Feature-based 3D non-manifold freeform object construction

Charlie C. L. Wang*

Department of Automation and Computer-Aided Engineering, The Chinese University of Hong Kong,

Shatin, N.T., Hong Kong

Yu Wang Matthew M. F. Yuen

Department of Mechanical Engineering, The Hong Kong University of Science and Technology,

Clear Water Bay, N.T., Hong Kong

Abstract

This paper presents a novel technique for modeling a 3D non-manifold freeform model around a 3D reference model. In order to represent both the design abstractions and the incomplete topological information, first of all, a new non-manifold data structure is defined. Our data structure embodies the functional vitalities of both the boundary representation data structure and the complex-based data structure. Along with our data structure, a set of topological operators is defined to manipulate the entities in the data structure. Based on the non-manifold data structure and the topological operators, we developed a technique to construct 3D freeform objects around a reference model. Intuitive 2D sketches are adopted to specify the detail profile of the constructed object. The construction method is feature based – every reference model has pre-defined features, and the feature template of the constructed object is related to the features of the reference model by feature node encoding. Therefore, the surfaces derived from one reference model can be regenerated automatically on another reference model with the same features. The geometry coverage of our geometric modeling approach includes both manifold and non-manifold 3D freeform objects.

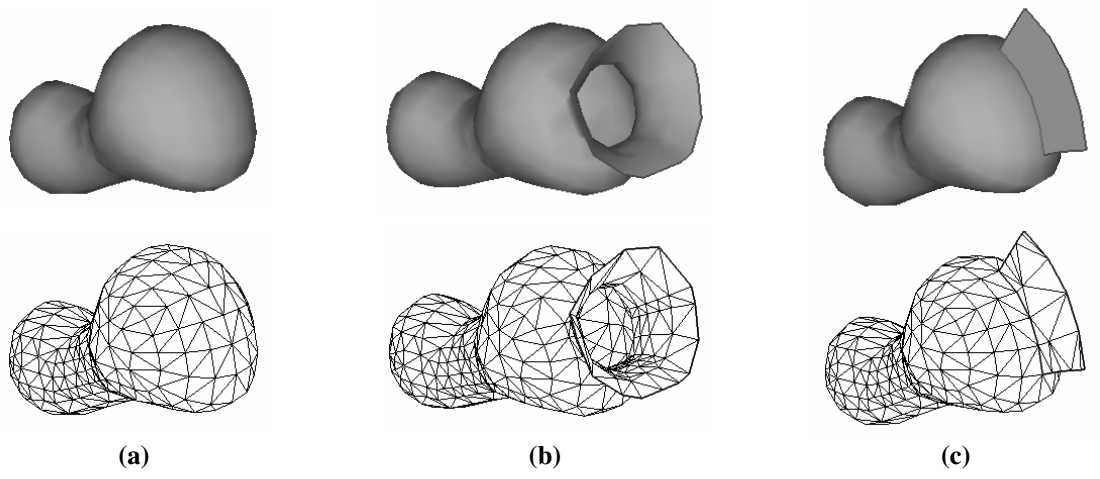
Keywords: non-manifold model, feature template, reference model, 2D sketches, geometric modeling.

1. Introduction

In design processes and engineering analyses, design abstractions expressed as lines or surfaces are integral parts of the conceptual model for a physical object. This conceptual modeling approach can be equally applied to non-physical objects. In geometric modeling, such an approach is commonly referred to as *non-manifold geometric modeling* or *non-homogeneous geometric modeling* in view of the nature of the modeling domain.

* Corresponding Author; E-mail: cwang@acae.cuhk.edu.hk or mewangcl@hotmail.com

This paper is going to develop a new approach for modeling 3D non-manifold freeform objects. The modeling method is feature-based – the constructed object is around a 3D reference model by a feature template; every reference model has pre-defined features; and the feature template of a constructed object is related to the features on the reference model. Therefore, the surfaces derived from one reference model can be regenerated automatically on another reference model with the same features, which greatly improves the efficiency of 3D object modeling. The geometry coverage of the modeling method presented in this paper includes closed surfaces (e.g., Fig. 1a), open surfaces (e.g., Fig. 1b), and more complex non-manifold surfaces (e.g., Fig. 1c).



(a) **(b)** **(c)**

Fig. 1 Closed surface, open surface, and more complex non-manifold surface

Abstractions are frequently used along with the models for physical objects, which leads to the non-manifold geometric modeling. With the exception of this, a non-manifold data structure is also used to store the incomplete topological information generated in the topology designing process of a product. Thus, first of all, a non-manifold data structure is defined in section 3. Our data structure embodies the functional vitalities of both the boundary representation data structure and the complex-based data structure, so the design abstractions and the incomplete topological information can be stored. Consideration has also been made to design a set of topological operators to relieve the geometric modeling system from specific and complex manipulation on the underlying data structure. Based on the non-manifold data structure and the topological operators, section 4 introduces a method to construct 3D freeform objects around a reference model. The feature template for a 3D freeform object is designed according to the features on a reference model by feature node encoding and topological graph construction. 2D sketches are conducted to specify the 3D profiles of the final object on the feature template. After that, the smooth mesh surfaces interpolating the feature nodes and profiles in a feature template are computed. Both manifold and non-manifold surfaces can be constructed. At the end of the paper, in order to demonstrate the functionality of our approach, an application in the apparel industry is given.

2. Related Work

Modeling of non-manifold object

Weiler (1986) [1] presented the first significant work on the non-manifold model. An edge-based data structure called the Radial Edge Structure (RES) was proposed. In order to represent the non-manifold adjacency relationships at vertices, edges, and faces, he introduced the *face-use*, *loop-use*, *edge-use*, and *vertex-use* topological entities in association with the *face*, *loop*, *edge*, and *vertex* entities, respectively. However, in the RES, it is impossible to form a correct shell using only topological data when a non-manifold vertex has to be traversed. To overcome this drawback of the RES, Choi (1989) [2, 3] proposed the Vertex-based Boundary Representation (VBR), in which the zone and disk topological entities are introduced to represent the inclusive relationships between the local regions at a vertex. Some other approaches are complex-based representation [4-6]. Since the complex-based data structure, unlike the afore-mentioned data structure, is based on a simple incidence graph that has no ordering information, it does not enable easy computation of certain important properties (orientability, for instance). A more recent research on the representation of non-manifold models is the Partial Entity Structure (PES) [7], which is a compact non-manifold boundary representation. The storage size of the PES is reduced to half of the radial edge structure (RES). However, incomplete boundaries cannot be represented in this data structure, which is vital to maintain in a conceptual design. Our data structure is a combination of the boundary representation and the complex-based representation, which can overcome the above inadequacies.

In topological modeling, a set of basic topological operators is utilized to manipulate the entities. The topological operators for non-manifold modeling are based on the *non-manifold Euler-Poincarè formula* [8-11]. Not only is a minimal set of the Euler operators required but also a practically sufficient set of the operators is needed to enable efficient implementation of high-level modeling capabilities. Masuda (1993) [10] defines operators for his complex-based data structure; and Lee (2001) [7] also presented a set of operators related to the PES. In this paper, the topological operators catering for the new non-manifold data structure are also included.

Feature based freeform mesh object modeling

In the past, many freeform modeling approaches have been developed. Some of them are related to surface construction [12-14]; some are interactive modification methods [15-17]; and others are deformation techniques [18-24]. The freeform object construction method presented in this paper is feature-based. It relates directly to the feature-based freeform object modeling and the mesh fitting techniques.

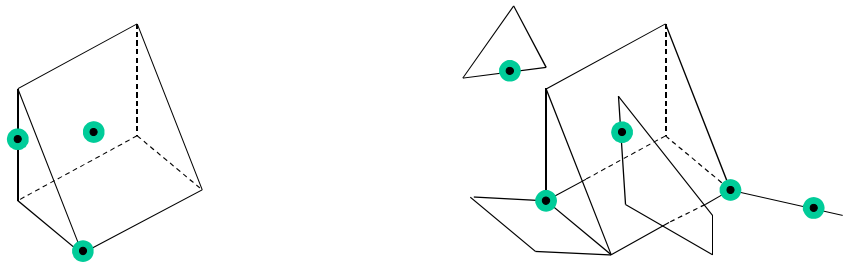
Feature-based object modeling has been studied for a long time [25-34]. In feature-based modeling, object semantics are systematically represented for a specific application domain; in other words, a semantic feature is an application-oriented feature defined on geometric elements. There are two approaches for building a feature model [32]: 1) The design by feature approach that creates the feature model of an object by composing the available features in a feature library; 2) The feature recognition approach that recognizes various features from a geometric model of an object according to the features defined in a feature library. In our feature-based object construction approach, a feature template related to a reference model is built and profiles of the feature template are specified using 2D sketches. After the profiles of a feature template are defined, a mesh surface fitting the profiles is constructed. This is a new approach as the technique involves no feature recognition and uses a feature template library rather than a feature library. It allows more flexibility in the design environment.

In our 3D object construction algorithm, we build the 3D mesh surface from the 3D profile curves, which is converted from 2D sketches input and with an arbitrary topology. The constructed surface interpolating the given 3D curves must be smooth. There are two approaches to construct smooth mesh surfaces interpolating given curves with an arbitrary topology: the combined subdivision scheme [35, 36], which is subdivision mask based; and the variational subdivision scheme [37], which is discrete fairing based. In our approach, not all the edges in the feature template mesh have interpolating curves; this does not satisfy the initial condition of the combined subdivision scheme. Thus, the variational subdivision scheme is chosen. The surface constructed is around a 3D reference model, and collision between the constructed surface and the reference model should be prevented during the shape construction process. The variational subdivision scheme is modified by integrating the collision detection. In order to achieve an efficient scheme, the voxel-based approach [38] is used to detect the collision.

3. Topological Data Structure for Non-manifold Mesh Modeling

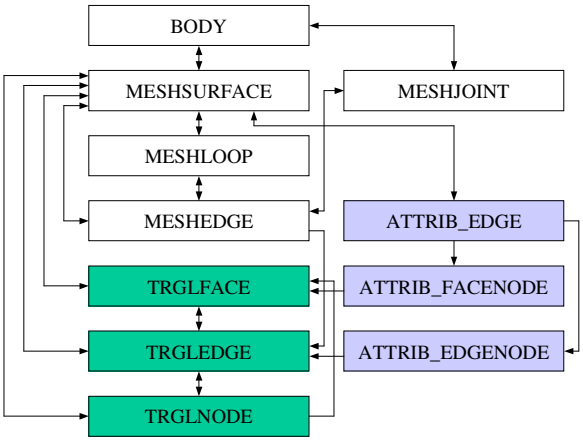
The geometric coverage of conventional boundary representation solid modelers is confined to the domain of two-manifold objects. For every point on the boundary of a two-manifold object, there exists a sufficiently small neighborhood that is topologically the same as an open disk in $\Re^2$. If there are any points on the boundary that do not satisfy the two-manifold condition, the object is classified as *non-two-manifold*, or simply *non-manifold* (see Fig. 2). While almost all physical artifacts in the world are two-manifold objects, in terms of modeling, the domains of two-manifold ones cannot easily accommodate the entities of a lower dimensionality, such as stand-alone faces and wireframe edges. However, such entities are important in engineering design for

representing the abstraction of a geometric shape (e.g., the 3D patterns in computer-aided garment design, which are assembled stand-alone surfaces; injection model parts or any thin-walled components).



(a) two-manifold **(b) non-two-manifold**

Fig. 2 Two-manifold vs. non-two-manifold



BODY	Entire Model Complex	MESHSURFACE	Mesh Surface
MESHLOOP	Mesh Surface Loop	TRGLFACE	Triangular Face
MESHEDGE	Mesh Surface Edge	TRGLEDGE	Triangular Face Edge
MESHJOINT	Mesh Surfaces Joint	TRGLNODE	Triangular Face Node
ATTRIB_EDGE	Curves on Mesh Surface	ATTRIB_FACENODE	Nodes on Triangular Face
ATTRIB_EDGENODE	Nodes on Triangular Edge		

Fig. 3 Data structure framework

Data structure

In order to catch the representation of geometric abstractions and the incomplete topological information, a non-manifold data structure for geometric object modeling by triangular meshes is to be constructed. The non-manifold elements such as stand-alone faces or wireframe edges can be stored in the new triangular mesh entity – MESHSURFACE. The important adjacency information for non-manifold model – the order of faces around an edge is stored in a new entity – MESHJOINT. The proposed data structure will embody the functional vitalities of both the boundary representation data structure and the complex-based data structure. The framework of our data structure is defined as shown in Fig. 3.

There are many ways of viewing this data structure: for instance, it can be thought of as being a tree, with BODY as its root. A BODY has a collection of MESHSURFACEs, each of which is comprised of many MESHEDGEs, MESHLOOPs, TRGLFACEs, TRGLEDGEs, and TRGLNODEs; and a BODY also has a collection of MESHJOINTs, each of which is comprised of some ordered MESHEDGEs. A TRGLFACE consists of 3 TRGLEDGEs. A TRGLEDGE is a line segment ended by 2 TRGLNODEs. Each MESHSURFACE is bounded by MESHLOOPs, which consists of several MESHEDGEs. Each MESHEDGE has a collection of TRGLEDGEs; and each TRGLEDGE has its own direction flag in the MESHEDGE. The adjacent information of MESHSURFACEs at some MESHEDGEs is stored in a new entity – MESHJOINT. Each MESHJOINT has a collection of MESHEDGEs, which contain the same number of TRGLEDGEs, and the TRGLEDGEs are one to one connected (as shown in Fig. 4). If a MESHEDGE is in the same direction with a MESHJOINT, it is defined as a positive one in the MESHJOINT; otherwise, it is defined as a negative one. The MESHEDGEs in a MESHJOINT are stored in a clockwise order by the right-hand rule (as shown in Fig. 4; where M1, M2, and M3 are three MESHSURFACEs, E1, E2, and E3 are their related MESHEDGEs which contain the same number of TRGLEDGEs, and EO is the MESHJOINT containing the adjacent information). The detail description of each entity is shown in Table 1 (Pseudo code is shown in Appendix I). Using the data structure, it is easy to carry out any topological and geometrical manipulation on the manifold or non-manifold triangular mesh models.

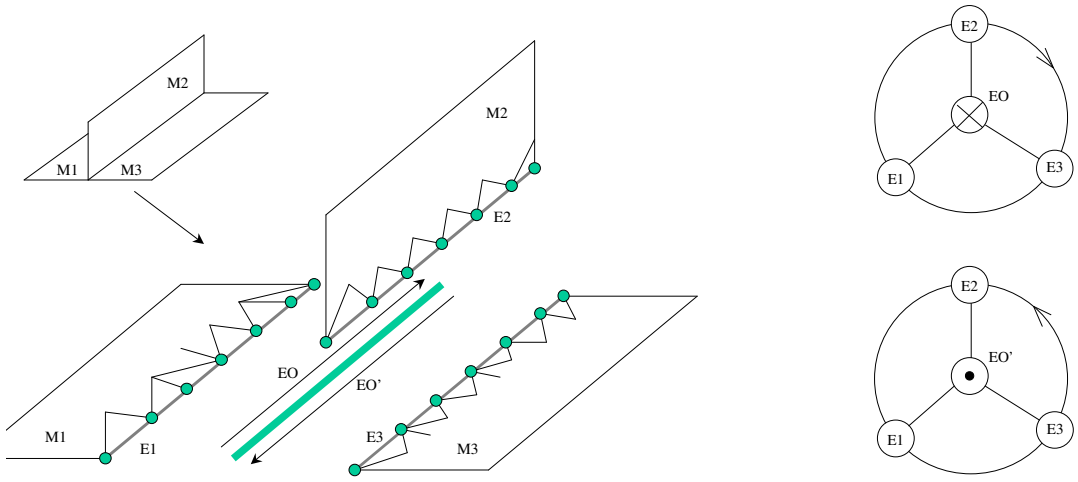


Fig. 4 Clockwise list of MESHEDGEs in a MESHJOINT

We define four attributes in our data structures. They include ATTRIB_NODE, ATTRIB_EDGE, ATTRIB_EDGENODE, and ATTRIB_FACENODE, where ATTRIB_EDGENODE and ATTRIB_FACENODE are derived from ATTRIB_NODE. ATTRIB_EDGENODE is the attribute node on a TRGLEDGE, and ATTRIB_FACENODE is the attribute node in a TRGLFACE. Their coordinates depend on the position of TRGLEDGE's nodes or the position of TRGLFACE's nodes. In detail, the coordinate of an

ATTRIB_EDGENODE is represented by a parameter u related to the nodes of a TRGLEDGE; and the coordinate of an ATTRIB_FACENODE is represented by (u,v,w) – the parametric area coordinate of a TRGLFACE. An ATTRIB_EDGE is an ordered collection of ATTRIB_NODEs, which can be either ATTRIB_EDGENODEs or ATTRIB_FACENODEs. The detail description of each attribute is shown in Table 2 (Pseudo code is shown in Appendix II).

Table 1 Representational Entities

Entity	Representation	Description
BODY	Complex of MESHSURFACEs and MESHJOINTs	Highest level entity in a model.
MESHSURFACE	Complex of MESHEDGEs, MESHLOOPs, TRGLFACEs, TRGLEDGEs, and TRGLNODEs	A portion of BODY's surface. It defines the shape of BODY, and is represented by many triangles.
MESHLOOP	Complex of MESHEDGEs	Connected portion of a MESHSURFACE's boundary.
MESHJOINT	Complex of MESHEDGEs	Assembly information of MESHSURFACEs.
MESHEDGE	Complex of TRGLEDGEs	A portion of MESHLOOP. It defines the shape of MESHLOOP. (+ve, clockwise; and -ve, anti-clockwise)
TRGLFACE	Complex of three TRGLEDGEs	Portion of a MESHSURFACE.
TRGLEDGE	Complex of two TRGLNODEs	Boundary of a TRGLFACE, holds the model together with adjacency information. (+ve, clockwise; -ve, anti-clockwise)
TRGLNODE	A point	Boundary of a TRGLEDGE.

Table 2 Representational Attributes

Attribute	Representation	Description
ATTRIB_NODE	A point	An attribute point on the surface of a MESHSURFACE.
ATTRIB_EDGE	Complex of ATTRIB_NODEs	An attribute curve lying on the surface of a MESHSURFACE. It is a list of ATTRIB_NODEs, and passes triangular faces of the MESHSURFACE.
ATTRIB_EDGENODE	A point	An attribute point on a triangular edge. Its position depends on the positions of the two endpoints of the edge.
ATTRIB_FACENODE	A point	An attribute point in a triangular face. Its position depends on the positions of the three nodes of the face.

Topological operators

The construction of a valid geometric model is achieved through the use of a proper set of topological operators. In geometric modeling, the fundamental topological operators are Euler operators [7, 10] that are consistent with the *Euler-Poincarè formula*. Likewise, the extended topological operators for non-manifold geometric modeling have to satisfy the same formula. Theoretically, just nine independent Euler operators and their inverse operators are sufficient to define all complex based non-manifold geometric models [10]. Volume is not included in our approach; therefore only eight extended Euler operators are utilized (shown in Appendix III), and they are restricted to the triangular meshes.

When editing a model, often several repeated sequences of the extended Euler operators are used. These sequences are formulated as high level editing operations. Five of these sequences are formulated as high level editing operations, these include *edge collapse*, *edge split*, *edge swap*, *face split*, and *face triangulation*. These high level operators are provided to automate the performance of the extended Euler operator sequences and increase the efficiency of topological operations, and they are frequently used in triangular mesh processing algorithms (i.e., *edge collapse*, *edge split*, and *edge swap* was utilized for mesh optimization [39]; *face split* operator was conducted in the famous *Loop Subdivision Scheme* [40], *Variational Subdivision Scheme* [37], and *Modified Butterfly Subdivision Scheme* [16]; and *face triangulation* was applied for remeshing [41]). The detail description of these operators is listed as follows. Their sequences of extended Euler operators are shown in Table 3; and their illustration is shown in Fig. 5.

Table 3 Sequences of Extended Euler Operators for High Level Operators

Operator	**Sequence of extended Euler operators**
edge split	*kill_face_make_Chole* (2 times) $\Rightarrow$ *split_edge* $\Rightarrow$ *make_edge_Chole* (2 times) $\Rightarrow$ *make_face_kill_Chole* (4 times)
edge collapse	*kill_face_make_Chole* (for all faces sharing P_d) $\Rightarrow$ *kill_edge_Chole* (for all edges sharing P_d) $\Rightarrow$ *kill_vertex_complex* (for P_d) $\Rightarrow$ *make_edge_Chole* (connecting to P_e for all original faces sharing P_d except f_1 and f_2) $\Rightarrow$ *make_face_kill_Chole* (appropriate times)
edge swap	*kill_face_make_Chole* (2 times) $\Rightarrow$ *kill_edge_Chole* $\Rightarrow$ *make_edge_Chole* $\Rightarrow$ *make_face_kill_Chole* (2 times)
face split	*kill_face_make_Chole* $\Rightarrow$ *split_edge* (3 times) $\Rightarrow$ *make_edge_Chole* (3 times) $\Rightarrow$ *make_face_kill_Chole* (4 times)
face triangulation	*kill_face_make_Chole* $\Rightarrow$ *make_vertex_complex* (by ATTRIB_FACENODE), *make_vertex_edge* (by ATTRIB_EDGE), split_edge (by ATTRIB_EDGENODE) $\Rightarrow$ *make_edge_Chole* (appropriate times) $\Rightarrow$ *make_face_kill_Chole* (appropriate times)

Edge split: For a triangular edge e, the edge split operator involves introducing the edge midpoint P_m that separates the edge e into two new edges, e_1 and e_2. The two triangles, f_1 and f_2, sharing edge e are replaced by four new triangles, $f_{j,j=1,\ldots,4}$. One new vertex, P_m, and four new edges, $e_{j,j=1,\ldots,4}$, are created. And all topological information of entities sharing edge e and its endpoints should also be altered (see Fig. 5a). If the edge e has a MESHJOINT connected, the corresponding TRGLEDGEs of e connecting the same MESHJOINT should be applied to the same split operation as e.

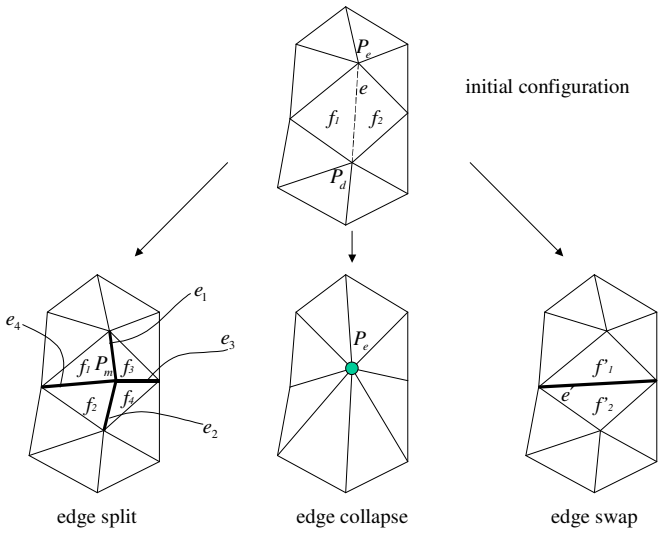
Edge collapse: The edge collapse operation is based on the identification of the two endpoints of edge e, thus leading to a unique point P_e that can be either one of the original edge endpoints. This operation removes the two triangles, f_1 and f_2, their shared edge e, and two edges sharing the deleted vertex P_d in f_1 and f_2; and replaces the endpoints of edges sharing P_d to P_e (see Fig. 5a). If the edge e has a MESHJOINT connected, the same as edge split, the corresponding TRGLEDGEs of e connecting the same MESHJOINT should be applied to the same collapse operation as e.

Edge swap: The two triangles (f_1 and f_2) sharing edge e are replaced by two new triangles (f_1' and f_2') sharing the dual edge e' of e. All topological information of entities sharing endpoints of edge e and endpoints of e' should be alternated (see Fig. 5a). If the edge e has a MESHJOINT connected, this operation is simply prevented.

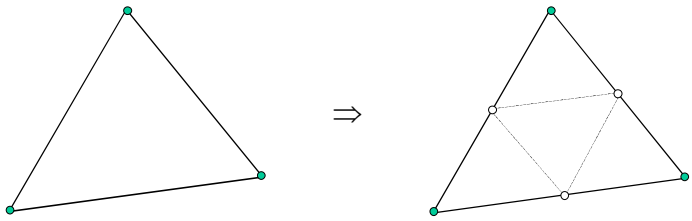
Face split: The face split operator subdivides one triangular face into four triangular faces uniformly, where three new vertices are introduced to divide each triangular edge into two edges, new triangular edges and faces are constructed to link these new vertices. The illustration of the face split operator is shown in Fig. 5b. If any edge of the triangular face shares a MESHJOINT with another edge, its corresponding TRGLEDGEs in the same MESHJOINT should have a stand-alone vertex inserted in the middle, and its related faces are re-triangulated by the following *face triangulation* operator.

Face triangulation: This operator triangulates the face with stand-alone attribute vertices and the edges on it. The vertices and edges are stored as ATTRIB_ FACENODEs, ATTRIB_EDGENODEs, and ATTRIB_EDGEs in the data structure. This operator converts ATTRIB_FACENODEs and ATTRIB_EDGENODEs to TRGLNODEs, and the TRGLEDGE with ATTRIB_EDGENODEs defined is divided into several TRGLEDGEs. The ATTRIB_EDGEs are converted to TRGLEDGEs. New TRGLEDGEs and TRGLFACEs are

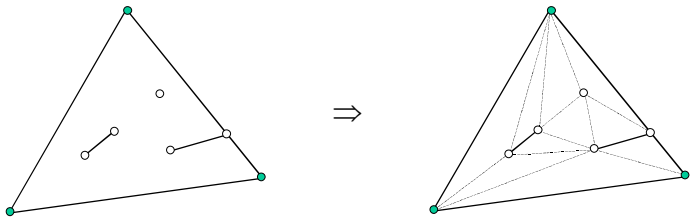
constructed to connect edges and nodes. The original TRGLFACE is removed. The new constructed TRGLFACEs must be in the same orientation and not overlapped, and the newly constructed TRGLEDGEs must not intersect with other triangles. One example of face triangulation is shown in Fig. 5c.



(a) *edge collapse*, *edge split*, and *edge swap*



(b) *face split*

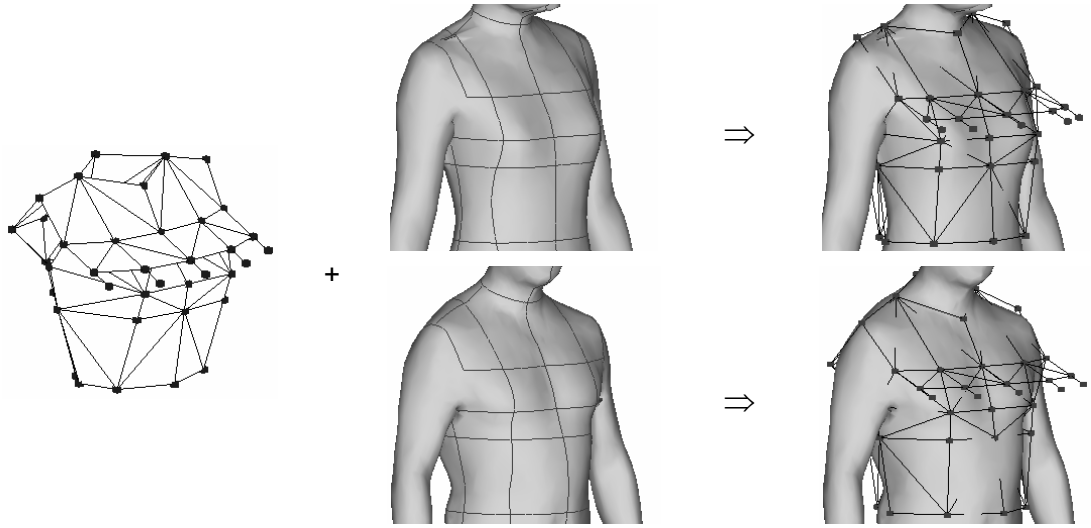


(c) *face triangulation*

Fig. 5 High level topological operators

4. Feature-based Model Construction

This section focuses on a new method of modeling a 3D freeform object around a 3D reference model. The modeling method is feature-based – the constructed object is around a 3D reference model by a feature template; every reference model has pre-defined features; and the feature template of the constructed surfaces are related to the features on the reference model. The feature template of the constructed object is conducted to regenerate the object automatically when using another reference model with the same features.



(a) feature template **(b) reference models** **(c) different shapes**

Fig. 8 Different shapes of the same template on different reference models

Feature template for freeform object models

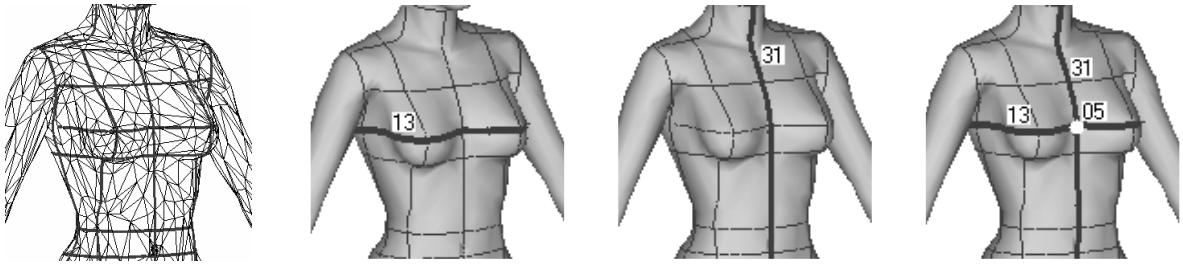
Freeform objects can be classified in accordance with their feature-based topological structure [33] and can be stored as different object feature templates. Each object feature template is represented by a set of triangular mesh surfaces - MESHSURFACEs, which is assembled by MESHJOINTs and stored in a BODY entity B^f. In a feature template B^f, the position of each vertex (which is called *feature node* and stored by TRGLNODE) is related to a relevant feature on a reference model. Thus, when we apply the same template to different reference models with the same features, we obtain the triangular meshes of different shapes (see Fig. 8).

The semantic feature curves and points of a reference feature model are first defined. In the following feature node encoding process, the relationship between the position of each feature node and the features on a reference model are determined. After that, the topological graph of B^f is designed interactively. The 3D profiles of the feature template are specified by 2D sketches and stored in B^f by ATTRIB_EDGENODEs.

4.1.1 Reference model

In our approach, a reference model H is also represented by a triangular mesh with a set of pre-defined semantic feature curves. Each semantic feature curve consists of a sorted set of line segments lying on the mesh surface of the reference model. The line segments might not be the topological edges on the reference model; they can pass through triangular faces (see Fig. 9a). The feature curves are stored by the ATTRIB_EDGE attributes in our data structure (defined in section 3.1). The line segments that belong to a specific semantic feature curve have a common feature ID number. Each semantic feature point is an intersection point of two semantic feature curves, and is stored by the ATTRIB_NODE attribute. Every semantic feature point has its own

ID number. For example, in Fig. 9b, the bold line segments with ID number 13 represents the chest feature curve of a human model; in Fig. 9c, the bold line segments with ID number 31 represents the center-front feature curve of a human model; and in Fig. 9d, the white point is the feature point determined by these two feature curves, the ID number of the feature point is 05.



(a) **(b)** **(c)** **(d)**

Fig. 9 Feature curves and points on a reference model

The position and orientation of the designed feature template is highly dependent on the position and distribution of the semantic feature curves and points. Thus, the location of the feature curves and points on the reference model is important. In other words, they must point out the features accurately. For example, if the chest feature curve of a human model (No.13 curve in Fig. 9b) were in a wrong position – upper or lower, the final constructed 3D object would have a bad shape at the real chest position. Therefore, the feature exaction algorithm should be robust enough to support variations in positioning. Since this is not a major focus of the paper, we only implement the method represented in [34] to get reference models with features.

4.1.2 Feature nodes encoding

As mentioned at the beginning of section 4.1, the position of each feature node is related to the features on reference model H. The relationship is built between a feature node $\vec{v}_i$ in the object feature template and a feature point q_j on H, where j is the ID number of point q_j. The position vector $\vec{p}(q_j)$ of q_j is determined from the intersection of two feature curves, f_1 and f_2. Since the feature point q_j lies on f_1 and f_2, the unit tangent vectors of feature curves f_1 and f_2 at q_j – $\vec{t}(f_1)$ and $\vec{t}(f_2)$, and the unit normal vector $\vec{n}(q_j)$ of the surface of H at q_j are well defined. The vectors $\vec{t}(f_1)$, $\vec{t}(f_2)$, and $\vec{n}(q_j)$ form a local coordinate frame as shown in Fig. 10. The three vectors may not be an orthogonal set, but it is obvious that the scalar product $\left(\vec{t}(f_1)\,\vec{t}(f_2)\,\vec{n}(q_j)\right) \neq 0$. Thus, the position $\vec{p}(\vec{v}_i)$ of any vertex $\vec{v}_i \in V^f$ can be represented by

$$\vec{p}(\vec{v}_i) = \vec{p}(q_j) + \alpha_i \vec{n}(q_j) + \beta_i \vec{t}(f_1) + \gamma_i \vec{t}(f_2). \quad (1)$$

Each vertex $\vec{v}_i$ thus consists of four elements: $(j, \alpha_i, \beta_i, \gamma_i)$. The encoding process, which relates one feature node $\vec{v}_i \in V^f$ on the object model to a feature point q_j on the reference model H, is actually a process to determine these four elements of vertex $\vec{v}_i$.

Equation (1) defines the relationship between the positions of $\vec{v}_i$ and q_j. Another popular form used to define the relationship between the positions of the two points is

$$\vec{p}(\vec{v}_i) = \vec{p}(q_j) + \vec{d} \tag{2}$$

where the position of vertex $\vec{v}_i$ is also determined by four elements: (j, d_x, d_y, d_z). When a new reference model H′ with the same features is used, the new position of each vertex $\vec{v}_i$ in Fig. 10 can be easily calculated by equation (1) or (2). As shown in Fig. 11, $\vec{p}_1(\vec{v}_i)$ is the new position determined by equation (1), and $\vec{p}_2(\vec{v}_i)$ is the new position determined by equation (2). It was found that the orientation of the feature node related to a reference model is not guaranteed when using equation (2), but equation (1) strongly preserves the orientation. This is an important factor in many industrial applications, so $(j, \alpha_i, \beta_i, \gamma_i)$ is chosen as the four elements to determine the position of $\vec{v}_i$ according to equation (1).

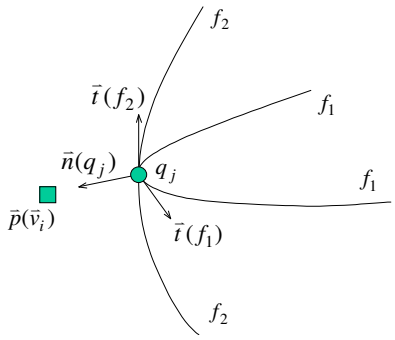


Fig. 10 Local frame on feature point q_j

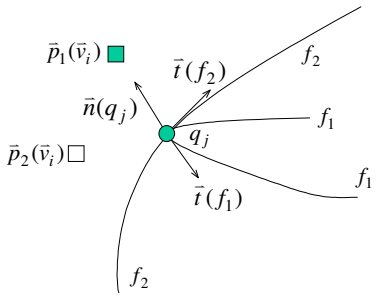


Fig. 11 New position of $\vec{v}_i$ determined

4.1.3 Build topological graph

Using interactive tools, we can define the feature nodes of a feature template B^f around the reference model H by determining the four parameters of a feature node. After that, the topological graph of B^f is to be specified. The topological graph is a collection of MESHSURFACEs that are connected by MESHJOINTs. For example, Fig. 12a shows the topological graph of a non-manifold object that consists of four MESHSURFACEs (in different colors). This object includes both stand-alone faces and wireframe edges. The bolded edges are MESHJOINTs. The topological graph can be constructed using interactive tools. Since our data structure is complex-based, the incomplete topology information during the construction process is easy to be stored. Fig.

12b-12h shows some fragments of the construction process of the topological graph. In Fig. 12b, the feature nodes have been defined around the reference model. The feature nodes are generally outside the reference model, and their positions actually define the final coarse shape of the constructed object; so the feature nodes should be chosen very carefully. An interactive tool is used to connect the feature nodes by edges as shown in Fig. 12c; and Fig. 12d shows the feature template after creating all edges. Triangular faces can also be created one by one interactively (Fig. 12e and 12f). Fig. 12f and 12h show the final result.

By the limitation of the shape construction that will be presented in section 4.2, the feature template should be in a good shape and orientation. Sharp angle triangles are required to be prevented; triangular faces belonging to one MESHSURFACE should have the same orientation (i.e., two adjacent triangles are not allowed to have reversed normal directions); and an asymmetrical topological graph will lead to an asymmetrical final object even if the distribution of the feature nodes is symmetric.

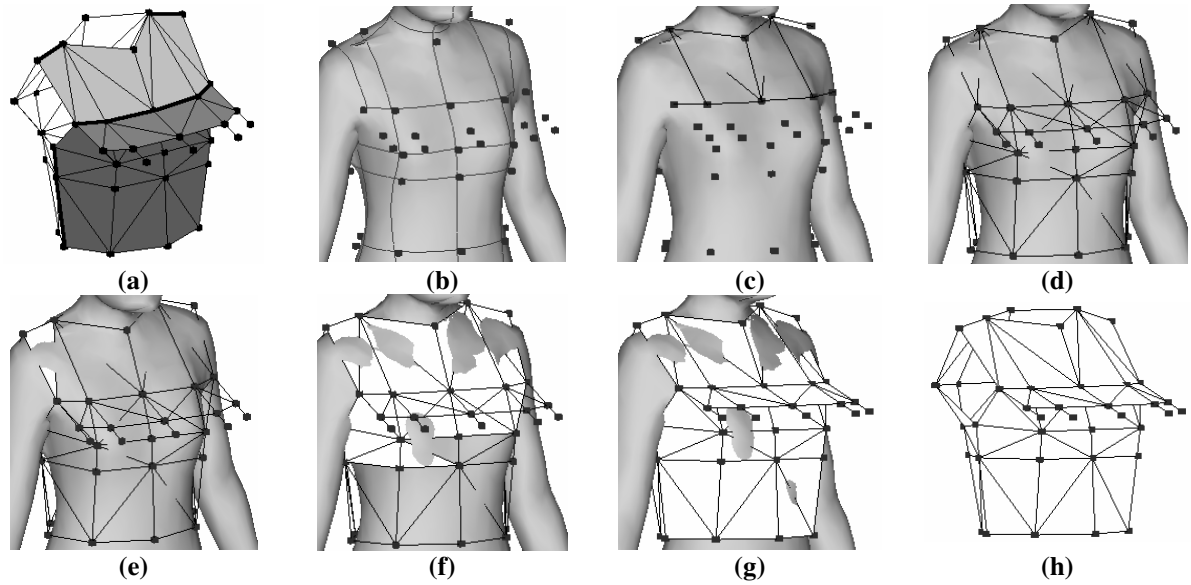


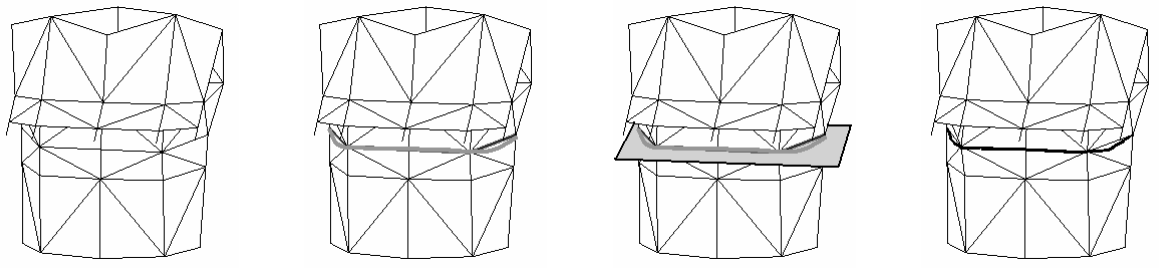
Fig. 12 An example of topological graph construction process

4.1.4 Profile specification

After the coarse shape of the feature template is determined, 2D sketches are used to specify the 3D profiles of some triangular edges in the feature template, which describe the detail shape of the constructed object and are interpolated in the shape construction step. The 3D profile of a triangular edge is a 3D curve whose two endpoints coincide with the two endpoints of the edge. In order to obtain a good shape of the constructed object, the 3D profiles are expected to be smooth at its endpoints. Each 3D profile is represented by a list of parameterized 3D points attached on a triangular edge. In our implementation, the points are stored by ATTRIB_EDGENODEs.

Two problems arise when using a 2D stroke Ψ to specify a 3D: 1) whose profiles are specified by Ψ, and 2) how to determine the positions of the 3D attached points by Ψ. After solving these two problems, the 2D stroke Ψ is converted to the 3D profiles attached to some edges.

For a given feature edge $\vec{v}_i\vec{v}_j$ with endpoints $\vec{v}_i$ and $\vec{v}_j$, it is selected by the stroke Ψ if $\vec{v}_i\vec{v}_j$ is visible and the distances between $\zeta(\vec{v}_i)$, $\zeta(\vec{v}_j)$ and Ψ are less than ε, where $\zeta:\Re^3 \to \Re^2$ is the map that sends spatial point $\vec{v}_i \in \Re^3$ to screen point $\psi_i \in \Re^2$, and ε is a small tolerance value (e.g., $\varepsilon = 4$ pixels). In the following, a plane is determined to project the points $\psi_i \in \Psi$ to convert them into 3D points $\psi^*_i \in \Psi^*$ in $\Re^3$, where Ψ^* is a list of 3D points. When multiple feature edges are selected, a plane that approximately passes through the selected feature edges is conducted. If only one feature edge is selected, we use the plane that bisects the dihedral angle along the chosen edges to project the points $\psi_i \in \Psi$. In this way, the sketched profile faces towards the camera as much as possible. After all $\psi_i \in \Psi$ are converted to $\psi^*_i \in \Psi^*$ in $\Re^3$, we separate them into intervals and store the points in one interval in its related edge. For a selected feature edge $\vec{v}_l\vec{v}_m$, we search the closest point ψ^*_l to $\vec{v}_l$ in Ψ^*, and the closest point ψ^*_m to $\vec{v}_m$ in Ψ^*. Then, the points $\psi^*_{j,\ j=l,\ldots,m}$ are stored as the attached points list in $\vec{v}_l\vec{v}_m$.



(a) **(b)** **(c)** **(d)**

Fig. 13 Specify a profile through a 2D stroke

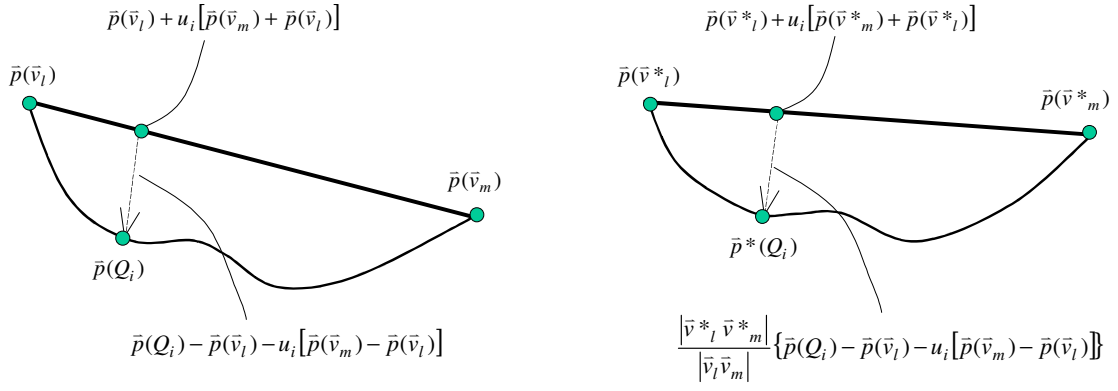
One example of specifying profiles through a 2D stroke is given in Fig. 13. Fig. 13a shows the feature graph before specifying a profile at the chest; in Fig. 13b, a 2D stroke is input; Fig. 13c shows the determined projection plane; and Fig. 13d shows the result.

When re-generating the 3D object on a reference object with a different shape, the positions of the feature nodes in the feature template are changed. We need to shift the position of the attached points on the triangular edges to re-generate the "new" 3D profiles. If an edge is moved from $\vec{v}_l\vec{v}_m$ to $\vec{v}^*_l\ \vec{v}^*_m$, the new positions of points $Q_{i,\ i=0\ldots n-1}$ attached to it are shifted by scaling the vector between them and $\vec{v}_l\vec{v}_m$. The idea is shown in

Fig. 14, where the vector between Q_i and $\vec{v}_l\vec{v}_m$ is scaled by $\frac{|\vec{v}^*_l\ \vec{v}^*_m|}{|\vec{v}_l\vec{v}_m|}$. The formulation to compute the new positions of Q_i is shown below

$$\vec{p}^*(Q_i) = \frac{|\vec{v}^*_l\ \vec{v}^*_m|}{|\vec{v}_l\vec{v}_m|}\{\vec{p}(Q_i) - \vec{p}(\vec{v}_l) - u_i[\vec{p}(\vec{v}_m) - \vec{p}(\vec{v}_l)]\} + \vec{p}(\vec{v}^*_l) + u_i[(\vec{p}(\vec{v}^*_m) - \vec{p}(\vec{v}^*_l)] \qquad (3)$$

where u_i is the parameter defined by the lengths of line segments [42].



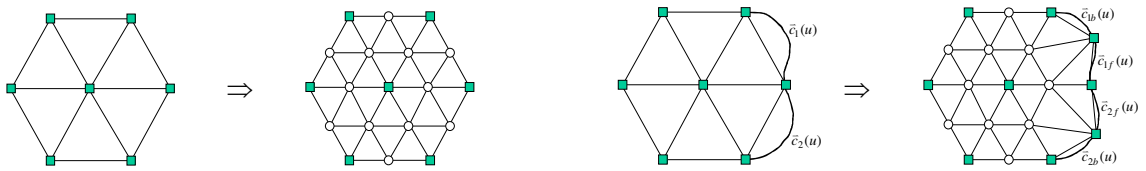
(a) before moving **(b) after moving**

Fig. 14 Shift a 3D profile

Shape construction

In this section, triangular mesh surfaces are constructed from the feature template interpolating its feature nodes and 3D profiles by applying a modified variational subdivision scheme. The feature template is actually assembled coarse triangular mesh surfaces with some interpolating curves (3D profiles). The basic idea of a variational subdivision scheme is to iteratively apply a topological splitting operator to introduce new vertices to increase the degrees of freedom, followed by a discrete fairing operator to increase the overall smoothness [37]. In our approach, the scheme is modified to construct mesh surfaces that interpolate not only the initial vertices but also the specified profiles. The topological splitting operator inserts new control vertices into the mesh. The split operation is chosen to be uniform so that all the new vertices are regular (valance is equal to 6, as shown in Fig. 15a). The position of the inserted new vertex $\vec{v}^*$, which lies on the edge $\vec{v}_s\vec{v}_e$, is determined by $\vec{p}(\vec{v}^*) = \frac{1}{2}(\vec{p}(\vec{v}_s) + \vec{p}(\vec{v}_e))$ if there is no profile specified on $\vec{v}_s\vec{v}_e$; or by $\vec{p}(\vec{v}^*) = \vec{c}(\frac{1}{2})$ if there is a profile specified on $\vec{v}_s\vec{v}_e$, where $\vec{c}(u)$ is the parametric curve that represents the profile shape. We also divide $\vec{c}(u)$ into two parts $\vec{c}_f(u)$ and $\vec{c}_b(u)$ at $\vec{c}(\frac{1}{2})$, and attach $\vec{c}_f(u)$ and $\vec{c}_b(u)$ to the newly created edges from

splitting (in Fig. 15b, $\vec{c}_{1f}(u)$ and $\vec{c}_{1b}(u)$ are from $\vec{c}_1(u)$, and $\vec{c}_{2f}(u)$ and $\vec{c}_{2b}(u)$ are from $\vec{c}_2(u)$). The smoothing operator moves the control vertices according to the weighted averages of neighboring vertices. The positions of vertices in the refined mesh are changed to achieve a global energy functional minimization. Here, we implement the 2nd order umbrella operator as an iterative solver of the problem. As mentioned by Kobblet [37], since each update step only computes a linear combination of nearby vertices, the computational complexity is linear if the number of umbrella iterations is bounded (in fact, a constant number). In order to guarantee that the resultant fine mesh interpolates the originally given vertices, the umbrella operator must not be applied to those vertices that already belong to the initial mesh. Also in order to guarantee that the resultant fine mesh interpolates the 3D profiles, the umbrella operator must not update the positions of the vertices lying on the profiles. The vertices whose positions can be updated are called free vertices (rounded white nodes in Fig. 15), and the vertices whose positions cannot be updated are called fixed vertices (rectangular gray nodes in Fig. 15).



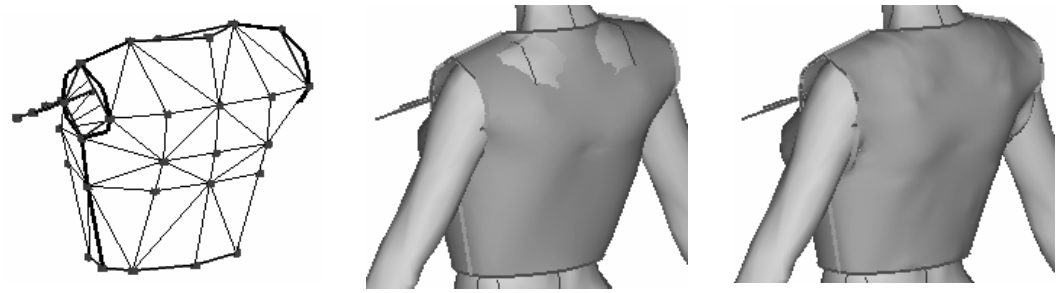
(a) without profile interpolation **(b) with profile interpolation**

Fig. 15 Subdivision

Since the umbrella operator can only be applied to the vertices within the same mesh surface, in order to maintain the connection in the assembled mesh surfaces, the position of vertices on a MESHJOINT must *not* be updated. By following this rule, the modified variational subdivision scheme can be applied to each MESHSURFACE individually to obtain the refined mesh surfaces interpolating the nodes and profiles in a feature template.

The mesh surfaces are constructed around a 3D reference model, so it is likely that collision between the constructed detail mesh surface and the reference object may occur during the surface construction procedure (e.g., the object in Fig. 16b is constructed from the feature template in Fig. 16a without collision detection). Here, the voxel-based collision detection scheme [38] is integrated to prevent collision between the constructed surface and the reference object. First of all, the whole detection space is uniformly subdivided into small voxel spaces. After the topological split operator inserts new vertices, using the coordinate of the new vertex to find a corresponding voxel space, the triangular faces in the voxel space are used to detect whether the free vertex is inside the reference object. If a vertex is inside the reference object, pull it to the outside space of the reference

object along the discrete surface normal direction at the free vertex. In the discrete fairing operation, if the 2nd umbrella-operator pulls a vertex from the outside space of the reference object to the inside space of the reference object, the update is prevented. To obtain a more accurate result, the position update of a vertex in the 2nd umbrella operator is subdivided into several steps, and the update is performed in steps while detecting the collision between the updated vertex and the reference model. The object construction result with collision detection is shown in Fig. 16c.



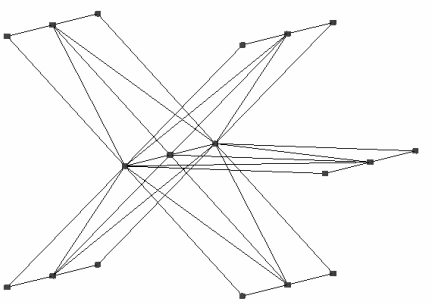
(a) feature template **(b) collision occurs** **(c) final result**

Fig. 16 Collision avoidance

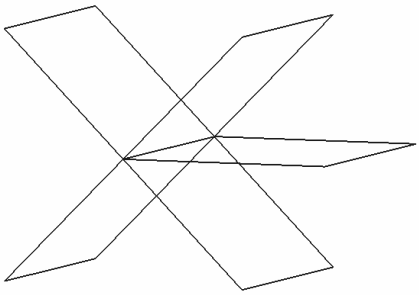
5. Experimental Results

The shape construction method applies the modified variational subdivision scheme to individual MESHSURFACEs to generate the final shape. Thus, as addressed by Hubeli and Gross in [6], different types of the connecting configuration lead to different fairing results. In our approach, different shapes can be constructed by different defined MESHJOINTs. The examples shown in Fig. 17 demonstrate this flexibility of modeling different shapes.

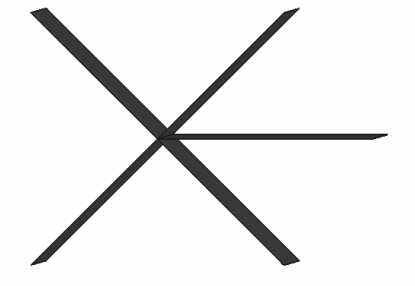
With the same wireframe and the same profiles of the feature template (Fig. 17a and17b), four different final shapes are generated by different connecting configurations. In example I (Fig. 17c and 17d), the boundary edges of the five mesh surfaces are connected by one MESHJOINT, the connecting order of the five mesh surfaces is M1, M2, M3, M4 and M5. In example II (Fig. 17e and 17f), the MESHEDGE of surface M2 connected by the MESHJOINT is an inner edge while the other three edges of M1, M3 and M4 are boundary edges. Two inner edges and one boundary edge are connected by one MESHJOINT in example III (Fig. 17g and 17h). If more complex results were expected, two or more MESHJOINTs can be used – as shown in example IV (Fig. 17i and 17j), one MESHJOINT connects the boundary edge of M1 and the inner edge of M2, and another MESHJOINT connects the inner edge of M2 and the inner edge of M3.



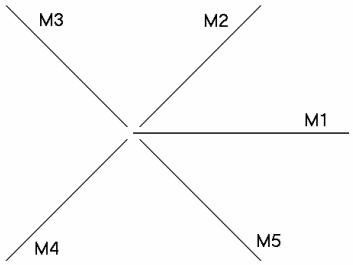
(a) the same wireframe



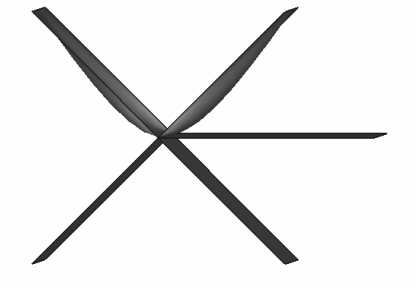
(b) the same profiles



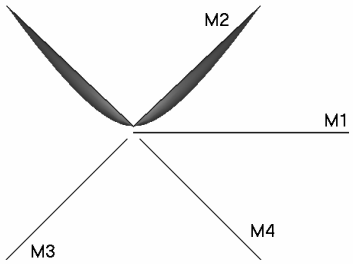
(c) example I



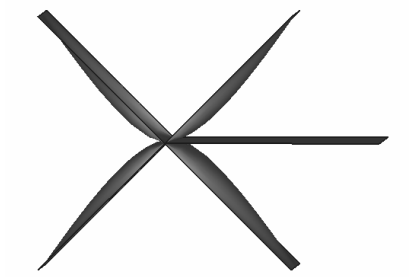
(d) illustration of example I: one MESHJOINT connecting five MESHSURFACEs in order – M1, M2, M3, M4, M5



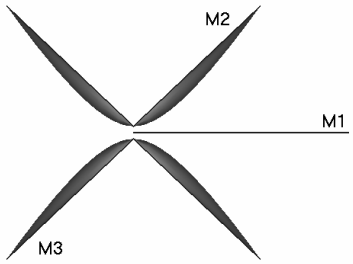
(e) example II



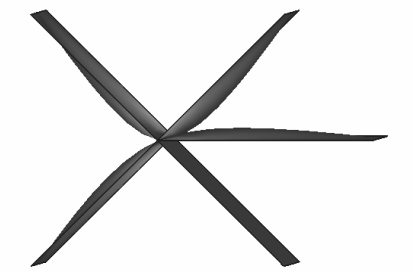
(f) illustration of example II: one MESHJOINT connecting four MESHSURFACEs in order – M1, M2, M3, M4



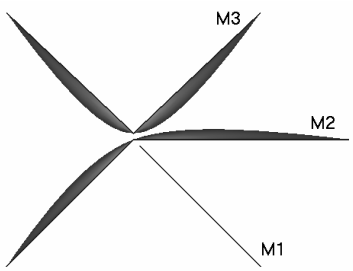
(g) example III



(h) illustration of example III: one MESHJOINT connecting three MESHSURFACEs in order – M1, M2, M3

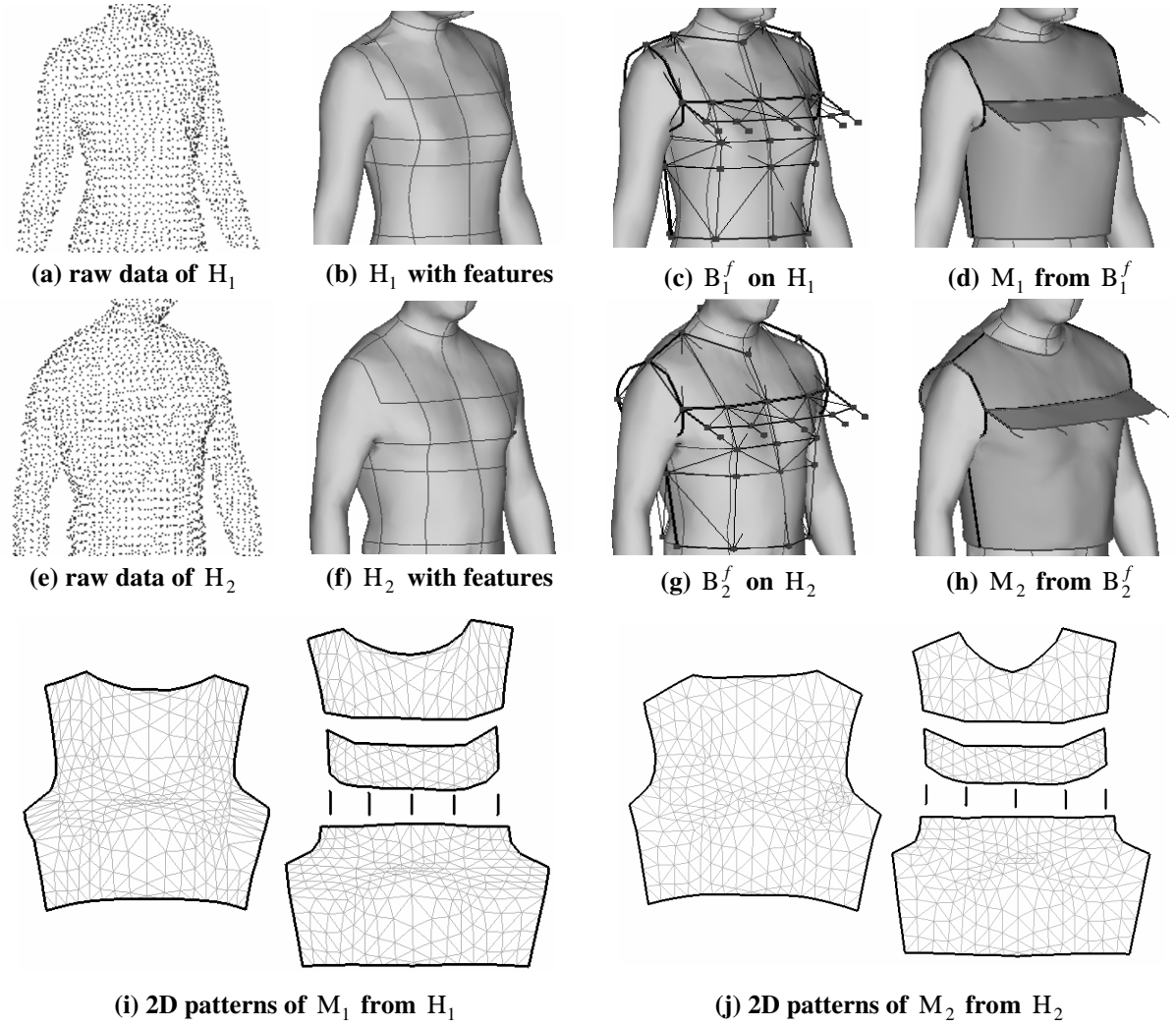


(i) example IV



(h) illustration of example IV: two MESHJOINTs defined – one connecting M1 and M2, and another connecting M2 and M3

Fig. 17 The flexibility of modeling different shapes around a MESHJOINT



(a) raw data of H_1 **(b) H_1 with features** **(c) B_1^f on H_1** **(d) M_1 from B_1^f**

(e) raw data of H_2 **(f) H_2 with features** **(g) B_2^f on H_2** **(h) M_2 from B_2^f**

(i) 2D patterns of M_1 from H_1 **(j) 2D patterns of M_2 from H_2**

Fig. 18 Application in the apparel industry

The technique presented in this paper can be widely used in many industrial applications (e.g., the apparel industry, the toy industry and the shoe industry). For example, in the apparel industry, after using the technique presented in [34] to generate the feature human model H_1 by the raw data that is output from a laser scanner (Fig. 18a and 18b), the approach presented in this paper is applied to build the feature template B_1^f of a piece of cloth on H_1 (Fig. 18c). After the final shape M_1 of B_1^f is constructed (Fig. 18d), it can be flattened into 2D patterns for manufacturing (Fig. 18i). When we regenerate the piece of cloth on another human model H_2 (Fig. 18e and 18f), after computing the new position of the feature nodes and 3D profiles in B_1^f, the new shape of the feature template B_2^f is determined (Fig. 18g). In the same way, we can construct its 3D final shape, M_2 (Fig. 18h) and its related 2D patterns (Fig. 18j). Obviously, the technique presented in this paper is very helpful to any industrial applications that attempt to automatically regenerate the 3D designed freeform objects on the

reference models with the same features but different shapes. In other words, to implement the automatic customization.

6. Conclusion

This paper presents a novel feature-based approach for modeling a 3D non-manifold freeform model around a 3D reference model. A new topological data structure is first defined with a set of operators. Based on the data structure and the operators, we developed the technique to construct 3D freeform objects around a reference model. Finally, an application of our technique in the apparel industry is given to demonstrate the functionality of our approach. In summary, our method has the following advantages:

- A new modeling method for 3D freeform objects is developed – the method is feature based, so the surfaces derived from one reference model can be regenerated automatically on another reference model with the same features;
- The geometry coverage of our modeling approach includes both manifold and non-manifold objects;
- Our non-manifold data structure embodies the functional vitalities of both the boundary data structure and the complex-based data structure, so incomplete topological information and the design abstractions can be easily stored; along with the data structure, a set of topological operators for non-manifold triangular mesh is developed;
- Intuitive 2D sketches are conducted to specify the detail shape of the constructed surfaces on the feature template of the designed 3D object.

While constructing the interpolating surface from the object template with the profiles specified, our current implementation conducts the uniform subdivision scheme, which leads to the non-uniform distribution of triangle density on the mesh surface. It is believed that an adaptive subdivision scheme can improve the non-uniform distribution of triangles. The fairing operator used in this paper can be applied only to two-manifold mesh surfaces. One possible further research is to consider a "volume-based" method to obtain a more flexible fairing result across the joints of mesh surfaces.

References

[1] Weiler K. (1986) The radial edge structure: a topological representation for non-manifold geometric boundary modeling, edited by Wozny M. J., McLaughlin H. W., and Encarnacao J. L., Geometric modeling for CAD applications, North-Holland, pp.3-36.

[2] Choi Y. (1989) Vertex-based boundary representation of non-manifold geometric models, PhD. Thesis, Carnegie Mellon University.

[3] Gursoz E. L., Choi Y., and Prinz F. B. (1990) Vertex-based boundary representation of non-manifold boundaries, edited by Wozny M. J., Turner J. U., and Preiss K., Geometric Modeling for Product Engineering, North-Holland, pp.107-130.

[4] Rossignac J., and O'Conner M. A. (1990) SGC: a dimensional –independent model for pointsets with internal structures and incomplete boundaries, edited by Wozny M. J., Turner J. U., and Preiss K., Geometric Modeling for Product Engineering, North-Holland, pp.145-180.

[5] Lienhardt P. (1991) Topological models for boundary representation: a comparison with n-dimensional generalized maps, Computer-Aided Design, vol.23, no.1, pp.59-82.

[6] Hubeli A., and Gross M. (2001) Multiresolution methods for nonmanifold models, IEEE Transactions on Visualization and Computer Graphics, vol.7, no.3, pp.207-221.

[7] Lee S. H., and Lee K. (2001) Partial entity structure: a compact non-manifold boundary representation based on partial topological entities, Proceedings of the 6th ACM Symposium on Solid Modeling and Applications, Ann Arbor, Michigan, pp.159-170.

[8] Heisserman J. A. (1991) A generalized Euler-Poincare Euation, Proceedings of the 1st ACM Symposium on Solid Modeling Foundations and CAD/CAM Applications, Austin, Texas, pp.533.

[9] Higashi M. (1993) High-quality solid modelling system with free-form surfaces, Computer-Aided Design, Vol. 25, No. 3, pp.172-183.

[10] Masuda H. (1993) Topological operators and Boolean operations for complex-based non-manifold geometric models, Computer-Aided Design, vol.25, no.2.

[11] Luo Y., and Gabor L. (1991) A boundary representation for form features and non-manifold solid objects", Proceedings of the 1st ACM Symposium on Solid Modeling Foundations and CAD/CAM Applications, Austin, Texas, pp.45-60.

[12] Meyers D., Skinner S., and Sloan K. (1992) Surface from Contours, ACM Transaction on Graphics, vol. 11, no. 3, pp. 228-258.

[13] Hoppe H., DeRose T., Duchamp T., Halstead M., Jin H., McDonald J., Schweitzer J., and Stuetzle W. (1994) Piecewise smooth surface reconstruction, SIGGRAPH 94 Conference Proceedings, pp.295-302.

[14] Igarashi T., Tanaka H., and Matsuoka S. (1999) Teddy: a sketching interface for 3D freeform design, SIGGRAPH 1999 Conference Proceedings, pp.409-416.

[15] Suzuki H., Sakurai Y., Kanai T., and Kimura F. (2000) Interactive mesh dragging with an adaptive remeshing technique, The Visual Computer, vol.16, no.3-4, pp.159-76.

[16] Zorin D., Schroder P., and Sweldens W. (1997) Interactive multiresolution mesh editing, SIGGRAPH 97 Conference Proceedings, pp.256-268.

[17] Khodakovsky A. and Schroder P. (1999) Fine level feature editing for subdivision surfaces, Proceedings ACM 5th Symposium on Solid Modeling and Applications, pp.203-211.

[18] Sederberg T. and Parry S. (1986) Free-form deformations of solid geometric models, Computer Graphics, vol.20, pp.151-160.

[19] Coquillart S. (1990) Extended free-form deformations: a sculpting tool for 3D geometric modeling, Computer Graphics, vol.24, no.4, pp.187-196.

[20] Chang Y. K. and Rockwood A. P. (1994) A generalized de Casteljau approach to 3D free-form deformation, Computer Graphics, vol.28, no.4, pp.257-260.

[21] Hsu W., Hughes J., and Kaufmann H. (1992) Direct manipulations of free-form deformations, Computer Graphics, vol.26, no.2, pp.177-184.

[22] MacCracken R. and Joy K. (1996) Free-form deformations with lattices of arbitrary topology, Computer Graphics, pp.181-189.

[23] Lazarus F., Coquillart S., and Jancene P. (1994) Axial deformations: an intuitive deformation technique, Computer-Aided Design, vol.26, no.8, pp.607-613.

[24] Singh K. and Fiume E. (1998) Wires: a geometric deformation technique, SIGGRAPH 98 Conference Proceedings, pp.405-414.

[25] Gindy N. N. Z. (1989) A hierarchical structure for form feature, International Journal of Production Research, vol.27, pp.2089-2103.

[26] Cavendish J. C. (1995) Integrating feature-based surface design with free-form deformation, Computer-Aided Design, vol.27, no.9, pp.703-711.

[27] Laakko T., and Mantyla M. (1996) Incremental constraint modelling in a feature modelling system, Computer Graphics Forum, vol.15, no.3, pp367-376.

[28] Takahashi S., Shinagawa Y., and Kunii T. L. (1997) A feature-based approach for smooth surface, ACM Symposium on Solid Modeling '97, Atlanta G.A., pp.97-110.

[29] Van Elsas P. A., and Vergeest J. S. M. (1998) Displacement feature modelling for conceptual design, Computer-Aided Design, vol.30, no.1, pp.19-27.

[30] Hoffmann C. H., and Joan-Arinyo R. (1998) On user-defined features, Computer-Aided Design, vol.30, no.5, pp.321-332.

[31] Gao S., and Shah J. J. (1998) Automatic recognition features based on minimal condition subgraph, Computer-Aided Design, vol30, no.9, pp727-739.

[32] Au C. K., and Yuen M. M. F. (1999) Feature-based reserve engineering of mannequin for garment design, Computer Aided Design, vo.31, no.12, pp.751-759.

[33] Au C. K., and Yuen M. M. F. (2000) A semantic feature language for sculptured object modeling, Computer Aided Design, vo.32, no.1, pp.63-74.

[34] Wang C. C. L., Chang T. K. K., and Yuen M. M. F. (2003) From laser-scanned data to feature human model: a system based on fuzzy logic concept, Computer-Aided Design, vol.35, no.3, pp.241-253, 2003.

[35] Levin A. (1999) Combined subdivision schemes for the design of surfaces satisfying boundary conditions, Computer Aided Geometry Design, vol. 16, no. 5, pp. 345-354.

[36] Levin A. (1999) Interpolating nets of curves by smooth subdivision surfaces", SIGGRAPH 99 Proceeding, ACM., pp.57-64, New York, USA.

[37] Kobbelt L. (2000) Discrete fairing and variational subdivision for freeform surface design, Visual Computer, vol. 16, no. 3/4, pp. 142-158.

[38] Zhang D. L. (2001) Efficient algorithms for cloth simulation, Ph.D. Thesis of the Hong Kong University of Science and Technology.

[39] Hoppe H., DeRose T., Duchamp T., McDonald J., and Stuetzle W. (1993) Mesh optimization, SIGGRAPH 1993 Conference Proceedings, pp.19-26.

[40] Loop C. (1987) Smooth subdivision surfaces based on triangles, Master thesis, Department of Mathematics, University of Utah.

[41] Wang C. C. L., and Yuen M. M. F. (2003) Freeform extrusion by sketched input, Computers & Graphics, vol.27, no.2.

[42] Piegl L., and Tiller W. (1997) The NURBS Book (2nd ed.), Berlin; Hong Kong: Springer.

Appendix I Entities

The data structure of entities is written in pseudo code as follows.

- Whole body

```
BODY {
        Int             indexNo;
        Double          box[6];         // Bounding box of this BODY
        FLAGS           flg;            // Status flags
```

```
        MESHSURFACE     **meshsurface_list;     // MESHSURAFCEs list
        MESHJOINT       **meshjoint_list;       // MESHJOINT list
};
```

- Mesh surface

```
MESHSURFACE {
        Int             indexNo;
        Double          box[6];                 // Bounding box of this mesh surface
        FLAGS           flg;                    // Status flags
        MESHLOOP        **meshloop_list;        // MESHLOOPs list
        MESHEDGE        **meshedge_list;        // MESHEDGEs list
        TRGLFACE        **trglface_list;        // TRGLFACEs list
        TRGLEDGE        **trgledge_list;        // TRGLEDGEs list
        TRGLNODE        **trglnode_list;        // TRGLNODEs list
        ATTRIB_EDGE     **attr_edge_list;       // ATTRIB_EDGEs list
};
```

- Joint of mesh surfaces

```
MESHJOINT {
        Int             indexNo;
        FLAGS           flg;                    // Status flags
        FLAGS           *dirFlg;                // Direction flags of MESHEDGEs
        MESHEDGE        **meshedge_list;        // MESHEDGEs (clockwise order)
};
```

- Loop of mesh surface

```
MESHLOOP {
        Int             indexNo;
        FLAGS           flg;                    // Status flags
        MESHEDGE        **meshedge_list;        // MESHEDGEs list
        MESHSURFACE     *mesh_surface;          // MESHSURFACE contains this loop
};
```

- Edge of mesh surface

```
MESHEDGE {
        Int             indexNo;
        FLAGS           flg;                    // Status flags
        FLAGS           *dirFlag;               // Direction flags of TRGLEDGEs
        TRGLEDGE        **trgledge_list;        // TRGLEDGEs list
        MESHSURFACE     *mesh_surface;          // Surface contains this mesh edge
        MESHLOOP        *mesh_loop;             // Loop contains this mesh edge
        MESHJOINT       *mesh_joint;            // Joint contains this mesh edge
};
```

- Triangular Face

```
TRGLFACE {
        Int             indexNo;
        FLAGS           flg;                    // Status flags
        FLAGS           dirFlg[3];              // Direction flags of edges
        Float           abcd[4];                // Plane equation of this triangle
        TRGLEDGE        *trgl_edge[3];          // Edges of this triangle
        MESHSURFACE     *mesh_surface;          // Surface contains this triangle
        ATTRIB_FACENODE **attr_node;            // ATTRIB_FACENODEs list
};
```

- Edge of triangular face

```
TRGLEDGE {
        Int                 indexNo;
        FLAGS               flg;                // Status flags
        TRGLNODE            *trgl_node[2];      // Nodes of this edge
        MESHSURFACE         *mesh_surface;      // Surface contain this edge
        TRGLFACE            *trgl_face[2];      // Left and right TRGLFACEs
        ATTRIB_EDGENODE     **attr_node;        // ATTRIB_EDGENODEs list
};
```

- Node of triangular face

```
TRGLNODE {
        Int                 indexNo;
        FLAGS               flg;                // Status flags
        Double              p_3d[3];            // Spatial position
        Double              p_3d_backup[3];     // Backup of spatial position
        Double              p_2d[3];            // Planar position
        MESHSURFACE         *mesh_surface;      // Surface contain this node
        TRGLFACE            **trglface_list;    // Adjacent TRGLFACEs list
        TRGLEDGE            **trgledge_list;    // Adjacent TRGLNODEs list
};
```

Appendix II Attributes

The data structure of attributes is written in pseudo code as follows.

- ATTRIB_NODE

```
ATTRIB_NODE {
        FLAGS               flg;                // Status flags
        ATTRIB_EDGE         *attr_edge;         // ATTRIB_EDGEs contain this node
};
```

- ATTRIB_EDGE

```
ATTRIB_EDGE {
        FLAGS               flg;                // Status flags
        MESHSURFACE         *mesh_surface;      // Surface contain this edge
        ATTRIB_NODE         **attr_node;        // Pointer of ATTRIB_NODEs list
};
```

- ATTRIB_EDGENODE

```
ATTRIB_EDGENODE : public ATTRIB_NODE {
        Double              u;                  // Parameter coordinate of this node
        Double              p_3d[3];            // Spatial position
        TRGLEDGE            *trgl_edge;         // TRGLEDGE contain this node
};
```

- ATTRIB_FACENODE

```
ATTRIB_FACENODE : public ATTRIB_NODE {
        Double              u, v, w;            // Areal coordinate of this node
        Double              p_3d[3];            // Spatial position
        TRGLFACE            *trgl_face;         // TRGLFACE contain this node
};
```

Appendix III Extended Euler Operators

The eight extended Euler operators are listed in Table A-1 and Fig. A-1. Reverse operators are enclosed in brackets, and *Chole* denotes a hole in a complex.

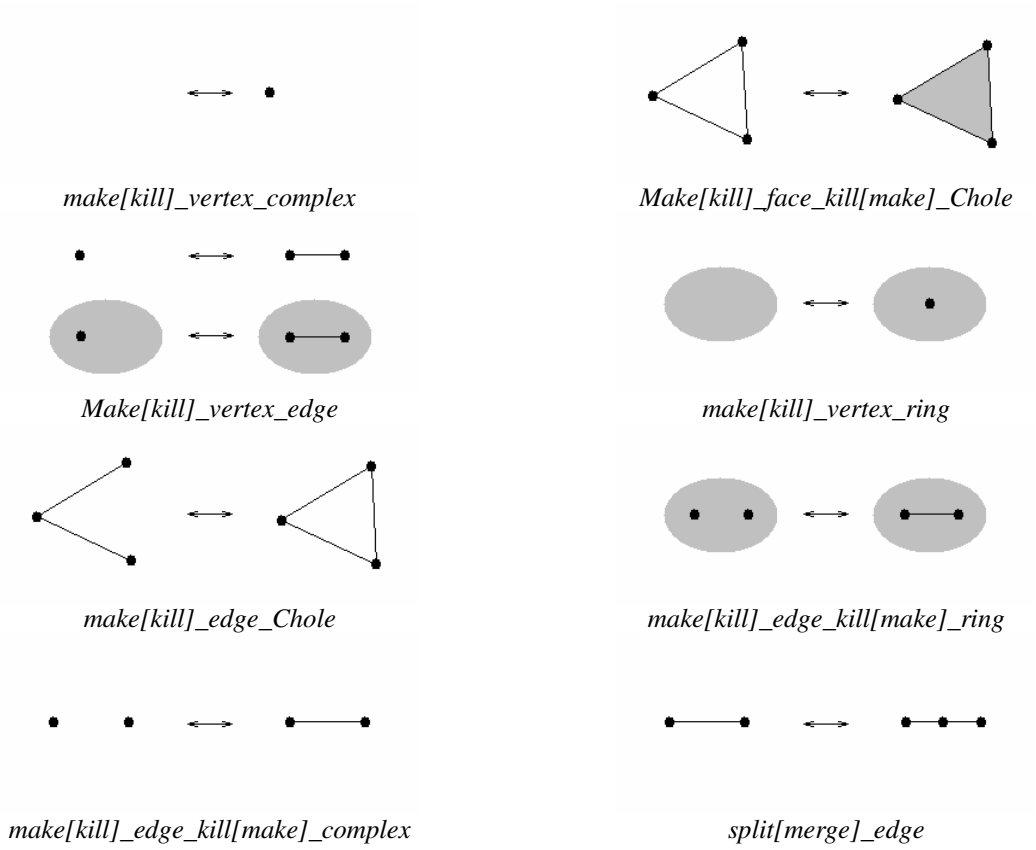


Fig. A-1 Extended Euler operators

Table A-1 Function Description of Extended Euler Operators

Operator	**Function of operator**
make[kill]_vertex_complex	Create a single vertex complex
make[kill]_vertex_edge	Create a vertex and an edge connecting to an existed vertex (the operator can be carried out freely or inside a face)
make[kill]_edge_Chole	Connect two vertex by a new edge to form a hole in a complex
make[kill]_edge_kill[make]_complex	Connect two vertex complexes by a new edge (since the two complexes are connected, one complex should be removed)
make[kill]_face_kill[make]_Chole	Create a new face on a complex hole and remove the hole
make[kill]_vertex_ring	Create a stand-alone vertex on a face (the single vertex forms a ring)
make[kill]_edge_kill[make]_ring	Connect two stand-alone vertices by a new edge on a face (two rings are merged into one ring, thus one ring should be removed)
Split[merge]_edge	Split one edge into two edges by adding a new vertex on the original edge