Consistent Solid and Boundary Representations from Arbitrary Polygonal Data

(in *Proceedings of the 1997 Symposium on Interactive 3D Graphics*, April 1997)

T. M. Murali*
Brown University

Thomas A. Funkhouser †
Bell Laboratories

Abstract

Consistent representations of the boundary and interior of three-dimensional solid objects are required by applications ranging from interactive visualization to finite element analysis. However, most commonly available models of solid objects contain errors and inconsistencies. We describe an algorithm that automatically constructs consistent representations of the solid objects modeled by an arbitrary set of polygons. The key feature of our algorithm is that it first partitions space into a set of polyhedral regions and then determines which regions are solid based on region adjacency relationships. From the solid polyhedral regions, we are able to output consistent boundary and solid representations in a variety of file formats. Unlike previous approaches, our solid-based approach is effective even when the input polygons intersect, overlap, are wrongly-oriented, have T-junctions, or are unconnected.

1 Introduction

We define a set of polygons in $\mathbb{R}^3$ to be *consistent* if the union of the polygons is a closed 2-manifold (see Hoffmann's book [12] for a definition) in which each polygon is oriented with its normal pointing away from the interior of the volume enclosed by the manifold. We say that a consistent set of polygons is a *correct representation* of a polyhedral solid object in $\mathbb{R}^3$ if the manifold formed by the polygons is identical to the boundary of the solid object. Intuitively, such a representation allows the unambiguous classification of a point as lying inside the object, outside the object, or on the surface of the object.

Correct representations of three-dimensional objects are useful in a number of applications. For instance, back-face culling, a technique used to render complex models quickly [6], requires that the polygons in the model be consistently and correctly oriented. More sophisticated rendering algorithms perform visibility culling by processing the interiors of the solid objects in the model [19, 20, 27]. In the case of interactive collision-detection, some algorithms first process "free" space, i.e., the complement of the union of all the obstacles in the environment [11]. Such algorithms require a correct representation of the boundary of the obstacles so that they can effectively construct the free space. Similarly, algorithms for lighting simulation process meshes constructed on the boundaries of the objects being lit or analysed [1]. If the boundaries have cracks, the mesh is malformed, causing errors and artifacts like spurious shadows in the result. "Bad" meshes can also produce errors in finite element analysis. Further, basic CAD/CAM operations like computing the mass or volume of an object, solid modeling techniques such as Constructive Solid Geometry that perform set operations on solid objects [18, 21, 28], and rapid prototyping [24], which is used to manufacture objects from CAD designs, need models with continuous and closed boundaries, with no cracks or improper intersections. Finally, systems that design and optimize wireless communication systems for a closed environment like a building require descriptions of the boundaries of the obstacles in the building [8, 14].

Unfortunately, most commonly available models of solid objects, whether created by hand or by using automatic tools, contain geometric and topological flaws. Typical errors are:

- wrongly-oriented polygons,
- intersecting or overlapping polygons,
- missing polygons,
- cracks (edges that should be connected have distinct endpoints, faces that should be adjacent have separated edges), and
- T-junctions (the endpoint of one edge lies in the interior of another edge).

For example, in Figure 1, there is a crack between segments a and b; segments c and d intersect; and the right endpoint of segment e lies in the interior of d, forming a

*Address: Department of Computer Science, Box 90129, Duke University, Durham, NC 27708–0129. Email: tmax@cs.duke.edu. This work was done when the author was visiting Bell Laboratories.

†Bell Laboratories, 700 Mountain Avenue, 2A-202, Murray Hill, NJ 07974. Email: funk@bell-labs.com

T-junction. Such errors may be caused by operational mistakes made by the person creating the model, may creep in when converting from one file format to another, or may occur because a particular modeler does not support some features (e.g., a snap grid).

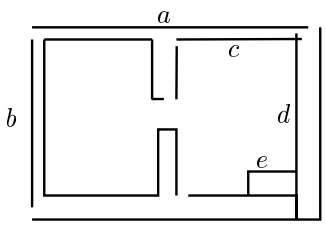


Figure 1: A non-manifold model

Motivated by the above applications and model imperfections, we consider the following *solid reconstruction* problem in this paper:

> *From an arbitrary set of polygons in $\mathbb{R}^3$, reconstruct a correct representation of the solid objects modeled by the polygons.*

We describe an automatic technique to solve the solid reconstruction problem that works well on many realistic models and is guaranteed to output a consistent set of polygons.

The remainder of this paper is organized as follows: In the next section, we discuss previous algorithms to solve the above problem. In Section 3, we briefly outline our approach. We describe our solid reconstruction algorithm in more detail Section 4. Section 5 contains experimental results and a brief discussion of the advantages and limitations of our approach. In Section 6, we discuss techniques that we plan to implement to overcome these limitations. The final section is a brief conclusion.

2 Previous Work

In the computational geometry and solid modeling communities, there has been a lot of work on the related problem of robust geometric computing [7, 12, 13, 23, 25, 26, 29, 31]. These techniques are not applicable to our problem since they attempt to avoid errors caused by *numerical* imprecision and cannot clean-up already incorrect data.

It has been noted in the literature that there are currently no robust techniques to solve the solid reconstruction problem [10, 17]. Previous approaches can be divided into two categories: boundary-based approaches and solid-based approaches.

Boundary-based techniques determine how the input polygons mesh together to form the boundaries of the objects modeled by them. Typically, these algorithms merge vertices and edges that are "close" or zip together the boundaries of two faces by merging pairs of "nearby" vertices, where "close" and "nearby" are defined in terms of a pre-specified tolerance. Some boundary-based methods assume that either all the input polygons are consistently oriented or that the orientation of a polygon can be determined from the order of the vertices on its boundary [1, 3]. Such an assumption is often invalid since many datasets contain inconsistently oriented polygons. Other algorithms require (a lot of) user intervention [5, 15], are inherently two-dimensional [14, 16] or are limited to removing parts of zero-volume (like internal walls) from CAD models [2]. Bøhn and Wozny [3] fill cracks or holes by adding polygons; their method can potentially add a lot of polygons to the model. However, the most common deficiency of many of the previous techniques is that they use scene-relative tolerances to "fill over" cracks and generate connectivity information about the model [1, 4, 24]. Determining the right tolerance for a given model is a difficult task, probably requiring input from the user. Moreover, such approaches do not work well when the size of some error in the input is larger than the smallest feature in the model. In this case, no suitable tolerance can be chosen that both fills the cracks and preserves small features.

Solid-based algorithms partition $\mathbb{R}^3$ into regions and determine which regions are solid. Thibault and Naylor [28] classify a region as solid exactly when there is at least one input polygon lying on the region's boundary whose normal is directed away from the interior of the region, while Teller [27] declares a region to be solid only if a majority of the polygons lying on the region's boundary have such normals. Both techniques assume that the orientations of the input polygons are correct. As we have pointed out earlier, this assumption is unwarranted for many datasets. Note that both algorithms were developed as a means to represent polyhedra; the authors did not set out to explicitly solve the solid reconstruction problem.

3 Our Approach

We have adopted a novel solid-based approach that uses region adjacency relationships to compute which regions are solid and constructs a consistent set of polygons from the solid regions. In contrast to previous boundary-based approaches that attempt to stitch and orient boundary polygons directly, we first focus on classifying *spatial* regions as solid or not and then derive a boundary representation from the solid regions. Also, in contrast to previous solid-based approaches that determine whether regions are solid or not based only on local input polygon orientations, we execute a *global* algorithm that focuses on the opacities of boundaries between regions. As a result, unlike previous approaches, our algorithm is: 1) effective for models containing intersecting, overlapping, and unconnected polygons, 2)

independent of the input polygon orientations, and 3) guaranteed to output a consistent set of polygons.

4 Reconstruction Algorithm

Our algorithm proceeds in three phases (as show in Figure 2): (a) spatial subdivision, (b) determination of solid regions, and (c) model output.

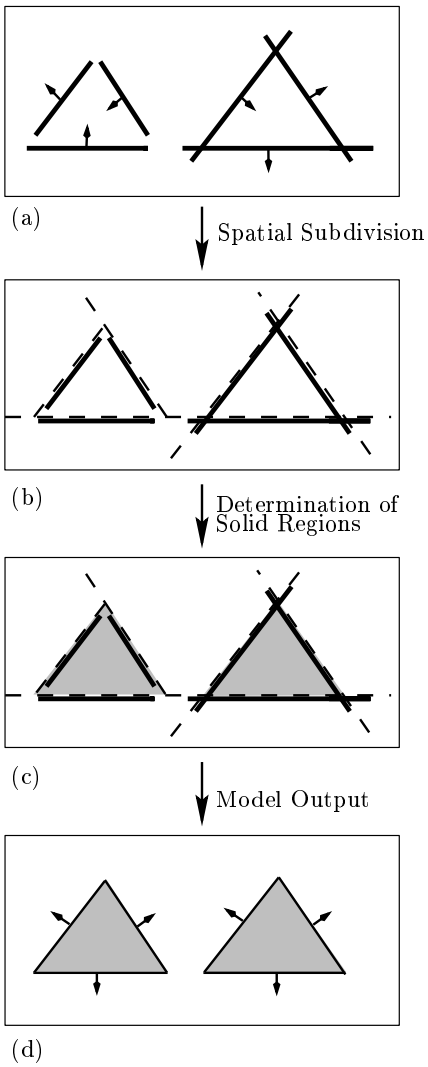


Figure 2: (a) Input model with incorrectly-oriented polygons (b) Subdivision of space (edges of the subdivision drawn slightly shifted from the input edges for clarity) (c) Solid regions (shaded) (d) Output model with correct polygon orientations.

4.1 Spatial Subdivision

During the spatial subdivision phase, we partition $\mathbb{R}^3$ into a set of polyhedral cells and build a graph that explicitly represents the adjacencies between the cells of the subdivision. Each cell in the spatial decomposition is represented by a node in the graph. Two nodes have a link between them if the corresponding cells are adjacent (i.e., the cells share a planar boundary).

Note that any partition of $\mathbb{R}^3$ (e.g., a tetrahedral decomposition) will satisfy our purposes, as long as the input polygons lie in the faces of the decomposition. In our current implementation, the cells correspond to the leaves of a Binary Space Partition (BSP) [9, 22]. To construct the BSP, we specify a list of splitting planes with which we recursively partition $\mathbb{R}^3$. Each plane in the list contains one of the polygons in the input; we create only one plane for a set of coplanar polygons. For each plane, we count the total area of the polygons lying in it. We process the planes in decreasing order of area to split $\mathbb{R}^3$.

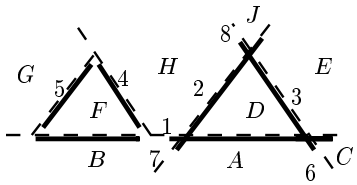
Construction of the cell adjacency graph is dovetailed with the construction of the BSP. Each node in the graph represents a convex polyhedron. Each link represents a convex polygon, and is augmented by lists of polygons describing the link's opaque portions (those covered by some input polygon) and the link's transparent portions (those not covered by any input polygon). If a leaf in the BSP is split into two by a plane, we create new nodes in the graph corresponding to the new leaves in the BSP, and update the links of the split leaf's neighbors to reflect the new adjacencies. For each updated link, we perform two more operations: (i) we subtract from the transparent part of the link any input polygon that is coplanar with the splitting plane and (ii) we add that input polygon to the link's opaque part.

Figure 3 shows an example spatial subdivision (in the plane). The input "polygons" are shown in Figure 3(a) as thick line segments. The BSP leaf nodes (regions labeled with letters) are constructed using splitting "planes" (dashed lines labeled with numbers) that support input "polygons," as shown in Figure 3(b). Finally, the cell adjacency graph for this example is shown in Figure 3(c) with the opacity of each link indicated by its line style (solid lines represent the opaque parts of a link, while dashed lines represent the transparent parts).

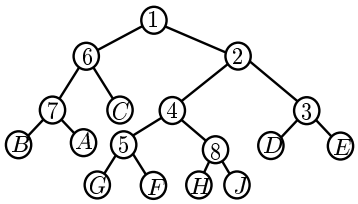
4.2 Determination of Solid Regions

During the solid determination phase, we compute whether each cell is solid or not, based on the properties of its links and neighbors. This approach is motivated by the following observations:

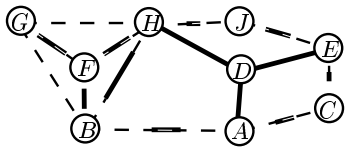
1. if two adjacent cells share a mostly transparent boundary, it is likely that they are both solid or both non-solid,



(a) Example input model with spatial subdivision.



(b) Binary space partition



(c) Cell adjacency graph

Figure 3: Spatial subdivision example

2. if two adjacent cells share a mostly opaque boundary, it is likely that one is solid and the other is non-solid, and

3. unbounded cells (i.e., the ones on the "outside" that contain points at infinity, like cell E in Figure 3(a)) are not solid.

We quantify "how solid" each cell C_i is by its *solidity*, s_i, a real number that ranges between -1 and 1. We use $s_i = 1$ to denote that C_i is solid (i.e., contained in the interior of a solid object), and $s_i = -1$ to denote that C_i is non-solid (i.e., lies in the exterior of all solid objects). An s_i value between -1 and 1 indicates that we are not entirely sure whether C_i is solid or not.

Our solid determination algorithm proceeds as follows. First, we assign a solidity value of -1 to all unbounded cells since they are in the exterior of all solid objects. Then, we compute the solidity s_i of each bounded cell C_i based on the solidities of its neighbor cells and the opacities of its links.

Formally, let $a_{i,j}$, $o_{i,j}$, and $t_{i,j}$ represent the total area, opaque area, and transparent area, respectively, of the link $L_{i,j}$ between cells C_i and C_j. Note that $a_{i,j} = o_{i,j} = t_{i,j} = 0$ iff C_i and C_j are not adjacent (do not share a planar boundary); otherwise, $a_{i,j} > 0$ and $o_{i,j}, t_{i,j} \geq 0$. Let the total area of the boundary of C_i be denoted by $A_i = \sum_j a_{i,j}$, where C_j ranges over all neighbors of C_i. Now, we can write an expression for the solidity of each bounded cell C_i in terms of the solidities of each other cell C_j:

$$s_i = \frac{\sum_j (t_{i,j} - o_{i,j})\, s_j}{A_i} \tag{1}$$

This formulation for computing cell solidities matches our intuition. When the link $L_{i,j}$ between two cells C_i and C_j is entirely transparent ($t_{i,j} = a_{i,j}$ and $o_{i,j} = 0$), s_j, the solidity that C_j contributes to s_i, is scaled by $a_{i,j}$, the maximum possible (positive) value. On the other hand, when $L_{i,j}$ is entirely opaque ($o_{i,j} = a_{i,j}$ and $t_{i,j} = 0$), the contribution from C_j to s_i is scaled by $-a_{i,j}$, the minimum possible (negative) number. Finally, when $L_{i,j}$ is partially opaque ($0 < t_{i,j} < a_{i,j}$ and $0 < o_{i,j} < a_{i,j}$) the contribution from C_j to s_i is a linear interpolation between these two extremes. We divide the total contribution to s_i by A_i to normalize the value of s_i between -1 and 1.

If the BSP has n bounded cells, (1) leads to a linear system of n equations, $Mx = b$, where

- $x \in \mathbb{R}^n$ is a vector of the (unknown) solidities of the bounded cells,
- $b \in \mathbb{R}^n$ is a vector representing the contributions from the (known) solidities of the unbounded cells ($b_i = \sum_k (o_{i,k} - t_{i,k})$, where k ranges over all unbounded neighbors of C_i),
- and M is an $n \times n$ matrix with the following properties (here i and j are integers with $1 \leq i, j \leq n$ and $i \neq j$):

  1. Each diagonal element is positive: $M_{i,i} = A_i > 0$.
  2. $M_{i,j} = o_{i,j} - t_{i,j}$. Thus,
     - $M_{i,j} > 0$ indicates $L_{i,j}$ is mostly opaque, and
     - $M_{i,j} < 0$ indicates $L_{i,j}$ is mostly transparent.
  3. M is symmetric, i.e., $M_{i,j} = M_{j,i}$.
  4. M has weak diagonal dominance, i.e.,

$$\sum_{j, j \neq i} |M_{i,j}| \leq |M_{i,i}|,$$

  for all i and there is a k such that

$$\sum_{j, j \neq k} |M_{k,j}| < |M_{k,k}|$$

  (if C_k has an unbounded neighbor or when C_k has at least one link that is not fully opaque or fully transparent).

These properties imply that M has an inverse (see the Appendix for a proof). Hence, the linear system $Mx = b$ has a unique solution. We can solve the linear system to obtain the the cell solidities by computing $x = M^{-1}b$. However, inverting M takes $O(n^2)$ time, which can be prohibitive costly if the BSP has many leaves, as is likely to be the case if there are many polygons in the input. In

such cases, we take advantage of the fact that each leaf in the BSP has a small number of neighbors. Therefore, M is sparse and we can use an iterative procedure to solve the linear system efficiently [30]. We would like to point out that it is not difficult to show that the elements of x have values between -1 and 1.

In our implementation, we use Gauss-Seidel iterations. Each iteration takes time proportional to the number of links in the adjacency graph. We set the initial values of the solidities of the bounded cells of the subdivision to be 0. We terminate the iterations when the change in the solidity of each cell is less than some small pre-specified tolerance. During each iteration, we update the solidities in arbitrary order. We plan to study techniques that order the updates cleverly so as to increase the rate of convergence.

After solving the linear system of equations, we classify each cell as solid or not (a binary choice) by looking at the sign of its solidity. A cell whose solidity is positive is determined to be solid, whereas a cell whose solidity is negative or zero is determined to be non-solid.

4.3 Model Output

Finally, in the model output phase, we write files containing consistent descriptions for the objects represented by the input polygons. To generate a consistent representation, we simply output a polygonal description of all links in the adjacency graph that represent the boundaries between cells that are solid ($s_i > 0$) and cells that are not solid ($s_i \leq 0$). We consistently orient all output polygons away from solid cells (see Figure 2(d)).

Another file format we currently support is a solid-based representation that represents the solid objects by explicitly encoding the BSP as a list of split planes augmented by a solidity value for each leaf cell.

5 Results and Discussion

We have implemented our solid reconstruction algorithm and run it on a number of datasets.

In the trivial case, when the input model is a manifold polygonal surface, the boundary representation output by our algorithm is identical to the input model (as it should be). In the manifold case, all cells lie entirely in the interior or exterior of the modeled objects, and the solidity computed by our algorithm is exactly 1 for every cell in the interior of the solid object and exactly -1 for every cell in the exterior of all solid objects. This follows readily from the fact that each link is either fully opaque or fully transparent. Note that unlike previous boundary-based approaches, our algorithm additionally outputs a representation of the modeled object as the union of a set of convex polyhedra.

In many complex cases, when the input model contains errors, our algorithm is able to fix errors automatically and output consistent solid and boundary representations, even in cases where previous approaches are unsuccessful. For instance, consider the 3D polygonal model of a coffee mug shown in Plate I(a). In this example, polygons are oriented randomly (back-facing polygons are drawn in black); the handle is modeled by several improperly intersecting and disconnected hollow cylinders (note the gaps along the top silhouette edge of the handle); and the polygons at both ends of the handle form improper intersections with the side of the cup.

All previous approaches known to the authors fail for this simple example. Boundary-based approaches that traverse the surface of the object [15] fail in the areas where polygons are unconnected (along the handle). Proximity-based approaches [1, 4, 24] that merge features within some tolerance of each other do not work as no suitable tolerance can be chosen for the entire model because the size of the largest error (a crack between polygons on the handle) is larger than the size of the smallest feature (a bevel on the top of the lip). Finally, solid-based approaches [27, 28] that decide whether each cell is solid or not based on the orientations of the input polygons along the cell's boundaries fail because the input has many wrongly-oriented polygons.

Our algorithm is able to fix the errors in this example and output a correct and consistent model (see Plates I(b)-I(d)). Plate I(b) shows outlines of the cells constructed during the spatial subdivision phase, with each cell labeled by its solidity computed during the solid determination phase. In addition, each cell C_i is outlined with a color that depends on the value of its solidity s_i (the color ranges from red when $s_i = 1$ to green when $s_i = -1$). Plate I(c) shows oriented polygons computed during the model output phase that are drawn along the boundaries between solid and non-solid cells. Finally, Plate I(d) shows the boundary representation output of our algorithm. The reconstructed model is both correct and consistent: the cracks in the handle have been filled in; intersections in the handle have been made explicit; and all polygons have been oriented correctly.

Plates II-IV show results derived from experiments with larger models from a variety of applications. The images on the left side of the last page show the input models, while the images on the right show different visualizations of the cell solidities computed for these models with our algorithm. In all these cases, we were able to construct correct and consistent solid and boundary representations. The three sets of images demonstrate the importance of appropriate visualization techniques for viewing the solid cells of the BSP. For instance, the text strings drawn in Plate II(b) would be overlapping if used in Plates III(b) and IV(b). Similarly, the opaque boundaries drawn in Plate III(b) would be inappropriate for use in Plate IV(b) as the outer-most solid cells (ceilings and floors) would mask the interior

non-solid cells. In complex cases, such as Plate IV(b), we simply represent cell solidities by colored dots drawn at the cells' centroids. In all cases, solidity information is drawn with a representative color linearly interpolated between red (for $s_i = 1$) and green (for $s_i = -1$). We use a 3D viewing program to allow the user to select different visualization options interactively.

We now present an analysis of the running time of our algorithm for the above models. In the table below, each row corresponds to the model whose name is specified in the first column. The second column displays the number of polygons in the model. The third column contains the number of cells in the spatial subdivision constructed for that model. The fourth and fifth columns specify the number of Gauss-Seidel iterations needed for convergence and the total running time for the model in seconds, respectively. These experiments were run on SGI Indigo2 with a 200MHz R4400 processor. For the mug, about 35% of the total time was spent on calculating solidities, while for the other models, this time ranged from 10-15%.

Model	#polys	#cells	#iter.	time
Mug	121	359	61	9.77
Clutch	420	159	39	8.64
Phone	1228	819	82	78.64
Building	1687	1956	92	240.34

The advantages of our algorithm are related to the fact that we use a *global* approach to classify *regions of space* rather than just considering local boundary relationships or feature proximities. First, our algorithm is effective for models containing intersecting, overlapping, or unconnected polygons for which it is difficult to traverse boundaries. Second, the output of our algorithm does not depend on the initial orientations of input polygons. Third, the boundary output by our algorithm is *always* guaranteed to be consistent (although it may not be a correct representation of the modeled object) since it is derived directly from the solid cells of the partition. Finally, we are able to output a solid representation of the model as well as a boundary representation, which may be critical to many applications.

However, our approach does have limitations. Its success depends on the spatial subdivision constructed. As a result, missing polygons may lead to the creation of cells that do not correctly model the shape of the solid object. Another limitation of our technique is that it is based on the assumption that input polygons separate solid and non-solid regions. Therefore, if the input model contains two solid objects that are intersecting or separated by a polygon in the input model (e.g., a mouse on a table), the solidities for the cells along the solid-on-solid boundary are driven by each other to values lower than 1. Intuitively, the polygon separating the solid objects is an "extra" polygon.

Fortunately, in many cases, cells with intermediate solidity values (s_i close to 0) identify parts of the model containing topological errors and inconsistencies like missing and extra polygons. This feature is useful for verifying model consistency and localizing model inaccuracies. For example, notice that in Plates I(a) and I(b) cells are red (s_i close to 1) in areas where the input model has no errors (on the left side of the cup) and that cells are yellow-ish (s_i close to 0) in areas where there are intersecting or unconnected polygons in the input model. This example demonstrates an important feature of our algorithm: it not only helps fix up errors, but also identifies where they are.

6 Future Work

There are two possible directions for future work. One approach we plan to examine is to determine the location of missing/extra polygons as the BSP is being constructed. During the construction of the BSP, polygons are placed on the boundaries of the current set of leaves (some of the current leaves will be interior nodes in the final BSP). We plan to maintain the solidities of the current leaves of the BSP. If a current leaf has a solidity close to 0 and there is a plane that splits the leaf so that at least one of the two new leaves has solidity close to -1 or 1, then it is likely that the splitting plane contains a missing polygon. A similar idea can be used to recognize extra polygons.

Another possible avenue for future work is to utilize final cell solidity values (those determined after the BSP has been fully constructed) to recognize missing/extra polygons. We plan to do so by defining a metric that measures the "goodness" of the BSP and using the metric to drive a simulated annealing or optimization process that maximizes the goodness of the BSP. For example, a "goodness" metric that penalizes solutions in which two cells with markedly different (respectively, similar) solidities share a mostly transparent (respectively, opaque) boundary is shown below (in the formula, i and j range over all the leaves of the BSP and $i \neq j$):

$$\text{goodness} = \frac{\sum_{i,j} \left(\left(1 - \frac{|s_i - s_j|}{2} \right) t_{i,j} + \frac{|s_i - s_j|}{2} o_{i,j} \right)}{\sum_{i,j} a_{i,j}}.$$

By associating weights with the transparent and opaque areas of each link, we can "change" the opacity of a link. Our goal then is to search for that set of weights that maximizes the goodness.

Note that the two ideas outlined above are not restricted to BSPs and can be generalized to any spatial decomposition. We expect these extensions to allow our algorithm to adapt dynamically when the model has missing polygons or solid-on-solid regions, and thus generate correct solutions for a wider class of input models.

7 Conclusions

We have described an algorithm that reconstructs consistent solid and boundary representations of objects from error-ridden polygonal data. The algorithm partitions $\mathbb{R}^3$ into a set of cells, computes the solidity of each cell based on cell adjacency relationships, and utilizes computed cell solidities to construct a consistent output representation. In contrast to previously described approaches, our method gives excellent results on many real 3D models containing intersecting, overlapping, or unconnected polygons. We believe that our technique is useful for interactive visualization, visibility culling, collision detection, radiosity, and many other interactive 3D graphics applications.

References

[1] Daniel R. Baum, Stephen Mann, Kevin P. Smith, and James M. Winget. Making radiosity usable: Automatic preprocessing and meshing techniques for the generation of accurate radiosity solutions. In *Proc. SIGGRAPH '91*, volume 25 of *Comput. Graph.*, pages 51–60, New York, 1991. ACM SIGGRAPH.

[2] Jan Helge Bøhn. Removing zero-volume parts from CAD models for layered manufacturing. *IEEE Computer Graphics and Applications*, 15(6):27–34, 1995.

[3] Jan Helge Bøhn and Michael J. Wozny. A topology-based approach for shell closure. In P. R. Wilson, M. J. Wozny, and M. J. Pratt, editors, *Geometric Modeling for Product Realization*, pages 297–319. North-Holland, Amsterdam, 1993.

[4] Geoffrey Butlin and Clive Stops. CAD data repair. In *Proceedings of the 5th International Meshing Roundtable*, October 1996. See also `http://www.fegs.co.uk/CADfix.html`.

[5] Carl Erikson. Error correction of a large architectural model: the Henderson County Courthouse. Technical Report TR95-013, Dept. of Computer Science, University of North Carolina at Chapel Hill, 1995.

[6] J. D. Foley, A. van Dam, S. K. Feiner, and J. F. Hughes. *Computer Graphics: Principles and Practice*. Addison-Wesley, Reading, MA, 1990.

[7] S. Fortune and C. J. Van Wyk. Efficient exact arithmetic for computational geometry. In *Proc. 9th Annu. ACM Sympos. Comput. Geom.*, pages 163–172, 1993.

[8] S. J. Fortune, D. M. Gay, B. W. Kernighan, O. Landron, R. A. Valenzuela, and M. H. Wright. WISE design of indoor wireless systems: practical computation and optimization. In *IEEE Computational Science and Engineering*, volume 2, pages 58–68, Spring 1995.

[9] H. Fuchs, Z. M. Kedem, and B. Naylor. On visible surface generation by a priori tree structures. *Comput. Graph.*, 14(3):124–133, 1980. Proc. SIGGRAPH '80.

[10] S. Gottschalk, M. C. Lin, and D. Manocha. OBBTree: A hierarchical structure for rapid interference detection. In Holly Rushmeier, editor, *Computer Graphics (SIGGRAPH '96 Proceedings)*, volume 30, pages 171–180, August 1996.

[11] Martin Held, James T. Klosowski, and Joseph S. B. Mitchell. Evaluation of collision detection methods for virtual reality fly-throughs. In *Proc. 7th Canad. Conf. Comput. Geom.*, pages 205–210, 1995.

[12] C. Hoffmann. *Geometric and Solid Modeling*. Morgan Kaufmann, San Mateo, California, 1989.

[13] C. M. Hoffmann, J. E. Hopcroft, and M. S. Karasick. Towards implementing robust geometric computations. In *Proc. 4th Annu. ACM Sympos. Comput. Geom.*, pages 106–117, 1988.

[14] Brian W. Kernighan and Christopher J. Van Wyk. Extracting geometrical information from architectural drawings. In *Proceedings of the Workshop on Applied Computational Geometry*, pages 82–87, May 1996.

[15] Delnaz Khorramabadi. A walk through the planned CS building. Technical Report UCB/CSD 91/652, Computer Science Dept., University of California at Berkeley, 1991.

[16] Robert Laurini and Françoise Milleret-Raffort. Topological reorganization of inconsistent geographical databases: a step towards their certification. *Computer and Graphics*, 18(6):803–813, 1994.

[17] Ernst Mücke. Comments on the Computational Geometry Impact Task Force Report. At `http://www.cs.duke.edu/~jeffe/compgeom/files/mucke.html`, June 1996.

[18] Bruce F. Naylor. SCULPT: an interactive solid modeling tool. In *Proc. Graphics Interface '90*, pages 138–148, 1990.

[19] Bruce F. Naylor. Interactive solid geometry via partitioning trees. In *Proc. Graphics Interface '92*, pages 11–18, 1992.

[20] Bruce F. Naylor. Partitioning tree image representation and generation from 3D geometric models. In *Proc. Graphics Interface '92*, pages 201–212, 1992.

[21] Bruce F. Naylor, John Amanatides, and William C. Thibault. Merging BSP trees yields polyhedral set operations. In *Proc. SIGGRAPH '90*, volume 24 of *Comput. Graph.*, pages 115–124, New York, 1990. ACM SIGGRAPH.

[22] R. A. Schumacker, R. Brand, M. Gilliland, and W. Sharp. Study for applying computer-generated images to visual simulation. Technical Report AFHRL–TR–69–14, U.S. Air Force Human Resources Laboratory, 1969.

[23] Mark Segal. Using tolerances to guarantee valid polyhedral modeling results. *Computer Graphics*, 24(4):105–114, August 1990.

[24] Xuejun Sheng and Ingo R. Meier. Generating topological structures for surface models. *IEEE Computer Graphics and Applications*, 15(6):35–41, 1995.

[25] Jonathan R. Shewchuk. Robust adaptive floating-point geometric predicates. In *Proc. 12th Annu. ACM Sympos. Comput. Geom.*, pages 141–150, 1996.

[26] K. Sugihara and M. Iri. A solid modelling system free from topological inconsistency. *Journal of Information Processing*, 12(4):380–393, 1989.

[27] Seth Jared Teller. *Visibility Computations in Densely Occluded Polyhedral Environments.* PhD thesis, Dept. of Computer Science, University of California, Berkeley, 1992.

[28] William C. Thibault and Bruce F. Naylor. Set operations on polyhedra using binary space partitioning trees. In *Proc. SIGGRAPH '87*, volume 21 of *Comput. Graph.*, pages 153–162, New York, 1987. ACM SIGGRAPH.

[29] C. Yap. Towards exact geometric computation. *Comput. Geom. Theory Appl.*, 1996. to appear.

[30] David M. Young. *Iterative solution of large linear systems.* Academic Press, New York, NY, USA, 1971.

[31] J. Yu. *Exact Arithmetic Solid Modeling.* PhD thesis, Purdue University, CS Dept., West Lafayette, IN 47907, USA, December 1991.

A Proof that M has an inverse

We prove that an inverse exists for any matrix M with weak diagonal dominance, i.e.,

$$\sum_{j,j\neq i} |M_{i,j}| \leq |M_{i,i}|,$$

for all i and there is a k such that

$$\sum_{j,j\neq k} |M_{k,j}| < |M_{k,k}|. \tag{2}$$

Assume that M has no inverse. Then the determinant $\det M = 0$. As a result, there exists a non-zero vector u such that $Mu = 0$ [30, Theorem 1.4]. Let i be the index such that $|u_i| = \max_j |u_j|$. Note that $|u_i| > 0$ since u is not a zero vector. Now, $Mu = 0$ implies that

$$M_{ii}u_i = -\sum_{j,j\neq i} M_{ij}u_j.$$

Using the diagonal dominance of M and taking absolute values in the above equation, we have

$$\begin{aligned}\Big(\sum_{j,j\neq i} |M_{ij}|\Big)|u_i| \leq |M_{ii}||u_i| &\leq \sum_{j,j\neq i} |M_{ij}||u_j| \\ &\leq \Big(\sum_{j,j\neq i} |M_{ij}|\Big)|u_i|,\end{aligned}$$

since $|u_i| = \max_j |u_j|$. As a result, all the inequalities above turn into equalities. Hence,

$$\sum_{j,j\neq i} |M_{ij}||u_j| = \Big(\sum_{j,j\neq i} |M_{ij}|\Big)|u_i|.$$

But $|u_i| = \max_j |u_j|$. Therefore,

$$|u_1| = |u_2| = \cdots = |u_n|. \tag{3}$$

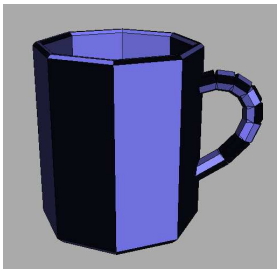
Solving for u_k in $Mu = 0$, where k is the row of M that causes M to be weakly diagonally dominant, we have

$$M_{kk}u_k = -\sum_{j,j\neq k} M_{kj}u_j.$$

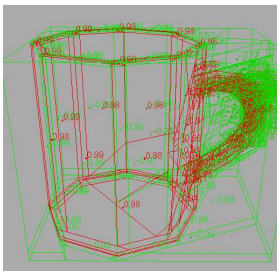
Taking absolute values in the above equation and using (2), we have

$$\begin{aligned}\Big(\sum_{j,j\neq k} |M_{kj}|\Big)|u_k| < |M_{kk}||u_k| &\leq \sum_{j,j\neq k} |M_{kj}||u_j| \\ &= \Big(\sum_{j,j\neq k} |M_{kj}|\Big)|u_k|, \qquad \text{by (3).}\end{aligned}$$

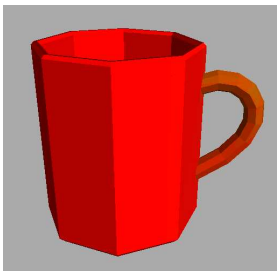
Therefore, $|u_k| < |u_k|$, which is a contradiction. Hence $\det M \neq 0$ and M has an inverse.



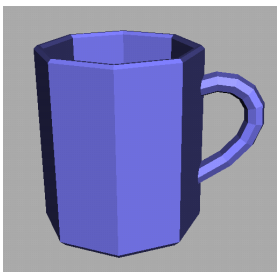
(a) Input model with inconsistencies.



(b) Subdivision cells labeled and outlined with colors based on their solidities.

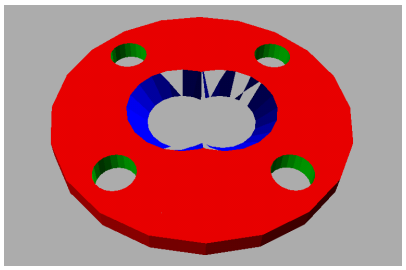


(c) Opaque boundaries between solid and free cells drawn with polygons oriented towards the model's exterior.

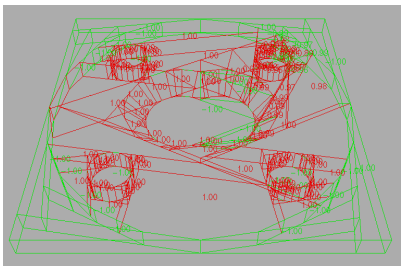


(d) Output model forming a manifold surface.

Plate I. Reconstructing solid model of a cup.

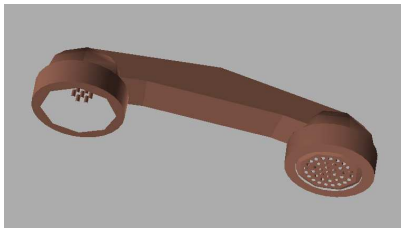


(a) Input model.

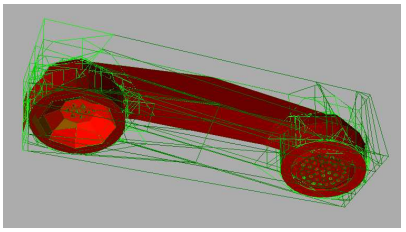


(b) Cell solidities indicated by numerical values.

Plate II. CAD/CAM: Automobile clutch with wrongly-oriented polygons (in the middle) and missing polygons (in the upper-right circular hole).

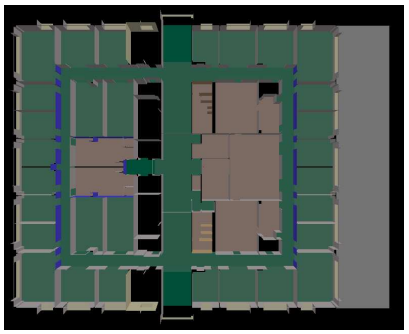


(a) Input model.

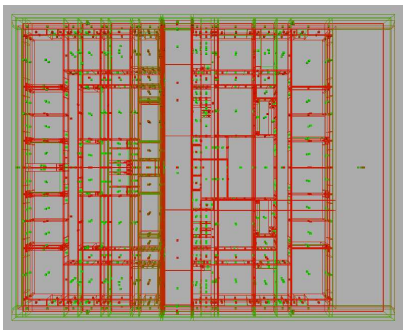


(b) Opaque polygons drawn between solid and free cells.

Plate III. Visualization: Telephone headset containing wrongly-oriented polygons (in the mouth and ear pieces) and intersecting polygons (along the handle).



(a) Input model.



(b) Cell solidities indicated by colored dots.

Plate IV. Architectural CAD: One floor of Soda Hall with numerous topological inconsistencies and other modeling errors.