WHY AND HOW TO DESIGN A GENERIC AND EFFICIENT IMAGE PROCESSING FRAMEWORK: THE CASE OF THE MILENA LIBRARY

Roland Levillain$^{1,2}$, Thierry Géraud$^{1,2}$

1 EPITA Research and Development Laboratory (LRDE)
14-16, rue Voltaire
FR-94276 Le Kremlin-Bicêtre
France

Laurent Najman$^2$
2 Université Paris-Est
Laboratoire d’Informatique Gaspard-Monge
Équipe A3SI, ESIEE Paris,
Cité Descartes, BP 99
FR-93162 Noisy-le-Grand, France

ABSTRACT
Most image processing frameworks are not generic enough to provide true reusability of data structures and algorithms. In fact, genericity allows users to write and experiment virtually any method on any compatible input(s). In this paper, we advocate the use of generic programming in the design of image processing software, while preserving performances close to dedicated code. The implementation of our proposal, Milena, a generic and efficient library, illustrates the benefits of our approach.

Index Terms— Genericity, Image Processing, Software Design, Reusability, Efficiency

1. INTRODUCTION
By its very nature, Image Processing (IP) is a science dealing with many image types and other data structures (lattices, graphs, topological structures, etc.). On the other hand, the IP literature proposes methods usually expressed in or adaptable to a general context: most of them are usually not tied to a specific image type. However, software for IP often addresses a small combinations of image types and algorithms. Many IP framework are dedicated to a specific area and provide a selection of algorithms for a few data structures.

For instance, many tools (CImg, ImLib3D, ImageJ, etc.) handle only regular $n$-dimensional images (with $n \in [1, 4]$) set on regular grids. Other are specialized in an application domain like medical imaging (ITK), remote sensing (OR-PHEO Toolbox), document image analysis (Leptonica). Others favor a class of IP methods : mathematical morphology (Morph-M), linear algebra (Gandalf), etc. To sum up, these tools are not generic, since they address a class of specific needs.

We believe a modern IP research framework should observe the following traits:

Genericity If a structure or an algorithm can be described by a general and unique definition independently of the context, it should have a single, generic implementation. The section 2 of this paper compares a classical, non-generic implementation of a simple algorithm with a generic counterpart.

Modular Design One of the goal of designing generic software is to make it truly reusable: methods from one domain or used on a certain kind of images should be transferable to other domains and images. The required architecture is highly dependent on the modularity and orthogonality of its components. We propose an architecture for a generic and modular IP framework in Sections 3 and 4.

Efficiency The generic implementation of a structure or of an algorithm should be as efficient as possible with respect to run time speed and memory usage. Dedicated implementations known to perform better (faster and/or more compact) in certain cases may be provided, and automatically selected by the framework when possible. The issue of retaining performances in a generic environment is addressed in Section 5.

Ease of Use Many IP practitioners are not computer scientists and should not have to deal with technical problems. Issues like memory management or arithmetic overflows should be handled efficiently and cleverly by the system.

Theory Resemblance The abstract and general aspect of the theory should be preserved as much as possible. Algorithms expressed in the considered environment should look natural to IP scientists used to mathematical notations.

Usability The framework should be implemented with portable, widely used and known tools, so that no specific knowledge is required from its users (e.g., an exotic programming language or a complicated platform). Besides,
Reproducible Research
A research environment should help its users to promote reproducible research, i.e., the possibility to analyze, compare, reproduce and extend published results. As the framework is generic and freely reusable, it is possible to implement new algorithms, compare them to reference methods and improve the reproducibility of benchmarks.

There are several generic IP libraries (ITK [1], VIGRA [2], Morph-M [3], GIL [4]), but in our opinion they fail to fulfill the whole conditions above-mentioned. In this article, we propose a framework for generic and efficient Image Processing, Milena [5]. Our contributions are the following:

- Rethinking IP tools under the light of genericity, using a Generic Programming (GP) paradigm.
- Establishing a set of abstractions for IP programming, called concepts.
- Proposing a library implementing these ideas.

To illustrate the benefits of a generic approach in IP, we present some examples of a simple yet effective generic processing chain in Section 6. Section 7 concludes.

**2. BRINGING GENERICITY TO IMAGE PROCESSING**

Let us consider a very simple algorithm, filling the pixels of an image with a given value (Algorithm 1). This code makes a few hypotheses on its inputs:

- The image is a 2-dimensional one.
- Its points have nonnegative coordinates starting at 0.
- Its values must be compatible with unsigned char.

Therefore we cannot reuse this code as-is to process an image

- with 3 dimensions;
- with points having negative coordinates;
- with points having floating-point coordinates;

as the framework is intended to process any kind of image, it should be able to handle voluminous inputs, like images of several gigabytes.

**Freedom of Use** Research software should minimize the barriers to help share and propagate knowledge. Free/Libre Open Source Software (FLOSS) is a guarantee that everybody can access and benefit from tools used in research experiments, adapt, improve and extend them, just like any other research medium.

**Reproducible Research** A research environment should present some examples of a simple yet effective generic processing chain in Section 6.
Models To be actually useful, the library must provide models for each concept. For instance, `image2d<T>` (see Figure 2) is a model of Image representing a 2D image associating values of type T to 2D discrete points.

Properties Models can define specific properties that will be used to select an optimal implementation of an algorithm. For example `image2d<T>` stores its values in a linear buffer and reflect this as a property.

Algorithms Generic algorithms are written using the concepts, not the models. Therefore, the most general formulation is used, and can be shared among all models.

Auxiliary Tools The library may provide syntactic sugar (programming shortcuts) to make reading and writing algorithms easier. For instance, the `for all` macro of Algorithm 2 simplifies the iteration over a point set. Likewise, `mln::piter(I)` deduces the type of point iterator (piter) from an image of type I.

4. CONCEPTS IN IMAGE PROCESSING

Milena contains more than 40 concepts organized in a hierarchy. In this section, we introduce some of these core concepts.

4.1. The Image Concept

In order to design a generic framework for image processing, we propose the following definition of an in image [5, 6].

Definition. An image I is a function from a domain D to a set of values V. The elements of D are called the sites of I, while the elements of V are its values.

For the sake of generality, we use the term site instead of point: if the domain of I were a Region Adjacency Graph (RAG), it would be awkward to refer to its elements (the regions) as “points”.

This definition translates easily into the concept of Figure 1. All images must satisfy (or model) this concept, i.e. they must give a valid definition for each associated type and each method. For instance, Figure 2 shows the associated types of `image2d<T>`, an image type modeling the Image concept. This type is built on a rectangular subset of the 2D plane set on an orthonormal grid (`box2d`), and associates values from the type T to its sites (`point2d`).

4.2. Site, Set, Site and Site Iterator Concepts

Sites generalize the notion of points. Almost every entity can be used as a site, i.e., to form the domain of an image. Information of spatial location is not part of the requirements of a Site, in order to handle “abstract” images built on domains with no actual geometrical parameters (e.g., an abstract graph). Likewise, a site itself may or may not convey topological information. For these reasons, the signature of the Site concept is almost empty, and requires only routines comparing sites between them.

Site sets are collections of sites. Such sets can be explicitly defined by an enumeration (examples include arrays of sites) or implicitly defined (e.g., with a bounding box). The only requirement on site sets is their ability to be browsed forward and backward by iterators: small objects used in site set iterations and convertible to sites, like p in Algorithm 2. Unlike many libraries, Milena’s iterators do not look like unfamiliar, non-IP programming artifacts, as they resemble points.

5. GENERIC ALGORITHMS AND EFFICIENCY

The efficiency of Milena lies in the following design choices:

Compiled Language Milena is written in C++, generating code running faster than an interpreted program.

Static Generic Programming We exercise genericity through a static programming paradigm, instead of a dynamic one based on polymorphic methods [1]: the cost of genericity is paid at compile time, letting algorithms execute at optimal speed at run time.

Property-Based Algorithm Selection Specialized versions of algorithms can be supported thanks to a mechanism more powerful than bare function overloading [8]. Each algorithm facade may inspect its inputs’ properties and use them to possibly select an implementation.
more efficient than the generic one. For example an algorithm iterating sequentially (e.g. fill) applied to an image with a linear buffer (e.g. image2d<T>) may use pointers instead of point2d objects to run faster.

**Access to Low-Level Features** C++ lets the user make use of low-level features either in general or specialized algorithms, thus benefiting from hardware capabilities.

6. ILLUSTRATIONS

In this section, we consider a simple, classical image processing chain: from an image ima, compute an area closing c using criterion value l; then, perform a watershed transform by flooding on c to obtain a segmentation s. We apply this chain on different images ima. All of the illustrations of Figure 3 use the exact same following Milena code:

```cpp
template<
typename L, typename I, typename N>
mln_ch_value(I, L)
chain(const I& ima, const N& nbh, int l, L& nb) {
    mln_concrete(I) c = closing::area(ima, nbh, l);
    return watershed::flooding(c, nbh, nb);
}
```

7. CONCLUSION

This paper advocates the use of generic programming to design and implement Image Processing (IP) frameworks. By following these guidelines, we have built an efficient and generic IP library. We have been using Milena for many applications, in particular in the fields of mathematical morphology and digital topology. Milena is Free Software released under the GNU General Public License. It is part of the Olena platform and can be freely downloaded at [http://olena.lrde.epita.fr/Download](http://olena.lrde.epita.fr/Download). To improve the usability of Milena, we are working on adding extra utilities on top of the library: bridges to other languages (starting with Python), dynamic interpreted environments, command-line tools, Graphical User Interfaces (GUIs), etc.

8. REFERENCES


