Milena: Write Generic Morphological Algorithms Once, Run on Many Kinds of Images

Roland Levillain[1,2], Thierry Géraud[1,2], Laurent Najman[2]

[1]EPITA Research and Development Laboratory (LRDE), Le Kremlin-Bicêtre, France

[2]Université Paris-Est, Laboratoire d'Informatique Gaspard-Monge (IGM), Équipe A3SI, ESIEE Paris, Noisy-le-Grand Cedex, France

International Symposium on Mathematical Morphology (ISMM)

Groningen, The Netherlands – August 24 - 27, 2009

Intent

Context

- Software solutions for Mathematical Morphology (MM).
- Reusability, flexibility (and efficiency).

Aim of this talk

- Think Generic!
- Advertise about a generic software framework proposal.

Genericity in MM at a Glance

Wouldn't it be nice to be able to implement an algorithm like this?

```
for_all(p)
  {
    sup = input(p);
    for_all(q)
      sup.take(input(q));
    output(p) = sup;
  }
```

Go to full code

Generic code: works on every kind of compatible images.

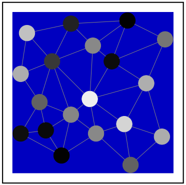
Genericity in MM at a Glance

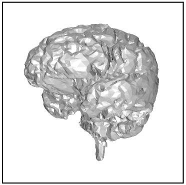
Wouldn't it be nice to be able to implement an algorithm like this?

```
for_all(p)
  {
    sup = input(p);
    for_all(q)
      sup.take(input(q));
    output(p) = sup;
  }
```

Go to full code

Generic code: works on every kind of compatible images.





Observations and Reflections

Observations

- Writing reusable MM software with good properties is hard.
- Even harder if you want to preserve other traits such as ease-of-use, efficiency, readability and flexibility.

Reflections

- Where should we start if we wanted to achieve the (impossible) goal of writing MM for all users and applications?
- Idea: Start with a core component with good features, and then build tools on top of it.
- What core?

The Nature of the Core

This core shall be:

- General enough to serve a reusable basis.
- Good enough to be used as-is.
- Accessible to all MM users.
- Extensible.
- Compatible with today's systems.

→ A generic library.

The Nature of the Core

This core shall be:

- General enough to serve a reusable basis.
- Good enough to be used as-is.
- Accessible to all MM users.
- Extensible.
- Compatible with today's systems.

$\rightarrow$ A generic library.

Implementing Mathematical Morphology Algorithms

A Very Simple Operator

Let's consider the morphological dilation of an image I with a flat structuring element B.
A mathematical definition:

$$\delta_B(I)(x) = \sup_{h \in B} I(x + h)$$

where

- I is a function $D \to V$ associating a point from the domain D to a value from the set V.
- B is a structuring element.

A Non Generic Implementation

```
for (unsigned int r = 0; r < input.nrows(); ++r)
  for (unsigned int c = 0; c < input.ncols(); ++c) {
    unsigned char sup = input(r, c);
    if (input(r-1, c) > sup) sup = input(r-1, c);
    if (input(r+1, c) > sup) sup = input(r+1, c);
    if (input(r, c-1) > sup) sup = input(r, c-1);
    if (input(r, c+1) > sup) sup = input(r, c+1);
    output(r, c) = sup;
  }
```

This solution makes a few hypotheses:

1. The image is 2-dimensional.
2. Points have nonnegative integers coordinates starting at 0.
3. Values have to be compatible with the 8-bit `unsigned char` type.
4. The values of the image form a totally ordered set.
5. The structuring element is based on the 4-connectivity.

A Non Generic Implementation

```
for (unsigned int r = 0; r < input.nrows(); ++r)
  for (unsigned int c = 0; c < input.ncols(); ++c) {
    unsigned char sup = input(r, c);
    if (input(r-1, c) > sup) sup = input(r-1, c);
    if (input(r+1, c) > sup) sup = input(r+1, c);
    if (input(r, c-1) > sup) sup = input(r, c-1);
    if (input(r, c+1) > sup) sup = input(r, c+1);
    output(r, c) = sup;
  }
```

This solution makes a few hypotheses:

1. The image is 2-dimensional.
2. Points have nonnegative integers coordinates starting at 0.
3. Values have to be compatible with the 8-bit `unsigned char` type.
4. The values of the image form a totally ordered set.
5. The structuring element is based on the 4-connectivity.

A Non Generic Implementation: Some Limitations

This code cannot handle (as-is) any of the following variations:

1. The input is a 3-dimensional image.
2. Its points are located on a box subset of a floating-point grid.
3. The values are encoded as 12-bit integers or as floating-point numbers.
4. The image is multivalued (e.g., a 3-channel color image).
5. The structuring element represents an 8-connectivity.

Though less common, images with these features can be found in fields like biomedical imaging, astronomy, document image analysis or arts.

A Non Generic Implementation: More Limitations

What if...

- ...the domain D of I...
  - ...is not an hyperrectangle (a “box”)?
  - ...is not a set of points located in a geometrical space, but a 3D triangle mesh?
  - ...is a restriction (subset) of another image’s domain?
- ...the neighbors of a site are not expressed with a fixed-set structuring element, but through a function associating a set of sites to any site of the image?
- ...the values are not scalar, nor vectorial (e.g., diffusion tensors)?

Back to the Generic Implementation

```
for_all(p)
  {
    sup = input(p);
    for_all(q)
      sup.take(input(q));
    output(p) = sup;
  }
```

where

- p $\in D$
- q $\in$ win(p)
- sup is a small object (accumulator) computing the supremum of a set of values.

A Generic Implementation: Benefits

```
for_all(p)
  {
    sup = input(p);
    for_all(q)
      sup.take(input(q));
    output(p) = sup;
  }
```

A few remarks:

- Small yet readable.
- Not tied to specific image type, i.e, generic.
- Example: dilating an image I where
  - $D(I) =$ a Region Adjacency Graph (RAG) where each site is a region of another image J.
  - $V(I) = \mathbb{R}^n$ (n-dimensional vectors expressing features from each underlying region of J).
  - $\forall$p, q browses the sites (i.e., regions) adjacent to p.

Introducing Genericity in MM

Main Idea

- MM software should be generic [Köthe, 1999, Darbon et al., 2002, d'Ornellas and van den Boomgaard, 2003].

Modus Operandi

- Genericity is possible provided abstractions of concepts of the domain are well defined.
- In Object-Oriented Programming (OOP), these abstractions are called interfaces.

Genericity in Mathematical Morphology

A Generic Definition of the Concept of Image

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.

Abstractions in Generic Programming

With Generic Programming (GP):

- Algorithms are no longer defined in terms of features specific to an image type.

```
for (unsigned int r = 0; r < input.nrows(); ++r)
  for (unsigned int c = 0; c < input.ncols(); ++c)
    ...
```

- Instead, abstractions are used.

```
mln_piter(I) p(input.domain()); // 'p' is a site iterator.
for_all(p)                      // ∀p ...
  ...
```

The Image Abstraction

The interface of an image type includes:

- Associated types.

```
typedef domain_t;  // Type of the domain (site set).
typedef site;      // Type of a site.
typedef piter;     // Associated iterator type.
typedef value;     // Type of a value.
typedef vset;      // Type of the set of values.
```

- Methods (services provided by the image).

```
value operator()(site p);  // ‘ima(p)’ → value
bool has(site p);          // Does ‘p’ belongs to ‘ima’?
vset values();             // Return the domain (D).
domain_t domain();         // Return the value set (V).
```

Other Abstractions

Site_Set Sets of sites must respect this interface.

```
typedef site;         // The type of the sites.
typedef fwd_piter;    // Forward iterator on the set's sites.
typedef bkd_piter;    // Backward iterator on the set's sites.

bool has(psite p);    // Does 'p' belongs to this set?
```

Also: Point_Site, Delta_Point_Site, Site_Iterator, Value, Value_Set, Value_Iterator, Neighborhood, Window, Weighted_Window, Accumulator, Function, ...

Introducing Milena

Good News

- The previous concepts can be translated to actual, working C++ code almost as-is.
- They have been implemented as a library (core component).
- Plus many, many more tools.

Milena

- A generic and efficient C++ library [Géraud and Levillain, 2008].
- Released within the Olena 1.0 platform.

Milena

What is in Milena?

- Abstractions
- Data structures, in particular site sets.
- Many image types (and their associated types), mostly built upon classical site sets (domains). E.g.:
  - Box on a regular 2D grid → `image2d`
  - Undirected graph → { image with values on vertices. / image with values on edges.
- Many auxiliary tools to make it easy to use and write algorithms (macros, accumulators, functions, etc.).
- Algorithms, in particular (but not only) in the field of on MM.

Site Sets

- Convey a lot of structural and topological information.
- Classical cases: hyperrectangles (boxes) on regular *n*-dimensional grids.
- But also: unstructured site sets based on usual data structures:
  - Arrays.
  - Sets.
  - Priority queues.
  - Hybrid containers.
  - etc.
- (Undirected) Graphs.
- Complexes.

Site Sets: Complexes

Definition

A simplicial complex is “a set of simplices”, where a simplex is the simplest manifold that can be created using *n* points.

- A 0-simplex ≡ a point.
- A 1-simplex ≡ a line segment.
- A 2-simplex ≡ a triangle.
- A 3-simplex ≡ a tetrahedron.

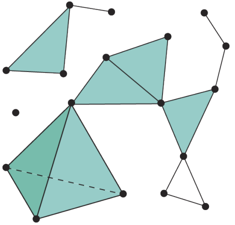


Figure: A simplicial 3-complex.

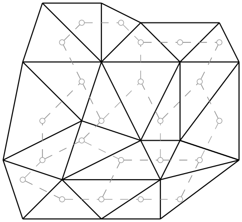


Figure: A simplicial 2-complex (mesh).

Site Sets: Complexes (cont.)

Ideal framework to process mesh-based “images”.

Figure: Triangle mesh, seen as a simplicial 2-complex.

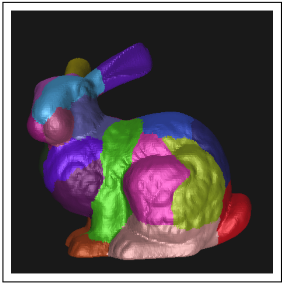


Figure: Watershed-based segmentation using curvature computed on edges [Meyer, 1991, Cousty et al., 2008].

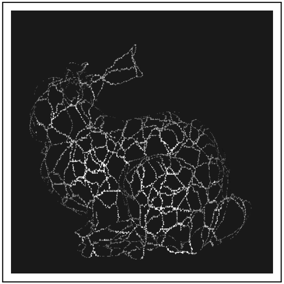


Figure: Skeleton using breadth-first thinning based on simple point characterization [Couprie and Bertrand, 2009].

Site Sets: More

- Milena's implementation of complexes is flexible enough to implement many structures: cubical complexes, simplicial complexes, etc.
- Providers can add new structures (either generic or not) to Milena and benefiting from the framework of the library (reuse algorithms, accumulators, etc.).
- Milena introduces no actual additional (run time/space/development) cost in itself.

Even More: Morphers

- Morphers: lightweight objects producing an image from an image.
- Kind of a function on an image.
- Example: Dilation by a 4-c structuring element:

```
dilation(ima, win_c4p());
```

- Likewise, but restricting the domain of `ima` to the subset `s`:

```
dilation(ima | s, win_c4p());
```

- Dilation of the red channel of an RGB color image:

```
dilation(red << rgb_ima, win_c4p());
```

- Many morphers provided by Milena: wrapping an image (torus), stacking several images, taking a 2D slice from a 3D volume, applying a geometrical transformation, etc.

Even More: Morphers

- Morphers: lightweight objects producing an image from an image.
- Kind of a function on an image.
- Example: Dilation by a 4-c structuring element:

```
dilation(ima, win_c4p());
```

- Likewise, but restricting the domain of `ima` to the subset `s`:

```
dilation(ima | s, win_c4p());
```

- Dilation of the red channel of an RGB color image:

```
dilation(red << rgb_ima, win_c4p());
```

- Many morphers provided by Milena: wrapping an image (torus), stacking several images, taking a 2D slice from a 3D volume, applying a geometrical transformation, etc.

Even More: Morphers

- Morphers: lightweight objects producing an image from an image.
- Kind of a function on an image.
- Example: Dilation by a 4-c structuring element:

```
dilation(ima, win_c4p());
```

- Likewise, but restricting the domain of `ima` to the subset `s`:

```
dilation(ima | s, win_c4p());
```

- Dilation of the red channel of an RGB color image:

```
dilation(red << rgb_ima, win_c4p());
```

- Many morphers provided by Milena: wrapping an image (torus), stacking several images, taking a 2D slice from a 3D volume, applying a geometrical transformation, etc.

Even More: Morphers

- Morphers: lightweight objects producing an image from an image.
- Kind of a function on an image.
- Example: Dilation by a 4-c structuring element:

```
dilation(ima, win_c4p());
```

- Likewise, but restricting the domain of `ima` to the subset `s`:

```
dilation(ima | s, win_c4p());
```

- Dilation of the red channel of an RGB color image:

```
dilation(red << rgb_ima, win_c4p());
```

- Many morphers provided by Milena: wrapping an image (torus), stacking several images, taking a 2D slice from a 3D volume, applying a geometrical transformation, etc.

A Simple Milena Processing Chain

- A generic code:

```
closed = morpho::closing::area(ima, nbh, lambda);
wshed  = morpho::watershed::flooding(closed, nbh, nb);
```

Go to full code

- Inputs:
  - `ima` Input image (e.g, image2d<int>, image3d<float>, complex_image, etc.).
  - `nbh` Neighborhood (e.g., `c4`, `c26`, `adjacent_vertices_neighborhood`, etc.).
  - `lambda` Value of the criterion (integer).
- Applicable to many different image types as-is.

Results (2D Image, 6-connectivity)

Figure: "Classical" image, with 6-connectivity.

Figure: Magnitude of the gradient.

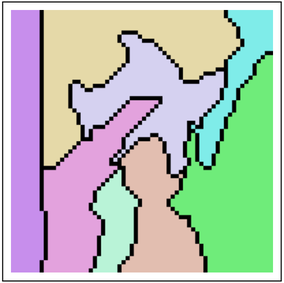


Figure: Result of the image processing chain on the magnitude of the gradient.

Results (Cubical 2-Complex)

Figure: Magnitude of the gradient computed on the edges of a cubical 2-complex.

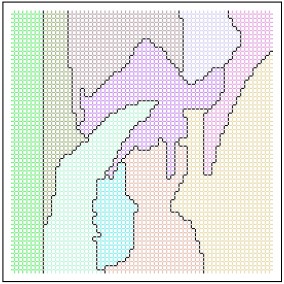


Figure: Result of the image processing chain.

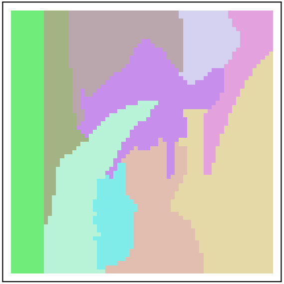


Figure: Extension of labels to 2-faces (squares) for visualization purpose.

Results (Graph)

Example of data clustering using MM methods.

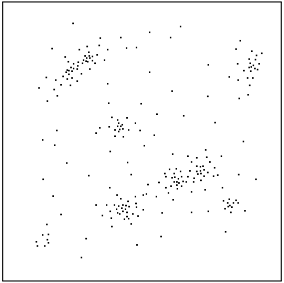


Figure: Vertices of a graph.

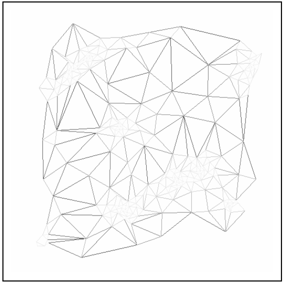


Figure: Distance-based magnitude computed on the edges of the triangulation of the graph.

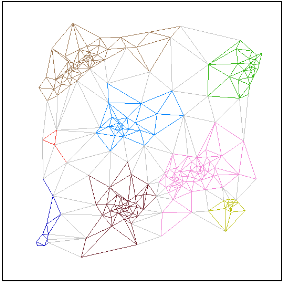


Figure: Result of the image processing chain on this magnitude.

Epilogue

The Present: Olena 1.0

- The latest version of Milena ships with the Olena 1.0 platform, released July 14, 2009.

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

- We invite you to download and try it.
- Olena is Free Software released under the GNU General Public License (GNU GPL).
- There is much more to say about Milena, in particular about efficiency.
- Send questions and comments to: olena@lrde.epita.fr.

The Future

- We are actively working on making the library easier to install, learn and use.
- Milena is the heart of the platform we are developing. We are working on adding
  - GUIs,
  - bridges to other languages (starting with Python),
  - command-line tools,
  - etc.

  while retaining as many advantage from Milena's core features as possible (namely genericity and efficiency).

Thank You!

Milena: Write Generic Morphological Algorithms Once, Run on Many Kinds of Images

Bibliography I

Couprie, M. and Bertrand, G. (2009).
New characterizations of simple points in 2d, 3d, and 4d discrete spaces.
IEEE Transactions on Pattern Analysis and Machine Intelligence, 31(4):637–648.

Cousty, J., Bertrand, G., Najman, L., and Couprie, M. (2008).
On watershed cuts and thinnings.
In *Proceedings of the 14th IAPR International Conference on Discrete Geometry for Computer Imagery (DGCI)*, pages 434–445, Lyon, France.

Bibliography II

Darbon, J., Géraud, Th., and Duret-Lutz, A. (2002).
Generic implementation of morphological image operators.
In *Mathematical Morphology, Proceedings of the 6th International Symposium (ISMM)*, pages 175–184, Sydney, Australia. CSIRO Publishing.

d'Ornellas, M. C. and van den Boomgaard, R. (2003).
The state of art and future development of morphological software towards generic algorithms.
International Journal of Pattern Recognition and Artificial Intelligence, 17(2):231—255.

Bibliography III

Géraud, Th. and Levillain, R. (2008).
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'08)*, Paphos, Cyprus.

Köthe, U. (1999).
Reusable software in computer vision.
In Jähne, B., Haussecker, H., and Geißler, P., editors, *Handbook of Computer Vision and Applications*, volume 3: Systems and Applications, pages 103–132. Academic Press, San Diego, CA, USA.

Bibliography IV

Meyer, F. (1991).
Un algorithme optimal de ligne de partage des eaux.
In *Actes du 8e Congrès AFCET*, pages 847–857, Lyon-Villeurbanne, France. AFCET.

Full Code of the Dilation

```
template <typename I, typename W>
mln_concrete(I)
dilation(const I& input, const W& win)
{
  mln_concrete(I) output;
  initialize(output, input);
  mln_piter(I) p(input.domain());
  mln_qiter(W) q(win, p);
  for_all(p)
  {
    accu::supremum<mln_value(I)> sup = input(p);
    for_all(q) if (input.has(q))
      sup.take(input(q));
    output(p) = sup;
  }
  return output;
}
```

Go to simplified code

Actual Code of the Illustrations

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

Go to simplified code