Discretized Marching Cubes

C. Montani‡, R. Scateni*, R. Scopigno†

‡ I.E.I. – Consiglio Nazionale delle Ricerche, Via S. Maria 46, 56126 Pisa, ITALY

* Centro di Ricerca, Sviluppo e Studi Superiori Sardegna (CRS4), Cagliari, ITALY

† CNUCE – Consiglio Nazionale delle Ricerche , Via S. Maria 36, 56126 Pisa, ITALY

Abstract

Since the introduction of standard techniques for isosurface extraction from volumetric datasets, one of the hardest problems has been to reduce the number of triangles (or polygons) generated.
This paper presents an algorithm that considerably reduces the number of polygons generated by a Marching Cubes-like scheme without excessively increasing the overall computational complexity. The algorithm 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 polygons.
An experimental evaluation of the proposed approach on datasets related to biomedical imaging and chemical modelling is reported.

1 Introduction

The use of the *Marching Cubes* (MC) technique, originally proposed by W. Lorensen and H. Cline [7], is considered to be a standard approach to the problem of extracting isosurfaces from a volumetric dataset. Marching Cubes is a very practical and simple algorithm and many implementations are available both as part of commercial systems or as public domain software.
Despite its extensive use in many applications, it does have some particular shortcomings: *topological inconsistency* [1], algorithm *computational efficiency* and excessive output *data fragmentation*. Standard MC produces no consistent notion of object connectivity; the *local* surface reconstruction criterion used give rise to a number of topological ambiguities, and therefore MC may output surfaces which are not necessarily coherent. These shortcomings have been extensively studied [11] and solutions have been proposed [12, 15, 8].
MC computational efficiency can be increased by exploiting implicit parallelism (each cell can be independently processed) [4] and by avoiding the visiting and testing of empty cells or regions of the volume [14].
Excessive fragmentation of the output data can prevent interactive rendering when high resolution datasets are processed. What has changed since the technique was introduced seven years ago, has been the amount of data to be processed while extracting such surfaces. Equipments that can generate volumetric datasets as large as $512*512*[\leq 512]$ are now generally available, and we are on the way to achieving machines capable of producing $1024 * 1024 * [\leq 1024]$ datasets or, in other words, 1 Gigavoxel per dataset. Although an isosurface does not usually cross all the voxels, we can understand how easy it is to generate more than one million triangles per surface. State-of-the-art hardware is not yet fast enough to manipulate such masses of data in real time.
These obstacles gave rise to substantial research aimed at reducing the number of triangles generated by MC. The solutions proposed can be classified into **adaptive** techniques, where the cell size is locally adapted to the shape of the surface [10] or the dataset is organized into high and low interest areas and more primitives are produced in selected areas only; and **filtering** techniques, where facet meshes returned by a surface fitting algorithm are filtered in order to merge or eliminate part of them.
Filtering–based approaches can be classified as:

a) *coplanar facets merging*, in which facets are filtered by searching for and merging coplanar and adjacent facets [6];

b) *elimination of tiny facets*, where the irregularity of the surface produced is reduced by eliminating the tiny triangles produced when the iso-surface passes near a vertex or an edge of a cubic cell; this is accomplished by bending the mesh so that a number of selected mesh nodes will lie on the iso–surface and the tiny triangles will degenerate into single vertices. The solution is based on a modified iso-surface fitting algorithm and a filtering phase; 40% reductions in the number of triangles are reported [9];

c) *approximated surface fitting*, based on trading off data reduction for a reduction in the precision of the representation generated, using error criteria to measure the suitability of the approximated surfaces.
Schroeder et al. [13] proposed an algorithm based on multiple filtering passes, that by analysing locally the geometry and topology of a triangle mesh removes vertices that pass a minimal distance or curvature angle criterion. The advantage of this approach is that any level of reduction can be obtained, on the condition that a sufficiently coarse approx-

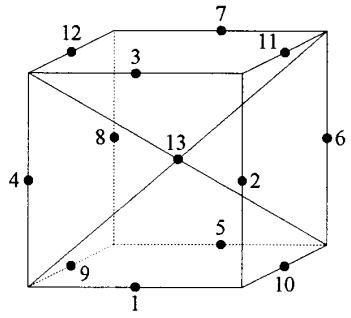

Figure 1: The set of different vertex locations produced by DiscMC.

imation threshold is set; reductions up to 90% have been obtained with an approximation error lower than the voxel size.
In another approach, by Hoppe et al. [5], mesh optimization is achieved by evaluating an energy function over the mesh, and then minimizing this function by either removing/moving vertices or collapsing/swapping edges.
Both approaches require a topological representation of the mesh to be decimated.

In this work we propose **Discretized Marching Cubes (DiscMC)**, an algorithm situated half-way between the *cuberille* method, which assumes constant value voxels and directly returns the voxels faces (orthogonal to the volume axes) [3], and the *cell interpolation* approach of MC. On the basis of two simple considerations, which both relate to data characteristics and visualization requirements, our solution leads to interesting reductions in output fragmentation by applying a very simple filtering approach. Moreover, the use of an unambiguous triangulation scheme [8] allows isosurfaces without topological anomalies to be obtained.

2 The Discretized Marching Cubes Algorithm

Given a binary dataset, linear interpolation is not needed to extract isosurfaces. When a cell edge in a binary dataset has both on and off corners, the midpoint of the edge is the intersection being looked for.
In a number of applications where approximated isosurfaces might be acceptable, the former assumption can be reasonably extended to *n–value* high resolution datasets. The maximal approximation error involved by adopting midpoint interpolation is 1/2 of the cell size, and in some applications the resolution of the dataset justifies such a lack of precision. Considering a $512 * 512 * [\leq 512]$ resolution, rendering the isosurface generated produces approximately the same images whether linear interpolation or midpoint selection is used.

Discretized Marching Cubes (DiscMC) is here proposed as an evolution of MC based on midpoint selection. The set of vertices that can be generated by DiscMC are shown in Figure 1: there are only 13 different spatial locations on which new vertices can be created (12 cell-edge midpoints plus the cell centroid). Moreover, applying midpoint selection in MC allows for a *finite set of planes* where the generated facets lie. We have only 13 different plane incidences onto which a facet can lie, and these are described by the following equations:

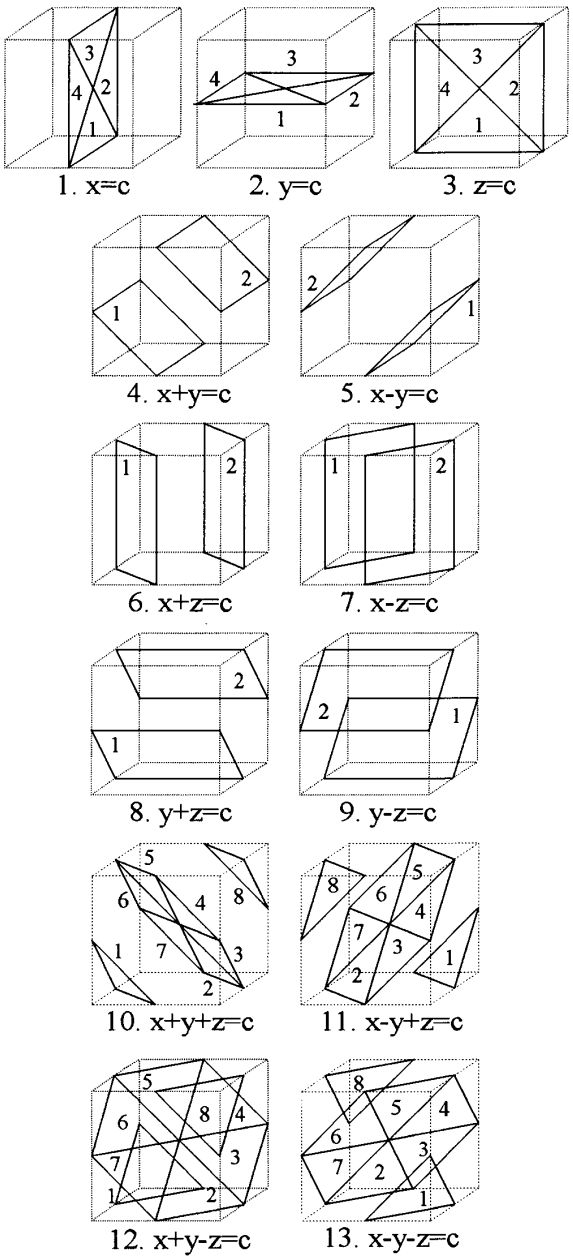

Figure 2: The facets returned by DiscMC for each different plane incidence.

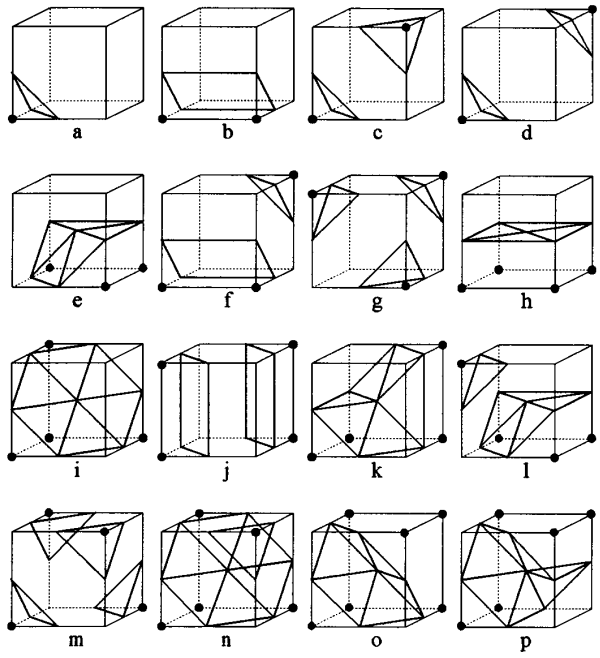

Figure 3: The sets of facets returned by DiscMC for each cell vertex configuration.

$$x = c,\ \ y = c,\ \ z = c,$$
$$x \pm y = c,\ \ x \pm z = c,\ \ y \pm z = c,$$
$$x \pm y \pm z = c.$$

As shown in Figure 2, for each incidence the algorithm generates a limited number of different facets.
The following considerations are the basis of our DiscMC algorithm:

a) each facet can be simply classified in terms of its shape and plane incidence;

b) the limited number of different plane incidences increases the percentage of coplanar adjacent facets and therefore drastically reduces the number of polygons returned while preserving small, but possibly significant, roughnesses;

c) the algorithm does not require interpolation of the surface intersections along the edges of the cells; this implies that it works in integer arithmetic (except for the computation of normals) at a higher speed than standard methods.

2.1 A new lookup table

For each on-off combination of the cell vertices (there are 256 different combinations), the standard MC lookup table (lut) codes the number of triangles produced and the cell edges on which these vertices lie.
DiscMC requires a simple reorganization of the standard MC lut. Midpoint selection means that the number of different facets returned by DiscMC is fixed, and we only have a constant number of different output primitives for each plane

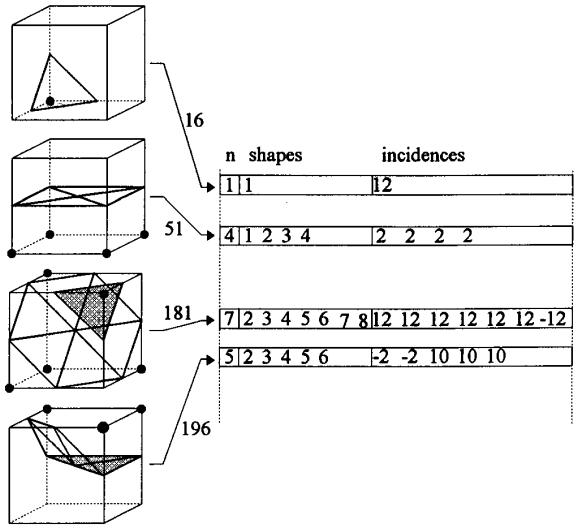

Figure 4: Some cell configurations and related DiscMC lookup table entries.

incidence: only right triangles are generated on planes $x = c$, $y = c$ and $z = c$ (Figures 2.1, 2.2 and 2.3); only rectangles on planes $x \pm y = c$, $x \pm z = c$ and $y \pm z = c$ (Figures 2.4, 2.5, 2.6, 2.7, 2.8 and 2.9); only equilateral triangles on planes $x \pm y \pm z = c$ (Figures 2.10, 2.11, 2.12 and 2.13). Moreover, using midpoint interpolation means that the geometrical location of facet vertices depends solely on the vertices configuration and the position of the cell in the dataset mesh.
Under these assumptions, the resulting facet set returned by DiscMC for each of the canonical MC configurations is reported in Figure 3. With respect to the original proposal by Lorensen and Cline we omit configuration 14 (this can be obtained by reflection from configuration 11, i.e. configuration k in Figure 3.). Furthermore, three more configurations have to be managed in order to prevent topological ambiguity (configurations n, o and p in Figure 3 [8]).
Each facet is coded in the DiscMC lut by using a *shape code*, which codifies the shape and position of the facet (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 facet lies. Geometrical information on the facet vertices is not explicitly stored in the DiscMC lut.
For each cell vertex configuration, DiscMC lut stores from zero up to seven facets, each represented by a shape code (1..8) and an incidence code (-13..13). We use *signed* incidences to store separately facets which lie on the same plane and have opposite normals direction (both in order to give an implicit representation of facet orientation and to fast facet search in the postprocessing merging phase).
Some cell configurations are graphically represented in Figure 4, together with the corresponding DiscMC lut entries.

2.2 Isosurface extraction

The isosurface reconstruction process returns intermediate results using a set of indexed data structures. The volume dataset is processed slice by slice. For each cell traversed by an isosurface, the DiscMC produces a set of facets by

means of the DiscMC lut. Each facet is coded by its shape, incidence and the index of the cell in which it lies (i.e. its geometical position).
In order to optimize the merging phase the facets produced are stored in a number of *hash tables*, one for each different incidence of the facets. Thus, 26 hash tables are used, and hash indexes are computed in terms of shape code and cell index.

2.3 Post-processing merging phase

The merging phase begins when the isosurfaces have been fitted. Each hash table is analyzed in order to search for adjacent faces, which by construction of the hash tables will also be iso-oriented and mergeable. Hash coding is chosen to allow a rapid search for adjacent facets (a nearly constant mean access time has been measured in a number of algorithm runs).
The merging algorithm does not work with the vertex coordinates of each merging polygon, but adopts Freeman's chains [2] as an intermediate representation scheme. In this scheme, a polygonal line is represented by the coordinates of the starting point of the chain and a set of directed links, that is, a set of relative displacements. This solution allows the unnecessary vertices to be rapidly eliminated.

The merging algorithm is simple and efficient. Due to the limited number of facet shapes and orientations, for each facet f and for each edge e of f the facet f' which might be adjacent on e to f is univocally determined. The algorithm is outlined in Figure 5 (an example is shown in Figure 6).

PUSH* verifies, for each edge pushed onto the *edgestack*, if an opposite edge exists on the stack, i.e. an edge with the same geometrical position but moving in the opposite direction. If this edge exists, mark both the edges as *connecting edges*. Marked edges will produce either *connecting links* (i.e. links which connect the starting point of the chain to the boundary of the region, or the boundary of the region to the boundary of the holes; see links 2 and 7 in the 15th tiles triple of Figure 6), or consecutive opposite links that have to be eliminated due to the reconstruction algorithm adopted.

The Merge algorithm main loop iterates until hash tables are empty. For each iteration of the first while loop, Merge produces the boundary of a region (anticlockwise in our implementation) and the boundaries of the holes (clockwise), if any. At the end of each iteration the boundaries of regions and holes are reconstructed by eliminating the marked links and, if necessary, by splitting the chain. Chains are then converted in the usual vertex–based representation.

The Merge algorithm uses a set of simple lookup tables which permit a general procedure to be designed irrespective of the type of the facets and the plane they belong to. These lookup tables store:

- the edges to be pushed onto the edgestack (depending on the starting point chosen);
- the edges to be pushed onto the edgestack when an adjacent facet has been found, or otherwise the link to be added to the Freeman's chain;
- the position (with respect to the current cell) of the cells to be inspected for adjacent facets.

In addition, through lookup tables we convert the chain links into relative displacements depending on the incidence plane we are examining.

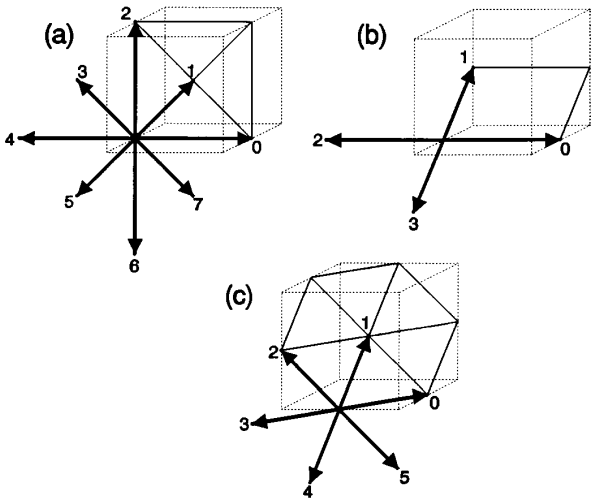

Figure 7: The links of Freeman's chains for (a) right triangles belonging to plane 3, (b) rectangles of plane 9, and (c) equilateral triangles of plane 12.

As previously introduced, with Freeman's chain representation scheme (Figure 7 shows the links used for three types of elementary primitives) unnecessary vertices can be removed by simply converting equal consecutive links into a single segment.
The *worst case* computational complexity of the merging phase is linear to the number of facets returned by the isosurface reconstructor. For each edge, the merger computes the potential adjacent facet and searches for such a facet in the hash table (a nearly constant time operation). In the worst case, when no mergeable facet pairs exist, the test is repeated e times for each facet f, with e the number of edges of facet f.

2.4 Vertex normals computation

Normals on the vertices of the isosurfaces extracted are needed in order to compute Gouraud or Phong shading. Normals can be computed during isosurface extraction (in terms of gradients [16], as in standard MC) or after the merging phase. In the current DiscMC implementation we computed vertex normals at the end of the merging process in order to avoid computing and storing a lot of unnecessary vertex normals. However, further processing of the volume data is thus needed.

3 Evaluation of results and conclusions

We tested DiscMC on a series of different datasets and compared results with a classic MC implementation. Table 1 reports the number of polygons generated and it refers to three datasets: *Sphere* is a voxelized sphere, *Buckyball* is the electron density around a molecule of C_{60} (courtesy of AVS International Centre) and *Head* is a CAT scanned dataset (courtesy of of Niguarda Hospital, Milan, Italy). The numbers of facets and vertices returned by Classic MC and DiscMC are reported in Table 1. DiscMC returns triangular ($3 - facets$), quadrilateral ($4 - facets$) or n-sided facets

```
Algorithm MERGE
    input HT_1, ..., HT_26:facet_hash_tables;
    output F:facet_list;
begin
    for each hash table HT_i do
        while HT_i is not empty do
            • extract a facet f from hash table HT_i;
            • select one of the vertices of f as the starting point of the
            current Freeman chain;
            • push the edges of the facet onto the edgestack (LIFO);
            {each edge is coded in the edgestack by the shape code of the current facet
            and the shape code and the cell coordinates of the potential adjacent facet.
            This notation will indicate, for each edge extracted from edgestack,
            the source facet and the adjacent facet to be searched for.}
            while edgestack is not empty do
                er:=POP(edgestack); {er:edge record}
                f_adj:=er.adjacent_facet;
                if facet f_adj is contained in HT_i
                    then
                        • extract the facet f_adj;
                        for each edge e_j ∈ f_adj such that e_j ≠ er do
                            PUSH*(edgestack, e_j); {PUSH*: see text in Section 2.3}
                    else
                        • add a link to the chain which is directed according to the current edge er;
                        • if the edge is a connecting edge, add a marked link to the current chain
                        (e.g. the link with a white arrow head in the 16th tile triple in Figure 6);

            • insert the current Freeman chain into F;
end algorithm.
```

Figure 5: Pseudocode of the Merge algorithm

$(n - facets)$; the respective numbers are in the rightmost three columns in Table 1.

Time comparison needs to be split into three steps: facet extraction, merging and generation of normals. The percentage of time spent in each stage of the computation varies from dataset to dataset; on average, it takes about 10% of the total time to extract facets, 85% to merge polygons, and about 5% to generate normals. The buckyball dataset and the head dataset, which are comparable in terms of voxel number, took around 2-3 minutes and 6-7 minutes, respectively, on an IBM RISC6000/550 workstation.

It is difficult to make a time comparison with other filtering approaches, because most of them do not report the running times but only the simplification percentages obtained. The mesh optimization approach by Hoppe et al. [5] is the only alternative technique which reports running times; the simplification of meshes (8000-18000 facets) with this method, which produces very good results indeed, took tens of minutes on a DEC Alpha workstation.
In the proposal by Schroeder et al. [13] running times are not reported, but the decimation phase is a much more complex task than the simple merging phase of DiscMC. In fact, the simplification of the mesh is obtained by multiple passes over the mesh. At each pass a vertex is selected for removal, all triangles that are incident on that vertex are removed, and the resulting hole is patched by computing a new local triangulation.
On the other hand, in the worst case, DiscMC has a complexity linear to the number of edges: for each edge of each facet, it searches for the adjacent facet on a hash list (a constant time and cheap operation), and makes an insertion/removal onto/from the edge stack.
The reduction in time complexity is significant, because the design goal of DiscMC was to give simplified meshes **with** high efficiency, to be used, for example, while searching for the correct threshold. Once this threshold has been selected, a more sophisticated method such as [13] can be used to obtain the best approximated mesh.

Another characteristic which differentiates DiscMC from other simplification approaches is that it does not entail managing a geo-topological representation of the triangle mesh. The topological relations are implicitly stored in the coding scheme used (facets shape and incidence) and this simplifies the implementation at the cost of single constant–time search into hash lists.

The results obtained and the good quality of the output images (the colour plates in Figures 9 and 11 were obtained with our algorithm, while the ones on Figures 8 and 10 were obtained with classic MC without mesh simplification) support our claim that Discretized Marching Cubes represents a valid tool for the rapid reconstruction and visualization of isosurfaces from medium and high resolution 3D datasets.
One of the most salient characteristics of the algorithm is that integer arithmetic is sufficient, and restricts the use of floating point computations to normals only. This is an important factor which enhances the overall performance.
Discretized Marching Cubes is both a valid solution for applications where the precision of the result is not critical or also as an intermediate solution to speed up the time needed to tune parameters, relegating to the final stage alone the use of techniques that are more precise in terms of visual results or geometrical approximation, such as ray tracing or standard MC.

(See color plates, page CP-32.)

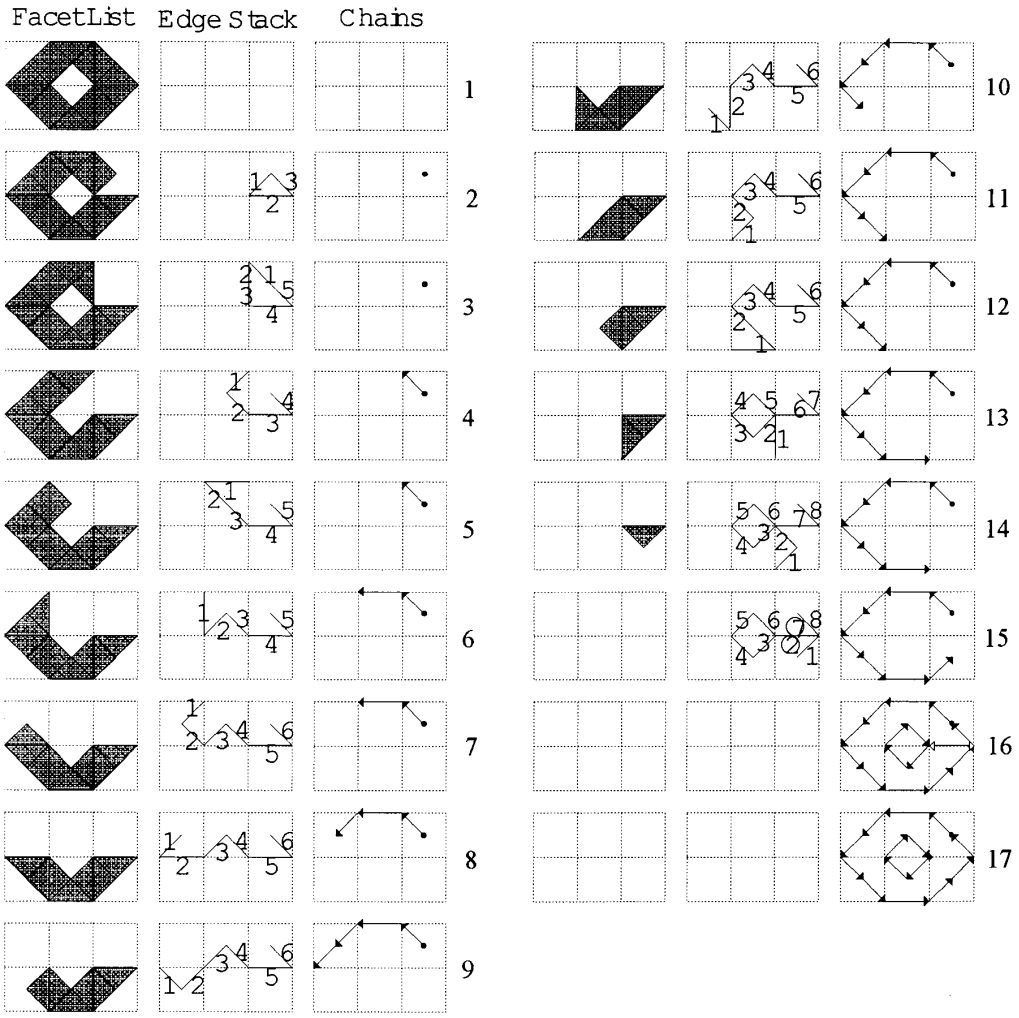

Figure 6: Steps required to merge a number of adjacent facets: for each iteration, the figure represents the facets remaining in the facet list (left tile), the edges present on the edge stack (centre tile) and the current Freeman chain (right tile). In the edge stack tiles, the label associated with the edges represents the order of insertion in the stack (1 is the top edge); edges which have a circled label represent *connecting edges*. In the chain tiles, arrows with a white end represent *connecting links*.

	Classic MC		*DiscMC*				
	# facets	*# vertices*	*# facets*	*# vertices*	*# 3-facet*	*# 4-facet*	*# n-facet*
Sphere (100^3)	37,784	18,556	5,501	9,594	0	5,167	334
Buckyball (128^3)	204,408	103,072	17,039	28,528	1,238	12,200	3,601
Head (256^2x33)	428,181	216,431	57,413	77,712	13,005	34,856	9,552

Table 1: The number of facets returned by the Discretized Marching Cubes and classic Marching Cubes algorithms on three different datasets.

4 Acknowledgements

This work has been partially carried out with the financial contribution of the Sardinian Regional Authorities.

References

[1] M. J. Dürst. Letters: Additional reference to "Marching Cubes. *ACM Computer Graphics*, 22(4):72–73, 1988.

[2] H. Freeman. Computer processing of line-drawing images. *ACM Computing Surveys*, 6:57–97, 1974.

[3] D. Gordon and J.K. Udupa. Fast surface tracking in 3D binary images. *Computer Vision, Graphics and Image Processing*, (45):196–214, 1989.

[4] C.D. Hansen and P. Hinker. Massively parallel isosurface extraction. In A.E. Kaufman and G.M. Nielson, editors, *Visualization '92 Proceedings*, pages 77–83. IEEE Computer Society Press, 1992.

[5] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Mesh optimization. *ACM Computer Graphics (SIGGRAPH '93 Conf. Proc.)*, pages 19–26, August 1-6 1993.

[6] A.D. Kalvin, C.B. Cutting, B. Haddad, and M.E. Noz. Constructing topologically connected surfaces for the comprehensive analysis of 3D medical structures. *SPIE Vol. 1445 Image Processing*, pages 247–259, 1991.

[7] W. Lorensen and H. Cline. Marching cubes: a high resolution 3D surface construction algorithm. *ACM Computer Graphics*, 21(4):163–170, 1987.

[8] C. Montani, R. Scateni, and R. Scopigno. A modified look-up table for implicit disambiguation of Marching Cubes. *The Visual Computer*, Vol.10, 1994, to appear.

[9] D. Moore and J. Warren. Compact isocontours from sampled data. In D. Kirk, editor, *Graphics Gems III*, pages 23–28. Academic Press, 1992.

[10] H. Muller and M. Stark. Adaptive generation of surfaces in volume data. *The Visual Computer*, 9(4):182–199, 1993.

[11] P. Ning and J. Bloomenthal. An Evaluation of Implicit Surface Tilers. *IEEE Computer Graphics & Applications*, 13(6):33–41, Nov. 1993.

[12] B.A. Payne and A.W. Toga. Surface mapping brain functions on 3D models. *IEEE Computer Graphics & Applications*, 10(2):41–53, Feb. 1990.

[13] W.J. Schroeder, J.A. Zarge, and W. Lorensen. Decimation of triangle mesh. *ACM Computer Graphics*, 26(2):65–70, July 1992.

[14] J. Wilhelms and A. Van Gelder. Octrees for faster isosurface generation. *ACM Computer Graphics*, 24(5):57–62, Nov. 1990.

[15] J. Wilhelms and A. Van Gelder. Topological considerations in isosurface generation. *ACM Computer Graphics*, 24(5):79–86, Nov 1990.

[16] R. Yagel, D. Cohen, and A. Kaufman. Normal estimation in 3D discrete space. *The Visual Computer*, 8:278–291, 1992.

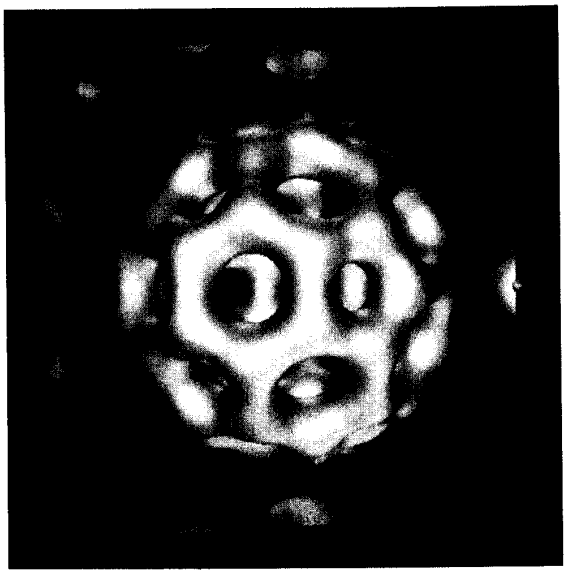

Figure 8: Isosurface reconstruction from the Buckyball dataset using standard MC (no mesh simplification).

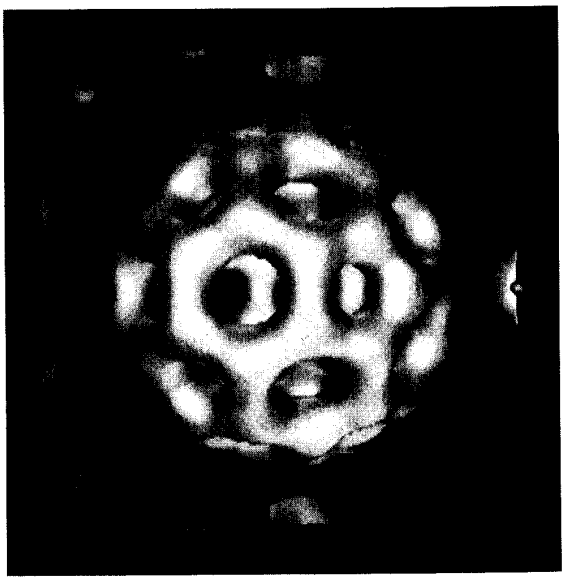

Figure 9: Isosurface reconstruction from the Buckyball dataset using DiscMC.