Mamba Image Library Python Reference

Automatically generated using pydoc

www.mamba-image.org

October 30, 2015

Contents

1 Introduction

This document is the Mamba library Python reference.

It applies to version 2.0.

It gives information regarding all the classes, functions and exceptions defined in the Python part of the Mamba library. This extends to all the functions defined in the mamba and mamba3D packages.

The document also gives information regarding the mambaDisplay peripheral module.

This document is intended for reference only. To learn more about Mamba, start with the Mamba User Manual.

2 mamba.arithmetic

Arithmetic and logical operators. This module provides arithmetic operators such as addition, subtraction, multiplication and division between images together with logical operators (and, or, not, xor ...) between these images.

2.1 Functions

2.1.1 add(imIn1, imIn2, imOut)

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

imOut = imIn1 + imIn2.

You can mix formats in the addition operation (a binary image can be added to a greyscale 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 greyscale images (e.g. on a 8-bit greyscale image, 255+1=255). With 32-bit images, the addition is not saturated.

2.1.2 addConst(imIn, v, imOut)

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

imOut = imIn + v

'imIn' and imOut' can be 8-bit or 32-bit images of same size and depth.

The operation is saturated (limited to 255) for greyscale images.

2.1.3 ceilingAdd(imIn1, imIn2, imOut)

Adds image 'imIn2' to image 'imIn1' and puts the result in 'imOut'. If imIn1 + imIn2 is larger than the maximal possible value in imOut, the result is truncated and limited to this maximal value.

Although it is possible to use a 8-bit image for imIn2, it is recommended to use the same depth for all the images.

Note that this operator is mainly useful for 32-bit images, as the result of the addition is always truncated for 8-bit images.

2.1.4 ceilingAddConst(imIn, v, imOut)

Adds a constant value 'v' to image 'imIn' and puts the result in 'imOut'. If imIn + v is larger than the maximal possible value in imOut, the result is truncated and limited to this maximal value.

Note that this operator is mainly useful for 32-bit images, as the result of the addition is always truncated for 8-bit images.

2.1.5 diff(imIn1, imIn2, imOut)

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 write 0 otherwise:

imOut = imIn1 if imIn1 > imin2 imOut = 0 otherwise.

'imIn1', imIn2' and 'imOut' can be 1-bit, 8-bit or 32-bit images of same size and depth.

2.1.6 div(imIn1, imIn2, imOut)

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

imOut = imIn1 / imIn2

Greyscale or 32-bit images can be used. You can mix formats in the divide operation. However you must ensure that the output image is as deep as 'imIn1'.

In order to avoid errors due to divisions by zero, each time a pixel in 'imIn2' is equal to zero, the result is set to the maximum value corresponding to the depth of the image.

2.1.7 divConst(imIn, v, imOut)

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

imOut = imIn / v (or more acurately : imIn = imOut * v + r, r being the ignored reminder)

A zero value in 'v' will return an error. For a 8-bit image, v will be restricted between 1 and 255. You cannot use it with binary images.

2.1.8 floorSub(imIn1, imIn2, imOut)

Subtracts image 'imIn2' from image 'imIn1' and puts the result in 'imOut'. If imIn1 - imIn2 is negative, the result is truncated and limited to 0.

Although it is possible to use a 8-bit image for imIn2, it is recommended to use the same depth for all the images.

Note that this operator is mainly useful for 32-bit images, as the result of the subtraction is always truncated for 8-bit images.

2.1.9 floorSubConst(imIn, v, imOut)

Subtracts a constant value 'v' to image 'imIn' and puts the result in 'imOut'. If imIn - v is negative, the result is truncated and limited to 0.

Note that this operator is mainly useful for 32-bit images, as the result of the subtraction is always truncated for 8-bit images.

2.1.10 logic(imIn1, imIn2, imOut, log)

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

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

"and" performs a bitwise AND operation, "or" a bitwise OR and "xor" a bitwise XOR. "inf" calculates the minimum and "sup" the maximum between corresponding pixel values.

'imIn1', imIn2' and 'imOut' can be 1-bit, 8-bit or 32-bit images of same size and depth.

2.1.11 mul(imIn1, imIn2, imOut)

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

imOut = imIn1 * imIn2

You can mix formats in the multiply operation (a binary image can be multiplied with a greyscale 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 greyscale images (e.g. on a greyscale image 255*255=255).

2.1.12 mulConst(imIn, v, imOut)

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

imOut = imIn * v

The operation is saturated for greyscale images. You cannot use it with binary images.

2.1.13 mulRealConst(imIn, v, imOut, nearest=False, precision=2)

Multiplies image 'imIn' by a real positive constant value 'v' and puts the result in image 'imOut'. 'imIn' and 'imOut' can be 8-bit or 32-bit images. If 'imOut' is greyscale (8-bit), the result is saturated (results of the multiplication greater than 255 are limited to this value). 'precision' indicates the number of decimal digits taken into account for the constant 'v' (default is 2). If 'nearest' is true, the result is rounded to the nearest integer value. If not (default), the result is simply truncated.

2.1.14 negate(imIn, imOut)

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

The operation is a binary complement for binary images and a negation for greyscale and 32-bit images.

2.1.15 sub(imIn1, imIn2, imOut)

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

imOut = imIn1 - imIn2

You can mix formats in the subtraction operation (a binary image can be subtracted to a greyscale 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) but not for 32-bit images.

2.1.16 subConst(imIn, v, imOut)

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

imOut = imIn - v

'imIn' and imOut' can be 8-bit or 32-bit images of same size and depth.

The operation is saturated (lower limit is 0) for greyscale images.

3 mamba.base

Image class definition. This is the base module of the Mamba Image library. It defines the imageMb class used to contain images.

3.1 Classes

3.1.1 imageMb

Defines the imageMb class and its methods. All Mamba images are represented by this class.

__del__(self)

__init__(self, *args, **kwargs) Constructor for a Mamba image object.

This constructor allows a wide range of possibilities for defining an image:

- imageMb(): without arguments will create an empty greyscale image.
- imageMb(im): will create an image using the same size and depth as 'im'.
- imageMb(depth): will create an image with the desired 'depth' (1, 8 or 32).
- imageMb(path): will load the image located in 'path'.
- imageMb(im, depth): will create an image using the same size as 'im' and the specified 'depth'.
- imageMb(path, depth): will load the image located in 'path' and convert it to the specified 'depth'.
- imageMb(width, height): will create an image with size 'width'x'height'.
- imageMb(width, height, depth): will create an image with size 'width'x'height' and the specified 'depth'.

When not specified, the width and height of the image will be set to 256x256. The default depth is 8 (greyscale).

When loading an image from a file, please note that Mamba accepts all kinds of images (actually all the PIL or PILLOW supported formats). You can specify the RGB filter that will be used to convert a color image into a greyscale image by adding the rgbfilter=<your_filter> to the argument of the constructor.

__str__(self)

convert(self, depth) Converts the image depth to the given 'depth'. The conversion algorithm is identical to the conversion used in the convert function (see this function for details).

extractRaw(self) Extracts and returns the image raw string data. This method only works on 8 and 32-bit images.

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 a faster drawing. So make sure to call the update() method once your drawing is finished, if you want to see the result.

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

freeze(self) Called to freeze the display of the image. Thus the image may change but the display will not show these modifications until the method unfreeze is called.

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.

getSize(self) Returns the size (a tuple width and height) of the image.

hide(self) Called to hide the display associated to the image. If the display is hidden, the computations go much 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 a color image into a 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. By default, the color conversion uses the ITU-R 601-2 luma transform (see PIL/PILLOW documentation for details).

loadRaw(self, dataOrPath, preprocfunc=None) Loads raw data inside the 3D image. You can give a filename or data directly through 'dataOrPath'. the data length must match the image size : width * height * (depth/8) If needed, you can preprocess the data using the optional argument 'preprocfunc' which will be called on the data before loading it. The preprocfunc must have the following prototype : outdata = preprocfunc(indata) The size verification is performed after the preprocessing (enabling you to use zip archives and such).

reset(self) Resets the image (all the pixels are put to 0). This method is equivalent to im.fill(0).

save(self, path, palette=None) Saves the image at the corresponding 'path' using PIL/PILLOW library. The format is automatically deduced by PIL/PILLOW from the image name extension. Make sure the format support the image depth (e.g 32-bit image cannot be stored in JPEG) and use uncompressed format if you wish to reload the image unaltered later. You can add a 'palette' to the saved image (note that it may alter the pixels value if you wish to reload the image later).

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

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

show(self, **options) Called to show the display associated to the image. Showing the display may significantly slow down your operations.

You can specify 'options' that will be given to the displayer.

unfreeze(self) Called to unfreeze the display of the image. Thus the image display will be updated along with the modifications inside the image.

update(self) Called when the display associated to the image must be updated (the image has changed).

3.2 Functions

3.2.1 getImageCounter()

Returns the number of images actually defined and allocated in the Mamba library. This function may be useful for debugging purposes.

3.2.2 getShowImages()

Returns the display status ('always_show' value).

3.2.3 setImageIndex(index)

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

3.2.4 setShowImages(showThem)

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

4 mamba.contrasts

Contrast operators. This module provides a set of functions to perform morphological contrast operators such as gradient, top-hat transform, ...

4.1 Functions

4.1.1 blackTopHat(imIn, imOut, n, se=structuringElement([0, 1, 2, 3, 4, 5, 6], mamba.HEXAGONAL))

Performs a black Top Hat operation on 'imIn' and puts the result in 'imOut'. This operator extracts from 'imIn' the dark objects thinner than 2*'n'+1.

The structuring element used is defined by 'se' ('DEFAULT_SE' by default).

4.1.2 gradient(imIn, imOut, n=1, se=structuringElement([0, 1, 2, 3, 4, 5, 6], mamba.HEXAGONAL))

Computes the morphological gradient of image 'imIn' and puts the result in 'imOut'. The thickness can be controlled using parameter 'n' (1 by default). The structuring element used by the erosion and dilation is defined by 'se' (DEFAULT_SE by default).

4.1.3 halfGradient(imIn, imOut, type='intern', n=1, se=structuringElement([0, 1, 2, 3, 4, 5, 6], mamba.HEXAGONAL))

Computes the half morphological gradient of image 'imIn' ond puts the result in 'imOut'.

'type' indicates if the half gradient should be internal or external. Possible values are : "extern" : dilation(imIn) - imIn "intern" : imIn - erosion(imIn)

The thickness can be controlled using parameter 'n' (1 by default). The structuring element used by the erosion or the dilation is defined by 'se'.

4.1.4 regularisedGradient(imIn, imOut, n, grid=HEXAGONAL)

Computes the regularized gradient of image 'imIn' of size 'n'. The result is put in 'imOut'. A regularized gradient of size 'n' extracts in the image contours thinner than 'n' while avoiding false detections.

This operation is only valid for omnidirectional structuring elements.

4.1.5 supBlackTopHat(imIn, imOut, n, grid=HEXAGONAL)

Performs a black Top Hat operation with the infimum of directional openings on 'imIn' and puts the result in 'imOut'. This operator partly extracts from 'imIn' the dark objects whose extension in at least one direction of 'grid' is smaller than 'n'.

4.1.6 supWhiteTopHat(imIn, imOut, n, grid=HEXAGONAL)

Performs a white Top Hat operation with the supremum of directional openings on 'imIn' and puts the result in 'imOut'. This operator partly extracts from 'imIn' the bright objects whose extension in at least one direction of 'grid' is smaller than 'n'.

4.1.7 whiteTopHat(imIn, imOut, n, se=structuringElement([0, 1, 2, 3, 4, 5, 6], mamba.HEXAGONAL))

Performs a white Top Hat operation on 'imIn' and puts the result in 'imOut'. This operator extracts from 'imIn' the bright objects thinner than 2*'n'+1.

The structuring element used is defined by 'se' ('DEFAULT_SE' by default).

5 mamba.conversion

Depth conversion operators. This module regroups various functions/operators to perform conversions based on image depth. It allows to transfer data from an image depth to another.

5.1 Functions

5.1.1 convert(imIn, imOut)

Converts the contents of 'imIn' to the depth of 'imOut' and puts the result in 'imOut'.

For greyscale/32-bit to binary conversion, value 255/0xffffffff in 'imIn' is converted to 1 in 'imOut'. All other values are transformed to 0. The reverse convention applies to binary to greyscale/32-bit conversion.

32-bit images are downscaled into greyscale images. Conversion from 8-bit to 32-bit is equivalent to copyBytePlane for plane 0.

When both images have the same depth this function is a simple copy.

5.1.2 convertByMask(imIn, imOut, mFalse, mTrue)

Converts a binary image 'imIn' into a greyscale image (8-bit) or a 32-bit image and puts the result in 'imOut'.

white pixels of 'imIn' are set to value 'mTrue' in the output image and the black pixels set to value 'mFalse'.

5.1.3 generateSupMask(imIn1, imIn2, imOut, strict)

Generates 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'.

'imIn1' and imIn2' can be 1-bit, 8-bit or 32-bit images of same size and depth.

5.1.4 lookup(imIn, imOut, 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 0 and the last one to 255.

5.1.5 threshold(imIn, imOut, 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.

'imIn' can be a 8-bit or 32-bit image.

6 mamba.copies

Copy operators. This module regroups various complete or partial copy operators.

6.1 Functions

6.1.1 copy(imIn, imOut)

Copies 'imIn' image into 'imOut' image.

'imIn' and 'imOut' can be 1-bit, 8-bit or 32-bit images. The images must have the same depth and size.

6.1.2 copyBitPlane(imIn, plane, imOut)

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

Plane values are from 0 (LSB) to 7 (MSB) for 8-bit images or from 0 (LSB) to 31 (MSB) for 32-bit images.

6.1.3 copyBytePlane(imIn, plane, imOut)

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

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

6.1.4 copyLine(imIn, nIn, imOut, nOut)

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

'imIn' and 'imOut' can be 1-bit, 8-bit or 32-bit images. The images must have the same depth and size.

6.1.5 cropCopy(imIn, posIn, imOut, posOut, size)

Copies the pixels of 'imIn' in 'imOut' starting from position 'posIn' (tuple x,y) in 'imIn' to position 'posOut' in 'imOut'. The size of the copy in controlled by 'size' (tuple w,h). The actual size will be computed so as not to exceed the images border.

The images must be of the same depth but can have different sizes. Only non binary images are accepted (8-bit or 32-bit).

7 mamba.draw

Drawing operators. This module defines functions to draw inside Mamba images. Drawing functions include lines, squares, ... The module also provides functions to extract pixel information.

7.1 Functions

7.1.1 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.

7.1.2 drawCircle(imOut, circle, value)

Draws a circle in 'imOut' using the tuple 'circle' containing 3 values (center and radius (x,y,r)) using 'value' to set the pixels.

7.1.3 drawFillCircle(imOut, circle, value)

Draws a filled circle in 'imOut' using the tuple 'circle' containing 3 values (center and radius (x,y,r)) using 'value' to set the pixels.

7.1.4 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 uses the Bresenham algorithm.

7.1.5 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.

7.1.6 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 uses the Bresenham algorithm.

8 mamba.erodil

Erosion and dilation operators. This module provides a set of functions and class to perform erosions and dilations. The module contains basic and complex operators that are based on neighbor comparisons. In particular, it defines the structuring element class which serve as the base for these operators. The module also contains distance functions based on erosion.

8.1 Classes

8.1.1 structuringElement

This class allows to define simple structuring elements with points belonging to the elementary neighborhood of the origin point. Points in use are defined by their direction, according to the grid in use (hexagonal or square one). All the used directions are put in a in a direction list. The central point (direction 0) may or may not belong to the structuring element. Example: »>HEXAGON = structuringElement([0,1,2,3,4,5,6], mamba.HEXAGONAL) defines a structuring element named HEXAGON on the hexagonal grid and containing the origin point and all the six neighboring points in directions 1 to 6 of the grid.

__eq__(self, otherSE)

__init__(self, directions, grid) Structuring element constructor. A structuring element is defined by the couple 'directions' (given in an ordered list) and 'grid'. You cannot defines a structuring element that holds a direction more than once.

You can look at the predefined structuring elements to get examples of how to make yours.

__ne__(self, otherSE)

__repr__(self)

getDirections(self, withoutZero=False) Returns a copy of the directions used by the structuring element. If 'withoutZero' is set to True the returned direction list will not include direction 0 (useful for some operators, such as erode or dilate, where direction 0 modifies the initial conditions).

Example:

```
>>>SQUARE3X3.getDirections()
[0, 1, 2, 3, 4, 5, 6, 7, 8]
```

getEncodedDirections(self, withoutZero=False) Returns the directions used by the structuring element encoded in a format directly usable with Neighbor functions. If 'withoutZero' is set to True the returned encoded direction will not include direction 0 (useful for some operators, such as erode or dilate, where direction 0 modifies the initial conditions).

getGrid(self) Returns the grid associated with the structuring element.
Example:

```
>>>HEXAGON.getGrid()
HEXAGONAL
```

hasZero(self) Returns True if the central point (0) is included in the direction list.

rotate(self, step=1) Rotates the structuring element 'step' times. When step is positive, rotation is clockwise. When step is negative, rotation is counterclockwise. When step is equal to zero, there is no rotation.
Example:

```
>>>SEGMENT.getDirections()
[0, 1]
>>>SEGMENT.rotate().getDirections()
[0, 2]
```

setAs(self, se) Copies the attributes (directions, grid) of the structuringElement 'se' into the structuring element which this method is applied to. This method is mainly used to modify the default structuring element.
Example:

```
>>>DEFAULT_SE.setAs(SQUARE3X3)
```

modifies the default structuring element to a square one. The erosions and dilations now will be performed with a square.
Warning! Although it is perfectly allowed, it is not wise to use setAs method with other pre-defined structuring elements. For instance, if you type:

```
>>>HEXAGON.setAs(SQUARE3X3)
```

the structuring element HEXAGON will be superseded by SQUARE3X3. This may have unwanted consequences.

transpose(self) Structuring element transposition (symmetry around the origin). Basically, it corresponds to a 3-steps rotation on an hexagonal grid, a 4-steps on a square one.
Example:

```
>>>TRIANGLE.getDirections()
[0, 3, 4]
>>>TRIANGLE.transpose().getDirections()
|0, 1, 6]
```

8.2 Functions

8.2.1 computeDistance(imIn, imOut, grid=HEXAGONAL, edge=EMPTY)

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

The distance computation will be performed according to the 'grid' (HEXAGONAL is 6-Neighbors and SQUARE is 8-Neighbors). 'edge' can be FILLED or EMPTY.

8.2.2 conjugateHexagonalDilate(imIn, imOut, size, edge=EMPTY)

Dilation by a conjugate hexagon (hexagon turned by 30 degrees). Be aware that the size of operation corresponds to twice the size of the regular hexagon: a conjugate hexagon of size 1 is inscribed in a regular hexagon of size 2.

8.2.3 conjugateHexagonalErode(imIn, imOut, size, edge=FILLED)

Erosion by a conjugate hexagon (hexagon turned by 30 degrees).

Be aware that the size of operation corresponds to twice the size of the regular hexagon: a conjugate hexagon of size 1 is inscribed in a regular hexagon of size 2.

8.2.4 diffNeighbor(imIn, imInout, nb, grid=HEXAGONAL, edge=EMPTY)

Performs a set difference operation between the 'imInout' image pixels and their neighbors according to 'grid' in image 'imIn'. Neighbors are encoded in 'nb'. If any considered neighbor point has a value greater than or equal to the center point, this center point point is set to 0. The result is put in 'imOut'.

'grid' value can be HEXAGONAL or SQUARE. 'edge' value can be EMPTY or FILLED.

If a neighboring point falls outside the image window, its value in the operation is defined by 'edge'. If 'edge' is EMPTY, its value is 0. If 'edge' is FILLED, its value equals the maximal allowed value according to the depth of 'imIn' image.

'imIn' and 'imOut' can be 1-bit, 8-bit and 32-bit images of same size and depth.

'nb' contains the coding of all the selected neighbor points. See the User Manual for details.

8.2.5 dilate(imIn, imOut, n=1, se=structuringElement([0, 1, 2, 3, 4, 5, 6], mamba.HEXAGONAL), edge=EMPTY)

This operator performs a dilation, using the structuring element 'se' (set by default as DEFAULT_SE), of image 'imIn' and puts the result in 'imOut'. The operation is repeated 'n' times (default is 1). This operator assumes an 'EMPTY' edge by default.

This operator always considers that the origin of the structuring element in use is at position 0 even if this point does not belong to it.

8.2.6 dodecagonalDilate(imIn, imOut, size, edge=EMPTY)

Dilation by a dodecagon (hexagonal grid). This operation is the result of an hexagonal dilation followed by a dilation by a conjugate hexagon. The respective sizes of the hexagon and of the conjugate hexagon are calculated in order that the final dodecagon be as isotropic as possible.

8.2.7 dodecagonalErode(imIn, imOut, size, edge=FILLED)

Erosion by a dodecagon (hexagonal grid). This operation is the result of an hexagonal erosion followed by an erosion by a conjugate hexagon. The respective sizes of the hexagon and of the conjugate hexagon are calculated in order that the final dodecagon be as isotropic as possible.

8.2.8 doublePointDilate(imIn, imOut, d, n, grid=HEXAGONAL, edge=EMPTY)

This operator 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' + 180 degrees (transposed direction) of 'grid'. The result is put in imOut. The direction d is selected according to the grid in use (DEFAULT_GRID).

Note that this operator is just an alias of the operator supFarNeighbor.

8.2.9 doublePointErode(imIn, imOut, d, n, grid=HEXAGONAL, edge=FILLED)

This operation 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 conjugate direction 'd' + 180 degrees of 'grid'. The result is put in imOut.

This operator is an alias of the infFarNeighbor operator.

8.2.10 erode(imIn, imOut, n=1, se=structuringElement([0, 1, 2, 3, 4, 5, 6], mamba.HEXAGONAL), edge=FILLED)

This operator corresponds, for erosion, to the dilation operator. It performs an erosion using the default structuring element of image 'imIn' and puts the result in 'imOut'. The operation is repeated 'n' times (default is 1). This function will assume a FILLED edge unless specified otherwise using 'edge'.

This operator always considers that the origin of the structuring element in use is at position 0 even if this point does not belong to it.

8.2.11 infNeighbor(imIn, imInout, nb, grid=HEXAGONAL, edge=FILLED)

Performs a minimum operation between the 'imInout' image pixels and their neighbors according to 'grid' in image 'imIn'. Neighbors are encoded in 'nb'. The result is put in 'imOut'.

'grid' value can be HEXAGONAL or SQUARE. 'edge' value can be EMPTY or FILLED.

If a neighboring point falls outside the image window, its value in the operation is defined by 'edge'. If 'edge' is EMPTY, its value is 0. If 'edge' is FILLED, its value equals the maximal allowed value according to the depth of 'imIn' image.

'imIn' and 'imOut' can be 1-bit, 8-bit or 32-bit images of same size and depth.

'nb' contains the coding of all the selected neighbor points. See the User Manual for details.

8.2.12 infVector(imIn, imInout, vector, edge=FILLED)

Performs a minimum operation between the 'imInout' image pixels and their corresponding pixels in image 'imIn' after it has been shifted by 'vector' (tuple dx,dy). The result is put in 'imOut'.

'edge' value can be EMPTY or FILLED.

If a neighboring point falls outside the image window, its value in the operation is defined by 'edge'. If 'edge' is EMPTY, its value is 0. If 'edge' is FILLED, its value equals the maximal allowed value according to the depth of 'imIn' image.

'imIn' and 'imOut' can be 1-bit, 8-bit or 32-bit images of same size and depth.

8.2.13 isotropicDistance(imIn, imOut, edge=FILLED)

Computes the distance function of a set in 'imIn'. This distance function uses dodecagonal erosions and the grid is assumed to be hexagonal. The procedure is quite slow but the result is more aesthetic. This operator also illustrates how to perform successive dodecagonal operations of increasing sizes.

8.2.14 linearDilate(imIn, imOut, d, n=1, grid=HEXAGONAL, edge=EMPTY)

Dilation by a segment in direction 'd' of image 'imIn', result in 'imOut'. The operation is repeated 'n' times (default is 1).This function will assume an EMPTY edge unless specified otherwise using 'edge'. The directions are defined according to the grid in use.

8.2.15 linearErode(imIn, imOut, d, n=1, grid=HEXAGONAL, edge=FILLED)

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 edge unless specified otherwise using 'edge'.

8.2.16 octogonalDilate(imIn, imOut, size, edge=EMPTY)

Dilation by an octogon (square grid). This operation is the result of a dilation by a square followed by a dilation by a diamond. The respective sizes of the square and of the diamond are calculated in order that the final octogon be as isotropic as possible.

8.2.17 octogonalErode(imIn, imOut, size, edge=FILLED)

Erosion by an octogon (square grid). This operation is the result of an erosion by a square followed by an erosion by a diamond (conjugate square). The respective sizes of the square and of the diamond are calculated in order that the final octogon be as isotropic as possible.

8.2.18 supNeighbor(imIn, imInout, nb, grid=HEXAGONAL, edge=EMPTY)

Performs a maximum operation between the 'imInout' image pixels and their neighbors according to 'grid' in image 'imIn'. Neighbors are encoded in 'nb'. The result is put in 'imOut'.

'grid' value can be HEXAGONAL or SQUARE. 'edge' value can be EMPTY or FILLED.

If a neighboring point falls outside the image window, its value in the operation is defined by 'edge'. If 'edge' is EMPTY, its value is 0. If 'edge' is FILLED, its value equals the maximal allowed value according to the depth of 'imIn' image.

'imIn' and 'imOut' can be 1-bit, 8-bit or 32-bit images of same size and depth.

'nb' contains the coding of all the selected neighbor points. See the User Manual for details.

8.2.19 supVector(imIn, imInout, vector, edge=EMPTY)

Performs a maximum operation between the 'imInout' image pixels and their corresponding pixels in image 'imIn' after it has been shifted by 'vector' (tuple dx,dy). The result is put in 'imOut'.

'edge' value can be EMPTY or FILLED.

If a neighboring point falls outside the image window, its value in the operation is defined by 'edge'. If 'edge' is EMPTY, its value is 0. If 'edge' is FILLED, its value equals the maximal allowed value according to the depth of 'imIn' image.

'imIn' and 'imOut' can be 1-bit, 8-bit or 32-bit images of same size and depth.

9 mamba.erodilLarge

Large erosion and dilation operators. This module provides a set of functions performing erosions and dilations with large structuring elements. These operators are to be prefered to the standard ones when working with large structuring elements.

9.1 Functions

9.1.1 infFarNeighbor(imIn, imInout, nb, amp, grid=HEXAGONAL, edge=FILLED)

Performs a minimum operation between the 'imInout' image pixels and their neighbor 'nb' at distance 'amp' according to 'grid' in image 'imIn'. The result is put in 'imInOut'.

'grid' value can be HEXAGONAL or SQUARE. 'edge' value can be EMPTY or FILLED.

If a neighboring point falls outside the image window, its value in the operation is defined by 'edge'. If 'edge' is EMPTY, its value is 0. If 'edge' is FILLED, its value equals the maximal allowed value according to the depth of 'imIn' image.

'imIn' and 'imInOut' can be 1-bit, 8-bit or 32-bit images of same size and depth.

9.1.2 largeDodecagonalDilate(imIn, imOut, size, edge=EMPTY)

Dilation by large dodecacagons (hexagonal grid). Basically, it is the same operation as the previous one where classical dilations have been replaced by dilations by large structuring elements, and where a "partial" dilation by a conjugate hexagon is used.

9.1.3 largeDodecagonalErode(imIn, imOut, size, edge=FILLED)

Erosion by large dodecacagons (hexagonal grid). Basically, it is the same operation as the previous one where classical erosions have been replaced by erosions by large structuring elements, and where a "partial" erosion by a conjugate hexagon is used.

9.1.4 largeHexagonalDilate(imIn, imOut, size, edge=EMPTY)

Dilation by large hexagons using dilations by large segments and the Steiner decomposition property of the hexagon. Edge effects are corrected by dilations with transposed decompositions combined with sup operators.

This operator is quite complex to avoid edge effects.

9.1.5 largeHexagonalErode(imIn, imOut, size, edge=FILLED)

Erosion by large hexagons using erosions by large segments and the Steiner decomposition property of the hexagon. Edge effects are corrected by erosions with transposed decompositions combined with inf operations (see documentation for further details).

This operator is quite complex to avoid edge effects.

9.1.6 largeLinearDilate(imIn, imOut, dir, size, grid=HEXAGONAL, edge=EMPTY)

Dilation by a large segment in direction 'dir' in a reduced number of iterations. Uses the dilations by doublets of points (supposed to be faster, thanks to an enhanced shift operator).

9.1.7 largeLinearErode(imIn, imOut, dir, size, grid=HEXAGONAL, edge=FILLED)

Erosion by a large segment in direction 'dir' in a reduced number of iterations. Uses the erosions by doublets of points (supposed to be faster, thanks to an enhanced shift operator).

9.1.8 largeOctogonalDilate(imIn, imOut, size, edge=EMPTY)

Dilation by a large octogon (square grid). This operation uses dilations by large squares and large diamonds previously defined.

9.1.9 largeOctogonalErode(imIn, imOut, size, edge=FILLED)

Erosion by a large octogon (square grid). This operation uses erosions by large squares and large diamonds previously defined.

9.1.10 largeSquareDilate(imIn, imOut, size, edge=EMPTY)

Dilation by a large square using dilations by large segments and the Steiner decomposition property of the square.

No edge effects are likely to happen with a square structuring element.

9.1.11 largeSquareErode(imIn, imOut, size, edge=FILLED)

Erosion by a large square using erosions by large segments and the Steiner decomposition property of the square.

No edge effects are likely to happen with a square structuring element.

9.1.12 supFarNeighbor(imIn, imInout, nb, amp, grid=HEXAGONAL, edge=EMPTY)

Performs a maximum operation between the 'imInout' image pixels and their neighbor 'nb' at distance 'amp' according to 'grid' in image 'imIn'. The result is put in 'imInOut'.

'grid' value can be HEXAGONAL or SQUARE. 'edge' value can be EMPTY or FILLED.

If a neighboring point falls outside the image window, its value in the operation is defined by 'edge'. If 'edge' is EMPTY, its value is 0. If 'edge' is FILLED, its value equals the maximal allowed value according to the depth of 'imIn' image.

'imIn' and 'imInOut' can be 1-bit, 8-bit or 32-bit images of same size and depth.

10 mamba.extrema

Extrema (min/max) operators. This module provides a set of operators dealing with maxima and minima of a function. New operators linked to the notion of dynamics are also provided.

10.1 Functions

10.1.1 deepMinima(imIn, imOut, h, grid=HEXAGONAL)

Computes the minima of the dual reconstruction of image 'imIn' by imIn + h and puts the result in 'imOut'.

Grid used by the dual build operation can be specified by 'grid'.

Only works with 8-bit or 32-bit images as input. 'imOut' must be binary.

10.1.2 highMaxima(imIn, imOut, h, grid=HEXAGONAL)

Computes the maxima of the reconstruction of image 'imIn' by imIn - h and puts the result in 'imOut'.

Grid used by the build operation can be specified by 'grid'.

Only works with 8-bit or 32-bit images as input. 'imOut' must be binary.

10.1.3 maxDynamics(imIn, imOut, h, grid=HEXAGONAL)

Extracts the maxima of 'imIn' with a dynamics higher or equal to 'h' and puts the result in 'imOut'.

Grid used by the dual build operation can be specified by 'grid'.

Only works with 8-bit or 32-bit images as input. 'imOut' must be binary.

10.1.4 maxPartialBuild(imIn, imMask, imOut, grid=HEXAGONAL)

Performs the partial reconstruction of 'imIn' with its maxima which are contained in the binary mask 'imMask'. The result is put in 'imOut'.

'imIn' and 'imOut' must be different and greyscale images.

10.1.5 maxima(imIn, imOut, h=1, grid=HEXAGONAL)

Computes the maxima of 'imIn' using a build operation and puts the result in 'imOut'. When 'h' is equal to 1 (default value), the operator provides the maxima of 'imIn'.

Grid used by the build operation can be specified by 'grid'.

Only works with with 8-bit or 32-bit images as input. 'imOut' must be binary.

10.1.6 minDynamics(imIn, imOut, h, grid=HEXAGONAL)

Extracts the minima of 'imIn' with a dynamics higher or equal to 'h' and puts the result in 'imOut'.

Grid used by the dual build operation can be specified by 'grid'.

Only works with 8-bit or 32-bit images as input. 'imOut' must be binary.

10.1.7 minPartialBuild(imIn, imMask, imOut, grid=HEXAGONAL)

Performs the partial reconstruction of 'imIn' with its minima which are contained in the binary mask 'imMask'. The result is put in 'imOut'.

'imIn' and 'imOut' must be different and greyscale images.

10.1.8 minima(imIn, imOut, h=1, grid=HEXAGONAL)

Computes the minima of 'imIn' using a dual build operation and puts the result in 'imOut'. When 'h' is equal to 1 (default value), the operator provides the minima of 'imIn'.

Grid used by the dual build operation can be specified by 'grid'.

Only works with 8-bit or 32-bit images as input. 'imOut' must be binary.

11 mamba.filter

Filtering operators. This module provides a set of functions to perform morphological filtering operations such as alternate filters.

11.1 Functions

11.1.1 alternateFilter(imIn, imOut, n, openFirst, se=structuringElement([0, 1, 2, 3, 4, 5, 6], mamba.HEXAGONAL))

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

11.1.2 autoMedian(imIn, imOut, n, se=structuringElement([0, 1, 2, 3, 4, 5, 6], mamba.HEXAGONAL))

Morphological automedian filter performed with alternate sequential filters.

11.1.3 fullAlternateFilter(imIn, imOut, n, openFirst, se=structuringElement([0, 1, 2, 3, 4, 5, 6], mamba.HEXAGONAL))

Performs a full alternate filter operation (successive alternate filters of increasing sizes, from 1 to 'n') 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.

11.1.4 largeDodecagonalAlternateFilter(imIn, imOut, start, end, step, openFirst)

Fast full alternate dodecagonal filter of image 'imIn'. The initial size is equal to 'start', the final one is bounded by 'end' (this size is not taken into account), the increment of size is 'step'. If 'openFirst' is true, the filter starts with on opening. It starts with a closing otherwise. The result is put in 'imOut'.

11.1.5 largeHexagonalAlternateFilter(imIn, imOut, start, end, step, openFirst)

Fast full alternate hexagonal filter of image 'imIn'. The initial size is equal to 'start', the final one is bounded by 'end' (this size is not taken into account), the increment of size is 'step'. If 'openFirst' is true, the filter starts with on opening. It starts with a closing otherwise. The result is put in 'imOut'. This operation is efficient if most of the sizes used in the filter are greater than 5. If it is not the case, the 'fullAlternateFilter' should be used instead.

11.1.6 largeOctogonalAlternateFilter(imIn, imOut, start, end, step, openFirst)

Fast full alternate octogonal filter of image 'imIn'. The initial size is equal to 'start', the final one is limited by 'end' (this size is not taken into account), the increment of size is 'step'. If 'openFirst' is true, the filter starts with on opening. It starts with a closing otherwise. The result is put in 'imOut'.

11.1.7 largeSquareAlternateFilter(imIn, imOut, start, end, step, openFirst)

Fast full alternate square filter of image 'imIn'. The initial size is equal to 'start', the final one is bounded by 'end' (this size is not taken into account), the increment of size is 'step'. If 'openFirst' is true, the filter starts with on opening. It starts with a closing otherwise. The result is put in 'imOut'. This operation is efficient if most of the sizes used in the filter are greater than 5. If it is not the case, the 'fullAlternateFilter' should be used instead (with a SQUARE structuring element).

11.1.8 linearAlternateFilter(imIn, imOut, n, openFirst, grid=HEXAGONAL)

Performs an alternate filter operation on image 'imIn' with openings and closings by segments of size 'n' (supremum of openings and infimum of closings) and puts the result in 'imOut'. If 'openFirst' is True, the filter begins with an opening, a closing otherwise.

11.1.9 simpleLevelling(imIn, imMask, imOut, grid=HEXAGONAL)

Performs a simple levelling of image 'imIn' controlled by image 'imMask' and puts the result in 'imOut'. This operation is composed of two geodesic reconstructions. This filter tends to level regions in the image of homogeneous grey values.

11.1.10 strongLevelling(imIn, imOut, n, eroFirst, grid=HEXAGONAL)

Strong levelling of 'imIn', result in 'imOut'. 'n' defines the size of the erosion and dilation of 'imIn' in the operation. If 'eroFirst' is true, the operation starts with an erosion, it starts with a dilation otherwise.

This filter is stronger (more efficient) that simpleLevelling. However, the order of the initial operations (erosion and dilation) matters.

12 mamba.geodesy

Geodesic operators. This module provides a set of functions to perform geodesic computations. It includes build and dualbuild operations, geodesic erosion and dilation ...

12.1 Functions

12.1.1 build(imMask, imInout, grid=HEXAGONAL)

Builds image 'imInout' using 'imMask' as a mask. This operator performs the geodesic reconstruction of 'imInout' inside the mask image and puts the result in the same image.

This operator uses a recursive implementation of the reconstruction.

This function will use the mamba default grid unless specified otherwise in 'grid'.

12.1.2 buildNeighbor(imMask, imInout, d, grid=HEXAGONAL)

Builds image 'imInout' in direction 'd' according to 'grid' using 'imMask' as a mask (the propagation is performed only in 'd' direction).

The function also returns the volume of the image 'imInout' after the build operation.

'grid' value can be HEXAGONAL or SQUARE.

12.1.3 closeHoles(imIn, imOut, grid=HEXAGONAL)

Close holes in image 'imIn' and puts the result in 'imOut'. This operator works on binary and greytone images. In this case, however, it should be used cautiously.

12.1.4 dualBuild(imMask, imInout, grid=HEXAGONAL)

Builds (dual build) image 'imInout' using 'imMask' as a mask. This operator performs the geodesic dual reconstruction (by erosions) of 'imInout' inside the mask image and puts the result in the same image.

This operator uses a recursive implementation of the reconstruction.

This function will use the mamba default grid unless specified otherwise in 'grid'.

12.1.5 dualbuildNeighbor(imMask, imInout, d, grid=HEXAGONAL)

Dual builds image 'imInout' in direction 'd' according to 'grid' using 'imMask' as a mask (the propagation is performed only in 'd' direction).

The function also returns the volume of the image 'imInout' after the build operation.

'grid' value can be HEXAGONAL or SQUARE.

12.1.6 geodesicDilate(imIn, imMask, imOut, n=1, se=structuringElement([0, 1, 2, 3, 4, 5, 6], mamba.HEXAGONAL))

This operator is simply an alias of lowerGeodesicDilate. It is kept for compatibility reasons.

12.1.7 geodesicDistance(imIn, imMask, imOut, se=structuringElement([0, 1, 2, 3, 4, 5, 6], mamba.HEXAGONAL))

Computes the geodesic distance function of a set in 'imIn'. This distance function uses successive geodesic erosions of 'imIn' performed in the geodesic space defined by 'imMask'. The result is stored in 'imOut'. Be sure to use an image of sufficient depth as output.

This geodesic distance is quite slow as it is performed by successive geodesic erosions.

12.1.8 geodesicErode(imIn, imMask, imOut, n=1, se=structuringElement([0, 1, 2, 3, 4, 5, 6], mamba.HEXAGONAL))

This transformation is identical to the previous version and it has been kept for compatibilty purposes.

Note that the binary and the greytone operators are different.

12.1.9 hierarBuild(imMask, imInout, grid=HEXAGONAL)

Builds image 'imInout' using 'imMask' as a mask. This function only works with greyscale and 32-bit images and uses a hierarchical queue algorithm to compute the result.

'grid' will set the number of neighbors considered by the algorithm (HEXAGONAL is 6-Neighbors and SQUARE is 8-Neighbors).

This function is identical to build but it is faster.

12.1.10 hierarDualBuild(imMask, imInout, grid=HEXAGONAL)

Builds (dual build) image 'imInout' using 'imMask' as a mask. This function works with greyscale and 32-bit images and uses a hierarchical queue algorithm to compute the result.

'grid' will set the number of neighbors considered by the algorithm (HEXAGONAL is 6-Neighbors and SQUARE is 8-Neighbors).

This function is identical to dualBuild but it is faster.

12.1.11 lowerGeodesicDilate(imIn, imMask, imOut, n=1, se=structuringElement([0, 1, 2, 3, 4, 5, 6], mamba.HEXAGONAL))

Performs a lower geodesic dilation of image 'imIn' below 'imMask'. The result is put inside 'imOut', 'n' controls the size of the dilation. 'se' specifies the type of structuring element used to perform the computation (DEFAULT_SE by default).

Warning! 'imMask' and 'imOut' must be different.

12.1.12 lowerGeodesicErode(imIn, imMask, imOut, n=1, se=structuringElement([0, 1, 2, 3, 4, 5, 6], mamba.HEXAGONAL))

Performs a lower geodesic erosion of image 'imIn' under 'imMask'. The result is put inside 'imOut', 'n' controls the size of the erosion. 'se' specifies the type of structuring element used to perform the computation (DEFAULT_SE by default).

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

Warning! 'imMask' and 'imOut' must be different.

12.1.13 removeEdgeParticles(imIn, imOut, grid=HEXAGONAL)

Removes particles (connected components) touching the edge in image 'imIn'. The resulting image is put in image 'imOut'. Although this operator may be used with greytone images, it should be considered with caution.

12.1.14 upperGeodesicDilate(imIn, imMask, imOut, n=1, se=structuringElement([0, 1, 2, 3, 4, 5, 6], mamba.HEXAGONAL))

Performs an upper geodesic dilation of image 'imIn' above 'imMask'. The result is put inside 'imOut', 'n' controls the size of the dilation. 'se' specifies the type of structuring element used to perform the computation (DEFAULT_SE by default).

Warning! 'imMask' and 'imOut' must be different.

12.1.15 upperGeodesicErode(imIn, imMask, imOut, n=1, se=structuringElement([0, 1, 2, 3, 4, 5, 6], mamba.HEXAGONAL))

Performs a upper geodesic erosion of image 'imIn' above 'imMask'. The result is put inside 'imOut', 'n' controls the size of the erosion. 'se' specifies the type of structuring element used to perform the computation (DEFAULT_SE by default).

Warning! 'imMask' and 'imOut' must be different.

13 mamba.grids

Grids handling and setting functions. This module defines various functions related to grid configurations and computations.

13.1 Functions

13.1.1 getDirections(grid=HEXAGONAL, withoutZero=False)

Returns a range of all the possible directions available in 'grid' (set to DEFAULT_GRID by default). If 'withoutZero' is set to True, the direction 0 is omitted.

If the 'grid' value is incorrect, the function returns an empty list.

13.1.2 gridNeighbors(grid=HEXAGONAL)

Returns the number of neighbors of a point in 'grid' (6 or 8).

If the 'grid' value is incorrect, the function returns 0.

13.1.3 rotateDirection(d, step=1, grid=HEXAGONAL)

Calculates the value of the new direction starting from direction 'd' after 'step' rotations (default value 1). If 'step' is positive, rotations are performed clockwise. They are counterclockwise if 'step' is negative. Calculation is made according to the grid. Direction 0 is taken into account (and always unchanged).

13.1.4 setDefaultGrid(grid)

This function will change the value of the default grid used in each operator that needs to specify one.

'grid' must be either HEXAGONAL or SQUARE.

You can of course manually change the variable DEFAULT_GRID by yourself. Using this function is however recommended if you are not sure of what you are doing.

13.1.5 transposeDirection(d, grid=HEXAGONAL)

Calculates the transposed (opposite) direction value of direction 'd' (corresponds to a rotation of gridNeighbors/2 steps).

14 mamba.hierarchies

Hierarchical segmentation operators. This module provides a set of functions to perform hierarchical segmentation operations. This module contains the waterfalls algorithm and various hierarchical operators (enhanced waterfalls, standard hierarchy and P algorithm).

14.1 Functions

14.1.1 enhancedWaterfalls(imIn, imOut, grid=HEXAGONAL)

Enhanced waterfall algorithm. Compared to the classical waterfalls algorithm, this one adds the contours of the watershed transform which are above the hierarchical image associated to the next level of hierarchy. This waterfalls transform also ends to an empty set. All the hierarchical levels of image 'imIn' (which is a valued watershed) are computed. 'imOut' contains all these hierarchies which are embedded, so that hierarchy i is simply obtained by a threshold [i+1, 255] of image 'imOut'. 'imIn' and 'imOut' must be greyscale images. 'imIn' and 'imOu't must be different. This transformation returns the number of hierarchical levels.

14.1.2 extendedSegment(imIn, imTest, imOut, offset=255, grid=HEXAGONAL)

Extended (experimental) segmentation algorithm. This algorithm is controlled by image 'imTest'. The current hierarchical image is compared to image 'imTest'. This image must be a greyscale image. The 'offset' indicates which level of hierarchy is compared to the current hierarchical image. The 'offset' is relative to the current hierarchical level (by default, 'offset' is equal to 255, so that the initial segmentation is used). Image 'imOut' contains all these hierarchies which are embedded. 'imIn', 'imTest' and 'imOut' must be greyscale images. 'imIn', 'imTest' and 'imOut' must be different. This transformation returns the number of hierarchical levels.

14.1.3 generalSegment(imIn, imOut, gain=2.0, offset=1, grid=HEXAGONAL)

General segmentation algorithm. This algorithm is controlled by two parameters: the 'gain' (identical to the gain used in standard and P segmentation) and a new one, the 'offset'. The 'offset' indicates which level of hierarchy is compared to the current hierarchical image. The 'offset' is relative to the current hierarchical level. If 'offset' is equal to 1, this operator corresponds to the standard segmentation, if 'offset' is equal to 255 (this value stands for the infinity), the operator is equivalent to P algorithm. Image 'imOut' contains all these hierarchies which are embedded. 'imIn' and 'imOut' must be greyscale images. 'imIn' and 'imOut' must be different. This transformation returns the number of hierarchical levels.

14.1.4 hierarchicalLevel(imIn, imOut, grid=HEXAGONAL)

Computes the next hierarchical level of image 'imIn' in the waterfalls transformation and puts the result in 'imOut'. This operation makes sure that the next hierarchical level is embedded in the previous one. 'imIn' must be a valued watershed image.

14.1.5 hierarchy(imIn, imMask, imOut, grid=HEXAGONAL)

Construction of a hierarchical image from image 'imIn' and with 'imMask'. The binary image 'imMask' controls the dual reconstruction (propagation) of 'imIn'. This operator is mainly used to build hierarchical images from valued watershed images. The hierarchical image is put in 'imOut'.

14.1.6 segmentByP(imIn, imOut, gain=2.0, grid=HEXAGONAL)

General segmentation by P algorithm. This algorithm keeps or reintroduces the contours of the initial watershed transform which are above or equal to the hierarchical image associated to the next level of hierarchy when the altitude of the contour is multiplied by a 'gain' factor (default is 2.0). This transform also ends by idempotence. All the hierarchical levels of image 'imIn' (which is a valued watershed) are computed. 'imOut' contains all these hierarchies which are embedded, so that hierarchy i is simply obtained by a threshold [i+1, 255] of image

imOut. 'imIn' and 'imOut' must be greyscale images. 'imIn' and 'imOut' must be different. This transformation returns the number of hierarchical levels.

14.1.7 standardSegment(imIn, imOut, gain=2.0, grid=HEXAGONAL)

General standard segmentation. This algorithm keeps the contours of the watershed transform which are above or equal to the hierarchical image associated to the next level of hierarchy when the altitude of the contour is multiplied by a 'gain' factor (default is 2.0). This transform also ends by idempotence. All the hierarchical levels of image 'imIn'(which is a valued watershed) are computed. 'imOut' contains all these hierarchies which are embedded, so that hierarchy i is simply obtained by a threshold [i+1, 255] of image 'imOut'. 'imIn' and 'imOut' must be greyscale images. 'imIn' and 'imOut' must be different. This transformation returns the number of hierarchical levels.

14.1.8 waterfalls(imIn, imOut, grid=HEXAGONAL)

Classical waterfall algorithm. All the hierarchical levels of greyscale image 'imIn' (which must be a valued watershed) are computed. 'imOut' contains all these hierarchies which are embedded, so that hierarchy i is simply obtained by a threshold at [i+1, 255]. This transformation returns the number of hierarchical levels.

15 mamba.labellings

This module contains various labelling procedures for sets and for partitions. They use the label operator now available for sets and for grey images.

15.1 Functions

15.1.1 areaLabelling(imIn, imOut)

Special case of measure labelling where each connected component of the binary image 'imIn' or each cell of the partition 'imIn' is labelled with its area. The label image is stored in the 32-bit image 'imOut'.

15.1.2 diameterLabelling(imIn, imOut, dir, grid=HEXAGONAL)

Labels each connected component of the binary image 'imIn' with its diameter in direction 'dir'. The labelled image is stored in the 32-bit image 'imOut'. If 'imIn' is a 8-bit or 32-bit image, this function works too. However, the 0-valued connected components are labelled with 0. This procedure works on hexagonal or square grid. 'dir' can be any strictly positive integer value.

15.1.3 feretDiameterLabelling(imIn, imOut, direc)

The Feret diameter of each connected component of the binary image or the partition image 'imIn' is computed and its value labels the corresponding component. The labelled image is stored in the 32-bit image 'imOut'. If 'direc' is "vertical", the vertical Feret diameter is computed. If it is set to "horizontal", the corresponding diameter is used.

15.1.4 measureLabelling(imIn, imMeasure, imOut)

Labelling each particle of the binary image or each cell of the partition 'imIn' with the number of pixels in the binary image 'imMeasure' contained in each particle or each cell of the partition. The result is put is the 32-bit image 'imOut'.

15.1.5 partitionLabel(imIn, imOut)

This procedure labels each cell of image 'imIn' and puts the result in 'imOut'. The number of cells is returned. 'imIn' can be a 1-bit, 8-bit or a 32-bit image. 'imOut' is a 32-bit image. When 'imIn' is a binary image, all the connected components of the background are also labelled. When 'imIn' is a grey image, the 0-valued cells are also labelled (which is not the case with the label operator. Warning! The label values of adjacent cells are not necessarily consecutive.

15.1.6 volumeLabelling(imIn1, imIn2, imOut)

Each connected component of the binary image or the partition 'imIn1' is labelled with the volume of the greyscale or 32-bit image 'imIn2' inside this component. The result is put in the 32-bit image 'imOut'.

16 mamba.measure

Measure operators. This module provides a set of functions which perform measure operations on an image. Measures include volume, range, area ...

16.1 Functions

16.1.1 computeArea(imIn, scale=(1.0, 1.0))

Calculates the area of the binary image 'imIn'. 'scale' is a tuple containing the horizontal scale factor (distance between two adjacent horizontal points) and the vertical scale factor (distance between two successive lines) of image 'imIn' (default is 1.0 for both). The result is a float (when default values are used, the result value is identical to the computeVolume operator).

Note that, with hexagonal grid, the "scale' default values do not correspond to an isotropic grid (where triangles would be equilateral).

Beware, if the input image 'imIn' is not a binary image, the function raises an error.

16.1.2 computeComponentsNumber(imIn, grid=HEXAGONAL)

Computes the number of connected components in image 'imIn'. The result is an integer value.

16.1.3 computeConnectivityNumber(imIn, grid=HEXAGONAL)

Computes the connectivity number (Euler_Poincare constant) of image 'ImIn'. The result is an integer number.

Beware, if the input image 'imIn' is not a binary image, the function raises an error.

16.1.4 computeDiameter(imIn, dir, scale=(1.0, 1.0), grid=HEXAGONAL)

Computes the diameter (diametral variation) of binary image 'imIn' in direction 'dir'. 'scale' is a tuple defining the horizontal and vertical scale factors (default is 1.0).

Beware, if the input image 'imIn' is not a binary image, the function raises an error.

16.1.5 computeFeretDiameters(imIn, scale=(1.0, 1.0))

Computes the global Feret diameters (horizontal and vertical) of binary image 'imIn' and returns the result in a tuple (hDf, vDf). These diameters correspond to the horizontal and vertical dimensions of the smallest bonding box containing all the particles of 'imIn'

16.1.6 computeMaxRange(imIn)

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

16.1.7 computePerimeter(imIn, scale=(1.0, 1.0), grid=HEXAGONAL)

Computes the perimeter of all particles in binary image 'imIn' according to the Cauchy-Crofton formula. 'scale' is a tuple defining the horizontal and vertical scale factors (default is 1.0).

The edge of the image is always set to 'EMPTY'.

Beware, if the input image 'imIn' is not a binary image, the function raises an error.

16.1.8 computeRange(imIn)

Computes 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.

16.1.9 computeVolume(imIn)

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

'imIn' can be a 1-bit, 8-bit or 32-bit image.

17 mamba.miscellaneous

Various unclassed operators. This module regroups functions/operators that could not be regrouped with other operators because of their unique nature or other peculiarity. As such, it regroups some utility functions.

17.1 Functions

17.1.1 Mamba2PIL(imIn)

Creates and returns a PIL/PILLOW image using the Mamba image 'imIn'.

If the mamba image uses a palette, it will be integrated inside the PIL/PILLOW image.

17.1.2 PIL2Mamba(pilim, imOut)

The PIL/PILLOW image 'pilim' is used to load the Mamba image 'imOut'.

17.1.3 checkEmptiness(imIn)

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

'imIn' can be a 1-bit, 8-bit or 32-bit image.

17.1.4 compare(imIn1, imIn2, imOut)

Compares the two images 'imIn1' and 'imIn2'. The comparison is performed pixelwise by scanning the two images from top left to bottom right and it 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.

'imOut' is not reset at the beginning of the comparison.

'imIn1', imIn2' and 'imOut' can be 1-bit, 8-bit or 32-bit images of same size and depth.

17.1.5 drawEdge(imOut, thick=1)

Draws a frame around the edge of 'imOut' whose value equals the maximum range value and whose thickness is given by 'thick' (default 1).

17.1.6 extractFrame(imIn, threshold)

Extracts the smallest frame (tuple containing the coordinates of the upper left point and the lower right point) inside the image 'imIn' that includes all the pixels whose value is greater or equal to 'threshold'.

'imIn' can be a 8-bit or 32-bit image.

17.1.7 mix(imInR, imInG, imInB)

Mixes mamba images 'imInR' (red channel), 'imInG' (green channel) and 'imInB' (blue channel) into a color image. The function returns a PIL/PILLOW image.

17.1.8 multiSuperpose(imInout, *imIns)

Superposes multiple binary images ('imIns') to the greyscale image 'imInout'. The binary images are put above the greyscale. The result is meant to be seen with an appropriate color palette.

17.1.9 shift(imIn, imOut, d, amp, fill, grid=HEXAGONAL)

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

'grid' value can be HEXAGONAL or SQUARE and is set to DEFAULT_GRID by default.

'imIn' and 'imOut' can be 1-bit, 8-bit or 32-bit images of same size and depth.

17.1.10 shiftVector(imIn, imOut, vector, fill)

Shifts image 'imIn' by 'vector' (tuple with dx,dy). The emptied space is filled with 'fill' value. The result is put in 'imOut'.

'imIn' and 'imOut' can be 1-bit, 8-bit or 32-bit images of same size and depth.

17.1.11 split(pilimIn, imOutR, imOutG, imOutB)

Splits a color PIL/PILLOW image 'pilimIn' into its three color channels (Red, Green and Blue) and puts the three resulting images into 'imOutR', 'imOutG' and 'imOutB' respectively.

18 mamba.openclose

Opening and closing operators. This module provides a set of functions to perform opening and closing operations. All the closing and opening operation defined in this module use erosion, dilation and build functions with user-defined edge settings. The functions define a default edge which can be changed (see the modules erodil and geodesy).

18.1 Functions

18.1.1 buildClose(imIn, imOut, n=1, se=structuringElement([0, 1, 2, 3, 4, 5, 6], mamba.HEXAGONAL))

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

18.1.2 buildOpen(imIn, imOut, n=1, se=structuringElement([0, 1, 2, 3, 4, 5, 6], mamba.HEXAGONAL))

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

18.1.3 closing(imIn, imOut, n=1, se=structuringElement([0, 1, 2, 3, 4, 5, 6], mamba.HEXAGONAL), edge=FILLED)

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

The default edge is set to 'FILLED'. If 'edge' is set to 'EMPTY', the operation is slightly modified to avoid errors (non extensivity).

18.1.4 infClose(imIn, imOut, n, grid=HEXAGONAL)

Performs the infimum of directional closings. A black particle is preserved if its length is larger than 'n' in at least one direction.

This operator is a closing. The image edge is set to 'FILLED' in order to take into account particles touching the edge (they are supposed not to extend outside the image window).

When square grid is used, the size in oblique directions are reduced to be similar to the horizontal and vertical size.

18.1.5 linearClose(imIn, imOut, dir, n, grid=HEXAGONAL, edge=FILLED)

Performs a closing by a segment of size 'n' in direction 'dir'.

If 'edge' is set to 'EMPTY', the operation must be modified to remain extensive.

18.1.6 linearOpen(imIn, imOut, dir, n, grid=HEXAGONAL, edge=FILLED)

Performs an opening by a segment of size 'n' in direction 'dir'.
'edge' is set to 'FILLED' by default.

18.1.7 opening(imIn, imOut, n=1, se=structuringElement([0, 1, 2, 3, 4, 5, 6], mamba.HEXAGONAL), edge=FILLED)

Performs an opening operation on image 'imIn' and puts the result in 'imOut'. 'n' controls the size of the opening and 'se' the structuring element used.
The default edge is set to 'FILLED'. Note that the edge setting operates in the erosion only.

18.1.8 supOpen(imIn, imOut, n, grid=HEXAGONAL)

Performs the supremum of directional openings. A white particle is preserved (but not entirely) if its length is larger than 'n' in at least one direction.
This operator is an opening. The image edge is set to 'EMPTY' in order to take into account particles touching the edge (they are considered as entirely included in the image window).
When square grid is used, the size in oblique directions are reduced to be similar to the horizontal and vertical size.

19 mamba.partitions

Partitioning operators. This module contains operators acting on partitions. Two types of operators are defined: operators acting on each cell of the partition independently, or operators considering each cell of the partition as a node of a weighted graph. In the first case, each cell of the partition is considered as a binary set and the defined operation is applied on each cell to produce a new transformation (which is not necessarily a partition). In the second case, the partition is considered as a graph and morphological operators are defined on this graph. These operators provide morphological transformations on graphs, without the need to explicitly define this graph structure from the partition.

19.1 Functions

19.1.1 cellsBuild(imIn, imInOut, grid=HEXAGONAL)

Geodesic reconstruction of the cells of the partition image 'imIn' which are marked by the image 'imInOut'. The marked cells take the value of their corresponding marker. Note that the background cells (labelled by 0) are also modified if they are marked. The result is stored in 'imInOut'. The images can be 8-bit or 32-bit images. 'grid' can be set to HEXAGONAL or SQUARE.

19.1.2 cellsComputeDistance(imIn, imOut, grid=HEXAGONAL, edge=EMPTY)

Computation of the distance function for each cell of the partition image 'imIn'. The result is put in the 32-bit image 'imOut'. This operator works on hexagonal or square 'grid' and 'edge' is set to EMPTY by default.

19.1.3 cellsErode(imIn, imOut, n=1, se=structuringElement([0, 1, 2, 3, 4, 5, 6], mamba.HEXAGONAL), edge=FILLED)

Simultaneous erosion of size 'n' (default 1) of all the cells of the partition image 'imIn' with 'se' structuring element. The resulting partition is put in 'imOut'. 'edge' is set to FILLED by default. This operation works on 8-bit and 32-bit partitions.

19.1.4 cellsExtract(imIn, imMarkers, imOut, grid=HEXAGONAL)

Geodesic reconstruction and extraction of the cells of the partition image 'imIn' which are marked by the binary marker image 'imMarkers'. The marked cells keep their initial value. The result is stored in 'imOut'. The images can be 8-bit or 32-bit images. 'grid' can be set to HEXAGONAL or SQUARE.

19.1.5 cellsFullThin(imIn, imOut, dse, edge=EMPTY)

A full thinning transform is performed on each cell of the partition 'imIn' until idempotence. 'dse' is a double structuring element. The result is put in 'imOut'. 'edge' is set to EMPTY by default.

19.1.6 cellsHMT(imIn, imOut, dse, edge=EMPTY)

A Hit-Or-Miss transform is performed on each cell of the partition 'imIn'. 'dse' is a double structuring element (see thinthick.py module). The result is put in 'imOut'. 'edge' is set to EMPTY by default.

19.1.7 cellsOpen(imIn, imOut, n=1, se=structuringElement([0, 1, 2, 3, 4, 5, 6], mamba.HEXAGONAL), edge=FILLED)

Simultaneous opening of size 'n' (default 1) of all the cells of the partition image 'imIn' with 'se' structuring element. The resulting partition is put in 'imOut'. 'edge' is set to FILLED by default. This operation works on 8-bit and 32-bit partitions.

19.1.8 cellsOpenByBuild(imIn, imOut, n=1, se=structuringElement([0, 1, 2, 3, 4, 5, 6], mamba.HEXAGON

Opening by reconstruction of size 'n' (default 1) of the partition image 'imIn'. 'se' defines the structuring element. The result is put in 'imOut'. The images can be 8-bit or 32-bit images.

19.1.9 cellsThin(imIn, imOut, dse, edge=EMPTY)

Performs a simple thinning transform on each cell of the partition 'imIn'. 'dse' is a double structuring element (see thinthick.py module). The result is put in 'imOut'. 'edge' is set to EMPTY by default.

19.1.10 equalNeighbors(imIn, imOut, nb, grid=HEXAGONAL, edge=FILLED)

This operator compares the value of each pixel of image 'imIn' with the value of its neighbors encoded in 'nb'. If all the neighbor values are equal, the pixel is unchanged. Otherwise, it takes value 0. This operator works on hexagonal or square 'grid' and 'edge' is set to FILLED by default. This operator works for 8-bit and 32-bit images.

19.1.11 nonEqualNeighbors(imIn, imOut, nb, grid=HEXAGONAL, edge=FILLED)

This operator compares the value of each pixel of image 'imIn' with the value of its neighbors encoded in 'nb'. If all the neighbor values are different, the pixel is unchanged. Otherwise, it takes value 0. This operator works on hexagonal or square 'grid' and 'edge' is set to FILLED by default. This operator works for 8-bit and 32-bit images.

19.1.12 partitionDilate(imIn, imOut, n=1, grid=HEXAGONAL)

Graph dilation of the corresponding partition image 'imIn'. The size is given by 'n'. The corresponding partition image of the resulting dilated graph is put in 'imOut'. 'grid' can be set to HEXAGONAL or SQUARE.

19.1.13 partitionErode(imIn, imOut, n=1, grid=HEXAGONAL)

Graph erosion of the corresponding partition image 'imIn'. The size is given by 'n'. The corresponding partition image of the resulting eroded graph is put in 'imOut'. 'grid' can be set to HEXAGONAL or SQUARE.

20 mamba.residues

Residual operators. This module provides a set of functions to perform residual operations. A residual transformation is built by subtracting two sequences of primitive operators to get residues and by computing the supremum of these residues. The position in the sequence where this maximum occurs is also computed (it is called associated function and is generally a 32-bit image). These residues are defined on binary and greytone images.

20.1 Functions

20.1.1 binarySkeletonByOpening(imIn, imOut1, imOut2, grid=HEXAGONAL, edge=FILLED)

Skeleton by openings (maximal balls skeleton) of binary image 'imIn'. 'imOut1' contains the skeleton points (centers of maximal balls) and 'imOut2' contains the associated function (that is the radius of each maximal ball included in the initial set.

The operation is fast because it is computed through the use of the distance function of 'imIn' (skeleton points can be obtained by a Top Hat transform on the distance function).

The edge is set to 'FILLED' by default.

20.1.2 binaryUltimateErosion(imIn, imOut1, imOut2, grid=HEXAGONAL, edge=FILLED)

Ultimate erosion of binary image 'imIn'. 'imOut1' contains the ultimate eroded set and 'imOut2' contains the associated function (that is the height of each connected component of the ultimate erosion).

An ultimate erosion is composed of the union of the last connected components of the successive erosions of the initial set. The associated function provides the size of the corresponding erosion.

Depth of 'imOut1' is 1, depth of 'imOut2' is 32.

The operation is fast because it is computed using the distance function of 'imIn' (the ultimate erosion is identical to the maxima of this distance function).

The edge is set to 'FILLED' by default.

20.1.3 fullRegularisedGradient(imIn, imOut1, imOut2, grid=HEXAGONAL, maxSize=16)

Full regularised morphological gradient of image 'imIn'. This operator is a residual transform which uses the regularised gradient of size i as a residue. The range of sizes i is limited to 16 by default, as beyond this value, the residue is most of the time equal to 0.

Warning! 'imOut2' is a greyscale image (depth equal to 8).

20.1.4 quasiDistance(imIn, imOut1, imOut2, grid=HEXAGONAL)

Quasi-distance of image 'imIn'. 'imOut1' contains the residues image and 'imOut2' contains the quasi-distance (associated function).

The quasi-distance of a greytone image is made of a patch of distance functions of some almost flat regions in the image. When the image is a simple indicator function of a set, the quasi-distance and the distance function are identical.

Depth of 'imOut1' is the same as 'imIn', depth of 'imOut2' is 32.

20.1.5 skeletonByOpening(imIn, imOut1, imOut2, grid=HEXAGONAL)

General skeleton by openings working on greytone image 'imIn'. 'imOut1' contains the skeleton function and 'imOut2' contains the associated function.

This skeleton corresponds to the centers of maximal cylinders included in the set under the graph of the image 'imIn'.

Depth of 'imOut1' is the same as 'imIn', depth of 'imOut2' is 32.

The edge is always set to 'FILLED'.

20.1.6 ultimateBuildOpening(imIn, imOut1, imOut2, grid=HEXAGONAL)

Ultimate opening by build of image 'imIn'. 'imOut1' contains the ultimate opening whereas 'imOut2' contains the granulometric function.

This ultimate opening is obtained by using successive openings by build.

Depth of 'imOut1' is the same as 'imIn', depth of 'imOut2' is 32.

20.1.7 ultimateErosion(imIn, imOut1, imOut2, grid=HEXAGONAL)

General ultimate erosion working on greytone image 'imIn'. 'imOut1' contains the ultimate eroded function and 'imOut2' contains the associated function.

This ultimate erosion can be applied to greytone images.

Depth of 'imOut1' is the same as 'imIn', depth of 'imOut2' is 32.

The edge is always set to 'FILLED'.

20.1.8 ultimateIsotropicOpening(imIn, imOut1, imOut2, step=1, grid=HEXAGONAL)

Ultimate opening of image 'imIn' with more isotropic structuring elements. Dodecagons are used on hexagonal grid, octogons on square grid. 'imOut1' contains the ultimate opening whereas 'imOut2' contains the granulometric function. 'step' is the increment of the size of the openings. Its default value is 1 but it can be increased to reduce the computation time.

Depth of 'imOut1' is the same as 'imIn', depth of 'imOut2' is 32.

20.1.9 ultimateOpening(imIn, imOut1, imOut2, grid=HEXAGONAL)

Ultimate opening of image 'imIn'. 'imOut1' contains the ultimate opening whereas 'imOut2' contains the granulometric function.

Ultimate opening is obtained by using successive openings by hexagons or squares as primitive functions depending of the grid in use.

Depth of 'imOut1' is the same as 'imIn', depth of 'imOut2' is 32.

21 mamba.segment

Segmentation operators. This module provides a set of functions to perform segmentation operations (such as watershed and basin). The module also contains the labelling operator.

21.1 Functions

21.1.1 basinSegment(imIn, imMarker, grid=HEXAGONAL, max_level=0)

Segments image 'imIn' (greyscale or 32-bit) using the watershed algorithm. 'imMarker' is used both as the marker image (the wells from which the flooding proceeds) 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). A zero value will make the flooding run to its completion.

'grid' will change the number of neighbors considered by the algorithm (HEXAGONAL is 6-Neighbors and SQUARE is 8-Neighbors).

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

21.1.2 fastSKIZ(imIn, imOut, grid=HEXAGONAL)

Fast skeleton by zones of influence of binary image 'imIn'. Result is put in binary image 'imOut'. The transformation is faster as it uses the watershed transform by hierarchical queues.

21.1.3 geodesicSKIZ(imIn, imMask, imOut, grid=HEXAGONAL)

Geodesic skeleton by zones of influence of binary image 'imIn' inside the geodesic mask 'imMask'. The result is in binary image 'imOut'.

21.1.4 label(imIn, imOut, lblow=0, lbhigh=256, grid=HEXAGONAL)

Labels the image 'imIn' and puts the result in 32-bit image 'imOut'. Returns the number of connected components found by the labelling algorithm. The labelling will be performed according to the 'grid' (HEXAGONAL is 6-Neighbors and SQUARE is 8-Neighbors).

'lblow' and 'lbhigh' are used to restrain the possible values in the lower byte of 'imOut' pixel values. these values (and all their multiples of 256) are then reserved for another use (see Mamba User Manual for further details).

21.1.5 markerControlledWatershed(imIn, imMarkers, imOut, grid=HEXAGONAL)

Marker-controlled watershed transform of greytone image 'imIn'. The binary image 'imMarkers' contains the markers which control the flooding process. 'imOut' contains the valued watershed.

21.1.6 mosaic(imIn, imOut, imWts, grid=HEXAGONAL)

Builds the mosaic image of 'imIn' and puts the results into 'imOut'. The watershed line (pixel values set to 255) is stored in the greytone image 'imWts'. A mosaic image is a simple image made of various tiles of uniform grey values. It is built using the watershed of 'imIn' gradient and original markers made of gradient minima which are labelled by the maximum value of 'imIn' pixels inside them.

21.1.7 mosaicGradient(imIn, imOut, grid=HEXAGONAL)

Builds the mosaic-gradient image of 'imIn' and puts the result in 'imOut'. The mosaic-gradient image is built by computing the differences of two mosaic images generated from 'imIn', the first one having its watershed lines valued by the suprema of the adjacent catchment basins values, the second one been valued by the infima.

21.1.8 valuedWatershed(imIn, imOut, grid=HEXAGONAL)

Returns the valued watershed of greyscale image 'imIn' into greyscale image 'imOut'. Each pixel of the watershed lines is given its corresponding value in initial image 'imIn'.

21.1.9 watershedSegment(imIn, imMarker, grid=HEXAGONAL, max_level=0)

Segments image 'imIn' (greyscale or 32-bit) using the watershed algorithm. 'imMarker' is used both as the marker image (the wells from which the flooding proceeds) 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). A zero value will make the flooding run to its completion.

'grid' will change the number of neighbors considered by the algorithm (HEXAGONAL is 6-Neighbors and SQUARE is 8-Neighbors).

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

22 mamba.statistic

Statistical operators. This module provides a set of functions to compute statistical values such as mean and median inside an image.

22.1 Functions

22.1.1 getHistogram(imIn)

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

22.1.2 getMean(imIn)

Returns the average value (float) of the pixels of 'imIn' (which must be a greyscale image).

22.1.3 getMedian(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 pixels are below it.

'imIn' must be a greyscale image.

22.1.4 getVariance(imIn)

Returns the pixels variance (estimator without bias) of image 'imIn' (which must be a greyscale image)..

23 mamba.thinthick

Thinning and thickening operators. This module contains morphological thinning and thickening operators based on the Hit-or-Miss transformation, together with various homotopic and geodesic functions derived from these operators. The module also defines the double structuring element class which serve as a base for these operators.

23.1 Classes

23.1.1 doubleStructuringElement

This class allows to define a doublet of structuring elements used in a coded format by Hit-or-Miss, thin and thick operations and their corresponding methods. The coding corresponds to the output of the 'hitormissPatternSelector' tool available in the extra module (mambaDisplay package).

__init__(self, *args) Double structuring element constructor. A double structuring element is defined by the first (background points) and second (foreground points) structuring elements.

You can define it in two ways:

- doubleStructuringElement(se0, se1): where 'se0' and 'se1' are instances of the class structuring element found in erodil module. These structuring elements must be defined on the same grid.
- doubleStructuringElement(dse0, dse1, grid): where 'dse0' and 'dse1' are direction lists and 'grid' defines the grid on which the two structuring elements are defined.

If the constructor is called with inapropriate arguments, it raises a ValueError exception.

__repr__(self)

flip(self) Flips the doublet of structuring elements. Flipping corresponds to a swap: the doublet (se0, se1) becomes (se1, se0).

getCSE(self) Returns the coded values corresponding to the background and foreground structuring elements se0 and se1 in a tuple so they can be used with the HitOrMiss function.

getGrid(self) Returns the grid on which the double stucturing element is defined.

getStructuringElement(self, ground) Returns the structuring element of the foreground if 'ground' is set to 1 or the structuring element of the background otherwise.

rotate(self, step=1) Rotates the double structuring element 'step' times (default=1). When 'step' is positive, rotation is clockwise. When 'step' is negative, rotation is counterclockwise. No rotation occurs when 'step' equals zero.

23.2 Functions

23.2.1 blackClip(imIn, imOut, step=0, grid=HEXAGONAL)

Performs a black skeleton clipping (clipping of a black skeleton image). If 'step' is not defined (or equal to 0), the clipping is performed until idempotence. If 'step' is defined, 'step' black points (if possible) will be removed from each branch of the black skeleton.

'edge' is always set to FILLED.

23.2.2 computeSKIZ(imIn, imOut, grid=HEXAGONAL)

Computes the influence zones of each connected component of 'imIn' and puts the result in 'imOut'. Inverting the result produces the skeleton by influence zones (SKIZ).

There exists a much faster way to compute the SKIZ operation (see segment.py module).

23.2.3 endPoints(imIn, imOut, grid=HEXAGONAL, edge=FILLED)

Extracts end points in 'imIn', supposed to be a "skeleton" image (connected components without thickness), and puts them in 'imOut'.

'edge' is FILLED by default and it can be modified to take into account extremities touching the edge.

23.2.4 fullGeodesicThick(imIn, imMask, imOut, dse)

Performs a complete geodesic thickening (until idempotence) of image 'imIn' inside mask 'imMask' with all the rotations of the double structuring element 'dse'. The result is put in 'imOut'.

23.2.5 fullGeodesicThin(imIn, imMask, imOut, dse)

Performs a complete geodesic thinning (until idempotence) of image 'imIn' inside mask 'imMask' with all the rotations of the double structuring element 'dse'. The result is put in 'imOut'.

23.2.6 fullThick(imIn, imOut, dse)

Performs a complete thickening of 'imIn' with the successive rotations of 'dse' (until idempotence) and puts the result in 'imOut'.

'imIn' and 'imOut' are binary images.

The edge is always set to EMPTY.

23.2.7 fullThin(imIn, imOut, dse, edge=EMPTY)

Performs a complete thinning of 'imIn' with the successive rotations of 'dse' (until idempotence) and puts the result in 'imOut'.

'imIn' and 'imOut' are binary images.

'edge' is set to EMPTY by default.

23.2.8 geodesicThick(imIn, imMask, imOut, dse)

Geodesic thickening of image 'imIn' inside 'imMask' by the double structuring element 'dse'. The result is stored in 'imOut'.

'imIn', 'imMask' and 'imOut' are binary images.

23.2.9 geodesicThin(imIn, imMask, imOut, dse)

Geodesic thinning of image 'imIn' inside 'imMask' by the double structuring element 'dse'. The result is stored in 'imOut'.

'imIn', 'imMask' and 'imOut' are binary images.

23.2.10 hitOrMiss(imIn, imOut, dse, edge=EMPTY)

Performs a binary Hit-or-miss operation on image 'imIn' using the double structuring element 'dse'. Result is put in 'imOut'.

WARNING! 'imIn' and 'imOut' must be different images.

'edge' value can be EMPTY or FILLED.

You can also find a helper function (hitormissPatternSelector) in the mambaExtra module.

23.2.11 homotopicReduction(imIn, imOut, grid=HEXAGONAL)

Reduces any simply connected component of 'imIn' (component without holes) to a single point. All other components are reduced to simpler ones, with same homotopy as the initial ones. This transformation is simply thinD on the hexagonal grid. It is a combination of thinnings with D and E structuring elements on square grid.

23.2.12 infThin(imIn, imOut, dse, edge=EMPTY)

Performs an inf of thinnings, each thinning being made with the successive rotations of 'dse'. The initial image 'imIn' is used at each step of thinning (intersection of thinnings).

'imIn' and 'imOut' are binary images.

'edge' is set to EMPTY by default.

23.2.13 multiplePoints(imIn, imOut, grid=HEXAGONAL)

Extracts multiple points in 'imIn', supposed to be a "skeleton" image (connected components without thickness), and puts the result in 'imOut'.

Note that, on a square grid, the resulting skeleton is supposed to be defined on a 4-connectivity grid. if it is not the case, some multiple points are likely to be missed.

23.2.14 rotatingGeodesicThick(imIn, imMask, imOut, dse)

Performs successive geodesic thickenings of 'imIn' inside 'imMask' with clockwise rotations of the double structuring element 'dse'. The number of rotations is either 6 or 8 according to the grid where 'dse' is defined. All the thickenings are concatenated.

'imIn', 'imMask' and 'imOut' are binary images.

23.2.15 rotatingGeodesicThin(imIn, imMask, imOut, dse)

Performs successive geodesic thinnings of 'imIn' inside 'imMask' with clockwise rotations of the double structuring element 'dse'. The number of rotations is either 6 or 8 according to the grid where 'dse' is defined. All the thinnings are concatenated.

'imIn', 'imMask' and 'imOut' are binary images.

23.2.16 rotatingThick(imIn, imOut, dse)

Performs a complete rotation of thickenings, the initial 'dse' double structuring element being turned one step clockwise after each thickening. At each rotation step, the previous result is used as input for the next thickening (chained thickenings). Depending on the grid where 'dse' is defined, 6 or 8 rotations are performed.

'imIn' and 'imOut' are binary images.

The edge is always set to EMPTY.

23.2.17 rotatingThin(imIn, imOut, dse, edge=FILLED)

Performs a complete rotation of thinnings , the initial 'dse' double structuring element being turned one step clockwise after each thinning. At each rotation step, the previous result is used as input for the next thinning (chained thinnings). Depending on the grid where 'dse' is defined, 6 or 8 rotations are performed.

'imIn' and 'imOut' are binary images.

'edge' is set to FILLED by default (default value is EMPTY in simple thin).

23.2.18 supThick(imIn, imOut, dse)

Performs a sup of thickenings, each thickening being made with the successive rotations of 'dse'. The initial image 'imIn' is used at each step of thickening (union of thickenings).

'imIn' and 'imOut' are binary images.

The edge is always set to EMPTY.

23.2.19 thick(imIn, imOut, dse)

Elementary thickening operator with 'dse' double structuring element. The

'imIn' and 'imOut' are binary images.

The edge is always EMPTY (as for hitOrMiss).

23.2.20 thickD(imIn, imOut, grid=HEXAGONAL)

Complete thickening with D structuring element. Depending on the grid in use, hexaM or squareM will be used. M structuring element is the flipping of D for the thickening.

This operator must be used with binary images.

23.2.21 thickL(imIn, imOut, grid=HEXAGONAL)

Complete thickening with L structuring element. Depending on the grid in use, hexagonalL or squareL will be used. Note that L is equal to its flipping (same structuring element is used in thinning and thickening).

This operator must be used with binary images.

The edge is always EMPTY.

23.2.22 thickM(imIn, imOut, grid=HEXAGONAL)

Complete thickening with M structuring element. Depending on the grid in use, hexagonalD or squareD will be used. D structuring element is the flipping of M structuring element for the thickening.

This operator must be used with binary images.

23.2.23 thin(imIn, imOut, dse, edge=EMPTY)

Elementary thinning operator with 'dse' double structuring element.
'imIn' and 'imOut' are binary images.
'edge' is set to EMPTY by default.

23.2.24 thinD(imIn, imOut, grid=HEXAGONAL, edge=EMPTY)

Complete thinning with D structuring element. Depending on the grid in use, hexagonalD or squareD will be used. This operator is mainly used to simplify each connected component to the simplest homotopic equivalent set (see homotopicReduction in this module).
This operator must be used with binary images.
The edge is set to EMPTY by default.

23.2.25 thinL(imIn, imOut, grid=HEXAGONAL, edge=EMPTY)

Complete thinning with L structuring element. Depending on the grid in use, hexagonalL or squareL will be used. This operator is also called skeleton, as it produces a result which looks like a connected skeleton of each connected component of 'imIn'.
This operator must be used with binary images.
The edge is set to EMPTY by default.

23.2.26 thinM(imIn, imOut, grid=HEXAGONAL, edge=EMPTY)

Complete thinning with M structuring element. Depending on the grid in use, hexagonalM or squareM will be used. This operator produces skeletons with lots of fishbones.
This operator must be used with binary images.
The edge is set to EMPTY by default.

23.2.27 whiteClip(imIn, imOut, step=0, grid=HEXAGONAL, edge=FILLED)

Performs a skeleton clipping of 'imIn' (supposed to contain a skeleton image) and puts the result in 'imOut'. If 'step' is not defined (or equal to 0), the clipping is performed until idempotence. If 'step' is defined, 'step' points (if possible) will be removed from each branch of the skeleton.
'edge' is set to FILLED by default.

24 mamba3D.arithmetic3D

Arithmetic and logical 3D operators. This module provides arithmetic operators such as addition, subtraction, multiplication, division and logical operators between 3D images. (and, or, not, xor ...).

24.1 Functions

24.1.1 add3D(imIn1, imIn2, imOut)

Adds 'imIn2' pixel values to 'imIn1' pixel values and puts the result in 'imOut'. The operation can be sum up in the following formula:
imOut = imIn1 + imIn2.
You can mix formats in the addition operation (a binary image can be added to a greyscale 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 greyscale images (e.g. on a 8-bit greyscale image, 255+1=255). With 32-bit images, the addition is not saturated.

24.1.2 addConst3D(imIn, v, imOut)

Adds 'imIn' pixel values to value 'v' and puts the result in 'imOut'. The operation can be sum up in the following formula:
imOut = imIn + v
'imIn' and imOut' can be 8-bit or 32-bit images of same size and depth.
The operation is saturated (limited to 255) for greyscale images.

24.1.3 ceilingAdd3D(imIn1, imIn2, imOut)

Adds image 'imIn2' to image 'imIn1' and puts the result in 'imOut'. If imIn1 + imIn2 is larger than the maximal possible value in imOut, the result is truncated and limited to this maximal value.

Although it is possible to use a 8-bit image for imIn2, it is recommended to use the same depth for all the images.

Note that this operator is mainly useful for 32-bit images, as the result of the addition is always truncated for 8-bit images.

24.1.4 ceilingAddConst3D(imIn, v, imOut)

Adds a constant value 'v' to image 'imIn' and puts the result in 'imOut'. If imIn + v is larger than the maximal possible value in imOut, the result is truncated and limited to this maximal value.

Note that this operator is mainly useful for 32-bit images, as the result of the addition is always truncated for 8-bit images.

24.1.5 diff3D(imIn1, imIn2, imOut)

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 write 0 otherwise:

imOut = imIn1 if imIn1 > imin2 imOut = 0 otherwise.

'imIn1', imIn2' and 'imOut' can be 1-bit, 8-bit or 32-bit images of same size and depth.

24.1.6 div3D(imIn1, imIn2, imOut)

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

imOut = imIn1 / imIn2

You can mix formats in the divide operation. However you must ensure that the output image is as deep as 'imIn1'.

In order to avoid divisions by zero, the result of the operation is set to the maximal pixel value whenever the corresponding pixel in 'imIn2' is equal to zero.

24.1.7 divConst3D(imIn, v, imOut)

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

imOut = imIn / v (or more acurately : imIn = imOut * v + r, r being the ignored reminder)

A zero value in 'v' will return an error. For a 8-bit image, v will be restricted between 1 and 255. You cannot use it with binary images.

24.1.8 floorSub3D(imIn1, imIn2, imOut)

subtracts image 'imIn2' from image 'imIn1' and puts the result in 'imOut'. If imIn1 - imIn2 is negative, the result is truncated and limited to 0.

Although it is possible to use a 8-bit image for imIn2, it is recommended to use the same depth for all the images.

Note that this operator is mainly useful for 32-bit images, as the result of the subtractiontion is always truncated for 8-bit images.

24.1.9 floorSubConst3D(imIn, v, imOut)

Subtracts a constant value 'v' to image 'imIn' and puts the result in 'imOut'. If imIn - v is negative, the result is truncated and limited to 0.

Note that this operator is mainly useful for 32-bit images, as the result of the subtraction is always truncated for 8-bit images.

24.1.10 logic3D(imIn1, imIn2, imOut, log)

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

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

"and" performs a bitwise AND operation, "or" a bitwise OR and "xor" a bitwise XOR. "inf" calculates the minimum and "sup" the maximum between corresponding pixel values.

'imIn1', imIn2' and 'imOut' can be 1-bit, 8-bit or 32-bit images of same size and depth.

24.1.11 mul3D(imIn1, imIn2, imOut)

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

imOut = imIn1 * imIn2

You can mix formats in the multiply operation (a binary image can be multiplied with a greyscale 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 greyscale images (e.g. on a greyscale image 255*255=255).

24.1.12 mulConst3D(imIn, v, imOut)

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

imOut = imIn * v

The operation is saturated for greyscale images. You cannot use it with binary images.

24.1.13 mulRealConst3D(imIn, v, imOut, nearest=False, precision=2)

Multiplies image 'imIn' by a real positive constant value 'v' and puts the result in image 'imOut'. 'imIn' and 'imOut' can be 8-bit or 32-bit images. If 'imOut' is greyscale (8-bit), the result is saturated (results of the multiplication greater than 255 are limited to this value). 'precision' indicates the number of decimal digits taken into account for the constant 'v' (default is 2). If 'nearest' is true, the result is rounded to the nearest integer value. If not (default), the result is simply truncated.

24.1.14 negate3D(imIn, imOut)

Negates the 3D image 'imIn' and puts the result in 'imOut'.

The operation is a binary complement for binary images and a negation for greyscale and 32-bit images.

24.1.15 sub3D(imIn1, imIn2, imOut)

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

imOut = imIn1 - imIn2

You can mix formats in the subtraction operation (a binary image can be subtracted to a greyscale 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) but not for 32-bit images.

24.1.16 subConst3D(imIn, v, imOut)

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

imOut = imIn - v

'imIn' and imOut' can be 8-bit or 32-bit images of same size and depth.

The operation is saturated (lower limit is 0) for greyscale images.

25 mamba3D.base3D

3D Image class definition. This is the base module of the Mamba 3D Image library. It defines the image3DMb class used to contain images. A 3D image can be considered as a stack or sequence of 2D images. The module also defines an alias to image3DMb named sequenceMb.

25.1 Classes

25.1.1 image3DMb

A 3D image is represented by an instance of this class.

__del__(self)

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

__init__(self, *args, **kwargs) Constructor for a mamba 3D image. A 3D image is a stack of images defined by width, height and length (the number of 2D images in it).

There is a wide range of possibilities :

- image3DMb() : without arguments will create an empty greyscale 3D image.
- image3DMb(im3D) : will create a 3D image using the same size, depth and length than 3D image 'im3D'.
- image3DMb(im2D) : will create a 3D image using the same size and depth than 2D image 'im2D'.
- image3DMb(depth) : will create a 3D image with the desired 'depth' (1, 8 or 32) for the mamba images.
- image3DMb(path) : will load the 3D image (sequence) located in 'path', see the load method.
- image3DMb(im3D, depth) : will create a 3D image using the same size than 3D image 'im3D' and the specified 'depth'.
- image3DMb(im2D, length) : will create a 3D image using the same size than 2D image 'im2D' and the specified 'length'.
- image3DMb(path, depth) : will load the 3D image (sequence) located in 'path' and convert it to the specified 'depth'.
- image3DMb(width, height, length) : will create a 3D image with size 'width'x'height' and 'length'.
- image3DMb(width, height, length, depth) : will create a 3D image with size 'width'x'height', 'depth' and 'length'.

When not specified, the width, height and length of the 3D image will be set to 256. The default depth is 8 (greyscale).

When loading a 3D image as a sequence make sure all the images have the same size.

__iter__(self) Makes a mamba image sequence iterable.

__len__(self) Returns the length of the 3D image.

__next__(self)

__str__(self)

convert(self, depth) Converts the image depth to the given 'depth'. The conversion algorithm is identical to the conversion used in the convert3D function (see this function for details).

extractRaw(self) Extracts and returns the image raw string data. This method only works on 8 and 32-bit images.

fill(self, v) Fills the 3D image with value 'v' A zero value makes the image completely dark.

freeze(self) Called to freeze the display of the image. Thus the image may change but the display will not show these modfications until the method unfreeze is called.

getDepth(self) Returns the depth of the 3D image.

getName(self) Returns the name of the 3D image.

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

getSize(self) Returns the size (tuple with width, height and length) of the 3D image.

hide(self) Called to hide the display associated to the image.

load(self, path, rgbfilter=None) Loads a 3D stack (sequence) of images found in directory 'path'.

To be valid, a sequence of images must be composed of at least 'length' images to be able to fill the sequence. Their file names must be of the form X.ext where X is a number and ext is an image file extension (like jpg or png). The sequence will be read in increasing order (1 then 2 and so on). However it is not mandatory that the numbers follow each other (1 then 5 is legal). You can also mix image formats (1.bmp then 2.jpg is legal) but you should make sure that files that are not images (txt, pdf, ...) are not named following that pattern. Also ensure that two images do not get the same number (like 1 and 01).

loadRaw(self, dataOrPath, preprocfunc=None) Loads raw data inside the 3D image. You can give a filename or data directly through 'dataOrPath'. The data length must match the image size: width * height * length * (depth/8) If needed you can preprocess the data using the optional argument 'preprocfunc' which will be called on the data before loading it. The preprocfunc must have the following prototype: outdata = preprocfunc(indata). The size verification is performed after the preprocessing (enabling you to use zip archives and such). This method only works on 8 and 32-bit images.

next(self) The next method for the iteration.

reset(self) Resets the 3D image (all the pixels are put to 0).

save(self, path, extension='.png', palette=None) Saves the images of the 3D image stack (sequence) inside a directory located at 'path'.

The function will create the directory if it doesn't exist. If the directory exists and already contains images they will be overwritten. The images are stored following this pattern (1.png, 2.png ...). You can modify the format of the image by changing the optional parameter 'extension' (refer to PIL documentation for supported format).

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

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

show(self, **options) Called to show the display associated to the image.

You can specify 'options' that will be given to the displayer.

unfreeze(self) Called to unfreeze the display of the image. Thus the image display will be updated along with the modifications inside the image.

update(self) Called when the display associated to the image must be updated (Contrary to mamba.imageMb, display is not automatically updated after any operation on your image due to loss of performance). You can update the display by hitting key F5 in the display window.

25.1.2 sequenceMb(image3DMb)

A sequence of images is represented by an instance of this class. This is a complete alias to image3DMb class kept for compatibility reasons.

26 mamba3D.contrasts3D

Contrast 3D operators. This module provides a set of functions to perform morphological contrast operators such as gradient, top-hat transform, ...

26.1 Functions

26.1.1 blackTopHat3D(imIn, imOut, n, se=structuringElement3D([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12], mamba3D.FACE_CENTER_CUBIC))

Performs a black Top Hat operation on 'imIn' and puts the result in 'imOut'. This operator extracts from 'imIn' the dark objects thinner than 2*'n'+1.

The structuring element used is defined by 'se' ('CUBOCTAHEDRON1' by default).

26.1.2 gradient3D(imIn, imOut, n=1, se=structuringElement3D([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12], mamba3D.FACE_CENTER_CUBIC))

Computes the morphological gradient of 3D image 'imIn' and puts the result in 'imOut'. The thickness can be controlled using parameter 'n' (1 by default). The structuring element used by the erosion and dilation is defined by 'se' (CUBOCTAHEDRON1 by default).

26.1.3 halfGradient3D(imIn, imOut, type='intern', n=1, se=structuringElement3D([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12], mamba3D.FACE_CENTER_CUBIC))

Computes the half morphological gradient of 3D image 'imIn' ond puts the result in 'imOut'.

'type' indicates if the half gradient should be internal or external. Possible values are : "extern" : dilation(imIn) - imIn "intern" : imIn - erosion(imIn)

The thickness can be controlled using parameter 'n' (1 by default). The structuring element used by the erosion or the dilation is defined by 'se'.

26.1.4 regularisedGradient3D(imIn, imOut, n, grid=mamba3D.FACE_CENTER_CUBIC)

Computes the regularized gradient of 3D image 'imIn' of size 'n'. The result is put in 'imOut'. A regularized gradient of size 'n' extracts in the 3D image contours thinner than 'n' while avoiding false detections.

This operation is only valid for omnidirectional structuring elements.

26.1.5 supBlackTopHat3D(imIn, imOut, n, grid=mamba3D.FACE_CENTER_CUBIC)

Performs a black Top Hat operation with the infimum of directional openings on 'imIn' and puts the result in 'imOut'. This operator partly extracts from 'imIn' the dark objects whose extension in at least one direction of 'grid' is smaller than 'n'.

26.1.6 supWhiteTopHat3D(imIn, imOut, n, grid=mamba3D.FACE_CENTER_CUBIC)

Performs a white Top Hat operation with the supremum of directional openings on 'imIn' and puts the result in 'imOut'. This operator partly extracts from 'imIn' the bright objects whose extension in at least one direction of 'grid' is smaller than 'n'.

26.1.7 whiteTopHat3D(imIn, imOut, n, se=structuringElement3D([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12], mamba3D.FACE_CENTER_CUBIC))

Performs a white Top Hat operation on 'imIn' and puts the result in 'imOut'. This operator extracts from 'imIn' the bright objects thinner than 2*'n'+1.

The structuring element used is defined by 'se' ('CUBOCTAHEDRON1' by default).

27 mamba3D.conversion3D

Depth conversion 3D operators. This module regroups various functions/operators to perform conversions based on image depth. It allows to transfer data from a 3D image depth to another.

27.1 Functions

27.1.1 convert3D(imIn, imOut)

Converts the contents of 'imIn' to the depth of 'imOut' and puts the result in 'imOut'.

Greyscale or 32-bit to binary and binary to greyscale or 32-bit conversions are supported. Value 255 in a greyscale image or 2**32-1 in a 32-bit image are considered as 1 in a binary one. All other values are transformed to 0. The reverse convention applies.

This function can also be used to downscale 32-bit images into greyscale images.

27.1.2 convertByMask3D(imIn, imOut, mFalse, mTrue)

Converts a binary image 'imIn' into a greyscale image (8-bit) or a 32-bit image and puts the result in 'imOut'.

White pixels of 'imIn' are set to value 'mTrue' in the output image and the black pixels set to value 'mFalse'.

This function works with 3D images.

27.1.3 generateSupMask3D(imIn1, imIn2, imOut, strict)

Generates a 3D 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 3D image 'imIn1' than in 3D image 'imIn2'.

'imIn1' and imIn2' can be 1-bit, 8-bit or 32-bit images of same size, length and depth.

27.1.4 lookup3D(imIn, imOut, lutable)

Converts the greyscale 3D image 'imIn' using the look-up table 'lutable' and puts the result in greyscale 3D image 'imOut'.

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

27.1.5 threshold3D(imIn, imOut, 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.

'imIn' can be a 8-bit or 32-bit image.

This function works with 3D images.

28 mamba3D.copies3D

Copy 3D operators. This module regroups various complete or partial copy operators for 3D images.

28.1 Functions

28.1.1 copy3D(imIn, imOut, firstPlaneIn=0, firstPlaneOut=0)

Copies 3D image 'imIn' into 'imOut'. 'firstPlaneIn' indicates the starting plane inside 'imIn' and 'firstPlaneOut' the starting plane inside 'imOut'.

28.1.2 copyBitPlane3D(imIn, plane, imOut)

Inserts or extracts a bit plane in a 3D image. If 'imIn' is a binary image, it is inserted at 'plane' position in greyscale 'imOut'. If 'imIn' is a greyscale image, its bit plane at 'plane' position is extracted and put into binary image 'imOut'.

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

28.1.3 copyBytePlane3D(imIn, plane, imOut)

Inserts or extracts a byte plane in a 3D image. If 'imIn' is a greyscale image, it is inserted at 'plane' position in 32-bit 'imOut'. If 'imIn' is a 32-bit image, its byte plane at 'plane' position is extracted and put into 'imOut'.

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

28.1.4 cropCopy3D(imIn, posin, imOut, posout, size)

Crops a cube of size 'size' at position 'posin' in image 'imIn' and inserts it in image 'imOut' at position 'posout'. 'posin', 'posout' and 'size' are tuples containing width, height and length values. This operator is similar to the 2D cropCopy operator. It shares the same restrictions.

29 mamba3D.draw3D

Drawing 3D operators. This module defines functions to draw inside Mamba 3D images. Drawing functions include lines, cubes, ... The module also provides functions to extract pixel information.

29.1 Functions

29.1.1 drawCube(imOut, cube, value)

Draws a cube in 'imOut' using the tuple 'cube' containing 4 values (nearest upper left to farest down right corners (x1,y1,z1,x2,y2,z2)) using 'value' to set the pixels.

29.1.2 drawLine3D(imOut, line, value)

Draws a line in 'imOut' using the tuple 'line' containing 4 values (starting and ending points (x1,y1,z1,x2,y2,z2)) using 'value' to set the pixels.

This function uses the Bresenham algorithm. It works with image3DMb instances.

29.1.3 getIntensityAlongLine3D(imOut, line)

Returns in a list the intensity profile along a line in 'imOut' using the tuple 'line' containing 6 values (starting and ending points (x1,y1,z1,x2,y2,z2)).

This function uses the Bresenham algorithm. It works with image3DMb instances.

30 mamba3D.erodil3D

Erosion and dilation 3D operators. This module provides a set of functions and class to perform erosions and dilations. The module contains basic and complex operators that are based on neighbor comparisons. In particular it defines the 3D structuring element class which serves as the base for these operators. The module also contains distance functions based on erosion.

30.1 Classes

30.1.1 structuringElement3D

__eq__(self, otherSE)

__init__(self, directions, grid) Structuring element constructor. A structuring element is defined by the couple 'directions' (given in an ordered list) and 'grid'. You cannot define a structuring element that holds a direction more than once.

You can look at the predefined structuring elements to get examples of how to make yours.

__ne__(self, otherSE)

__repr__(self)

getDirections(self, withoutZero=False) Returns a copy of the directions used by the structuring element. if 'withoutZero' is set to True the returned direction list will not include direction 0 (useful for some operators, such as erode or dilate, where direction 0 modifies the initial conditions).

getGrid(self) Returns the grid associated with the structuring element.
Example:

```
>>>CUBOCTAHEDRON1.getGrid()
mamba3D.FACE_CENTER_CUBIC
```

hasZero(self) Returns True if the central point (0) is included in the direction list.

transpose(self) Structuring element transposition (symmetry around the origin).

30.2 Functions

30.2.1 computeDistance3D(imIn, imOut, grid=mamba3D.FACE_CENTER_CUBIC, edge=EMPTY)

Computes for each white pixel of binary 3D 'imIn' the minimum distance to reach a connected component boundary while constantly staying in the set. The result is put in 32-bit 'imOut'.

The distance computation will be performed according to the 'grid' (CUBIC is 26-Neighbors and FACE_CENTER_CUBIC is 12-Neighbors, CENTER_CUBIC is unsupported by this operator). 'edge' can be FILLED or EMPTY.

30.2.2 dilate3D(imIn, imOut, n=1, se=structuringElement3D([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12], mamba3D.FACE_CENTER_CUBIC), edge=EMPTY)

This operator performs a dilation, using the structuring element 'se' (set by default as CUBOCTAHEDRON1), of 3D image 'imIn' and puts the result in 'imOut'. The operation is repeated 'n' times (default is 1). This operator assumes an 'EMPTY' edge by default.

This operator always considers that the origin of the structuring element in use is at position 0 even if this point does not belong to it.

30.2.3 dilateByCylinder3D(imInOut, height, section)

Dilates 3D image 'imInOut' using a cylinder with an hexagonal section of size 2x'section' and a height of 2x'height'. The image is modified by this function. The edge is always set to EMPTY.

30.2.4 erode3D(imIn, imOut, n=1, se=structuringElement3D([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12], mamba3D.FACE_CENTER_CUBIC), edge=FILLED)

This operator performs an erosion, using the structuring element 'se' (set by default as CUBOCTAHEDRON1), of 3D image 'imIn' and puts the result in 'imOut'. The operation is repeated 'n' times (default is 1). This operator assumes a 'FILLED' edge by default.

This operator always considers that the origin of the structuring element in use is at position 0 even if this point does not belong to it.

30.2.5 erodeByCylinder3D(imInOut, height, section)

Erodes 3D image 'imInOut' using a cylinder with an hexagonal section of size 2x'section' and a height of 2x'height'. The image is modified by this function. The edge is always set to FILLED.

30.2.6 linearDilate3D(imIn, imOut, d, n=1, grid=mamba3D.FACE_CENTER_CUBIC, edge=EMPTY)

Dilation by a segment in direction 'd' of 3D image 'imIn', result in 'imOut'. The operation is repeated 'n' times (default is 1).This function will assume an EMPTY edge unless specified otherwise using 'edge'. The directions are defined according to the grid in use.

30.2.7 linearErode3D(imIn, imOut, d, n=1, grid=mamba3D.FACE_CENTER_CUBIC, edge=FILLED)

Performs an erosion in direction 'd' of 3D image 'imIn' and puts the result in 'imOut'. The operation is repeated 'n' times (default is 1). This function will assume a FILLED edge unless specified otherwise using 'edge'.

31 mamba3D.erodilLarge3D

Erosion and dilation 3D operators for large structuring elements. This module provides a set of functions and class to perform erosions and dilations with large structuring elements on 3D images.

31.1 Functions

31.1.1 infFarNeighbor3D(imIn, imInOut, nb, amp, grid=mamba3D.FACE_CENTER_CUBIC, edge=FILLED)

Performs an infimum operation between the 'imInOut' 3D image pixels and their neighbor 'nb' at distance 'amp' according to 'grid' in 3D image 'imIn'. The result is put in 'imInOut'. "grid' value can be CUBIC, CENTER_CUBIC or FACE_CENTER_CUBIC. 'edge' value can be EMPTY or FILLED.

31.1.2 largeCubeDilate(imIn, imOut, size, edge=EMPTY)

Dilation by a large cube using dilations by large segments and the Steiner decomposition property of the cube. No edge effects are likely to happen with a cubic structuring element.

31.1.3 largeCubeErode(imIn, imOut, size, edge=FILLED)

Erosion by a large cube using erosions by large segments and the Steiner decomposition property of the cube. No edge effects are likely to happen with a cubic structuring element.

31.1.4 largeLinearDilate3D(imIn, imOut, dir, size, grid=mamba3D.FACE_CENTER_CUBIC, edge=EMPTY)

Dilation of the 3D image 'imIn' by a large segment in direction 'dir' in a reduced number of iterations. Uses the dilations by doublets of points (supposed to be faster, thanks to an enhanced shift operator).

31.1.5 largeLinearErode3D(imIn, imOut, dir, size, grid=mamba3D.FACE_CENTER_CUBIC, edge=FILLED)

Erosion of a 3D image 'imIn' by a large segment in direction 'dir' in a reduced number of iterations. Uses the erosions by doublets of points (supposed to be faster, thanks to an enhanced shift operator).

31.1.6 supFarNeighbor3D(imIn, imInOut, nb, amp, grid=mamba3D.FACE_CENTER_CUBIC, edge=EMPTY)

Performs a supremum operation between the 'imInOut' 3D image pixels and their neighbor 'nb' at distance 'amp' according to 'grid' in 3D image 'imIn'. The result is put in 'imInOut'. "grid' value can be CUBIC, CENTER_CUBIC or FACE_CENTER_CUBIC. 'edge' value can be EMPTY or FILLED.

32 mamba3D.extrema3D

Extrema (min/max) 3D operators. This module provides a set of operators dealing with maxima and minima of a function. New operators linked to the notion of dynamics are also provided.

32.1 Functions

32.1.1 deepMinima3D(imIn, imOut, h, grid=mamba3D.FACE_CENTER_CUBIC)

Computes the minima of the dual reconstruction of image 'imIn' by imIn + h and puts the result in 'imOut'.
Grid used by the dual build operation can be specified by 'grid'.
Only works with 8-bit or 32-bit images as input. 'imOut' must be binary.

32.1.2 highMaxima3D(imIn, imOut, h, grid=mamba3D.FACE_CENTER_CUBIC)

Computes the maxima of the reconstruction of image 'imIn' by imIn - h and puts the result in 'imOut'.
Grid used by the build operation can be specified by 'grid'.
Only works with 8-bit or 32-bit images as input. 'imOut' must be binary.

32.1.3 maxDynamics3D(imIn, imOut, h, grid=mamba3D.FACE_CENTER_CUBIC)

Extracts the maxima of 'imIn' with a dynamics higher or equal to 'h' and puts the result in 'imOut'.
Grid used by the dual build operation can be specified by 'grid'.
Only works with 8-bit or 32-bit images as input. 'imOut' must be binary.

32.1.4 maxPartialBuild3D(imIn, imMask, imOut, grid=mamba3D.FACE_CENTER_CUBIC)

Performs the partial reconstruction of 'imIn' with its maxima which are contained in the binary mask 'imMask'. The result is put in 'imOut'.
'imIn' and 'imOut' must be different and greyscale images.

32.1.5 maxima3D(imIn, imOut, h=1, grid=mamba3D.FACE_CENTER_CUBIC)

Computes the maxima of 'imIn' using a build operation and puts the result in 'imOut'.
'h' can be used to define the maxima height. Grid used by the build operation can be specified by 'grid'.
Only works with 8-bit or 32-bit as input. 'imOut' must be binary.

32.1.6 minDynamics3D(imIn, imOut, h, grid=mamba3D.FACE_CENTER_CUBIC)

Extracts the minima of 'imIn' with a dynamics higher or equal to 'h' and puts the result in 'imOut'.
Grid used by the dual build operation can be specified by 'grid'.
Only works with 8-bit or 32-bit images as input. 'imOut' must be binary.

32.1.7 minPartialBuild3D(imIn, imMask, imOut, grid=mamba3D.FACE_CENTER_CUBIC)

Performs the partial reconstruction of 'imIn' with its minima which are contained in the binary mask 'imMask'. The result is put in 'imOut'.
'imIn' and 'imOut' must be different and greyscale images.

32.1.8 minima3D(imIn, imOut, h=1, grid=mamba3D.FACE_CENTER_CUBIC)

Computes the minima of 'imIn' using a dual build operation and puts the result in 'imOut'.
'h' can be used to define the minima depth. Grid used by the dual build operation can be specified by 'grid'.
Only works with 8-bit or 32-bit as input. 'imOut' must be binary.

33 mamba3D.filter3D

Filtering operators. This module provides a set of functions to perform morphological filtering operations such as alternate filters.

33.1 Functions

33.1.1 alternateFilter3D(imIn, imOut, n, openFirst, se=structuringElement3D([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12], mamba3D.FACE_CENTER_CUBIC))

Performs an alternate filter operation of size 'n' on 3D image 'imIn' and puts the result in 'imOut'. If 'openFirst' is True, the filter begins with an opening, a closing otherwise.

33.1.2 autoMedian3D(imIn, imOut, n, se=structuringElement3D([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12], mamba3D.FACE_CENTER_CUBIC))

Morphological automedian filter performed with alternate sequential filters.

33.1.3 fullAlternateFilter3D(imIn, imOut, n, openFirst, se=structuringElement3D([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12], mamba3D.FACE_CENTER_CUBIC))

Performs a full alternate filter operation (successive alternate filters of increasing sizes, from 1 to 'n') on 3D 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.

33.1.4 linearAlternateFilter3D(imIn, imOut, n, openFirst, grid=mamba3D.FACE_CENTER_CUBIC)

Performs an alternate filter operation on 3D image 'imIn' with openings and closings by segments of size 'n' (supremeum of openings and infimum of closings) and puts the result in 'imOut'. If 'openFirst' is True, the filter begins with an opening, a closing otherwise.

33.1.5 simpleLevelling3D(imIn, imMask, imOut, grid=mamba3D.FACE_CENTER_CUBIC)

Performs a simple levelling of 3D image 'imIn' controlled by image 'imMask' and puts the result in 'imOut'. This operation is composed of two geodesic reconstructions. This filter tends to level regions in the image of homogeneous grey values.

33.1.6 strongLevelling3D(imIn, imOut, n, eroFirst, grid=mamba3D.FACE_CENTER_CUBIC)

Strong levelling of 'imIn', result in 'imOut'. 'n' defines the size of the erosion and dilation of 'imIn' in the operation. If 'eroFirst' is true, the operation starts with an erosion, it starts with a dilation otherwise.

This filter is stronger (more efficient) that simpleLevelling3D. However, the order of the initial operations (erosion and dilation) matters.

34 mamba3D.geodesy3D

Geodesic 3D operators. This module provides a set of functions to perform geodesic computations. It includes build and dualbuild operations, geodesic erosion and dilation ...

34.1 Functions

34.1.1 build3D(imMask, imInout, grid=mamba3D.FACE_CENTER_CUBIC)

Builds 3D image 'imInout' using 'imMask' as a mask. This operator performs the geodesic reconstruction of 'imInout' inside the mask image and puts the result in the same image.

This operator uses a hierarchical implementation of the reconstruction.

This function will use the mamba3D default grid unless specified otherwise in 'grid'. It works only with grids FACE_CENTER_CUBIC and CUBIC.

34.1.2 buildNeighbor3D(imMask, imInOut, d, grid=mamba3D.FACE_CENTER_CUBIC)

Builds 3D image 'imInout' in direction 'd' according to 'grid' using 'imMask' as a mask (the propagation is performed only in 'd' direction).

The function also returns the volume of the image 'imInout' after the build operation.

'grid' value can be any 3D grid.

34.1.3 closeHoles3D(imIn, imOut, grid=mamba3D.FACE_CENTER_CUBIC)

Close holes in 3D image 'imIn' and puts the result in 'imOut'. This operator works on binary and greytone images. In this case, however, it should be used cautiously.

34.1.4 dualBuild3D(imMask, imInout, grid=mamba3D.FACE_CENTER_CUBIC)

Builds (dual build) 3D image 'imInout' using 'imMask' as a mask. This operator performs the geodesic dual reconstruction (by erosions) of 'imInout' inside the mask image and puts the result in the same image.

This operator uses a hierarchical implementation of the reconstruction.

This function will use the mamba3D default grid unless specified otherwise in 'grid'. It works only with grids FACE_CENTER_CUBIC and CUBIC.

34.1.5 dualbuildNeighbor3D(imMask, imInOut, d, grid=mamba3D.FACE_CENTER_CUBIC)

Dual builds image 'imInout' in direction 'd' according to 'grid' using 'imMask' as a mask (the propagation is performed only in 'd' direction).

The function also returns the volume of the image 'imInout' after the build operation.

'grid' value can be any 3D grid.

34.1.6 geodesicDilate3D(imIn, imMask, imOut, n=1, se=structuringElement3D([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12], mamba3D.FACE_CENTER_CUBIC))

This operator is simply an alias of lowerGeodesicDilate3D. It is kept for compatibility reasons.

34.1.7 geodesicErode3D(imIn, imMask, imOut, n=1, se=structuringElement3D([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12], mamba3D.FACE_CENTER_CUBIC))

This transformation is identical to the previous version and it has been kept for compatibilty purposes.

Note that the binary and the greytone operators are different.

34.1.8 lowerGeodesicDilate3D(imIn, imMask, imOut, n=1, se=structuringElement3D([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12], mamba3D.FACE_CENTER_CUBIC))

Performs a lower geodesic dilation of 3D image 'imIn' below 'imMask'. The result is put inside 'imOut', 'n' controls the size of the dilation. 'se' specifies the type of structuring element used to perform the computation (CUBOCTAHEDRON by default).

Warning! 'imMask' and 'imOut' must be different.

34.1.9 lowerGeodesicErode3D(imIn, imMask, imOut, n=1, se=structuringElement3D([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12], mamba3D.FACE_CENTER_CUBIC))

Performs a lower geodesic erosion of 3D image 'imIn' under 'imMask'. The result is put inside 'imOut', 'n' controls the size of the erosion. 'se' specifies the type of structuring element used to perform the computation (CUBOCTAHEDRON by default).

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

Warning! 'imMask' and 'imOut' must be different.

34.1.10 removeEdgeParticles3D(imIn, imOut, grid=mamba3D.FACE_CENTER_CUBIC)

Removes particles (connected components) touching the edge in 3D image 'imIn'. The resulting image is put in image 'imOut'. Although this operator may be used with greytone images, it should be considered with caution.

34.1.11 upperGeodesicDilate3D(imIn, imMask, imOut, n=1, se=structuringElement3D([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12], mamba3D.FACE_CENTER_CUBIC))

Performs a upper geodesic dilation of 3D image 'imIn' above 'imMask'. The result is put inside 'imOut', 'n' controls the size of the dilation. 'se' specifies the type of structuring element used to perform the computation (CUBOCTAHEDRON by default).

Warning! 'imMask' and 'imOut' must be different.

34.1.12 upperGeodesicErode3D(imIn, imMask, imOut, n=1, se=structuringElement3D([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12], mamba3D.FACE_CENTER_CUBIC))

Performs a upper geodesic erosion of 3D image 'imIn' above 'imMask'. The result is put inside 'imOut', 'n' controls the size of the erosion. 'se' specifies the type of structuring element used to perform the computation (CUBOCTAHEDRON by default).

Warning! 'imMask' and 'imOut' must be different.

35 mamba3D.grids3D

3D Grids handling and setting functions. This module defines various functions related to grid configurations and computations. It also defines the 3D grids (cubic, face-centered cubic ...) used with the 3D operators.

35.1 Functions

35.1.1 getDirections3D(grid=mamba3D.FACE_CENTER_CUBIC, withoutZero=False)

Returns a range of all the possible directions available in 'grid' (set to DEFAULT_GRID3D by default). If 'withoutZero' is set to True, the direction 0 is omitted.

If the 'grid' value is incorrect, the function returns an empty list.

35.1.2 gridNeighbors3D(grid=mamba3D.FACE_CENTER_CUBIC)

Returns the number of neighbors of a point in 'grid'.

If the 'grid' value is incorrect, the function returns 0.

35.1.3 setDefaultGrid3D(grid)

This function will change the value of the default grid used in each operator that needs to specify one.

'grid' must be a valid 3D grid.

You can of course manually change the variable DEFAULT_GRID3D by yourself. Using this function is however recommended if you are not sure of what you are doing.

35.1.4 transposeDirection3D(d, grid=mamba3D.FACE_CENTER_CUBIC)

Calculates the transposed (opposite) direction value of direction 'd'

36 mamba3D.measure3D

Measure 3D operators. This module provides a set of functions which perform measure operations on a 3D image. Measures include volume, range ...

36.1 Functions

36.1.1 computeMaxRange3D(imIn)

Returns a tuple with the minimum and maximum possible pixel values given the depth of 3D image 'imIn'. The values are returned in a tuple holding the minimum and the maximum.

36.1.2 computeRange3D(imIn)

Computes the range, i.e. the minimum and maximum values, of 3D image 'imIn'. The values are returned in a tuple holding the minimum and the maximum.

36.1.3 computeVolume3D(imIn)

Computes the volume of the 3D image 'imIn', i.e. the sum of its pixel values. The computed integer value is returned by the function.

'imIn' can be a 1-bit, 8-bit or 32-bit image.

Be aware that because this operator runs on 3D image, the returned value can be very high.

37 mamba3D.miscellaneous3D

Various unclassed 3D operators. This module regroups functions/operators that could not be regrouped with other operators because of their unique nature or other peculiarity. As such, it regroups some utility functions.

37.1 Functions

37.1.1 checkEmptiness3D(imIn)

Checks if 3D image 'imIn' is empty (i.e. completely black). Returns True if so, False otherwise.

'imIn' can be a 1-bit, 8-bit or 32-bit image.

37.1.2 compare3D(imIn1, imIn2, imOut)

Compares the two 3D images 'imIn1' and 'imIn2'. The comparison is performed pixelwise by scanning the two images from top left to bottom right starting with plane 0 and it 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,-1) if the two images are identical.

'imOut' is not reset at the beginning of the comparison.

'imIn1', imIn2' and 'imOut' can be 1-bit, 8-bit or 32-bit images of same size and depth.

37.1.3 drawEdge3D(imOut, thick=1)

Draws a frame around the edge of 'imOut' whose value equals the maximum range value and whose thickness is given by 'thick' (default 1).

37.1.4 shift3D(imIn, imOut, d, amp, fill, grid=mamba3D.FACE_CENTER_CUBIC)

Shifts 3D image 'imIn' in direction 'd' of the 'grid' over an amplitude of 'amp'. The emptied space is filled with 'fill' value. This implementation is fast as a minimal number of shifts is used. The result is put in 'imOut'.

38 mamba3D.openclose3D

Opening and closing operators. This module provides a set of functions to perform opening and closing operations. All the closing and opening operation defined in this module use erosion, dilation and build functions with user-defined edge settings. The functions define a default edge which can be changed (see the modules erodil3D and geodesy3D).

38.1 Functions

38.1.1 buildClose3D(imIn, imOut, n=1, se=structuringElement3D([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12], mamba3D.FACE_CENTER_CUBIC))

Performs a closing by dual reconstruction operation on 3D image 'imIn' and puts the result in 'imOut'. 'n' controls the size of the closing.

38.1.2 buildOpen3D(imIn, imOut, n=1, se=structuringElement3D([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12], mamba3D.FACE_CENTER_CUBIC))

Performs an opening by reconstruction operation on 3D image 'imIn' and puts the result in 'imOut'. 'n' controls the size of the opening.

38.1.3 closeByCylinder3D(imInOut, height, section)

Closing using the dilation and erosion by a cylinder.

38.1.4 closing3D(imIn, imOut, n=1, se=structuringElement3D([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12], mamba3D.FACE_CENTER_CUBIC), edge=FILLED)

Performs a closing operation on 3D image 'imIn' and puts the result in 'imOut'. 'n' controls the size of the closing and 'se' the structuring element used.

The default edge is set to 'FILLED'. If 'edge' is set to 'EMPTY', the operation is slightly modified to avoid errors (non extensivity).

38.1.5 infClose3D(imIn, imOut, n, grid=mamba3D.FACE_CENTER_CUBIC)

Performs the infimum of directional closings. A black particle is preserved if its length is larger than 'n' in at least one direction.

This operator is a closing. The image edge is set to 'FILLED' in order to take into account particles touching the edge (they are supposed not to extend outside the image window).

When square grid is used, the size in oblique directions are reduced to be similar to the horizontal and vertical size.

38.1.6 linearClose3D(imIn, imOut, dir, n, grid=mamba3D.FACE_CENTER_CUBIC, edge=FILLED)

Performs a closing by a segment of size 'n' in direction 'dir'.

If 'edge' is set to 'EMPTY', the operation must be modified to remain extensive.

38.1.7 linearOpen3D(imIn, imOut, dir, n, grid=mamba3D.FACE_CENTER_CUBIC, edge=FILLED)

Performs an opening by a segment of size 'n' in direction 'dir'.

'edge' is set to 'FILLED' by default.

38.1.8 openByCylinder3D(imInOut, height, section)

Opening using the dilation and erosion by a cylinder.

38.1.9 opening3D(imIn, imOut, n=1, se=structuringElement3D([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12], mamba3D.FACE_CENTER_CUBIC), edge=FILLED)

Performs an opening operation on 3D image 'imIn' and puts the result in 'imOut'. 'n' controls the size of the opening and 'se' the structuring element used.

The default edge is set to 'FILLED'. Note that the edge setting operates in the erosion only.

38.1.10 supOpen3D(imIn, imOut, n, grid=mamba3D.FACE_CENTER_CUBIC)

Performs the supremum of directional openings. A white particle is preserved (but not entirely) if its length is larger than 'n' in at least one direction.

This operator is an opening. The image edge is set to 'EMPTY' in order to take into account particles touching the edge (they are considered as entirely included in the image window).

When square grid is used, the size in oblique directions are reduced to be similar to the horizontal and vertical size.

39 mamba3D.segment3D

Segmentation 3D operators. This module provides a set of functions to perform 3D segmentation operations (such as watershed and basin). The module also contains the labelling operator.

39.1 Functions

39.1.1 basinSegment3D(imIn, imMarker, grid=mamba3D.FACE_CENTER_CUBIC, max_level=0)

Segments 3D image 'imIn' (greyscale or 32-bit) using the watershed algorithm. 'imMarker' is used both as the marker image (the wells from which the flooding proceeds) and as the output image. It is a 32-bit 3D 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).

'grid' will change the number and position of neighbors considered by the algorithm.

The result is put inside 'imMarker'. The three first byte planes contain 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 faster than watershedSegment3D if you are not interested in the watershed line.

This operator works only with grids FACE_CENTER_CUBIC and CUBIC.

39.1.2 fastSKIZ3D(imIn, imOut, grid=mamba3D.FACE_CENTER_CUBIC)

Fast skeleton by zones of influence of binary 3D image 'imIn'. Result is put in binary 3D image 'imOut'. The transformation is faster as it uses the watershed transform by hierarchical queues.

39.1.3 label3D(imIn, imOut, lblow=0, lbhigh=256, grid=mamba3D.FACE_CENTER_CUBIC)

Labels the 3D image 'imIn' and puts the result in 32-bit 3D image 'imOut'. Returns the number of connected components found by the labeling algorithm. The labelling will be performed according to the 'grid'.

'lblow' and 'lbhigh' are used to restrain the possible values in the lower byte of 'imOut' pixel values. these values (and all their multiples of 256) are then reserved for another use (see Mamba User Manual for further details).

This operator works only with grids FACE_CENTER_CUBIC and CUBIC.

39.1.4 markerControlledWatershed3D(imIn, imMarkers, imOut, grid=mamba3D.FACE_CENTER_CUBI

Marker-controlled watershed transform of greyscale or 32-bit 3D image 'imIn'. The binary 3D image 'imMarkers' contains the markers which control the flooding process. 'imOut' contains the valued watershed.

39.1.5 mosaic3D(imIn, imOut, imWts, grid=mamba3D.FACE_CENTER_CUBIC)

Builds the mosaic 3D image of 'imIn' and puts the results into 'imOut'. The watershed line (pixel values set to 255) is stored in the greytone 3D image 'imWts'. A mosaic image is a simple image made of various tiles of uniform grey values. It is built using the watershed of 'imIn' gradient and original markers made of gradient minima which are labelled by the maximum value of 'imIn' pixels inside them.

39.1.6 mosaicGradient3D(imIn, imOut, grid=mamba3D.FACE_CENTER_CUBIC)

Builds the mosaic-gradient 3D image of 'imIn' and puts the result in 'imOut'. The mosaic-gradient image is built by computing the differences of two mosaic images generated from 'imIn', the first one having its watershed lines valued by the suprema of the adjacent catchment basins values, the second one been valued by the infima.

39.1.7 valuedWatershed3D(imIn, imOut, grid=mamba3D.FACE_CENTER_CUBIC)

Returns the valued watershed of greyscale or 32-bit 3D image 'imIn' into greyscale 3D image 'imOut'. Each pixel of the watershed lines is given its corresponding value in initial image 'imIn'.

39.1.8 watershedSegment3D(imIn, imMarker, grid=mamba3D.FACE_CENTER_CUBIC, max_level=0)

Segments greyscale or 32-bit 3D image 'imIn' using the watershed algorithm. 'imMarker' is used both as the marker image (the wells from which the flooding proceeds) and as the output image. It is a 32-bit 3D 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).

'grid' will change the number and position of neighbors considered by the algorithm.

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

This operator works only with grids FACE_CENTER_CUBIC and CUBIC.

40 mamba3D.statistic3D

Statistical 3D operators. This module provides a set of functions to compute statistical values such as mean and median inside a 3D image.

40.1 Functions

40.1.1 getHistogram3D(imIn)

Returns a list holding the histogram of the greyscale 3D image 'imIn' (0 to 255).

40.1.2 getMean3D(imIn)

Returns the average value (float) of the pixels of 'imIn' (which must be a greyscale 3D image).

40.1.3 getMedian3D(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 pixels are below it.

'imIn' must be a greyscale 3D image.

40.1.4 getVariance3D(imIn)

Returns the pixels variance (estimator without bias) of 3D image 'imIn' (which must be a greyscale image).

41 mamba3D.thinthick3D

Thinning and thickening 3D operators. This module contains morphological thinning and thickening operators based on the Hit-or-Miss transformation. The module also defines the double 3D structuring element class which serves as a base for these operators.

41.1 Classes

41.1.1 doubleStructuringElement3D

This class allows to define a doublet of structuring elements used in a coded format by Hit-or-Miss, thin and thick operations and their corresponding methods.

__init__(self, *args) Double structuring 3D element constructor. A double structuring element is defined by the first (background points) and second (foreground points) structuring elements 3D.
You can define it in two ways:

- doubleStructuringElement3D(se0, se1): where 'se0' and 'se1' are instances of the class structuringElement3D found in erodil3D module. These structuring elements must be defined on the same grid.
- doubleStructuringElement3D(dse0, dse1, grid): where 'dse0' and 'dse1' are direction lists and 'grid' defines the grid on which the two structuring elements are defined.

If the constructor is called with inapropriate arguments, it raises a ValueError exception.

__repr__(self)

flip(self) Flips the doublet of structuring elements. Flipping corresponds to a swap: the doublet (se0, se1) becomes (se1, se0).

getGrid(self) Returns the grid on which the double stucturing element is defined.

getStructuringElement3D(self, ground) Returns the structuring element of the foreground if 'ground' is set to 1 or the structuring element of the background otherwise.

41.2 Functions

41.2.1 hitOrMiss3D(imIn, imOut, dse, edge=EMPTY)

Performs a binary Hit-or-miss operation on 3D image 'imIn' using the doubleStructuringElement3D 'dse'. Result is put in 'imOut'.
WARNING! 'imIn' and 'imOut' must be different images.

41.2.2 thick3D(imIn, imOut, dse)

Elementary thickening operator with 'dse' double structuring element. 'imIn' and 'imOut' are binary 3D images.
The edge is always EMPTY (as for mamba.hitOrMiss).

41.2.3 thin3D(imIn, imOut, dse, edge=EMPTY)

Elementary thinning operator with 'dse' double structuring element. 'imIn' and 'imOut' are binary 3D images.
'edge' is set to EMPTY by default.

42 mambaDisplay

Display and palette functions. mambaDisplay contains all the elements allowing to display mamba images (2D and 3D). The documentation for this package is intended for advanced users who wish to define their own display. Palette definitions. Basic users can use it to list and use existing palettes and build new ones through the provided functions.

42.1 Classes

42.1.1 Displayer

This generic class is provided to allow advanced users to define their own way to display mamba images. To do so, you must create your own displayer class inheriting this one. Then call the setDisplayer function from this package like this : setDisplayer(displayer=your_own_displayer_class) As an example, you can look into the mambaDisplay package to see the standard displayer provided with mamba based on Tkinter.

addWindow(self, im) Creates a window for mamba or mamba3D image 'im' (imageMb or image3DMb). You can access name, palette information and other information related to the mamba image object.

The function must return the id of the window (also called its key) that the mamba image (imageMb) will store for later interaction with the display. If an error occurred, returns an empty string.

controlWindow(self, wKey, ctrl) Method used to control the display of a window identified by 'wKey'. The 'ctrl' parameter indicates the type of operation to perform. Here are the value the displayer must support : - "FREEZE" : freeze the display so that update will no longer be possible until the window is unfreezed. - "UNFREEZE" : unfreeze the display and automatically update it.

Other controls must be ignored quietly.

destroyWindow(self, wKey) Destroys the window identified by 'wKey'.

hideWindow(self, wKey) Method used to hide a window from the screen. 'wKey' indicates the particular window to withdraw.

The function can be called even if the window is already hidden, iconified or withdrawn.

showWindow(self, wKey, **options) Method used to recall and redisplay a window that has been hidden, iconified or withdrawn from the screen. 'wKey' indicates the particular window to redisplay. 'options' can be used to control specific element of the windows. It depends on the displayer.

The function is also called right after the creation of the window. It can be called even if the window was not hidden, iconified or withdrawn.

tidyWindows(self) Tidies the display to ensure that all the windows are visible.

In particular, this method is called by the mamba tidyDisplays() function.

updateWindow(self, wKey) If an event occurred that modified the mamba image associated to window 'wKey', this method will be called.

For optimization sake, it is advised to disregard calls to this function when the concerned window is hidden.

42.2 Functions

42.2.1 getDisplayer()

Returns the reference to the displayer used by mamba images when they need to be displayed.

42.2.2 setDisplayer(displayer)

Will set the 'displayer' to use. Use this function to overrun the default display. This will have no effect if you have already displayed images.

42.2.3 setMaxDisplay(size)

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

42.2.4 setMinDisplay(size)

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

42.2.5 tidyDisplays()

Tidies the displayed images. This function will try to optimize, given the actual screen size, the position of the images so that every one may be visible (not always) possible if many images are displayed).

42.2.6 addPalette(name, palette)

Adds a 'palette' with its 'name' to the existing palettes. A palette is a tuple containing 256*3 value (r0,g0,b0,r1,g1,b1 ...).

42.2.7 getPalette(name)

Returns the palette with 'name'.

42.2.8 listPalettes()

Returns the list of all the defined palettes.

42.2.9 tagOneColorPalette(value, color)

Creates a palette that tags a specific 'value' inside an image with a given 'color', a tuple (red, green, blue), while the rest of the image stays in greyscale.

43 mambaDisplay.extra

Extra displays. This module defines specific extra displays that are meant to be used interactively with the user. This module is not loaded by default with mambaDisplay.

43.1 Functions

43.1.1 dynamicThreshold(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 a tuple (low, high) used to obtain the image displayed using the threshold operation 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 keys S (increasing) and X (decreasing).

43.1.2 hitormissPatternSelector(grid=HEXAGONAL)

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

The function returns a double structuring elements 'es0' and 'es1' (in that order) used as entry in the hitOrMiss 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 hitOrMiss function : hitOrMiss(imIn, imOut, hitormissPatternSelector())

43.1.3 interactiveSegment(imIn, imOut)

Opens an interactive display where you can select the marker and perform a segmentation using the watershed algorithm on greyscale image 'imIn'.

Once the close button is pressed, the result of the segmentation is returned in 32-bit imgae 'imOut'.

Returns the list of markers selected by the user (list of tuple in the (x,y) format).

43.1.4 superpose(imIn1, imIn2)

Draws 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 (default color) if the pixel is set in both images, green (default color) if the pixel is set only in 'imIn1' and red (default color) if it is only set in 'imIn2'.

If one image is greyscale and the other is binary, the binary image is redrawn over the greyscale image in purple (default color).

Image superposition is not possible if both images are greyscale.

The default colors can be changed while displaying by clicking the corresponding color box in the caption above the display window. A color palette will appear where a new color can be selected.