Reverse Engineering Polygonal Meshes
Using Discrete Differential Geometry

Rama A. Vorray[1], Gerard O'Driscoll[2] and Anthony Steed[3]

[1]University College London and Vero UK Ltd., r.vorray@cs.ucl.ac.uk, rvorray@vero.co.uk
[2]Vero UK Ltd., godriscoll@vero.co.uk
[3]University College London, a.steed@cs.ucl.ac.uk

ABSTRACT

Reverse Engineering computer representations of physical objects is still a substantial research topic either it be in the manufacturing industry, virtual reality or computer vision. This paper explains a semi-automatic approach to the conversion of polygonal meshes into standard high-level CAD representations. User intervention is sought only for choosing appropriate tolerance values and fine-tuning the model. The uniqueness of the current approach is its usage of Discrete Differential Geometry. We take regular triangle meshes and compute Discrete Differential Curvature measurements over them. We then sub-divide them into smaller 2D geometric entities and 3D triangular faceted patches that highlight sharp edges and curved surfaces, respectively. The techniques we have implemented automate a substantial part of the structure and topology discovery and we aim for further automation in future. All features described in this paper have been implemented as part of the CAD package VISI®, owned by Vero UK Ltd.

1. INTRODUCTION

Reverse engineering has seen a rapid growth in its research, development and applications since its first introduction in 1970's [3]. Reverse engineering has various application domains and roles within those domains. In computer graphics, reverse engineering might be targeted at efficiently creating a visual representation from real data; whereas in advanced manufacturing, reverse engineering might be targeted at creating a high-level CAD model from which to derive a machining process [3]. In this paper we focus on the reverse engineering of CAD models comprised of planar and curved surfaces. The goal is to create a CAD model that can subsequently be edited efficiently, or used to generate different discretizations. The input data will be triangle models in STL files that might originate from laser scan data or that might have been exported from other CAD/CAM pipelines.

There is extensive literature available on reverse engineering objects represented in 3D point cloud format, e.g., [3], [8], [9], [13], [16]. In [8] Fudos describes a feature-based constraint-based exploitation of information for small objects like jewelry in constructing CAD models but the source is only 3D point cloud data. In [16] Thompson et al. describe a domain-specific modeling and domain-specific constraint based approach similar to [8] and aims at real mechanical parts but is again based on point cloud data. In [13] Owen, Sloan & Thompson describe a feature-based approach but using an interactive tool REFAB where user should specify approximate locations of manufacturing features within the point cloud for them to be fitted. In [9] Germain et al. describe various constraints that can be used to simplify the reverse engineering process but it again uses a feature-based approach as described in [13] and [16].

There are reverse engineering tools available in the market that address part of the reverse engineering process. For example, CopyCAD from DelCam is a tool to convert point cloud datasets to polygonal representations. Reverse Engineering 2 (RE2) from Dasault is a tool to reverse engineer polygonal meshes [3]. MeshToSolid, a plug-in to Rhino3D®, is another reverse engineering tool which reduces number of triangles in mesh data by grouping triangle facets that describe a geometric feature and fits a single trimmed NURBS surface to that group of facets [12]. Although this technique gives far fewer primitives to work with, it doesn't take 2D features of the object into consideration, and thus it creates complex surfaces in regions that could be described more efficiently with different primitives.

The technique described in this paper is aimed at the reverse engineering of triangular meshes. It is based on mesh analysis, fragmentation of mesh into similar geometric regions and fitting geometric primitives to those regions. The input polygonal mesh comes from STL files. The first stage is to build topological information on the mesh using the winged-edge data structure [2]. This topological information enables us to understand local geometry of objects. After understanding the local geometry, objects are segmented into 2D geometric primitives and 3D surface patches representing boundaries and shapes, respectively. This segmentation is achieved using curvature measurements from Discrete Differential Geometry (DDG) [5]. The overall process is semi-automated and requires users nothing more than to vary tolerance values that constrain shape fitting. The user can optionally fine-tune the model, such as specifying best allocation of input geometry to output surfaces, when there is no clear geometric discriminator to make the choice.

Section 2 gives an overview of building topological information over triangle meshes using winged-edge data structure along with some of its advantages relevant to this application area. Section 3 explains various DDG curvature measurements over the meshes and how they provide necessary information. Section 4 gives a description of segmenting objects into 2D geometric primitives like arcs, line-segments and circles and several three-dimensional triangular faceted surface patches. Section 5 lays out future work in this project which is towards surface fitting, model validation or *model beautification* as mentioned in [3] and further automation.

2. TOPOLOGICAL INFORMATION OVER THE MESH

A triangular representation of an object only provides local geometric surface information. In this representation there is none of the topological information that is necessary for computing discrete curvature measurements. These measures, which are discussed in Section 3, require adjacency information at each and every vertex and edge of the object. We use the Winged Edge Data Structure (WEDS) to build and store such information. WEDS is an edge-based hierarchical data structure where for each edge - its two vertices, two faces on either side of it and two connecting edges from each vertex at both ends, are held. This provides a connected network of vertices, edges and faces through which the whole mesh can be traversed. Detailed study of this data structure can be found in [2], [4], [15]. In [11] Kettner describes other variants of winged-edge data structure and other hierarchical data structures with similar properties.

Geometric information obtained from STL files is stored using Vertex-Face data structure [1], [4], [15]. This data structure holds a list of triangles with indices pointing into a unique list of vertices. Details about generation of Vertex-Face data structure can be found in [4] which also describes a clear-cut technique of converting vertex-face data structure into winged-edge data structure. Converting STL files into Winged-Edge representation is $O(n^2)$ complex and hence it is necessary to pre-process STL files and store basic topological information in an intermediate file which does not levy huge computational loads while converting them into winged-edge representations in real time.

3. DISCRETE DIFFERENTIAL GEOMETRY

DDG is a recently explored area of mathematics where analogies of continuous forms of curvature measurements are applied on discrete structures [5]. Triangle meshes can be considered to be discretized models of original objects that were continuous. Hence discretized equations of continuous differential geometry can give a good numerical analysis at discrete levels of objects. Specifically, DDG provides metrics that let us analyze geometry of objects at the faces, edges and vertices of meshes. The two metrics that we use are Gaussian Curvature and Integrated Mean Curvature [5]. Gaussian curvature is the integral of *angle defect* at all vertices contained in a non-empty convex body. Angle defect at a vertex is the sum of interior angles of all triangles meeting at that vertex and is also called *excess angle*. Integrated mean curvature is the integral of dihedral angles at all edges in the non-empty convex body weighted by their lengths. Dihedral angle is the angle made by two triangle normals adjoining an edge [5].

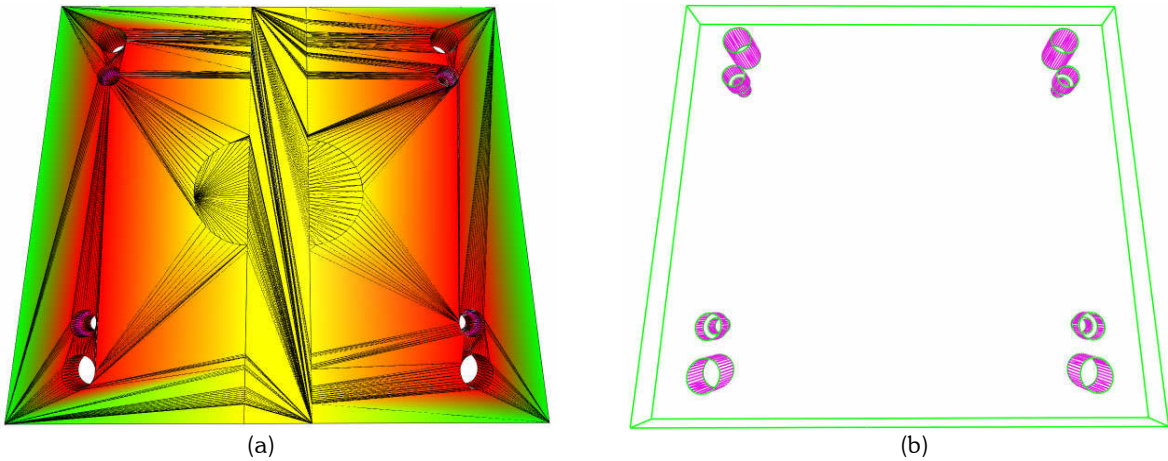(a)   (b)

Fig. 1(a): A simple mechanical object with Gaussian Curvature measured at the vertices of the object. Colors are interpolated across each triangle. Fig. 1(b): The same Mechanical Object with Integrated Mean Curvature measured and edges filtered on the basis of their dihedral angle.

Fig. 1(a) and fig. 1(b) demonstrate a simple mechanical object with the two curvature measurements taken on it. Fig. 1(a) shows the object colored by the Gaussian curvature. Vertices where object possesses a greater than 360° defect angle are shown in red, vertices with near 360° defect angle are shown in yellow (planar regions) and vertices with less than 360° defect angle are shown in green and magenta. It can be observed that corners of the outline cuboid are shown in green color, indicating the fact that they have an angle defect of 270°. Fig. 1(b) demonstrates the same object with Integrated Mean Curvature calculated on it. It can be observed that significant edges of the body (sharp edges and edges on curved surfaces) are highlighted. Edges shown in green have a dihedral angle of near 90° and edges shown in magenta have a dihedral angle less than 90°. The image shown here is after filtering edges based on their dihedral angle. Filtering in this case is the selection of edges whose dihedral angle falls within a particular range. For example, selecting edges whose dihedral angle is between 89.9° & 90.1° and 0.1° & 89.8° (inclusive) from objects shown in fig. 1(a)., and fig. 2(a)., will result in objects shows in fig. 1(b)., and 2(c)., respectively. Section 3.1 gives further details on the two discrete curvature measurements. Further explanation on edge filtering can be found in section 4.1.

3.1 Gauss Curvature and Integrated Mean Curvature
An in-depth explanation about these measurements on a polyhedron is given in [5]. Let P be a polyhedron with vertex set V and edge set E and B any small region in linear space R^3, then Integrated Gaussian and Mean curvatures can be defined as:

$$\varnothing_P^K(B) = \sum_{v \in V \cap B} K_v \quad \text{Where} \quad K_v = 2\pi - \sum_j \alpha_j \tag{3.1}$$

$$\varnothing_P^H(B) = \sum_{e \in E} l(e \cap B)\theta_e \tag{3.2}$$

Eqn. (3.1) is for Gaussian Curvature and Eqn. (3.2) is for Integrated Mean Curvature. $\varnothing$ indicates the function while $\textbf{\textit{K}}$ and $\textbf{\textit{H}}$ given as superscripts to $\varnothing$ indicate type of measures in R^3. $\textbf{\textit{Kv}}$ is the excess angle sum at vertex $\textbf{\textit{v}}$ made by all incident triangle angles at $\textbf{\textit{v}}$. $\textbf{\textit{l(.)}}$ denotes the length of an edge and θ_e is the signed dihedral angle at $\textbf{\textit{e}}$ made by incident triangle normals [5]. Once local topological information is available using a data structure such as WEDS, computing these measures is extremely quick.

4. FEATURE EXTRACTION

With a WEDS built and DDG measurements taken on the object, we can then extract required features. The aim is to convert the triangle mesh into standard CAD representation consisting of geometric primitives such as line-segments, arcs, circles and NURBS patches. In addition, extraction of cylinders can also be observed, utilizing the basic elements already extracted. There are two steps involved in this process. They are: extraction of 2D geometric entities like line-segments, arcs and circles; extraction of triangular faceted patches

4.1 Classification of Mesh

Dihedral angle at edges of the mesh can be used to classify the mesh into groups where all edges in a group have similar properties. Fig. 2(a)., fig. 2(b)., and fig. 2(c)., depict the classification of edges.
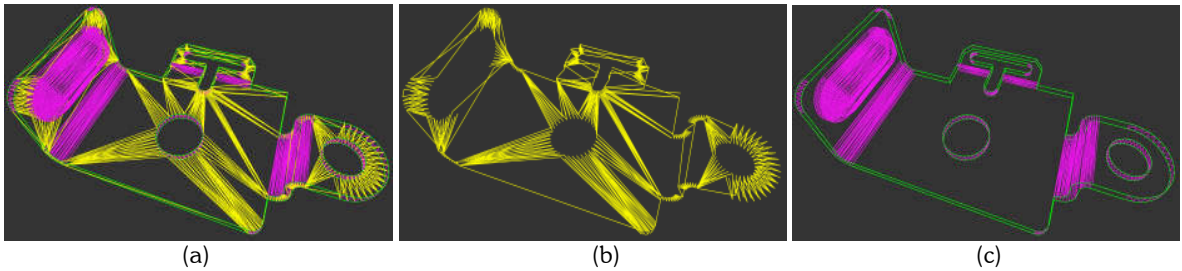

(a)   (b)   (c)

Fig. 2(a): An object with integrated mean curvature measured and coloured according to the dihedral angle of edges. Fig. 2(b): Object with edges only in planar regions. Dihedral angle of these edges is 0° since triangle normals adjacent to each of those edges are parallel to each other. Fig. 2(c): Edges of the body at sharp boundaries and curved surfaces. Edges shown in green have a dihedral angle of 90° (indicating sharp edges) and that shown in magenta have a dihedral angle of 8.11° (indicating a uniform curved shape). In this mesh no edges have a curvature other than 0°, 8.11° and 90°.

(a)   (b)

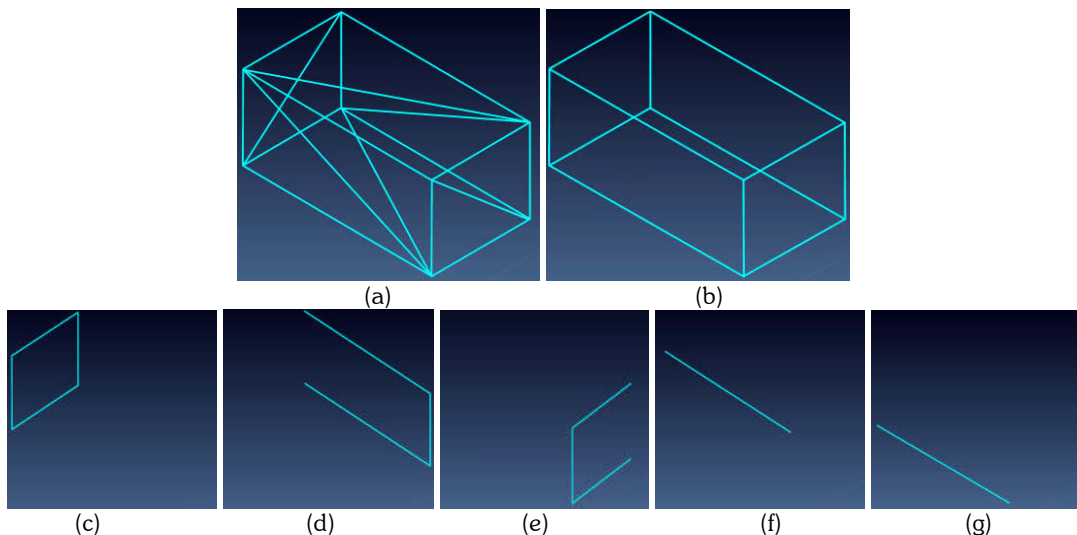(c)   (d)   (e)   (f)   (g)

Fig. 3(a): A cuboid represented as a triangle mesh. Fig. 3(b): Sharp edges of cuboid grouped together using mean curvature of edges. Fig. 3(c) through Fig. 3(g): Network of edges shown in (b) broken down into smaller entities of polylines which are being considered as *paths* and these entities will be further partitioned into *segments*.

The sharp edges of the object, as shown in green in Fig. 2(c), will have a dihedral angle of 90°, as normals of triangles incident on each of those edges are perpendicular to each other. These edges play a significant role in extracting 2D

features of the object. Fig. 2(c), also shows edges in magenta which indicate curved parts of the object. In general all edges of an object can be classified as planar with dihedral angle of near 0°, curved with dihedral angle between 0° and 90°, boundary edges with dihedral angle of near 90° or acute edges with dihedral angle between 90° and 180°. Groups of edges are gathered in this procedure using Integrated Mean Curvature measure. It is assumed and taken care that all triangle normals point outwards from the object.

4.2 Extraction of 2D Geometric Features

Edges are grouped according to their curvature measures. For example, edges having a dihedral angle of near 90° are considered as one group indicating all sharp boundaries of the object. Similarly, edges with other angle ranges, for example 8.11° as shown in Fig. 2(c), are considered as other groups for 3D feature extraction. The group of edges having dihedral angle of near 90° (also called as *sharp edges*) is again classified into different *paths* and then into small *segments*. These *segments* are polylines that represent either line-segments or arcs or circles. A *path* is a trail of edges in which all vertices are distinct and a *segment* in this case can be considered as a trail of edges where direction change between all consecutive edges is similar. A special case is a *circle* which is also a *path* [1]. The pseudo code procedure MESH_TO_PATHS describes the procedure of extracting *paths* out of *connected sharp edges*. Each path and each segment thereafter is considered as a separate group of edges. Fig. 3, above depicts the MESH_TO_PATHS feature performed on a cuboid.

PROCEDURE MESH_TO_PATHS
BEGIN

    For each edge $e(i)$ **and** (e->dihedral_angle > 89° and < 91°) **and EDGE_UNVISITED**
    {For each edge $e(i)$ in the object whose dihedral angle is > 89° and < 91° and the edge is unvisited before}

        FIND_CONNECTED_EDGES_TO e(i) AND ASSIGN_THEM_GROUPID
        {Build a network of connected edges starting from $e(i)$ having similar dihedral angle. Store the Group ID}

        If (no connected edges found to $e(i)$) **then**
            $e(i)$ IS_A_SIMPLE_LINE_SEGMENT
            Continue WITH_ANOTHER_EDGE
    Else
        PROCEED_BELOW

        COLLECT_EDGES_WITH_CURRENT_GROUPID_INTO_A_LIST

        For each edge $e(j)$ IN_THE_LIST
            If ($e(j)$ is unvisited in the current loop) **then**
                PROCEED_BELOW
        Else
                Continue WITH_NEW_EDGE

        CONSIDER_NEW_GROUPID
        {Consider new group ID to be assigned for the new path being found. Simply increment current group ID by one. This value gets incremented as new paths are being found.}

        ASSIGN_GROUPID_TO_CURRENT_EDGE
        {Assign the new group ID to the current edge}

        FIND_UNGROUPED_EDGE_ADJACENT_TO_CURRENT_EDGE
        {Find an unvisited edge from the list that is immediately adjacent to the current edge in either direction. Edge should be unvisited in the current loop}

        {For this new edge}

If (either of the new edge's vertices are not shared by more than one edge with the same group ID) **then**

      TAG_EDGE_WITH_GROUPID

      Continue AGAIN_FROM_FIRST_EDGE

      {Tag new edge with the updated group ID and continue the loop from first edge}

Else {there is a *node* present at one of the vertices}

      Continue FROM_START_OF_THE_PROCEDURE

      {Do not tag edge with the new group ID and continue from start of the procedure considering the current edge and finding a new network of edges and new paths within it.}

      End {For Each Edge *e*(j)}

    End {For Each Edge *e*(i)}

End {End Procedure}

At the end of this procedure all *edge paths* of the object have been extracted and grouped together. Each group is assigned a numerical ID as can be understood from the pseudo code above. These groups indicate individual paths of edges which have to be classified again into segments to indicate an arc or a circle or a simple line-segment. The pseudo code procedure PATHS_TO_SEGMENTS below describes the procedure of classifying edge paths into segments. This algorithm works on the principle of direction change of edges during traversal of that path. Fig. 4 below depicts an original connected edge path and it being partitioned into further smaller entities representing line-segments and arcs.
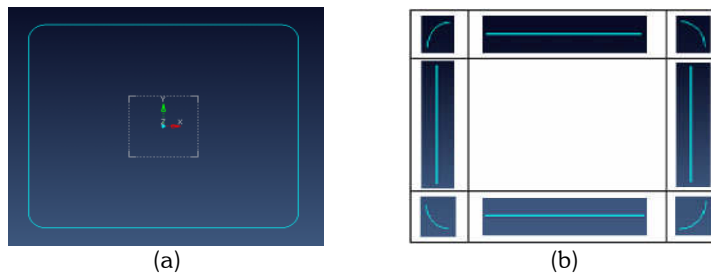

(a)   (b)

Fig. 4(a): A *path* extracted from an object. Fig. 4(b): The same path partitioned into smaller segments representing arcs and line-segments. Images in each cell on the circumference of the table show partitioned segments.

PROCEDURE PATHS_TO_SEGMENTS

BEGIN

    For each path p(i) WITH_CORRESPONDING_GROUPID

      COLLECT_EDGES_OF_PATH p(i) INTO_A_LIST

      {Collect edges of path p(i) with the current group ID into a list}

      If (only one edge is in the list) **then**

        RETURN_IT_AS_A_SIMPLE_LINE-SEGMENT

        Continue WITH_NEXT_PATH

      Else PROCEED_BELOW

      SORT_EDGES_IN_LIST_BASED_ON_ADJACENCY

      {Sort edges in the current list based on their adjacency. vertices of all edges in the path should be geometrically adjacent.}

      CONSIDER_NEW_GROUPID

      {Consider a new group ID to be assigned for the segments found. Simply increment current group ID by one. This value gets incremented as groups of segments are found.}

TAG_FIRST_EDGE_IN_LIST_WITH_UPDATED_GROUPID

For each edge *e*(j) IN_THE_LIST
FIND_DIRECTION_CHANGE_FOR_FIRST_TWO_EDGES_AND_GET_SEED
{Find direction change between first two *edges* and consider it as a *seed* of tolerance. Consider *edges* as normals in any one direction and find Euler angle between them.}

If (*seed* is > 45o and <225o) **then** {sharp corner between first two edges}
FIRST_EDGE = GROUPID
SECOND_EDGE = GROUPID++
Else If (the length of any of the two *edges* is > mean of lengths of all *edges* in the object) **then**
FIRST_EDGE->GROUPID = GROUPID
SECOND_EDGE->GROUPID = GROUPID++
Else
FIRST_EDGE->GROUPID=SECOND_EDGE->GROUPID=GROUPID

For each edge *e*(k) STARTING_FROM_SECOND_EDGE_OF_THE_LIST AND WITH_LOOP_VARIABLE_AS_IN_ABOVE_LOOP
{Loop from second *edge* of the list with same loop variable as in loop immediately above}

FIND_DIRECTION_CHANGE_FOR_TWO_CONSECUTIVE_EDGES_AND_GET_NEW_SEED

If (new *seed* == old *seed*) **then**
SECOND_EDGE->GROUPID=GROUPID
{Tag the second of the two *edges* with same group ID and continue in this loop}
Else
GROUPID++
SECOND_EDGE->GROUPID=GROUPID
{Tag the second of the two *edges* with updated group ID}
Continue FROM_PARENT_LOOP
{Increment loop variable and continue from the parent loop}
End {For each edge *e*(k)}
End {For each edge *e*(j)}
End {For each path p(i)}
End {End of Procedure}

At the end of this procedure all edge paths have been classified into several small segments. These groups of edges are all connected line-segments which can be a simple line-segment or an arc or a circle.

4.2.1 Fitting Line-Segments, Arcs and Circles to Edge Segments (Polylines)
After edge segments have been extracted out of the paths they should be represent either a line segment, a circle or an arc. Respective geometric entities have to be fitted to these segments. The procedure followed for this is based on *shape recognition by decomposing NURBS curves* which can be found in [14]. The procedure as described in this document is that, a primitive shape can be recognized by decomposing a NURBS curve into Bezier pieces, sampling those pieces at uniform intervals and using these sampled points to determine the shape of the curve. In the current situation instead of sampling curves, unique list of vertices given by edge-segments are considered as sampled points and the shape of the polyline (connected edge segments) is determined. First, centres of curvatures and unit normals at these sampled points are computed using DDG as described in [5] and then two criteria are followed for detecting line-segments and circles separately. For line-segments the procedure is that, to check if $dist(Pi, line(P0, Pn)) < \varepsilon$ where ε is a user-defined tolerance and that projection of Pi onto $(P0, Pn)$ lies between $P0$ and Pn. As described by Piegl &

Tiller in [14], **n=2(p+1)** and **p** is the degree of the original curve but as no curve is being considered here and only points are being dealt with, **n** is considered to be the number of unique edge vertices on the polyline. For a circle, the procedure is, to compute centers of curvature (centre of osculating circle) and unit normals to the plane of the curve at each of the sampled points and to check that the centers and unit normals at all these points do not deviate more than the set tolerance [14]. Again, as there are no curves considered here, unit normals are computed to the plane of the polyline and sampled points are the unique list of edge vertices on the polyline. If the polyline in this case is not closed, it is considered as an arc after checks are done for degenerate points.

The tolerances considered are values that depend on the source and nature of the data; they may need user specification. For example, the tolerance for the detection of a circle can depend on the resolution at which the original mesh was generated, and whether it has been reduced in detail at any point. This makes the current application semi-automatic, but since each mesh is usually generated in a uniform way only a few values need be set.

The pseudo-code procedure SEGMENTS_TO_GEOMETRIC_PRIMITIVES explains the method of classifying a polyline to an arc, line-segment or a circle. The specific method of fitting (simplifying) these primitives to a polyline is described above.

PROCEDURE SEGMENTS_TO_GEOMETRIC_PRIMITIVES
BEGIN

    For each segment s(i)
    {For each segment of edges obtained from the Procedure PATHS_TO_SEGMENTS}
        COLLECT_ALL_UNIQUE_VERTICES_OF_POLYLINE_EDGE_SEGMENTS_INTO_A_LIST

        PASS_LIST_ONTO_SIMPLIFICATION_PROCEDURE

        If (can be simplified to a single point) **then**
            FIT_POINT
        Else If (can be simplified to a line-segment) **then**
            FIT_A_LINE_SEGMENT
            {Procedure followed to check for a line-segment is described above}
        Else If (segment can be simplified as a circle or an arc) **then**
            If (polyline is closed) **then**
                FIT_A_CIRCLE
                {Procedure followed to check for a circle is described above}
            Else
                FIT_AN_ARC
                {Computer radius of arc using its two end points and validate the arc by checking that all points of the polyline lie on the arc}
    End {For each segment}
End {End of Procedure}

Fig. 5 shows extracted 2D features of the object shown in Fig. 2. The object is entirely represented using circles, arcs and line-segments. It can be observed that the curved surface part of the object is not shown in this figure. This is because it is a smooth deformation of a planar surface, and thus it has no simple characterizing 2D boundary. Refer to Fig. 8(a-1) through 8(d-3) in Section 9 for more examples.

4.3 Extraction of Triangular Faceted Patches
Extraction of 2D features brings out many of the significant features of an object. The remaining parts that have to be extracted are curved surface that define the shape of the object at places where there are no sharp boundaries. This surface extraction can be accomplished in two steps: grouping of triangles based on the angle defect of their vertices and dihedral angle of their edges; and considering these groups of triangular faceted patches and fitting surfaces.
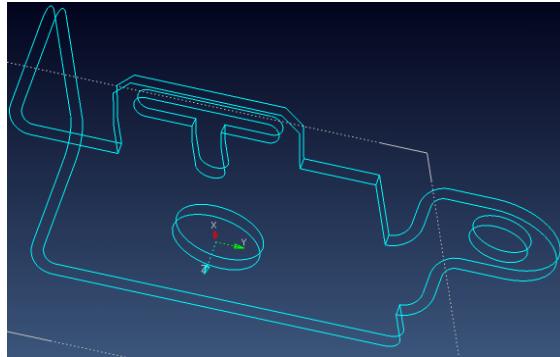
Fig. 5: Reverse Engineered form of object shown in Fig. 2, rendered inside VISI CAD package. The object is almost entirely represented using 2D geometric primitives, circles, arcs and line-segments. It can be observed that curved surface part of the object show in magenta in Fig. 2, has not been captured.

The following section describes the procedure of extracting groups of triangles indicating surface patches and Section 4.3.2 describes a method of smoothing boundaries between patches with an analysis on possible surface fitting techniques appropriate for the situation.

4.3.1 Grouping of Triangles

A group of triangles whose vertices have equivalent excess angle and possess similar features, is taken indicate a patch on the object's surface. Vertices of triangles in this patch might provide geometric information, such as *control points*, through which surfaces interpolating those points shall be built. In reality however there is no technique which can fit a surface that can pass through all the vertices. We explore a couple of techniques which allow us to fit surfaces for extracted patches keeping error within user specified tolerance. The pseudo code MESH_TO_FACETED_PATCHES below explains the procedure of grouping triangles based on their vertices' excess angle (angle defect) and dihedral angle. Triangular facets are grouped using a numerical group ID similar to the grouping procedure followed for edges.

PROCEDURE MESH_TO_FACETED_PATCHES
BEGIN

        For each facet f(i) UNGROUPED
        {For each unvisited facet in the mesh}
                If (all three vertices of f(i) have same excess angle && f(i) is not on a flat surface) **then**
                        PROCEED_BELOW
                Else
                        CONTINUE_WITH_NEW_FACET

                INCREMENT_FACET_GROUPID
                TAG_CURRENT_FACET_WITH_UPDATED_GROUPID
                PLACE_CURRENT_FACET_IN_AN_EMPTY_LIST
                {Initialize a List and place the current facet in that empty list. This list extends progressively.}

                For each facet f(j) FROM_THE_LIST
                        FIND_ALL_FACETS_AROUND f(j)
                        {A maximum of three facets can only be found around any facet.}

                        For each facet f(k) OF_THE_THREE_NEW_FACES
                        {For each facet of the three new faces found surrounding the facet from list.}

If (*angle defect* of vertex from facet f(k) across the *edge* adjoining facet f(j) is
equivalent to that of either of the three vertices' *angle defect* of facet f(j) && none
of the edges' dihedral angle of f(k) is 0° && f(k) is not visited before) **then**
{If facet has same gauss curvature and is not on a flat surface}
    FACET->GROUPID = UPDATED_GROUPID
    APPEND_FACET f(k) TO_THE_LIST
    {Append the facet to the list of faces considered before for the
    procedure to extend progressively to find a patch.}
Else If (f(k) is not visited before && none of edges' dihedral angle of f(k) is 0°)
{Facet is on the boundary of a patch}
    FACET->GROUPID = UPDATED_GROUPID
    {Tag the facet to the current group (patch) but do not append it to the
    list of faces for progressive patch build-up}
      End {For each facet f(k)}
    End {For each facet f(j)}
  End {For each facet f(i)}
End {End of Procedure}

At the end of this procedure, groups of triangular faceted patches are indentified to which spline surfaces can be fitted. Initially, these surfaces may possess an irregular shape and appear with zigzag boundaries which are not ideal as input data for any of the existing surface fitting techniques. Hence, we smooth boundaries to the extent possible by following a strategy whereby it is decided which patch a particular boundary facet should fall into so as to avoid saw tooth pattern at the patch boundaries as seen in Fig. 6. Section 4.3.2 describes this procedure of smoothing boundaries and describes possible methods of fitting surfaces automatically with minimal user input.

(a)

(b)   (c)   (d)
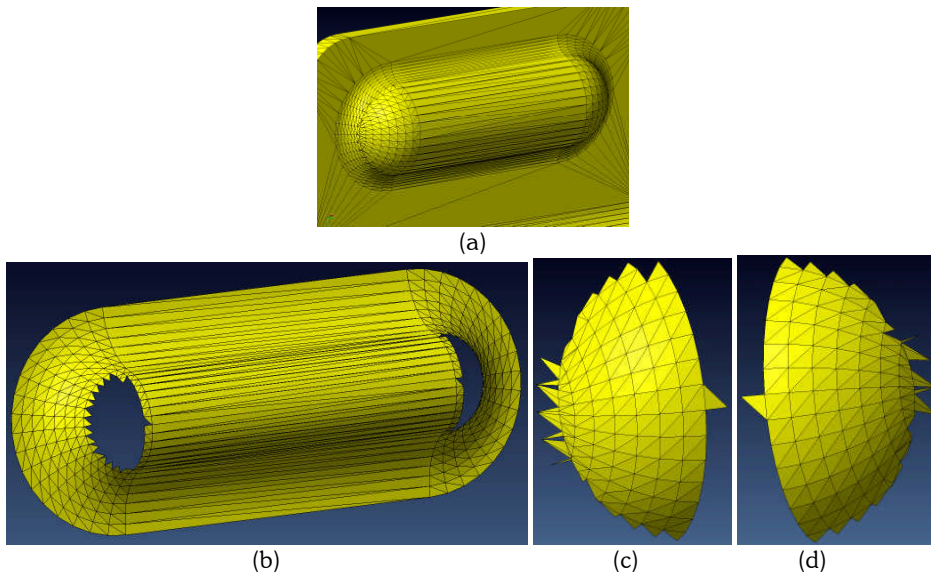
Fig. 6(a): The original object with a curved surface which has to be converted into a real spline based surface. Fig. 6(b) – Fig. 6(d): Extracted parts of the surface. A sawtooth effect at the boundaries of patches can be observed here, due to effectively random allocation of triangles when they could lie in either of two adjacent parts.

4.3.2 Smoothing Patch Boundaries and Surface Fitting Techniques
As can be seen in Fig. 6, surface patches often will be created with irregular boundaries. This is for two reasons. Firstly, the underlying geometry might be ambiguous, in that a particular triangle might be similar to two adjacent parts. This case can be fixed by a heuristic where a border between two parts is simplified, when this would not alter the fitting properties of the respective surfaces. The strategy followed is that if any triangle in any patch has more than two edges

of it shared by other patches, that triangle will be added into one of those patches so that not more than one edge should be shared by another patch, thereby shortening distance between two consecutive vertices. This strategy has been applied on shapes shown in Fig. 6(b) – Fig. 6(d) and the results can be seen in Fig. 7. The second reason for the irregular boundary is that some meshes are poorly structured, either because of resolution of scan, or because they were sampled naively from high-level descriptions and thus they contain very small or very thin triangles that are difficult to classify. This can be seen in Fig. 7(b) & Fig. 7(d), where a sharp corner still exists. The corner is the result of very small adjacent triangles which mean that this can be automatically allocated to the other, more appropriate, part of the mesh.

| (a) | (b) | (c) | (d) |
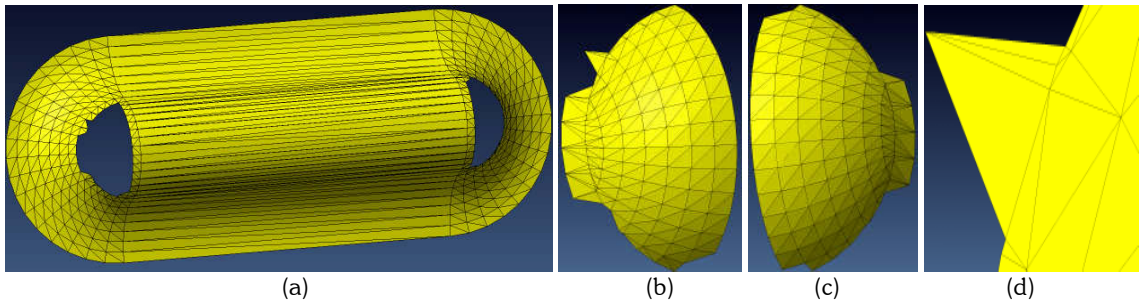
Fig. 7(a) - Fig., 7(c): Boundary smoothed forms of patches shown in Fig. 6(b) - Fig. 6(d) respectively. A sharp corner can be observed in Fig. 7(b). This is a group of three triangles as shown in Fig. 7(d).

After parts of surfaces have been extracted as patches, the task is to fit NURBS or any other B-Spline based surfaces to these patches. Existing surface fitting techniques [6], [7], [10], [17] expect data to be organized into quadrilateral regions to fit surfaces. In order for such irregular shapes to be organized into quadrilateral regions their boundaries should be smoothed so that definite quadrilateral regions can be defined over the patch. A couple of techniques have been explored for surface fitting. First the technique described in [10] is being studied and analysed for implementation. Other techniques which can be found in [6], [7] will be utilized in future depending upon the results obtained through the implementation of technique described in [10]. In [10] Hongwei, Guojin & Chenshi describe an iterative b-spline surface fitting technique which takes a given point set as control point set and gradually adjusts control points using an iterative formula. This technique is only applicable to surfaces that are two-manifold. In [6] Eck & Hoppe describe a technique of fitting surfaces using sub-division surfaces. This technique is built up on the technique described in [7]. Eck & Hope in [6] first classify a mesh of n-manifold into several Voronoi tiles, partitions tiles into triangular regions, smoothens their edges and creates quadrilateral regions by joining pair-wise triangular regions. All triangles in these quadrilateral regions are re-meshed into quadrilaterals by joining triangles pair-wise and a tensor product b-spline patch is fitted for each individual region, resulting in a network of G^1 continuous patches. This technique is not yet guaranteed for open surface patches with irregular boundaries [6]. Although our aim is towards a totally automated reverse engineering application, for some surfaces, an assisted manual process is being considered. The assistance will be automated measures of fit to vertices and fit to the curvature implied by the DDG on the input mesh.

5. FUTURE WORK

This paper has described a new approach for reverse engineering of triangular meshes. To date the techniques yield an useful representation of the original model using geometric primitives such as lines, arcs and circles and classification of curved parts of the object into a network of patches for which surfaces have to be fitted. This is already of immediate use in a CAD/CAM process, as features detected need not be reworked. Some manual work is currently required to fit patches. Future work, in addition to surface fitting, is aimed at automating the procedure to a further extent, automatically constructing analytical shapes such as cylinders from the 2D geometric features extracted and validating the transformed model by checking that the whole model lies within a set tolerance from the original model. All these improvements will hopefully deliver an end-to-end pipeline for reverse engineering polygonal meshes and producing high-level CAD primitives.

6. ACKNOWLEDGEMENTS

7. COPYRIGHTS

8. REFERENCES

[1]   Anderson, I.: A First Course in Discrete Mathematics, Springer, London, 2001.
[2]   Baumgart, B. G.: Winged Edge Polyhedron Representation for Computer Vision, National Computer Conference, Stanford, California, May 1975.
[3]   Bradley, C.; Currie, B.: Advances in the Field of Reverse Engineering, Computer-Aided Design & Applications, 2(5), 2005, 697-706.
[4]   Chan, K.-C.; Tan, S.-T.: Hierarchical Structure to Winged-Edge Structure: A Conversion Algorithm, The Visual Computer, 4, 1988, 133-141.
[5]   Desbrun, M.; Polthier, K.; Schroeder, P.; Stern A.: Discrete Differential Geometry: An Applied Introduction, Course Notes, SIGGRAPH 2006, Boston, MA, 2006.
[6]   Eck, M.; Hoppe, H.: Automatic Reconstruction of B-Spline Surfaces of Arbitrary Topological Type, Proc. SIGGRAPH 1996, 325-334.
[7]   Hoppe, H.; DeRose, T.; Duchamp, T.: Surface Reconstruction from Unorganized Points, Proc. SIGGRAPH 1992, 71-78.
[8]   Fudos, I.: CAD/CAM Methods for Reverse Engineering: A Case Study of Re-engineering Jewelry, Computer-Aided Design & Applications, 3(6), 2006, 683-700.
[9]   Germain, H. J.-de st.; Stark, S.-R.; Thompson, W.-B.; Henderson, T.-C.: Constraint Optimization and Feature-based Model Construction for Reverse Engineering, Proc. ARPA Image Understanding Workshop, February 1996, http://citeseer.ist.psu.edu/dest96constraint.html. (Accessed on 25 February, 2008)
[10]   Hongwei, L.; Guojin, W.; Chenshi, D.: Constructing Iterative Non-Uniform B-Spline Curve and Surface to Fit Data Points, Science in China Series F - Information Sciences, 47(3), 2004, 315-331.
[11]   Kettner, L.: Designing a Data Structure for Polyhedral Surfaces, Proc. 14th Annual ACM Symposium on Computational Geometry, June 1998, 146-154.
[12]   MeshToSolid, http://www2.rhino3d.com/resources/display.asp?language=&listing=522, McNeel. (Accessed on 25 February, 2008)
[13]   Owen, J.-C.; Sloan P.-J.; Thompson, W.-B.: Interactive Feature-based Reverse Engineering of Mechanical Parts, Proc. ARPA Image Understanding Workshop, November 1994, 1115-1124.
[14]   Piegl, L. A.; Tiller, W.: Computing offsets of NURBS curves and surfaces, Computer-Aided Design, 31(2), 1999, 147-156.
[15]   Slater, M.; Steed, A.; Chrysanthou, Y.: Computer Graphics and Virtual Environments: From Realism to Real-Time, Addison-Wesley, London, 2001.
[16]   Thompson, W.-B.; Germain, H. J.-de st.; Henderson, T.-C.; Owen, J.-C.: Constructing High-Precision Geometric Models from Sensed Position Data, Proc. ARPA Image Understanding Workshop, February 1996, 987-994.
[17]   Venkat, K.; Marc, L.: Fitting Smooth Surfaces to Dense Polygon Meshes, Proc. SIGGRAPH 1996, 313-324.

9. EXAMPLE FIGURES

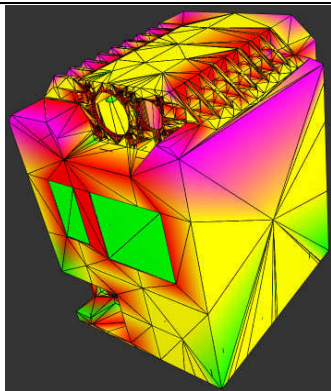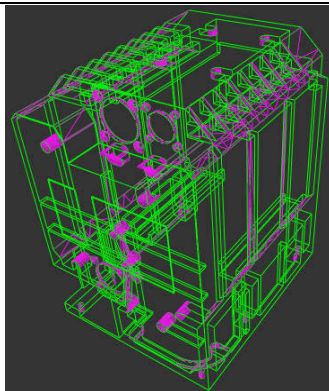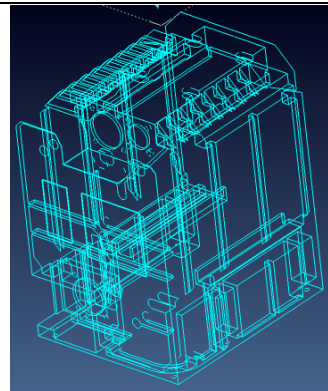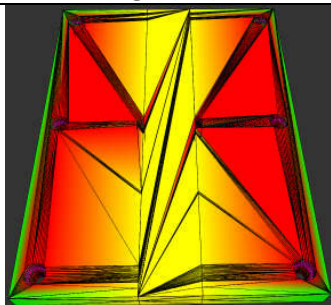| *Mesh with angle defect measured at vertices.*	*Edges filtered on the basis of their dihedral angle.*	*Reverse Engineered object represented using 2D geometric features. Rendered in VISI®.*

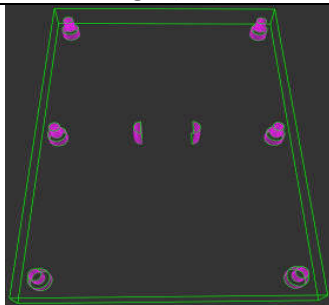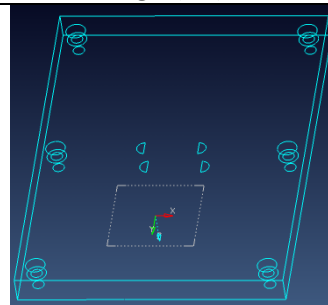Fig. 8(a-1).

Fig. 8(a-2).

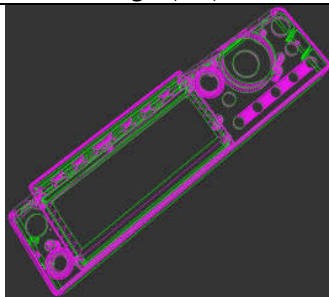Fig. 8(a-3).

Fig. 8(b-1).

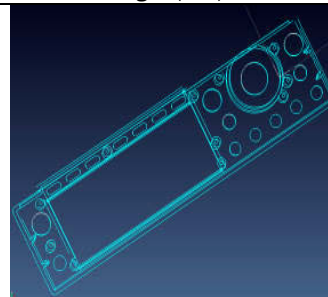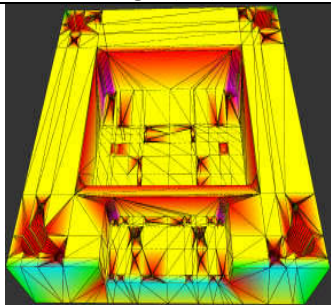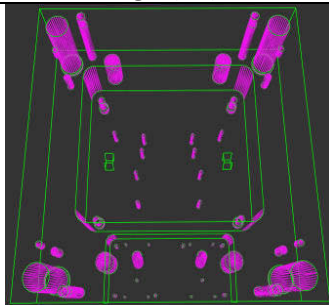Fig. 8(b-2).

Fig. 8(b-3).
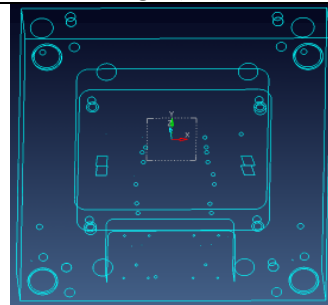

Fig. 8(c-1).

Fig. 8(c-2).

Fig. 8(c-3).

Fig. 8(d-1).

Fig. 8(d-2).

Fig. 8(d-3).