cvd

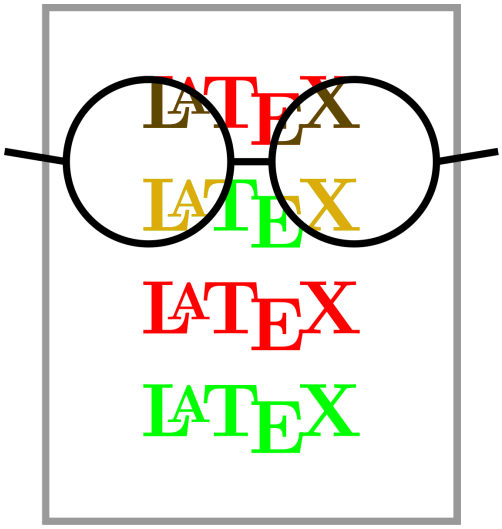
A LaTeX Package for Simulating Color Vision Deficiency

Johan Larsson Simon Pfahler

24 June 2026

Table of contents

Introduction



The `cvd` package is a LaTeX package for simulating color vision deficiency.

Part I

User Guide

1 Getting Started

The `cvd` package helps you create colorblind-safe LaTeX documents by simulating various types of color vision deficiency (CVD). This allows you to check whether your color choices are accessible to readers with different kinds and severities of CVDs without needing to consult someone affected by color blindness.

1.1 Quick Start

To quickly check your document for deuteranopia (the most common form of color blindness), simply load the package like this:

```
\usepackage[deuteranopia]{cvd}
```

All colors in your document will then be transformed to appear as they would to someone with deuteranopia.

1.2 What is CVD Simulation?

Color vision deficiency simulation transforms colors according to scientific models of how different types of color blindness affect color perception. By applying these transformations to your document, you can verify that information conveyed through color remains distinguishable even for colorblind readers.

1.3 When to Use This Package

CVD simulation is useful for a wide range of documents, including:

- Scientific papers with color-coded data, charts, or graphs
- Educational materials that use color to convey information
- Presentations and slides
- Maps with color-coded regions

Essentially, it is useful for any document where color plays a role in conveying information.

1.4 Basic Usage

The package provides several preset options for common CVD types:

- `protanopia` — red-blind (complete)
- `deuteranopia` — green-blind (complete)
- `tritanopia` — blue-blind (complete)
- `protanomaly` — red-weak (partial, severity 0.5)
- `deuteranomaly` — green-weak (partial, severity 0.5)
- `tritanomaly` — blue-weak (partial, severity 0.5)

Example usage:

```
\usepackage[protanopia]{cvd}      % Red-blind simulation
\usepackage[tritanomaly]{cvd}     % Blue-weak simulation
```

You can also customize the severity (0 to 1, where 1 is maximum):

```
\usepackage[type=deuteranopia, severity=0.5]{cvd}
```

and even set change the type and severity halfway through your document:

```
\cvdset{type=tritanopia, severity=0.2}
```

2 Package Options

The behavior of the CVD simulation is configured through package options. These options apply to the entire document or until they are changed later using the commands described in Commands.

2.1 CVD Type Options

2.1.1 Type Presets

The package provides convenient preset options for common color vision deficiencies:

Option	Description	Equivalent to
`protanopia`	Red-blind (complete red-green color blindness)	`type=protanopia, severity=1.0`
`deuteranopia`	Green-blind (complete red-green color blindness)	`type=deuteranopia, severity=1.0`
`tritanopia`	Blue-blind (complete blue-yellow color blindness)	`type=tritanopia, severity=1.0`
`protanomaly`	Red-weak (partial red-green color blindness)	`type=protanopia, severity=0.5`
`deuteranomaly`	Green-weak (partial red-green color blindness)	`type=deuteranopia, severity=0.5`
`tritanomaly`	Blue-weak (partial blue-yellow color blindness)	`type=tritanopia, severity=0.5`

2.1.2 Custom Type and Severity

For more control, you can specify the type and severity explicitly:

Option	Values	Default	Description
`type`	`protanopia`, `deuteranopia`, `tritanopia`	none	Sets the type of color-vision deficiency
`severity`	0.0 to 1.0	none	Severity level, where 1.0 means maximum severity (full colorblindness) and 0 means normal vision

2.2 Graphics Options

Option	Values	Default	Description
`graphics hook`	`true`, `false`	`true`	Enable or disable the simulation for included PDF images
`graphics convert`	`true`, `false`	`false`	Enable or disable ImageMagick conversion for raster images (PNG/JPG)

2.3 Complete Examples

```
% Most common use case: check for deuteranopia
\usepackage[deuteranopia]{cvd}

% Full customization
\usepackage[
  type=protanopia,
  severity=0.7,
  graphics hook=true,
  graphics convert=true
]{cvd}
```

2.4 Option Interaction

- Preset options (`protanopia`, `deuteranopia`, etc.) override individual `type` and `severity` settings

- If both a preset and individual options are specified, the preset takes precedence
- Graphics options are independent of CVD type and severity settings
- Options can be changed mid-document using the commands in Commands

3 Commands

The `cvd` package provides several commands to control the CVD simulation at any point in your document. These commands allow you to enable, disable, and configure the simulation dynamically.

3.1 Simulation Control

3.1.1 Enable/Disable Simulation

Command	Description
`\cvdenable`	Enable color vision deficiency simulation
`\cvddisable`	Disable color vision deficiency simulation

When you disable the simulation using `\cvddisable`, the type and severity settings are stored in the background. A subsequent `\cvdenable` restores the original CVD simulation behavior.

```
\cvdenable   % Turn on CVD simulation
...          % Content with CVD applied
\cvddisable  % Turn off CVD simulation
...          % Content without CVD
\cvdenable   % Turn on CVD simulation again (restores previous settings)
```

3.2 Configuration Commands

3.2.1 Set CVD Type

`\cvdtype{<type>}`

Sets the type of color-vision deficiency. Possible values are: - `protanopia` — red-blind - `deuteranopia` — green-blind - `tritanopia` — blue-blind

Example:

```
\cvdtype{deuteranopia}
```

3.2.2 Set Severity

```
\cvdseverity{<value>}
```

Sets the severity level on a scale from 0 to 1. - `1` means maximum severity (full colorblindness of the given type) - `0` means completely normal vision

Example:

```
\cvdseverity{0.5}  % 50% severity
```

3.2.3 Set Multiple Options

```
\cvdset{<options>}
```

Set one or more CVD options at once using key-value syntax. All options available at package load time can be used here.

Example:

```
\cvdset{type=protanopia, severity=0.8, graphics hook=true}
```

3.3 Image Commands

3.3.1 Include Raster Image with CVD

```
\cvdincludegraphics[<options>]{<path>}
```

Include a raster image (PNG/JPG) with CVD transformation applied.

This macro behaves identically to `\includegraphics` when CVD simulation is enabled. However, when CVD simulation is disabled, this macro can be used to apply the CVD transformation to a single raster image.

Note: This macro has no effect for PDF images; PDF images are transformed according to the current state of the CVD simulation via the `graphics hook` option.

Example:

```
\cvdincludegraphics[width=0.5\textwidth]{example.png}
```

3.4 Color Commands

3.4.1 Define CVD Color

```
\cvddefinecolor[<options>]{<source color>}{<target color>}
```

Define a new color named `<target color>` by applying the current CVD transformation to the existing color `<source color>`.

The optional `<options>` argument can be used to override the current CVD settings (e.g., `type`, `severity`) for this specific color definition, using the same syntax as `\cvdset`.

This is particularly useful for creating pre-computed CVD colors that can be used consistently throughout your document, especially with shadings and gradients (see Shadings).

Example:

```
% Define a color that appears as blue would to someone with protanopia
\cvddefinecolor[protanopia, severity=0.5]{blue}{blue-cvd}

% Use the pre-computed color
\textcolor{blue-cvd}{This text uses the CVD-transformed blue color}
```

Example with multiple CVD variants:

```
\cvddefinecolor[protanopia]{red}{red-p}
\cvddefinecolor[deuteranopia]{red}{red-d}
\cvddefinecolor[tritanopia]{red}{red-t}
```

3.5 Complete Example

```
\documentclass{article}
\usepackage{xcolor}
\usepackage{cvd}

\begin{document}

% Start with normal vision
\cvddisable
Normal vision: \textcolor{red}{Red text} \textcolor{green}{Green text}

% Enable deuteranopia simulation
\cvdenable
\cvdtype{deuteranopia}
```

```
\cvdseverity{1.0}
\bigskip

Deuteranopia (severity 1.0): \textcolor{red}{Red text} \textcolor{green}{Green text}

% Try a different type
\cvdtype{protanopia}
\cvdseverity{0.5}
\bigskip

Protanopia (severity 0.5): \textcolor{red}{Red text} \textcolor{green}{Green text}

% Define and use pre-computed colors
\cvddisable
\cvddefinecolor[protanopia]{red}{red-p}
\cvddefinecolor[protanopia]{green}{green-p}
\bigskip

Pre-computed protanopia colors: \textcolor{red-p}{Red (as protanope)} \textcolor{green-p}{Gr

\end{document}
```

4 Images and Graphics

The `cvd` package handles both vector (PDF) and raster (PNG/JPG) images, transforming colors according to the configured CVD simulation. The transformation behavior depends on the image type and the package options.

4.1 Image Types

4.1.1 PDF Vector Images

PDF images are transformed automatically when the `graphics hook` option is enabled (default: `true`). The package uses LuaTeX's `process_pdf_image_content` callback to intercept and transform color operators in the PDF content streams.

Supported color operators: - `rg` / `RG` — RGB color (device-color) - `k` / `K` — CMYK color (device-color)

Not supported: - `scn` / `SCN` — Spot/separation colors, ICCBased colors, and pattern colors

Example:

```
\usepackage[graphics hook=true]{cvd}
\cvdtype{deuteranopia}

\begin{document}
% This PDF image will have its colors transformed
\includegraphics{vector-image.pdf}
\end{document}
```

4.1.2 Raster Images (PNG/JPG)

Raster images require the `graphics convert` option to be enabled and additional system dependencies:

Requirements: - Compile with `--shell-escape` flag - ImageMagick must be installed on the system

When these requirements are met, raster images are converted on-the-fly with the CVD transformation applied.

Example:

```
\usepackage[graphics convert=true]{cvd}
\cvdtype{protanopia}

\begin{document}
% This PNG image will be converted with CVD transformation
\includegraphics{photo.png}
\end{document}
```

Compile with:

```
lualatex --shell-escape mydocument.tex
```

4.2 Graphics Options

Option	Default	Description
`graphics hook`	`true`	Enable transformation for PDF vector images
`graphics convert`	`false`	Enable ImageMagick conversion for raster images

4.3 The CVD Cache

To store the transformed raster images when using `graphics convert=true`. This directory is created automatically in the output directory.

The cache stores: - Transformed versions of raster images - Metadata about the CVD settings used for each transformation in the filename

This also means that subsequent compilations are faster, as images only need to be transformed once per CVD settings.

4.4 Alternative: `\cvdincludegraphics`

The `\cvdincludegraphics` command provides explicit control over CVD transformation for raster images:

```
\cvdincludegraphics[<options>]{<path>}
```

This command: - Applies CVD transformation to raster images even when `graphics convert=false` - Behaves identically to `\includegraphics` when CVD simulation is enabled - Has **no effect** on PDF images (use `graphics hook` for PDFs)

Example:

```
% Transform a single raster image, even with CVD disabled globally
\cvddisable
\cvdincludegraphics[width=0.5\textwidth]{chart.png}
```

4.5 Limitations and Workarounds

4.5.1 Raster Image Requirements

If ImageMagick is not installed or you don't compile with `--shell-escape`, raster images will pass through untransformed. In this case: - PDF vector images can still be transformed (if `graphics hook=true`) - Consider using vector formats (PDF, SVG) instead of raster formats

4.5.2 Large PDF Files

When transforming PDF content streams, significant growth may cause truncation or rendering issues in some PDF viewers. This can happen with complex PDFs containing many color operations.

Mitigation: - Use raster formats for complex PDFs - Split complex documents into multiple files - A warning is issued when significant stream growth is detected

4.6 Best Practices

1. **Use vector formats when possible** — PDF images are transformed more reliably and don't require external dependencies
2. **Enable both options for full coverage:**

   ```
   \usepackage[graphics hook=true, graphics convert=true]{cvd}
   ```

3. **Always compile with –shell-escape** when using raster images:

   ```
   lualatex --shell-escape mydocument.tex
   ```

4. **Use `\cvdincludegraphics` for specific images** when you want to apply CVD transformation to select raster images without enabling global raster conversion

5. **Pre-compute colors for complex graphics** — When working with shadings and gradients, use `\cvddefinecolor` to create pre-computed colors (see Shadings)

5 Examples

This page provides practical examples demonstrating various use cases for the `cvd` package.

5.1 Basic Usage

5.1.1 Quick Check for Deuteranopia

The simplest way to check your document for the most common form of color blindness:

```
\documentclass{article}
\usepackage[deuteranopia]{cvd}
\usepackage{xcolor}

\begin{document}

This document simulates deuteranopia (green-blind) color vision.

\textcolor{red}{Red text} \textcolor{green}{Green text} \textcolor{blue}{Blue text}

\end{document}
```

5.1.2 Comparing Multiple CVD Types

Create a comparison document showing how colors appear with different CVD types:

```
\documentclass{article}
\usepackage{cvd}
\usepackage{xcolor}

\begin{document}

\section{Normal Vision}
\cvddisable
\textcolor{red}{Red} \textcolor{green}{Green} \textcolor{blue}{Blue}
```

```
\section{Protanopia (Red-blind)}
\cvdenable
\cvdtype{protanopia}
\cvdseverity{1.0}
\textcolor{red}{Red} \textcolor{green}{Green} \textcolor{blue}{Blue}

\section{Deuteranopia (Green-blind)}
\cvdtype{deuteranopia}
\cvdseverity{1.0}
\textcolor{red}{Red} \textcolor{green}{Green} \textcolor{blue}{Blue}

\section{Tritanopia (Blue-blind)}
\cvdtype{tritanopia}
\cvdseverity{1.0}
\textcolor{red}{Red} \textcolor{green}{Green} \textcolor{blue}{Blue}

\end{document}
```

5.2 Scientific Papers and Charts

5.2.1 Color-Coded Data

When presenting color-coded data, check that your color scheme is accessible:

```
\documentclass{article}
\usepackage{pgfplots}
\usepackage[deuteranopia]{cvd}
\pgfplotsset{compat=1.18}

\begin{document}

\begin{tikzpicture}
\begin{axis}[xlabel=X, ylabel=Y]
  \addplot[red, mark=*] coordinates {(0,0) (1,1) (2,2)};
  \addplot[green, mark=square*] coordinates {(0,1) (1,0) (2,1)};
  \addplot[blue, mark=triangle*] coordinates {(0,2) (1,1) (2,0)};
\end{axis}
\end{tikzpicture}

\end{document}
```

If the colors become hard to distinguish, consider changing them until they can also be

distinguished under the CVD simulation.

5.2.2 Multiple Charts with Different CVD Types

```
\documentclass{article}
\usepackage{pgfplots}
\usepackage{cvd}
\pgfplotsset{compat=1.18}

\begin{document}

% Define CVD-transformed colors for each type
\cvddefinecolor[type=protanopia]{red}{red-p}
\cvddefinecolor[type=protanopia]{green}{green-p}
\cvddefinecolor[type=protanopia]{blue}{blue-p}

\cvddefinecolor[type=deuteranopia]{red}{red-d}
\cvddefinecolor[type=deuteranopia]{green}{green-d}
\cvddefinecolor[type=deuteranopia]{blue}{blue-d}

\begin{tikzpicture}
\begin{axis}[xlabel=X, ylabel=Y, title=Protanopia]
  \addplot[red-p, mark=*] coordinates {(0,0) (1,1) (2,2)};
  \addplot[green-p, mark=square*] coordinates {(0,1) (1,0) (2,1)};
\end{axis}
\end{tikzpicture}

\begin{tikzpicture}
\begin{axis}[xlabel=X, ylabel=Y, title=Deuteranopia]
  \addplot[red-d, mark=*] coordinates {(0,0) (1,1) (2,2)};
  \addplot[green-d, mark=square*] coordinates {(0,1) (1,0) (2,1)};
\end{axis}
\end{tikzpicture}

\end{document}
```

5.3 Maps and Color-Coded Regions

5.3.1 Simple Color-Coded Map

```
\documentclass{article}
\usepackage{tikz}
\usepackage[deuteranomaly]{cvd}

\begin{document}

\begin{tikzpicture}
  % Draw a simple map with color-coded regions
  \fill[red!30] (0,0) rectangle (2,2);
  \fill[green!30] (2,0) rectangle (4,2);
  \fill[blue!30] (4,0) rectangle (6,2);

  \node at (1,1) {Region A};
  \node at (3,1) {Region B};
  \node at (5,1) {Region C};
\end{tikzpicture}

\end{document}
```

5.4 Shadings and Gradients

5.4.1 Basic Gradient with CVD

```
\documentclass{article}
\usepackage{tikz}
\usepackage[deuteranopia]{cvd}

\begin{document}

\begin{tikzpicture}
  % This gradient will be transformed with deuteranopia
  \shade[left color=red, right color=blue] (0,0) rectangle (6,2);
  \node at (3,1) {Deuteranopia Gradient};
\end{tikzpicture}

\end{document}
```

5.4.2 Multiple Gradients with Different CVD Types

See the Shadings page for detailed examples on handling shadings with different CVD types, including the caching workaround.

5.5 Raster Images

5.5.1 Transforming a Photo

```
\documentclass{article}
\usepackage[graphics convert=true]{cvd}
\cvdtype{protanopia}

\begin{document}

% Requires --shell-escape and ImageMagick
\includegraphics[width=\textwidth]{my-photo.jpg}

\end{document}
```

Compile with:

```
lualatex --shell-escape mydocument.tex
```

5.5.2 Transforming Specific Images Only

```
\documentclass{article}
\usepackage[protanopia, graphics convert=false]{cvd}
% CVD simulation is disabled globally

\begin{document}

% This image is NOT transformed
\includegraphics[width=0.5\textwidth]{normal-image.png}

% This specific image IS transformed using \cvdincludegraphics
\cvdincludegraphics[width=0.5\textwidth]{cvd-image.png}

\end{document}
```

6 Shadings (gradients)

`cvd` transforms TikZ/pgf shadings—both axial (linear) and radial gradients emitted by `\shade`, `\shadedraw`, and friends. The transformation is applied to the RGB or CMYK color stops at the moment pgf emits the shading's PDF object. For CMYK, the K component is left unchanged.

6.1 Basic usage

Set the cvd type once before `\begin{document}` and any shading you draw will come out simulated:

```
\documentclass{article}
\usepackage{tikz}
\usepackage{cvd}

\cvdtype{deuteranopia}
\cvdseverity{1.0}
\cvdenable

\begin{document}
\begin{tikzpicture}
  \shade[left color=red, right color=blue] (0,0) rectangle (4,1);
\end{tikzpicture}
\end{document}
```

6.2 Comparing several cvd types in one document

Toggling `\cvdtype` between two `\shade` calls that use the **same** input colors does **not** produce two different gradients. pgf caches each shading by its input color stops (see `\pgfuseshading` in `pgfcoreshade.code.tex`); the second call hits the cache and reuses the PDF object emitted by the first, so the cvd state at the time of the second call has no effect.

The workaround is to pre-compute simulated colors with `\cvddefinecolor` and shade with those, leaving cvd disabled at draw time:

```
\cvddefinecolor[type=deuteranopia]{red}{red-d}
\cvddefinecolor[type=deuteranopia]{blue}{blue-d}
\cvddefinecolor[type=protanopia]{red}{red-p}
\cvddefinecolor[type=protanopia]{blue}{blue-p}

\begin{tikzpicture}
  \shade[left color=red,   right color=blue]   (0,0) rectangle (4,0.8);
  \shade[left color=red-d, right color=blue-d] (5,0) rectangle (9,0.8);
  \shade[left color=red-p, right color=blue-p] (0,-1.5) rectangle (4,-0.7);
\end{tikzpicture}
```

Each row uses different color names, so pgf's cache key differs and each shading gets its own PDF object.

6.3 Known limitations

- **Functional shadings (`\pgfdeclarefunctionalshading`, `ShadingType 1`)** embed their colors inside a PostScript Type 4 function. cvd does not parse or rewrite that function, so functional shadings pass through unchanged.
- **Cached duplicate shadings** – as described above, two `\shade` calls with identical input colors share one PDF object. Use `\cvddefinecolor` to side-step the cache.

7 Current Limitations

While the `cvd` package provides powerful CVD simulation capabilities, there are some technical limitations to be aware of. This page documents all known limitations and their workarounds.

7.1 Engine Requirements

Only LuaLaTeX is currently supported. Support for other engines is under development.

7.2 Raster Image Requirements

Raster image (PNG/JPG) transformation requires compiling with `-shell-escape` and ImageMagick to be installed on the system. Without these, raster images pass through untransformed.

7.3 PDF/Vector Graphics Limitations

- Functional shadings (`\pgfdeclarefunctionalshading`, ShadingType 1) pass through unchanged as their colors are embedded in PostScript Type 4 functions.
- pgf caches shadings, so changing CVD settings between identical `\shade` calls has no effect. Workaround: use `\cvddefinecolor` to create unique color names for each CVD variant you need.
- Spot/separation colors, ICCBased colors, and pattern colors are not transformed. Only device-color operators `rg/RG` (RGB) and `k/K` (CMYK) are supported.
- When transforming PDF content streams, significant growth may cause truncation or rendering issues in some PDF viewers. Mitigation: use raster formats for complex PDFs, or split them into multiple documents. When a significiant growth of te PDF stream is detected, a warning for this is issued.

7.4 Feature Requests

If you encounter limitations that affect your use case, consider:

1. Checking the GitHub repository for open issues
2. Opening a new issue with a minimal working example
3. Contributing a pull request with improvements

The package is actively developed, and many limitations may be addressed in future versions.

Part II

Background

8 Theory: Color Vision Deficiency Simulation

This page explains the mathematical foundation behind how the `cvd` package simulates color vision deficiencies. Understanding this theory helps you appreciate how the transformations work and why they produce realistic simulations of color blindness. This is based on "A Physiologically-based Model for Simulation of Color Vision Deficiency" by Gustavo M. Machado, Manuel M. Oliveira.

8.1 Overview

Color vision deficiency (CVD) simulation transforms colors according to scientific models of how different types of color blindness affect human color perception. The process involves adjusting the cone sensitivities in the human eye and then transforming colors through a well-defined mathematical pipeline.

8.2 The Simulation Pipeline

The CVD simulation follows a systematic six-step process:

8.2.1 Step 1: Adjusted Cone Sensitivities

For a given CVD type, we first obtain adjusted cone sensitivities. The human eye has three types of cone cells: - **L cones** (Long wavelength) - most sensitive to red - **M cones** (Medium wavelength) - most sensitive to green - **S cones** (Short wavelength) - most sensitive to blue

The adjusted sensitivities account for: - **Cone shift** ($\Delta\lambda_S \in [0, 60]$) for S cone cells - **Severity factors** ($\alpha_L, \alpha_M \in [0, 1]$) for protanopia and deuteranopia

The mathematical formulas for adjusted cone sensitivities are:

$$L_a(\lambda) = (1 - \alpha_L)L(\lambda) + 0.96\frac{\text{Area}_L}{\text{Area}_M}\alpha_L M(\lambda)$$
$$M_a(\lambda) = (1 - \alpha_M)M(\lambda) + \frac{1}{0.96}\frac{\text{Area}_M}{\text{Area}_L}\alpha_M L(\lambda)$$
$$S_a(\lambda) = S(\lambda + \Delta\lambda_S)$$

Where: - $L(\lambda), M(\lambda), S(\lambda)$ are the original cone sensitivity functions - $L_a(\lambda), M_a(\lambda), S_a(\lambda)$ are the adjusted sensitivity functions - α_L, α_M control the severity (0 = normal vision, 1 = full colorblindness) - The area ratios account for the different densities of cone types in the retina - The factor 0.96 is a magic number that Machado et al. figured out is helpful

8.2.2 Step 2: Opponent-Color-Model Basis Functions

Human color vision can be understood via opponent color theory, where colors are perceived through opposing pairs: - **White-Black (Luminance)** - WS - **Yellow-Blue** - YB - **Red-Green** - RG

For the adjusted cone sensitivities, we compute the opponent-color-model basis functions:

$$\begin{pmatrix} WS(\lambda) \\ YB(\lambda) \\ RG(\lambda) \end{pmatrix} = \begin{pmatrix} 0.6 & 0.4 & 0 \\ 0.24 & 0.105 & -0.7 \\ 1.2 & -1.6 & 0.4 \end{pmatrix} \begin{pmatrix} L_a(\lambda) \\ M_a(\lambda) \\ S_a(\lambda) \end{pmatrix}$$

This matrix transformation converts from cone space to the opponent color space that allows us to better model CVDs.

8.2.3 Step 3: Spectral Power Distributions

For each color channel in the color model (e.g., R, G, B in RGB), we obtain the spectral power distributions $\varphi_C(\lambda)$. These describe how much light of each wavelength is emitted or reflected by a color.

The spectral power distributions depend on the color model: - **RGB**: Based on monitor characteristics - **CMYK**: Based on printer ink and paper characteristics - **Other models**: Converted to RGB/CMYK first

8.2.4 Cone Sensitivity Functions

The simulation uses the CIE 2015 2° Standard Observer cone sensitivity data. These functions describe how sensitive each type of cone cell is to different wavelengths of light.

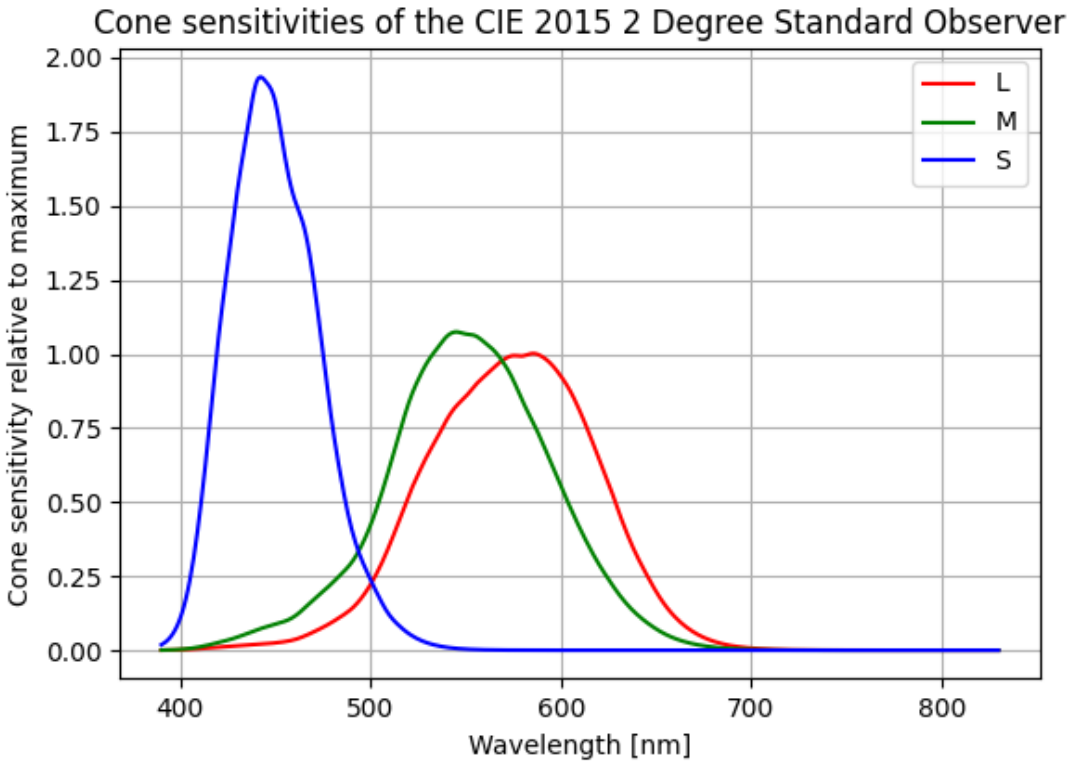


Figure 8.1: Cone Sensitivity Functions

8.2.5 RGB Spectral Power Distributions

For RGB colors, we use the measured spectral power distributions for an Apple Studio LCD display. This approximates the color characteristics of a typical monitor.

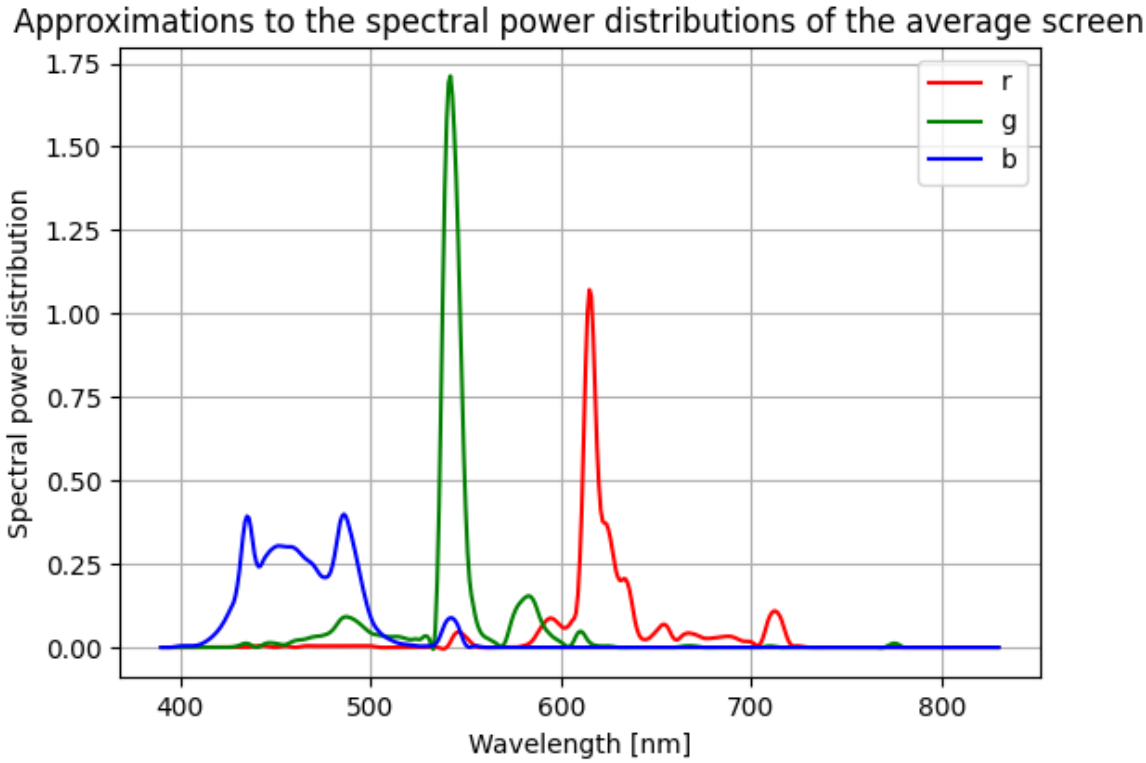


Figure 8.2: RGB Spectral Power Distributions

8.2.6 CMYK Spectral Power Distributions

For CMYK colors, we use example reflectivities for an ink-jet printer from Abebe 2011, multiplied by standard illuminant A. This approximates the average printer, paper, and lighting conditions for a subtractive color model.

Note that CMYK is a subtractive color model, so the spectral power distributions are modeled with a dip around peak absorption rather than a Gaussian peak.

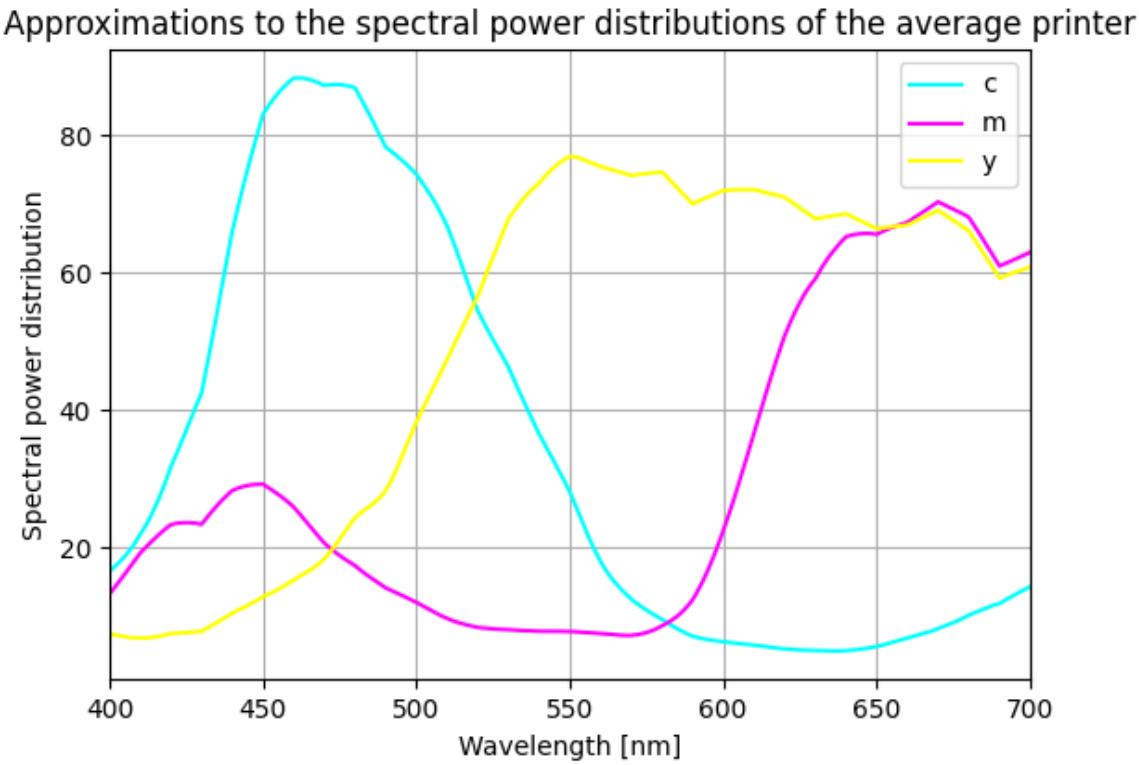


Figure 8.3: CMYK Spectral Power Distributions

8.2.7 Step 4: Projection to Opponent Color Space

We project the spectral power distributions onto the opponent color model basis functions. This is done by forming a matrix of convolutions:

$$\hat{T}_{i,C} = \int \varphi_C(\lambda) O_i(\lambda) \, \mathrm{d}\lambda$$

Where $O_i(\lambda)$ are the opponent color basis functions (WS, YB, RG).

This step essentially calculates how much each color channel stimulates each of the opponent color mechanisms.

8.2.8 Step 5: Normalization

We form the normalized transformation matrix T by applying normalization factors ρ_i:

$$\rho_i = \frac{1}{\sum_C \hat{T}_{i,C}}$$

And then:

$$T_{i,C} = \rho_i \hat{T}_{i,C}$$

This normalization ensures that the transformation preserves certain perceptual properties and maintains consistency across different color spaces.

8.2.9 Step 6: Final Color Transformation

To obtain the simulated color for a person with normal color vision experiencing a particular CVD, we apply the transformation:

$$\vec{C_s} = T_{\text{normal}}^{-1} T \vec{C}$$

Where: - $\vec{C}$ are the original color values (e.g., (r, g, b)) - T is the transformation matrix for the CVD being simulated - T_{normal} is the transformation matrix for normal vision - $\vec{C_s}$ are the simulated color values as perceived by someone with normal vision but showing what a person with CVD would see

8.3 References

The simulation methodology is based on established color vision science and implemented in the Python `colour` library (colour-science/colour), which provides the underlying color science computations.

An implementation of the described pipeline of creating the 3x3 transformation matrices is implemented in simon-pfahler/simon-pfahler/CVD-simulation-generator.

For more details on the mathematical foundations, "A Physiologically-based Model for Simulation of Color Vision Deficiency" by Gustavo M. Machado, Manuel M. Oliveira.

Part III

Implementation

A Main

> **Note**
>
> **Source file:** src/cvd.dtx

A.1 Package Dependencies

Load required packages.

```
\RequirePackage{iftex}
\RequirePackage{xcolor}
\RequirePackage{graphicx}
```

A.2 Engine Check

Currently only LuaTeX is fully supported.

```
\sys_if_engine_luatex:F
{
  \msg_error:nn { cvd } { luatex-required }
}
\msg_new:nnn { cvd } { luatex-required }
{
  LuaTeX~required.\\
  This~package~currently~only~works~with~LuaLaTeX.\\
  pdfLaTeX~support~is~under~development.
}
```

A.3 Load Lua Module

Load the Lua module that implements the CVD transformations. The `install_pdf_image_hook` function registers a callback that transforms colors in em-

bedded PDF pages (vector graphics only). We also load the Lua File System module for file timestamp checking.

```
\directlua{lfs = require("lfs"); cvd = require("cvd"); cvd.install_pdf_image_hook()}
```

A.4 Hook into xcolor

Use `xcolor`'s hook to transform RGB values before display. This handles text colors, color boxes, and other `xcolor`-based content.

```
\cs_set:Npn \XC@bcolor
{
  \directlua
  {
    token.set_macro("current@color",~
    cvd.transform_current_color("\luaescapestring{\current@color}"),~
    "global")
  }
}
```

A.5 Hook into pgf Shadings

TikZ/pgf shadings (linear and radial gradients) store their colors in PDF `Shading` dictionaries whose `Function` carries `/C0` and `/C1` color arrays. These objects sit in the page resources rather than in any content stream, so none of 's other transform paths reach them and they need a dedicated hook.

Patch pgf's leaf tuple emitters so the RGB or CMYK tuple is filtered through `cvd.transform` first. From one transform each emitter sets both the space-separated macro (/) that ends up in `/C0` and `/C1` for the pdf/luatex driver, and the brace-grouped system-layer record (/) used by the dvisvgm driver, so the two never disagree. Guarded with so loading without or is a no-op. `ShadingType 1` (functional) shadings and `DeviceGray` shadings are intentionally not handled.

```
\def \__cvd_patch_pgf_shadings:
  {
    \@ifundefined { pgf@getrgb@@ } { } {
      \def \pgf@getrgb@@ ##1,##2,##3!
        {
          \directlua { cvd.set_pgf_rgb("##1",~"##2",~"##3") }
        }
    }
```

```
    \@ifundefined { pgf@getcmyk@@ } { } {
      \def \pgf@getcmyk@@ ##1,##2,##3,##4!
        {
          \directlua { cvd.set_pgf_cmyk("##1",~"##2",~"##3",~"##4") }
        }
    }
  }
\__cvd_patch_pgf_shadings:
\AtBeginDocument { \__cvd_patch_pgf_shadings: }
```

A.6 User Commands

A.6.1 `\cvdtype`

Set the type of color vision deficiency to simulate.

```
\NewDocumentCommand \cvdtype { m }
{
  \directlua { cvd.set_type("#1") }
}
```

A.6.2 `\cvdseverity`

Set the severity of the simulation (0.0 to 1.0).

```
\NewDocumentCommand \cvdseverity { m }
  {
    \directlua { cvd.set_severity(#1) }
  }
```

A.6.3 `\cvdenable`

Enable CVD simulation.

```
\NewDocumentCommand \cvdenable { }
{
  \directlua { cvd.enable() }
}
```

A.6.4 \cvddisable

Disable CVD simulation.

```
\NewDocumentCommand \cvddisable { }
{
  \directlua { cvd.disable() }
}
```

A.6.5 \cvdincludegraphics

Include a graphics file with CVD transformation applied to raster images.

```
\tl_new:N \l__cvd_imgpath_tl
\NewDocumentCommand \cvdincludegraphics { O{} m }
{
  \tl_set:Nx \l__cvd_imgpath_tl
  {
    \directlua
    { tex.sprint(cvd.get_image_path("\luaescapestring{#2}")) }
  }
  \__cvd_orig_includegraphics[#1]{\tl_use:N \l__cvd_imgpath_tl}
}
```

A.6.6 \cvddefinecolor

Define a new color by applying CVD transformation to an existing color. Usage:

```
\tl_new:N \l__cvd_model_tl
\tl_new:N \l__cvd_values_tl

\NewDocumentCommand \cvddefinecolor { O{} m m }
{
  % Extract the original color
  \extractcolorspecs{#2}{\l__cvd_model_tl}{\l__cvd_values_tl}

  % Apply CVD transformation with specified settings
  \keys_set:nn { cvd } { #1 }
  \cvdenable

  % Transform the RGB values directly via Lua
  \directlua{
    local~values~=~"\luaescapestring{\l__cvd_values_tl}"
```

```
    local~r,~g,~b~=~values:match("([^,]+),([^,]+),([^,]+)")
    r,~g,~b~=~tonumber(r),~tonumber(g),~tonumber(b)
    r,~g,~b~=~cvd.transform("rgb",~r,~g,~b)
    token.set_macro("l__cvd_values_tl",~string.format("\csstring\%.6f,\csstring\%.6f,\csstri
  }

  % Define the color with transformed values
  \use:x { \definecolor {#3} { \exp_not:V \l__cvd_model_tl } { \exp_not:V \l__cvd_values_tl }

  \cvddisable
}
```

A.7 Package Configuration

Define keys for package configuration using . Keys are available both as package load-time options and via the command.

```
\bool_new:N \l__cvd_graphics_hook_bool
\bool_new:N \l__cvd_graphics_convert_bool
\cs_new_eq:NN \__cvd_orig_includegraphics \includegraphics

\keys_define:nn { cvd }
  {
    type          .code:n = { \cvdtype{#1} } ,
    severity      .code:n = { \cvdseverity{#1} } ,
    graphics~hook .bool_set:N = \l__cvd_graphics_hook_bool ,
    graphics~hook .initial:n = true ,
    graphics~hook .default:n = true ,
    graphics~hook / true .code:n = { \directlua { cvd.enable_graphics_hook() } } ,
    graphics~hook / false .code:n = { \directlua { cvd.disable_graphics_hook() } } ,
    graphics~convert .bool_set:N = \l__cvd_graphics_convert_bool ,
    graphics~convert .initial:n = false ,
    graphics~convert .default:n = true ,
    graphics~convert / true .code:n = { \__cvd_patch_includegraphics: } ,
    graphics~convert / false .code:n = { \__cvd_unpatch_includegraphics: } ,
    protanopia    .code:n = { \cvdtype{protanopia} \cvdseverity{1.0} } ,
    deuteranopia  .code:n = { \cvdtype{deuteranopia} \cvdseverity{1.0} } ,
    tritanopia    .code:n = { \cvdtype{tritanopia} \cvdseverity{1.0} } ,
    protanomaly   .code:n = { \cvdtype{protanopia} \cvdseverity{0.5} } ,
    deuteranomaly .code:n = { \cvdtype{deuteranopia} \cvdseverity{0.5} } ,
    tritanomaly   .code:n = { \cvdtype{tritanopia} \cvdseverity{0.5} } ,
    unknown       .code:n =
```

```
      { \msg_warning:nnx { cvd } { unknown-option } { \l_keys_key_str } }
  }
\msg_new:nnn { cvd } { unknown-option }
  { Unknown~option~'#1'. }

\cs_new:Npn \__cvd_patch_includegraphics:
{
  \RenewDocumentCommand \includegraphics { O{} m }
  {
    \cvdincludegraphics[##1]{##2}
  }
  \directlua { cvd.enable_graphics_convert() }
}

\cs_new:Npn \__cvd_unpatch_includegraphics:
{
  \cs_set_eq:NN \includegraphics \__cvd_orig_includegraphics
  \directlua { cvd.disable_graphics_convert() }
}

\NewDocumentCommand \cvdset { m }
{
  \keys_set:nn { cvd } { #1 }
}
\ProcessKeyOptions [ cvd ]
```