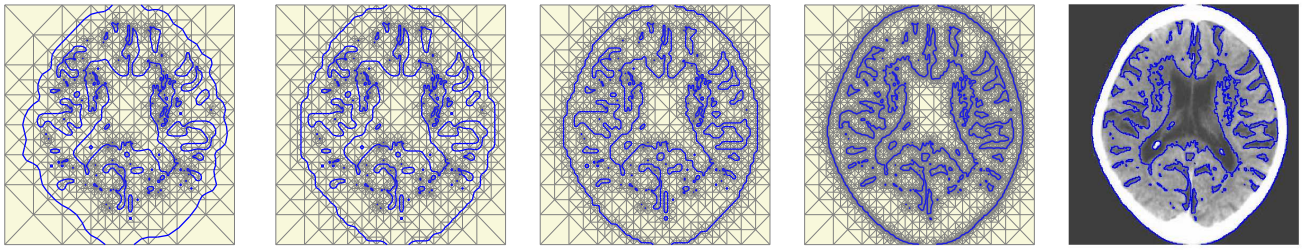
Extraction and compression of hierarchical isocontours from image data

THOMAS LEWINER[1,2], HÉLIO LOPES[1], LUIZ VELHO[3] AND VINÍCIUS MELLO[3]

[1] Department of Mathematics — Pontifícia Universidade Católica — Rio de Janeiro — Brazil
[2] Géométrica Project — INRIA – Sophia Antipolis — France
[3] Visgraf Project — IMPA — Rio de Janeiro — Brazil
{tomlew, lopes}@mat.puc--rio.br. {lvelho, vinicius}@visgraf.impa.br.

Abstract. In this work, we introduce a new scheme to extract hierarchical isocontours from regular and irregular 2D sampled data and to encode it at single rate or progressively. A dynamic tessellation is used to represent and adapt the 2D data to the isocontour. This adaptation induces a controlled multi–resolution representation of the isocontour. This representation can then be encoded while controlling the geometry and topology of the decoded isocontour. The resulting algorithms form an efficient and flexible isocontour extraction and compression scheme.
Keywords: ***Level sets. Data Compression. Simplicial Methods. Progressive Transmission. Geometry Processing.***



(a) 764 bytes (b) 853 bytes (c) 1089 bytes (d) 2011 bytes (e) 5369 bytes

Figure 1: *Topology controlled extraction of a Computerized Tomography image of the cortex, and progressive compression.*

1 Introduction

Curves are one of the basic building blocks of geometry processing. They are used to represent shape in 2D images, terrain elevation on maps, and equations in mathematical visualization. In most of those applications, the curves can be interpreted as an isocontour of a 2D dataset, possibly mapped on a more complex space. Those isocontours are flexible objects that can be refined or reduced, that can deform with differential simulations or mathematical morphology, and that can be described for shape classification or automatic diagnostic in medicine or geosciences.

Problem statement. Given the sampling $\hat{f}$ of scalar function f defined over a domain D embedded in $\mathbb{R}^2$ (such as a 2D image or a discrete surface), the *isocontour* of an *iso*value α is the curve $f^{-1}(\alpha)$. Such an isocontour corresponds to only a small part of D, but usually covers a large area of the domain. For example, the cortex corresponds to only specific x–ray scintillation inside the scan of the whole head (see Figure 1), the elevation curve is only a small part inside a topographic map (see Figure 20). Therefore, specific compression techniques for isocontours should provide better compression rate than the encoding of the entire 2D data.

Contributions. In this paper we introduce a new method for extracting and compressing isocontours based on a dynamic tessellation of the 2D data. This structure shows very nice adaptation properties, allowing extraction of the isocontour with different level of details. The main idea is to encode the tubular neighbourhood of the isocontour extracted from different levels of detail of the tessellation. Moreover, the adaptation the tessellation can depend on the isocontour, providing to our compression scheme a full control on the geometry and topology of the decoded isocontour. The resulting algorithms are flexible, can handle irregular 2D data, single–rate and progressive transmission together with uniform and adapted refinements.

2 Related work

In this work, we will use dynamic adaptive triangulations to represent and encode two–dimensional isocontours. This section describes some relevant works related to dynamic adaptive tessellations, and hierarchical isocontour extraction and isocontour compression.

Preprint MAT. 21/05, communicated on August 15^{th}, 2005 to the Department of Mathematics, Pontifícia Universidade Católica — Rio de Janeiro, Brazil. The corresponding work was published in Computerized Medical Imaging and Graphics, volume 30, number 4, pp. 231-242. Elsevier 2006..

Adaptive Tessellations. Hierarchical data structures are traditionally used for progressive compression and visualization of images. The usual representation for images relies on a rectangular grid that is subdivided uniformly or adaptively with a quad–tree. However, these structures are restricted to rectangular data sets. The size of those rectangles reduces twice as fast as the sizes of triangles in triangular tessellations, resulting in less adaptability. We will therefore focus on triangulations. [19] introduced multi–triangulations as a general concept for adapted variable resolution simplicial structures. [17] developed a binary multi–resolution structure based on stellar operators, which is a multi–triangulation with optimal properties. [18] proposed SGS, a Slow Growing Subdivision scheme for tetrahedral meshes that employs the refinement mechanism of a binary multi–triangulation. [23] proposed a very simple scheme for dynamically adapting triangulations while maintaining regularity conditions.

Hierarchical Isocontour Extraction. A hierarchical representation of an isocontour can be obtained by reduction of the polygonal curve of a single isocontour: [6] introduced the first algorithm for reduction of the polygonal approximation of a curve in the plane. Since then, this algorithm has been extended and improved in many aspects (see [24] for guarantees on the consistency of the reduced curve).

Nevertheless, this hierarchy can be obtained by extracting isocontours at each level of detail of a multi–resolution representation of the 2D data [20, 15]. The approximation of complex implicit curves usually requires robust computation, whose result can be seen as an isocontour. Hierarchical structures such as quad–trees usually provide simple and efficient solutions [14]. Similarly, the hierarchical representation of the image data can be adapted to the specified isocontour. For example [11] provides a hierarchy of rectangles to represent the isocontour, using genetic algorithm to optimize the dimensions of the rectangles. Those hierarchical representations are numerous when dedicated to a specific application, in particular for shape analysis [16, 3] and compression [5].

Isocontour compression. Isocontour compression is usually done as a compression of a non–self–intersecting curve, for example as two separate signals for each coordinate, or as a vector displacement [4, 21]. It can also be compressed by the popular chain code when the curve points are limited to pixel quantization [8]. In that case, it can be compressed as a 2D signal [10, 2]. [9] introduced another concept by encoding a hierarchical representation of the isocontour induced by a multi–resolution of the 2D data. The progression tries to maintain the chessboard distance from the original curve to the encoded one. The coarser resolution is encoded as two separate signals, and the position of the points introduced by the refinements are encoded as a difference with the near-by points.

3 Overview

In this work, we intend to encode a hierarchy of isocontours by their tubular neighbourhood. The tubular neighbourhoods are extracted from an adapted triangulation of the original 2D data. The data structure we will use to represent the 2D data is the one of [17, 23]. We adapt it to the neighbourhood of the isocontour, and reduce it according to the isocontour topology, geometry and position inside the triangulation. This structure allows to a very simple multi–resolution isocontour extraction, and enables us to compress the curve at single rate or progressively. Moreover, it is well suited for uniform and adapted progression on both regular and irregular 2D data. It can prevent topological changes or high geometric distortion during the progression. Finally, it works with sub–pixel interpolation for the curve, which enables smooth curve reconstruction at any level of detail.

Paper outline. We will introduce the adaptive triangulation we used in section 4 *Adaptive Tessellation*. This structure can represent regularly sampled data with the same amount of sample points as the classical quad-tree representation, but it can also adapt to irregular data. It also provides simple and effective controls on the topology and geometry of the isocontour as explained in section 5 *Isocontour Extraction*. Our compression scheme is introduced in section 6 *Isocontour Compression* and section 7 *Arithmetic encoding*, and the results are showed in section 8 *Results*.

4 Adaptive Tessellation

The multi–resolution structure we will use here is the Regular Binary Multi–Triangulation (RBMT) [17, 23]. This structure is constructed using stellar operators on edges and can decompose adaptively the 2D data. These decompositions can be regarded as hierarchies of conforming triangulations. In this section, we will give a brief description of the Stellar operators on edges and the RBMT.

(a) Stellar Operators

Stellar theory [1] studies the equivalences between simplicial complexes (i.e., a generalization of a triangulation) and defines combinatorial operators that change these complexes while maintaining their topology, integrity and coherence with the modelled object. The building blocks of these operators are the *stellar subdivision* and the *stellar simplification*.

The stellar subdivision operation in a triangulation inserts a vertex into an edge σ. The inverse of this operation is called *stellar simplification*, and removes the *welded vertex* w modifying its *star* (see Figure 4). Stellar theory states that these stellar operators are sufficient to map any two equivalent triangulated manifolds [1].

(b) Binary Multi–Triangulation

The Binary Multi–Triangulation (BMT) is a multi–resolution structure based on stellar subdivision on edges. When subdividing an edge, its incident triangles are sub-

The corresponding work was published in Computerized Medical Imaging and Graphics, volume 30, number 4, pp. 231-242. Elsevier 2006..

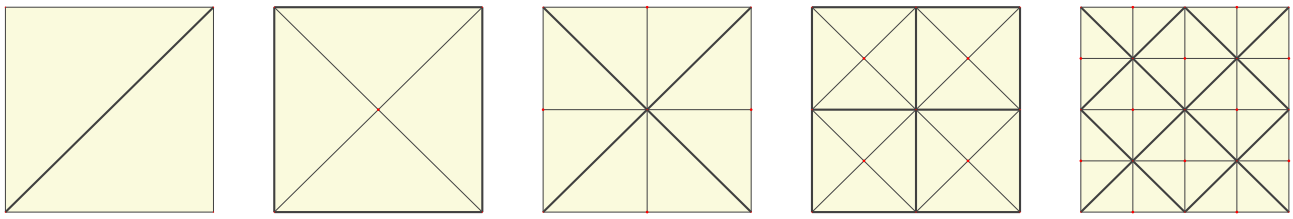


Figure 2: *BMT example: T^i can be transformed into T^{i+1} by subdividing one of the subdivision edges (in bold).*

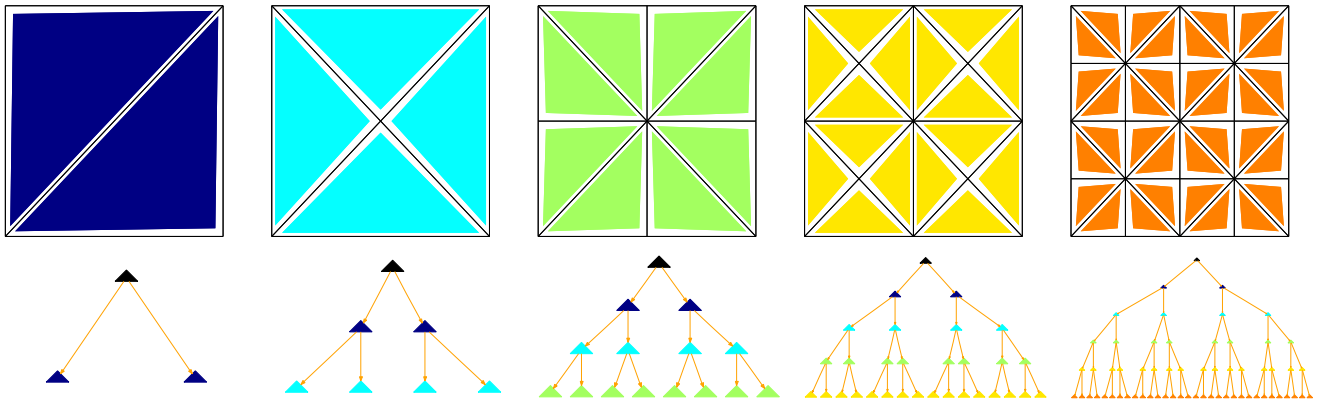


Figure 3: *The BMT example of Figure 2 can be represented as a binary tree.*

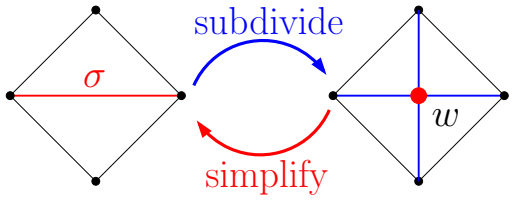


Figure 4: *Subdividing an edge $\sigma \leftrightarrows$ simplifying the welded vertex w. The two triangles on the right are the star of σ, while the four triangles on the left are the star of w.*

divided in two. Therefore, a sequence of subdivisions on edges can be represented as a binary tree structure, where each node represents a triangle and the two sons of a node t are the two triangles obtained by subdividing t (see Figure 3). This binary tree (actually a forest) adapts more nicely than the classical quad-tree for image decomposition, since it provides twice more intermediate levels. We will call the *level* of a triangle its depth in the binary tree. This property is close to the partially ordered set representation of [7], which provides a general framework for adaptive multiresolution structures. Moreover this binary tree structures leads to very simple algorithmic formulations.

The BMT is *reduced* or *refined* by walking up and down the binary tree, creating a hierarchy of triangulations at different *resolutions*. We perform those operations efficiently by maintaining for each triangle t, the vertex w that has been inserted during the subdivision that created t. The vertex w is called the *simplification vertex* of t, and the edge opposite to w is called the *subdivision edge* of t. Figure 2 shows a BMT whose root is a the two triangles of the square (T^0) and whose leaves uniformly covers a grid (T^{21}). The bold edges are the subdivision edges, and the bold vertices are the simplification vertices.

(c) Adaptation Properties and Regularity

The adaptability of the BMT comes from the possibility to refine and reduce the triangulation locally, while maintaining dependencies between adjacent triangles. In particular, we will maintain gradual transitions by preventing two adjacent triangles from differing by more than one level. With this restriction, the resulting structure is called a *regular binary multi–triangulation* (RBMT). This type of structure is also called a restricted hierarchy [22].

For example, Figure 5 illustrates a sequence of refinements adapting the triangulation to the bold line. In order to preserve gradual transition between resolution levels, local refinements around the bold line propagates inside the triangulation (in this example, far away from the bold line), as what happens to the bottom left part. The corresponding binary tree is represented on Figure 6.

Notice that the resulting modifications of the binary tree are not local. Therefore, it is necessary to propagate a subdivision or simplification to adjacent triangles. This propagation of a subdivision on an edge e is performed by checking

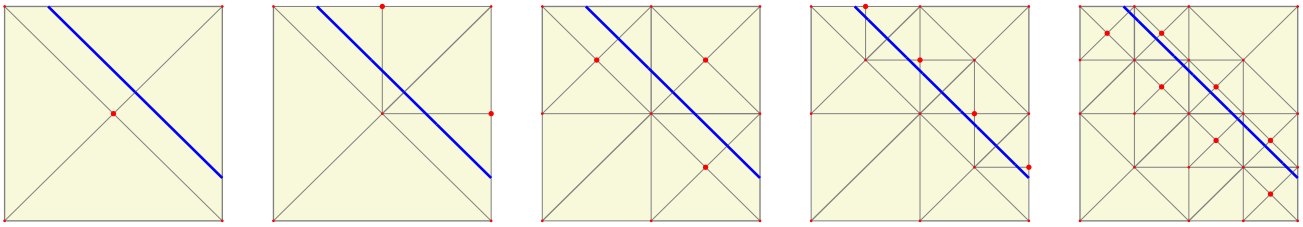


Figure 5: *RBMT adapted to the bold line: at each level, every triangle crossing the bold line is refined, but subdivisions in the upper–right part of the square propagate to the lower–left part.*

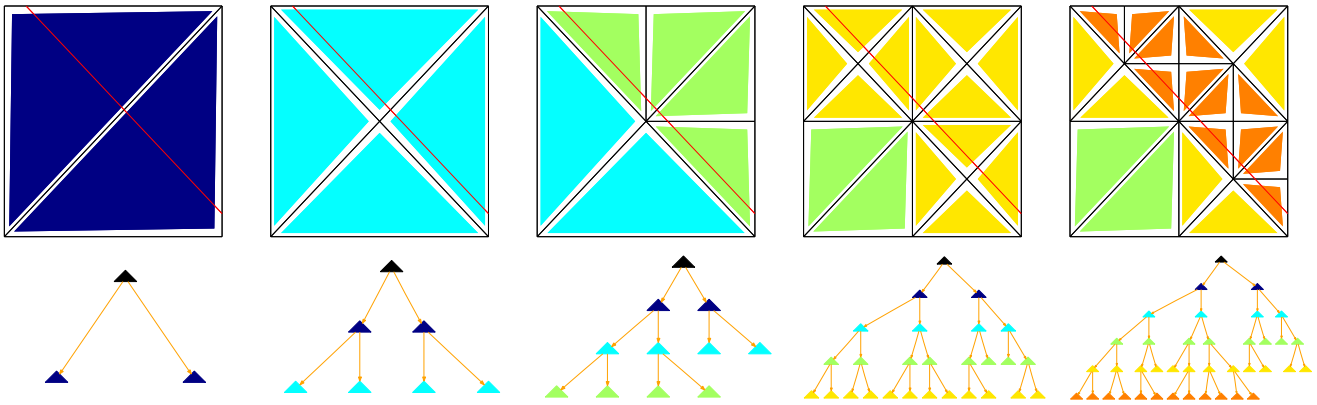


Figure 6: *The BMT example of Figure 2 can be represented as a binary tree.*

that each triangle t adjacent to e has e as a subdivision edge. If it is not the case, a subdivision is performed on the subdivision edge of t. This subdivision can require other triangles to be subdivided if they do share their subdivision edges. The propagation of a simplification on a vertex is done in a similar way (see Algorithm reduce).

In such a structure as the RBMT, subdividing a triangle means subdividing its subdivision edge, while simplifying it means simplifying its simplification vertex. Adaptive decomposition is then achieved by performing restricted refinement and simplification to strengthen and preserve a certain criterium (see Algorithm reduce), such as topology, curvature or decoding distortion as detailed in the next section (see Figure 9).

5 Isocontour Extraction

The *2D data* is given as a collection of samples v_i, each of which is associated with its *isovalue* $\hat{f}(v_i)$. Those samples are triangulated. In particular, when the samples are regularly spaced on a 2D grid (a gray–scale image for example), this triangulation can be automatically generated by the subdivision of a two–triangles square.

(a) Interpolation in a Triangulation

Assuming that we want to obtain the isocontour corresponding to the isovalue α, a sample of the 2D data is classified as *positive* or *negative* depending whether its isovalue is greater than α or not. An edge of the triangulation is *crossing*

Algorithm 1 reduce(criterion) : reduces the RBMT according to criterion

```
1:  repeat                                   // until the propagation stabilizes
2:    changed ← false                        // until now unchanged
3:    for all triangles t do                 // test the criterion on all triangles
4:      w ← t.welded_vertex                  // used to check if t can be simplified
5:      for all triangles t' ∈ w.star do     // front of the propagation?
6:        if t'.welded_vertex = w then       // t' must be simplified before t
7:          next t                           // will process t after
8:        end if
9:      end for
10:     if criterion(w) then                 // t must be simplified
11:       simplify(t)                        // simplify t
12:       changed ← true                     // unchanged the structure
13:     end if
14:   end for
15: until not changed                        // repeat until the propagation stabilizes
```

The corresponding work was published in Computerized Medical Imaging and Graphics, volume 30, number 4, pp. 231-242. Elsevier 2006..

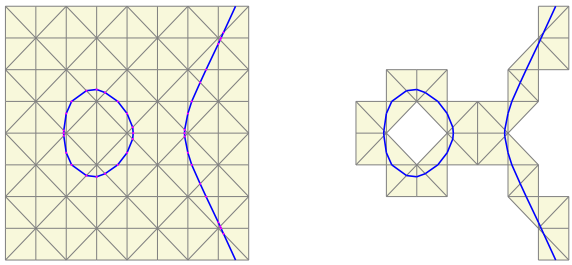


Figure 7: *A regular BMT simplified to preserve the tubular neighbourhood of an isocontour.*

when its end–points have opposite signs. A triangle is crossing when it contains a crossing edge. The *tubular neighbourhood* of the isocontour is the set of all the crossing triangles (see Figure 7).

The isocontour is approximated by a polygonal line linking the *isocontour points* interpolated at each crossing edge of the tessellation. In the case of a gray–scale image, this corresponds to a linear sub–pixel interpolation (as the purple points of Figure 7), although this extra precision does not generate extra cost during the compression. The samples of the 2D data will be referred as the *vertices* of the triangulation, as opposed to the *points* of the isocontour. More

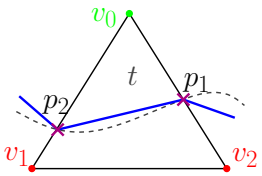


Figure 8: *Interpolation of the isocontour.*

precisely, the isocontour is computed as a polygonal curve. Each crossing triangle t of the RBMT contains a segment $[p_1, p_2]$ of the isocontour. The extremities p_1 and p_2 of the segment are linearly interpolated on the two crossing edges v_0v_1 and v_0v_2 of t (see Figure 8):

$$p_i = \lambda \cdot v_0.\mathsf{position} + (1-\lambda) \cdot v_i.\mathsf{position}$$

with

$$\lambda = \frac{v_i.\mathsf{isovalue} - \alpha}{v_i.\mathsf{isovalue} - v_0.\mathsf{isovalue}} \quad .$$

If the isocontour is extracted from an image, the sub-pixel interpolation can generate an ambiguity. This ambiguity is solved (somehow arbitrarily) by the tessellation, since only triangular elements are interpolated. The resulting isocontour is controlled redundantly by the isovalue of the vertices (gray level of the representing pixels) and by the size and position of the edge. This double control provides more flexibility on the isocontour progressive compression.

(b) Extraction from a RBMT

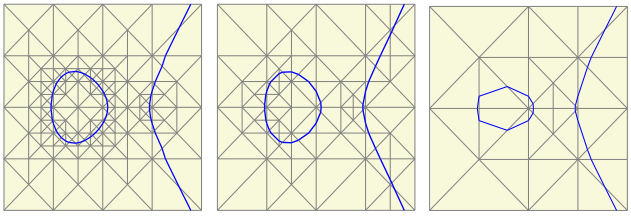
As an example of our formalism, we can already reduce the RBMT to represent with less triangles the tubular neighbourhood of the isocontour. This can be done using Algorithm reduce with the criterion function of Algorithm crossing. Figure 5 was generated using such a criterion on the different steps of Figure 2.

Algorithm 2 crossing(w) : test if w can be simplified without altering the tubular neighbourhood

```
1: for all edges e ∈ w.star do          // edges adjacent to w
2:   if e.crossing then                  // e is crossing
3:     return false                      // simplification would alter the
       tubular neighbourhood
4:   end if
5: end for
6: return true                           // simplification is safe
```

Now, this operation does not alter the tubular neighbourhood, and thus will leave the isocontour unchanged. Other criterion functions will change the interpolation of the isocontour, generating different representations for it. These representations correspond to the polygonal interpolations for different resolutions of the RBMT. These resolutions will be obtained by successive reductions of the finest level.

(c) Controls on the Extraction



(a) curvature (b) distortion (c) topology

Figure 9: *The isocontour of Figure 7 adapted according to (a) the curvature, (b) the distortion and (c) the topology.*

The reductions can be uniform, simplifying all triangles whose level is greater than a given threshold l (see Figure 2): $\mathsf{criterion}(w) = w.\mathsf{level} > l$. It can also be adapted to preserve the topology of the curve, or to preserve small triangles where the curve has a high curvature, and to reduce the triangulation where it is more flat (see Figure 9(a)). In a similar way, during compressing, we will be able to minimize the distortion of the curve (see Figure 9(b)) and to preserve its topology (see Figure 9(c)).

Geometry control. The criterion function of Algorithm crossing can be easily modified to provide a more flexible control on the isocontour. In particular, a curve is well approximated by a segment when its curvature is close to zero. Therefore a very simple geometric control will prohibit simplifications of triangles containing highly

Algorithm 3 curvature$_\kappa(w)$: test if w is adjacent to a highly curved part of the curve

```
1: n ← 0 ; k ← 0   // number of segments ; total curvature
2: for all edges e ∈ w.star do        // edges adjacent to w
3:    if t.crossing then                    // e is crossing
4:       n ← n+1 ; k ← e.curv       // adds local curvature
5:    end if
6: end for
7: return k > κ· n                          // curvature threshold
```

curved isocontour. Other geometric criterion can be used in a similar way, using Algorithm curvature as a model.

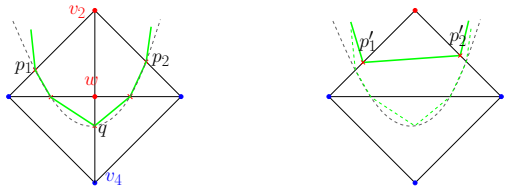


Figure 10: *The simplification of w induces a distortion that can be measured as* $\max\{d(q,p'_1);d(q,p'_2)\}$.

Distortion control. The distortion induced by a simplification of a vertex w can be estimated as the distance between the curve before and after the simplification. To perform a vertex simplification, the star of w needs first to be composed of only four triangles, which simplifies the cases to study to compute this distortion Figure 10: the distortion is estimated as the maximal distance between q and p'_i. In particular for coding, this distance can take into account the quantization error of the decoder, in order for the encoder to compute exactly the final distortion of the decoder: the points of the reduced curve will be the interpolation of the isocontour on the remaining edges, with the isovalues of the vertices quantized according to the compression tuning.

Topology control. Similarly, the topology of the isocontour can be easily controlled during the simplification of a vertex w. Actually, there are only two prohibited simplifications for a vertex inside the RBMT, and similar ones on the boundary (see Figure 11 and Algorithm topology):

1. The destruction of a connected component occurs when all the four edges incident to w are crossing.

2. The separation of a connected component in two or the merge of two connected components happen when the subdivision edge v_0v_1 is not crossing, while wv_0 and wv_1 were crossing edges.

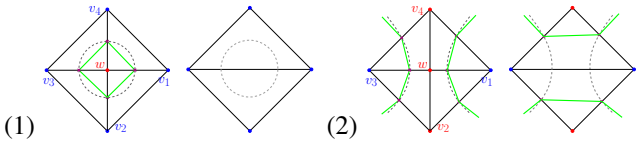


Figure 11: *There are two cases where the simplification of w induces a change on the topology: (1) destruction of a connected component, (2) exchange of local connected components.*

Algorithm 4 topology(w) : test if the simplification of w would alter the topology of the isocontour (valence 4 case)

```
1: (e1, e2, e3, e4) ← edges of w.star   // e2 ∪ e4 is the subdivision edge
2: (c1, c2, c3, c4) ← (e1.cross, e2.cross, e3.cross, e4.cross)
3: if (c1, c2, c3, c4) = (true, true, true, true) then
4:    return false   // (1) destruction of a connected component
5: end if
6: if (c1, c2, c3, c4) = (true, false, true, false) then
7:    return false   // (2) exchange of local connected components
8: end if
9: return true                          // simplification is safe
```

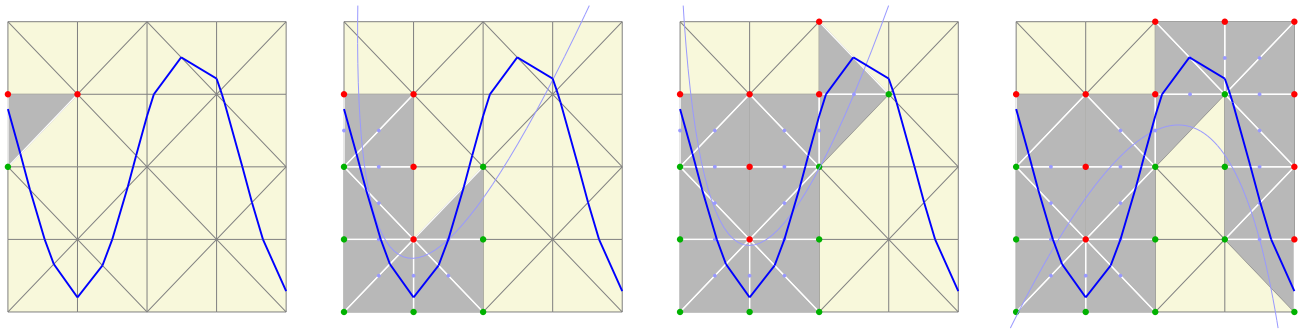
6 Isocontour Compression

The techniques we are now to introduce perform both direct and progressive encoding of the tubular neighbourhood of an isocontour. Moreover, the encoding of this neighbourhood can be done uniformly (i.e., the triangles of the tubular neighbourhood of the isocontour have constant size, see Figure 12) or adaptively (i.e. the size of the triangles varies according to the contour see Figure 14).

This section is organized as follows. We will introduce our method for encoding the tubular neighbourhood of the isocontour first uniformly and then adaptively. These methods can be used to encode the coarser resolution of the progression, or to encode the entire isocontour at single rate. Then, we will detail our techniques to encode the refinements, used for the progressive compression. Again, this process can be done uniformly or adaptively. Finally, we will explain the geometry encoding, and describe how to reuse the compressed data of one isocontour to encode other near–by ones on the same scalar data.

(a) Coarser Resolution

Our main idea is to encode the tubular neighbourhood going along with the isocontour, from one crossing triangle to an adjacent one in the tubular neighbourhood. The algorithm encodes first the localization of an unvisited crossing triangle t_0, and the signs of its vertices. From this initial triangle, it follows the isocontour, encoding the sign of each unvisited vertex encountered. When the traversal is done, it continues on the next connected component. Notice that the vertices of



(a) Location: level 4, 1, $L\ R\ L\ R$. $+\ -\ -$. (b) Vertex signs: $--+++++$. (c) 2 known vertices, then: $-\ +\ -$. (d) $----++-+-$ End.

Figure 12: *Uniform encoding of the coarser resolution of a small sinusoid. The light curve is a second–order fitting of the decoder's points (in the middle of the crossing edges), and serves as geometrical predictor.*

the only tubular neighbourhood have their isovalue encoded, leaving our algorithm almost independent of the initial size of the 2D data size.

Algorithm 5 send localize(t_0) : send localization of t

```
1:  t ← t_0                                   // temporary variable
2:  stack ∈ {0,1}^ℕ                           // stack of the positions of the
    ancestors of t_0
3:  while t.parent ≠ ∅ do                     // not reached the root
4:      stack.push(t.parent.left = t)         // stores as Left (true)
        or Right (false)
5:      t ← t.parent                          // one level up
6:  end while
7:  send bit(t = first)                       // encodes the root of the binary
    tree
8:  while stack ≠ ∅ do                        // pops the stack
9:      send bit(top.push)                    // encodes the Left or Right
        symbol
10: end while
```

Localization. The location of a triangle t_0 can be encoded using the binary tree inherent to the RBMT (see Algorithm send localize). The root of the tree contains only a few triangles: 2 for a regular grid. The triangle containing t_0 is encoded by its index. Then, knowing the level l of the triangle to encode, it can be localized using a sequence of l symbols Left and Right (see Figure 12(a), Figure 14(a) and Figure 14(d)). The signs of its vertices are then encoded and all of them are marked as visited; the initial triangle t_0 is also marked as visited.

Uniform encoding. Once the signs of the vertices of the initial triangle t_0 have been encoded, its crossing edges are known to the decoder. The first one is chosen to be the first gate e_0 (see Figure 13). The triangle t_1 ($\neq t_0$) incident to the gate e_0 is also crossing (see Algorithm next triangle). The sign of the vertex v_1 opposite to e_0 is encoded, and both t_1 and v_1 are marked as visited. The algorithm then continues the same way starting as t_1 (see Algorithm receive coarse uniform). The traversal ends when both triangles incident to the gate are marked, or when the gate is a boundary edge of the RBMT.

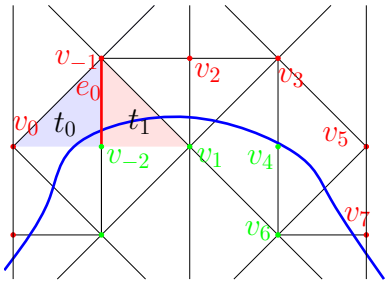


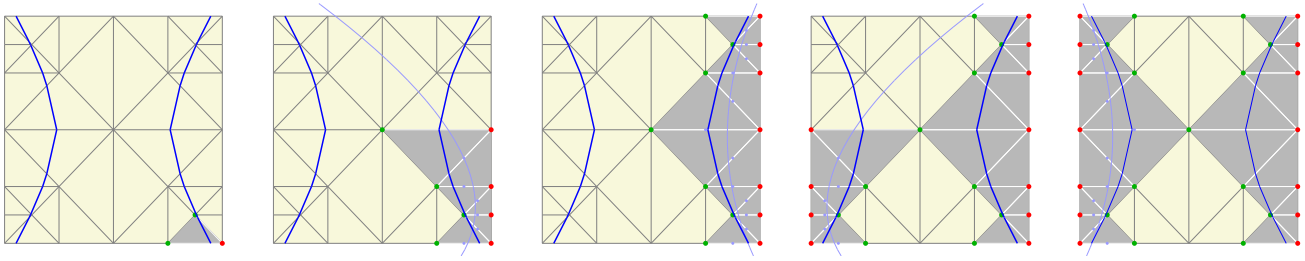
Figure 13: *Coarser resolution compression: the traversal goes from t_0 to t_1 through the gate e_0, and encodes the vertex v_1.*

Algorithm 6 next triangle(e, t) : gets the next triangle t' from edge e

```
1: for all edges e' = (t, t') of t do // look for the out edge
2:     if e' ≠ e and e'.crossing and not t'.encoded then //
       crossing edge
3:         return (e', t')                    // next triangle
4:     end if
5:     return ∅                               // closed component
6: end for
```

Actually, the sign of the vertex v_1 is encoded only when it has not been marked. This guarantees to encode exactly one sign bit per vertex, with an overhead of a few bits per connected component, used for the localization procedure (see Figure 12).

Adaptive encoding. The above algorithm can be easily modified to encode the tubular neighbourhood of the isocontour when it is composed of triangles of different levels. In that case, the algorithm also encodes the level of the current triangle (t_1) during the traversal. The decoder will read the required level for t_1 and subdivide or simplify t_1 if necessary. The RBMT we use maintain a difference of at most one



(a) Location: level 5, 0, $L\ R\ L\ L\ L. - + +.$ (b) Vertex levels and signs: $> - = - < + < - < +.$ (c) $= + > - > + > - = - < +.$ (d) level 5, trig 0, $L\ L\ L\ L\ L. - + +. > - = - < + < - <.$ (e) $= + > - > + > - = - < +$*End.*

Figure 14: *Adaptive encoding of the coarser resolution of a small hyperbola.*

Algorithm 7 receive coarse uniform : decodes the coarser level uniformly

1: $t \leftarrow$ receivelocalize ; $e \leftarrow t.\text{edge}_0$ *// retrieve the first triangle*
2: **repeat** *// traverse the curve*
3: $(e', t') \leftarrow$ next triangle(t) *// get the next triangle*
4: $curve \leftarrow curve \cup (e', t')$ *// add to the decoded curve*
5: e'.opposite.isovalue $\leftarrow$ receive bit *// decode the sign of the next vertex*
6: **until** [closed the component]$t' = \emptyset$
7: **return** $curve$ *// the curve is used for the refinements*

level between adjacent triangles (see section 4 *Adaptive Tessellation*). Therefore, the decoder will have to subdivide or simplify an unvisited triangle at most once, and the encoder will manage only 3 symbols: $' ='$ when the levels match, $' >'$ or $' <'$ when the levels differ. The encoding of the signs is done similarly to the uniform one.

This guarantees to encode exactly one bit per vertex of the tubular neighbourhood, plus one symbol of $\{=, <, >\}$ per triangle of the tubular neighbourhood, with an overhead of a few bits per connected components, used for the localization procedure (see Figure 14).

(b) Progressive refinement

The refinement methods allow a progressive adaptation of the tubular neighbourhood to the isocontour, using each time smaller triangles. The algorithm can simply encode the signs of the new vertices created by the subdivisions of all triangles on the tubular neighbourhood, or it can first specify which triangles to subdivide and then send the sign of the new vertices inserted.

Uniform refinement. The uniform refinement simply subdivides first all the triangles of the tubular neighbourhood, and then en/decodes the signs of the vertices created by the subdivision inside the former tubular neighbourhood. The order of the new vertices is induced by the traversal used for the coarser resolution en/decoding.

When subdividing a triangle crossing the isocontour, the configuration of its sub–triangles is determined by the sign

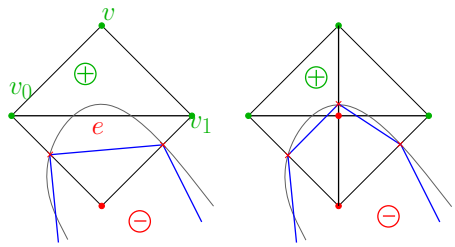


Figure 15: *A subdivision can extend the tubular neighbourhood of the isocontour if the sign of $vv_0 > 0$ and $vv_1 > 0$.*

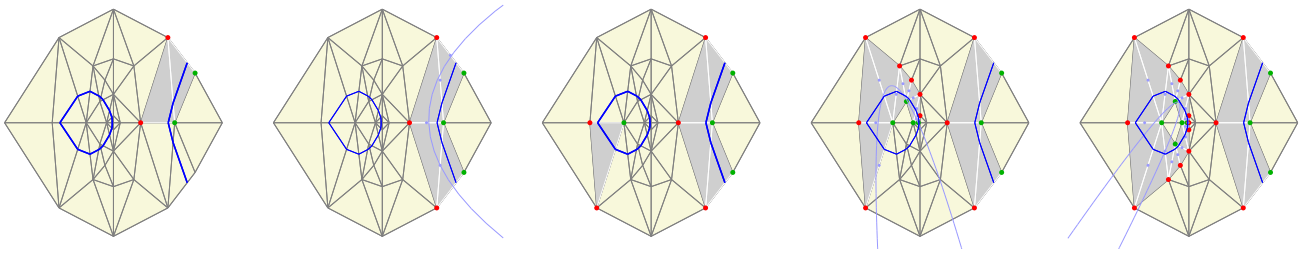
of the new vertex introduced on the subdivision edge. This sign is encoded during the compression. This subdivision can locally extend the tubular neighbourhood (see Figure 15). This case occurs when a non–crossing subdivision edge $e = v_1v_2$ gives rise to two crossing sub–edges. In that case, the vertex v opposite to e will be included in the tubular neighbourhood, but its sign need not to be encoded as it can be deduced by the decoder as the sign of v_1 (see Figure 15).

Adaptive refinement. Our method also allows an adaptive progression, creating smaller triangles where the isocontour is more complex, and leaving bigger triangles where it is simple. For each subdivision edge in the tubular neighbourhood, we encode the Refine and Keep code. When a Keep symbol is encoded, the subdivision edge will not be considered in future refinements, keeping it as a leaf on the hierarchy. The subdivision edges of the tubular neighbourhood are collected in the order of the coarser resolution traversal. The sign of the newly inserted vertices is encoded in the same way as for uniform refinement (see Algorithm receive adaptive refinement).

Actually two successive resolution of the RBMT can differ locally by more than one level in the binary tree. Therefore, the above sequence will be repeated as long as a Cont/Stop bit is read by the decoder after the vertex signs are received.

(c) Final geometry encoding

Once the tubular neighbourhood of a resolution level is encoded, only one bit for each vertex of the RBMT is known



(a) Location: level 6, 1, $L\ L\ L\ L\ L\ R. - + +. < -.$ (b) Vertex levels and signs: $= - > +.$ (c) level 5, trig 0, $L\ L\ L\ L\ L.$ (d) $- + -. = - = - > + > - = - < + = -.$ (e) $> - = - = - =< + = - > - = + << End.$

Figure 16: *Adaptive encoding of the coarser resolution of a cubic: irregular tessellation can reduce the distortion.*

Algorithm 8 receive adaptive refinement($curve$) : decodes the adaptive refinement

Refine the curve
$newverts \leftarrow \emptyset$ *// vertices created by the subdivision*
$subedges \leftarrow$ subdivision_edges($curve$) *// subdivision edges of the curve*
for all $edgee \in subedges$ **do** *// potential subdivisions*
  if receive bit **then** *// e should be subdivided*
    $(\{w_i\}, \{e_i\}) \leftarrow$ subdivide(e) *// new vertices and subdivision edges*
    $newverts \leftarrow newverts \cup \{w_i\} \cap$ neighbour($curve$)
    $subedges \leftarrow subedges \cup \{e_i\} \cap$ neighbour($curve$)
  end if
  $subedges \leftarrow subedges \setminus e$ *// remove e from subedges*
end for

Decode the isovalues of the new vertices
for all $vertexw \in newverts$ **do** *// new vertices*
  w.isovalue $\leftarrow$ receive bit *// retreive the sign of w*
end for

to the decoder. Therefore, the position of each point p of the isocontour is *a priori* the middle point of a crossing edge e. This can be improved by sending the isovalue of the endpoints of e. Moreover, this scalar value will be used to encode other isocontour generated with a different isovalue.

The input 2D data regularity can actually influence the quality of the compression (see Figure 16). For specific applications such as human face contour compression, the input data can be sampled according to the mean and variances of the original isocontour, resulting in an elliptic–radial initial distribution of the 2D data.

(d) Close–by isocontour encoding

Once an isocontour C corresponding to an isovalue α_C has been decoded, the decoder knows its entire tubular neighbourhood. It can be used to encode also isocontours C' corresponding to isovalues $\alpha_{C'}$ close to α_C, especially for medical applications where the contrast of the 2D data is not well defined. C' can be easily encoded as a new coarser resolution by sending first the isovalue $\alpha_{C'}$ and then the quantized isovalues of the vertices not precise enough or unknown to the decoder. The algorithm is almost identical to Algorithm receive coarse uniform.

7 Arithmetic encoding

In order to improve the compression rate of our scheme, each compression step (coarser resolution, refinements and isovalue compression) is composed of the encoder described in the previous section, followed by an arithmetic coder. Those coders can be improved by the use of good predictors and eventually by the memory of the former symbols encoded, known as the *order* of the encoder. Moreover, the predictor can estimate the quality of its prediction. Depending on the prediction and on the order, the encoder will use different probability models, or *contexts*, which will be used for the arithmetic coding. Those models are then updated according to the current symbol, in order to the encoder adapt itself to the data. We will now describe the predictors we used for the different techniques of encoding, and detail the contexts we used with them.

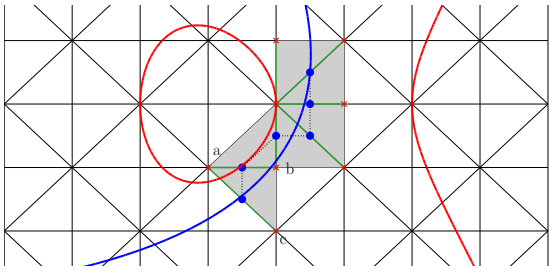


Figure 17: *The next crossing edge can be predicted using a simple curve fitting (in blue) on the decoder points, situated at the middle of each crossing edge for the first pass.*

(a) Prediction

Coarser resolution. The sign of the vertex being decoded can be predicted by approximating the decoded curve and extrapolating this approximation. More specifically, we fit a parabola on the last 20 decoded isocontour points using [12]. Observe that during the decompression, the isocontour points are at the center of each decoded crossing edge. This parabola passes closer to one of the two unknown edges the triangle returned by Algorithm next triangle. The algorithm predicts that edge to be the next crossing edge. Moreover, the distance to the central vertex allows defining 3 different cases, each of which corresponds to a specific probability model (context). This prediction succeeds in average in 80% of the cases [13]. We encode if prediction succeeds using an order 0 arithmetic coder.

Refinements. We distinguish two cases for the vertex v created during the refinement process: v is on the border of the tubular neighbourhood, or inside the tubular neighbourhood. On the border, the general case is not to extend the tubular neighbourhood, as mentioned in section 6(b) *Progressive refinement*. Inside the tubular neighbourhood, the prediction is be made using the parabola fitting of the last paragraph. We encode if prediction for the refinements succeeds using an order 2 arithmetic coder.

Geometry. The geometry can be predicted smoothing the isocontour, by standard signal processing. However, since we already used the parabola fitting with success, we use it again for the prediction of the geometry. This strategy showed nice results when encoding the geometry as a final step. We used an order 1 arithmetic coder to encode the difference between the predicted isovalue and the real one.

(b) Context designs

The encoding schemes we introduced here accompany the isocontour to compress. Therefore, the next symbols to encode are closely related to the symbols just encoded before. This fact gives a priori more accurate probability models for the context–based arithmetic encoding. The context model can be enhanced with a good initialization. In particular, we use a Gaussian probability centered at 0 for the geometry encoding, an $80\%/20\%$ probability for the sign encodings and a doubled probability to the $=$ symbol in the adaptive base mesh encoding.

We consider a unique probability distribution of the symbols (context) for the order 0 encoders, four of them for the order 2 coders (for each combination of the 2 last bits) and two for the geometry (since we send one bit at a time). Each context is updated at each symbol encoded. Moreover, the encoder does not encode a RBMT grid vertex twice, although it visits it more than once. This multiple visit can be used to update the probability model of the context, even without encoding the corresponding symbol. This accelerates the learning delay of the arithmetic coder, without any cost.

8 Results

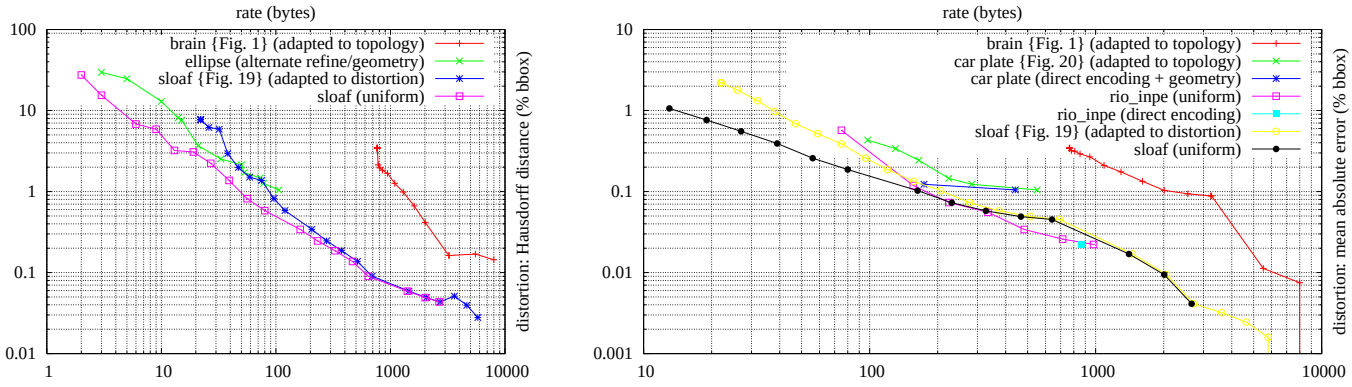
The methods we introduced here are flexible and can be used in many ways. For isocontours of images, we chose to reduce the complete triangulation of the pixels. We then encoded the isocontour at each step of this reduction, which can be uniform or adapted to the isocontour. For implicit curves, we sent in–between two levels of detail a part of the geometry (2 bits). We implemented our compression scheme with a context arithmetic coder of order 0 for the coarser level transmission, 2 for the refinements and 1 for the final geometry encoding. We used a simple predictor based on a second–order curve fitting [12, 13] (see Figure 12 and Figure 14).

Our method resulted in efficient rate/distortion curves (see Figure 18). The different controls on the adaptation of the multi–resolution can have a significant cost. For example, on regular models such as the elevation curve of the Sugar Loaf of Figure 20, the distortion and topology controls provide nicer results to the eyes at the beginning of the compression than uniform refinements, while finally resulting in similar performances. Topology control means an extra cost depending on the complexity of the model (compare the car number plate of Figure 21 with the brain model of Figure 1 on Figure 18), but the rate/distortion performance has a similar evolution for all methods and models (see Figure 18(a)). Geometry encoding seems a good alternative to refinements for regular models such as the ellipse of Figure 18(a). Single rate encoding is, as usual, a little more efficient than progressive encoding, but can be complemented by the final geometry compression of section 6(c) *Final geometry encoding*.

9 Next steps

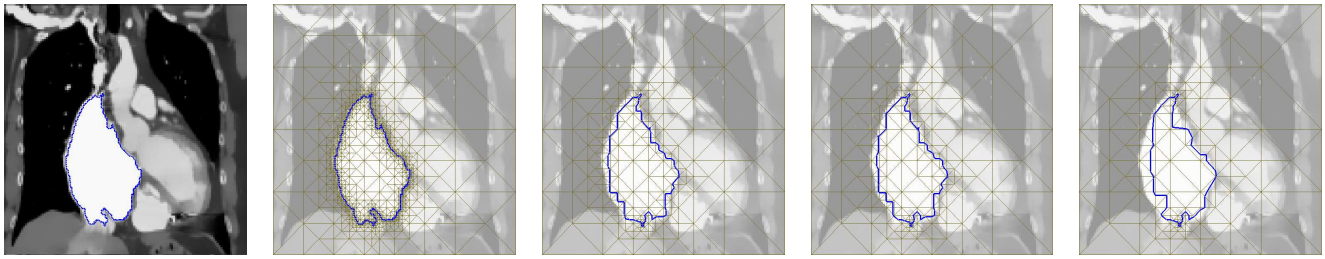
We introduced a complete isocontour multi–resolution extraction and compression scheme. The dynamic triangulation we used provides a very simple way of creating a multi–resolution structure of the 2D data adapted to the isocontour. This structure allowed us to achieve both direct and progressive compression, and to encode the isocontour uniformly or adaptively. The single rate, uniform compression is guaranteed to compress less than 2 bits per pixels intersecting the curve *before* the arithmetic coder. Moreover, it provides a full control on the progression, granting topological and geometrical control on the decoded isocontour.

The corresponding work was published in Computerized Medical Imaging and Graphics, volume 30, number 4, pp. 231-242. Elsevier 2006..



(a) Hausdorff distance. (b) Mean absolute error.

Figure 18: *Compression results: on complex models, topology control is mode expensive with progressive encoding.*



(a) Original. (b) Tessellation. (c) Curvature. (d) Distortion. (e) Topology.

Figure 19: *Extraction of an isocontour with different controls.*

We used binary multi–triangulations for the scalar data representation, implemented using dimension–independent generic programming [17]. This offers a very simple extension of our algorithm to compress level sets in any dimension with a similar efficiency. This will require first to optimize the arithmetic coding and prediction parameters we use for the compression, and to quantify the advantages of geometry encoding upon refinement operations.

References

[1] J. W. Alexander. The combinatorial theory of complexes. *Annals of Mathematics*, 31:219–320, 1930.

[2] F. Bossen and T. Ebrahimi. A simple and efficient binary shape coding technique based on bitmap representation. In *Acoustics, Speech, and Signal Processing*, pages 3129–3132, 1997.

[3] L. da Fontoura Costa and R. M. Cesar Jr. *Shape analysis and classification*. CRC Press, 2000.

[4] M. Craizer, D. A. Fonini Jr and E. A. B. da Silva. Alpha–expansions: a class of frame decompositions. *Applied and Computational Harmonic Analysis*, 13:103–115, 2002.

[5] E. Danovaro, L. de Floriani, M. Lee and H. Samet. Multiresolution tetrahedral meshes: an analysis and a comparison. In G. Wyvill, editor, *Shape Modeling International*, pages 83–91, Banff (Canada), 2002. IEEE.

[6] D. Douglas and T. Peucker. Algorithms for the reduction of the number of points required for represent digitized line or its caricature. *Canadian Cartographer*, 10(2):112–122, 1973.

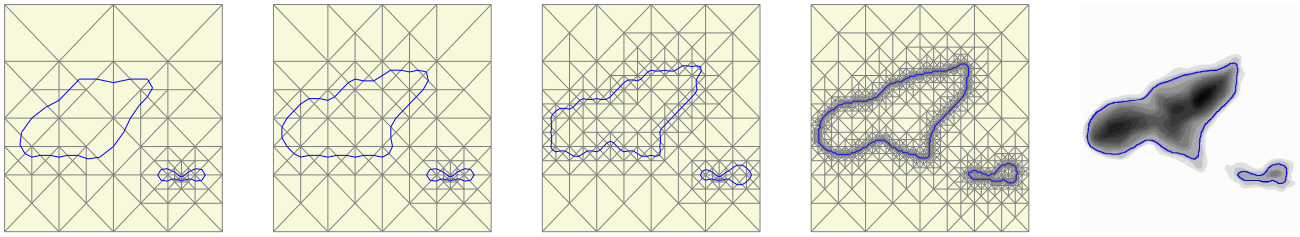
[7] L. de Floriani, E. Puppo and P. Magillo. A formal approach to multiresolution modeling. In W. Straßer, R. Klein and R. Rau, editors, *Theory and Practice of Geometric Modeling*, pages 302–323. Springer, 1997.

[8] H. Freeman. Computer processing of line drawing images. *Computing Surveys*, 6(1):57–97, 1974.

[9] C. Le Buhan and T. Ebrahimi. Progressive polygon encoding of shape contours. In *Image Processing and its Applications*, pages 17–21, 1997.

[10] G. Langdon Jr and J. Rissanen. Compression of black–white images with arithmetic coding. *Transactions on Communications*, 29(6):858–867, 1981.

[11] P. Lee and T. Nagao. Hierarchical description of two dimensional shapes using a genetic algorithm. In *IEEE Evolutionary Computation*, pages 637–640, 1995.



(a) 21 bytes (b) 31 bytes (c) 46 bytes (d) 156 bytes (e) 690 bytes

Figure 20: *Progressive compression of an elevation curve of the Sugar Loaf. The tessellation is adapted to the curve to minimize the distortion and to preserve the topology. (a) The shape of the tubular neighbourhood of the curve is sent first, with the nodes sign. (b),(c),(d) Then the refinements of the tessellation are encoded with the signs of the new nodes. (e) Finally, the values of the nodes in the tubular neighbourhood are refined.*

(a) progressive: 97 bytes (b) 129 bytes (c) 164 bytes (d) 223 bytes (e) direct: 173 bytes, progr.: 283 bytes

Figure 21: *Progressive compression of a car number plate (with enhanced contrast), topology controlled: direct encoding and progressive encoding both provide efficient compression ratio.*

[12] T. Lewiner, J. Gomes Jr, H. Lopes and M. Craizer. Arc–length based curvature estimator. In *Sibgrapi*, pages 250–257, Curitiba, Oct. 2004. IEEE.

[13] T. Lewiner, J. Gomes Jr, H. Lopes and M. Craizer. Curvature and torsion estimators based on parametric curve fitting. *Computers & Graphics*, 29(5):641–655, 2005.

[14] H. Lopes, J. B. Oliveira and L. H. de Figueiredo. Robust adaptive polygonal approximation of implicit curves. *Computers & Graphics*, 26(6):841–852, 2002.

[15] S. Mallat. *A wavelet tour of signal processing*. Academic Press, 1999.

[16] S. Marshall. Review of shape coding techniques. *Image and Vision Computing*, 7(4):281–294, 1989.

[17] V. Mello, L. Velho, P. Roma Cavalcanti and C. Silva. A generic programming approach to multiresolution spatial decompositions. In H.-C. Hege and K. Polthier, editors, *Visualization and Mathematics III*, pages 337–360. Springer, Heidelberg, 2003.

[18] V. Pascucci. Slow growing subdivision (SGS) in any dimension: towards removing the curse of dimensionality. *Computer Graphics Forum*, 21(3):451–460, 2002.

[19] E. Puppo. Variable resolution triangulations. *Computational Geometry: Theory and Applications*, 11(3–4):219–238, 1998.

[20] A. Rosenfeld, editor. *Multiresolution image processing and analysis*. Springer, Berlin, 1984.

[21] A. Safonova and J. Rossignac. Compressed piecewise–circular approximations of 3D curves. *Computer–Aided Design*, 35(6):533–547, 2003.

[22] B. von Herzen and A. H. Barr. Accurate triangulations of deformed, intersecting surfaces. In *Siggraph*, pages 103–110. IEEE, 1987.

[23] L. Velho. A dynamic adaptive mesh library based on stellar operators. *Journal of Graphics Tools*, 9(2), 2004.

[24] S. T. Wu and M. R. G. Márquez. A non–self–intersection douglas–peucker algorithm. In *Sibgrapi*, pages 60–66. IEEE, 2003.

The corresponding work was published in Computerized Medical Imaging and Graphics, volume 30, number 4, pp. 231-242. Elsevier 2006..