COLOUR - DEMOSAICING

Colour - Demosaicing Documentation

Release 0.2.5

Colour Developers

Feb 04, 2024

CONTENTS

1 1.1 Features **3**
1.1 1.1.1 Examples 3

2 1.2 User Guide **5**
2.1 User Guide 5

3 1.3 API Reference **7**
3.1 API Reference 7

4 1.4 Code of Conduct **15**

5 1.5 Contact & Social **17**

6 1.6 About **19**

Bibliography **21**

Index **23**

A Python package implementing various CFA (Colour Filter Array) demosaicing algorithms and related utilities.

It is open source and freely available under the BSD-3-Clause terms.

CHAPTER

ONE

1.1 FEATURES

The following CFA (Colour Filter Array) demosaicing algorithms are implemented:

- Bilinear
- Malvar (2004)
- DDFAPD - Menon (2007)

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 - Demosaicing** 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 - Demosaicing requires various dependencies in order to run:

- python >= 3.9, < 4
- colour-science >= 4.3
- imageio >= 2, < 3
- numpy >= 1.22, < 2
- scipy >= 1.8, < 2

Pypi

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

```
pip install --user colour-demosaicing
```

The overall development dependencies are installed as follows:

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

2.1.2 Bibliography

CHAPTER

THREE

1.3 API REFERENCE

3.1 API Reference

3.1.1 Colour - Demosaicing

Bayer CFA Demosaicing and Mosaicing

Demosaicing

colour_demosaicing

demosaicing_CFA_Bayer_bilinear(CFA[, pattern])	Return the demosaiced *RGB* colourspace array from given *Bayer* CFA using bilinear interpolation.
demosaicing_CFA_Bayer_Malvar2004(CFA[, pattern])	Return the demosaiced *RGB* colourspace array from given *Bayer* CFA using *Malvar (2004)* demosaicing algorithm.
demosaicing_CFA_Bayer_Menon2007(CFA[, ...])	Return the demosaiced *RGB* colourspace array from given *Bayer* CFA using DDFAPD - *Menon (2007)* demosaicing algorithm.

colour_demosaicing.demosaicing_CFA_Bayer_bilinear

colour_demosaicing.**demosaicing_CFA_Bayer_bilinear**(*CFA: ArrayLike*, *pattern: Literal['RGGB', 'BGGR', 'GRBG', 'GBRG'] | str = 'RGGB'*) → NDArrayFloat

Return the demosaiced *RGB* colourspace array from given *Bayer* CFA using bilinear interpolation.

Parameters

- **CFA** (ArrayLike) – *Bayer* CFA.
- **pattern** (Literal['RGGB', 'BGGR', 'GRBG', 'GBRG'] | str) – Arrangement of the colour filters on the pixel array.

Returns

RGB colourspace array.

Return type

numpy.ndarray

Notes

- The definition output is not clipped in range [0, 1] : this allows for direct HDRI image generation on *Bayer* CFA data and post demosaicing of the high dynamic range data as showcased in this Jupyter Notebook.

References

[LMY10]

Examples

```
>>> import numpy as np
>>> CFA = np.array(
...     [
...         [0.30980393, 0.36078432, 0.30588236, 0.3764706],
...         [0.35686275, 0.39607844, 0.36078432, 0.40000001],
...     ]
... )
>>> demosaicing_CFA_Bayer_bilinear(CFA)
array([[[ 0.69705884,  0.17941177,  0.09901961],
        [ 0.46176472,  0.4509804 ,  0.19803922],
        [ 0.45882354,  0.27450981,  0.19901961],
        [ 0.22941177,  0.5647059 ,  0.30000001]],

       [[ 0.23235295,  0.53529412,  0.29705883],
        [ 0.15392157,  0.26960785,  0.59411766],
        [ 0.15294118,  0.4509804 ,  0.59705884],
        [ 0.07647059,  0.18431373,  0.90000002]]])
>>> CFA = np.array(
...     [
...         [0.3764706, 0.360784320, 0.40784314, 0.3764706],
...         [0.35686275, 0.30980393, 0.36078432, 0.29803923],
...     ]
... )
>>> demosaicing_CFA_Bayer_bilinear(CFA, "BGGR")
array([[[ 0.07745098,  0.17941177,  0.84705885],
        [ 0.15490197,  0.4509804 ,  0.5882353 ],
        [ 0.15196079,  0.27450981,  0.61176471],
        [ 0.22352942,  0.5647059 ,  0.30588235]],

       [[ 0.23235295,  0.53529412,  0.28235295],
        [ 0.4647059 ,  0.26960785,  0.19607843],
        [ 0.45588237,  0.4509804 ,  0.20392157],
        [ 0.67058827,  0.18431373,  0.10196078]]])
```

colour_demosaicing.demosaicing_CFA_Bayer_Malvar2004

colour_demosaicing.**demosaicing_CFA_Bayer_Malvar2004**(*CFA: ArrayLike*, *pattern: Literal['RGGB', 'BGGR', 'GRBG', 'GBRG'] | str = 'RGGB'*) → NDArrayFloat

Return the demosaiced *RGB* colourspace array from given *Bayer* CFA using *Malvar (2004)* demosaicing algorithm.

Parameters

- **CFA** (ArrayLike) – *Bayer* CFA.
- **pattern** (Literal['RGGB', 'BGGR', 'GRBG', 'GBRG'] | str) – Arrangement of the colour filters on the pixel array.

Returns
RGB colourspace array.

Return type
numpy.ndarray

Notes

- The definition output is not clipped in range [0, 1] : this allows for direct HDRI image generation on *Bayer* CFA data and post demosaicing of the high dynamic range data as showcased in this Jupyter Notebook.

References

[MHCW04]

Examples

```
>>> CFA = np.array(
...     [
...         [0.30980393, 0.36078432, 0.30588236, 0.3764706],
...         [0.35686275, 0.39607844, 0.36078432, 0.40000001],
...     ]
... )
>>> demosaicing_CFA_Bayer_Malvar2004(CFA)
array([[[ 0.30980393,  0.31666668,  0.32941177],
        [ 0.33039216,  0.36078432,  0.38112746],
        [ 0.30588236,  0.32794118,  0.34877452],
        [ 0.36274511,  0.3764706 ,  0.38480393]],

       [[ 0.34828432,  0.35686275,  0.36568628],
        [ 0.35318628,  0.38186275,  0.39607844],
        [ 0.3379902 ,  0.36078432,  0.3754902 ],
        [ 0.37769609,  0.39558825,  0.40000001]]])
>>> CFA = np.array(
...     [
...         [0.3764706, 0.360784320, 0.40784314, 0.3764706],
...         [0.35686275, 0.30980393, 0.36078432, 0.29803923],
...     ]
... )
>>> demosaicing_CFA_Bayer_Malvar2004(CFA, "BGGR")
array([[[ 0.35539217,  0.37058825,  0.3764706 ],
```

(continues on next page)

(continued from previous page)

```
        [ 0.34264707,  0.36078432,  0.37450981],
        [ 0.36568628,  0.39607844,  0.40784314],
        [ 0.36568629,  0.3764706 ,  0.3882353 ]],

       [[ 0.34411765,  0.35686275,  0.36200981],
        [ 0.30980393,  0.32990197,  0.34975491],
        [ 0.33039216,  0.36078432,  0.38063726],
        [ 0.29803923,  0.30441178,  0.31740197]]])
```

colour_demosaicing.demosaicing_CFA_Bayer_Menon2007

colour_demosaicing.**demosaicing_CFA_Bayer_Menon2007**(*CFA: ArrayLike, pattern: Literal['RGGB', 'BGGR', 'GRBG', 'GBRG'] | str = 'RGGB', refining_step: bool = True*)

Return the demosaiced *RGB* colourspace array from given *Bayer* CFA using DDFAPD - *Menon (2007)* demosaicing algorithm.

Parameters

- **CFA** (ArrayLike) – *Bayer* CFA.
- **pattern** (Literal['RGGB', 'BGGR', 'GRBG', 'GBRG'] | str) – Arrangement of the colour filters on the pixel array.
- **refining_step** (bool) – Perform refining step.

Returns

RGB colourspace array.

Return type

numpy.ndarray

Notes

- The definition output is not clipped in range [0, 1] : this allows for direct HDRI image generation on *Bayer* CFA data and post demosaicing of the high dynamic range data as showcased in this Jupyter Notebook.

References

[MAC07]

Examples

```
>>> CFA = np.array(
...     [
...         [0.30980393, 0.36078432, 0.30588236, 0.3764706],
...         [0.35686275, 0.39607844, 0.36078432, 0.40000001],
...     ]
... )
>>> demosaicing_CFA_Bayer_Menon2007(CFA)
array([[[ 0.30980393,  0.35686275,  0.39215687],
        [ 0.30980393,  0.36078432,  0.39607844],
        [ 0.30588236,  0.36078432,  0.39019608],
        [ 0.32156864,  0.3764706 ,  0.40000001]],
```

(continues on next page)

(continued from previous page)

```
       [[ 0.30980393,  0.35686275,  0.39215687],
        [ 0.30980393,  0.36078432,  0.39607844],
        [ 0.30588236,  0.36078432,  0.39019609],
        [ 0.32156864,  0.3764706 ,  0.40000001]]])
>>> CFA = np.array(
...     [
...         [0.3764706, 0.36078432, 0.40784314, 0.3764706],
...         [0.35686275, 0.30980393, 0.36078432, 0.29803923],
...     ]
... )
>>> demosaicing_CFA_Bayer_Menon2007(CFA, "BGGR")
array([[[ 0.30588236,  0.35686275,  0.3764706 ],
        [ 0.30980393,  0.36078432,  0.39411766],
        [ 0.29607844,  0.36078432,  0.40784314],
        [ 0.29803923,  0.3764706 ,  0.42352942]],

       [[ 0.30588236,  0.35686275,  0.3764706 ],
        [ 0.30980393,  0.36078432,  0.39411766],
        [ 0.29607844,  0.36078432,  0.40784314],
        [ 0.29803923,  0.3764706 ,  0.42352942]]])
```

Ancillary Objects

`colour_demosaicing`

`demosaicing_CFA_Bayer_DDFAPD`(CFA[, pattern, ...])	Return the demosaiced *RGB* colourspace array from given *Bayer* CFA using DDFAPD - *Menon (2007)* demosaicing algorithm.

colour_demosaicing.demosaicing_CFA_Bayer_DDFAPD

`colour_demosaicing.`**`demosaicing_CFA_Bayer_DDFAPD`**(*CFA: ArrayLike, pattern: Literal['RGGB', 'BGGR', 'GRBG', 'GBRG'] | str = 'RGGB', refining_step: bool = True*)

Return the demosaiced *RGB* colourspace array from given *Bayer* CFA using DDFAPD - *Menon (2007)* demosaicing algorithm.

Parameters

- **CFA** (`ArrayLike`) – *Bayer* CFA.
- **pattern** (`Literal['RGGB', 'BGGR', 'GRBG', 'GBRG'] | str`) – Arrangement of the colour filters on the pixel array.
- **refining_step** (`bool`) – Perform refining step.

Returns

RGB colourspace array.

Return type

`numpy.ndarray`

Notes

- The definition output is not clipped in range [0, 1] : this allows for direct HDRI image generation on *Bayer* CFA data and post demosaicing of the high dynamic range data as showcased in this Jupyter Notebook.

References

[MAC07]

Examples

```
>>> CFA = np.array(
...     [
...         [0.30980393, 0.36078432, 0.30588236, 0.3764706],
...         [0.35686275, 0.39607844, 0.36078432, 0.40000001],
...     ]
... )
>>> demosaicing_CFA_Bayer_Menon2007(CFA)
array([[[ 0.30980393,  0.35686275,  0.39215687],
        [ 0.30980393,  0.36078432,  0.39607844],
        [ 0.30588236,  0.36078432,  0.39019608],
        [ 0.32156864,  0.3764706 ,  0.40000001]],

       [[ 0.30980393,  0.35686275,  0.39215687],
        [ 0.30980393,  0.36078432,  0.39607844],
        [ 0.30588236,  0.36078432,  0.39019609],
        [ 0.32156864,  0.3764706 ,  0.40000001]]])
>>> CFA = np.array(
...     [
...         [0.3764706, 0.36078432, 0.40784314, 0.3764706],
...         [0.35686275, 0.30980393, 0.36078432, 0.29803923],
...     ]
... )
>>> demosaicing_CFA_Bayer_Menon2007(CFA, "BGGR")
array([[[ 0.30588236,  0.35686275,  0.3764706 ],
        [ 0.30980393,  0.36078432,  0.39411766],
        [ 0.29607844,  0.36078432,  0.40784314],
        [ 0.29803923,  0.3764706 ,  0.42352942]],

       [[ 0.30588236,  0.35686275,  0.3764706 ],
        [ 0.30980393,  0.36078432,  0.39411766],
        [ 0.29607844,  0.36078432,  0.40784314],
        [ 0.29803923,  0.3764706 ,  0.42352942]]])
```

Mosaicing

colour_demosaicing

mosaicing_CFA_Bayer(RGB[, pattern])	Return the *Bayer* CFA mosaic for a given *RGB* colourspace array.

colour_demosaicing.mosaicing_CFA_Bayer

colour_demosaicing.**mosaicing_CFA_Bayer**(*RGB: ArrayLike*, *pattern: Literal['RGGB', 'BGGR', 'GRBG', 'GBRG'] | str = 'RGGB'*) → ndarray[Any, dtype[_ScalarType_co]]

Return the *Bayer* CFA mosaic for a given *RGB* colourspace array.

Parameters

- **RGB** (ArrayLike) – *RGB* colourspace array.
- **pattern** (Literal['RGGB', 'BGGR', 'GRBG', 'GBRG'] | str) – Arrangement of the colour filters on the pixel array.

Returns

Bayer CFA mosaic.

Return type

numpy.ndarray

Examples

```
>>> import numpy as np
>>> RGB = np.array([[[0, 1, 2], [0, 1, 2]], [[0, 1, 2], [0, 1, 2]]])
>>> mosaicing_CFA_Bayer(RGB)
array([[ 0.,  1.],
       [ 1.,  2.]])
>>> mosaicing_CFA_Bayer(RGB, pattern="BGGR")
array([[ 2.,  1.],
       [ 1.,  0.]])
```

Masks

colour_demosaicing

masks_CFA_Bayer(shape[, pattern])	Return the *Bayer* CFA red, green and blue masks for given pattern.

colour_demosaicing.masks_CFA_Bayer

colour_demosaicing.**masks_CFA_Bayer**(*shape: int | Tuple[int, ...], pattern: Literal['RGGB', 'BGGR', 'GRBG', 'GBRG'] | str = 'RGGB'*) → Tuple[ndarray[Any, dtype[_ScalarType_co]], ...]

Return the *Bayer* CFA red, green and blue masks for given pattern.

Parameters

- **shape** (`int | Tuple[int, ...]`) – Dimensions of the *Bayer* CFA.
- **pattern** (`Literal['RGGB', 'BGGR', 'GRBG', 'GBRG'] | str`) – Arrangement of the colour filters on the pixel array.

Returns

Bayer CFA red, green and blue masks.

Return type

`tuple`

Examples

```
>>> from pprint import pprint
>>> shape = (3, 3)
>>> pprint(masks_CFA_Bayer(shape))
(array([[ True, False,  True],
       [False, False, False],
       [ True, False,  True]], dtype=bool),
 array([[False,  True, False],
       [ True, False,  True],
       [False,  True, False]], dtype=bool),
 array([[False, False, False],
       [False,  True, False],
       [False, False, False]], dtype=bool))
>>> pprint(masks_CFA_Bayer(shape, "BGGR"))
(array([[False, False, False],
       [False,  True, False],
       [False, False, False]], dtype=bool),
 array([[False,  True, False],
       [ True, False,  True],
       [False,  True, False]], dtype=bool),
 array([[ True, False,  True],
       [False, False, False],
       [ True, False,  True]], dtype=bool))
```

3.1.2 Indices and tables

- genindex
- search

CHAPTER

FOUR

1.4 CODE OF CONDUCT

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

CHAPTER

FIVE

1.5 CONTACT & SOCIAL

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

- Email
- Facebook
- Github Discussions
- Gitter
- Twitter

CHAPTER

SIX

1.6 ABOUT

Colour - Demosaicing by Colour Developers
Copyright 2015 Colour Developers – colour-developers@colour-science.org
This software is released under terms of BSD-3-Clause: https://opensource.org/licenses/BSD-3-Clause
https://github.com/colour-science/colour-demosaicing

BIBLIOGRAPHY

[LMY10] O. Losson, L. Macaire, and Y. Yang. Comparison of color demosaicing methods. In *Advances in Imaging and Electron Physics*, volume 162, pages 173–265. 2010. doi:10.1016/S1076-5670(10)62005-8.

[MHCW04] Henrique S Malvar, Li-Wei He, Ross Cutler, and One Microsoft Way. High-quality linear interpolation for demosaicing of bayer-patterned color images. In *International Conference of Acoustic, Speech and Signal Processing*, 5–8. Institute of Electrical and Electronics Engineers, Inc., May 2004. URL: http://research.microsoft.com/apps/pubs/default.aspx?id=102068.

[MAC07] Daniele Menon, Stefano Andriani, and Giancarlo Calvagno. Demosaicing with directional filtering and a posteriori decision. *IEEE Transactions on Image Processing*, 16(1):132–141, January 2007. doi:10.1109/TIP.2006.884928.

INDEX

D

`demosaicing_CFA_Bayer_bilinear()` (*in module colour_demosaicing*), 7
`demosaicing_CFA_Bayer_DDFAPD()` (*in module colour_demosaicing*), 11
`demosaicing_CFA_Bayer_Malvar2004()` (*in module colour_demosaicing*), 9
`demosaicing_CFA_Bayer_Menon2007()` (*in module colour_demosaicing*), 10

M

`masks_CFA_Bayer()` (*in module colour_demosaicing*), 14
`mosaicing_CFA_Bayer()` (*in module colour_demosaicing*), 13