SM37: Reconstruction of Feature Volumes and Feature Suppression

Sashikumar Venkataraman[1]
Geometric Software Solutions Ltd.
Plant 14, Pirojshanagar,
Vikhroli, Mumbai-400079, India
92-22-5960982
sashiv@geometricsoftware.com

Milind Sohoni
Dept. of Computer Science and Engg.
Indian Institute of Technology, Powai
Mumbai-400076, India
91-22-5767729
sohoni@cse.iitb.ac.in

ABSTRACT

This paper describes a systematic algorithm for reconstructing the feature volume from a set of faces in a solid model. This algorithm serves a dual purpose. Firstly, the algorithm generates the feature volume by extending or contracting the neighboring faces of the set of faces. Secondly, the algorithm may also be used to remove (or suppress) the face-set from the model. The algorithm uses a divide-and-conquer strategy and geometric cues to identify the correct topology. It robustly handles a wide class of feature volumes with complex topology and geometry. A simplified version of the algorithm has also been presented to handle volumes resulting from 2.5D features.

Keywords

Feature volume reconstruction, feature suppression, delete faces.

1. INTRODUCTION

Feature recognition is an important step in many CAD/CAM applications. In the area of manufacturing, feature recognition provides an important bridge between CAD and CAM by automatically or interactively extracting manufacturing features from a part. These features represent negative volumes associated with a particular machining operation. In the area of design, feature recognition is used to extract specific set of form features from parts that do not have design information. The original design tree may be lost in translation of data from one CAD system to another. Feature recognition is then used to recreate the design intent. For a comprehensive review of feature recognition, the reader is referred to Refs [1,2,3].

An important step during feature recognition is the creation of feature volume corresponding to the feature face-set. Feature volumes are used in many applications such as tool accessibility analysis [4] and process planning [5]. Feature volumes also aids in resolving feature interactions during feature recognition [6,7,8]. For example in [6], the same authors had presented a graph-based framework for feature recognition. Simple or non-interacting features are first recognized as a set of faces, and the corresponding feature volume is constructed. The recognized feature is then suppressed by removing/filling the feature volume, and thereby updating the part. The part updation simplifies the part, and facilitates further recognition of interacting features.

Apart from feature recognition, the delete face operator can also be used for local editing and modification of features. For example, deleting certain kinds of features is frequently used to simplify parts to aid several downstream applications such as finite element analysis (FEA). This operator is also very useful in modification of features in parts that do not have the complete design tree available.

Several researchers have attempted to construct feature volumes from feature face-sets. One common technique adopted to create feature volumes is by sweeping 2D-profiles [9, 10]. However, this technique can be adopted only for swept features. Attempts have also been made to create feature volume by extending adjacent faces of the feature faces [8,11,12,13]. These approaches begin by intersecting all the neighboring faces, and then finding the feature volume from the intersection graph using some heuristics and constraints. However, selecting the right edges and vertices is a difficult problem, and no systematic approach has been presented to resolve them. Overall, the approaches presented so far are mainly heuristic in nature and usually work for simple and specific kinds of feature volumes. No rigorous algorithm has yet been presented that handles general feature volumes that could be concave and of arbitrary topology.

In this paper, we present a general algorithm to remove a set of faces from a part and reconstruct the corresponding feature volume by extending/contracting the neighboring faces of the feature. This is referred to as the *delete face operation*. Our algorithm is unique from other face-extension algorithms in many aspects. We use a *divide-and-conquer* approach in which a complex feature volume is divided into simpler sub-problems. Unlike other face-extension approaches, our algorithm does not perform all possible intersections initially; but resolves edges and vertices incrementally. Heuristics are used in clearly defined steps, and are independent of the basic framework of the algorithm. Our algorithm handles many complex configurations including concave feature volumes, and can be applied to faces of any surface type that extends smoothly. Unlike some other approaches, our algorithm allows extension as well as shrinking of neighbor faces. Furthermore, the suppression of the feature from the part and the creation of feature volume take place in a single step without needing an extra boolean operation.

---

[1] Sashikumar V. is also pursuing a Ph.D. in the Dept of Computer Science and Engg. at the Indian Institute of Technology, Powai.

The delete face operator is offered by many geometry kernels such as Parasolid [15] and ACIS [16]. However, the functionality and robustness varies across kernels and several do not offer it at all. Our algorithm deals with many cases that are not handled by the existing geometry kernels. These examples are discussed later in the paper. Moreover, none of the existing kernels gives the feature volume as an output of the delete face operator. Furthermore, the presented algorithm is implemented in a kernel-independent platform [17], and can be easily ported to any geometry kernel.

The paper is organized as follows. Section 2 presents the overview of the delete face operator. Section 3 discusses the main algorithm of delete faces. Section 4 describes finer details of the algorithm and explains the working of the algorithm in specific situations. Section 5 describes issues relating to implementation of the delete face operation. Section 6 presents several examples that are handled by our algorithm. Section 7 discusses a special case of the operation applied to 2.5D manufacturing features. A conclusion to the paper is presented in the final section.

2. OVERVIEW

The input of the delete face operator is a boundary representation model (brep) and a set of connected faces that is to be removed from the model. The input set of faces is assumed to correspond to a feature that either adds or removes volume from a solid. The output of the operator is a modified model obtained by removing the set of faces, and patching up the deleted region. We assume that extension/contraction of neighbor faces suffice to construct the feature volume without requiring additional faces. Figure 1(a) shows an example of a pocket feature interacting with two slots. The bottom faces of the two slots are at different levels. The pocket feature can be recognized using common feature recognition techniques. The feature volume is then created using the delete face operation and is used to suppress the feature as shown in Figure 1(b).

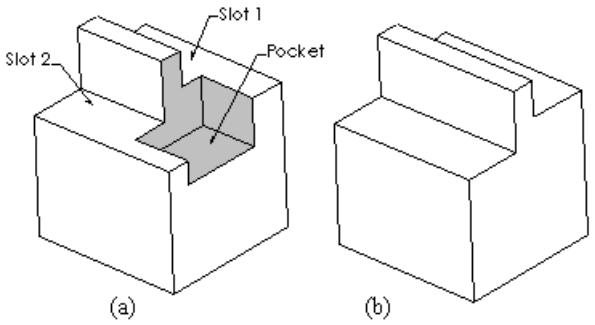

Figure 1 (a) Feature interaction between a pocket and slots. (b) Model obtained after suppressing the pocket feature.

2.1 Terminology

The input to the delete face procedure is a set of faces. These faces are referred to as the *feature faces* and is denoted by $f_0, f_1, f_2, \ldots f_m$. The output is an updated model formed after removing the set of faces and patching the region by extending/shrinking the neighbor faces. These neighboring faces are referred to as the *external faces* and are denoted by $F_0, F_1, F_2, \ldots F_n$. The edges that are along the boundary of the feature faces are termed as *boundary edges*, and the vertices along the boundary are termed as the *boundary vertices*. The edges that are not along the boundary but touch the boundary vertices are termed as *external edges*. The set of boundary edges forms one or more *boundary loops*. Figure 2 shows the boundary loop and external edges of a corner slot. The boundary edges of the slot form one boundary loop that has three external edges.

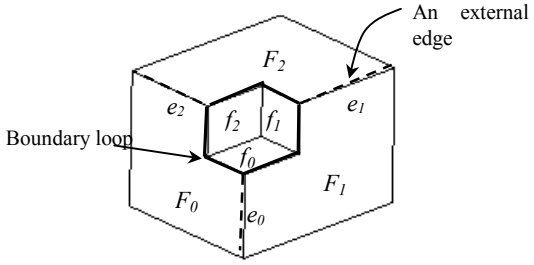

Figure 2 Boundary and external edges for a corner slot.

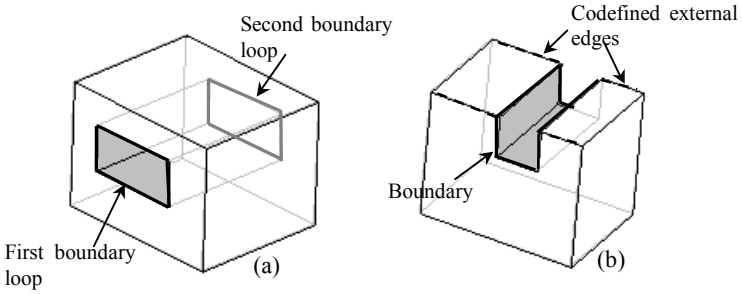

Figure 3 Boundary and external edges for a through pocket and through slot.

Figure 3 shows the boundary loops and external edges for a through pocket and a slot. Figure 3(a) shows an example having two boundary loops. Figure 3(b) shows an example of two external edges between the same geometric entities. These are termed as *co-defined edges*. Each external edge has a left and a right face associated with it. This is initially assigned from the neighbor faces of the external edges. The left face is defined as the neighboring face on the left of the edge as one travels along the edge towards the boundary vertex. The other neighboring face is assigned as the right face of the external edge. The external edges in a loop are ordered in a circular list such that the left face of the next external edge is the right face of the current external edge. For simplifying the analysis, it is convenient to visualize the boundary loop and the external edges in a planar graph as shown in Figure 4 below.

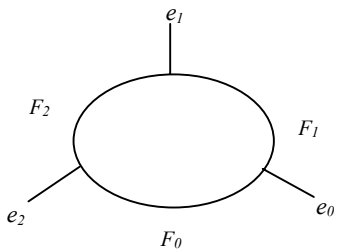

Figure 4 Schematic representation of the external edges of the corner slot.

2.2 Nature of Solutions

The solution to the delete face operation is better understood in terms of the feature volume. As discussed before, we assume that the face-set to be deleted arose from a boolean subtract (or unite) of the feature volume from the part. Figure 6 shows a volumetric feature created by an extrude cut and the corresponding feature volume.

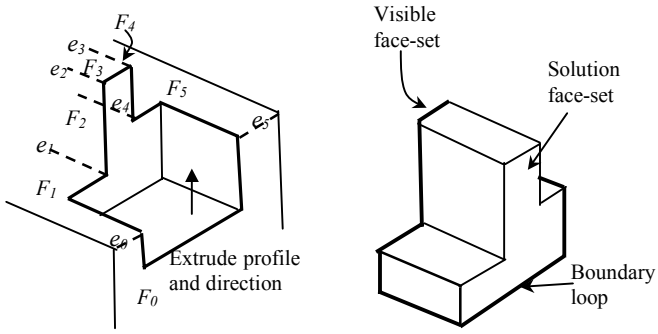

Figure 5 Pocket created as an extruded cut.

The boundary loop is seen on the feature solid, and partitions the set of faces into two parts, the visible face-set and the solution face-set. The visible face-set is the collection of feature faces that need to be deleted from the part. The solution face-set is the required output of the delete face and is formed by extending or shrinking the neighbor faces. The feature volume is formed by stitching the visible face-set and the solution face-set. The solution face-set is visualized as a graph whose nodes are the vertices and whose arcs are the edges of the solution face-set. By its very construction, this is a planar graph and every face of this graph has one or more boundary edges and corresponds to the extension/contraction of the corresponding neighbor face. Furthermore, the external edges must extend into (or contract from) the solution graph with the same geometry that is completely specified by its neighboring faces. Thus in effect, the solution to the delete face problem may be constructed by non-deterministically "guessing" the solution graph, and then constructing the geometry. Using a recursion and some geometric heuristics, this is roughly the approach we follow. Figure 6 shows the formation of the solution graph for the above example by the delete face operation.

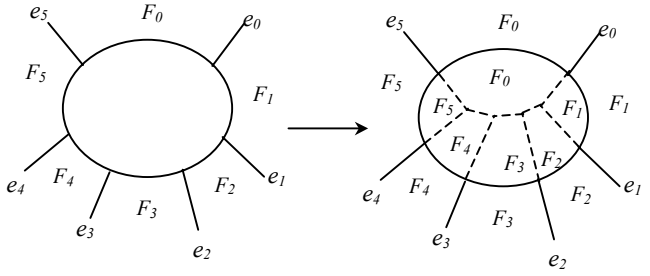

Figure 6 Solution graph obtained by the delete face operation for the boundary loop shown in Figure 5.

There is, however, another ingredient required to construct the geometry from a candidate solution graph, namely, the construction of the local geometry at a vertex. In the purely convex or concave vertex case, the face geometry suffices to create the local geometry at the vertex. This is not the case, when the vertex has negative gaussian curvature. In all, the choice of the vertex geometry has a total of eight possibilities and is termed as the *sense* of the vertex. These have been shown in Figure 7 and Figure 8 where X, Y, and Z denote faces with fixed face outward normal along the x, y and z axis. It is important to note that the vertex sense is directly related to the convexity and direction of the adjoining edges.

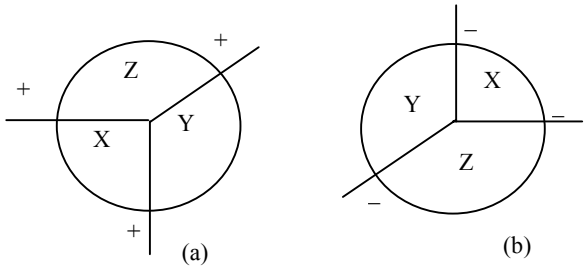

Figure 7 Vertex with positive Gaussian curvature. (a) convex vertex; (b) concave vertex.

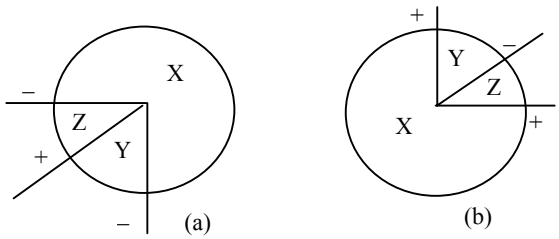

Figure 8 Two examples of vertices with negative Gaussian curvature.

3. DELETE FACE ALGORITHM

The delete face algorithm mainly solves a *loop problem* defined by a set of external edges in a well-defined circular order. The initial loop problem consists of all external edges corresponding to the boundary loop(s) of the face-set to be deleted. The output of the algorithm is a solution graph that matches along the boundary loop.

The delete face algorithm begins by selecting an external edge, and extending the underlying curve to an appropriate interval around the boundary vertex of the edge. A face is determined from the set of external faces that most likely intersects this external edge. This face is termed as the *opposite face*. The end vertex of the extended external edge v_c is determined as the intersection of the external edge and the opposite face. This simultaneously creates two edges of the solution graph emanating from v_c that are obtained by intersection of the corresponding adjacent faces of the external edge and the opposite face. Figure 9 shows this step starting at external edge e_2. The opposite face is determined as face F_0. The edges e_l and e_r are determined as the intersection of faces F_2 and F_3 with F_0.

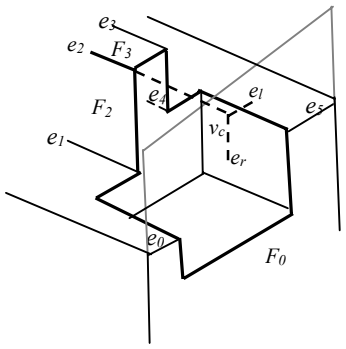

Figure 9 Extending an external edge upto an opposite face.

The above step naturally breaks the delete face problem into two sub-problems corresponding to the external edges on either side of the selected external edge. In the above example e_l, e_3, e_4 and e_5 form one loop problem, while e_r, e_0 and e_1 form another loop problem. These problems are recursively solved as individual loop problems. The sub-graphs obtained by the sub-problems are then combined to produce the final solution graph. Figure 10 shows the breaking of the loop problem into two sub-problems schematically. The above recursion continues as long as the loop problem has any external edge. A loop problem with no external edge is trivially discarded since it does not change the solution graph.

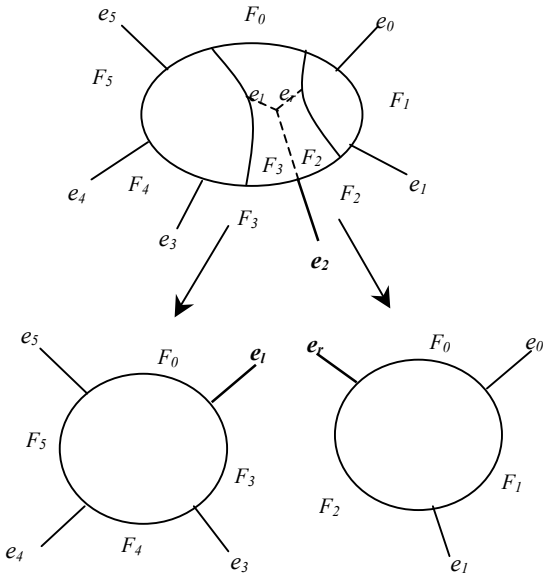

Figure 10 Breaking the loop problem into two sub-problems.

In the above algorithm, heuristics are used to determine the opposite face and vertex sense of the newly created vertex v_c. These heuristics are described later in this section in detail.

In certain situations, instead of an opposite face, an opposite co-defined external edge is found for the selected external edge. In this case, the extended external edge merges with the other external edge, and no new edges are created. The loop problem again splits into two sub-problems corresponding to the external edges on either side of the selected external edge. Figure 11 shows the external edges of a closed pocket feature over a rib feature. The figure shows the working of the delete face algorithm starting at external edge e_1. No opposite face is determined for e_1, but an opposite external edge e_6 is found that merges with the extended edge of e_1. In this example e_2, e_3, e_4 and e_5 form one loop problem, while e_7 and e_0 form another loop problem. These two sub problems are shown in Figure 12. Note that face F_2 and F_6 , have co-defined geometry, while F_1 and F_7 have co-defined geometry.

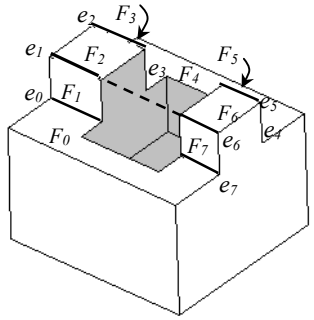

Figure 11 Pocket on a rib. External edge e_1 is picked as the starting external edge.

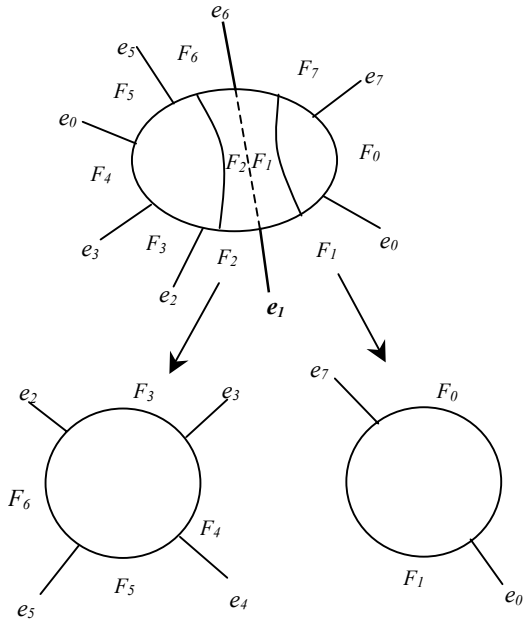

Figure 12 Sub-problems created while deleting the pocket.

3.1 Algorithm Steps

The delete face algorithm is a systematic recursive procedure that breaks a loop problem into two sub-problems with lesser number of external edges. The individual steps of the algorithm are described below.

Algorithm : Loop problem of delete face operator.

Input : A set of circularly ordered external edges.

Output : Inner solution graph corresponding to the loop.

Procedure DeleteFace (e_0, e_1, …. e_n : circular list)

(1) Pick an external edge e_i from the loop with boundary vertex v_i, left face F_l and right face F_r.

(2) **Find** *opposite* face F_c or co-defined edge e_c using heuristics.

If opposite face found **do**

(3) Create vertex v_c on F_c and an edge $e_{i'}$ between v_c and v_i.

(4) Create edge e_i between F_l and F_c and edge e_r between F_r and F_c.

(5) **Assign** *vertex sense* for v_c using angle heuristics.

(6) DeleteFace(e_l, e_{i+1}, …. e_{c-1}).

(7) DeleteFace(e_r, e_{c+1}, …. e_{i-1}).

End do

If codefined edge found **do**

(8) Create edge between e_i and e_c.

(9) DeleteFace(e_{i+1}, e_{i+2}, …. e_{c-1}).

(10) DeleteFace(e_{c+1}, e_{c+2}, …. e_{i-1}).

End do

End DeleteFaces

The above recursion continues till the sub-problems are reduced to the terminating step of the recursion that is a problem containing no external edges which is trivially solved since it does not affect the final solution graph. In the sections below we describe the important steps of the above algorithm in detail. Specifically, we describe the heuristics used in step 2 and 5 of the algorithm that determine the opposite face and the vertex sense respectively.

3.2 Opposite face heuristic

An important step in the delete face algorithm is the finding the opposite face F_c to e_i. A simple heuristic is to pick the face whose underlying surface is closest to the external vertex v_i along e_i. This is found by intersecting the extended curve of e_i and the geometry of all the external faces, and finding the face that first intersects the extended external edge starting from the boundary vertex. In Figure 11, the opposite face of e_2 is found as F_0 using this heuristic. While this heuristic works fine for convex feature volumes, it may fail when the feature volume has concavities. In the example in Figure 11, if e_0 is chosen as the starting external edge, the opposite face is found as F_2 using this heuristic. This is shown in the Figure 13 below. The next step of recursion would continue by finding the opposite face to e_l and e_r. The opposite face of e_r is wrongly calculated as F_5 instead of F_3 using the above heuristic since v_{c1} comes before v_{c2}.

In order to tackle such problems the heuristic is enhanced to give preference to vertices that are within the convex hull of the face excluding the boundary edges. In the above example, v_{c1} is not within the convex hull of face F_5, while v_{c2} is within the convex hull of face F_3 excluding the boundary edges. Hence, v_{c2} is chosen in preference to v_{c1} using the modified heuristic.

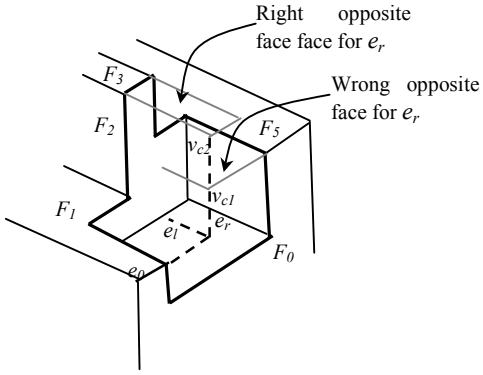

Figure 13 Conflict in choosing opposite face for edge e_r.

3.3 Vertex sense heuristic

Another important step in the delete face algorithm is the assignment of the vertex sense for the opposite vertex v_c. As discussed in the previous section, there are eight possibilities for the sense of a vertex formed by three faces in a particular ordering. Since the convexity of e_i is known, there are four possibilities for the sense assigned to vertex v_c. These possibilities can be also be interpreted as setting either the convexity or direction of the edges e_l and e_r.

In certain cases, the vertex sense can be predicted directly from the gaussian curvature. The sign of the gaussian curvature can be computed from the dot product of the tangent vector of e_i, the convexity of e_i, and the normal of the opposite face. If the vertex has positive gaussian curvature, then the convexity of the two edges e_l and e_r are the same as that of e_i and hence the vertex sense is completely determined. For example, in Figure 9, the vertex v_c has positive gaussian curvature. Hence the edges e_l and e_r are convex, and the vertex sense is completely determined.

If the gaussian curvature of the vertex is found to be negative, then there are three possibilities for the vertex sense, and the sense cannot be predicted deterministically. Figure 14 shows the same example as in Figure 13 with e_0 chosen as the starting external edge. The vertex sense for v_c is wrongly computed due to wrong setting of direction for e_r. The edge e_r is set wrongly as concave instead of convex in this case.

The vertex sense in such cases is determined using certain heuristics. One such heuristic is obtained using the external edges of the neighboring faces. Every neighbor face typically has two external edges. The angle between these two external edges is termed as the external angle for the face. This angle denotes the total of the *signed* angle turn as one traverses from one external edge to the other along the solution graph. This quantity is independent of the nature of the solution graph and thus may be computed beforehand. Figure 15 shows the external angle for faces F_0, F_1 and F_2. The external angle for face F_0 is *0*, while the external angle for F_1 is *Pi/2*.

The total of the absolute value of the angle turn for each face depends on the solution graph, and is equal to or more than the external angle. This is referred to as the *absolute angle*. The heuristic to assign the direction of edges aims to minimize the sum of the absolute angles of the neighbor faces. This roughly corresponds to the "minimum variation" feature volume. Using this heuristic, the angle variation for the edge directions shown in Figure 14 result in more variation than those shown in Figure 13. This is because in Figure 14, the variation for face F_2 becomes large due to the wrong direction set for edge e_r (since e_r will have to turn a full *3*Pi/2* to depart along F_2 as e_2).

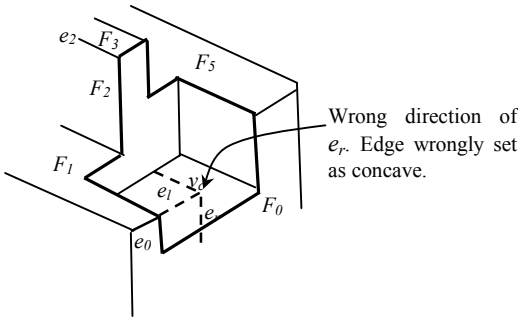

Figure 14 Wrong setting of vertex sense due to wrong direction chosen for edge e_r.

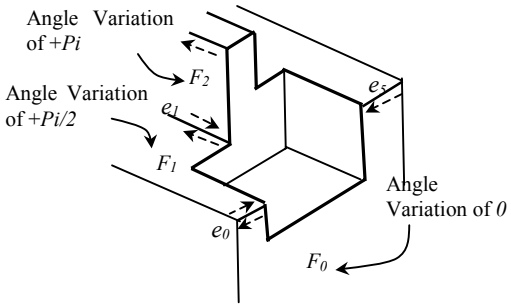

Figure 15 External angle shown for faces F_0 and F_1. These are used in heuristic to predict vertex sense.

4. ALGORITHM DETAILS

The examples considered so far demonstrate the broad working of the delete face algorithm. In this section, we describe finer details of the algorithm, and demonstrate the working of the algorithm in specific situations. These situations include delete face problems with multiple boundary loops, problems involving shrinking of neighbor faces and degeneracy. This section also discusses the validity of the final solution obtained by the algorithm, and situations that give rise to multiple solutions.

4.1 Multiple boundary loops

In the example dealt in Figure 9, there is a single boundary loop. However in certain situations such as in Figure 3(a), there are multiple boundary loops. In such cases, we first need to determine whether the neighbor faces of the boundary loops interact with each other during the delete face operation. Figure 3(a) shows a through pocket with two boundary loops. In that case, the neighbor faces of the boundary loops do not interact with each other during the deletion operation.

Figure 16(a) shows an example of a groove around the corner that can be visualized as an open pocket with an island. The groove has two boundary loops and four external edges. In this case, the neighbor faces of the two boundary loops interact with each other during the deletion operation. The model obtained after the delete face operation in shown in Figure 16(b).

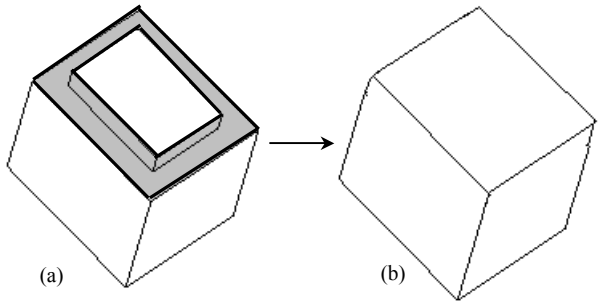

Figure 16 Delete face of a feature with two boundary loops.

The interaction between boundary loops is taken as an input by the delete face algorithm. This can be set by the caller based on the type of feature that is passed for deletion. For example, for through pockets and holes the loops do not interact with other; while for blind extruded pockets and bosses, the loops would interact with each other.

In case when the loops do not interact with each other, the delete face algorithm makes a separate loop problem for each boundary loop. However, if the loops interact, then a single loop problem is created for all the boundary loops and external edges. The algorithm is suitably modified to consider such loop problems.

4.2 Shrinking of faces

In all the examples presented so far, the delete face operation is achieved by only extending neighboring faces. However, the presented algorithm allows neighboring faces to either extend or contract. Face contraction is achieved by allowing the external edge to contract from the boundary vertex while finding the opposite face in step 2 of the algorithm.

Figure 17(a) shows an example of an uneven slot. In this example, the bottom face f_1 and side face f_0 are passed for deletion. During the delete face algorithm, face F_0, F_1, and F_3 extend outward, while F_2 contracts within to obtain the result shown in Figure 17(b).

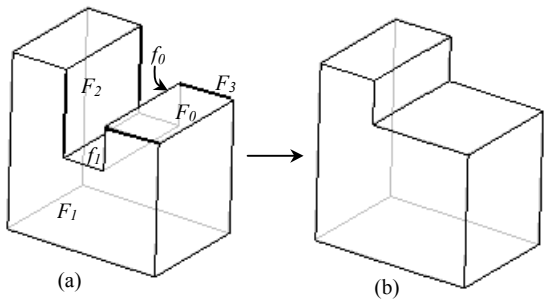

Figure 17 Shrinking of neighbor faces during delete face.

The choice of the face-set to be sent to the delete face algorithm is crucial as indicated in Figure 17. In this example, F_2 is not passed in the face set for deletion and is treated as a neighbor face. In general, the determination of the faces passed for deletion must be done in the recognition stage itself. Faces of the feature that merge with other faces of the model are termed as *virtual* faces (refer [6]), and are not passed for deletion. Figure 18 shows an example of pocket with a single virtual face. During suppression, only the non-virtual faces of the pocket are passed to the delete face operation. The virtual face acts as a neighbor face and contracts during the delete face operation.

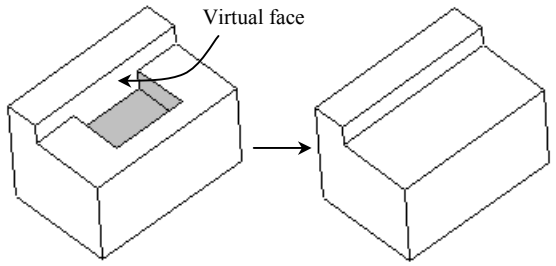

Figure 18 Deletion of a pocket with a virtual face. The virtual face is not passed for deletion.

4.3 Degeneracy handling

The discussion so far dealt with manifold bodies that have only three faces adjacent to each vertex. Degenerate vertices arise when more than three faces are adjacent to a vertex. In this section, we show how the algorithm is enhanced to handle bodies with degenerate vertices.

Degenerate vertices can either arise in the final solution face-set or in the input boundary loop itself. Figure 19 shows an example in which face-deletion results in a degenerate vertex. Such cases are handled by enhancing step 2 of the algorithm. While finding the opposite face, we allow faces that are at zero distance from the start external edge. This creates zero length edges in the solution graph that are finally deleted to create the degenerate vertices.

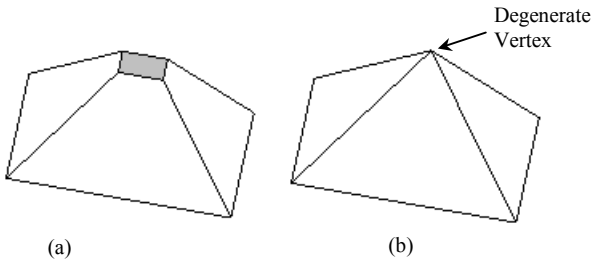

Figure 19 Example in which delete face operation creates a degenerate vertex.

The algorithm also handles degenerate vertices that may be present along the boundary loop. Figure 20 shows an example of a of a slot that has a degenerate vertex along the boundary loop. In such situations, zero length edges are introduced between the external edges along the boundary loop to break the degeneracy prior to the delete face operation. These edges are merged at the end of the delete face operation. Figure 21 shows the breaking of the vertex degeneracy along the boundary loop.

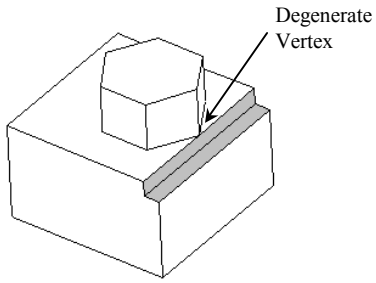

Figure 20 Example in which a degenerate vertex is present along the boundary loop

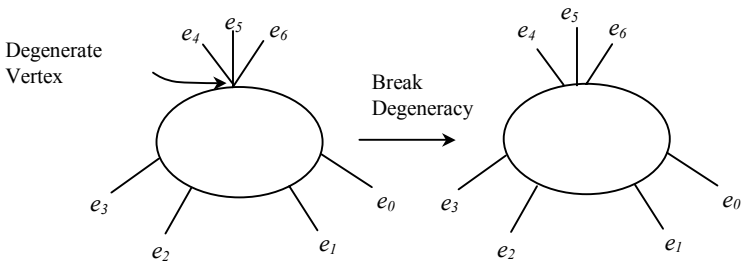

Figure 21 Breaking of degeneracy prior to delete face operation.

4.4 Validity of solutions

There are two main steps in each recursion of the delete face algorithm; namely; determining the opposite face and fixing the vertex sense of the newly created vertex. Theoretically speaking, there is always a right opposite face and vertex sense that will work towards the correct solution graph. In our implementation, these are determined using geometric heuristics as described in the previous section.

If the feature volume is fully convex, then the opposite face and vertex sense is always predicted rightly by our heuristics, and the correct solution is found. However, if the feature volume is concave such as in Figure 1, then the algorithm obtains the right solution as long as the heuristics work correctly at each step of the algorithm. If the heuristics fail at any step, then a wrong solution or no solution is obtained. A wrong solution is determined by performing a *check* on the topological and geometry structure of the final feature volume. If the check on the feature volume fails, then the model is restored back to the original state.

In order to avoid overall failure due to the failure of a particular heuristic, *back-tracking* can be introduced in the delete face algorithm to control and refine the search of the correct opposite face and sense. However, this could take substantially more time, and may not be feasible in applications such as feature recognition where feature suppression is used repeatedly.

It is also possible for the delete face algorithm to have multiple or no solutions. Figure 22 shows a blend face that interacts with a rib. In this case, two solutions are possible when the blend face is passed to the delete face operation. From the design perspective, the first solution is preferable to the second one. The delete face algorithm outputs any one of the solutions depending on the starting external edge and the results of the heuristics at the intermediate stages.

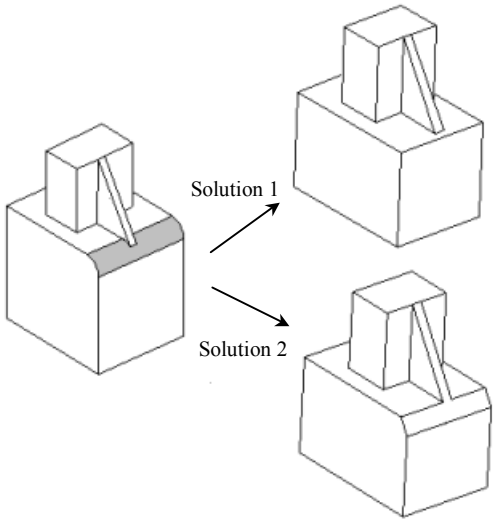

Figure 22 Example showing multiple solutions that may arise from the delete face operation.

It is of course possible for the delete face problem to have no possible solution. These are usually cases that need new faces to be inserted in the solution, and are not within the scope of the current algorithm.

5. IMPLEMENTATION

The delete face algorithm has been implemented as part of the Feature Recognition system library at Geometric Software Solutions Ltd [17]. The implementation uses local operations in the form of *Euler Operators* [14] to modify topology locally. Euler Operations are low-level functions that modify a small region of topology. Using these operators, topological elements such as faces, loops, edges and vertices may be added, removed, or modified in a model. Together with functions to attach and detach geometry, these functions enable applications to implement their own modeling operations, such as local operations and feature creation.

Euler Operation functions always return a body with valid topological data-structures. However, the functions do not alter geometry - new topology has no geometry attached, and any topology that is deleted has its geometry deleted first. This means that the resulting body is normally invalid. Geometry is later associated with the model to make the model geometrically valid.

The Euler operators can be divided into two groups; the *make group* and the *kill group*. The make group consists of operators for adding some elements into the existing model, while the kill group does exactly the inverse of the make group. For example, MEV is an operator that makes an edge and a vertex, while KEV is an operator that kills (or deletes) an edge and vertex. Similarly, MEF is an Euler operator that makes an edge and face and KEF is the corresponding inverse operator. These operators are shown pictorially in Figure 23 below.

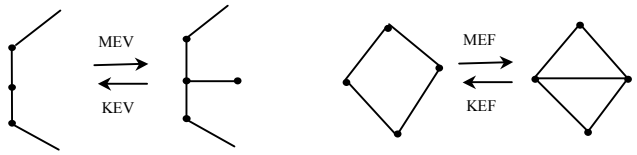

Figure 23 Example of Eulers operators. (a) Operators for making/killing of edge and vertex. (b) Operators for making/killing of edge and face.

The delete face algorithm calls these operators to modify the topology of the model. The sequence of these operations in the delete face algorithm is in three stages.

In the first stage, the faces passed for deletion are removed and the boundary loops are capped with a single face or a set of faces. These are termed as rubber faces. In the second stage, as the recursion unfolds, these rubber faces are recursively subdivided into individual faces corresponding to the evolving solution graph as described in the previous section. Each individual face corresponds to an extension/contraction of a neighbor face. In the third stage, the feature solid is obtained using the newly created faces and the faces that are passed for deletion. The new faces are then merged with the corresponding neighbor face to obtain the updated model. These stages are shown in Figure 24. The individual stages are explained below in detail.

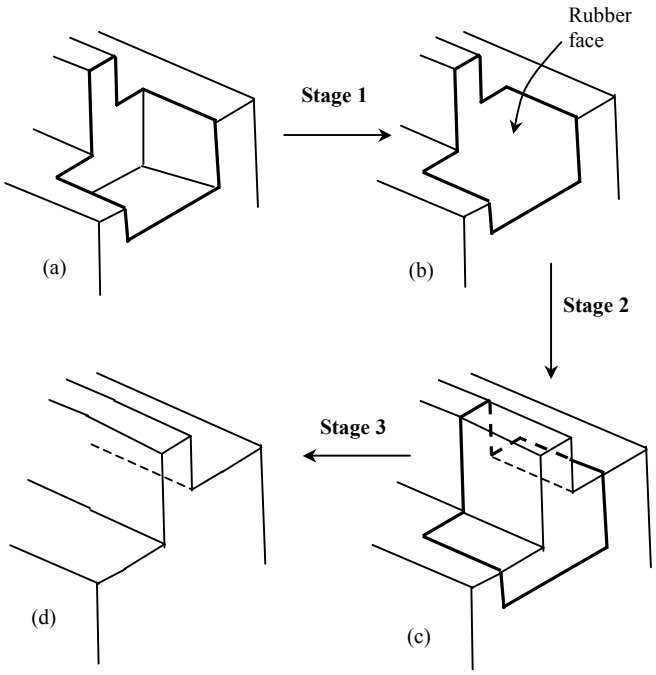

Figure 24 Figure showing intermediate stages of the delete face algorithm for the pocket feature in Figure 5.

Stage 1: *Create rubber face(s)*

In this stage, the faces to be deleted are removed from the model, and the boundary loops are capped using new faces without any associated geometry. Such faces are termed as *rubber* faces. Figure 24(b) shows the rubber face created during the suppression of the slot. At this stage, though the model is topologically valid, no geometry is yet associated with the rubber face. If there is a single boundary loop such as in Figure 24, a single rubber face is capped. However, if there are 2 or more boundary loops, it is possible to cap each of the boundary loops with a separate face *or* cap all boundary loops with a single face with multiple loops (refer section 4.1). Figure 25 shows the capping of two boundary loops with a single rubber face.

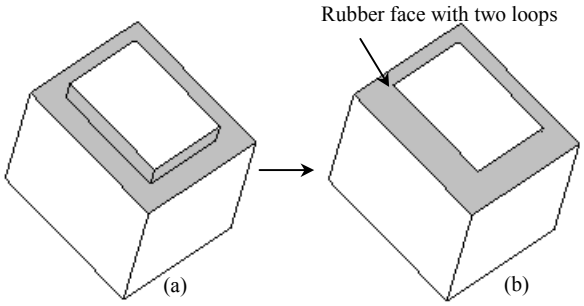

Figure 25 Creation of rubber face with two boundary loops.

Stage 2: *Find the solution graph*

In the second stage, the rubber face(s) are recursively subdivided into multiple faces using the recursive algorithm described in the previous section. The topology change is accomplished using Euler operators. The steps 3 and 4 of the algorithm use the MEV euler operator, while step 8 uses the MEF operator. Figure 24(c) shows the model after the delete face recursion has imposed the solution graph on the rubber face.

Each new face created in the algorithm corresponds to an extended/contracted portion of a neighbor face, and is assigned the corresponding face geometry. The inner edges and vertices of the solution graph are assigned geometry immediately after the corresponding Euler operation via which they got generated. In step 3 and 8, edge $e_{i'}$ is assigned the extended geometry of edge e_i. The geometry of vertex v_c is found as the point of intersection between the extended curve of edge e_i and the surface underlying F_c. In case of multiple points of intersection, the intersection point is selected as the point closest to vertex v_i. In step 4, the geometry of edges e_l and e_r are computed by intersecting the underlying surface of F_c with the surfaces of F_l and F_r respectively. In case of multiple curves of intersection, the curve that is closest to vertex v_c is chosen as the respective geometry.

An important aspect during geometry computations is the orientation of the newly created faces. As mentioned before, each newly created face is adjacent to one neighboring face and is assigned the corresponding face geometry. However, the orientation is based on whether the new face corresponds to an expansion or a contraction of the corresponding neighbor face. This is tested by choosing a point on the created face, and checking whether the point lies within the interior of the neighboring face. If the point is in the exterior, then we conclude that the neighbor face has expanded, and the orientation of the newly created face is same as the corresponding neighbor face. If the point is in the interior, then the neighbor face has contracted and the face normal is set to opposite orientation.

Stage 3: *Remove redundant edges and create feature solid*

The above face deletion algorithm creates an intermediate solid that contains new faces that cap the boundary loops. These faces along with the original deleted faces are stitched together to create the feature volume. This has been shown pictorially in Figure 5. During stitching, the faces belonging the visible face-set need to reversed in orientation before stitching with the newly created faces of the solution graph. The model can then be simplified by merging the newly created faces with the neighbor faces. This is also done using Euler operators that remove the edges and vertices along the boundary loop, and merge the newly created faces with the corresponding neighbor face. Figure 24(d) shows the final updated model after merging the newly created faces with the corresponding neighbor faces.

Figure 26 shows the individual stages during deletion of an uneven slot. During step 2 of the implementation as discussed in the previous section, new faces are created for each of the neighboring faces as shown in Figure 26(b). The new face G_2 fully overlaps with face F_2 and hence is set opposite orientation, while the other new faces G_0 and G_1 have the same orientation as the corresponding neighbor face. In stage 3, when these faces get merged with the neighboring faces, face G_2 contracts face F_2, while other faces expand their neighbors. Figure 26(c) shows the model obtained after merging the newly created faces with the adjacent neighbor faces.

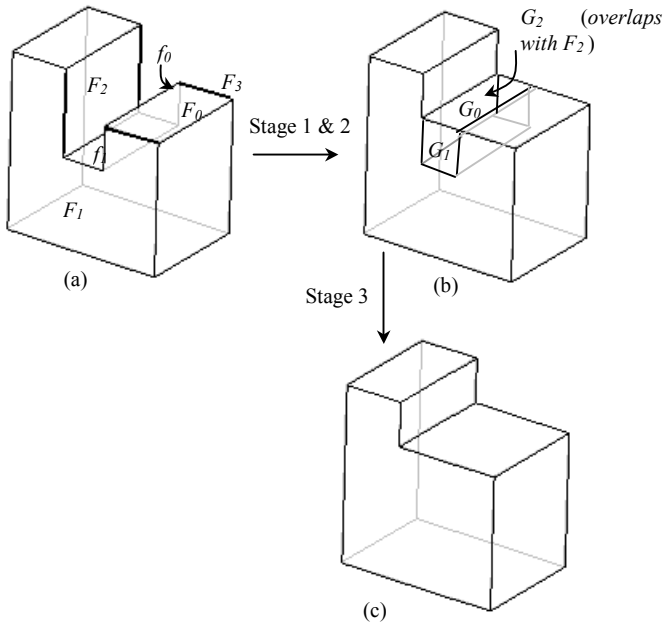

Figure 26 Shrinking of neighbor faces during delete face.

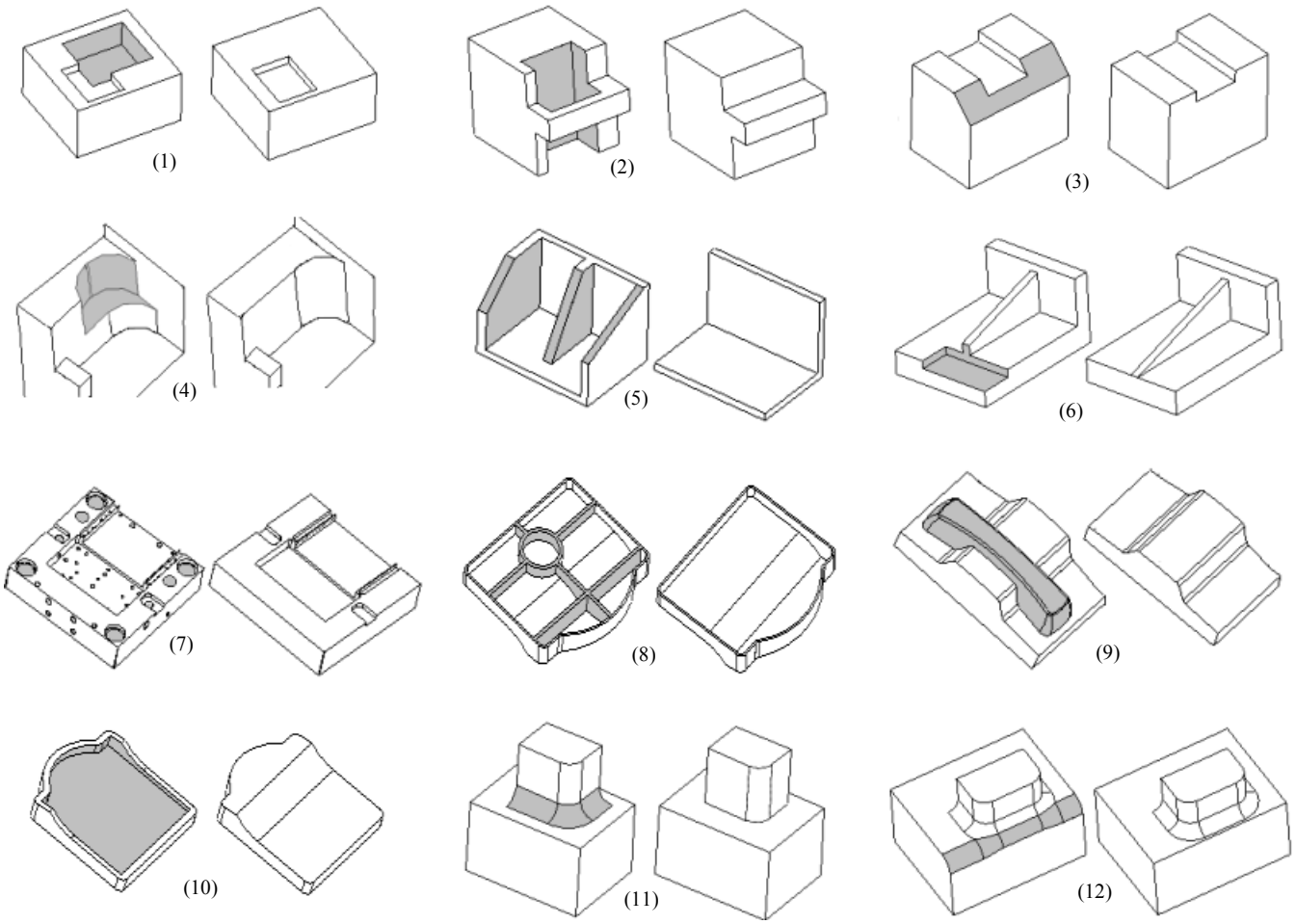

Figure 27 Examples showing the working of the delete face operation in 12 parts.

6. EXAMPLES

Figure 27 shows several examples that demonstrate the working of the delete face algorithm. Each example shows the input model and the final model after the delete face algorithm. The faces passed for deletion are shown in grey. It may be noted that some of these examples such as example 3 do not work in any existing geometry kernel, but works correctly in our algorithm.

Example 1 shows the deletion of a pocket interacting with another pocket. Example 2 shows a through pocket interacting with ribs on either sides and having two boundary loops. The two boundary loops are solved independently as separate loop problems. In this example, neighboring faces are merged during the delete face operation. Example 3 shows an example of slot through a chamfer. The chamfer is passed to the delete face algorithm to get back the slot feature. This example involves creation of a concave feature volume during the delete face opeartion.

Example 4 and 5 show examples in which faces contract during the delete face operation. Example 4 is a complex *virtual* slot in which a face of the slot is shared by other features in the model. The non-virtual faces of the slot are passed to the delete face algorithm. The virtual face is treated as a neighbor face and contracts during the delete face algorithm. Example 5 shows three ribs that have virtual faces. The virtual faces of the ribs are not passed to the delete face operator. Example 6 shows a pocket on a rib. The pocket is deleted to get back the rib feature. Two degenerate vertices are created in this operation.

Examples 7 to 12 show the deletion of several other types of features. Example 7 shows the deletion of all holes in a part. This is usually done in the beginning of feature recognition to simplify the part. Example 8 shows the deletion of a rib-network while example 9 shows the deletion of a free-form protrusion feature. Both these examples involve merging of several neighboring faces. Example 10 shows the usage of the delete face operator to unshell a thin-walled solid.

Examples 11 and 12 show the deletion of blends using the delete face operator. Example 11 shows a complex blend chain in which a blend chain rolls on another chain. During deletion of the blend chain, the other blend chain is partially reconstructed. At this point, we remark that although the algorithm presented in this paper can be used to handle blend features, it could fail for complex blend chains. This is because finding the opposite face in step 2 of the algorithm involves many tangential intersections that are unstable in nature. Blend deletion is best handled using a separate algorithm that is considered in a separate paper.

7. SUPPRESSION OF 2.5D FEATURES

So far we have discussed a general algorithm for suppressing any volumetric feature from a model. However, the problem can be greatly simplified when the nature of the feature and its neighborhood is well understood. In this section, we show how the delete face operation is simplified for parts that are purely made up of 2.5D features. These features arise due to extrusions of profiles between two levels, and are probably the most commonly used kind of features in manufacturing parts.

Figure 28 shows an example of a pocket that has been made at an intersection of crossing ribs. The boundary of the pocket is fairly complex, and the delete face algorithm would need powerful heuristics to suppress the pocket, and reconstruct the right feature volume. However, the problem can be viewed much more easily when the part is seen in the "top view" as shown in Figure 29. In this projected view, the part is represented as a tiling of faces connected in a two dimensional plane. Each face has a level associated with it that corresponds to the height at which the bottom face of the pocket feature is located. The vertical faces of the pocket reduce to edges in the projected view. The problem of delete face of a pocket feature reduces to deleting a face in the plane by extending the neighboring faces (and levels) in the same plane. The set of newly created faces caused by the extensions/contractions of the neighborhood can then be pulled back in 3D to create the feature volume.

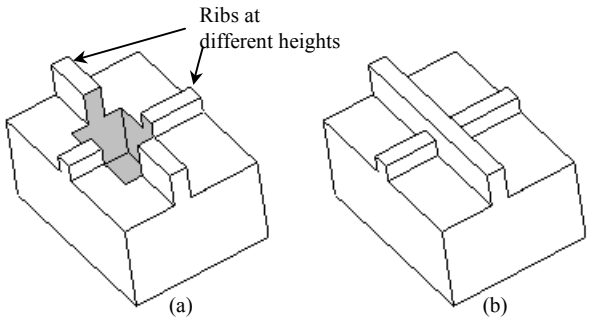

Figure 28 Delete face operation of a 2.5D part.

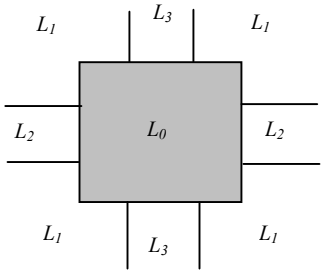

Figure 29 Projected view of the part from top. The various levels L_0, L_1, L_2, L_3 shown are in ascending order.

The algorithm of delete face in two dimensions also begins similarly at an external edge of the face. Instead of an opposite face that was found for the 3D delete face problem, an opposite edge is found in the 2D delete face problem. Unlike in the 3D case, the opposite edge can be found predictably without the need of clever heuristics. This is because of the limited number of configurations possible when two external edges meet at a vertex. The three cases that are permissible are shown in Figure 30. Figure 30(a) shows the case when two external edges terminate at a vertex. In this case, the faces on either side of the edges are at the same level. Figure 30(b) and 30(c) shows the case when two external edges meet at a vertex and one of the edges extends beyond the vertex. In this case, the external edges have a common adjacent level as shown. Due to these limited configurations and due to the fact that edges cannot cross each other in the projected view, the opposite edge is predictably found as the closest external edge that matches these constraints.

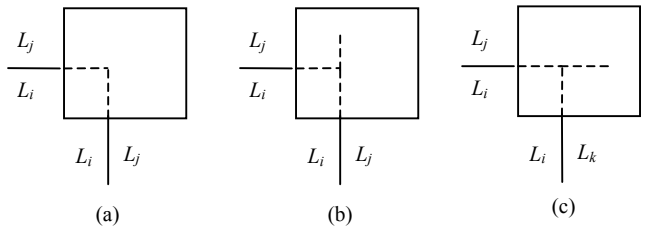

Figure 30 The possible configurations when external edges intersect after extension.

The computation of the vertex sense is also greatly simplified in most situations. When the external edges terminate at a vertex, there is no ambiguity for the vertex sense. In the other case when the two edges intersect and one of the edges extend furthers, there are two possible configurations depending on the edge that gets extended beyond the vertex. Even in this case, the configuration is determined completely if the common level L_i is the lowest or the highest level among the three levels. In this case, the edge that is adjacent to the higher level extends from the intersection vertex. The only ambiguous situation arises when two external edges intersect and the middle level is in between the two side levels. In such a situation, it is possible for either of the edges to extend beyond the vertex, and hence there are two possible choices for the vertex sense. The configuration is then chosen based on similar heuristics as described in the previous section. For the example shown in Figure 29, the final configuration is completely determined without any heuristics. This is shown in Figure 31 below.

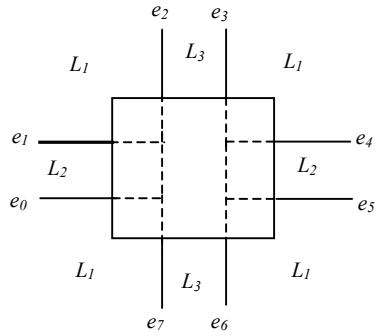

Figure 31 Solving the delete face problem in two dimensions.

The final configuration is determined starting at external edge e_1. The first edge intersecting this edge meeting the constraints is

edge e_2. Since level L_3 is higher than level L_2, edge e_2 extends beyond and proceeds downwards. The edge then meets e_0 and proceeds further downward since L_3 is also higher than level L_1. Finally it then merges with edge e_7. The other set of external edges is also similarly solved resulting in the final solution graph.

8. CONCLUSION

This paper presents an algorithm to delete a set of faces of a feature from a model, and create the corresponding feature volume. This operation is very useful in feature recognition algorithm in suppressing the already recognized features, and thereby enabling recognition of interacting features. Apart from feature recognition, the delete face operation is also useful for local editing and modification of parts. For example, deletion of certain types of features such as holes and fillets can be used to simplify the part and aid downstream applications.

In literature, several algorithms have been proposed and implemented to solve the delete face problem. However, the solutions presented so far are mainly heuristic in nature and work for simple and specific kinds of feature volumes. Our paper is unique in this regard, and provides a systematic algorithm to remove face-set features of arbitrary topology and geometry. Heuristics are used at clearly defined steps, and they are directly related to important characteristics of the final topology and geometry. These heuristics have been implemented as separate functions, and are independent of the basic recursive algorithm. Moreover, our algorithm uses a divide-and-conquer approach, and hence is computationally less expensive than other presented algorithms. Furthermore, unlike previous approaches, the face deletion and the feature volume construction take place in a single step without the need of a separate Boolean operation.

The delete face algorithm has been tested on a wide variety of parts. Both depression and protrusion features are handled by the algorithm. The algorithm allows extension, contraction and merging of neighbor faces during face deletion. Many situations such as multiple boundary loops and degeneracy are handled robustly by our algorithm.

In many situations, prior knowledge of the feature volume and the neighborhood can be used to simplify the algorithm to a large extent. This has been demonstrated for 2.5D features in this paper.

9. ACKNOWLEDGMENTS

We would like to thank Geometric Software Solutions Ltd. (GSSL) for extending co-operation in promoting the research. We would like to specially thank Dr. Vinay Kulkarni and Anil Risbud for their encouragement throughout the project.

10. REFERENCES

[1] Shah, J.J., Manyla, M. amd Nau, D., eds. *Advances in Feature Based Manufacturing*, Elsevier/North-Holland, Amsterdam. 1994.

[2] Ji Qiang, Marefat M., Machinable Interpretation of CAD Data for Manufacturing. *ACM Computing surveys*, 1997. 24(3), 264-311.

[3] Tseng, Y. J. and Joshi, S. B., Recognizing multiple interpretations of interacting machining features. *Computer Aided Design*, 1994. Vol. 26(9), 667-688.

[4] Dong, X. and M. Wozny, "Feature Volume Creation for Computer Aided Process Planning," in book Geometric Modeling for Product Engineering, Elsevier Science Publishers, B.V. (North-Holland), Edited by M. Wozny, J. Turner and K. Preiss, pp. 208-223.

[5] Karinthi, R., and D. Nau, "An Approach to address Geometric Feature Interactions in Concurrent Design", *Proc. ASME Computers in Engineening Conference*, Boston, August 5-8, 1990, Vol. 1, pp. 243-250.

[6] Sashikumar V., Sohoni M., Graph based framework for recognizing user defined features. *ACM Symposium on Solid Modeling and Applications*, June 2001, Ann Arbor, Michigan.

[7] Nalluri Rao SRP, Form feature generation model for feature technology, PhD thesis, Department of Mechanical Engineering, Indian Institute of Science, Bangalore, India, 1994.

[8] Sakurai, H. and D. Gossard, "Recognizing Shape Features in Solid Models," *IEEE Computer Graphics & Applications*, September, 1990, pp. 22-32.

[9] Henderson, M. R., Extraction of Feature Information from Three dimensional CAD Data, Ph.D. Dissertation, Purdue University, May 1984.

[10] Vandenbrande, J. and A. A. G. Requicha, "Spatial Reasoning for Automatic Recognition of Interacting Form Features," *Proc. ASME Computers in Engineering Conference*, 1990, pp. 251–256.

[11] Dong, X. and M. Wozny, "A method for generating volumetric features from surface features," *ACM Symposium on Solid Modeling and Applications*, 1991.

[12] Sakurai, H. and D. Gossard, "Shape Feature Recognition from 3D Solid Models," *Proc. ASME Computers in Engineening Conference*, San Francisco, August 1-4, 1988, Vol. 1, pp. 515–519.

[13] Sandiford D., Hinduja S., Construction of feature volumes using intersection of adjacent surfaces. *Computer Aided Design,*2001. Vol 33, pp 455-473.

[14] Mäntylä, M. (1988). *An Introduction to Solid Modeling. Principles of Computer Science*. Computer Science Press, Maryland, U.S.A.

[15] Parasolid, Functional Description Manual, Version 11, Unigraphics Solutions, (www.parasolid.com) May 2000.

[16] ACIS Geometric Modeller, Format Manual, Version 6.0, Spatial Technologies, (www.spatial.com) June 2000.

[17] Feature Recognition Library, Version 11, Geometric Software Solutions Limited, (www.geometricsoftware.com) October 2001.