



COLOUR - DEMOSAICING

# Colour - Demosaicing Documentation

*Release 0.1.6*

Colour Developers

Nov 09, 2021



# CONTENTS

<b>1</b>	<b>1.1</b>	<b>Features</b>	<b>3</b>
<b>2</b>	<b>1.2</b>	<b>Installation</b>	<b>5</b>
	2.1	1.2.1 Primary Dependencies . . . . .	5
	2.2	1.2.2 Pypi . . . . .	5
<b>3</b>	<b>1.3</b>	<b>Usage</b>	<b>7</b>
	3.1	1.3.1 API . . . . .	7
		3.1.1 Colour - Demosaicing Manual . . . . .	7
		3.1.1.1 Reference . . . . .	7
		3.1.1.2 Bibliography . . . . .	14
	3.2	1.3.2 Examples . . . . .	14
<b>4</b>	<b>1.4</b>	<b>Contributing</b>	<b>15</b>
<b>5</b>	<b>1.5</b>	<b>Bibliography</b>	<b>17</b>
<b>6</b>	<b>1.6</b>	<b>Code of Conduct</b>	<b>19</b>
<b>7</b>	<b>1.7</b>	<b>Contact &amp; Social</b>	<b>21</b>
<b>8</b>	<b>1.8</b>	<b>About</b>	<b>23</b>
		<b>Bibliography</b>	<b>25</b>
		<b>Index</b>	<b>27</b>



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.



## Table of Contents

- 1 *Colour - Demosaicing*

- 1.1 *Features*
- 1.2 *Installation*
  - \* 1.2.1 *Primary Dependencies*
  - \* 1.2.2 *Pypi*
- 1.3 *Usage*
  - \* 1.3.1 *API*
  - \* 1.3.2 *Examples*
- 1.4 *Contributing*
- 1.5 *Bibliography*
- 1.6 *Code of Conduct*
- 1.7 *Contact & Social*
- 1.8 *About*

## **1.1 FEATURES**

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

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





## 1.2 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 1.2.1 Primary Dependencies

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

- `python>=3.5`
- `colour-science`

### 2.2 1.2.2 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 tests suite dependencies are installed as follows:

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

The documentation building dependencies are installed as follows:

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

The overall development dependencies are installed as follows:

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



## 1.3 USAGE

### 3.1 1.3.1 API

The main reference for *Colour - Demosaicing* is the manual:

#### 3.1.1 Colour - Demosaicing Manual

##### 3.1.1.1 Reference

*Colour - Demosaicing*

*Bayer CFA Demosaicing and Mosaicing*

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

#### Demosaicing

colour\_demosaicing

<code>demosaicing_CFA_Bayer_bilinear(CFA[, pattern])</code>	Returns the demosaiced <i>RGB</i> colourspace array from given <i>Bayer</i> CFA using bilinear interpolation.
<code>demosaicing_CFA_Bayer_Malvar2004(CFA[, pattern])</code>	Returns the demosaiced <i>RGB</i> colourspace array from given <i>Bayer</i> CFA using <i>Malvar (2004)</i> demosaicing algorithm.
<code>demosaicing_CFA_Bayer_Menon2007(CFA[, ...])</code>	Returns the demosaiced <i>RGB</i> colourspace array from given <i>Bayer</i> CFA using <i>DDFAPD - Menon (2007)</i> 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** (*str*, 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]],

        [[ 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, pattern='RGB')`

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

**Parameters**

- **CFA** (*array\_like*) – *Bayer* CFA.
- **pattern** (*str*, 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]],

       [[ 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, 0.36078432, 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 ]],

       [[ 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** (*str*, 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]],

       [[ 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

---

<code>demosaicing_CFA_Bayer_DDFAPD(CFA[, pattern, ...])</code>	Returns the demosaiced <i>RGB</i> colourspace array from given <i>Bayer</i> CFA using DDFAPD - <i>Menon (2007)</i> demosaicing algorithm.
--	---

---

### colour\_demosaicing.demosaicing\_CFA\_Bayer\_DDFAPD

`colour_demosaicing.demosaicing_CFA_Bayer_DDFAPD(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** (`str`, 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]],

       [[ 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 ],
```

(continues on next page)

(continued from previous page)

```
[ 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

---

<code>mosaicing_CFA_Bayer(</code> <i>RGB</i> <code>[, </code> <i>pattern</i> <code>])</code>	Returns the <i>Bayer</i> CFA mosaic for a given <i>RGB</i> 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** (*str*, 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

---

<code>masks_CFA_Bayer(shape[, pattern])</code>	Returns the <i>Bayer</i> 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** (str, 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],
       [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 1.3.2 Examples

Various usage examples are available from the [examples directory](#).

## 1.4 CONTRIBUTING

If you would like to contribute to [Colour - Demosaicing](#), please refer to the following [Contributing](#) guide for [Colour](#).



## 1.5 BIBLIOGRAPHY

The bibliography is available in the repository in [BibTeX](#) format.



## 1.6 CODE OF CONDUCT

The *Code of Conduct*, adapted from the [Contributor Covenant 1.4](#), is available on the [Code of Conduct](#) page.





## 1.7 CONTACT & SOCIAL

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

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



## 1.8 ABOUT

**Colour - Demosaicing** by Colour Developers

Copyright © 2015-2021 – Colour Developers – [colour-developers@colour-science.org](mailto:colour-developers@colour-science.org)

This software is released under terms of New BSD License:

<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

demosaijing\_CFA\_Bayer\_bilinear() (*in module colour\_demosaijing*), 8  
demosaijing\_CFA\_Bayer\_DDFAPD() (*in module colour\_demosaijing*), 11  
demosaijing\_CFA\_Bayer\_Malvar2004() (*in module colour\_demosaijing*), 9  
demosaijing\_CFA\_Bayer\_Menon2007() (*in module colour\_demosaijing*), 10

### M

masks\_CFA\_Bayer() (*in module colour\_demosaijing*), 13  
mosaicing\_CFA\_Bayer() (*in module colour\_demosaijing*), 12