

SMIL in a nutshell

Centre de Morphologie Mathématique
MINES ParisTech

November, 25th 2019

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

- 1 Installation
- 2 Python in 5 minutes
- 3 Using SMIL
- 4 Conclusion

Installation

Installation

Download source code from:

“git clone

<https://github.com/MinesParis-MorphoMath/smil>”

(available under a BSD-style open source license)

Requirements

- cmake
- swig
- python 2.7 or greater
- Qt 4 or greater (option)
- Image libraries: libpng, libtiff, libjpeg

Jupyter notebook server

Server connection

- open a web browser (firefox, chrome...)
- URL: <http://notebooks.cmm.mines-paristech.fr>
- Use your username and password to log-in.

Server connection

Login: Use your username and password

Fichier Édition Affichage Historique Marque-pages Outils ?

JupyterHub

notebooks.cmm.mines-paristech.fr/hub/login

jupyter

Sign in

Warning: JupyterHub seems to be served over an unsecured HTTP connection. We strongly recommend enabling HTTPS for JupyterHub.

Username:

morpho-mat-XX

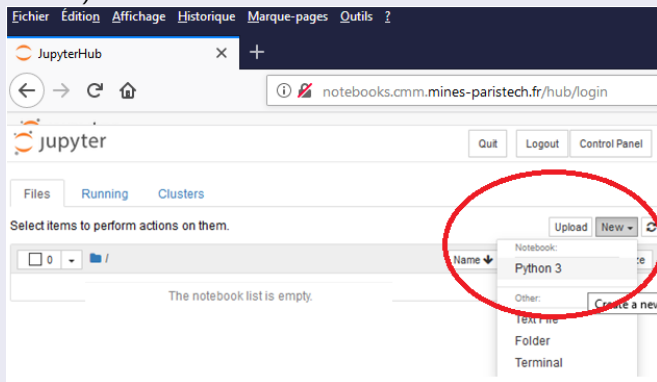
Cette connexion n'est pas sécurisée. Les identifiants saisis ici pourraient être compromis. En savoir plus

Sign In

Create new notebook

New notebook

Click on “New” button, select “Python 3” (on the right top corner).

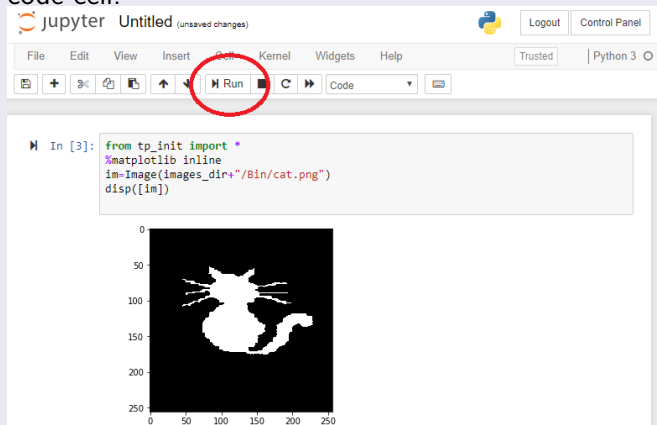


The screenshot shows the JupyterHub web interface. At the top, there is a navigation bar with buttons for 'Quit', 'Logout', and 'Control Panel'. Below this, there are tabs for 'Files', 'Running', and 'Clusters'. The main content area displays 'Select items to perform actions on them.' and a file browser showing an empty directory. A red circle highlights the 'New' button in the top right corner of the file browser, which has opened a dropdown menu. The menu lists 'Notebook: Python 3' as the selected option, along with other options like 'Text File', 'Folder', and 'Terminal'.

Run a script

Write your own script (or open an existing one) and execute it.

Shift-Enter or “Run” button executes the code of the selected code-cell.



The screenshot shows the Jupyter Notebook interface. The top bar includes the Jupyter logo, the text "jupyter Untitled (unsaved changes)", a Python logo, and buttons for "Logout" and "Control Panel". The menu bar contains "File", "Edit", "View", "Insert", "Cell", "Kernel", "Widgets", and "Help". The "Cell" menu is open, and the "Run" button is circled in red. The code cell contains the following Python code:

```
In [3]: from tp_init import *
%matplotlib inline
im=Image(images_dir+"/Bin/cat.png")
disp([im])
```

Below the code cell, a plot displays a white silhouette of a cat on a black background. The plot has x and y axes ranging from 0 to 250.

Outline

- 1 Installation
- 2 Python in 5 minutes
 - Expressions, variables and types
 - Functions
 - Loops and tests
 - Packages
 - Files and paths
- 3 Using SMIL
- 4 Conclusion

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

- 1 Installation
- 2 Python in 5 minutes
- 3 Using SMIL**
 - Input/Output, copying, ...
 - Direct access to the image
 - Example
- 4 Conclusion

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

You have to define the matplotlib backend: `inline` (just display) or `notebook` (interactive mode)

`%matplotlib inline`

`disp(im)`: displays a single image.

`disp(im, True)`: displays a single image in false colors.

`disp([im1,im2])`: displays a list of images.

`disp([im1,im2], [True,False])`: displays a list of images, the first in false colors (labels)

`%matplotlib notebook`

Interactive mode, but only a single image can be shown in this mode.

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 )
disp(imOut)
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 usually the last one.

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

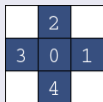
Modify default SE

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

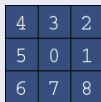
Set default SE : `Morpho.setDefaultSE(CrossSE())`. `CrossSE()` becomes default SE.

Structuring elements

Predefined structuring elements



CrossSE()



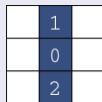
SquSE()



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

- 1 Installation
- 2 Python in 5 minutes
- 3 Using SMIL
- 4 Conclusion

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 !