Writing Reusable Digital Geometry Algorithms in a Generic Image Processing Framework

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

[1] EPITA Research and Development Laboratory (LRDE)
[2] Université Paris-Est, Laboratoire d'Informatique Gaspard-Monge, Équipe A3SI, ESIEE Paris
roland.levillain@lrde.epita.fr, thierry.geraud@lrde.epita.fr, l.najman@esiee.fr

Abstract Digital Geometry software should reflect the generality of the underlying mathematics: mapping the latter to the former requires genericity. By designing generic solutions, one can effectively reuse digital geometry data structures and algorithms. We propose an image processing framework focused on the Generic Programming paradigm in which an algorithm on the paper can be turned into a single code, written once and usable with various input types. This approach enables users to design and implement new methods at a lower cost, try cross-domain experiments and help generalize results[1].

Keywords Generic Programming, Interface, Skeleton, Complex

1 Introduction

Like Mathematical Morphology (MM), Digital Geometry (DG) has many applications in image analysis and processing. Both present sound mathematical foundations to handle many types of discrete images. In fact, most methods from digital geometry or mathematical morphology are not tied to a specific context (image type, neighborhood, topology): they are most often described in abstract and general terms, thus not limiting their field of application. However, software packages for MM and DG rarely take (enough) advantage of this generality: an algorithm is sometimes reimplemented for each image and/or value type, or worse, written for a single input type. Such implementations are not reusable because of their lack of *genericity*. These limitations often come from the implementation framework, which prohibits a generic design of algorithms.

Thanks to the Generic Programming (GP) paradigm, provided in particular by the C++ language, one can design and implement generic frameworks, in particular in the field of scientific applications where the efficiency, widespread availability and standardization of C++ are real assets. To this end, we have designed a paradigm dedicated to generic and efficient scientific software [4] and applied the idea of generic algorithms to MM in Image Processing (IP) [6], as suggested by d'Ornellas and van den Boomgaard [3]. The result of our experiments is a generic library, Milena, part of the Olena image processing platform.

Lamy proposes to implement digital topology in IP libraries [5]. The suggested solution, applied to the ITK library "*works for any image dimension*". In this paper, we present a framework for the generic implementation of DG methods within the Milena library, working for *any image type* supporting the required notions (value types, geometric and topological properties, etc.). Such a generic framework requires the definition of concepts from the domain (in particular, of an image) to organize data structures and algorithms (see Section 2). Given these concepts one can write generic algorithms, like homotopic skeletonizations based on various definitions of the notion of simple point, as illustrated in Section 3. Section 4 concludes on the extensibility of this work along different axes: existing algorithms, new data structures and efficiency.

[1] This work has been conducted in the context of the SCRIBO project (http://www.scribo.ws/) of the Free Software Thematic Group, part of the "System@tic Paris-Région" Cluster (France). This project is partially funded by the French Government, its economic development agencies, and by the Paris-Région institutions.

2 Genericity in Image Processing

In order to design a generic framework for image processing, we have proposed the following definition of an image [6]: 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*; e.g. sites could represents the triangle of surface mesh used as the domain of an image. Classical site sets used as image domains encompass hyperrectangles (boxes) on regular n-dimensional grids, graphs and complexes (see Section 3).

In the GP paradigm, these essential notions (image, site set, site, value) must be translated into interfaces called *concepts* in Milena (Image, Site_Set, etc.) [7]. These interfaces contain the list of *services* provided by each type belonging to the concept, as well its *associated types*. For instance, a type satisfying the Image concept must provide a `domain()` routine (to retrieve D), as well as a `domain_t` type (i.e. the type of D) satisfying the Site_Set concept. Concepts act as contracts between providers (types satisfying the concept) and users (algorithms expressing requirements on their inputs and outputs through concepts). For instance, the `breadth_first_thinning` routine from Algorithm 3 expects the type `I` (of `input`) to fulfill the requirements of the Image concept. Likewise `nbh` must be a Neighborhood; and `is_simple` and `constraint`, functions taking a value of arbitrary type and returning a Boolean value (`Function_v2b` concept).

3 Generic Implementation of Digital Geometry

Let us consider the example of a homotopic skeletonization by thinning. Such an operation can be obtained by the removal of *simple points* (or simple sites in the Milena parlance) using Algorithm 1 [1]. A point of an object is said to be simple if its deletion does not change the topology of the object. This algorithm takes an object X and a constraint K (a set of points that must not be removed) and iteratively deletes simple points of $X \backslash K$ until stability is reached. Algorithm 1 is an example of an algorithm with a general definition that could be applied to many input types in theory. But in practice, software tools often allow a limited set of such input types (or even just a single one), because some operations (like "is simple") are bound to the definition of the algorithm [6].

Algorithm 2 shows a more general version of Algorithm 1, where implementation-specific elements have been replaced by *mutable* parts: a predicate stating whether a point A is simple with respect to a set X (*is_simple*); a routine "detaching" a (simple) point A from a set X (*detach*); and a predicate declaring whether a condition (or a set of conditions) on A is satisfied before considering it for removal (*constraint*). The algorithm takes these three functions as arguments in addition to the input X. Algorithm 2 is a good candidate for a generic C++ implementation of the breadth-first thinning strategy and has been implemented as Algorithm 3 in Milena. The set X is represented by a binary image ($V = \{\top, \bot\}$), that must be compatible with operations performed within the algorithm. Inputs *is_simple*, *detach* and *constraint* have been turned into function objects (also called *functors*).

There are local characterizations of simple points in 2D, 3D and 4D, which can lead to look-up table (LTU) based implementations [2]. However, since the number of configurations of simple and non-simple points in $\mathbb{Z}^d$ is 2^{3^d-1}, this approach can only be used in practice in 2D (256 configurations, requiring a LTU of 32 bytes) and possibly in 3D (67,108,864 configurations, requiring a LTU of 8 megabytes). The 4D case exhibits 2^{80} configurations, which is intractable using a LTU, as it would need 128 zettabytes (128 billions of terabytes). Couprie and Bertrand have proposed a more general framework for checking for simple points using complexes [2] and the collapse operation. Intuitively, complexes can be seen as a generalization of graphs. An informal definition of a *simplicial complex* (or simplicial d-complex) is "a set of simplices" (plural of simplex), where a simplex or n-simplex is the simplest manifold that can be created using n points (with $0 \leq n \leq d$). A 0-simplex is a point, a 1-simplex a line segment, a 2-simplex a triangle, a 3-simplex a tetrahedron. A graph is indeed a 1-complex. Figure 1(a) shows an example of a simplicial complex. Likewise, a *cubical complex* or cubical d-complex can be thought as a set of n-faces (with $0 \leq n \leq d$) in

Algorithm 1: Breadth-First Thinning.

Data : E (a set of points/sites),
$X \subseteq E$ (initial set of points),
$K \subseteq X$ (a set of points (constraint) that cannot be removed)
Result : X
$P \leftarrow \{A \in X | A \text{ is simple for } X\}$
while $P \neq \emptyset$ **do**
  $Q \leftarrow \emptyset$
  for each $A \in P$ **do**
    if $A \notin K$ **and** A is simple for X **then**
      $X \leftarrow X \backslash A$
      for each $B \in N(A) \cap X$ **do**
        $Q \leftarrow Q \cup \{B\}$
  $P \leftarrow \emptyset$
  for each $A \in Q$ **do**
    if A is simple for X **then** $P \leftarrow P \cup A$

Algorithm 2: A generic version of Algorithm 1.

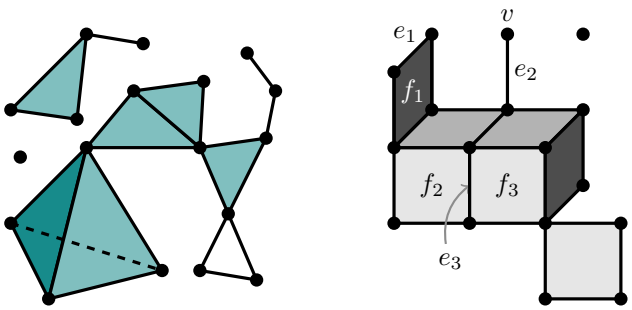
Data : E, $X \subseteq E$, N (neighborhood),
is_simple (a function saying if a point is simple),
detach (a routine detaching a point from X),
constraint (a function representing a constraint)
Result : X
$P \leftarrow \{A \in X |$ *is_simple*$(A, X)\}$
while $P \neq \emptyset$ **do**
  $Q \leftarrow \emptyset$
  for each $A \in P$ **do**
    if *constraint*(A) **and** *is_simple*(A, X) **then**
      $X \leftarrow$ *detach*(X, A)
      for each $B \in N(A) \cap X$ **do**
        $Q \leftarrow Q \cup \{B\}$
  $P \leftarrow \emptyset$
  for each $A \in Q$ **do**
    if *is_simple*(A, X) **then** $P \leftarrow P \cup A$

Algorithm 3: A generic C++ implementation of Algorithm 2 in Milena. Functors are highlighted.

```
template <typename I, typename N, typename F, typename G, typename H>
mln_concrete(I)
breadth_first_thinning(const Image<I>& input, const Neighborhood<N>& nbh,
                       Function_v2b<F>& is_simple, G detach,
                       const Function_v2b<H>& constraint) {
  mln_concrete(I) output = duplicate(input);
  is_simple.set_image(output); // Have 'output' be the subject of 'is_simple'.

  typedef p_set<mln_psite(I)> set_t; // Type of a set of points (sites).
  set_t set;
  mln_piter(I) p(output.domain());
  for_all(p) // ∀ 'p' ∈ 'output.domain()'...
    if (output(p) && constraint(p) && is_simple(p))
      set.insert(p); // Initialize 'set' with simple points from the border.

  while (!set.is_empty()) {
    set_t next_set;
    mln_piter(set_t) p(set);
    for_all (p)
      if (constraint(p) && is_simple(p)) {
        detach(p, output); // 'p' is simple, passes the constraint:  remove it.
        mln_niter(N) n(nbh, p);
        for_all(n) // ∀ 'n' in the neighborhood of 'p'...
          if (output.domain().has(n) // Prevent out-of-bound accesses.
              && output(n) && constraint(n) && is_simple(n))
            next_set.insert(n); // Add neighbor 'n' to the next candidate set.
      }
    set.clear(); swap(set, next_set);
  }
  return output;
}
```



(a) A simplicial 3-complex. (b) A cubical 2-complex.

Figure 1: Complexes.

$\mathbb{Z}^d$, like points (0-faces), edges (1-faces), squares (2-faces), cubes (3-faces) or hypercubes (4-faces). Figure 1(b) depicts a cubical complex sample.

Complexes support a topology-preserving transformation called *collapse*. An *elementary collapse* removes a free pair of faces of a complex, like the square face f_1 and its top edge e_1, or the edge e_2 and its top vertex v, in Figure 1(b). The pair (f_2, e_3) cannot be removed, since e_3 also belongs to f_3. Successive elementary collapses form a *collapse sequence* that can be used to remove simple points. Collapse-based implementations of simple-point deletion can be always be used in 2D, 3D and 4D, though they are less efficient than their LTU-based counterparts. On the other hand, they provide some genericity as the collapse operation can have a single generic implementation on complexes regardless of their structure.

Illustrations

Using this generic approach, Algorithm 3 can be used to compute the skeleton of a complex-based surface, as in Figure 2:

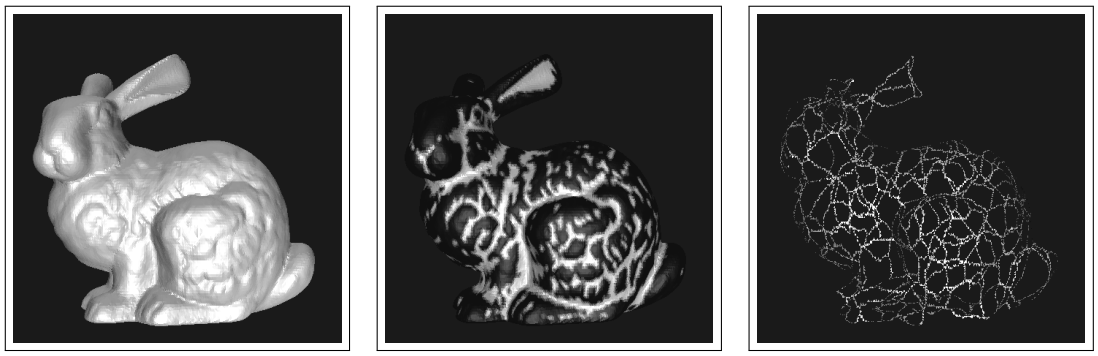
```
skel = breadth_first_thinning(surf, nbh, is_simple_cell(), detach(), tautology());
```

In the previous code, `surf` is a triangle-mesh surface representing X and `nbh` represents an adjacency relationship between triangles sharing a common edge. Function objects `is_simple_cell` and `detach` are operations compatible with `surf`'s type; they are generic routines based on the collapse operation working on any complex-based binary image. Finally, `tautology()` is a functor always returning the value `true`, so as to materialize a lack of constraint. Other input types (e.g. a 2D image on a regular square grid) can be processed similarly, by using compatible definitions for `nbh` (e.g. a 4-connectivity neighborhood), for `is_simple` (e.g. a 2D mask-based predicate) and for other arguments.

4 Conclusion

We have presented building blocks to implement reusable Digital Geometry algorithms in an image processing framework, Milena. Given a set of theoretical constraints on its inputs, an algorithm can be written once and reused with many compatible input types. This design has previously been proposed for Mathematical Morphology, and can be applied to virtually any image processing field. Milena is Free Software released under the GNU General Public License, and can be freely downloaded at `http://olena.lrde.epita.fr/Download`.

A strength of generic designs is their ability to extend and scale easily and efficiently. First, generic algorithms are extensible because of their parameterization. For instance, the behavior of Algorithm 3 can be significantly changed by acting on the simple point definition or the set of constraints. Secondly, the framework can be extended with respect to data structures. Milena



(a) Triangle mesh surface. (b) Surface curvature. (c) Surface skeleton.

Figure 2: Computation of a skeleton using breadth-first thinning. The triangle mesh surface 2(a) is seen as a simplicial 2-complex. The image of curvature 2(b) is computed on the edges of the mesh, and simplified using an area opening filter. All curvature regional minima are then removed from the mesh, and the skeleton 2(c) is obtained with Algorithm 3 using the collapse operation.

provides site sets based on boxes, graphs and complexes, but more can be added to the library (e.g. combinatorial maps, orders, etc.) and benefit from existing algorithms and tools. Finally, our approach can take advantage of *properties* of input types (regularity of the site set, isotropic adjacency relationship, etc.) and allow users to write specialized versions of their algorithms for such subsets of data types, leading to faster or less memory-consuming implementations.

Acknowledgments The authors thank Benoît Sigoure and the anonymous reviewers for their helpful comments on this paper.

References

[1] G. Bertrand and M. Couprie. Transformations topologiques discrètes. In D. Coeurjolly, A. Montanvert, and J.-M. Chassery, editors, *Géométrie discrète et images numériques*, chapter 8, pages 187–209. Hermes Sciences Publications, 2007.

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

[3] M. C. d'Ornellas and R. van den Boomgaard. 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, Mar. 2003.

[4] Th. Géraud and R. 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*, Paphos, Cyprus, July 2008.

[5] J. Lamy. Integrating digital topology in image-processing libraries. *Computer Methods and Programs in Biomedicine*, 85(1):51–58, 2007.

[6] R. Levillain, Th. Géraud, and L. 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, Aug. 2009.

[7] R. Levillain, Th. Géraud, and L. Najman. Why and how to design a generic and efficient image processing framework: The case of the Milena library. In *Proceedings of the IEEE International Conference on Image Processing (ICIP)*, Hong Kong, Sept. 2010. To appear.