THEME ARTICLE

Using Geometric Hashing To Repair CAD Objects

GILL BAREQUET
Johns Hopkins University

Problems in CAD software sometimes cause defects in the boundaries of polyhedral objects—small gaps bounded by edges incident to one polyhedron face. Using geometric hashing, algorithms can resolve this problem, which has vexed layered manufacturing. Similar techniques hold promise for solving problems in such areas as computer vision, medical imaging, and molecular biology.

Geometric hashing was first introduced in the context of object recognition, an important area of robotics and computer vision dealing with such things as robot task and motion planning and automatic image understanding and learning.[1,2] In a two- or three-dimensional image of a scene, certain objects—some perhaps only partially visible—need to be identified, both for their position and for their orientation in the scene. Many approaches to object recognition have been developed, including pose clustering[3] (also known as transformation clustering and generalized Hough transform), subgraph isomorphism,[4] alignment,[5] iterative closest point,[6] and many indexing techniques—including geometric hashing.

Geometric hashing is a model-based technique for object recognition, identifying objects using prior knowledge about them stored in a database (see the tutorial on p. 10). For the CAD software problem described here, geometric hashing is used as in object recognition: to detect the similar features of two entities. In this case, the goal is not object recognition, but the identification of similar portions of curves in three dimensions. Finding these similarities between curves is only part of the solution. Finding the remaining, nonsimilar parts is accomplished in a postprocessing step.

Defective CAD objects are an acute problem for systems that rely on the global continuity and consistency of the object boundary, such as finite element analysis algorithms and various manufacturing processes. Often the object input to the system is a processed version of the original object, and it is not practical to correct the design of the original object. Therefore, a repairing algorithm—applied to the processed version of the object—is required.

Geometric hashing

Automatic object recognition has greatly benefited from the partial-curve matching technique, first suggested by Kalvin et al.[7] This technique—which uses geometric hashing—solved the curve-matching problem in the plane, assuming that one curve is a proper subcurve of the other. That is, given two curves in the plane such that one is a proper, though slightly deformed, subcurve of the other, the technique finds the translation and rotation of the subcurve that yields the best least-squares fit to the appropriate portion of the larger curve.

This technique has been extended for use in computer vision, allowing automatic identification of partially obscured objects in two or three dimensions. It uses geometric hashing as a recognition technique to identify partial curve matches between an input scene boundary and a preprocessed set of known object boundaries.[8-10] Through this, it determines the objects participating in a scene and computes their position and orientation.

Put simply, geometric hashing is a model-based recognition technique that can efficiently identify partially occluded objects with objects stored in a model database. It is based on an offline model preprocessing step where model features are encoded through some transformation-invariant function and stored in a hashing table, thus facilitating a particularly efficient subsequent online recognition step. Geometric hashing is a general technique that can be applied to recognition tasks in any dimension under different classes of transformation. It has three basic characteristics:

1. Object features are represented using transformation invariants, allowing recognition of an object that is subject to any allowed transformation.
2. These invariants are stored in a hashing table, allowing efficient matching that is nearly independent of the complexity of the model database.
3. A robust matching scheme guarantees reliable recognition even with relatively small overlap and the presence of considerable noise.

The original partial-curve-matching version of geometric hashing (which later will be shown useful for repairing gaps) found partial matches between curves in the plane. It used a database of preprocessed curves and found matches between portions of a composite query curve and portions of the curves in the database, subject to a rigid motion in the plane.

In the preprocessing step, the system generates, encodes, and stores in the database features of all the curves. It scans each curve, with *footprints* (or invariant values under the class of allowed transformations; in this case rigid motions) generated at equally spaced points along the curve, each point labeled by its sequential number (proportional to the arc length) along the curve. The footprint is chosen so that it is invariant under the rigid motion of the curve. For example, a typical footprint choice is the second derivative (with respect to arc length) of the curve function—that is, the change in the direction of the tangent line to the curve at each point. Each footprint is a key to a hashing table, which records the curve and the label of the point at which the footprint was generated.

With respect to the total number of sample points on the curves stored in the database, the time complexity of the preprocessing step is linear. Since the processing of each curve is independent of the others, this technique can process curves in parallel. Moreover, it can add or delete curves in the database without recomputing the entire hashing table.

In the recognition step, a voting scheme determines which portions of curves stored in the database match portions of the query curve and with which shifts of the label sequences. The system scans the query curve and computes the footprints at equally spaced points, with the same discretization parameter as the preprocessed curves. It finds the appropriate entry for each footprint in the hashing table and retrieves all the curve/label pairs stored there. Each pair contributes one vote for the model curve and for the relative shift, or difference in labels of matched points, between this curve and the query curve. That is, if a footprint of the *i*th sample point of the query curve is close enough to the footprint of the *j*th point of model curve *c*, then we add one vote to the curve *c* with the relative shift $j - i$. In order to tolerate small deviations in the footprints, the system fetches not only entries with the same footprints as the points along the query curve, but also entries within some small neighborhood of the image footprint.

The goal is not object recognition, but the identification of similar portions of curves in three dimensions.

The voting mechanism rests principally on the assumption that real matches between long portions of curves result in a large number of footprint similarities—and hence votes—between the appropriate model curve and the query curve, giving almost identical shifts. By the end of the voting process, curve/shift pairs with the most votes are identified, and for each pair the approximate endpoints of the matched portions of the model and query curves are determined.

It is then straightforward to compute the rigid transformation between the two curves, with accuracy that increases with the length of the matched portions. The running time of the matching step is, on average, linear in the number of sample points generated along the query curve. This is based on the assumptions that on average each access to the hashing table requires a constant time, and that the output of the corresponding range-searching query has a constant size. Thus, the expected running time of this step does not depend on the number of

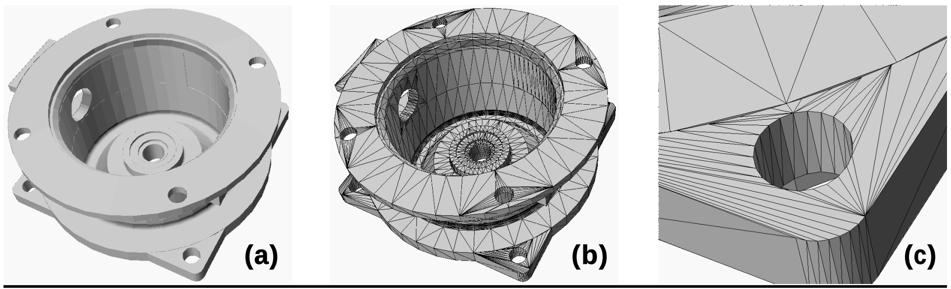

Figure 1. A real CAD example: (a) smooth display; (b) faceted display; (c) zoom view of a problematic region in the bottom-right area of the original image file. The "ear" doesn't match the body of the object. (Images courtesy of Cubital Ltd.)

curves stored in the database or on the total number of points on the curves.

There have been many variations of the geometric hashing technique, including different choices of allowed transformation, different applications (for example, locating an object in a raster image, registration of medical images, and molecule docking), and generalizations to higher dimensions. In any situation, the key to success is in defining a good footprint so that true votes outnumber false votes. In practice, all applications of geometric hashing have their own special footprint setting, which strongly depends on the nature of the problem.

Repairing CAD objects

In polyhedral approximations of CAD objects, whose boundaries are described using curved entities of higher levels, gaps often appear in the boundary of the polyhedron.[11-14] In constructive solid geometry, an object is described as the unions and intersections of spheres, cones, and other 3D shapes, but when the object's boundary representation is used, it is described by such things as nonuniform rational bisplines and Bézier surfaces.

Gaps are caused by missing surfaces, by incorrect handling of adjacent patches within a surface, or most commonly by incorrect handling of trimming curves, a CAD feature defined by the intersections of adjacent surfaces. The vertices of the polyhedral approximation along an intersection curve between those two adjacent surfaces are often computed separately along each of the curve's two sides, thus creating two different copies of the same curve and causing a gap between them. In the simple case, different point sets may still use the same curve equation. In the more complicated case, mesh point evaluations use different equations of the curve, one for each surface. Figure 1 shows a real CAD object, displayed with both smooth and faceted surfaces, and a close-up view of a gap.

Invariably these approximation errors cause edges in the boundary of the resulting polyhedron that are incident to only one face—not the two faces of a valid representation—thus creating gaps in the boundary and making the resulting representation invalid. Such gaps may disconnect parts of the boundary from other parts or may create small holes bounded by a cycle of invalid edges. (The parts that are connected usually result from the polyhedral approximation of a single surface or part of a solid in the original representation.) This problem does not usually disturb graphics applications, where the gaps between the surfaces are often too small to be seen or are handled straightforwardly. However, it may cause severe problems in applications that rely on the continuity of the boundary, such as finite-element analysis[15] and rasterization algorithms.[16]

Such gaps arise in almost every sufficiently large CAD file, so their detection and elimination is indeed a rather acute practical problem. Sheng and Tucholke[11] referred to these errors as one of the most severe software problems in layered manufacturing. Many authors, such as Dolenc and Mäkelä,[14] and Sheng and Hirsch,[17] have described attempts to avoid the gap problem already in the surface-fitting triangulation process. Previous attempts to solve the problem, based only on the polyhedral description of an object, used local information and did not check for global consistency violations. Bøhn and Wozny treated only local gaps by iteratively triangulating them, eliminating at each step the vertex that spans the smallest angle with its two neighboring vertices.[12] Similarly, Mäkelä and Dolenc used a minimum-distance heuristic to locally fill cracks in the boundary of the polyhedron.[13] Barequet and Kumar merged vertices of the polyhedron to close gaps in its boundary.[18]

A geometric hashing approach

Another approach, however, would be an algorithm that uses geometric hashing to fill gaps.[19] The main problem is to stitch together the borders—the collection of cycles of invalid edges on the polyhedron boundary—adding new faces to close the gaps, thus making the resulting polyhedron valid. The algorithm adds new faces by connecting points along the same or different borders, following these steps:

1. Identify matching portions of the borders.
2. Choose the best consistent set of matches.
3. Construct new facets, or planar faces, and connect them.
4. Fill the remaining holes by triangulation.

An alternative approach would be to replace each pair of matching portions of borders by a single polygonal curve.

To identify matching portions of borders, the geometric hashing algorithm uses a simplified version of the partial-curve matching technique, mentioned earlier. This version matches 3D curves, but does not allow motion of one curve relative to the other. Since the scope of the problem is wider than object recognition, the information obtained from matching is further processed. The results are used to repair most of the defects, and then 3D triangulation is used to close the remaining holes.

To match border portions, the algorithm discretizes each border polygon into a cyclic sequence of points. This is accomplished by choosing some sufficiently small arc-length parameter s and generating equally spaced points at distance s from each other along the polygon boundary. As in object recognition, the resulting discretization can be considered footprints of the borders.

Because footprints in this algorithm are the 3D coordinates of the points, the two parts of the original object boundary—which should have shared a common polygonal curve, but were split apart in the approximation—must naturally have similar sequences of footprints along their common curve (unless the approximation was very bad). The goal then is to search for pairs of sufficiently long "similar" subsequences. In this approach, two subsequences, $(p_i, \ldots, p_{i+\ell-1})$ and $(q_j, \ldots, q_{j+\ell-1})$, are said to closely match each other if, for some chosen parameter $\varepsilon > 0$, the number of indices k for which $||p_{i+k} - q_{j+k}|| \le \varepsilon$ is sufficiently close to ℓ. The algorithm then performs the following voting process, where votes are given to good point-to-point matches. The borders are given as cyclic ordered sequences of vertices, and each cycle is broken at an arbitrarily chosen vertex. Also, the direction of borders is implied by the chosen orientation of their connected component in the polyhedron boundary. (Had it been chosen the other way, the border direction would have been reversed.) A match between two border subsequences is *direct* when the sequences of vertex indices of both borders are both increasing or decreasing. A match is *inverted* when one of the sequences is increasing, but the other is decreasing.

Adjacent components whose orientations are consistent should have an inverted match, which if accepted gives the combined component the same orientation as its subparts, or the opposite of these orientations. A direct match implies that the orientations of the two components are not consistent, so if the match is accepted, all the facets of just one of the components should invert their orientation before the two components are glued together. If two border portions that bound the same component are matched, then only inverted matches are acceptable because otherwise the component cannot be oriented after gluing.

In order to efficiently locate all vertices that lie in some ε-neighborhood of vertex v, all border vertices are preprocessed—either by using a range-tree data structure[20] or by fractional cascading[21]—for range searching. The ε parameter is not a function of the input, but rather an a priori estimate of the physical size of the gaps between the original surfaces. An estimate that is too small or too large may lower the algorithm's performance. If it is too small, close points will not be matched, keeping border matches from being found. If it is too large, too many false votes will cause correct border matches to be lost among too many incorrect matches.

Gaps may cause severe problems in applications that rely on continuous boundries, such as finite-element analysis and rasterization algorithms.

The positions along border sequence b, whose length is ℓ_b, are numbered from 0 to $\ell_b - 1$. Assume that the querying vertex v is in position i of border sequence b_1. Then, each vertex retrieved by the query, which is in position j in border sequence b_2, contributes a vote for a direct match between borders b_1 and b_2 with the shift $(j - i)$ (mod ℓ_b) and a vote for a match between the borders $\bar{b}_1$ (the inverted b_1) and b_2 with the shift $(j - (\ell_{b_1} - 1 - i))$ (mod ℓ_{b_2}), the inverted match between the borders b_1 and b_2. As noted earlier,

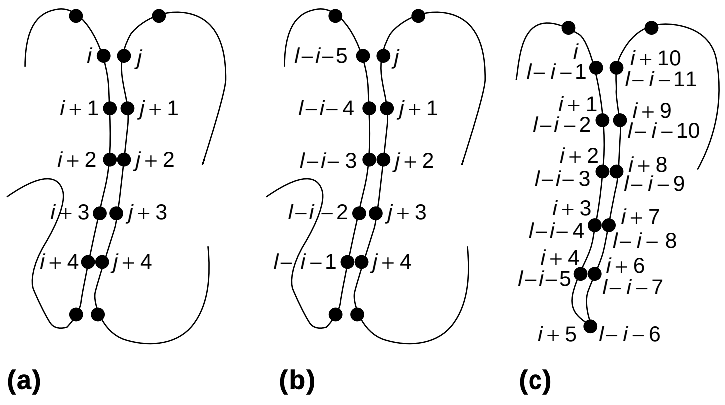

Figure 2. Matches between borders: (a) a direct match, (b) an inverted match, and (c) a match between a border and itself.

only inverted matches are allowed between two portions of the same border or between borders that bound the same component.

All these cases are illustrated in Figure 2. The match shown in Figure 2a is direct, hence the orientation of one of the components is inverted and the corresponding shift is $(j - i)$. The match in Figure 2b is inverted, hence the two involved components are consistent, and the corresponding shift is $(j - \ell + i + 5)$, where the indices of the left border already indicate that it is inverted. Finally, the match in Figure 2c is between a border and itself, where the shift, when inverting the left portion, is $(2i - \ell + 11)$.

Obviously, matches between long portions of borders are indicated by a large number of votes for the appropriate shift between the matching borders. Since small mismatches might occur between the two portions of the matching borders, or the arc length along one portion might not exactly coincide with the arc length along the other, a real match most likely will be shown by a significant peak of a few successive shifts in the graph plotting the number of votes between two borders (or possibly the same one) as a function of the mutual shift.

There may, as well, be several peaks in the same graph, which implies that there are several good matches between the same pair of borders, but with different alignments. For each peak, the portions of the borders voting for this alignment help find the endpoints of the corresponding match or matches between the two borders. Based on the neighborhood of the peak, the matches are extended as much as possible, allowing, up to specified limits, sporadic mismatches, insertions, or deletions. In almost all cases, small portions of the borders are included in more than one candidate match. This happens when several borders occur in close locations, usually at the junction of three or more gaps. The algorithm simply eliminates those portions common to more than one significant candidate match.

Repairing the object

Each match is then given a score, which may reflect not only the number of votes for the appropriate shift, but also the Euclidean length of the match and its quality, which is measured by the closeness of the vertices on the match's two sides. Now the algorithm chooses a consistent subset of the collection of suggestions of partial border matches with maximum scores. This step turns out to be NP-hard,[19] so the algorithm implements it by using a standard approximation scheme.

Next the gaps in the boundary of the polyhedron are filled. Each pair of border portions that have been matched in the above step and whose match has been accepted are stitched together with new triangles, which are consistently oriented with the facets along the borders.

Finally, the remaining holes, which usually appear at the junctions of several matches, are identified and triangulated using a 3D minimum-area triangulation technique. This technique is similar to the dynamic programming triangulation of simple polygons developed by Klincsek.[22]

The entire algorithm has been implemented and tested on dozens of CAD files, mostly for automotive parts, whose boundaries contained gaps. The tuning of the parameters was robust, and the algorithm obtained good results in most cases. The voting threshold ε was usually set to 0.5 for objects whose global size was 2 to 20 cm in all dimensions.

As noted earlier, the value of ε has a crucial effect on the success of the algorithm. Although there was in most cases a large enough range of valid ε settings, the algorithm failed more often if this parameter was chosen improperly. Values of ε on the margins of this range degraded the significance of the peak in the voting graph. An ε that was too small resulted in the loss of matches due to the lack of votes. An ε that was too large resulted in a noisy voting table, in which "intuitive" matches were not given sufficiently larger scores than incorrect match suggestions. Even when these intuitive matches were identified, they overlapped with erroneous match suggestions and were therefore eliminated. In all cases, too many unmatched border portions were passed to the next step of the algorithm, causing its failure.

Figure 3 shows a CAD object that underwent gap filling. Figure 3a shows the entire object, and Figure 3b shows the gaps in the object. Figures 3c and 3d are close-ups of two pairs of borders (pointed to by an arrow in Figure 3b) before and after stitching, respectively. Figure 4 shows the graph that plots the number of votes and the scores, as a function of the mutual shift, of matches between the two borders of the upper circular pair, as seen in Figure 3b.

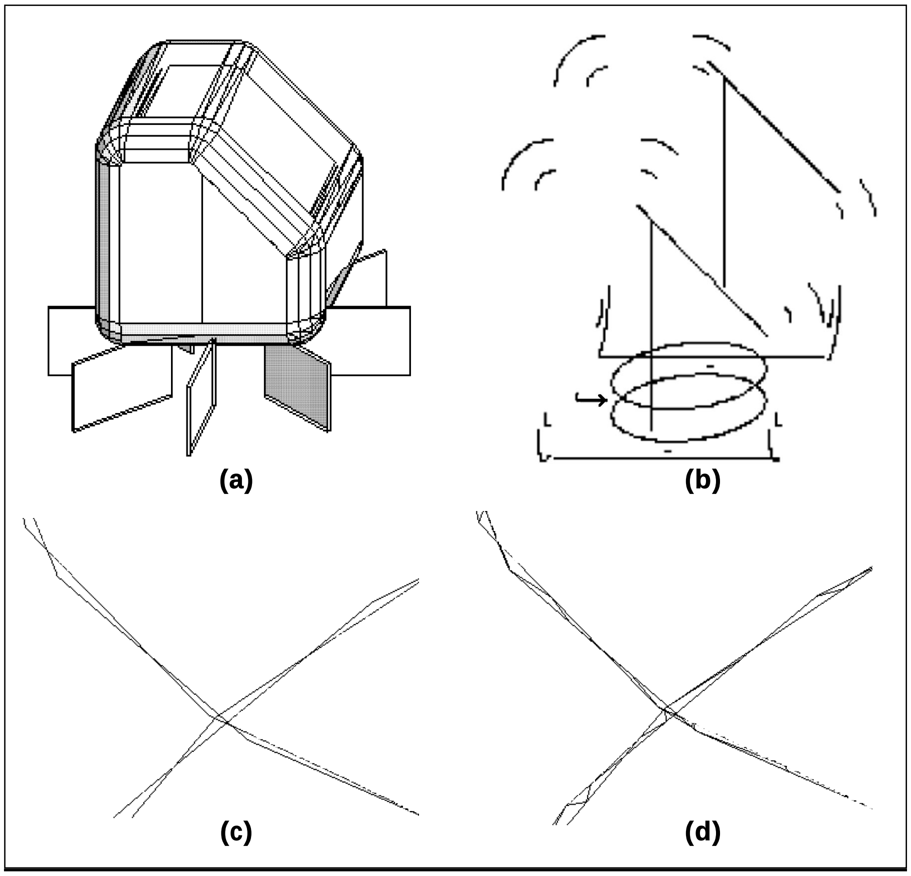

Figure 3. Repairing a CAD object, showing (a) the original object, (b) identified gaps, (c) the borders of the object before stitching, and (d) the borders of the object after stitching.

This algorithm, using geometric hashing to repair CAD objects, is significant because it is the first algorithm to solve the acute problem of defective CAD objects, combining robust detection of matching portions of the gaps with maintenance of global consistency of the boundary orientation. The technique holds promise for such problems as object recognition in computer vision, the interpolation between scanned cross-sections of human organs in medical imaging, and protein docking in molecular biology—wherever there is a need to find a partial geometric fit between a priori unknown shapes. ♦

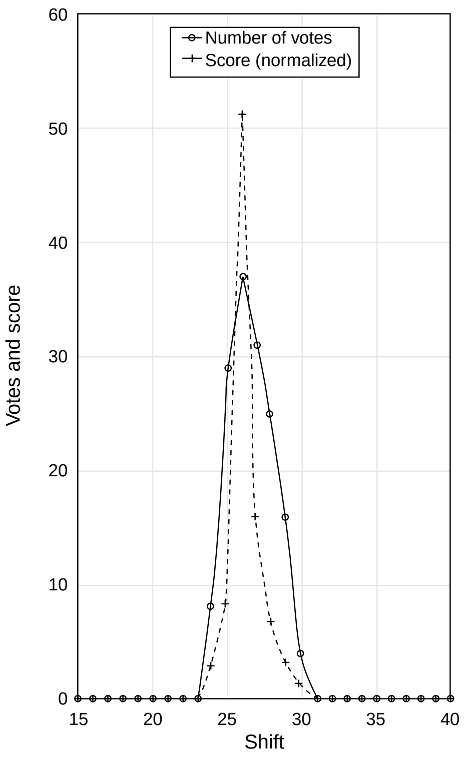

Figure 4. This voting graph plots the number of votes and the scores of matches between the borders of the upper circular pair in Figure 3b.

Acknowledgments

The author thanks Micha Sharir and Haim Wolfson for introducing him to the fascinating geometric hashing technique. Work on this paper has been supported in part by the US Army Research Office under grant number DAAH04-96-1-0013.

References

1. P.J. Besl and R.C. Jain, “Three-Dimensional Object Recognition,” *ACM Computing Surveys,* Vol. 17, No. 1, Mar. 1985, pp. 75–154.
2. R.T. Chin and C.R. Dyer, “Model-Based Recognition in Robot Vision,” *ACM Computing Surveys,* Vol. 18, No. 1, Mar. 1986, pp. 67–108.

3. G. Stockman, "Object Recognition and Localization Via Pose Clustering," *Computer Vision, Graphics, and Image Processing,* Vol. 40, No. 3, Dec. 1987, pp. 361–387.
4. R.C. Bolles and R.A. Cain, "Recognizing and Locating Partially Visible Objects: The Local-Feature-Focus Method," *Int'l J. Robotics Research,* Vol. 1, No. 3, Fall 1982, pp. 637–643.
5. D.P. Huttenlocher and S. Ullman, "Recognizing Solid Objects by Alignment with an Image," *Int'l J. Computer Vision,* Vol. 5, No. 2, Nov. 1990, pp. 195–212.
6. P.J. Besl and N.D. McKay, "A Method for Registration of 3-D Shapes," *IEEE Trans. Pattern Analysis and Machine Intelligence,* Vol. 14, No. 2, Feb. 1992, pp. 239–256.
7. A. Kalvin et al., "Two-Dimensional, Model-Based, Boundary Matching Using Footprints," *Int'l J. Robotics Research,* Vol. 5, No. 4, Winter 1986, pp. 38–55.
8. H.J. Wolfson, "On Curve Matching," *IEEE Trans. Pattern Analysis and Machine Intelligence,* Vol. 12, No. 5, May 1990, pp. 483–489.
9. Y. Lamdan, J.T. Schwartz, and H.J. Wolfson, "Affine Invariant Model-Based Object Recognition," *IEEE Trans. Robotics and Automation,* Vol. 6, No. 5, Oct. 1991, pp. 578–589.
10. E. Kishon, T. Hastie, and H. Wolfson, "3-D Curve Matching Using Splines," *J. Robotic Systems,* Vol. 8, No. 6, Oct. 1991, pp. 723–743.
11. X. Sheng and U. Tucholke, "On Triangulating Surface Model for SLA," *Proc. Second Int'l Conf. Rapid Prototyping,* Univ. of Dayton, Dayton, Ohio, 1991, pp. 236–239.
12. J.H. Bøhn and M.J. Wozny, "Automatic CAD-Model Repair: Shell-Closure," *Proc. Symp. Solid Freeform Fabrication,* Univ. of Texas, Austin, Texas, 1992, pp. 86–94.
13. I. Mäkelä and A. Dolenc, "Some Efficient Procedures for Correcting Triangulated Models," *Proc. Symp. Solid Freeform Fabrication,* Univ. of Texas, Austin, Texas, 1993, pp. 126–134.
14. A. Dolenc and I. Mäkelä, "Optimized Triangulation of Parametric Surfaces," *Computer-Aided Surface Geometry and Design,* A. Bowyer, ed., Clarendon Press, Oxford, U.K., 1994, pp. 169–183.
15. K. Ho-Le, "Finite Element Mesh Generation Methods: A Review And Classification," *Computer-Aided Design,* Vol. 20, No. 1, Jan. 1988, pp. 27–38.
16. J.D. Foley and A. van Dam, *Fundamentals of Interactive Computer Graphics,* Addison Wesley Longman, Reading, Mass., 1984.
17. X. Sheng and B.E. Hirsch, "Triangulation of Trimmed Surfaces in Parametric Space," *Computer-Aided Design,* Vol. 24, No. 8, Aug. 1992, pp. 437–444.
18. G. Barequet and S. Kumar, "Repairing CAD Models," *Proc. IEEE Visualization '97,* ACM, New York, 1997, pp. 363–370.
19. G. Barequet and M. Sharir, "Filling Gaps in the Boundary of a Polyhedron," *Computer-Aided Geometric Design,* Vol. 12, No. 2, Mar. 1995, pp. 207–229.
20. K. Mehlhorn, *Data Structures and Algorithms 3: Multi-Dimensional Searching and Computational Geometry*, Springer-Verlag, Berlin, 1984.
21. B. Chazelle, "A Functional Approach to Data Structures and Its Use in Multidimensional Searching," *SIAM J. Computing,* Vol. 17, No. 3, June 1988, pp. 427–462.
22. G.T. Klincsek, "Minimal Triangulations of Polygonal Domains," *Annals of Discrete Mathematics,* Vol. 9, 1980, pp. 121–123.

Gill Barequet is a postdoctoral fellow at the Department of Computer Science of Johns Hopkins University. His research interests include discrete and computational geometry, interpolation and reconstruction algorithms, geometric software and computing over the Internet, and geometric applications in CAD, medical imaging, and molecular biology. He holds five US patents in related areas. He received a BS in mathematics and computer science in 1985, an MS in 1987, and a PhD in 1994 in computer science, all from Tel Aviv University.

Barequet can be reached at the Center for Geometric Computing, Dept. of Computer Science, Johns Hopkins University, 3400 N. Charles St., Baltimore, MD 21218; e-mail, barequet@cs.jhu.edu; WWW, http://www.cs.jhu.edu/~barequet.