

# Robust Repair of Polygonal Models

Tao Ju\*  
Rice University

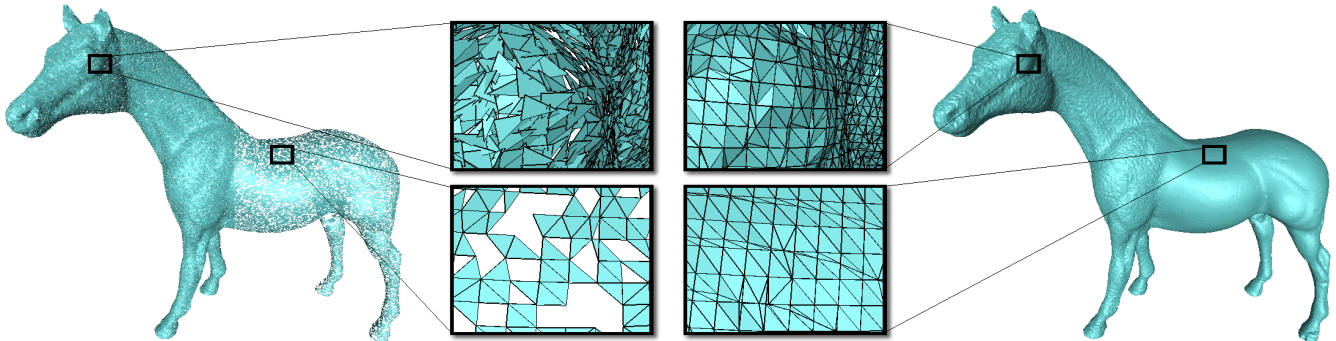


Figure 1: A synthetically distorted Horse model (left) containing numerous self-intersecting polygons, gaps and holes, and the repaired model (right) with a closed surface.

## Abstract

We present a robust method for repairing arbitrary polygon models. The method is guaranteed to produce a closed surface that partitions the space into disjoint internal and external volumes. Given any model represented as a polygon soup, we construct an inside/outside volume using an octree grid, and reconstruct the surface by contouring. Our novel algorithm can efficiently process large models containing millions of polygons and is capable of reproducing sharp features in the original geometry.

**CR Categories:** I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Boundary representations; Curve, surface, solid, and object representations; Geometric algorithms, languages, and systems

**Keywords:** model repair, robustness, scan conversion, octree

## 1 Introduction

Polygonal representations are widely used in computer systems and applications for modeling 3-dimensional geometry. Polygonal models can be created from various sources, such as 3D range scans and computer-aided design software. However, due to limitations of these creation methods, the resulting polygonal models often cannot be directly utilized by applications that require a closed model as input. By saying *closed*, we mean that the surface of the model partitions the entire space into disjoint internal and external volumes, so that each polygon lies between an internal volume and an external volume. A non-closed model often contains mesh defects such as gaps, holes, self-intersecting polygons, etc. Figure

2 compares two groups of curve models in 2D, one group being closed (bottom) and the other not (top).

We seek a robust method for repairing an arbitrary polygonal model, so that the repaired model is always closed. Due to the diversity and complexity of polygonal models, mesh repair faces daunting challenges. Specifically, an ideal repair method should possess the following properties:

1. **Robustness:** The method should always produce a closed surface for any input model.
2. **Efficiency:** The method should be able to process huge models within reasonable time and space.
3. **Accuracy:** The method should preserve the geometry of the input model whenever possible.

Unfortunately, no existing methods are known to the authors that satisfy all the desired properties. In particular, the robustness of the repair method is hard to guarantee, especially for huge input models with numerous mesh defects. In this paper, we present a satisfactory solution using a volumetric approach. Our method takes a polygon soup (e.g., the horse in figure 1 left) as input, constructs an intermediate volume grid, and generates the output surface (e.g., the horse in figure 1 right) by contouring the grid. The method is guaranteed to produce a closed and consistently oriented surface for any arbitrary input polygonal model. By using memory-less operations and divide-and-conquer techniques, input models with millions of triangles can be repaired on a consumer level PC in minutes. As an option, our method is also capable of reproducing sharp features on the input surface, which is particularly suitable for repairing CAD models.

### 1.1 Contributions

Although other volumetric techniques have been proposed for repairing polygon models [Nooruddin and Turk 2003] or filling holes on the model surface [Curless and Levoy 1996; Davis et al. 2002], our method differs substantially in the following aspects:

\*e-mail: jutao@rice.edu

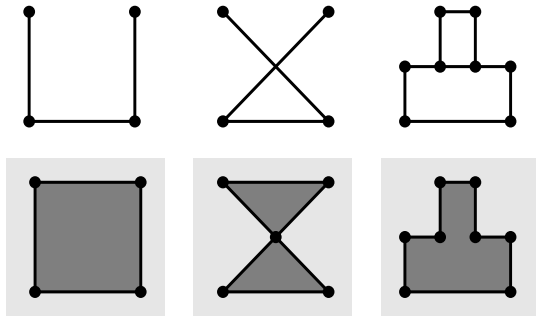


Figure 2: Three non-closed curves (top) and three closed curves (bottom). Vertices are represented by dots, internal volumes are colored dark gray and external volumes are colored light gray.

1. Unlike previous methods that rely on a uniform volume grid, we employ a space-efficient octree grid that allows the input model to be repaired and reconstructed at a higher resolution with less space consumption. Correspondingly, we present a fast, memory-less and numerically robust algorithm for scan-converting an arbitrary input model onto an octree grid.
2. The core of our method is a simple and robust algorithm for determining, at each point on the grid, a sign indicating whether the point lies inside or outside the input model. The signs are generated as a post-process on the scan-converted grid and requires no global iterations. Our algorithm can be efficiently implemented as recursive procedures on the octree.
3. Unlike previous volumetric approaches that generate blobby surfaces, by recording Hermite data, we are able to reproduce sharp features in the original model using Dual Contouring [Ju et al. 2002].

## 2 Related works

### 2.1 Mesh-based model repair

One class of model repair methods, which we refer to as *mesh-based* methods, operate directly on the polygons in the model to repair various geometric and topological errors. Turk and Levoy [Turk and Levoy 1994] introduce mesh zippering for removing overlapping regions on the mesh when merging multiple range scan images. Barequet and Kumar [Barequet and Kumar 1997] describe an interactive system that closes small cracks by stitching corresponding edges and fills big holes by triangulating the detected hole boundary. A different approach is proposed by Borodin et al. [Borodin et al. 2002] who describe gap closing as progressive boundary decimation. Recently, Liepa [Liepa 2003] applies mesh fairing after hole triangulation so that the patch interpolates the shape and density of the surrounding mesh.

Mesh-based methods are good at reproducing the original geometry, since only regions with mesh defects are detected and repaired. However, the output model is not guaranteed to be closed and may contain self-intersecting polygons. Moreover, detection of defective regions (such as holes) often requires global traversal of the entire model, which is both time-consuming and space-consuming.

A related method is proposed by Murali and Funkhouser [Murali and Funkhouser 1997], in which the input model is represented as a BSP tree and cells on the tree are classified as solid or non-solid. The output surface always encloses a solid space. Their method, however, involves expensive operations such as construction of a BSP tree from the input model and solving systems of linear equations whose size equals the number of input polygons.

### 2.2 Volumetric model repair

A volumetric approach to model repair involves representing the input model on a volumetric grid and reconstructing an output surface from the grid. Several approaches have been proposed for performing each task, which we shall review separately. (Note that a related problem considers surface reconstruction from scattered points via an intermediate volume, such as [Ohtake et al. 2003].)

#### 2.2.1 Volume construction

Converting a polygonal model into a volume representation is often referred to as *scan-conversion* or *voxelization*. To be able to reconstruct a closed surface, the key is to determine, at each grid point, a sign indicating whether it lies inside or outside the model. Hence we can classify scan-conversion techniques by the way they generate signs on the volume.

The first class of scan-conversion methods only detect cells on the volume that intersect with the model surface without generating signs, such as [Huang et al. 1998] and [Dachille and Kaufman 2000]. However, the volumetric representations generated by these methods are thin-shelled, thus do not possess inside and outside partitions.

The second class of methods generate a signed volume directly from the input model. Frisken et al. [Frisken et al. 2000] generate a signed distance at each grid point by computing the minimum signed distance from the grid point to each input polygon. However, the method fails when input model contains inconsistently oriented polygons (e.g., non-orientable surfaces).

In a different approach, Nooruddin and Turk [Nooruddin and Turk 2003] proposed two methods for determining the sign at a given grid point directly from input polygons: parity count and ray stabbing. Both methods involve casting rays from each grid point and voting based on the parity or locations of intersections on each ray with the model. However, since ray casting is a global operation, a small error on the surface may cause a large portion on the grid at a distant location to be misclassified. Moreover, sign generation relies on a uniform grid, which makes it difficult for representing the model at a high grid resolution.

The last class of scan-conversion methods generate inside/outside partitions on the volume as a post process. Andujar et al. [Andujar et al. 2002] first construct an octree grid and then determine the inside/outside property of each cell using a robust seed algorithm. Similarly, Oomes et al. [Oomes et al. 1997] first voxelize triangles onto a uniform grid and then fill the cells inside the model using a boundary filling algorithm [Foley et al. 1990]. These filling algorithms, however, fail if the model contains holes on the surface.

To be able to determine signs in the presence of holes, two methods were introduced for processing complex surfaces: the Space Carving method [Curless and Levoy 1996], and the Volumetric Diffusion method [Davis et al. 2002]. Space carving works on surfaces reconstructed from range scans and relies on line of sight information from the scanners to determine empty regions on the volume. Volumetric diffusion, on the other hand, utilizes pre-existing signed distances at each grid point and employs global iterations that converge to form a smooth and naturally looking surface. Both methods depend on extra information (i.e., line of sight or signed distances) that can not be easily or reliably obtained from an arbitrary polygonal model other than range scans. Moreover, diffusion is a global process, which traverses a much larger space than necessary to close a hole and can be slow to converge. As a final note, these two methods are specialized for filling holes, hence the reconstructed surface is not guaranteed to partition the space into disjoint internal and external volumes.

## 2.2.2 Surface reconstruction

Surface reconstruction from a signed volume is usually performed through *contouring*, which produces a polygonal approximation of the zero-value isosurface. Contouring algorithms can be classified into two types: primal methods (such as Marching Cubes [Lorensen and Cline 1987]) and hybrid methods (such as the Feature Sensitive Surface Extraction method [Kobbelt et al. 2001] and Dual Contouring method [Ju et al. 2002]). Primal contouring methods extract polygons by connecting points lying on the grid lines, and generate blobby surfaces with rounded corners. Hybrid methods allow polygon vertices to be placed inside grid cells to reproduce sharp edges and corners. The volumetric model repair methods that we have reviewed [Nooruddin and Turk 2003; Curless and Levoy 1996; Davis et al. 2002] use primal contouring methods for surface reconstruction, since the information stored on the volume is not adequate for reproducing sharp features from the original geometry.

## 3 Robust model repair pipeline

As illustrated in figure 3, our model repair process consists of three steps:

1. **Scan-conversion:** Embed the input model in a uniformly spaced grid, and mark edges on the grid that intersect the polygons as *intersection edges*. For efficiency, cells containing intersection edges are stored in an octree (figure 3 (b)).
2. **Sign generation:** At the grid points, generate signs that are *consistent* with the intersection edges, so that each cell edge intersecting the model should exhibit a sign change (figure 3 (c)).
3. **Surface reconstruction:** Reconstruct a closed surface on the signed grid by contouring. Dual contouring can be used to reproduce sharp features when Hermite data is stored on the intersection edges (figure 3 (d)).

Sign generation is the central step, which results in a partitioning of space that is essential to the construction of a closed surface. Our approach relies on the grid edges intersected with the input polygons, which can be robustly obtained for any type of model (even non-orientable surfaces). By adding or removing intersection edges at appropriate locations, we can always generate consistent signs at the grid points.

## 4 Scan-conversion

In the first step of model repair, we convert the input model to a volumetric form by constructing an octree grid that records edges intersecting the input model (i.e., intersection edges). The octree can be built incrementally as the polygons are read from the input model. Specifically, for each polygon to be processed, we recursively walk down the octree, expanding nodes when necessary, until all the leaf cells at the bottom level of the tree that intersect the polygon are located (figure 4 (a) to (c)). Then, cell edges that intersect the polygon are identified in those leaf cells and are marked as intersection edges (figure 4 (d)).

Although intersection edges suffice for the purpose of sign generation, extra information (e.g., exact intersection points and triangle normals) can be recorded on each intersection edge for better surface reconstruction (see Section 6). To avoid duplication, each leaf cell only maintains the extra information on its three primal edges (i.e., edges adjacent to the lower-left-far cell corner), and stores the intersection properties of all its edges in a 12-bit mask for fast querying.

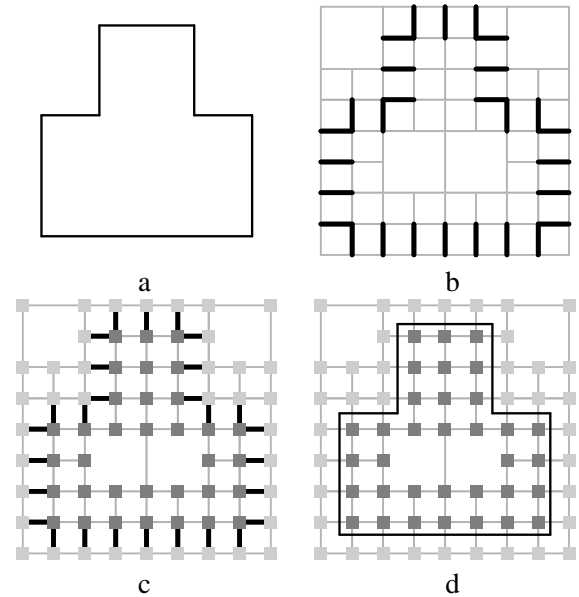


Figure 3: Model repair pipeline: the original model (a), an octree grid with intersection edges highlighted (b), signs at grid points generated from the intersection edges (c), and the final model reconstructed by contouring (d).

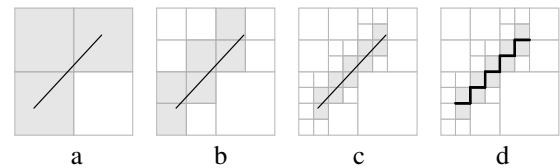


Figure 4: Recursive steps in scan-converting a polygon onto an octree.

Incremental update of the octree is a *memory-less* operation, in that no space is necessary for storing the input model. Polygons are processed one at a time, and are discarded after they are scan-converted. It is therefore particularly suitable for processing large polygon models, such as 3D range scan data. For example, the David model at 1mm resolution with over 56 million triangles can be scan-converted onto an octree grid at the same resolution (at depth 13) using less than 500 megabytes memory (consider that storing the model itself would take over 1 gigabyte).

### 4.1 Efficient and robust intersection tests

Assuming the input polygons are triangles, the main operations involved in the scan-conversion process are triangle-cube intersections and triangle-edge intersections. Both operations can be performed efficiently using the Separating Axes method. Specifically, a triangle and a cube are disjoint if their projections on any one of the following 13 vectors are disjoint: the triangle face normal, 3 cube face normals, and 9 pair-wise cross-products of the 3 edges of the cube and the 3 edges of the triangle. Treating a cell edge as a degenerate cube, triangle-edge intersection can be tested using the same set of projection vectors as those computed from the triangle and the cube containing the edge. Since the octree cells and their cell edges are aligned to the primal axes, the projection vectors can be easily computed once for each triangle and used for perform-

ing all intersection tests on the octree. Table 1 reports the resulting scan-conversion time for models of various sizes.

Intersection tests using Separating Axes involve floating point operations that are prone to numerical errors. These errors cause incorrect identification of cell edges that intersect the polygons, which may lead to erroneous signs generated in the next step. To eliminate floating-point errors, we use integer operations to obtain exact results. Specifically, we embed the octree grid and the input model inside a fine integer lattice of size  $2^N$  in each dimension, and clamp each grid point and polygon vertex to the nearest lattice point. We choose  $N = 20$  to achieve a comparable accuracy as single-precision floating-point coordinates. Since vertices of the triangle and the grid cells can be represented by  $N$ -bit integers, their differences are vectors representable by  $N$ -bit integers too. Consequently, the 13 projection vectors used in the intersection tests, which are cross-products of these difference vectors, require  $2N$ -bit integers. Finally, the projections of triangle and cube vertices on any of the 13 projection vectors can be represented by integers with  $2N + N = 3N$  bits. As a result, a 64-bit representation suffices for our integer computation.

## 5 Sign generation

Given a scan-converted grid, our next task is to determine signs at the grid points such that each intersection edge exhibits a sign change (i.e., a consistent sign configuration). However, consistent signs do not exist when a cell face contains an odd number of edges that intersect the model, as shown in figure 5. Such cases may arise either due to an insufficient grid resolution (figure 5 left) or a non-closed input model (figure 5 middle and right). Denote the set of original intersection edges on the scan-converted grid as  $E$ , our goal is to obtain a modified set  $\hat{E}$  of intersection edges that possess a consistent sign configuration.

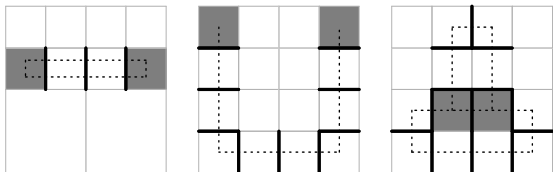


Figure 5: Cell faces (colored gray) containing an odd number of intersection edges (drawn in black). Input models are shown as broken lines.

### 5.1 Overview

The set  $E$  defines a *dual surface*  $S$ , which consists of quads that are dual to the edges in  $E$ . Specifically, each quad in  $S$  is perpendicular to an edge in  $E$  and centered at the midpoint of that edge. Figure 6 (a) shows an input model, and figure (b) shows the edges on a depth 4 octree grid intersecting the model and the corresponding dual surface. We introduce a boundary operator  $\partial$ , so that  $\partial(S)$  extracts the *boundary edges* (i.e., edges shared by an odd number of quads) on  $S$  (highlighted in figure 6 (b)). Note that each edge in  $\partial(S)$  is dual to a cell face on the primal grid that contains an odd number of intersection edges. In other words,  $\partial(S)$  is empty when all cell faces on the primal grid are well-formed: they contain an even number (0, 2 or 4) of intersection edges.

The key observation is that, a consistent sign configuration exists for  $E$  on the primal grid *if and only if*  $\partial(S)$  is empty (a brief proof is given in Appendix A). Therefore, we can proceed in the following three steps for sign generation:

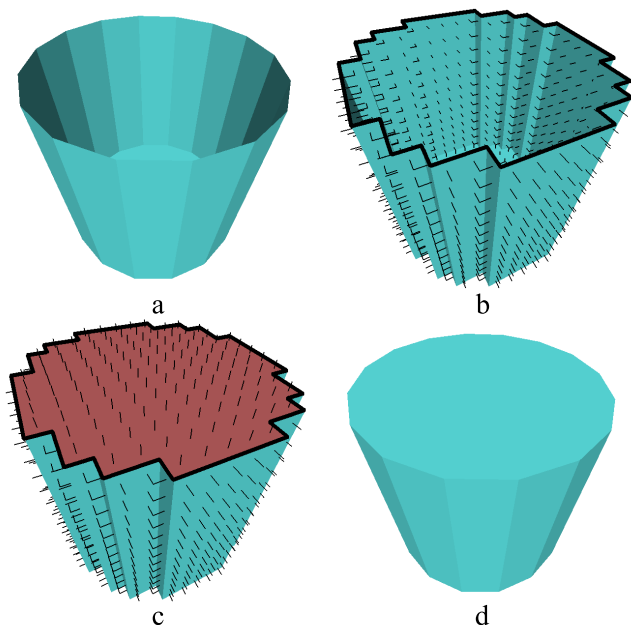


Figure 6: Sign generation via the dual surface: the input model with an open top (a), edges intersected with the model and the corresponding dual surface (b) (boundary edges are highlighted), the patched dual surface with corresponding intersection edges (c), and the repaired model based on the consistent signs generated (d).

1. **Detect boundary cycles.**  $\partial(S)$  is a collection of closed cycles  $b_i$ . In fact, since a vertex on  $S$  is shared by two edges on each quad that contains the vertex, every vertex on  $S$  is shared by an even number of boundary edges. Hence  $\partial(S)$  is an Eulerian graph and can be partitioned into disjoint cycles  $b_i$  [Bollobas 1979]. The dual surface in figure 6 (b) contains one boundary cycle at the top.
2. **Patch boundary cycles.** For each boundary cycle  $b_i$ , we construct a patch  $P_i$  so that  $\partial(P_i) = b_i$ . Let  $\ominus$  be the symmetric difference operator<sup>1</sup>, we note that  $\partial(S \ominus P_i) = \partial(S) \ominus \partial(P_i) = \partial(S) - b_i$ . Hence taking the symmetric differences between every  $P_i$  and  $S$  results in a patched surface  $\hat{S}$ , such that  $\partial(\hat{S}) = \emptyset$ . Figure 6 (c) shows the patched result of the dual surface in figure 6 (b).
3. **Generate signs.** The patched dual surface  $\hat{S}$  corresponds to a new set of intersection edges  $\hat{E}$  on the primal grid, where consistent signs can be generated. Figure 6 (d) shows the repaired model based on the intersection edges shown in 6 (c).

Next, we will demonstrate how each step can be efficiently implemented on the octree grid as recursive procedures.

### 5.2 Detect boundary cycles

Detecting the boundary edges in  $\partial(S)$  simply involves enumerating the cell faces on the primal grid containing an odd number of intersection edges. To form cycles, we introduce a bottom-up procedure **detectProc[N]** that returns all complete cycles  $B$  and incomplete cycles (i.e., paths of boundary edges)  $R$  inside the octree node  $\mathbf{N}$ .

<sup>1</sup>The symmetric difference between two sets  $X$  and  $Y$  is defined as  $X \ominus Y = X \cup Y - X \cap Y$ .

The key to the recursion is, a cycle or an incomplete cycle either lies inside a child node, or consists of incomplete cycles in multiple children nodes connected by edges crossing the faces between the nodes. At a leaf node, `detectProc[N]` returns  $B = \emptyset$  and  $R = \emptyset$ . At an internal node, `detectProc[N]` proceeds as follows:

1. Call `detectProc[Ni]` for every child node  $\mathbf{N}_i$ , which return cycles  $B_i$  and incomplete cycles  $R_i$ .
2. Detect the boundary edges  $E$  crossing the faces between adjacent children nodes.
3. Connect  $R_i$  by  $E$  to form complete cycles  $\bar{B}$  and incomplete cycles  $R$ .  $B$  is the union of  $\bar{B}$  and the  $B_i$ s.

### 5.3 Patch boundary cycles

Given a boundary cycle  $b$  on the dual surface, we need to construct a patch  $P$  so that  $\partial(P) = b$ . We would also like  $P$  to contain as few quads as possible, so that a minimum portion of the grid is affected by patching. The continuous version of the problem is known as the Plateau’s Problem, which seeks the surface of minimum area spanning a given curve in 3-D space. The solution to the Plateau’s Problem involves methods in differential geometry [Douglas 1931]. For efficiency and robustness, we introduce a divide-and-conquer method for constructing a  $P$  with a boundary cycle  $b$ , so that  $P$  lies in the *discrete convex hull* (i.e., the set of all grid cells contained or partially contained in the convex hull) of  $b$ . (A similar procedure was used in [Schroeder et al. 1992] for triangulating a loop of polygon edges.)

The method relies on the fact that, as shown on the left of figure 7, there exists a *splitting plane* on the primal grid that intersects a cycle  $b$  on the dual surface with two edges  $e_1$  and  $e_2$  orthogonal to the plane. To find  $P$ , we first construct a band of quads  $Q$  connecting  $e_1$  and  $e_2$ , which splits  $b$  into two sub-cycles  $b_1$  and  $b_2$  such that  $b = \partial(Q) \oplus b_1 \oplus b_2$ . Assuming that recursions in sub-cycles  $b_1$  and  $b_2$  produce two sub-patches  $P_1$  and  $P_2$ , respectively,  $P$  is formed by  $Q \oplus P_1 \oplus P_2$ .

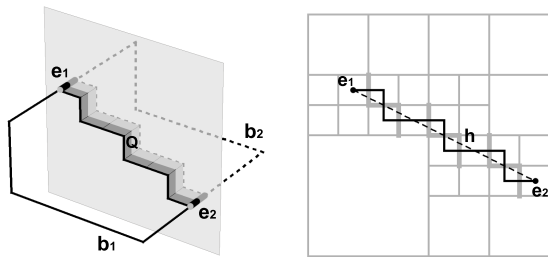


Figure 7: Patch construction by divide-and-conquer. Left: quads  $Q$  connect edges  $e_1$  and  $e_2$  orthogonal to the splitting plane and divide the original cycle into  $b_1$  (solid lines) and  $b_2$  (broken lines). Right: on the splitting plane, quads  $Q$  are dual to cell edges (thickened) crossing the line  $h$  (dashed) connecting the projections of  $e_1$  and  $e_2$ .

To construct a patch with low surface area, we let the band of quads  $Q$  follow the plane on which  $e_1$  and  $e_2$  lie. Specifically, as shown on the right of figure 7, we connect the projections of  $e_1$  and  $e_2$  on the splitting plane with a line segment  $h$  (in practice the projections are computed by averaging the locations and normals of existing edge intersections on the cell faces dual to  $e_1$  and  $e_2$ ). We then construct  $Q$  as the quads dual to the cell edges on the primal grid that cross  $h$  (new edge intersections are created when necessary by intersecting  $h$  with the cell edges and interpolating the normals associated with the end points of  $h$ ). Since  $h$  lies inside the convex

hull of the boundary cycle  $b$ , the resulting quads on the patch  $P$  always stay inside the discrete convex hull of  $b$ .

We shall prove that the recursive procedure always terminates and constructs a  $P$  with the desired boundary  $b$ . In fact, by building  $Q$  on a Manhattan path between  $e_1$  and  $e_2$ , the length of the two sub-cycles  $b_1$  and  $b_2$  are strictly less than the length of  $b$ . Hence recursions in  $b_1$  and  $b_2$  are guaranteed to terminate. To show that  $\partial(P) = b$ , we observe that when  $b = \emptyset$ ,  $\partial(P) = \emptyset = b$ . When  $b \neq \emptyset$ , assuming that  $\partial(P_1) = b_1$  and  $\partial(P_2) = b_2$ , we have  $\partial(P) = \partial(Q \oplus P_1 \oplus P_2) = \partial(Q) \oplus b_1 \oplus b_2 = b$ .

The divide-and-conquer method naturally leads to a top-down implementation on the octree grid. Let `patchProc[N,b]` be the recursive procedure that patches a boundary cycle  $\mathbf{b}$  inside an octree node  $\mathbf{N}$ . By choosing the splitting planes as the center planes of  $\mathbf{N}$ , `patchProc[N,b]` first splits  $\mathbf{b}$  into sub-cycles  $\mathbf{b}_{i,j}$  that lie in the children nodes  $\mathbf{N}_i$ . Then `patchProc[Ni,bi,j]` is called for every child node  $\mathbf{N}_i$  and its enclosing sub-cycles  $\mathbf{b}_{i,j}$ . Note that each  $\mathbf{b}_{i,j}$  is bounded by a cell that is half the size of  $\mathbf{b}$ , hence the recursion is guaranteed to stop after  $H$  levels, where  $H$  is the depth of the octree. To compute the symmetric difference of the patch and the existing dual surface, we only need to negate the intersection/non-intersection property of the cell edges dual to the quads in the patch.

### 5.4 Generate signs

The patching process results in a new set of intersection edges  $\hat{E}$  on the primal grid dual to the patched dual surface  $\hat{S}$ . Since  $\partial(\hat{S})$  is empty,  $\hat{E}$  possesses a consistent sign configuration. We introduce a recursive procedure `signProc[N,s]`, which generates signs in an octree node  $\mathbf{N}$  given a sign  $\mathbf{s}$  at the lower-left-far corner, and returns the resulting signs at its eight corners as a list. At a leaf node, `signProc[N,s]` infers signs at the other seven corners based on the intersection edges. At an internal node, `signProc[N,s]` first calls `signProc[N1,s]`, where  $\mathbf{N}_1$  refers to the lower-left-far child, and obtains signs at the corners of  $\mathbf{N}_1$  as  $\mathbf{s}_i (i = 1, \dots, 8)$ . Then `signProc[Ni,si]` is called for every other child node  $\mathbf{N}_i (i \neq 1)$  with the sign  $\mathbf{s}_i$  at their lower-left-far corners.

## 6 Surface reconstruction

After signs are determined at each grid point, a closed surface separating grid points with opposite signs can be constructed using contouring algorithms. If the locations of the intersection points are stored on the edges during scan-conversion, a primal contouring algorithm can be used, such as the Marching Cubes algorithm [Lorensen and Cline 1987]. If normals are attached to the intersection points in addition to their locations (i.e., a Hermite representation), we can use Dual Contouring [Ju et al. 2002] to reproduce sharp features in the original model. In either case, the polygons on the contoured surface do not self-intersect, and can be consistently oriented to face the external volume.

### 6.1 Examples

Figures 8 and 9 demonstrate repairing CAD models with sharp features using the presented method. Figure 8 (a) shows a mechanical part built using a state-of-the-art 3D design software. Note that it is not unusual for CAD software to produce hanging polygons (shown in this example) during CSG operations. The gears model in figure 9 (a) is also not closed, since the polygons between the top and the bottom gear does not lie between an internal volume and an external volume. In both examples, boundary cycles are detected on the corresponding dual surfaces, in which each boundary edge is shared by one quad (figure 8 (b)) or three quads (figure 9 (b)). The final repaired surfaces are reconstructed using the Marching Cubes algorithm (figure 8 (c) and figure 9 (c)) and Dual Contouring (figure 8 (d) and figure 9 (d)). Observe that the dual-contoured surfaces elim-

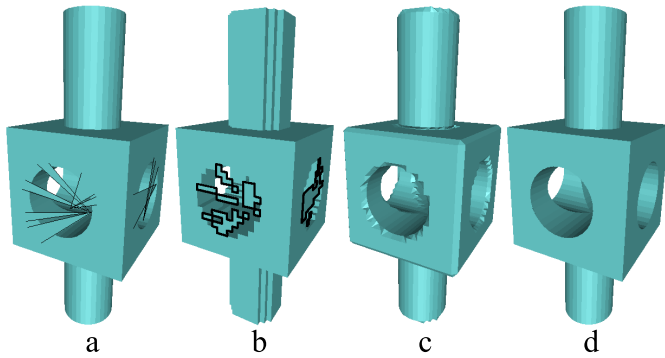


Figure 8: A mechanical part model containing hanging triangles (a), the dual surface with highlighted boundary cycles (b), reconstructed surfaces using Marching Cubes (c) and Dual Contouring (d).

inate the errors in the original models while accurately preserving the rest of the geometry, such as sharp edges and corners.

Polygonal models obtained from 3D range images typically have gaps, holes or intersecting polygons. In figure 10, the Bunny model is repaired using octree grids at different depths and reconstructed using Marching Cubes. Observe that the holes at the bottom of the Bunny are repaired at every grid resolution with reasonable appearance. The geometry of the original model away from the holes are faithfully reproduced to the extent that the grid resolution allows.

To test the robustness of our algorithm, we synthetically distort the Horse model on the left of figure 1 by randomly perturbing the vertex locations in each triangle in the front part, and removing one third of the triangles at random locations in the second part. Despite the numerous self-intersections, gaps and holes on the distorted model, the repaired model on the right consists of a single closed surface with no self-intersecting triangles. Observe that the repaired surface reproduces both noisy and smooth geometry on the original model.

Our model repair method can process huge models efficiently. Figure 11 (a) shows the statue of Michelangelo’s David reconstructed at 1mm resolution from 3D range scans. The model contains over 56 million triangles, and takes up over 1 Gigabyte of memory to store. Figure 11 (b) shows the repaired model using an octree of depth 13, in which each leaf cell on the octree measures approximately 1mm on the model. The repair process finishes within an hour using less than half a gigabyte of memory. Sign generation on the entire grid, in particular, takes only 45 seconds. Figure 11 (c) takes close-up looks at the model before and after repair. Observe again that the repaired surface accurately reproduces the geometry details, and generates reasonable patches where the data is missing from the original model.

Table 1 reports the time and space consumed for repairing the various models in the paper. We also include the results of repairing other models from the Stanford 3D Scanning Repository, such as the Dragon and the Happy Buddha, as well as the David model reconstructed at 2mm resolution. The experiments are performed on a consumer-level PC with 1.5GHz CPU and 2GB memory, and I/O operations are included when measuring the scan-conversion and contouring time. Observe that while scan-conversion is sensitive to the size of the input model, the performance of sign generation and surface reconstruction only depends on the complexity of the octree.

## 7 Discussion and future work

The goal of our work is to design a simple algorithm for generating a closed surface robustly for any arbitrary input mesh. Due to the simplicity of our approach, however, the repaired region on the

reconstructed surface may not look optimal with respect to the surrounding mesh, particularly in places of complex holes with multiple boundaries or highly curved shapes. Moreover, curved internal membranes may not be satisfactorily removed as a result of the simple divide-and-conquer approach. Without compromising the robustness of our method, we can improve the appearance of the repaired surface by exploring other means by which the boundary cycles on the dual surface can be patched. For example, boundary cycles that are within a distance threshold can be combined, and existing geometry can be taken into consideration when patching the cycles. We are also investigating incorporation of diffusion-based methods [Davis et al. 2002] for creating more natural looking patches on the dual surface with the surrounding geometry.

Due to the use of an intermediate volume, the reconstructed surface using our method may contain topological redundancy in the form of handles, cavities, and disconnected components (the genus of each repaired model in this paper are reported in Table 1). Since a large number of topology repair methods operate directly in the volume domain, we can adapt these methods to work on the octree volume produced by the second step (i.e., sign generation) so that a topologically acceptable surface is produced by contouring. In particular, we can apply the robust seed algorithm [Andujar et al. 2002] to remove internal cavities, and the topology simplification algorithm [Wood et al. 2002] to reduce the genus of the surface. Morphological operators introduced in [Nooruddin and Turk 2003] can also be applied to remove small features on the repaired volume if desired.

Finally, to reduce triangle count due to the use of contouring algorithms, we can output the reconstructed mesh to a polygon simplifier, such as QSLim [Garland and Heckbert 1997], to obtain a decimated model. Alternatively, if Hermite data are present, QEF simplification method [Ju et al. 2002] can be applied directly on the octree grid, so that the dual-contoured mesh is already simplified to the desired extent.

## 8 Conclusion

In this paper we present a robust method for repairing polygonal models. Given an arbitrary polygonal model represented as polygon soups, our method always produces a closed and consistently oriented surface. As a volumetric approach, our method employs an octree grid and memory-efficient operations that are capable of processing huge models. The output surface is guaranteed to be defect-free due to a simple and robust algorithm for generating signs on the octree grid. Sharp features in the input model can also be faithfully reproduced by storing Hermite data.

## 9 Acknowledgement

I would like to thank my advisor, Joe Warren, for his continuous support and insightful comments.

## References

- ANDUJAR, C., BRUNET, P., AND AYALA, D. 2002. Topology-reducing surface simplification using a discrete solid representation. *ACM Transaction on Graphics* 21, 2, 88–105.
- BAREQUET, G., AND KUMAR, S. 1997. Repairing CAD models. In *IEEE Visualization '97*, R. Yagel and H. Hagen, Eds., 363–370.
- BOLLOBAS, B. 1979. *Graph Theory: An Introductory Course*. New York: Springer-Verlag.
- BORODIN, P., NOVOTNI, M., AND KLEIN, R. 2002. Progressive gap closing for mesh repairing. In *Advances in Modelling, Animation and Rendering*, J. Vince and R. Earnshaw, Eds. Springer Verlag, July, 201–213.

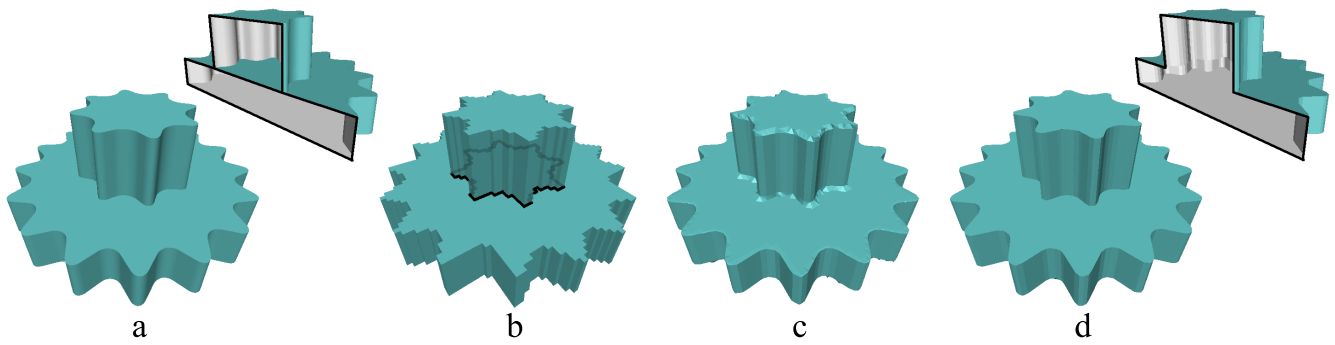


Figure 9: A gear model (a) that contains a membrane in the middle, the dual surface with a highlighted boundary cycle (b), reconstructed surfaces using Marching Cubes (c) and Dual Contouring (d). The cross-section views of the original and the repaired model are shown at their top-right corners.

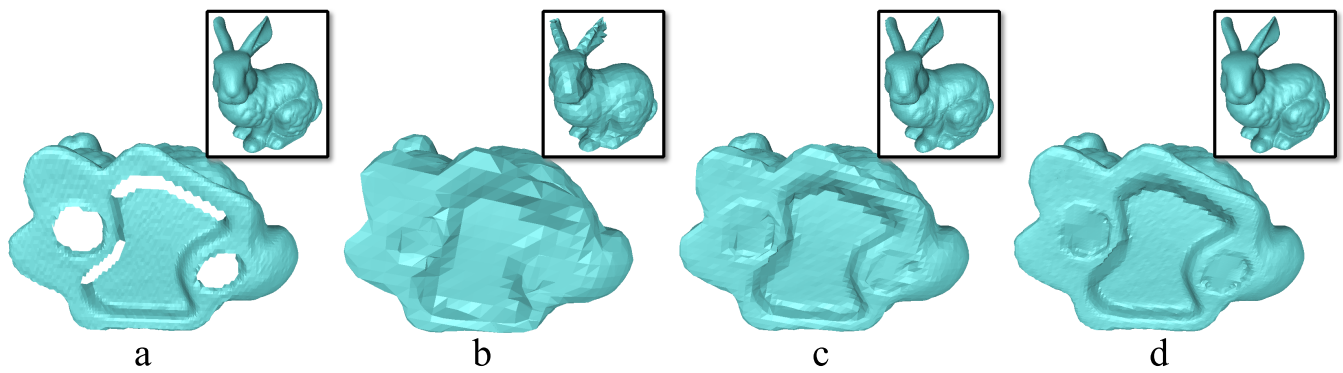


Figure 10: The original bunny model (a) repaired at octree depth 5 (b), 6 (c) and 7 (d).

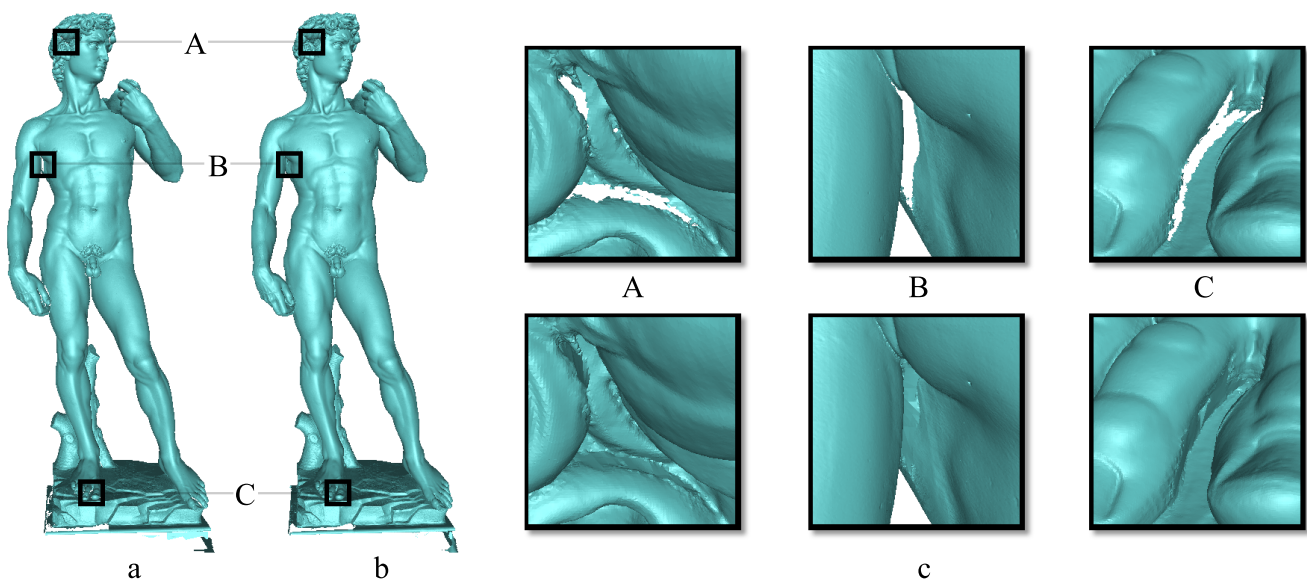


Figure 11: Repairing David: the original model at 1mm resolution (a), the repaired model at the same resolution (b), and close-ups on the model before repair (top row in (c)) and after repair (bottom row in (c)).

Model	Input Triangles	Octree Depth	Octree Leaf Cells	Scan-conversion Time (sec)	Sign generation Time (sec)	Contouring Time (sec)	Memory Usage (MB)	Output Triangles	Surface Genus
Mechanic Part	25,487	6	5,449	0.89	0.01	0.094	< 10	5,480	3
Gears	11,610	6	9,316	0.86	0.03	0.14	< 10	8,926	0
Bunny	69,451	7	46,028	2.75	0.06	0.79	< 10	91,716	0
Horse	80,805	8	93,164	3.62	0.31	2.14	< 10	163,692	1
Dragon	871,414	9	543,449	37.16	0.69	7.92	16	1,085,404	2
Buddha	1,087,716	10	1,716,718	56.47	2.02	18.22	28	3,425,781	11
David (2mm)	8,254,150	12	6,539,335	362.98	8.20	133.75	92	13,059,000	15
David (1mm)	56,230,343	13	31,561,029	2482.14	45.82	661.64	417	62,825,040	179

Table 1: Performance results on repairing various models on a consumer level PC with 1.5GHz CPU and 2GB memory.

- CURLESS, B., AND LEVOY, M. 1996. A volumetric method for building complex models from range images. In *Proceedings of ACM SIGGRAPH 1996*, ACM Press / ACM SIGGRAPH, New York. E. Fiume, Ed., Computer Graphics Proceedings, Annual Conference Series, ACM, vol. 30, 303–312.
- DACHILLE, F., AND KAUFMAN, A. 2000. Incremental triangle voxelization. In *Graphics Interface*, 205–212.
- DAVIS, J., MARSCHNER, S. R., GARR, M., AND LEVOY, M. 2002. Filling holes in complex surfaces using volumetric diffusion. In *First International Symposium on 3D Data Processing, Visualization, and Transmission*.
- DOUGLAS, J. 1931. Solution of the problem of plateau. *Transactions of the American Mathematical Society* 33, 263–321.
- FOLEY, J. D., VAN DAM, A., FEINER, S. K., AND HUGHES, J. F. 1990. *Computer Graphics (2nd Edition)*. Addison-Wesley Publication Company.
- FRISKEN, S. F., PERRY, R. N., ROCKWOOD, A. P., AND JONES, T. R. 2000. Adaptively sampled distance fields: A general representation of shape for computer graphics. In *Proceedings of ACM SIGGRAPH 2000*, ACM Press / ACM SIGGRAPH, New York. E. Fiume, Ed., Computer Graphics Proceedings, Annual Conference Series, ACM, K. Akeley, Ed., 249–254.
- GARLAND, M., AND HECKBERT, P. S. 1997. Surface simplification using quadric error metrics. In *Proceedings of ACM SIGGRAPH 1997*, ACM Press / ACM SIGGRAPH, New York. E. Fiume, Ed., Computer Graphics Proceedings, Annual Conference Series, ACM, 209–216.
- HUANG, J., YAGEL, R., FILIPPOV, V., AND KURZION, Y. 1998. An accurate method for voxelizing polygon meshes. In *IEEE Symposium on Volume Visualization*, 119–126.
- JU, T., LOSASSO, F., SCHAEFER, S., AND WARREN, J. 2002. Dual contouring of hermite data. *ACM Transactions on Graphics* 21, 3, 339–346.
- KOBELT, L. P., BOTSCH, M., SCHWANECKE, U., AND SEIDEL, H.-P. 2001. Feature sensitive surface extraction from volume data. In *Proceedings of ACM SIGGRAPH 2001*, ACM Press / ACM SIGGRAPH, New York. E. Fiume, Ed., Computer Graphics Proceedings, Annual Conference Series, ACM, 57–66.
- LIEPA, P. 2003. Filling holes in meshes. In *Proceedings of the Eurographics/ACM SIGGRAPH symposium on Geometry processing*, Eurographics Association, 200–205.
- LORENSEN, W., AND CLINE, H. 1987. Marching cubes: A high resolution 3d surface construction algorithm. In *Computer Graphics (Proceedings of ACM SIGGRAPH 87)*, ACM Press, vol. 21, 163–169.
- MURALI, T. M., AND FUNKHOUSER, T. A. 1997. Consistent solid and boundary representations from arbitrary polygonal data. In *Proceedings of the 1997 symposium on Interactive 3D graphics*, ACM Press, 155–ff.
- NOORUDDIN, F. S., AND TURK, G. 2003. Simplification and repair of polygonal models using volumetric techniques. *IEEE Transactions on Visualization and Computer Graphics* 9, 2, 191–205.
- OHTAKE, Y., BELYAEV, A., ALEXA, M., TURK, G., AND SEIDEL, H.-P. 2003. Multi-level partition of unity implicits. *ACM Transactions on Graphics* 22, 3, 463–470.
- OOMES, S., SNOEREN, P., AND DIJKSTRA, T. 1997. 3d shape representation: Transforming polygons into voxels. In *ScaleSpace97*.
- SCHROEDER, W. J., ZARGE, J. A., AND LORENSEN, W. E. 1992. Decimation of triangle meshes. In *Computer Graphics (Proceedings of ACM SIGGRAPH 92)*, ACM Press, 65–70.
- TURK, G., AND LEVOY, M. 1994. Zippered polygon meshes from range images. In *Proceedings of ACM SIGGRAPH 1994*, ACM Press / ACM SIGGRAPH, New York. E. Fiume, Ed., Computer Graphics Proceedings, Annual Conference Series, ACM, 311–318.
- WOOD, Z., HOPPE, H., DESBRUN, M., AND SCHRODER, P. 2002. Isosurface topology simplification. *Technical Report MSR-TR-2002-28, Microsoft Research* (January).

## A Existence of consistent signs

**Proposition:** *An octree grid with intersection edges  $E$  can be consistently signed if and only if its dual surface  $S$  contains no boundary edges, that is,  $\partial(S) = \emptyset$ .*

**Proof:** Observe that a consistent sign configuration exists if and only if every closed circuit of edges on the grid contains an even number of intersection edges. In particular, since a cell face is a closed circuit of four edges, every cell face should contain an even number of intersection edges if a consistent sign configuration exists, hence the dual surface contains no boundary edges. To prove sufficiency, we assume that a consistent sign configuration does not exist. Among all closed circuits on the grid containing an odd number of intersection edges (referred to as *odd circuits*), we find the one with the shortest length and denote it as  $b$ . Following a similar argument as in Section 5.3,  $b$  can be split into two sub-circuits,  $b_1$  and  $b_2$ , by a band of cell faces  $Q$  in the middle (see figure 7 left). Since  $b_1$  and  $b_2$  are strictly shorter than  $b$ , they can not be odd circuits. Hence at least one cell face in  $Q$  contains an odd number of intersection edges. In other words, a boundary edge must exist on the dual surface. This completes the proof.