



Mamba Image Library Coding Rules and Standards

Author:
Nicolas BEUCHER

August 23, 2009

Contents

1	Introduction	4
2	Policy	4
2.1	A Mathematical Morphology library	4
2.2	Simple yet Fast	4
2.3	Portable	4
2.4	... and Free	4
3	Programming	5
3.1	Languages	5
3.2	Rules for C	5
3.3	Rules for Python	5
4	Documentation	6
4.1	In code	6
4.2	Other documents	6
5	Other contributions	6
6	Licensing	6

List of Figures

standards, n.:

The principles we use to reject other people's code.

1 Introduction

Mamba is an open-source library and any contribution is welcome. In order to help programmers, users and would be contributors this document presents the basic policy ruling the development of Mamba. It is our vision, as its creator, of what is Mamba and where we want it to go in the future. Anyone who wish to participate (by giving ideas, suggestions or code) should read at least the policy section [2](#).

This document also provides a set of rules, guidelines and general standards to use when developing Mamba. Its intended audience is programmers contributing to the Mamba library. These rules are not intended to be followed blindly but are meant to make it easier for the various programmers to understand the code they are not directly responsible for but still need to interact with. As a programmer for Mamba you should take care to fulfil these rules and ultimately, if you have a good reason for breaking them, to correctly explain and document your reason.

Documentation contribution and licensing aspect are also covered by this document.

2 Policy

As you may already know, Mamba objective is to be a fast, simple, portable and free (as in free speech but also as in free beer, this being the case only because no one on earth could afford it if we had decided to sell it) mathematical morphology library. To cover this purpose, a basic policy was decided by its creators.

2.1 A Mathematical Morphology library

Mamba is a mathematical morphology library only. It basically means that there is no convolution, fast fourier transform and such in it. Only algorithms related to mathematical morphology may be present in it.

This policy is one of the founding aspect of Mamba. The objective is to prevent Mamba from dispersing itself into aspects that are not related to its original purpose. We believe that there is already enough good libraries out there to compute images using other techniques than mathematical morphology and thus that there is no need for Mamba to include them.

2.2 Simple yet Fast

Fast and simple can appear somehow contradictory as complex algorithms may deliver faster performances. However, what we mean here is to make sure Mamba is simple to use. As such, we believe that it is important not to have to wait a very long time to get the result of your algorithm. Mamba is meant to be used in research and education where you are not always sure of your idea. Being able to rapidly evaluate the soundness of an algorithm is also what will make Mamba simple to use. In other words, the faster Mamba is, the simpler it becomes to try new ideas with it.

It seems also important for us to provide tools, inside Mamba, helping users to "visualize" their ideas and thus be able to easily assess their worthiness. As a result, Mamba comes with a set of tools to fulfil this objective, the most visible being the capability to see manipulated images in live. These tools are a very important part of the library (equally in our eyes to the core functions used in computations). Again, the objective is to provide a simple framework for testing and trying new ideas in mathematical morphology.

2.3 Portable

Mathematical morphology algorithms variety and diversity make it somehow stupid to try to restrain their use to a very limited set of computer and device (or so we think). Thus the effort towards portability.

2.4 ... and Free

Last but not least, as we hope Mamba will become a huge success and help spread knowledge and use of mathematical morphology, we believe it is important not to restrain its future users with a complex or obscure license. Similarly we think that a commercial license is inappropriate for some of the intended audience of this library (students, university...) and too complex for us to handle with others (industry, ...). Thus we decided to license Mamba under a slightly modified version of the X11 license (also known as the MIT license). This is one of the least restrictive free license. Any contributor to Mamba will have to make sure his contribution is licensed under a similar license (see [6](#)).

3 Programming

3.1 Languages

- Mamba low level is written in C.
- Mamba high level is written in Python.

This language choice is meant to cover the Mamba objective of being a fast portable and easy to use library.

C choice for low-level answers the speed requirement and partly covers the portability. Indeed, Mamba is meant to be portable to embedded systems. Of course, the part written in C must be compilable on various environments (Windows, Unix ...) and for various processors (Pentium, Core 2 Duo, ARM, ...).

Python choice is here to ensure easiness of use. As a high level language, it makes it easier to develop programs and algorithms without taking care of memory management, OS portability ... Other languages could be used for high level but currently all the high level functionalities are written in Python. Of course, if you have the time and the need to develop a high level interface in another language, feel free to do so. However, maintaining multiple high level interfaces in different languages may prove too much hassle so your high level interface may not be integrated to the official Mamba distribution.

3.2 Rules for C

Three words can be used to sum up the philosophy of rules/standards applying to Mamba C code.

- simplicity
- readability
- portability

Simplicity means that your code must not try to answer all the problems or to take into account all the possible situations. You should leave the complexity to the high level interface (Where generally it is much easier to handle the real life situations).

Readability is a vague notion. Mainly, you have to make sure your code is understandable. Comments, coherent naming, and so on are strongly advised. More specific rules are :

- Use of english in comments is mandatory.
- Function descriptions in comments use doxygen style (see the documentation section below for more details regarding documentation).
- Indentations are done using 4 spaces (no tabs, they are ugly because they messed up when changing the editor).
- Functions and variables that are exported (visible by the exterior) should begin with "MB_".

Portability means that you should always take care to write your program so that it will run on the greatest number of machines and equipments. Of course this is not always feasible (particularly if you are going very low level). Anyway, it simply means that if, for example, you write an algorithm using SSE instructions of modern Intel/AMD processors, you should also include your algorithm in standard C code (even if that means it's excruciatingly slow).

3.3 Rules for Python

When coding in Python, no specific rules apply. However, you must take great care to make your code and the various functionalities you are implementing as simple to use as possible.

If the basic C functions requires a lot of arguments, try to categorize them in the Python code. For example, the C function :

```
MB_errcode MB_InfNbb(MB_Image *src ,
                    MB_Image *srcdest ,
                    unsigned int dirnum ,
                    unsigned int count ,
                    enum MB_grid_t grid ,
                    enum MB_bordermode_t border );
```

is wrapped in the Python by the function :

```
def infNeighborImage(imInout, imIn, d, count):
```

Information regarding border and grid are global variables in the Python module and thus do not need to be repeated every time.

Similarly, complexity is handled in the Python interface. In the previous example, the same function is used for binary, greyscale and 32-bit images in Python and corresponds to three functions in C (one for each depth). Complexity should only be handled in C if this makes computations go significantly faster.

As for comments, you need to document each function that are meant to be public with docstring. You should also begin your internal functions with a "_" that will tell Python that it's an internal function (the same goes for global variables).

4 Documentation

4.1 In code

As was explained in the programming section, documentation related to code must take two forms :

For Python code, use docstring.

For C code, use doxygen style.

You should indeed know that part of the actual documentation is created automatically using these form of documentation. All the documentation must be written in english.

4.2 Other documents

All the other documents existing to date were created using Latex. Mamba comes with a specific Latex style that allows creating an homogeneous set of documents (with header, code listing style, etc... coherent between documents).

If you have an idea of documentation, we strongly advise you to use Latex along with the Mamba style (which can be found in the source). To use it, you simply need to create a directory texmf/tex/latex in your \$HOME or wherever your Latex distribution may find it and add the files mamba.sty and mamba_log_white.png in it. We prefer Latex because it makes it easier to track modification inside our versioning repository (Latex being text format and not binary)

For those of you who do not have Latex or who do not wish to use it, we can accept documents in other formats provided you also give along a PDF version of them (making sure that way that the document will be readable by anyone).

And of course, whatever the format you choose, make sure your name appears in the document.

5 Other contributions

There is lot of things you could do for Mamba, even if your are not a programmer or a writer.

First of all, you can give us feedback regarding the way you use Mamba. Any comment, criticism or suggestion is welcome and will be taken into consideration (as long as it does not infringe our policy).

6 Licensing

For code contribution, make sure your work is licensed under a similar license than the one covering Mamba. Here is a reminder of the license :

Copyright (c) <2009>, <Your name here>

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions: The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

Except as contained in this notice, the names of the above copyright holders shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Software without their prior written authorization.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Any similar license is good (see X11 license, BSD license). Make sure your license is compatible with GNU GPL and that it has NO copyleft obligations.

For documentation contributions, make sure that your work license allow us to distribute it freely.