SMIL survival kit

November 23, 2015

1 Introduction

This document contains a short list of functions with a quick description. The complete documentation can be found at *http://smil.cmm.mines-paristech.fr/doc/modules.html.*

Note: optional parameters are between square brackets [*optional*].

2 import SMIL library

In order to import SMIL library type the following command:
from **smilPython** *import **
or
import **smilPython** *as* **sm**
If the second version is used, the suffix **sm** will be necessary when calling SMIL functions.
In order to run an existing script type:
execfile("myscript.py")

3 I/O images

- To read an image:
  im = **Image**("images/toto.png")
- To write an image:
  write(im,"filename.png")
- To create a new image based on the dimensions of a given image (but not to copy the contents):
  im2 = Image(im)
- To create a new image based on the dimensions of a given image but with a different depth:
  im2 = Image(im,"UINT16")
- To display an image:
  im.**show**()
- To display an image in false colors:
  im.**showLabel**()

4 Pixel-based functions

- To copy an image into another one:

  copy(im,im2)

- To set the image to zero:

  im << 0

- To invert the image

  inv(imin,imout)

- To add a constant (or an image):

  add(imin,constant_or_image,imout)

- To substract a constant (or an image):

  sub(imin,constant_or_image,imout)

- Returns the maximum value of im:

  maxVal(im)

- Returns the maximum value of im:

  minVal(im)

- To compare an image to a constant or another image:

  compare(imin,condition,a,b,c, imout)

  a, b and *c* can be images or scalars. imin is compared to *a* according to the given condition. If result is true, parameter *b* is set in *imout*. Otherwise the corresponding imout pixel is set to *c*.

  For example

  compare(im1,">", im2, im1,im2,imomut)

  is equivalent to sup(im1,im2,imout)

- To compute the sup of two images:

  sup(im1,im2,imout)

- To compute the inf of two images:

  inf(im1,im2,imout)

- Threshold:

  threshold(im,minval,maxval,trueval,falseval,imout)

- Otsu threshold :

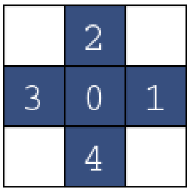
  threshold(im,imout)

- Scale : if im has the size (W,H), the size of imout will be (W*factor_x, H* factor_y). imout should be allocated first (imout = Image()).

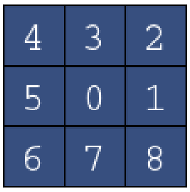
  scale(im,factor_x, factor_y,imout)

5 Structuring elements

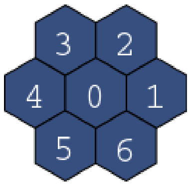
5.1 Already defined structuring elements :



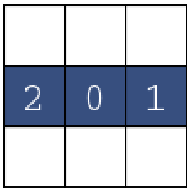
CrossSE()



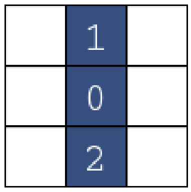
SquSE()



HexSE()



HorizSE()



VertSE()

5.2 Default structuring element

If not specified, default SE is used in most morphological functions. As an optional parameter, SE is usualy the last one.

erode(imin,imout)
erode(imin,imout, CrossSE())
erode(imin,imout,CrossSE(size))

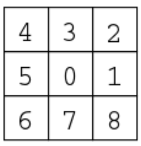
5.3 Modify default SE

Get Default SE: **print Morpho.getDefaultSE()**. If not modified, HexSE() is used by default.

Set default SE : **Morpho.setDefaultSE(CrossSE())**. CrossSE() becomes default SE.

5.4 Define other SEs

Construct a structuring element with points defined by their indexes:
StrElt(HexFlag, PointList) :
For example:
mySE1= StrElt(False,(0,1,5)) is equivalent to HorizSE().



Square

Hexagonal

6 Morphological Erosion, Dilation, Opening, Closing

If not specified, default SE is used in most morphological functions. As an optional parameter, SE is usualy the last one.

- Erosion with default SE:

  erode(im,imout)

- Erosion with other SE:
  erode(im,imout,CrossSE())
- Erosion with an homothetic SE:
  erode(im,imout[, *CrossSE(size)*])
- Dilation (idem for optional SE).
  dilate(im,imout[,*CrossSE(size)*])
- Morphological opening (idem for optional SE).
  open(im,imout[,*CrossSE(size)*])
- Morphological closing (idem for optional SE).
  close(im,imout[,*CrossSE(size)*])

7 Reconstruction

- Reconstruction by dilation
  build(imMark,imRef,imOut[,*nl*])
- Reconstruction by erosion
  dualBuild(imMark,imRef,imOut[,*nl*])
- hBuild: reconstruct by dilation from f-h:
  hBuild(im,h,immout[,*nl*])
- hDualBuild: reconstruct by erosion from f+h:
  hDualBuild(im,h,immout[,*nl*])
- hMaxima: regional maxima after hBuild (reconstruction from f-h),
  hMaxima(im,h,immax[,*nl*])
- hMinima: regional minima after hDualBuild (reconstruction from f+h),
  hMinima(im,h,immin[,*nl*])
- hMaxima: regional maxima after razing function f constrained by f-h,
  hMaxima(im,h,immax[,*nl*])
- hMinima: regional minima after flooding function f constrained by f+h,
  hMinima(im,h,immin[,*nl*])

8 Filtering

Opening and closing are also morphological filter. Other filters are given in this section:

- Alternate filter: an opening of size *size* followed by a closing of the same size.

  AF(im,size,imout[,*nl*])

- Alternate sequential filter: $\phi_{size}\gamma_{size}...\phi_2\gamma_2\phi_1\gamma_1(im)$

  ASF(im,size,imout[,*nl*])

- Opening by reconstruction: erosion of size *size* followed by reconstruction (by dilation):

  buildOpen(imIn, imOut, se(size))

- Closing by reconstruction: dilation of size *size* followed by reconstruction (by erosion):

  buildClose(imIn, imOut, se(size))

- Alternate filter combining buildOpen and buildClose of a give size:

  buildAF(imIn, imOut, se(size))

- Alternate sequential filter combining buildOpen and buildClose of increasing sizes (up to size *size*:

  buildASF(imIn, imOut, se(size))

- AreaOpen:

  areaOpen(imIn, size, imOut)

- Alternate levelings:

  ASF_Leveling(imIn, size, imOut,nl)

9 Connexity oriented functions

- Regional minima of imin :

  minima(imin,imout[, *nl*])

- Regional maxima of imin:

  maxima(imin,imout[, *nl*])

- **label**(im,imlabel[, *nl*]). Assign a different *label* (identifier) to each connected component.

10 Segmentation

- To compute the gradient:

  gradient(im,imout[,*nl*])

- Watershed of *imgra* into *imws*:

  watershed(imgra,imws[,nl])

- Watershed of *imgra* from markers *immark* into *imws*. Note: each marker should be identified with a different *label*.

  watershed(imgra,immark,imws[,nl])

- Watershed of *imgra* from markers *immark*. Two output parameters*imws* with the watershed line and *imbasins* the labelled mosaic without the watershed line:

  watershed(imgra,immark,imws,imbasins[,nl])

- Basins (labelled mosaic without watershed line) of *imgra*:

  basins(imgra,imbasins[,nl])

- Basins (labelled mosaic without watershed line) of *imgra* from markers *immark*:

  basins(imgra,immark,imbasins[,nl])

- Hierarchical segmentation based on extinction values: imgra is flooded and the computed extinction value is assigned to the minima in *imEV* image. *extinction_type* can be "d", "a" or "v" for dynamics, area or volume respectively.

  watershedExtinction(imgra,imEV, extinction_type [,nl])

- A waterfall iteration of *imgra0* from *imws0*:

  waterfall(imgra0,imws0,imgra1,imws1[,nl])

- Iteration of waterfall, *level* times:

  waterfall(imgra,level,imwf[,nl])

- Extinction values based hierarchical segmentation:

  imFineSeg,MST = watershedEV(imgra,EVType[,nl])

- Get a partition imSeg with Nregions from the hierarchy stored in MST graph:

  getEVLevel(imFineSeg, MST, Nregions, imSeg)

11 Color

- Extract channels from a color image:

  im1,im2,im3 = extractChannels(colorim)

- Combine channels into a color image:
  colorout = combineChannels(im1,im2,im3)
- Get luminance:
  im8 = Image(colorim,"UINT8")
  RGBToLuminance(colorim,im8)
- Color conversions:
  colorim2 = Image(colorim)
  - RTBToXYZ(colorim,colorim2)
  - RGBToLAB(colorim,colorim2)
  - RGBToHLS(colorim,colorim2)
- Color gradients:
  - imgraLAB= gradient_LAB(colorim,nl)
  - imgraHLS= gradient_HLS(colorim,nl)