1 Introduction

This document contains a short list of functions with a quick description. The complete documentation can be found at http://smil.cnam.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:

\texttt{from smilPython import *}

or

\texttt{import smilPython as sm}

If the second version is used, the suffix \texttt{sm} will be necessary when calling SMIL functions.

In order to run an existing script type:

\texttt{execfile(“myscript.py”)}

3 I/O images

- To read an image:
  \texttt{im = Image(“images/toto.png”)}

- To write an image:
  \texttt{write(im,”filename.png”)}

- To create a new image based on the dimensions of a given image (but not to copy the contents):
  \texttt{im2 = Image(im)}

- To create a new image based on the dimensions of a given image but with a different depth:
  \texttt{im2 = Image(im,”UINT16”)}

- To display an image:
  \texttt{im.show()}

- To display an image in false colors:
  \texttt{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)`

2
5 Structuring elements

5.1 Already defined structuring elements:

<table>
<thead>
<tr>
<th>CrossSE()</th>
<th>SquSE()</th>
<th>HexSE()</th>
<th>HorizSE()</th>
<th>VertSE()</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 4 3 1</td>
<td>4 3 2  1</td>
<td>3 2</td>
<td>2 0 1</td>
<td>1 0 2</td>
</tr>
<tr>
<td>3 0 1</td>
<td>5 0 1</td>
<td>4 0 1</td>
<td>2 1</td>
<td>2 1</td>
</tr>
<tr>
<td>4</td>
<td>6 7 8</td>
<td>5 6</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

5.2 Default structuring element

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

```python
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:

```python
StrElt(HexFlag, PointList)
```

For example:

```python
mySE1 = StrElt(False, (0, 1, 5))
```

6 Morphological Erosion, Dilation, Opening, Closing

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

- Erosion with default SE:
  ```python
  erode(im, imout)
  ```
• Erosion with other SE:
  \texttt{erode(im,imout,CrossSE())}

• Erosion with an homothetic SE:
  \texttt{erode(im,imout[,CrossSE(size)])}

• Dilation (idem for optional SE).
  \texttt{dilate(im,imout[,CrossSE(size)])}

• Morphological opening (idem for optional SE).
  \texttt{open(im,imout[,CrossSE(size)])}

• Morphological closing (idem for optional SE).
  \texttt{close(im,imout[,CrossSE(size)])}

7 Reconstruction

• Reconstruction by dilation
  \texttt{build(imMark,imRef,imOut[,n])}

• Reconstruction by erosion
  \texttt{dualBuild(imMark,imRef,imOut[,n])}

• hBuild: reconstruct by dilation from f-h:
  \texttt{hBuild(im,h,immout[,n])}

• hDualBuild: reconstruct by erosion from f+h:
  \texttt{hDualBuild(im,h,immout[,n])}

• hMaxima: regional maxima after hBuild (reconstruction from f-h),
  \texttt{hMaxima(im,h,immax[,n])}

• hMinima: regional minima after hDualBuild (reconstruction from f+h),
  \texttt{hMinima(im,h,immin[,n])}

• hMaxima: regional maxima after razing function f constrained by f-h,
  \texttt{hMaxima(im,h,immax[,n])}

• hMinima: regional minima after flooding function f constrained by f+h,
  \texttt{hMinima(im,h,immin[,n])}
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.
  $\text{AF}(im, size, imout[, nl])$

- Alternate sequential filter: $\phi_{size} \gamma_{size} ... \phi_{2} \gamma_{2} \phi_{1} \gamma_{1}(im)$
  $\text{ASF}(im, size, imout[, nl])$

- Opening by reconstruction: erosion of size $size$ followed by reconstruction (by dilation):
  $\text{buildOpen}(imIn, imOut, se(size))$

- Closing by reconstruction: dilation of size $size$ followed by reconstruction (by erosion):
  $\text{buildClose}(imIn, imOut, se(size))$

- Alternate filter combining $\text{buildOpen}$ and $\text{buildClose}$ of a give size:
  $\text{buildAF}(imIn, imOut, se(size))$

- Alternate sequential filter combining $\text{buildOpen}$ and $\text{buildClose}$ of increasing sizes (up to size $size$):
  $\text{buildASF}(imIn, imOut, se(size))$

- AreaOpen:
  $\text{areaOpen}(imIn, size, imOut)$

- Alternate levelings:
  $\text{ASF\_Leveling}(imIn, size, imOut, nl)$

9 Connexity oriented functions

- Regional minima of imin:
  $\text{minima}(imin, imout[, nl])$

- Regional maxima of imin:
  $\text{maxima}(imin, imout[, nl])$

- $\text{label}(im, imlabel[, nl])$. Assign a different label (identifier) to each connected component.
10 Segmentation

- To compute the gradient:
  \texttt{gradient\text{(}\texttt{im,imout[,]nl}\text{)}}

- Watershed of \textit{imgra} into \textit{imws}:
  \texttt{watershed\text{(}\texttt{imgra,imws[,]nl}\text{)}}

- Watershed of \textit{imgra} from markers \textit{immark} into \textit{imws}. Note: each marker should be identified with a different label.
  \texttt{watershed\text{(}\texttt{imgra,immark,imws[,]nl}\text{)}}

- Watershed of \textit{imgra} from markers \textit{immark}. Two output parameters \textit{imws} with the watershed line and \textit{imbasins} the labelled mosaic without the watershed line:
  \texttt{watershed\text{(}\texttt{imgra,immark,imws,imbasins[,]nl}\text{)}}

- Basins (labelled mosaic without watershed line) of \textit{imgra}:
  \texttt{basins\text{(}\texttt{imgra,imbasins[,]nl}\text{)}}

- Basins (labelled mosaic without watershed line) of \textit{imgra} from markers \textit{immark}:
  \texttt{basins\text{(}\texttt{imgra,immark,imbasins[,]nl}\text{)}}

- Hierarchical segmentation based on extinction values: \textit{imgra} is flooded and the computed extinction value is assigned to the minima in \textit{imEV} image. \texttt{extinction\_type} can be “d”, ”a” or ”v” for dynamics, area or volume respectively.
  \texttt{watershedExtinction\text{(}\texttt{imgra,imEV, extinction\_type [,nl]}\text{)}}

- A waterfall iteration of \textit{imgra0} from \textit{imws0}:
  \texttt{waterfall\text{(}\texttt{imgra0,imws0,imgra1,imws1[,]nl}\text{)}}

- Iteration of waterfall, level times:
  \texttt{waterfall\text{(}\texttt{imgra,level,imwf[,]nl}\text{)}}

- Extinction values based hierarchical segmentation:
  \texttt{imFineSeg,MST = watershedEV\text{(}\texttt{imgra,EVType[,]nl}\text{)}}

- Get a partition \textit{imSeg} with Nregions from the hierarchy stored in MST graph:
  \texttt{getEVLevel\text{(}\texttt{imFineSeg, MST, Nregions, imSeg}\text{)}}

11 Color

- Extract channels from a color image:
  \texttt{im1,im2,im3 = extractChannels\text{(}colorim\text{)}}
• Combine channels into a color image:
  \[
  \text{colorout} = \text{combineChannels}(\text{im1}, \text{im2}, \text{im3})
  \]

• Get luminance:
  \[
  \text{im8} = \text{Image}(\text{colorim}, \text{"UINT8"})
  \]
  \[
  \text{RGBToLuminance}(\text{colorim}, \text{im8})
  \]

• Color conversions:
  \[
  \text{colorim2} = \text{Image}(\text{colorim})
  \]
  \[
  \begin{align*}
  &- \text{RTBToXYZ}(\text{colorim}, \text{colorim2}) \\
  &- \text{RGBToLAB}(\text{colorim}, \text{colorim2}) \\
  &- \text{RGBToHLS}(\text{colorim}, \text{colorim2})
  \end{align*}
  \]

• Color gradients:
  \[
  \begin{align*}
  &- \text{imgraLAB} = \text{gradient}_\text{LAB}(\text{colorim}, \text{nl}) \\
  &- \text{imgraHLS} = \text{gradient}_\text{HLS}(\text{colorim}, \text{nl})
  \end{align*}
  \]