SMIL in a nutshell

Centre de Morphologie Mathématique
MINES ParisTech

September, 4th 2014

Simple* Morphological Image Library (* but efficient)

Introduction

- SMIL is an image processing library, specialized for mathematical morphology, developed at the CMM since 2012 by Matthieu Faessel.
- It aims to be lightweight, fast, easy to use and to extend.
- It uses swig and can be wrapped in several languages: for now, python, java, ruby and octave are available.

Two layers:

- a C++ core
- a Python interface

Outline

Installation

Installation

Download installer (linux or windows) from
http://smil.cmm.mines-paristech.fr/doc/download_page.html

Start SMIL

Write your python script and launch it with your favorite python interpreter (Idle, ipython...). You can also run SMIL without previous installation, using a java interpreter
http://smil.cmm.mines-paristech.fr/doc/webstart_page.html

Outline

About Python

Python

- high-level language, inspired by Lisp
- object-oriented with dynamic typing
- part of the “agile” language family
- easy to interface with C++
- easy to learn (?)

Expressions

Examples of expressions

A session in the interactive interpreter:

```
>>> 2
2
>>> 2+2
4
>>> "hello"
'hello'
>>> # this is a comment
```

Variables

Using variables

Variables are not declared and their content is copied:

```
>>> a = 2
>>> a
2
>>> b = 3
>>> print b
3
>>> b = a
>>> a = 1
>>> print a,b
1 2
```

Compound types: tuples and lists

Tuples

```
>>> t1 = ( 1, "zz")
>>> type(t1)
<type 'tuple'>
>>> print t1, t1[0], t1[1]
(1,"zz") 1 "zz"
>>> a,b = t1
>>> print a
1
>>> print b
'zz'
```

Compound types: tuples and lists

Lists

```
>>> l1=[ "abc" ]
>>> l1.append(3.14159) # lists can be heterogeneous
>>> l1 [ 'abc', 3.14159 ]
>>> l1.append( [ 1, t1 ] ) # lists can contain lists
>>> l1 ['abc', 3.14159, [1, (1, 2)]]
>>> print l1[1] # indexing starts at 0 3.14159
```

Functions

Declaring a function

In the interpreter:

```
>>> def f(a): # note the ':' (colon)
... return a*2 # there are 4 spaces or 1 tab
# before the "return"
...
>>> f("abcd")
'abcdabcd'
```

Indentation is *critical*: it's part of the syntax

Tests

Tests and conditions

```
if 1 == 2 : # note the ':' !
     print "Wow, Problem!" # <- 4 spaces or 1 tab

if( myGlass.isFull() ): # ':' and braces ()
     myGlass.doEmpty()
elif myGlass.isEmpty(): # or just ':' and no braces
     myGlass.doFill()
else:
     pass # "empty" action
```

Loops (1)

Simple loops

```
for i in range( 5 ):
print i
```

Will display:

```
0
1
2
3
4
```

Loops (2)

Loops on lists

To loop over the items in a list, 2 possibilities:

```
L = [ 1 , 2 , "abc" ]
for i in range( len(L) ):
    print L[i]
```

which is equivalent to:

```
for x in L:
    print x
```

Loops (3)

"while" loops

Infinite "while" loop:

```
a = 0
while True:
     a += 1
     if a == 10:
          break # "continue" also exists
```

"while" loop with a test:

```
while a < 10:
     a+=1
```

Packages

Importing packages

```
>>> import math
>>> math.cos(0)
1.0
```

For the lazy ones:

```
>>> from math import *
>>> cos(0)
1.0
```

Files

Reading and writing (text mode)

To read the content of a text file:

```
>>> f = open( "hop.txt" , "r" )
>>> for l in f.readlines():
        print l
```

To write:

```
>>> f2 = open( "hop2.txt" , "w" ) # "a" for "append"
>>> f2.write( "Hello, world!" + str( 3.14 ) + "\n" )
>>> f2.close() # optional
```

Paths

Manipulating paths

In the **os** package:

```
>>> import os
>>> os.getcwd() # current directory
'/home/romain'
>>> os.path.join( os.getcwd() , "hop" , "hip" )
'/home/romain/hop/hip'
```

Paths (2)

Packages import paths

In the file `/home/matthieu/Python/myOwnProgram.py`

```
def hello():      return "bonjour"
```

In the main program, using the **sys** package:

```
>>> import sys
>>> sys.path.append("/home/matthieu/Python")
>>> import myOwnProgram
>>> print myOwnProgram.hello()
'bonjour'
```

Outline

Input/Output

Reading and writing images

Reading:

```
>>> im1 = Image( "toto.png" )
```

Writing:

```
>>> write(im1, "/tmp/toto2.png")
```

PNG is the best supported format.

Input/Output

Copying images

```
>>> im2 = Image(im1)
>>> copy(im1, im2)
```

Changing type (8/16/32 bits, floats, ...):

```
>>> im2 = Image(im1, "UINT16" )
>>> copy(im1,im2)
```

IMPORTANT: `ImCopy` copies in *existing* images !

Image display

internal module

im.show()
im3d.show() : slice by slice for 3D images
im.showLabel(): display in false colors.

external module: nxv

```
>>> ImDisplay(im)
>>> ImDisplay(im1, im2, im3)
```

External module (nxv) is called. 16 bits images are copied to 8 bits images before displaying. Several images can be displayed in the same window. Stwich between them with page up/down (zoom is preserved for all images of the same window).

Direct access to the pixels

2d image

```
>>> print im.getPixel(x,y)
235
>>> im.setPixel(x,y, value)
```

3d image

```
>>> print im.getPixel(x,y,z)
235
>>> im.setPixel(x, y, z, value)
```

Example

Displaying erosions of increasing size

```
imIn = Image( images_dir + "Gray/foreman.png" )
imOut = Image( imIn )
imOut.show()
for i in range( 5 ):
    erode( imIn , imOut, CrossSE(i+1))
```

Input and Output images can be the same.

Structuring elements

Default SE

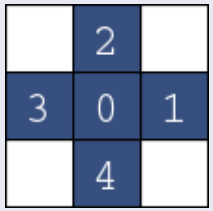
If not specified, default SE is used in most morphological functions. As an optional parameter, SE is usualy the last one.
erode(imin,imout)
erode(imin,imout, CrossSE())
erode(imin,imout,CrossSE(size))

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.

Structuring elements

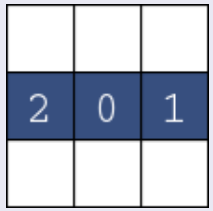
Predefined structuring elements



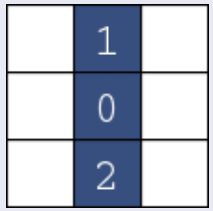
CrossSE()

SquSE()

HexSE()



HorizSE()

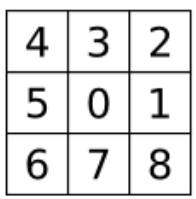


VertSE()

Define other SEs

StrElt(HexFlag, PointList)

Construct a structuring element with points defined by their indexes: mySE1= StrElt(False,(0,1,5)) -> HorizSE()



Square

Hexagonal

Outline

Conventions

Function parameter order

Usually, we specify first the input images, then the parameters and finally the output images:

```
inv( imIn, imOut)
add( imIn , 3 , imOut)
```

But there are exceptions ! :)
For example SE is usually the last parameter, when specified.

First aid...

Help !

Python has a useful "help" command:

```
>>> help(abs)
abs(...)
    abs(number) -> number
    Return the absolute value of the argument.
```

Also for functions:

```
>>> help(dilate)
Help on function dilate in module
smilPython.smilBasePython:
```

The end

Good Luck !