Efficient Representation and Extraction of 2-Manifold Isosurfaces Using kd-Trees

Alexander Greß
gress@cs.uni-bonn.de

Reinhard Klein
rk@cs.uni-bonn.de

University of Bonn
Institute of Computer Science II
Römerstraße 164, 53117 Bonn, Germany

Abstract

In this paper, we propose the utilization of a kd-tree based hierarchy as an implicit object representation. Compared to an octree, the kd-tree based hierarchy is superior in terms of adaptation to the object surface. In consequence, we obtain considerably more compact implicit representations especially in case of thin object structures. We describe a new isosurface extraction algorithm for this kind of implicit representation. In contrast to related algorithms for octrees, it generates 2-manifold meshes even for kd-trees with cells containing multiple surface components. The algorithm retains all the good properties of the Dual Contouring approach [10] like feature preservation, computational efficiency, etc. In addition, we present a simplification framework for the surfaces represented by the kd-tree based on quadric error metrics. We adapt this framework to quantify the influence of topological changes, thereby allowing controlled topological simplification of the object. The advantages of the new algorithm are demonstrated by several examples.

Keywords: *Isosurfaces, Mesh Generation, Hierarchical Structures, Topological Simplification*

1. Introduction

During recent years implicit modeling has gained a lot of attention in computer graphics, especially since the introduction of octree based hierarchies that allow an efficient adaptive implicit representation of arbitrary solid objects [6]. CSG operations are particularly efficient and simple to implement for such a representation, and a number of new modeling techniques were build on the basis of implicit representations [16]. For rendering as well as further postprocessing the implicit representation is converted into a polygonal mesh by an isosurface extraction algorithm. Also the reconstruction of surfaces from range images or unstructured point clouds is often done by estimating a signed distance function and then applying an isosurface extraction algorithm to such a distance field [4, 5]. This kind of implicit representation has been used for various tasks of geometry processing as well. For example, the data of an intermediate stage in the industrial CAD process pose a number of problems (e.g. cracks in tessellations, topological inconsistencies, degenerated triangles, topological noise, etc.) to any further use of such models in downstream applications. In order to overcome these problems, the CAD data is converted into an implicit representation and then converted back into a surface mesh with desired properties [3]. Such geometry remeshing allows to close gaps, to remove topological noise [23] and also to generate a multi-resolution representation of the objects [6, 16, 22]. Last but not least, implicit representations of surfaces are of great importance in medicine, biology, and geology, where a huge amount of volumetric data (CT, NMR, Ultrasound, etc.) is acquired and has to be processed.

1.1. Related work

The standard approach for the extraction of surfaces from an implicit representation is the well-known Marching Cubes algorithm by Lorensen and Cline [14]. It operates on a uniform volumetric grid that specifies for each grid point whether it lies inside or outside the represented object. The output is a polygonal mesh of a surface that separates the grid points inside from those outside the object. This surface is called contour surface or *isosurface*. The Marching Cubes algorithm visits all cells of the grid and computes the isosurface for each cell independently. If and how a cell is intersected by the isosurface is determined dependent on the inside/outside states of the eight cell corners, using a look-up table. If ambiguous cell configurations are handled properly the algorithm generates a closed 2-manifold

Proceedings of the 11th Pacific Conference on Computer Graphics and Applications (PG'03)
0-7695-2028-6/03 $17.00 © 2003 IEEE

IEEE COMPUTER SOCIETY

mesh [21]. The vertices of this mesh correspond to the intersection points of the isosurface with cell edges. These intersections are typically estimated by linear interpolation of the implicit values given at the grid points that are the end points of those edges. Instead of directly using sampled values from acquired volumetric data, which may lead to jagged or terraced surfaces due to high spatial frequencies in the data, Gibson proposes the use of a distance map, where at each grid point the shortest distance to the object surface is given [9]. But, as noted in [3], this still leads to a bad estimation for the reconstructed surface in the vicinity of sharp features.

The Extended Marching Cubes approach [3] resolves this problem. By the use of a directed distance field, where at each grid point the distances to the object surface in direction of the three axes are given, the exact intersection points of the isosurface with the uniform grid can be determined. Furthermore, it is assumed that surface normals associated with these points are known. By checking the variation of these normals the presence of a sharp feature inside a cell can be detected. In this case an additional vertex inside the cell is created and incorporated in the generated mesh such that the detected feature will be preserved.

An alternative surface extraction method is the SurfaceNets algorithm [8]. It produces meshes that are dual to the ones produced by Marching Cubes in the sense that SurfaceNets meshes consist of one quadrilateral (or equivalently two triangles) for each grid edge that crosses the surface, whereas Marching Cubes meshes contain one vertex for each such edge. The SurfaceNets algorithm simply generates a vertex for each cell that intersects the surface and connects it with the vertices generated in neighboring cells. Contrary to Marching Cubes, the vertex positions are not restricted to the grid edges, and thus the polygons of the resulting meshes tend to have better aspect ratios.

The surface extraction approaches reviewed above have the common property that they use uniform grids as input. This has several disadvantages. The storage space required for cells that are not crossed by the surface as well as the computation time spent for empty cells is considerable. Moreover, the number of polygons generated is usually very high since it is independent of local surface properties like surface curvature and depends only on the number of cells intersected by the isosurface.

To address these problems several adaptive isosurface extraction methods based on octrees or closely related hierarchical structures have been proposed.

Marching Cubes based adaptive approaches [2, 15, 19, 18] have to deal with the fact that applying Marching Cubes to cells of different sizes lead to cracks in the generated surface since the Marching Cubes algorithm handles neighboring cells independently. In these approaches the cracks are patched either by closing them with polygons of the same shape as those cracks or by forcing the polyline on the common faces of neighboring cells to fall together.

The Adaptive Skeleton Climbing approach [17] avoids the crack patching problem by first creating those polylines and then combining them to form the polygons of the resulting mesh.

Contrary to these approaches, the SurfaceNets algorithm can be extended much more easily to work on an octree hierarchy [16]. The Dual Contouring approach by Ju et al. [10] combines this Enhanced SurfaceNets algorithm with ideas from the Extended Marching Cubes approach to preserve sharp features in the reconstructed meshes and further removes the restriction of prior approaches that the octree depth of neighboring cells may differ only by one.

Unfortunately, the SurfaceNets algorithm as well as the adaptive approaches based on it [6, 16, 10] have the restriction that they produce 2-manifold surfaces only if not more than one surface component is contained in one cell. But especially in case of thin object structures it is likely that more than one surface component is contained in one cell. In this case vertices with non-manifold neighborhood and most likely also non-manifold edges will be generated. Hence, topologically incorrect reconstructions will be produced.

In contrast to the recent approaches for adaptive isosurface extraction that use octree based hierarchies, in this paper we will introduce a more general data structure that is based on a kd-tree hierarchy. A kd-tree is a multi-dimensional binary tree for spatial subdivision with arbitrary axis-aligned splitting planes that is widely used in computational geometry.

In context of surface extraction kd-trees have been used in an approach by Livnat et al. [13] to accelerate surface extraction on a uniform grid. In this context kd-trees already have been proven to be much more efficient than similar approaches based on octrees [20]. But it still remains to show that a kd-tree also has several advantages over an octree when used as basis for an adaptive implicit representation.

1.2. Contributions

In addition to several improvements over previous algorithms the main novel contributions of this paper are

- The introduction of a kd-tree based hierarchy as an implicit object representation together with an appropriate surface extraction algorithm. Compared to an octree, this data structure allows for better adaptation to the object surface and thus for a more compact object representation especially in case of fine topological detail.
- The algorithm proposed in this paper resolves the

IEEE COMPUTER SOCIETY

topology correctly within cells containing more than one surface component and even allows for simplification of the geometry in those cells. Furthermore, during simplification the topology is modified in a reasonable and controlled way.

The algorithm guarantees that 2-manifolds (or manifolds with boundary in case of non-solid objects) are generated. In the special case of a uniform space subdivision 2-manifold quad meshes are generated, which is of interest e.g. in finite element mesh generation.

The rest of the paper is organized as follows. First, we present a variant of our isosurface extraction algorithm for uniform grids in section 2. Then, we introduce our new kd-tree based representation and extend the extraction algorithm to this representation in section 3. Section 4 describes how the kd-tree can be reduced and thus the object can be simplified. In section 5 we show how our approach can be used to represent arbitrary polygonal meshes by an adaptive grid and how CSG operations can be applied to those adaptive grids. Finally, we summarize our results in section 6.

2. Isosurface extraction on uniform grids

Before extending the method to adaptive volumetric representations, we first describe how the new surface extraction algorithm operates on a uniform representation.

We assume that a uniform grid is given, i.e. a set of $m \times n \times o$ adjacent, equally sized cubes that we call *grid cell*s. We refer to the six sides of each cell as *grid half-face*s, as for two adjacent cells there are two coincident half-faces that together build a *grid face*. Each grid face consists of four successive *grid edges*.

We further assume that all edges of the grid intersected by the isosurface, which we call *active grid edges*, are tagged with information about the exact intersection point and a surface normal in this point. Furthermore, inside/outside information is necessary that might be given in the form of in/out signs at the grid points.

The new algorithm is similar to SurfaceNets [8] and the uniform variant of the Dual Contouring approach [10] in the sense that it is a dual algorithm and that on regular grids exactly one quadrilateral is generated per active grid edge, thus producing a quadrilateral mesh. However, more care is taken in reconstructing the face connectivity to assure that the resulting mesh is 2-manifold and the approximation of the geometry is improved if multiple components are contained in one cell. Figure 1 shows an example. On the left hand the output of the uniform Dual Contouring algorithm is shown, on the right hand a 2-manifold isosurface was extracted using our new algorithm. While both algorithms produce quad meshes the main difference is that no complex vertices are generated by our algorithm. In many postprocessing algorithms like finite element mesh generation this property is essential.

A more complex example of a 2-manifold quad mesh generated by our algorithm is given in figure 2.

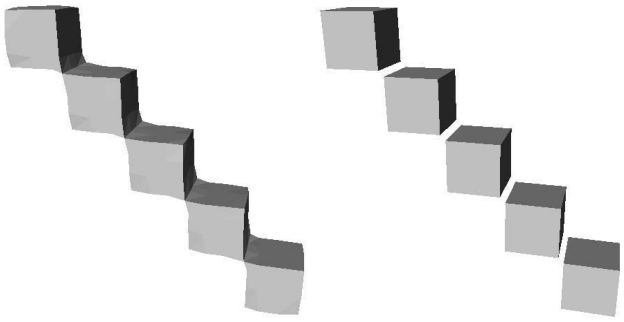


Figure 1. The mesh on the left was generated using the Dual Contouring algorithm on a coarse uniform grid. It contains several complex edges, and therefore also the features of the original object could not be fully preserved. The mesh on the right was generated using our new isosurface extraction algorithm on the same coarse grid. The topology as well as the features of the original object have been preserved.

2.1. Connectivity reconstruction

Let us first make a simple observation: If a grid face is intersected by the isosurface the chain of successive grid edges of that face exhibits always an even number of sign changes (two or four). Therefore, during isosurface extraction two or four quadrilaterals have to be generated for each grid face that is intersected by the isosurface. If four sign changes are present in a grid face, the SurfaceNets algorithm would generate one common edge that is incident to the four quadrilaterals corresponding to the intersection points on the grid face and connects the two vertices corresponding to the two neighboring grid cells, see figure 3a. This results in non-manifold surfaces. In order to overcome this problem instead of using always one edge per grid face two edges are used if four sign changes are present and two vertices are generated in the neighboring grid cells, respectively. In the algorithm this is realized by generating exactly one half-edge for each intersection point on the grid face, see figure 3b. Since the number of generated half-edges is two or four, respectively, we can always combine two half-edges corresponding to consecutive intersection points on the grid face: if only two sign changes are present, the

IEEE COMPUTER SOCIETY



Figure 2. 2-manifold quad meshes generated by our algorithm from grids of different resolutions. While the topology of the original object has been preserved well using a high resolution grid, as shown on the right, some fine topological structures smaller than the grid spacing have been lost in the lower resolution grid, shown on the left.

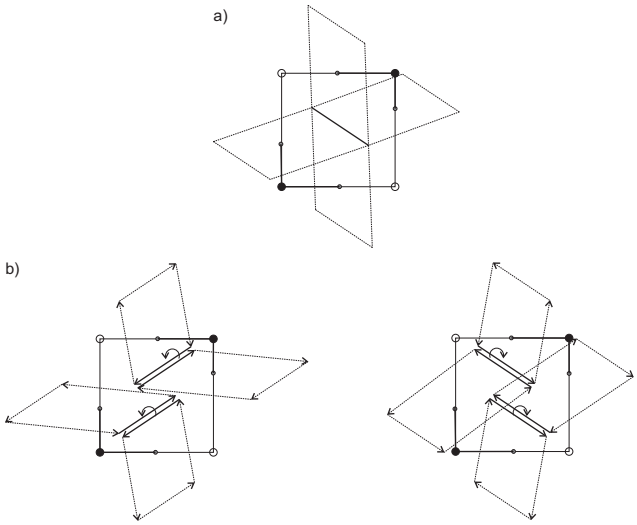


Figure 3. Handling of grid faces with four sign changes: a) The SurfaceNets algorithm produces a non-manifold solution in this case. b) In contrast, the disambiguation strategies used with our algorithm avoid the generation of such complex edges.

two half-edges build one common edge, if four half-edges are present two edges will be constructed. In the latter case there are two possible solutions to combine consecutive half-edges, as shown in figure 3b. Which one is chosen does not matter since in both cases a closed mesh will be generated. The simplest strategy obviously is to choose always the same one of the two solutions. For input data with thin structures the solution shown on the right side of figure 3b is usually a good choice.

Incorporating the idea of using half-edges leads to a crucial difference to the SurfaceNets algorithm. The sequence in which the grid edges are visited in order to generate the quadrilaterals is changed. Instead of iterating through the grid edges and building the quadrilaterals, we iterate through the grid faces and generate for each active grid edge the associated half-edge. In the same iteration the half-edges belonging together are set as described above. In this step, in addition to the opposite pointers also the next pointers of the generated half-edges have to be set, i.e. four half-edges have to be connected to build a quadrilateral. Obviously each active grid edge is contained in exactly four grid faces. Therefore, the grid edge is visited four times during the iteration through the grid faces, and four half-edges are created, which build a quadrilateral, see figure 4. Hence a quadrilateral can be constructed by simply collecting the half-edges generated for their associated grid edge during the iteration through the grid faces and connecting them in the order in which the grid faces are arranged around the grid edge. The orientation of the half-edges depends on the signs given at the end points of that grid edge.

2.2. Computing vertex locations

At this point of the extraction algorithm we have generated the whole connectivity information of the resulting mesh, but so far we have not defined the corresponding vertices and their locations. This is done in a second step as follows:

We iterate through the list of all half-edges generated above. For each half-edge that has not been assigned to a vertex so far, we generate a new vertex and assign it to the half-edge itself. Then we iterate around this new vertex using the opposite and next pointers of our half-edge data structure and assign this vertex to all incident half-edges. The optimal location for the new vertex is determined using a quadratic error function as described below. Note that this kind of vertex generation assures that the neighborhood of each vertex is topologically equivalent to a disk, and thus a closed 2-manifold is constructed.

2.2.1. Primal quadrics

Each quadrilateral of the isosurface mesh corresponds to an intersection between the isosurface and a grid edge. For

IEEE COMPUTER SOCIETY

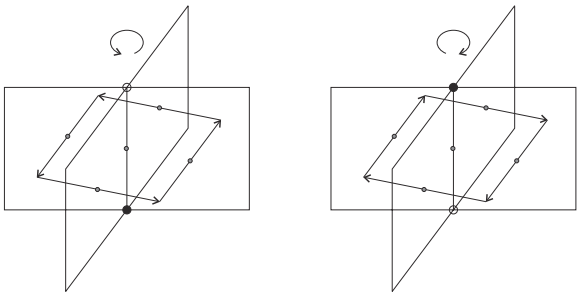


Figure 4. There are four grid faces adjacent to an active grid edge. For each grid face a half-edge is created, and the four half-edges are connected to build a quadrilateral. The orientation of the half-edges depends on the signs given at the end points of the intersected grid edge.

these intersections the locations and normals are given. We therefore try to position the vertices of the quadrilaterals in such a way, that their incident quadrilaterals lie approximately in the planes determined by their corresponding intersection points. Following the approach of Ju et al. [10], we position each vertex in such a way that the quadratic function

$$E(x) = \frac{1}{k}\sum_{i=1}^{k}(n_i \cdot (x - p_i))^2 \quad (1)$$

where p_i, n_i correspond to the points and normals associated with the polygons adjacent to the vertex, is minimized.

Following Lindstrom [11] and writing

$$A = \sum_{i=1}^{k} n_i n_i^T,\ b = \sum_{i=1}^{k} n_i \cdot (n_i^T p_i),\ c = \sum_{i=1}^{k} (n_i^T p_i)^2 \quad (2)$$

where A is a symmetric 3×3 matrix, b is a 3-vector, and c is a scalar, we can rewrite eq. 1 as follows:

$$E(x) = \frac{1}{k}(x^T A x - 2x^T b + c) \quad (3)$$

Motivated by [7] we call $Q = (A, b, c)$ *primal quadric*.

A minimum of $E(x)$ can be found by solving $A^T x = b$. This can be done efficiently by computing the singular value decomposition of A, see Ju et al. [10]. In the case that the resulting vertex position is not contained in the corresponding grid cell, which would contradict the assumption that no more intersection points are located on the grid edges, the resulting vertex position can be projected back to the boundary of the grid cell as suggested by Ju et al. [10]. An alternative suggested by Lindstrom et al. [12] is to search for the minimum under the constraint of lying on the boundary of the grid cell.

2.3. Extensions

In this subsection we present some optional improvements for the uniform extraction algorithm.

2.3.1. Multiple features inside a cell

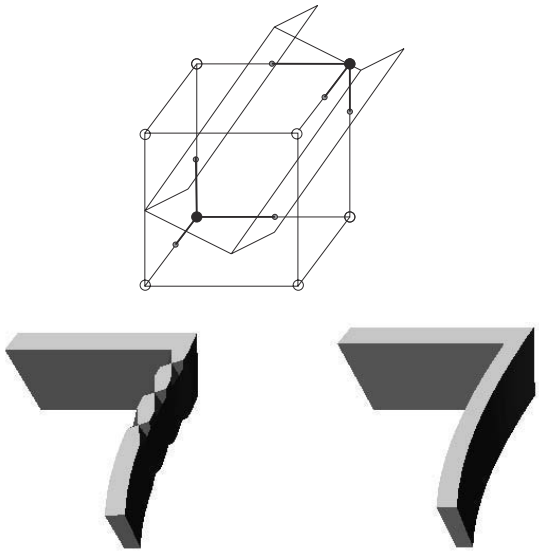


Figure 5. While the mesh on the left hand contains only one vertex per surface component inside a cell, the mesh on the right hand contains some additional vertices and thus reproduces even multiple feature lines inside a cell that belong to the same surface component.

Since the extraction algorithm as described above generates exactly one vertex per isosurface component inside a cell, only one feature on such component can be reconstructed. In some cases this leads to disturbing artifacts, as shown in figure 5. In such cases we split the vertex and introduce a new edge and two triangles at those edge. Note that this results in meshes that contain triangles as well as quadrilaterals, see figure 5.

To detect if it is worthwhile to apply a vertex split we check if $E(x) \geq \varepsilon$ for the resulting vertex position x and a threshold ε. After detecting that a vertex split is necessary the main task is to determine, at which edges the polygons in the one-ring of the vertex should be split.

A simple solution to check if and how a vertex split should be performed is to start with an arbitrary polygon f_0 in the one-ring and then to iterate clockwise through its adjacent faces accumulating the corresponding quadrics until either all adjacent faces have been visited without exceeding the given threshold ε or a face f_{i+1} is added causing

IEEE COMPUTER SOCIETY

$E(x) \geq \varepsilon$. In the first case no vertex split is necessary. Otherwise, we subtract the quadric corresponding to polygon f_{i+1} and store the edge between f_i and f_{i+1} as first split edge. Then we iterate backwards through the adjacent faces of v, again starting at face f_0 and accumulating the corresponding quadrics further until the threshold is exceeded again. This way we have found one section of the one-ring that can be represented by one vertex without exceeding the approximation threshold ε. The location of this vertex is determined as minimum of the accumulated quadric. The remaining section of the one-ring is processed in the same way. Figure 6 shows an example of such a vertex split.

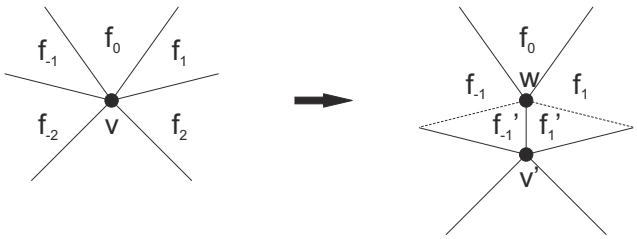


Figure 6. Example of a vertex split: Here, neither the quadric defined by faces f_0, f_1, f_2 nor the quadric defined by faces f_0, f_1, f_{-1}, f_{-2} yields a good solution. Therefore vertex v is split such that the new vertex w is positioned at the minimizer of the quadric defined by faces f_0, f_1, f_{-1}, i.e. at the intersection point of the corresponding planes, and the process is repeated for the neighborhood of vertex v'.

2.3.2. Representation of non-solid objects

If an arbitrary input should be converted into its implicit representation, we have to deal with the fact that it might not describe a solid object. A simple solution to this problem is to provide the inside/outside information required by the extraction algorithm not by signs at the grid points, but by an orientation flag at each grid edge that is crossed by the surface specifying which side of the intersection belongs to the interior of the represented object. As this information is uniquely defined even for objects that are not solid, we can in fact construct such a representation for arbitrary objects even if they are not solid, as we will show in section 5.1. If we adapt the extraction algorithm described above to this case, not every half-edge gets an opposite half-edge anymore, but it is still assured that the generated vertices have a neighborhood that is topologically equivalent to a disk or half-disk, and thus a manifold with boundary is constructed.

3. Adaptive isosurface extraction on kd-tree grids

In this section we introduce our new kd-tree based representation, explain the operations needed for the construction and reduction of this representation and show how isosurfaces can be extracted from such a representation.

3.1. Introduction

In the following subsection we briefly review the recently used adaptive implicit representations, since they already deal with the basic problems of adaptive surface representation and adaptive isosurface extraction that we have to solve in our approach as well.

3.1.1. Recent adaptive representations

The Enhanced SurfaceNets algorithm [16] as well as the Dual Contouring approach [10] both use signed octrees as an adaptive representation, i.e. they partition the given space using an octree and store signs at all cell corners of this octree thereby separating spatial regions inside the represented object from those outside. Furthermore, the octree holds additional information required for the positioning of the vertices that will be generated during surface extraction. In case of the octree structure used in the Enhanced SurfaceNets approach, called Adaptively Sampled Distance Field [6], this is the shortest distance from the cell corners to the represented object. Instead of this, the Dual Contouring approach assumes that the edges of leaf cells are tagged with exact intersection points and normals (Hermite data), and associates a quadratic error function to each octree cell based on this data [10].

For arbitrary sign configurations of an octree, the surfaces produced by the Enhanced SurfaceNets algorithm may contain cracks at those places where neighboring octree leaf cells of different levels in the hierarchy meet. This problem is caused by the fact that multiple cell edges may be contained in one larger cell edge of a common neighboring cell. However, in their Dual Contouring approach Ju et al. [10] show that such cracks can be avoided by using a different polygon generation rule which says that only for those edges of octree leaf cells a polygon should be generated that do not properly contain a smaller edge of a neighboring cell.

3.1.2. The new kd-tree based adaptive representation

In contrast to the previously used octree based representations described above, in this section we introduce a more general data structure for representation of adaptive axis-aligned grids.

IEEE COMPUTER SOCIETY

Such an adaptive grid is given by a set of grid cells organized in a kd-tree hierarchy (*kd-tree grid*). Each *grid cell* corresponds to a box with six axis-aligned sides that we call *grid half-faces*. A grid cell may be subdivided into two child cells according to an arbitrary axis-aligned splitting plane. This introduces two new grid half-faces lying on the splitting plane that together constitute the *splitting face* of the subdivided cell. The grid cells that are not further subdivided (i.e. the leafs in the kd-tree hierarchy) and their half-faces are called *minimal* grid cells and *minimal* grid half-faces, respectively. A (minimal) *grid face* is given by the intersection of two coincident or overlapping (minimal) half-faces. The intersection of adjacent minimal grid faces defines a minimal *grid edge*. The grid edges that cross the isosurface are called *active* edges.

The idea of our uniform isosurface extraction algorithm to iterate over all grid faces, see section 2, is also the key to handle the adaptive grid. While in the uniform case a grid face simply corresponds to a pair of coincident grid half-faces, grid faces may overlap in the adaptive case, thus similar to the afore-mentioned crack prevention rule by Ju et al. we have to enumerate the *minimal* faces of the adaptive grid in order to extract the isosurface.

3.1.3. Comparison to previous approaches

If we subdivide grid cells always by a plane through their center points, successively in x-, y-, and z-direction, our hierarchy corresponds to the octree hierarchy used in previous approaches. However, such an octree hierarchy has several limitation we wish to overcome with our more general hierarchy. We will demonstrate this in the following.

Consider an isosurface extraction algorithm operating on a uniform or adaptive grid and having no information given about the topology of the isosurface other than where it intersects the edges of the grid. Then a basic issue is that we have to ensure that no surface component intersects a cell face while not intersecting its edges as otherwise the reconstructed object may suffer from uncontrolled topology changes or the loss of thin object structures, see figure 7a.

To overcome this problem in an octree based hierarchy, in principle the octree cells containing such edges could be subdivided until this case does not occur anymore. Unfortunately, since octree cells are always subdivided by a plane through its center point this might require a large number of subdivision steps leading to a large depth of the octree, especially if the isosurface component has a small diameter compared to the corresponding grid face, as in the example in figure 2.

However, with our kd-tree based hierarchy the position of the axis-aligned splitting plane can be chosen arbitrarily, thus allowing to resolve such a problematic case with a single subdivision step during the construction of the adaptive grid.

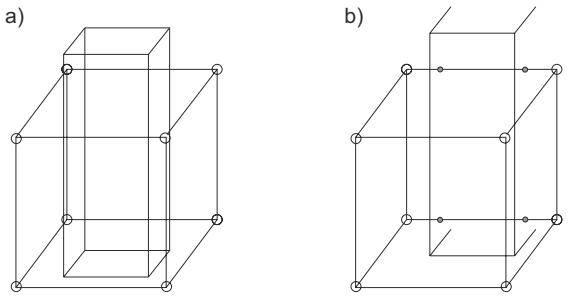


Figure 7. The object on the left intersects a face while not intersecting its edges. The object on the right intersects a grid edge twice.

In general, the kd-tree based hierarchy leads to a more compact representation of the underlying geometry. Furthermore, using a kd-tree based hierarchy more features can be represented than by an octree of the same depth. This is nicely illustrated in figure 8.



Figure 8. This example shows a street model (just an incomplete part of a larger city model), converted to an octree representation (on the top) and to a kd-tree representation (at the bottom), both restricted to a maximum depth of 18. While in the octree representation just the basic shape of the buildings at the street have been preserved, the kd-tree representation of the same depth was able to preserve even the fine detail of the input model like the doors and windows, the street lamps etc.

The Enhanced SurfaceNets algorithm as well as the Dual Contouring approach reconstruct the topology of the isosurface based only on the inside/outside states (*signs*) of the given grid points. Therefore, they could not detect if a grid

IEEE COMPUTER SOCIETY

edge is intersected by the isosurface multiple times, which is likely to happen in case of objects with thin structures. If a grid edge is intersected an even number of times these algorithms would not even detect an intersection at all, as there is no sign change at the corresponding grid points, see figure 7b. To overcome this problem and to be able to reconstruct the topology of an isosurface that crosses a grid edge multiple times, we store all intersections of the isosurface together with a grid edge in our adaptive data structure. As in the uniform case, the intersection information in our algorithm consists of the exact intersection point as well as the surface normal at this point. Note, that the intersection point on an axis-aligned grid edge can simply be represented by a single scalar value. In our data structure the list of intersections associated with a grid edge is sorted by these values. We also allow such a list to hold two intersections with identical intersection points but opposite surface normals. This way we are even able to represent and reconstruct thin-shelled surfaces. Without storing multiple intersections per grid edge topologically correct representation of objects with thin structures would require highly subdivided grids (at least with an octree hierarchy) while thin-shelled surfaces could not be represented at all, see figure 9.

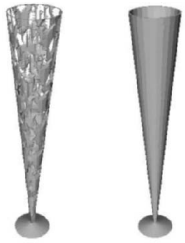


Figure 9. Reconstructing the topology of the isosurface based only on signs given at the grid points, as done in the Dual Contouring approach, leads to artifacts in case of thin object structures, as shown on the left hand. In contrast, our extraction algorithm handles the cases where a grid edge is intersected by the isosurface twice correctly and is able to reconstruct the object without artifacts, as shown on the right.

3.2. The data structure in detail

An adaptive grid according to the definition in section 3.1.2 can be constructed by recursively subdividing an initial root cell. The construction of the hierarchy obviously depends on the application, respectively in which form the original input data is given. Theoretically, the only information needed is if a cell should be subdivided further and in this case the position and axis of the corresponding splitting plane (*splitting axis and value*).

As mentioned above, in addition to the kd-tree hierarchy our isosurface extraction algorithm requires information about all intersections of the grid edges with the isosurface. Since every grid edge of the adaptive grid is generated together with exactly one splitting face, this additional information about the intersection points and normals of each edge of the splitting face of a grid cell can be uniquely attached to the splitting face.

The hierarchy of grid cells implicitly defines a hierarchy of grid half-faces as follows: Every time a grid cell is subdivided by a splitting plane, four of its six half-faces are also subdivided by this splitting plane. Thus each splitting half-face (i.e. a half-face newly introduced by a subdivision of a cell) is the root of a 2-dimensional kd-tree hierarchy of grid half-faces whose leafs are exactly the minimal grid half-faces. Likewise, the hierarchy of grid half-faces implicitly defines a 1-dimensional hierarchy of grid half-edges, since every time a grid half-face is subdivided by a splitting line, two of its four half-edges are also subdivided by this splitting line.

Since we have to enumerate all minimal grid faces and the intersections on its grid edges during isosurface extraction, we prefer to have a data structure for the adaptive grid, for which these operations can be implemented efficiently and which could be easily constructed and maintained during modification (for example simplification) of the kd-tree hierarchy.

For these reasons we store the nested kd-tree hierarchy described above explicitly, which is done as follows. For each non-minimal grid cell we store its splitting axis and value, pointers to its two child cells and for each of its two splitting half-faces a pointer to the root of a 2-dimensional kd-tree of grid half-faces. Likewise, for each non-minimal grid half-face we store pointers to its two child half-faces, and optionally its splitting axis and value (which is identical to the splitting axis and value of the corresponding grid cell). In principle the grid half-edges could also be stored as a 1-dimesional kd-tree. But since this would introduce unnecessary overhead we simply store for each non-minimal grid half-face a sorted list of intersections of its splitting edge with the isosurface, and for each of its two grid half-edges that are subdivided by the splitting line an index to an element in the corresponding list, separating this list of intersections into two sub-lists, one for each child half-edge. This way for each grid half-edge the indices to the first and last intersection point in the corresponding list of the splitting edge are known.

If the adaptive grid is used to represent a solid object and if inside/outside information for all grid points is required (for example to be able to apply CSG operations) we might

IEEE COMPUTER SOCIETY

also store four sign values for each splitting face indicating if its corners lie inside or outside the represented object.

Note that there is no information to store for minimal grid cells or minimal grid half-faces. Therefore, we do not need to store them in our hierarchy at all.

In addition, the nested kd-tree structure can be further optimized by taking into account that there is no data to store for non-active grid edges. Thus, the 2-dimensional kd-trees can be reduced such that they contain only those grid half-faces that have active edges or children with active edges. This reduction can be done during construction of the nested kd-tree structure or afterwards by successively removing leaf nodes that do not contain any non-null data.

Note, that our nested kd-tree structure does not contain any redundant intersection data as long as two adjacent grid cells are always subdivided using different splitting values. Only if two cells are split along the same axis with the same splitting value there are two coincident splitting edges in our data structure.

3.3. The adaptive extraction algorithm

We now extend our extraction algorithm that was described in section 2 such that it can be used to generate adaptive tessellations from a given kd-tree grid.

As in the uniform case, the algorithm consists of two steps: First, polygons are generated and the connectivity is reconstructed. Second, vertices are generated and vertex locations are computed. The second step does not differ from the uniform case at all, so that we concentrate on the first step only.

Like in the uniform case one polygon is generated per active grid edge. But in this case these polygons are triangles instead of quadrilaterals, since the number of vertices of a polygon corresponds to the number of cells adjacent to the active grid edge, which is always three in our adaptive grid.

Recall, that a minimal grid face corresponds to the intersection of two overlapping minimal grid half-faces that are explicitly given in our adaptive grid. Starting at each non-minimal grid cell we simultaneously traverse the two kd-tree hierarchies of grid half-faces down to all pairs of overlapping minimal grid half-faces. This way we obtain for each edge of the two grid half-faces a list of their intersections with the isosurface. Four of these eight edges are incident to the minimal grid face. Therefore, we find the information about all intersections of the isosurface with the boundary of the considered minimal grid face in the four corresponding intersection lists.

Like in the uniform case, we generate a half-edge for each intersection on the boundary of the minimal grid face. If we construct more than two half-edges we again have to apply a disambiguation strategy as described in section 2.1.

4. Simplification framework

In this section we describe an error-controlled reduction technique for kd-tree grids. Similar to the simplification method for signed octrees by Ju et al. [10], the simplification framework for kd-tree grids presented here is controlled by error metrics. However, in contrast to their approach, our method assures that multiple surface components inside a cell are handled independently. Furthermore, apart from simplifying the object surface, we also allow controlled topological simplification of the represented object.

Figure 10 shows two examples generated using our simplification method. While the first example object has been simplified without allowing topological modifications, the second object has been simplified while allowing small topological changes.

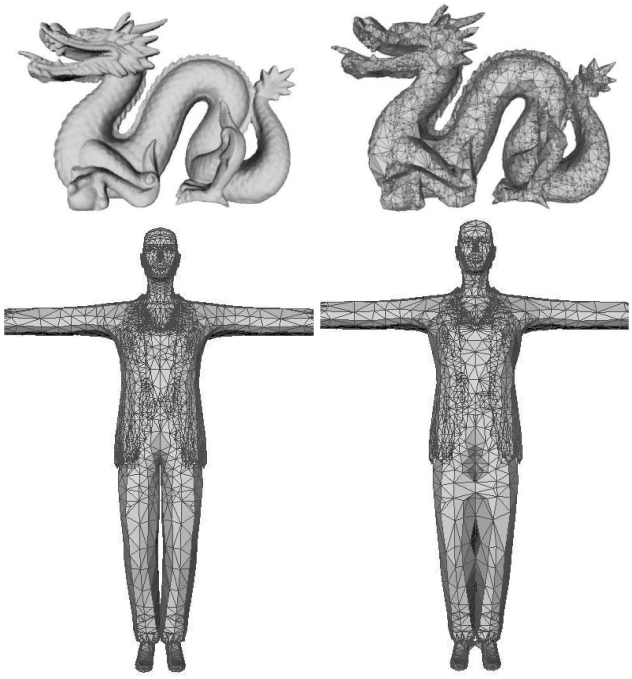


Figure 10. The example on the top shows the isosurface of an adaptive grid that has been simplified without allowing topological modifications. (The object on the top right was simplified up to an error of $\varepsilon_{on} = 0.25$, taking about 5.7 sec.) The example at the bottom shows the isosurface of an adaptive grid that has been simplified while allowing small topological changes.

IEEE COMPUTER SOCIETY

4.1. Metrics

In this subsection we show how the error between the original and the simplified representation is quantified and how it can be evaluated efficiently during the simplification process.

Given a kd-tree grid that should be reduced, containing signs at all grid points as well as intersection points for all active grid edges, the following information about the represented object is known:

- a set P_{on} of points *on* the object surface (the intersection points)
- a set P_{in} of grid points *inside* the object
- a set P_{out} of grid points *outside* the object

During the simplification process we approximately quantify the shortest Euclidean distances $d_S(v)$ from the points $v \in P_{on}$ to the isosurface of the object that is represented by the reduced kd-tree, as well as the distances $d_I(p_{in})$ and $d_O(p_{out})$ from the points $p_{in} \in P_{in}$ and $p_{out} \in P_{out}$ to the closure of its interior and exterior, respectively.

Based on these distances we define three metrics called *on-metric*, *in-metric*, and *out-metric* by Andujar et al. [1]. While the on-metric E_{on}, based on the distances $d_S(v)$, allows to quantify the distance between the two isosurfaces contained in the original and in the simplified representation, the in-metric E_{in} and out-metric E_{out}, based on the distances $d_I(g_{in})$ and $d_O(g_{out})$, allow to quantify the error introduced by topological modification, namely the removal of material in case of the in-metric and the adding of material in case of the out-metric.

The goal is to reduce our kd-tree based representation such that for the reduced representation the following property is valid:

$$E_{on} \leq \varepsilon_{on}, \quad E_{in} \leq \varepsilon_{in}, \quad E_{out} \leq \varepsilon_{out} \tag{4}$$

for user-defined error tolerances $\varepsilon_{on}, \varepsilon_{in}, \varepsilon_{out}$. By adjusting these thresholds the simplification process described in the following section can be guided.

The isosurface contained in a kd-tree grid representation is determined by a set of planes S corresponding to the intersection points and normals given at the active grid edges. To evaluate the distance of a single point to the isosurface we need to know which of these planes would correspond to nearest polygon after surface extraction. For a plane $s \in S$ let P_s be the set of points from P_{on} which are nearest to exactly that polygon of the extracted mesh which corresponds to s. Then our on-metric is defined as follows:

$$E_{on} = \max_{s \in S} \frac{1}{|P_s|} \sum_{v \in P_S} d_s(v)^2 \tag{5}$$

where $d_s(v)$ is the distance of point v to plane s.

Further, let I be the closure of the interior of an object given by its kd-tree grid representation, and let O be the closure of its exterior. Then our in-metric and out-metric are defined as follows:

$$E_{in} = \max_{p_{in} \in P_{in}} d_I(p_{in}) \tag{6}$$

$$E_{out} = \max_{p_{out} \in P_{out}} d_O(p_{out}) \tag{7}$$

Since the distance d_I of a point p_{in} to the closure of the interior I corresponds to the distance to the surface S if $p_{in} \notin I$ and is zero otherwise, we can rewrite eq. 6,7 as follows:

$$E_{in} = \max_{p_{in} \in P_{in} \setminus I} d_S(p_{in}) \tag{8}$$

$$E_{out} = \max_{p_{out} \in P_{out} \setminus O} d_S(p_{out}) \tag{9}$$

4.1.1. Dual quadrics

The evaluation of the on-metric requires the calculation of a sum of squared distances $E(n, d)$ between points $v_1, \ldots, v_k \in P_{on}$ and a plane $s \in S$ that is given by a normal n and a scalar d.

$$E(n, d) = \frac{1}{k} \sum_{i=1}^{k} (n^T v_i + d)^2 \tag{10}$$

Following Garland et al. [7] and writing

$$A = \sum_{i=1}^{k} v_i v_i^T, \quad b = \sum_{i=1}^{k} v_i, \quad c = k \tag{11}$$

where again A is a symmetric 3×3 matrix, b is a 3-vector, and c is a scalar, we can rewrite eq. 10 as follows:

$$E(n, d) = \frac{1}{k} (n^T A n - 2 d n^T b + d^2 c) \tag{12}$$

$P = (A, b, c)$ is called *dual quadric*.

As shown in [7], the optimal plane that minimizes $E(n, d)$ can be determined as follows. Assuming that the plane passes through the mean of points v_i, we get $d = -n^T b / c$. Furthermore, the normal n corresponds to the eigenvector of the covariance matrix Z of points v_i corresponding to its smallest eigenvalue. Z can be constructed from the dual quadric as

$$Z = \frac{1}{k-1} \left(A - \frac{b b^T}{c} \right) \tag{13}$$

where the factor $1/(k-1)$ may be omitted since it has no influence on the eigenvalue decomposition. The normal n computed this way is only unique up to sign. However, in

IEEE COMPUTER SOCIETY

our case the correct sign is already given by the orientation of the corresponding active grid edge.

During the simplification process we store a dual quadric for each intersection of a grid edge with the isosurface to be able to evaluate the on-metric efficiently. The dual quadrics can be stored as replacement for the explicitly given intersection information, since the intersection point and normal is already given implicitly by the dual quadric.

4.2. Simplification method

The simplification process starts with the leafs of the kd-tree hierarchy and successively merges pairs of minimal grid cells as long as the merged cell does not violate the described error criteria.

Whenever a pair of minimal grid cells is removed from the hierarchy, also the corresponding splitting face has to be removed. If any of the grid edges of that face cross the isosurface the associated dual quadrics have to be distributed to nearby active grid edges, as otherwise the intersection information associated with those grid edges would be totally lost. We recall that during surface extraction for each active grid edge a triangle is generated. Therefore the removal of an active grid edge corresponds to the removal of a triangle in the resulting mesh by collapsing the half-edge that would be generated while iterating around the splitting face. We can find the two active grid edges that correspond to the edge-adjacent neighbor triangles in the resulting mesh by iterating around the minimal grid faces that contain the active grid edge that has to be removed. Before removing the active grid edge, we first evaluate the error according to the on-metric that would be caused by the removal of the intersections on that grid edge. This is done by eigenvalue decomposition of the accumulated dual quadrics. If the removal of a pair of intersections would introduce a topological change, i.e. if in between of those two intersections material would be removed or added, we further evaluate the error according to the in- or out-metric, respectively. If none of these error criteria are violated the dual quadrics stored together with the active grid edges corresponding to the edge-adjacent neighbor triangles are replaced by accumulated dual quadrics and the splitting face is removed. This process is repeated until no pair of minimal grid cells can be merged without violating the error criteria.

5. Applications

5.1. Generation of a kd-tree grid representation for arbitrary polygonal meshes

In this section we present a method for the construction of an implicit representation of an arbitrary polygonal mesh without given connectivity information (polygon soups) by a kd-tree. Such an implicit representation can be useful for many purposes, for example if the object should be topologically simplified, if a mesh should be retesselated, or if additional tasks should be performed that require an implicit object representation and/or profit by the kd-tree hierarchy, like CSG operations.

5.1.1. Constructing the hierarchy

Based on a given polygonal mesh, we construct a kd-tree grid such that each face of the given polygonal mesh is intersected by at least one grid edge parallel to that coordinate axis which has the smallest angle to the normal of the face. This criterion ensures that each face in the input mesh has an influence on the reconstruction.

To construct the kd-tree hierarchy in a top-down manner a suitable subdivision criterion for the subdividing splitting planes has to be devised. To balance the number of polygons in each kd-tree cell we use the centers of the polygons contained in the cell as input for a median cut criterion and during subdivision we maintain three sorted lists of the x, y, z values of the polygon centers as potential splitting values. As shown in figure 7 a necessary condition for a polygon to be reconstructed is that it is intersected by at least two adjacent grid faces, i.e. two splitting planes are necessary for each polygon. Therefore, to optimize splitting results, we consider only those two planes as potential splitting axes that are most orthogonal to the polygon. Note that both strategies, the straightforward one and the optimized one lead to a depth of the kd-tree hierarchy of at most $O(\log n)$, where n is the number of polygons in the original object. After splitting a cell the polygons are divided among its children dependent on their center points. Note, that a further improvement can be made if we remove all polygons from further consideration with respect to the current splitting axes that are intersected by the currently introduced splitting plane. This can be realized by projecting the corresponding potential splitting coordinates onto the introduced splitting plane.

5.1.2. Finding intersection points

After computing the kd-tree hierarchy the active grid edges have to be detected and the corresponding intersection information for these edges have to be computed. One way to compute the intersection information would be to shoot rays and intersect them against the given polygonal mesh. However, this technique leads to problems. Due to numerical instabilities the decision on which side of a grid point an intersection lies might be inconsistent for different axes. This leads to topologically wrong representations.

Therefore, we determine the active edges and the corresponding intersections during the construction of the kd-tree grid by the following technique that is robust and also

IEEE COMPUTER SOCIETY

more efficient. In addition to the lists of potential splitting values we maintain a list of faces inside the cell during the subdivision process. In each subdivision step the faces of this list are clipped against the splitting plane, and the clipped faces are handed over to the child cells. For each vertex generated by the clipping process, we remember the axes of the planes against that it was clipped. Thus we can easily identify those vertices that correspond to the intersection points on each of the four grid edges belonging to a splitting face. The intersection information is stored together with those grid edges, either in form of an intersection point and normal or, as required for the simplification framework, as dual quadric.

5.2. CSG

CSG operations are a very nice example for the power of an adaptive implicit object representation especially when used in conjunction with a feature-sensitive isosurface extraction algorithm. This was already shown in previous octree based approaches [10].

The high efficiency of CSG operations on an adaptive representation can be even improved by our kd-tree based hierarchy. For example performing CSG operations on objects of different complexity profits by the fact that the kd-tree better adapts to local object properties.

In combination with the method for converting an arbitrary polygonal mesh into its kd-tree grid representation presented in the previous subsection, we are also able to perform CSG operations on multiple polygonal input meshes efficiently.

In comparison with the isosurface extraction, the only additional information required for performing CSG operations is an in/out sign at each grid point. Therefore, we explain in this subsection how the method described above can be extended in a straightforward manner to construct a kd-tree grid with signs at all grid points from arbitrary solid polygonal input meshes.

Assuming that the bounding box of the input object is fully contained in the root grid cell, it is known that the boundary of the root grid cell lies outside the object. Whenever a grid cell is subdivided during the construction of the kd-tree hierarchy we have to determine the signs of the corners of the splitting face. Since each of these four grid points lie on a grid edge for whose end points the signs are already known, we simply propagate the signs given at these end points to the new grid points by checking if the intersection list of the corresponding grid edge contains any intersections in between the end points and the new grid point.

As long as the input mesh represents a solid object the result is consistent, no matter from whose of the two end points the sign is propagated. Otherwise the inside/outside state for a newly introduced grid point may not be uniquely defined. However, dependent on the strategy used some nearly solid objects can be converted quite faithfully to solid ones by this simple technique, thereby allowing to apply CSG operations on that object and to produce a closed mesh by isosurface extraction.

Figure 11 shows the result of two CSG operations applied to three polygonal input meshes.



Figure 11. In this example two CSG operations were performed: the union of two slightly rotated Dragon models, followed by the subtraction of a cylinder.

6. Results

The examples in this paper were generated on an Intel Pentium IV 2.6 GHz processor with 512 MB of main memory. The adaptive grids from which the shown isosurfaces have been extracted were all build from polygonal input meshes using the technique described in section 5.1. The octree examples that were shown for comparison of the octree and kd-tree based hierarchies have been constructed using the same data structure but with a simple octree subdivision rule. Table 1 shows the times required to construct the adaptive grids and to extract the isosurfaces from those grids using our adaptive isosurface extraction algorithm described in section 3. In addition, it contains the number of polygons of the used input meshes, the maximum depth of the generated kd-trees, and the number of triangles generated by the extraction algorithm.

References

[1] C. Andújar, P. Brunet, and D. Ayala. Topology-reducing surface simplification using a discrete solid representation. *ACM Transactions on Graphics*, 21(2):88–105, Apr. 2002.

[2] J. Bloomenthal. Polygonization of implicit surfaces. *Computer Aided Geometric Design*, 5:341–355, 1988.

[3] M. Botsch, L. P. Kobbelt, U. Schwanecke, and H. Seidel. Feature-Sensitive surface extraction from volume data. *SIGGRAPH 2001 Conference Proceedings*, pages 57–66, Aug. 12–17 2001.

0-7695-2028-6/03 $17.00 © 2003 **IEEE**

IEEE COMPUTER SOCIETY

example	#polys input	gen. time	kd depth	extr. time	#polys output
1. boxes	30	0.01	10	0.01	540
9. glass	142	0.18	15	0.21	15844
2. chair	1464	1.65	21	2.02	140924
8. street	87751	8.85	18	4.54	262687
11. CSG	114976	12.38	19	6.21	304754

Table 1. Timings (in seconds) for generating the adaptive implicit representations and for extracting their isosurfaces.

[4] B. Curless and M. Levoy. A volumetric method for building complex models from range images. *Computer Graphics*, 30(Annual Conference Series):303–312, 1996.

[5] S. F. Frisken and R. N. Perry. Efficient estimation of 3D euclidean distance fields from 2D range images. *Proceedings of the 2002 IEEE symposium on Volume visualization and graphics*, pages 81–88, Oct. 28–29 2002.

[6] S. F. Frisken, R. N. Perry, A. P. Rockwood, and T. R. Jones. Adaptively sampled distance fields: A general representation of shape for computer graphics. *SIGGRAPH 2000 Conference Proceedings*, pages 249–254, July 23–28 2000.

[7] M. Garland, A. Willmott, and P. S. Heckbert. Hierarchical face clustering on polygonal surfaces. *Proceedings of ACM Symposium on Interactive 3D Graphics*, Mar. 2001.

[8] S. F. F. Gibson. Constrained elastic surface nets: Generating smooth surfaces from binary segmented data. *Lecture Notes in Computer Science*, 1496:888–??, 1998.

[9] S. F. F. Gibson. Using distance maps for accurate surface reconstruction in sampled volumes. *Proceedings of the 1998 Symposium on Volume Visualization*, pages 23–30, Oct. 19–20 1998.

[10] T. Ju, F. Losasso, S. Schaefer, and J. Warren. Dual contouring of hermite data. *ACM Transactions on Graphics*, 21(3):339–346, July 2002.

[11] P. Lindstrom. Out-of-Core simplification of large polygonal models. *SIGGRAPH 2000 Conference Proceedings*, pages 259–262, July 23–28 2000.

[12] P. Lindstrom and C. T. Silva. A memory insensitive technique for large model simplification. *Proceedings of IEEE Visualization 2001*, pages 121–126, Oct. 2001.

[13] Y. Livnat, H.-W. Shen, and C. R. Johnson. A near optimal isosurface extraction algorithm using the span space. *IEEE Transactions on Visualization and Computer Graphics*, 2(1):73–84, Mar. 1996.

[14] W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3D surface construction algorithm. *Computer Graphics*, 21(4):163–169, July 1987.

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

[16] R. N. Perry and S. F. Frisken. Kizamu: a system for sculpting digital characters. *SIGGRAPH 2001 Conference Proceedings*, pages 47–56, 2001.

[17] T. Poston, T.-T. Wong, and P.-A. Heng. Multiresolution isosurface extraction with adaptive skeleton climbing. *Computer Graphics Forum*, 17(3):137–148, 1998.

[18] R. Shekhar, E. Fayyad, R. Yagel, and J. F. Cornhill. Octree-based decimation of marching cubes surfaces. *Proceedings of IEEE Visualization'96*, pages 335–342, 1996.

[19] R. Shu, C. Zhou, and M. S. Kankanhalli. Adaptive marching cubes. *The Visual Computer*, 11(4):202–217, 1995.

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

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

[22] Z. J. Wood, M. Desbrun, P. Schröder, and D. Breen. Semi-regular mesh extraction from volumes. *Proceedings of IEEE Visualization 2000*, pages 275–282, Oct. 2000.

[23] Z. J. Wood, H. Hoppe, M. Desbrun, and P. Schröder. Isosurface topology simplification. *ACM Transactions on Graphics, submitted*, 2002.