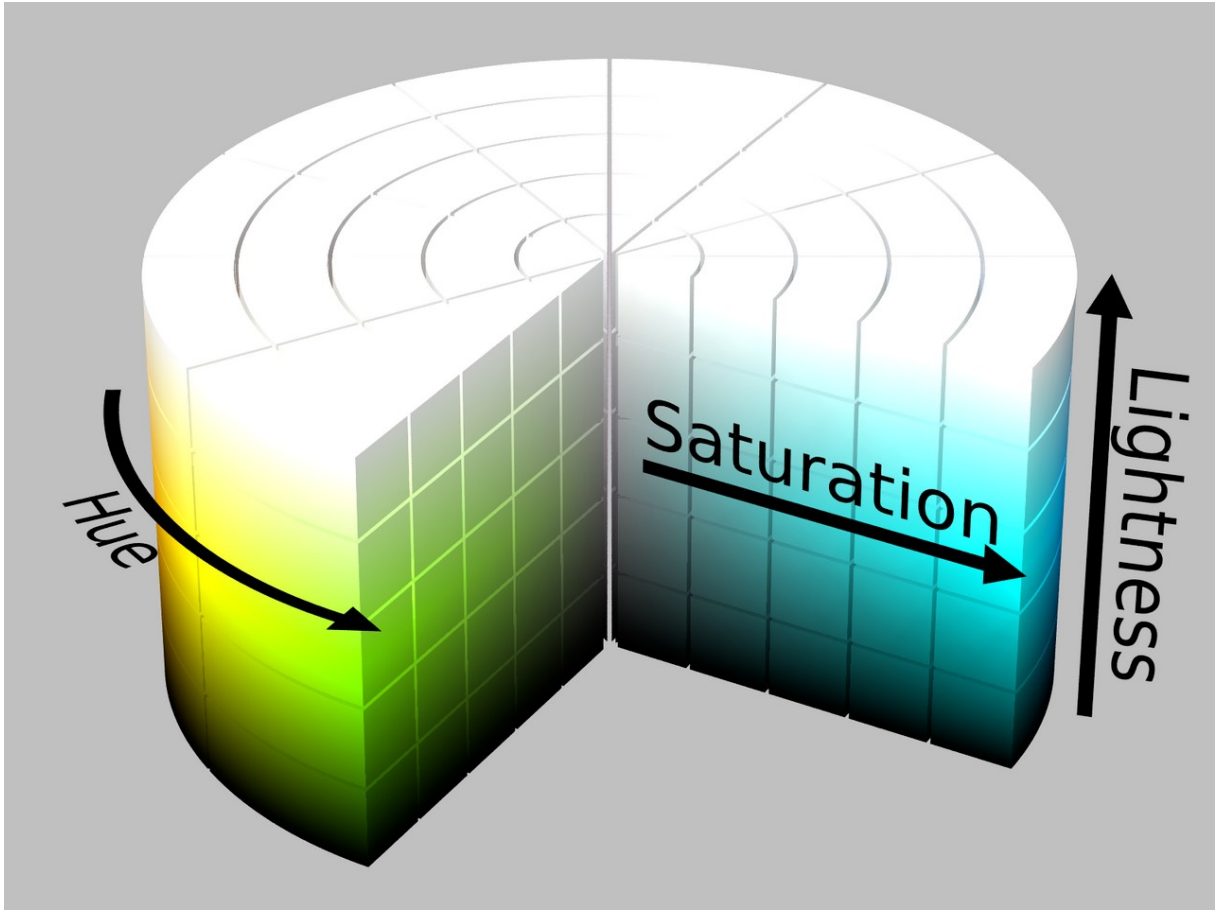
Additive Colour Mixing



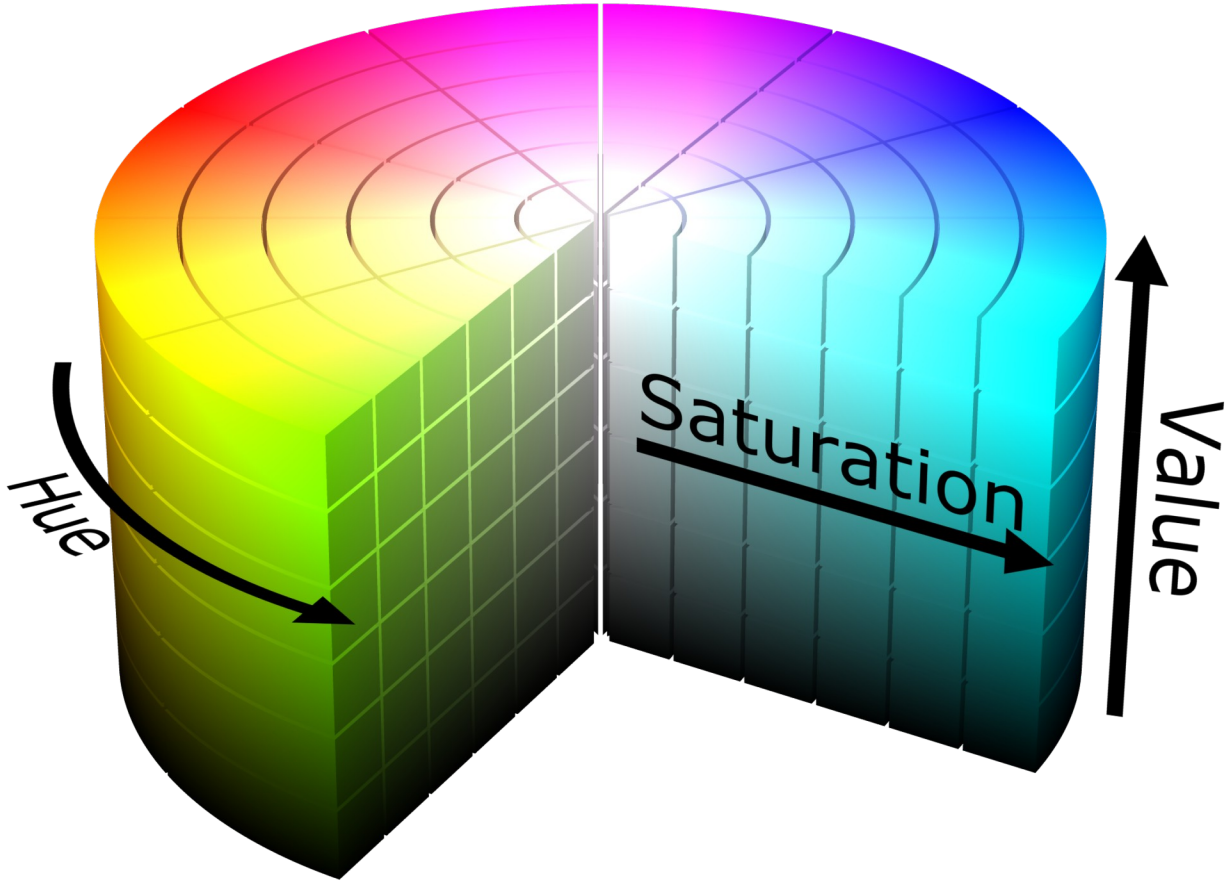
CMYK

Subtractive Colour Mixing

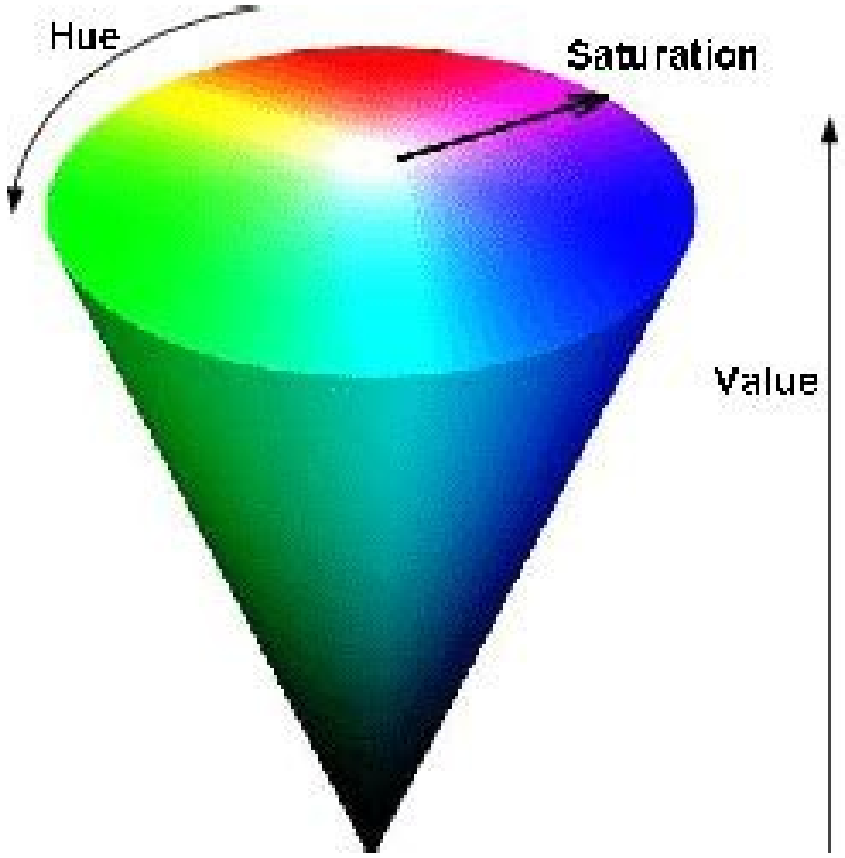
Zylindrischer HSL Farbraum



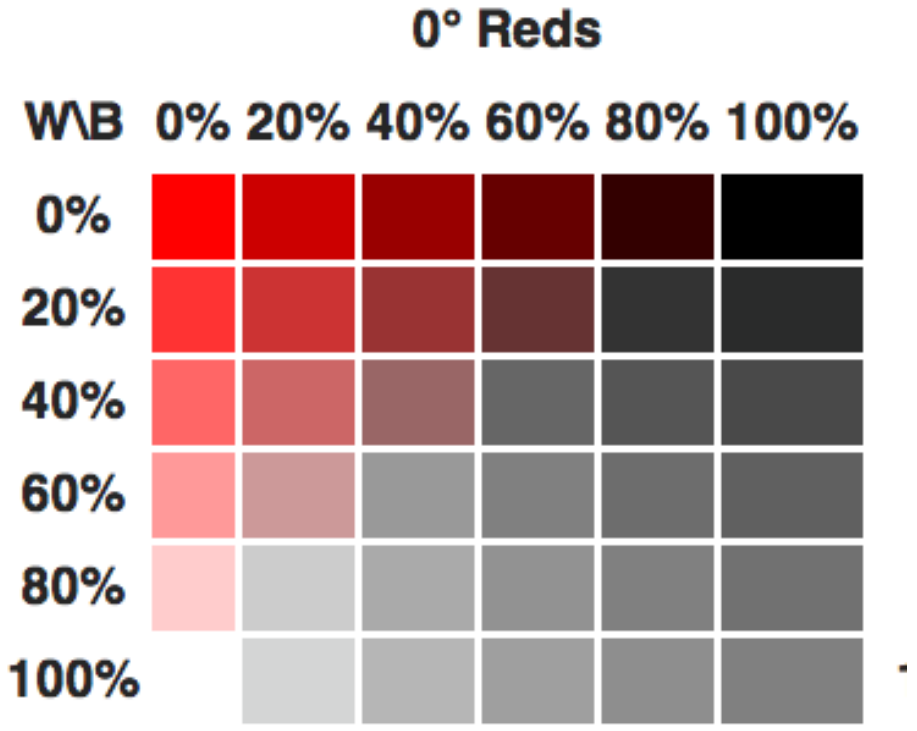
Zylindrischer HSV Farbraum



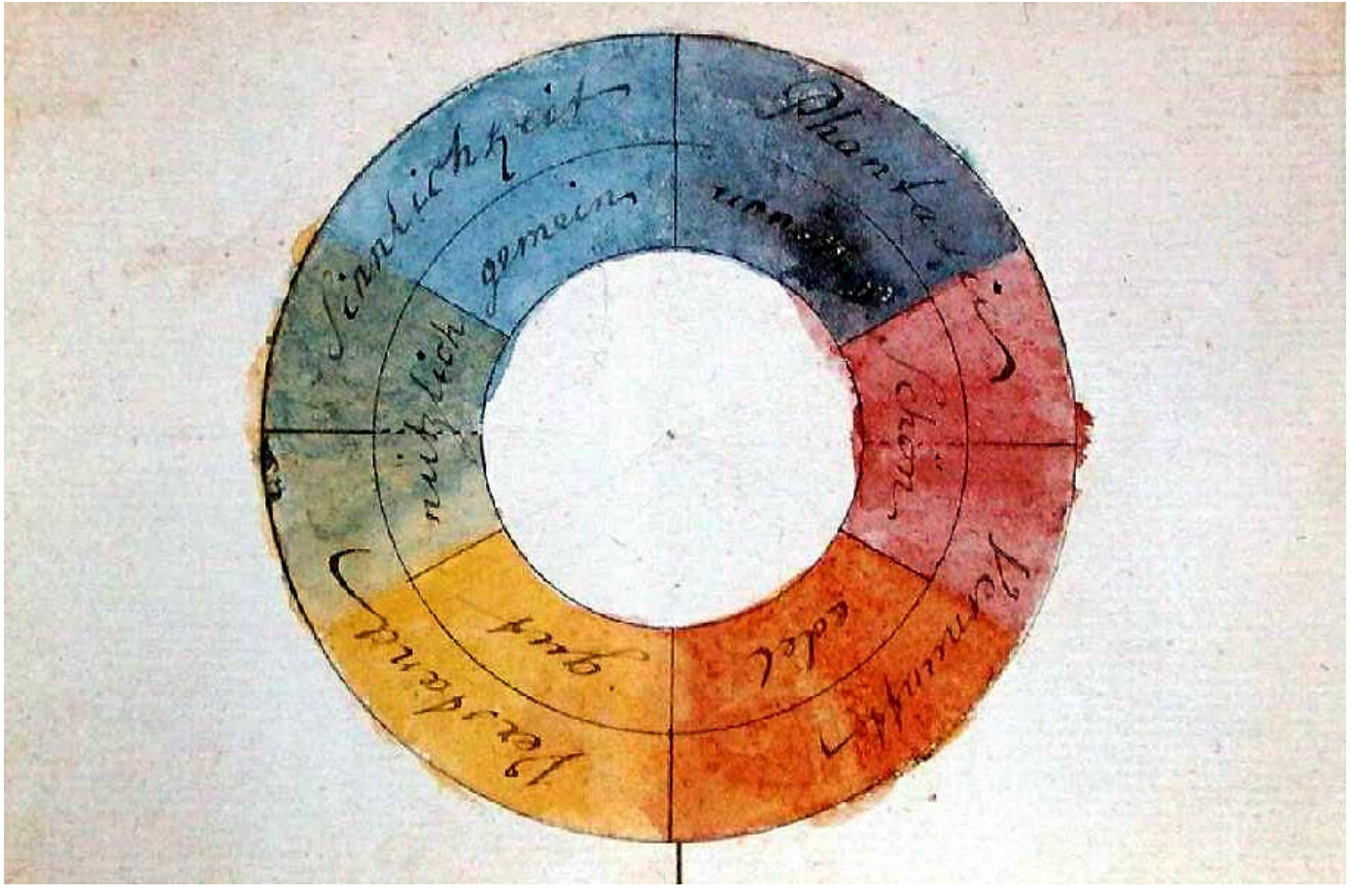
Kegelförmiger HSV Farbraum



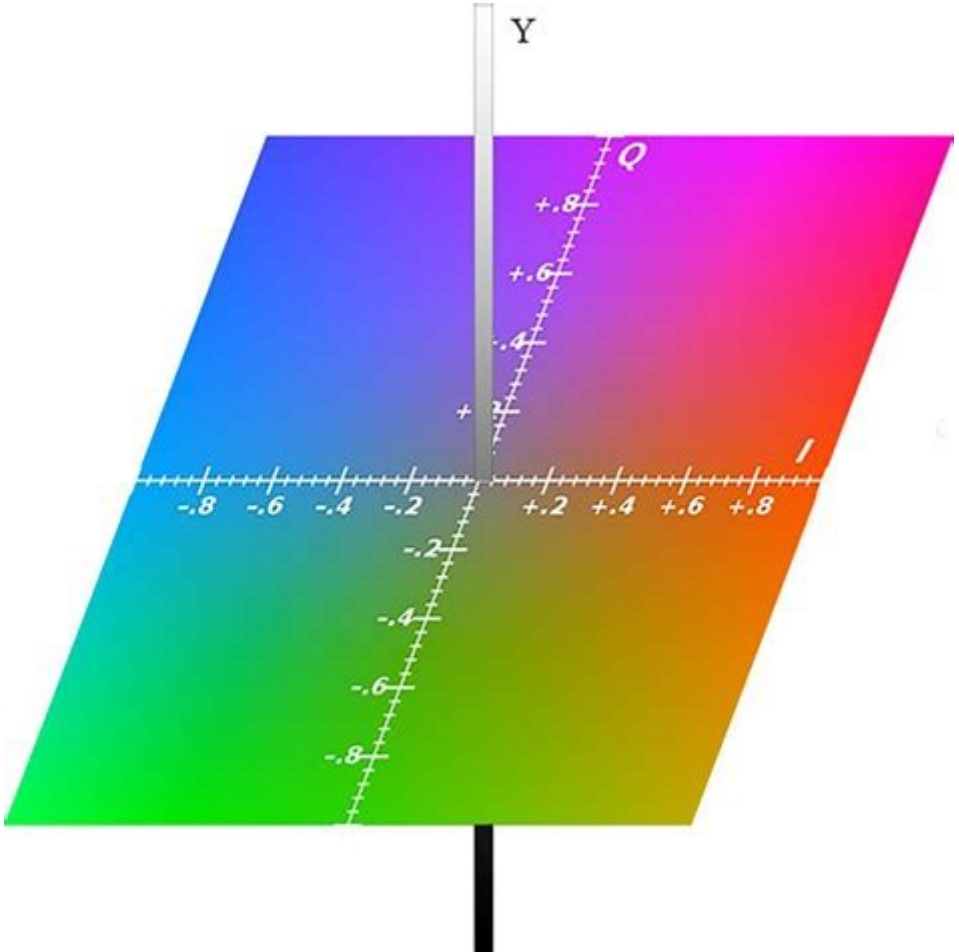
Zylindrischer HWB Farbraum



Ncol Hue Werte RYGCBM + %



Kubischer YIQ Farbraum



Das Gehirn regelt nach

grün mit rot angestrahlt bleibt grün

(Edwin Lang mondrian experiment)

Sonnenlicht ist farblos aus gewöhnung

510 nm grünliches Gelb - Helligkeitsmaximum

stein auf mond: blaugrau bis grünlich

dunkelgrau auf erde

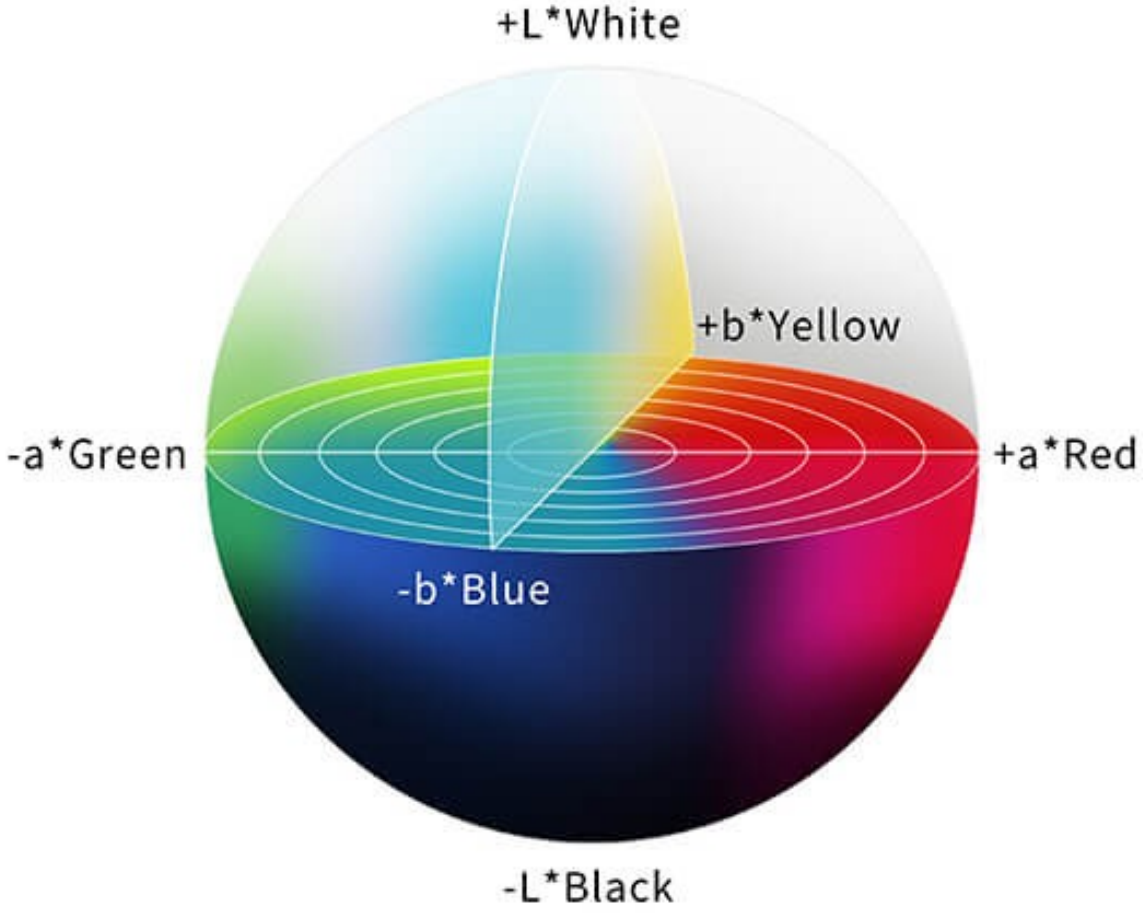
Das Gehirn regelt nach

1. Die Helligkeit

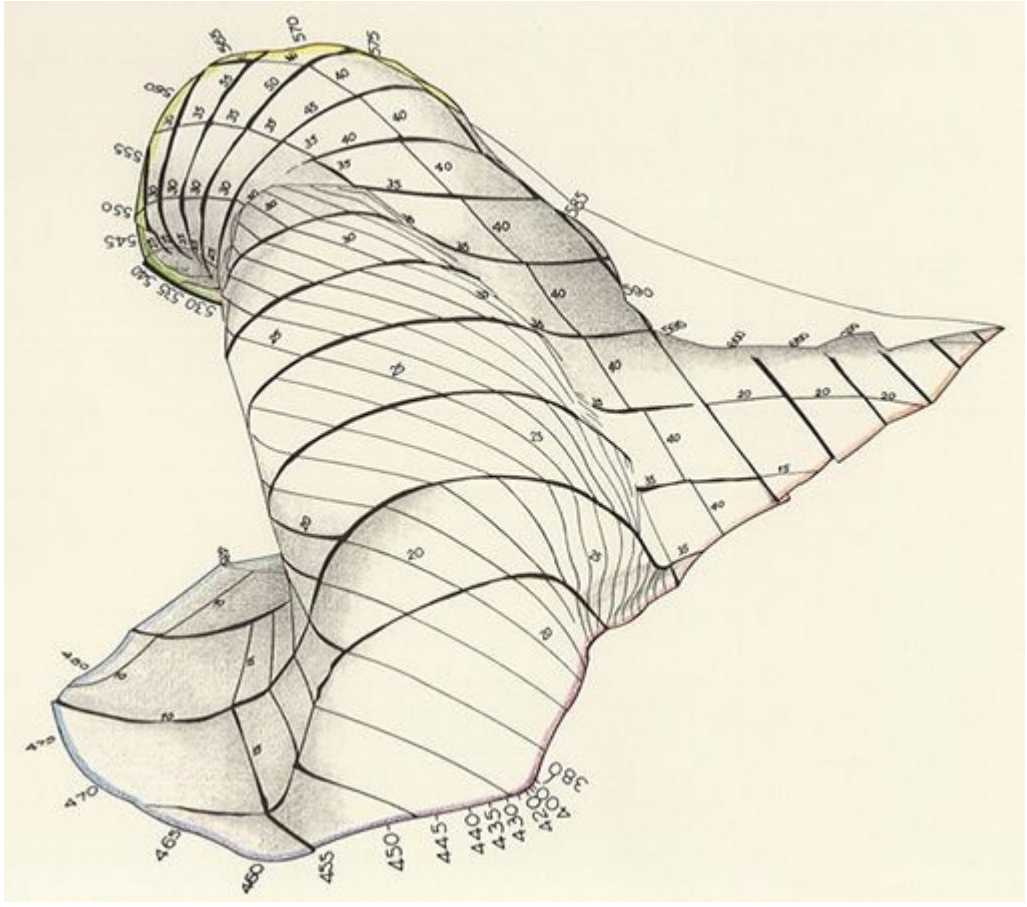
2. RGB-Werte um buntes Licht auszugleichen

3. RGB-Werte um Eindruck der Umgebungsfarbe auszugleichen

Kugelförmiger Lab Farbraum



Uniform Chromatic Scale D.L.M.



Teil 2 - API

kein Vorlesen der Doku

Sinn hinter der Sache

ein paar schöne Details

Anwendungsfälle

Code !

Zweck von GTC

algorithmisches (Laufzeit!)

finden und analysieren

von Farben und Farbmengen

Philosophie

- Werkzeugkasten Farbe (~ Perl-Philosophie)
- zuverlässig, mächtig
- kombinierbare Befehle (Summe > Teile)
  - ! ∀ Details haben Methode (rand map grep)
- kleine API (wenig zu merken)
- deklarativ (sprechend, keine Kürzel)

Umsetzung der Philosophie

- Werkzeugkasten Farbe (~ Perl-Philosophie)
- zuverlässig, mächtig
- kombinierbare Befehle (Summe > Teile)
  - ! ∀ Details haben Methode (rand map grep)
- kleine API (wenig zu merken)
- deklarativ (sprechend, keine Kürzel)

Zuverlässig

- große Testsuite

Zuverlässig

- große Testsuite
- Bug Tracker wird beachtet – ist leer

Zuverlässig

- große Testsuite
- Bug Tracker wird beachtet – ist leer
- sauberer, übersichtlicher Code

Zuverlässig

- große Testsuite
- Bug Tracker wird beachtet – ist leer
- sauberer, übersichtlicher Code
- ausführliche Dokumentation

Zuverlässig

- große Testsuite
- Bug Tracker wird beachtet – ist leer
- sauberer, übersichtlicher Code
- ausführliche Dokumentation
- Verwendung in Projekten (*graph, Chart)

Zuverlässig

- große Testsuite
- Bug Tracker wird beachtet – ist leer
- sauberer, übersichtlicher Code
- ausführliche Dokumentation
- Verwendung in Projekten (*graph, Chart)
- keine Abhängikeiten (Carp Exporter)

Zuverlässig

- Tests, keine Bugs
- sauberer, übersichtlicher Code
- ausführliche Dokumentation
- Verwendung in Projekten (*graph, Chart)
- keine Abhängikeiten (Carp Exporter)
- Opt. Abhängigkeiten (Graphics::ColorNames::*)

Zuverlässig

- Tests, keine Bugs
- sauberer Code, viel Dokumentation
- Verwendung in Projekten (*graph, Chart)
- keine Abhängikeiten (Carp Exporter)
- opt. Abhängigkeiten (Graphics::ColorNames::*)
- read only Objekte

Zuverlässig

- Tests, keine Bugs
- sauberer Code, viel Dokumentation
- Verwendung in Projekten (*graph, Chart)
- keine Abhängikeiten (Carp Exporter)
- opt. Abhängigkeiten (Graphics::ColorNames::*)
- read only Objekte, serialisierbar (2 str & back)

GTC ist Mächtig

- konvertiert zwischen 12 Farbräumen
- erkennt und konvertiert zu Farbnamen
- misst Farbunterschiede
- erzeugt verwandte Einzelfarben
- erzeugt Farbpaletten
- Leistet kein anderes Modul im CPAN

Alternative Module:

• Color::Similarity	Mattia Barbon	2007
• Convert::Color	Paul Evans	2023
• Graphics::ColorObject	Alex Izvorski	2005
• Graphics::ColorUtils	Philipp K. Janert	2007
• Color::Calc	Claus Färber	2014
• Color::Fade	Noah Petherbridge	2008
• Color::Scheme	Ricardo Signes	2023

Unterschiede:

- Color::Similarity RGB, Lab Distanzen + 00% - 90%
- Convert::Color viele Plugins + 00% - 75%
- Graphics::ColorObject bester Konverter + 15% - 55%
- Graphics::ColorUtils allg. --Räume + 00% - 45%
- Color::Calc allgemein einfacher + 05% - 50%
- Color::Fade nur Verläufe + 00% - 85%
- Color::Scheme anderer Focus + 20% - 60%

Alternatives:

- Best (for specialized schemes):

  Color::Scheme Ricardo Signes 2023

- Worst (can animate GIF):

  Color FigAnim 0.1 2004

Alternatives:

- Color::Similarity RGB, Lab Distanzen + 00% - 90%
- Convert::Color viele Plugins + 00% - 75%
- Graphics::ColorObject bester Konverter + 15% - 55%
- Graphics::ColorUtils allg. --Räume + 00% - 45%
- Color::Calc allgemein einfacher + 05% - 50%
- Color::Fade nur Verläufe + 00% - 85%
- Color::Scheme anderer Focus + 20% - 60%

Kleine API

- 10 Methoden
  - verständlich
  - Doku navigierbar
- 11 Argumentnamen
  - werden wieder verwendet wenn möglich
  - machen API stabiler

deklarative API

- 10 Methoden:
  - Werte: new, name, values, distance
  - Farben: set, add, blend
  - Mengen: gradient, complement, ball
- 11 Argumente: in, as, to(with), steps
  range, select, pos, dynamic + 3

deklarative API

- 10 Methoden:
  - new, name, values, distance, set, add, blend
  - gradient, complement, ball
- 11 Argumente: range, select, pos, dynamic + 3
  - in (Farbraum); as (Format)
  - to(with) (Zielfarbe); steps (Mengengröße)

einfache API

- Argumentnamen:
  - 11 (4 + 4 + 3) [+2]
  - 37 Achsennamen (16 verschiedene)

Konstruktor für GTC Objekt

```
use Graphics::Toolkit::Color qw/color/;
$blue = Graphics::Toolkit::Color->new ( ‘blue‘ );
$color = color ( ... );
```

Konstruktoreingabe

Farbname: ‘blue‘, ‘SVG:blue‘

String(CSS): 'HSL: 240,100, 100' 'hsl(240, 100, 100)'

ARRAY: [HSL => 240, 100, 100]

char_hash, HASH: { h => 240, s => 100, l => 100 }

{ hue => 240, saturation =>100, lightness => 100 }

einfache liste und array für RGB Werte

Getter, Konverter

```
$color->name( );          # name als string
$color->values( );        # RGB Werte als Liste
$color->values( 'HSL' );  # HSL Werte als Liste
$color->values( in => 'HSL', as => 'HASH' )->{hue};
$color->values( in => 'RGB', range => 2**16 );
# HSL Werte als HASH ; RGB16 Werte als Liste
```

einfache API

- alle Formate, Farbräume, Namen
- erhältlich als Ein- und Ausgabe
- in allen Methoden - (Argumenten)
- nie Teil der Methodennamen
- Im Zweifel: RGB

Abstand

```
$c->distance( $color2 );    # real number in RGB
$c->distance( to => 'green', in => 'HSL' );
$c->distance( to => 'green', in => 'HSL',
                    select => 'hue', range => 'normal' );
# deklarativer Syntax
# gleiche Argumentnamen wie zuvor (in range to)
```

Verwandte Farbe

```
$color->set ( saturation => 80 );    # absolut
$color->add( lightness => -10 );    # relativ
```

Achsennamen bekannter Räume

Verwandte Farbe

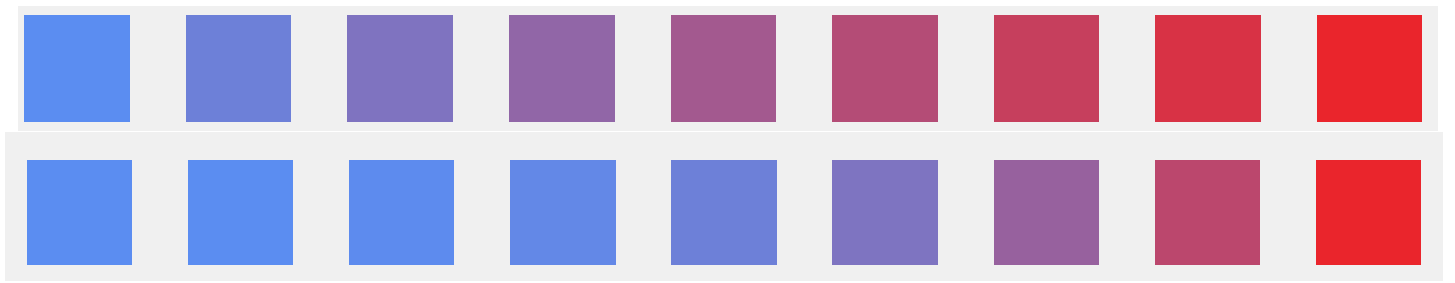
```
$color->set ( saturation => 80 );    # absolut
$color->add( lightness => -10 );    # relativ
$color->blend( with => 'green', in => 'HSL',
                        pos => 0.7 );
# 70% green  30% color , RGB ist default
# gleiche Argumentnamen wie zuvor (in)
```

Gruppe von Farbobjekten

```
$c->gradient( to => $c2, steps => 9,
              in => 'HSL', dynamic => 3);
```

linear :

dyn. 3 :

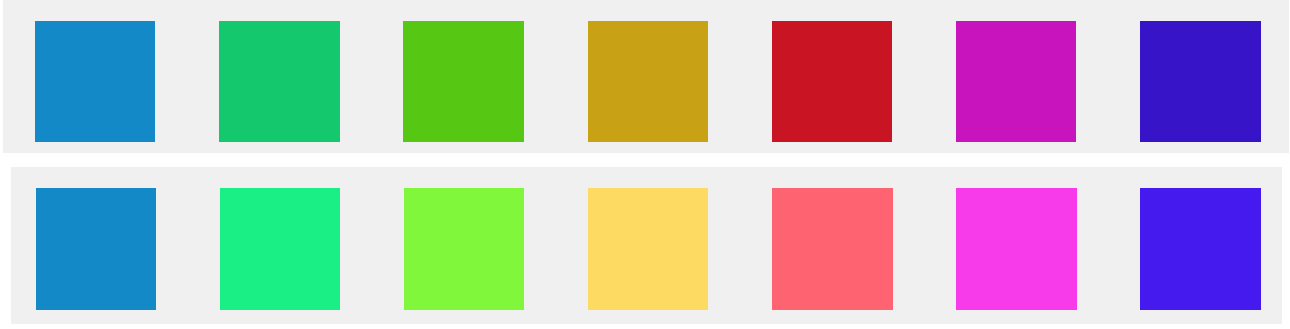


Gruppe von Farbobjekten

```
$c->complement( steps => 7,
                s => 20, l => 30);
```

no tilt:

with tilt:



Teil 3 - Softwaredesign

3 Gedanken zum Nachdenken

… als Anregung präsentiert

Guter Stil (Cell. Prog.)

- kleine Module, kleine Routinen
- wenige Argumente (signatur syntax)
- einfacher Kode # Kommentare
- vollständig getestet (kein TDD)
- vollständige Dokumentation (DDD!)

Modul in Zahlen

- 73 kB Download
- 21 Code - Dateien
- 173.2 kB (< 9 kB / Datei)
- 1.6 kLoC (< 80 Zeilen / Datei)
- 4 Abstraktionsebenen
- German Overengineering?

Was verzerrt Statistik ?

- 121 kB / 23 Files Tests waren nicht inklusive
- viel Dokumentation (deswegen LoC)
- 720 Farbkonstanten (50.4 kB – 30%)
- 6 util sub (45 Zeilen statt 2 Abhängigkeiten)
- 12 Farbräume (30-60 Zeilen)
  - deklarativ & dicht

CMY Namensraum

```
package Graphics::Toolkit::Color::Space::Instance::CMY;
use Graphics::Toolkit::Color::Space;
my $cmy_def = Graphics::Toolkit::Color::Space->new (
                        axis => [qw/cyan magenta yellow/] );
$cmy_def->add_converter ( 'RGB', \&to_rgb, \&from_rgb );
sub from_rgb { map { 1 - $_} @_ }
sub to_rgb   { map { 1 - $_} @_ }
$cmy_def;
```

5 Stufen einer Konvertierung

- Deformatierung: ‘#FFFFFF‘ → 255, 255, 255
- Normalisierung: 255, 255, 255 → 1, 1, 1
- Konvertierung: 1, 1, 1 → 0, 0, 1
- Denormalisierung: 0, 0, 1 → 0, 0, 100
- Formatierung: 0, 0, 100 → ‘hsl(0, 0, 100)‘

Unabhängig!

- Formatierung: Liste, ARRAY, String, CSS
  HASH, Char_hash, hex
- Normalisierung: beliebig mit defaults
- Konvertierung: RGB, CMY, CMYK,
  HSL, HSV, HSB, HWB,
  YIQ, XYQ, LAB, LUV, HCL

Definiert im Farbraum!

- Formatierung: zusätzliche
- Normalisierung: optional
- Konvertierung: Pflicht!
- Namen: Raum, Achsen, Abkürzungen
- Geometrie: Euklid / Zylinder

CMY Namensraum

```
package Graphics::Toolkit::Color::Space::Instance::CMY;
use Graphics::Toolkit::Color::Space;
my $cmy_def = Graphics::Toolkit::Color::Space->new (
                        axis => [qw/cyan magenta yellow/] );
$cmy_def->add_converter ( 'RGB', \&to_rgb, \&from_rgb );
sub from_rgb { map { 1 - $_} @_ }
sub to_rgb   { map { 1 - $_} @_ }
$cmy_def;
```

RGB Namensraum

```
package Graphics::Toolkit::Color::Space::Instance::RGB;
use Graphics::Toolkit::Color::Space;
my $rgb_def = Graphics::Toolkit::Color::Space->new (
                        axis => [qw/red green blue/], range => 255 );
$rgb_def->add_formatter(   'hex',   \&hex_from_rgb );
$rgb_def->add_deformatter( 'hex',
                    sub { rgb_from_hex(@_) if is_hex(@_) } );
$rgb_def;
```

HSL Namensraum

```
package Graphics::Toolkit::Color::Space::Instance::HSL;
use Graphics::Toolkit::Color::Space;
my $hsl_def = Graphics::Toolkit::Color::Space->new (
                    axis => [qw/hue saturation lightness/],
                  range => [ 360, 100, 100 ],
                    type => [qw/angular linear linear/]);
$hsl_def->add_converter( 'RGB', \&to_rgb, \&from_rgb );
$hsl_def;
```

YIQ Namensraum

```
package Graphics::Toolkit::Color::Space::Instance::YIQ;

use Graphics::Toolkit::Color::Space;

my $yiq_def = Graphics::Toolkit::Color::Space->new (
                    axis => [qw/luminance in_phase quadrature/],
                   short => [qw/Y I Q/],
                  range => [1, [-$i_max, $i_max], [-$q_max, $q_max]] );


$yiq_def;
```

Was verzerrt Statistik ?

- 121 kB / 23 Files Tests waren nicht inklusive
- viel Dokumentation (deswegen LoC)
- 720 Farbkonstanten (50.4 kB / 173)
- 6 util sub (45 Zeilen statt 2 Abhängigkeiten)
- 12 Farbräume (30-60 Zeilen)
  - deklarativ & dicht

Was bleibt übrig?

- 7 Dateien
- 103.3 kB (~14.7kB / Datei)
- 1100 LoC (158 / Datei – 1.5 Bildschirme)
- 4 Abstraktionsebenen (7 ?)
- German Overengineering?

Graphics::Toolkit::Color

- öffentliche API
  - Dokumentation
  - Argumentencheck (entferne Magie)
  - leite Arbeit weiter
- Doku: berechnet Mengen an Farben
  - geschieht (noch) hier

Graphics::Toolkit::Color

- öffentliche API
  - Dokumentation
  - Argumentencheck (entferne Magie)
  - leite Arbeit weiter
- Doku: berechnet Mengen an Farben
  - geschieht (noch) hier (316 LoC)

Graphics::Toolkit::Color

- G::T::C Nutzer-API
  - [G::T::C::Set] Farbmengen
  - G::T::C::Values Einzelfarben (numerisch)
    Top-Level-Funktionen
    mit normal. RGB Tripeln
  - G::T::C::Names übersetze Farbnamen

Graphics::Toolkit::Color

- G::T::C Nutzer-API
  - [G::T::C::Set] Farbmengen
  - G::T::C::Values Einzelfarben (numerisch)

    Top-Level-Funktionen

    mit normal. RGB Tripeln
  - G::T::C::Names übersetze Farbnamen

Graphics::Toolkit::Color

- G::T::C — Nutzer-API
  - [G::T::C::Set] — Farbmengen
  - G::T::C::Values — Einzelfarben (numerisch)

mit normal. RGB Tripeln

kein Wissen über Räume

Formate, Distanzen

unterhalb GTC::Values:

- G::T::C::Values höchste Abstraktion
  - GTC::Space::Hub verwaltet Farbräume
  - GTC::Space Farbraum
  - GTC::Space::Shape Abstände, (De-) Normal.
  - GTC::Space::Basis Vektor & Raum Namen

Vererbung Vermeiden:

- G::T::C::Values real Vektor Objekt
  - GTC::Space::Hub Paket verwaltet Objekte
  - GTC::Space Klasse
  - GTC::Space::Shape Klasse: Space Attribut
  - GTC::Space::Basis Klasse: Space Attribut

Vererbung Vermeiden:

- Attribute sind Objekte (kapseln Funktionalität)
  - nennt sich Delegation
  - wie OOP eigentlich sein sollte

Vererbung Vermeiden:

- Attribute sind Objekte (kapseln Funktionalität)
- Farbräume sind Objekte
  - erben nichts
  - mutieren eigne Methoden (mit Introspektion)
  - Objekte einfacher handhabbar als Klassen

Referenz statt 1 als Rückgabe

```
package Graphics::Toolkit::Color::Space::Instance::CMY;
use Graphics::Toolkit::Color::Space;
my $cmy_def = Graphics::Toolkit::Color::Space->new (
                        axis => [qw/cyan magenta yellow/] );
$cmy_def->add_converter ( 'RGB', \&to_rgb, \&from_rgb );
sub from_rgb { map { 1 - $_} @_ }
sub to_rgb   { map { 1 - $_} @_ }
$cmy_def;
```

Ref. verwaltet der Space::Hub

```
package Graphics::Toolkit::Color::Space::Instance::CMY;
use Graphics::Toolkit::Color::Space;
my $cmy_def = Graphics::Toolkit::Color::Space->new (
                        axis => [qw/cyan magenta yellow/] );
$cmy_def->add_converter ( 'RGB', \&to_rgb, \&from_rgb );
sub from_rgb { map { 1 - $_} @_ }
sub to_rgb   { map { 1 - $_} @_ }
$cmy_def;
```

Ref. verwaltet der Space::Hub

```
my @space_packages = $base_package,
        qw/CMY CMYK HSL HSV HSB HWB YIQ XYZ LAB LUV HCL/;
my %space_obj = map { $_ =>
          require "Graphics/Toolkit/Color/Space/Instance/$_.pm"
                                  } @space_packages;
```

Danke !

Fragen ?