Feature Recognition from Mesh Models

Deepali Tatkar [1]
Govind Kelkar [1]
Venkatesh Kamat [1]

Department of Computer Science and Technology, Goa University, India [1]

Abstract: In this paper we propose a methodology for manufacturing feature recognition from a segmented triangulated mesh model. Proposed methodology has two phases, segment preprocessing and feature recognition. Input to the algorithm is triangulated mesh model which is simplified by segmenting the mesh model into high level regions approximated by a simple primitive such as planes, sphere and cylinders. In segment preprocessing phase, we gather adjacency information related to each identified primitive. For each feature to be identified, feature rules are defined using geometric properties of approximated primitives. Finally, feature recognition phase checks if any of the connected set of primitives satisfy the feature rules and highlights the recognized feature. At present, feature recognition is restricted only to simple features composed of plane primitives such as pocket, step, slot and cylindrical primitive such as holes. Given a segmented mesh model as an input, the algorithm automatically recognizes different manufacturing features present in model. This extracted features information then can be used for downstream CAD/CAM application such as process planning, cost estimation and generating feature tree.

Keywords: CAD mesh model, reverse engineering, approximated primitives, segmented mesh, manufacturing features, feature recognition.

1 Introduction

Features play a key role in mechanical design and manufacturing. Feature recognition from low level geometric entities in the solid model has been of significant importance in Computer Integrated Manufacturing (CIM). Typically, design features and manufacturing features are different for the solid model. The type of features to be used in a particular application will depend upon the intention, i.e. whether the intent is to design or to manufacture. In this paper, we want to address the problem of automatic manufacturing feature recognition from triangulated mesh models. Manufacturing features play a crucial role in integration of design and down stream manufacturing applications such as process planning, cost estimation and tool path generation.

Most CAD models are represented parametrically using boundary representation (B-rep) data in either proprietary file formats or neutral formats such as STEP or IGES.

Copyright © 2009 Inderscience Enterprises Ltd.

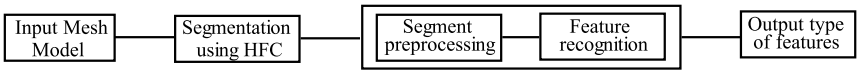


Figure 1: Framework for proposed algorithm (HFC: Hierarchical Face Clustering Algorithm (5))

Extracting simple manufacturing features from B-rep data models is straight forward task. But at times due to loss of primary geometric information during data exchange or feature interaction, extracting exact manufacturing features becomes hard. Significant research has been reported during the past two decades addressing these problems (4). Extracting high level semantic information becomes even more difficult, when solid model is represented with discrete geometry using mesh models. Today triangulated mesh model represented using STL file format is becoming popular due to advancement in 3D scanning and rapid prototyping (RP) technology. Triangulated mesh models can also be generated directly from B-rep models using automatic mesh generation software, for rapid prototyping, engineering analysis and visualization applications.

3D shape reconstruction from mesh geometry is an active area of research (1). It covers large application areas such as computer graphics, reverse engineering, biomedical engineering and visualization. Reconstructing complete CAD model from triangular mesh into high level B-rep model is a challenging problem (2). Mesh reconstruction, segmentation and surface fitting are the fundamental processes which try to associate high level geometry/topology information to raw 3D data referred to as point cloud. Although the major focus of research till date in 3D shape reconstruction has been to extract primary surface geometry from mesh model, recent trend is towards extracting higher level semantic information for shape understanding (3). Here not only primary surfaces but connected set of surfaces forming a particular composite feature with some functional requirements will be detected as a single entity. The goal of our research is to extract design related information of the mechanical CAD object at high level of abstraction.

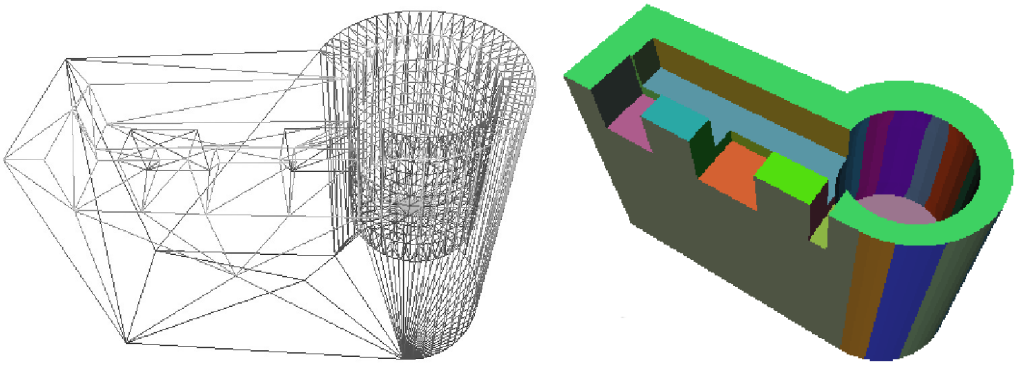
In this paper we are addressing the problem of manufacturing feature recognition from reverse engineering perspective and assume that input model is a segmented mesh model that gives primary surface geometry. In our study, segmentation of mesh model is done using a hierarchical mesh segmentation algorithm based on fitting primitives[(5)][(15)]. However mesh model can be segmented into high level regions by applying any of the segmentation algorithm present in the literature[(7)][(8)]. Hierarchical Mesh Segmentation algorithm has direct relevance to reverse engineering, since result of segmentation is set of simple primitives such as planes, cylinder or sphere that define most of the CAD models. Subsequently in feature recognition, we look for connected set of simple primitives which define meaningful features in the context of machining operations. The framework for the proposed methodology is shown in Figure 1 Rest of the paper is organized as follows. Section two gives brief description of previous work. Section three gives overview of proposed method and underlying segmentation. Section four describes implementation and results. Concluding section discusses findings and scope for future work.

2 Related Work

In the last decade, digital geometry processing received much attention with lots of work reported in core mesh processing, surface reconstruction and mesh segmentation (6). Many ideas in surface reconstruction originate from disciplines such as computational geometry and computer vision. Mesh segmentation itself is conceived as an optimization problem with set of constrains and evaluation criteria. Different mesh segmentation techniques reported in the literature are specific to application domain. Survey paper (7) describes in detail various segmentation techniques and their evaluation criteria. Much of the research work carried out on shape reconstruction deals primarily with organic objects such as human body, animals or art objects. Their objective is to reconstruct digital shape for rendering and visualization purpose. Engineering objects on the contrary are created with precision and have functional relevance during manufacturing. Reverse engineering field has reported different techniques for extracting primary surface information required for mesh model which segment the CAD model with its basic entities such as plane, cylinder, sphere etc. In (9) Rama Vorray et al. describes a semi-automatic approach for converting polygonal mesh model into standard high-level CAD representations using discrete differential geometry formulation. Recent paper by Manish Goyal et al.(10) describes a novel method that avoids the actual segmentation of the mesh and obtains direct parametrization of 3D mesh by sweeping prominent cross-section. In (11) T.Varady reviews different techniques to create complete and consistent topological structures over polygonal meshes and highlights more advanced issues such as design intent with reconstructed CAD models. K.Chang et al. (12) describes in detail, a process in which discrete point clouds are converted into feature-based parametric solid models using auto-surfacing technology. They also shed light on reverse engineering features available in the state-of-the-art commercial CAD software. Sunil and Pande (13) uses the sign of local curvature(s) property for segmenting and feature detection in sheet metal mesh models. They define rules for the following features, bend, dart, hole, louvre, dent and dimple that are specific to sheet metal components. Cheuk Yiu Ip and William C. Regli (14) describe automatic method for classifying mesh CAD models based on their manufacturing processes. Our work reported in this paper is on the similar lines, wherein we are trying to discover the function from the form.

3 Proposed methodology

Here we propose a methodology for manufacturing feature recognition from a triangulated mesh model where even primary surface information is not available. Reconstructing features from mesh data is proposed as a step by step process,reconstructing the high level geometry from low level triangles at each step. A rule based approach is used to define each manufacturing feature uniquely in terms of geometric primitives and its connectivity information. The approach has been widely used in CAD models represented using B-Rep (4). However, this is the first time a rule based approach is being tried on mesh data where very limited geometric/topological information is available for approximated surface primitive. The algorithm takes approximate primitive with adjacency information as input. At present feature rules are defined in terms of two simple geometric primitives mainly planar surface and cylindrical surface. The primary reason for choosing these primitives being, much of the manufacturing features are dominated by machining operations such as drilling and milling. This extracted features information then can be used for downstream



(a) Triangulated mesh object (b) An Object with mixed features

Figure 2: A CAD object mixed with different features

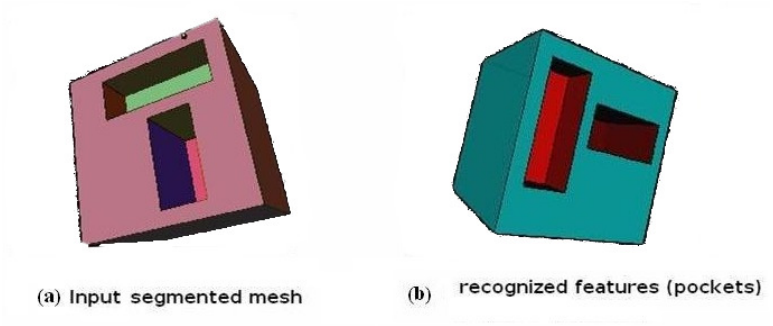


Figure 3: Feature recognition from mesh model

CAD/CAM application such as process planning, cost estimation and generating feature tree.

Proposed methodology is implemented in two phases, in the first stage we simplify input using segmentation and segment preprocessing by recognizing connected set of segments. In the second stage we identify specific manufacturing features.

Figure 2a exhibits a triangulated mesh model having multiple features. The colours show the segments identified by the earlier segmentation method.

- Segment preprocessing : Segmented model is accepted as input and processed further to detect different geometric/topology properties. Figure 3.(a) and Figure 2b.

- Feature recognition : Rules are formulated for each manufacturing features to be identified and finally feature rules are executed to extract any of defined feature is present in given model. Figure 3(b).

4 Implementation

4.1 Segment Preprocessing

After the segmentation, next step is to interactively take meaningfully segmented regions from the expert and processes these segments in order to extract the adjacency information. These segments are represented as clusters as explained in segmentation step. Each segment in the hierarchy has unique id associated with it along with total number of triangles present in that segment. At this stage, only the list of segments, approximated with primary surfaces primitives is available. But in order to get high level feature information, one need to find which segment is connected to which other segment. Let us assume a set of representative segments as S,

$$S = \{s_i; i = 1..N_c\}$$

where N_c denotes number of segments found and each s_i is represented as

$$s_i = \{v_{j=1..jn}, a_i, n_i, p_i\}$$

where v_j denotes set of vertices of triangles in that segment and v_j is subset of V which is a global set of vertices of given mesh model. a_i is areas of segment, n_i represent the normals direction of the segment, p_i denotes the primitives which takes values 0: planar, 1: circular, 2: spherical.Let the adjacency relation between S_i and S_j segment be defined as $r\,(S_i, S_j) \iff v = S_i \cap S_j$ and $v \geq 2$, then adjacency relation for all segments can be given as $\{r\,(S_i, S_j)$; i, j = $1..N_c$ and $i \neq j\}$.

The adjacency information of each segments is calculated based on number of shared vertices(at least two). For every segment, all its connected segment information is extracted and stored in data structure with primitive type and area information. Segments are sorted based on number of adjacent segments in descending order. We use the heuristics that the face with large no of adjacent segments is likely to have a feature. The segment with highest number of adjacent segments is selected as a parent segment for feature recognition step.

4.2 Data Structure

Every segment in the segmented model contains the information such as area, primitive, normal and it's adjacent segments(*adj*). Segments are represented by the id assigned to it by the preprocessing. These segments are represented by face(a dictionary data type with attributes area, primitive, normal, *adj*). Here the area and primitive are of float type. Ordinates of Normal are integers and *adj* is of type set. Figure. 4 shows the data structure used. The faces are sorted based on no of *adj* and stored.

4.3 Feature recognition

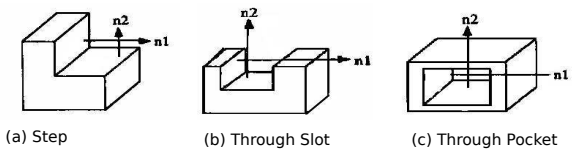
This section describes core feature recognition algorithm. The algorithm takes approximate primitive with adjacency information as input. A rule based approach is used to define each manufacturing feature uniquely in terms of geometric primitives and its

The segmented mesh object having 11 faces numbered: 1, 4, 6, 9, 13, 27, 29, 33, 35, 37, 39

face	=>	{ Area	Primitive	Normal vector	Set of adjacent face }
1	=>	{ 22500	0	[-1, 0, 0]	{4, 6, 9, 13} }
4	=>	{ 11250	0	[0, 1, 0]	{1, 6, 13, 39} }
6	=>	{ 11250	0	[0, 0, -1]	{1, 4, 9, 39} }
9	=>	{ 11250	0	[0, -1, 0]	{1, 6, 13, 39 } }
13	=>	{ 78750	0	[0, 0, -1]	{1, 4, 9, 27, 29, 33, 35, 39} }
27	=>	{ 90000	2	[0, 1, 0]	{13, 29, 35, 37 } }
29	=>	{ 90000	0	[1, 0, 0]	{13, 27, 33, 37} }
33	=>	{ 90000	0	[0, -1, 0]	{13, 29, 35, 37 } }
35	=>	{ 90000	0	[-1, 0, 0]	{13, 27, 33, 37} }
37	=>	{ 90000	0	[0, 0, 1]	{27, 29, 33, 35} }
39	=>	{ 22500	0	[1, 0, 0]	{4, 6, 9, 13} }

(a) Data structure

Figure 4: The Segmented mesh object representation using our data structure



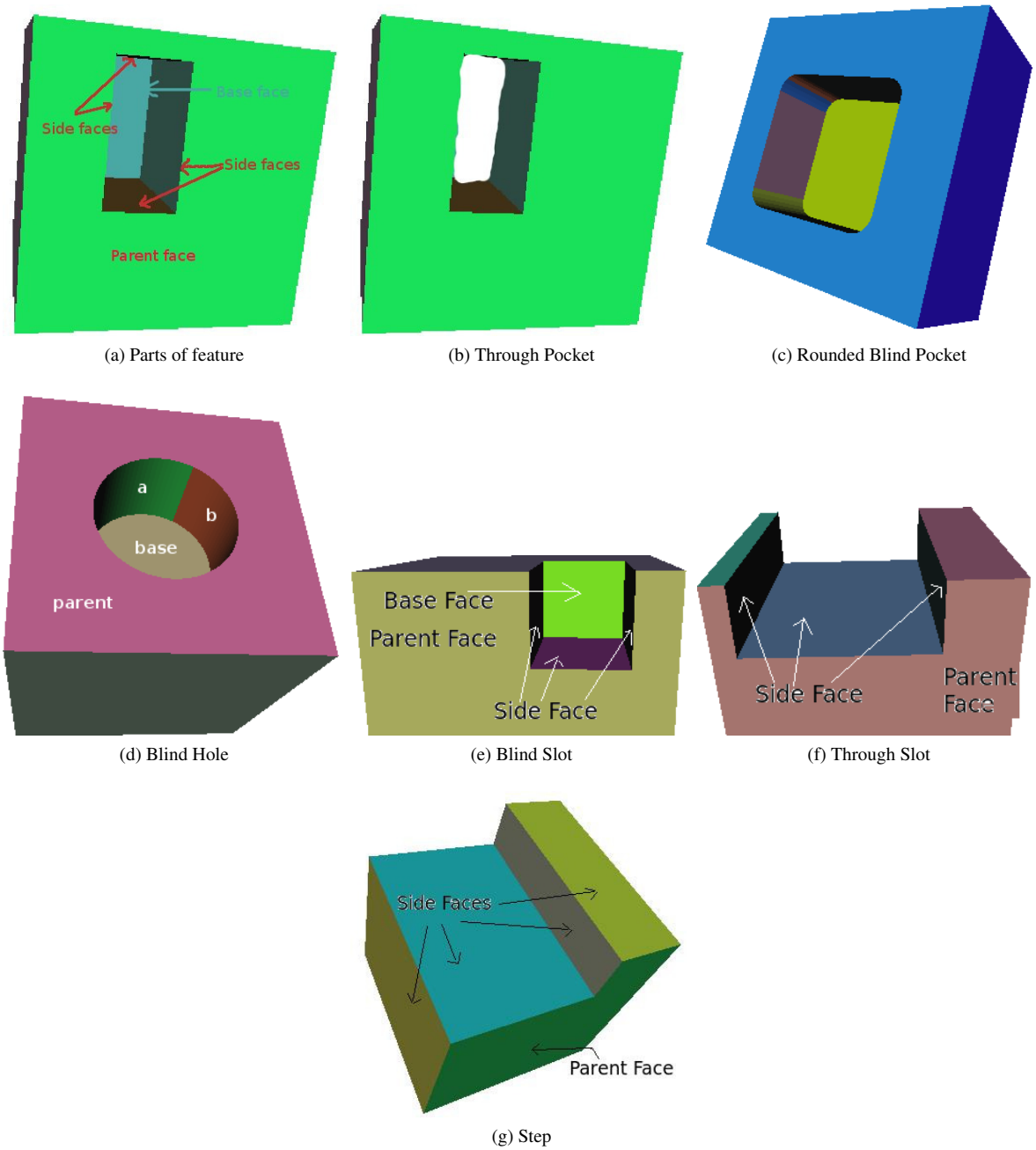
(a) Step (b) Through Slot (c) Through Pocket

Figure 5: Features and normal direction

connectivity information. The approach has been widely used in CAD models represented using B-Rep. However, this is the first time a rule based approach is being tried on mesh data where very limited geometric/topological information available for approximated surface primitive. At present feature rules are defined in terms of two simple geometric primitives mainly planar surface and cylindrical surface. The primary reason for choosing these primitives being, much of the manufacturing features are dominated by machining operations such as drilling and milling. We have explained below the rules for recognizing various features defined in terms of planar and cylindrical surfaces.

4.4 Basic terminology for recognition method

In Figure 6a, the terminology used in describing a feature is depicted. Feature is assumed to be on a parent face having side faces and a base face. The parent face has principal axis which is the axis of the surface normal of parent face. The normal of the face are depicted in Figure 5. In case of Figure 6b, the base face is the one where feature emerges out. The side faces are meeting exactly in right angle here whereas they may be separated by few circular faces (Figure 6c). In Figure 6c, the sum of area of circular face is less than sum of area of planar faces (embedded between circular faces), hence these figures are termed as rounded features whereas the feature in Figure 6a and Figure 6b are termed as square



(a) Parts of feature (b) Through Pocket (c) Rounded Blind Pocket

(d) Blind Hole (e) Blind Slot (f) Through Slot

(g) Step

Figure 6: Different types of features and feature parts

features. Figure 6d represents a Hole feature with its parts identified as base face, parent face and side face (circular faces). The figure has a base face making it a Blind hole feature

and the absence of base face gives a through hole feature. The normals of Base face and Parent face in Figure 6d and Figure 6a are facing in same direction which tells that these features are blind where as normals of Base face and Parent Face in Figure 6b are in opposite direction making the feature as through type. Slot features shown in Figure 6e and 6f have parent face, base face and three side faces. The missing side face is considered as open side. Step feature is seen having Parent face, Base face and Side faces.

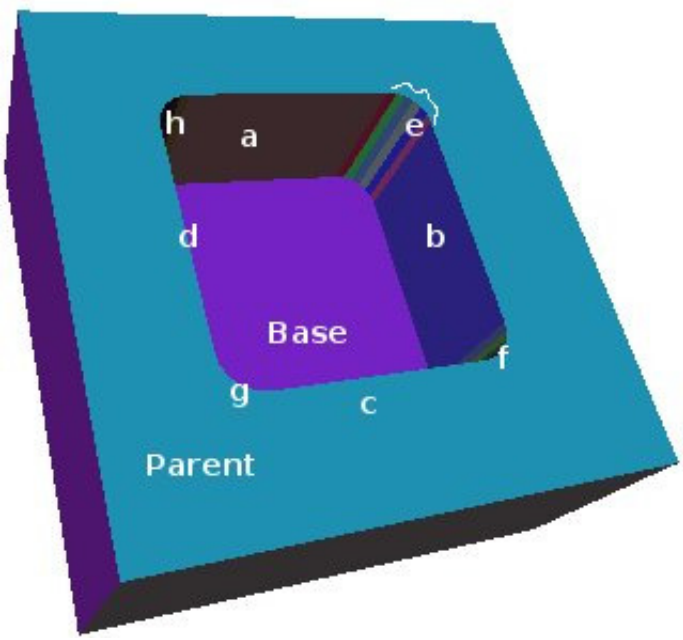


Figure 7: rounded pocket feature and its parts

A rounded feature as shown in Figure. 7. can have side faces of type circular or planar. The *sides* named as a, b, c and d are plane type and their primitive type will be 0. The *sides* e, f, g, h are circular type with primitive set to greater than 0. From the figure we see that e, f, g, h are made up of many small segments of type circular. The number of these small segments depends upon when the segmentation process was stopped.

5 Recognition procedures in details

5.1 GetHoleAdjacents

```
/* getHoleAdjacents
 * input: seedadj, adjacents, parent
 * output: elem, ne, roundArea, flatArea
 */
```

```
initialise elem as list, sides as dict and ne as set
A = seedadj    /* non-planar segment(primitive != 0) */
B = next adjacent non-planar segment
add A and B to elem   with any planar adjacents in between A and B
initialise done to false
while not done
       initialise ne to adjacents of B minus adjacents of A
       remove elem from ne
       intersect ne with adjacents
       if ne has an element
              A = B
              B = first element of ne
              append B to elem
              done = false
       else
              done = true
initialise ne to intersection of adjacents of all elements of
    elem /* base of feature */
initialise roundArea to sum of areas of non-planar elements of
    elem(primitive != 0)
initialise flatArea to sum of areas of planar elements of elem
return
```

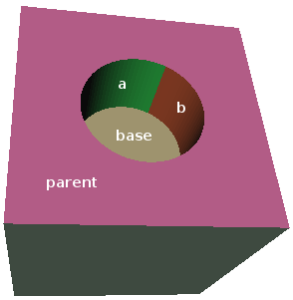


Figure 8: Adjacents of hole feature

In this procedure we try to get all the adjacent segments of cicular type that form a hole feature. This procedure have a set of all circular type of adjacents to parent face. We start with topmost circular segment(a). Search for next circular segment(b) that is adjacent to a. Put these a and b (Figure 8)in the order they are found in a list called *elem*. The Figure 8 depicts that consecutive segments make up for the faces of feature. Thus at last we get all the adjacent segments making the feature. This same method is used to identify the adjacents in the procedure GetAdjacents.
Subtracting adjacents of a and elements of *elem* from b will give us next segment which is part of hole. This new found element to be appended to *elem*. Repeat this until we get a new element which is part of the feature. The intersection of set of adjacents of all elements of *elem* will be stored in a list called *ne*. This will include parent as well as base face and as we are interested in getting base face, the parent segment will be removed from *ne*. *elem* is

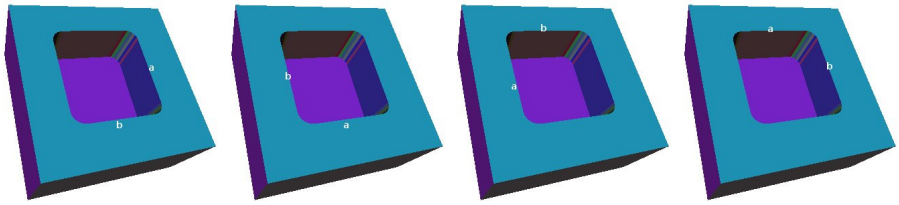
making *sides* of the feature and *ne* making base side of the feature. Compute the *roundarea* and *flatarea* of elements of *elem* based on their primitive. Area of all elements with primitive 0 will be added to *flatarea* and rest will be added to *roundarea*. This procedure will return *elem*, *ne*, *roundarea*, *flatarea* parameters.

5.2 GetHoleFeatures

```
/* getHoles
 * input: parent, adjs
 * output: ThHole, blHole
 * non-planar adjacent: primitive != 0
 * planar adjacent: primitive = 0
 */
initialise circ to non-planar adjacents from adjs
initialise initCirc to circ
initialise i to 0
initialise done to true
initialise holes to 0
while done
    call getHoleAdjacents(circ[i], adjs, parent)
    if roundArea > flatArea and len(elem) > 1
        if normal of ne is same as normal of parent
            add hole to thHole set
        else
            add hole to blHole set
        remove elem from parent and ne adjacents list
        remove elem from circ
        i = 0
        holes = holes + 1
    if initCirc == circ
        i = i + 1
        if i > length(circ)
            done = false
    else
        initCirc = circ
return blHoles, thHoles, holes
```

The adjacents with primitive greater than 0 are stored in a list *circ*. The procedure GetHoleAdjacents is called repeatedly until all hole features are extracted from this parent segment. The process continues as follows: GetHoleAdjacents is called to fetch the adjacents forming a hole feature. If *elem* has more than 1 segment and *roundarea* is more than *flatarea* only then the feature is a hole. Further if the normal of parent and base segment is same then we get a blind hole otherwise it is through hole. Now remove the elements of *elem* from adjacents of parent and *ne* and also remove them from *circ*.If hole was not found than consider next element of *circ* as seed. Finally all the holes extracted are returned by this function.

5.3 GetPlanFeatureAdjacents



(i) Iter-1 (ii) Iter-2 (iii) Iter-3 (iv) Iter-4

Figure 9: Iterations of getting adjacents in case of rounded pocket

```
/* getPlanarFeatureAdjacents
 * input: seedadj, adjacents, parent
 * output: elem, ne, sids, openSide, roundArea, flatArea
 */
initialise elem as list, sides as dict and ne as set
a = seedadj    /* planar segment(primitive = 0) */
b = next adjacent planar segment
add a and b to elem   with any adjacents in between a and b
add area of a, b to sides
initialise done to false
while not done
       initialise ne to adjacents of b minus adjacents of a
       remove elem from ne
       intersect ne with adjacents
       if ne has an element
              a = b
              b = first element of ne
              append b to elem
              if b is planar then add area of b to sides
              done = false
       else
              done = true
if length(elem) > length(sides) then /* case of rounded feature */
       collect planar segments from sides into sids
else
       initialise sids to elem
initialise openSide to the elements of elem having more than 4
    adjacents
remove parent from ne
initialise ne to intersection of adjacents of all elements of
    elem /* base of feature */
initialise roundArea to sum of areas of non-planar elements of
    elem(primitive != 0)
initialise flatArea to sum of areas of planar elements of elem
return
```

These features will be of at least 4 *sides* plus one base side. Slot feature will have an open side(in the figure 6e and 6f, it is clear that there are 3 visible *sides* and fourth side is non-existent. This non-existent side is considered as open side of the feature).

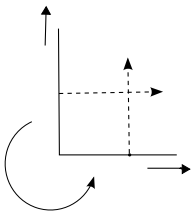


Figure 10: Direction shift of faces

The algorithm starts with a seed segment passed to it. Search for the first *adj* that is adjacent to the seed. Put both these elements in *elem* in their order of appearance. Consider first element of *elem* as *a* and second as *b*. Maintain a dictionary(*sides*) of planar segments with key(segment id) and value(segment area). Now get the difference of the adjacent(*ne*) of *a* from *b*. Take the difference of *elem* from *ne* which will give us a list of next segments in the order of *a*, *b*. IF this list is non-empty than consider the first element of the list found and append it to *elem*. If the new found element is planar then append it to *sides* also. Repeat this procedure until *ne* is non empty. Now, if *elem* has more elements then *sides*, this means we have found a rounded feature so we have to get a separate list of planar *sides* of the feature. For this sort the *sides* on descending order of area and get top 4 elements in *sids* variable. Now refer to order of elements in *elem* and place the elements of *sids* to get the planar faces of the feature in the order as *elem* is having all rounded as well as planar faces in the order of appearance. If there is any element of *elem* with size of *adj* more than 4 then it is considered as *open* side. Now get the intersection of *adj* of all elements of *elem* and remove parent from this result to get the base side of the feature. Compute the *roundarea* and *flatarea* of elements of *elem* based on their primitive. Area of all elements with primitive 0 will be added to *flatarea* and rest will be added to *roundarea*. This procedure will return back *elem*, *open*, *sides*, *ne*, *flatarea*, *roundarea*

6 Feature Recognition Rules for planar features

6.1 Direction Shift Pattern

```
/* getDirection
 * input: n1, n2, paxis
 */
if paxis = 1 then take y and z of n1 and n2 as x and y
if paxis = 2 then take x and z of n1 and n2 as x and y
if paxis = 3 then take x and y of n1 and n2 as x and y
q1 = n1(x, y)
q2 = n2(x, y)
if (q1 along x and q2 is along -y) then /* shift in same quadrant
    */
return 1
else   /* shift is not in same quadrant */
return 0
in case of error return -1
```

In Figure **??**, "face 1" and "face 2" are 2 faces of a feature. They are aligned along X-axis and Y-axis respectively. Normal of "face 1" is along Y-axis and that of "face 2" is along X-axis. Thus the direction of normals (n1 to n2) is shifting from Y-axis to X-axis. we consider this shift of direction as 1(concave) and otherwise 0(convex). This is used to get a direction shift pattern formed by the side faces of a feature(non-circular feature) which will be discussed after having a look at the features considered for retrieval.

```
/* getDirectionShiftPattern
 * input: segments, paxis, normals of segments
 * output: dirs, sumd, f
 */
initialize i to 0
while i < length(segments) - 1
       dirs.append(getDirection(segments[i], segments[i + 1],
           paxis))
       increment i
dirs.append(getDirection(segments[i], segments[0], paxis))
initialize sumd to 0
add all dirs to sumd
store str(dirs) to f
return dirs, sumd, f
```

It is clear from the above discussion that step, slot and pocket features are composed of side faces(Plan face) in a certain layout formation. There is concave or convex shift of normal direction from one face to other face. The faces are arranged as per the neighbourhoodness among themselves and then the direction shift is accumulated starting from first face till the last face. This makes a pattern like '1111', '101', '0110' which decides what feature is formed by these faces.

The normal of each face has 3 ordinates and we use only 2 of them. The ordinate representing the principal axis of the parent face is dropped from the ordinates of the face normals and the remaining are used for finding the direction shift. If principal axis is along Y axis then the y ordinate is dropped of both normals and the remaining x and z ordinate is considered as x and y ordinate. if principal axis is along X axis, then y and z ordinates makes x and y ordinate for direction shift.

6.2 GetFeatures

```
/* getFeatures
 * output: blHole, thHole, thPocket, blPocket, thSlot, blSlot,
    Step
 * r : holds dict of count of adjacents of segment
 */
initialise t to keys of r
while pars to next key from t
    while parent to next pars
        initialise adj to set of adjacent segments of parent
        call getHoles(parent, adj)
        initialise done to true
        initialise i to 0
```

```
        initialise paxis to axis of normal to parent
        while done
            initAdj = adj
            if roundArea < flatArea and len(nb) > 3
                initialise found to false
                initialise normals to normals of nb
                [ptype, sums, pattern] = call
                    getDirectionShiftPattern(nb, paxis, normals)
                if sums is 4 and pattern = '1111' or sums = 0 and
                    pattern = '0000'
                    if length(side) < 1
                        initialise found to true
                        if normal of ne = normal of parent
                            feature is blind Pocket
                        else
                            feature is through Pocket
                    elseif sums > 1:
                        if pattern contains '101'
                            feature is step
                            initialise found to true
                        else
                            if pattern contains '0110'
                                initialise found to true
                                if length(side) > 0
                                    feature is blind slot
                                else
                                    feature is through slot
                    if found is true
                        i = 0
                        remove elem from base, parent, adj, side
                            adjacents
                if (initAdj = adj)
                    done = false
                else
                    if length(adj) < 5
                        done = false
                    else
                        initAdj = adj
return features
```

All the segments are sorted based on the number of adjacent segments they have with a feeling that larger the number of adjacent segments there is certainly going to be a feature. We start with topmost segment to search for the feature. This will be considered as parent segment(parent face) on which the features will be available. The adjacents of this segment(parent segment) are sorted(descending order) based on area with the heuristics that inner adjacent will be having area less than outer adjacent. Call GetHoleFeatures procedure to extract all the holes from parent.

Copy adjacents in *adj* and *initAdj*. Call GetPlanFeatureAdjacents to extract a planar feature in *adj* of parent. If size of *sides* is more than 3 and *flatarea* is large than *roundarea* then we have a feature. Now to decide what type of feature it is we compute direction shift pattern. if the sum of digits of direction pattern is 4 and pattern is '1111' and *open* is empty

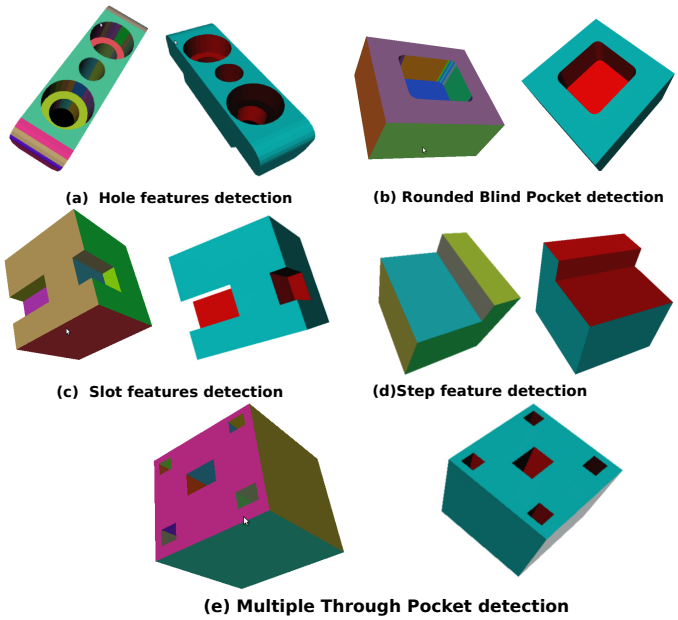


Figure 11: Input segmented mesh models and extracted features.

then we get a pocket. If normal of parent and base is same then we have a blind pocket otherwise it is a through pocket. If sum is more than 1 and pattern is '101' then we have a step feature.

If sum is more than 1 and pattern is '0110' then we have a slot feature. Further if normal of parent is same as normal of base then it is a blind slot other a through slot. If a feature is found then the elements of *elem* and *open* are removed from *adj* of parent and base. Also they are removed from *adj*. This procedure is repeated until size of *adj* of parent is greater then 5 or *initAdj* is not same as *adj*.

Repeat the GetFeature procedure with other segments with size of *adj* greater than 4.

7 Discussion and results

Computing segment adjacency and its attributes is implemented in C++ and feature rule definition and feature recognition algorithm is implemented in python. All visualizations are done using Coin3D library [16]. CAD models in triangulated mesh formats such as off, ply, stl, wrl and obj are accepted as input for segmentation step. Figure 11 (a),(b),(c),(d) and (e) shows, some of the result of feature recognition with input segmented mesh models and its corresponding extracted features highlighted in red color.

8 Conclusion and future scope

In this paper we present a method for recognition of manufacturing features from CAD models represented in triangulated mesh formats. Given a segmented mesh

model(Mechanical CAD mesh models with regular surfaces and moreover in the models where neighborhood exist between given any two surfaces of the model) as an input, our algorithm successfully detect the manufacturing features such as pocket (blind/through), hole (blind/through),slot (blind/through) and step. Some complex features such as rounded pocket and slot are too detected. It extracts the interacting features to some extent as we tried with a case of hole interacting with another hole. Our algorithm is working successfully for simple mechanical CAD model. This extracted features information then can be used for downstream CAD/CAM application such as process planning, cost estimation etc. Our aim is to generalize the algorithm to work on complex model with more features and extract dimension information of the features for process planning and NC tool path generation.

This method can be extended to freeform mesh models also provided neighbourhood can be specified between every pair of the surface on such model. The idea of feature extraction can be extended further for extracting more complex feature or interacting feature recognition from mesh data. It can be also extended for feature patterns recognition, feature tree generation or design intent recognition.It can be used for mesh compression by representing model at high level of semantics with feature level representation.

References

[1] M. Attene, S. Biasotti, M. Mortara, G. PatanÃ¨, M. Spagnuolo, and B. Falcidieno,2006 'Computational methods for understanding 3D shapes',*Computers and Graphics*, Vol. 30(3), pp.323-333.

[2] R. Beniere, G. Subsol, G. GesquiÃ¨re, F. Le Breton, and W. Puech,2013 'A comprehensive process of reverse engineering from 3D meshes to CAD models',*Computer-Aided Design*, vol. 45(11), pp.1382-1393.

[3] M. Attene, F. Robbiano, M. Spagnuolo, and B. Falcidieno,2009 'Characterization of 3D shape parts for semantic annotation',*Computer-Aided Design*, vol. 41(10),pp.756-763.

[4] B. Babic, N. Nesic, and Z. Miljkovic,2008 'A review of automated feature recognition with rule-based pattern recognition',*Computers in Industry*, vol. 59(4), pp. 321-337.

[5] Attene, M.; Falcidieno, B.; Spagnuolo,2006 'Hierarchical mesh segmentation based on fitting primitives',*The Visual Computer*, 22(3), 181-193.

[6] Hugues Hoppe,1994 'Surface reconstruction from unorganized points',*PhD Thesis*, Department of Computer Science and Engineering, University of Washington.

[7] A. Shamir,2008 'A survey on Mesh Segmentation Techniques', *Computer Graphics Forum*, vol. 27, no. 6, pp. 1539-1556.

[8] A. Agathos, I. Pratikakis, S. Perantonis, N. Sapidis, and P. Azariadis,2007 '3D Mesh Segmentation Methodologies for CAD applications',*Computer-Aided Design*, vol. 4, no. 6, pp. 827-841.

[9] R. A. Vorray, G. O. Driscoll, and A. Steed,2008 'Reverse Engineering Polygonal Meshes Using Discrete Differential Geometry',*Computer-Aided Design and Applications*, vol. 5, pp. 86-98.

[10] M. Goyal et al.,2012 'Towards locally and globally shape-aware reverse 3D modelling',*Computer-Aided Design*, vol. 44, no. 6, pp. 537-553.

[11] T. Varady,2008 'Automatic Procedures to Create CAD Models from Measured Data', *Computer-Aided Design and Applications*, vol. 5, no. 5, pp. 577-588.

[12] Kuang-Hua Chang and Chienchih Chen,2011 '3D Shape Engineering and Design Parametrization'*Computer-Aided Design and Applications*, vol. 8, no. 5, pp. 681-692.

[13] V. B. Sunil and S. S. Pande,2008 'Automatic recognition of features from freeform surface CAD models',*Transition*, vol. 40, pp. 502-517.

[14] C. Y. Ip and W. C. Regli,2006 ' A 3D object classifier for discriminating manufacturing processes' *Computers and Graphics*, vol. 30, no. 6, pp. 903-916.

[15] Garland, M.; Willmott, A.; Heckbert, P. S,2001 'Hierarchical face clustering on polygonal surfaces',*Proc. Symposium on Interactive 3D Graphics* 49-58.

[16] https://bitbucket.org/Coin3D/coin/wiki/Home