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], Laurent Najman[2]

[1]EPITA Research and Development Laboratory (LRDE), France
[2]Université Paris-Est, Laboratoire d'Informatique Gaspard-Monge (IGM), ESIEE Paris, France
{roland.levillain,thierry.geraud}@lrde.epita.fr, l.najman@esiee.fr

At a Glance

The Problem Most Image Processing (IP) frameworks are not generic enough to provide true reusability of data structures and algorithms.

The Point Genericity allows users to write and experiment virtually any method on any compatible input(s).

Our Contribution A generic programming framework to design IP software, able to preserve performances close to dedicated code.

The Outcome The implementation of our proposal, **Milena**, a generic and efficient C++ library, illustrates the benefits of our approach.

Desired Properties of a Modern IP Framework

Genericity A single structure or algorithm definition ⇒ a single, generic implementation.

Modular Design Modular, orthogonal components ⇒ reusability (on other images or contexts).

Efficiency An algorithm: a generic version + optional, dedicated, more efficient variants.

Ease of Use Look familiar to IP practitioners. Hide technical difficulties.

Theory Resemblance Use classical, mathematical notations preserving the generality of the theory.

Usability Made with portable, widely used tools. Able to handle large data (gigabytes).

Freedom of Use Share and spread knowledge with Free/Libre Open Source Software (FLOSS).

Reproducible Research Reusability helps to analyze, compare, reproduce and extend results.

The `fill` Example: Non Generic vs Generic Algorithm

A Non Generic Algorithm

```
void fill(image& ima, unsigned char v) {
  for (unsigned int r = 0; r < ima.nrows(); ++r)
    for (unsigned int c = 0; c < ima.ncols(); ++c)
      ima(r, c) = v;
}
```

An Abstract Definition A general definition of `fill` where D is ima's domain:

$$\forall p \in D \quad ima(p) \leftarrow v$$

A Generic Algorithm

```
template <typename I, typename V>
void fill(Image<I>& ima_, const V& v) {
  I& ima = exact(ima_);            // Convert to concrete type
  mln_piter(I) p(ima.domain());    // Let p ∈ D
  for_all(p) ima(p) = v;           // ∀p  ima(p) ← v
}
```

Components of a Generic IP Library

Concepts General description of an abstract notion of the domain. In IP: Image, Site, Value, Neighborhood, Function, etc. E.g.: An image I is a function from a domain D (the *sites* of I) to a set of values V [3].

Models Instance of a concept.

Properties Traits of an object, used to select the optimal version of an algorithm.

Algorithms Written using concepts, not models ⇒ generic behavior.

Auxiliary Tools E.g.: `for_all` loops; `mln_piter(I)`: image type ↦ site type.

Image concept	
Associated types	
`domain_t`	Type of the domain
`site`	Type of a site
`fwd_piter`	Forward iterator type
`bkd_piter`	Backward iterator type
`vset`	Type of the set of values
`value`	Type of a value
Services (methods)	
`value operator()(site& p)`	Value at `ima(p)`
`bool has(const psite& p)`	Site membership test
`const domain_t& domain()`	Return the domain (D)
`const vset& values()`	Return the value set (V)

image2d<T>, a model of Image		
Associated types		
`domain_t`	:	`box2d`
`site`	:	`point2d`
`fwd_piter`	:	`box2d::fwd_piter`
`bkd_piter`	:	`box2d::bkd_piter`
`vset`	:	`value::set<T>`
`value`	:	`T`

Efficiency Considerations

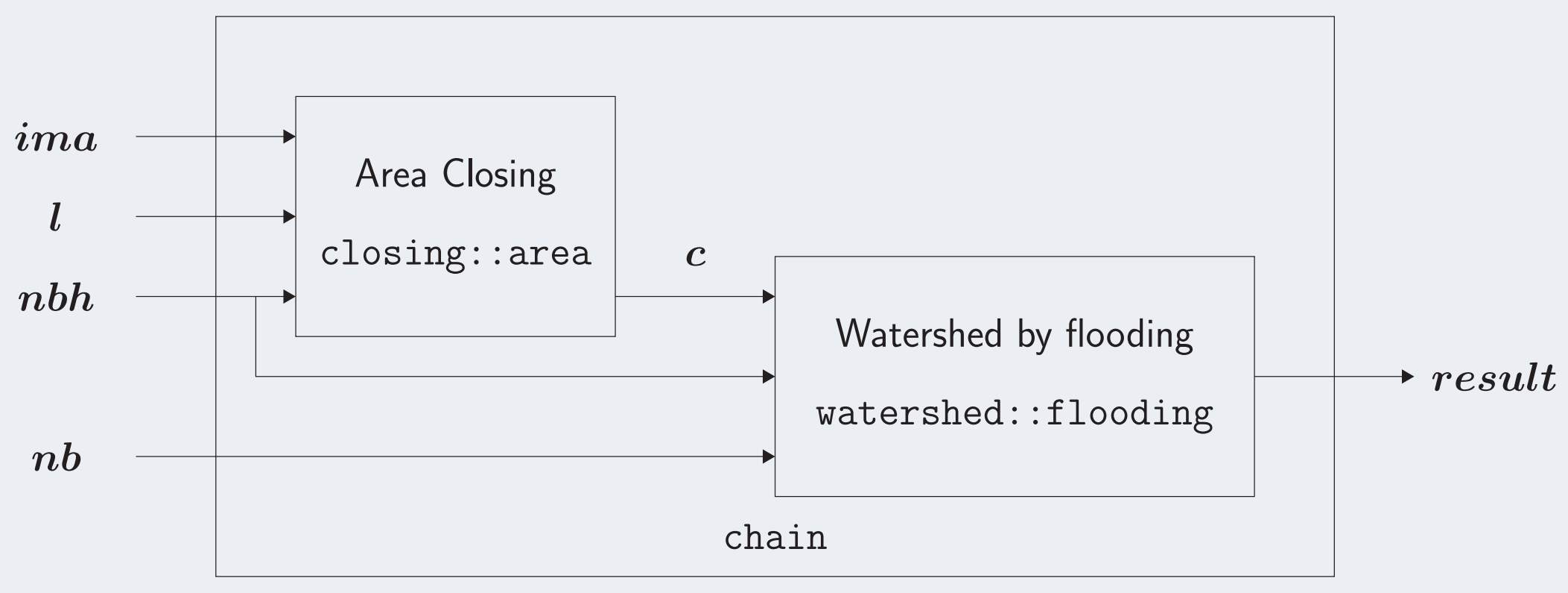
Compiled Language Written in C++, code running faster than interpreted programs.

Static Generic Programming No dynamic polymorphic methods (`virtual`) [1].

Property-Based Algorithm Selection Automatic selection of the best variant [2].

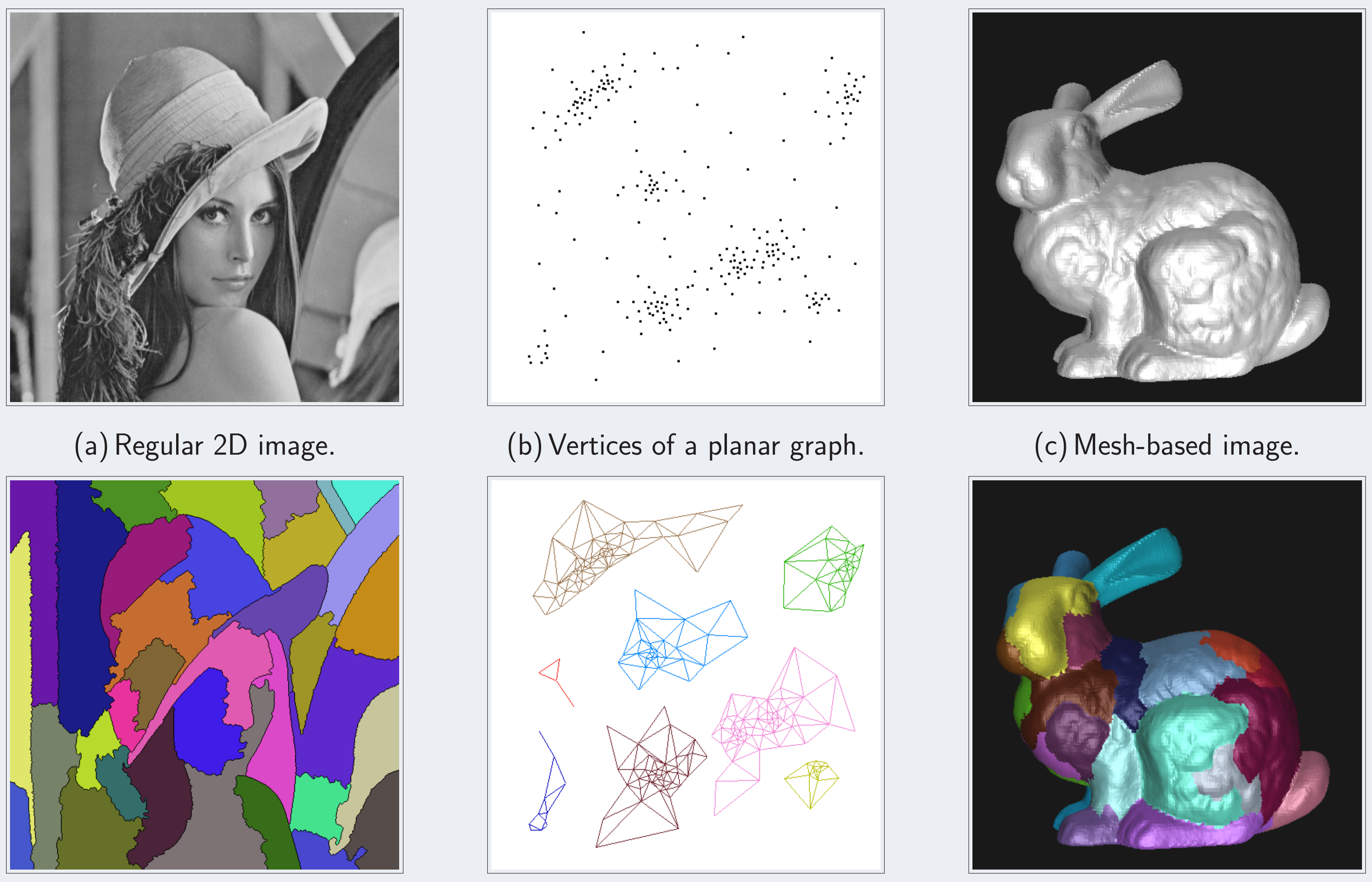
Access to Low-Level Features Naturally available from C++.

Illustration: a Generic Segmentation Processing Chain



ima	Input image.
l	Area closing criterion.
nbh	Neighborhood.
nb	Resulting number of regions.
c	Image simplified by area closing.
$result$	Output image.

```
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);
  mln_ch_value(I, L) result = watershed::flooding(c, nbh, nb);
  return output;
}
```



(a) Regular 2D image. (b) Vertices of a planar graph. (c) Mesh-based image.

(d) Result on the gradient of (a). (e) Result on edges' lengths of (b). (f) Result on the curvature of (c).

More information on Milena

Project A part of the **Olena** platform.

Latest Version 1.0 (July 14, 2009).

License GNU General Public License (GNU GPL).

Web `http://olena.lrde.epita.fr`

Contact `olena@lrde.epita.fr`

Contributors More than 50 (for 10 years).

Code Size >150.000 lines (according to David A. Wheeler's 'SLOCCount').

References

[1] Nicolas Burrus, Alexandre Duret-Lutz, Thierry Géraud, David Lesage, and Raphaël Poss.
A static C++ object-oriented programming (SCOOP) paradigm mixing benefits of traditional OOP and generic programming.
In *Proceedings of the Workshop on Multiple Paradigm with Object-Oriented Languages (MPOOL)*, Anaheim, CA, USA, October 2003.

[2] Thierry Géraud and Roland Levillain.
Semantics-driven genericity: A sequel to the static C++ object-oriented programming paradigm (SCOOP 2).
In *Proceedings of the 6th International Workshop on Multiparadigm Programming with Object-Oriented Languages (MPOOL)*, Paphos, Cyprus, July 2008.

[3] Roland Levillain, Thierry Géraud, and Laurent Najman.
Milena: Write generic morphological algorithms once, run on many kinds of images.
In Springer-Verlag, editor, *Proceedings of the Ninth International Symposium on Mathematical Morphology (ISMM)*, Lecture Notes in Computer Science Series, pages 295–306, Groningen, The Netherlands, August 2009.