A Linear Algorithm for Constructing the Polygon Adjacency Relation in Iso-Surfaces of 3D Images

Serge Miguet
Laboratoire ERIC
Bât. L, Univ. Lyon II
5 av. P. Mendès-France
69676 Bron (France)
miguet@univ-lyon2.fr

Jean-Marc Nicod*
LIB
IUT Belfort-Montbeliard
BP 527 Rue Engel Gros
90016 Belfort Cedex
Jean-Marc.Nicod@iut-bm.univ-fcomte.fr

David Sarrut
Laboratoire ERIC
Bât. L, Univ. Lyon II
5 av. P. Mendès-France
69676 Bron (France)
dsarrut@univ-lyon2.fr

Abstract

This paper proposes an optimal algorithm for constructing the surface adjacency relation in a list of polygons extracted from 3D medical images. The discrete nature of data allows us to build this adjacency relation in a time proportional to the number of triangles $\mathcal{T}$. We have payed a special attention on the memory requirements, since our method takes as input the surface extracted by the Marching-Cubes algorithm and does not make reference to the initial 3D dataset. Moreover, no additional temporary storage is needed to compute the relation.

1 Introduction

Since the development of 3D medical scanning devices, a tremendous number of techniques have been proposed for reconstructing, processing and visualizing the anatomical data. One of the most used approach for understanding the 3D structure of objects consists in extracting iso-surfaces from the volume. The geometric description of these data can then be visualized by the help of classical rendering algorithms, using various lighting and shading models.

The Marching-Cubes algorithm (MC) is probably the most popular of these surface-based techniques, and is widely used in medical imaging as well as in

*This research has been done while the author was member of ERIC, Université Lyon-2

other application domains (geoscience, biology, molecular systems, etc). The input dataset consists in a 3D regular mesh whose elements (the voxels) have gray-level values depending on the acquisition device (MRI, CT, etc). In its original version [LC87], the algorithm generates a list of polygons (triangles) corresponding to a threshold value. The extracted surface separates the voxels having value above and below this threshold. Despite this list being sufficient for visualization, some processing for geometric modeling, topological considerations or image analysis need more information such as connected surface components, surface orientation or curvature, inner volume, total area, and so on. All of these kernels need to access in a simple way to the neighborhood of a given polygon. Although this adjacency relation could be computed by the MC itself, it would need an important amount of additional memory and computing time.

The main goal of this paper is to propose an efficient algorithm for constructing the surface adjacency relation, taking as input the original extracted surface, without using the initial 3D dataset. We show that our algorithm has a linear (optimal) complexity in the number of polygons of the surface, without requiring additional memory. Experimental results on medical images containing more than 1 million triangles fully demonstrate the efficiency of our approach, since the time for constructing the adjacency relation is less than 10 seconds.

2 Statement of the problem

The principle of the MC algorithm is to consider a cell made by 8 input samples. If the iso-surface intersects this cell, we build this intersection with triangles. The whole surface is made by moving this cell in the image successively along the x, y, and z axis (see a part of such a surface on figure 3).

The data generated by MC consists of two lists: a list of vertices and a list of triangles (solid tables in figure 1). Each vertex is represented by three position coordinates and three coordinates of the vector normal to the surface . One triangle is a triple of integers corresponding to the three vertex indices in the first list. This allows to store the surface in a compact and non-redundant way [MN95, ZN94]. Our goal is to build the surface adjacency relation $\mathcal{R}$ such that two distinct triangles t and t' are in relation by $\mathcal{R}$ if and only if they have an edge in common. We use a slightly modified version of MC as proposed in [Lac96], that has been proved to generate simple, oriented and closed surfaces. Thanks to these properties, an edge is shared by two and only two faces, which implies that a given triangle of the surface has exactly three neighbors. The adjacency relation will thus be stored by adding to each triangle the indices of its three neighbors (grayed table in figure 1).

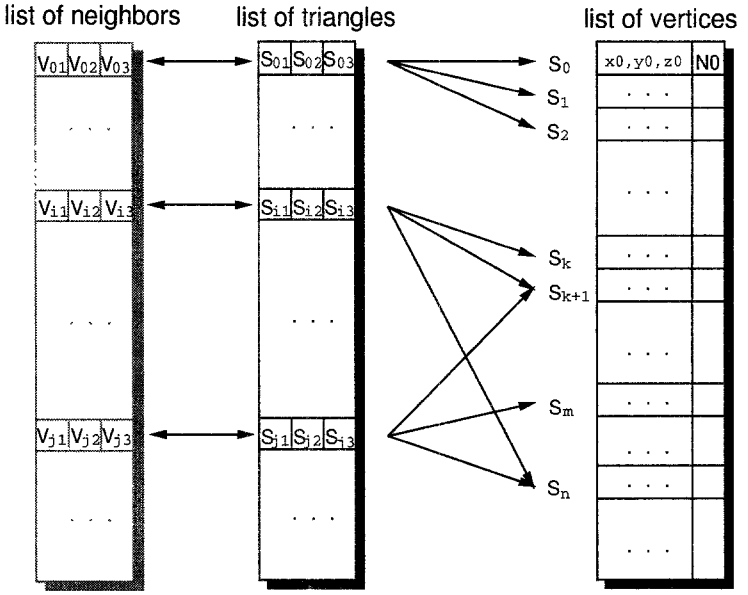


Figure 1: Output data structure of the MC algorithm

3 Algorithms for adjacency construction

In this section, we introduce two different adjacency construction algorithms that require no additional memory. Indeed, it would be possible to use a 3D data structure or an additional list associated with vertices to store the relation, but the important needs of memory make these solutions practically unusable with real medical images.

The first algorithm we present uses an intuitive approach, but its complexity makes it unsuitable for surfaces with a lot of triangles. We then explain our modified algorithm, that is proved to be optimal in section 3.3.

3.1 Notations

- let t be a triangle of the surface,
- let $\mathcal{C}(t) = (i, j, k)$ be the integer coordinates of the cell containing t,
- let $\mathcal{I}(t)$ be position of t in the triangles list,
- let ω be the maximum number of triangles that might be generated in one cell. $\omega = 6$ in the version of MC algorithm we use.

- let n be the average side of the 3D dataset. We assume thus that we have $O(n^3)$ voxels.
- let $\mathcal{T}$ be the length of the triangles list.

3.2 The naive algorithm

The principle of this algorithm is to find the neighbors of each triangle t, by searching in the remainder of the list, three other triangles having an edge in common with t. The number of operations done on each triangle is related to the maximal distance Δ in the list, between two adjacent triangles. An obvious upper bound on Δ is $O(\mathcal{T})$ leading to a $O(\mathcal{T}^2)$ complexity for the algorithm. In the same way, an obvious lower bound on Δ is $O(1)$ leading to a $\Omega(\mathcal{T})$ lower bound complexity. Although, we can exhibit families of surfaces for which these upper and lower bounds are reached, these bounds can be tightened for real life datasets. The idea of the improvement is to give more realistic bounds on Δ.

We have to make the following hypotheses on our 3D images:

H1: the number of surface elements is a quadratic function of n, i.e. $\mathcal{T} = \Theta(n^2)$.

H2: the number $\mathcal{T}_k$ of surface elements generated in each layer k is bounded by a linear function of n, i.e. $\mathcal{T}_k = O(n)$.

These hypotheses exclude mathematically generated families of images, such as fractals that can generate $\Theta(n^3)$ surface elements, or wireframe objects that can lead to $O(n)$ surface elements only.

Theorem 1

The naive algorithm has a $O(\mathcal{T}^{\frac{3}{2}})$ complexity on images verifying hypotheses H1 *and* H2.

Proof:

- *from* H1, *we have:* $\mathcal{T} = \Theta(n^2) \Rightarrow \exists \alpha > 0 \mid \alpha n^2 \leq \mathcal{T}$, *thus*

$$n \leq \sqrt{\frac{\mathcal{T}}{\alpha}} \tag{1}$$

- *from* H2, *we have:*

$$\mathcal{T}_k = O(n) \Rightarrow \exists \beta > 0 \mid \mathcal{T}_k \leq \beta n, \tag{2}$$

- *let t be a triangle generated on layer k. The number of triangles we have to scan to find the neighbors of t can be bounded by the number of triangles generated on both layers k and $k+1$:*

$$\Delta \leq \mathcal{T}_k + \mathcal{T}_{k+1}$$

from equations (1) and (2), we have:

$$\Delta \leq 2\beta\sqrt{\frac{\mathcal{T}}{\alpha}},$$

thus $\Delta = O(\mathcal{T}^{\frac{1}{2}})$.

An upper bound on the complexity of the naive algorithm is thus $O(\Delta \times \mathcal{T}) = O(\mathcal{T}^{\frac{3}{2}})$. □

Nevertheless, this complexity is still too high to use this algorithm for a surface coming from 3D medical images ($\mathcal{T} > 10^6$). So we have to propose an other way to construct the adjacency.

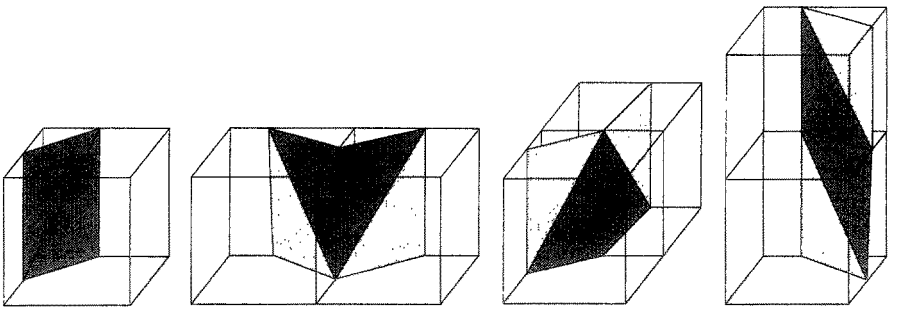
3.3 A linear adjacency construction algorithm

The idea is now to avoid the entire scan of the triangle list from the current position. We propose to start the scan from different positions defined by the previous searches. Let t be a triangle of the surface belonging to cell $\mathcal{C}(t) = (i, j, k)$. The neighbors t' of t that are still unknown can be found in four different cells only, that are $\mathcal{C}_I = \mathcal{C}(t)$ itself or the three 6–neighboring cells of $\mathcal{C}(t)$ that have not yet been scanned, i.e. $\mathcal{C}_X = (i+1, j, k)$, $\mathcal{C}_Y = (i, j+1, k)$ and $\mathcal{C}_Z = (i, j, k+1)$. The corresponding subscript I, X, Y or Z is called the *direction* of the neighbor t' and is noted $\mathcal{D}$ in the following. This direction can easily be determined from the coordinates of the edge common to t and t'. Figure 2 illustrates these four cases. An additional fifth case noted $\mathcal{D} = \emptyset$ will be used in section 4 when the triangle has already been scanned by the algorithm, because it was located before t in the list.

In the case where $\mathcal{D} = I$ or $\mathcal{D} = X$, the neighbor t' is known to be close to t in the list and can be searched exhaustively in a constant time. It is only if $\mathcal{D} = Y$ or $\mathcal{D} = Z$, that we use an optimization.

Even if we do not know $\mathcal{I}(t')$, we know the exact position $\mathcal{I}(u')$ of the last triangle we searched for (from a given triangle u) in direction $\mathcal{D}$. The main idea of our algorithm consists in starting the search of t' in a neighborhood of u'.

The algorithm is justified by the following result:



(a) $\mathcal{D} = I$ (b) $\mathcal{D} = X$ (c) $\mathcal{D} = Y$ (d) $\mathcal{D} = Z$

Figure 2: the 4 configurations corresponding to the 4 values of $\mathcal{D}$.

Theorem 2

Let t be the current triangle in the list known to have a neighbor t' in direction $\mathcal{D}$ ($\mathcal{D} = Y$ or $\mathcal{D} = Z$), and let u be the last scanned triangle having had a neighbor u' in the same direction $\mathcal{D}$. The following property holds:

$$\mathcal{I}(t') > \mathcal{I}(u') - \omega$$

Proof:

The different cells (i, j, k) are implicitly ordered by the scanning performed by MC. Following this order, the number of cells separating $\mathcal{C}(u)$ and $\mathcal{C}(t)$ is equal to the number of cells separating $\mathcal{C}(u')$ and $\mathcal{C}(t')$.

- *If u and t have been generated in the same cell (i.e. $\mathcal{C}(u) = \mathcal{C}(t)$), then u' and t' are also generated in the same cell (i.e. $\mathcal{C}(u') = \mathcal{C}(t')$). We do not know if u' is generated before t', but since there are at most ω triangles per cell, it is sufficient to start the search of t' from index $\mathcal{I}(u') - \omega + 1$.*
- *Else (i.e. $\mathcal{C}(u) \neq \mathcal{C}(t)$), the fact that u is before t in the list implies that $\mathcal{C}(u)$ was scanned before $\mathcal{C}(t)$ by MC. We deduce that $\mathcal{C}(u')$ and $\mathcal{C}(t')$ were also scanned in that order which allows to conclude that $\mathcal{I}(t') > \mathcal{I}(u')$.*

In these two cases, it is easy to check that the announced result is verified.

The only data structure needed to control this adjacency construction algorithm is reduced to three addresses in the triangles list: the current index $\mathcal{I}(t)$, and the last values $\mathcal{I}_Y$ and $\mathcal{I}_Z$ of $\mathcal{I}(u')$, corresponding respectively to $\mathcal{D} = Y$ and $\mathcal{D} = Z$.

The figure 4 represents a part of the triangles list generated from the example shown in figure 3. In this case, the triangle t has one of its neighbors, named t', in the direction $\mathcal{D} = Y$. Before storing the relation between t and t', $\mathcal{I}_Y$ was equal to $\mathcal{I}(u')$ since (u, u') was the last couple of triangles in direction $\mathcal{D} = Y$. t' could thus be searched from the position of u' instead of the one of t as in the naive algorithm. After having established the relation between t and t', the new value of $\mathcal{I}_Y$ is $\mathcal{I}(t')$.

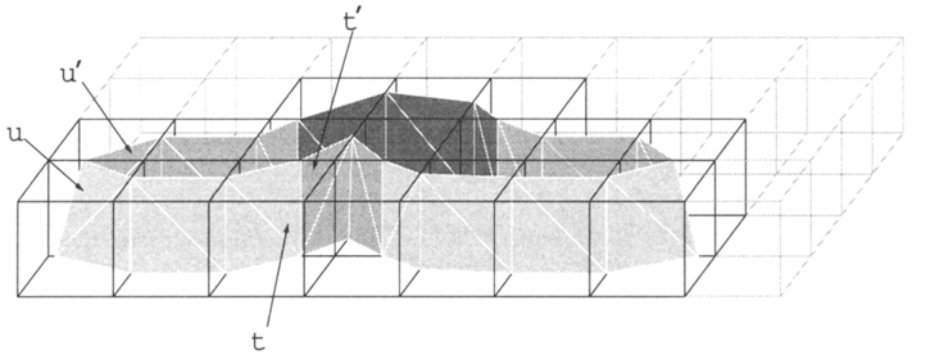


Figure 3: An example of a generated surface in a layer.

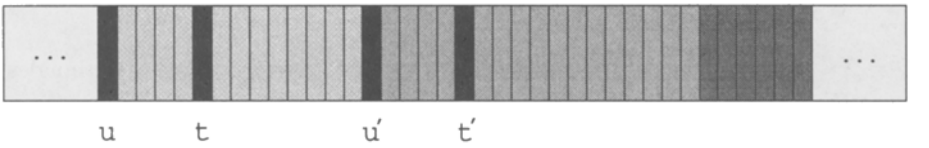


Figure 4: A example of a triangle list generated from the previous example

4 Complexity analysis

In this section, we present a proof of the linearity of the algorithm we have presented. The algorithm scans the whole list of triangles, and searches for the neighbors of the current one. Although some of these searches are costly (and can be as expensive as in the naive algorithm), we will show that they occur seldom enough to be amortized by many constant cost of other searches. This

is a typical proof using the amortized analysis. We use the potential method of D. D. Sleator, with the notations of [CLR90]:

Let D_i be the data structure at the i^{th} step of the algorithm. This method associates to the data structure a potential function Φ who maps each data structure D_i to a real number $\Phi(D_i)$. Let c_i be the actual cost of the i^{th} operation. By definition, the amortized cost $\hat{c}_i$ of the i^{th} operation is : $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$.

The main property used in amortized analysis is that the total amortized cost after n steps is equal to the total actual cost of these n steps plus the increase of potential:

$$\sum_{i=1}^{n} \hat{c}_i = \sum_{i=1}^{n} c_i + \Phi(D_n) - \Phi(D_0) \tag{3}$$

It is sufficient to check that the final potential is larger than the initial one, to guarantee that the total amortized cost is an upper bound of the total actual cost.

The i^{th} step of the algorithm consists in the search of one neighbor t' of a given triangle t. The actual cost of searches depends on the direction $\mathcal{D}$ of t':

$\mathcal{D} = \emptyset$: this is the notation we use when t' has already been scanned because located before t in the list. No search is needed, and we can let $c_i = 1$.

$\mathcal{D} = I$: t and t' are in the same cell, with distant of at most $\omega - 1$ triangles. We have thus $c_i < \omega$.

$\mathcal{D} = X$: t' is in the cell following t, with distant of at most $2\omega - 1$ triangles. We have thus $c_i < 2\omega$.

$\mathcal{D} = Y$: the search for t' starts from index $\mathcal{I}_Y$. Let Δ_Y be the number of scanned triangles until t' is reached. We have thus $c_i = \Delta_Y$.

$\mathcal{D} = Z$: the search for t' starts from index $\mathcal{I}_Z$. Let Δ_Z be the number of scanned triangles until t' is reached. We have thus $c_i = \Delta_Z$.

We are now ready to announce the main result of the paper:

Theorem 3

The algorithm described in section 3.3 has a linear complexity in the number of triangles

Proof:

The expression of Φ we use for the potential method is:

$$\Phi(D_i) = i - \mathcal{I}_Y - \mathcal{I}_Z \tag{4}$$

It is easy to check that this potential is null at the beginning of the algorithm, and positive at the end. Indeed, there are $\mathcal{T}$ triangles having each 3 edges. The last value of i is thus $3\mathcal{T}$, and $\mathcal{I}_Y$ and $\mathcal{I}_Z$ are smaller than $\mathcal{T}$.

Let us now compute the increase of potential associated to the five possible values for $\mathcal{D}$:

- $\mathcal{D} = \emptyset$, $\mathcal{D} = I$ **or** $\mathcal{D} = X$: *in these three cases, $\mathcal{I}_Y$ and $\mathcal{I}_Z$ are not modified, thus $\Phi(D_i) - \Phi(D_{i-1}) = 1$,*
- $\mathcal{D} = Y$: *the new value of $\mathcal{I}_Y$ corresponds to the position of t'. It has thus been increased of Δ_Y, so $\Phi(D_i) - \Phi(D_{i-1}) = 1 - \Delta_Y$,*
- $\mathcal{D} = Z$: *the new value of $\mathcal{I}_Z$ corresponds to the position of t'. It has thus been increased of Δ_Z, so $\Phi(D_i) - \Phi(D_{i-1}) = 1 - \Delta_Z$,*

By adding the actual cost with this increase of potential, we check that the amortized cost of the each of the five operations is $O(1)$. We can thus deduce that the $3\mathcal{T}$ operations are done in complexity $O(\mathcal{T})$.

□

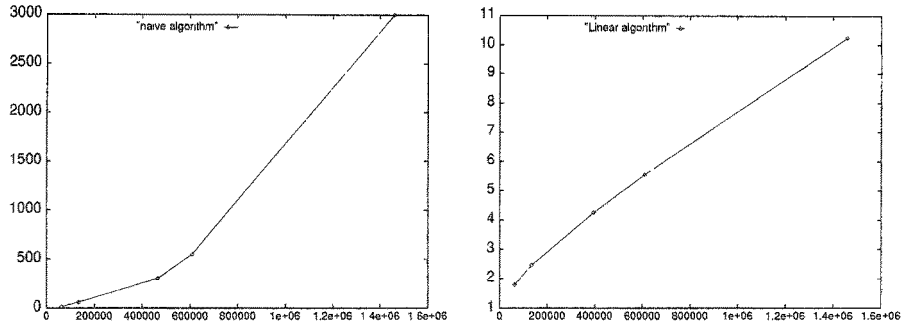
Since any algorithm for this problem has to visit at least once each triangle, our approach is optimal.

5 Experiments and results

In this section, we show experiments on the implementation of our two approaches for constructing the polygon adjacency relation.

In figure 5, we test the two algorithms on several surfaces extracted from 3D medical images. The computation time, for both figures, is obtained with the same set of surfaces. We can observe experimentally the linearity of our approach. For the surface made with 1.5×10^6 triangles, the time decreases from 3000 seconds to 10 seconds.

Figure 6, shows graphically the benefits of our improvement: we plot the cumulated cost of searches as a function of the displacement Δ, performed to



(a) naive algorithm (b) linear algorithm

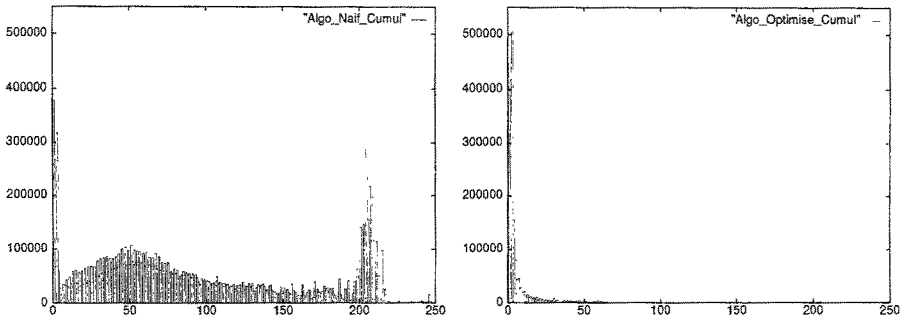
Figure 5: Comparison of the computation time between two different algorithms for constructing the polygon adjacency relation.

find the neighbor. The surface of the histograms correspond to the cost of the algorithm. We clearly observe three modes in the first histogram 6(a) corresponding to the three directions of searches $\mathcal{D} = I, X$, $\mathcal{D} = Y$ or $\mathcal{D} = Z$. These modes completely disappear in figure 6(b) meaning that most of the searches are completely local.

6 Conclusion

We have introduced an optimal algorithm for constructing the adjacency relation in a triangles list computed by the Marching-Cubes algorithm. The main advantage of this technique is to require no additional memory, which is a crucial point in 3D medical imaging. We have used amortized analysis to prove the optimal complexity of our algorithm. Numerical experiments demonstrate practically the efficiency of the approach.

A parallel version of this algorithm has been developed and integrated in our parallel 3D medical imaging environment [Mig95]. It has been chained with a parallel connected components algorithm, adapted from Perroton [Per94]. This will make it possible to interactively select connected objects from a 3D visualization, or to initiate a segmentation algorithm based on active surfaces as proposed in [LB94].



(a) naive algorithm

(b) linear algorithm

Figure 6: Distribution of the cumulative cost for each different displacement in the list.

References

[CLR90] Thomas Cormen, Charles Leiserson, and Ronald Rivest. *Introduction to Algorithms.* MIT Press, 1990.

[Lac96] Jacques-Olivier Lachaud. Topologically defined isosurfaces. In Serge Miguet, Annie Montanvert, and Stéphane Ubéda, editors, *Discrete Geometry for Computer Imagery*, volume 1176 of *Lecture Notes in Computer Science*, pages 245–256. Springer, November 1996.

[LB94] Jacques-Olivier Lachaud and Eric Bainville. A discrete model following topological modifications of volumes. In Jean-Marc Chassery and Annick Montanvert, editors, *Discrete geometry for computer imagery*, pages 183–194, Grenoble (France), September 1994.

[LC87] William E. Lorensen and Harvey E. Cline. Marching cubes : a high resolution 3D surface construction algorithm. *Computer Graphics*, 21(4):163–169, July 1987.

[Mig95] Serge Miguet. Un environnement parallèle pour l'imagerie 3D. Mémoire d'habilitation à diriger des recherches, Laboratoire de l'informatique du Parallélisme, 46 allée d'Italie, 69364 Lyon Cedex 07, December 1995.

[MN95] Serge Miguet and Jean-Marc Nicod. An optimal parallel iso-surface extraction algorithm. In *Fourth International Workshop on Parallel Image Analysis (IWPIA '95)*, pages 65–78. Laboratoire de l'Informatique du Parallélisme, ENS Lyon (France), December 1995.

[Per94] Laurent Perroton. *Segmentation Parallèle d'Images Volumiques.* PhD thesis, Ecole Normale Supérieure de Lyon, December 1994.

[ZN94] Meiyun Zheng and H.T. Nguyen. An Efficient Parallel Implementation of the Marching-cubes Algorithm. In L. Decker, W. Smit, and J.C. Zuidervaart, editors, *Massively Parallel Processing Applications and Development*, pages 903–910, 1994.