A Generic and Parallel Algorithm for 2D Digital Curve Polygonal Approximation

Guillaume Damiand, David Coeurjolly

▶ **To cite this version:**

Guillaume Damiand, David Coeurjolly. A Generic and Parallel Algorithm for 2D Digital Curve Polygonal Approximation. Journal of Real-Time Image Processing, 2011, 6 (3), pp.145-157. 10.1007/s11554-011-0193-x . hal-00636402

HAL Id: hal-00636402
https://hal.science/hal-00636402

Submitted on 27 Oct 2011

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Journal of Real-Time Image Processing (2011) 6:145-157
DOI 10.1007/s11554-011-0193-x

Guillaume Damiand · David Coeurjolly

A Generic and Parallel Algorithm for 2D Digital Curve Polygonal Approximation

Received: 15 July 2009 / Accepted: 5 January 2011 / Published online: 15 February 2011

Abstract In this paper, we present a generic topological and geometrical framework which allows to define and control several parallel algorithms for 2D digital curve approximation. The proposed technique is based on combinatorial map simplifications guided by geometrical criteria. We illustrate the genericity of the framework by defining three contour simplification methods: a polygonal approximation one based an area deviation computation; a digital straight segments reconstruction one which guaranties to obtain a loss-less representation; and a moment preserving simplification one which simplifies the contours while preserving geometrical moments of the image regions. Thanks to a complete experimental evaluation, we demonstrate that the proposed methods can be efficiently implemented in a multi-thread environment to simplify labeled image contours.

Keywords combinatorial maps · parallel contour simplification · polygonal approximation · multi-thread image processing

1 Introduction

Image processing algorithms often need to compute, extract or analyze information contained in images. These information can be computed by decomposing an image into regions and by extracting topological and geometrical features from them. As a consequence of the discrete nature of images, region contours are composed of discrete curves (set of pixels or set of inter-pixel elements, see Klette and Rosenfeld (2004a)) whose topology can be complex (several connected components, junctions,...). If we want to efficiently represent all region contours, we are facing two difficulties. First, we need a data-structure to represent the topology of regions and thus to represent the topology of their contours. Then, we need a process to approximate the discrete contours with polygonal lines while preserving the geometry of the contours. In this paper we present a generic and parallel algorithm to simplify contours of a multi-region image using combinatorial maps and geometrical criteria.

In computer vision, many segmentation algorithm outputs are decompositions of the image into labeled regions such that pixels in a region have uniform features (image intensities, texture characteristics,...). To represent regions and to be able to perform efficient operations on them, a topological data structure is required in order to describes boundaries and adjacency information. There are many different structures to represent the region boundaries of a given image, the first one being the Region Adjacency Graph (RAG) (Rosenfeld 1974). However, a RAG does not describe all the information contained in the image (like multi-adjacencies or inclusion relations). To represent all the information, several models based on combinatorial maps were defined (Domenger 1992; Fiorio 1996; Brun and Domenger 1997; Damiand et al 2004). The main advantage of these models is to describe the subdivision of regions into cells (vertices, edges and faces) and to describe all the incidence and adjacency relations between these cells and thus to represent the topology of the image. Combinatorial maps allow us to link geometrical features to edges and regions, and to design a simple and generic contour simplification by removing combinatorial maps vertices. Using a shared memory parallel model, we show that the simplification algorithm can be easily parallelized by adding mutexes

Author manuscript, published in Journal of Real-Time Image Processing, 6(3), pp. 145-157, September 2011. Thanks to Elsevier. The original publication is available at http://dx.doi.org/10.1007/s11554-011-0193-x

G. Damiand
Université Lyon, CNRS, LIRIS, UMR5205, F-69622 Villeurbanne Cedex, France
Tel.: +33 (0)4.72.43.26.62
Fax: +33 (0)4.72.43.15.36
E-mail: guillaume.damiand@liris.cnrs.fr

D. Coeurjolly
Université Lyon, CNRS, LIRIS, UMR5205, F-69622 Villeurbanne Cedex, France
Tel.: +33 (0)4.72.43.82.40
Fax: +33 (0)4.72.43.15.36
E-mail: david.coeurjolly@liris.cnrs.fr

to some cells and by using the model to avoid concurrent access.

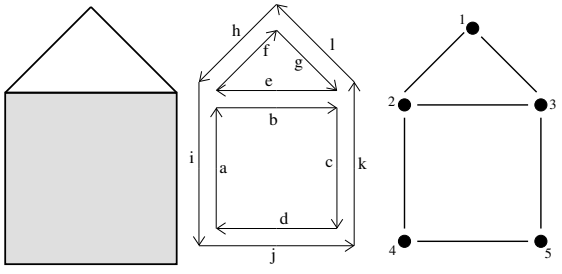
Once the topology of regions and contours has been handled, we need a simplification algorithm to approximate all the region contours with polygonal lines. For nearly half a century, a wide literature exists in the polygonal approximation of digital curves. Among all published surveys on the subject, Bhowmick and Bhattacharya (2007) provides an interesting classification of approximation techniques: there exist the curvature maxima based techniques which set vertices to high curvature loci (Teh and Chin 1988); the combinatorial optimization using ant colonies to minimize the number of edges and their location (Yin 2003); the perceptual organization based approaches (Hu and Yan 1997); the area deviation between the discrete curve and the approximation techniques (Wall and Danielsson 1984); and finally the approximation using Digital Straight Segments (DSS) which main advantage is that the representation is loss-less since the original curve can be retrieved from the digitization of the approximated curve (Klette and Rosenfeld 2004a). The algorithms in the first three approaches cannot be considered in our parallel multi-region framework since either they cannot be applied on complex digital curve topologies, or the approximation cannot be decided locally, making the parallelization more complex or impossible.

In this paper we illustrate our generic framework with three different geometrical criteria: First a polygonal simplification process based on a maximal distance threshold (area deviation based approach). Then we have also considered a digital reconstruction criterion based on DSS (Kovalevsky 1990; Dorst and Smeulders 1991; Lindenbaum and Bruckstein 1993; Debled-Rennesson and Reveillès 1995; Klette and Rosenfeld 2004a). The last criterion exploits the topological data structure to construct an approximation based on the region geometry preservation (using geometrical moments) instead of a region contour simplification process.

In section 2, we first introduce combinatorial maps. Section 3 details our two generic simplification algorithms, a sequential and a parallel one, taking a given criterion as parameter. In section 4, we present two contour based criteria: a distance-to-curve criterion, and a digital straight segment recognition, and in section 5 we give a region based criterion based on geometrical moments associated to regions. In section 6, we finally present some experiments and we conclude in section 7.

2 Combinatorial Maps Presentation

A combinatorial map is a mathematical model of space subdivision representation based on a planar map (Edmonds 1960; Tutte 1963). The subdivision of a 2D topological space is a partition of the space into 3 subsets whose elements are *cells* of 0, 1 and 2 dimension (respectively called vertices, edges and faces, and denoted i-cell for an i-dimensional cell). In 2D, a combinatorial map is equivalent to other structures such as the Winged-Edge data structure (Baumgart 1975) or the doubly-connected edge list (Preparata and Ian Shamos 1990; de Berg et al 2008). The main advantage of combinatorial maps is the generic definition which is valid in any dimension. This is not used in this work since we consider only 2D images, but allow us to plan extension of this work to 3D as explain in the conclusion.



	a	b	c	d	e	f	g	h	i	j	k	l
β_1	b	c	d	a	f	g	e	i	j	k	l	h
β_2	i	e	k	j	b	h	l	f	a	d	c	g

Fig. 1 2D combinatorial map example. (a) A 2D object. (b) Corresponding combinatorial map. Darts are represented by labeled black arrows (to represent the orientation). Two darts linked by β_1 are drawn consecutively, and two darts linked by β_2 are concurrently drawn and in reverse orientation. (c) Corresponding subdivision made of 5 vertices (labeled from 1 to 5), 6 edges and 3 faces (by counting the infinite face). We use sometimes this representation instead of drawing all the darts of the map as in (b) to make figures lighter.

Intuitively a 2D combinatorial map (or 2-map) is a decomposition of 2D objects into faces, edges and vertices. The basic element of a 2-map is called a *dart* (sometimes called half-edge in 2D). Each dart is incident to a vertex, an edge and a face. Darts are linked together with two one-to-one mappings β_1 and β_2 which describe the structure of the subdivision: β_1 connects one dart belonging to an edge to the dart of the next edge of the same face; β_2 connects one dart belonging to an edge to the dart of the other face of the same edge. Definition 1 is the definition of 2D combinatorial map (see Lienhardt (1991) and Fig. 1).

Definition 1 (2D combinatorial map) A 2D combinatorial map is a triplet $M = (D, \beta_1, \beta_2)$ where:

1. D is a finite set of darts;
2. β_1 is a *permutation*[1] on D;
3. β_2 is an *involution*[2] on D;

[1] A *permutation* on a set S is a one-to-one mapping from S onto S.

[2] An *involution* f on a set S is a one-to-one mapping from S onto S such that $f = f^{-1}$.

Combinatorial maps encode subdivisions and incidence relations between all the different cells of the space, and so represent the topology of this space. Thanks to this model and its implementation (Damiand et al 2004), we can directly (i.e. in $O(1)$) retrieve all the different relations associated to darts and one-to-one mappings. We denote $v(d)$ (resp. $e(d)$, $f(d)$) for the vertex (resp. edge, face) incident to dart d.

Moreover, several operations exist to modify a combinatorial map. The main operation used in this paper is the removal of an i-cell, called the i-removal operation, which removes the i-cell and merges the two incident *(i+1)*-cells. The i-removal operation is possible only for degree[3] one or two cells. Indeed, otherwise it is not possible to decide how to connect cells around the removed cell. Thanks to combinatorial maps, it is possible to test in $O(1)$ if the degree of a vertex is 2. Moreover, the removal of a vertex does not modify the degree of the other vertices (see (Damiand and Lienhardt 2003; Damiand et al 2004) for more details on removal operations and algorithms).

In Fig. 1, vertices 1, 4 and 5 are degree two vertices and can be removed, while vertices 2 and 3 are degree three vertices and thus cannot be removed.

In this work, we use a combinatorial map to describe regions contained in a *labeled image*. Such image can be the result of any segmentation algorithm (for example (Damiand and Resch 2003; Dupas and Damiand 2008)). Regions in a labeled image are the maximal sets of 4-connected pixels with same label. Note that we can consider any type of image and use the color of each pixel as a label. Regions are important for image processing since they contain colorimetric information about objects embedded in the image (mean color, texture characteristics, ...). Each region of the image is represented with a data structure added to the combinatorial map. This allows to easily add some information to regions. Moreover, each region is linked to a dart associated to the external boundary of the region, and all the darts of a region are linked with its belonging region. This allows to retrieve in constant time, given a dart, its belonging region, and given a region, we can retrieve in linear time all of its darts.

We can see in Fig. 2 an example of a 2D combinatorial map describing the labeled image shown in Fig. 3. Vertex 3 is a degree one vertex because it is incident to one edge (a self loop). Even if this vertex can be technically removed, its removal involves the lost of the boundary between region R_2 and R_3. Vertex 4 is also a degree one vertex, but this case is not possible when the combinatorial map describes a labeled image. Indeed, such a map represents boundary between regions, and these boundaries are closed curves. To summarize, vertices with degree greater than two can not be removed, and degree 1 vertices either must not be removed or does not exist: this ensures that only degree two vertices must be considered during our simplification algorithms.

[3] The degree of an i-cell c is the number of distinct *(i+1)*-cells incident to c.

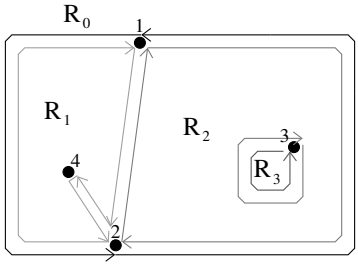


Fig. 2 An example of a 2D combinatorial map describing the labeled image shown in Fig. 3 containing three regions, plus an infinite region R_0 which is the complementary of the image (except the edge between vertices 2 and 4 which must not exist normally). Vertices 1 and 2 are degree three vertices and thus cannot be removed. Vertex 3 is a degree one vertex (it is incident to one edge). Lastly, vertex 4 is also a degree one vertex, but this case is not possible when the combinatorial map describes a labeled image.

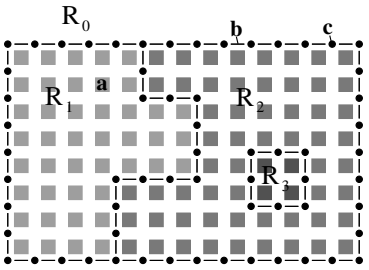


Fig. 3 Interpixel example. Pixels (for example a), linels (for example b) and pointels (for example c).

Combinatorial maps encode only the topology of the image. Geometry is described by using the cellular framework that decomposes the digital space pixels into linels, pointels and pixels: linels are unit one dimensional elements separating two pixels and pointels are zero dimensional elements between linels (see (Klette and Rosenfeld 2004a) and Fig. 3).

In the rest of the paper, we present several techniques to perform a labeled image contour simplification. All these processes are based on the same generic algorithm using combinatorial map data structure properties.

3 The Generic and Parallel Algorithms for Polygonal Approximation

We present now the generic algorithms allowing to simplify a given set of digital curves according to a given criterion. These algorithms use the high level representation of the curves by a combinatorial map. This allows to handle efficiently the vertices, the edges and the faces of the subdivision, and to use removal operations allowing to simplify the model while updating its proper-

ties. First we present a straightforward sequential algorithm to sketch the main principle of our method, then we present the parallel version which is a direct extension of the sequential one.

3.1 The sequential algorithm

The main principle of the sequential algorithm, presented in Algorithm 1, is first to compute a combinatorial map where each edge corresponds to a linel between two pixels belonging to two different regions (by using algorithm presented in Damiand et al (2004)). Then we scan the vertices of the map and remove each degree two vertex such that the geometrical criterion used for the polygonal approximation method is satisfied. When a vertex is removed, as detailed above, both incident edges are merged into a unique edge. The geometrical embedding of the new edge corresponds to the concatenation of two adjacent discrete curves into a single one.

Algorithm 1: Sequential polygonal approximation of contours of a labeled image.

Input: A labeled image I;
A criterion `criterion()`.
Output: A polygonal approximation of contours of I.

$M \leftarrow$ combinatorial map where each edge corresponds to a linel contour of I;
foreach *dart d of M* **do**
  if *the degree of $v(d)$ is 2* **and** `criterion(`d`)` **then**
    Remove $v(d)$;

The main loop of Algorithm 1 considers each dart successively, and processes only degree two vertices. Indeed, vertices with degree greater than 2 are at the junction of several branches and thus cannot be removed. Moreover, since the removal of a vertex does not modify the degree of the other vertices, we are sure that these vertices can not become removable later. This property ensures that during the simplification process, each vertex is tested exactly once.

When the current vertex is removed during the main loop, we jump over removed darts and continue the loop with the next dart. At the end of the algorithm, we have considered each vertex of the map, and we have removed the ones satisfying our merging criterion. This allows us to prove that the polygonal approximation is stable since no more vertex can be removed.

Fig. 4 illustrates each step of Algorithm 1: Fig. 4a is the input labeled image; Fig. 4b is the initial combinatorial map where each edge corresponds to a linel between two pixels with different labels, and Fig. 4c is the result obtained at the end of the algorithm where each edge corresponds to an approximation of the corresponding curve (by using the distance to curve criterion presented in section 4.1).

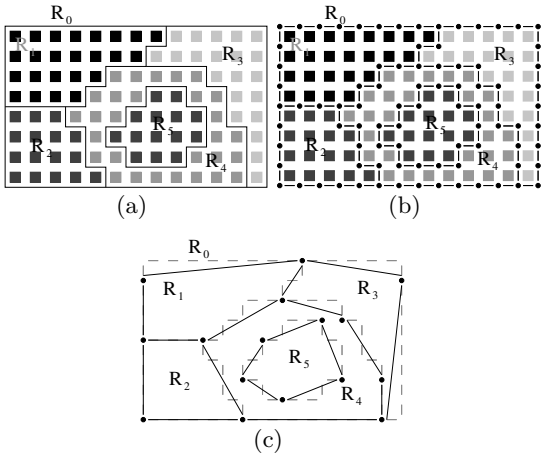


Fig. 4 Example of 2D discrete reconstruction of contours of a labeled image. (a) A labeled image with 5 regions (plus the infinite region R_0). (b) Initial combinatorial map where each edge of the map corresponds to a linel between two pixels with different labels. (c) Result obtained after the polygonal approximation. Each edge of this map corresponds to an approximation of the corresponding curve satisfying the given criterion.

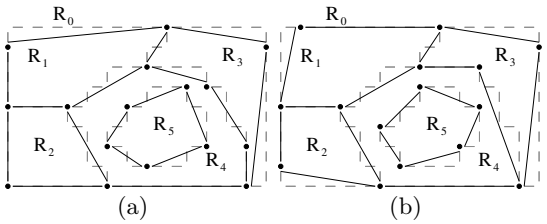


Fig. 5 Two possible results of 2D polygonal approximation of contours of the same labeled image (image given in Fig. 4a). Results can be different for the segment extremities, but can also be different in number of segments. (a) A result with 16 vertices, and 19 edges. (b) A second result with 15 vertices and 18 edges.

Note that the result of polygonal approximation depends on the order in which vertices are processed in the main loop of Algorithm 1. Indeed, each result is composed of a stable polygonal reconstruction in the sense that the union of two adjacent segments does not satisfy the given criterion. However, depending on the order of processed vertices, the extremities of segments can be different, and moreover the number of segments can also be different (see the example in Fig. 5). It is possible to add a pre-processing step in order to sort vertices and thus to add some properties on the resulting polygonal approximation. A discussion on this point is addressed in the conclusion.

The key point of Algorithm 1 is that each operation used in the main loop is a local process, and that each adjacency and incidence relations can be retrieved directly in $O(1)$ thanks to our model. Thus, the overall

computational cost of Algorithm 1 is linear in number of darts, times the cost of the geometrical criterion.

3.2 The parallel algorithm

The main idea of the parallel algorithm is to share the combinatorial map, to split the main loop of the sequential algorithm into several independent loops, and to execute each loop in parallel using threads. Thanks to the combinatorial map data-structure, the information associated to darts (degree, neighbors,...) can be retrieved in $O(1)$ in a thread-safe manner. The remaining bottleneck is the generic criterion computation and the removal of the considered dart. Since the algorithm is generic, we do not know yet the problems that can arise due to concurrent access of different threads. We avoid these problems by using the function `takeMutexes()` which must be defined according to each simplification method in order to guaranty correct access to the shared memory. Algorithm 2 details the pseudo-code executed by each thread independently.

Algorithm 2: Parallel polygonal approximation: per-thread algorithm.

Input: A combinatorial map M where each edge corresponds to a linel of the image contours;
L a list of darts to process;
A criterion `criterion()`;
A method entering in critical section `takeMutexes()`.
Output: Sub-part of M corresponding to L is modified such that each edge corresponds to a segment of a polygonal approximation.

```
Let P be a stack of darts to consider latter;
foreach dart d of L ∪ P do
    if the degree of v(d) is 2 then
        if takeMutexes(d) then
            if criterion(d) then
                Remove v(d);
            Release taken mutexes;
        else
            Push d in P;
```

The main loop of this algorithm is very similar to Algorithm 1 one. We can point out two main differences: first, the list of darts to process is now only a part of the whole darts of the map since other darts are processed by other threads. Secondly, we have added a particular process to avoid conflicts due to concurrent access. This is achieved by function `takeMutexes()`. As detailed in the sections 4 and 5, it is enough to protect the map locally with mutexes around the removed vertices.

The function return false if it was not able to take all the required mutexes. In such a case, the current vertex cannot be processed because the modifications made by another thread can modify the vertex, and thus we need to re-test this vertex later. For that, the vertex is pushed in a stack of vertices to be considered later. Note that mutexes are taken in non-blocking mode to avoid inter-blocking situations, and that the termination is guaranteed by using a total order on cells, as explained in the following sections.

4 Approximation with Contour Based Criteria

4.1 Distance-to-Curve Criterion

The first simplification process we present is based on a very simple contour polygonal approximation using a *distance-to-curve* criterion which corresponds to an adaptation of Wall and Danielsson (1984).

The criterion is given in function `criterionPolygon()`. It consists in computing the maximal distance between the edge obtained if we remove the vertex incident to the considered dart (the Euclidean segment $[v_1, v_2]$) and all the pointels of the corresponding curve in the image associated to edges $e(d)$ and $e(\beta_0(d))$ (see Fig. 6). The criterion is satisfied if the distance is smaller that a threshold specified by the user. To compute the distance between segment $[v_1, v_2]$ and the curve, we compute successively the distance between $[v_1, v_2]$ and each pointel belonging to both edges incident to d and to $\beta_0(d)$ (the two edges incident to vertex $v(d)$). The complexity of the function `criterionPolygon()` is linear in the length of the discrete curve between v_1 and v_2.

Function criterionPolygon(d: a dart, ϵ: a number):Boolean

```
Let v1 the vertex incident to β0(d) and v2 the vertex
incident to β1(d) ;
if distance-to-curve(e(d),[v1, v2])< ε and
distance-to-curve(e(β0(d)),[v1, v2])< ε then
    return true;
return false;
```

Since the processing of a vertex is limited to the two incident edges, the parallel algorithm needs to protect concurrent accesses on both edges incident to the current vertex. This is simply achieved by adding a mutex to each edge of the combinatorial map. Then, in the `takeMutexesPolygon()` procedure, we try to take the mutexes associated to edges e_1 and e_2. If both mutexes are taken, we have a guaranty that there is no other thread processing a vertex incident to either edge e_1 or e_2. As discussed in section 4.3, we use a total order on the edges to decide which mutex we take first (order $\prec$ and its associated min and max functions). The order can be

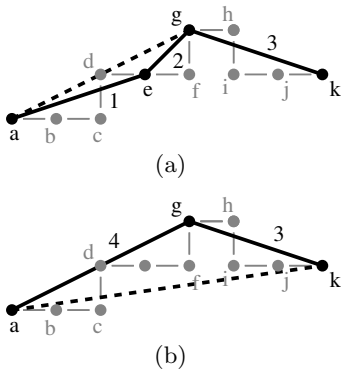


Fig. 6 Criterion used to allow or deny the removal of a vertex for the polygonal approximation. Pointels are labeled from a to k and edges are numbered from 1 to 4. (a) We try to remove the vertex incident to pointel e. We compute the maximal distance between segment $[a, g]$ and each pointel of edges $e(1)$ and $e(2)$ (i.e. pointels from a to g). If this distance is smaller than ϵ, the vertex is removed and edges $e(1)$ and $e(2)$ are merged into edge $e(4)$ in (b). (b) We want to remove vertex incident to pointel g, we compute the maximal distance between segment $[a, k]$ and pointels from a to k.

arbitrarily chosen and in our experiments, we have used the order induced by edge memory addresses.

Function takeMutexesPolygon(d: a dart):Boolean

```
Let e1 the edge incident to d (resp. e2 to β0(d));
Let e_inf = min≺(e1, e2) and e_sup = max≺(e1, e2);
Try to take mutex m_inf associated to e_inf;
if mutexes m_inf is taken then
    Try to take mutex m_sup associated to e_sup;
    if mutexes m_sup is taken then
        return true;
    else
        Release m_inf;
return false;
```

4.2 Discrete Straight Segment Reconstruction

To illustrate the genericity of the proposed simplification algorithm, we consider a second method of polygonal approximation using digital straight segment recognition. DSS recognition algorithms are widely used to convert a digital contour into a polygon (see Klette and Rosenfeld (2004a) or Klette and Rosenfeld (2004b) for a complete survey). Indeed, this class of algorithms provides several advantages compared to the contour approximation algorithm presented above: a DSS contains arithmetical structures allowing to speed up the recognition process using only integer number computations; and using

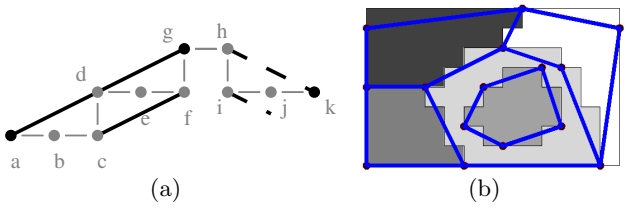


Fig. 7 Illustration of the DSS based contour simplification. (a) Both sequences $\{a, \ldots, g\}$ and $\{g, \ldots, k\}$ are DSS but the union test on these two DSS will fail. (b) Contour reconstruction defined by the DSS extermities.

DSS, we can derive a loss-less representation of the digital curve since the digitization of the obtained polygon exactly coincides with the input set.

In the literature, many DSS characterizations exist (Klette and Rosenfeld 2004a), we consider here a digital straight line as the set of pixels $(x, y) \in \mathbb{Z}^2$ satisfying:

$$\mu \leq ax - by < \mu + |a| + |b| \tag{1}$$

with $a,\ b, \mu \in \mathbb{Z}$. Hence, b/a is the DSS slope and μ its intercept. According to Reveillès (1991), the resulting set of pixels is a 4−arc, which means that each pixel of the DSL has exactly two 4-adjacent neighbors. A DSS is defined as a finite connected subset of a DSL. For example in Fig. 7a, both sequences $\{a, \ldots, g\}$ and $\{g, \ldots, k\}$ are DSS. Bold lines (dashed or not) represent the DSS leaning lines defined by $ax - by = \mu$ and $ax - by = \mu + |a| + |b| - 1$. Based on this definition, a recognition problem may arise: given a set of grid points, does there exists a DSS containing it ? In the literature, many algorithms have been proposed (Kovalevsky 1990; Dorst and Smeulders 1991; Lindenbaum and Bruckstein 1993; Debled-Rennesson and Reveillès 1995; Klette and Rosenfeld 2004a). In our framework, we consider a recognition process based on a union predicate: given two 4-adjacent DSS, we decide if the union of the two pixel sets forms a DSS. Indeed, thanks to the combinatorial map representation, DSS parameters are attached to darts and thus the access to the adjacent DSS parameters can be obtained in $O(1)$. Then, a naive algorithm to decide if the union of two DSS S and T (*i.e.* the union of the two grid point sets) is a DSS or not can be done in $O(|T|)$. Note that more complex algorithms using preimages (Lindenbaum and Bruckstein 1993) can be designed to obtain a computational cost in $O(\log(\alpha))$ where α corresponds to largest side of the bounding box containing S and T.

For this method, we use as criterion the following function `criterionDSS()`. Note that contrary to the previous criterion `criterionPolygon()`, the parameter ϵ has disappeared since it is embedded in the DSS definition. In our implementation, the computational cost of function `criterionDSS()` is equal to the cost of function `criterionPolygon()`.

In the initial map where each edge corresponds to a linel, we add an attribute to each edge containing the

Function criterionDSS(d: a dart):Boolean

let d_1 the dss associated to d (resp d_2 associated to $\beta_0(d)$);
if $d_1 \cup d_2$ *is a DSS* **then**
  return *true*;
return *false*;

straight line equation of the corresponding edge (parameters a, b and μ). This equation is easy to initialize since each segment of the initial map corresponds to a linel. Then, during the simplification algorithm, when we process vertex $v(d)$, we test if the union of the two edges incident to $v(d)$ is a DSS (these two edges are obtained from d and $\beta_0(d)$). When this criterion is satisfied, we remove $v(d)$ and update the DSS parameters of the new edge by modifying the corresponding attributes.

As for the polygonal approximation, we can prove that, at the end of the process, we obtain a stable decomposition into DSSs since the union predicate fails on each couple of remaining adjacent DSS. Fig. 7 illustrates the DSS based contour reconstruction.

For the parallel algorithm, we use Algorithm 2 and the criterion function `criterionDSS()`. The function `takeMutexes()` is the same than `takeMutexesPolygon()` because for both methods, modifications are only made locally and concern the two incident edges around the removed vertex.

4.3 Termination Analysis

In these contour based criteria, an infinite loop situation consists in a state where all threads fail taking their two mutexes and thus no vertex is processed. Such situation only occurs when a cycle of degree 2 vertices is considered, with one thread associated to each vertex of the cycle, and when each thread starts taking its *left* mutex (and succeeds). In this case, no thread will be able to take its *right* mutex (see example in Fig. 8). Thus, no vertex is processed at this step and the same configuration could occur in the remaining execution flow. As a consequence, the program may not terminate. In the other cases (chain of degree 2 vertices, one vertex of the cycle not associated to a thread, or a thread starting with its *right* mutex), we can demonstrate that at least one thread can process its vertex.

To avoid the pathological case, we define a total order of the edges (relationship $\prec$ in `takeMutexesPolygon()`). Using this order to take the two mutexes, we can ensure that at least one thread in the cycle can take its two mutexes (and thus process the point). Indeed, the situation where each thread takes its *left* mutex is not consistent with the total order and thus could never occur.

Conversely, the situation is the same if all threads take their *right* mutex first. Again, the $\prec$ order solves the problem.

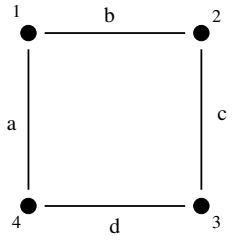


Fig. 8 Termination analysis: four threads working on the four vertices 1, 2, 3 and 4. Each thread $i \in \{1, 2, 3, 4\}$ succeeds to take its *left* mutex: $(1 - a)$, $(2 - b)$, $(3 - c)$, $(4 - d)$. If $a \prec b \prec c \prec d$ and if we have $(1-a)$, $(2-b)$, $(3-c)$, then thread 4 tries to take a but fails and thus `takeMutexesPolygon()` returns *false* on 4 (and 3 can take d and process its vertex).

5 Approximation with a Region Based Criterion

The main idea of this third method of contour simplification is to preserve geometrical characteristics of the regions with the help of geometrical moments: geometrical moments associated to each region of the topological map are computed and controlled during the contour simplification.

5.1 Geometrical moments

In many computer vision or shape modeling applications geometrical moments have been widely used to provide a powerful tools to describe shape geometry (Teh and Chin 1988; Mukundan and Ramakrishnan 1998). Considering a compact domain Ω on $\mathbb{R}^2$, the geometrical moment m_{pq} of order p and q is defined as follows:

$$m_{pq} = \int\int_{\Omega} x^p y^q dx dy \,. \tag{2}$$

From equation 2, we can first observe that m_{00} is the area of the domain Ω and $(\frac{m_{10}}{m_{00}}, \frac{m_{10}}{m_{00}})$ its center of gravity. Note that geometrical moments m_{pq} may not be relevant for shape description purposes since they are not invariant to basic transformation such as translation or scaling. However, from the geometrical moments, other invariant moments can be derived. For example, central moments (Mukundan and Ramakrishnan 1998) are invariant to translation and can be computed as polynoms of m_{pq} moments. In shape matching, complex moments can be designed to achieve rotational invariance. For example, Zernike moments (Teh and Chin 1988; Novotni and Klein 2004) can be defined by linear combination of geometrical moments.

When the domain Ω is either discrete (subset of $\mathbb{Z}^2$) or defined by a polygonal boundary, a fast geometrical moment computation algorithm can be designed. First, let us consider a triangular domain T with vertices $\{(0,0), (x_1, y_1), (x_2, y_2)\}$ on which we want to integrate a function $f : \mathbb{R}^2 \to \mathbb{R}$. Using a linear mapping of the

triangle vertices to the unit triangle T_0 vertices $\{(0,0)$, $(1,0)$, $(0,1)\}$, we have:

$$\iint_T f(x,y)dxdy = det\begin{pmatrix} x_1 & x_2 \\ y_1 & y_2 \end{pmatrix} \int_0^1 \int_0^{1-Y} g(X,Y)dXdY \quad (3)$$

with $g(X,Y) = f(x_1X + x_2Y, y_1X + y_2Y)$.

Hence, using an analytical evaluation of the integration of the geometrical moments over T_0, we obtain the following formulas to compute the geometrical moments of T:

$$m_{00} = \frac{1}{2}(x_1y_2 - y_1x_2) \quad (4)$$
$$m_{10} = \frac{(x_1y_2 - y_1x_2)\ (x_2 + x_1)}{6}$$
$$m_{01} = \frac{(x_1y_2 - y_1x_2)\ (y_2 + y_1)}{6}$$
$$m_{11} = \frac{(x_1y_2 - y_1x_2)\ ((2\,x_2 + x_1)\ y_2 + y_1\,x_2 + 2\,x_1\,y_1)}{24}$$
$$\dots$$

If we consider now a polygonal domain P, a classical approach to compute the integration of f over P is to sum the contributions of the integration of f over all triangles $\{(0,0)$, (x_i,y_i), $(x_{i+1},y_{i+1})\}$ $(i = \{0,\ 1,\ \dots, n\}\ mod\ n)$. Thanks to the determinant in equation 3, as depicted in Fig. 9, the sign a contribution depends on the edge orientation. Such an evaluation framework has been widely used to evaluate integrals over polygonal domains, such as geometrical moments (Sheue-Ling-Chang 1984) or Spherical Harmonics (Mousa et al 2006).

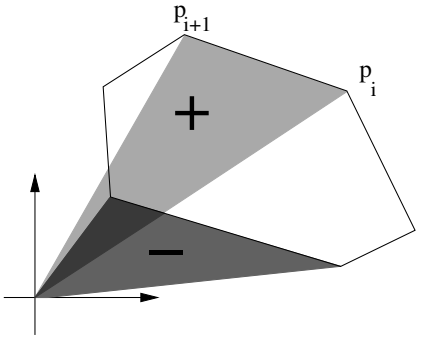


Fig. 9 Integral computation over a polygonal domain based on triangular contributions.

In our framework, in order to preserve the geometrical characteristics of the labeled image regions, the idea is to define a contour simplification algorithm controlled by geometrical moment variations. We can sketch the algorithm formalized below as follows: to decide if a vertex is removed, the error is evaluated in terms of geometrical moment changes of the adjacent regions.

5.2 Moment based simplification algorithm

For this method, contrary to both previous ones, attributes are associated to regions and not to edges. Indeed, each region of the topological map will contain the set of moments we consider. The criterion which allows or denies to remove a vertex needs here to compute the new moment of both regions incident to the current vertex (there are always two regions since we process only degree two vertices).

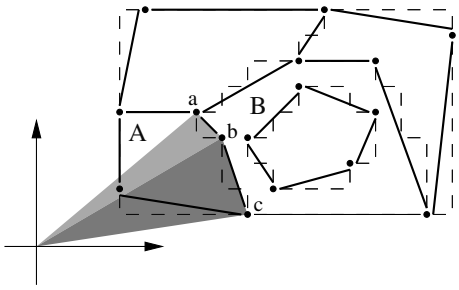


Fig. 10 Moment based contour simplification.

Let us consider the example given in Fig. 10: let us suppose that we decide to remove the vertex b. Hence both regions A and B change leading to regions A' and B'. If we denote T_{uv} the triangle $\{(0,0), u, v\}$ and m^P_{pq} the geometrical moment of order p, q of the domain P, it follows from the additivity of the geometrical moments that

$$m^{A'}_{pq} = m^A_{pq} - m^{T_{ab}}_{pq} - m^{T_{bc}}_{pq} + m^{T_{ac}}_{pq} \quad (5)$$
$$m^{B'}_{pq} = m^B_{pq} - m^{T_{ab}}_{pq} - m^{T_{bc}}_{pq} + m^{T_{ac}}_{pq} \quad (6)$$

Hence, the quantity $\delta_{pq} = -m^{T_{ab}}_{pq} - m^{T_{bc}}_{pq} + m^{T_{ac}}_{pq}$ can be viewed as an error measurement induced by the removal of b. Note that δ_{pq} and thus $m^{A'}_{pq}$ and $m^{A'}_{pq}$ can be computed in $O(1)$ thanks to equations 4.

The overall simplification process can now be formalized as follows: we first fix the number of moments we want to control and we compute the analytic formulas of equation 4. Then, we compute the geometrical moments of the initial image regions. This step can be performed without increasing the computational complexity during the topological map extraction. Then, to decide if a given vertex of the map can be removed, we construct a predicate based on a threshold on the ratio between updated moments and initial ones (see function `criterionMoment()`).

In a computational point of view, the criterion is evaluated in $O(1)$ since the topological information (adjacent regions) can be retrieved from the map in $O(1)$ and since the moments can be updated in $O(1)$ if the number of moments is fixed.

In the parallel method, to avoid problems due to concurrent access, mutexes need here to be linked to regions (see function `takeMutexesMoment()`). Otherwise,

Function criterionMoment(d: a dart, τ: a percentage):Boolean

```
Let r1 the region incident to d (resp. r2 incident to β2(d));
Let m1 the moments of r1 when removing vertex v(d)
(resp m2 of r2);
foreach moment m of m1 (resp. m2) do
    if m differs more than τ percent from
    corresponding original moment of r1 (resp. r2)
    then
        return false;
return true;
```

if two concurrent threads process two vertices incident to a same region, both criteria can return true to allow both removals because each modification is smaller than the threshold while the sum of the two moment modifications is greater than the threshold. Adding mutexes onto regions avoid this situation because the concurrent threads can not process two vertices incident to a same region.

Function takeMutexesMoment(d: a dart):Boolean

```
Let r1 the region incident to d (resp. r2 to β2(d));
Let r_inf = min≺(r1, r2) and r_sup = max≺(r1, r2);
Try to take mutex m_inf associated to r_inf;
if mutexes m_inf is taken then
    Try to take mutex m_sup associated to r_sup;
    if mutexes m_sup is taken then
        return true;
    else
        Release m_inf;
return false;
```

This shows another interest of our approach which allows to use any cells of the cellular decomposition (vertices, edges and faces), and depending on the needs of each application, we can add different attributes onto different cells.

5.3 Termination Analysis

Similarly to section 4.3, termination problem may occur when thread dependencies during the non-atomic `takeMutexesMoment()` process induce a cycle. Again, we have used an arbitrary total order on regions ($\prec$ in `takeMutexesMoment()`) to break the cycle and ensure the termination of the simplification.

Table 1 Image characteristics: *Size* is the size of the image in number of pixels (x and y), *Regions*, *Vertices* and *Edges* are respectively the number of regions, of vertices and edges of the initial combinatorial map corresponding to the image.

	Size	Regions	Vertices	Edges
`Airplane`	512×512	1689	28201	29481
`Baboon`	512×512	6291	81050	85647
`Cornouaille`	256×256	1177	15828	16797
`Goldhill`	720×576	4705	66484	69858
`Lamp`	256×256	1633	16736	18040
`Lena`	512×512	3118	39577	41937
`Peppers`	512×512	3402	35338	38120
`Table`	256×256	698	12622	13171

6 Experiments

To evaluate our generic framework for labeled image contour simplification, we have set up several experiments. First, we evaluate the genericity in terms of time efficiency of the algorithm parallel versions compared to sequential ones. Secondly we demonstrate the scale-up property considering high resolution images. Lastly, we evaluate the criteria in terms of visual quality.

Our experiments were made with an "Intel Core i7 CPU" at 2.80GHz, with 12 giga-bytes of memory and 8 mega-bytes of cache. This processor has four cores and uses hyper-Threading technology, allowing to run eight threads simultaneously. We have used eight segmented images: `Airplane`, `Baboon`, `Cornouaille`, `Goldhill`, `Lamp`, `Lena`, `Peppers`, `Table` (see Fig. 11). These images are classical images used in image processing, segmented with a basic region growing algorithm.

Table 1 presents all the characteristics of these images (size, number of regions) and the number of vertices and edges of the initial map (where each edge corresponds to a linel). The number of darts is not given since it is twice the number of edges. Table 2 gives the times taken by our methods (values are means of ten tests for each image). For the polygonal approximation method, we have used a threshold $\epsilon = 1.0$, and for the moment preserving method a percentage of modification allowed $\tau = 5\%$.

First, we can remark that our methods are very efficient. Indeed, the time required for each method is very small: the worst case is in 40.15 milliseconds to simplify `Baboon` with the sequential moment methods. `Baboon` is the worst case image because it is composed with many small regions which avoid contour simplifications. Secondly, we can remark that in general, the discrete reconstruction method and the polygonal simplification are faster than the moment preserving simplification. The moment method is slower than both other methods due to the computation of moments: there are six moments to compute for three edges, and moments used complex formula.

We can observe that the speed-up of the parallel method is interesting: about 42% for the polygonal ap-

Fig. 11 The eight images used in our experiments: `Airplane`, `Baboon`, `Cornouaille`, `Goldhill`, `Lamp`, `Lena`, `Peppers` and `Table`.

Table 2 Times in milliseconds (10^{-3} seconds) taken to compute the polygonal approximation with the sequential (*poly seq.*) and parallel (*poly par.*) method, to compute the discrete reconstruction with the sequential (*dss seq.*) and parallel (*dss par.*) algorithm, and to compute the simplification preserving moments with the sequential (*mom seq.*) and parallel (*mom par.*) algorithm. The three rows *speed-up* give the percentage of speed-up of the parallel method on the sequential one for each type of simplification, the three rows *push* give the number of vertices pushed in the stack of elements to reconsider, and the three rows *rem. vertices* give the number of removed vertices for each method (means of ten tests).

	`Airpl.`	`Baboon`	`Cornou.`	`Goldhi.`	`Lamp`	`Lena`	`Peppers`	`Table`
poly seq.	7.14	25.28	3.42	19.86	3.6	10.98	9.41	2.57
poly par.	3.65	11	2.27	8.8	2.61	5.05	4.66	2.16
poly speed-up	48.86%	56.46%	33.70%	55.67%	27.62%	54.04%	50.46%	16.05%
poly push	58	50	47	48	42	57	44	53
rem vertices	21,597	58,557	11,857	48,740	11,951	29,115	25,210	10,179
dss seq.	7.7	28.77	3.45	22.82	3.74	12.08	10.44	2.49
dss par.	3.29	10.14	2.02	8.16	2.26	4.59	4.21	1.86
dss speed-up	57.26%	64.77%	41.59%	64.24%	39.52%	61.97%	59.67%	25.14%
dss push	58	44	52	45	38	56	46	37
rem vertices	21,405	57,735	11,694	48,065	11,790	28,688	24,861	10,037
mom seq.	11.05	40.15	5.55	31.11	5.88	17.6	15.08	3.68
mom par.	9.01	31.84	2.75	20.28	3.11	6.66	6.05	2.5
mom speed-up	18.44%	20.68%	50.45%	34.79%	47.18%	62.15%	59.88%	32.09%
mom push	657,693	2,556,919	20,237	1,459,085	14,545	87,108	73,450	19,350
rem vertices	20,344	50,907	10,749	44,806	10,129	26,465	22,170	9,555

proximation, 51% for the discrete reconstruction method and 40% for the moment preserving method. First, we must notice that the gain for parallel methods is related to the time required by the sequential one. Indeed, fast sequential algorithms are really difficult to improve. This is for example the case for `Cornouaille`, `Lamp` and `Table` images, which are small images of sizes 256×256.

Secondly, this speed-up must be considered relatively to the number of conflicts during the simplification. The rows *poly push*, *dss push* and *mom push* in table 2 give the number of vertices pushed in the stack of dart to reconsider (a vertex is pushed in this stack when the thread was not able to take the two required mutexes). We can observe than the number of conflicts is really small for the two first methods (polygonal and DSS simplifications), while these numbers are important for the moment method. Indeed, for the two first methods, mutexes are local since they correspond to the two edges incident to the current vertex, while for the moment method, mutexes correspond to the two regions incident to the current vertex, which prohibits the process of many other vertices.

This explain why we have for example a speed up of 20% for the image `Baboon`, and 62% for the im-

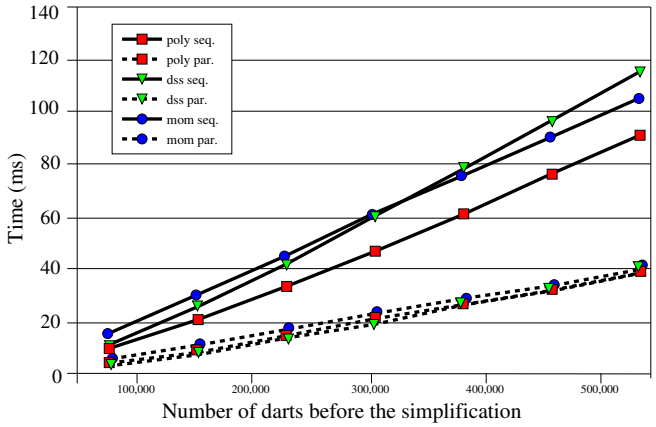


Fig. 12 Time (in milliseconds) taken by the three contour simplification methods in sequential and parallel for the image peppers, by increasing its size from 512×512 to 3584×3584.

Table 3 Slopes of the linear regression between computation times and size of combinatorial maps in number of darts for our 6 methods: polygonal, DSS and moments for sequential and parallel methods (slopes are multiply by 10^5).

Polygonal		DSS		Moments	
Seq.	Par.	Seq.	Par.	Seq.	Par.
18	7,6	23	7,7	19,7	7,45

age `Lena` for the moment method. For `Baboon`, we obtain better results by decreasing the number of threads. We have 33.23ms with two threads, 26.8ms with four threads, and 31.84ms with eight threads. These times are related to the number of conflicts: 1 101 506 for two threads, 12 528 498 for four threads, and 25 569 191 for eight threads. Using four threads seems in this case to be a good compromise between number of vertices to treat by each thread, and number of conflicts.

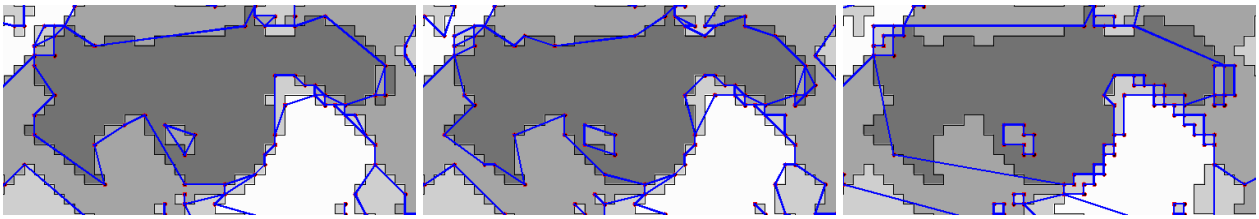
We have also considered an experiment in order to study the scale-up property of our algorithms. For that, we have taken the `Peppers` image, and multiply its size by 2, 3, 4, 5, 6 and 7. Thus, we obtain 7 images of size from 512×512 to 3584×3584. We have run our three simplification methods in sequential and in parallel onto these 7 images. Results are presented in Fig. 12. We can first remark that even for large images, our methods are really quick: less than 120 milliseconds for image of size 3584×3584 for the DSS method. Secondly, we can observe that despite the fact that the worst case complexity is not linear for the first two methods, the time efficiency behaves linearly. As expected, Fig. 12 also illustrates the speed-up between sequential and parallel methods. This is confirmed while studying the slopes of the linear regression between computation times and size of images given in table 3. We can observe that slopes are smaller for parallel methods than for sequential ones: this shows that the speed-up of parallel methods comparing to sequential ones increases for bigger images.

To evaluate the results of the proposed simplification methods, Fig. 13 shows three zooms onto the left eye of `Lena`. We can observe that the contours obtained with the DSS method describe more precisely the original regions than contours obtained with the polygonal approximation. This can be verified in table 2 by looking at the number of removed vertices: there are always less removed vertices for DSS method than for polygonal one. We can also see that the moment based method preserves more precisely the small region contours while allowing more important modifications on large region contours. Indeed, in this method, the modifications allowed are a percentage of the original moments, and thus for small regions, removing a pixel involves, in percentage, a bigger modification than for a large region. This can be also verified in Fig. 14 for experiments on other classical images (`Soldier` and `Bird`). The DSS method gives results with more vertices than both other approaches, due to reversibility. The moment preserving method simplify more strongly small details due to the merging criterion which is given in a percentage of the whole region. This shows once again the interest of our generic approach allowing to easily propose a new criterion mixing local and global features depending on the need of a particular application.

7 Conclusion

In this paper, we have presented a generic algorithm which allows to simplify the contour of any labeled image. By using a combinatorial map which describes the image, we use cells of the subdivision to add some parameters, and we use incident and adjacency relations to merge two edges when it is possible. This gives a simple and efficient algorithm. Moreover, since all processes are local, the algorithm can easily be parallelized with independent threads sharing the same map. Our experiments show that the speed-up of the parallel algorithm is about 45% with a "Intel Core i7 CPU CPU". Since our algorithm is fully parallel, the speed of our method will be improved by using future generation of processors with more cores. Furthermore, thanks to the time efficiency evaluation presented above, we have shown that a real-time multi-region contour simplification can be obtained.

As a consequence the genericity of our framework, we have presented three simplification methods: a first one which is a polygonal approximation of contours, a second one which is a discrete polygonalization and a third one which is a moment preserving simplification. These three methods illustrate the interest of using combinatorial map because we use different type of cells (edges for polygonal approximation and for discrete reconstruction and regions for moment preserving simplification) and several adjacency and incidence relations.



(a) (b) (c)

Fig. 13 Zoom around the left eye of `Lena` for the three contour simplifications: (a) Polygonal approximation, (b) Discrete reconstruction and (c) Moment preserving simplification.

An advantage of our method is the possibility to process vertices in any order. This can be easily achieved by adding a step which sorts vertices of the initial combinatorial map in a specific order. For example, we can use random orders to obtain results which can be statistically studied. This order of vertices must also be studied in order to improve the parallel methods by decreasing the number of conflicts.

Another future work could be to schedule the vertices to process considering geometrical information. For example, we could order points according their estimated curvature measurement and then start the simplification process with low curvature value vertices. Moreover, we can easily add the notion of critical points (vertices that cannot be removed) by marking these particular vertices and do not process them during our algorithm. Lastly, we plan to extend this method in 3D by using 3D combinatorial maps (Damiand 2008).

Acknowledgments

A preliminary version of this work, describing only the discrete segment recognition, was presented at the International Symposium on Visual Computing, Las Vegas, NV, 2008, LNCS, Springer, Special Track on Discrete and Computational Geometry and their Applications in Visual Computing (Damiand and Coeurjolly 2008).

References

Baumgart B (1975) A polyhedron representation for computer vision. In: Proc. AFIPS, vol 44, pp 589–596

de Berg M, Cheong O, van Kreveld M, Overmars M (2008) Computational Geometry: Algorithms and Applications. Springer

Bhowmick P, Bhattacharya BB (2007) Fast polygonal approximation of digital curves using relaxed straightness properties. IEEE Transactions on Pattern Analysis and Machine Intelligence 29(9):1590–1602

Brun L, Domenger JP (1997) A new split and merge algorithm with topological maps and inter-pixel boundaries. In: Proc. WSCG

Damiand G (2008) Topological model for 3d image representation: Definition and incremental extraction algorithm. Computer Vision and Image Understanding 109(3):260–289

Damiand G, Coeurjolly D (2008) A generic and parallel algorithm for 2d image discrete contour reconstruction. In: Proc. ISVC, Springer, Las vegas, Nevada, USA, LNCS, vol 5359, pp 792–801

Damiand G, Lienhardt P (2003) Removal and contraction for n-dimensional generalized maps. In: Proc. DGCI, Naples, Italy, LNCS, vol 2886, pp 408–419

Damiand G, Resch P (2003) Split and merge algorithms defined on topological maps for 3d image segmentation. Graphical Models 65(1-3):149–167

Damiand G, Bertrand Y, Fiorio C (2004) Topological model for two-dimensional image representation: definition and optimal extraction algorithm. Computer Vision and Image Understanding 93(2):111–154

Debled-Rennesson I, Reveillès JP (1995) A linear algorithm for segmentation of digital curves. International Journal on Pattern Recognition and Artificial Intelligence 9:635–662

Domenger J (1992) Conception et implémentation du noyau graphique d'un environnement 2d1/2 d'édition d'images discrètes. Thèse de doctorat, Université Bordeaux I

Dorst L, Smeulders AWM (1991) Decomposition of discrete curves into piecewise straight segments in linear time. In: Proc. VG, Contemporary Mathematics, vol 119, pp 169–195

Dupas A, Damiand G (2008) First results for 3d image segmentation with topological map. In: Proc. DGCI, Springer Berlin/Heidelberg, Lyon, France, LNCS, vol 4992, pp 507–518

Edmonds J (1960) A combinatorial representation for polyhedral surfaces. Notices of the American Mathematical Society 7

Fiorio C (1996) A topologically consistent representation for image analysis: the frontiers topological graph. In: Proc. DGCI, Lyon, France, LNCS, vol 1176, pp 151–162

Hu JM, Yan H (1997) Polygonal-approximation of digital curves based on the principles of perceptual organization. Pattern Recognition 30(5):701–718

Klette R, Rosenfeld A (2004a) Digital Geometry: Geometric Methods for Digital Picture Analysis. Series in Computer Graphics and Geometric Modelin, Morgan Kaufmann

Klette R, Rosenfeld A (2004b) Digital straightness–a review. Discrete Applied Mathematics 139(1-3):197–230

Kovalevsky V (1990) New definition and fast recognition of digital straight segments and arcs. In: Proc. ICPR, pp 31–34

Lienhardt P (1991) Topological models for boundary representation: a comparison with n-dimensional generalized maps. Computer-Aided Design 23(1)

Lindenbaum M, Bruckstein A (1993) On recursive, $o(n)$ partitioning of a digitized curve into digital straigth segments. IEEE Transactions on Pattern Analysis and Machine In-

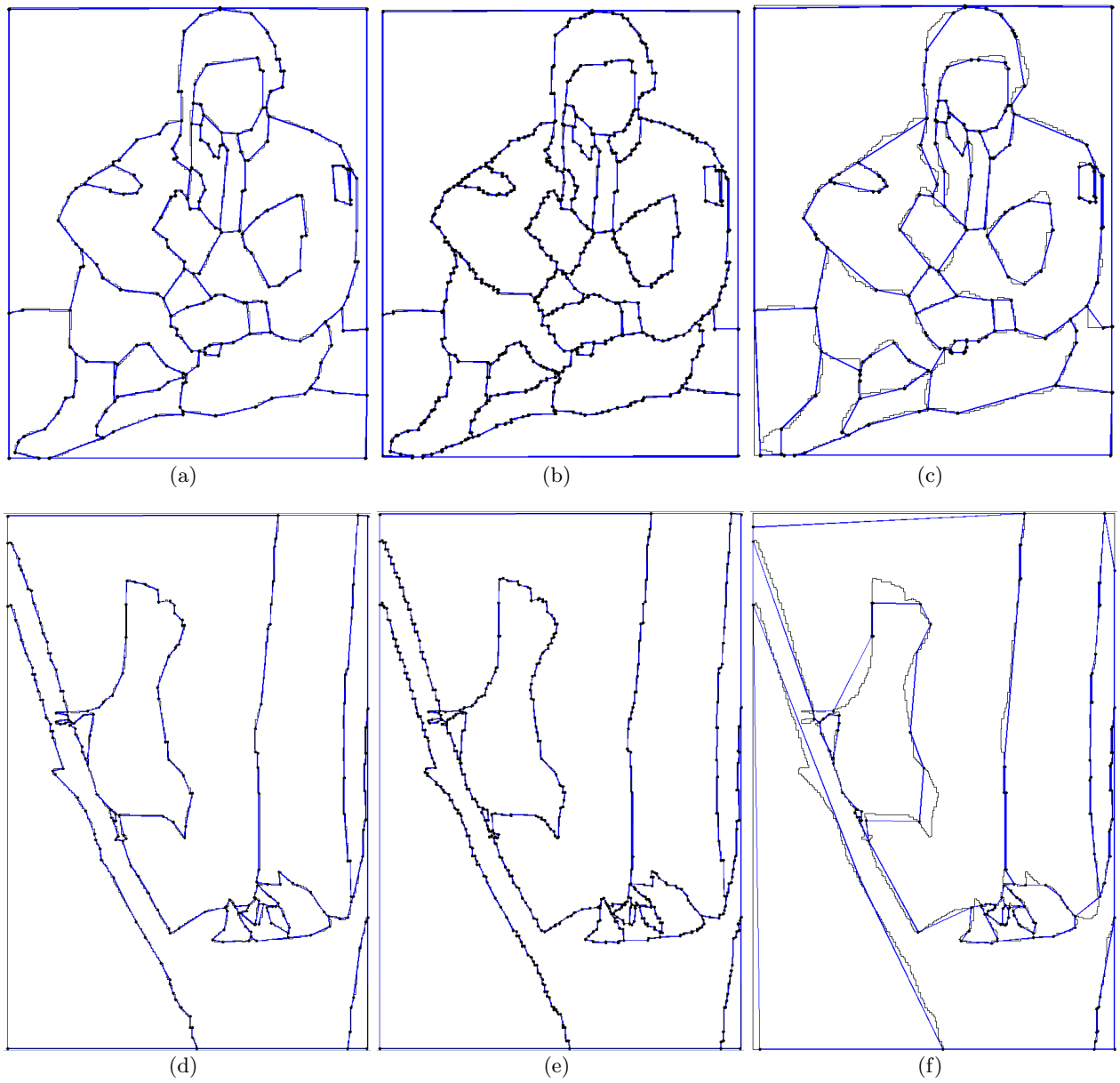


Fig. 14 More experiments on different images. First line (a) to (c) for `Soldier` image, initially with 4465 vertices. (a) Result for distance to curve: 215 vertices ($\epsilon = 1.8$). (b) Result for DSS reconstruction: 574 vertices. (c) Result for moment preserving approximation: 156 vertices ($\tau = 5\%$). Second line (d) to (f) for `Bird` image, initially with 5625 vertices. (d) Result for distance to curve: 227 vertices ($\epsilon = 1.8$). (e) Result for DSS reconstruction: 546 vertices. (f) Result for moment preserving approximation: 144 vertices ($\tau = 5\%$).

telligence 15(9):949–953
Mousa M, Chaine R, Akkouche S (2006) Direct spherical harmonic transform of a triangulated mesh. Journal of graphics tools 11(2):17–26
Mukundan R, Ramakrishnan KR (1998) Moment Functions in Image Analysis, Theory and Applications. World Scientific
Novotni M, Klein R (2004) Shape retrieval using 3D zernike descriptors. Computer-Aided Design 36(11):1047–1062
Preparata FP, Ian Shamos M (1990) Computational Geometry: An Introduction. Springer-Verlag
Reveillès JP (1991) Géométrie discrète, calcul en nombres entiers et algorithmique. PhD thesis, Université Louis Pasteur - Strasbourg
Rosenfeld A (1974) Adjacency in digital pictures. Information and Control 26(1):24–33
Sheue-Ling-Chang L (1984) Combining computation with geometry. PhD thesis, California Institute of Technology, Pasadena, Californie, Etats-unis
Teh CH, Chin R (1988) On image analysis by the methods of moments. IEEE Transactions on Pattern Analysis and Machine Intelligence 10(4):496–513

Tutte W (1963) A census of planar maps. Canadian Journal of Mathematics 15:249–271
Wall K, Danielsson PE (1984) A fast sequential method for polygonal approximation of digitized curves. Computer Vision, Graphics and Image Processing 28:220–227
Yin PY (2003) Ant colony search algorithms for optimal polygonal approximation of plane curves. Pattern Recognition 36(8):1783–1797