



# Mamba Image Library Python Reference

*Author:*  
Nicolas BEUCHER

February 16, 2010

## Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>mamba</b>	<b>3</b>
2.1	Classes	3
2.1.1	imageMb	3
2.2	Functions	4
2.2.1	ImageHitOrMiss(imOut, imIn, es0, es1)	4
2.2.2	absImage(imOut, imIn)	4
2.2.3	addImages(imOut, imIn1, imIn2)	4
2.2.4	addconstImage(imOut, imIn, v)	4
2.2.5	basinSegmentImage(imMarker, imIn, max_level=256)	5
2.2.6	buildNeighborImage(imInout, imMask, d)	5
2.2.7	checkEmptinessImage(imIn)	5
2.2.8	compareImages(imOut, imIn1, imIn2)	5
2.2.9	computeMaxRangeImage(imIn)	5
2.2.10	computeRangeImage(imIn)	5
2.2.11	computeVolumeImage(imIn)	5
2.2.12	convertByMaskImage(imOut, imIn, mFalse, mTrue)	5
2.2.13	convertImage(imOut, imIn)	5
2.2.14	copyBitPlaneImage(imOut, imIn, plane)	5
2.2.15	copyBytePlaneImage(imOut, imIn, plane)	5
2.2.16	copyImage(imOut, imIn)	6
2.2.17	copyLineImage(imOut, imIn, nOut, nIn)	6
2.2.18	diffImages(imOut, imIn1, imIn2)	6
2.2.19	diffNeighborImage(imInout, imIn, d, count)	6
2.2.20	divconstImage(imOut, imIn, v)	6
2.2.21	dualbuildNeighborImage(imInout, imMask, d)	6
2.2.22	extractWindowImage(imIn, threshold)	6
2.2.23	generateSupMaskImage(imOut, imIn1, imIn2, strict)	6
2.2.24	getBorder()	6
2.2.25	getDirections()	6
2.2.26	getGrid()	6
2.2.27	getImageCounter()	6
2.2.28	getImageHistogram(imIn)	6
2.2.29	getImageSize()	7
2.2.30	getOriginalImageSize()	7
2.2.31	getWindowPosition()	7
2.2.32	getWindowSize()	7
2.2.33	infNeighborImage(imInout, imIn, d, count)	7
2.2.34	labelImage(imOut, imIn, lblow=1, lbhigh=256)	7
2.2.35	logicImages(imOut, imIn1, imIn2, log)	7
2.2.36	lookupImage(imOut, imIn, lutable)	7
2.2.37	mulImages(imOut, imIn1, imIn2)	7
2.2.38	mulconstImage(imOut, imIn, v)	7
2.2.39	negateImage(imOut, imIn)	7
2.2.40	printBorder()	8
2.2.41	printGrid()	8
2.2.42	resetWindow()	8
2.2.43	setBorder(new_border)	8
2.2.44	setBorderDistanceImage(imOut, imIn)	8
2.2.45	setGrid(new_grid)	8
2.2.46	setImageIndex(index)	8
2.2.47	setImageSize(size)	8
2.2.48	setMaxImageDisplay(size)	8
2.2.49	setMinImageDisplay(size)	8
2.2.50	setShowImages(showThem)	8
2.2.51	setWindow(window)	8

2.2.52	shiftImage(imOut, imIn, d, amp, fill)	8
2.2.53	subImages(imOut, imIn1, imIn2)	9
2.2.54	subconstImage(imOut, imIn, v)	9
2.2.55	supNeighborImage(imInout, imIn, d, count)	9
2.2.56	thresholdImage(imOut, imIn, low, high)	9
2.2.57	watershedSegmentImage(imMarker, imIn, max_level=256)	9
<b>3</b>	<b>mambaDraw</b>	<b>9</b>
3.1	Functions	9
3.1.1	drawBox(imOut, square, value)	9
3.1.2	drawLine(imOut, line, value)	9
3.1.3	drawSquare(imOut, square, value)	9
3.1.4	getIntensityAlongLine(imOut, line)	10
<b>4</b>	<b>mambaExtra</b>	<b>10</b>
4.1	Functions	10
4.1.1	dynamicThresholdImage(imIn)	10
4.1.2	hitormissPatternSelector(grid=None)	10
4.1.3	mixImage(imInR, imInG, imInB)	10
4.1.4	overimposeImage(imIn1, imIn2)	10
<b>5</b>	<b>mambaComposed.sequence</b>	<b>10</b>
5.1	Classes	10
5.1.1	sequenceMb	10
5.2	Functions	11
5.2.1	closeByCylinderSequence(sequence, height, section)	11
5.2.2	copySequence(sequenceOut, sequenceIn)	11
5.2.3	dilateByCylinderSequence(sequence, height, section)	11
5.2.4	erodeByCylinderSequence(sequence, height, section)	12
5.2.5	openByCylinderSequence(sequence, height, section)	12
<b>6</b>	<b>mambaComposed.statistic</b>	<b>12</b>
6.1	Functions	12
6.1.1	getImageMean(imIn)	12
6.1.2	getImageMedian(imIn)	12
6.1.3	getImageVariance(imIn)	12
<b>7</b>	<b>mambaComposed.dilation</b>	<b>12</b>
7.1	Functions	12
7.1.1	dilateByDoublePointImage(imOut, imIn, d, n, border=0)	12
7.1.2	dilateBySmallSquareImage(imOut, imIn, border=0)	12
7.1.3	dilateByTriangleImage(imOut, imIn, border=0)	12
7.1.4	dilateInAllDirImage(imOut, imIn, n=1, border=0)	12
7.1.5	dilateInOneDirImage(imOut, imIn, d, n=1, border=0)	13
7.1.6	geodesicDilateImage(imOut, imIn, imMask, n=1)	13
<b>8</b>	<b>mambaComposed.gradient</b>	<b>13</b>
8.1	Functions	13
8.1.1	gradientImage(imOut, imIn, n=1)	13
8.1.2	halfgradientImage(imOut, imIn, type='intern', n=1)	13
8.1.3	regularGradientImage(imOut, imIn, n)	13
8.1.4	setContourImage(imOut, imIn)	13
<b>9</b>	<b>mambaComposed.erosion</b>	<b>13</b>
9.1	Functions	13
9.1.1	erodeByDoublePointImage(imOut, imIn, d, n, border=1)	13
9.1.2	erodeBySmallSquareImage(imOut, imIn, border=1)	13
9.1.3	erodeByTriangleImage(imOut, imIn, border=1)	13
9.1.4	erodeInAllDirImage(imOut, imIn, n=1, border=1)	14

9.1.5	<code>erodeInOneDirImage(imOut, imIn, d, n=1, border=1)</code>	14
9.1.6	<code>geodesicErodeImage(imOut, imIn, imMask, n=1)</code>	14
<b>10</b>	<b><code>mambaComposed.build</code></b>	<b>14</b>
10.1	Functions	14
10.1.1	<code>buildImage(imInout, imIn, border=0)</code>	14
10.1.2	<code>dualbuildImage(imInout, imIn, border=1)</code>	14
<b>11</b>	<b><code>mambaComposed.openclose</code></b>	<b>14</b>
11.1	Functions	14
11.1.1	<code>closeImage(imOut, imIn, n=1)</code>	14
11.1.2	<code>closebyBuildImage(imOut, imIn, n=1)</code>	14
11.1.3	<code>openImage(imOut, imIn, n=1)</code>	14
11.1.4	<code>openbyBuildImage(imOut, imIn, n=1)</code>	15
<b>12</b>	<b><code>mambaComposed.filter</code></b>	<b>15</b>
12.1	Functions	15
12.1.1	<code>alternateFilterImage(imOut, imIn, n, openFirst)</code>	15
12.1.2	<code>blackTopHatFilterImage(imOut, imIn, n)</code>	15
12.1.3	<code>whiteTopHatFilterImage(imOut, imIn, n)</code>	15

## List of Figures

## 1 Introduction

This document is the mamba library python reference.

It gives information regarding all the classes, functions and exceptions defined in the python part of the mamba library. This extends to all the basic functions found in mamba but also to all the modules found in the mambaComposed package. The document also gives informations regarding peripheral modules such as mambaDraw.

This document is intended for reference only. To learn more about mamba, start with the tutorial.

## 2 mamba

This module provides base functions and classes to manipulate images and do mathematical morphology. The library provides a display and user friendly features.

### 2.1 Classes

#### 2.1.1 imageMb

All mamba images are represented by this class.

**`__del__`**(self)

**`__init__`**(self, path=None, depth=8, rgbfilter=None) Constructor for a mamba image object. Without arguments will create an empty greyscale image you can specify the source image and the desired depth for the image. The depth is an integer whose values can be 1, 8 or 32. Mamba accepts all kind of image (actually all the PIL supported formats), the rgb filter will be used to convert color image into greyscale image.

**`__str__`**(self)

**`convert`**(self, depth) Converts the image depth to the given 'depth'.

**`disableTrack`**(self) Disables the tracking of mouse events inside the window displaying the image.

**`enableTrack`**(self) Enables the tracking of mouse events inside the window displaying the image. This allows to manually select pixel or areas of computations.

**`fastSetPixel`**(self, value, position) Sets the pixel at 'position' with 'value'. 'position' is a tuple holding (x,y). This function will not update the display to enable faster drawing so make sure to call the `updateDisplay()` method once you are finished with your drawing.

**`fill`**(self, v) Completely fills the image with a given value 'v'. A zero value makes the image completely dark.

**`getDepth`**(self) Returns the depth of the image.

**`getName`**(self) Returns the name of the image.

**`getPixel`**(self, position) Gets the pixel value at 'position'. 'position' is a tuple holding (x,y). Returns the value of the pixel.

**`getTrack`**(self) Returns the stored list of mouse events occurred while tracking was enabled. The list contains mouse events (list) where the first element indicates the type, either "PIXEL" or "SELECTION", and the following represent the position of the clicking ["PIXEL", x, y] or the window selected ["SELECTION", x1, y1, x2, y2] (there is no guarantee that x2 or y2 is greater than x1 or y1 as the selection can be made backward). The recorded event list is erased when this function is called.

**`hideDisplay`**(self) Call to hide the display associated to the image. If the display is hidden, the computations goes faster.

**load(self, path, rgbfilter=None)** Loads the image in 'path' into the mamba image. The optional 'rgbfilter' argument can be used to specify how to convert color image into greyscale image. It is a sequence of 3 float values indicating the amount of red, green and blue to take from the image to obtain the grey value.

**reset(self)** Reset the image (all the pixel are put to 0). This method is equivalent to `im.fill(0)`.

**resetPalette(self)** Undefines the palette to use to convert the image in color for display and save. The image will be grey scale.

**save(self, path)** Saves the image at the corresponding 'path' using PIL library. The format is automatically deduced from the extension by PIL.

**setName(self, name)** Use this function to set the image 'name'.

**setPalette(self, pal)** Defines the palette to use to convert the image in color for display and save.

**setPixel(self, value, position)** Sets the pixel at 'position' with 'value'. 'position' is a tuple holding (x,y).

**showDisplay(self)** Call to show the display associated to the image. Showing the display may slow significantly your algorithm.

**updateDisplay(self)** Call when the display associated to the image must be updated (the image changed).

## 2.2 Functions

### 2.2.1 ImageHitOrMiss(imOut, imIn, es0, es1)

Performs a binary Hit or miss operation on image 'imIn' using the structuring element 'es0' and 'es1'. Result is put in 'imOut'.

Structuring elements are integers value coding which direction must be taken into account. 'es0' indicating which neighbor of the current pixel will be checked for 0 value and 'es1' those which will be evaluated for 1 value. for neighbor in direction d the correct code is  $\text{pow}(2,d)$ . If you want to add multiple direction, just add their code.

You can also found a helper function in the `mambaExtra` module.

### 2.2.2 absImage(imOut, imIn)

returns in 'imOut' the absolute value of image 'imIn'.

### 2.2.3 addImages(imOut, imIn1, imIn2)

Adds 'imIn2' pixel values to 'imIn1' pixel values and put the result in 'imOut'. The operation can be sum up in the following formula :

$$\text{imOut} = \text{imIn1} + \text{imIn2}.$$

You can mix format in the addition operation (A binary image can be added to a grey scale image, etc...). However you must ensure that the output image is as deep as the deepest of the two added images.

The operation is also saturated for grey-scale images (e.g. on a grey scale image  $255+1=255$ ).

### 2.2.4 addconstImage(imOut, imIn, v)

Adds 'imIn' pixel values to value 'v' and put the result in 'imOut'. The operation can be sum up in the following formula :

$$\text{imOut} = \text{imIn} + v$$

The operation is saturated for grey-scale images.

### 2.2.5 `basinSegmentImage(imMarker, imIn, max_level=256)`

Segments greyscale image 'imIn' using the watershed algorithm. 'imMarker' is used both as the marker image (the wells from which the flooding proceed) and as the output image. It is a 32-bit image. 'max\_level' can be used to limit the flooding process to a specific level (useful if you want to survey the flooding level by level).

The result is put inside 'imMarker'. The three first byte planes contains the actual segmentation (each segment has a specific label according to the original marker). This function only return catchment basins (no watershed line) and is fastest than watershedSegmentImage if you are not interested in the watershed line.

### 2.2.6 `buildNeighborImage(imInout, imMask, d)`

Builds image 'imInout' in direction 'd' using 'imMask' as a mask. The function also returns the volume of the image 'imInout' after the build operation.

### 2.2.7 `checkEmptinessImage(imIn)`

Checks if image 'imIn' is empty (i.e. completely black). Returns True if so, False otherwise.

### 2.2.8 `compareImages(imOut, imIn1, imIn2)`

Compare the two images 'imIn1' and 'imIn2'. The comparison stops as soon as a pixel is different in the two images. The corresponding pixel in 'imOut' is set to the value of the pixel of 'imIn1'.

The function returns a tuple holding the position of the first mismatching pixel. The tuple value is (-1,-1) if the two images are identical.

### 2.2.9 `computeMaxRangeImage(imIn)`

Returns a tuple with the minimum and maximum pixel values possible given the depth of image 'imIn'. The values are returned in a tuple holding the minimum and the maximum.

### 2.2.10 `computeRangeImage(imIn)`

Compute the range, i.e. the minimum and maximum values, of image 'imIn'. The values are returned in a tuple holding the minimum and the maximum.

### 2.2.11 `computeVolumeImage(imIn)`

Compute the volume of the image 'imIn', i.e. the sum of its pixel values. The computed integer value is returned by the function.

### 2.2.12 `convertByMaskImage(imOut, imIn, mFalse, mTrue)`

Converts a binary image 'imIn' into a greyscale image or a 32-bit image and put the result in 'imOut'.

The pixel set to true are set to value 'mTrue' in the output image and the pixel set to false to value 'mFalse'.

### 2.2.13 `convertImage(imOut, imIn)`

Convert the content 'imIn' to the the depth of 'imOut' and puts the result in 'imOut'.

Only grey-scale to binary and binary to grey-scale conversion are supported.

### 2.2.14 `copyBitPlaneImage(imOut, imIn, plane)`

Inserts or extracts a bit plane. If 'imIn' is is a binary image it is inserted at 'plane' position in greyscale 'imOut'. If 'imIn' is is a greyscale image its bit plane at 'plane' position is extracted and put in 'imOut'.

Plane values are 0 (LSB) to 7 (MSB).

### 2.2.15 `copyBytePlaneImage(imOut, imIn, plane)`

Inserts or extracts a byte plane. If 'imIn' is is a greyscale image it is inserted at 'plane' position in 32-bit 'imOut'. If 'imIn' is is a 32-bit image its byte plane at 'plane' position is extracted and put in 'imOut'.

Plane values are 0 (LSByte) to 3 (MSByte).

### 2.2.16 copyImage(imOut, imIn)

Copy the pixels of 'imIn' in 'imOut'.

The images must be in the same depth.

### 2.2.17 copyLineImage(imOut, imIn, nOut, nIn)

Copies the line number 'nIn' of image 'imIn' into 'imOut' at line index 'nOut'.

The images must be in the same depth.

### 2.2.18 diffImages(imOut, imIn1, imIn2)

Performs a set difference between 'imIn1' and 'imIn2' and puts the result in 'imOut'. The set difference will copy 'imIn1' pixels in 'imOut' if the corresponding pixel in 'imIn2' is lower and will put 0 otherwise.

### 2.2.19 diffNeighborImage(imInout, imIn, d, count)

Performs a set difference operation between the 'imInout' image and the 'imIn' image shifted in direction 'd'.

The operation is repeated 'count' times if the two images are referring to the same data (only for 8 bit images).

### 2.2.20 divconstImage(imOut, imIn, v)

Divides 'imIn' pixel values by value 'v' and put the result in 'imOut'. The operation can be sum up in the following formula :

$$\text{imOut} = \text{imIn} / v \text{ (or more accurately : } \text{imIn} = \text{imOut} * v + r, r \text{ being the ignored remainder)}$$

A zero value in 'v' will return an error. For 8-bits image, v will be restricted between 0 and 255. You cannot use it with binary images.

### 2.2.21 dualbuildNeighborImage(imInout, imMask, d)

Dual Builds image 'imInout' in direction 'd' using 'imMask' as a mask. The function also returns the volume of the image 'imInout' after the build operation.

### 2.2.22 extractWindowImage(imIn, threshold)

Extracts the smallest window inside the image 'imIn' that includes all the pixels whose value is greater or equals to 'threshold'.

### 2.2.23 generateSupMaskImage(imOut, imIn1, imIn2, strict)

Generate a binary mask image in 'imOut' where pixels are set to 1 when they are greater (strictly if 'strict' is set to True, greater or equal otherwise) in image 'imIn1' than in image 'imIn2'.

### 2.2.24 getBorder()

Returns the current setting for border

### 2.2.25 getDirections()

Returns a list containing all the possible directions available in the currently used grid.

### 2.2.26 getGrid()

Returns the current setting for grid

### 2.2.27 getImageCounter()

Returns the number of image actually defined in the mamba library.

### 2.2.28 getImageHistogram(imIn)

Returns a list holding the histogram of the greyscale image 'imIn' (0 to 255).

### 2.2.29 `getImageSize()`

Returns a tuple holding the image size.

### 2.2.30 `getOriginalImageSize()`

Returns a tuple holding the original image size.

### 2.2.31 `getWindowPosition()`

Returns a tuple holding the current computation window position in the image.

### 2.2.32 `getWindowSize()`

Returns a tuple holding the current computation window size.

### 2.2.33 `infNeighborImage(imInout, imIn, d, count)`

Performs a minimum operation between the 'imInout' image and the 'imIn' image shifted in direction 'd'. The operation is repeated 'count' times if the two images are referring to the same data.

### 2.2.34 `labelImage(imOut, imIn, lblow=1, lbhigh=256)`

Labels the binary image 'imIn' and put the result in 32-bit image 'imOut'. Returns the number of connected components found by the labeling algorithm.

'lblow' and 'lbhigh' are used to restrain the possible values in the lower byte of 'imOut' pixel values.

### 2.2.35 `logicImages(imOut, imIn1, imIn2, log)`

Performs a logic operation between the pixel of images 'imIn1' and 'imIn2' and put the result in 'imOut'. The logic operation to perform is indicated through argument 'log'. The allowed logical operations in 'log' are :

"and", "or", "xor", "'inf' or 'sup'".

The images must be in the same depth.

### 2.2.36 `lookupImage(imOut, imIn, lutable)`

Converts the greyscale image 'imIn' using the look-up table 'lutable' and puts the result in greyscale image 'imOut'.

'lutable' is a list containing 256 values with the first one corresponding to the 0 and the last to 255.

### 2.2.37 `mullImages(imOut, imIn1, imIn2)`

Multiplies 'imIn2' pixel values with 'imIn1' pixel values and put the result in 'imOut'. The operation can be sum up in the following formula :

$$\text{imOut} = \text{imIn1} * \text{imIn2}$$

You can mix format in the substraction operation (A binary image can be multiplied with a grey scale image, etc...). However you must ensure that the output image is as deep as the deepest of the two input images.

The operation is also saturated for grey-scale images (e.g. on a grey scale image  $255*255=255$ ).

### 2.2.38 `mulconstImage(imOut, imIn, v)`

Multiplies 'imIn' pixel values with value 'v' and put the result in 'imOut'. The operation can be sum up in the following formula :

$$\text{imOut} = \text{imIn} * v$$

The operation is saturated for grey-scale images. You cannot use it with binary images.

### 2.2.39 `negateImage(imOut, imIn)`

Negates the image 'imIn' and puts the result in 'imOut'.

The operation is a binary complement for binary and grey-scale images and a negation for signed 32-bit images.

**2.2.40 printBorder()**

Prints the border status.

**2.2.41 printGrid()**

Prints the grid status.

**2.2.42 resetWindow()**

Resets the computation window inside the image so that all the image is processed.

**2.2.43 setBorder(new\_border)**

Sets the border used for computation to 'new\_border'. Allowed values are FILLED and EMPTY.

**2.2.44 setBorderDistanceImage(imOut, imIn)**

Computes for each pixel of binary 'imIn' set to True the minimum distance to reach a border while constantly staying in the set. The result is put in 32-bit 'imOut'.

**2.2.45 setGrid(new\_grid)**

Sets the grid used for computation to 'new\_grid'. Change global variables along to Allowed values are HEXAGONAL and SQUARE.

**2.2.46 setImageIndex(index)**

Set the image index used for naming to a given value 'index'

**2.2.47 setImageSize(size)**

Allows the definition of image 'size' that will be used afterward. This function will have no effects once an image has been created.

**2.2.48 setMaxImageDisplay(size)**

Set the maximum 'size' (tuple with w and h) above which the image is automatically downsized upon displaying.

**2.2.49 setMinImageDisplay(size)**

Set the minimum 'size' (tuple with w and h) below which the image is automatically upsized upon displaying.

**2.2.50 setShowImages(showThem)**

Activate automatically the display for new images when 'showThem' is set to True.

**2.2.51 setWindow(window)**

Sets the computation window inside the image so that only a specific part of the image is processed. 'window' is a tuple with 2 or 4 values. With two values, the window is set to the minimum size possible that includes the given pixel. With 4 values, the windows is set to the minimum size that include the region defined, the first pixel is the upper left corner and the second the lower right. If window does not contain 2 or 4 value the function does nothing.

**2.2.52 shiftImage(imOut, imIn, d, amp, fill)**

Shifts image 'imIn' in direction 'd' over an amplitude of 'amp'. The created space is filled with 'fill' value. The result is put in 'imOut'.

### 2.2.53 `subImages(imOut, imIn1, imIn2)`

Subtracts 'imIn2' pixel values to 'imIn1' pixel values and put the result in 'imOut'. The operation can be sum up in the following formula :

$$\text{imOut} = \text{imIn1} - \text{imIn2}$$

You can mix format in the subtraction operation (A binary image can be subtracted to a grey scale image, etc...). However you must ensure that the output image is as deep as the deepest of the two subtracted images.

The operation is also saturated for grey-scale images (e.g. on a grey scale image 0-1=0).

### 2.2.54 `subconstImage(imOut, imIn, v)`

Subtracts 'v' value to 'imIn' pixel values and put the result in 'imOut'. The operation can be sum up in the following formula :

$$\text{imOut} = \text{imIn} - v$$

The operation is saturated for grey-scale images.

### 2.2.55 `supNeighborImage(imInout, imIn, d, count)`

Performs a maximum operation between the 'imInout' image and the 'imIn' image shifted in direction 'd'.

The operation is repeated 'count' times if the two images are referring to the same data.

### 2.2.56 `thresholdImage(imOut, imIn, low, high)`

Performs a threshold operation over image 'imIn'. The result is put in binary image 'imOut'.

All the pixels that have a strictly lower value than 'low' or strictly higher than 'high' are set to false. Otherwise they are set to true.

### 2.2.57 `watershedSegmentImage(imMarker, imIn, max_level=256)`

Segments greyscale image 'imIn' using the watershed algorithm. 'imMarker' is used both as the marker image (the wells from which the flooding proceed) and as the output image. It is a 32-bit image. 'max\_level' can be used to limit the flooding process to a specific level (useful if you want to survey the flooding level by level).

The result is put inside 'imMarker'. The three first byte planes contains the actual segmentation (each segment has a specific label according to the original marker). The last plane represent the actual watershed line (pixel set to 255).

## 3 mambaDraw

This module defines functions to draw inside mamba image. Drawing functions includes lines, square, ... The module also provides function to extract pixel information

### 3.1 Functions

#### 3.1.1 `drawBox(imOut, square, value)`

Draws a box (empty square) in 'imOut' using the tuple 'square' containing 4 values (upper left and down right corners (x1,y1,x2,y2)) using 'value' to set the pixels.

#### 3.1.2 `drawLine(imOut, line, value)`

Draws a line in 'imOut' using the tuple 'line' containing 4 values (starting and ending points (x1,y1,x2,y2)) using 'value' to set the pixels.

This function use the Bresenham algorithm.

#### 3.1.3 `drawSquare(imOut, square, value)`

Draws a square in 'imOut' using the tuple 'square' containing 4 values (upper left and down right corners (x1,y1,x2,y2)) using 'value' to set the pixels.

### 3.1.4 `getIntensityAlongLine(imOut, line)`

Returns in a list the intensity profile along a line in 'imOut' using the tuple 'line' containing 4 values (starting and ending points (x1,y1,x2,y2)).

This function use the Bresenham algorithm.

## 4 `mambaExtra`

This module defines specific functions, class and so on that can be considered as extra for mamba. They usually concern optional display method.

### 4.1 Functions

#### 4.1.1 `dynamicThresholdImage(imIn)`

Opens a separate display in which you can dynamically perform a threshold operation over image 'imIn'.

Once the close button is pressed, the result of the dynamic threshold is returned. This result is the a tuple (low, high) used to obtain the image displayed using the threshold operation `thresholdImage` from mamba.

While the window is opened, you can increase or decrease the low level using keys Q and W respectively. The high level is modified by the key S (increasing) and X (decreasing).

#### 4.1.2 `hitormissPatternSelector(grid=None)`

Helps the user to create pattern for the Hit or Miss operator defined in the Mamba module.

The function returns inside a tuple the structuring elements 'es0' and 'es1' (in that order) used as entry in the `ImageHitOrMiss` function.

You can select the desired grid for the pattern selector. If not specified the function will use the grid currently in use.

example with the `ImageHitOrMiss` function : `ImageHitOrMiss(imOut, imIn, *hitormissPatternSelector())`

#### 4.1.3 `mixImage(imInR, imInG, imInB)`

Mixes images 'imInR' (red channel), 'imInG' (green channel) and 'imInB' (blue channel) into a color image.

#### 4.1.4 `overimposeImage(imIn1, imIn2)`

Draw images 'imIn1' and 'imIn2' in a common display.

If both images are binary, the display is a combination of their pixel values, i.e. black where the pixel is black in both images, blue if the pixel is set in both images, green if the pixel is set only in 'imIn1' and red if it is only set in 'imIn2'.

If one of the image is greyscale and the other is binary, the binary image is redrawn over the greyscale image in purple.

If both images are greyscale, TO DO ...

## 5 `mambaComposed.sequence`

This module provides class, method and functions to perform morphological computations over sequence of image using mamba. This module can be seen as 3-D extension of mamba.

### 5.1 Classes

#### 5.1.1 `sequenceMb`

A sequence of images is represented by an instance of this class.

`__getitem__`(self, key) Handles direct acces to the image inside the sequence

**\_\_init\_\_(self, length, path=None, depth=8)** Constructor for a mamba image sequence. 'length' controls the length of the sequence. 'path' can be used to indicate a directory in which a sequence of image can be found (see method load for explanation regarding the loading of a sequence). 'depth' indicates the image depth.

**\_\_iter\_\_(self)** Makes a mamba image sequence iterable.

**fill(self, v)** Fills all the images in the sequence with value 'v' A zero value makes the image completely dark.

**getDepth(self)** Returns the depth of the sequence.

**getLength(self)** Returns the length of the sequence.

**getName(self)** Returns the name of the sequence.

**hideAllImages(self)** Deactivates the image display for all the image in the sequence.

**hideImage(self, index)** Deactivates the image display for image at 'index' in the sequence.

**load(self, path)** Loads a sequence of image as found in directory 'path'. To be valid a sequence of image must be composed of at least 'length' image to be able to fill the sequence. Their file names must be of the form XXX.ext where XXX is a number on three digits and ext is an image file extension (like jpg or png). The sequence will be read in increasing order (001 then 002 and so on) however it is not mandatory that the number follow each other (001 then 005 is legal). You can also mix image formats (001.bmp then 002.jpg is legal) but you should make sure that files that are not images (txt, pdf, ...) are not named following that pattern.

**next(self)** The next method for the iteration.

**reset(self)** Reset the sequence (all the pixel are put to 0).

**resetPalette(self)** Undefined the palette to use to convert the images in color for display and save. The images will be grey scale.

**setPalette(self, pal)** Defines the palette to use to convert the images in color for display and save. Apply to all the images in the sequence.

**showAllImages(self)** Activates the image display for all the image in the sequence.

**showImage(self, index)** Activates the image display for image at 'index' in the sequence.

## 5.2 Functions

### 5.2.1 closeByCylinderSequence(sequence, height, section)

Closing using the dilation and erosion by cylinder.

### 5.2.2 copySequence(sequenceOut, sequenceIn)

Copies the content of 'sequenceIn' into 'sequenceOut'. The copy is stopped by the smallest sequence.

### 5.2.3 dilateByCylinderSequence(sequence, height, section)

Dilates the 'sequence' using a cylinder with an hexagonal section of size 2x'section' and a height of 2x'height'. The sequence is modified by this function.

#### 5.2.4 `erodeByCylinderSequence(sequence, height, section)`

Erodes the 'sequence' using a cylinder with an hexagonal section of size  $2 \times \text{section}$  and a height of  $2 \times \text{height}$ . The sequence is modified by this function.

#### 5.2.5 `openByCylinderSequence(sequence, height, section)`

Opening using the dilation and erosion by cylinder.

## 6 `mambaComposed.statistic`

This module provides a set of functions to compute statistical value inside mamba image.

### 6.1 Functions

#### 6.1.1 `getImageMean(imIn)`

Returns the average value of the pixels of 'imIn'.

#### 6.1.2 `getImageMedian(imIn)`

Returns the median value of the pixels of 'imIn'.

The median value is defined as the first pixel value for which at least half of the are below it.

#### 6.1.3 `getImageVariance(imIn)`

Returns the pixels variance of image 'imIn'.

## 7 `mambaComposed.dilation`

This module provides a set of functions to perform dilation using Mamba. it works with `imageMb` instance as defined in mamba.

### 7.1 Functions

#### 7.1.1 `dilateByDoublePointImage(imOut, imIn, d, n, border=0)`

Performs a dilation of 'imIn' using a double point as a structuring element. To build the double point the first point is considered in position (0,0) and the second is built using a shift 'n' times in the direction 'd'. The result is put in imOut.

This function will assume an EMPTY border unless specified otherwise using 'border'.

#### 7.1.2 `dilateBySmallSquareImage(imOut, imIn, border=0)`

Performs a dilation of 'imIn' using a small square (2x2) as a structuring element. The result is put in imOut.

This function will assume an EMPTY border unless specified otherwise using 'border'.

#### 7.1.3 `dilateByTriangleImage(imOut, imIn, border=0)`

Performs a dilation of 'imIn' using a triangle as a structuring element. The result is put in imOut. The triangle points upward.

This function will assume an EMPTY border unless specified otherwise using 'border'.

#### 7.1.4 `dilateInAllDirImage(imOut, imIn, n=1, border=0)`

Performs a dilation in all the directions of image 'imIn' and puts the result in 'imOut'. The operations is repeated 'n' times (default is 1).

This function will assume an EMPTY border unless specified otherwise using 'border'.

### 7.1.5 dilateInOneDirImage(imOut, imIn, d, n=1, border=0)

Performs a dilatation in direction 'd' of image 'imIn' and puts the result in 'imOut'. The operation is repeated 'n' times (default is 1).

This function will assume an EMPTY border unless specified otherwise using 'border'.

### 7.1.6 geodesicDilateImage(imOut, imIn, imMask, n=1)

Performs a geodesic dilation of image 'imIn' inside 'imMask'. The result is put inside 'imOut', 'n' controls the size of the dilation.

## 8 mambaComposed.gradient

This module provides a set of functions to perform morphological gradient using mamba. it works with imageMb instance as defined in mamba.

### 8.1 Functions

#### 8.1.1 gradientImage(imOut, imIn, n=1)

Computes the morphological gradient of image 'imIn' and puts the result in 'imOut'. The thickness can be controlled using parameter 'n'.

#### 8.1.2 halfgradientImage(imOut, imIn, type='intern', n=1)

Computes the half morphological gradient of image 'imIn' and puts the result in 'imOut'. 'type' indicates if the half gradient should be internal or external. The thickness can be controlled using parameter 'n'.

#### 8.1.3 regularGradientImage(imOut, imIn, n)

Computes the regularized gradient of image 'imIn' of size 'n'. The result is put inside 'imOut'.

#### 8.1.4 setContourImage(imOut, imIn)

Computes the set contour of a binary image.

The function will work with a greyscale image but will not be useful.

## 9 mambaComposed.erosion

This module provides a set of functions to perform erosion using Mamba. it works with imageMb instance as defined in mamba.

### 9.1 Functions

#### 9.1.1 erodeByDoublePointImage(imOut, imIn, d, n, border=1)

Performs an erosion of 'imIn' using a double point as a structuring element. To build the double point the first point is considered in position (0,0) and the second is built using a shift 'n' times in the direction 'd'. The result is put in imOut.

This function will assume a FILLED border unless specified otherwise using 'border'.

#### 9.1.2 erodeBySmallSquareImage(imOut, imIn, border=1)

Performs an erosion of 'imIn' using a small square (2x2) as a structuring element. The result is put in imOut.

This function will assume a FILLED border unless specified otherwise using 'border'.

#### 9.1.3 erodeByTriangleImage(imOut, imIn, border=1)

Performs an erosion of 'imIn' using a triangle as a structuring element. The result is put in imOut. The triangle points upward.

This function will assume a FILLED border unless specified otherwise using 'border'.

#### 9.1.4 `erodeInAllDirImage(imOut, imIn, n=1, border=1)`

Performs an erosion in all the directions of image 'imIn' and puts the result in 'imOut'. The operations is repeated 'n' times (default is 1).

This function will assume a FILLED border unless specified otherwise using 'border'.

#### 9.1.5 `erodeInOneDirImage(imOut, imIn, d, n=1, border=1)`

Performs an erosion in direction 'd' of image 'imIn' and puts the result in 'imOut'. The operation is repeated 'n' times (default is 1).

This function will assume a FILLED border unless specified otherwise using 'border'.

#### 9.1.6 `geodesicErodeImage(imOut, imIn, imMask, n=1)`

Performs a geodesic erosion of image 'imIn' inside 'imMask'. The result is put inside 'imOut', 'n' controls the size of the erosion.

The geodesic erosion is realised using the fact that the dilation is the dual operation of the erosion.

## 10 `mambaComposed.build`

This module provides a set of functions to perform build and dualbuild operation it works with imageMb instance as defined in mamba.

### 10.1 Functions

#### 10.1.1 `buildImage(imInout, imIn, border=0)`

Builds image 'imInout' using 'imIn' as a mask.

This function will assume an EMPTY border unless specified otherwise using 'border'.

#### 10.1.2 `dualbuildImage(imInout, imIn, border=1)`

Builds (dual build) image 'imInout' using 'imIn' as a mask.

This function will assume a FILLED border unless specified otherwise using 'border'.

## 11 `mambaComposed.openclose`

This module provides a set of functions to perform opening and closing operation using mamba. it works with imageMb instance as defined in mamba. All the closing and opening operation defined in this module use erosion, dilation and build functions without consideration for the current border setting. The functions define a default border (see the modules erosion, dilation and build).

### 11.1 Functions

#### 11.1.1 `closeImage(imOut, imIn, n=1)`

Performs a closing operation on image 'imIn' and puts the result in 'imOut'. 'n' controls the size of the closing.

#### 11.1.2 `closebyBuildImage(imOut, imIn, n=1)`

Performs a closing by dual reconstruction operation on image 'imIn' and puts the result in 'imOut'. 'n' controls the size of the closing.

#### 11.1.3 `openImage(imOut, imIn, n=1)`

Performs an opening operation on image 'imIn' and puts the result in 'imOut'. 'n' controls the size of the opening.

#### 11.1.4 `openbyBuildImage(imOut, imIn, n=1)`

Performs an opening by reconstruction operation on image 'imIn' and puts the result in 'imOut'. 'n' controls the size of the opening.

## 12 `mambaComposed.filter`

This module provides a set of functions to perform filtering operation using mamba. it works with imageMb instance as defined in mamba.

### 12.1 Functions

#### 12.1.1 `alternateFilterImage(imOut, imIn, n, openFirst)`

Performs an alternate filter operation on image 'imIn' and puts the result in 'imOut'. 'n' controls the filter size. If 'openFirst' is True, the filter begins with an opening, a closing otherwise.

#### 12.1.2 `blackTopHatFilterImage(imOut, imIn, n)`

Performs a black Top Hat filter operation on 'imIn' and puts the result in 'imOut'. This filter extracts from 'imIn' the dark objects smaller than 'n'.

#### 12.1.3 `whiteTopHatFilterImage(imOut, imIn, n)`

Performs a white Top Hat filter operation on 'imIn' and puts the result in 'imOut'. This filter extracts from 'imIn' the bright objects smaller than 'n'.