Proceedings of DETC'01
2001 ASME Design Engineering Technical Conferences
September 9-12, 2001, Pittsburgh, Pennsylvania, USA

DETC2001/DAC-21068

FINDING AND REMOVING FEATURES FROM POLYHEDRA

José Ribelles
Dep. de Lenguajes y Sistemas Informáticos
Universitat Jaume I
Campus de Riu Sec, E-12080
Castellón, Spain
http://nuvol.uji.es/~ribelles/

Paul S. Heckbert*
Computer Science Dept.
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213-3891
http://www.cs.cmu.edu/~ph/

Michael Garland
Dept. of Computer Science
University of Illinois
1304 West Springfield Ave
Urbana, Illinois 61801
http://graphics.cs.uiuc.edu/~garland/

Tom Stahovich
Dept. of Mechanical Engineering
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213-3890
http://www.me.cmu.edu/faculty1/stahovich/

Vinit Srivastava
Dept. of Mechanical Engineering
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213-3890
vinit@andrew.cmu.edu

ABSTRACT

Geometric models of solids often contain small features that we would like to isolate and remove. Examples include bumps, holes, tabs, notches, and decorations. Feature removal can be desirable for numerous reasons, including economical meshing and finite element simulation, analysis of feature purpose, and compact shape representation. In this work, an algorithm is presented that inputs a polyhedral solid, identifies and ranks its candidate features, and outputs solid models of the feature and the original object with the feature removed. Ranking permits a user or higher level software to quickly find the most desirable features for the task at hand. Features are defined in terms of portions of the surface that are classified differently from the rest of the solid's surface with respect to one or more split planes. This approach to feature definition is more general than many previous methods, and generalizes naturally to quadric surfaces and other implicit surfaces.

Keywords: feature removal, defeaturing, surface simplification, hole filling

*Address all correspondence to this author.

1 Introduction

Most geometric shapes have features whose removal can facilitate mathematical modeling. For example, removal of bolt heads, rivets, holes, or tabs from a solid model can facilitate finite element analysis. Their removal will often cause negligible error to a structural analysis, for example, while reducing the number of mesh elements and speeding the analysis significantly. Feature identification and removal is also useful in reverse engineering.

Both for design conceptualization and for manufacturing, it is commonplace to represent solid objects using constructive solid geometry (CSG), describing complex shapes as nested unions or differences of a big "body" part and a small "feature" part.

If we are given a boundary representation for a detailed solid shape, and no CSG model is not available, often we would like to find its features and to derive a CSG representation. Such a CSG model is not unique, and often a user prefers to be "in the loop" in the search for a good CSG model, since human designers have application-specific definitions of the features of interest.

In this paper we present a general technique for feature identification and removal that is applicable to polyhedral objects and is amenable to interactive applications.

Copyright © 2001 by ASME

2 Background

When describing or analyzing shapes, it is quite natural for us to conceive of them as some base shape together with some set of smaller features which modulate the underlying shape. For instance, most users would probably describe the object in Figure 3 as a box with a box-shaped protrusion on top of it. For this reason, and because it often simplifies automated analysis, there has been significant interest over the years in automatic methods for segmenting and identifying individual features from object models.

Feature extraction is the problem of particular concern for us. This is the process by which features are separated from the rest of the model. A related problem is feature classification, which is concerned with classifying the resulting features based on their form into protrusions, holes, slots, and so on. We will not address the problem of classification, although it might be a post-process operating on the results of feature extraction.

2.1 Representation

We will assume that we are given a 2-manifold[1] M that is composed of a set of vertices V and a set of polygonal faces F. For the sake of simplicity, and without loss of generality, we will assume that the set of faces F consists solely of triangles. While in general M may be an open surface (i.e. a manifold with boundary) we will focus on the case where M is a closed surface, defining the boundary of a solid model. To avoid ambiguity in our discussion, we will use the term *facet* to refer to any maximal, connected planar region of the surface. Thus, a cube always has exactly 6 facets, but may be represented using any number of triangular faces. It is important to keep in mind that essentially all prior methods discussed in the next section assume that the surface is decomposed into faces which are identical to its facets, thus making the two terms interchangeable.

A common construction used in feature extraction, and one which we will use in later sections, is the *dual graph*, or *edge–face graph*, of the surface M. This dual is constructed by creating a node for every face of M and connecting every pair of nodes whose corresponding faces are adjacent on the surface.

2.2 Feature Extraction

Before reviewing possible methods for performing feature extraction, we must consider how to define what we mean by a "feature". An often cited definition, due to Pratt and Wilson [12], is that "a feature is a region of interest on the surface of a part". While quite vague, and hence almost universally applicable, this definition does not help us in forming an operational definition for use in a computational process. Instead, we propose the following general definition: A feature is a connected region of a surface that can be easily separated from the rest of the surface. This definition fits well with many of the actual algorithms that have been developed to perform feature extraction. We will now review the feature extraction methods which are most relevant to our work. More extensive surveys of related methods are available elsewhere [15].

2.3 Volume Decomposition

Kim [11] generates volume features using a convex decomposition method called the alternating sum of volumes with partitioning (ASVP). This method is an extension of an earlier approach called alternating sum of volumes (ASV) decomposition [14]. The procedure is iterative. In each iteration, the polyhedron is subtracted from its convex hull, using a reguralized difference operator. The result is taken as the polyhedron for the next iteration. When the process converges, the resulting shape is a feature. Sometimes the iterations do not converge, and the polyhedron is partitioned into pieces, each of which is then decomposed with ASV and further partitioning, if necessary. Their method is best at isolating indented features. The method we describe works well for both indented and protruding features.

Sakurai [13] presents an approach for identifying form features by decomposing a polyhedron into maximal convex cells (MCCs). A polyhedron can be decomposed into convex cells by intersecting the polyhedron with the half spaces of its faces having concave edges. A maximal convex cell is a convex cell whose half spaces are those of the polyhedron and that is not totally included by any other such cell. Individual features are obtained by subtracting various MCCs from each other. Currently this method constructs only convex features, however, Sakurai, suggests that it is possible to produce concave features by considering combinations of MCCs rather than differences. Our approach produces both convex and concave features.

The primary difference between Sakurai's approach and ours, is the type of features produced. Sakurai's approach produces features that are decompositions of the volume, and thus the approach does not directly identify negative features such as pockets. To identify negative features, it is first first necessary to select a shape for the raw material from which the part is made. A delta volume is obtained by subtracting the part from the raw material. Negative features are then identified by decomposing the delta volume. Our approach directly identifies both positive and negative features. Furthermore, identifying negative features with our method does not require a delta volume, but instead existing part faces are used to define the volumes of the features.

2.4 Loop-Based Methods

One common approach to feature extraction is to select a set of one or more edge loops which separate features from the base shape. Selection of these separating loops may be based on some simple criterion, such as requiring that they be composed solely

[1] Recall that a manifold is a surface all of whose points have a neighborhood which is homeomorphic to a disk (or half-disk in the case of manifolds with boundary).

Copyright © 2001 by ASME

of concave edges. Other more complex criteria have also been proposed.

Gadh and Prinz [4, 6, 5] describe a technique that recognizes feature by identifying loops consisting entirely of “positive” or entirely of “negative” edges. The positive edges are obtained by viewing the solid from a variety of viewing directions and identifying those edges that are adjacent to both a visible and a hidden face (i.e., convex edges). The negative edges are obtained by performing the same operations on the complement of the solid. They define five fundamental feature classes on the basis of the loops that are identified. For example, a blind depression is defined by a single positive loop, while a through depression has two positive loops. Because features are defined by *closed* loops, the approach cannot identify certain classes of features. For example, if a small cube is on top of a large cube, and the two cubes have a vertical face in common, the small cube will not be identified as a protrusion. There is no closed loop at the base because there is no edge separating the two cubes along the common face.

Lu *et. al.* [16] extend this approach to handle certain features that do not have closed loops of positive or negative edges. They allow a loop to simultaneously contain convex, concave, hybrid, and neural edges. Hybrid edges are convex in some regions and concave in others. Neutral edges are created by extending parts faces and intersecting them with other faces. There are some restrictions on the allowable combinations of edge types within a single loop, thus there are likely to be features that our method will find and their method will not. Furthermore, there approach for filling the hole in the surface that results when a feature removed is less general than ours.

2.5 Connected Components Methods

As an alternative, others have suggested partitioning the set of faces directly into disjoint connected subsets, which we will term *clusters*. Separating loops are thus implicitly defined by the boundaries between adjacent clusters, and we are thus freed from tracking their many interrelationships. Other authors work with the *dual graph* – the graph of faces and their adjacencies. De Floriani [3] proposed cutting the dual graph into biconnected and triconnected components [2], which has the effect of partitioning the set of faces into clusters. Note that this is a purely topological approach; the geometry of the surface is merely implicit in its decomposition into planar faces. Gavankar and Henderson [9] explore heuristics that attempt to accelerate the decomposition into biconnected components. These methods can isolate a given feature as a single cluster provided that it meets the base shape in only 1 or 2 faces. Features with higher degrees of connectivity cannot be extracted without using additional methods.

2.6 Other Methods

In the last decade several methods, surveyed in detail elsewhere [1, 7, 10], have been developed for simplifying polygonal surfaces. When given a triangulated surface, these simplification systems attempt to produce an approximation containing fewer triangles which is nevertheless as similar to the original surface as possible. However, existing simplification methods are not suitable for the task of feature extraction. Since preserving surface shape is generally their primary goal, they tend to approximate rather than cleanly remove features. More importantly, while they may produce a resulting surface with some set of features (more or less) removed, they do not provide any representation of these removed features.

In the following sections we present a new algorithm for feature extraction. It combines an emphasis on face partitioning, thus avoiding some of the shortcomings of loop-based methods, with geometric separators, thus being sensitive to the object’s shape rather than merely its connectivity.

3 Feature Removal

Our approach to feature definition is explained first intuitively, in terms of simple examples, and then presented more precisely.

3.1 Feature Removal Examples

In this work, we define a “feature” of a polyhedral solid using a classification of space relative to one or more split planes. A single oriented plane separates space into three regions: IN, ON, and OUT corresponding to inside relative to the plane, on the plane, and outside, respectively. Figure 1 shows how a two-dimensional shape can be decomposed, using a split line, into the union of a simpler “body” and a feature. We draw the ON region with exaggerated thickness for clarity. Note that, in this case, all of the feature edges are OUT, and all of the body edges are classified as IN or ON.

This first approach to feature definition has limitations, however. Many features of interest cannot be cut off with a single line or plane, (see figure 2a, for example). More troubling, if the feature is a pit and not a bump, both the feature and body polygons are classified as IN, so splitting and classification alone are not sufficient to identify such a feature.

These difficulties are easily overcome, however. The use of multiple split planes permits features on edges and corners to be defined. And to isolate indentated features, we use adjacency information. Both of these generalizations are illustrated in the two-dimensional example of figure 2. We describe the algorithm in detail later in the paper; for now we concentrate on intuition. Here, the clearest feature is the small rectangle that has been subtracted from the bigger rectangle (there are others, but let’s concentrate on this one for now). This feature can be defined by

Copyright © 2001 by ASME

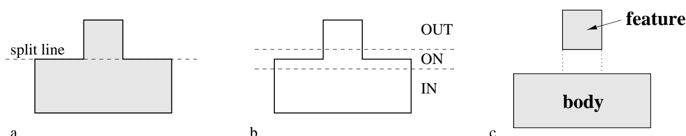

Figure 1. Simple example of splitting for feature removal: a small rectangular feature on a bigger rectangle. a) original object, b) classified by one split line, c) feature and body exploded.

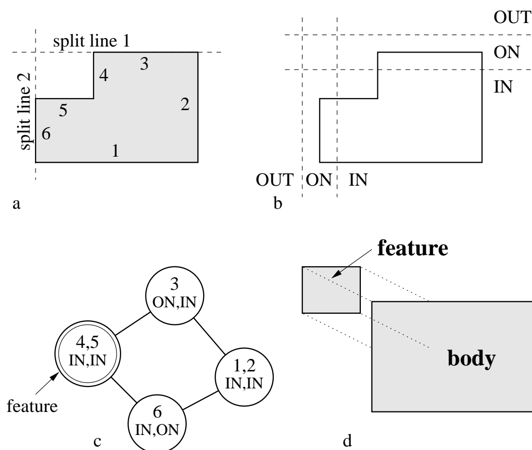

Figure 2. Removal of a more complex feature in 2-D: A small rectangular feature subtracted from a big rectangle. a) original, b) classified by two split lines, c) cluster graph, showing the set of edges stored with each node and its class label, d) feature and body exploded.

splitting with the two lines defined by edges 3 and 6. This classifies edges 1, 2, 4, and 5 as (IN, IN) – (IN with respect to both line 1 and line 2). Edge 3 is (ON, IN) and edge 6 is (IN, ON). Now we introduce the central data structure of our method. If we collect all edges that are classified identically and also adjacent into a cluster, we get the *cluster graph* of Figure 2c. Each node stores a set of edges and a class label. Edges 1 and 2 form a cluster because they are both (IN, IN), but none of their neighbors (edges 6 and 3) are, and similarly for edges 4 and 5.

Note that the cluster graph combines geometric classification information and topological adjacency information. Neither of these two types of information is by itself sufficient to isolate the feature, but their combination is.

Every node of the cluster graph defines a potential feature (some invalid). In this example, the feature we have chosen corresponds to the node labeled “4,5”. Solid models of the feature and body can be built using the two feature edges (4 and 5) and the two split lines (3 and 6).

A simple three-dimensional example is shown in figure 3. The small box feature on top of the larger box is extracted by

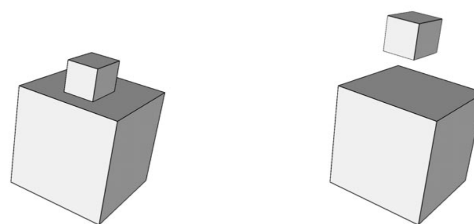

Figure 3. Simple three-dimensional feature showing removal of a small box feature from a larger box. Left: original object. Right: feature and body exploded.

splitting with a single plane.

3.2 Feature Removal Method

More generally, our approach to feature removal uses the following steps:

input: triangulated surface model of a bounded solid polyhedron.

1. **splitting** the original solid with n chosen split planes, subdividing any triangles that are split by a plane;
2. **classifying** each triangle as IN, ON, or OUT with respect to each split plane;
3. **clustering** adjacent, identically classified triangles to build a cluster graph;
4. regarding each node in this graph which is not ON with respect to any of the split planes as a potential feature; and
5. **hole filling** to build solid models of the body and feature.

output: triangulated surface models of a solid feature and solid body and the Boolean operator (union or difference) to combine them.

We elaborate on each of these steps in the following sections.

An advantage of this approach to feature definition is that it generalizes naturally to solid models built by constructive solid geometry from implicit surface primitives such as quadric surfaces, toroids, and other algebraic surfaces. The basic operations required are splitting of one surface by another, point classification queries, and adjacency tests. These are already supported in many CSG modeling systems.

Splitting. To simplify classification, each triangle that intersects a split plane is split (figure 4). Doing so permits each triangle of the surface model to be classified as IN, ON, or OUT with respect to each plane. Splitting changes the surface mesh, but not the solid.

Copyright © 2001 by ASME

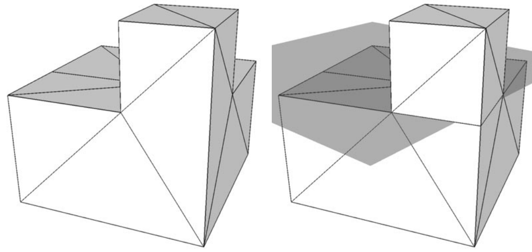

Figure 4. When an object is split with a plane (shown transparent), triangles are subdivided so that each triangle can be classified as IN, ON, or OUT with respect to that plane. Left: before split, right: after split.

Classifying. Once the model has been split, each triangle is classified with respect to the split planes. The three classes are coded using two bits per triangle, per plane. For example, 00=IN, 01=ON, 10=OUT. Thus, the classification with respect to up to 16 planes can be stored in a 32 bit word that is stored with the triangle. Typically, we will isolate features using five or fewer planes, so 32 bits is plenty.

Clustering. Once the model has been classified, the cluster graph is constructed (e.g. Figure 1c). The nodes of the cluster graph correspond to adjacent, identically-classified groups of triangles, and the links correspond to the adjacencies between these groups. This graph is constructed starting from a dual graph of the triangulation, where nodes correspond to triangles and links correspond to pairs of triangles that share a triangle edge. Each node holds a set of triangles and a classification bit string. All links of this dual graph that lie between identically-classified nodes are then collapsed (a link is deleted and two nodes are merged). On completion, the result is the cluster graph.

Feature Selection. Next, the feature of interest is selected. Of course, the definition of a "feature" is subjective and application-specific. For generality, our system imposes only loose constraints on this definition. We regard each node in the cluster graph which is not ON with respect to any of the split planes as a potential feature. More precisely, any such node is a group of adjacent, identically-classified triangles that is either IN or OUT with respect to each plane (note that the cluster might be IN with respect to one plane but OUT with respect to another). The cluster will typically not be a closed surface, however, so it is not a complete model of the feature. But these triangles can often be used to generate a solid feature, so we refer to them as the *feature polygon set*.

Separation of the feature polygons from the rest of the surface leaves a hole in the surface. We attempt to fill that hole to build solid models of the feature and the body. If hole filling is successful, we regard the feature as valid, otherwise it is invalid. Other cluster graph nodes can yield other features. As discussed in section 5, a number of higher level feature ranking algorithms and user interfaces can be built on this low level foundation. First we discuss hole filling, the most technically challenging step.

4 Hole Filling

Once a node of the cluster graph has been selected, this identifies a feature polygon set. Typically, this polygon set is not a closed polyhedron, however, and neither is its complement (the original polygon set minus the feature polygon set). Instead they are triangulated manifolds with boundary. To clarify our terminology: We call the void inside each boundary cycle a *hole*. Filling holes builds solid models of the feature and the body. Typically (but not necessarily), the feature is the smaller shape and the body is the larger shape.

We first explain the hole filling algorithm abstractly, then describe its implementation using quadric error metrics.

4.1 Hole Filling Approach

The hole filling algorithm takes two triangulated manifolds with boundary as input, namely the feature polygon set and the body polygon set, and produces two triangulated solid models as output: the feature and the body. Hole filling is done incrementally by inserting triangles into the hole, shortening the boundary one edge at a time.

When the boundary is planar, the problem reduces to triangulating a concave polygon, for which solutions are well known. As seen in figure 5, however, many features have non-planar boundaries, so hole filling is not as trivial as it might seem.

There are potentially many ways to fill a hole. To best accomplish feature removal, the rule we follow is that all inserted triangles must lie in one of the split planes.

In our figures, we will illustrate the hole filling process by showing hole filling applied to the body, but recall that hole filling is applied to the body and the feature simultaneously.

To represent the planes along which we want to fill a hole, we associate a *constraint set* with each boundary vertex. These are used to guide the choice of triangles to fill the hole. A constraint set is a set of split planes coincident with the vertex, in particular, it represents the positional constraints imposed on the point by the split planes passing through the vertex.

A vertex with one constraint plane corresponds to a boundary vertex in a planar region; hole filling in its local neighborhood must lie in this plane. A vertex with two constraint planes corresponds to a boundary vertex on an edge between two planes (e.g. an edge of a cube); hole filling in its neighborhood must proceed in one plane or the other, and the edge between these two planes can be extended. A vertex with three constraint planes corresponds to a boundary vertex where three or more planes intersect (e.g. a corner of a cube); hole filling in its neighborhood can

Copyright © 2001 by ASME

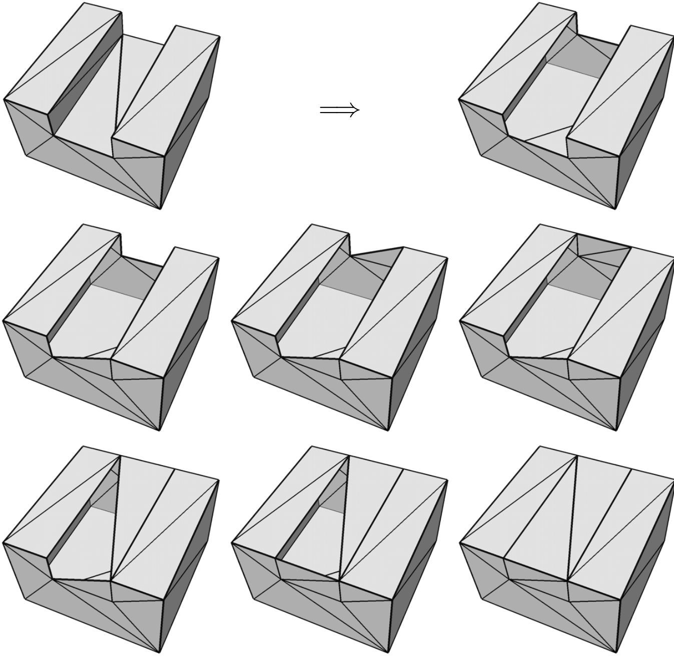

Figure 5. Steps of hole filling. Image 1: Original object. The feature of interest is the groove. Image 2: Triangles of groove are isolated by splitting, classification, and clustering with three split planes (near, top, and far planes of the larger box) and made invisible, leaving a hole that allows us to see the inside. Images 3-8: Six new triangles are inserted sequentially, each coplanar with one of the split planes, to fill the hole and form a solid model of the body. A similar procedure is followed to form a solid model of the feature.

proceed in one of these three planes.

The algorithm repeatedly inserts valid triangles formed by triples of consecutive vertices along the boundary. In order to be value, a there must be a split plane common to its three vertices' constraint sets. Triangles can be inserted in any order subject to the above conditions. This procedure uniquely determines the resulting solid, even though the triangulation is not unique. (Note, for example, that there are two ways to triangulate a planar, rectangular hole). Figure 5 shows how the boundary for a three-plane feature is filled with this algorithm.

4.2 Quadric Error Metric

The constraint sets can be represented in constant space and with built in numerical tolerances using the quadric error metric. Our implementation of the hole filling algorithm uses these quadrics.

A *quadric error metric* is a general second degree function of position. Previously, it has been used primarily for surface simplification [8, 7], but it is also closely related to techniques in multidimensional least squares fitting.

In its standard form, the quadric error metric measures the sum of squared distances between an arbitrary point in space and a set of planes. If $\mathbf{p}_i$ and $\mathbf{n}_i$ are a point and the unit normal of plane i, and $\mathbf{v}$ is the point of interest, then the sum is given by

$$Q(\mathbf{v}) = \sum_i ((\mathbf{v} - \mathbf{p}_i) \cdot \mathbf{n}_i)^2 \quad . \tag{1}$$

This sum can be expanded into a second degree polynomial in $\mathbf{v}$:

$$\begin{aligned} Q(\mathbf{v}) &= \mathbf{v}^\mathsf{T}\mathbf{A}\mathbf{v} + 2\mathbf{b}^\mathsf{T}\mathbf{v} + c \\ &= \mathbf{v}^\mathsf{T}\big(\sum_i \mathbf{n}_i\mathbf{n}_i^\mathsf{T}\big)\mathbf{v} - 2\big(\sum_i \mathbf{n}_i\mathbf{n}_i^\mathsf{T}\mathbf{p}_i\big)^\mathsf{T}\mathbf{v} + \big(\mathbf{p}_i^\mathsf{T}\mathbf{n}_i\mathbf{n}_i^\mathsf{T}\mathbf{p}_i\big) \quad . \end{aligned} \tag{2}$$

This function $Q(\mathbf{v})$ is called a quadric error metric because its isosurfaces are quadric surfaces (ellipsoids, cylinders, or parallel planes). As defined, the quadric matrix $\mathbf{A}$ will always be positive semidefinite. Rather than keep track of each of the planes in the constraint set, the quadric error metric stores $(\mathbf{A}, \mathbf{b}, c)$, using 10 coefficients to represent the symmetric 3×3 matrix $\mathbf{A}$, the 3-vector $\mathbf{b}$ and the scalar c. We refer to computing these coefficients as "computing the quadric" and we sometimes refer to the quadric $(\mathbf{A}, \mathbf{b}, c)$ in terms of the shape of its quadric isosurface. Intuitively, the quadric represents how much freedom a vertex on the surface has while still staying on all the planes in its constraint set.

We can associate a quadric with each vertex of a triangulated surface using the planes of the surrounding triangles. When the neighborhood is planar, the quadric will be two parallel planes, and the rank of the matrix $\mathbf{A}$ is one. When the neighborhood is linear in one direction, as in the middle of an edge on a polyhedron, the quadric will be a cylinder, and the rank is two. When the neighborhood is curved in both directions, as at the vertex of a cube, the quadric will be an ellipsoid, and the rank of $\mathbf{A}$ is three. The quadric is thus a natural representation for the constraint set spoken of earlier.

The *optimal point* – the point that minimizes the quadric error metric – is

$$\bar{\mathbf{v}} = -\mathbf{A}^{-1}\mathbf{b} \tag{3}$$

and its *cost* is

$$Q(\bar{\mathbf{v}}) = \mathbf{b}^\mathsf{T}\bar{\mathbf{v}} + c = -\mathbf{b}^\mathsf{T}\mathbf{A}^{-1}\mathbf{b} + c \quad . \tag{4}$$

The quadric error metric $Q(\mathbf{v})$ quantitatively encodes a set of geometric coplanarity constraints. The optimal point satisfies the constraints iff its cost is zero: $Q(\bar{\mathbf{v}}) = 0$.

Copyright © 2001 by ASME

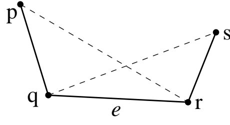

Figure 6. Inserting a triangle near four boundary points **p**, **q**, **r**, and **s**.

Note that in the case of **A** matrices of rank 1 or 2 (which are particularly important for hole filling), $\mathbf{A}^{-1}$ does not exist, and the optimal point is not unique, but would be all points on a plane or line, respectively. When the rank of A is less than three, or equivalently when the quadric has no single optimal point, we say that the vertex "can move", otherwise it is "fixed".

If we define the quadric for each edge to be the sum of the quadrics for its two endpoints, then the cost of an edge is zero if and only if the optimal point for the edge has zero error with respect to the quadrics of each of the edge's endpoints. Relating the definition to constraint sets, an edge has cost zero if and only if the edge's optimal point satisfies the constraints of each of the edge's endpoints.

4.3 Hole Filling Algorithm

To fill a hole, we start by computing the quadric for each vertex in the boundary using only the split planes that contain each vertex. Next, the quadric for each edge of the boundary is computed by summing the two endpoint quadrics.

On each iteration of the hole filling algorithm, we make one or two passes over the edges of the boundary, attempting to insert one or more triangles.

On the first pass, we visit each edge attempting to insert a single triangle connecting three consecutive boundary vertices. To test an edge, the edge quadric is evaluated at each of the edge's endpoints. As shown in figure 6 we test edge e between vertices **q** and **r**. The quadric error associated with e is evaluated at points **q** and **r**. Four cases result:

$Q_e(\mathbf{q}) = 0$ and $Q_e(\mathbf{r}) = 0$: consider inserting triangle **qrs** (or **pqr**),
$Q_e(\mathbf{q}) = 0$ and $Q_e(\mathbf{r}) \neq 0$: consider inserting triangle **pqr**,
$Q_e(\mathbf{q}) \neq 0$ and $Q_e(\mathbf{r}) = 0$: consider inserting triangle **qrs**,
$Q_e(\mathbf{q}) \neq 0$ and $Q_e(\mathbf{r}) \neq 0$: do nothing.

If any of the above succeeds with an insertion for any edge, the algorithm advances to the next iteration.

If no triangle is inserted in the first pass, a second pass over the edges of the boundary is made, attempting to insert a pair of triangles connecting three consecutive vertices to a new vertex. To test edge e, we query the quadrics of the edge and of the edge's first vertex **q**. If **q** can move, and the optimal point **v** for edge e is zero, triangles **pqv** and **qrv** are inserted. An example is shown in figure 7.

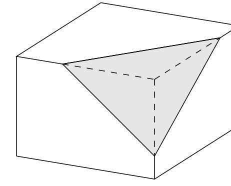

Figure 7. If input object is a shape from which a corner has been cut, insertion of an inferred vertex not present in the input is necessary in order to build solid models of the body and the feature, in this case a box and a tetrahedron, respectively.

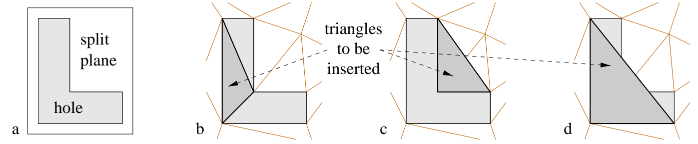

Figure 8. Overlap detection. a) A concave, planar hole (light gray). b-d) Region surrounding hole is a triangulated surface. b) Test triangle (dark gray) does not intersect existing triangles, so it can be inserted. c,d) Test triangles intersect existing triangles, so they cannot be inserted.

At each iteration, boundary and quadrics are updated.

4.4 Validity Checks

Coincident triangles forming *overlaps* cannot be permitted during hole filling. As shown in figure 8, triangles being considered for insertion are checked for overlap with existing triangles. If an overlap is detected, the triangle is not inserted.

5 Feature Finding Algorithm

The feature removal algorithm is built on the methods for feature representation and hole filling described earlier. Various algorithms could be built using the approach we are describing in this paper. Below, we describe the algorithm of our current implementation to demonstrate one way in which a simple program for polyhedral feature detection and removal can be constructed. The algorithm is first summarized in pseudocode, after which we explain some of the new components.

Copyright © 2001 by ASME

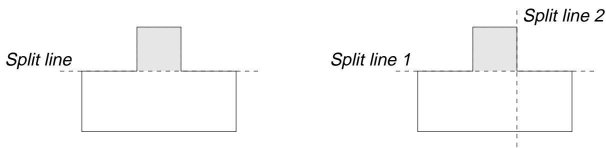

Figure 9. Elimination of redundant features. The small shaded box feature on top could be generated by a single split line (left) or using two split lines (right). Clearly these are identical solid features; we must detect that they are redundant and eliminate the latter.

```
read original_model from file
make a list of the planes of the faces of the model
ask user how many planes are desired (nmax)
featurelist = {}
for n = 1 .. nmax
  loop over all combinations of n planes
    model = original_model
    split the model using the chosen planes
    classify each triangle relative to these planes
    cluster adjacent, identically labeled triangles to build cluster graph
    for all clusters that are not ON with respect to any plane
      each such cluster is a potential feature polygon set
      if feature polygons are adjacent to split plane polygons and
        feature polygons are not in featurelist then
          fill hole inside boundary
          if hole filling successful then
            results of hole filling are solid models of body and feature
            compute volume of feature
            if volume > half of original, reject this feature
            compute "value" of feature
            save feature polygon set, plane list, and value in featurelist
          else
            this is an invalid feature, ignore

sort features by value
present features and allow user to pick one
```

The approach we have chosen is to iterate through the various combinations of split planes, and for each, do the appropriate splitting and classification. This keeps the preprocessing relatively small. An alternative approach would be to split and classify the triangles of the model with respect to all split planes as a preprocess. There might be many tens or even hundreds of potential split planes, so splitting could generate a highly subdivided model. Consequently, the preprocessing could be slow and take up a lot of memory. On the other hand, this approach could be faster overall, since it could eliminate redundant splitting and classification. We have chosen the former, more "lightweight" approach, since it provides greater flexibility.

5.1 Removal of Redundant Features

In a brute force enumeration of feature polygon sets, many redundant features are found. We can reduce the number of features detected by the algorithm and ease the job of the user or software examining the feature list by eliminating redundancy. Figure 9 illustrates the problem, in 2-D. Briefly, this can be done using subset tests on the sets of triangles in a feature polygon set and coplanarity tests with respect to each of the split planes.

5.2 Feature Ranking

The above criteria generate a number of features but do not distinguish or rate features as "good" or "bad" the way a subjective user might. After experimenting with various volume-based and surface-area-based formulas, we have found the following quantitative measure to provide a ranking fairly consistent with most human users:

$$\text{value} = \frac{(\text{ surface area of inserted triangles })}{\left(\begin{array}{c}\text{surface area of original object}\\ \text{coincident with one of the splitting planes}\end{array}\right)}$$

This value will be small for "good" features.

6 Results

The above algorithm has been implemented in C++. The software reads in polyhedral models in a variant of the Wavefront OBJ file format, allowing the user to either manually select split planes and invoke hole filling, or use the automatic algorithm described above to search for the leading features. All of the perspective 3-D figures in this paper were generated with this software.

Figure 10 shows the results when our system is run on a model of a vice. Automatic feature identification using up to three split planes runs in 2 minutes total. After examining 575 combinations of three or fewer split planes, the system identified 1761 potential features, filled holes, and eliminated redundancies, outputing 73 valid features. As shown in the figure, the top-ranked features conform closely to those that a human might pick.

Figure 11 shows the features of a more complex object. Processing of this polyhedron with three planes took 18 minutes, examined 1793 combinations of planes and 6127 potential features, from which 128 were found to be valid. The figure shows the ability of our system to deal with rather complex, abutting features.

7 Conclusions

The use of a combination of topological and geometric information permits a general definition of features in solid models. In particular, we define a feature in terms of a connected surface region that is classified consistently when an object is split by one or more surfaces. Filling the holes created with the feature is topologically removed from the body builds solid models of both

Copyright © 2001 by ASME

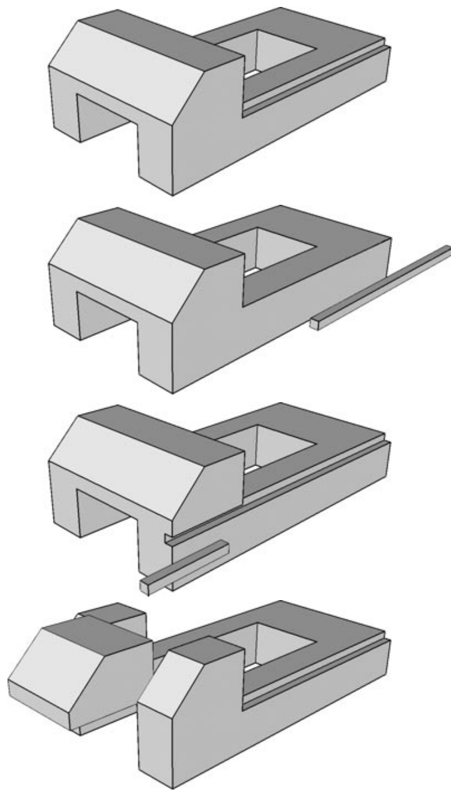

Figure 10. Top features of a vice model after splitting with three planes. a) original object, b) feature 1 is one corner groove, feature 2 is its mate, c) feature 3 is extension of groove, feature 4 is its mate, d) feature 5 is block between the rails.

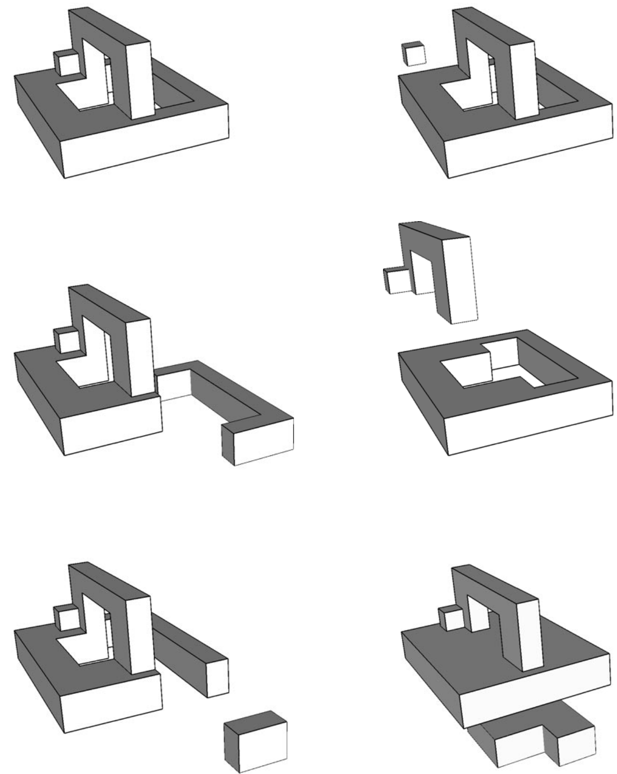

Figure 11. Top features of a multi-holed object after splitting with three planes. Original object, plus features 1, 2, 3, 4, and 38.

the feature and the body. A variety of features can be found automatically by splitting an object with extensions of its surfaces, elimination of redundancy, and intelligent ranking. The approach could be used to help designers coarsen CAD models for finite element analysis, to aid in the analysis and documentation of the purpose of features.

This approach to feature identification and removal has been implemented for polyhedral objects. The approach has advantages over previous methods as it employs a very general definition of "feature". It could also be generalized for solid models employing quadrics and other implicit surfaces.

8 Acknowledgments

We are grateful to NSF grant DMI-9813259 and Fundació Caixa Castelló - Bancaixa for financial support.

REFERENCES

[1] P. Cignoni, C. Montani, and R. Scopigno. A comparison of mesh simplification algorithms. *Computers & Graphics*, 22(1):37–54, 1998.

[2] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. MIT Press, Cambridge, MA, 1990.

[3] Leila De Floriani. Feature extraction from boundary models of three-dimensional objects. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(8):785–798, August 1989.

[4] R. Gadh and F. B. Prinz. Recognition of geometric forms using the differential depth filter. *Computer-Aided Design*, 24(11):583–598, November 1992.

[5] Rajit Gadh and F. B Prinz. Automatic determination of feature interactions in design-for-manufacturing analysis. *ASME Journal of Mechanical Design*, 117(1):2–9, 1995.

[6] Rajit Gadh and F. B Prinz. A computationally efficient approach to feature abstraction in design-manufacturing integration. *ASME Journal of Engineering for Industry*, 117(1):16–27, 1995.

[7] Michael Garland. *Quadric-Based Polygonal Surface*

Copyright © 2001 by ASME

Simplification. PhD thesis, Carnegie Mellon University, CS Dept., 1999. Tech. Rept. CMU-CS-99-105. http://www.cs.cmu.edu/~garland/thesis/.

[8] Michael Garland and Paul S. Heckbert. Surface simplification using quadric error metrics. In *SIGGRAPH 97 Proc.*, pages 209–216, August 1997. http://www.cs.cmu.edu/~garland/quadrics/.

[9] P. Gavankar and M. R. Henderson. Graph-based extraction of protrusions and depressions from boundary representations. *Computer-Aided Design*, 22(7):442–450, September 1990.

[10] Paul S. Heckbert and Michael Garland. Survey of polygonal surface simplification algorithms. In *Multiresolution Surface Modeling Course Notes*. ACM SIGGRAPH, 1997. http://www.cs.cmu.edu/~ph.

[11] Y. S. Kim. Recognition of form features using convex decompositon. *Computer Aided Design*, 24(9):461–476, 1992.

[12] M. J. Pratt and P. R. Wilson. Requirements for support of form features in a solid modeling system. Technical Report R-85-ASPP-01, CAM-I Inc., Arlington, Texas, June 1985.

[13] Hiroshi Sakurai. Volume decomposition and feature recognition: Part 1 — polyhedral objects. *Computer-Aided Design*, 27(11):833–843, November 1995.

[14] T. Woo. Feature extraction by volume decomposition. In *CAD/CAM Technology in Mechanical Engineering*, 1982.

[15] M. C. Wu and C. R. Liu. Analysis on machined feature recognition techniques based on B-rep. *Computer-Aided Design*, 28(8):603–616, 1996.

[16] Rajit Gadh Yong Lu and Timothy J. Tautges. Feature decomposition for hexahedral meshing. In *ASME Design Automation Conference*, Las Vegas, NV, September 1999. DETC99/DAC-8618.

Copyright © 2001 by ASME