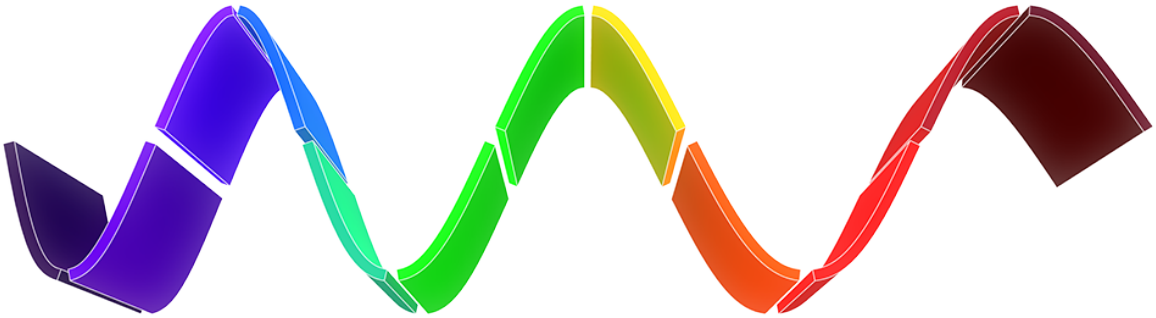

Colour - Demosaicing Documentation

Release 0.1.4

Colour Developers

Mar 24, 2019

Contents

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

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

CHAPTER 1

Features

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

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

CHAPTER 2

Installation

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.

2.1 Primary Dependencies

Colour - Demosaicing requires various dependencies in order to run:

- Python 2.7 or Python 3.5
- Colour Science
- NumPy
- OpenImageIO

2.2 Pypi

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

```
pip install colour-demosaicing
```

The tests suite dependencies are installed as follows:

```
pip install 'colour-demosaicing[tests]'
```

The documentation building dependencies are installed as follows:

```
pip install 'colour-demosaicing[docs]'
```

The overall development dependencies are installed as follows:

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

CHAPTER 3

Usage

3.1 API

The main reference for Colour - Demosaicing is the manual:

3.1.1 Colour - Demoisaicing Manual

3.1.1.1 Reference

Colour - Demosaicing

Bayer CFA Demosaicing and Mosaicing

- *Demosaicing*
- *Mosaicing*
- *Masks*

Demosaicing

`colour_demosaicing`

`demosaicing_CFA_Bayer_bilinear`(CFA[, pattern])	Returns the demosaiced *RGB* colourspace array from given *Bayer* CFA using bilinear interpolation.

Continued on next page

Table 1 – continued from previous page

demosaicing_CFA_Bayer_Malvar2004(CFA[, pattern])	Returns the demosaiced *RGB* colourspace array from given *Bayer* CFA using *Malvar (2004)* demosaicing algorithm.
demosaicing_CFA_Bayer_Menon2007(CFA[, ...])	Returns 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*, *pattern='RGGB'*)

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

Parameters

- **CFA** (array_like) – *Bayer* CFA.
- **pattern** (unicode, optional) – **{'RGGB', 'BGGR', 'GRBG', 'GBRG'}**, Arrangement of the colour filters on the pixel array.

Returns *RGB* colourspace array.

Return type ndarray

Notes

- The definition output is not clipped in range [0, 1] : this allows for direct HDRI / radiance 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]],
<BLANKLINE>
       [[ 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]])
```

(continues on next page)

(continued from previous page)

```
>>> 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]],
<BLANKLINE>
       [[ 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*, *pattern='RGGB'*)

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

Parameters

- **CFA** (array_like) – *Bayer* CFA.
- **pattern** (unicode, optional) – **{'RGGB', 'BGGR', 'GRBG', 'GBRG'}**, Arrangement of the colour filters on the pixel array.

Returns *RGB* colourspace array.

Return type ndarray

Notes

- The definition output is not clipped in range [0, 1] : this allows for direct HDRI / radiance 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]],
<BLANKLINE>
       [[ 0.34828432,  0.35686275,  0.36568628],
        [ 0.35318628,  0.38186275,  0.39607844],
        [ 0.3379902 ,  0.36078432,  0.3754902 ],
```

(continues on next page)

(continued from previous page)

```
       [ 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 ],
        [ 0.34264707,  0.36078432,  0.37450981],
        [ 0.36568628,  0.39607844,  0.40784314],
        [ 0.36568629,  0.3764706 ,  0.3882353 ]],
<BLANKLINE>
       [[ 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*, *pattern='RGGB'*, *refining_step=True*)

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

Parameters

- **CFA** (array_like) – *Bayer* CFA.
- **pattern** (unicode, optional) – **{'RGGB', 'BGGR', 'GRBG', 'GBRG'}**, Arrangement of the colour filters on the pixel array.
- **refining_step** (bool) – Perform refining step.

Returns *RGB* colourspace array.

Return type ndarray

Notes

- The definition output is not clipped in range [0, 1] : this allows for direct HDRI / radiance 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],
```

(continues on next page)

(continued from previous page)

```
        [ 0.32156864,  0.3764706 ,  0.40000001]],
<BLANKLINE>
       [[ 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]],
<BLANKLINE>
       [[ 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, ...])	Returns 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*, *pattern=u'RGGB'*, *refining_step=True*)

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

Parameters

- **`CFA`** (`array_like`) – *Bayer* CFA.
- **`pattern`** (`unicode, optional`) – **{'RGGB', 'BGGR', 'GRBG', 'GBRG'}**, Arrangement of the colour filters on the pixel array.
- **`refining_step`** (`bool`) – Perform refining step.

Returns *RGB* colourspace array.

Return type ndarray

Notes

- The definition output is not clipped in range [0, 1] : this allows for direct HDRI / radiance 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]],
<BLANKLINE>
       [[ 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]],
<BLANKLINE>
       [[ 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])	Returns the *Bayer* CFA mosaic for a given *RGB* colourspace array.

colour_demosaicing.mosaicing_CFA_Bayer

colour_demosaicing.**mosaicing_CFA_Bayer**(*RGB*, *pattern='RGGB'*)

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

Parameters

- **RGB** (array_like) – *RGB* colourspace array.
- **pattern** (unicode, optional) – **{'RGGB', 'BGGR', 'GRBG', 'GBRG'}**, Arrangement of the colour filters on the pixel array.

Returns *Bayer* CFA mosaic.

Return type 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])	Returns the *Bayer* CFA red, green and blue masks for given pattern.

colour_demosaicing.masks_CFA_Bayer

colour_demosaicing.**masks_CFA_Bayer**(*shape*, *pattern='RGGB'*)

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

Parameters

- **shape** (array_like) – Dimensions of the *Bayer* CFA.
- **pattern** (unicode, optional) – **{'RGGB', 'BGGR', 'GRBG', 'GBRG'}**, 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],
```

(continues on next page)

(continued from previous page)

```
       [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))
```

Indices and tables

- genindex
- search

3.1.1.2 Bibliography

3.2 Examples

Various usage examples are available from the examples directory.

CHAPTER 4

Contributing

If you would like to contribute to Colour - Demosaicing, please refer to the following Contributing guide for Colour.

CHAPTER 5

Bibliography

The bibliography is available in the repository in BibTeX format.

CHAPTER 6

About

Colour - Demosaicing by Colour Developers

http://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. URL: http://linkinghub.elsevier.com/retrieve/pii/S1076567010620058, 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. URL: http://ieeexplore.ieee.org/document/4032820/, doi:10.1109/TIP.2006.884928.

Index

D

M