Colour - HDRI Documentation

Release 0.2.0

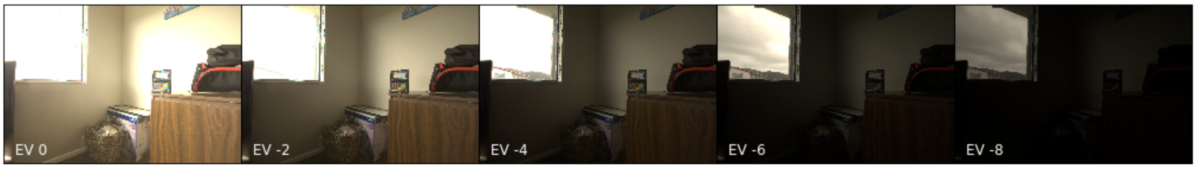
Colour Developers

Feb 26, 2022

CONTENTS

A Python package implementing various HDRI / Radiance image processing algorithms.

It is open source and freely available under the New BSD License terms.



CHAPTER

ONE

1.1 FEATURES

The following features are available:

- HDRI / Radiance Image Generation
- Debevec (1997) Camera Response Function Computation
- Grossberg (2003) Histogram Based Image Sampling
- Variance Minimization Light Probe Sampling
- Global Tonemapping Operators
- Adobe DNG SDK Colour Processing
- Absolute Luminance Calibration
- Digital Still Camera (DSC) Exposure Model
- Raw Processing Helpers

1.1 1.1.1 Examples

Various usage examples are available from the examples directory.

CHAPTER

TWO

1.2 USER GUIDE

2.1 User Guide

The user guide provides an overview of **Colour - HDRI** and explains important concepts and features, details can be found in the API Reference.

2.1.1 Installation Guide

Because of their size, the resources dependencies needed to run the various examples and unit tests are not provided within the Pypi package. They are separately available as Git Submodules when cloning the repository.

Primary Dependencies

Colour - HDRI requires various dependencies in order to run:

- python >= 3.8, < 4
- colour-science
- recordclass

Optional Features Dependencies

- colour-demosaicing
- Adobe DNG Converter
- dcraw
- ExifTool
- rawpy

Pypi

Once the dependencies are satisfied, **Colour - HDRI** can be installed from the Python Package Index by issuing this command in a shell:

```
pip install --user colour-hdri
```

The optional features dependencies are installed as follows:

```
pip install --user 'colour-hdri[optional]'
```

The figures plotting dependencies are installed as follows:

```
pip install --user 'colour-hdri[plotting]'
```

The overall development dependencies are installed as follows:

```
pip install --user 'colour-hdri[development]'
```

2.1.2 Bibliography

Indirect References

Some extra references used in the codebase but not directly part of the public api:

- [AdobeSystems15b]
- [AdobeSystems15c]

CHAPTER

THREE

1.3 API REFERENCE

3.1 API Reference

3.1.1 Colour - HDRI

Camera Calibration

Absolute Luminance - Lagarde (2016)

`colour_hdri`

`absolute_luminance_calibration_Lagarde2016(...)`	Perform absolute *Luminance* calibration of given *RGB* panoramic image using *Lagarde (2016)* method.
`upper_hemisphere_illuminance_weights_Lagarde2016(...)`	Compute upper hemisphere illuminance weights for use with applications unable to perform the computation directly, i.e. *Adobe Photoshop*.

colour_hdri.absolute_luminance_calibration_Lagarde2016

`colour_hdri.`**`absolute_luminance_calibration_Lagarde2016`**(*RGB: ArrayLike, measured_illuminance: Floating, colourspace: colour.models.rgb.rgb_colourspace.RGB_Colourspace = RGB_COLOURSPACES['sRGB']*) → numpy.ndarray

Perform absolute *Luminance* calibration of given *RGB* panoramic image using *Lagarde (2016)* method.

Parameters

- **`RGB`** (`ArrayLike`) – *RGB* panoramic image to calibrate.
- **`measured_illuminance`** (`Floating`) – Measured illuminance E_v.
- **`colourspace`** (`colour.models.rgb.rgb_colourspace.RGB_Colourspace`) – *RGB* colourspace used for internal *Luminance* computation.

Returns Absolute *Luminance* calibrated *RGB* panoramic image.

Return type `numpy.ndarray`

Examples

```
>>> RGB = np.ones((4, 8, 3))
>>> absolute_luminance_calibration_Lagarde2016(
...     RGB, 500)
array([[[ 233.9912506...,  233.9912506...,  233.9912506...],
        [ 233.9912506...,  233.9912506...,  233.9912506...],
        [ 233.9912506...,  233.9912506...,  233.9912506...],
        [ 233.9912506...,  233.9912506...,  233.9912506...],
        [ 233.9912506...,  233.9912506...,  233.9912506...],
        [ 233.9912506...,  233.9912506...,  233.9912506...],
        [ 233.9912506...,  233.9912506...,  233.9912506...],
        [ 233.9912506...,  233.9912506...,  233.9912506...]],

       [[ 233.9912506...,  233.9912506...,  233.9912506...],
        [ 233.9912506...,  233.9912506...,  233.9912506...],
        [ 233.9912506...,  233.9912506...,  233.9912506...],
        [ 233.9912506...,  233.9912506...,  233.9912506...],
        [ 233.9912506...,  233.9912506...,  233.9912506...],
        [ 233.9912506...,  233.9912506...,  233.9912506...],
        [ 233.9912506...,  233.9912506...,  233.9912506...],
        [ 233.9912506...,  233.9912506...,  233.9912506...]],

       [[ 233.9912506...,  233.9912506...,  233.9912506...],
        [ 233.9912506...,  233.9912506...,  233.9912506...],
        [ 233.9912506...,  233.9912506...,  233.9912506...],
        [ 233.9912506...,  233.9912506...,  233.9912506...],
        [ 233.9912506...,  233.9912506...,  233.9912506...],
        [ 233.9912506...,  233.9912506...,  233.9912506...],
        [ 233.9912506...,  233.9912506...,  233.9912506...],
        [ 233.9912506...,  233.9912506...,  233.9912506...]],

       [[ 233.9912506...,  233.9912506...,  233.9912506...],
        [ 233.9912506...,  233.9912506...,  233.9912506...],
        [ 233.9912506...,  233.9912506...,  233.9912506...],
        [ 233.9912506...,  233.9912506...,  233.9912506...],
        [ 233.9912506...,  233.9912506...,  233.9912506...],
        [ 233.9912506...,  233.9912506...,  233.9912506...],
        [ 233.9912506...,  233.9912506...,  233.9912506...],
        [ 233.9912506...,  233.9912506...,  233.9912506...]]])
```

colour_hdri.upper_hemisphere_illuminance_weights_Lagarde2016

`colour_hdri.`**`upper_hemisphere_illuminance_weights_Lagarde2016`**(*height: Integer*, *width: Integer*) → numpy.ndarray

Compute upper hemisphere illuminance weights for use with applications unable to perform the computation directly, i.e. *Adobe Photoshop*.

Parameters

- **`height`** (`Integer`) – Output array height.
- **`width`** (`Integer`) – Output array width.

Returns Upper hemisphere illuminance weights.

Return type `numpy.ndarray`

References

[LLJ16]

Examples

```
>>> upper_hemisphere_illuminance_weights_Lagarde2016(
...     16, 1)
array([[ 0...        ],
       [ 4.0143297...],
       [ 7.3345454...],
       [ 9.3865515...],
       [ 9.8155376...],
       [ 8.5473281...],
       [ 5.8012079...],
       [ 2.0520061...],
       [ 0...        ],
       [ 0...        ],
       [ 0...        ],
       [ 0...        ],
       [ 0...        ],
       [ 0...        ],
       [ 0...        ],
       [ 0...        ]])
```

Debevec (1997)

`colour_hdri`

`g_solve`(Z, B[, l_s, w, n])	Given a set of pixel values observed for several pixels in several images with different exposure times, this function returns the imaging system's response function g as well as the log film irradiance values lE for the observed pixels.
`camera_response_functions_Debevec1997`(...[, ...])	Return the camera response functions for given image stack using *Debevec (1997)* method.

colour_hdri.g_solve

`colour_hdri.`**`g_solve`**(*Z: ArrayLike*, *B: ArrayLike*, *l_s: Floating = 30*, *w: Callable = weighting_function_Debevec1997*, *n: Integer = 256*) → Tuple[numpy.ndarray, numpy.ndarray]

Given a set of pixel values observed for several pixels in several images with different exposure times, this function returns the imaging system's response function g as well as the log film irradiance values lE for the observed pixels.

Parameters

- **Z** (`ArrayLike`) – Set of pixel values observed for several pixels in several images.
- **B** (`ArrayLike`) – Log Δt, or log shutter speed for images.
- **l_s** (`Floating`) – λ smoothing term.
- **w** (`Callable`) – Weighting function w.
- **n** (`Integer`) – n constant.

Returns Camera response functions $g(z)$ and log film irradiance values lE.

Return type tuple

References

[DM97]

colour_hdri.camera_response_functions_Debevec1997

colour_hdri.**camera_response_functions_Debevec1997**(*image_stack: colour_hdri.utilities.image.ImageStack, sampling_function: Callable = samples_Grossberg2003, sampling_function_kwargs: Optional[Dict] = None, weighting_function: Callable = weighting_function_Debevec1997, weighting_function_kwargs: Optional[Dict] = None, extrapolating_function: Callable = extrapolating_function_polynomial, extrapolating_function_kwargs: Optional[Dict] = None, l_s: Floating = 30, n: Integer = 256, normalise: Boolean = True*) → numpy.ndarray

Return the camera response functions for given image stack using *Debevec (1997)* method.

Image channels are sampled with s sampling function and the output samples are passed to `colour_hdri.g_solve()`.

Parameters

- **image_stack** (`colour_hdri.utilities.image.ImageStack`) – Stack of single channel or multi-channel floating point images.
- **sampling_function** (`Callable`) – Sampling function s.
- **sampling_function_kwargs** (`Optional[Dict]`) – Arguments to use when calling the sampling function.
- **weighting_function** (`Callable`) – Weighting function w.
- **weighting_function_kwargs** (`Optional[Dict]`) – Arguments to use when calling the weighting function.
- **extrapolating_function** (`Callable`) – Extrapolating function used to handle zero-weighted data.
- **extrapolating_function_kwargs** (`Optional[Dict]`) – Arguments to use when calling the extrapolating function.
- **l_s** (`Floating`) – λ smoothing term.
- **n** (`Integer`) – n constant.
- **normalise** (`Boolean`) – Enables the camera response functions normalisation.

Returns Camera response functions $g(z)$.

Return type numpy.ndarray

References

[DM97]

Exposure Computation

Common

`colour_hdri`

`average_luminance`(N, t, S[, k])	Compute the average luminance L in $cd \cdot m^{-2}$ from given relative aperture *F-Number* N, *Exposure Time* t, *ISO* arithmetic speed S and *reflected light calibration constant* k.
`average_illuminance`(N, t, S[, c])	Compute the average illuminance E in Lux from given relative aperture *F-Number* N, *Exposure Time* t, *ISO* arithmetic speed S and *incident light calibration constant* c.
`luminance_to_exposure_value`(L, S[, k])	Compute the exposure value EV from given scene luminance L in $cd \cdot m^{-2}$, *ISO* arithmetic speed S and *reflected light calibration constant* k.
`illuminance_to_exposure_value`(E, S[, c])	Compute the exposure value EV from given scene illuminance E in Lux, *ISO* arithmetic speed S and *incident light calibration constant* c.
`adjust_exposure`(a, EV)	Adjust given array exposure using given EV exposure value.

colour_hdri.average_luminance

`colour_hdri.`**`average_luminance`**(*N: FloatingOrArrayLike*, *t: FloatingOrArrayLike*, *S: FloatingOrArrayLike*, *k: FloatingOrArrayLike = 12.5*) → FloatingOrNDArray

Compute the average luminance L in $cd \cdot m^{-2}$ from given relative aperture *F-Number* N, *Exposure Time* t, *ISO* arithmetic speed S and *reflected light calibration constant* k.

Parameters

- **N** (`FloatingOrArrayLike`) – Relative aperture *F-Number* N.
- **t** (`FloatingOrArrayLike`) – *Exposure Time* t.
- **S** (`FloatingOrArrayLike`) – *ISO* arithmetic speed S.
- **k** (`FloatingOrArrayLike`) – *Reflected light calibration constant* k. *ISO 2720:1974* recommends a range for k of 10.6 to 13.4 with luminance in $cd \cdot m^{-2}$. Two values for k are in common use: 12.5 (Canon, Nikon, and Sekonic) and 14 (Minolta, Kenko, and Pentax).

Returns Average luminance L in $cd \cdot m^{-2}$.

Return type `np.floating` or `numpy.ndarray`

References

[Wikipediaa]

Examples

```
>>> average_luminance(8, 1, 100)
8.0
```

colour_hdri.average_illuminance

colour_hdri.**average_illuminance**(*N: FloatingOrArrayLike, t: FloatingOrArrayLike, S: FloatingOrArrayLike, c: FloatingOrArrayLike = 250*) → FloatingOrNDArray

Compute the average illuminance E in Lux from given relative aperture *F-Number* N, *Exposure Time* t, *ISO* arithmetic speed S and *incident light calibration constant* c.

Parameters

- **N** (FloatingOrArrayLike) – Relative aperture *F-Number* N.
- **t** (FloatingOrArrayLike) – *Exposure Time* t.
- **S** (FloatingOrArrayLike) – *ISO* arithmetic speed S.
- **c** (FloatingOrArrayLike) – *Incident light calibration constant* c. With a flat receptor, *ISO 2720:1974* recommends a range for c. of 240 to 400 with illuminance in Lux; a value of 250 is commonly used. With a hemispherical receptor, *ISO 2720:1974* recommends a range for c of 320 to 540 with illuminance in Lux; in practice, values typically are between 320 (Minolta) and 340 (Sekonic).

Returns Average illuminance E in Lux.

Return type np.floating or numpy.ndarray

References

[Wikipediaa]

Examples

```
>>> average_illuminance(8, 1, 100)
160.0
```

colour_hdri.luminance_to_exposure_value

colour_hdri.**luminance_to_exposure_value**(*L: FloatingOrArrayLike, S: FloatingOrArrayLike, k: FloatingOrArrayLike = 12.5*) → FloatingOrNDArray

Compute the exposure value EV from given scene luminance L in $cd \cdot m^{-2}$, *ISO* arithmetic speed S and *reflected light calibration constant* k.

Parameters

- **L** (FloatingOrArrayLike) – Scene luminance L in $cd \cdot m^{-2}$.
- **S** (FloatingOrArrayLike) – *ISO* arithmetic speed S.

- **k** (FloatingOrArrayLike) – *Reflected light calibration constant* k. *ISO 2720:1974* recommends a range for k of 10.6 to 13.4 with luminance in $cd \cdot m^{-2}$. Two values for k are in common use: 12.5 (Canon, Nikon, and Sekonic) and 14 (Minolta, Kenko, and Pentax).

Returns Exposure value EV.

Return type np.floating or numpy.ndarray

Notes

- The exposure value EV indicates a combination of camera settings rather than the focal plane exposure, i.e. luminous exposure, photometric exposure, H. The focal plane exposure is time-integrated illuminance.

References

[Wikipediaa]

Examples

```
>>> luminance_to_exposure_value(0.125, 100)
0.0
```

colour_hdri.illuminance_to_exposure_value

colour_hdri.**illuminance_to_exposure_value**(*E: FloatingOrArrayLike*, *S: FloatingOrArrayLike*, *c: FloatingOrArrayLike = 250*) → FloatingOrNDArray

Compute the exposure value EV from given scene illuminance E in Lux, *ISO* arithmetic speed S and *incident light calibration constant* c.

Parameters

- **E** (FloatingOrArrayLike) – Scene illuminance E in Lux.
- **S** (FloatingOrArrayLike) – *ISO* arithmetic speed S.
- **c** (FloatingOrArrayLike) – *Incident light calibration constant* c. With a flat receptor, *ISO 2720:1974* recommends a range for c. of 240 to 400 with illuminance in Lux; a value of 250 is commonly used. With a hemispherical receptor, *ISO 2720:1974* recommends a range for c of 320 to 540 with illuminance in Lux; in practice, values typically are between 320 (Minolta) and 340 (Sekonic).

Returns Exposure value EV.

Return type np.floating or numpy.ndarray

Notes

- The exposure value EV indicates a combination of camera settings rather than the focal plane exposure, i.e. luminous exposure, photometric exposure, H. The focal plane exposure is time-integrated illuminance.

References

[Wikipediaa]

Examples

```
>>> illuminance_to_exposure_value(2.5, 100)
0.0
```

colour_hdri.adjust_exposure

colour_hdri.**adjust_exposure**(*a: FloatingOrArrayLike*, *EV: FloatingOrArrayLike*) → FloatingOrNDArray

Adjust given array exposure using given EV exposure value.

Parameters

- **a** (FloatingOrArrayLike) – Array to adjust the exposure.
- **EV** (FloatingOrArrayLike) – Exposure adjustment value.

Returns Exposure adjusted array.

Return type np.floating or numpy.ndarray

Examples

```
>>> adjust_exposure(np.array([0.25, 0.5, 0.75, 1]), 1)
array([ 0.5,  1. ,  1.5,  2. ])
```

Digital Still Camera Exposure

colour_hdri

focal_plane_exposure(L, A, t, F, i, H_f[, ...])	Compute the focal plane exposure H in lux-seconds $(lx.s)$.
arithmetic_mean_focal_plane_exposure(L_a, A, t)	Compute the arithmetic mean focal plane exposure H_a for a camera focused on infinity, $H_f << H$, $T = 9/10$, $\theta = 10°$ and $f_v = 98/100$.
saturation_based_speed_focal_plane_exposure(L, ...)	Compute the Saturation-Based Speed (SBS) focal plane exposure H_{SBS} in lux-seconds $(lx.s)$.
exposure_index_values(H_a)	Compute the exposure index values I_{EI} from given focal plane exposure H_a.
exposure_value_100(N, t, S)	Compute the exposure value $EV100$ from given relative aperture *F-Number* N, *Exposure Time* t and *ISO* arithmetic speed S.
photometric_exposure_scale_factor_Lagarde2014(EV100)	Convert the exposure value $EV100$ to photometric exposure scale factor using *Lagarde and de Rousiers (2014)* formulation derived from the *ISO 12232:2006 Saturation Based Sensitivity* (SBS) recommendation.

colour_hdri.focal_plane_exposure

colour_hdri.**focal_plane_exposure**(*L: FloatingOrArrayLike, A: FloatingOrArrayLike, t: FloatingOrArrayLike, F: FloatingOrArrayLike, i: FloatingOrArrayLike, H_f: FloatingOrArrayLike, T: FloatingOrArrayLike = 9 / 10, f_v: FloatingOrArrayLike = 98 / 100, theta: FloatingOrArrayLike = 10*) → FloatingOrNDArray

Compute the focal plane exposure H in lux-seconds $(lx.s)$.

Parameters

- **L** (FloatingOrArrayLike) – Scene luminance L, expressed in cd/m^2.
- **A** (FloatingOrArrayLike) – Lens *F-Number* A.
- **t** (FloatingOrArrayLike) – *Exposure Time* t, expressed in seconds.
- **F** (FloatingOrArrayLike) – Lens focal length F, expressed in meters.
- **i** (FloatingOrArrayLike) – Image distance i, expressed in meters.
- **H_f** (FloatingOrArrayLike) – Focal plane flare exposure H_f, expressed in lux-seconds $(lx.s)$.
- **T** (FloatingOrArrayLike) – Transmission factor of the lens T.
- **f_v** (FloatingOrArrayLike) – Vignetting factor f_v.
- **theta** (FloatingOrArrayLike) – Angle of image point off axis θ.

Returns Focal plane exposure H in lux-seconds $(lx.s)$.

Return type np.floating or numpy.ndarray

Notes

- Focal plane exposure is also named luminous exposure or photometric exposure and is time-integrated illuminance.
- Object distance o, focal length F, and image distance i are related by the thin-lens equation: $\frac{1}{f}=\frac{1}{o}+\frac{1}{i}$
- This method ignores the *ISO* arithmetic speed S and is not concerned with determining an appropriate minimum or maximum exposure level.

References

[ISO06]

Examples

```
>>> focal_plane_exposure(4000, 8, 1 / 250, 50 / 1000, 50 / 1000, 0.0015)
...
0.1643937...
```

colour_hdri.arithmetic_mean_focal_plane_exposure

colour_hdri.**arithmetic_mean_focal_plane_exposure**(*L_a: FloatingOrArrayLike, A: FloatingOrArrayLike, t: FloatingOrArrayLike*) → FloatingOrNDArray

Compute the arithmetic mean focal plane exposure H_a for a camera focused on infinity, $H_f << H$, $T = 9/10$, $\theta = 10°$ and $f_v = 98/100$.

Parameters

- **L_a** (FloatingOrArrayLike) – Arithmetic scene luminance L_a, expressed in cd/m^2.
- **A** (FloatingOrArrayLike) – Lens *F-Number* A.
- **t** (FloatingOrArrayLike) – *Exposure Time* t, expressed in seconds.

Returns Focal plane exposure H_a.

Return type np.floating or numpy.ndarray

Notes

- Focal plane exposure is also named luminous exposure or photometric exposure and is time-integrated illuminance.
- Object distance o, focal length F, and image distance i are related by the thin-lens equation: $\frac{1}{f} = \frac{1}{o} + \frac{1}{i}$
- This method ignores the *ISO* arithmetic speed S and is not concerned with determining an appropriate minimum or maximum exposure level.

References

[ISO06]

Examples

```
>>> arithmetic_mean_focal_plane_exposure(4000, 8, 1 / 250)
...
0.1628937...
```

colour_hdri.saturation_based_speed_focal_plane_exposure

colour_hdri.**saturation_based_speed_focal_plane_exposure**(*L: FloatingOrArrayLike, A: FloatingOrArrayLike, t: FloatingOrArrayLike, S: FloatingOrArrayLike, F: FloatingOrArrayLike = 50 / 1000, i: FloatingOrArrayLike = 1 / - 1 / 5 + 1 / 50 / 1000, H_f: FloatingOrArrayLike = 0, T: FloatingOrArrayLike = 9 / 10, f_v: FloatingOrArrayLike = 98 / 100, theta: FloatingOrArrayLike = 10*) → FloatingOrNDArray

Compute the Saturation-Based Speed (SBS) focal plane exposure H_{SBS} in lux-seconds ($lx.s$).

The model implemented by this definition is appropriate to simulate a physical camera in an offline or realtime renderer.

Parameters

- **L** (FloatingOrArrayLike) – Scene luminance L, expressed in cd/m^2.
- **A** (FloatingOrArrayLike) – Lens *F-Number* A.
- **t** (FloatingOrArrayLike) – *Exposure Time* t, expressed in seconds.
- **S** (FloatingOrArrayLike) – *ISO* arithmetic speed S.
- **F** (FloatingOrArrayLike) – Lens focal length F, expressed in meters.
- **i** (FloatingOrArrayLike) – Image distance i, expressed in meters.
- **H_f** (FloatingOrArrayLike) – Focal plane flare exposure H_f, expressed in lux-seconds ($lx.s$).
- **T** (FloatingOrArrayLike) – Transmission factor of the lens T.
- **f_v** (FloatingOrArrayLike) – Vignetting factor f_v.
- **theta** (FloatingOrArrayLike) – Angle of image point off axis θ.

Returns Saturation-Based Speed focal plane exposure H_{SBS} in lux-seconds ($lx.s$).

Return type `np.floating` or `numpy.ndarray`

Notes

- Focal plane exposure is also named luminous exposure or photometric exposure and is time-integrated illuminance.
- Object distance o, focal length F, and image distance i are related by the thin-lens equation: $\frac{1}{f}=\frac{1}{o}+\frac{1}{i}$
- The image distance default value is that of an object located at 5m and imaged with a 50mm lens.
- The saturation based speed, S_{sat}, of an electronic still picture camera is defined as: $S_{sat}=\frac{78}{H_{sat}}$ where H_{sat} is the minimum focal plane exposure, expressed in lux-seconds ($lx.s$), that produces the maximum valid (not clipped or bloomed) camera output signal. This provides $1/2$ "stop" of headroom (41% additional headroom) for specular highlights above the signal level that would be obtained from a theoretical 100% reflectance object in the scene, so that a theoretical 141% reflectance object in the scene would produce a focal plane exposure of H_{sat}.
- The focal plane exposure H_{SBS} computed by this definition is almost equal to that given by scene luminance L scaled with the output of `colour_hdri.photometric_exposure_scale_factor_Lagarde2014()` definition.

References

[ISO06]

Examples

```
>>> saturation_based_speed_focal_plane_exposure(
...     4000, 8, 1 / 250, 400, 50 / 1000, 50 / 1000, 0.0015)
0.8430446...
```

colour_hdri.exposure_index_values

colour_hdri.**exposure_index_values**(*H_a: FloatingOrArrayLike*) → FloatingOrNDArray

Compute the exposure index values I_{EI} from given focal plane exposure H_a.

Parameters **H_a** (FloatingOrArrayLike) – Focal plane exposure H_a.

Returns Exposure index values I_{EI}.

Return type np.floating or numpy.ndarray

References

[ISO06]

Examples

```
>>> exposure_index_values(0.1628937086212269)
61.3897251...
```

colour_hdri.exposure_value_100

colour_hdri.**exposure_value_100**(*N: FloatingOrArrayLike, t: FloatingOrArrayLike, S: FloatingOrArrayLike*) → FloatingOrNDArray

Compute the exposure value $EV100$ from given relative aperture *F-Number* N, *Exposure Time* t and *ISO* arithmetic speed S.

Parameters

- **N** (FloatingOrArrayLike) – Relative aperture *F-Number* N.
- **t** (FloatingOrArrayLike) – *Exposure Time* t.
- **S** (FloatingOrArrayLike) – *ISO* arithmetic speed S.

Returns Exposure value $EV100$.

Return type np.floating or numpy.ndarray

References

[ISO06], [LdeRousiers14]

Notes

- The underlying implementation uses the `colour_hdri.luminance_to_exposure_value()` and `colour_hdri.average_luminance()` definitions with same fixed value for the *reflected light calibration constant* k which cancels its scaling effect and produces a value equal to $log_2(\frac{N^2}{t}) - log_2(\frac{S}{100})$ as given in [LdeRousiers14].

Examples

```
>>> exposure_value_100(8, 1 / 250, 400)
11.9657842...
```

colour_hdri.photometric_exposure_scale_factor_Lagarde2014

`colour_hdri.`**`photometric_exposure_scale_factor_Lagarde2014`**(*EV100: FloatingOrArrayLike, T: FloatingOrArrayLike = 9 / 10, f_v: FloatingOrArrayLike = 98 / 100, theta: FloatingOrArrayLike = 10*) → FloatingOrNDArray

Convert the exposure value $EV100$ to photometric exposure scale factor using *Lagarde and de Rousiers (2014)* formulation derived from the *ISO 12232:2006 Saturation Based Sensitivity* (SBS) recommendation.

The model implemented by this definition is appropriate to simulate a physical camera in an offline or realtime renderer.

Parameters

- **T** (`FloatingOrArrayLike`) – Exposure value $EV100$.
- **T** – Transmission factor of the lens T.
- **f_v** (`FloatingOrArrayLike`) – Vignetting factor f_v.
- **theta** (`FloatingOrArrayLike`) – Angle of image point off axis θ.
- **EV100** (`FloatingOrArrayLike`) –

Returns Photometric exposure in lux-seconds ($lx.s$).

Return type `np.floating` or `numpy.ndarray`

Notes

- The saturation based speed, S_{sat}, of an electronic still picture camera is defined as: $S_{sat} = \frac{78}{H_{sat}}$ where H_{sat} is the minimum focal plane exposure, expressed in lux-seconds ($lx.s$), that produces the maximum valid (not clipped or bloomed) camera output signal. This provides $1/2$ "stop" of headroom (41% additional headroom) for specular highlights above the signal level that would be obtained from a theoretical 100% reflectance object in the scene, so that a theoretical 141% reflectance object in the scene would produce a focal plane exposure of H_{sat}.

- Scene luminance L scaled with the photometric exposure value computed by this definition is almost equal to that given by the `colour_hdri.saturation_based_speed_focal_plane_exposure()` definition.

References

[ISO06], [LdeRousiers14]

Examples

```
>>> EV100 = exposure_value_100(8, 1 / 250, 400)
>>> H = photometric_exposure_scale_factor_Lagarde2014(EV100)
>>> print(H)
0.0002088...
>>> H * 4000
0.8353523...
```

HDRI / Radiance Image Generation

Generation

`colour_hdri`

`image_stack_to_radiance_image`(image_stack[, ...])	Generate a HDRI / radiance image from given image stack.

colour_hdri.image_stack_to_radiance_image

`colour_hdri.`**`image_stack_to_radiance_image`**(*image_stack: ImageStack*, *weighting_function: Callable = weighting_function_Debevec1997*, *weighting_average: Boolean = False*, *camera_response_functions: Optional[ArrayLike] = None*) → Optional[NDArray]

Generate a HDRI / radiance image from given image stack.

Parameters

- **`image_stack`** (`ImageStack`) – Stack of single channel or multi-channel floating point images. The stack is assumed to be representing linear values except if `camera_response_functions` argument is provided.
- **`weighting_function`** (`Callable`) – Weighting function w.
- **`weighting_average`** (`Boolean`) – Enables weighting function w computation on channels average instead of on a per-channel basis.
- **`camera_response_functions`** (`Optional[ArrayLike]`) – Camera response functions $g(z)$ of the imaging system / camera if the stack is representing non-linear values.

Returns Radiance image.

Return type `numpy.ndarray`

> **Warning:** If the image stack contains images with negative or equal to zero values, unpredictable results may occur and NaNs might be generated. It is thus recommended encoding the

images in a wider RGB colourspace or clamp negative values.

References

[BADC11a]

Weighting Functions

colour_hdri

normal_distribution_function(a[, mu, sigma])	Return given array weighted by a normal distribution function.
hat_function(a)	Return given array weighted by a hat function.
weighting_function_Debevec1997(a[, ...])	Return given array weighted by *Debevec (1997)* function.

colour_hdri.normal_distribution_function

colour_hdri.**normal_distribution_function**(*a: ArrayLike*, *mu: Floating = 0.5*, *sigma: Floating = 0.15*) → numpy.ndarray

Return given array weighted by a normal distribution function.

Parameters

- **a** (ArrayLike) – Array to apply the weighting function onto.
- **mu** (Floating) – Mean or expectation.
- **sigma** (Floating) – Standard deviation.

Returns Weighted array.

Return type numpy.ndarray

Examples

```
>>> normal_distribution_function(np.linspace(0, 1, 10))
array([ 0.00386592,  0.03470859,  0.18002174,  0.53940751,  0.93371212,
        0.93371212,  0.53940751,  0.18002174,  0.03470859,  0.00386592])
```

colour_hdri.hat_function

colour_hdri.**hat_function**(*a: ArrayLike*) → numpy.ndarray

Return given array weighted by a hat function.

Parameters **a** (ArrayLike) – Array to apply the weighting function onto.

Returns Weighted array.

Return type numpy.ndarray

Examples

```
>>> hat_function(np.linspace(0, 1, 10))
array([ 0.        ,  0.95099207,  0.99913557,  0.99999812,  1.        ,
        1.        ,  0.99999812,  0.99913557,  0.95099207,  0.        ])
```

colour_hdri.weighting_function_Debevec1997

colour_hdri.**weighting_function_Debevec1997**(*a: ArrayLike*, *domain_l: Floating = 0.01*, *domain_h: Floating = 0.99*) → numpy.ndarray

Return given array weighted by *Debevec (1997)* function.

Parameters

- **a** (ArrayLike) – Array to apply the weighting function onto.
- **domain_l** (Floating) – Domain lowest possible value, values less than domain_l will be set to zero.
- **domain_h** (Floating) – Domain highest possible value, values greater than domain_h will be set to zero.

Returns Weighted array.

Return type numpy.ndarray

References

[DM97]

Examples

```
>>> weighting_function_Debevec1997(np.linspace(0, 1, 10))
array([ 0.        ,  0.23273657,  0.48849105,  0.74424552,  1.        ,
        1.        ,  0.74424552,  0.48849105,  0.23273657,  0.        ])
```

Colour Models

Adobe DNG SDK

colour_hdri

xy_to_camera_neutral(xy, ...)	Convert given *xy* white balance chromaticity coordinates to *Camera Neutral* coordinates.
camera_neutral_to_xy(camera_neutral, ...[, ...])	Convert given *Camera Neutral* coordinates to *xy* white balance chromaticity coordinates.
XYZ_to_camera_space_matrix(xy, ...)	Return the *CIE XYZ* to *Camera Space* matrix for given *xy* white balance chromaticity coordinates.
camera_space_to_XYZ_matrix(xy, ...[, ...])	Return the *Camera Space* to *CIE XYZ* matrix for given *xy* white balance chromaticity coordinates.

colour_hdri.xy_to_camera_neutral

colour_hdri.**xy_to_camera_neutral**(*xy: ArrayLike*, *CCT_calibration_illuminant_1: Floating*, *CCT_calibration_illuminant_2: Floating*, *M_color_matrix_1: ArrayLike*, *M_color_matrix_2: ArrayLike*, *M_camera_calibration_1: ArrayLike*, *M_camera_calibration_2: ArrayLike*, *analog_balance: ArrayLike*) → numpy.ndarray

Convert given *xy* white balance chromaticity coordinates to *Camera Neutral* coordinates.

Parameters

- **xy** (ArrayLike) – *xy* white balance chromaticity coordinates.
- **CCT_calibration_illuminant_1** (Floating) – Correlated colour temperature of *CalibrationIlluminant1*.
- **CCT_calibration_illuminant_2** (Floating) – Correlated colour temperature of *CalibrationIlluminant2*.
- **M_color_matrix_1** (ArrayLike) – *ColorMatrix1* tag matrix.
- **M_color_matrix_2** (ArrayLike) – *ColorMatrix2* tag matrix.
- **M_camera_calibration_1** (ArrayLike) – *CameraCalibration1* tag matrix.
- **M_camera_calibration_2** (ArrayLike) – *CameraCalibration2* tag matrix.
- **analog_balance** (ArrayLike) – *AnalogBalance* tag vector.

Returns *Camera Neutral* coordinates.

Return type numpy.ndarray

References

[AdobeSystems12d], [AdobeSystems12b], [AdobeSystems15a], [McG12]

Examples

```
>>> M_color_matrix_1 = np.array(
...     [[0.5309, -0.0229, -0.0336],
...      [-0.6241, 1.3265, 0.3337],
...      [-0.0817, 0.1215, 0.6664]])
>>> M_color_matrix_2 = np.array(
...     [[0.4716, 0.0603, -0.0830],
...      [-0.7798, 1.5474, 0.2480],
...      [-0.1496, 0.1937, 0.6651]])
>>> M_camera_calibration_1 = np.identity(3)
>>> M_camera_calibration_2 = np.identity(3)
>>> analog_balance = np.ones(3)
>>> xy_to_camera_neutral(
...     np.array([0.32816244, 0.34698169]),
...     2850,
...     6500,
...     M_color_matrix_1,
...     M_color_matrix_2,
...     M_camera_calibration_1,
...     M_camera_calibration_2,
...     analog_balance)
array([ 0.4130699...,  1...        ,  0.646465...])
```

colour_hdri.camera_neutral_to_xy

colour_hdri.**camera_neutral_to_xy**(*camera_neutral: ArrayLike, CCT_calibration_illuminant_1: Floating, CCT_calibration_illuminant_2: Floating, M_color_matrix_1: ArrayLike, M_color_matrix_2: ArrayLike, M_camera_calibration_1: ArrayLike, M_camera_calibration_2: ArrayLike, analog_balance: ArrayLike, epsilon: Floating = EPSILON*) → numpy.ndarray

Convert given *Camera Neutral* coordinates to *xy* white balance chromaticity coordinates.

Parameters

- **camera_neutral** (ArrayLike) – *Camera Neutral* coordinates.
- **CCT_calibration_illuminant_1** (Floating) – Correlated colour temperature of *CalibrationIlluminant1*.
- **CCT_calibration_illuminant_2** (Floating) – Correlated colour temperature of *CalibrationIlluminant2*.
- **M_color_matrix_1** (ArrayLike) – *ColorMatrix1* tag matrix.
- **M_color_matrix_2** (ArrayLike) – *ColorMatrix2* tag matrix.
- **M_camera_calibration_1** (ArrayLike) – *CameraCalibration1* tag matrix.
- **M_camera_calibration_2** (ArrayLike) – *CameraCalibration2* tag matrix.
- **analog_balance** (ArrayLike) – *AnalogBalance* tag vector.
- **epsilon** (Floating) – Threshold value for computation convergence.

Returns *xy* white balance chromaticity coordinates.

Return type numpy.ndarray

Raises **RuntimeError** – If the given *Camera Neutral* coordinates did not converge to *xy* white balance chromaticity coordinates.

References

[AdobeSystems12c], [AdobeSystems12b], [AdobeSystems15a], [McG12]

Examples

```
>>> M_color_matrix_1 = np.array(
...     [[0.5309, -0.0229, -0.0336],
...      [-0.6241, 1.3265, 0.3337],
...      [-0.0817, 0.1215, 0.6664]])
>>> M_color_matrix_2 = np.array(
...     [[0.4716, 0.0603, -0.0830],
...      [-0.7798, 1.5474, 0.2480],
...      [-0.1496, 0.1937, 0.6651]])
>>> M_camera_calibration_1 = np.identity(3)
>>> M_camera_calibration_2 = np.identity(3)
>>> analog_balance = np.ones(3)
>>> camera_neutral_to_xy(
...     np.array([0.413070, 1.000000, 0.646465]),
...     2850,
...     6500,
...     M_color_matrix_1,
...     M_color_matrix_2,
```

(continues on next page)

(continued from previous page)

```
...     M_camera_calibration_1,
...     M_camera_calibration_2,
...     analog_balance)
array([ 0.3281624...,  0.3469816...])
```

colour_hdri.XYZ_to_camera_space_matrix

colour_hdri.**XYZ_to_camera_space_matrix**(*xy: ArrayLike, CCT_calibration_illuminant_1: Floating, CCT_calibration_illuminant_2: Floating, M_color_matrix_1: ArrayLike, M_color_matrix_2: ArrayLike, M_camera_calibration_1: ArrayLike, M_camera_calibration_2: ArrayLike, analog_balance: ArrayLike*) → numpy.ndarray

Return the *CIE XYZ* to *Camera Space* matrix for given *xy* white balance chromaticity coordinates.

Parameters

- **xy** (ArrayLike) – *xy* white balance chromaticity coordinates.
- **CCT_calibration_illuminant_1** (Floating) – Correlated colour temperature of *CalibrationIlluminant1*.
- **CCT_calibration_illuminant_2** (Floating) – Correlated colour temperature of *CalibrationIlluminant2*.
- **M_color_matrix_1** (ArrayLike) – *ColorMatrix1* tag matrix.
- **M_color_matrix_2** (ArrayLike) – *ColorMatrix2* tag matrix.
- **M_camera_calibration_1** (ArrayLike) – *CameraCalibration1* tag matrix.
- **M_camera_calibration_2** (ArrayLike) – *CameraCalibration2* tag matrix.
- **analog_balance** (ArrayLike) – *AnalogBalance* tag vector.

Returns *CIE XYZ* to *Camera Space* matrix.

Return type numpy.ndarray

Notes

- The reference illuminant is D50 as defined per colour_hdri.models.datasets.dng.CCS_ILLUMINANT_ADOBEDNG attribute.

References

[AdobeSystems12b], [AdobeSystems15a], [McG12]

Examples

```
>>> M_color_matrix_1 = np.array(
...     [[0.5309, -0.0229, -0.0336],
...      [-0.6241, 1.3265, 0.3337],
...      [-0.0817, 0.1215, 0.6664]])
>>> M_color_matrix_2 = np.array(
...     [[0.4716, 0.0603, -0.0830],
...      [-0.7798, 1.5474, 0.2480],
...      [-0.1496, 0.1937, 0.6651]])
```

(continues on next page)

(continued from previous page)

```
>>> M_camera_calibration_1 = np.identity(3)
>>> M_camera_calibration_2 = np.identity(3)
>>> analog_balance = np.ones(3)
>>> XYZ_to_camera_space_matrix(
...     np.array([0.34510414, 0.35162252]),
...     2850,
...     6500,
...     M_color_matrix_1,
...     M_color_matrix_2,
...     M_camera_calibration_1,
...     M_camera_calibration_2,
...     analog_balance)
array([[ 0.4854908...,  0.0408106..., -0.0714282...],
       [-0.7433278...,  1.4956549...,  0.2680749...],
       [-0.1336946...,  0.1767874...,  0.6654045...]])
```

colour_hdri.camera_space_to_XYZ_matrix

colour_hdri.**camera_space_to_XYZ_matrix**(*xy: ArrayLike, CCT_calibration_illuminant_1: Floating, CCT_calibration_illuminant_2: Floating, M_color_matrix_1: ArrayLike, M_color_matrix_2: ArrayLike, M_camera_calibration_1: ArrayLike, M_camera_calibration_2: ArrayLike, analog_balance: ArrayLike, M_forward_matrix_1: ArrayLike, M_forward_matrix_2: ArrayLike, chromatic_adaptation_transform: Union[Literal['Bianco 2010', 'Bianco PC 2010', 'Bradford', 'CAT02 Brill 2008', 'CAT02', 'CAT16', 'CMCCAT2000', 'CMCCAT97', 'Fairchild', 'Sharp', 'Von Kries', 'XYZ Scaling'], str] = 'Bradford'*) → numpy.ndarray

Return the *Camera Space* to *CIE XYZ* matrix for given *xy* white balance chromaticity coordinates.

Parameters

- **xy** (ArrayLike) – *xy* white balance chromaticity coordinates.
- **CCT_calibration_illuminant_1** (Floating) – Correlated colour temperature of *CalibrationIlluminant1*.
- **CCT_calibration_illuminant_2** (Floating) – Correlated colour temperature of *CalibrationIlluminant2*.
- **M_color_matrix_1** (ArrayLike) – *ColorMatrix1* tag matrix.
- **M_color_matrix_2** (ArrayLike) – *ColorMatrix2* tag matrix.
- **M_camera_calibration_1** (ArrayLike) – *CameraCalibration1* tag matrix.
- **M_camera_calibration_2** (ArrayLike) – *CameraCalibration2* tag matrix.
- **analog_balance** (ArrayLike) – *AnalogBalance* tag vector.
- **M_forward_matrix_1** (ArrayLike) – *ForwardMatrix1* tag matrix.
- **M_forward_matrix_2** (ArrayLike) – *ForwardMatrix2* tag matrix.
- **chromatic_adaptation_transform** (Union[Literal['Bianco 2010', 'Bianco PC 2010', 'Bradford', 'CAT02 Brill 2008', 'CAT02', 'CAT16', 'CMCCAT2000', 'CMCCAT97', 'Fairchild', 'Sharp', 'Von Kries', 'XYZ Scaling'], str]) – Chromatic adaptation transform.

Returns *Camera Space* to *CIE XYZ* matrix.

Return type numpy.ndarray

Notes

- The reference illuminant is D50 as defined per colour_hdri.models.datasets.dng.CCS_ILLUMINANT_ADOBEDNG attribute.

References

[AdobeSystems12b], [AdobeSystems12a], [AdobeSystems15a], [McG12]

Examples

```
>>> M_color_matrix_1 = np.array(
...     [[0.5309, -0.0229, -0.0336],
...      [-0.6241, 1.3265, 0.3337],
...      [-0.0817, 0.1215, 0.6664]])
>>> M_color_matrix_2 = np.array(
...     [[0.4716, 0.0603, -0.0830],
...      [-0.7798, 1.5474, 0.2480],
...      [-0.1496, 0.1937, 0.6651]])
>>> M_camera_calibration_1 = np.identity(3)
>>> M_camera_calibration_2 = np.identity(3)
>>> analog_balance = np.ones(3)
>>> M_forward_matrix_1 = np.array(
...     [[0.8924, -0.1041, 0.1760],
...      [0.4351, 0.6621, -0.0972],
...      [0.0505, -0.1562, 0.9308]])
>>> M_forward_matrix_2 = np.array(
...     [[0.8924, -0.1041, 0.1760],
...      [0.4351, 0.6621, -0.0972],
...      [0.0505, -0.1562, 0.9308]])
>>> camera_space_to_XYZ_matrix(
...     np.array([0.32816244, 0.34698169]),
...     2850,
...     6500,
...     M_color_matrix_1,
...     M_color_matrix_2,
...     M_camera_calibration_1,
...     M_camera_calibration_2,
...     analog_balance,
...     M_forward_matrix_1,
...     M_forward_matrix_2)
array([[ 2.1604087...,  -0.1041...    ,  0.2722498...],
       [ 1.0533324...,  0.6621...    , -0.1503561...],
       [ 0.1222553...,  -0.1562...    ,  1.4398304...]])
```

RGB Models

`colour_hdri`

camera_space_to_RGB(RGB, ...)	Convert given *RGB* array from *camera space* to given *RGB* colourspace.
camera_space_to_sRGB(RGB, M_XYZ_to_camera_space)	Convert given *RGB* array from *camera space* to *sRGB* colourspace.

colour_hdri.camera_space_to_RGB

colour_hdri.**camera_space_to_RGB**(*RGB: ArrayLike*, *M_XYZ_to_camera_space: ArrayLike*, *matrix_RGB_to_XYZ: ArrayLike*) → numpy.ndarray

Convert given *RGB* array from *camera space* to given *RGB* colourspace.

Parameters

- **RGB** (ArrayLike) – Camera space *RGB* colourspace array.
- **M_XYZ_to_camera_space** (ArrayLike) – Matrix converting from *CIE XYZ* tristimulus values to *camera space*.
- **matrix_RGB_to_XYZ** (ArrayLike) – Matrix converting from *RGB* colourspace to *CIE XYZ* tristimulus values.

Returns *RGB* colourspace array.

Return type numpy.ndarray

Examples

```
>>> RGB = np.array([0.80660, 0.81638, 0.65885])
>>> M_XYZ_to_camera_space = np.array([
...     [0.47160000, 0.06030000, -0.08300000],
...     [-0.77980000, 1.54740000, 0.24800000],
...     [-0.14960000, 0.19370000, 0.66510000]])
>>> matrix_RGB_to_XYZ = np.array([
...     [0.41238656, 0.35759149, 0.18045049],
...     [0.21263682, 0.71518298, 0.07218020],
...     [0.01933062, 0.11919716, 0.95037259]])
>>> camera_space_to_RGB(
...     RGB,
...     M_XYZ_to_camera_space,
...     matrix_RGB_to_XYZ)
array([ 0.7564180...,  0.8683192...,  0.6044589...])
```

colour_hdri.camera_space_to_sRGB

colour_hdri.**camera_space_to_sRGB**(*RGB: ArrayLike*, *M_XYZ_to_camera_space: ArrayLike*) → numpy.ndarray

Convert given *RGB* array from *camera space* to *sRGB* colourspace.

Parameters

- **RGB** (ArrayLike) – Camera space *RGB* colourspace array.
- **M_XYZ_to_camera_space** (ArrayLike) – Matrix converting from *CIE XYZ* tristimulus values to *camera space*.

Returns *sRGB* colourspace array.

Return type numpy.ndarray

Examples

```
>>> RGB = np.array([0.80660, 0.81638, 0.65885])
>>> M_XYZ_to_camera_space = np.array([
...     [0.47160000, 0.06030000, -0.08300000],
...     [-0.77980000, 1.54740000, 0.24800000],
...     [-0.14960000, 0.19370000, 0.66510000]])
>>> camera_space_to_sRGB(RGB, M_XYZ_to_camera_space)
array([ 0.7564350...,  0.8683155...,  0.6044706...])
```

Plotting

HDRI / Radiance Image

colour_hdri.plotting

plot_radiance_image_strip(image[, count, ...])	Plot given HDRI / radiance image as strip of images of varying exposure.

colour_hdri.plotting.plot_radiance_image_strip

colour_hdri.plotting.**plot_radiance_image_strip**(*image: ArrayLike*, *count: Integer = 5*, *ev_steps: Floating = - 2*, *cctf_encoding: Callable = CONSTANTS_COLOUR_STYLE.colour.colourspace.cctf_encoding*, ***kwargs: Any*) → Tuple[matplotlib.figure.Figure, matplotlib.axes._axes.Axes]

Plot given HDRI / radiance image as strip of images of varying exposure.

Parameters

- **image** (ArrayLike) – HDRI / radiance image to plot.
- **count** (Integer) – Strip images count.
- **ev_steps** (Floating) – Exposure variation for each image of the strip.
- **cctf_encoding** (Callable) – Encoding colour component transfer function / opto-electronic transfer function used for plotting.
- **kwargs** (Any) – {colour.plotting.display()}, Please refer to the documentation of the previously listed definition.

Returns Current figure and axes.

Return type tuple

Tonemapping Operators

colour_hdri.plotting

plot_tonemapping_operator_image(image, ...)	Plot given tonemapped image with superimposed luminance mapping function.

colour_hdri.plotting.plot_tonemapping_operator_image

colour_hdri.plotting.**plot_tonemapping_operator_image**(*image: ArrayLike, luminance_function: ArrayLike, log_scale: Boolean = False, cctf_encoding: Callable = CONSTANTS_COLOUR_STYLE.colour.colourspace.cctf_encoding, **kwargs: Any*) → Tuple[matplotlib.figure.Figure, matplotlib.axes._axes.Axes]

Plot given tonemapped image with superimposed luminance mapping function.

Parameters

- **image** (ArrayLike) – Tonemapped image to plot.
- **luminance_function** (ArrayLike) – Luminance mapping function.
- **log_scale** (Boolean) – Use a log scale for plotting the luminance mapping function.
- **cctf_encoding** (Callable) – Encoding colour component transfer function / opto-electronic transfer function used for plotting.
- **kwargs** (Any) – {colour.plotting.render()}, Please refer to the documentation of the previously listed definition.

Returns Current figure and axes.

Return type tuple

Image Processing

Adobe DNG SDK

Raw Files

colour_hdri

convert_raw_files_to_dng_files(raw_files, ...)	Convert given raw files to *dng* files using given output directory.
RAW_CONVERTER	Command line raw conversion application, usually Dave Coffin's *dcraw*.
RAW_CONVERSION_ARGUMENTS	Arguments for the command line raw conversion application for non demosaiced linear *tiff* file format output.
RAW_D_CONVERSION_ARGUMENTS	Arguments for the command line raw conversion application for demosaiced linear *tiff* file format output.

colour_hdri.convert_raw_files_to_dng_files

`colour_hdri.convert_raw_files_to_dng_files(raw_files: Sequence[str], output_directory: str) → List[str]`

Convert given raw files to *dng* files using given output directory.

Parameters

- **raw_files** (Sequence[str]) – Raw files to convert to *dng* files.
- **output_directory** (str) – Output directory.

Returns *dng* files.

Return type list

Raises RuntimeError – If the *Adobe DNG Converter* is not available.

colour_hdri.RAW_CONVERTER

`colour_hdri.RAW_CONVERTER = 'dcraw'`

Command line raw conversion application, usually Dave Coffin's *dcraw*.

colour_hdri.RAW_CONVERSION_ARGUMENTS

`colour_hdri.RAW_CONVERSION_ARGUMENTS = '-t 0 -D -W -4 -T "{0}"'`

Arguments for the command line raw conversion application for non demosaiced linear *tiff* file format output.

colour_hdri.RAW_D_CONVERSION_ARGUMENTS

`colour_hdri.RAW_D_CONVERSION_ARGUMENTS = '-t 0 -H 1 -r 1 1 1 1 -4 -q 3 -o 0 -T "{0}"'`

Arguments for the command line raw conversion application for demosaiced linear *tiff* file format output.

DNG Files

colour_hdri

convert_dng_files_to_intermediate_files(...)	Convert given *dng* files to intermediate *tiff* files using given output directory.
DNG_CONVERTER	
DNG_CONVERSION_ARGUMENTS	Arguments for the command line *dng* conversion application.
DNG_EXIF_TAGS_BINDING	Exif tags binding for a *dng* file.
read_dng_files_exif_tags(dng_files[, ...])	Read given *dng* files exif tags using given binding.

colour_hdri.convert_dng_files_to_intermediate_files

`colour_hdri.`**`convert_dng_files_to_intermediate_files`**(*dng_files: Sequence[str], output_directory: str, demosaicing: Boolean = False*) → List[str]

Convert given *dng* files to intermediate *tiff* files using given output directory.

Parameters

- **`dng_files`** (`Sequence[str]`) – *dng* files to convert to intermediate *tiff* files.
- **`output_directory`** (`str`) – Output directory.
- **`demosaicing`** (`Boolean`) – Perform demosaicing on conversion.

Returns Intermediate *tiff* files.

Return type `list`

colour_hdri.DNG_CONVERTER

`colour_hdri.`**`DNG_CONVERTER = None`**

colour_hdri.DNG_CONVERSION_ARGUMENTS

`colour_hdri.`**`DNG_CONVERSION_ARGUMENTS = '-cr7.1 -l -d "{0}" "{1}"'`**

Arguments for the command line *dng* conversion application.

colour_hdri.DNG_EXIF_TAGS_BINDING

`colour_hdri.`**`DNG_EXIF_TAGS_BINDING = CaseInsensitiveMapping({'EXIF': ...})`**

Exif tags binding for a *dng* file.

colour_hdri.read_dng_files_exif_tags

`colour_hdri.`**`read_dng_files_exif_tags`**(*dng_files: Sequence[str], exif_tags_binding: Mapping[str, Mapping[str, Tuple[Callable, Optional[str]]]] = DNG_EXIF_TAGS_BINDING*) → List[colour.utilities.data_structures.CaseInsensitiveMapping]

Read given *dng* files exif tags using given binding.

Parameters

- **`dng_files`** (`Sequence[str]`) – *dng* files to read the exif tags from.
- **`exif_tags_binding`** (`Mapping[str, Mapping[str, Tuple[Callable, Optional[str]]]]`) – Exif tags binding.

Returns *dng* files exif tags.

Return type `list`

Highlights Recovery

Clipped Highlights Recovery

`colour_hdri`

`highlights_recovery_blend`(RGB, multipliers)	Perform highlights recovery using *Coffin (1997)* method from *dcraw*.
`highlights_recovery_LCHab`(RGB[, threshold, ...])	Perform highlights recovery in *CIE L*C*Hab* colourspace.

colour_hdri.highlights_recovery_blend

`colour_hdri.`**`highlights_recovery_blend`**(*RGB: ArrayLike*, *multipliers: ArrayLike*, *threshold: Floating = 0.99*) → numpy.ndarray

Perform highlights recovery using *Coffin (1997)* method from *dcraw*.

Parameters

- **`RGB`** (`ArrayLike`) – *RGB* colourspace array.
- **`multipliers`** (`ArrayLike`) – Normalised camera white level or white balance multipliers.
- **`threshold`** (`Floating`) – Threshold for highlights selection.

Returns Highlights recovered *RGB* colourspace array.

Return type `numpy.ndarray`

References

[Cof15]

colour_hdri.highlights_recovery_LCHab

`colour_hdri.`**`highlights_recovery_LCHab`**(*RGB: ArrayLike*, *threshold: Optional[Floating] = None*, *RGB_colourspace: RGB_Colourspace = RGB_COLOURSPACE_sRGB*) → NDArray

Perform highlights recovery in *CIE L*C*Hab* colourspace.

Parameters

- **`RGB`** (`ArrayLike`) – *RGB* colourspace array.
- **`threshold`** (`Optional[Floating]`) – Threshold for highlights selection, automatically computed if not given.
- **`RGB_colourspace`** (`RGB_Colourspace`) – Working *RGB* colourspace to perform the *CIE L*C*Hab* to and from.

Returns Highlights recovered *RGB* colourspace array.

Return type `numpy.ndarray`

Image Sampling

Viriyothai (2009)

colour_hdri

light_probe_sampling_variance_minimization_Viriyothai2009(...)	Sample given light probe to find lights using *Viriyothai (2009)* variance minimization light probe sampling algorithm.

colour_hdri.light_probe_sampling_variance_minimization_Viriyothai2009

colour_hdri.**light_probe_sampling_variance_minimization_Viriyothai2009**(*light_probe*, *lights_count=16*, *colourspace=RGB_COLOURSPACES['sRGB*

Sample given light probe to find lights using *Viriyothai (2009)* variance minimization light probe sampling algorithm.

Parameters

- **light_probe** (array_like) – Array to sample for lights.
- **lights_count** (int) – Amount of lights to generate.
- **colourspace** (*colour.RGB_Colourspace*, optional) – *RGB* colourspace used for internal *Luminance* computation.

Returns list of colour_hdri.sampling.variance_minimization.Light_Specification lights.

Return type list

References

[VD09]

Grossberg (2013)

colour_hdri

samples_Grossberg2003(image_stack[, samples, n])	Return the samples for given image stack intensity histograms using *Grossberg (2003)* method.

colour_hdri.samples_Grossberg2003

colour_hdri.**samples_Grossberg2003**(*image_stack: ArrayLike*, *samples: Integer = 1000*, *n: Integer = 256*) → numpy.ndarray

Return the samples for given image stack intensity histograms using *Grossberg (2003)* method.

Parameters

- **image_stack** (ArrayLike) – Stack of single channel or multi-channel floating point images.
- **samples** (Integer) – Samples count.
- **n** (Integer) – Histograms bins count.

Returns Intensity histograms samples.

Return type numpy.ndarray

References

[BB14], [GN03]

Tonemapping Operators

Global

Simple

colour_hdri

tonemapping_operator_simple(RGB)	Perform given *RGB* array tonemapping using the simple method: $\frac{RGB}{RGB+1}$.

colour_hdri.tonemapping_operator_simple

colour_hdri.**tonemapping_operator_simple**(*RGB: ArrayLike*) → numpy.ndarray

Perform given *RGB* array tonemapping using the simple method: $\frac{RGB}{RGB+1}$.

Parameters **RGB** (ArrayLike) – *RGB* array to perform tonemapping onto.

Returns Tonemapped *RGB* array.

Return type numpy.ndarray

References

[Wikipediab]

Examples

```
>>> tonemapping_operator_simple(np.array(
...     [[[0.48046875, 0.35156256, 0.23632812],
...       [1.39843753, 0.55468757, 0.39062594]],
...      [[4.40625388, 2.15625895, 1.34375372],
...       [6.59375023, 3.43751395, 2.21875829]]]))
array([[[ 0.3245382...,  0.2601156...,  0.1911532...],
        [ 0.5830618...,  0.3567839...,  0.2808993...]],

       [[ 0.8150290...,  0.6831692...,  0.5733340...],
        [ 0.8683127...,  0.7746486...,  0.6893211...]]])
```

Normalisation

colour_hdri

tonemapping_operator_normalisation(RGB[, ...])	Perform given *RGB* array tonemapping using the normalisation method.

colour_hdri.tonemapping_operator_normalisation

colour_hdri.**tonemapping_operator_normalisation**(*RGB: ArrayLike*, *colourspace: colour.models.rgb.rgb_colourspace.RGB_Colourspace = RGB_COLOURSPACES['sRGB']*) → numpy.ndarray

Perform given *RGB* array tonemapping using the normalisation method.

Parameters

- **RGB** (ArrayLike) – *RGB* array to perform tonemapping onto.
- **colourspace** (colour.models.rgb.rgb_colourspace.RGB_Colourspace) – *RGB* colourspace used for internal *Luminance* computation.

Returns Tonemapped *RGB* array.

Return type numpy.ndarray

References

[BADC11b]

Examples

```
>>> tonemapping_operator_normalisation(np.array(
...     [[[0.48046875, 0.35156256, 0.23632812],
...       [1.39843753, 0.55468757, 0.39062594]],
...      [[4.40625388, 2.15625895, 1.34375372],
...       [6.59375023, 3.43751395, 2.21875829]]]))
array([[[ 0.1194997...,  0.0874388...,  0.0587783...],
        [ 0.3478122...,  0.1379590...,  0.0971544...]],

       [[ 1.0959009...,  0.5362936...,  0.3342115...],
        [ 1.6399638...,  0.8549608...,  0.5518382...]]])
```

Gamma

colour_hdri

tonemapping_operator_gamma(RGB[, gamma, EV])	Perform given *RGB* array tonemapping using the gamma and exposure correction method.

colour_hdri.tonemapping_operator_gamma

colour_hdri.**tonemapping_operator_gamma**(*RGB: ArrayLike*, *gamma: Floating = 1*, *EV: Floating = 0*) → numpy.ndarray

Perform given *RGB* array tonemapping using the gamma and exposure correction method.

Parameters

- **RGB** (ArrayLike) – *RGB* array to perform tonemapping onto.
- **gamma** (Floating) – γ correction value.
- **EV** (Floating) – Exposure adjustment value.

Returns Tonemapped *RGB* array.

Return type numpy.ndarray

References

[BADC11b]

Examples

```
>>> tonemapping_operator_gamma(np.array(
...     [[[0.48046875, 0.35156256, 0.23632812],
...       [1.39843753, 0.55468757, 0.39062594]],
...      [[4.40625388, 2.15625895, 1.34375372],
...       [6.59375023, 3.43751395, 2.21875829]]]),
...      1.0, -3.0)
array([[[ 0.0600585...,  0.0439453...,  0.0295410...],
        [ 0.1748046...,  0.0693359...,  0.0488282...]],

       [[ 0.5507817...,  0.2695323...,  0.1679692...],
        [ 0.8242187...,  0.4296892...,  0.2773447...]]])
```

Logarithmic

colour_hdri

tonemapping_operator_logarithmic(RGB[, q, ...])	Perform given *RGB* array tonemapping using the logarithmic method.
tonemapping_operator_exponential(RGB[, q, ...])	Perform given *RGB* array tonemapping using the exponential method.
tonemapping_operator_logarithmic_mapping(RGB)	Perform given *RGB* array tonemapping using the logarithmic mapping method.
tonemapping_operator_exponentiation_mapping(RGB)	Perform given *RGB* array tonemapping using the exponentiation mapping method.
tonemapping_operator_Schlick1994(RGB[, p, ...])	Perform given *RGB* array tonemapping using *Schlick (1994)* method.
tonemapping_operator_Tumblin1999(RGB[, ...])	Perform given *RGB* array tonemapping using *Tumblin, Hodgins and Guenter (1999)* method.
tonemapping_operator_Reinhard2004(RGB[, f, ...])	Perform given *RGB* array tonemapping using *Reinhard and Devlin (2004)* method.
tonemapping_operator_filmic(RGB[, ...])	Perform given *RGB* array tonemapping using *Hable (2010)* method.

colour_hdri.tonemapping_operator_logarithmic

colour_hdri.**tonemapping_operator_logarithmic**(*RGB: ArrayLike, q: Floating = 1, k: Floating = 1, colourspace: colour.models.rgb.rgb_colourspace.RGB_Colourspace = RGB_COLOURSPACES['sRGB']*) → numpy.ndarray

Perform given *RGB* array tonemapping using the logarithmic method.

Parameters

- **RGB** (ArrayLike) – *RGB* array to perform tonemapping onto.
- **q** (Floating) – q.
- **k** (Floating) – k.
- **colourspace** (colour.models.rgb.rgb_colourspace.RGB_Colourspace) – *RGB* colourspace used for internal *Luminance* computation.

Returns Tonemapped *RGB* array.

Return type numpy.ndarray

References

[BADC11b]

Examples

```
>>> tonemapping_operator_logarithmic(np.array(
...     [[[0.48046875, 0.35156256, 0.23632812],
...       [1.39843753, 0.55468757, 0.39062594]],
...      [[4.40625388, 2.15625895, 1.34375372],
...       [6.59375023, 3.43751395, 2.21875829]]]),
...       1.0, 25)
array([[[ 0.0884587...,  0.0647259...,  0.0435102...],
        [ 0.2278222...,  0.0903652...,  0.0636376...]],

       [[ 0.4717487...,  0.2308565...,  0.1438669...],
        [ 0.5727396...,  0.2985858...,  0.1927235...]]])
```

colour_hdri.tonemapping_operator_exponential

colour_hdri.**tonemapping_operator_exponential**(*RGB: ArrayLike, q: Floating = 1, k: Floating = 1, colourspace: colour.models.rgb.rgb_colourspace.RGB_Colourspace = RGB_COLOURSPACES['sRGB']*) → numpy.ndarray

Perform given *RGB* array tonemapping using the exponential method.

Parameters

- **RGB** (ArrayLike) – *RGB* array to perform tonemapping onto.
- **q** (Floating) – q.
- **k** (Floating) – k.
- **colourspace** (colour.models.rgb.rgb_colourspace.RGB_Colourspace) – *RGB* colourspace used for internal *Luminance* computation.

Returns Tonemapped *RGB* array.

Return type `numpy.ndarray`

References

[BADC11b]

Examples

```
>>> tonemapping_operator_exponential(np.array(
...     [[[0.48046875, 0.35156256, 0.23632812],
...       [1.39843753, 0.55468757, 0.39062594]],
...      [[4.40625388, 2.15625895, 1.34375372],
...       [6.59375023, 3.43751395, 2.21875829]]]),
...       1.0, 25)
array([[[ 0.0148082...,  0.0108353...,  0.0072837...],
        [ 0.0428669...,  0.0170031...,  0.0119740...]],

       [[ 0.1312736...,  0.0642404...,  0.0400338...],
        [ 0.1921684...,  0.1001830...,  0.0646635...]]])
```

colour_hdri.tonemapping_operator_logarithmic_mapping

`colour_hdri.`**`tonemapping_operator_logarithmic_mapping`**(*RGB: ArrayLike, p: Floating = 1, q: Floating = 1, colourspace: colour.models.rgb.rgb_colourspace.RGB_Colourspace = RGB_COLOURSPACES['sRGB']*) → numpy.ndarray

Perform given *RGB* array tonemapping using the logarithmic mapping method.

Parameters

- **RGB** (`ArrayLike`) – *RGB* array to perform tonemapping onto.
- **p** (`Floating`) – p.
- **q** (`Floating`) – q.
- **colourspace** (`colour.models.rgb.rgb_colourspace.RGB_Colourspace`) – *RGB* colourspace used for internal *Luminance* computation.

Returns Tonemapped *RGB* array.

Return type `numpy.ndarray`

References

[Sch94]

Examples

```
>>> tonemapping_operator_logarithmic_mapping(np.array(
...     [[[0.48046875, 0.35156256, 0.23632812],
...       [1.39843753, 0.55468757, 0.39062594]],
...      [[4.40625388, 2.15625895, 1.34375372],
...       [6.59375023, 3.43751395, 2.21875829]]]))
array([[[ 0.2532899...,  0.1853341...,  0.1245857...],
        [ 0.6523387...,  0.2587489...,  0.1822179...]],

       [[ 1.3507897...,  0.6610269...,  0.4119437...],
        [ 1.6399638...,  0.8549608...,  0.5518382...]]])
```

colour_hdri.tonemapping_operator_exponentiation_mapping

colour_hdri.**tonemapping_operator_exponentiation_mapping**(*RGB: ArrayLike, p: Floating = 1, q: Floating = 1, colourspace: colour.models.rgb.rgb_colourspace.RGB_Colourspace = RGB_COLOURSPACES['sRGB']*) → numpy.ndarray

Perform given *RGB* array tonemapping using the exponentiation mapping method.

Parameters

- **RGB** (ArrayLike) – *RGB* array to perform tonemapping onto.
- **p** (Floating) – p.
- **q** (Floating) – q.
- **colourspace** (colour.models.rgb.rgb_colourspace.RGB_Colourspace) – *RGB* colourspace used for internal *Luminance* computation.

Returns Tonemapped *RGB* array.

Return type numpy.ndarray

References

[Sch94]

Examples

```
>>> tonemapping_operator_exponentiation_mapping(np.array(
...     [[[0.48046875, 0.35156256, 0.23632812],
...       [1.39843753, 0.55468757, 0.39062594]],
...      [[4.40625388, 2.15625895, 1.34375372],
...       [6.59375023, 3.43751395, 2.21875829]]]))
array([[[ 0.1194997...,  0.0874388...,  0.0587783...],
        [ 0.3478122...,  0.1379590...,  0.0971544...]],

       [[ 1.0959009...,  0.5362936...,  0.3342115...],
        [ 1.6399638...,  0.8549608...,  0.5518382...]]])
```

colour_hdri.tonemapping_operator_Schlick1994

colour_hdri.**tonemapping_operator_Schlick1994**(*RGB: ArrayLike, p: Floating = 1, colourspace: colour.models.rgb.rgb_colourspace.RGB_Colourspace = RGB_COLOURSPACES['sRGB']*) → numpy.ndarray

Perform given *RGB* array tonemapping using *Schlick (1994)* method.

Parameters

- **RGB** (ArrayLike) – *RGB* array to perform tonemapping onto.
- **p** (Floating) – p.
- **colourspace** (colour.models.rgb.rgb_colourspace.RGB_Colourspace) – *RGB* colourspace used for internal *Luminance* computation.

Returns Tonemapped *RGB* array.

Return type numpy.ndarray

References

[BADC11b], [Sch94]

Examples

```
>>> tonemapping_operator_Schlick1994(np.array(
...     [[[0.48046875, 0.35156256, 0.23632812],
...       [1.39843753, 0.55468757, 0.39062594]],
...      [[4.40625388, 2.15625895, 1.34375372],
...       [6.59375023, 3.43751395, 2.21875829]]]))
array([[[ 0.1194997...,  0.0874388...,  0.0587783...],
        [ 0.3478122...,  0.1379590...,  0.0971544...]],

       [[ 1.0959009...,  0.5362936...,  0.3342115...],
        [ 1.6399638...,  0.8549608...,  0.5518382...]]])
```

colour_hdri.tonemapping_operator_Tumblin1999

colour_hdri.**tonemapping_operator_Tumblin1999**(*RGB: ArrayLike, L_da: Floating = 20, C_max: Floating = 100, L_max: Floating = 100, colourspace: colour.models.rgb.rgb_colourspace.RGB_Colourspace = RGB_COLOURSPACES['sRGB']*) → numpy.ndarray

Perform given *RGB* array tonemapping using *Tumblin, Hodgins and Guenter (1999)* method.

Parameters

- **RGB** (ArrayLike) – *RGB* array to perform tonemapping onto.
- **L_da** (Floating) – L_{da} display adaptation luminance, a mid-range display value.
- **C_max** (Floating) – C_{max} maximum contrast available from the display.
- **L_max** (Floating) – L_{max} maximum display luminance.
- **colourspace** (colour.models.rgb.rgb_colourspace.RGB_Colourspace) – *RGB* colourspace used for internal *Luminance* computation.

Returns Tonemapped *RGB* array.

Return type numpy.ndarray

References

[THG99]

Examples

```
>>> tonemapping_operator_Tumblin1999(np.array(
...     [[[0.48046875, 0.35156256, 0.23632812],
...       [1.39843753, 0.55468757, 0.39062594]],
...      [[4.40625388, 2.15625895, 1.34375372],
...       [6.59375023, 3.43751395, 2.21875829]]]))
array([[[ 0.0400492...,  0.0293043...,  0.0196990...],
        [ 0.1019768...,  0.0404489...,  0.0284852...]],

       [[ 0.2490212...,  0.1218618...,  0.0759427...],
        [ 0.3408366...,  0.1776880...,  0.1146895...]]])
```

colour_hdri.tonemapping_operator_Reinhard2004

`colour_hdri.`**`tonemapping_operator_Reinhard2004`**(*RGB: ArrayLike, f: Floating = 0, m: Floating = 0.3, a: Floating = 0, c: Floating = 0, colourspace: colour.models.rgb.rgb_colourspace.RGB_Colourspace = RGB_COLOURSPACES['sRGB']*) → numpy.ndarray

Perform given *RGB* array tonemapping using *Reinhard and Devlin (2004)* method.

Parameters

- **RGB** (`ArrayLike`) – *RGB* array to perform tonemapping onto.
- **f** (`Floating`) – f.
- **m** (`Floating`) – m.
- **a** (`Floating`) – a.
- **c** (`Floating`) – c.
- **colourspace** (`colour.models.rgb.rgb_colourspace.RGB_Colourspace`) – *RGB* colourspace used for internal *Luminance* computation.

Returns Tonemapped *RGB* array.

Return type `numpy.ndarray`

References

[RD05]

Examples

```
>>> tonemapping_operator_Reinhard2004(np.array(
...     [[[0.48046875, 0.35156256, 0.23632812],
...       [1.39843753, 0.55468757, 0.39062594]],
...      [[4.40625388, 2.15625895, 1.34375372],
...       [6.59375023, 3.43751395, 2.21875829]]]),
...     -10)
array([[[ 0.0216792...,  0.0159556...,  0.0107821...],
        [ 0.0605894...,  0.0249445...,  0.0176972...]],

       [[ 0.1688972...,  0.0904532...,  0.0583584...],
        [ 0.2331935...,  0.1368456...,  0.0928316...]]])
```

colour_hdri.tonemapping_operator_filmic

`colour_hdri.`**`tonemapping_operator_filmic`**(*RGB: ArrayLike, shoulder_strength: Floating = 0.22, linear_strength: Floating = 0.3, linear_angle: Floating = 0.1, toe_strength: Floating = 0.2, toe_numerator: Floating = 0.01, toe_denominator: Floating = 0.3, exposure_bias: Floating = 2, linear_whitepoint: Floating = 11.2*) → numpy.ndarray

Perform given *RGB* array tonemapping using *Habble (2010)* method.

Parameters

- **`RGB`** (`ArrayLike`) – *RGB* array to perform tonemapping onto.
- **`shoulder_strength`** (`Floating`) – Shoulder strength.
- **`linear_strength`** (`Floating`) – Linear strength.
- **`linear_angle`** (`Floating`) – Linear angle.
- **`toe_strength`** (`Floating`) – Toe strength.
- **`toe_numerator`** (`Floating`) – Toe numerator.
- **`toe_denominator`** (`Floating`) – Toe denominator.
- **`exposure_bias`** (`Floating`) – Exposure bias.
- **`linear_whitepoint`** (`Floating`) – Linear whitepoint.

Returns Tonemapped *RGB* array.

Return type `numpy.ndarray`

References

[Hab10a], [Hab10b]

Examples

```
>>> tonemapping_operator_filmic(np.array(
...     [[[0.48046875, 0.35156256, 0.23632812],
...       [1.39843753, 0.55468757, 0.39062594]],
...      [[4.40625388, 2.15625895, 1.34375372],
...       [6.59375023, 3.43751395, 2.21875829]]]))
array([[[ 0.4507954...,  0.3619673...,  0.2617269...],
        [ 0.7567191...,  0.4933310...,  0.3911730...]],

       [[ 0.9725554...,  0.8557374...,  0.7465713...],
        [ 1.0158782...,  0.9382937...,  0.8615161...]]])
```

Logarithmic Mapping

colour_hdri

tonemapping_operator_logarithmic_mapping(RGB)	Perform given *RGB* array tonemapping using the logarithmic mapping method.

Exponential

colour_hdri

tonemapping_operator_exponential(RGB[, q, ...])	Perform given *RGB* array tonemapping using the exponential method.

Exponentiation Mapping

colour_hdri

tonemapping_operator_exponentiation_mapping(RGB)	Perform given *RGB* array tonemapping using the exponentiation mapping method.
tonemapping_operator_Schlick1994(RGB[, p, ...])	Perform given *RGB* array tonemapping using *Schlick (1994)* method.
tonemapping_operator_Tumblin1999(RGB[, ...])	Perform given *RGB* array tonemapping using *Tumblin, Hodgins and Guenter (1999)* method.
tonemapping_operator_Reinhard2004(RGB[, f, ...])	Perform given *RGB* array tonemapping using *Reinhard and Devlin (2004)* method.
tonemapping_operator_filmic(RGB[, ...])	Perform given *RGB* array tonemapping using *Habble (2010)* method.

Schlick (1994)

colour_hdri

tonemapping_operator_Schlick1994(RGB[, p, ...])	Perform given *RGB* array tonemapping using *Schlick (1994)* method.

Tumblin, Hodgins and Guenter (1999)

colour_hdri

tonemapping_operator_Tumblin1999(RGB[, ...])	Perform given *RGB* array tonemapping using *Tumblin, Hodgins and Guenter (1999)* method.

Reinhard and Devlin (2004)

colour_hdri

tonemapping_operator_Reinhard2004(RGB[, f, ...])	Perform given *RGB* array tonemapping using *Reinhard and Devlin (2004)* method.

Habble (2010) - Filmic

colour_hdri

tonemapping_operator_filmic(RGB[, ...])	Perform given *RGB* array tonemapping using *Habble (2010)* method.

Utilities

Common

colour_hdri

vivification()	Implement supports for vivification of the underlying dict like data-structure, magical!
vivified_to_dict(vivified)	Convert given vivified data-structure to dictionary.
path_exists(path)	Return whether given path exists.
filter_files(directory, extensions)	Filter given directory for files matching given extensions.

colour_hdri.vivification

colour_hdri.**vivification**() → collections.defaultdict

Implement supports for vivification of the underlying dict like data-structure, magical!

Return type defaultdict

Examples

```
>>> vivified = vivification()
>>> vivified['my']['attribute'] = 1
>>> vivified['my']
defaultdict(<function vivification at 0x...>, {u'attribute': 1})
>>> vivified['my']['attribute']
1
```

colour_hdri.vivified_to_dict

colour_hdri.**vivified_to_dict**(*vivified: Union[Dict, collections.defaultdict]*) → Dict

Convert given vivified data-structure to dictionary.

Parameters **vivified** (Union[Dict, collections.defaultdict]) – Vivified data-structure.

Return type dict

Examples

```
>>> vivified = vivification()
>>> vivified['my']['attribute'] = 1
>>> vivified_to_dict(vivified)
{u'my': {u'attribute': 1}}
```

colour_hdri.path_exists

colour_hdri.**path_exists**(*path: Optional[str]*) → Boolean

Return whether given path exists.

Parameters **path** (Optional[str]) – Path to check the existence.

Returns Whether given path exists.

Return type bool

Examples

```
>>> path_exists(__file__)
True
>>> path_exists('')
False
```

colour_hdri.filter_files

`colour_hdri.filter_files(directory: str, extensions: Sequence[str]) → List[str]`

Filter given directory for files matching given extensions.

Parameters

- **directory** (str) – Directory to filter.
- **extensions** (Sequence[str]) – Extensions to filter on.

Returns Filtered files.

Return type list

EXIF Data Manipulation

colour_hdri

EXIF_EXECUTABLE	Command line EXIF manipulation application, usually Phil Harvey's *ExifTool*.
EXIFTag(group, name, value, identifier)	EXIF tag data.
parse_exif_string(exif_tag)	Parse given EXIF tag assuming it is a string and return its value.
parse_exif_number(exif_tag[, dtype])	Parse given EXIF tag assuming it is a number type and return its value.
parse_exif_fraction(exif_tag[, dtype])	Parse given EXIF tag assuming it is a fraction and return its value.
parse_exif_array(exif_tag[, dtype, shape])	Parse given EXIF tag assuming it is an array and return its value.
parse_exif_data(data)	Parse given EXIF data output from *exiftool*.
read_exif_tags(image)	Return given image EXIF image tags.
copy_exif_tags(source, target)	Copy given source image file EXIF tag to given image target.
update_exif_tags(images)	Update given images pairs EXIF tags.
delete_exif_tags(image)	Delete all given image EXIF tags.
read_exif_tag(image, tag)	Return given image EXIF tag value.
write_exif_tag(image, tag, value)	Set given image EXIF tag value.

colour_hdri.EXIF_EXECUTABLE

`colour_hdri.EXIF_EXECUTABLE = 'exiftool'`

Command line EXIF manipulation application, usually Phil Harvey's *ExifTool*.

colour_hdri.EXIFTag

`class colour_hdri.EXIFTag(group: typing.Optional[str] = <factory>, name: typing.Optional[str] = <factory>, value: typing.Optional[str] = <factory>, identifier: typing.Optional[str] = <factory>)`

EXIF tag data.

Parameters

- **group** (Optional[str]) – EXIF tag group name.
- **name** (Optional[str]) – EXIF tag name.
- **value** (Optional[str]) – EXIF tag value.

- **identifier** (Optional[str]) – EXIF tag identifier.

Return type None

__init__(*group: typing.Optional[str] = <factory>, name: typing.Optional[str] = <factory>, value: typing.Optional[str] = <factory>, identifier: typing.Optional[str] = <factory>*) → None

Parameters

- **group** (Optional[str]) –
- **name** (Optional[str]) –
- **value** (Optional[str]) –
- **identifier** (Optional[str]) –

Return type None

Methods

`__init__`([group, name, value, identifier])

Attributes

`group`
`name`
`value`
`identifier`

colour_hdri.parse_exif_string

`colour_hdri.`**`parse_exif_string`**(*exif_tag: colour_hdri.utilities.exif.EXIFTag*) → str

Parse given EXIF tag assuming it is a string and return its value.

Parameters **exif_tag** (`colour_hdri.utilities.exif.EXIFTag`) – EXIF tag to parse.

Returns Parsed EXIF tag value.

Return type `str`

colour_hdri.parse_exif_number

`colour_hdri.`**`parse_exif_number`**(*exif_tag: EXIFTag, dtype: Optional[Type[DTypeNumber]] = None*) → Number

Parse given EXIF tag assuming it is a number type and return its value.

Parameters

- **exif_tag** (`EXIFTag`) – EXIF tag to parse.
- **dtype** (`Optional[Type[DTypeNumber]]`) – Return value data type.

Returns Parsed EXIF tag value.

Return type `numpy.floating` or `numpy.integer`

colour_hdri.parse_exif_fraction

`colour_hdri.`**`parse_exif_fraction`**(*exif_tag: EXIFTag*, *dtype: Optional[Type[DTypeFloating]] = None*) → Floating

Parse given EXIF tag assuming it is a fraction and return its value.

Parameters

- **exif_tag** (`EXIFTag`) – EXIF tag to parse.
- **dtype** (`Optional[Type[DTypeFloating]]`) – Return value data type.

Returns Parsed EXIF tag value.

Return type `numpy.floating`

colour_hdri.parse_exif_array

`colour_hdri.`**`parse_exif_array`**(*exif_tag: EXIFTag*, *dtype: Optional[Type[DTypeNumber]] = None*, *shape: Optional[Union[SupportsIndex, Sequence[SupportsIndex]]] = None*) → NDArray

Parse given EXIF tag assuming it is an array and return its value.

Parameters

- **exif_tag** (`EXIFTag`) – EXIF tag to parse.
- **dtype** (`Optional[Type[DTypeNumber]]`) – Return value data type.
- **shape** (`Optional[Union[SupportsIndex, Sequence[SupportsIndex]]]`) – Shape of the array to be returned.

Returns Parsed EXIF tag value.

Return type `numpy.ndarray`

colour_hdri.parse_exif_data

`colour_hdri.`**`parse_exif_data`**(*data: str*) → List

Parse given EXIF data output from *exiftool*.

Parameters **data** (`str`) – EXIF data output.

Returns Parsed EXIF data output.

Return type `list`

Raises **`ValueError`** – If the EXIF data output cannot be parsed.

colour_hdri.read_exif_tags

colour_hdri.**read_exif_tags**(*image: str*) → collections.defaultdict
Return given image EXIF image tags.

Parameters **image** (str) – Image file.

Returns EXIF tags.

Return type defaultdict

colour_hdri.copy_exif_tags

colour_hdri.**copy_exif_tags**(*source: str*, *target: str*) → Boolean
Copy given source image file EXIF tag to given image target.

Parameters

- **source** (str) – Source image file.
- **target** (str) – Target image file.

Returns Definition success.

Return type bool

colour_hdri.update_exif_tags

colour_hdri.**update_exif_tags**(*images: Sequence[Sequence[str]]*) → Boolean
Update given images pairs EXIF tags.

Parameters **images** (Sequence[Sequence[str]]) – Image pairs to update the EXIF tags of.

Returns Definition success.

Return type bool

colour_hdri.delete_exif_tags

colour_hdri.**delete_exif_tags**(*image: str*) → Boolean
Delete all given image EXIF tags.

Parameters **image** (str) – Image file to delete the EXIF tags from.

Returns Definition success.

Return type bool

colour_hdri.read_exif_tag

colour_hdri.**read_exif_tag**(*image: str*, *tag: str*) → str
Return given image EXIF tag value.

Parameters

- **image** (str) – Image file to read the EXIF tag value of.
- **tag** (str) – Tag to read the value of.

Returns Tag value.

Return type str

colour_hdri.write_exif_tag

colour_hdri.**write_exif_tag**(*image: str*, *tag: str*, *value: str*) → Boolean

Set given image EXIF tag value.

Parameters

- **image** (str) – Image file to set the EXIF tag value of.
- **tag** (str) – Tag to set the value of.
- **value** (str) – Value to set.

Returns Definition success.

Return type bool

Image Data & Metadata Utilities

colour_hdri

Metadata(f_number, exposure_time, iso, ...)	Define the base object for storing exif metadata relevant to HDRI / radiance image generation.
Image([path, data, metadata])	Define the base object for storing an image along its path, pixel data and metadata needed for HDRI / radiance images generation.
ImageStack()	Define a convenient stack storing a sequence of images for HDRI / radiance images generation.

colour_hdri.Metadata

class colour_hdri.**Metadata**(*f_number: typing.Optional[numpy.ndarray] = <factory>*, *exposure_time: typing.Optional[numpy.ndarray] = <factory>*, *iso: typing.Optional[numpy.ndarray] = <factory>*, *black_level: typing.Optional[numpy.ndarray] = <factory>*, *white_level: typing.Optional[numpy.ndarray] = <factory>*, *white_balance_multipliers: typing.Optional[numpy.ndarray] = <factory>*)

Bases: colour.utilities.array.MixinDataclassArray

Define the base object for storing exif metadata relevant to HDRI / radiance image generation.

Parameters

- **f_number** (Optional[numpy.ndarray]) – Image *FNumber*.
- **exposure_time** (Optional[numpy.ndarray]) – Image *Exposure Time*.
- **iso** (Optional[numpy.ndarray]) – Image *ISO*.
- **black_level** (Optional[numpy.ndarray]) – Image *Black Level*.
- **white_level** (Optional[numpy.ndarray]) – Image *White Level*.
- **white_balance_multipliers** (Optional[numpy.ndarray]) – Image white balance multipliers, usually the *As Shot Neutral* matrix.

Return type None

colour_hdri.Image

class colour_hdri.**Image**(*path: Optional[str] = None*, *data: Optional[ArrayLike] = None*, *metadata: Optional[Metadata] = None*)

Bases: `object`

Define the base object for storing an image along its path, pixel data and metadata needed for HDRI / radiance images generation.

Parameters

- **path** (Optional[str]) – Image path.
- **data** (Optional[ArrayLike]) – Image pixel data array.
- **metadata** (Optional[Metadata]) – Image exif metadata.

Attributes

- `colour_hdri.Image.path`
- `colour_hdri.Image.data`
- `colour_hdri.Image.metadata`

Methods

- `colour_hdri.Image.__init__()`
- `colour_hdri.Image.read_data()`
- `colour_hdri.Image.read_metadata()`

property path: Optional[str]

Getter and setter property for the image path.

Parameters **value** – Value to set the image path with.

Returns Image path.

Return type `None` or `str`

property data: Optional[numpy.ndarray]

Getter and setter property for the image data.

Parameters **value** – Value to set the image data with.

Returns Image data.

Return type `None` or `numpy.ndarray`

property metadata: Optional[colour_hdri.utilities.image.Metadata]

Getter and setter property for the image metadata.

Parameters **value** – Value to set the image metadata with.

Returns Image metadata.

Return type `None` or `colour_hdri.Metadata`

read_data(*cctf_decoding: Optional[Callable] = None*) → numpy.ndarray

Read image pixel data at `Image.path` attribute.

Parameters **cctf_decoding** (Optional[Callable]) – Decoding colour component transfer function (Decoding CCTF) or electro-optical transfer function (EOTF / EOCF).

Returns Image pixel data.

Return type `numpy.ndarray`

Raises `ValueError` – If the image path is undefined.

read_metadata() → *colour_hdri.utilities.image.Metadata*

Read image relevant exif metadata at `Image.path` attribute.

Returns Image relevant exif metadata.

Return type `colour_hdri.Metadata`

Raises `ValueError` – If the image path is undefined.

colour_hdri.ImageStack

class colour_hdri.**ImageStack**

Bases: `collections.abc.MutableSequence`

Define a convenient stack storing a sequence of images for HDRI / radiance images generation.

Methods

- `colour_hdri.ImageStack.__init__()`
- `colour_hdri.ImageStack.__getitem__()`
- `colour_hdri.ImageStack.__setitem__()`
- `colour_hdri.ImageStack.__delitem__()`
- `colour_hdri.ImageStack.__len__()`
- `colour_hdri.ImageStack.__getattr__()`
- `colour_hdri.ImageStack.__setattr__()`
- `colour_hdri.ImageStack.sort()`
- `colour_hdri.ImageStack.insert()`
- `colour_hdri.ImageStack.from_files()`

insert(*index: Integer*, *value: Any*)

Insert given `colour_hdri.Image` class instance at given index.

Parameters

- **index** (`Integer`) – `colour_hdri.Image` class instance index.
- **value** (`Any`) – `colour_hdri.Image` class instance to set.

sort(*key: Optional[Callable] = None*)

Sort the underlying data structure.

Parameters **key** (`Optional[Callable]`) – Function of one argument that is used to extract a comparison key from each data structure.

static from_files(*image_files: Sequence[str]*, *cctf_decoding: Optional[Callable] = None*) → *colour_hdri.utilities.image.ImageStack*

Return a `colour_hdri.ImageStack` instance from given image files.

Parameters

- **image_files** (`Sequence[str]`) – Image files.
- **cctf_decoding** (`Optional[Callable]`) – Decoding colour component transfer function (Decoding CCTF) or electro-optical transfer function (EOTF / EOCF).

Return type `colour_hdri.ImageStack`

3.1.2 Indices and tables

- genindex
- search

CHAPTER

FOUR

1.4 SEE ALSO

4.1 1.4.1 Publications

- Advanced High Dynamic Range Imaging: Theory and Practice by Banterle, F. et al.

Advanced High Dynamic Range Imaging: Theory and Practice was used as a reference for some of the algorithms of **Colour - HDRI**.

4.2 1.4.2 Software

C/C++

- OpenCV by Bradski, G.
- Piccante by Banterle, F. and Benedetti, L.,

Piccante was used to verify the Grossberg (2003) Histogram Based Image Sampling.

Matlab

- HDR Toolbox by Banterle, F. et al.

CHAPTER

FIVE

1.5 CODE OF CONDUCT

The *Code of Conduct*, adapted from the Contributor Covenant 1.4, is available on the Code of Conduct page.

CHAPTER

SIX

1.6 CONTACT & SOCIAL

The *Colour Developers* can be reached via different means:

- Email
- Discourse
- Facebook
- Github Discussions
- Gitter
- Twitter

CHAPTER

SEVEN

1.7 ABOUT

Colour - HDRI by Colour Developers

https://github.com/colour-science/colour-hdri

BIBLIOGRAPHY

[BADC11a] Francesco Banterle, Alessandro Artusi, Kurt Debattista, and Alan Chalmers. *2.1.1 Generating HDR Content by Combining Multiple Exposures*. A K Peters/CRC Press, 2011. ISBN 978-1-56881-719-4.

[BADC11b] Francesco Banterle, Alessandro Artusi, Kurt Debattista, and Alan Chalmers. 3.2.1 simple mapping methods. In *Advanced High Dynamic Range Imaging*, pages 38–41. A K Peters/CRC Press, 2011.

[BB14] Francesco Banterle and Luca Benedetti. Piccante: an open and portable library for hdr imaging. 2014.

[Cof15] Dave Coffin. Dcraw. 2015. URL: https://www.cybercom.net/~dcoffin/dcraw/.

[DM97] Paul E. Debevec and Jitendra Malik. Recovering high dynamic range radiance maps from photographs. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques - SIGGRAPH '97*, 369–378. New York, New York, USA, 1997. ACM Press. doi:10.1145/258734.258884.

[GN03] M.D. Grossberg and S.K. Nayar. Determining the camera response from images: what is knowable? *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(11):1455–1467, November 2003. doi:10.1109/TPAMI.2003.1240119.

[Hab10a] John Habble. Filmic tonemapping operators. 2010. URL: http://filmicgames.com/archives/75 (visited on 2015-03-15).

[Hab10b] John Habble. Uncharted 2: hdr lighting. 2010. URL: http://www.slideshare.net/ozlael/hable-john-uncharted2-hdr-lighting (visited on 2015-03-15).

[LLJ16] Sebastien Lagarde, Sebastien Lachambre, and Cyril Jover. An artist-friendly workflow for panoramic hdri. 2016. URL: http://blog.selfshadow.com/publications/s2016-shading-course/unity/s2016_pbs_unity_hdri_notes.pdf.

[LdeRousiers14] Sébastian Lagarde and Charles de Rousiers. Moving frostbite to physically based rendering 3.0. *Siggraph 2014*, pages 119, 2014.

[McG12] Sandy McGuffog. Hue twists in dng camera profiles. 2012. URL: http://dcptool.sourceforge.net/Hue\T1\textbackslash{}%20Twists.html (visited on 2016-10-29).

[RD05] Erik Reinhard and Kate Devlin. Dynamic range reduction inspired by photoreceptor physiology. *IEEE Transactions on Visualization and Computer Graphics*, 11(01):13–24, January 2005. doi:10.1109/TVCG.2005.9.

[Sch94] Christophe Schlick. Quantization techniques for visualization of high dynamic range pictures. *Proceedings of the Fifth Eurographics Workshop on Rendering*, pages 7–18, 1994.

[THG99] Jack Tumblin, Jessica K. Hodgins, and Brian K. Guenter. Two methods for display of high contrast images. *ACM Transactions on Graphics*, 18(1):56–94, January 1999. doi:10.1145/300776.300783.

[VD09] Kuntee Viriyothai and Paul Debevec. Variance minimization light probe sampling. In *SIGGRAPH '09: Posters on - SIGGRAPH '09*, 1–1. New York, New York, USA, 2009. ACM Press. doi:10.1145/1599301.1599393.

[AdobeSystems12a] Adobe Systems. Camera to xyz (d50) transform. In *Digital Negative (DNG) Specification*, pages 81. 2012.

[AdobeSystems12b] Adobe Systems. Digital negative (dng) specification. 2012.

[AdobeSystems12c] Adobe Systems. Translating camera neutral coordinates to white balance xy coordinates. In *Digital Negative (DNG) Specification*, pages 80–81. 2012.

[AdobeSystems12d] Adobe Systems. Translating white balance xy coordinates to camera neutral coordinates. In *Digital Negative (DNG) Specification*, pages 80. 2012.

[AdobeSystems15a] Adobe Systems. Adobe dng sdk 1.4. 2015. URL: http://download.adobe.com/pub/adobe/dng/dng_sdk_1_4.zip.

[AdobeSystems15b] Adobe Systems. Adobe dng sdk 1.4 - dng_sdk_1_4/dng_sdk/source/dng_camera_profile.cpp - dng_camera_profile::illuminanttotemperature. 2015. URL: http://download.adobe.com/pub/adobe/dng/dng_sdk_1_4.zip.

[AdobeSystems15c] Adobe Systems. Adobe dng sdk 1.4 - dng_sdk_1_4/dng_sdk/source/dng_tag_values.h - lightsource tag. 2015. URL: http://download.adobe.com/pub/adobe/dng/dng_sdk_1_4.zip.

[ISO06] ISO. International standard iso12232-2006 - photography - digital still cameras - determination of exposure index, iso speed ratings, standard output sensitivity, and recommended exposure index. 2006.

[Wikipediaa] Wikipedia. Ev as a measure of luminance and illuminance. URL: https://en.wikipedia.org/wiki/Exposure_value\T1\textbackslash{}#EV_as_a_measure_of_luminance_and_illuminance (visited on 2015-11-14).

[Wikipediab] Wikipedia. Tonemapping - purpose and methods. URL: http://en.wikipedia.org/wiki/Tone_mapping\T1\textbackslash{}#Purpose_and_methods (visited on 2015-03-15).

INDEX