tinycio

Release 0.8.2 a

Sam Izdat

Aug 13, 2025

HOW TO:

Primitive, lightweight Python color library for PyTorch-involved projects. It implements color space conversion, tone mapping, LUT usage and creation, basic color correction and color balancing, and HDR-LDR encoding/decoding.

CHAPTER

ONE

GETTING STARTED

1. Recommended: set up a clean Python environment
2. Install PyTorch as instructed here
3. Run `pip install tinycio`
4. Run `tcio-setup` (installs freeimage binaries; iio docs on fi)

CHAPTER

TWO

ABOUT

Version 0.8.2 a

As pip has a tendency to screw up PyTorch installations, torch is deliberately left out of the explicit package requirements; install it whichever way is most appropriate, if you haven't done so already.

Note on limitations

I feel like a disclaimer is in order.

- This package is (obviously) not suitable for realtime rendering
- This package is *generally* not suitable for differentiable rendering
- I am not a color expert and can't guarantee scientific accuracy

This package was motivated by a frustration with the staggering complexities of digital color and put together as part of a larger project, already using PyTorch, to minimize external dependencies. In self-flattering terms, the goal here is a kind of spartan-utilitarian approach to color for image processing with some light machine learning. In less self-flattering terms, I have no idea what I'm doing. This is not meant to be a replacement for a proper color management solution.

While the code was tested within reason, some things are likely still not quite correct. For dependable color management, defer to OCIO or consider a more comprehensive and scientifically-minded Python project like Colour.

Supported color spaces (and other things treated as if they were):

- CIE XYZ
- CIE xyY
- sRGB
- Rec. 709
- sRGB / Rec. 709 (scene-linear)
- Rec. 2020
- Rec. 2020 (scene-linear)
- DCI-P3
- DCI-P3 (scene-linear)
- Display P3
- ACEScg
- ACEScc
- ACES 2065-1

- LMS
- OKLAB
- CIELAB
- HSL
- HSV
- OKHSL
- OKHSV

Supported tone mapping operators:

- AgX / AgX punchy
- ACEScg (fitted RRT+ODT)
- Hable (a.k.a. Uncharted 2)
- Reinhard (extended)

Supported LUT formats:

- CUBE

License

MIT License on all original code - see source for details

CHAPTER

THREE

HOW TO

3.1 Hello Color

A basic example:

```
from tinycio import TransferFunction, fsio

try:
    # Linearize an image
    im_srgb = fsio.load_image('my/srgb_image.png')
    im_linear = TransferFunction.srgb_eotf(im_srgb)
    fsio.save_image(im_linear, 'my/linear_image.png')
except Exception as e:
    print(e) # your error handling here
```

This package is divided into four user modules:

- *tinycio* - for all main color-related features and types
- *tinycio.fsio* - for loading from and saving to the file system
- *tinycio.util* - for miscellaneous color and non-color utility functions
- *tinycio.numerics* - for base numeric vector data types

It also exposes a few (optional) high-level abstractions to cut out a lot of the tedium:

- *ColorImage* - 3-channel *float32* color image type
- *Color* - 3-component *float32* color type
- *Chromaticity* - 2-component *float32* CIE xy chromaticity type

Exceptions may occur at any point, but exception handling will be omitted from subsequent code snippets for brevity.

3.2 Look up a wavelength

```
from tinycio import Spectral

# Look up a wavelength (nm) as CIE XYZ color
col_xyz = Spectral.wl_to_xyz(490)   # Float3([0.0320, 0.2080, 0.4652])

# Look up a wavelength as sRGB color
col_srgb = Spectral.wl_to_srgb(490) # Float3([0.    , 0.9257, 1.    ])
```

See: *Spectral*

3.3 Look up a color value

The tersest way:

```
from tinycio import Color

# Convert color from linear sRGB to CIE xyY
result = Color(0.8, 0.4, 0.2).convert('SRGB_LIN', 'CIE_XYY')
# returns Color([0.4129, 0.3817, 0.4706])
```

Slightly more verbose:

```
from tinycio import Color

# Specify the color and make a 1x1 linear sRGB ColorImage
col_im = Color(0.8, 0.4, 0.2).image('SRGB_LIN')

# Convert sRGB linear image to CIE xyY
result = Color(col_im.to_color_space('CIE_XYY'))
# returns Color([0.4129, 0.3817, 0.4706])
```

This is functionally equivalent to:

```
import torch
from tinycio import ColorSpace

# Specify the color as a tensor
col_im = torch.tensor([0.8, 0.4, 0.2], dtype=torch.float32)

# Expand it to a [3, 1, 1] sized image tensor
col_im = col_im.unsqueeze(-1).unsqueeze(-1)

# Convert
cs_in = ColorSpace.Variant.SRGB_LIN
cs_out = ColorSpace.Variant.CIE_XYY
result = ColorSpace.convert(col_im, source=cs_in, destination=cs_out)
# returns tensor([0.4129, 0.3817, 0.4706]) (squeezed)
```

Note

Most color spaces are implemented as operations on image tensors, but a few (OKHSL, OKHSV) are only available as direct color value conversions.

See: *Color.convert()*, *ColorSpace*

3.4 Change an image's color space

```
from tinycio import ColorImage

# Color space is assumed to be sRGB, bit depth assumed to be uint8 in, float32 out
ColorImage.load('my/image.png').to_color_space('ACES2065_1').save('my/image.exr')
```

A little more explicit:

```
from tinycio import ColorImage

# Load sRGB image from disk
im_in = ColorImage.load('my/image.png', graphics_format='UINT8', color_space='SRGB')

# Convert the image
im_out = im_in.to_color_space('ACES2065_1')

# Save as 32-bit-per-channel EXR file
im_out.save('my/image.exr', graphics_format='SFLOAT32')
```

This is functionally equivalent to:

```
from tinycio import ColorSpace, fsio

# Load image - returns [C, H, W] sized torch.Tensor
im_in = fsio.load_image('my/image.png')

# Specify color spaces
cs_in = ColorSpace.Variant.SRGB
cs_out = ColorSpace.Variant.ACES2065_1

# Covert the image
im_out = ColorSpace.convert(im_in, cs_in, cs_out)

# Save as 32-bit-per-channel EXR file
fmt_out = fsio.GraphicsFormat.SFLOAT32
fsio.save_image(im_out, 'my/image.exr', graphics_format=fmt_out)
```

Note

Tone mapping is treated as a discrete step and, in the general case, converted color values will not be normalized or clamped to a valid range for the destination color space automatically.

See: *ColorImage.to_color_space()*, *ColorSpace*

3.5 Encode HDR data as RGBA PNG

Formats like Logluv and RGBE can encode three channels of HDR data as a 32-bpp RGBA image.

```
from tinycio import fsio, Codec

# Load 3-channel HDR image
im_hdr = fsio.load_image('my/hdr_image.exr')

# Encode logluv - returns 4-channel tensor
im_logluv = Codec.logluv_encode(im_hdr)

# Save 4-channel LogLuv image
fsio.save_image(im_logluv, 'my/logluv_image.png')
```

```
from tinycio import fsio, Codec

# Load 4-channel LogLuv image
im_logluv = fsio.load_image('my/logluv_image.png')

# Decode LogLuv - returns 3-channel tensor
im_hdr = Codec.logluv_decode(im_logluv)

# Save 3-channel HDR image
fsio.save_image(im_hdr, 'my/hdr_image.exr')
```

Note

tinycio.Codec doesn't care about your color space and you can feed it any HDR data you like. You can encode a *tinycio.ColorImage*, but the effect will be the same as any other image tensor.

See: *Codec*

3.6 Tone map HDR images

```
from tinycio import ColorImage

im = ColorImage.load('my/hdr_image.tif', 'SRGB_LIN')
im.tone_map('AGX_PUNCHY', target_color_space='SRGB_LIN').save('my/ldr_image.tif')
```

ColorImage handles any necessary color space conversions silently. If you do this manually and step-by-step instead, take note of the inputs expected.

```python
from tinycio import ColorSpace, ToneMapping, fsio

# Load HDR image from disk
im_hdr = fsio.load_image('my/hdr_image.exr')

cs_in = ColorSpace.Variant.SRGB_LIN
cs_tm = ColorSpace.Variant.ACESCG
cs_out = ColorSpace.Variant.SRGB
tm = ToneMapping.Variant.ACESCG

# This tone mapper expects scene-referred ACEScg data
im_ap1 = ColorSpace.convert(im_hdr, cs_in, cs_tm)

# Apply (fitted) ACES RRT+ODT
im_ldr = ToneMapping.apply(im_ap1, tone_mapper=tm)

# Take image to sRGB and apply the gamma curve
im_srgb = ColorSpace.convert(im_ldr, cs_tm, cs_out)

# Save final image as 24-bit sRGB PNG file
fsio.save_image(im_srgb, 'my/ldr_image.png')
```

3.6.1 Comparison

This HDR environment map was put through the different tone mapping options.

3.6.2 Clamp/saturate (no tone mapping)

3.6.3 AgX

3.6.4 AgX Punchy

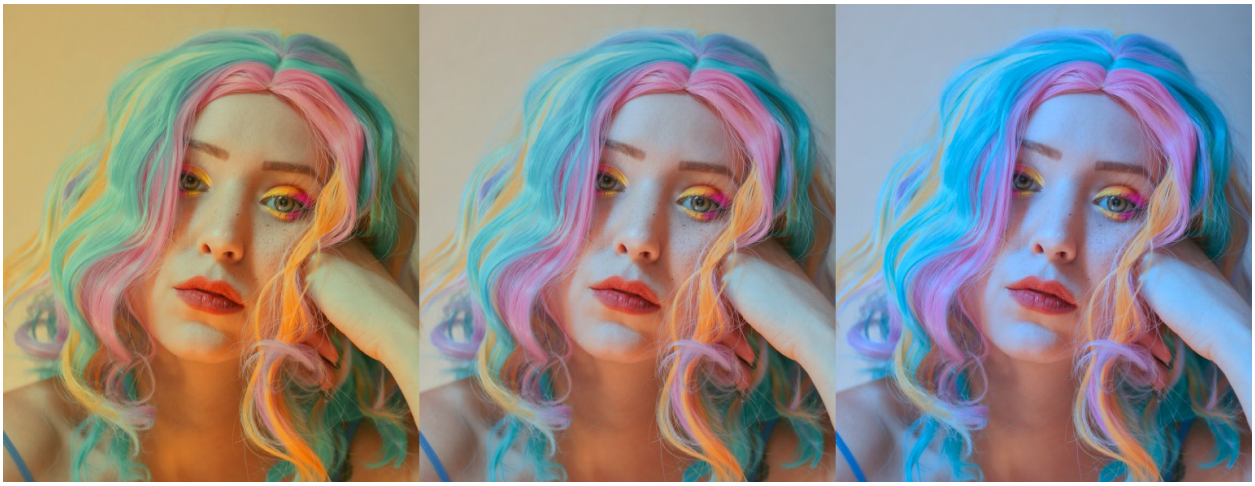
3.6.5 ACEScg (fitted)

3.6.6 Hable

3.6.7 Reinhard (extended)

See: *ColorImage.tone_map()*, *ToneMapping*

3.7 Apply white balance



```
from tinycio import ColorImage

im = ColorImage.load('my/image.tif', 'SRGB_LIN')
im = im.white_balance(source_white='auto', target_white='HORIZON')
im.save('my/new_image.tif', graphics_format='SFLOAT16')
```

Note

For specifying a white point, you can also use a correlated color temperature or *Chromaticity*.

While *ColorImage* will figure out the details automatically, there's a more explicit way to do the same, with a few more steps involved.

```
from tinycio import fsio, WhiteBalance, ColorSpace

cs_lin = ColorSpace.Variant.SRGB_LIN
cs_xyz = ColorSpace.Variant.CIE_XYZ
cs_lms = ColorSpace.Variant.LMS
wp_target = WhiteBalance.Illuminant.HORIZON
fmt_out = fsio.GraphicsFormat.SFLOAT16

# Load the image
im = fsio.load_image('my/image.tif')

# Convert to XYZ for white point estimation
im_xyz = ColorSpace.convert(im, cs_lin, cs_xyz)

# Get approximate source white point
src_white = WhiteBalance.wp_from_image(im_xyz)

# Get xy chromaticity from standard illuminant
tgt_white = WhiteBalance.wp_from_illuminant(wp_target)
```

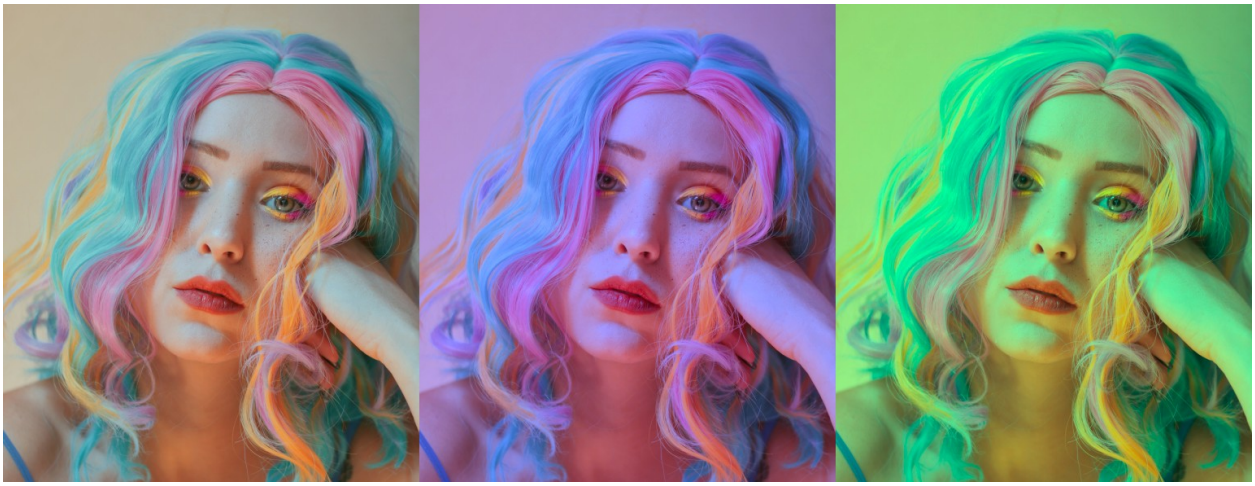
(continues on next page)

(continued from previous page)

```
# Convert image to LMS for white balancing
im_lms = ColorSpace.convert(im, cs_lin, cs_lms)

# Apply
im_lms = WhiteBalance.apply(im_lms, source_white=src_white, target_white=tgt_white)

# Convert back to sRGB linear
im_out = ColorSpace.convert(im, cs_lms, cs_lin)

# Finally save
fsio.save_image(im_out, 'my/new_image.tif', graphics_format=fmt_out)
```

See: *ColorImage.white_balance()*, *WhiteBalance*

3.8 Apply basic color correction



```
from tinycio import ColorImage, ColorCorrection

im = ColorImage.load('my/image.exr', 'SRGB_LIN')
cc = ColorCorrection()
cc.set_contrast(1.2)
cc.set_saturation(0.8)
cc.save('/my/cc_settings.toml') # if needed
im.correct(cc).save('my/corrected_image.exr')
```

Alternatively, without *ColorImage*:

```
from tinycio import ColorSpace, fsio

im_in = fsio.load_image('my/image.exr')
cs_in = ColorSpace.Variant.SRGB_LIN
cs_cc = ColorSpace.Variant.ACESCC
im_cc = ColorSpace.convert(im_in, cs_in, cs_cc)
```

(continues on next page)

(continued from previous page)

```
cc = ColorCorrection()
cc.set_contrast(1.2)
cc.set_saturation(1.2)
im_corrected = cc.apply(im_cc)
fsio.save_image(im_corrected, 'my/corrected_image.exr')
```

Most color correction uses *ACEScc*, so *ColorCorrection.apply()* expects inputs in that color space. *ColorImage.correct()* handles this conversion automatically.

Your instructions to the *ColorCorrection* object will change the state of the instance rather than returning a new one and will all be idempotent (no difference between calling *set_hue_delta()* once or five times). It is recommended to use the setter functions as they perform color value conversions and check value ranges.

Note

The arguments *hue* and *saturation* below are perceptually-linear values (the H and S from the OKHSV color space); *hue delta* has a [-1, +1] range but also operates in OKLAB.

3.8.1 Color filter

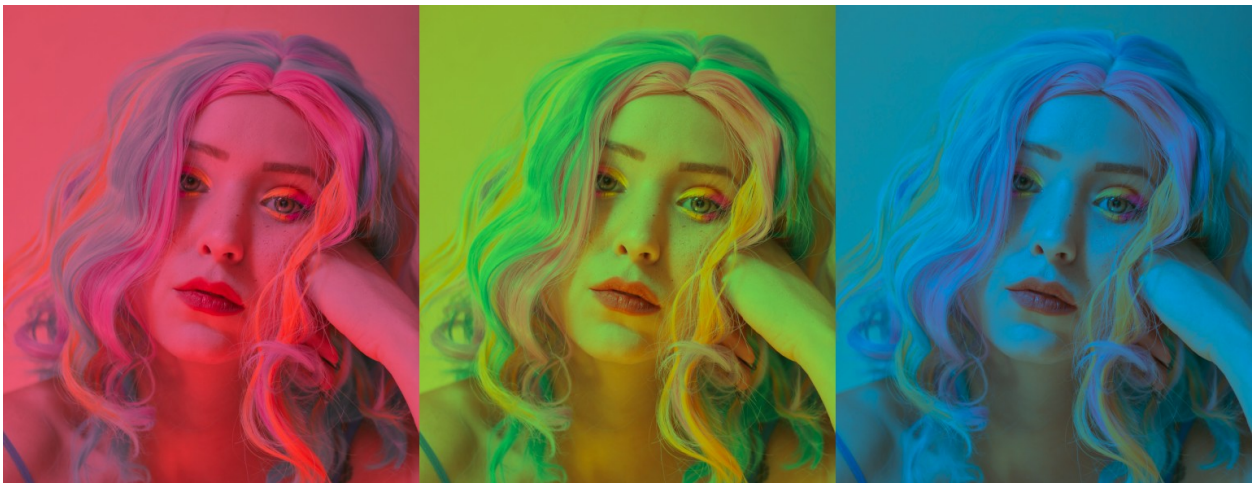


Fig. 3.1: Color filters at: [0, 0.5], [0.33, 0.5], [0.66, 0.5]

```
cc = ColorCorrection()
cc.set_color_filter(hue = 0., saturation = 0.5) # red filter
```

See: *set_color_filter()*

3.8.2 Exposure bias

```
cc = ColorCorrection()
cc.set_exposure_bias(1) # +1 f-stop
```

See: *set_exposure_bias()*

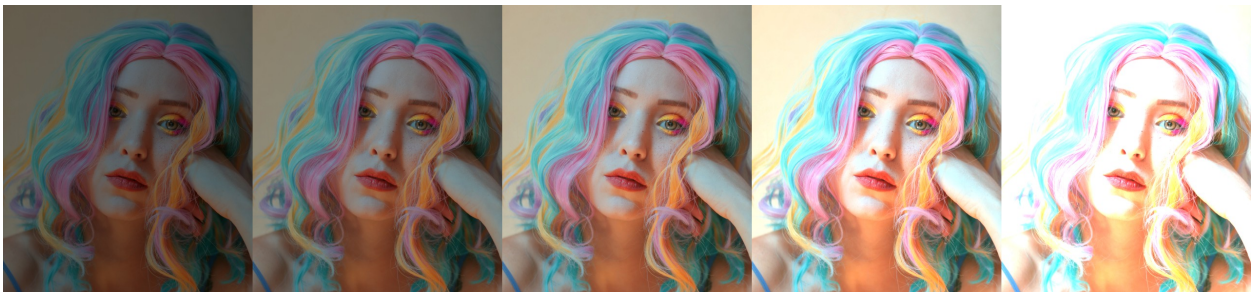


Fig. 3.2: Exposure bias at: -2, -1, 0, +1, +2

3.8.3 Hue delta

Fig. 3.3: Hue delta at: -0.66, -0.33, +0.33, +0.66

```
cc = ColorCorrection()
cc.set_hue_delta(-0.5) # shift hue by -0.5 (-1 and +1 are identical)
```

See: *set_hue_delta()*

3.8.4 Saturation

Fig. 3.4: Saturation at: 0.33, 0.66, 1.33, 1.66

```
cc = ColorCorrection()
cc.set_saturation(1.3) # increase image saturation
```

See: *set_saturation()*

3.8.5 Contrast

Fig. 3.5: Contrast at: 0.5, 1, 1.5, 2

```
cc = ColorCorrection()
cc.set_contrast(1.3) # boost contrast
```

See: *set_contrast()*

3.8.6 Shadows, midtones and highlights

Fig. 3.6: Shadow pulled to blue, highlights to red and vice-versa

```
# Arbitrary values - just for example
cc = ColorCorrection()
cc.set_highlight_offset(-0.02)
cc.set_highlight_color(hue = 0.33, saturation = 0.5)
```

(continues on next page)

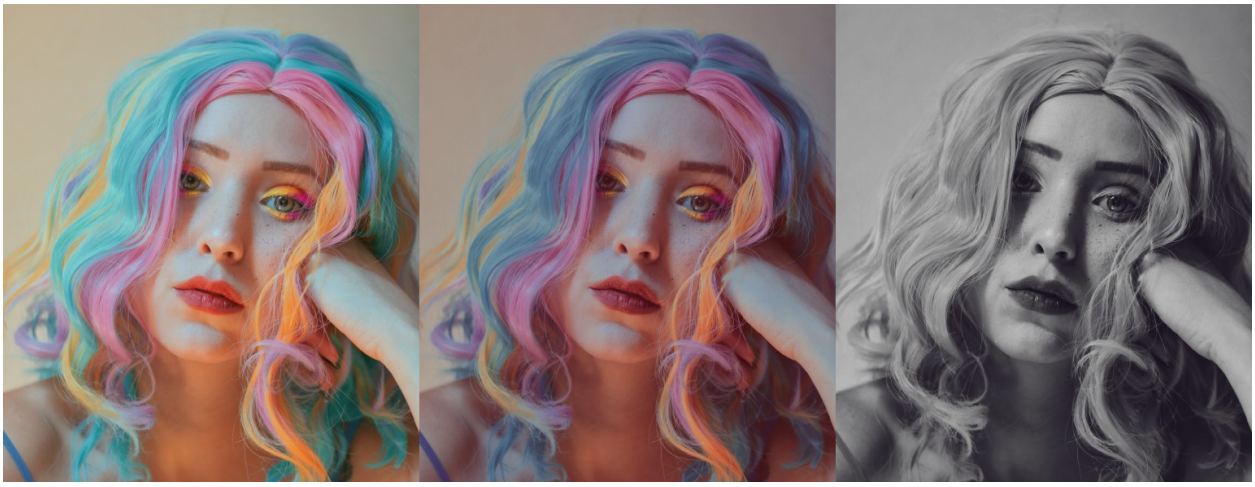
(continued from previous page)

```
cc.set_midtone_offset(0.022)
cc.set_midtone_color(0.66, 0.15)
cc.set_shadow_offset(0.1)
cc.set_shadow_color(0., 0.08)
```

This is a little awkward to describe without a GUI, but if you've ever used the color wheels on common color grading software, you probably know what these settings do.

See: *ColorImage.correct()*, *ColorCorrection*

3.9 Apply LUTs



```
from tinycio import ColorImage

ColorImage.load('my/image.png').lut('my/lut.cube').save('my/new_image.png')
```

If this is for a batch of images, it would be more performant to load and reuse the LUT.

```
from tinycio import LookupTable, ColorImage

mylut = LookupTable.load('my/lut.cube')
im = ColorImage.load('my/image.png')
im.lut(mylut).save('my/new_image.png')
```

If the LUT is meant for a particular log profile, it may be necessary to use a transfer function.

```
from tinycio import LookupTable, ColorImage, TransferFunction

lut = LookupTable.load('my/log_c_lut.cube')

# Load and linearize
im = ColorImage.load('my/image.png', 'SRGB').to_color_space('SRGB_LIN')

# Apply desired curve
im_log_c = TransferFunction.log_c_oetf(im)
```

(continues on next page)

(continued from previous page)

```
# Apply LUT
im_log_c = lut.apply(im_log_c)

# Back to linear (only if needed - a display-ready LUT should have the transform baked-
↪in)
im_out = TransferFunction.log_c_eotf(im)

# Apply sRGB gamma curve (if needed) and save
ColorImage(im_out, 'SRGB_LIN').to_color_space('SRGB').save('my/new_image.png')
```

Although the above hypothetically loads a 24-bit sRGB PNG file, this probably makes more sense to do with uncompressed or minimally-compressed image data, in whatever representation the LUT requires. LUTs have many uses for DCC, but processing raw photographic image data is beyond the scope of this library.

See: *ColorImage.lut()*, *LookupTable*

3.10 Bake color correction to a LUT

```
from tinycio import ColorCorrection,

cc = ColorCorrection()
cc.set_saturation(1.5)
cc.set_contrast(1.2)
cc.set_hue_delta(0.1)
lut = cc.bake_lut()
lut.save('my/lut.cube')
```

Keep in mind that, by default, the LUT will be baked for the color grading color space – *ACEScc*. So, before using that LUT, you would need to put the image in its color space.

```
from tinycio import ColorImage, LookupTable

lut = LookupTable.load('my/lut.cube')
im = ColorImage.load('my/image.exr', 'SRGB_LIN')
im_cc = im.to_color_space('ACESCC')
im_out = im_cc.lut(lut)
im_out.to_color_space('SRGB_LIN').save('my/corrected_image.exr')
```

Alternatively, you can bake the LUT in the color space of your preference:

```
from tinycio import ColorCorrection

cc = ColorCorrection()
# [...apply color correction...]
lut = cc.bake_lut(lut_size=64, lut_color_space='SRGB')
lut.save('my/lut_srgb_lut.cube')
```

See: *ColorCorrection.bake_lut()*

CHAPTER

FOUR

EXAMPLES

4.1 Color sweeps

4.1.1 CCT sweep

```
import torch
from tinycio import Chromaticity, ColorImage, WhiteBalance

res = []
width = 800
for x in range(width):
    cct = x / width * 21000 + 4000
    xy = Chromaticity(WhiteBalance.wp_from_cct(cct))
    col = xy.to_xyz(0.75)
    res.append(col.image())

out = ColorImage(torch.cat(res, dim = 2).repeat(1, 50, 1), 'CIE_XYZ')
out.to_color_space('SRGB').clamp(0., 1.).save('../out/sweep_cct.png')
```

4.1.2 Illuminant sweep

```
import torch
from tinycio import Chromaticity, ColorImage, WhiteBalance

res = []
width = 800
illum = WhiteBalance.Illuminant.TUNGSTEN
illum_xy = Chromaticity(WhiteBalance.wp_from_illuminant(illum))
light_scale, dist_scale = 2000., 1000.
for x in range(width):
    col = illum_xy.to_xyz(light_scale * (1./(((x / width) * dist_scale)**2 + 1.)))
    res.append(col.image())

out = ColorImage(torch.cat(res, dim = 2).repeat(1, 50, 1).clamp(0., 1.), 'CIE_XYZ')
out.to_color_space('SRGB').save('../out/sweep_illuminant.png')
```

4.1.3 Visible spectrum sweep

```
import torch
from tinycio import Color, ColorImage, Spectral

res = []
width = 800
for x in range(width):
    wl = (x / width) * 400 + 380
    res.append(Color(Spectral.wl_to_srgb_linear(wl, normalize=False)).image())

out = ColorImage(torch.cat(res, dim = 2).repeat(1, 50, 1), 'SRGB_LIN').clamp(0., 1.)
out.to_color_space('SRGB').save('../out/sweep_spectrum.png')
```

4.1.4 HSL hue sweep

```
import torch
from tinycio import Color, ColorImage

res = []
width = 800
col = Color(1, 1, 0.5)
for x in range(width):
    col.x = (x / width + 0.5) % 1
    res.append(col.image())

out = ColorImage(torch.cat(res, dim = 2).repeat(1, 50, 1).clamp(0., 1.), 'HSL')
out.to_color_space('SRGB').save('../out/sweep_hue_hsl.png')
```

4.1.5 OKLAB hue sweep

```
import torch
from tinycio import Color, ColorImage
from tinycio.util import apply_hue_oklab

res = []
width = 800
for x in range(width):
    im = Color(1, 0, 0).image('SRGB_LIN').to_color_space('OKLAB')
    res.append(apply_hue_oklab(im, x / width * 2. - 1))

out = ColorImage(torch.cat(res, dim = 2).repeat(1,50,1), 'OKLAB')
out.to_color_space('SRGB').save('../out/sweep_hue_oklab.png')
```

4.2 Lower-level image manipulation

It is generally far better to operate on whole tensors rather than individual pixels, because reading and writing per-pixel or even per-line data is very slow with native Python. But you can make it work in a pinch. If you need actual performance, take a look at Taichi Lang or Numba.

4.2.1 Source image

4.2.2 Line sweep

```
import torch
import time
from tinycio import ColorImage
from tinycio.util import apply_hue_oklab

# Each vertical line of the image hue shifted as a tensor.
```

(continues on next page)

(continued from previous page)

```
res = []
im_in = ColorImage.load('../doc/images/examples_ll/horizon.png', 'SRGB')
start = time.time()
im_in = im_in.to_color_space('OKLAB')
_, H, W = im_in.size()
for x in range(W):
        res.append(apply_hue_oklab(im_in[:, :, x:x+1], x / W * 2. - 1))
im_out = ColorImage(torch.cat(res, dim = 2), 'OKLAB')
im_out = im_out.to_color_space('SRGB').clamp(0., 1.)
end = time.time()
im_out.save('../out/horizon_hue_sweep.png')
print(f'Code execution: {end - start} seconds')
```

Fig. 4.1: ~0.2s, depending on hardware - not great.

4.2.3 Pixel sweep

```
import torch
import time
import numpy as np
from tinycio import Color, ColorImage

# Keying into PyTorch tensors for each pixel is prohibitively expensive.
# We can instead hand it over to NumPy - still slow, but relatively tolerable.
im_in = ColorImage.load('../doc/images/examples_ll/horizon.png', color_space='SRGB')
start = time.time()
im_in = im_in.to_color_space('SRGB_LIN').numpy()
C, H, W = im_in.shape
for y in range(H):
    for x in range(W):
        col = Color(im_in[:, y, x])
        col.r *= 1. - (x / W)
        col.g *= 1. - (y / H)
        im_in[:, y, x] = col.rgb
im_out = ColorImage(im_in, 'SRGB_LIN').to_color_space('SRGB').clamp(0., 1.)
end = time.time()
im_out.save('../out/horizon_rg_sweep.png')
print(f'Code execution: {end - start} seconds')
```

4.2.4 Baseline

```
import torch
import time
from tinycio import ColorImage

im_in = ColorImage.load('../doc/images/examples_ll/horizon.png', color_space='SRGB')
start = time.time()
im_in = im_in.to_color_space('SRGB_LIN')
C, H, W = im_in.shape
xw = torch.linspace(start=1., end=0., steps=W).unsqueeze(0).repeat(H, 1)
yh = torch.linspace(start=1., end=0., steps=H).unsqueeze(-1).repeat(1, W)
im_in[0,...] *= xw
im_in[1,...] *= yh
im_out = ColorImage(im_in, 'SRGB_LIN').to_color_space('SRGB').clamp(0., 1.)
end = time.time()
im_out.save('../out/horizon_rg_torch.png')
print(f'Code execution: {end - start} seconds')
```

Same operation, but with PyTorch functions: ~0.04s

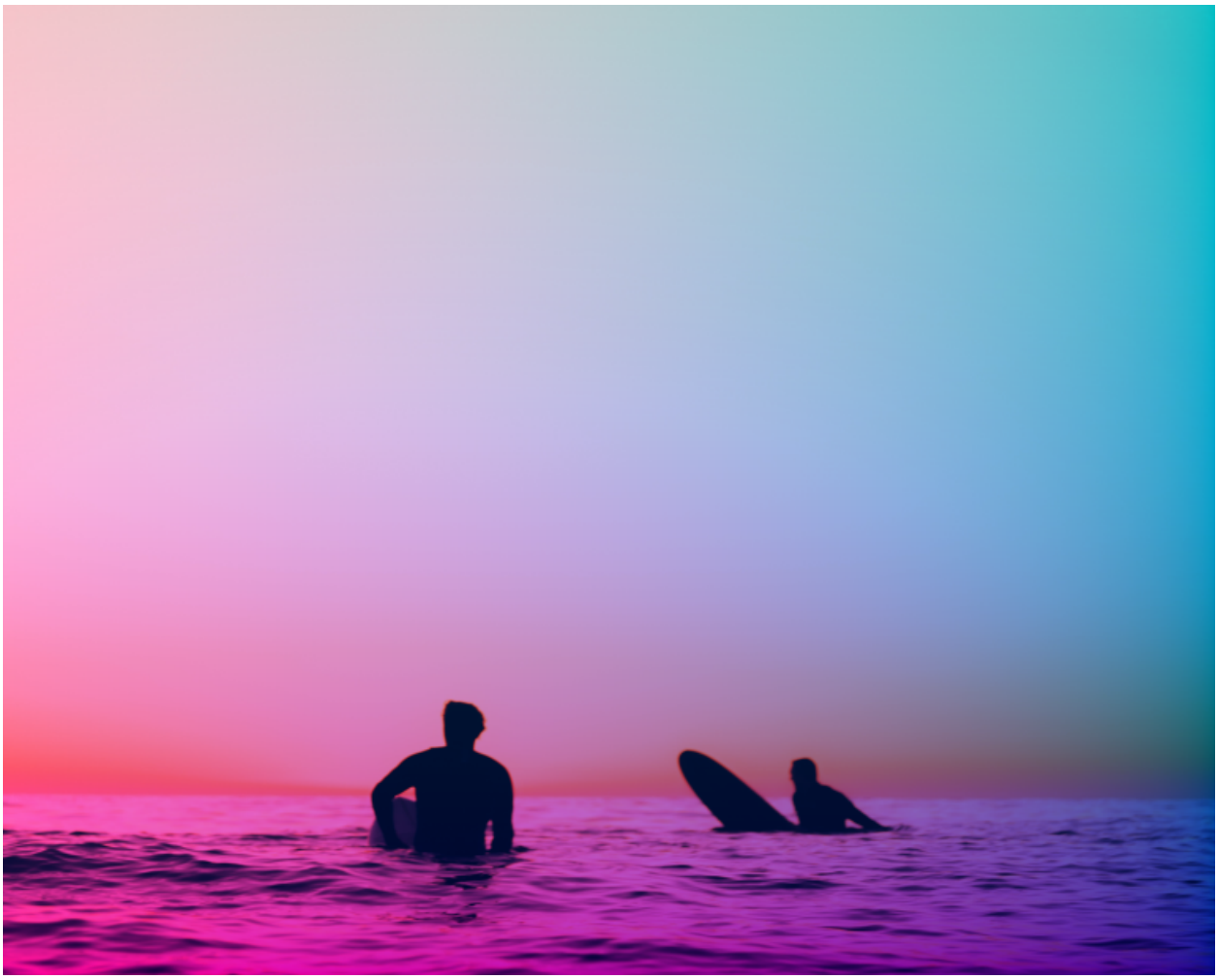


Fig. 4.2: ~3.8s - getting yikesy

CHAPTER

FIVE

DEEPER DIVE

CHAPTER

SIX

REFERENCE

6.1 API Reference

ColorImage(*args, **kwargs)	2D 3-channel color image using float32.
MonoImage(*args, **kwargs)	2D monochromatic image using float32.
Color(*args, **kwargs)	3-component float32 color type.
Chromaticity(*args, **kwargs)	2-component float32 CIE xy chromaticity type.
ColorSpace()	Color space conversion.
TransferFunction()	Opto-electronic/electro-optical transfer functions.
ToneMapping()	Map high-dynamic-range values to low-dynamic-range.
ColorCorrection()	Apply color correction to an image.
LookupTable(size[, lattice, lut_format])	Color lookup table.
WhiteBalance()	Adjust white balance of an image.
Spectral()	Spectral color matching functions.
Codec()	Encoder/decoder for high-dynamic-range data.

class tinycio.**ColorImage**(**args*, ***kwargs*)

Bases: `Tensor`

2D 3-channel color image using float32. Convenience wrapper around a [C=3, H, W] sized PyTorch image tensor. This is the most hands-off abstraction, as it automatically handles color space conversion as needed.

Note

All transformative methods will create a new *ColorImage* instance without altering the current state. PyTorch methods and operations will work as expected but will return with the color space set to UNKNOWN. Exceptions include `.clamp()`, `.lerp()`, `.minimum()`, and `.maximum()`, which will retain the current state for convenience.

Note

Assumes size/shape [C, H, W] for both PyTorch tensor or NumPy array inputs.

Parameters

- **param0** (*torch.Tensor | numpy.ndarray*) – [C=3, H, W] sized tensor/array
- **color_space** (*str | ColorSpace.Variant*) – Color space of the image.

correct(*cc*)

Apply color correction and return as new *ColorImage*.

Parameters
cc (ColorCorrection) – Color correction object.

Returns
New *ColorImage* with color correction applied.

Return type
ColorImage

info(*printout=True*)

Print or return a string describing the properties of the image for debugging.

Parameters
printout – Print string if True, return string if False

Return type
bool | str

static load(*fp*, *color_space=Variant.UNKNOWN*, *graphics_format=GraphicsFormat.UNKNOWN*)

Load image from file

Parameters

- **fp** (*str*) – Image file path.
- **color_space** (*str | ColorSpace.Variant*) – Image color space.
- **graphics_format** (*str | fsio.GraphicsFormat*) – Image graphics format.

Returns
New *ColorImage* loaded from file.

Return type
ColorImage

lut(*lut*, *lut_format=LUTFormat.UNKNOWN*)

Apply a *LookupTable* and return as new *ColorImage*.

Parameters

- **lut** (*str | LookupTable*) – *LookupTable* or LUT file path
- **lut_format** – Format of the LUT, if loading from file.

Returns
New *ColorImage* with LUT applied.

Return type
ColorImage

save(*fp*, *graphics_format=GraphicsFormat.UNKNOWN*)

Save image to file

Warning

This will overwrite existing files.

Parameters

- **fp** (*str*) – Image file path.
- **graphics_format** (*str | fsio.GraphicsFormat*) – Image graphics format.

Returns
True if successful

Return type
bool

to_color_space(*color_space*)

Convert the image to a different color space and return as new *ColorImage*.

Parameters
color_space (*str | ColorSpace.Variant*) – the destination color space

Returns
New *ColorImage* in designated color space.

Return type
ColorImage

tone_map(*tone_mapper*, *target_color_space=Variant.SRGB_LIN*)

Apply tone mapping and return as new *ColorImage*.

Note

Image will be returned and (if the TMO allows) tone mapped in the desired target color space. Any needed color space conversion will be handled automatically.

Parameters

- **tone_mapper** (*str | ToneMapping.Variant*) – Tone mapper to use.
- **target_color_space** (*str | ColorSpace.Variant*) – The desired color space the image should be tone mapped for. Must be a scene-linear RGB color space. The image will be returned in this color space. If the TMO requires a different color space, the image will converted after tone mapping. Some TMOs may have been designed with linear sRGB in mind and may not perform as expected in a different color space.

Returns
New tone mapped *ColorImage*.

Return type
ColorImage

white_balance(*source_white='auto'*, *target_white=Illuminant.NONE*)

White balance the image and return as new *ColorImage*.

Note

Any needed color space conversion will be handled automatically.

Parameters

- **source_white** (*str | Float2 | Chromaticity | int | Illuminant*) – Source white point - can be any of:

- *string* "auto" for automatic (*wp_from_image()*)
- *int* for correlated color temperature (*wp_from_cct()*)
- *Illuminant* or matching *string* (*wp_from_illuminant()*)
- CIE 1931 xy coordinates as *Float2*/*Chromaticity*

target_white (*str* | Float2 | Chromaticity | *int* | Illuminant) – Target white point - can be any of:

- *int* for correlated color temperature (*wp_from_cct()*)
- *Illuminant* or matching *string* (*wp_from_illuminant()*)
- CIE 1931 xy coordinates as *Float2*/*Chromaticity*

Returns
New *ColorImage* with white balance applied.

Return type
ColorImage

class tinycio.**MonoImage**(**args*, ***kwargs*)

Bases: Tensor

2D monochromatic image using float32. Convenience wrapper around a [C=1, H, W] sized PyTorch image tensor. This is a utility class that has no built-in functionality beyond enforcing a single channel for initial inputs. Can be converted to *ColorImage* with `ColorImage(mono_im.repeat(3,1,1))`.

Note

Assumes size/shape [C, H, W] for both PyTorch tensor or NumPy array inputs.

Parameters
param0 (*torch.Tensor* | *numpy.ndarray*) – [C=1, H, W] sized tensor/array

info(*printout=True*)

Print or return a string describing the properties of the image for debugging.

Parameters
printout – Print string if True, return string if False

Return type
bool | str

class tinycio.**Color**(**args*, ***kwargs*)

Bases: *Float3*

3-component float32 color type. Convenience wrapper around *Float3*, which is in turn a shape [3] numpy.ndarray. Example:

```
a = Color(1, 2, 3)
b = Color(torch.tensor([1, 2, 3]))
c = Color(numpy.array([1, 2, 3]))
d = Color(torch.tensor([1, 2, 3]).unsqueeze(-1).unsqueeze(-1))
numpy.array_equal(a,b) & numpy.array_equal(b,c) & numpy.array_equal(c,d) # True
```

Note

This data type is color space agnostic until a *ColorImage* is requested. Swizzling is allowed, but a numeric type will be returned if the number of swizzled components is not 3.

Parameters

- **option1_param0** (*float*) – x/r component
- **option1_param1** (*float*) – y/g component
- **option1_param2** (*float*) – z/b component
- **option2_param0** (*float*) – value of every component
- **option3_param0** (*torch.Tensor | numpy.ndarray | Float3 | list | tuple*) – 3-component tensor, array, list or tuple (a 2D tensor/array image is allowed if the number of pixel values is 1)

convert(*source*, *destination*)

Convert color value from one color space to another.

Parameters

- **source** (*str | ColorSpace.Variant*) – Color space to convert from.
- **destination** (*str | ColorSpace.Variant*) – Color space to convert to.

Return type
Color

image(*color_space=Variant.UNKNOWN*)

Unsqueeze to a [C=3, H, W] sized PyTorch image tensor.

Parameters
color_space (*str | ColorSpace.Variant*) – Color space of the image.

Returns
Expanded [C=3, H=1, W=1] float32 "image" tensor.

Return type
ColorImage

class tinycio.**Chromaticity**(**args*, ***kwargs*)

Bases: *Float2*

2-component float32 CIE xy chromaticity type. Convenience wrapper around *Float2*, which is in turn a shape [2] numpy.ndarray.

Parameters

- **option1_param0** (*float*) – x component
- **option1_param1** (*float*) – y component
- **option2_param0** (*float*) – Value of both components.
- **option3_param0** (*torch.Tensor | numpy.ndarray | Float2*) – 2-component tensor or array. (a 2D tensor/array image is allowed if the number of pixel values is 1)

to_xyy(*luminance=1.0*)

Returns CIE xyY color

Parameters
luminance (*float*) – Y luminance

Returns
CIE xyY color

Return type
Color

to_xyz(*luminance=1.0*)

Returns CIE XYZ color

Parameters
luminance (*float*) – Y luminance

Returns
CIE XYZ color

Return type
Color

class tinycio.**ColorSpace**

Bases: object

Color space conversion. Applies OETFs and EOTFs as needed but omits tonemapping. Cylindrical transformations are treated as distinct color spaces. Example:

```
cs_in = ColorSpace.Variant.SRGB_LIN
cs_out = ColorSpace.Variant.OKLAB
oklab_image = ColorSpace.convert(srgb_image, source=cs_in, destination=cs_out)
```

class Variant(*value*)

Bases: IntEnum

Color space enum. For a list of available options, see *Color spaces*.

classmethod convert(*im*, *source*, *destination*)

Change the color space of an image. Cylindrical transformations HSV/HSL are treated as their own color spaces and assumed to be relative to sRGB linear. Unless otherwise noted or required by specification (e.g. ACES), we assume D65 white point.

Warning

Tone mapping is not automatically included. Thus, converting a wide-gamut color space to one with a narrower gamut, or converting HDR values to a nominally LDR-designated color space will not automatically remap out-of-gamut color to in-gamut or reduce dynamic range. It will likely yield out-of-gamut (positive of negative) or out-of-range values. These values should still, at some point, be tone mapped or clamped to the desired output range.

Warning

Cylindrical transformations (HSL, HSV) should be given input in [0, 1] linear sRGB range (or equivalent). This is not strictly enforced but input outside this range may yield unpredictable results or *NaN* values.

Parameters

- **im** (*torch.Tensor* | ColorImage) – [C=3, H, W] image tensor
- **source** (Variant) – color space to convert from
- **destination** (Variant) – color space to convert to

Returns
image tensor in designated color space

Return type
torch.Tensor

class tinycio.**TransferFunction**

Bases: object

Opto-electronic/electro-optical transfer functions. Example:

```
im_srgb = TransferFunction.srgb_oetf(im_linear)
```

Note

These transfer functions are applied automatically by ColorSpace.convert when appropriate, but can instead be used explicitly.

Note

Out-of-gamut values will be accepted, but the values returned may not be well-defined or meaningful. The goal is only to keep them consistent and out of range.

static **dcip3_eotf**(*im*)

DCI P3 electro-optical transfer function (DCI P3 gamma to linear)

Parameters
im (*torch.Tensor*) – DCI P3 image tensor

Returns
linear P3 gamut image tensor

Return type
torch.Tensor

static **dcip3_oetf**(*im*)

DCI P3 opto-electronic transfer function (linear to DCI P3 gamma)

Parameters
im (*torch.Tensor*) – linear P3 gamut image tensor

Returns
DCI P3 image tensor

Return type
torch.Tensor

`static log_c_eotf(im)`

LogC electro-optical transfer function

Parameters
im (*torch.Tensor*) – LogC encoded image tensor

Returns
linear image tensor

Return type
torch.Tensor

`static log_c_oetf(im)`

LogC opto-electronic transfer function

Parameters
im (*torch.Tensor*) – linear image tensor

Returns
LogC encoded image tensor

Return type
torch.Tensor

`static rec2020_eotf(im)`

Rec. 2020 electro-optical transfer function (Rec. 2020 gamma to linear)

Parameters
im (*torch.Tensor*) – Rec. 2020 image tensor

Returns
linear Rec. 2020 gamut image tensor

Return type
torch.Tensor

`static rec2020_oetf(im)`

Rec. 2020 opto-electronic transfer function (linear to Rec. 2020 gamma)

Parameters
im (*torch.Tensor*) – linear Rec. 2020 gamut image tensor

Returns
Rec. 2020 image tensor

Return type
torch.Tensor

`static rec709_eotf(im)`

Rec. 709 electro-optical transfer function (Rec. 709 gamma to linear sRGB)

Parameters
im (*torch.Tensor*) – Rec. 709 image tensor

Returns
linear sRGB image tensor (same primaries)

Return type
torch.Tensor

static rec709_oetf(*im*)

Rec. 709 opto-electronic transfer function (linear sRGB to Rec. 709 gamma)

Parameters
im (*torch.Tensor*) – linear sRGB image tensor (same primaries)

Returns
Rec. 709 image tensor

Return type
torch.Tensor

static s_log_eotf(*im*)

S-Log electro-optical transfer function

Parameters
im (*torch.Tensor*) – S-Log encoded image tensor

Returns
linear image tensor

Return type
torch.Tensor

static s_log_oetf(*im*)

S-Log opto-electronic transfer function

Parameters
im (*torch.Tensor*) – linear image tensor

Returns
S-Log encoded image tensor

Return type
torch.Tensor

static srgb_eotf(*im*)

sRGB electro-optical transfer function (sRGB gamma to linear sRGB)

Parameters
im (*torch.Tensor*) – sRGB image tensor

Returns
linear sRGB image tensor

Return type
torch.Tensor

static srgb_oetf(*im*)

sRGB opto-electronic transfer function (linear sRGB to sRGB gamma)

Parameters
im (*torch.Tensor*) – linear sRGB image tensor

Returns
sRGB image tensor

Return type
torch.Tensor

class tinycio.**ToneMapping**

Bases: object

Map high-dynamic-range values to low-dynamic-range. LDR is typically sRGB in [0, 1] range. Example:

```
tm = ToneMapping.Variant.HABLE
tonemapped_image = ToneMapping.apply(input_im, tone_mapper=tm)
```

class **Variant**(*value*)

Bases: IntEnum

Tone mapper enum. Available options are:

```
- NONE
- CLAMP
- AGX
- AGX_PUNCHY
- HABLE
- REINHARD
- ACESCG
```

classmethod **apply**(*im*, *tone_mapper*)

Apply tone mapping to HDR image tensor. Input data is expected to be in the correct color space for the chosen tone mapper.

Note

ACESCG tone mapping is performed on AP1 primaries and expects input in the ACESCG color space. All other tone mappers expect SRGB_LIN. The tone_map() method of *ColorImage* handles this conversion automatically.

Parameters

- **im** (*torch.Tensor*) – [C=3, H, W] sized image tensor
- **tone_mapper** (ToneMapping.Variant) – tonemapper to be used

Returns
image tensor

Return type
torch.Tensor

class tinycio.**ColorCorrection**

Bases: object

Apply color correction to an image. Example:

```
cc = ColorCorrection()
cc.set_contrast(1.3)
im_corrected = cc.apply(im_cc)
```

Note

Any *hue* and *saturation* parameters use perceptually linear values of the OKHSV color space.

apply(*im*)

Apply color correction to image tensor in ACEScc color space.

Parameters
im (*torch.Tensor*) – Image tensor sized [C=3, H, W] in ACEScc color space.

Returns
Color corrected image tensor.

Return type
torch.Tensor

bake_lut(*lut_size=64*, *lut_color_space=Variant.ACESCC*)

Bake color correction to a CUBE LUT.

Note

Regardless of working color space, the LUT will be limited to a [0, 1] range of values.

Parameters

- **lut_size** (*int*) – Size of the LUT. Range [4, 512].
- **lut_color_space** (*str | Variant*) – The color space that the LUT should be baked for, as string or IntEnum.

Returns
The baked LUT.

Return type
LookupTable

fit_to_image(*im_source*, *im_target*, *steps=500*, *learning_rate=0.003*, *strength=1.0*, *allow_hue_shift=False*, *fit_height=512*, *fit_width=512*, *device=None*, *context=None*)

Perform gradient descent on the color correction settings, so that the appearance of the source image matches the target.

Note

Images need to be in *ACEScc* color space.

Parameters

- **im_source** (*torch.Tensor | ColorImage*) – Source image tensor in *ACEScc* color space. Values must be in range [0, 1].
- **im_target** (*torch.Tensor | ColorImage*) – Target image tensor in *ACEScc* color space.
- **steps** (*int*) – Number of optimization steps.
- **learning_rate** (*float*) – Learning rate for gradient descent.
- **strength** (*float*) – Strength of the effect in range [0, 1].
- **allow_hue_shift** (*bool*) – Allow the optimizer to shift the hue.
- **fit_height** (*int*) – Image tensors will be interpolated to this height for evaluation.

- **fit_width** (*int*) – Image tensors will be interpolated to this width for evaluation.
- **context** (*callable*) – The Python context to use (see *util.progress_bar()*).
- **device** (*str*) – Device for gradient descent (if None will use input tensor device).

Returns
True when completed

Return type
ColorCorrection

info(*printout=True*)

Print or return a string describing the current color correction settings.

Parameters
printout (*bool*) – Print string if True, return string if False

Return type
bool | str

classmethod load(*fp*, *force=False*)

Load color correction settings from *toml* file.

Parameters

- **fp** (*str*) – File path of *toml* file to be loaded.
- **force** (*bool*) – If set to True, ignores version mismatch forces loading the file.

Returns
ColorCorrection object with settings loaded.

Return type
ColorCorrection

save(*fp*)

Save color correction settings to a *toml* file.

Parameters
fp (*str*) – Output file path of *toml* file to be saved.

Returns
True if successful.

Return type
bool

Warning

This will overwrite existing files.

set_color_filter(*hue=0.0*, *saturation=0.0*)

Set a color filter. Idempotent assignment.

Parameters

- **hue** (*float*) – Filter hue - perceptually linear (the H from OKHSV). Range [0, 1].
- **saturation** (*float*) – Filter saturation - perceptually linear (the S from OKHSV). Range [0, 1].

Returns
True if successful

Return type
bool

`set_contrast`(*contrast=1.0*)

Set the contrast. Idempotent assignment.

Parameters
contrast (*float*) – Contrast. Range [0, 4].

Returns
True if successful

Return type
bool

`set_exposure_bias`(*exposure_bias=0.0*)

Set the exposure bias. Idempotent assignment.

Parameters
exposure_bias (*float*) – Exposure bias in f-stops. Range [-5, 5].

Returns
True if successful

Return type
bool

`set_highlight_color`(*hue=0.0*, *saturation=0.0*)

Set the highlight color. Uses OKHSV model - value fixed at 1. Idempotent assignment.

Parameters

- **hue** (*float*) – Highlight hue in range [0, 1] - perceptually linear (the H from OKHSV)
- **saturation** (*float*) – Highlight saturation in range [0, 1] - perceptually linear (the S from OKHSV)

Returns
True if successful

Return type
bool

`set_highlight_offset`(*offset=0.0*)

Set the shadow offset in range [-2, 2]. Idempotent assignment.

Parameters
offset (*float*) – Highlight offset.

Returns
True if successful

Return type
bool

`set_hue_delta`(*hue_delta=0.0*)

Set the hue delta, shifting an image's hues. Idempotent assignment.

Parameters
hue_delta (*float*) – Amount of hue shift - perceptually linear. Range [-1, 1].

Returns
True if successful

Return type
bool

`set_midtone_color`(*hue=0.0*, *saturation=0.0*)

Set the midtone color. Uses OKHSV model - value fixed at 1. Idempotent assignment.

Parameters

- **hue** (*float*) – Midtone hue in range [0, 1] - perceptually linear (the H from OKHSV)
- **saturation** (*float*) – Midtone saturation in range [0, 1] - perceptually linear (the S from OKHSV)

Returns
True if successful

Return type
bool

`set_midtone_offset`(*offset=0.0*)

Set the shadow offset in range [-2, 2]. Idempotent assignment.

Parameters
offset (*float*) – Midtone offset.

Returns
True if successful

Return type
bool

`set_saturation`(*saturation=1.0*)

Set the color saturation. Idempotent assignment.

Parameters
saturation (*float*) – Amount of saturation. Range [0, 4].

Returns
True if successful

Return type
bool

`set_shadow_color`(*hue=0.0*, *saturation=0.0*)

Set the shadow color. Uses OKHSV model - value fixed at 1. Idempotent assignment.

Parameters

- **hue** (*float*) – Shadow hue in range [0, 1] - perceptually linear (the H from OKHSV)
- **saturation** (*float*) – Shadow saturation in range [0, 1] - perceptually linear (the S from OKHSV)

Returns
True if successful

Return type
bool

set_shadow_offset(*offset=0.0*)

Set the shadow offset in range [-2, 2]. Idempotent assignment.

Parameters
offset (*float*) – Shadow offset.

Returns
True if successful

Return type
bool

class tinycio.**LookupTable**(*size*, *lattice=None*, *lut_format=LUTFormat.CUBE_3D*)

Bases: `object`

Color lookup table. Example:

```
lut = LookupTable.get_negative()
im_negative = lut.apply(im)
```

Parameters

- **size** (*int*) – Size of the LUT.
- **lattice** (*torch.Tensor*) – Lattice as tensor (defaults to linear).
- **lut_format** (*LUTFormat*) – Format of the LUT.

apply(*im*)

Apply LUT to image tensor.

Parameters
im (*torch.Tensor | ColorImage*) – Input image tensor

Returns
Image tensor with LUT applied

Return type
torch.Tensor

fit_to_image(*im_source*, *im_target*, *steps=500*, *learning_rate=0.003*, *strength=1.0*, *fit_height=512*, *fit_width=512*, *device='cuda'*, *context=None*)

Perform gradient descent on the lattice, so that the appearance of the source image matches the target.

Parameters

- **im_source** (*torch.Tensor | ColorImage*) – Source image tensor. Values must be in range [0, 1].
- **im_target** (*torch.Tensor | ColorImage*) – Target image tensor.
- **steps** (*int*) – Number of optimization steps.
- **learning_rate** (*float*) – Learning rate for gradient descent.
- **strength** (*float*) – Strength of the effect in range [0, 1].
- **fit_height** (*int*) – Image tensors will be interpolated to this height for evaluation.
- **fit_width** (*int*) – Image tensors will be interpolated to this width for evaluation.
- **device** (*Union[str, None]*) – Device for gradient descent (if None will use input tensor device).

- **context** (*callable*)

Returns
True when completed

Return type
bool

classmethod get_empty(*size=32*, *lut_format=LUTFormat.CUBE_3D*)

Returns empty LUT. All values mapped to 0.

Parameters

- **size** (*int*) – Size of the LUT.
- **lut_format** (*LUTFormat*) – Format of the LUT.

Return type
LookupTable

classmethod get_linear(*size=32*, *lut_format=LUTFormat.CUBE_3D*)

Returns linear LUT. Has no effect: when applied, output matches input ([0, 1] range).

Parameters

- **size** (*int*) – Size of the LUT.
- **lut_format** (*LUTFormat*) – Format of the LUT.

Return type
LookupTable

classmethod get_negative(*size=32*, *lut_format=LUTFormat.CUBE_3D*)

Returns negative LUT. Output is inverted ([0, 1] range).

Parameters

- **size** (*int*) – Size of the LUT.
- **lut_format** (*LUTFormat*) – Format of the LUT.

Return type
LookupTable

classmethod get_random(*size=32*, *lut_format=LUTFormat.CUBE_3D*)

Returns random LUT. Everything mapped to random values ([0, 1] range).

Parameters

- **size** (*int*) – Size of the LUT.
- **lut_format** (*LUTFormat*) – Format of the LUT.

Return type
LookupTable

classmethod load(*fp*, *lut_format=LUTFormat.UNKNOWN*)

Load LUT from file.

Parameters

- **fp** (*str*) – File path.
- **lut_format** (*LUTFormat*) – Format of the LUT.

Return type
LookupTable

save(*fp*, *lut_format=LUTFormat.UNKNOWN*)

Save LUT to file.

Warning

This will overwrite existing files.

Parameters

- **fp** (*str*) – File path.
- **lut_format** (*LUTFormat*) – Format of the LUT.

class tinycio.**WhiteBalance**

Bases: object

Adjust white balance of an image. Example:

```
source_white = WhiteBalance.wp_from_image(input_image)
target_white = WhiteBalance.wp_from_illuminant(WhiteBalance.Illuminant.NORTH_SKY)
white_balanced_image = WhiteBalance.apply(input_image, source_white, target_white)
```

class Illuminant(*value*)

Bases: IntEnum

CIE 1931 2° standard illuminant. Available options are:

```
- NONE
- A (INCANDESCENT, TUNGSTEN)
- D50 (HORIZON)
- D55 (MIDMORNING)
- D65 (DAYLIGHT_NOON)
- D75 (NORTH SKY)
- D93 (BLUE_PHOSPHOR)
- E (EQUAL_ENERGY)
- F1 (FLUORESCENT_DAYLIGHT1)
- F2 (FLUORESCENT_COOL_WHITE)
- F3 (FLUORESCENT_WHITE)
- F4 (FLUORESCENT_WARM_WHITE)
- F5 (FLUORESCENT_DAYLIGHT2)
- F6 (FLUORESCENT_LIGHT_WHITE)
- F7 (D65_SIMULATOR, DAYLIGHT_SIMULATOR)
- F8 (D50_SIMULATOR, SYLVANIA_F40)
- F9 (FLOURESCENT_COOL_WHITE_DELUXE)
- F10 (PHILIPS_TL85, ULTRALUME_50)
- F11 (PHILIPS_TL84, ULTRALUME_40)
- F12 (PHILIPS_TL83, ULTRALUME_30)
- LED_B1 (PHOSPHOR_CONVERTED_BLUE1)
- LED_B2 (PHOSPHOR_CONVERTED_BLUE2)
- LED_B3 (PHOSPHOR_CONVERTED_BLUE3)
- LED_B4 (PHOSPHOR_CONVERTED_BLUE4)
```

(continues on next page)

(continued from previous page)

```
- LED_B5 (PHOSPHOR_CONVERTED_BLUE5)
- LED_BH1
- LED_RGB1
- LED_V1 (LED_VIOLET1)
- LED_V2 (LED_VIOLET2)
```

static apply(*im_lms*, *source_white*, *target_white*)

Apply white balance.

Parameters

- **im_lms** (*torch.Tensor*) – Image tensor in LMS color space
- **source_white** (*torch.Tensor*) – Source white point in LMS space
- **target_white** (*torch.Tensor*) – Target white point in LMS space

Returns
White balanced image tensor

Return type
torch.Tensor

static wp_from_cct(*cct*)

Compute CIE xy chromaticity coordinates (white point) from correlated color temperature.

Parameters
cct (*int*) – Correlated color temperature in range [4000, 25000]

Returns
White point coordinates (CIE xy)

Return type
Float2

classmethod wp_from_illuminant(*illuminant*)

Look up chromaticity coordinates (white point) of a CIE 1931 2° standard illuminant.

Parameters
illuminant (Illuminant) – Standard illuminant

Returns
White point coordinates (CIE xy)

Return type
Float2

static wp_from_image(*im_xyz*)

Estimate the dominant illuminant of an environment map or a target image directly and return its approximate CIE xy chromaticity coordinates (white point).

Warning

This is a lazy method that just averages the pixels in the image tensor. There is no spherical mapping, nor PCA, nor any serious attempt to analyze the image.

Parameters
im_xyz (*torch.Tensor* | ColorImage) – Image tensor in CIE XYZ color space

Returns
Estimated white point coordinates (CIE xy)

Return type
Float2

class tinycio.**Spectral**

Bases: object

Spectral color matching functions. Example:

```
xyz_col = Spectral.wl_to_xyz(550)
```

classmethod cm_table(*normalize_xyz=False*)

Returns color matching table as tensor of size [81, 3]. The table contains CIE XYZ values, arranged by wavelength, in increments of 5nm, from 380nm to 780nm.

Parameters
normalize_xyz (*bool*) – Normalize XYZ color, such that X=X/(X+Y+Z), etc

Returns
Color matching function table

Return type
torch.Tensor

classmethod wl_to_srgb(*wl*, *normalize=False*, *lum_scale=0.25*)

Wavelength (nm) to normalized, approximate sRGB color, clamped to [0, 1] range, with sRGB gamma curve.

Note

Wolfram Alpha doesn't quite agree, but it's rather close. This produces a plausible-looking spectrum, but there is probably some missing (pre?) normalization step. Take the RGB outputs with a grain of salt.

Parameters

- **wl** (*float*) – wavelength in nm
- **normalize** (*bool*) – normalize sRGB color
- **lum_scale** (*float*)

Returns
sRGB linear color

Return type
Float3

classmethod wl_to_srgb_linear(*wl*, *normalize=False*, *lum_scale=0.25*)

Wavelength (nm) to normalized, approximate linear sRGB color, clamped to [0, 1] range.

Parameters

- **wl** (*float*) – wavelength in nm
- **normalize** (*bool*) – normalize sRGB color
- **lum_scale** (*float*)

Returns
sRGB linear color

Return type
Float3

classmethod wl_to_xyz(*wl*)

Wavelength (nm) to CIE XYZ color, linearly interpolated. Precision is limited to interpolated 5nm increments.

Note

These coordinates will often fall outside the sRGB gamut. Direct color space conversion will yield invalid sRGB values.

Parameters
wl (*float*) – wavelength in nm

Returns
CIE XYZ color

Return type
Float3

class tinycio.**Codec**

Bases: object

Encoder/decoder for high-dynamic-range data. Example:

```
encoded_image = Codec.logluv_encode(hdr_rgb_image)
```

classmethod logluv_decode(*im*)

Decode LOGLUV to HDR floating point RGB data.

Note

4 channels in, 3 channels out.

Parameters
im (*torch.Tensor*) – LOGLUV encoded [C=4, H, W] sized image tensor (4 channels)

Returns
Decoded HDR image tensor (3 channels)

Return type
torch.Tensor

classmethod logluv_encode(*im*)

Encode HDR floating point RGB data to LOGLUV for 32bpp (RGBA) image file.

Note

3 channels in, 4 channels out.

Parameters
im (*torch.Tensor* | ColorImage) – HDR [C=3, H, W] sized image tensor

Returns
LOGLUV encoded image tensor (4 channels)

Return type
torch.Tensor

6.1.1 fsio module

File system input and output.

GraphicsFormat(value)	The graphics format of an image file to be saved or loaded.
LUTFormat(value)	Lookup table format.
load_image(fp[, graphics_format])	Load image file as [C, H, W] float32 PyTorch image tensor.
save_image(im, fp[, graphics_format])	Save [C, H, W] float32 PyTorch image tensor as image file.
load_lut(fp[, lut_format])	Load LUT from file.
save_lut(lattice, fp[, lut_format])	Save LUT to a file.
truncate_image(im)	Returns the first three color channels of a multi-channel image.

6.1.2 numerics module

Warning

The FloatX/IntX types here are Python wrappers around numpy arrays. They are glacially slow and should only be used for convenience inside of Python scope and well outside of any high-traffic loops.

The numerics module offers shading-language-like syntax for float and int vector data types, as well as a few utility functions. These types are an extension of numpy arrays, and you can use them as such. Matrices are not natively supported.

```
from tinycio.numerics import *
import numpy as np
import torch

# Several possible inputs
Float4(4,3,2,1)             # Float4([4., 3., 2., 1.])
Float4(1.23)                # Float4([1.23, 1.23, 1.23, 1.23])
Float4([4,3,2,1])           # Float4([4., 3., 2., 1.])
Float4((4,3,2,1))           # Float4([4., 3., 2., 1.])
Float4(np.array([3,2,1,0])) # Float4([3., 2., 1., 0.])
Float4(torch.rand(4))       # Float4([0.22407186, 0.26193792, 0.89055574, 0.57386285])
Float4(torch.rand(4,1,1))   # Float4([0.99545109, 0.46160549, 0.78145427, 0.02302521])

# Swizzling
foo = Float3(1,2,3)
foo.y                       # 2.0 (float)
```

(continues on next page)

(continued from previous page)

```
foo.rg                      # Float2([1., 2.])
foo.bgr                     # Float3([3., 2., 1.])
foo.xxyx                    # Float4([1., 1., 2., 1.])

# Utility functions
bar = Float3.y_axis()       # Float3([0., 1., 0.])
bar.list()                  # [0., 1., 0.]
bar.tuple()                 # (0., 1., 0.)
lerp(foo.bbb, bar, 0.5)     # Float3([1.5, 2. , 1.5])
saturate(Float2(-2,5))      # Float2([0., 1.])
sign(Float2(-2,5))          # Float2([-1.,  1.])
Float4(1) == Float4.one()   # Float4([ True,  True,  True,  True])
```

Float2(*args)	Float2 type using numpy.ndarray.
Float3(*args)	Float3 type using numpy.ndarray.
Float4(*args)	Float4 type using numpy.ndarray.
Int2(*args)	Int2 type using numpy.ndarray.
Int3(*args)	Int3 type using numpy.ndarray.
Int4(*args)	Int4 type using numpy.ndarray.
lerp(a, b, w)	Linearly interpolate two values.
saturate(x)	Saturate x, such that x is clamped to range [0, 1].
sign(x)	Sign of x as 1, -1 or 0.
normalize(v)	Normalize vector v to a unit vector.
reflect(n, l)	Reflect vector l over n.

class tinycio.numerics.**Float2**(**args*)

Bases: ndarray

Float2 type using numpy.ndarray.

list()

Returns values as Python list

Return type
list

static one()

Returns numeric type filled with one values

tuple()

Returns values as Python tuple

Return type
tuple

static x_axis()

Returns numeric type with x-axis set to 1 and all others to 0

static y_axis()

Returns numeric type with y-axis set to 1 and all others to 0

static zero()

Returns numeric type filled with zero values

class tinycio.numerics.**Float3**(**args*)

Bases: ndarray

Float3 type using numpy.ndarray.

list()

Returns values as Python list

Return type
list

static one()

Returns numeric type filled with one values

tuple()

Returns values as Python tuple

Return type
tuple

static x_axis()

Returns numeric type with x-axis set to 1 and all others to 0

static y_axis()

Returns numeric type with y-axis set to 1 and all others to 0

static z_axis()

Returns numeric type with z-axis set to 1 and all others to 0

static zero()

Returns numeric type filled with zero values

class tinycio.numerics.**Float4**(**args*)

Bases: ndarray

Float4 type using numpy.ndarray.

list()

Returns values as Python list

Return type
list

static one()

Returns numeric type filled with one values

tuple()

Returns values as Python tuple

Return type
tuple

static x_axis()

Returns numeric type with x-axis set to 1 and all others to 0

static y_axis()

Returns numeric type with y-axis set to 1 and all others to 0

static z_axis()

Returns numeric type with z-axis set to 1 and all others to 0

static zero()

Returns numeric type filled with zero values

class tinycio.numerics.Int2(*args)

Bases: ndarray

Int2 type using numpy.ndarray.

list()

Returns values as Python list

Return type
list

tuple()

Returns values as Python tuple

Return type
tuple

class tinycio.numerics.Int3(*args)

Bases: ndarray

Int3 type using numpy.ndarray.

list()

Returns values as Python list

Return type
list

tuple()

Returns values as Python tuple

Return type
tuple

class tinycio.numerics.Int4(*args)

Bases: ndarray

Int4 type using numpy.ndarray.

list()

Returns values as Python list

Return type
list

tuple()

Returns values as Python tuple

Return type
tuple

tinycio.numerics.lerp(*a*, *b*, *w*)

Linearly interpolate two values.

Parameters

- **a** (*float | list | numpy.ndarray | torch.Tensor*) – first value or list to interpolate

- **b** (*float | list | numpy.ndarray | torch.Tensor*) – second value or list to interpolate
- **w** (*float | list | numpy.ndarray | torch.Tensor*) – 0-1 weight ratio of first to second value

Returns
linearly interpolated value(s)

Return type
float | list | numpy.ndarray | torch.Tensor

tinycio.numerics.**saturate**(*x*)

Saturate x, such that x is clamped to range [0, 1].

Parameters
x (*float | list | numpy.ndarray | torch.Tensor*) – input value

Returns
saturated input value(s)

Return type
float | list | numpy.ndarray | torch.Tensor

tinycio.numerics.**sign**(*x*)

Sign of x as 1, -1 or 0. Returns:

- 0 if/where x == 0
- 1 if/where x > 0
- -1 if/where x < 0

Parameters
x (*float | list | numpy.ndarray | torch.Tensor*) – input value

Returns
sign(s) of input values

Return type
float | list | numpy.ndarray | torch.Tensor

tinycio.numerics.**normalize**(*v*)

Normalize vector v to a unit vector.

Parameters
v (*numpy.ndarray | torch.Tensor*) – input vector

Returns
normalized vector

Return type
numpy.ndarray | torch.Tensor

tinycio.numerics.**reflect**(*n*, *l*)

Reflect vector l over n.

Parameters

- **l** (*numpy.ndarray or torch.Tensor*) – input "light" vector
- **n** (*numpy.ndarray | torch.Tensor*) – input normal vector

Returns
reflected vector

Return type
numpy.ndarray | torch.Tensor

6.1.3 util module

Warning

Using the color functions here directly is not advised, if an interface exists in the root module.

version()	Get current tinycio version.
version_check_minor(ver_str)	Verify tinycio version.
remap(x, from_start, from_end, to_start, to_end)	Linearly remap scalar or tensor.
remap_to_01(x, start, end)	Remap value to [0, 1] range.
remap_from_01(x, start, end)	Remap [0, 1] value back to specified range.
smoothstep(edge0, edge1, x)	Smooth Hermite interpolation between 0 and 1.
softsign(x)	Smooth nonlinearity.
fract(x)	Get the fractional part of input (x - floor(x)).
serialize_tensor(val)	Convert a tensor into a float or list of floats.
trilinear_interpolation(im_3d, indices)	Interpolate 3D image tensor.
fitted_polynomial_curve_6th_order(x, fit)	Evaluate 6th-order polynomial curve.
fitted_polynomial_curve_7th_order(x, fit)	Evaluate 7th-order polynomial curve.
progress_bar()	Context to display a progressbar with tqdm.
apply_gamma(im, gamma)	Apply arbitrary gamma correction.
apply_hue_oklab(im_oklab, hue_delta)	Manually shift hue by a -1 to +1 delta value.
srgb_luminance(im_srgb)	Return relative luminance of linear sRGB image.
col_hsv_to_rgb(hsv)	Convert HSV color to RGB.
col_rgb_to_hsv(rgb)	Convert RGB color to HSV.
col_oklab_to_linear_srgb(lab)	Convert OKLAB color to linear sRGB.
col_linear_srgb_to_oklab(srgb)	Convert linear sRGB color to OKLAB.
col_okhsv_to_srgb(hsv)	Convert OKHSV color to linear sRGB.
col_srgb_to_okhsv(srgb)	Convert linear sRGB color to OKHSV.
col_okhsl_to_srgb(hsl)	Convert OKHSL color to linear sRGB.
col_srgb_to_okhsl(srgb)	Convert linear sRGB color to OKHSL.
asc_cdl(im[, slope, offset, power, eps])	Apply ASC DCL
lgg(im[, lift, gamma, gain, eps])	Apply Lift/Gamma/Gain
xy_to_XYZ(xy[, luminance])	Convert xy chromaticity to CIE XYZ.
xyz_mat_from_primaries(rgb, wp)	Get XYZ transform for arbitrary RGB primaries.
mat_von_kries_cat(src_wp_xy, dst_wp_xy, ...)	Compute simple raw LMS Von Kries CAT matrix.

tinycio.util.**version**()

Get current tinycio version. :return: version string (“major.minor.patch”)

Return type
str

tinycio.util.**version_check_minor**(*ver_str*)

Verify tinycio version. Check if the major and minor version of *ver_str* matches current.

Parameters
ver_str (*str*) – Version string to compare

Returns
True if major.minor match, else False

Return type
bool

tinycio.util.**remap**(*x*, *from_start*, *from_end*, *to_start*, *to_end*)

Linearly remap scalar or tensor.

Parameters

- **x** (*float | torch.Tensor*) – Input value or tensor
- **from_start** (*float*) – Start of input range
- **from_end** (*float*) – End of input range
- **to_start** (*float*) – Start of target range
- **to_end** (*float*) – End of target range

Returns
Remapped and clamped value

Return type
float | torch.Tensor

tinycio.util.**remap_to_01**(*x*, *start*, *end*)

Remap value to [0, 1] range.

Parameters

- **x** (*float | torch.Tensor*) – Input value or tensor
- **start** (*float*) – Start of original range
- **end** (*float*) – End of original range

Returns
Normalized value clamped to [0, 1]

Return type
float | torch.Tensor

tinycio.util.**remap_from_01**(*x*, *start*, *end*)

Remap [0, 1] value back to specified range.

Parameters

- **x** (*float | torch.Tensor*) – Normalized value or tensor
- **start** (*float*) – Target range start
- **end** (*float*) – Target range end

Returns
Rescaled value clamped to [start, end]

Return type
float | torch.Tensor

tinycio.util.**smoothstep**(*edge0*, *edge1*, *x*)

Smooth Hermite interpolation between 0 and 1. For x in [edge0, edge1].

Parameters

- **edge0** (*float*) – Lower bound of transition
- **edge1** (*float*) – Upper bound of transition
- **x** (*torch.Tensor*) – Input tensor

Returns
Smoothly interpolated tensor

Return type
torch.Tensor

tinycio.util.**softsign**(*x*)

Smooth nonlinearity. x / (1 + |x|), useful for range compression.

Parameters
x (*float | torch.Tensor*) – Input scalar or tensor

Returns
Softsign result

Return type
float | torch.Tensor

tinycio.util.**fract**(*x*)

Get the fractional part of input (x - floor(x)).

Parameters
x (*float | torch.Tensor*) – Input scalar or tensor

Returns
Fractional part

Return type
float | torch.Tensor

tinycio.util.**serialize_tensor**(*val*)

Convert a tensor into a float or list of floats.

Parameters
val (*torch.Tensor*) – Tensor to serialize

Returns
Scalar if 1-element tensor, else flattened list

Return type
Union[float, List[float]]

tinycio.util.**trilinear_interpolation**(*im_3d*, *indices*)

Interpolate 3D image tensor.

Parameters

- **im_3d** (*torch.Tensor*) – Input 3D image tensor of shape (C, D, H, W).
- **indices** (*Union[ColorImage, torch.Tensor]*) – Indices into the tensor.

Returns
Interpolated color values.

Return type
torch.Tensor

tinycio.util.**fitted_polynomial_curve_6th_order**(*x*, *fit*)

Evaluate 6th-order polynomial curve.

Parameters

- **x** (*torch.Tensor*) – Input tensor
- **fit** (*torch.Tensor*) – Coefficient tensor of shape [7]

Returns
Evaluated tensor

Return type
torch.Tensor

tinycio.util.**fitted_polynomial_curve_7th_order**(*x*, *fit*)

Evaluate 7th-order polynomial curve.

Parameters

- **x** (*torch.Tensor*) – Input tensor
- **fit** (*torch.Tensor*) – Coefficient tensor of shape [8]

Returns
Evaluated tensor

Return type
torch.Tensor

tinycio.util.**progress_bar**()

Context to display a progressbar with tqdm.

tinycio.util.**apply_gamma**(*im*, *gamma*)

Apply arbitrary gamma correction.

Parameters

- **im** (*torch.Tensor | ColorImage*) – Image tensor
- **gamma** (*float*) – Gamma correction (should be in the range [0.1, 10.0])

Returns
Image tensor with correction applied

Return type
torch.Tensor

tinycio.util.**apply_hue_oklab**(*im_oklab*, *hue_delta*)

Manually shift hue by a -1 to +1 delta value.

Parameters

- **im_oklab** (*torch.Tensor | ColorImage*) – Image tensor in OKLAB color space
- **hue_delta** (*float*) – Hue shift value in the range [-1., 1.]

Returns
Image tensor in OKLAB color space with adjusted hue

Return type
torch.Tensor

tinycio.util.**srgb_luminance**(*im_srgb*)

Return relative luminance of linear sRGB image.

Parameters
im_srgb (*torch.Tensor* | ColorImage) – [C=3, H, W] color image tensor in sRGB color space

Returns
[C=1, H, W] image tensor

Return type
torch.Tensor

tinycio.util.**col_hsv_to_rgb**(*hsv*)

Convert HSV color to RGB.

Parameters
hsv (Float3 | Color) – HSV color

Return type
Float3

tinycio.util.**col_rgb_to_hsv**(*rgb*)

Convert RGB color to HSV.

Parameters
rgb (Float3 | Color) – RGB color

Return type
Float3

tinycio.util.**col_oklab_to_linear_srgb**(*lab*)

Convert OKLAB color to linear sRGB.

Parameters
lab (Float3 | Color) – OKLAB color

Return type
Float3

tinycio.util.**col_linear_srgb_to_oklab**(*srgb*)

Convert linear sRGB color to OKLAB.

Parameters
srgb (Float3 | Color) – linear sRGB color

Return type
Float3

tinycio.util.**col_okhsv_to_srgb**(*hsv*)

Convert OKHSV color to linear sRGB.

Parameters
hsv (Float3 | Color) – OKHSV color

Return type
Float3

tinycio.util.**col_srgb_to_okhsv**(*srgb*)

Convert linear sRGB color to OKHSV.

Parameters
srgb (Float3 | Color) – linear sRGB color

Return type
Float3

tinycio.util.**col_okhsl_to_srgb**(*hsl*)

Convert OKHSL color to linear sRGB.

Parameters
hsl (Float3 | Color) – OKHSL color

Return type
Float3

tinycio.util.**col_srgb_to_okhsl**(*srgb*)

Convert linear sRGB color to OKHSL.

Parameters
srgb (Float3 | Color) – linear sRGB color

Return type
Float3

tinycio.util.**asc_cdl**(*im*, *slope=1.0*, *offset=0.0*, *power=1.0*, *eps=1e-07*)

Apply ASC DCL

Parameters

- **im** (*torch.Tensor* | ColorImage) – Image tensor
- **slope** (*float*) – slope
- **offset** (*float*) – offset
- **power** (*float*) – power
- **eps** (*float*) – epsilon

Returns
Image tensor with correction applied

tinycio.util.**lgg**(*im*, *lift=1.0*, *gamma=1.0*, *gain=1.0*, *eps=1e-07*)

Apply Lift/Gamma/Gain

Parameters

- **im** (*torch.Tensor* | ColorImage) – Image tensor
- **lift** (*float*) – slope
- **gamma** – offset
- **gain** (*float*) – power
- **eps** (*float*) – epsilon

Returns
Image tensor with correction applied

tinycio.util.**xy_to_XYZ**(*xy*, *luminance=1.0*)

Convert xy chromaticity to CIE XYZ. (CIE 1931 2°)

Parameters

- **xy** (*tuple* | Float2 | Chromaticity) – xy chromaticity

- **luminance** (*float*) – Y luminance

Returns
CIE XYZ color

Return type
tuple

tinycio.util.**xyz_mat_from_primaries**(*rgb*, *wp*)

Get XYZ transform for arbitrary RGB primaries. (CIE 1931 2°)

Parameters

- **rgb** (*list[tuple | Float2 | Chromaticity]*) – List of RGB chromaticities
- **wp** (*tuple | Float2 | Chromaticity*) – white point chromaticity

Returns
3×3 RGB to XYZ matrix

Return type
numpy.ndarray

tinycio.util.**mat_von_kries_cat**(*src_wp_xy*, *dst_wp_xy*, *cs_to_lms*, *lms_to_cs*)

Compute simple raw LMS Von Kries CAT matrix.

Parameters

- **src_wp_xy** (*list[tuple | Float2 | Chromaticity]*) – Source white point
- **dst_wp_xy** (*list[tuple | Float2 | Chromaticity]*) – Destination/target white point
- **cs_to_lms** (*numpy.ndarray*) – 3×3 to-LMS matrix
- **lms_to_cs** (*numpy.ndarray*) – 3×3 from-LMS matrix

Returns
3×3 chromatic adaptation transform matrix

Return type
numpy.ndarray

6.2 Color spaces and graphics formats

6.2.1 Color spaces

Table 6.5: Available color spaces

Identifier	Description
UNKNOWN	no color space specified - flag for “take a guess”
NONCOLOR	non-color data or color space not applicable
CIE_XYZ	CIE 1931 XYZ color space
CIE_XYY	xyY color space derived from above
SRGB	sRGB with gamma curve
SRGB_LIN	scene-linear sRGB/Rec. 709 - no gamma curve
REC709	almost the same as sRGB, different gamma
REC2020	a wide-gamut RGB color space
REC2020_LIN	scene-linear Rec. 2020
DCI_P3	a wide-gamut RGB color space
DCI_P3_LIN	scene-linear DCI-P3
DISPLAY_P3	some apple DCI P3 clone with different gamma, because apple
ACESCG	AP1 primaries; scene-linear, less suitable for color grading
ACESCC	AP1 primaries; logarithmic, more suitable for color grading
ACESCCT[1]	AP1 primaries; logarithmic, more suitable for color grading
ACES2065_1[2]	AP0 primaries; scene-linear, core ACES color space
LMS	“long, medium, short” / tristimulus
OKLAB	perceptual; Björn Ottosson’s improvement over CIELAB
CIELAB	perceptual; for “characterization of colored surfaces and dyes”
CIELUV[3]	perceptual; for “characterization of color displays”
HSV	“hue, saturation, value”; assumed sRGB-relative
HSL	“hue, saturation, lightness”; assumed sRGB-relative
OKHSV[4]	perceptually linear HSV based on OKLAB
OKHSL[4]	perceptually linear HSL based on OKLAB

6.2.2 Graphics formats

Table 6.6: Available graphics formats

Identifier	Description
UNKNOWN	Unknown; “take a guess”
UINT8	Unsigned 8-bit integers
UINT16	Unsigned 16-bit integers
UINT32	Unsigned 32-bit integers
SFLOAT16	Signed 16-bit floats
SFLOAT32	Signed 32-bit floats
UNORM8	Normalized unsigned 8-bit integers
UNORM16	Normalized unsigned 16-bit integers
UNORM32	Normalized unsigned 32-bit integers

[1] not yet implemented
[2] scene-linear but still bad for rendering
[3] currently disabled
[4] intended for color picking; not implemented for image tensor conversion

6.3 Release notes

v 0.8.0 a - May 2025

- Reworked util module
- Reworked docs
- Some other minor changes and fixes
- Updated dependencies

v 0.7.2 a - Apr. 2024

- Fixed ACES RRT/ODT

v 0.7.1 a - Apr. 2024

- Misc doc and script corrections

v 0.7.0 a - Apr. 2024

- Added `target_color_space` argument to *ColorImage* tone mapping. Image will be returned and, if possible, tone mapped in the desired color space.
- Added ACEScc color space to `tcio-color2color` script

v 0.6.3 a - Mar. 2024

- Minor correction to sRGB transfer functions (insignificant for 8-bit values)
- Added luminance scaling for spectral wavelength-to-sRGB functions

v 0.6.2 a - Dec. 2023

- Fixed color space conversion bugs resulting from clipping values
- Fixed color correction bug: midtone saturation was being assigned wrong values
- Added better transforms for REC2020/REC2020_LIN and DCI_P3/DCI_P3_LIN - fixed DCI chromatic adaptation

v 0.6.1 a - Dec. 2023

- Corrected linear color space definitions for scripts
- Updated numerics unit test

v 0.6.0 a - Dec. 2023

- **API change**: simplified `GraphicsFormat` (options now just UINT8, SFLOAT32, UNORM8, etc).
- Added denormalization/normalization for UNORM image loading/saving.
- API extension: extended numeric types to accept and return lists and tuples.
- Moved scripts to src and added them to setuptools config. Scripts can now be run with:
  - `tcio-color2color`
  - `tcio-hdr-codec`
  - `tcio-img2cube`

    - `tcio-white-balance`

- Added `tcio-setup` script. Currently just an alias for `imageio_download_bin freeimage`.

v 0.5.4 a - Dec. 2023

- Fixed tonemapping color space conversion bug with *`ColorImage`*.

v 0.5.3 a - Dec. 2023

- Updated examples to reflect API changes.
- Fixed potential problem with ColorImage prematurely losing state.
- Updated docs.

v 0.5.2 a - Dec. 2023

- Very minor corrections and fixes, mostly to docs and examples.

v 0.5.1 a - Dec. 2023

- just docs and webhosting housekeeping

v 0.5.0 a - Dec. 2023

- *gestures at the docs*

Fin.

6.4 Modules

The list of user-facing modules is just what you see here and in *API Reference*.

Note

The internal modules are not meant to be directly imported, unless you are modifying or developing the library. As they are not part of the user API, their names may change at any time. Import from their root module name instead - e.g.

- `from tinycio.fsio import GraphicsFormat` - like this
- `from tinycio.fsio.format import GraphicsFormat` - not like this

6.5 License

```
MIT License

Copyright (c) 2023 Sam Izdat

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
```

(continues on next page)

(continued from previous page)

```
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.
```

CHAPTER

SEVEN

SCRIPTS

7.1 Color to color

Image-to-image color space conversion and tone mapping.

Summary:

```
usage: tcio-color2color [-h] [--tmcs ] [--igf ] [--ogf ] [--tonemapper ] [--keep-alpha]
                        input input-color-space output output-color-space

Convert the color space an image, with an optional tone mapping stage.

positional arguments:
  input                 Input image file path
  input-color-space     Input color space
                        CHOICES:
                            cie_xyz, cie_xyy, srgb, srgb_lin,
                            rec709, rec2020, rec_2020_lin,
                            dci_p3, dci_p3_lin, display_p3,
                            acescg, acescc, aces2065_1,
                            lms, hsl, hsv, oklab, cielab
  output                Output image file path
  output-color-space    Output color space
                        CHOICES: [same as above]

optional arguments:
  -h, --help            show this help message and exit
  --tmcs []             Tone mapping color space (default: srgb_lin)
                        CHOICES:
                            srgb_lin, rec2020_lin
                            dci_p3_lin, acescg, aces2065_1
  --igf []              Input graphics format (default: unknown)
                        CHOICES:
                            sfloat16, sfloat32
                            uint8, uint16, uint32
  --ogf []              Output graphics format (default: unknown)
                        CHOICES: [same as above]
  --tonemapper [], -t []
                        Tone mapper (default: none)
                        CHOICES:
                            none, clamp, agx, agx_punchy
```

(continues on next page)

(continued from previous page)

```
                            acescg, hable, reinhard
  --keep-alpha, -a      Preserve alpha channel
```

Example usage:

```
$ tcio-color2color -t agx input.tif cie_xyz output.png srgb
```

More explicit example:

```
$ tcio-color2color --igf uint8 --ogf sfloat16 --keep-alpha input.png srgb output.tif␣
↪aces2065_1
```

Also, note that the ACEScg RRT+ODT will not be applied automatically, so in this case "acescg" needs to be repeated - once for the "output color space" and once for the "tonemapper" option:

```
$ tcio-color2color -t acescg input.exr acescg output.png srgb
```

Script:

```python
#!/usr/bin/env python
"""Convert the color space an image, with an optional tone mapping stage."""
import os
import argparse
from argparse import RawTextHelpFormatter
import torch
from tinycio import ColorImage, ColorSpace, ToneMapping, fsio

def main_cli():
    parser = argparse.ArgumentParser(description=__doc__, formatter_class=argparse.
↪RawTextHelpFormatter)
    parser.add_argument('input', type=str, help='Input image file path')
    parser.add_argument(
        'ics',
        type=str,
        choices=[
            'cie_xyz', 'cie_xyy', 'srgb', 'srgb_lin',
            'rec709', 'rec2020', 'rec2020_lin',
            'dci_p3', 'dci_p3_lin', 'display_p3',
            'acescg', 'aces2065_1', 'acescc', 'lms',
            'hsl', 'hsv', 'oklab', 'cielab'],
        metavar='input-color-space', # because bad formatting is bad :(
        help='Input color space\n' + \
            'CHOICES:\n' + \
            '    cie_xyz, cie_xyy, srgb, srgb_lin, \n' + \
            '    rec709, rec2020, rec_2020_lin, \n' + \
            '    dci_p3, dci_p3_lin, display_p3, \n' + \
            '    acescg, acescc, aces2065_1, \n' + \
            '    lms, hsl, hsv, oklab, cielab'
        )
    parser.add_argument('output', type=str, help='Output image file path')
    parser.add_argument(
```

(continues on next page)

(continued from previous page)

```
        'ocs',
        type=str,
        choices=[
            'cie_xyz', 'cie_xyy', 'srgb', 'srgb_lin',
            'rec709', 'rec2020', 'rec2020_lin',
            'dci_p3', 'dci_p3_lin', 'display_p3',
            'acescg', 'aces2065_1', 'acescc', 'lms',
            'hsl', 'hsv', 'oklab', 'cielab'],
        metavar='output-color-space',
        help='Output color space\n' + \
            'CHOICES: [same as above]'
        )
    parser.add_argument(
        '--tmcs',
        type=str,
        default="srgb_lin",
        const="srgb_lin",
        nargs='?',
        choices=['srgb_lin', 'rec2020_lin', 'dci_p3_lin', 'acescg', 'aces2065_1'],
        metavar='',
        help='Tone mapping color space (default: %(default)s)\n' + \
            'CHOICES:\n' + \
            '    srgb_lin, rec2020_lin \n' + \
            '    dci_p3_lin, acescg, aces2065_1 \n'
        )
    parser.add_argument(
        '--igf',
        type=str,
        default="unknown",
        const="unknown",
        nargs='?',
        choices=['sfloat16','sfloat32','uint8','uint16','uint32'],
        metavar='',
        help='Input graphics format (default: %(default)s)\n' + \
            'CHOICES:\n' + \
            '    sfloat16, sfloat32 \n' + \
            '    uint8, uint16, uint32 \n'
        )
    parser.add_argument(
        '--ogf',
        type=str,
        default="unknown",
        const="unknown",
        nargs='?',
        choices=['sfloat16','sfloat32','uint8','uint16','uint32'],
        metavar='',
        help='Output graphics format (default: %(default)s)\n' + \
            'CHOICES: [same as above]'
        )
    parser.add_argument(
        '--tonemapper',
        '-t',
```

(continues on next page)

(continued from previous page)

```
        type=str,
        default="none",
        const="unknown",
        nargs='?',
        choices=[
            'none', 'clamp', 'agx', 'agx_punchy',
            'acescg', 'hable', 'reinhard'],
        metavar='',
        help='Tone mapper (default: %(default)s)\n' + \
            'CHOICES:\n' + \
            '    none, clamp, agx, agx_punchy \n' + \
            '    acescg, hable, reinhard'
        )
    parser.add_argument('--keep-alpha', '-a', action='store_true', help='Preserve alpha␣
↪channel')
    args = parser.parse_args()

    fp_in   = os.path.realpath(args.input)
    fp_out  = os.path.realpath(args.output)
    gf_in   = args.igf.strip().upper()
    gf_out  = args.ogf.strip().upper()
    cs_in   = args.ics.strip().upper()
    cs_out  = args.ocs.strip().upper()
    tm      = args.tonemapper.strip().upper()
    alpha   = args.keep_alpha

    try:
        im, ac = None, None
        if alpha:
            im = fsio.load_image(fp_in, graphics_format=fsio.GraphicsFormat[gf_in])
            ac = im[3:4,...] if im.size(0) == 4 else None
            im = ColorImage(fsio.truncate_image(im), color_space=cs_in)
        else:
            im = ColorImage.load(fp_in, color_space=cs_in, graphics_format=gf_in)

        im = im.tone_map(tm, target_color_space=args.tmcs) # it's okay to pass "NONE"
        im = im.to_color_space(cs_out)
        if ac is not None: im = torch.cat([im, ac], dim=0)
        im.save(fp_out, gf_out)
        print(f'saved image to: {os.path.realpath(fp_out)}')
    except Exception as e:
        print(f'cannot convert: {e}')

if __name__ == '__main__':
    main_cli()
```

7.2 HDR Codec

Encode and decode HDR color data to and from low-bit-depth RGBA files.

Summary:

```
usage: tcio-hdr-codec [-h] (--encode | --decode) [--igf ] [--ogf ] [--format {logluv}]␣
↪input output

Encode and decode high dynamic range color data.

positional arguments:
  input               Input image file path
  output              Output image file path

optional arguments:
  -h, --help          show this help message and exit
  --encode, -e        Encode mode
  --decode, -d        Decode mode
  --igf []            Input graphics format (default: unknown)
                      CHOICES:
                          sfloat16, sfloat32
                          uint8, uint16, uint32
  --ogf []            Output graphics format (default: unknown)
                      CHOICES: [same as above]
  --format {logluv}   Format
```

Example usage:

```
$ tcio-hdr-codec --encode hdr_image.exr encoded_image.png
```

```
$ tcio-hdr-codec --decode encoded_image.png hdr_image.exr
```

Script:

```python
#!/usr/bin/env python
"""Encode and decode high dynamic range color data."""
import os
import argparse
from argparse import RawTextHelpFormatter
from tinycio import fsio, Codec

def main_cli():
    parser = argparse.ArgumentParser(description=__doc__, formatter_class=argparse.
↪RawTextHelpFormatter)
    parser.add_argument('input', type=str, help='Input image file path')
    parser.add_argument('output', type=str, help='Output image file path')
    mode = parser.add_mutually_exclusive_group(required=True)
    mode.add_argument('--encode', '-e', action='store_true', help='Encode mode')
    mode.add_argument('--decode', '-d', action='store_true', help='Decode mode')
    parser.add_argument(
        '--igf',
        type=str,
        default="unknown",
        const="unknown",
        nargs='?',
```

(continues on next page)

(continued from previous page)

```
        choices=['sfloat16','sfloat32','uint8','uint16','uint32'],
        metavar='',
        help='Input graphics format (default: %(default)s)\n' + \
            'CHOICES:\n' + \
            '    sfloat16, sfloat32 \n' + \
            '    uint8, uint16, uint32 \n'
        )
    parser.add_argument(
        '--ogf',
        type=str,
        default="unknown",
        const="unknown",
        nargs='?',
        choices=['sfloat16','sfloat32','uint8','uint16','uint32'],
        metavar='',
        help='Output graphics format (default: %(default)s)\n' + \
            'CHOICES: [same as above]'
        )
    parser.add_argument(
        '--format',
        type=str,
        choices=['logluv'],
        default='logluv',
        help='Format'
        )
    args = parser.parse_args()

    fp_in   = os.path.realpath(args.input)
    fp_out  = os.path.realpath(args.output)
    gf_in   = args.igf.strip().upper()
    gf_out  = args.ogf.strip().upper()

    try:
        if args.format == 'logluv':
            if args.encode:
                im_hdr = fsio.load_image(fp_in, graphics_format=fsio.GraphicsFormat[gf_
↪in])
                im_hdr = fsio.truncate_image(im_hdr)
                im_logluv = Codec.logluv_encode(im_hdr)
                fsio.save_image(im_logluv, fp_out)
            elif args.decode:
                im_logluv = fsio.load_image(fp_in, graphics_format=fsio.
↪GraphicsFormat[gf_in])
                im_hdr = Codec.logluv_decode(im_logluv)
                fsio.save_image(im_hdr, fp_out)
            else:
                raise Exception('unexpected mode')
        else:
            raise Exception('unexpected format')
        print(f'saved image to: {fp_out}')
    except Exception as e:
        print(f'cannot encode/decode: {e}')
```

(continues on next page)

(continued from previous page)

```
if __name__ == '__main__':
    main_cli()
```

7.3 White balance

White balance an image.

Summary:

```
usage: tcio-white-balance [-h] --source-white SOURCE_WHITE --target-white TARGET_WHITE
                          [--color-space] [--igf ] [--ogf ]
                          input output

White balance an image.

positional arguments:
  input                 Input image file path
  output                Output image file path

optional arguments:
  -h, --help            show this help message and exit
  --source-white SOURCE_WHITE, -s SOURCE_WHITE
                        Source white point - can be:
                         - string "auto"
                         - name of standard illuminant as e.g. "DAYLIGHT_NOON"
                         - correlated color temperature as e.g. 4600
                         - chromaticity xy as e.g. "0.123, 0.321"
  --target-white TARGET_WHITE, -t TARGET_WHITE
                        Target white point [as above, except no "auto"]
  --color-space , -c    Color space of input and output (default: srgb)
                        CHOICES:
                            cie_xyz, cie_xyy, srgb, srgb_lin
                            rec709, rec2020, rec2020_lin,
                            dci_p3, dci_p3_lin, display_p3,
                            acescg, aces2065_1, lms, hsl, hsv
                            oklab, cielab
  --igf []              Input graphics format (default: unknown)
                        CHOICES:
                            sfloat16, sfloat32
                            uint8, uint16, uint32
  --ogf []              Output graphics format (default: unknown)
                        CHOICES: [same as above]
```

Example usage:

```
$ tcio-white-balance -s auto -t EQUAL_ENERGY -c srgb_linear env_map.exr env_map_bal.exr
```

Script:

```
#!/usr/bin/env python
"""White balance an image."""
import os
import argparse
from argparse import RawTextHelpFormatter
from tinycio import ColorImage, Chromaticity

def main_cli():
    parser = argparse.ArgumentParser(description=__doc__, formatter_class=argparse.
↪RawTextHelpFormatter)
    parser.add_argument('input', type=str, help='Input image file path')
    parser.add_argument('output', type=str, help='Output image file path')
    parser.add_argument(
        '--source-white',
        '-s',
        required=True,
        type=str,
        help='Source white point - can be:\n' + \
            ' - string "auto"\n' + \
            ' - name of standard illuminant as e.g. "DAYLIGHT_NOON"\n' + \
            ' - correlated color temperature as e.g. 4600\n' + \
            ' - chromaticity xy as e.g. "0.123, 0.321"\n'
        )
    parser.add_argument(
        '--target-white',
        '-t',
        required=True,
        type=str,
        help='Target white point [as above, except no "auto"]'
        )
    parser.add_argument(
        '--color-space',
        '-c',
        type=str,
        default='srgb',
        metavar='',
        choices=[
            'cie_xyz', 'cie_xyy', 'srgb', 'srgb_lin',
            'rec709', 'rec2020', 'rec2020_lin',
            'dci_p3', 'dci_p3_lin', 'display_p3',
            'acescg', 'aces2065_1', 'lms', 'hsl', 'hsv',
            'oklab', 'cielab'],
        help='Input color space\n' + \
            'CHOICES:\n' + \
            '    cie_xyz, cie_xyy, srgb, srgb_lin, \n' + \
            '    rec709, rec2020, rec_2020_lin, \n' + \
            '    dci_p3, dci_p3_lin, display_p3, \n' + \
            '    acescg, aces2065_1, lms, hsl, hsv \n' + \
            '    oklab, cielab'
        )
    parser.add_argument(
```

(continues on next page)

(continued from previous page)

```
        '--igf',
        type=str,
        default='unknown',
        const='unknown',
        nargs='?',
        choices=['sfloat16','sfloat32','uint8','uint16','uint32'],
        metavar='',
        help='Input graphics format (default: %(default)s)\n' + \
            'CHOICES:\n' + \
            '    sfloat16, sfloat32 \n' + \
            '    uint8, uint16, uint32 \n'
        )
    parser.add_argument(
        '--ogf',
        type=str,
        default='unknown',
        const='unknown',
        nargs='?',
        choices=['sfloat16','sfloat32','uint8','uint16','uint32'],
        metavar='',
        help='Output graphics format (default: %(default)s)\n' + \
            'CHOICES: [same as above]'
        )
    args = parser.parse_args()

    fp_in       = os.path.realpath(args.input)
    fp_out      = os.path.realpath(args.output)
    cs          = args.color_space.strip().upper()
    gf_in       = args.igf.strip().upper()
    gf_out      = args.ogf.strip().upper()

    src_white   = None
    tgt_white   = None

    try:
        if "," in args.source_white:
            src_white = args.source_white.split(',')
            if len(src_white) == 2:
                src_white = Chromaticity(float(src_white[0]), float(src_white[1]))
            else:
                raise Exception('script could not interpret source white')
        else:
            src_white = args.source_white.strip()
            if src_white.isnumeric(): src_white = int(src_white)

        if "," in args.target_white:
            tgt_white = args.target_white.split(',')
            if len(tgt_white) == 2:
                tgt_white = Chromaticity(float(tgt_white[0]), float(tgt_white[1]))
            else:
                raise Exception('script could not interpret target white')
        else:
```

(continues on next page)

(continued from previous page)

```
            tgt_white = args.target_white.strip()
            if tgt_white.isnumeric(): tgt_white = int(tgt_white)

        im = ColorImage.load(fp_in, color_space=cs, graphics_format=gf_in)
        im = im.white_balance(src_white, tgt_white)
        im.save(fp_out, graphics_format=gf_out)
    except Exception as e:
        print(f'cannot apply: {e}')

if __name__ == '__main__':
    main_cli()
```

7.4 Image to CUBE LUT

Create a color grading LUT by aligning the appearance of a source image to that of a target image.

Summary:

```
usage: tcio-img2cube [-h] [--save-image SAVE_IMAGE] [--save-lut SAVE_LUT]
                     [--size SIZE] [--steps STEPS] [--learning-rate LEARNING_RATE]
                     [--strength STRENGTH] [--empty-lut] [--igfs ] [--igft ]
                     [--ogf ] [--device ]
                     source target

Apply an automatic color grade to an image and/or generate a color grading CUBE LUT
by aligning the look of a source image to that of a target image.

positional arguments:
  source                Source image file path
  target                Target image file path

optional arguments:
  -h, --help            show this help message and exit
  --save-image SAVE_IMAGE, -i SAVE_IMAGE
                        Output image file path
  --save-lut SAVE_LUT, -l SAVE_LUT
                        Output LUT file path
  --size SIZE, -s SIZE  LUT size (range [0, 128]) (default: 64)
  --steps STEPS, -t STEPS
                        Steps (range [0, 10000]) (default: 1000)
  --learning-rate LEARNING_RATE, -r LEARNING_RATE
                        Learning rate (range [0, 1]) (default: 0.003)
  --strength STRENGTH   Strength of the effect (range [0, 1]) (default: 1.0)
  --empty-lut           Initialize empty LUT (instead of linear)
  --igfs []             Source image graphics format (default: unknown)
                        CHOICES:
                            sfloat16, sfloat32
                            uint8, uint16, uint32
  --igft []             Target image graphics format (default: unknown)
                        CHOICES: [same as above]
  --ogf []              Output image graphics format (default: unknown)
```

(continues on next page)

(continued from previous page)

```
                        CHOICES: [same as above]
  --device []           Device for gradient descent (default: cuda)
```

Example usage:

```
$ tcio-img2cube --t 500 --save-image out.png source.png target.png
```

Script:

```
#!/usr/bin/env python
"""
Apply an automatic color grade to an image and/or generate a color grading CUBE LUT
by aligning the look of a source image to that of a target image.
"""
import os
import argparse
from argparse import RawTextHelpFormatter
from tinycio import fsio, LookupTable
from tinycio.util import progress_bar

def main_cli():
    parser = argparse.ArgumentParser(description=__doc__, formatter_class=argparse.
↪RawTextHelpFormatter)
    parser.add_argument('source', type=str, help='Source image file path')
    parser.add_argument('target', type=str, help='Target image file path')
    parser.add_argument('--save-image', '-i', type=str, help='Output image file path')
    parser.add_argument('--save-lut', '-l', type=str, help='Output LUT file path')
    parser.add_argument('--size', '-s',
        type=int,
        default=64,
        help='LUT size (range [0, 128]) (default: %(default)s)')
    parser.add_argument('--steps', '-t',
        type=int,
        default=1000,
        help='Steps (range [0, 10000]) (default: %(default)s)')
    parser.add_argument('--learning-rate', '-r',
        type=float,
        default=0.003,
        help='Learning rate (range [0, 1]) (default: %(default)s)')
    parser.add_argument('--strength',
        type=float,
        default=1.,
        help='Strength of the effect (range [0, 1]) (default: %(default)s)')
    parser.add_argument('--empty-lut', action='store_true', help='Initialize empty LUT␣
↪(instead of linear)')
    parser.add_argument(
        '--igfs',
        type=str,
        default="unknown",
        const="unknown",
        nargs='?',
```

(continues on next page)

(continued from previous page)

```
        choices=['sfloat16','sfloat32','uint8','uint16','uint32'],
        metavar='',
        help='Source image graphics format (default: %(default)s)\n' + \
            'CHOICES:\n' + \
            '    sfloat16, sfloat32 \n' + \
            '    uint8, uint16, uint32 \n'
        )
    parser.add_argument(
        '--igft',
        type=str,
        default="unknown",
        const="unknown",
        nargs='?',
        choices=['sfloat16','sfloat32','uint8','uint16','uint32'],
        metavar='',
        help='Target image graphics format (default: %(default)s)\n' + \
            'CHOICES: [same as above]'
        )
    parser.add_argument(
        '--ogf',
        type=str,
        default="unknown",
        const="unknown",
        nargs='?',
        choices=['sfloat16','sfloat32','uint8','uint16','uint32'],
        metavar='',
        help='Output image graphics format (default: %(default)s)\n' + \
            'CHOICES: [same as above]'
        )
    parser.add_argument(
        '--device',
        type=str,
        default="cuda",
        const="cuda",
        nargs='?',
        choices=['cpu','cuda'],
        metavar='',
        help='Device for gradient descent (default: %(default)s)\n'
        )
    args = parser.parse_args()

    try:
        if not (args.save_lut or args.save_image):
            parser.error('no output requested - need at least one of: --save-lut or --
↪save-image')

        assert 8. <= args.size <= 128, "size must be in range [0, 128]"
        assert 1 <= args.steps <= 10000, "steps must be in range [0, 10000]"
        assert 0. <= args.strength <= 1., "strength must be in range [0, 1]"
        assert 0. <= args.learning_rate <= 1., "learning-rate must be in range [0, 1]"

        steps   = int(args.steps)
```

(continues on next page)

(continued from previous page)

```
        fp_src  = os.path.realpath(args.source) if args.source else None
        fp_tgt  = os.path.realpath(args.target) if args.target else None
        fp_out  = os.path.realpath(args.save_image) if args.save_image else None
        fp_lut  = os.path.realpath(args.save_lut) if args.save_lut else None
        gfs_in  = args.igfs.strip().upper()
        gft_in  = args.igft.strip().upper()
        gf_out  = args.ogf.strip().upper()

        im_src = fsio.load_image(fp_src, graphics_format=fsio.GraphicsFormat[gfs_in])
        im_dst = fsio.load_image(fp_tgt, graphics_format=fsio.GraphicsFormat[gft_in])
        im_src = fsio.truncate_image(im_src)
        im_dst = fsio.truncate_image(im_dst)

        lut = LookupTable.get_empty(args.size) if args.empty_lut else LookupTable.get_
↪linear(args.size)
        lut.fit_to_image(
            im_source=im_src,
            im_target=im_dst,
            strength=args.strength,
            steps=steps,
            device=args.device,
            context=progress_bar
            )

        im_out = lut.apply(im_src)
        if fp_lut: lut.save(fp_lut)
        if fp_out: fsio.save_image(im_out, fp_out, graphics_format=fsio.
↪GraphicsFormat[gf_out])
    except Exception as e:
        print(f'cannot proceed: {e}')

if __name__ == '__main__':
    main_cli()
```

CHAPTER

EIGHT

LINKS

- GitHub
- PyPi

CHAPTER

NINE

SIBLING PROJECTS

- tinytex
- tinyfilm

CHAPTER

TEN

SPECIAL THANKS

- Some color space conversion is from S2CRNet, convert-colors-py and seal-3d.
- Several matrices were computed with colorspace-routines.
- CCT calculation is from colour-science.
- The AgX implementation is owed to Troy James Sobotka and Liam Collod.
- Some loss computations are borrowed from NLUT.
- Thanks to the @64 blog for explaining common tone mapping algorithms.
- Stephen Hill (@self_shadow) for ACES tone mapping fit.
- John Hable for the Uncharted 2 operator.
- Reinhard et al. for their operator.
- The white balancing and the von Kries transform were kindly explained by pbr-book.
- The OKLAB color space was developed by Björn Ottosson
- The OKHSL and OKHSV color space conversions originally by Brian Holbrook
- Matt of Ready At Dawn Studios for explaining Logluv.
- Test photograph is from Bianca Salgado

PYTHON MODULE INDEX

t

INDEX

V

W

X

Y

Z