Decimation of Isosurfaces with Deformable Models

H. Delingette

I.N.R.I.A.
06902 Sophia-Antipolis Cedex, BP 93, France

Abstract

For many medical applications including computer-assisted surgery, it is necessary to perform scientific computations, such as mechanical deformation, on anatomical structure models. Such patient-based anatomical models are often extracted from volumetric medical images as isosurfaces. In this paper, we introduce a new algorithm for the decimation of isosurfaces based on deformable models. The method emphasizes the creation of mesh of high geometric and topological properties well suited for performing scientific computation. It allows a close control of the distance of the mesh to the isosurface as well a the overall smoothness of the mesh. The isosurface is stored in a data-structure that enables the fast computation of the distance to the isosurface. Finally, our method can handle very large datasets by merging pieces of isosurfaces.

1 Introduction

Isosurfacing is a popular technique for reconstructing models from volumetric images such as CT-scan images. The main advantage of the Marching Cubes (MC) algorithm over the contour-based reconstruction of Boissonnat [2] is that it achieves an automatic extraction of isosurfaces with subvoxel accuracy, independently of the complexity of the object topology.

As computer-assisted procedures are being introduced in today's surgery, the need for scientific computation of anatomic models is increasing. Today, the computation of mechanical resistance are already performed in orthopedics surgery for bones. However, the MC algorithm generates a triangulation that is not suited for scientific computation due to the very large number of triangles and the poor mesh quality.

1.1 Previous Work

Most decimation methods of isosurfaces have focused on the level of decimation achieved by the algorithm as well as the time of computation needed to achieve this decimation. A first class of methods, such as Shroeder [10], remeshes a subset of the original mesh vertices to achieve a level of decimation. Similarly, Hoppe [6] minimizes a functional in order to select the local mesh transformation.

A second class of decimation algorithms moves vertices in order to optimize some geometric constraint. For instance, Gourdon [4] uses a curvature criterion to remove edges or vertices. Guéziec [5] decimates isosurfaces while preserving the overall enclosed volume of the isosurface.

Finally, a third class of algorithms performs the decimation while extracting the isosurface. Shu [12] applies the MC algorithm at increasing resolutions of the image, and adapt the voxel size as a function of the isosurface curvature. Similarly, Shekhar [11] uses octrees to decimate flat areas of isosurfaces.

1.2 Originality and Contributions

Unlike previous work, our method creates meshes satisfying prescribed geometric and topological criteria such as its distance to the isosurface. Unlike previous approaches, we introduce an additional constraint on the smoothness of the decimated mesh. Smoothness control is achieved by representing the mesh as a deformable model.

Furthermore, we introduce a data-structure for storing isosurfaces that allow an efficient computation of distance to the isosurface. Finally, since isosurfaces can have more than 1 million polygons, we introduce a mechanism for decimating pieces of isosurfaces and subsequently merging decimated mesh together.

2 Decimation with Deformable Models

2.1 Description of the Algorithm

The basic algorithm consists of the following stages : after the extraction of the isosurface, we initialize a deformable model as a dual mesh of the isosurface triangulation. This mesh is then decimated by selecting edges in a random manner. For each edge, we apply a removal edge test followed by edge swap tests on the neighboring edges. At each step, we evaluate if the resulting mesh still satisfies geometric and topological constraints.

2.2 Deformable Models as Simplex Meshes

The originality of this algorithm stems from the use of a deformable model as the representation of isosurface. The deformable models framework has been introduced by Kaas, Witkin and Terzopoulos in their seminal paper on active contours [7]. In this paper, we use simplex meshes as deformable models for the representation of isosurfaces. An extensive description of the simplex mesh deformable models can be found in [3]. Simplex meshes have the property of being simply connected (3 neighbors for each vertex) and have a dual structure of triangulations (see figure 1(a)).

Unlike most deformable surface models, simplex meshes are not parametric models and therefore can handle all types of topology. Furthermore, simplex meshes include the notion of deformable contour which is useful to represent the boundary of isosurfaces.

With the duality between triangulations and simplex meshes, a simplex mesh can be created from any type of isosurface, as can be seen in figure 1(b). One mesh is created for every connected component.

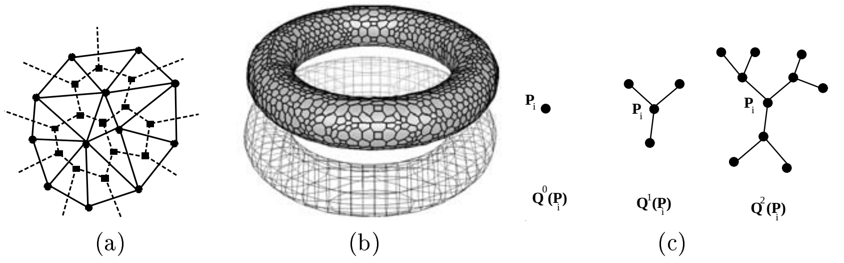


Fig. 1. (a) The duality principle between a triangulation and a simplex mesh; (b) Example of an isosurface and the resulting simplex mesh; (c) Description of a neighborhood $Q^{s_i}(P_i)$

A vertex P_i of a deformable model is moved according to the law of motion :

$$m\frac{d^2P_i}{dt^2} = -\gamma\frac{dP_i}{dt} + \mathbf{F}_{int} + \mathbf{F}_{ext} \tag{1}$$

$\mathbf{F}_{int}$ is the internal force consisting in smoothing the mesh. The smoothness constraint applied on the mesh is the *simplex angle regularity constraint.* The simplex angle describes the local curvature of the mesh and the force $\mathbf{F}_{int}$ is linked with the variation of curvature on the mesh. An important parameter is the rigidity parameter s_i that controls the extent of the neighborhood $Q^{s_i}(P_i)$ on which the smoothing is performed. The neighborhood $Q^{s_i}(P_i)$ is defined recursively as shown in figure 1 (c).

$\mathbf{F}_{ext}$ on the other hand, is proportional to the distance of P_i to the isosurface. If M_i is the closest point of the isosurface from P_i, then the external force is simply : $\mathbf{F}_{ext} = \beta_i\ \overrightarrow{P_iM_i}$. β_i is the stiffness parameter that controls the trade-off between smoothness and closeness to the isosurface. The section 3.2 describes how to compute the closest isosurface point M_i.

3 Representation of Isosurfaces

3.1 Isosurface Data Structure

We represent an isosurface with a data structure that is a trade off between a surface triangulation and a regular grid data structure. Surface meshes such as triangulations, are commonly used representation of surfaces that are well suited for representing complex shapes. However, computing intersections or closest points with triangulations is of high complexity and cannot be done efficiently.

Regular grids on the other hand are efficient data structure for the computation of intersection or closest point because of the direct spatial indexing. However, the representation of objects as sets of voxels or *cuberilles* does not provide subvoxel accuracy and is not accurate enough for most applications in computer assisted surgery.

Isosurfaces are computed with a subvoxel accuracy with the MC algorithm [8]. We actually use the implementation of Nielson [9] that guarantees the topological closure of isosurfaces. In most implementations, the extraction of an isosurface can be decomposed into the computation of the intersection of the isosurface with each voxel. The isosurface is then represented as a set of polygons whose vertices belong to voxel edges. Because those polygons may not be planar, we choose to tessellate the polygons into triangles if the a number of vertices per polygon is greater than 4.

Since the edges of the triangulation of an isosurface are either inside a voxel or on a face of voxel, we store the triangulation in a voxel-based data structure. The data structure is briefly described in figure 2 (a). In each voxel intersecting the isosurface, we store the list of triangles lying inside the voxel. Since every vertex belongs to at most 4 voxels, we build a separate vertex table and we reference vertices inside a isosurface voxel with their index entry into that table.

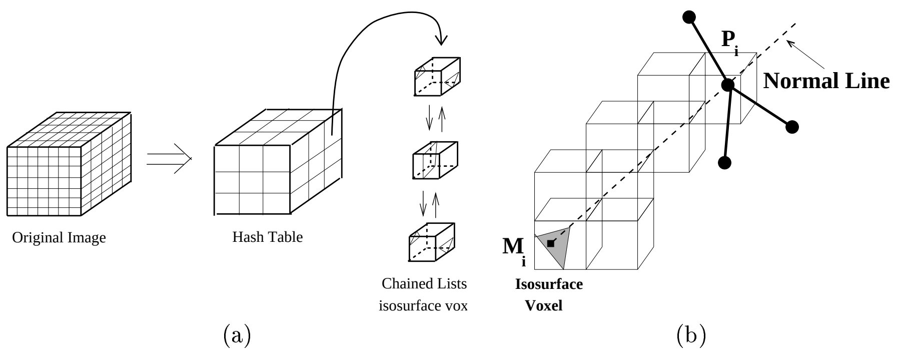


Fig. 2. (a) The data structure for storing isosurfaces; (b) The computation of the intersection point M_i

The isosurface voxels are stored in a hash table with three indexes (one index for each dimension of the image) whose size depends on the amount of memory available. In general, we use a $10 \times 10 \times 10$ hash table corresponding to a reasonable trade off between computing time and memory size.

To find if voxel (r, c, d) intersects the isosurface, we first compute the three indices of the hash table, and then browse the double chained lists of isosurface voxels. In practice, this data structure enables a very fast computation of intersection with the isosurface.

3.2 Computing the isosurface point M_i

Given a vertex P_i, we need to compute its closest point M_i on the isosurface. The computation of the true closest point is time consuming since it requires the computation of the distance to all vertices, edges and triangles in the neighborhood of P_i.

We choose instead to compute the isosurface point M_i as the intersection of the isosurface with the normal line at P_i. The reasons are threefold. First, the mesh is in general very close to the isosurface and therefore the normal at P_i is close to the normal at the isosurface. That means that the intersection point is, in general, very close to the true closest point. Second, because M_i is located on the normal line, the resulting external force $\mathbf{F}_{ext}$ is directed along the normal which ensures smooth deformation of the mesh. Finally, the computation of the intersection point M_i can be done very efficiently.

Figure 2 (b) illustrates the algorithm for the search of M_i. Starting from the voxel containing P_i, we scan the neighboring voxels along the normal line using a tridimensional line-scanning algorithm. For each voxel, we test if the voxel intersects the isosurface, in which case, we check if the normal line intersects the triangles of the isosurface voxel. We only test a few voxels on the normal line since P_i is close to the isosurface.

3.3 Cutting and Merging Isosurfaces

When processing large isosurfaces, we need a method for decimating only subparts of the original isosurface and then sewing the different pieces into a single mesh. Because isosurfaces are stored as set of voxels, they can be easily merged or cut along the three orthogonal planes of the original image. To cut an isosurface along a plane, we split the isosurface voxels into two sets according to their relative position with the plane. We only need to rebuild the vertex table and update the vertex indices inside each isosurface voxels. Similarly, to merge two isosurfaces, we find the common vertices and then rebuild the hash table as well as the vertex table. Examples of cutting and merging of isosurfaces are provided in figure 4 (a).

4 Decimation of Simplex Meshes

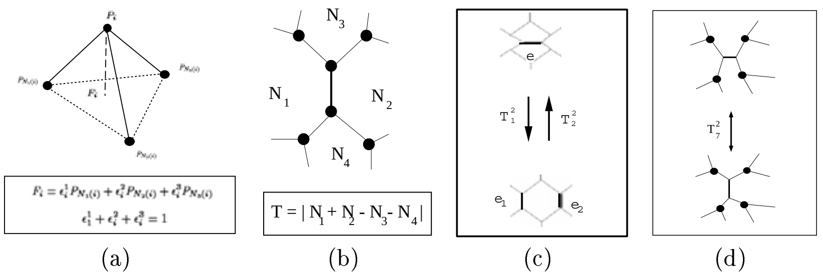
4.1 Geometric and Topological Quality of a Mesh

Scientific computation such as the computation of heat transfer, mechanical deformation or flow requires geometric meshes of high quality in order to guarantee the convergence and stability of algorithms. There are many ways to measure the quality of a mesh : a complete survey is provided in [1]. The mesh resulting from our decimation method satisfies the three criteria :

1. **Isosurface Distance Criterion:** This criterion controls the maximum distance of vertices and face centers to the isosurface.
2. **Geometric Criterion:** In simplex meshes, each vertex has three neighbors. If we project a vertex P_i on the triangle made by its three neighbors, we can compute the barycentric coordinates $(\epsilon_i^1, \epsilon_i^2, \epsilon_i^3)$ of the projection of P_i in the triangle (see figure 3 (a)). We guarantee that each barycentric coordinate is

greater than a given threshold (in general 0.1). This threshold controls also the minimum angle between vertices.

3. **Topological Criterion:** The topological criterion is related with the number of vertices per face. This number should be as close as possible to 6. The topological criterion for an edge is equal to the absolute value of the difference between the number of vertices of the two adjacent faces of an edges and the number of vertices of the two opposing faces (see figure 3(b)). We constrain all mesh edges, to have a topological criterion less than a threshold $\mathcal{T}$ ($\mathcal{T} > 2$).



P_i
$P_{N_1(i)}$
$P_{N_2(i)}$
$P_{N_3(i)}$
F_i
$F_i = \epsilon_i^1 P_{N_1(i)} + \epsilon_i^2 P_{N_2(i)} + \epsilon_i^3 P_{N_3(i)}$
$\epsilon_i^1 + \epsilon_i^2 + \epsilon_i^3 = 1$
N_1
N_2
N_3
N_4
$T = | N_1 + N_2 - N_3 - N_4 |$
e
T_1^2
T_2^2
e_1
e_2
T_7^2
(a)
(b)
(c)
(d)

Fig. 3. (a) The barycentric coordinates $(\epsilon_i^1, \epsilon_i^2, \epsilon_i^3)$ of the projection F_i; (b) The topological criterion of an edge; (c) The edge removal and edge swap (d) operators;

In addition to those criteria, two parameters influence the decimation of the mesh : the rigidity parameter s_i and the external force parameter β_i.

4.2 Decimation Scheme

After the creation of the simplex mesh from the isosurface, we proceed by first applying to all edges, the edge swap test, because the initial mesh created from the MC algorithm is, in general, of poor topological quality. The edge swap test tries to swap edges adjacent to large or small faces while satisfying both geometric and distance criteria.

In a second stage, all edges are labeled as *unprocessed*. We then iteratively pick at random, edges labeled as *unprocessed* and apply the edge removal test. If the test fails, the edge is labeled as *processed*. Otherwise, the edge is removed, thus creating a face with more vertices. We then apply a set edge swap tests on edges belonging to the newly created face. The purpose is to decrease the topological criterion of those edges and therefore increase locally the topological quality of the mesh. Once all neighboring edges have been submitted to the edge swap test, the edges that have failed the edge swap test are labeled as *processed* while the others are labeled as *unprocessed*. We proceed until all edges are labeled as *processed*. n.

4.3 Edge Swap Test

The edge swap test is performed with the help of the operator T_7^2 as described in figure 3 (d). After swapping the edge, the test consists in locally deforming the mesh according to the law of motion described in equation 1. The number of vertices depends on the neighborhood size s_i. The external force parameter β_i as well as s_i control the resulting local deformation of the mesh. Applying few iterations (5 iterations in general), we check if the moved vertices verify the geometric and distance criterion. If not, all vertices are moved back to their position before deformations, we swap back the edge and label it as *processed*. If all vertices satisfy both criteria, we label all adjacent vertices as *unprocessed*.

4.4 Edge Removal Test

The edge removal test is similar to the edge swap test. We remove the edge with the operator T_1^2 described in figure 3 (c). This operator is Eulerian since it does not change the genus or Euler number of the mesh. The mesh is then locally deformed as for the edge swap test, and consequently we test if the moved vertices verify the geometric and distance criterion. If the test is negative, we apply the operator T_2^2, reverse of operator T_1^2 , that recreates the previously removed edge.

4.5 Decimating Large Isosurfaces

One important drawback of the MC algorithm is that it generates a large number of triangles even in presence of flat surfaces. As large volumetric images are becoming more and more frequent, it is important that our algorithm can handle large datasets. However, since we are storing in memory the isosurface as well as the mesh, our algorithm cannot accept large datasets (with more than 100 000 vertices) on a 64 megabytes memory machine.

Given an isosurface, our program computes the minimum number of isosurface pieces, based on the memory available on the machine. In addition to the number of planes of cut, we need to know along which direction (there are three possible directions) we should slice the isosurface. We select the the direction of cut that minimizes the number of isocontours resulting from the intersection between a plane and the isosurface. The isosurface is then sliced along the preferred direction into a set of isosurfaces of smaller size.

Each connected component of isosurface is decimated separately with the same criteria. An automatic merging algorithm sews neighboring meshes together. Two meshes are connected with a topological operator applied on two contours. Once two corresponding isocontours are merged, we smooth the mesh around the junction and increase the topological quality of the meshes by swapping selected edges. Figure 4(a) shows an isosurface representing a vessel after slicing along 3 isocontours. There are 4 connected components that have been decimated and merged automatically.

5 Results

The first example demonstrates the importance of controlling the smoothness of the mesh during decimation. Figure 4(b) shows an isosurface of a cube having large step effects due to the MC algorithm. Figure 4(c) shows a decimated mesh with a distance criterion equal to 40% of the isotropic voxel size. The resulting reduction factor is 90%. We have used a high rigidity parameter $s_i = 3$ and a low stiffness parameter $\beta_i = 0.15$. The recovered model has smoothly approximated the cube shape.

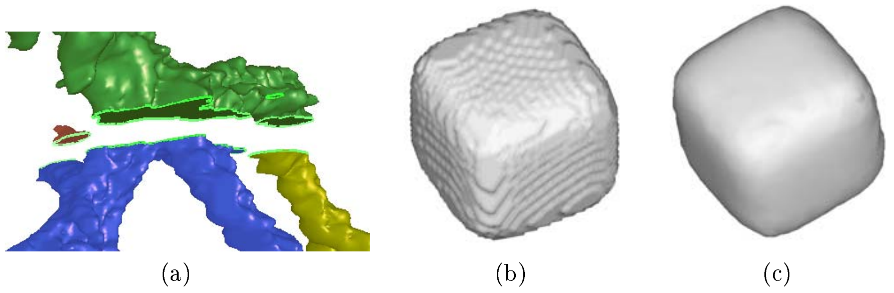


Fig. 4. (a) Four simplex meshes are merged by connecting corresponding contours; (b) The isosurface of a cube; (c) The smoothed decimated mesh.

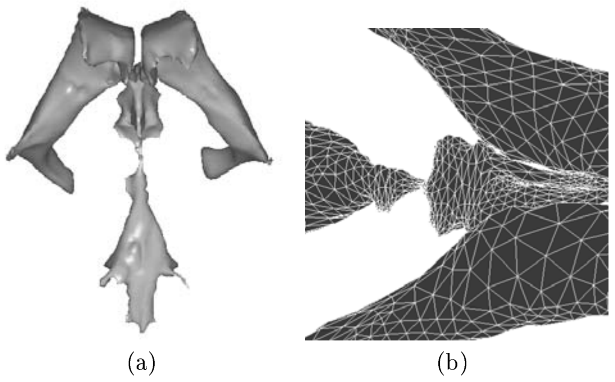


Fig. 5. (a) The decimated mesh of a brain ventricle; (b) Details of the decimation

The second example shows the decimation of an isosurface with parts of high curvature. The brain ventricle of figure 5 (a) has been decimated from an isosurface with 40 000 triangles. The isosurface had a lot of step artifacts since it was extracted from a binary image. The distance threshold was equal to 40% of the maximum voxel size. The mesh reduction was only of 65% because of the highly curved areas. We used a low rigidity value $s_i = 1$ and a large stiffness parameter $\beta_i = 0.5$. The topological threshold was $\mathcal{T} = 3$. The decimated simplex mesh has been triangulated and displayed in figure 5(b). Because of the duality relationship between simplex meshes and triangulations, the associated triangulation has the same high geometric quality as the original mesh, and satisfies the distance criterion. The computation time was 4 hours on a DEC Alphastation 200/233 without slicing the isosurface.

	Number of triangles	Decimation rate
Distance = 20%	23174	81%
Distance = 30%	10114	92%

Table 1. Decimation of the portal vein with two different distance criteria.

The third example shows the decimation a portal vein (see figure 6). The original isosurface had 126 738 triangles. We obtain a decimation rate of 81 % with a distance criterion of 20% of the maximum voxel size. Since voxels are anisotropic ($0.33 \times 0.33 \times 1.0$), it actually represents 60 % of the pixel size. The isosurface was cut into two pieces creating 9 different connected components. Table 1 summarizes the results obtained with this isosurface.

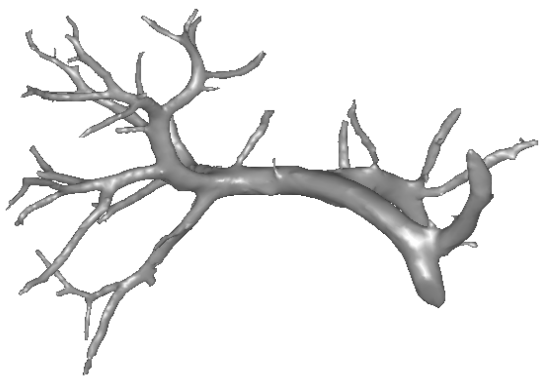


Fig. 6. The decimated isosurface of a portal vein with 23174 triangles with a decimation rate of 81%

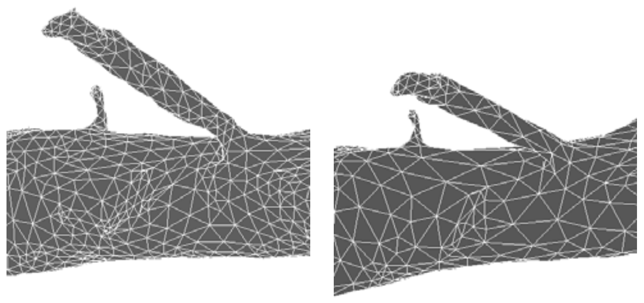
Figure 7 shows the layout of triangles of the decimated mesh. The vertices are concentrated at parts of high curvature. This is because in those areas, a vertex cannot be removed without noticeably increasing the local mesh distance to the isosurface.

6 Conclusion and Perspectives

We are currently using the decimation method proposed in this paper for visualization and computation purposes. In particular, it is used for generating patient-based models suitable for a surgical simulator. In the future, we would like to improve the speed of computation of our algorithm and to increase the smoothness control over the mesh.

Acknowledgment

We are grateful to J. Marescaux, J.M. Clément and V. Tassetti from *IRCAD* (Institut de Recherche sur les Cancers de l'Appareil Digestif, Strasbourg, France) for their help in extracting the portal vein model. We thank Dr Kikinis from the Brigham and Women's Hospital, (Boston, USA), for providing the model of the ventricle.



(a) (b)

Fig. 7. (a) Details of the decimated triangulation with a distance criterion of 20% of the voxel size; (b) same as (a) with a distance of 30%.

References

1. M. Bern and D. Eppstein. Mesh generation and optimal triangulation. In D. Du and F. Hwang, editors, *Computing in Euclidean Geometry*, volume 1, pages 23–90. World Scientific Publishing Co, 1991.
2. J. Boissonnat and B. Geiger. Three dimensional reconstruction of complex shapes based on the delaunay triangulation. In R. Acharya and D. Goldgof, editors, *SPIE Conference on Biomedical Image Processing and Biomedical Visualization*, volume 1905, San Jose, CA, Feb. 1993.
3. H. Delingette. Simplex meshes: a general representation for 3d shape reconstruction. Technical Report 2214, INRIA, Mar. 1994.
4. A. Gourdon. Simplification of irregular surfaces meshes in 3d medical images. In *First International Conference on Computer Vision, Virtual Reality and Robotics in Medicine*, pages 413–419, Nice, France, Apr. 1995.
5. A. Gueziec and D. Dean. The wrapper : A surface optimization algorithm that preserves highly curved areas. In *Visualisation In Biomedical Computing (VBC'94)*, volume 2359, pages 631–641, Rochester, USA, Sept. 1994. SPIE.
6. H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Mesh optimization. In *Computer Graphics (SIGGRAPH'93)*, pages 19–25, Anaheim, July 1993.
7. M. Kass, A. Witkin, and D. Terzopoulos. Snakes: Active contour models. *International Journal of Computer Vision*, 1:321–331, 1988.
8. W. Lorensen and H. Cline. Marching cubes: a high resolution 3d surface construction algorithm. *ACM Computer Graphics (SIGGRAPH'87)*, 21:163–169, 1987.
9. G. Nielson and B. Hamann. The asymptotic decider: Resolving the ambiguityin marching cubes. *Visualisation'91*, pages 83–91, Oct. 1991.
10. W. J. Schroeder, J. Zarge, and W. Lorensen. Decimation of triangles meshes. In *Computer Graphics (SIGGRAPH'92)*, volume 26. ACM, Aug. 1992.
11. R. Shekhar, E. Fayyad, R. Yagel, and J. Cornhill. Octree-based decimation of marching cubes surfaces. In *Visualisation'96*, San Francisco, Sept. 1996.
12. R. Shu, Z. Chen, and M. Kankanhalli. Adaptive marching cubes. *The Visual Computer*, 11:202–217, 1995.