

## Decreasing isosurface complexity via discrete fitting

C. Montani<sup>a</sup>, R. Scateni<sup>b</sup>, R. Scopigno<sup>c</sup>

<sup>a</sup> IEI—CNR, Via S. Maria 46, I-56126 Pisa, Italy

<sup>b</sup> CRS4, Via N. Sauro 10, I-09123 Cagliari, Italy

<sup>c</sup> CNUCE—CNR, Via S. Maria 36, I-56126 Pisa, Italy

Received May 1998; revised June 1999

---

### Abstract

Since the introduction of techniques for isosurface extraction from volumetric datasets, one of the hardest problems has been to reduce the number of generated triangles (or polygons).

This paper presents an algorithm that considerably reduces the number of triangles generated by a Marching Cubes algorithm, while presenting very close or shorter running times. The algorithm first assumes discretization of the dataset space and replaces cell edge interpolation by midpoint selection. Under these assumptions the extracted surfaces are composed of polygons lying within a finite number of incidences, thus allowing simple merging of the output facets into large coplanar triangular facets. Lastly, the vertices which survived the decimation process are located on their exact positions and normals are computed.

An experimental evaluation of the proposed approach on datasets relevant to biomedical imaging and chemical modeling is reported. © 2000 Elsevier Science B.V. All rights reserved.

*Keywords:* Volume rendering; Isosurface extraction; Marching cubes; Surface simplification

---

### 1. Introduction

The use of the *Marching Cubes* (MC) technique, originally proposed by Lorensen and Cline (1987), is nowadays considered to be the standard approach to the problem of extracting isosurfaces from a volumetric dataset. As a member of the large class of surface tracking techniques, surveyed by Styzt et al. (1991), MC is a very practical and simple algorithm, and many implementations are available either as part of commercial systems or as public domain software.

Despite its extensive use in many applications, shortcomings of the approach have been pointed out regarding *topological consistency and correctness*, algorithm *computational efficiency*, and excessive output *data fragmentation*.

Standard MC produces no consistent notion of object connectivity; the local surface reconstruction criterion used allows a number of *topological ambiguities* (Dürst, 1988), and therefore MC may generate surfaces which are not necessarily coherent. These shortcomings have been extensively studied (Ning and Bloomenthal, 1993) and solutions have been proposed (Lachaud, 1996; Montani et al., 1994a; Natarajan, 1994; Nielson and Hamann, 1991; Payne and Toga, 1990; Van Gelder and Wilhelms, 1994).

MC *computational efficiency* can be increased by exploiting implicit parallelism (each cell can be independently processed) (Hansen and Hinker, 1992) and by avoiding visiting and testing empty cells (previous research indicated that between 80% and 90% of voxels do not contain the isosurface searched for). Many different data structures and algorithms have been used to prevent fruitless exploration of regions of the volume: octrees (Wilhelms and Gelder, 1992) and other data structures (Itoh and Koyamada, 1995) in the case of regular datasets, *kd*-trees (Livnat et al., 1996) and interval trees (Cignoni, 1997) for scattered datasets. Both for regular and scattered datasets, the *seed sets* approach (Bajaj et al., 1996; van Kreveld, 1997) turns out to be compact and efficient as well.

*Excessive fragmentation* of the output data can prevent interactive rendering when high resolution datasets are processed. What has changed since the introduction of the technique, in 1987, has been the amount of data to be processed while extracting such surfaces. Equipment able to generate volumetric datasets as large as  $512 \times 512 \times [\leq 512]$  are now generally available, for example the MRI machines used in radiology, and we are on the way to achieve machines capable of producing datasets of up to  $10^9$  voxels. Although an isosurface does not usually cross more than a small subset of the voxels, we can understand how easy it is to generate surfaces defined by millions of triangles. State-of-the-art hardware is not yet fast enough to manipulate such masses of data in real time.

Two different classes of algorithms have been developed to overcome these limitations:

- *Adaptive fitting* techniques apply ad hoc data traversing and fitting in order to reduce output data complexity. They exploit the spatial coherence of the dataset and operate during the isosurface fitting process;
- *Simplification* techniques simplify the extracted isosurface either by merging nearly co-planar faces, or by removing geometric elements (vertices, edges, faces), or by re-sampling vertices. They work a posteriori and are completely independent of the fitting process.

In this work we present Discrete Marching Cubes (DiscMC), an efficient isosurface fitting and simplification algorithm. DiscMC is based on a simple discretizing approach and its founding idea was first presented by the authors in (Montani, 1994b). With respect to the previous version, DiscMC now returns isosurfaces which are almost undistinguishable from those returned by a standard MC algorithm, it achieves a high simplification rate, and it returns very compact convex polygons represented by means of efficient triangle strips and fans. DiscMC solves the problem of the topological ambiguities by adopting a solution which is very similar to that one we proposed in (Montani et al., 1994a) for the general case of the MC algorithm. The use of a pyramidal data structure to prevent fruitless exploration of empty parts of the volume ensures high computational efficiency. Finally, DiscMC presents the advantages of both adaptive fitting and simplification techniques. As in a simplification method, its ability to merge co-planar elements is independent of the

fitting process and not limited to few adjacent cells; as in an adaptive method, it exploits the spatial coherence of the regular input dataset.

The paper is organized as follows. Section 2 describes previous work in the related areas of isosurface fitting and mesh simplification. Section 3 describes DiscMC in details by discussing the three main steps of the algorithm: *marching*, the extraction of an approximated isosurface formed by geometric primitives with prearranged shapes and incidence; *merging*, the fusion of the geometric primitives into larger triangle strips and fans; *interpolating*, the computation of the exact positions and normals of the vertices. Results and conclusions are discussed in Sections 4 and 5, respectively.

## 2. Related work

Substantial results have been reported in the last few years aimed at reducing surface complexity. As previously introduced, the proposed solutions can be classified into two broad classes: *adaptive fitting* techniques and *simplification* techniques.

*Adaptive fitting* techniques are peculiar to volume rendering applications. On the whole, these approaches reduce output data complexity, either by:

- Bending the mesh to eliminate the tiny facets produced when the isosurface passes near a vertex or an edge of a cubic cell (Moore and Warren, 1992). This technique generally improves the rendering process but it does not achieve high reduction factors in the number of generated triangles;
- Using adaptive down-sampling of the volume in the regions where the isosurface is nearly flat and maintaining high resolution in the parts with finer details (Mueller and Stark, 1993; Shekhar et al., 1996; Shu et al., 1995).

Though with different approaches and data structures, the algorithms using adaptive down-sampling replace almost co-planar geometric primitives belonging to adjacent cells with larger planar polygons and then they have to face with the problem of the turned up cracks (surface discontinuities due to the presence of patches at different resolution) by properly moving vertices or adding new facets. DiscMC avoids this problem because it does not down-sample the volume; it simply merges co-planar,  $C^0$  continuous geometric primitives.

Moreover, extracting isosurfaces at different resolution or patching cracks might generate rendering anomalies. These occur when a large polygon, adjacent on one edge to smaller ones, does not incorporate all of the vertices lying on that edge. This problem is ignored by the mentioned algorithms. Its solution generally leads to an increased number of triangles of the output surface.

*Simplification* techniques simplify a [triangular] mesh either by merging elements or by removing elements (vertices, edges or faces) or by re-sampling vertices, using different error criteria to measure the fit of the approximated surface. Any level of reduction can be generally obtained with these approaches, on the condition that a sufficiently coarse approximation threshold is set. Since any type of mesh (e.g., laser range data, terrain, synthetic surfaces) can be managed, these approaches are more general than the previous ones but they often require the explicit representation of the topology of the mesh and they can't exploit the spatial coherence of the regular input grid (Heckbert and Garland, 1997; Puppo et al., 1997).

Simplification proposals include the following:

- *Coplanar facets merging*: coplanar or nearly coplanar data are searched in the mesh, merged into larger polygons, and then re-triangulated into fewer simple facets than those originally required (De Hamer and Zyda, 1991; Hinker and Hansen, 1993; Kalvin and Taylor, 1996);
- *Controlled vertex/edge/face decimation*: these methods work by the iterative elimination of components (vertices, edges, faces), chosen upon local geometric optimality criteria (Algorri and Schmitt, 1996; Garland and Heckbert, 1997; Hamann, 1994; Schroeder et al., 1992);
- *Energy function optimization*: mesh reduction is iteratively obtained by performing legal moves on mesh edges (collapsing, swapping or splitting). Legal moves selection is driven by an optimization process of an energy function, which measures the *quality* of each reduced mesh (Hoppe, 1996; Hoppe et al., 1993);
- *Vertex clustering*: based on geometric proximity, this approach groups vertices into clusters, and for each cluster it computes a new representative vertex (Low and Tan, 1997; Rossignac and Borrel, 1993);
- *Wavelet-based approaches*: the wavelet decomposition approach has been proposed to manage regularly gridded meshes (Gross et al., 1996; Hebert and Kim, 1995) or more generic meshes (Eck et al., 1995).

The results of the empirical comparison of six different simplification codes are discussed in (Cignoni et al., 1998a). One of the results of this comparison was that the lower empirical computational complexity is still hold by the Mesh Decimation (Schroeder et al., 1992) approach. We therefore report the results of empirical comparisons between Mesh Decimation and DiscMC in Section 4.

### 3. The Discrete Marching Cubes algorithm

#### 3.1. Algorithm's overview

Let's introduce the DiscMC algorithm by means of the simple example in Fig. 1. Starting with the four contiguous cells (on the left) in which all the back vertices are classified *on* (i.e., the associated data values are greater than the user selected threshold) and all the front vertices are *off*, the upper and lower sequences in the figure represent the logical pipeline of the MC and DiscMC algorithms, respectively.

For the cells of the example, the MC algorithm (upper sequence) directly produces 8 triangles whose 9 vertices are computed by linear interpolations on the data values of the corresponding edges and, for each vertex, a normal to the isosurface.

The DiscMC pipeline consists of three different steps:

- A *marching step*, in which each active cell is represented by one out of a set of predefined geometric patterns. Geometric patterns consist of a limited number of different geometric primitives and plane incidences. No vertex coordinates are computed in this phase: each geometric primitive is simply represented by a code and by the address of the cell it belongs to;

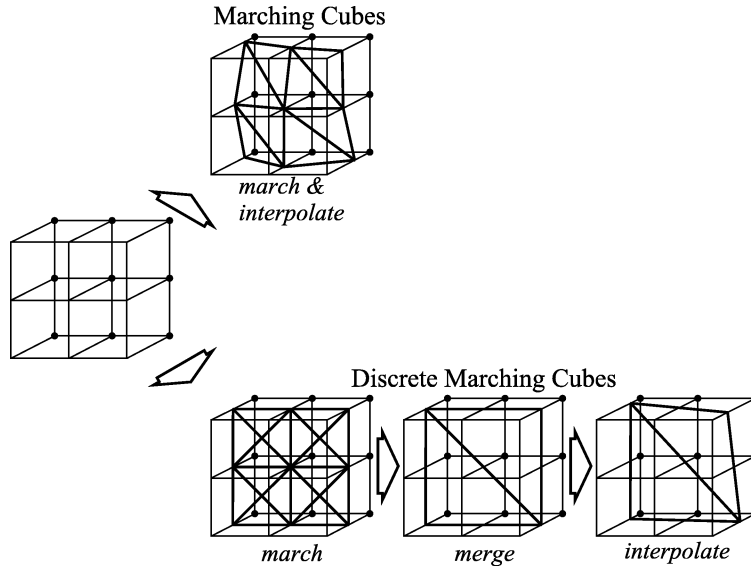


Fig. 1. The Marching Cubes (upper sequence) and the Discrete Marching Cubes (lower sequence) logical pipelines.

- A *merging step*, in which all the coplanar adjacent primitives are merged into larger polygons; polygons are subdivided into convex parts and these latter are finally converted into triangle strips and/or fans. Again, no vertex coordinates are computed: they are simply expressed in the coordinate space of the regular input dataset;
- An *interpolating step*, in which all the vertices which survived the merging phase are located, by linear interpolations, to their exact positions and the corresponding normals computed.

For the four cells of the example in Fig. 1, the DiscMC algorithm (lower sequence) first produces 16 coplanar right triangles (marching step); these primitives are then merged into a large square which is then converted into 2 triangles (merging step); finally, the exact position of the 4 vertices and the normals to the isosurface are computed (interpolating step).

Based on the example, we can anticipate the results produced by our algorithm. With respect to the classic MC approach, DiscMC returns topologically identical isosurfaces in which the number of representing triangles is greatly reduced and the corresponding vertices are in the same locations. The error DiscMC introduces, in terms of the maximum distance between the two isosurfaces, is lower than the largest cell edge.

In the following, more details about the algorithm are given by referring to the three mentioned logical steps.

### 3.2. The marching step

The DiscMC's basic idea is to first approximate the requested isosurface by means of a predefined limited number of geometric primitives and plane incidences. DiscMC provides, for each cell, only 13 different spatial locations on which vertices can be

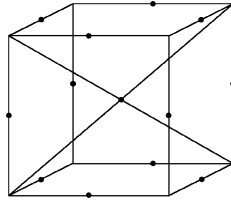


Fig. 2. The set of different vertex locations adopted by DiscMC.

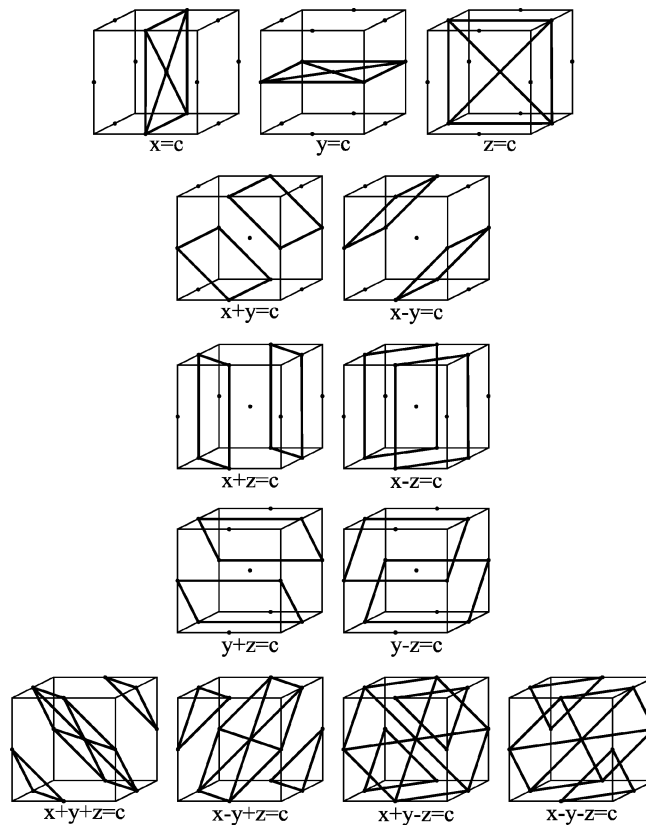


Fig. 3. The geometric primitives returned by DiscMC for each different plane incidence.

originated: 12 cell-edge midpoints plus the cell centre (Fig. 2). Moreover, DiscMC adopts only 13 different plane incidences on which primitives can lie and, for each incidence, a limited number of different geometric primitives (Fig. 3):

- four right triangles for the incidences  $x = c$ ,  $y = c$ , and  $z = c$ ;
- two rectangles for the incidences  $x \pm y = c$ ,  $x \pm z = c$ , and  $y \pm z = c$ ;
- eight equilateral triangles for the incidences  $x \pm y \pm z = c$ .

The mesh fitted by DiscMC is therefore very similar to that one produced by MC while extracting the isosurface with isovalue  $t = 0.5$  from a binary dataset (i.e., with only 0's and 1's as data values).

### 3.2.1. A new lookup table

For each *on-off* combination of the cell vertices (there are 256 possible combinations due to the classification of the cell's vertices with respect the selected threshold  $t$ ), the MC lookup table (lut) codes the number of triangles to be generated and the cell edges on which the vertices lie.

A similar lookup table has been defined for DiscMC; the geometric pattern for each cell configuration is coded by using one or more of the primitives shown in Fig. 3. The 16 canonical configurations of the DiscMC lut are shown in Fig. 4. Each primitive is coded in the DiscMC lut by using a *shape code*, which codifies the shape and position (with reference to the cell) of the primitive (1..4 for right triangles, 1..2 for rectangles and 1..8 for equilateral triangles), and an *incidence code*, i.e., the plane on which the primitive lies. Geometrical information on the facet vertices isn't explicitly stored in the DiscMC lut.

For each cell configuration, DiscMC lut stores from zero up to seven geometric primitives, univocally individuated by a shape code (1..8) and an incidence code (−13..13).

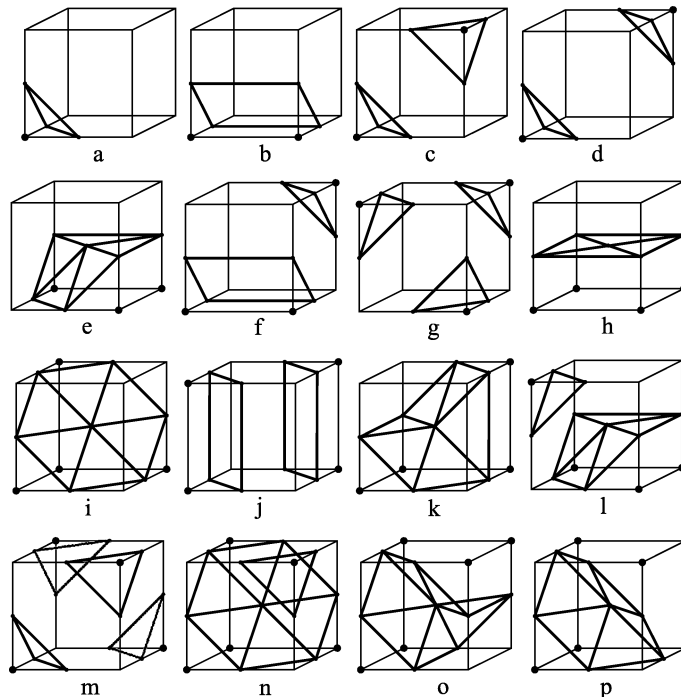


Fig. 4. The sets of geometric patterns returned by DiscMC for each cell vertex configuration. Only the topologically different configurations are shown.

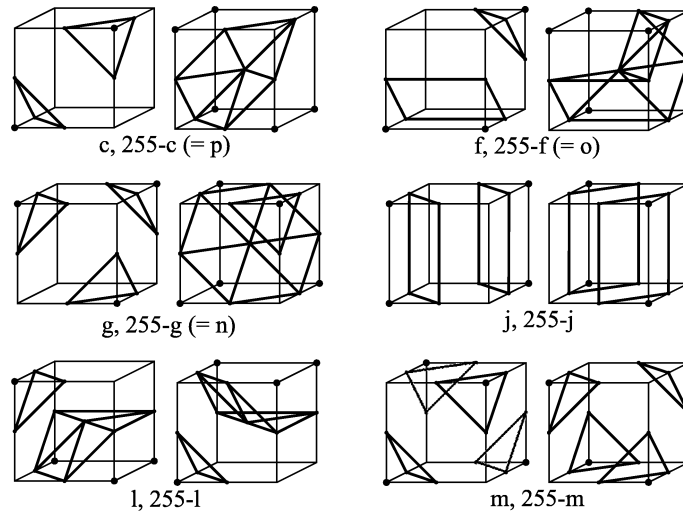


Fig. 5. Ambiguous configurations. For each couple of cells the patterns returned by DiscMC for the ambiguous configuration (left) and for the complementary one (right).

We use signed incidences to store separately facets which lie on the same plane and have opposite normal directions. This avoids merging possibly adjacent primitives which belong to topologically different surfaces.

As recalled in the introduction, the MC algorithm can produce topological inconsistencies (Dürst, 1988) when dealing with ambiguous configurations. A cell is ambiguous if it contains one or more ambiguous faces, that is faces in which the *on* (above threshold) and *off* (below threshold) vertices are diagonally opposite. Configurations *c*, *f*, *g*, *j*, *l*, *m*, *n*, *o*, and *p* in Fig. 4 are ambiguous. MC inconsistencies derive from the use of a symmetric lut: the geometric pattern adopted for a cell with configuration *x* is the same adopted for the complementary cell  $255 - x$ . While this is correct in many cases (the triangle adopted for configuration *a* in Fig. 4 still holds for the complementary cell with seven *on* vertices and one *off* vertex), lut's symmetry can produce small holes in the case of ambiguous configurations.

DiscMC solves this problem by introducing three new geometric patterns with respect to the original MC (configurations *n*, *o*, and *p* in Fig. 4) and by adopting different solutions for the ambiguous complementary configurations. Fig. 5 shows, for each couple of cells, the geometric pattern used for the ambiguous configuration and for the complementary one. In (Montani et al., 1994a) we proposed a similar approach for the solution of the so-called ambiguity problem of MC.

### 3.2.2. Storing the geometric primitives

The data structures used to store the geometric primitives are designed to guarantee efficient access during the merging phase, even if this can cause a loss of efficiency in memory usage.

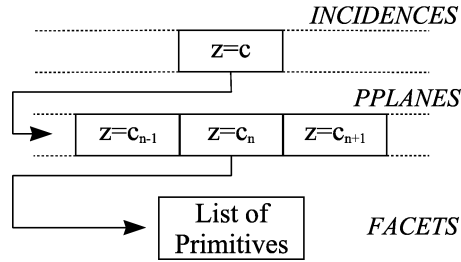


Fig. 6. The two-levels bucketed data structure for the geometric primitives.

The used data representation is a two-level bucketed structure, as shown in Fig. 6. Each entry of the 26 elements vector *INCIDENCES* holds a pointer to a vector of type *PPLANES* composed of as many elements as the maximum possible number of parallel planes at that particular incidence. Each entry of any *PPLANES* vector points to the list, if there is one, of all the facets belonging to that plane; in other words all the coplanar facets will be grouped into one list for faster access.

For each primitive extracted from the volume data, it's quite trivial to identify the plane it lies over: it depends on the incidence code and the spatial coordinates of the cell containing the facet.

Each primitive is stored in the corresponding primitives' list by simply recording its shape code and the coordinates of the cell it belongs to. Coplanar primitives belonging to the same cell are stored in a single entry in the list.

The memory overhead due to this solution is proportional to  $18(X + Y + Z)$ , where  $X$ ,  $Y$  and  $Z$  are the dimensions of the dataset. This is because there are at most  $X$ ,  $Y$ , and  $Z$  parallel planes for incidences  $x = c$ ,  $y = c$ , and  $z = c$ , respectively;  $2(X + Y)$ ,  $2(X + Z)$ , and  $2(Y + Z)$  parallel planes for incidences  $x \pm y = c$ ,  $x \pm z = c$ , and  $y \pm z = c$ , respectively;  $4(X + Y + Z)$  parallel planes for incidences  $x \pm y \pm z = c$ . This holds  $9(X + Y + Z)$  which is doubled because we consider signed plane incidences.

At the end of the extraction process we are ready to merge adjacent coplanar facets, and each list contains only those facets which are eligible to be merged.

### 3.2.3. Speeding up the marching step

As noted in the introduction, most of the cells visited in the marching step are not usually crossed by the isosurface. Since the computations performed by DiscMC on each not-empty cell is limited to one access to the lut and to the storing of the returned facets into the corresponding primitives' lists, most of the marching step's time is spent visiting and testing empty cells.

Octrees have been largely used to avoid the unnecessary exploration of regular datasets (Wilhelms and Van Gelder, 1992). Since in real applications it is not common to have datasets with large regions of uniform data values and so it is difficult to limit the depth of a hierarchical data structure, DiscMC adopts a more simple pyramidal data structure to speed up the localization of the active cells (i.e., the cells crossed by the isosurface). The data structure consists of a pyramid of 3D *min-max* interval arrays in which:

- the generic element of the lowest level of the pyramid holds the *min–max* extents of the corresponding (possibly) 8 cells in the volume dataset (27 data values);
- the generic element of each intermediate level of the pyramid holds the *min–max* extents of the corresponding (possibly) 8 elements of the underlying level;
- the highest level of the pyramid is an  $1 \times 1 \times 1$  array holding the *min–max* extents of the corresponding (possibly) 8 elements of the underlying level.

The search for active cells is performed by recursively visiting the active elements of the data structure. With respect to the octree (Wilhelms and Van Gelder, 1992) data structure, in which an uniform node is not further expanded at lower levels, pyramids do not exploit this kind of hierarchical data compression but, on the other hand, they are more simple to build. This memory cost is slightly greater in the case one or more dimensions of the volume are not powers of 2.

The pyramid's construction time is of the same order as a complete visit of the dataset but the data structure can obviously be used for any selected threshold. If  $S$  is the size of the dataset, then the pyramid costs, in terms of memory usage,  $\sim 2S/7$  (the levels of the data structure have sizes  $S/8, S/64, \dots$ , i.e.,  $S/7$ , and each entry contains two data values).

#### 3.2.4. Volume computation

An approximation of the amount of space delimited by the isosurface we are looking for can be easily computed during the marching step, without any additional cost. We coded the volume contribution of each not-empty cell in a field of the lookup table. Each contribution is codified by an integer number which represents, in terms of 48th's of the volume of a cell, the volume portion bounded by the isosurface crossing the cell. Volume computation is a simple summation of these contributions. Table 1 reports, for each configuration of Fig. 4, the volume contribution of the configuration and, when applicable, of the complementary one.

Table 1  
Volume contribution, expressed in 48th's of a cell, of each configuration and, when applicable, of the complementary one

Config.	Volume (compl.)	Config.	Volume (compl.)
a	1 (47)	i	24 (24)
b	6 (42)	j	12 (na)
c	2 (na)	k	24 (24)
d	2 (46)	l	18 (na)
e	17 (31)	m	4 (na)
f	7 (na)	n	25 (na)
g	3 (na)	o	17 (na)
h	24 (24)	p	38 (na)

### 3.3. The merging step

The real kernel of the algorithm is the merging step. The goal of this phase is to merge all the adjacent sets of coplanar primitives, to split the generated polygons into the largest possible convex sub-polygons, and then to triangulate them. We decided to transform via software the  $n$ -sided polygons into triangle strips to produce output data in a format which guarantees rapid visualization on the current state-of-the-art graphics libraries (e.g., OpenGL). In particular, our triangulation produces only triangle-strips and, if necessary, triangle-fans.

For the sake of clarity, we will describe these steps going informally through the example in Fig. 7. The example refers to a set of triangular primitives lying on a plane belonging to the incidence  $z = c$ .

#### 3.3.1. Primitives-to-MERGER conversion

For each plane, all the generated primitives (Fig. 7a) are written in XOR mode in a two-dimensional bytes array named MERGER. More precisely, the not-horizontal edges of

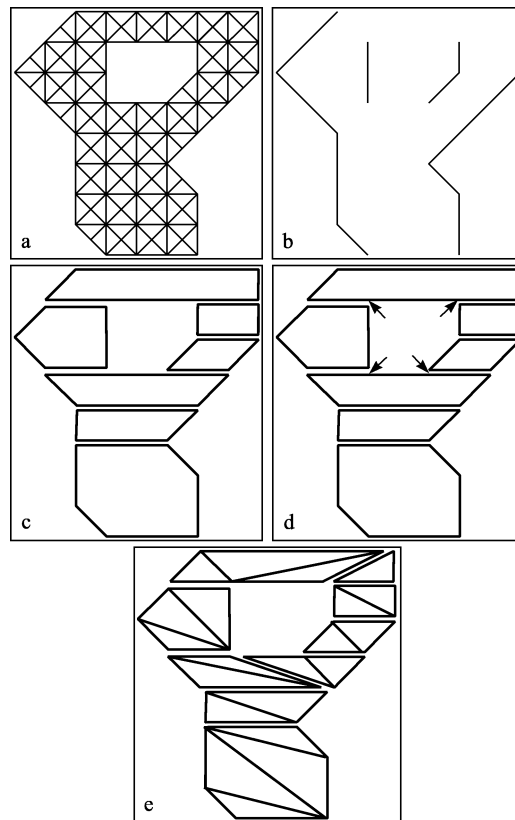


Fig. 7. The merging step at a glance.

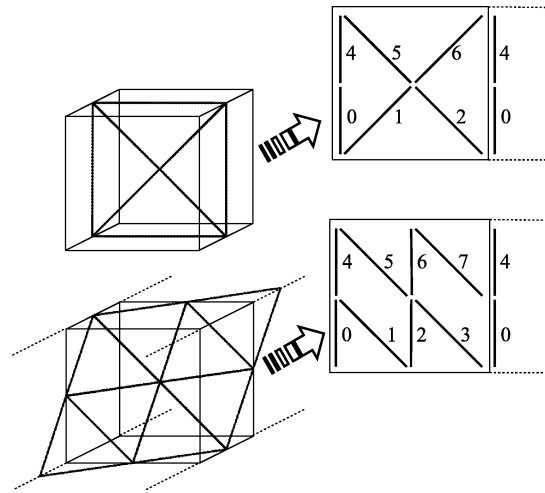


Fig. 8. Primitives-to-MERGER conversion.

each facet are encoded using one or two bits of a MERGER entry. The use of the XOR writing mode while updating the array MERGER leads to the automatic elimination from the array of all the edges shared by facet pairs, as shown in Fig. 7b. This means that at the end of the facets-to-MERGER conversion phase, the matrix will contain the encoding of the not-horizontal edges of the boundary of one or more coplanar polygonal regions.

Fig. 8 shows how geometric primitives are mapped into MERGER entries. The examples refer to the facets of the incidences  $z = c$  (upper) and  $x + y - z = c$  (lower). The squares on the right represent entries of the MERGER array and the numbers indicate the bits used to codify the not-horizontal edges of the geometric primitives. It has to be noted that, for the reported examples, the right sides of the rightmost primitives which map into the entry  $[x, y]$  are XOR written into the entry  $[x + 1, y]$  so that the left sides of the adjacent facets in position  $[x + 1, y]$  (if present) are correctly eliminated. Similar mapping rules have been defined for all of the incidences. As shown in the lower example of Fig. 8, eight bits are sufficient to codify all the sides of the primitives mapping onto an array entry. Hereafter we will refer to bits 0..3 in each entry as bits belonging to the *low stripe* of the entry, and to bits 4..7 as bits of the *high stripe*.

While constructing the MERGER array we also compute and store the bounding box enclosing all of the edges inserted in order to speed up the subsequent phase.

The MERGER array, whose dimensions are  $(\max[X, Y, Z], \max[X, Y, Z])$ , is allocated once and then used for any plane of each incidence, since the merging step is serial over the planes.

### 3.3.2. Reconstructing convex polygons

Once coplanar regions have been encoded in the MERGER array, our next step is to decompose them into a set of connected *convex* polygons (Fig. 7c).

The algorithm to perform the reconstruction first finds the left-most and bottom-most *stripe* encoded into the array. A *stripe* (Montani, 1984) is defined as a planar region, half-entry high, and delimited by two vertical or oblique sides.

Each *stripe* is represented by a pair of two consecutive *on* bits placed in even-odd positions in the same row of the MERGER array and belonging to the *low stripe* (bits 0..3) or to the *high stripe* (bits 4..7) of the row (Fig. 8).

The identification of each convex polygon proceeds upward by adding new *stripes* to the current polygon as long as the region remains convex. The convexity test is as follows: the examined low (high) *stripe* can be connected to the current high (low) *stripe* if and only if the lower vertices of the *stripe* coincide with the upper vertices of the current one and the internal angles at joints are 180 degrees or less. Due to the limited number of different slopes of the primitives' edges, this test is very simple to perform and, moreover, the polygons obtained are bounded by a limited number of sides: 3 to 8 for incidences  $x = c$ ,  $y = c$ , and  $z = c$ , 3 to 6 for incidences  $x \pm y \pm z = c$ , and just rectangles for the other incidences.

The lists of vertices describing the polygons are arranged in clockwise or counterclockwise order depending on the sign of the incidence they belong to. Polygons' 3D vertex coordinates are expressed as integer numbers with respect to a double-resolution input regular grid.<sup>1</sup> The conversion between the 2D space of the MERGER structure and this 3D space can easily be done, given the incidence the primitives belong to and the original PPLANES index.

### 3.3.3. Inserting *T*-vertices and triangulating

At this point, the only computation to be performed, before proceeding with the next step of the algorithm, is the triangulation of the convex polygons. Triangulating a convex polygons is easy: the generation of a triangle strip can be performed by simply connecting the  $n$  vertices of the polygon in the sequence:

$$v_1, v_n, v_2, v_{n-1}, v_3, \dots$$

Unfortunately, a rendering problem may arise if the algorithm to reconstruct convex polygons does not take into account the neighboring ones. This is because some polygons can have vertices which are not present in the adjacent ones (the missing vertices are indicated by arrows in the example of Fig. 7d). This would lead to aliasing problems while Gouraud-shading the surfaces.

Before triangulating, we need to detect these vertices, called *T-vertices* in literature, and insert them in the boundary of the corresponding polygons. This is performed by using the 3D array VERTICES, with an entry for each possible vertex in the final triangulation. During the construction of each convex polygon, we store in VERTICES the vertices which survived the primitives' merging. The array of the vertices is an array of bits (to indicate whether the corresponding vertex is in the final tessellation or not) of dimensions  $(\lceil X/8 \rceil, 2Y, 2Z)^2$  bytes.

<sup>1</sup> In order to represent coordinates by means of integer numbers, we assume that the cell  $[x, y, z]$  of the input dataset be located at the address  $[2x, 2y, 2z]$ . In this way the centre of the input cell  $[0, 0, 0]$  can be expressed by the coordinates  $[1, 1, 1]$ .

<sup>2</sup> Again, the VERTICES array refers to a double-resolution grid.

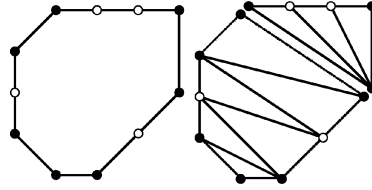


Fig. 9. Triangulating a polygon presenting T-vertices.

Once the merging for all PLANES has terminated, we search for possible missing T-vertices on the boundary of each polygon. Each of its edges is therefore tested with a sort of simple 3D rasterization algorithm: using appropriate steps and directions we verify whether a vertex to be inserted in the current edge exists in the VERTICES array. T-vertices are marked when inserted in the polygon's vertices list in order to be recognized in the next triangulation step.

With respect to the simple algorithm described above, the triangulation of a convex polygon having T-vertices requires a slightly different approach (Cignoni et al., 1996) in order to prevent the generation of zero-area triangles. The algorithm adopted in DiscMC (see Fig. 9) decomposes the convex polygon into one triangle strip and, at most, one triangle fan. The zigzag construction of the strip terminates when the part of the polygon to be triangulated reduces itself to a triangle in which just one side presents one or more T-vertices. This triangle is then converted into a triangle-fan. For our leading example, the final triangulation is shown in Fig. 7e. In it, due to the presence of T-vertices, two of the convex polygons have been split into strips and fans.

### 3.4. The interpolating step

The fastest way to conclude the algorithm is to generate a list of vertices, a list of normals, and a list of indexed triangle strips and fans. With respect to triangles strips and fans, the indexed ones are a more compact and efficient representation for polygonal meshes. The list of vertices can be simply built by reading all the *on* bits in the VERTICES array and transforming the corresponding integer coordinates into physical space coordinates. Normals can be computed, in terms of gradients as in the standard MC approach, on the vertices of the isosurfaces. Indexed triangle strips and fans can be generated by transforming integer vertex coordinates into indices to the vertices' list. DiscMC performs this step by means of an hashing indexing technique. Finally, indexed triangle strips and fans can be sent to the final renderer.

Though all the vertices are located on the mid-points of the cells' edges or in the cells' centres, the quality of the generated isosurfaces is good. We claim that, for many applications and for high-resolution datasets the quality is sufficient. Examples are shown in the next section.

However, if a greater approximation of the resulting isosurface is required, DiscMC can provide a more precise computation of the vertices' locations by a simple linear interpolation on the data values of the corresponding cell edges. This transformation assures that all of the vertices lying on edges will have, in the final surface, the same

positions of the corresponding vertices of the isosurface extracted by means of a standard MC algorithm. Also the vertices located in the cell centre undergo this interpolation process, but they do not have corresponding vertices in the MC isosurfaces. Centre linear interpolation is performed by moving the point on a cell diagonal having the extreme data values in the *on-off* configuration.

## 4. Results

DiscMC was evaluated in two different testing sessions, on two different machines. The goal of the first session was to measure the simplification capabilities of DiscMC and the percentage of computer times spent in each algorithm phase. The second session was devoted to a comparison of DiscMC with *standard* MC and surface simplification algorithms. This comparison was based on the results presented in (Lorenzen, 1995).

### 4.1. DiscMC performance evaluation

This first set of measures was obtained using an SGI O2 workstation, with one R5000 (180 MHz IP32) processor, 32 KB data and instruction caches, 96 MB main memory, running Irix 6.3.

We measured DiscMC performances on a number of different datasets and compared it with our implementation of the MC algorithm. This latter makes use of the same pyramidal data structure introduced in Section 3.2.3 and of a modified lookup table for the correct handling of the ambiguous configurations (Montani et al., 1994a).

Five datasets were used in the first testing session:

- *Femur* ( $256 \times 256 \times 116$ , voxel size  $dx = dy = 0.6dz$ ) is the CAT scanned head of a human femur (courtesy of the Istituto Ortopedico Rizzoli, Bologna, Italy);
- *Head* ( $256 \times 256 \times 34$ ,  $dx = dy = 0.5dz$ ) is the CAT scanned occipital region of a human head (courtesy of the Niguarda Hospital, Milan, Italy);
- *Bucky* ( $128^3$ ,  $dx = dy = dz$ ) is the electron density around a molecule of  $C_{60}$  (courtesy of AVS International Center);
- *Hydro* ( $127^3$ ,  $dx = dy = dz$ ) is the electron density around a molecule of  $H_2$ ;
- *Sphere* ( $65^3$ ,  $dx = dy = dz$ ) is the classic sphere dataset.

Table 2 reports the number of triangles generated by MC and DiscMC for the indicated thresholds (*thr*). The last column shows the percentage of reduction in the number of generated triangles obtained by DiscMC with respect to the MC algorithm. In our experiments, we obtained reduction percentages in the range 40–85%. Low reduction factors (for example, in the case of the *Bucky* dataset and threshold  $t = 231.9$ ) are obtained when the extracted isosurface results in many disjointed components with a few triangles each.

Table 3 reports timing results for our implementation of MC and DiscMC algorithms. Times (in CPU seconds) do not take into account I/O operations nor the time spent in the construction of the pyramid data structure. The algorithms can be properly compared because they share the pyramidal data structure for the localization of the active cells, the hashing indexing technique to transform triangle strips into indexed ones, and the function

Table 2  
Triangles produced by the MC and DiscMC algorithms

Dataset	Res.	Thr.	MC tri's	DiscMC tri's	$1 - \frac{\#DiscMC}{\#MC} \%$
Femur	$256^2 \times 116$	112.2	172,096	53,149	69.1
Head	$256^2 \times 34$	41.7	138,252	52,201	62.2
		105.1	250,369	87,562	65.0
Bucky	$128^3$	23.1	257,032	78,324	69.5
		127.8	183,480	62,508	65.9
		231.9	19,424	10,728	44.8
Hydro	$127^3$	40.1	124,736	32,496	73.9
		127.6	44,704	11,512	74.2
Sphere	$65^3$	746.5	28,904	6,908	76.1

Table 3  
Running times (in CPU seconds) of the MC and DiscMC algorithms

Dataset	Res.	Thr.	MC sec.	DiscMC sec.
Femur	$256^2 \times 116$	112.2	5.4	4.8
Head	$256^2 \times 34$	41.7	5.9	5.7
		105.1	6.8	7.0
Bucky	$128^3$	23.1	7.6	7.0
		127.8	6.6	6.4
		231.9	3.4	3.6
Hydro	$127^3$	40.1	4.5	3.9
		127.6	1.5	1.3
Sphere	$65^3$	746.5	0.69	0.54

to compute vertex locations and normals. For each of the sample datasets used, Table 4 reports the pyramid construction times.

Table 5 reports timing results for each phase of the DiscMC algorithm. Times (in CPU seconds) do not take into account I/O operations nor the pyramid construction time. They are split into *marching* times (localization of the active cells, extraction and storing of the geometric patterns), *merging* times (primitives' merging, construction of convex polygons, insertion of the missing *t*-vertices, triangulation, and transformation of triangles strips and fans into indexed ones), and *interpolating* times (exact location of the vertices and

Table 4  
Pyramid construction times

Dataset	Res.	Sec.
Femur	$256^2 \times 116$	5.98
Head	$256^2 \times 34$	1.69
Bucky	$128^3$	1.61
Hydro	$127^3$	0.77
Sphere	$65^3$	0.10

Table 5  
Running times of the three steps of the DiscMC algorithm

Dataset	Res.	Thr.	Marching sec.	Merging sec.	Interpolating sec.	Total sec.
Femur	$256^2 \times 116$	112.2	0.6	1.2	3.0	4.8
Head	$256^2 \times 34$	41.7	0.4	1.3	4.0	5.7
		105.1	0.8	2.6	3.6	7.0
Bucky	$128^3$	23.1	0.7	2.1	4.2	7.0
		127.8	0.6	1.6	4.2	6.4
		231.9	0.1	0.4	3.1	3.6
Hydro	$127^3$	40.1	0.3	0.7	2.9	3.9
		127.6	0.1	0.2	1.0	1.3
Sphere	$65^3$	746.5	0.07	0.12	0.35	0.54

normals computation). The table shows that a consistent part of the algorithm running time (between 50% and 80%) is spent in the *interpolating* step. This is mainly due to the normals computation, in fact the times do not present a valuable decrease if we give up the vertex relocation phase of the algorithm.

In our experiments, if a triangles' reduction factor greater than 50% is achieved, then the DiscMC total running time results generally lower than the MC one. This is because the time spent in the *merging* step is counterbalanced by the lower numbers of vertex locations and normals to be computed.

Figs. 10–12 allow a visual comparison between the MC and the DiscMC algorithms. They refer to the *Head* (threshold  $t = 41.7$ ), *Bucky* (threshold  $t = 127.8$ ) and *Femur* (threshold  $t = 112.2$ ) datasets, respectively. The lower image in Fig. 12 shows the result obtained if the vertex relocation phase of the algorithm is not performed.

Fig. 13 shows the approximation error of the DiscMC algorithm with respect to MC. The example refers to a magnified isosurface section extracted from the *Sphere* dataset.

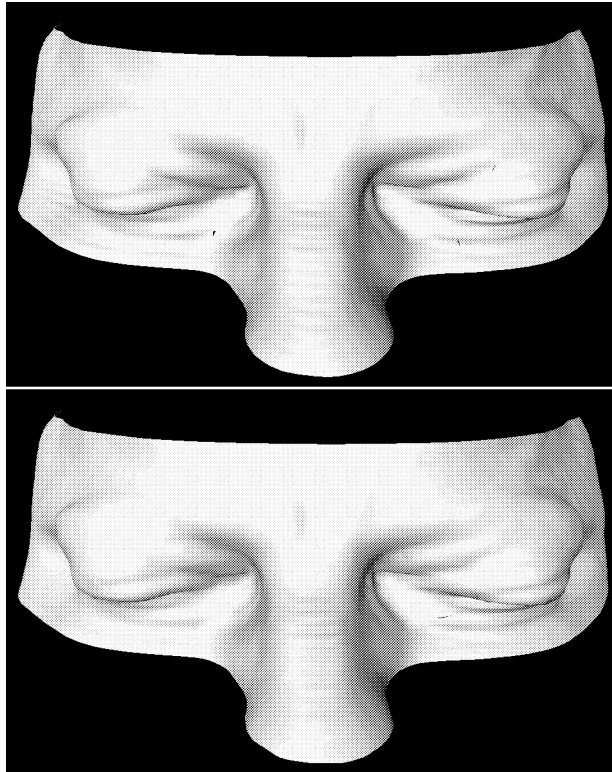


Fig. 10. *Head* dataset: MC (top) and DiscMC (bottom) output.

#### 4.2. Comparing DiscMC with published results

To compare the performances of DiscMC we decided to follow the advice given by Lorensen. In (Lorensen, 1995) he proposed a methodology for reporting algorithmic results, based on a precise description of the method adopted, the algorithm parameters, and the HW/SW configuration. This approach should allow fair comparisons of different solutions. Moreover, he reports a number of results related to isosurface fitting (by means of the MC algorithm (Lorensen and Cline, 1987)) and simplification (the simplification algorithm used is described in (Schroeder et al., 1992)) over a complex volumetric dataset, the Visible Human.

A comparison of DiscMC performances with standard MC and surface decimation techniques is therefore possible by comparing our results with those presented by Lorensen.

A second session of tests was therefore run on the same data used by Lorensen and adopting the same HW configuration: an SGI Onix workstation, with two R4400 (150 MHz IP19) processors, 16 KB data and instruction caches, 1MB secondary cache, 256 MB main memory, running Irix 6.2. We fitted isosurfaces from the National Library of

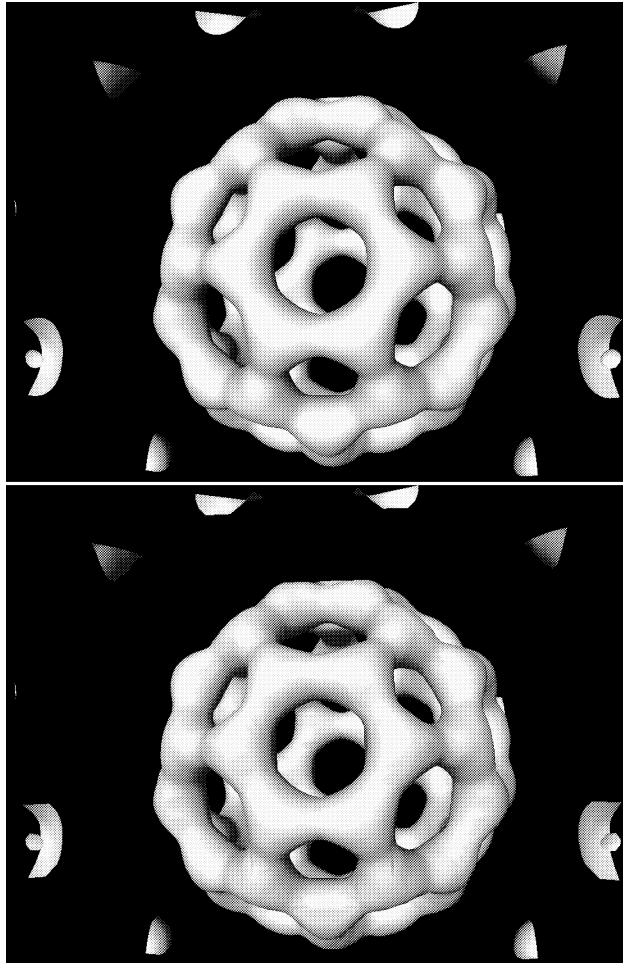


Fig. 11. *Bucky* dataset: MC (top) and DiscMC (bottom) output.

Medicine (NLM) “Visible Man” dataset,<sup>3</sup> using, for the fresh CT data, the same threshold (or density value) proposed by Lorensen for the bone ( $t = 1224.0$ ).

Table 6 compares the number of triangles produced using DiscMC with those obtained by Lorensen using MC, and left after surface decimation (Lorensen, 1995). Lorensen’s decimated meshes have been obtained with two iterations of the simplification algorithm and maximum error threshold equal to 0.0002 of the field of view.

Table 7 compares the CPU times needed by the different methods. In this case, we also report the running times of our implementation of the MC algorithm which makes use of the pyramidal data structure to speed up the localization of the active cells.

<sup>3</sup> [http://www.nlm.nih.gov/extramural\\_research.dir/visible\\_human.html](http://www.nlm.nih.gov/extramural_research.dir/visible_human.html).

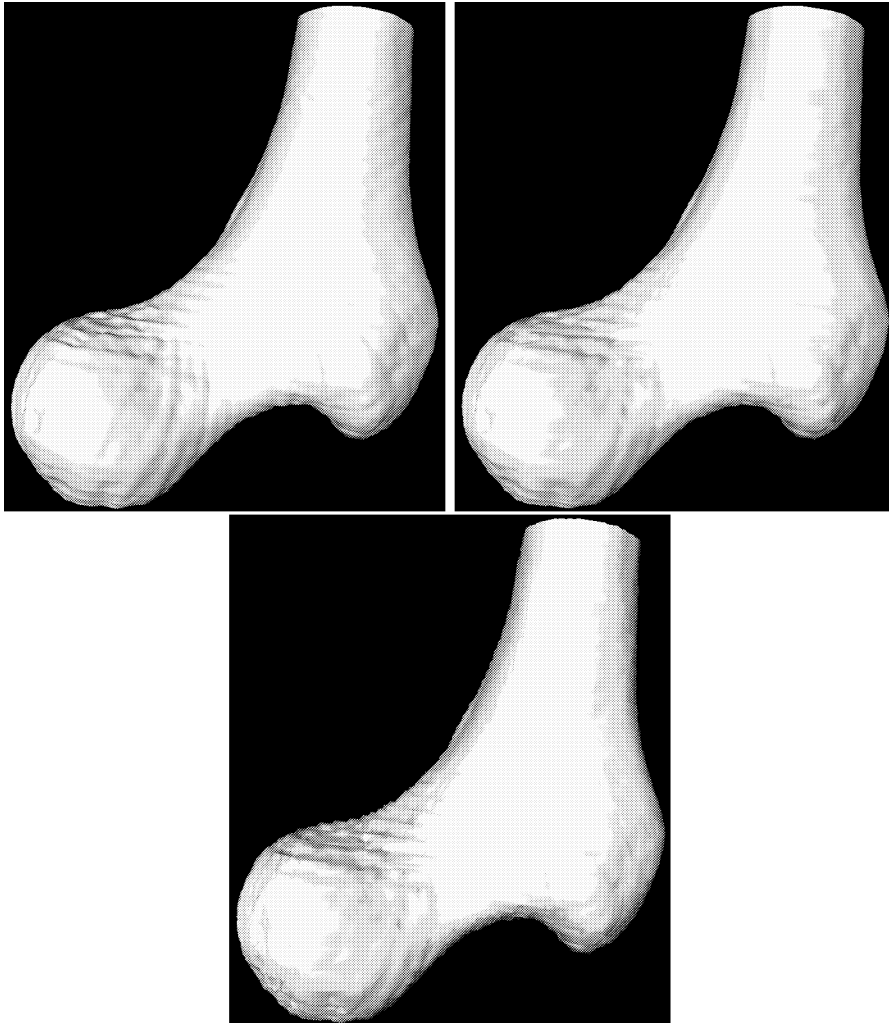


Fig. 12. *Femur* dataset: MC (left), DiscMC without vertex relocation (bottom), and DiscMC (right) output.

The results reported show that the merging phase of DiscMC is definitively faster than surface decimation. In fact, each decimation step reduces the mesh complexity by one element. This is a fairly complex phase: the decimation algorithm selects the vertex that has to be removed, removes all the triangles incident in it, and patches the resulting hole by

---

Fig. 13. *Sphere* dataset: a magnified wire-frame example of MC (top), DiscMC (center), and DiscMC without vertex relocation (bottom) output.

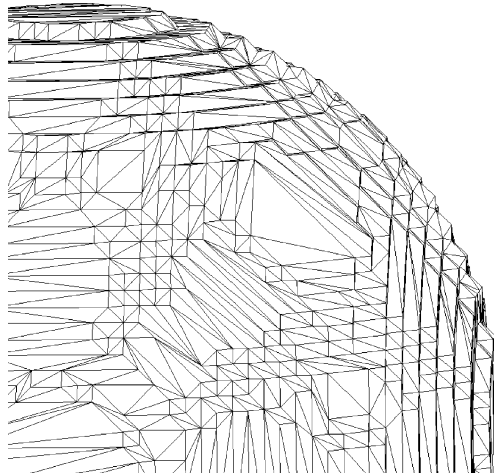
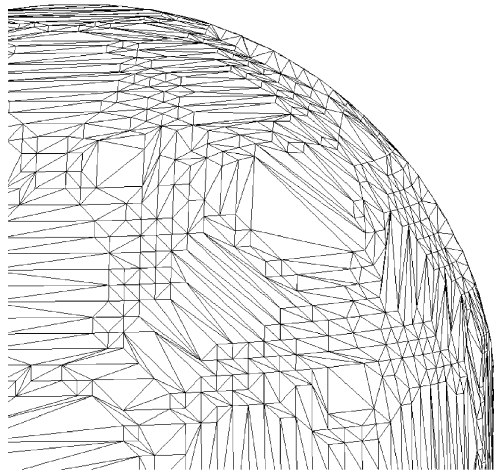
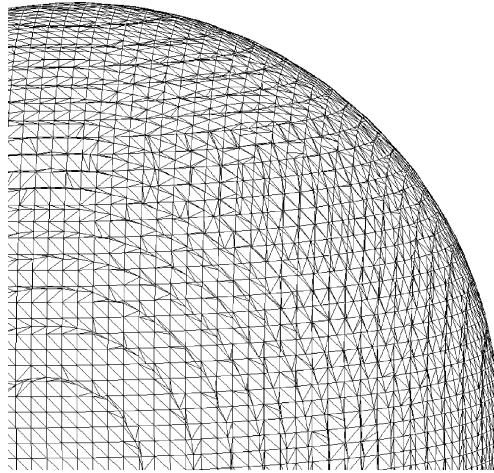


Table 6  
The MC and Decimation output size compared with the DiscMC one

Section	Res.	MC tri's	Decimation tri's	DiscMC tri's
2	$512^2 \times 83$	865,454	541,654	374,299
3	$512^2 \times 41$	193,102	137,717	84,118
4	$512^2 \times 55$	915,904	679,246	480,426

Table 7  
The MC and Decimation running times compared with the times of our MC and DiscMC algorithms

Section	Res.	Lorensen's MC sec.	Our MC sec.	Decimation sec.	DiscMC sec.
2	$512^2 \times 83$	93.7	15.2	135.4	17.4
3	$512^2 \times 41$	34.2	5.2	28.6	5.8
4	$512^2 \times 55$	77.9	16.2	125.0	22.7

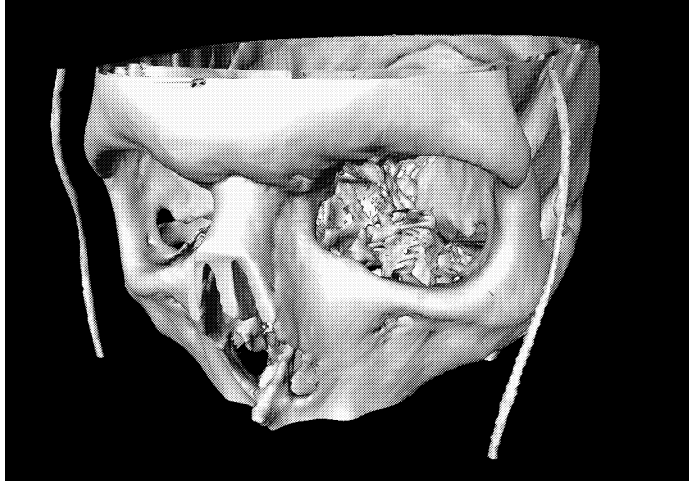


Fig. 14. Results from the NLM *Visible Man* dataset, Section 2, obtained by DiscMC for the threshold  $t = 1224$ .

computing a new local triangulation. The number of decimation steps obviously depends on the required simplification degree.

Fig. 14 shows the results obtained by DiscMC working on Section 2 of the NLM *Visible Man* dataset.

## 5. Conclusions

The results presented in the previous section demonstrate the high efficiency of the DiscMC algorithm. This is crucial because the design goal of DiscMC was to produce simplified meshes with high efficiency.

The efficiency of DiscMC is mainly due to three factors:

- (a) with respect to the usual isosurface fitting and simplification process, it avoids to perform computations (locations and normals) on the vertices which will be later removed;
- (b) thanks to the pyramid data structure, it avoids the analysis of empty cells;
- (c) it adopts a very efficient merging approach; the topological relations required for the merging phase are implicitly stored in the coding scheme adopted (facet shapes and incidences) and this reduces processing times and simplifies the algorithm implementation.

From an algorithmic point, an important characteristic of DiscMC is its geometrical robustness. Most of the process relies on the use of look up tables and combinatorial rules; none of the choices is based on geometrical computations, which might be affected by the limited numerical precision.

The results obtained also highlight the good performance of DiscMC in terms of obtainable simplification rates. DiscMC generally reduces the number of triangles to 1/3 of those returned by the MC algorithm.

The mesh generated by DiscMC is *approximated*, due to vertex simplification strategy adopted, but the error introduced is bounded. The error on each vertex depends on data resolution and is not greater than the largest cell edge.

We measured the effective DiscMC approximation error using *Metro* (Cignoni et al., 1998b), a tool designed to compare the difference between triangulated surfaces. *Metro* adopts an approach based on surface sampling and point-surface distance computation. We ran *Metro* on two meshes extracted from the *Hydro* dataset by DiscMC and MC. The maximum difference (i.e., the maximum distance between the two meshes) is 0.8054 cell edge units and the mean difference is 0.08183 cell edge units which is about 0.05% of the length of the diagonal of the input grid. These results are in accordance with the above assumption and also show that the mean error is lower than the given upper bound.

## Acknowledgements

DiscMC is part of *Surfactor 1.2*, available as public domain software at the Visual Computing Group WWW site of the CNUCE-IEI CNR Institutes in Pisa.<sup>4</sup> *Surfactor*, developed in OpenInventor and ViewKit, implements the MC and DiscMC (called *Precise DiscMC* in *Surfactor*) algorithms together with a version of DiscMC without vertex relocation. *Surfactor* works on 16 bits datasets and allows the user to interactively browse

---

<sup>4</sup> <http://vcg.iei.pi.cnr.it/homepage.html>.

the slices of the dataset and, thank to the pyramid data structure, to extract isosurfaces at lower resolutions.

The *Metro* tool is also available as public domain software at the same address.

Many thanks to Paolo Cignoni and Claudio Rocchini for their valuable help in developing and testing the Surfator's graphical user interface.

This work was partially financed by the Sardinian Regional Authorities and by the Progetto Coordinato *Modelli Multirisoluzione per la Visualizzazione di Campi Scalari Multidimensionali* of the Italian National Research Council. We also acknowledge the support received by HP Italy.

## References

- Algorri, M.E. and Schmitt, F. (1996), Mesh simplification, in: Computer Graphics Forum (Eurographics '96 Conference Proceeding), 15 (3), 78–86.
- Bajaj, C.L., Pascucci, V. and Schikore, D.R. (1996), Fast isocontouring for improved interactivity, in: Proc. 1996 Symposium on Volume Visualization, ACM Press, 39–46.
- Cignoni, P., Marino, P., Montani, C., Puppo, E. and Scopigno, R. (1997), Speeding up isosurface extraction using interval trees, IEEE Transactions on Visualization and Computer Graphics 3 (2), 158–170.
- Cignoni, P., Montani, C. and Scopigno, R. (1996), Triangulating convex polygons having T-vertices, J. Graphics Tools 1 (2), 1–4.
- Cignoni, P., Montani, C. and Scopigno, R. (1998a), A comparison of mesh simplification algorithms, Computers & Graphics 22 (1), 37–54.
- Cignoni, P., Rocchini, C. and Scopigno, R. (1998b), Metro: Measuring error on simplified surfaces, Computer Graphics Forum 17 (2), 167–174.
- De Hamer, M.J. and Zyda, M.J. (1991), Simplification of objects rendered by polygonal approximations, Computer & Graphics 15 (2), 175–184.
- Dürst, M.J. (1988), Letters: Additional reference to marching cubes, ACM Computer Graphics 22 (4), 72–73.
- Eck, M., De Rose, T., Duchamp, T., Hoppe, H., Lounsbery, M. and Stuetzle, W. (1995), Multiresolution analysis of arbitrary meshes, in: SIGGRAPH '95 Conference Proceedings, 173–181.
- Garland, M. and Heckbert, P. (1997), Surface simplification using quadric error metrics, in: SIGGRAPH '97 Conference Proceedings, 209–216.
- Gross, M.H., Staadt, O.G. and Gatti, R. (1996), Efficient triangular surface approximations using wavelets and quadtree data structures, IEEE Transactions on Visualization and Computer Graphics 2 (2), 130–144.
- Hamann, B. (1994), A data reduction scheme for triangulated surfaces, Computer Aided Geometric Design 11 (2), 197–214.
- Hansen, C.D. and Hinker, P. (1992), Massively parallel isosurface extraction, in: Proceedings of Visualization '92, IEEE Computer Society Press, 77–83.
- Hebert, D.J. and Kim, H.J. (1995), Image encoding with triangulation wavelets, in: Proceedings SPIE, Vol. 2569 (1), 381–392.
- Heckbert, P. and Garland, M. (1997), Survey of polygonal surface simplification algorithms, Technical Report, Carnegie Mellon University, Department of Computer Science.
- Hinker, P. and Hansen, C. (1993), Geometric optimization, in: Proceedings of Visualization '93, IEEE Computer Society Press, 189–195.

- Hoppe, H. (1996), Progressive meshes, in: SIGGRAPH '96 Conference Proceedings, 99–108.
- Hoppe, H., DeRose, T., Duchamp, T., McDonald, J. and Stuetzle, W. (1993), Mesh optimization, in: SIGGRAPH '93 Conference Proceedings, 19–26.
- Itoh, T. and Koyamada, K. (1995), Automatic isosurface propagation using an extrema graph and sorted boundary cell lists, *IEEE Transactions on Visualization and Computer Graphics* 1 (4), 319–327.
- Kalvin, A.D. and Taylor, R.H. (1996), Superfaces: Polygonal mesh simplification with bounded error, *IEEE Computer Graphics & Applications* 1 (3), 64–77.
- van Kreveld, M., van Oostrum, R., Bajaj, C., Schikore, D. and Pascucci, V. (1997), Contour trees and small seed sets for isosurface traversal, in: *Proceedings Thirteenth ACM Symposium on Computational Geometry (Theory Track)*, Nice, France, June 4–6, ACM Press, 212–219.
- Lachaud, J.O. (1996), Topologically defined isosurfaces, in: Miguet, S., Montanvert, A. and Ubeda, S., eds., *Proc. Conference Discrete Geometry for Computer Imagery '96*, Springer, Berlin, 245–256.
- Livnat, Y., Shen, H.W. and Johnson, C.R. (1996), A near optimal isosurface extraction algorithm for unstructured grids, *IEEE Transactions on Visualization and Computer Graphics* 2 (1), 73–84.
- Lorensen, W.E. (1995), Marching through the visible man, in: *Proceedings of Visualization '95*, IEEE Computer Society Press, 368–373.
- Lorensen, W.E. and Cline, H. (1987), Marching cubes: A high resolution 3D surface construction algorithm, in: *ACM Computer Graphics (Proceedings of SIGGRAPH '87)*, Vol. 21 (4), 163–170.
- Low, K.L. and Tan, T.S. (1997), Model simplification using vertex clustering, in: *1997 ACM Symposium on Interactive 3D Graphics*, 75–81.
- Montani, C. (1984), Region representation: Parallel connected stripes, *Computer Vision Graphics and Image Processing* 4 (11), 139–165.
- Montani, C., Scateni, R. and Scopigno, R. (1994a), A modified look-up table for implicit disambiguation of marching cubes, *The Visual Computer* 10 (6), 353–355.
- Montani, C., Scateni, R. and Scopigno, R. (1994b), Discretized marching cubes, in: Bergeron, R.D. and Kaufman, A.E., eds., *Proceedings of Visualization '94*, IEEE Computer Society Press, 281–287.
- Moore, D. and Warren, J. (1992), Compact isocontours from sampled data, in: Kirk, D., ed., *Graphics Gems III*, Academic Press, 23–28.
- Mueller, H. and Stark, M. (1993), Adaptive generation of surfaces in volume data, *The Visual Computer* 9 (4), 182–199.
- Natarajan, B.K. (1994), On generating topologically consistent isosurfaces from uniform samples, *The Visual Computer* 11 (1), 52–62.
- Nielson, G.M. and Hamann, B. (1991), The asymptotic decider: resolving the ambiguity in marching cubes, in: *Proceedings of Visualization '91*, ACM press, 83–90.
- Ning, P. and Bloomenthal, J. (1993), An evaluation of implicit surface tilers, *IEEE Computer Graphics and Applications* 13 (6), 33–41.
- Payne, B.A. and Toga, A.W. (1990), Surface mapping brain functions on 3D models, *IEEE Computer Graphics and Applications* 10 (2), 41–53.
- Puppo, E. and Scopigno, R. (1997), Simplification, LOD, and multiresolution—principles and applications, in: *EUROGRAPHICS '97 Tutorial Notes (ISSN 1017-4656)*, Eurographics Association, Aire-la-Ville (CH), PS97 TN4.
- Rossignac, J. and Borrel, P. (1993), Multi-resolution 3D approximation for rendering complex scenes, in: Falcidieno, B. and Kunii, T.L., eds., *Geometric Modeling in Computer Graphics*, Springer, Berlin, 445–465.
- Schroeder, W.J., Zarge, J.A. and Lorensen, W.E. (1992), Decimation of triangle mesh, in: *ACM Computer Graphics (SIGGRAPH '92 Conference Proceedings)*, Vol. 26 (2), 65–70.

- Shekhar, R., Fayyad, E., Yagel, R. and Cornhill, J.F. (1996), Octree-based decimation of marching cubes surfaces, in: *Proceedings of Visualization '96*, ACM Press, 335–342.
- Shu, R., Chen, Z. and Kankanhalli, M.S. (1995), Adaptive marching cubes, *The Visual Computer* 11 202–217.
- Stytz, M.R., Frieder, G. and Frieder, O. (1991), Three-dimensional medical imaging: Algorithms and computer systems, *ACM Computing Survey* 23 (4), 421–499.
- Van Gelder, A. and Wilhelms, J. (1994), Topological considerations in isosurface generation, *ACM Transactions on Graphics* 13 (4), 337–375.
- Wilhelms, J. and Van Gelder, A. (1992), Octrees for faster isosurface generation, *ACM Transaction on Graphics* 11 (3), 201–227.