Proceedings of DETC'02
ASME 2002 Design Engineering Technical Conferences
and Computer and Information in Engineering Conference
Montreal, Canada, September 29-October 2, 2002

DETC2002/CIE-34494

FEATURE SIMPLIFICATION IN SURFACE MODELS FOR EFFICIENT FINITE ELEMENT MESH GENERATION

Nikhil Joshi

Department of Mechanical Engineering
University of Michigan
Email: njoshi@umich.edu

Debasish Dutta

Department of Mechanical Engineering
University of Michigan
Email: dutta@umich.edu

ABSTRACT

Sheet metal components are typically modelled as freeform surface models. Finite element meshes generated automatically for such models have poor quality around small detailed features. These features need to be simplified in order to obtain an acceptable mesh. Simplification involves recognition of the feature and modification of its geometry or complete suppression of the feature. This paper proposes techniques to directly query the CAD data structure to recognise and suppress two basic features, viz. holes and fillets in freeform surface models. Results of a software implementation for the same are discussed with suitable examples.

1 INTRODUCTION

Finite element analysis is an important step in the validation of a design. Generation of the finite element mesh is a tedious process. Many a time, certain features are unimportant for finite element analysis and can safely be suppressed without affecting the accuracy or validity of the results. Recognition of such features and their selective suppression or simplification, the topic of this paper, greatly reduces the efforts expended in the generation of an acceptable quality mesh.

Geometric models store design information in the form of points, curves and surfaces. However, the decision making and reasoning processes of most engineering tasks require functional entities and attributes, such as "holes", "distance between holes", etc. that are not explicitly available. Features are geometric or topological patterns of interest in a part model, which represent high-level entities with respect to their engineering significance [1]. Recognition and extraction of features from a geometric model is, therefore, required for various design and manufacturing activities. Features are widely used for tasks such as automatic process plan generation, classification of part families [2], reconstruction of solid models from scanned data [3], etc.

Sheet metal components, such as automobile body parts, are typically modelled as freeform surface models. Finite element analysis is used to verify their structural strength and deformation characteristics. Automatic mesh generation algorithms do not generate meshes of acceptable quality, especially near small features in the surface such as holes, fillets, beads, bosses, etc. The automatically generated mesh, therefore, needs to be corrected manually. This manual process is extremely tedious and time consuming, owing to which, approximately 75% of the time required for FEA is spent in mesh generation. Automatic recognition of such features and their selective suppression or simplification can, therefore, greatly reduce the efforts of mesh generation. In this paper we propose new techniques to detect and suppress two commonly occurring features, viz. *holes* and *fillets*, using definite rules to analyse the geometry of the models.

The remainder of the paper is organised as follows: In section 2, we review existing literature related to features. In section 3, we explain the methodology we use to detect features in such models. We also present examples of its implementation on functional parts to highlight the salient points and limitations of the technique. In section 4, we discuss the implementation of the algorithms and the overall architecture envisaged for the feature recognition system. We conclude in section 5 with a summary of the work done and the scope for future work on these techniques.

2 PREVIOUS RELEVANT WORK

To the best of our knowledge, the topic of feature modification or simplification in surface models has not received much attention in academic literature. Feature simplification requires recognition of the feature as a preliminary step. Many methodologies have been proposed for the recognition of form features in solid models. Depending upon the feature representation scheme adopted, they can be divided into two categories, viz. surface and volume feature recognition schemes. Surface recognition schemes assume that a feature is formed by specific arrangement of faces that satisfy certain conditions. Graph-based methods, syntax-based methods, rule-based methods, hint-based methods or geometric reasoning, etc. are examples of such schemes. Only a few systems characterise a feature by its volume. These include volume decomposition methods, the convex hull algorithm and backward growing.

Most of the graph based systems use the notion of an Attributed Adjacency Graph (AAG) [4]. In these methods the topology of the model is represented in the form of a graph. The recogniser then scans the object's graph to find sub-graphs that match the description of a feature. Syntactic pattern methods [5] use sequences of geometric elements to describe features. Such a sequence of geometric elements is represented as a string of codes. A parser checks whether any sub-strings can be generated by a grammar describing a feature. Rule based methods [6] use sets of rules to describe features. The rules enumerate the necessary and sufficient conditions for a portion of a model to be identified as a feature. The scope of feature instances that can be recognised can be very wide. However, formulation of rules is tedious and subjective. Scanning the model for regions that satisfy the rules can be time consuming, especially for complex models. In the convex-hull method [7], the difference between the object and its convex hull is computed recursively. The object is finally represented as a sequence of convex volumes with alternating signs. Volume decomposition [8] is based on decomposing a delta volume into cells, which are further processed and mapped to standard machining features to form the cells. Volumetric representation schemes, in general, enable a more complete description of feature-feature interactions.

In addition to these methods, heuristics and techniques from artificial intelligence, such as genetic algorithms and artificial neural networks (ANN) have also been employed to solve the problem of feature recognition. Neural networks have used in conjunction with Attributed Adjacency Graphs [9], [10] to recognise feature classes, such as pockets, slots, protrusions, passages, steps, and holes, in solid models. Neural networks have also been trained to recognise features in 2D components [11], [12], though the work has been restricted to flat 2D surfaces.

All the schemes discussed above are, as yet, only able to detect features composed of planar and cylindrical surfaces in solid models. Most of these generic methods cannot be easily extended to include features on freeform surfaces. Formulation of description patterns to define features on surface models is a subjective process. Neural networks are better suited for this task since they can learn from examples, i.e. training patterns. However, neural networks are not suited for complex numerical computations. CAD systems store the models as a mathematical description of constituent faces and edges. Thus, a pre-processing step is required to convert the CAD model into a form of input suitable for the neural network. Determining the representation that will be most useful to recognise features is the major hurdle in the use of neural networks in freeform surface models. As a preliminary technique, we resort to defining rules to describe features on freeform surfaces.

3 DIRECT MANIPULATION OF CAD DATA STRUCTURE

Sheet metal parts have freeform shapes that are obtained by processes such as form pressing, drawing, bending, etc. These parts have a very small thickness and are generally modelled using surfaces to specify the shape of the part. In B-Rep models, NURBS surfaces are used to represent the complex freeform shapes. The surface model is made up of a large number of NURBS patches (called faces) that are trimmed to specify the outer boundary. Adjoining patches are stitched to each other so as to form a single contiguous part. Owing to this, different designers may construct the same feature in various different ways. e.g. a hole boundary may be defined by a single edge or sequence of many edges. Consequently, it is almost impossible to define some features based on the topological relationships between the faces, as is done in "graph based" and "hint based" approaches. Volumetric approaches cannot be used, as surface models are non-manifold, i.e. they are modelled as surfaces having zero thickness and consequently have no volume. ANNs and other approaches require the feature to be presented in special formats in order to be recognised. This involves locating the region containing the feature (usually done manually) and then presenting it to the ANN in the required representation for identification classification and parameterisation. Directly querying the data structure of the CAD model ensures that none of the information about the geometry of the part is lost through any change of representation. It also requires minimum human intervention during the feature recognition stage. However, the technique has similar shortcomings to any other "rule based" feature recognition method. i.e. scanning the entire model by the set of rules can be very time consuming, especially if the object is complicated with many faces.

We make the following assumptions about the CAD data that we receive as input:

1) Input is the form of a B-rep model
2) Trimmed surfaces have been defined
3) There are no overlapping surfaces
4) NURBS representation is available for all surface patches

3.1 SIMPLIFICATION OF *HOLES*

3.1.1 DEFINITIONS AND CONCEPTS:

In B-Rep models, the surface model is made using a large number of trimmed surface patches, which are stitched at the coincident edges. Each patch is stored in the data structure as a *face*. Each *face* has one outer *loop* and any number of inner *loop*s. *Loop*s are sequences of *edge*s, which define the trimming

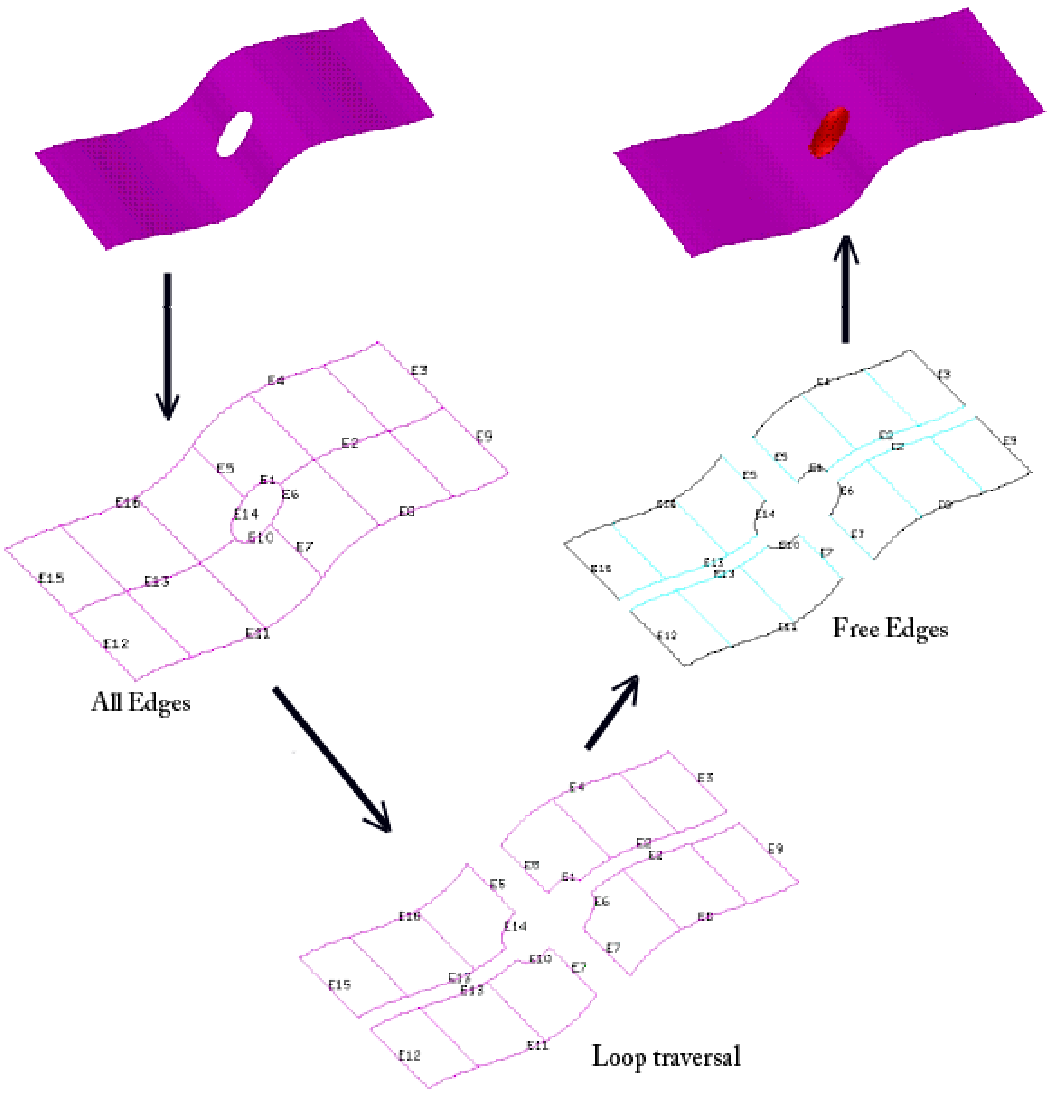

Figure 1 - Suppression of holes

curves for the underlying NURBS surface. *Edge*s in turn point to the *loop* they belong to. We define a *hole* to be any loop of edges that has no surface on the inside. All the edges in a B-Rep surface model will either be shared by 2 surface patches (internal edge) or will have a surface on one side and none on the other. Such an edge, with a surface on one side only, is called a *free edge*. In addition, to the assumptions already mentioned, we also assume that surface patches are stitched so that no gaps exist between them. If they exist, they will be treated as intended holes.

3.1.2 RECOGNITION OF HOLES:

We have already assumed that the geometry is cleaned and there are no overlapping surfaces. It can then be observed that edges forming the boundary of a hole are characterised by having a face only one side, while all other edges will be sandwiched between two faces. We have already defined such edges as free edges. The algorithm implemented initially finds all such free edges in the model. It then forms closed chains of these edges. Each such closed loop of edges corresponds to a hole. The algorithm used to detect holes is enumerated below:

Algorithm:

Input – B-Rep surface model with holes
Output – B-Rep surface model with patches covering original holes

1) Get set F of all free edges in the model
Get number of edges of model
For each face get all loops
For each loop traverse through all the edges of the loop
Keep count of the number of times each edge was encountered
All edges that were encountered just once are free edges
2) Select and edge e ∈ F

3) Find start and end vertices of e
4) Find edge starting from end vertex of e and ∈ F
5) Find chain of edges till end vertex of chain = start vertex
6) Repeat steps 2 to 5 to define new holes till all edges in F are exhausted

The working of the algorithm can be best studied with the help of a simple example. Consider the model shown in Figure 1, containing one hole. The algorithm first tags all edges of the model to identify each edge uniquely. The model is made up of four trimmed and stitched faces. For each face, the algorithm traverses all the loops and keeps a record of the number of times an edge is encountered. As can be seen in the figure, edges E2, E5, E13 and E7 are encountered twice. All other edges are encountered only once, and are thus marked as free edges. In order to chain the free edges into holes, it starts with an arbitrary edge, say E1, which belongs to the set of free edges. It then seeks another free edge from the remaining edges that shares the end vertex of E1, which in this case will be edge E6. The process continues till the end vertex of the new edge is the same as the start vertex of the original edge. The sequence of these edges (E1 - E6 - E10 - E14) is stored as the definition of a hole. The perimeter of the hole is also calculated as the sum of individual edge lengths. If there are any more free edges remaining, the process is repeated till all free edges are accounted for. The hole with the largest perimeter is designated as the outer boundary and not considered for simplification. Thus, in Figure 1, the sequence (E3 – E4 – E16 – E15 – E12 – E11 – E8 – E9) will initially be detected as a hole, but will eventually be removed from further consideration.

It should be noted that the detection of free edges cannot be done by merely querying the edge for number of loops it belongs to. It might be possible that an edge is contained in only one loop, but is not a free edge. E.g. In the case of a cylindrical face, there exists an edge that runs longitudinally along the cylindrical surface. This edge has the same face on both sides and is obviously contained in only one loop. However, it isn't a free edge. To detect the free edges we need to traverse all the loops and find the number of times each edge is encountered during the traversals. All edges that are encountered only once will be free edges. It can be seen that the longitudinal edge on the cylindrical face will be traversed twice in opposite directions in the same loop. Certain checks are also required to eliminate degeneracies and special conditions e.g. edges with '0' length showing up as holes.

3.1.3 SIMPLIFICATION OF HOLES:

Simplification of a hole will consist of suppressing the hole, i.e. covering the hole by a smooth surface patch. Trying to estimate the internal shape of a surface given its surrounding surfaces is a difficult problem. However, in certain cases, we can use the existing surface patches to predict the surface that would cover the hole. If the hole to be suppressed is the internal loop in any one face, then the mathematical representation of the required interpolation surface already exists and is merely suppressed by the corresponding trimming curve. The suppression of the hole then entails the removal of this trimming curve. Alternately, we can create a new face, having the same underlying surface equation, with edges of the hole as the outer trimming curve. If the edges of the hole belong to separate patches, we can still try to use their underlying surface equations. If the underlying surfaces form a watertight intersection, then we can remove the trimming curve corresponding to the hole and trim the surfaces at their intersection instead. It might further be necessary to stitch and blend the faces at the intersection to maintain continuity of slope. Alternatively, we can loft a new surface using the edges of the holes as an outer loop and adding tangency constraints for the adjacent faces.

In our implementation we create new surface patches for all holes, whose perimeter is below a user-defined threshold. The threshold value of the perimeter needs to be determined by the user and will depend upon the fineness, i.e. the mean element size, of the finite element mesh that will be constructed. Smaller elements will usually mesh large holes adequately and, therefore, only small holes will need to be suppressed. Using the definition of the boundary of the holes, it tries to loft a surface that would cover the hole. Consequently, the limitations of the lofting operation of the native CAD software will extend to the creation of surface patches. The patches are stored as separate entities that can be stitched to the original part by the user.

We present, here, the results of the algorithm obtained when tested on three separate sheet metal models. The threshold perimeter for patching holes was kept at 0.1 inches for all samples, however the program allows the user to modify this threshold value depending upon the size of elements that will be used for meshing. As can be seen from Figure 2 (a), the sample part has 14 holes of various sizes on different faces. Figure 2 (b) shows the patches that have been created to patch the holes. All 14 holes were detected by the program. As can be seen, patches were not created for two of the holes as their perimeter exceeded the threshold value. All holes, except the two holes on the side face (indicated in Figure 2 (a)), were situated on planar faces, and can be joined to the original part. Patches were created for 12 holes including the ones on the non-planar surface. Two holes were not suppressed owing to their large perimeters.

Figure 3 (a) shows another specimen part with 26 holes. Figure 3 (b) shows the patches created by the program for the specimen. Patches have been created for 24 holes, while two holes were found to have perimeter greater than the threshold. It is important to note that the algorithm has successfully created patches for holes of arbitrary shapes as shown in Figure 3 and is thus not confined to circular holes.

Figure 4 (a) shows a complex sheet metal model having 30 holes. Figure 4 (b) shows the result obtained from the program. All 30 holes were recognised by the program. Patches were created for 21 holes. Two holes, viz. holes labelled 'A' and 'B' in Figure 4, were found to have perimeter exceeding the threshold value. The remaining 7 holes could not be patched due to limitations of the lofting command, e.g. hole 'C' in Figure 4. The boundaries of the hole can, however, be used by the user to create a new surface.

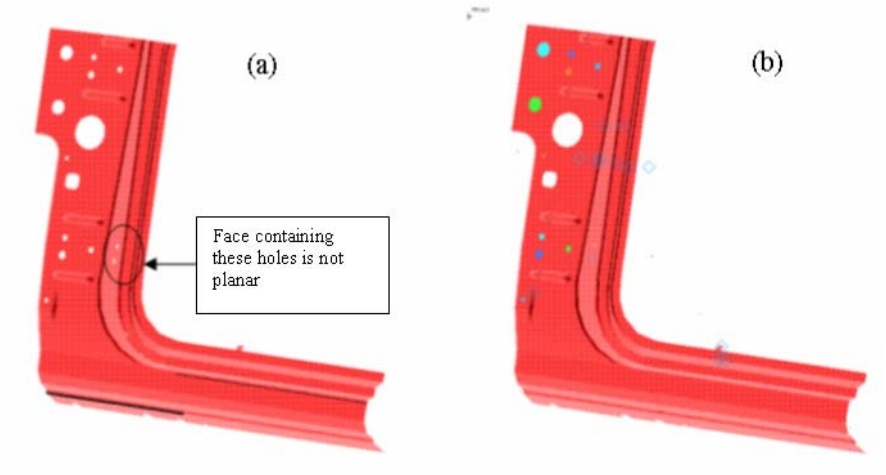

Figure 4 - Sample Part #1 - Simplification of holes

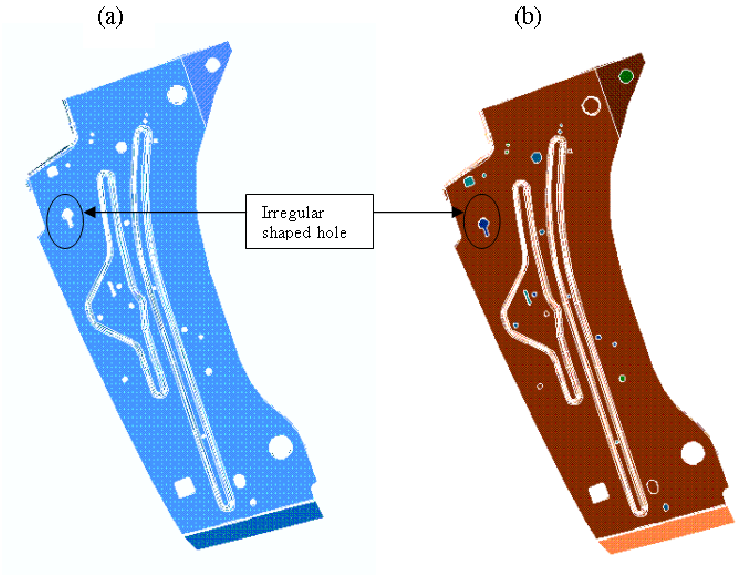

Figure 2 - Sample Part #2 - Simplification of holes

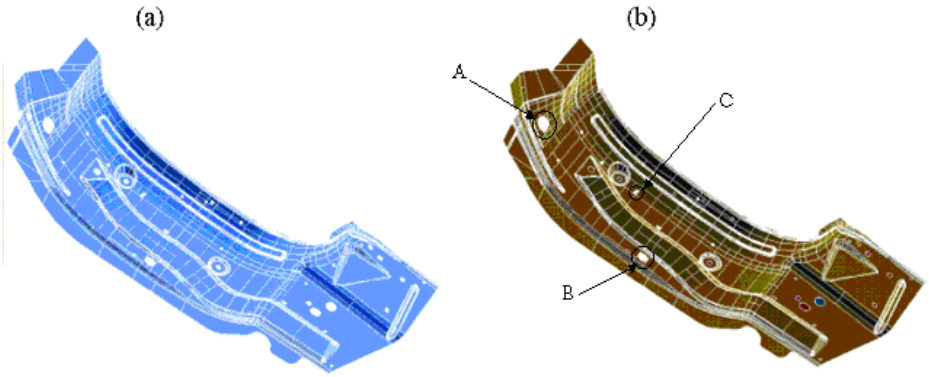

Figure 3 - Sample Part #3 - Simplification of holes

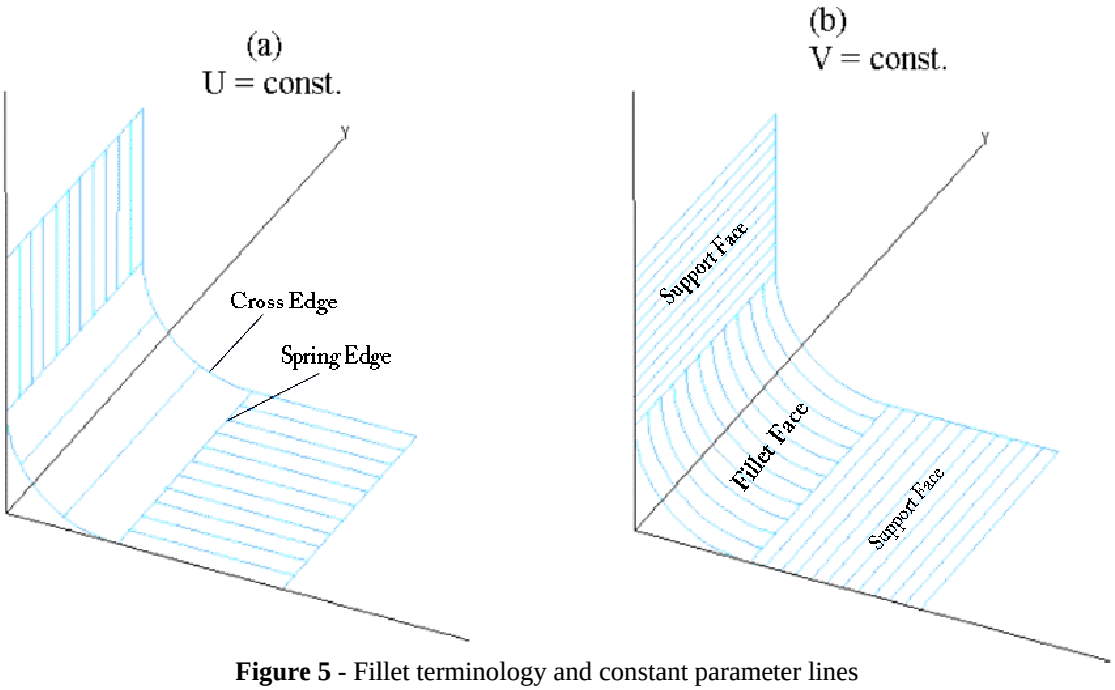

Figure 5 - Fillet terminology and constant parameter lines

3.2 SIMPLIFICATION OF *FILLETS*

3.2.1 DEFINITIONS AND CONCEPTS:

A *fillet* or *blend* can be defined as a curved surface (or a chain of curved surfaces) that links surfaces together to form a continuous smooth bend (see Figure 5) [13], [14]. The blend geometry is computed using an imaginary rolling ball that maintains contact with the surfaces to be blended. The side faces of the blend (which support the imaginary rolling ball) are called *support faces*. The locus of points traced by the rolling ball centre is called the *spine curve*. The edges of contact between the blend face and the support faces are called *spring edges*. The edges that connect adjacent faces in a blend chain are called *cross edges*. The blend surface usually has a circular cross section, and the plane of the cross section is always perpendicular to both the faces. The blend *curvature* at a point on the blend surface is the curvature along the cross-section of the blend surface at that point. The cross-sectional radius may be constant throughout the surface, called *constant radius blends*, or may vary at different cross-sections, called *variable radius blends*. Recognition of blends in freeform surface models is difficult since topologically a fillet is no different from a normal face. Moreover, there might be a surface patch where only a portion of the patch forms the fillet. The variation of curvature at various points on the face provides the only clues to detect the existence of a fillet. Accordingly, we make the following assumptions about the fillets that the algorithm can detect:

1. Blend faces are parameterised such that the blend radius is along one parametric direction.
2. The blend face describes only the geometry of the blend.

3.2.2 RECOGNITION OF FILLETS:

Most of the present day modeling softwares store information about the history, i.e. the steps involved in the creation of the model, in the form of a chronological tree structure. If such a *history tree* is available for the model one can search for filleting operations in the tree and suppress them. However, this approach has its shortcomings. Firstly, operations such as stitching and geometry healing, which are common in surface models, result in loss of history and hence the history tree may not be reliable to detect all fillets. Secondly, in the creation of surface models the fillet faces themselves may be used as references to create newer faces. Suppression of a filleting operation may, therefore, disallow creation of these faces and thus result in an invalid object upon reconstruction using the modified history tree. Moreover, the history tree for a model is not unique. It depends upon the way in which the model was generated by the designer. Many a time, a gap between two neighbouring fillets of different radii is connected by a lofted face, with variable radius, rather than an explicit filleting operation. Such a face should also be recognised as a fillet for correct suppression of fillets.

The blend recognition technique used by us collects information about the variation of curvature across a face to determine whether it is fillet face. The geometry of each face in a B-Rep model is stored as a NURBS surface representation. A NURBS surface is parameterised is two directions, usually named U and V. For every face the algorithm essentially steps in U and V directions and calculates the curvatures of the constant U and constant V curves at various points on the surface. If the surface is given by X(U,V), where U and V are the parametric variables then the curvatures at any point are given by

$$K_U = \frac{\left|\frac{\partial X}{\partial U} \times \frac{\partial^2 X}{\partial U^2}\right|}{\left|\frac{\partial X}{\partial U}\right|^3} \quad \& \quad K_V = \frac{\left|\frac{\partial X}{\partial V} \times \frac{\partial^2 X}{\partial V^2}\right|}{\left|\frac{\partial X}{\partial V}\right|^3}$$

If it is found that for every constant U curve, K_v remains constant as we step in V direction and is greater than a predefined threshold, the face is marked as a potential fillet in V direction. The threshold for the curvature is calculated depending upon the bounding box for the object so that fillet recognition is independent of the scale of the model. If the curvatures of all such U curves are the same, then it is a uniform radius fillet; otherwise it is marked as a variable radius fillet. Likewise similar observations for V curves lead the face to be marked as a potential fillet in U direction. For each face that is a potential fillet, the program attempts to find spring and cross edge. The face is classified as a fillet only if the curves in the direction of the fillet subtend an angle between 0 and 180 at the spine curve. Additional checks are included to prevent detection of parts of cylindrical faces from being detected as fillets. E.g. The fillet face in Figure 5 is a constant radius fillet in U direction.

Spring and cross edges are identified by comparing the curvatures along the edge and perpendicular to the edge for face under consideration and its adjacent face sharing the same edge. If the curvature along the edge is equal on both faces and corresponds to the radius of the fillet, then the edge is a cross edge. If the curvature in a direction normal to the edge is greater than the threshold and greater than the curvature of the adjacent face in the perpendicular direction, the edge is classified as a spring edge. Presently, the program checks the curvature properties only at the midpoint of the edge under consideration. Once the spring and cross edges are detected, the entire information of the blend, i.e. type of blend, radii at each cross-section, angle subtended by blend, list of spring and cross edge, is written as attributes to the face which further used for chaining and simplification.

Chaining and sequencing of blends is essential to determine the sequence in which the blends must be suppressed so that a valid object is obtained. As can be observed in blend-blend interactions, when a blend precedes another, it acts as a support face for the new blend. Thus the shared edge is a cross edge for original blend face and a spring edge for the new blend. This allows us to formulate rules to determine the sequence of blends. In order to chain blends, the program essentially queries each of the adjoining faces of a fillet face recursively to find out whether it is a blend, and if yes, whether it was created in the same filleting operation or whether it preceded/followed this fillet. The information about the chain to which a face belongs is recorded as another attribute and the process is continued till all the blend faces are accounted for. The algorithm for detection of blends is charted down below.

Algorithm:

Input – B-Rep surface model with blends
Output – List of fillet faces and ordered chains of blends and file containing only fillet faces

Algorithm to find fillet faces
1) For each face in the model get its NURBS representation
2) Evaluate curvatures in U and V parametric directions at equal intervals on the surface
3) If the face satisfies criteria for being a fillet face
4) Determine type of fillet
5) Determine its spring and cross edges
6) Create chronologically ordered chains of fillet faces

Rules for creating blend chains
1) If the edge between adjoining fillet faces is a cross edge for both faces, then both faces were created in the same filleting operation.
2) If the edge between adjoining fillet faces is a cross edge for one and spring edge for another, then the face for which it is a cross edge was formed in an earlier filleting operation

3.2.3 SUPPRESSION OF FILLETS:

In order to suppress the fillets, we need to calculate the intersection edge that existed before the blend was made. For any given cross-section of a fillet, it is possible to calculate the intersection point knowing the tangential directions and extremities of the blend. As can be seen from Figure 6, we can calculate the angle (α) subtended at the centre using the formula

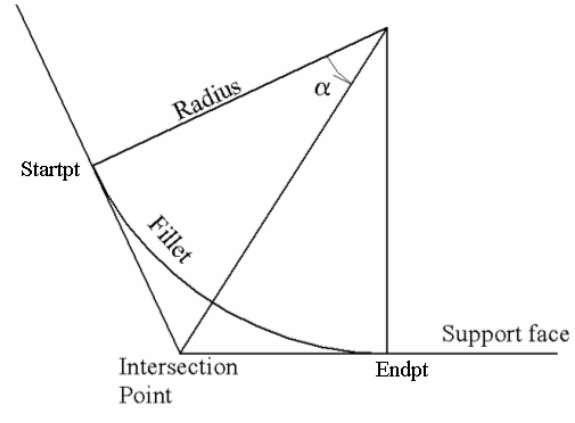

Figure 6 - Cross section of a fillet

$$\cos\alpha = \frac{\left|\left(\frac{dX}{dt}\right)_{Startpt} \bullet \left(\frac{dX}{dt}\right)_{Endpt}\right|}{\left|\left(\frac{dX}{dt}\right)_{Startpt}\right|\left|\left(\frac{dX}{dt}\right)_{Endpt}\right|}$$

…..where X(t) is the curve of the blend cross section.

The distance of the intersection point from the start point or end point is given by

$$extension = radius \times \tan\alpha$$

Using the distances from the Start point and the End point, the program determines the point of intersection. The program calculates these intersection points for various cross-sections of the blend chain. It then passes a spline interpolation curve through all these points to estimate the intersection curve. The original blend face is then removed from the model. It is replaced by new faces that are created by skinning surfaces between the spring edge and the spine curve for each spring edge. Constraints are added to ensure that the new faces will have tangent continuity with the support faces.

It should be noted here that the intersection curve so calculated, might not be the actual intersection that existed before the filleting operation. This is because the effects of curvature of the freeform support face in the blended region are not accounted for. It may be sometimes possible to find the underlying surface geometry of the support face and calculate the new face using that geometry and appropriate trimming curves. However, as indicated before, all the faces that are detected as fillets might not be generated by filleting operation and thus the surface representation for the required region may not exist.

Presently only the latest blend chain can be suppressed in this way. This is because during suppression a new face may be created that would be a part of the preceding blend. This face is not included in the list as a part of the preceding blend chain since it did not exist then. One way to account for this is to run the detection algorithm again to detect these new blend faces. However, this results in unnecessary repetition of work. Another way to overcome this problem is to make a note of the preceding blend on all fillet faces that are formed at the blend chain intersections. Using this information, we should be able to decide which chain the new face belongs to without having to repeat the recognition algorithm. This forms a part of our ongoing research.

The working of the program can be seen with the simple

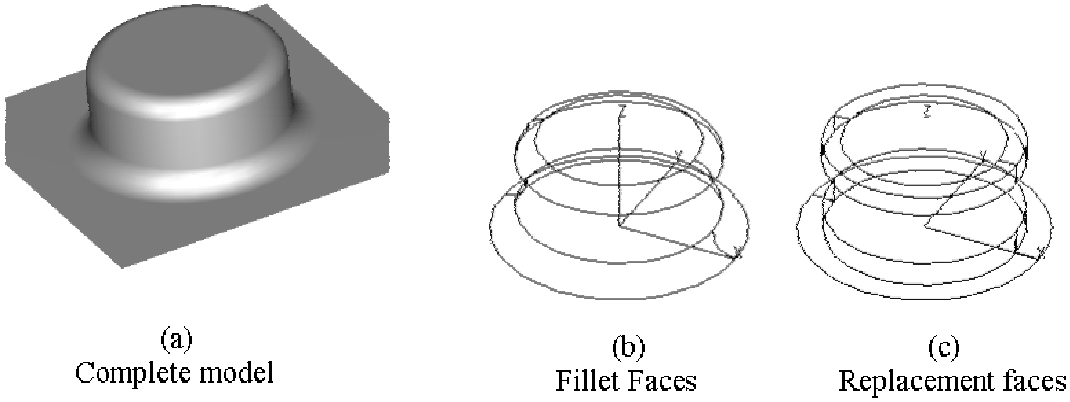

Figure 7 - Fillet simplification in a "boss"

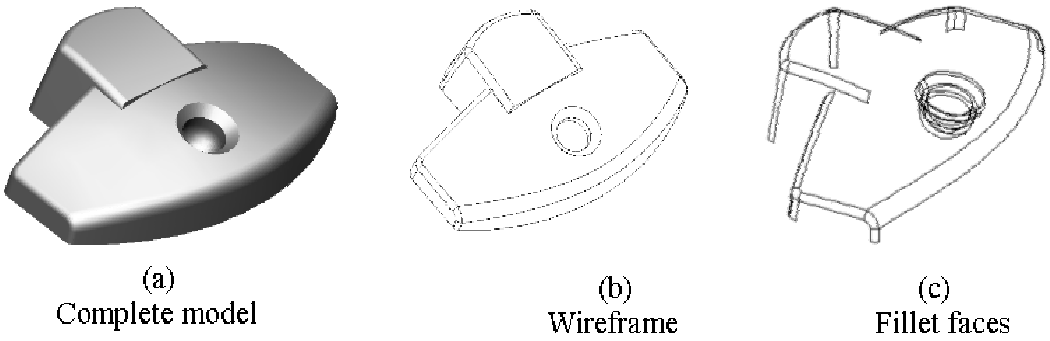

Figure 8 - Fillet recognition in the base of a joystick

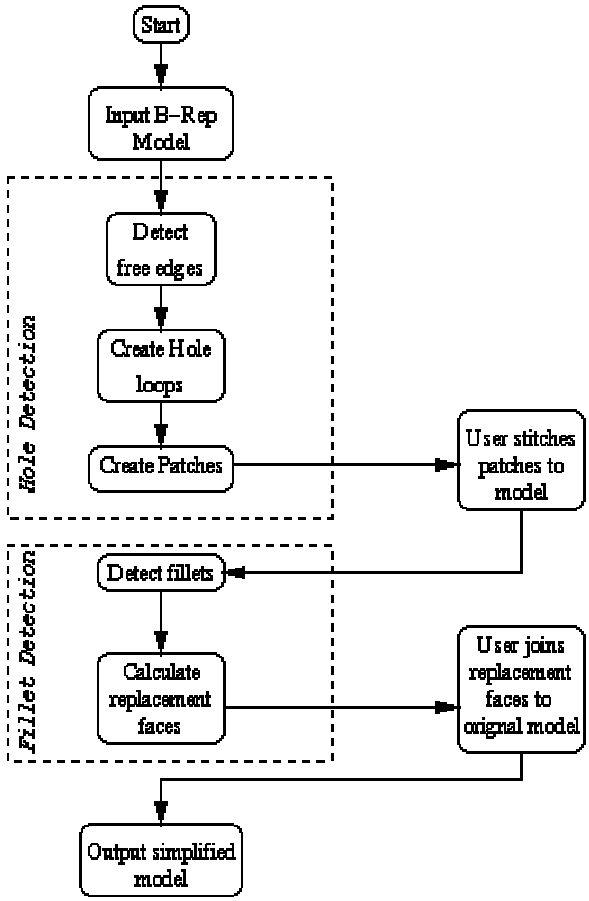

Figure 9 - Flow chart of overall feature simplification scheme

example of a boss. As can be seen from Figure 7, the program is able to detect the blend faces. It accurately calculates the intersection curves. The new faces generated to replace the fillet faces are shown in Figure 7(c). The blend detection algorithm was also verified using more complicated parts. The example shown in Figure 8, considers the surface model of the base of a joystick. All blend faces were accurately detected as can be seen in Figure 8(b). Accurate intersection edges were also calculated.

4 IMPLEMENTATION

The algorithms were coded for the SUN Solaris (UNIX) platform. The program is envisaged to act as a module in conjunction with a complete geometric modelling system. Since it is not meant to act as a stand-alone program, it does not have any graphical user interface. The algorithms are coded in C++ using ACIS API function libraries. The program records the results in ASCII text and .sat files. The same algorithms were also coded using C++ and Open-IDEAS to act as a module with the SDRC – IDEAS modelling system. In this case, the ASCII files are written but the modifications to the models are made in the current IDEAS session and need to be saved by the user. The overall working of the module can be summarised with the help of the flow chart shown in Figure 9.

Suppose that the designer wishes to detect and extract the 2 holes and 2 fillets in the model shown in Figure 10. The program first invokes the hole-detection module. The user is prompted for the threshold value of perimeter to be used for suppression of holes. Accordingly, the 4 holes are detected and suppressed. The output of the hole-detection phase is in the form of an ASCII file storing the definitions of the hole and patches that cover the holes, as shown in Figure 11.

The program then invokes the fillet-detection module. The fillet-detection module stores the fillet face and replacement faces in separate .sat files. It also deletes the fillet face from the original model and stores it along with the replacement faces. The replacement faces need to be manually stitched to the model by the user. The model at the end of the fillet-detection phase is shown in Figure 12.

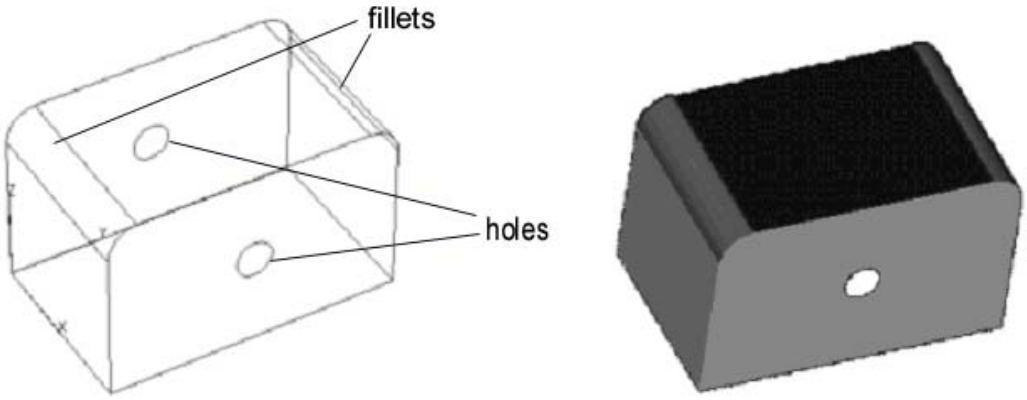

Figure 12 - Sample for feature simplification

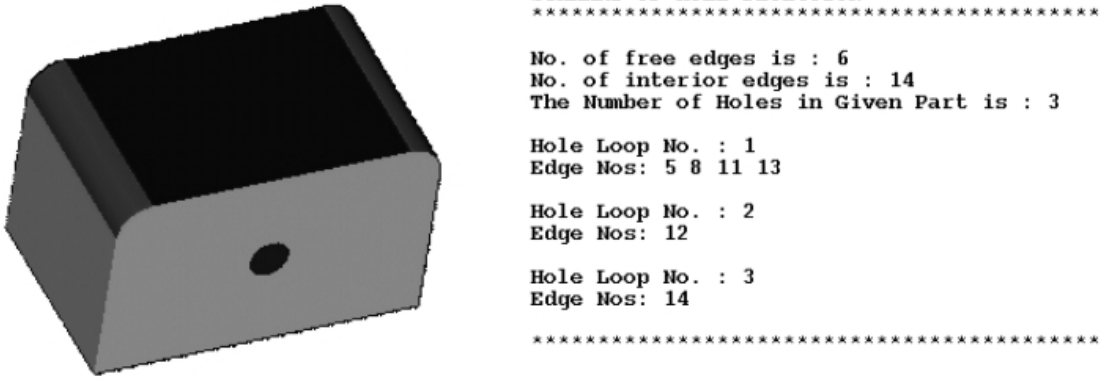

Figure 11 - Output of hole simplification phase

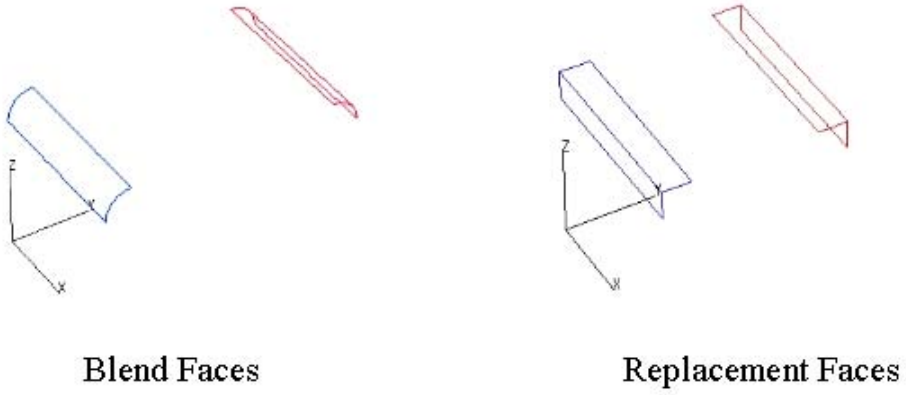

Figure 10 - Output of blend simplification phase

5 CONCLUSIONS AND FUTURE WORK

In this paper, we have proposed methods for directly accessing the CAD model data structure to recognise and simplify features in case of freeform surface models using predefined sets of rules. The results of an implementation of these methods are encouraging. The methods can successfully detect *hole* and *fillet* features. Methods for suppression or simplification of these features have been successful for certain classes of parts and have presented certain limitations.

The research presented in this paper was carried out with the aim of simplifying the geometry of a complex freeform surface model so as to allow efficient automatic mesh generation for Finite Element Analysis. However, most advantages of feature recognition in solid models also extend to freeform surface models. The ability to detect fillets will be helpful in recognition of more complicated form features, such as bosses or beads, since such features will generally be surrounded by fillets. The ability to detect features on freeform surfaces will allow comparison and of completely new families of parts. It will also provide measures to determine complexity of a given part. Features such as holes need to be further classified with regards to their function and shape so as to be more useful in part comparison. Such a classification, of course, will depend upon the family of parts under consideration. When aimed at facilitating finite element mesh generation, we also need to calculate the effect of suppression of features on the results of finite element analysis and formulate rules and conditions for simplification.

ACKNOWLEDGMENTS

We are grateful to the *Ford Motor Company* for their technical input during this project.

REFERENCES

[1] Shah, J. and Mäntylä, M., 1995, "Parametric and Feature-Based CAD/CAM", John Wiley and Sons, Inc.

[2] Kakazu, Y and Okino, N., 1984, "Pattern Recognition Approaches to GT Code Generation on CSG", *Proceedings of 16th CIRP International Seminar on Manufacturing Systems*, Tokyo.

[3] Vergeest, J. S. M. & Horvath, I., 2000, "Fitting freeform shape patterns to scanned 3D objects", *Proceedings of ASME2000 DETC*, Baltimore.

[4] Joshi, S. and Chang, T. C., 1988, "Graph-based heuristics for recognition of machined features from 3D solid model", *Computer-Aided Design*, Vol. 20, No. 2, pp. 58-86.

[5] Stanley, S. M., Henderson, M. R. and Anderson, D. C., "Using Syntactic Pattern Recognition to extract Feature Information from a solid geometric database, 1983, *Computers in Mechanical Engineering*, Vol. 2, No. 2, pp. 61-66.

[6] Vosniakos, G. C., and Davies, B. J., 1993, "A Shape Feature Recognition framework and its application to holes in prismatic parts, *International Journal of Advanced Manufacturing Technology*, Vol. 8, No. 5, pp. 345-351.

[7] Ferreira, J. C. E. and Hinduja, S., 1990, "Convex-Hull-based Feature Recognition Method for 2.5D components", *Computer-Aided Design*, Vol. 22, No. 1, 41-49.

[8] Kim, Y. S., 1994, "Volumetric Features Recognition using Convex Decomposition", *Advances in Feature Based Manufacturing*, Ed: Shah, J. J., Mäntylä, M., and Nau, D. S., Elsevier.

[9] Prabhakar, S. and Henderson, M. R., 1992, "Automatic Form-Feature Recognition using Neural-Network-based techniques on Boundary Representations of Solid Models", *Computer-Aided Design*, Vol. 24, No. 7, pp.381-393.

[10] Nezis, K. & Vosniakos, G., 1997, "Recognizing 21/2 D shape features using a neural network and heuristics", *Computer-Aided Design*, Vol. 29, No. 7, pp. 523-539.

[11] Wu, M. C., Chen, J. R. and Jen, S. R., 1994, "Global Shape Information Modelling and Classification of 2D workpieces", *International Journal of Computer Integrated Manufacturing*, Vol. 7, No. 5, pp. 261-275.

[12] Greska, W., Franke, V. & Geiger, M., 1996, "Classification problems in manufacturing of sheet metal parts", *Computers in Industry*, 33, pp. 17-30.

[13] Rossignac, J. R. and Requicha, A. A. G., 1984, "Constant Radius Blending in Solid Modeling", *Computers in Mechanical Engineering*.

[14] Venkatraman, S and Sohoni, M., 2001, "Blend Recognition Algorithm and Applications", *Sixth ACM Symposium on Solid Modeling and Applications, Conference Proceedings*, pp. 99-108

[15] Xu, X & Hinduja, S., 1998, "Recognition of rough machining features in 21/2 D components", *Computer-Aided Design*, Vol. 30, No. 7, pp. 503-516.

[16] Cohen, S., Elber, G. & Bar-Yehuda, R., 1997, "Matching of freeform curves", *Computer-Aided Design*, Vol. 29, No. 5, pp. 369-378.

[17] Lentz, D. H. and Sowerby, R, 1994, "Hole Extraction for Sheet Metal Components", *Computer-Aided Design*

[18] DoCarmo, M. P., 1976, "Differential Geometry of Curves and Surfaces", Prentice-Hall, Inc.

