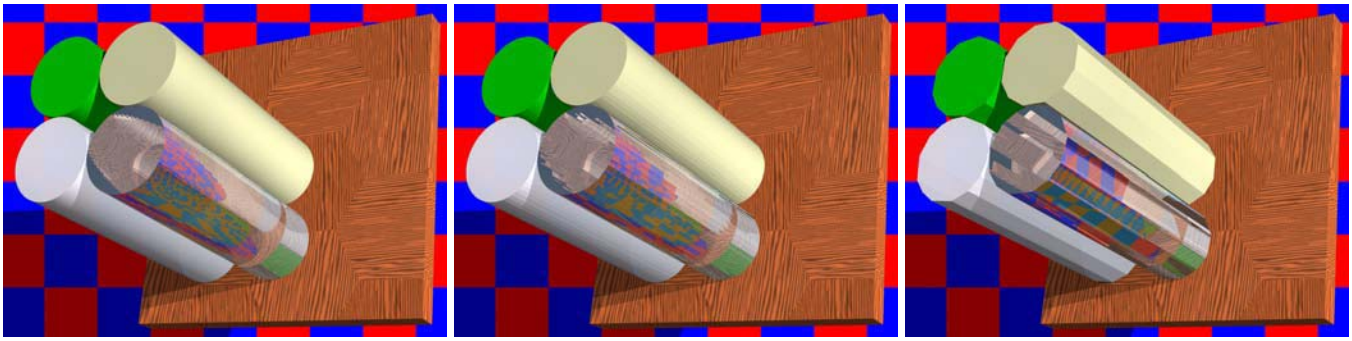
# Dual Contouring with Topology-Preserving Simplification Using Enhanced Cell Representation

Nan Zhang Wei Hong Arie Kaufman

Center for Visual Computing (CVC) and Department of Computer Science
Stony Brook University, Stony Brook, NY 11794-4400, USA *



(a) (b) (c)

Figure 1: Simplifying the zero-isosurface of a directed distance volume of $256^3$ using our topology-preserving isosurface simplification algorithm. Although the cylinders and the box are very close to each other, they don't touch, and thus there are several disconnected surface components in this volume. Note that disconnected surface components are assigned different materials and are clustered independently. (a) $\delta^2 = 0$ and $t = 97K$. (b) $\delta^2 = 10^{-6}$ and $t = 23K$. (c) $\delta^2 = 10^{-4}$ and $t = 2494$. ($\delta^2$: quadric error threshold, $t$: triangle count.)

**ABSTRACT**

We present a fast, topology-preserving approach for isosurface simplification. The underlying concept behind our approach is to preserve the disconnected surface components in cells during isosurface simplification. We represent isosurface components in a novel representation, called enhanced cell, where each surface component in a cell is represented by a vertex and its connectivity information. A topology-preserving vertex clustering algorithm is applied to build a vertex octree. An enhanced dual contouring algorithm is applied to extract error-bounded multiresolution isosurfaces from the vertex octree while preserving the finest resolution isosurface topology. Cells containing multiple vertices are properly handled during contouring. Our approach demonstrates better results than existing octree-based simplification techniques.

**CR Categories:** I.3.6 [Computer Graphics]: Methodology and Techniques—Graphics Data Structure and Data Types

**Keywords:** isosurface simplification, isosurface extraction, topology preservation, vertex clustering

## 1 INTRODUCTION

Isosurface extraction is a very useful tool for visualizing volume datasets. Due to the improvements in resolution and accuracy of acquisition devices, the resolution of volume datasets has been increasing dramatically. For large volumes, the extracted triangles which make up an isosurface can easily overwhelm the interactive rendering capability of modern computers. Multiresolution isosurface extraction methods have been developed to address this issue. Usually, we want to preserve the topology of the finest resolution isosurface for the simplified one. Most of the topology-preserving isosurface simplification algorithms extract and maintain all critical points of the volume for topology preservation. They follow the concept used in a top-down based isosurfacing approach [20]: if the adaptive volume contains all critical points (points where an isosurface would change genus or number of components), then the extracted isosurface has the same topology as the corresponding isosurface on the finest resolution. However, the expense is a lower simplification capability at the cells containing critical points since these cells cannot be merged.

Our solution to the topology-preserving isosurfacing problem on large, uniformly-sampled volumes is a bottom-up approach and therefore is based on a different idea: preserving the disconnected surface components in each cell during isosurface simplification. It is inspired by the observation that the volume grid cuts a surface into patches and these patches can be actually merged into a whole surface if the cell merging sequence can be altered. During isosurfacing, we extract the isosurface from the finest resolution volume. The isosurface components are then hierarchically simplified using a connectivity-guided method, where connected components are merged and disconnected components are preserved. The simplification capability of our approach is determined by the capability of representing the disconnected surface components inside a cell. There is no need to extract and maintain critical points, as in many other solutions.

To represent the surface components in a cell, we introduce a new data structure—the enhanced cell, where each isosurface component is represented inside a cell by a vertex with encoded connectivity information. These vertices can be simplified using a vertex clustering algorithm [16], where vertices that fall into the same cell are clustered into one representative vertex. Since vertices are clus-

*Email: {nanzhang|weihong|ari}@cs.sunysb.edu

IEEE Visualization 2004
October 10-15, Austin, Texas, USA
0-7803-8788-0/04/$20.00 ©2004 IEEE

tered regardless of the topology, directly applying this algorithm for isosurface simplification will introduce more serious problems than for a mesh. The tiny triangles which make up an isosurface are more sensitive to position changes. Besides, human eyes are especially sensitive to topology errors and the resulting gradient abnormality (cf. [18]). To solve the topology problem in vertex clustering, a connectivity-guided clustering algorithm is applied, where the vertices from different surface components are clustered independently. Surface components are detected by a graph algorithm that uses the connectivity information of the vertices.

Our simplification algorithm contains three steps: (1) extracting and representing the finest resolution isosurface components, (2) building a vertex octree, and (3) extracting simplified isosurfaces from the vertex octree. The simplified isosurfaces preserve the topology of the isosurface in the finest resolution. Due to the space limitation, we omit the discussion here [1].

The primary contributions of this paper are:

- An encoding scheme, called enhanced cell, to represent the surface components in a cell. Our scheme handles multiple disconnected surface components, singular connections and up to two intersections per cell edge (including loops).
- A bottom-up cell merging algorithm, where topology-preserving vertex clustering is used to cluster the vertices in a cell. Thanks to the enhanced cell definition, we are able to encode complex surface components created in a cell during hierarchical merging.
- An enhanced dual contouring algorithm for the vertex octree built from bottom-up cell merging. We extend the SurfaceNets algorithm [5, 14] to handle multiple representative vertices. Several vertex selection strategies have been supported to specify the active vertices used for contouring.

## 2 Related Work

**Isosurface Extraction and Simplification**: The problem of isosurface extraction from volume datasets has been studied extensively. Limited by the space here, we only discuss the most relevant documents. The Marching Cubes (MC) algorithm [10] has been widely used to extract triangular isosurfaces from the scalar values sampled in rectilinear lattices. It can be naturally integrated into the hierarchical octree data structure for multiresolution isosurface extraction. The MC algorithm has been further extended by Nielson [13]. The major issue in multiresolution MC algorithms is that a crack patching stage is explicitly required when cells of different size meet, since the piecewise linear approximation surface is no longer continuous. Various patching schemes have been proposed, such as point deletion [19] and point insertion [23]. However, these strategies usually don't allow level differences of more than one. Furthermore, the original MC algorithm is unable to recover sharp features in a volume.

Recently, feature-preserving isosurface extraction techniques have been reported. Kobbelt et al. [8] have presented a directed distance field representation and an extended MC algorithm for feature-sensitive isosurface extraction. Ju et al. [7] have described a dual contouring algorithm for Hermite data. This algorithm avoids the explicit feature testing stage in the extended MC algorithm by computing a minimized representative vertex for each cell and connecting the vertices using the SurfaceNets algorithm [5, 14]. Both surface extraction algorithms assume up to one feature per cell. Varadhan et al. [22] have further extended Ju et al.'s work by handling more than one feature in a cell. We are interested in the SurfaceNets family algorithms because of its feature-preserving ability and crack-free nature in multiresolution contouring. However, we have a different assumption than Varadhan et al. [22]. We are more inclined to believe that most of the finest level cells are simple and multiple surface components are formed in the hierarchical simplification process.

Tetrahedral-based isosurface extraction has been presented [4, 6]. Gerstner and Pajarola [4] have further studied the rules for extracting a simplified isosurface while maintaining its topology in adaptive tetrahedral refinement. A lookup-table method has been used to extract critical points. Isosurface topology is preserved by preserving all the critical points in the adaptive tetrahedral mesh. Ju et al. [7] have applied a similar idea to simplify a volume grid tagged with material information. There are existing techniques for isosurface and volume topology simplification. In order to simplify the topology, these algorithm usually alter the voxel values. Wood et al. [24] have analyzed the topology of the input isosurface based on its Reeb Graph. Small handles are located. Then, the corresponding area in the original volume is changed to remove these handles. Szymczak and Vanderhyde [21] have described a volume topology simplification algorithm where a topology-sensitive carving technique is applied to progressively remove the boundary cells. A variant of the MC algorithm is used to extract an isosurface with simplified topology. However, it is not clear how to combine topology simplification with multiresolution isosurface extraction. Also, the topology-simple isosurface is not guaranteed to have fewer triangles than the complex one.

**Vertex Clustering for Meshes**: Although it lacks topology-preservation capability, vertex clustering remains a very powerful simplification technique for large mesh data. Rossignac and Borrel [16] have reported one of the earliest vertex clustering algorithms on a uniform grid. Many enhancements have been presented for this prototype algorithm. Lindstrom [9] has used quadric error metrics [3] to improve the positioning of the representative vertices. His algorithm is further designed for out-of-core simplification using polygon soups. Luebke and Erikson [11] have employed a hierarchical octree data structure to partition the space. Shaffer and Garland [18] have presented BSP-tree partitioning. Although these algorithms reduce the clustering errors, the topology preservation problem has not been tackled. Our algorithm avoids the topology problem by allowing multiple representative vertices in a cell and clusters vertices using connectivity.

## 3 Definition of Enhanced Cells

Although it appears straightforward to add more representative vertices into a cell, the representation of their connectivity is not that trivial. Both the MC algorithm and the discretized MC algorithm [12] use a cube-based approach, where the binary values of the eight corner points are used to determine the surface components and the connectivity of each component. However, each representative vertex is unable to find its own connectivity using a shared encoding vector of the 8 corner points. Storing the intersection points on edges, such as the directed distance method, may be another choice. However, the intersection points themselves are useless in our simplification process. Also, the intersection points are not associated with the representative vertices explicitly. Consequently, we decide to extend the cube-based encoding by assigning each vertex an encoding vector. Furthermore, we want to incorporate the ability of encoding 2 intersection points per edge, as presented by Varadhan et al. [22]. Therefore, we define an enhanced cell as follows:

[1] A report of proof can be downloaded from our website: http://www.cs.sunysb.edu/∼vislab/papers/tpvc.pdf.

**An enhanced cell** is a cell with a list of representative vertices. Each representative vertex has its own inside/outside classification information for the eight cell corner points.

For each representative vertex $v_i$, the classification information $C_i$ encodes the connectivity of $v_i$ with vertices in its neighborhood. $C_i$ is an encoded vector of $(c_0,...,c_7)$. Each $c_i$ is a tri-value scalar in the range $[0,X,1]$, where 0 stands for outside the isosurface object, 1 stands for inside the object, and $X$ means unknown. When the classification code of the two end points of an edge are $(0,1)$ or $(1,0)$, we refer to it as a sign change, since the isosurface must pass through this edge. The sign changes are directed, which means $(0,1)$ is different from $(1,0)$ [2]. The directed sign changes are used to distinguish two vertices that share the same edge for connectivity encoding. The $X$ value is introduced for encoding singular connectivity. In detecting sign changes on edges, no sign changes are indicated if one end-point on an edge has a value $X$. Within a cell, the connectivity encoding vector of each vertex is unique. The connectivity set of all vertices must satisfy the following rule:

- There are no two same directed sign changes constructed from the coding vectors.

This rule enforces that every directed sign change on the edge corresponds to only one vertex. Therefore, the encoding power of the new representation is limited to at most two different directed sign changes per cell edge. Our scheme has higher connectivity encoding capability than the cube-based encoding schemes. It can represent more complex surface components in a cell. We show some examples in 2D. In Figure 2a, the coding vectors for the two representative vertices $v_0$ and $v_1$ are $(1,0,0,0)$ and $(0,0,0,1)$, respectively. In Figure 2b, the coding vectors for the two vertices $v_0$ and $v_1$, which lie on the two parallel lines, are $(0,0,1,1)$ and $(1,1,0,0)$, respectively. The enhanced cell representation cannot represent the shape shown in Figure 3, where the coding rule is violated.

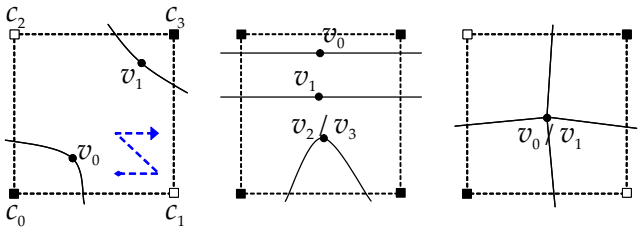



Figure 2: The enhanced cell representation for several cases, where the blue dashed line in (a) shows the corner indexing sequence. (a) Two disconnected surface components. (b) Two intersection points per edge, including a loop case. (c) Non-manifold topology.

Our new representation can further encode loops and singular connections inside a cell. Bloomenthal and Ferguson [1] have addressed the non-manifold polygonization problem. Their method can't handle loops in cells. In our method, we introduce a vertex duping technique, where a vertex can be split into two or more, each with the same position but different coding. For example, for vertex $v_2$ on the loop of Figure 2b, we split it into $v_2$ and $v_3$, which have the same position but different classification codes: $(1,0,X,X)$ and $(0,1,X,X)$. Each coding vector has only one sign change for all the four edges. Therefore, the surface components in Figure 2b can be represented by the four vertices and their classification codes:

$$v_0:(0,0,1,1),\ v_1:(1,1,0,0),\ v_2:(1,0,X,X),\ v_3:(0,1,X,X).$$

[2] We consider the direction of the sign changes along the axis-aligned grid lines only. The directions can be $+x$, $-x$, $+y$, $-y$, $+z$ or $-z$.

Figure 2c shows a non-manifold topology example. Similarly, we split the vertex $v_0$ into two to guarantee manifold topology. The difference is that in the loop case, we mark the two vertices as the same vertex, while in Figure 2c, they are treated as different vertices.

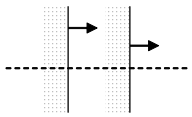


Figure 3: Our encoding scheme can't represent two surface patches which generate the same directed sign change on an edge (shown as a dashed line).

## 4 Data Conversion

Using the enhanced cells, the isosurfaces of various volume representations can be encoded. For a user-defined isovalue, the data conversion process extracts the isosurface and represents it by the enhanced cell representation. Although we only use regular volumes, adaptively-sampled volumes can be represented in the same way. Volume datasets usually include: scanned data, such as the data from CT, MRI and UltraSound, and synthesized data, such as the data generated by scan-conversion algorithms. Volume datasets can also be classified into density/distance volumes and directed distance volumes. The scanned datasets are all density volumes. The synthesized volumes can be density/distance volumes or directed distance volumes, depending on the scan-conversion algorithms used. After conversion, a vertex octree is initialized, where all the homogeneous regions are maximally collapsed and the boundary leaf cells are represented in the enhanced cell form.

### 4.1 Density/Distance Volume

In density/distance volume datasets, for a given isovalue, we use the lookup table of the MC algorithm to determine the disconnected surface components. For each component $S_i$, we generate a representative vertex $v_i$ by averaging the intersection points between this surface component and the cell edges. The coding vector $C_i$ for each vertex $v_i$ is determined by considering only the existence of the cluster $S_i$ in the cell. One coding vector is used for each representative vertex.

### 4.2 Directed Distance Volume

Meshes and implicit surfaces are usually scan-converted (or voxelized) into regular distance fields. Alternatively, they can be sampled into directed distance volumes for feature-sensitive isosurface reconstruction. In the directed distance volumes, explicit intersection points with optional normal information are stored for each edge of the cell. The difference between voxelizing meshes and voxelizing implicit surfaces is mainly the intersection computation. The data conversion process inputs a directed distance volume of uniform resolution. Each cell edge may contain up to two intersection points with associated normals [22]. The intersection points are grouped to find the surface components they belong to. A representative vertex is computed by minimizing a quadric error function (see Section 5.3 for detail).

Similar to the ambiguity problem in the MC algorithm, there is ambiguity in constructing surface components from the edges with up to two intersection points per edge. To find the surface components in a cell, we develop a robust algorithm, where repeated edge collapsing tests [4] are performed. Starting from a corner point, the algorithm performs a depth-first traversal on the cell edges and stops at the first intersection point it meets. The edges with no intersection points are skipped. The intersection points which are first

met are grouped into one surface component. At the same time, these intersection points are deleted from the cell. A new starting point is selected and the traversal is continued until all the intersection points are deleted from the cell. To robustly find the surface components, the choice of the starting point is crucial. We build a score board for all the corner points. The corner point with the highest score is chosen as the starting point. The scoring formula is determined by the number of intersection points associated with its three corner edges, the maximum normal deviation of the intersection points, and the in/out classification value of this corner point. In addition, we always prefer a starting point that groups all the vertices into one surface component.

Because of the existence of edges with two intersection points in a surface component, the computation of the coding vectors is more complicated than that of the density/distance volume data. The number of split vertices depends on the number of coding vectors generated for each surface component. The following greedy algorithm computes the coding vectors of a cluster of intersection points:

1. Initialize the coding vector buffer with $X$. Classify the cell edges into two sets: 1-edge (only one intersection point per edge) and 2-edge (two intersection points per edge).
2. Construct a coding vector $C_0$ using only the 1-edge edges. The inside/out classification values of the corner points are obtained using the normal directions of these intersection points.
3. Update $C_0$ using the intersection points from the 2-edge set, until unable to proceed. In such cases, there will be a wrong coding vector created (introducing conflicting corner classification or non-existing sign changes) if we add one more.
4. Construct a new coding vector $C_i$ ($i > 0$) using the intersection points from the 2-edge set, until unable to proceed.
5. Repeat step 4 until all intersection points have been exhausted.

Usually 1-2 coding vectors are sufficient to encode the connectivity, even for complicated cases. Figure 4a shows the topologically equivalent graph of a cube. In Figure 4b, even a complete edge graph where each edge has two intersection points can be encoded using two complement coding vectors. However, this is not always true. In Figure 4c, the graph with three 2-edge edges can't be encoded with only two vectors since there should be no sign changes for the top edge. Four coding vectors are required, instead. In our experiments, four is the maximum number for all examples. After computing the coding vectors, the representative vertex will be split if two or more coding vectors are used. These vertices are given one common id to indicate that they are the same split vertices.

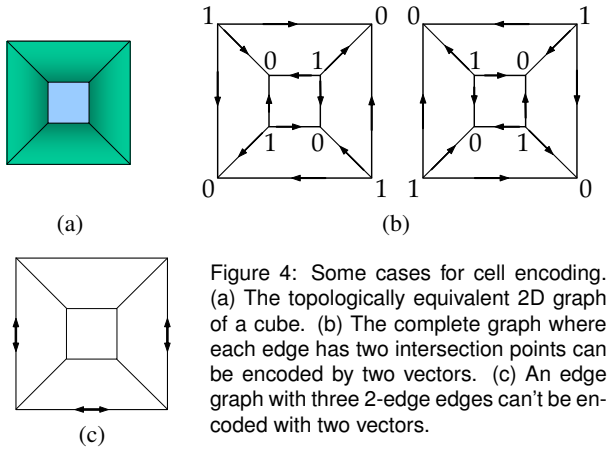



Figure 4: Some cases for cell encoding. (a) The topologically equivalent 2D graph of a cube. (b) The complete graph where each edge has two intersection points can be encoded by two vectors. (c) An edge graph with three 2-edge edges can't be encoded with two vectors.

## 5 Building Vertex Hierarchy

The vertex hierarchy is built by hierarchically merging the octree cells in a bottom-up manner. During cell merging, the vertices in the child cells are clustered and a new representative vertex is created for each surface component. A new cell can't be created if the surface components in the cell cannot be represented by the enhanced cell structure. For each cell merging, there are three steps:

1. Identify the connected surface components for the representative vertices in the child cells.
2. Test whether the coding vectors of the child vertices in each surface component can be merged or not. If yes, go to step 3, otherwise quit.
3. Construct a new representative vertex and new coding vectors for each distinct surface component.

### 5.1 Identifying Surface Components

During bottom-up cell merging, it is very common that the representative vertices in the child cells are from independent surface components. Clustering them into one single vertex will change the topology. Although in level-of-detail rendering, topology change is sometimes advocated, such as the edge contraction operator [3], the topology change will lead to much more error in isosurface simplification. To preserve topology, we separate the surface components and cluster them independently. In our approach, we first compute a connectivity graph of the vertices. Then, we compute the distinct surface components from this graph. To construct the connectivity graph, we use the coding vector associated with each vertex to recover the connection between vertices. Our algorithm is described as follows:

1. Initialize the vertex sets with the representative vertices from the eight child cells. Each vertex is an independent set.
2. Find connected surface components. For each cell edge with a directed sign change, the 4 vertices sharing that directed sign change are considered to be in the same component.
3. Merge the connected vertex sets to minimize the set count.
4. For the vertices with identical vertex id in a child cell, merge the sets they belong to into one set.

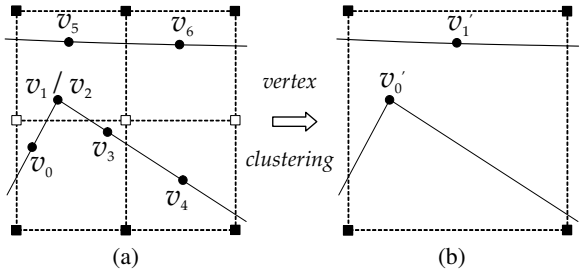



Figure 5: Identifying surface components in cells. (a) The child cells. (b) The merged cell.

In Figure 5, we show an example of identifying surface components in 2D. Simplifying surfaces such as this is straightforward for existing mesh simplification algorithms. However, for isosurface simplification algorithms which rely on one representative vertex per cell, the simplification is impossible. According to the components identifying steps, we generate the results and list them in Table 1. Note that in the third step of Table 1, since $v_1$ and $v_2$ are

the same vertex, the two sets associated with them: $\{v_0, v_1\}$ and $\{v_2, v_3, v_4\}$ are merged into one set. Finally, two distinct surface components are found by the identifying algorithm.

Table 1: Steps for identifying the surface components in Figure 5.

| | |
|---|---|
| 1 | $\{v_0\}, \{v_1\}, \{v_2\}, \{v_3\}, \{v_4\}, \{v_5\}, \{v_6\}$ |
| 2 | $\{v_0, v_1\}, \{v_2, v_3\}, \{v_3, v_4\}, \{v_5, v_6\}$ |
| 3 | $\{v_0, v_1\}, \{v_2, v_3, v_4\}, \{v_5, v_6\}$ |
| 4 | $\{v_0, v_1, v_2, v_3, v_4\}, \{v_5, v_6\}$ |

### 5.2 Merging Coding Vectors

For each surface component, a set of coding vectors needs to be generated from the existing coding vectors of the child vertices $\{v_i\}$. To ensure that the enhanced cell definition is not violated, we need to count the directed sign changes on edges of the coarse cell. This definition enforces that for each edge, at most two different directed sign changes exist, one in each direction. The child cells can't be merged if this rule is violated.

The edges of the child cells can be classified into three types: inner-edges (edges that are not mapped to the boundary of the merged cell), edge-edges (edges that are mapped to the boundary edges of the merged cell), and face-edges (edges that are mapped to the boundary faces of the merged cell). During the coding vector merging process, the edge-edges of the child cells are mapped to the corresponding edges of the coarse cell. Their coding information is used to generate the new coding vectors. The merging process uses an algorithm similar to the one used for computing the coding vectors of intersection points in Section 4.2. The inner-edges and the face-edges are discarded since they can't be mapped to the edges of a coarse cell and have no contribution to the coding vectors. It is possible that the merged coding vectors show no connection with other cells. In such cases, we reject merging the child cells to prevent losing surface components.

### 5.3 Positioning Representative Vertices

We use the quadric error function (QEF) [2] as the vertex positioning metric. QEF has been widely use as an error metric for mesh simplification [3, 9, 18]. Different from the previous vertex clustering algorithms [7, 9, 18], where one cell is associated with a QEF, we associate each surface component $S_i$ in the cell with a quadric error function $Q$, which is defined as:

$$Q(x) = \sum Q_i(x) \tag{1}$$

where $Q_i(x)$ is the QEF of a vertex $v_i$, $v_i \in S_i$ and $v_i$ is in a child cell. For boundary leaf cells, $Q_i(x)$ can be evaluated using the planes passing through the cells. The representative vertex of $S_i$ can be evaluated by minimizing $Q$.

The advantage of using a quadric error function is that each $Q_i$ can be represented by the following quadric form:

$$Q_i(x) = x^\top A_i x - 2b_i^\top x + c_i \tag{2}$$

where $x$ is the representative vertex, $A_i$ is a $3\times3$ matrix, $b_i$ is a column vector, and $c_i$ is a constant. Similarly, their summarization Q is:

$$Q(x) = x^\top A x - 2b^\top x + c \tag{3}$$

$Q(x)$ represents the sum of square distance of $x$ to a set of planes passing the cell. Normally, the position of $x$ can be solved by solving a linear equation $Ax = b$ to minimize the square distance. To robustly produce the best vertex, we use the following equation to solve the least square optimization problem [9].

$$x = \bar{x} + A^{-1}(b - A\bar{x}) \tag{4}$$

where the seed point $\bar{x}$ is the center of the cell. If the matrix $A$ is singular or near singular, the minimization can't be solved by the Gaussian elimination method. In these cases, we adopt the singular value decomposition method (SVD) [15]. As pointed out by Schaefer et al. [17], using the center as the seed point sometimes causes the minimized point to be outside the cell. In the SVD method, although part of the solution space is inside the cell, the computed solution is not guaranteed to be in the cell. Actually, Equation 4 only gives a solution that has the minimized distance to the set of planes and is close to the seed point. In our system, we simply use the average of all the representative vertices, $\bar{v} = (\sum v_i)/n$, as the seed point.

## 6 Enhanced Dual Contouring

After the hierarchical clustering stage, a vertex octree has been created, but no isosurface generated. In the contouring stage, the surface that satisfies a specified condition is reconstructed. Instead of storing several levels of simplified isosurfaces, we favor on-the-fly polygon extraction from such a hierarchy. There are several reasons: (1) For any user-defined error threshold, a topology-preserving isosurface can be extracted from the vertex hierarchy. (2) The vertex hierarchy allows a vertex to be dynamically refined or collapsed in a topology-preserving style, thus makes view-dependent rendering possible. (3) Other methods of isosurface visualization can be supported, such as the Region of Interest (ROI) isosurfacing. The ROI isosurfacing specifies a low error threshold on the regions of interest, while making the outside regions coarse.

We name our polygonization algorithm enhanced dual contouring since it extracts meshes from a vertex octree where multiple vertices may be stored in a cell and uses a vertex selection procedure to decide the active vertex set. To shorten our narration, we only discuss the polygonization strategy that employs single error-based strategy. The ROI isosurfacing can be easily derived by using region-based vertex selection strategies. In the single error-based strategy, the vertices used in polygonization are chosen according to a given quadric error threshold $\delta^2$. A vertex $v$ can be marked as active and selected for polygonization only if its quadric error $Err_v \le \delta^2$. We discuss two types of polygonization:

- Static polygonization, where only the vertices in the boundary leaf cells of the octree are used to construct the isosurface. This polygonization algorithm is similar to the extended dual contouring algorithm [22].
- Dynamic polygonization, where the vertices in the intermediate level cells, which are created by hierarchical merging, are also used to connect the surface.

### 6.1 Static Polygonization

The static polygonization corresponds to the case where $\delta^2 = 0$. By default, all the representative vertices in the boundary leaf cells are initialized as active. In such a case, the polygonization algorithm is a straightforward extension of the SurfaceNets algorithm for multiple representative vertices. In a uniform resolution volume, for each edge with a directed sign change, the vertices from the four cells sharing that edge are connected to form a quad. Since there are multiple vertices in a cell, for each edge with a directed sign

change, finding the corresponding vertex is performed by a sequential search of the vertex list stored in the cell. In the multiresolution cases, there might be only three cells involved and only one triangle is constructed. Note that the static polygonization algorithm also supports the crack-free multi-resolution isosurface extraction since it is still a dual method to the MC algorithm.

### 6.2 Dynamic Polygonization

The dynamic polygonization corresponds to the case where $\delta^2 > 0$. Every vertex $v$ in the isosurface is required to satisfy $Err_v <= \delta^2$. Before polygonization, an explicit active vertex selection stage is applied, where the whole octree is recursively visited and all the representative vertices are checked for activeness. The potential problem in the selection is that there might be cases where some of the vertices in a cell have errors higher than the threshold and the remainder lower than the threshold. Figure 6 gives such an example, where vertex $a_0$ is merged from $a_1$, $a_2$ and $a_3$ and vertex $b_0$ is merged from $b_1$, $b_2$ and $b_3$. Assume $Err_{a_0}, Err_{b_1}, Err_{b_2}, Err_{b_3} \leq \delta^2$ and $Err_{b_0} > \delta^2$. Therefore, although $a_0$ and $b_0$ are in the same cell, only $a_0$ can be used for polygonization. To extract the surface component represented by $b_0$, the child vertices of $b_0$ have to be used. However, the static polygonization algorithm is performed in a cell-by-cell order. An indication is required to extract the surface components in the child cells.

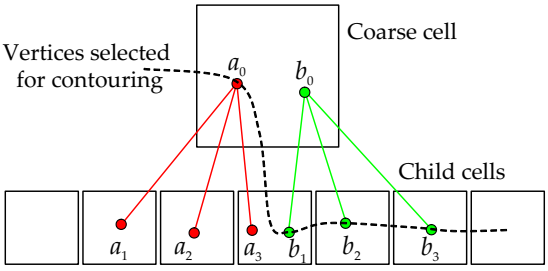



Figure 6: During dynamic polygonization, vertices which have error $Err > \delta^2$ are not selected, such as the vertex $b_0$. Instead, the child vertices ($b_1$, $b_2$, and $b_3$) are selected for contouring.

To handle the above problem in our dynamic polygonization algorithm, for each vertex $v$, a *parent-vertex* field is introduced. The *parent-vertex* field stores a pointer to the vertex that $v$ is merged to. This field is initialized during the hierarchical merging stage. The vertex selection stage is modified. The *parent-vertex* field is checked for each vertex. If the parent vertex or ancestor has been activated, all the decedents must be deactivated. Otherwise, the representative vertices are classified according to $\delta^2$. The vertices within the error threshold are marked as active. In the modified polygonization algorithm, the *parent-vertex* is also checked to prevent missing surface components. Now, in Figure 6, since $a_0$ is used, $a_1$, $a_2$, and $a_3$ are disabled. On the contrary, since $b_0$ is not used, $b_1$, $b_2$, and $b_3$ are enabled. Note that vertices $a_3$ and $b_1$ are in the same cell whereas only $b_1$ is enabled.

## 7 Implementation and Results

### 7.1 Implementation

We have implemented our simplification algorithm in C/C++ and run all the experiments on a uni-processor 3.0GHz Intel Pentium IV PC running Windows XP, with 512M RAM and NVIDIA GeForce4 graphics board. In all the examples, the physical size of the octree is set to 1.0. The vertex octree has been implemented as a Hash tree to ease the neighborhood searching operations frequently used in our polygonization algorithms. In the hierarchical clustering algorithm, the quadric error metrics are used locally. Therefore, a buffer of 30 entries is sufficient for simplification. In the final vertex tree, for each vertex we store the quantized vertex position (6 bytes), a quadric error (4 bytes), a parent pointer (4 bytes), the vertex id (1 byte), a coding vector (2 bytes), and some tagging information (1 byte). Totally 18 bytes are used for each vertex. The simplification algorithm generates about $0.3n$ extra vertices in the vertex hierarchy, where $n$ is the count of vertices in the finest resolution. Therefore, the space requirement for this algorithm is about $O(1.3n)$, plus the octree cost. The simplification time complexity is also $O(1.3n)$, since each vertex is visited only once.

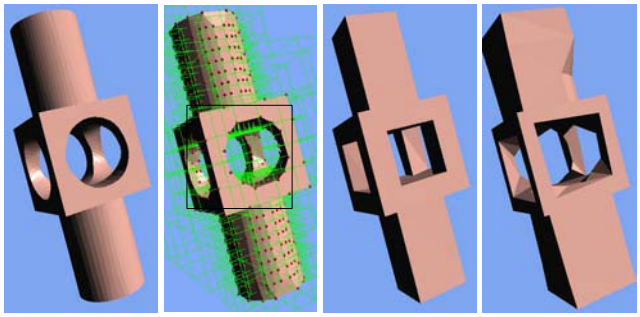
### 7.2 Performance

We have applied our algorithm on different volume datasets: density/distance volumes, scan-converted polygon meshes, and scan-converted implicit surfaces. The voxelization time for polygonal meshes and implicit surfaces varies from several seconds to about one minute. However, it is purely preprocessing cost. The time for data conversion is usually within seconds. There are two stages in the performance measurement: the simplification stage, where the vertex octree is built hierarchically, and the polygonization stage, where a user-specified error threshold is given and the isosurface satisfying that tolerance is extracted. The timing results for some test data are shown in Table 2. The advantage of separating the two stages is reflected in the timing for polygonization. As indicated in Table 2, since the representative vertices are computed in advance and stored in the cells, the surface extraction is much faster, compared to the result reported by Varadhan et al. [22], despite the fact that a deeper octree is used.

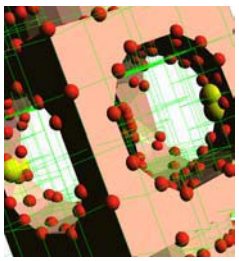
Table 2: Simplification and polygonization timing (in sec.) for voxelized objects.

| Model | Octree level | Simp. | Polygonization time | | |
|---|---|---|---|---|---|
| | | | $\delta^2 = 10^{-6}$ | $\delta^2 = 10^{-4}$ | $\delta^2 = 10^{-2}$ |
| Fig. 1 | 8 | 1.03 | 0.023 | 0.01 | 0.00 |
| Fig. 7 | 8 | 0.82 | 0.02 | 0.00 | 0.00 |
| Fig. 8 | 9 | 5.56 | 0.28 | 0.04 | 0.01 |

Figure 1 demonstrates that disconnected, voxelized models are clustered independently. These objects are assigned different material properties. After simplification, the isosurfaces remain disconnected. In Figure 7, we further show the effectiveness of caching the surface components in the intermediate levels. The zero-isosurface of a voxelized mechanical part dataset is simplified to a symmetric object of 152 triangles (Figure 7c), where each octree cell contains only one representative vertex. However, the intermediate simplification result (Figure 7b) does contain some cells with multiple representative vertices (drawn as yellow spheres) in each cell. With the help of the enhanced cell representation as a temporary storage, these cells can be merged with other cells. Another interesting observation we gain from this dataset is that the adaptive sampling process with maximum octree level of 4 creates an inferior model (Figure 7d) than our simplification result, although more cells are used. This effect can be explained by the fact that during the adaptive sampling, the representative vertices are positioned according to the QEFs constructed from the intersection points on the cell edges. Since the sampling grid is coarse, these vertices are not as good as a simplification result where the vertices are actually the average of many vertices.



(a) $t = 128K$ (b) $t = 1442$ (c) $t = 152$ (d) $t = 224$



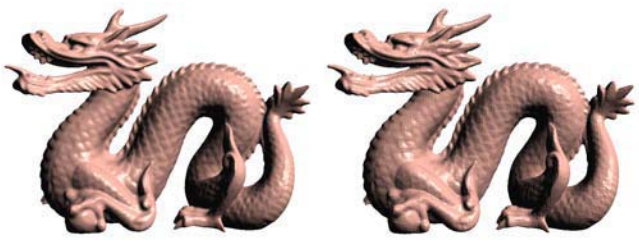
(e)

Figure 7: Simplifying the zero-isosurface of a voxelized mechanical part dataset. (a) The original model. (b) Simplified, where $\delta^2 = 2.5 \times 10^{-3}$. The octree subdivision is shown in green lines. (c) The final simplification result, where $\delta^2 = 1$. (d) An adaptively sampled example. (e) Zoom-in view of the rectangular region in (b). The representative vertices are shown as red (one point per cell) or yellow (multiple points per cell) spheres.

Figure 8 shows quadric error based simplification for the isosurface of a distance volume, where about 91% triangles are decimated. Figure 9 shows ROI isosurfacing on two density volumes, where two different ROI methods are used: a rectangular and a spherical. The spherical ROI has gradual error increasing from its center. The versatility of our polygonization algorithm is attributed to a flexible vertex selection procedure on the vertex hierarchy. For a given error threshold, a vertex is selected based on its own error and is independent of the errors of other vertices. At the same time, the polygonization speed is only slightly slower than the dual contouring algorithm. The added cost is mainly the time for matching vertices and traversing the vertex hierarchy in finding the active vertices.

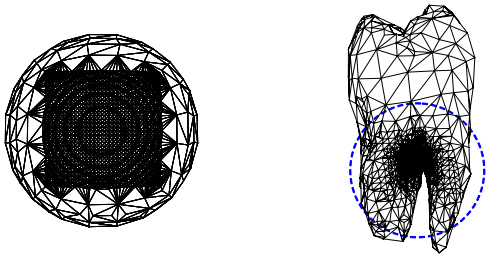
### 7.3 Comparison with Previous Methods

Compared with our algorithm, the simplification algorithm described by Ju et al. [7] is a more conservative algorithm. The core of their algorithm is a manifold test based on Gerstner and Pajarola's algorithm [4]. Their algorithm requires that in each coarse cell, the shape should be a pseudo-manifold (each material is a manifold). For contouring, their method has the limitation of only one vertex per cell. However, we have shown that two disconnected surface components in a fine level cell can be merged into one surface component in the coarse level, such as the example in Figure 5 where the two surface components represented by vertices $v_0$ and $v_3$ can



(a) $\delta^2 = 0, t = 900K$. (b) $\delta^2 = 2 \times 10^{-5}, t = 80K$.

Figure 8: Simplifying the isosurface from a distance volume, where the distance value 0.001 is used.



(a) Rectangular ROI. (b) Spherical ROI.

Figure 9: Isosurface visualization using ROI.

be clustered together in the cell merging process. Therefore, the simplification capability of their algorithm is restricted.

In Figure 10, we show the results of simplifying a 2D Chinese character based on a quadtree data structure. This character is sampled in $256^2$ resolution. Several variations of the vertex clustering algorithm have been applied: the octree based algorithm where topology is ignored [11], the topology-preserving algorithm of Ju et al. [7], and ours. Some differences are highlighted with circles. In the non-topology-preserving result of Figure 10b, several regions have become disjointed. In the conservative result of Figure 10c, many tiny cells are kept since they violate the simplification rules defined by Ju et al. [7]. In contrast, we are able to handle these cases. Table 3 gives the comparison of cell and vertex counts generated in Figure 10. Under the same error threshold, we generate fewer cells and vertices than the other topology-preserving algorithm. Tiny cells are merged so that a more balanced simplification result is achieved. Our method also generates fewer cells and vertices than the non-topology-preserving algorithm. It shows that fewer errors are accumulated on the vertices. Furthermore, the contouring cost is reduced by the facts: (1) fewer cells and vertices are visited, and (2) fewer lines are generated. The effectiveness of our algorithm is more significant in 3D cases.

Table 3: Comparison of the simplification results of a Chinese character, where $\delta^2 = 0.003$ (EC: our enhanced cell method; VC: non-topology-preserving vertex clustering; JA: Ju et al.'s algorithm).

| Algorithm | # cells | # vertices | # lines |
|---|---|---|---|
| EC | 279(100%) | 300(100%) | 300(100%) |
| VC | 327(117%) | 327(109%) | 327(109%) |
| JA | 470(168%) | 470(157%) | 470(157%) |

It is not easy to give a direct comparison between our algorithm and the octree-based MC decimation algorithm reported by Shekhar et al. [19], since ours is basically a dual method to the MC algorithm. However, because of the encoding power, we could expect that ours will generate better results. Furthermore, ours has the advantage of no crack-patching between different octree levels.

## 8 Conclusions and Future Work

We have presented a topology-preserving isosurface simplification algorithm using a novel concept of preserving disconnected surface components during simplification. The problem is transformed into representing isosurface components and topology-preserving isosurface clustering. The enhanced cell representation is used to encode isosurface components in cells. A connectivity-guided vertex clustering algorithm is used to simplify the isosurface components. After building a hierarchically clustered vertex octree, topology-preserved isosurfaces can be extracted under various error bounds by the enhanced dual contouring algorithm.

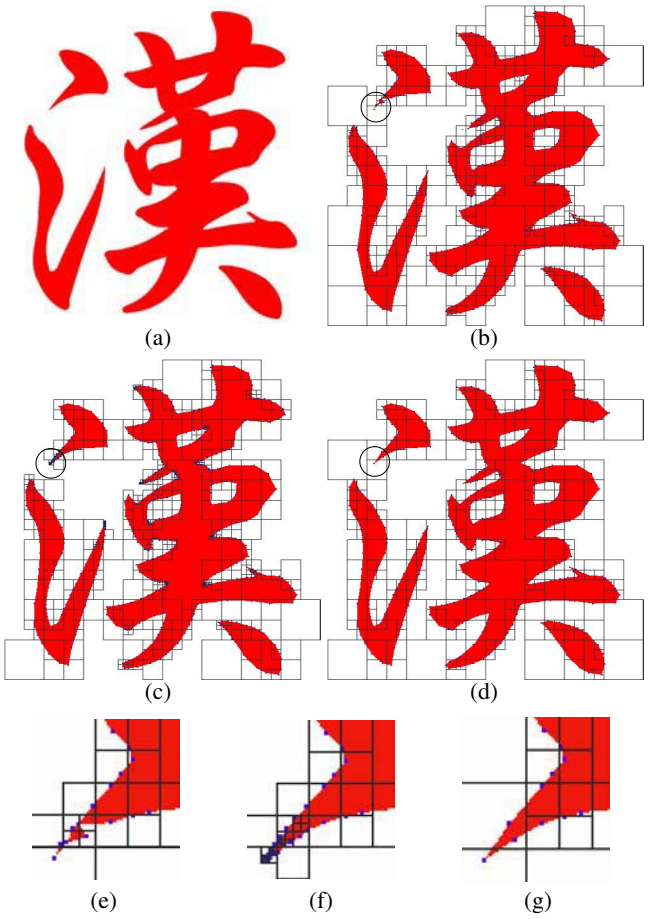



Figure 10: Simplifying a Chinese character in 2D. (a) The original model. (b) Simplification result using the non-topology-preserving vertex clustering. (c) Simplification result using Ju et al.'s algorithm. (d) Simplification result using our algorithm. (e), (f), and (g) show zoom-in view of the circled regions of (b), (c), and (d), respectively.

Our algorithm is efficient in simplifying isosurfaces of a fixed isovalue. Although it is not a general isosurface extraction algorithm with topology preservation, such as Gerstner and Pajarola's algorithm [4], the efficiency is attractive to those who are interested in fixed isovalues and concerned more about real-time visualization. In these cases, offline simplification algorithms, such as QSlim, are not suitable. Furthermore, the efficiency of our algorithm makes it possible to be combined with fast isosurface extraction algorithms to extract multiresolution isosurfaces with topology-preservation for different isovalues.

We have some on-going research work. We are studying powerful encoding schemes for isosurfaces. We are also interested in applying our approach in view-dependent isosurface rendering. Our pre-simplified vertex octree allows vertex refinement and collapsing in a topology-preserving manner. The screen-space errors can be considered in vertex selection. Finally, we plan to apply our new representation in point cloud reconstruction by better handling surface topology in cells.

## Acknowledgments

This work is supported by NSF grant CCR0306438 and ONR grant N000140110034. We would like to thank the anonymous reviewers for their help in improving this paper.

## References

[1] J. Bloomenthal and K. Ferguson. Polygonization of non-manifold implicit surfaces. In *SIGGRAPH Proceedings*, pages 55–62, July 1995.

[2] M. Garland. *Quadric-Based Polygonal Surface Simplification*. PhD thesis, Carnegie Mellon University, May 1999.

[3] M. Garland and P. S. Heckbert. Surface simplification using quadric error metrics. In *SIGGRAPH Proceedings*, pages 209–216, August 1997.

[4] T. Gerstner and R. Pajarola. Topology preserving and controlled topology simplifying multiresolution isosurface extraction. In *IEEE Visualization*, pages 259–266, October 2000.

[5] S. Gibson. Using distance maps for smooth surface representation in sampled volumes. In *IEEE Visualization*, pages 23–30, October 1998.

[6] B. Gregorski, M. Duchaineau, P. Lindstrom, and V. Pascucci. Interactive view-dependent rendering of large isosurfaces. In *IEEE Visualization*, pages 475–482, October 2002.

[7] T. Ju, F. Losasso, S. Schaefer, and J. Warren. Dual contouring of Hermite data. In *SIGGRAPH Proceedings*, pages 339–346, July 2002.

[8] L. P. Kobbelt, M. Botsch, U. Schwanecke, and H. Seidel. Feature-sensitive surface extraction from volume data. In *SIGGRAPH Proceedings*, pages 57–66, August 2001.

[9] P. Lindstrom. Out-of-core simplification of large polygonal models. In *SIGGRAPH Proceedings*, pages 259–262, July 2000.

[10] W. E. Lorensen and H. E. Cline. Marching Cubes: A high resolution 3D surface construction algorithm. In *SIGGRAPH Proceedings*, pages 163–169, July 1987.

[11] D. Luebke and C. Erikson. View-dependent simplification of arbitrary polygonal environments. In *SIGGRAPH Proceedings*, pages 199–208, August 1997.

[12] C. Montani, R. Scateni, and R. Scopigno. Discretized marching cubes. In *IEEE Visualization*, pages 281–287, October 1994.

[13] G. M. Nielson. On marching cubes. *IEEE Transactions on Visualization and Computer Graphics*, 9(3):283–297, 2003.

[14] R. N. Perry and S. F. Frisken. Kizamu: A system for sculpting digital characters. In *SIGGRAPH Proceedings*, pages 47–56, August 2001.

[15] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in C++*. Cambridge University Press, Cambridge, UK, 2001.

[16] J. Rossignac and P. Borrell. Multi-resolution 3D approximation for rendering complex scenes. In *Modeling in Computer Graphics*, pages 455–465. Springer-Verlag, Berlin, 1993.

[17] S. Schaefer and J. Warren. Dual contouring: The secret sauce. Technical Report 02-408, Department of Computer Science, Rice University, 2002.

[18] E. Shaffer and M. Garland. Efficient adaptive simplification of massive meshes. In *IEEE Visualization*, pages 127–134, October 2001.

[19] R. Shekhar, E. Fayyad, R. Yagel, and J. Cornhill. Octree-based decimation of marching cubes surfaces. In *IEEE Visualization*, pages 335–342, October 1996.

[20] B. T. Stander and J. C. Hart. Guaranteeing the topology of an implicit surface polygonization for interactive modeling. In *SIGGRAPH Proceedings*, pages 279–286, August 1997.

[21] A. Szymczak and J. Vanderhyde. Extraction of topologically simple isosurfaces from volume datasets. In *IEEE Visualization*, pages 67–74, October 2003.

[22] G. Varadhan, S. Krishnan, Y. J. Kim, and D. Manocha. Feature-sensitive subdivision and isosurface reconstruction. In *IEEE Visualization*, pages 99–106, October 2003.

[23] R. Westermann, L. Kobbelt, and T. Ertl. Real-time exploration of regular volume data by adaptive reconstruction of isosurfaces. *The Visual Computer*, 15(2):100–111, 1999.

[24] Z. Wood, H. Hoppe, M. Desbrun, and P. Schröder. Isosurface topology simplification. Technical Report MSR-TR-2002-28, Microsoft Research, 2002.