SMIL - smilPython Quick Reference

Revision : 1.0.4 - November 15, 2024

1 Introduction

This document contains a short list of functions (for `Python` users) with a quick description. The complete documentation can be found at `https://smil.cmm.minesparis.psl.eu/doc/modules.html`.

Notes

- this isn't an exhaustive listing of all features available on Smil, but just the most common features. For a complete listing, please refer to the complete documentation (*RTFM*);
- some functions may have some other variants on the parameters (number or presence). Most of the time, these functions are marked with a **"*"**. Take a look on the complete documentation;
- To make **smilPython** available in your programs you must type one of the following commands:

```
from smilPython import *
import smilPython as sp

```

- parameters between square brackets indicates an optional parameter. For example :

```
# erode() function prototype :
erode(imIn, imOut[, se])
# function call
erode(imIn, imOut)
erode(imIn, imOut, SquSE())
erode(imIn, imOut, CrossSE(3))

```

- if not specified, default Structuring Element is used in most morphological functions, usualy the last one;
- **all Smil functions** returns some value : a **data** or a **result** indicating an error condition (**1** means **no error**);
- output image passed as a parameter **must always** be created (even if not initialized) before any function call.

2 Functions Quick Reference

I/O images	
Function	**Description**
`im = Image('images/toto.png')` `im = Image('http://server/toto.png')`	Read an image from file Read an image from some internet server.
`im2 = Image(im)`	Create an image (as `im`) - no content copy
`im2 = Image(im,True)`	Create an image (as `im`) and copy its content
`im2 = Image(im,'UINT16')`	Create an image (as `im`) but with another data type

`read('titi.png',im)`
`write(im,'titi.png')`
`getHttpFile(url,fout)`
	Read a file (or a list of files for 3D) into an image
	Write *im* into a file (or a list of files for 3D)
	Retrieve a file from internet

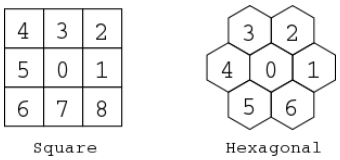
`im.show()` `im.showLabel()`	Display an image Display an image with false colors
`val = im.getPixel(x,y[,z])` `im.setPixel(x,y[,z],val)`	Get and set pixel values

`width = im.getWidth()`
`height = im.getHeight()`
`depth = im.getDepth()`
`dimensions = getDimensions()`
	Get image size
	Get image dimensions (2D or 3D)

`count = im.getPixelCount()`	Number of pixels ($width \times height \times depth$)

Numpy interface	
Function	**Description**
im.fromNumArray(numpyArray**)**	Fill an image from a ***Numpy*** array
arr = **im.getNumArray(**c_contigous = False**)**	Get an image data pointer as a ***Numpy*** array

Define, modify and restore default Structuring Element	
Function	**Description**
se = **Morpho.getDefaultSE()**	Get default Structuring Element
Morpho.setDefaultSE(CrossSE()**)**	Set default Structuring Element to CrossSE()
se.printSelf()	Print Structuring Element content
mySE = **StrElt(**HexFlag,PointList**)**	Construct a structuring element with points defined by their indexes. Ex. : mySE = StrElt(False,(0,1,5) equals HorizSE()



Structuring Element Grids

Pre-defined Structuring Elements			
Structuring Element	**Grid**	**Points**	**Description**
CrossSE()	Square	5	(0,1,3,5,7)
SquSE()	Square	9	(0,1,2,3,4,5,6,7)
HorizSE()	Square	3	(0,1,5)
VertSE()	Square	3	(0,3,7)
HexSE()	Hexagonal	7	(0,1,2,3,4,5,6)
Cross3DSE()	Cubic	7	(0,1,3,5,7,8,16)
CubeSE()	Cubic	27	(0,1,2,3,4,5,6,7,8,...,26)
RhombicubeoctaedronSE()	Cubic	81	(..)
LineSE(len,θ**)**	Square		2D line with arbitrary length and angle starting from origin
Line3DSE(len,θ,ϕ**)**	Square		3D line with arbitrary length and angle starting from origin

Operations on Structuring Elements	
Methods	**Description**
se = **StrElt(**size = 1 | se**)**	Constructor
se = **transpose()**	
se = **merge()**	
se = **homothety()**	
se = **clone()**	
se = **noCenter()**	
se = **operator(**size = 1**)**	
setName(name = ""**)**	
name = **getName()**	
addPoint(Point | x,y[,z]**)**	
point = **getPoint(**i**)**	
point = **getPoint(**i**)**	
printSelf()	

Basic Morphological Operators : Erosion, Dilation, Opening, Closing	
Function	**Description**
erode(imIn,imOut[,se]**)**	Erosion
dilation(imIn,imOut[,se]**)**	Dilation
open(imIn,imOut[,se]**)**	Opening
close(imIn,imOut[,se]**)**	Closing

Morphological Filters	
Function	**Description**
open(imIn,imOut[,se]**)**	Opening
close(imIn,imOut[,se]**)**	Closing
asfOpen(imIn,imOut[,se]**)**	Alternate Open/Close with sizes from 1 to $size(se)$
asfClose(imIn,imOut[,se]**)**	Alternate Close/Open with sizes from 1 to $size(se)$
buildOpen(imIn,imOut,se**)**	Erosion followed by reconstruction by dilation
buildClose(imIn,imOut,se**)**	Dilation followed by reconstruction by erosion
asBuildOpen(imIn,imOut[,se]**)**	Alternate buildOpen/buildClose with sizes from 1 to $size(se)$
asBuildClose(imIn,imOut[,se]**)**	Alternate buildClose/buildOpen with sizes from 1 to $size(se)$
areaOpen(imIn,size,imOut**)** **areaClose(**imIn,size,imOut**)** **widthOpen(**imIn,size,imOut**)** **widthClose(**imIn,size,imOut**)** **heightOpen(**imIn,size,imOut**)** **heightClose(**imIn,size,imOut**)**	Opening and closing by attributes : area, width and height
ultimateOpen(imIn,imTrans,imIndic,se,stopSize,delta**)**	

Connexity oriented functions - Labelling	
Function	**Description**
minima(imIn,imOut[,se]**)**	Regional minima de imIn
minimaLabeled(imIn,imOut[,se]**)**	... and label it
hMinima(imIn,height,imOut[,se]**)**	Regional minima after hDualBuild
hMinimaLabeled(imIn,height,imOut[,se]**)**	... and label it
maxima(imIn,imOut[,se]**)**	Regional maxima de imIn
maximaLabeled(imIn,imOut[,se]**)**	... and label it
hMaxima(imIn,height,imOut[,se]**)**	Regional maxima after hBuild
hMaximaLabeled(imIn,height,imOut[,se]**)**	... and label it
fastMinima(imIn,imOut[,se]**)**	Regional minima/maxima computation
fastMaxima(imIn,imOut[,se]**)**	based on arrowing graphs
label(imIn,imLabel[,se]**)**	Assign a different label to each connected component
fastLabel(imIn,imLabel[,se]**)**	Fast parallelized
labelWithArea(imIn,imLabel[,se]**)**	Labell an image (with it's already labeled image)
labelWithVolume(imIn,imLabelIn,imLabel[,se]**)**	with some image descriptor
LabelWithMaxima(imIn,imLabelIn,imLabel[,se]**)**	
labelWithMean(imIn,imLabelIn,imLabel[,se]**)**	
lambdaLabel(imIn,lambdaVal,imOut[,se]**)**	Flat zones labelization

Image reconstruction	
Function	**Description**
geoErode(imIn,imMask,imOut[,se]**)**	Geodesic Dilation of imIn over the reference imMask
geoDilate(imIn,imMask,imOut[,se]**)**	Geodesic Dilation of imIn under the reference imMask
build(imMark,imRef,imOut[,se]**)**	Reconstruction by dilation of imMark under the reference imRef
dualBuild(imMark,imRef,imOut[,se]**)**	Reconstruction by erosion of imMark over the reference imRef
hBuild(imMark,h,imOut[,se]**)**	Reconstruction by dilation of $(f - h)$ under the reference f
hDualBuild(imMark,h,imOut[,se]**)**	Reconstruction by erosion of $(f + h)$ over the reference f

Morphological Residues	
Function	**Description**
gradient(imIn,imOut[,se]**)** **gradient(**imIn,imOut,dilSe,eroSe]**)**	Morphological gradient ($dilation - erosion$)
topHat(imIn,imOut[,se]**)** **dualTopHat(**imIn,imOut[,se]**)**	Top Hat (*Open TopHat or White TopHat*) : $im - \gamma(im)$ Dual Top Hat (*Close TopHat or Black TopHat*) : $\phi(im) - im$

Segmentation	
Function	**Description**
gradient(imIn,imOut[,se]**)** *	Compute image gradient
watershed(imGrad,imWs[,se]**)**	Watershed of imGrad into imWs
watershed(imGrad,imMark,imWs[,se]**)**	Watershed of imGrad into imWs. imMark shall be labeled
watershed(imGrad,imMark,imWs,imBasins[,se]**)**	Watershed of imGrad into imWs, as above. imBasins is generated with a labeled mosaic without the watershed line.
basins(imGrad,imBasins[,se]**)**	Basins (labelled mosaic without watershed line) of imGrad
basins(imGrad,imMark,imBasins[,se]**)**	Basins (labelled mosaic without watershed line) of imGrad from markers
waterfall(imGrad,level,imWf[,se]**)** *	level waterfall iterations of imGrad into imWf
stochasticWatershed(imMark,imGrad,imWs,nSeed,se**)**	Stochastic Watershed (see doc)

Hit-or-Miss Morphological Transforms	
Function	**Description**
hitOrMiss(imIn,foreSE,backSE,imOut,borderVal**)** **hitOrMiss(**imIn,compSE,imOut,borderVal**)** **hitOrMiss(**imIn,compSEList,imOut,borderVal**)**	
thin(imIn,foreSE,backSE,imOut**)** **thin(**imIn,compSE,imOut**)** **thin(**imIn,compSEList,imOut**)** **fullThin(**imIn,foreSE,backSE,imOut**)** **fullThin(**imIn,compSE,imOut**)** **fullThin(**imIn,compSEList,imOut**)**	
thick(imIn,foreSE,backSE,imOut**)** **thick(**imIn,compSE,imOut**)** **thick(**imIn,compSEList,imOut**)** **fullThick(**imIn,foreSE,backSE,imOut**)** **fullThick(**imIn,compSE,imOut**)** **fullThick(**imIn,compSEList,imOut**)**	
skiz(imIn,imOut**)** **pruneSkiz(**imIn,imOut[,se]**)** **skeleton(**imIn,imOut[,se]**)** **extinctionValues(**imIn,imOut[,se]**)**	
zhangSkeleton()	

Line based Morphology	
Function	**Description**
lineDilate(imIn,angle,halfLength,imOut) **lineErode**(imIn,angle,halfLength,imOut) **lineOpen**(imIn,angle,halfLength,imOut) **lineClose**(imIn,angle,halfLength,imOut)	Base morphological operators using a segment as SE
squareDilate(imIn,halfSide,imOut) **squareErode**(imIn,halfSide,imOut) **squareOpen**(imIn,halfSide,imOut) **squareClose**(imIn,halfSide,imOut)	Base morphological operators using segments as SE One Horizontal followed by a Vertical one
circleDilate(imIn,radius,imOut) **circleErode**(imIn,radius,imOut) **circleOpen**(imIn,radius,imOut) **circleClose**(imIn,radius,imOut)	Base morphological operators using segments as SE Rotation of a segment

Measures	
Function	**Description**
flag = **isBinary**(im)	Two levels image
area = **area**(im)	Area of the image
volume = **volume**(im)	Volume of the image
min = **minVal**(im) max = **maxVal**(im) mean,stddev = **meanVal**(im) mode = **modeVal**(im) median = **medianVal**(im) range = **rangeVal**(im) values = **valueList**(im)	Statistical descriptors
values = **measBarycenter**(imIn)	Barycenter
values = **measMoments**(imIn,onlyNonZero,centered)	First and second order moments
mat = **measCovariance**(im1,im2,dx,dy,dz,maxSteps,centered) mat = **measAutoCovariance**(imIn,dx,dy,dz,maxSteps,centered)	Covariance between two images Auto covariance
values = **measEntropy**(imIn) values = **measEntropy**(imIn,imMask)	Entropy of an image
values = **measBoundBox**(imIn)	Bounding Box
values = **measGranulometry**(imIn,se,stepSize,CDF,maxSeSize)	Granulometry

Blobs	
Function	**Description**
blobs = **createBlobs**(imLabel,onlyNonZero)	Create a map of blobs from a labeled image
areas = **blobsArea**(imLabel,onlyNonZero) areas = **blobsArea**(blobs)	Create blobs and return the area for each one Gets the area of each blob
volumes = **blobsVolume**(imIn,blobs)	Get the volume of each blob
mins = **blobsMinVal**(imIn,blobs) max = **blobsMaxVal**(imIn,blobs) mean,stddev = **blobsMeanVal**(imIn,blobs) modes = **blobsModeVal**(imIn,blobs) medians = **blobsMedianVal**(imIn,blobs) ranges = **blobsRangeVal**(imIn,blobs) values = **blobsValueList**(imIn,blobs)	Get a map of statistical descriptors
values = **blobsBarycenter**(imIn,blobs) values = **blobsBarycenter**(imLabel,onlyNonZero)	Get the barycenter of each blob Create blobs and return the barycenters
values = **blobsMoments**(imIn,blobs,central) values = **blobsMoments**(imLabel,onlyNonZero,central)	Moments till 2nd order for each blob Create blobs and return moments
mat = **blobsInertiaMatrix**(imIn,blobs,central) mat = **blobsInertiaMatrix**(imLbl,onlyNonZero,central)	Create Inertia Matrix for each blob Create blobs and return Inertia Matrices
values = **blobsEntropy**(imIn,blobs)	Calculate the Entropy of each blob
values = **blobsBoundBox**(imIn,blobs) values = **blobsBoundBox**(imLabel,onlyNonZero)	Bounding Box of each blob
areaThreshold(imIn,threshold,gt,imOut)	Filters regions based on their area

Distance	
Function	**Description**
distance(imIn,imOut[,se]**)**	Morphological Distance
distanceGeodesic(imIn,imMask,imOut[,se]**)**	Geodesic Distance
distanceEuclidean(imIn,imOut**)**	Euclidean Distance

Pixel-based arithmetic and logic functions	
Function	**Description**
inv(imIn,imOut**)**	Invert the image
add(imIn,imageOrValue,imOut**)** **addNoSat(**imIn,imageOrValue,imOut**)**	Add an image to an image or value to *imIn* with or without bounds value check
sub(imIn,imageOrValue,imOut**)** **subNoSat(**imIn,imageOrValue,imOut**)**	Subtract an image from an image or value from *imIn* with or without bounds value check
mul(imIn,imageOrValue,imOut**)** **mulNoSat(**imIn,imageOrValue,imOut**)**	Multiply an image by an image or value from *imIn* with or without bounds value check
div(imIn,imageOrValue,imOut**)**	Divide an image by an image or value from *imIn*
grt(imIn,imageOrValue,imOut**)** **grtOrEqu(**imIn,imageOrValue,imOut**)** **equ(**imIn,imageOrValue,imOut**)** **lowOrEqu(**imIn,imageOrValue,imOut**)** **low(**imIn,imageOrValue,imOut**)**	Arithmetic comparison between an image and an image (or value). Pixels in the output image are set to max(T) if result is true and 0 otherwise
diff(imIn,imageOrValue,imOut**)**	Same as **equ**()
absDiff(imIn,imageOrValue,imOut**)**	Output image with the absolute difference an image and an image or value
logicAnd(im1,im2,imOut**)** **logicOr(**im1,im2,imOut**)** **logicXOr(**im1,im2,imOut**)**	Logic comparison between between pixels of two images. Pixels in the output image are set to **1** if result is **true** and **0** otherwise.
bitAnd(im1,im2,imOut**)** **bitOr(**im1,im2,imOut**)** **bitXOr(**im1,im2,imOut**)**	Same as above but comparison is done **bitwise**.
sup(im1,im2,imOut**)** **inf(**im1,im2,imOut**)**	Compute the sup of two images Compute the inf of two images
log(imIn,imOut,base**)** **exp(**imIn,imOut,base**)**	Transform range of values of input image to a logarithmic scale. Revert a **log** transform.
mask(imIn,imMask,imOut**)**	Apply a mask on an input image
applyLookup(imIn,lutMap,imOut**)**	Transform pixel values based on a lookup table (map)

Filters (non morphological)	
Function	**Description**
gaussianFilter(imIn,radius,imOut**)**	Gaussian filter
horizConvolve(imIn,kernel,imOut**)** **vertConvolve(**imIn,kernel,imOut**)** **convolve(**imIn,kernel,imOut**)**	Convolution against an horizontal kernel ... a vertical one Both. **2D** filters.
recursiveBilateralFilter(imIn,sigmaW,sigmaW,imOut**)**	Fast bilateral filter
cannyEdgeDetection(imIn,sigma,imOut**)**	**2D** filter
dericheEdgeDetection(imIn,alpha,imOut**)**	**2D** filter
kuwaharaFilter(imIn,radius,imOut**)**	**2D** filter
meanShiftFilter(imIn,radius,imOut**)**	**2D** filter
sigmaFilter(imIn,radius,sigma,pctNbMinPixel,excOutlier,imOut**)**	**2D** filter
gaborFilter(imIn,sigma,theta,lambda,psi,gamma,imOut**)**	**2D** filter

Transformations	
Function	**Description**
vertFlip(imIn,imOut**)** **horizFlip(**imIn,imOut**)**	Vertical and Horizontal mirror
rotateX90(imIn,count,imOut**)**	Image rotate by a multiple of 90 degrees
translate(imIn,dx,dy,dz,imOut,border**)** *	Image translation by **dx, dy, dz** offset
resize(imIn,sx,sy,sz,imOut,method**)** * **scale(**imIn,kx,ky,kz,imOut,method**)** *	Change (or multiply) image size. method can be **auto**, **bilinear** (2D), **trilinear** (3D) or **closest** (B/W).
copy(imIn,start(X,Y,Z),sz(X,Y,Z),imOut,oStart(X,Y,Z)**)** * **clone(**imIn,imOut**)**	Copy imIn into imOut Copy imIn into imOut
crop(imIn,start(X,Y,Z),sz(X,Y,Z),imOut**)** *	Copy imIn into imOut
im << **value** **fill(**im,value**)** **randFill(**im**)**	Set all pixels in im.
cast(imIn,imOut**)** **rangeScale(**imIn,iMin,iMax,oMin,oMax,imOut**)** **rangeScale(**imIn,oMin,oMax,imOut,onlyNonZero**)** **rangeScale(**imIn,imOut,onlyNonZero**)** **sCurve(**imIn,pivot,ratio,imOut**)**	Copy imIn to imOut expanding range of values. Expand range of values of imIn to imOut (See documentation for details) Modify contrast with help of a logistic function
compare(imIn,condition,a,tVal,fVal,imOut**)** * **test(**imIn,imTrue,imFalse,imOut**)** *	use condition to compare imIn with image or values check for each pixel if it's nonZero

Histogram	
Function	**Description**
hMap = **histogram(**imIn[,imMask][,fullRange]**)**	Image histogram
hMap = **histogramMap(**imIn[,binSize]**)** hMap = **histogramMap(**imIn,imMask[,binSize]**)**	Histogram compressed in *bins* (option) (returned map doesn't contains empty slots)
stretchHistogram(imIn,iMin,iMax,imOut,oMin,oMax**)** **stretchHistogram(**imIn,imOut,oMin,oMax**)** **stretchHistogram(**imIn,imOut**)**	Output image is a histogram based linear transform of input image. (See documentation for details)
threshold(imIn,iMin,iMax,tVal,fVal,imOut**)** **threshold(**imIn,iMin,iMax,imOut**)** **threshold(**imIn,iMin,imOut**)**	Threshold
threshold(imIn,imOut**)** **otsuThreshold(**imIn,imOut,nTresholds**)** * **otsuThreshold(**imIn,imMask,imOut[,nThresholds]**)** **otsuThresholdValues(**imIn,nThresholds**)** **otsuThresholdValues(**hist,nThresholds**)**	Otsu's threshold with two classes Multi-Otsu
enhanceContrast(imIn,imOut,saturation**)**	Enhance image contrast

Matrix transforms	
Function	**Description**
matTranspose(imIn, imOut[,order="yxz]**)**	Matrix traspose (**3D**)
matMultiply(im1,im2,imOut**)**	Matrix multiplication (**2D**)

Color	
Function	**Description**
`im1,im2,im3 = extractChannels(colorIm)`	Extract channels from a color image
`colorOut = combineChannels(im1,im2,im3)`	Combine channels into a color image
`im8 = Image(colorImage,'UINT8')` `RGBToLuminance(colorim,im8)`	Convert color image to luminance

`RGBToXYZ(imIn,imOut)` `XYZToRGB(imIn,imOut)`
`RGBToLAB(imIn,imOut)` `LABToRGB(imIn,imOut)`
`RGBToHLS(imIn,imOut)` `HLSToRGB(imIn,imOut)`
	Color space conversions

`gradientLAB(imRGB,im8,se)`
`im8 = gradientLAB(imRGB,se)`
`gradientHLS(imRGB,im8,se)`
`im8 = gradientHLS(imRGB,se)`
	Color gradient in HLS and LAB space

Operations on Multichannel images (Color)	
Function	**Description**
`splitChannels(imMCTIn,im3DOut)`	Split channels of a **RGB** image into slices of a **3D** image
`mergeChannels(im3DIn,imMCTOut)`	Create a **RGB** image from slices of a **3D** image
`copyChannel(imMCTIn, chanNum, imOut)`	Copy a channel of a **RGB** image into a **2D** image
`copyToChannel(imIn, chanNum, imMCTOut)`	Copy a **2D** image into a channel of a **RGB** image

Drawing	
Function	**Description**
`drawLine(im,x0,y0,x1,y1,value)`	
`drawRectangle(im,x0,y0,w,h,value,fill,zSlice)` `drawRectangle(im,coords,value,fill)`	
`drawCircle(im,x0,y0,radius,value)`	
`drawSphere(im,x0,z0,radius,value)`	
`drawBox(im,x0,y0,z0,w,h,d,value,fill)`	
`drawBlobs(blobs,imOut,blobsValue,fillFirst,defaultValue)`	

Advanced geodesic functions	
Function	**Description**

`geodesicMeasure(imIn,imOut,method,sliceBySlice,dzOverDx)`
`geodesicDiameter(imIn,imOut,sliceBySlice,dzOverDx)`
`geodesicElongation(imIn,imOut,sliceBySlice,dzOverDx)`
`geodesicTortuosity(imIn,imOut,sliceBySlice,dzOverDx)`
`geodesicExtremities(imIn,imOut,sliceBySlice,dzOverDx)`	

`labelFlatZones(imIn,method,imOut)`	
`geodesicPathOpening(imIn,method,lenght,imOut,sx,sy,sz)` `geodesicPathClosing(imIn,method,lenght,imOut,sx,sy,sz)`	
`geodesicUltimatePathOpening(In,T,Ind,sx,sy,sz,stop,lAtt,tMin)` `geodesicUltimatePathClosing(In,T,Ind,sx,sy,sz,stop,lAtt,tMin)`	

Evaluating image similarity	
Function	**Description**
indexJaccard(imGt,imIn**)**	Jaccard index between two images
indexRuzicka(imGt,imIn**)**	
distanceHamming(imGt,imIn**)**	
distanceHausdorff(imGt,imIn**)**	
indexAccuracy(imGt,imIn,threshold=0**)**	
indexPrecision(imGt,imIn**)**	
indexRecall(imGt,imIn**)**	
indexFScore(imGt,imIn,beta=1.**)**	
indexSensitivity(imGt,imIn**)**	
indexSpecificity(imGt,imIn**)**	
indexFallOut(imGt,imIn**)**	
indexMissRate(imGt,imIn**)**	
indexOverlap(imGt,imIn**)**	

3 Table of contents