Dual Contouring of Hermite Data

Tao Ju, Frank Losasso, Scott Schaefer, Joe Warren
Rice University*

Figure 1: A temple undergoing destructive modi£cations. Both models were generated by dual contouring a signed octree whose edges contain Hermite data. The modi£ed model on the right was computed from the lefthand model in real-time.

Abstract

This paper describes a new method for contouring a signed grid whose edges are tagged by Hermite data (i.e; exact intersection points and normals). This method avoids the need to explicitly identify and process "features" as required in previous Hermite contouring methods. Using a new, numerically stable representation for quadratic error functions, we develop an octree-based method for simplifying contours produced by this method. We next extend our contouring method to these simpli£ed octrees. This new method imposes no constraints on the octree (such as being a restricted octree) and requires no "crack patching". We conclude with a simple test for preserving the topology of the contour during simpli£cation.

CR Categories: I.3.5 [Computation Geometry and Object Modeling]: CSG—Curve, surface, solid and object representations

Keywords: implicit functions, contouring, crack prevention, quadratic error functions, polyhedral simpli£cation

*e-mail: {jutao,losasso,sschaefe,jwarren}@rice.edu

Copyright © 2002 by the Association for Computing Machinery, Inc.
Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Permissions Dept, ACM Inc., fax +1 (212-869-0481 or e-mail permissions@acm.org.
© 2002 ACM 1-58113-521-1/02/0007 $5.00

1 Introduction

In the spring of 2001, 17 students in an advanced computer graphics class set out on a semester-long group project to develop a cutting-edge computer game. One of the primary goals for the game was to incorporate technology that allowed real-time modi£cation of the game geometry. (In gaming terminology, such geometry is referred to as "destructible".) The geometric engine for the resulting game was based on implicit modeling with the environment modeled as the zero contour of a 3D grid of scalar values. Our choice of this representation was guided by the fact that CSG operations are particularly simple to implement for implicit models. Although this game was a relative success, we noted that our implicit approach to modeling had several disadvantages. During a post-project review, we identi£ed several problems:

- Due to the use of a uniform grid, we were restricted to relatively small grid sizes. In particular, the £nal game could only process environments de£ned using grids of size 64^3 due to the requirement that the game run in real-time.
- The resulting environment lacked the sharp edges found in most polyhedral models. Although we could simulate a small class of shapes such as rooms and hallways by cleverly manipulating the sign £eld, the resulting environment was geometrically simple in comparison to those modeled using BSP trees.
- The polyhedral meshes produced by contouring often contained large ¤at regions tiled by numerous small polygons. The tiling of these ¤at regions trivially in¤ated the number of polygons in our model and often overwhelmed the graphics card used for the game.

In preparation for the next version of the gaming class, the instructor and three members of the class (the authors) decided to pursue a yearlong project to rewrite the game engine to address these de£ciencies. In particular, we focused on adapting three pieces of recently developed modeling technology for our program. Each of these pieces addresses one of the problems:

- First, we use an octree in place of a 3D uniform grid. In particular, our octree is inspired by those used in Adaptive Distance Fields [Frisken et al. 2000; Perry and Frisken 2001] in which signs are maintained at corners of cubes in the octree.

- At the leaves of the octree, we tag those edges with sign changes by exact intersection points and their normals from the contour. This choice is inspired by the Extended Marching Cubes method of [Kobbelt et al. 2001]. Adding normals allows this method to exactly reproduce a wide class of polyhedral shapes as well as curve or sharp edges on the contour.

- Third, we use these normals to de£ne a quadratic error function (QEF) for each leaf of the octree. These QEFs are then used in an octree-based polyhedral simpli£cation method similar to that of [Lindstrom 2000]. Our method uses the added information speci£ed by the signs attached to the corners of cubes in the octree to preserve the topology of this contour during simpli£cation.

The resulting representation is an octree whose leaf cubes have signs at their corners with exact intersections and normals tagging edges that exhibit sign changes. (See the upper left portion of £gure 2 for an example). Interior nodes in the octree contain QEFs used during simpli£cation. This representation can accurately approximate implicit shapes as well as parametric shapes such as subdivision surfaces. (These parametric shapes are imported as polygonal approximations and scan converted into a signed octree.) The adaptive structure of the octree allows for real-time approximate CSG operations and simpli£cation of the resulting shapes.

Given that we are building on several pieces of previous work, we should make clear our original contributions in this paper. First, we propose a new method for contouring a 3D grid of Hermite data that avoids the need to explicitly identify and process "features" as done in the Extended Marching Cubes method. After extending this contouring method to the case of multiple materials, we demonstrate how to model textured contours. We also introduce a new, numerically stable representation for quadratic error functions that we use in a standard octree-based method for simplifying these contours and their textured regions. We then develop a version of our contouring method for simpli£ed octrees that imposes no constraints on the octree (such as being a restricted octree) and requires no "crack patching". We conclude with a simple new test for preserving the topology of both the contour and its textured regions during simpli£cation.

2 Dual contouring on uniform grids

Although our ultimate goal is to develop a simple contouring method that is suitable for octrees, we £rst consider various methods for contouring signed uniform grids. The upper left portion of £gure 2 shows a typical example of a signed uniform grid. Those edges of the grid that exhibit a sign change are tagged by Hermite data consisting of exact intersection points and normals from the contour. This Hermite data can be computed directly from the implicit de£nition of the contour or by scan converting a closed polygonal mesh.

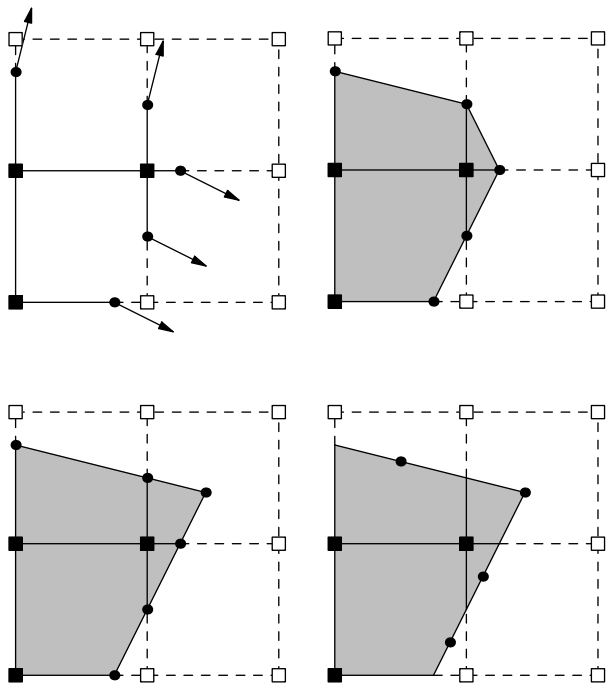

Figure 2: A signed grid with edges tagged by Hermite data (upper left), its Marching Cubes contour (upper right), its Extended Marching Cubes contour (lower left), and its dual contour (lower right).

2.1 Previous contouring methods

Cube-based methods such as the Marching Cubes (MC) algorithm and its variants generate one or more polygons for each cube in the grid that intersects the contour. Typically, these methods generate one polygon for each portion of the contour that interest a particular cube with the vertices of these polygons being positioned at the intersection of the contour with the edges of the cube. The upper right portion of £gure 2 shows a 2D example of the MC contour generated from the signed grid to its left. The left-hand side of £gure 3 shows a 3D example of a sphere generated as the zero contour of the function $f[x,y,z] = 1-x^2-y^2-z^2$. This contour consists of a collection of polygons that approximate the restriction of the contour to individual cubes in the grid.

Dual methods such as the *SurfaceNets* algorithm of [Gibson 1998] generate one vertex lying on or near the contour for each cube that intersects the contour. For each edge in the grid that exhibits a sign change, the vertices associated with the four cubes that contain the edge are joined to form a quad. The result is a continuous polygonal surface that approximates the contour. The right-hand side of £gure 3 shows an example of the same sphere contoured using the SurfaceNets method. Note that the polygonal mesh produced by the SurfaceNets method is dual to the mesh produced by MC in the standard topological sense: vertices of the SurfaceNets mesh correspond to faces of the MC mesh and vice versa. Dual methods typically deliver polygonal meshes with better aspect ratios since the vertices of the mesh are free to move inside the cube as opposed to being restricted to edges of the grid as in cube-based methods. [1]

[1]Note that other methods such as [Wood et al. 2000] contour without respect to the underlying £ne grid. We focus our attention on grid-based

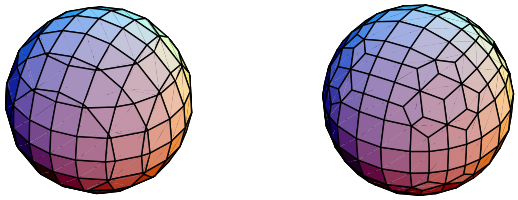

Figure 3: A sphere contoured using the Marching Cubes method (left) and the SurfaceNets method (right).

The *Extended Marching Cubes* (EMC) method is a hybrid between a cube-based method and a dual method. The EMC method detects the presence of sharp "features" inside a cube by examining normals associated with the intersection points on the edges of the cube. Those cubes whose normals lie inside a user-speci£ed cone are deemed to be featureless. In this case, the EMC method generates a polygon(s) using standard MC. For those cubes that do contain a feature, the method generates a vertex positioned at the minimizer of the quadratic function

$$E[x] = \sum_i (n_i \cdot (x - p_i))^2 \tag{1}$$

where the pairs p_i, n_i correspond to the intersections (and unit normals) of the contour with the edges of the cube. Once this vertex has been positioned, the method generates a triangle fan to the edges on the boundary of the cube. Finally, if two adjacent cubes both contain feature vertices, then the pair of triangles generated by the fan to their common face has its common edge ¤ipped to form a feature edge. The lower left portion of £gure 2 shows a 2D example of the contour generated by EMC.

2.2 Dual contouring of Hermite data

The main advantage of the EMC method is that it uses Hermite data and QEFs in positioning the vertices associated with cubes that contain features. This Hermite approach can generate contours that contain both sharp vertices and sharp edges. One drawback of this method is the need to explicitly test for such features and to then perform some type of special processing in these cases. As an alternative to the EMC method, we propose the following dual contouring method for Hermite data:

1. For each cube that exhibits a sign change, generate a vertex positioned at the minimizer of the quadratic function of equation 1.

2. For each edge that exhibits a sign change, generate a quad connecting the minimizing vertices of the four cubes containing the edge.

This method is an interesting hybrid of the EMC method and the SurfaceNets method. It uses the EMC method's feature vertex rule for positioning all vertices of the contour while using the *SurfaceNets* method to determine the connectivity of these vertices. (Note that the SurfaceNets method uses a completely different rule for positioning vertices on the contour.) By using QEFs to position all of the vertices of the contour, this method avoids the need to explicit test for features. Vertices on the contour are simply positioned to be consistent with the normals associated with the data. The lower right portion of £gure 2 shows a 2D example of the dual contour generated by the Hermite data in the upper left portion of the £gure.

methods like the ones above since this grid structure is the basis of our fast CSG operations.

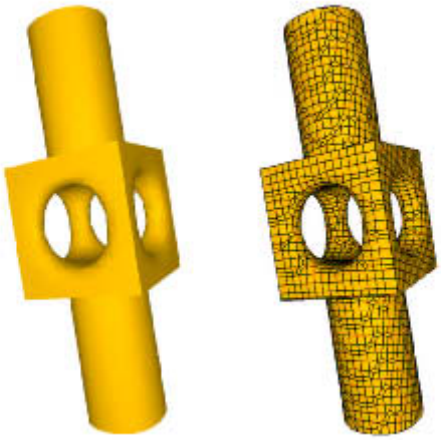

Figure 4: A mechanical part generated by dual contouring Hermite data on a 64^3 grid.

Figure 4 shows a 3D example of a mechanical part modeled by dual contouring Hermite data on a 64^3 grid. The left image shows a smooth shaded version of the part while the right image shows the polygonal mesh produced by dual contouring. The intersection points and normals for the model were generated from a closed subdivision surface. A sign £eld denoting the inside/outside of the model was computed using a standard scan conversion algorithm as described in [Foley et al. 1995].

2.3 Representing and minimizing QEFs

At this point, we should make a few comments concerning how we represent and minimize quadratic error functions. The function $E[x]$ of equation 1 is constructed from a collection of intersection points p_i and normals n_i. This function $E[x]$ can be expressed as the inner product $(Ax-b)^T(Ax-b)$ where A is a matrix whose rows are the normals n_i and b is a vector whose entries are $n_i \cdot p_i$. Typically, the quadratic function $E[x]$ is expanded into the form

$$E[x] = x^T A^T A x - 2x^T A^T b + b^T b \tag{2}$$

where the matrix $A^T A$ is a symmetric 3×3 matrix, $A^T b$ is a column vector of length three and $b^T b$ is a scalar. The advantage of this expansion is that only the matrices $A^T A$, $A^T b$ and $b^T b$ need be stored (10 ¤oats), as opposed to storing the matrices A and b. Furthermore, a minimizing value $\hat{x}$ for $E[x]$ can be computed by solving the normal equations $A^T A\hat{x} = A^T b$.

One drawback of this representation is that it is numerically unstable. For example, consider computing the value of $E[x]$ in ¤oating point arithmetic when the intersection points and normals used in constructing $E[x]$ are sampled from a ¤at area. For a grid of size 256^3 (as in £gure 1), the magnitude of $b^T b$ can be on the order of 10^6. Since ¤oats are only accurate to six decimal digits, if $E[x]$ is evaluated at points on the original ¤at area (where $E[x]$ should be zero), the resulting value has an error on the order of 1.

One possible solution to this problem is to use double precision numbers instead of ¤oats in representing $A^T A$, $A^T b$ and $b^T b$. Us-

ing doubles, the value of $E[x]$ in our ¤at example now has an error of 10^{-6}. Of course, the main drawback of using doubles in place of ¤oats is that the space require to store a QEF is doubled. For our application, we found this solution to be problematic since our program tended to be space bound as opposed to being time bound. (See the last section for details.)

An alternative representation for QEFs that delivers the accuracy of doubles while using only ¤oats is based on the *QR decomposition* [Golub and Van Loan 1989]. If $(A\ b)$ is the matrix formed by appending the column vector b to the matrix A, the idea behind this decomposition is to commpute an orthogonal matrix Q whose product with $(A\ b)$ is an upper triangular matrix of the form

$$\begin{pmatrix} x & x & x & x \\ 0 & x & x & x \\ 0 & 0 & x & x \\ 0 & 0 & 0 & x \\ 0 & 0 & 0 & 0 \\ \dots & \dots & \dots & \dots \end{pmatrix} = \begin{pmatrix} \hat{A} & \hat{b} \\ 0 & r \\ 0 & 0 \\ \dots & \dots \end{pmatrix}. \qquad (3)$$

Here, $\hat{A}$ is an upper triangular 3×3 matrix, $\hat{b}$ is a column vector of length 3 and r is a scalar. This matrix Q can be expressed as the product of a sequence of Givens rotations where each rotation zeroes a single entry in the lower part of $(A\ b)$.

Since any orthogonal matrix Q satis£es the relation $Q^T Q = I$, $E[x]$ can be rewritten as

$$\begin{aligned} (Ax-b)^T(Ax-b) &= (Ax-b)^T Q^T Q(Ax-b) \\ &= (QAx-Qb)^T(QAx-Qb) \\ &= (\hat{A}x-\hat{b})^T(\hat{A}x-\hat{b})+r^2 \end{aligned}$$

To evaluate $E[x]$ in this form, we compute the product of the vector $\hat{A}x-\hat{b}$ with itself and then add r^2. Returning to our previous ¤at example, we note that b has entries on the order of 10^3 and therefore $\hat{A}x-\hat{b}$ has entries that are on the order of 10^{-3} when x is chosen from the ¤at regions. Therefore, the computed value of $E[x]$ will be on the order of 10^{-6}.

If $\hat{A}$ is non-singular, the minimizing $\hat{x}$ can be computed by solving $\hat{A}\hat{x} = \hat{b}$ using back substitution. However, during dual contouring, $\hat{A}$ is often computed from noisy normals that are nearly coplanar. In this case, the matrix $\hat{A}$ is nearly singular. As a result, the minimizing $\hat{x}$ may lie far outside the de£ning cube. To solve this problem, we compute the SVD decomposition of $\hat{A}$ and form its pseudo-inverse by truncating its small singular values as done in [Kobbelt et al. 2001; Lindstrom 2000]. Based on experimentation, we typically truncate those singular values with a magnitude of less than 0.1. Using the resulting pseudo-inverse, we then approximately solve $\hat{A}\hat{x} = \hat{b}$ while minimizing the distance of $\hat{x}$ to the centroid of the intersection points p_i.

2.4 Modeling textured contours

In £gure 3, both contouring algorithms produced surfaces that bounded the transition from negative (empty) space to positive (solid) space. In a realistic environment, solids are not composed of a single homogeneous material. In practice, solids are composed of a collection of materials; each of which induces a region with a distinct texture on the contour. Figure 5 shows an example of a cube consisting of two materials, a gold material formed by extruding a Chinese character through the cube and a red material forming the remaining portion of the cube. Note that the gold material is a true solid (and not a surface texture) since the gold character extends all the way through the cube (as evidenced by the cube after a spherical cut on the right).

This partition of solids into distinct materials can be modeled implicitly by replacing the signs $-$ and $+$ (corresponding to empty and solid space) by a material *index*. In this representation, each grid point has an index corresponding to a distinct material.(See [Bloomenthal and Ferguson 1995; Bonnell et al. 2000] for examples of similar approaches.) Figure 6 shows a 2D grid with three distinct indices; the gray and black grid points denote distinct solid materials while white grid points denote empty space. As before, edges that exhibit index changes are also tagged by exact intersection points and normals. During contouring, this Hermite data is used in equation 1 to de£ne a QEF associated with each cube in the grid. Next, for each edge that exhibits a index change, dual contouring generates a quad connecting the minimizers of the QEFs for the four cubes containing the edge. The left portion of igure 6 shows the dual contour that separates the three materials.

If the viewer is restrict to empty space, we can optimize this contouring method for solids that consist several different materials. In particular, quads generated by solid/solid edges are not visible from empty space. The remaining quads correspond to solid/empty edges and can be textured using the material properties of the solid endpoint of the edge [2]. In £gure 5, the underlying grid contains three material; empty space, gold material, and red material. The resulting dual contour consists of red quads generated by red/empty edges and gold quads generated by gold/empty edges. Red/gold edges do not generate quads.

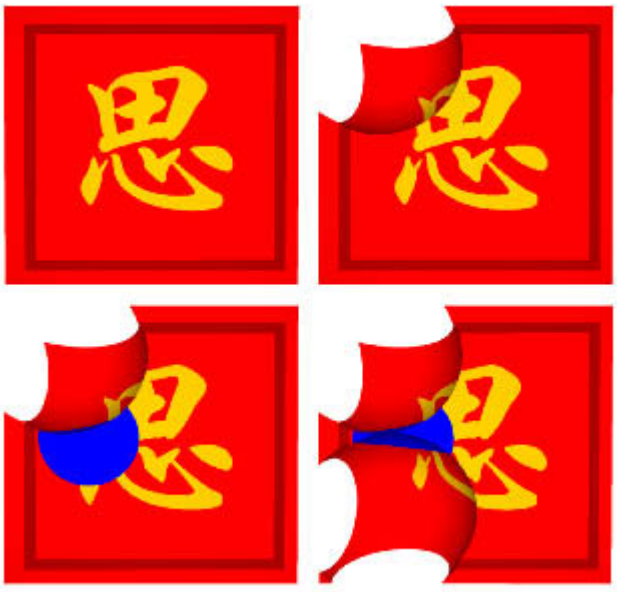

Figure 5: A solid cube undergoing a sequence of CSG operations.

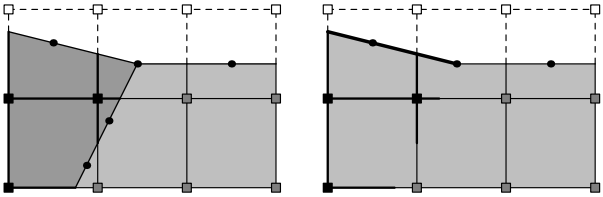

Figure 6: The dual contour for a three-index grid (left), treating the two solid (dark) indices as single index(right)

[2] Each material has an associated "black box" de£ned during the material's addition to the model that converts 3D geometric coordinates into 2D texture coordinates.

When three or more materials meet inside a single cube, dual contouring places the minimizing vertex at or near their intersection point. This positioning allows the outlines of letters and characters embossed on a surface to be reproduced very accurately. (The right-hand portion of £gure 12 shows a close-up example of this effect.) Cube-based contouring methods constrain the vertices of the contour to lie on the edges of the 3D grid making this effect dif£cult to achieve.

The move to the multi-material case allows for several interesting variations on the CSG operations used in the two-material case. In place of the standard CSG operations, we use a single operation *Add* that overwrites a portion of the existing model with a new material. Subtractive operations such as the spherical cut in the upper right of £gure 5 can be represented as adding a sphere of empty space to the model. Another useful variant of *Add* is the operation *Replace* that overwrites only the solid portion of the model. This operation can also be used to simulate texturing a portion of the contour. For example, the lower images in £gure 5 show a portion of the solid replaced by a blue material and then subsequently cut by a sphere.

3 Adaptive dual contouring

The previous algorithm for dual contouring has the obvious disadvantage of being formulated for uniform grids. In practice, most of a uniform grid is devoted to storing homogeneous cubes (i.e; cubes whose vertices all have the same sign). Only a small fraction of the cubes are heterogeneous and thus, intersect the contour. One way to avoid this waste of space is to replace the uniform grid by an octree. In this section, we describe an adaptive version dual contouring based on simplifying an octree whose leaves contain QEFs. This method is essentially an adaptive variant of a uniform simpli£cation method due to [Lindstrom 2000]. Our method has three steps:

- Generate a signed octree whose homogeneous leaves are maximally collapsed.
- Construct a QEF for each heterogeneous leaf and simplify the octree using these QEFs.
- Recursively generate polygons for this simpli£ed octree.

Note that our method also differs from Lindstrom's method in that we generate polygons from the signed octree instead of collapsing polygons in an existing mesh.

Generating the signed octree in the £rst step is relatively straightforward. For implicit or polygonal models, this octree can be constructed recursively in a top-down manner by spatially partitioning the models. For signed data on a uniform grid, this octree can be generated in a bottom-up manner by recursively collapsing homogeneous regions. The next two subsections examine the second and third parts of this process in more detail. (Note that the adaptive method described here works for multi-material case without modi£cation.)

3.1 Octree simplification using QEFs

Our approach to the second step of the adaptive method is to construct a QEF associated with each heterogeneous leaf using equation 1. Note that the residual associated with the minimizer of this QEF estimates how well the minimizing vertex approximates the original geometry [Garland and Heckbert 1998]. Our approach to simplifying the resulting octree is to form QEFs at interior nodes of the octree by adding the QEFs associated with the leaves of the subtree rooted by the node. Those interior nodes whose QEFs have a residual less than a given tolerance are collapsed into leaves.

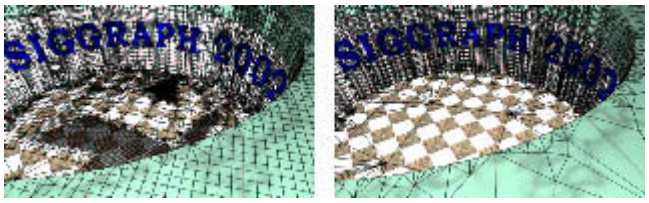

Figure 7: Closeups of two polygonal approximations to the temple computed using the standard (left) and QR (right) representation for QEFs.

The only modi£cation that we make to this method is to change our internal representation for QEFs to take advantage of the QR decomposition discussed in the previous section. To this end, we represent a QEF in terms of 10 ¤oats corresponding to the entries of the upper triangular matrix of equation 3. Adding two QEFs corresponds to merging the rows of their two upper triangular matrices to form a single 8×4 matrix of the form

$$\begin{pmatrix} x & x & x & x \\ 0 & x & x & x \\ 0 & 0 & x & x \\ 0 & 0 & 0 & x \\ x & x & x & x \\ 0 & x & x & x \\ 0 & 0 & x & x \\ 0 & 0 & 0 & x \end{pmatrix}$$

and then performing a sequence of Givens rotations to bring the matrix back into upper triangular form of equation 3. Due to the orthogonality of the Givens rotations, the QEF for the merged system is the sum of QEFs associated with the unmerged systems. Note that bringing the merged system back into upper triangular form is slower (around 150 arithmetic operations) than simply adding 10 ¤oats as done in the standard representation. However, the improved stability of the representation leads to better simpli£cations.

Figure 7 gives a concrete illustration of the advantage of the QR representation's stability. The meshes in this £gure show two simpli£cations of the temple from £gure 1 to an error of 0.014; the left mesh was computed using the standard representation for QEFs, the right mesh was computed using the QR representation for QEFs. (To give a sense of scale, the temple was de£ned over a 256^3 unit grid.) Due to the numerical error introduced by the instability of the standard representation, the mesh on the left contains $78K$ polygons while the mesh on the right has $36K$ polygons.

3.2 Polygon generation for simplified octrees

Given this simpli£ed octree, our next task is to modify the polygon generation phase of dual contouring appropriately. For cube-based methods, this problem of generating contours from octrees has been extensively studied [Bloomenthal 1988; Wilhelms and Gelder 1992; Livnat et al. 1996; Shekhar et al. 1996; Westermann et al. 1999; Frisken et al. 2000; Cignoni et al. 2000]. Typically, these methods restrict the octree to have neighboring leaves that differ by at most one level (i.e; "restricted" octrees) and usually perform some type of "crack repair" to ensure a closed contour. [Perry and Frisken 2001] describes a variant of the SurfaceNets algorithm for signed octrees based on enumerating the edges associated with leaves of the octree. In particular,

- For each edge that exhibits a sign change, generate all triangles that connect the vertices associated with any three distinct cubes containing the edge.

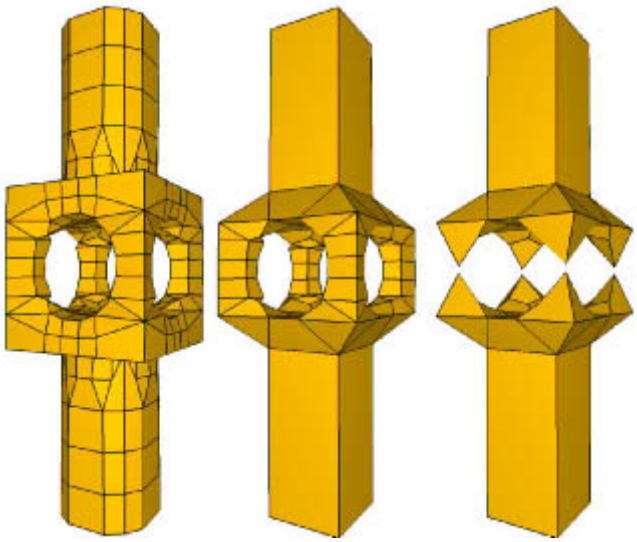

Figure 8: Three simpli£ed versions of the mechanical part.

To avoid generating redundant triangles, this method culls some of the generated triangles based on the relative positions of their corresponding edges inside a leaf cube. In the £nal mesh, each edge corresponds to either one or two triangles (a quad) on the resulting contour. The main disadvantage of this method (as acknowledged by the authors) is that it occasionally yields contours with cracks. The authors identify these problem con£gurations and avoid them by performing extra subdivision on the octree.

We propose a simpler rule for dual contouring signed octrees that avoids this need for extra subdivision. The rule is based on the observation that only those edges of leaf cubes that do not properly contain an edge of a neighboring leaf should generate a polygon. We refer to such edges as the *minimal* edges of the octree. Thus, our rule for polygon generation is

- For each minimal edge that exhibits a sign change, generate a polygon connecting the minimizing vertices of cubes that contain the edge.

This rule has the property that it always produces a closed polygonal mesh for any simpli£ed octree. In particular, every eedge in the mesh is contained by an even number of polygons. To prove this fact, we observe that edges in the dual contour are generated by pairs of face-adjacent leaf cubes. The minimal edges tiling the boundary of their common square face always exhibit an even number of sign changes since the boundary of the square is a closed curve. Therefore, the rule always generates an even number of polygons containing the edge. For example, the common square face always consists of four consecutive edges in the uniform case. This chain of four edges can exhibit either two or four sign changes and consequently generate two or four polygons containing the common edge. (It is possible to construct signed octrees that generate dual contours with 6 or more polygons share a common edge.) Figure 8 shows three simpli£ed approximations to the mechanical part. Note that the rightmost mesh has undergone a topology change.

This rule generates triangles instead of quads in transitional areas of the octree where a single coarse cube is face-adjacent to four £ne cubes. Minimal edges in the middle of the shared coarse face are contained by only three cubes and generate triangles that form a transition between coarse quads and £ne quads. Figure 7 shows many examples of such transition triangles produced by contouring a simpli£ed octree.

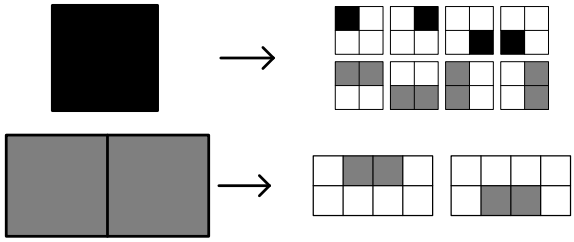

Figure 9: Recursive functions `faceProc` (black) and `edgeProc` (gray) used in enumerating pairs of leaf squares that contain a common edge.

Note that the Perry/Frisken rule enumerates edges in the octree and then locates those cubes that contain the edge. This neighbor £nding entails either walking up and down the octree or explicitly maintaining links between neighboring cubes. Instead of enumerating edges and trying to £nd neighbors, we propose a recursive method for enumerating those sets of cubes that contain a common minimal edge. For the sake of simplicity, we explain this enumeration method for quadtrees while noting that a similar method works for octrees.

The key to this enumeration procedure are two recursive functions `faceProc`$[q]$ and `edgeProc`$[q_1, q_2]$. Given an interior node q in the quadtree, `faceProc`$[q]$ recursively calls itself on the four children of q as well as calling `edgeProc` on all four pairs of edge-adjacent children of q. Given a pair of edge-adjacent interior nodes q_1 and q_2, `edgeProc`$[q_1, q_2]$ recursively calls itself on the two pairs of edge-adjacent children spanning the common edge between q_1 and q_2. Figure 9 depicts the mutually recursive structure of these two functions.

The recursive calls to `edgeProc`$[q_1, q_2]$ terminate when both q_1 and q_2 are leaves of the quadtree. At this point, the call to `edgeProc` has all of the information necessary to generate the segment associated with the minimal edge shared by q_1 and q_2. Note the running time of this method is linear in the size of the quadtree since there is one call to `faceProc` for each square in the quadtree and one call to `edgeProc` for each edge in the quadtree.

Contouring octrees requires three functions `cellProc`$[q]$, `faceProc`$[q_1, q_2]$ and `edgeProc`$[q_1, q_2, q_3, q_4]$. The function `cellProc` spawns eight calls to `cellProc`, twelve calls to `faceProc` and six calls to `edgeProc`. `faceProc` spawns four calls to `faceProc` and four calls to `edgeProc`. Finally, `edgeProc` spawns two calls to `edgeProc`. The recursive calls to `edgeProc`$[q_1, q_2, q_3, q_4]$ terminate at minimal edges of the octrees where all of the q_i's are leaves.

4 Simplification with topology safety

Simpli£cation methods such as [Rossignac and Borrell 1993; Lindstrom 2000] have the property that the topological connectivity of the polygonal mesh may change during simpli£cation. More sophisticated methods such as [Stander and Hart 1997; Gerstner and Pajarola 2000; Wood et al. 2000; Guskov and Wood 2001] were developed to maintain the connectivity of the mesh during simpli£cation. Unfortunately, in our setting, not only can the topological connectivity of the contour change during simpli£cation, but also the connectivity of its textured regions. The left side of £gure 12 shows an example of a simpli£cation in which distinct parts of the Chinese character have merged making it dif£cult to recognize. While these topological changes are not always undesirable, we wish to have the option of maintaining the topological connectivity of the contour and its textured regions during simpli£cation.

Speci£cally, given an interior node in the octree whose eight children are leaves, we desire a test based on the signs (or indices) at the corners of these leaves that guarantees that the topological connectivity of the dual contour and its textured regions is preserved during collapse of the node.

4.1 The two-signed case

Consider a coarse cube consisting of eight leaf cubes. The signs at the corners of the eight leaf cubes de£ne a $3 \times 3 \times 3$ grid whose corners de£ned a $2 \times 2 \times 2$ coarse grid. Our goal is to develop a test for determining whether the dual contour generated by this £ne grid is topologically equivalent to the dual contour generated by the coarse grid [3].

Before presenting the test, we recall that a d-dimensional contour is locally a *manifold* if it is topologically equivalent to a d-dimensional disc. Since a cube has twelve edges, dual contouring can generate up to twelve polygons that meet at the central vertex associated with the cube. For most common sign con£gurations on the cube, these polygons de£ne a manifold at this vertex. However, there exist sign con£gurations for which the dual contour is non-manifold. (These con£gurations correspond to the "ambiguous" sign con£gurations in standard cube-based methods.) Given this de£nition, the safety test has three checks:

1. Test whether the dual contour for the coarse cube is a manifold. If not, stop.
2. Test whether the dual contour for each individual £ne cube is a manifold. If not, stop.
3. Test whether the £ne contour is topologically equivalent to the coarse contour on each of the sub-faces of the coarse cube. If not, stop; otherwise safely collapse.

The £rst two checks restrict the simpli£cation process to manifold dual contours. (Note that the second check can be dropped if the £ne leaf cubes are themselves the results of a previous collapse.) In practice, this restriction is acceptable since most £ne resolution contours are manifold with non-manifold contours usually arising due to unsafe simpli£cation. For the £rst two checks, [Gerstner and Pajarola 2000] describe a simple test for determining whether the contour associated with a single cube is a manifold. The idea is to repeatedly collapse the edges of the cube whose corners have the same sign to a single vertex. Now, the contour associated with the cube is manifold if and only if the result of this reduction is a single edge. The result of this test can be pre-computed for all possible sign con£gurations associated with a single cube and stored in a table of size 2^8.

The third check tests topological equivalence of the coarse and £ne contours as follows: First, the method checks for topological equivalence on the edges of the coarse cube. Next, the method checks for topological equivalence on the faces of the coarse cube. Finally, the method checks for equivalence on the interior of the coarse cube. These checks can be implemented as a sequence of sign comparisons on the $3 \times 3 \times 3$ grid of signs.

- The sign in the middle of a coarse edge must agree with the sign of at least one of the edge's two endpoints.
- The sign in the middle of a coarse face must agree with the sign of at least one of the face's four corners.
- The sign in the middle of a coarse cube must agree with the sign of at least one of the cube's eight corners.

[3] Two shapes are *topologically equivalent* if they can be deformed into each other by a continuous, invertible mapping.

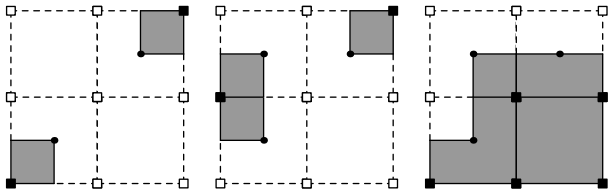

Figure 10: Three signed quadtrees and their dual contours.

Figure 10 shows three signed quadtrees that are candidates for simpli£cation. The dual contour for the left quadtree has two distinct connected components. In this case, the £rst check rejects the simpli£cation as unsafe since the contour for the collapsed quadtree is non-manifold. The dual contour for the middle quadtree also has two distinct components. In this case, the third check rejects the simpli£cation since the left edge of the quadtree cannot be safely simpli£ed. The signs for the rightmost quadtree satisfy all three checks and therefore the quadtree can be safely simpli£ed.

The proof of correctness for these sign checks is based on establishing topological equivalence for subfaces of the coarse cube in order of increasing dimension. The right mesh in £gure 8 shows an example of a simpli£ed version of the mechanical part that has undergone a topology change that disconnects the mesh. The middle mesh in £gure 8 shows an example of the part after safe simpli£cation with the topology checks preventing further unsafe simpli£cation.

4.2 The multi-material case

One nice feature of the contouring and simpli£cation methods discussed in the previous sections is that these methods handle the case of multiple materials without any extra dif£culty. Luckily, the safety test described in the previous subsection also generalizes to the contours of multi-material regions with one small change. An apparent dif£culty is that the contours of multi-material regions are inherently non-manifold in the two-material sense. For example, £gure 11 shows three examples of dual contours separating the three materials. Two of the contours have a vertex where three materials meet. Note that if we consider the boundary of each material's region separately, we can still classify whether this portion of the dual contour is a manifold. Speci£cally, a multi-material dual contour is a *quasi-manifold* if the boundary of each material's region is a manifold. In the two-material case, being a quasi-manifold is equivalent to being a manifold.

Now, the multi-material safety test determines whether it is topologically safe to simplify dual contours that are quasi-manifolds. As before, this restriction is not particularly problematic since most portions of a multi-material contour are quasi-manifold. This new test again consists of three phases and is identical to the two-material test with the exception that we replace the £rst and second checks for whether the contour inside a single cube is a manifold by an equivalent test for whether the contour is a quasi-manifold. The index tests in phase three remain unchanged.

In analogy with the manifold case, the quasi-manifold test for a multi-material cube involves collapsing each edge of the cube whose endpoints have the same index. Now, the dual contour associated with the cube is a quasi-manifold if and only if the collapsed edge graph is a simplex (i.e; a point, a segment, a triangle or a tetrahedron). As in the two-sign case, the values of this function can be pre-computed and stored in a lookup table of size 4^8. (If a cube has 5 or more distinct indices, its edge graph cannot collapse to a simplex.) The correctness of this test can be veri£ed by selecting an index on the cube and treating all of the remaining indices as being equivalent. Since the resulting edge graph collapses to a segment,

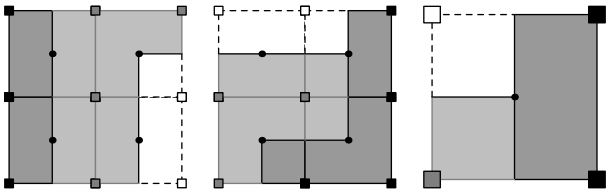

Figure 11: Three multi-material quadtrees and their dual contours.

Figure 12: The Chinese cube after unsafe simpli£cation (left) and safe simpli£cation (right). Note that distinct parts of the character have merged during unsafe simpli£cation.

the portion of the contour corresponding to the chosen index is a manifold.

Figure 11 shows two multi-material quadtrees that are candidates for simpli£cation. The left quadtree is rejected since the third check fails on the bottom edge of the quadtree. The middle quadtree passes all three checks and collapses to the quadtree on the right. Note that the contour for this collapsed quadtree is a quasi-manifold since the collapsed edge graph for this square is a triangle.

Figure 12 shows two simpli£ed versions of the Chinese cube from £gure 5. The left version has been simpli£ed without any type of topological safety. Note that the disjoint components of the Chinese character have fused together. The right version has been safely simpli£ed using the multi-sign test with separate regions of the character remaining distinct after simpli£cation.

5 Results

The current version of our geometric program runs on a consumer-grade PC equipped with a GeForce 3 video card. The program performs adaptive dual contouring on an indexed octree. The table below shows the number of quads generated by our method for various examples after simpli£cation to an error tolerance of 0.01. (All grids have unit spacing.) The time £eld represents the sum of the times to simplify the initial octree (with topological safety) and then generate polygons from the simpli£ed octree. The CSG operations (spheres of radius 6) in £gure 1 took approximately 30 milliseconds to compute.

model	# quads	time (MS)	space (MB)
part 64^3	2578	44	1.3
Chinese cube 128^3	1646	636	32
temple 256^3	39201	3586	156
david 512^3	143533	2948	91

Acknowledgements We would like to thank Danny Sorenson for his suggestion to use the QR decomposition to represent QEFs. We would also like to thank Marc Levoy and the Michelangelo Project for their gracious permission to use the scanned version of Michelangelo's David.

References

BLOOMENTHAL, J., AND FERGUSON, K. 1995. Polygonization of non-manifold implicit surfaces. In *Proceedings of SIGGRAPH 95*, ACM SIGGRAPH / Addison Wesley, Los Angeles, California, Computer Graphics Proceedings, Annual Conference Series, 309–316. ISBN 0-201-84776-0.

BLOOMENTHAL, J. 1988. Polygonization of implicit surfaces. *Computer Aided Geometric Design 5*, 4, 341–356.

BONNELL, K. S., SCHIKORE, D. R., JOY, K. I., DUCHAINEAU, M., AND HAMANN, B. 2000. Constructing material interfaces from data sets with volume-fraction information. In *IEEE Visualization 2000*, 367–372. ISBN 0-7803-6478-3.

CIGNONI, P., GANOVELLI, F., MONTANI, C., AND SCOPIGNO, R. 2000. Reconstruction of topologically correct and adaptive trilinear isosurfaces. 399–418.

FOLEY, J., VAN DAM, A., FEINER, S., AND HUGHES, J. 1995. *Computer Graphics: Principles and Practice.* Addison Wesley.

FRISKEN, S. F., PERRY, R. N., ROCKWOOD, A. P., AND JONES, T. R. 2000. Adaptively sampled distance £elds: A general representation of shape for computer graphics. In *Proceedings of SIGGRAPH 2000*, ACM Press / ACM SIGGRAPH / Addison Wesley Longman, Computer Graphics Proceedings, Annual Conference Series, 249–254.

GARLAND, M., AND HECKBERT, P. S. 1998. Simplifying surfaces with color and texture using quadric error metrics. In *IEEE Visualization '98*, IEEE, 263–270.

GERSTNER, T., AND PAJAROLA, R. 2000. Topology preserving and controlled topology simplifying multiresolution isosurface extraction. In *IEEE Visualization 2000*, 259–266.

GIBSON, S. F. F. 1998. Using distance maps for accurate surface reconstruction in sampled volumes. In *1998 Volume Visualization Symposium*, IEEE, 23–30.

GOLUB, G. A., AND VAN LOAN, C. F. 1989. *Matrix Computations*, second ed. The Johns Hopkins University Press.

GUSKOV, I., AND WOOD, Z. 2001. Topological noise removal. In *Graphics Interface 2001*, 19–26.

KOBBELT, L. P., BOTSCH, M., SCHWANECKE, U., AND SEIDEL, H.-P. 2001. Feature-sensitive surface extraction from volume data. In *Proceedings of SIGGRAPH 2001*, ACM Press / ACM SIGGRAPH, Computer Graphics Proceedings, Annual Conference Series, 57–66.

LINDSTROM, P. 2000. Out-of-core simpli£cation of large polygonal models. In *Proceedings of SIGGRAPH 2000*, ACM Press / ACM SIGGRAPH / Addison Wesley Longman, Computer Graphics Proceedings, Annual Conference Series, 259–262.

LIVNAT, Y., SHEN, H.-W., AND JOHNSON, C. R. 1996. A near optimal isosurface extraction algorithm using the span space. 73–84.

PERRY, R. N., AND FRISKEN, S. F. 2001. Kizamu: A system for sculpting digital characters. In *Proceedings of SIGGRAPH 2001*, ACM Press / ACM SIGGRAPH, Computer Graphics Proceedings, Annual Conference Series, 47–56.

ROSSIGNAC, J., AND BORRELL, P. 1993. Multi-resolution 3d approximation for rendering complex scenes. In *Modeling in Computer Graphics*, 455–465.

SHEKHAR, R., FAYYAD, E., YAGEL, R., AND CORNHILL, J. F. 1996. Octree-based decimation of marching cubes surfaces. In *IEEE Visualization '96*, IEEE, 335–344.

STANDER, B. T., AND HART, J. C. 1997. Guaranteeing the topology of an implicit surface polygonization for interactive modeling. In *Proceedings of SIGGRAPH 97*, ACM SIGGRAPH / Addison Wesley, Los Angeles, California, Computer Graphics Proceedings, Annual Conference Series, 279–286.

WESTERMANN, R., KOBBELT, L., AND ERTL, T. 1999. Real-time exploration of regular volume data by adaptive reconstruction of isosurfaces. 100–111.

WILHELMS, J., AND GELDER, A. V. 1992. Octrees for faster isosurface generation. 201–227.

WOOD, Z. J., DESBRUN, M., SCHRÖDER, P., AND BREEN, D. 2000. Semi-regular mesh extraction from volumes. In *IEEE Visualization 2000*, 275–282.