International Journal of CAD/CAM Vol. 3, No. 2, pp. 77~83 (2003)

INTERNATIONAL JOURNAL OF CAD/CAM
www.ijcc.org

Generalized Cylinder based on Linear Interpolation by Direction Map

Joo-Haeng Lee*, Hyun Kim and Hyoung-Sun Kim
Intelligent Robot Research Team, ETRI, Daejeon, Korea

Abstract – We propose two algorithms to generate (1) polygonal meshes and (2) developable surface patches for generalized cylinders defined by contours of discrete curves. To solve the contour blending problem of generalized cylinder, the presented algorithms have adopted the algorithm and related properties of LIDM (linear interpolation by direction map) that interpolate geometric shapes based on direction map merging and group scaling operations. Proposed methods are fast to compute and easy to implement.

Keywords: **Generalized cylinder, Linear interpolation by direction map, Developable surface**

1. Introduction

Generalized cylinder (GC) is a well-known modeling technique to design tube-like shapes whose surfaces are constructed over skeletal frames composed of a finite sequence of contours (2D cross-sectional curves) that are systematically arranged on a 3D spine curve. Generally, the spine curve determines the overall shape, and contours expresses detailed features on the surface. By interpolating (or blending) the given contours, we can generate a one-parameter family of contours which conceptually sweeps along the spine curve while changing its orientation and shape. Using this general sweep analogy, we can represent a GC as a tensor product surface with two parameters representing the spine and the contour directions, respectively. After generating the surface from the initial design, we can interact with the skeletal curves or the surface itself to change the shape characteristics. With these simple building blocks and mechanisms coupled with other geometric modeling techniques, we can design various kinds of artificial shapes (i.e., pipes, vessels, and tires) and natural shapes (i.e., human bodies, flowers, and seashells) as GC models for CAD and computer graphics applications.

Previous researches have focused on various GC topics such as surface representations, deformation and interaction techniques, and orientation arrangements. Note that, to be integrated with general-purpose geometric modeling tools, it is a common practice to generate surface models directly from the intrinsic definition of GC: i.e., a spine and contour curves. Hence, there are abundant research works presenting how to represent GC surfaces as a polyhedral mesh [1], Bézier [6], B-spline [14], or NURBS surfaces [4] from the given skeletal frames. Some representation methods focus on the direct ray-casting [14] without converting into specific representations. We can find researches presenting special representations suitable for interactive deformation [4, 6]. Contour arrangement with smooth orientation change is simply achieved by embedding a contour on the normal plane of Frenet frame [3, 4, 6, 14]. For better results, rotation minimizing frame can optimizes distortion [7]. As a more sophisticated topic, Gansca *et al.* deals with the problem of self-intersection avoidance in the generation of GC surfaces [5].

Another important GC topic is contour blending, which is required to generate a one-parameter family of contours continuously. One of the fundamental steps for contour blending is to set up correspondences between features of neighboring contours. However, in most of the previous approaches, these correspondences are assumed to be described manually or implicitly. For example, although very complicated contours were illustrated in B-Spline surface approach of de Voogt *et al.* [14], no explicit step is specified for setting correspondences between every pair of control points from adjacent contours. This is partly because every contour has the same number of control points, which may lead to trivial correspondences in a certain case. However, this is not the case of real-world examples where correspondences are rather complicated to be described manually or assumed implicitly.

This correspondence problem is also fundamental in morphing. (Note that morphing is composed of two steps: (1) correspondences and (2) path interpolation.) When key-frames are not so complex, existing geometric morphing techniques works fine. However, we can hardly expect full automation: for better result, a human intervention is inevitable. For example, when we want to generate in-betweens interpolating given key-frames

*Corresponding author:
Tel: +82-42-860-1338
Fax: +82-42-860-6790
Homepage: http://joohaeng.etri.re.kr
E-mail: joohaeng@etri.re.kr

representing different postures of a dancer [13], we could hardly expect one of dancer's arms is never corresponded to one of legs based on geometric intelligence of the previous algorithms. In addition, most of morphing techniques cannot express interpolating path in the form of parametric curves. If we consider contours as key-frames, parametric form of moving path is critical in representing a GC surface.

In this paper, we suggest to adopt a geometric morphing technique referred to as LIDM (linear interpolation by direction map) proposed by Lee *et al.* [10] to solve both correspondence and parametric interpolation problems in contour blending. LIDM is closely related to previous morphing technique referred to as LIMS (linear interpolation by Minkowski sum) [9, 12], but more generalized and computationally efficient.

In addition, we present methods to construct GC surface with (1) polygonal mesh and (2) developable surface patches using the geometric properties of LIDM [8]. The overall computation of proposed methods is fast enough to be applied in interactive geometric design applications.

The remainder of this paper is organized as follows. In Section 2, we describe a typical representation of parametric GC surface and explain why contour blending problem is important in GC. In Section 3, we propose to adopt LIDM as a contour blending method in GC design. In Section 4, we describe how to build a polygonal mesh of GC whose contours are blended by LIDM. In Section 5, we describe how to generate developable surface patches representing GC. In Section 6, we demonstrate the example results.

2. Parametric Representation of GC

For a parameterized spine curve $K(u)$ in 3D, we can choose the normal plane $N(u_0)$ at $K(u_0)$ as the contour plane, which is intrinsically defined by Frenet frame [3]. (As in Chang *et al.* [4], we can define the orientation more generally – independently of differential characteristics of the given spine curve.) In this case, $N(u_0)$ is spanned by its normal and binormal vectors in 3D, $n_x(u_0)$ and $n_y(u_0)$, and its local origin is placed at $K(u_0)$. When a 2D contour curve $C_{u_0}(v)=(x_{u_0}(v), y_{u_0}(v))$ is embedded in $N(u_0)$, it has the following parametric form:

$$C_{u_0}(v)=x_{u_0}(v)\cdot n_x(u_0)+y_{u_0}(v)\cdot n_y(u_0) \tag{1}$$

However, considering that every contour $C_u(v)$ at $K(u)$ may have a different shape, above parametric form should be further generalized as follows:

$$\begin{aligned} C_u(v) &= x_u(v)\cdot n_x(u)+y_u(v)\cdot n_y(u) \\ &\equiv C(u,v)=x(u,v)\cdot n_x(u)+y(u,v)\cdot n_y(u) \end{aligned} \tag{2}$$

When we consider GC as a sweep surface of a moving contour, the parametric form of GC surface S is generated by sweeping $C_u(v)$ along $K(u)$ as follows:

$$\begin{aligned} S=S(u,v) &= C(u,v)+K(u) \\ &= x(u,v)\cdot n_x(u)+y(u,v)\cdot n_y(u)+K(u) \end{aligned} \tag{3}$$

In the above equation, we assume that a pair of coordinate functions $(x_u(v), y_u(v))$ is defined at every point $K(u)$ of the spine curve; however, a human designer cannot specify infinite number of coordinate functions manually at each value of u. This is the point where contour blending problem arises: how do we smoothly interpolate a finite set of key-frame contours to generate a certain number of in-between contours required to satisfy the given precision criteria.

In this paper, to be more focused on the blending problem itself, we can confine the bases of the contour plane to be fixed over u. In this case, a GC surface has a following parametric form:

$$S(u,v)=C(u,v)+K(u)=x(v)\cdot n_x+y(v)\cdot n_y+K(u) \tag{4}$$

This is the typical case when the spine curve is a straight-line segment. The examples presented in this paper are of this type.

3. Contour Blending by LIDM

Recently, Lee *et al.* [10] proposed an efficient algorithm referred to as LIDM (linear interpolation by direction map) to interpolate two polygonal shapes. Moreover, a designer can specify additional control shapes, which enables a Bézier-curve (or blossom) like control structure. The result can be represented as a parametric form of a one-parameter family of polygons. Specially, the automatic correspondences work quite well for relatively simple shapes rather than the complex ones of character animations. Hence, we propose to adopt LIDM for contour blending in GC design.

In LIDM, a polygon is represented by a circular list of direction vectors, which is referred to as a direction map. A direction vector is defined as a connecting vector of two neighboring polygon vertices. (See Fig. 1. The number describes the correspondence between a direction vector and an edge.) A group of consecutive direction vectors may represent a geometric feature such as a pocket. We assume that, in LIDM, the direction itself of any direction vector is invariant. Hence, (1) the sequence of directions and (2) lengths of individual direction vectors are deciding factors of a shape feature.

We can generate a new polygon by merging two direction maps, each of which represents different shapes. This step corresponds to blending features from

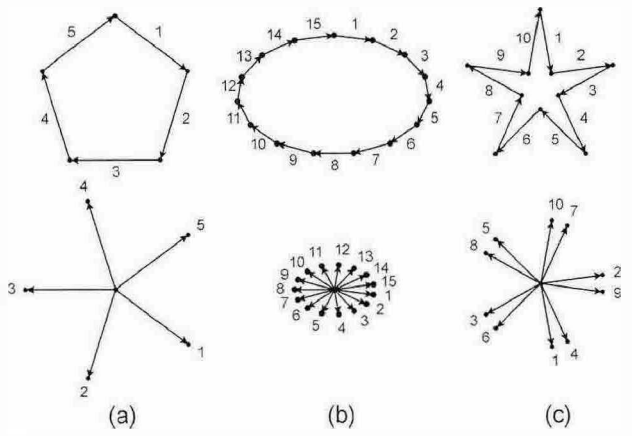


Fig. 1. Shapes (upper row) and their direction maps (lower row): (a) pentagon, (b) discrete oval, and (c) 5-star.

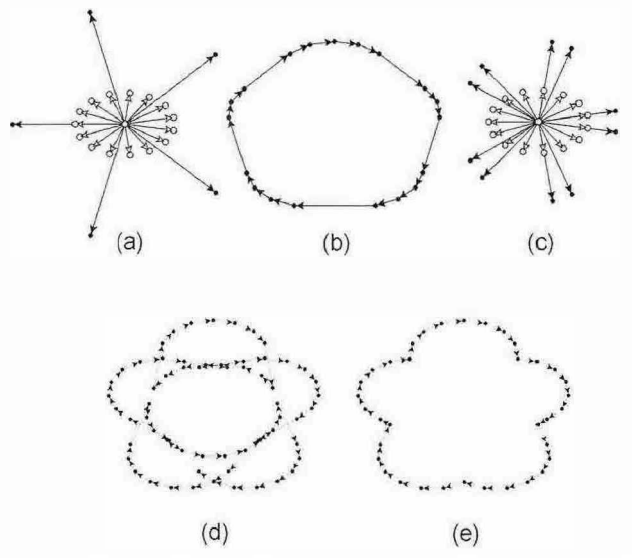


Fig. 2. Convolution merging: (a) the merged direction map of Fig. 1(a) and (b), (b) newly generated shape from (a), (c) the merged direction map of Fig. 1(b) and (c), (d) newly generated shape with self-intersections, and (e) trimmed result of (d).

different polygons. In merging operation, we change neither the direction nor the length of any direction vector. Only a new merged sequence is generated by applying a certain geometric correspondence rule. Among various feature correspondence strategies, convolution merging (see Fig. 2) is closely related to Minkowski sum or convolution operations, where vertex-wise feature correspondences are set up by geometric rules [10]. Note that LIMS (linear interpolation by Minkowski sum) is a special case of LIDM [9, 12]. Another merging method is convex-hull merging (see Fig. 3) where no self-intersection occurs; hence, trimming is not required. For convex shapes, the convolution and convex-hull merging generate the same result.

To generate a one-parameter family of in-between

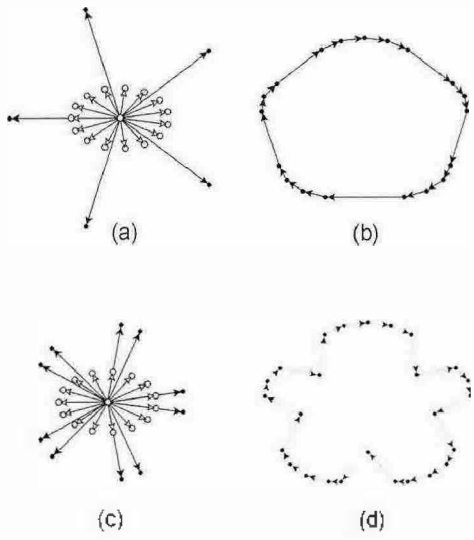


Fig. 3. Convex-hull merging: (a) the merged direction map of Fig. 1(a) and (b), (b) newly generated shape from (a), (c) the merged direction map of Fig. 1(b) and (c), (d) newly generated shape without self-intersections.

shapes, we have to smoothly change the degree of dominating feature. In LIDM, this is accomplished by changing the length of direction vectors using group scaling operation where different values of scaling factors are assigned to each group of direction vectors. (Note that each group is identified by a *group id*, which is assigned to every direction vector in a merged direction map.) The examples of scaling factors can be Bézier or blossom basis functions. Using both merging and group scaling operations, Lee *et al.* proposed interpolation algorithm as follows (for details, we refer readers to [10]):

Algorithm 1: LIDM
Input: A merged direction map: $D \leftarrow D_0 + \cdots + D_m$; and
Scalar functions for the group scaling operation:
$\mathbf{B} = \{b_0(t), \ldots, b_m(t)\}$; and
The blending parameter : t.
Output: A contour generated by LIDM algorithm:
$C = C(t)$.

Fig. 4 is an example output of LIDM algorithm: (a) two input direction maps represent a triangle and an octagon; (b) by group scaling operations (in this example, with linear scaling functions), the dominating direction vectors become longer and the others shorter; however, (c) after the lengths are normalized into one, we find that the directions are invariant over t. Fig. 5 shows a result for the four input direction maps: the initial circle becomes longer in width and height in order according to the sequence of control polygons; and comes back to a circle satisfying the final key-frame. In Fig. 5, we used cubic Bernstein polynomials as blending functions.

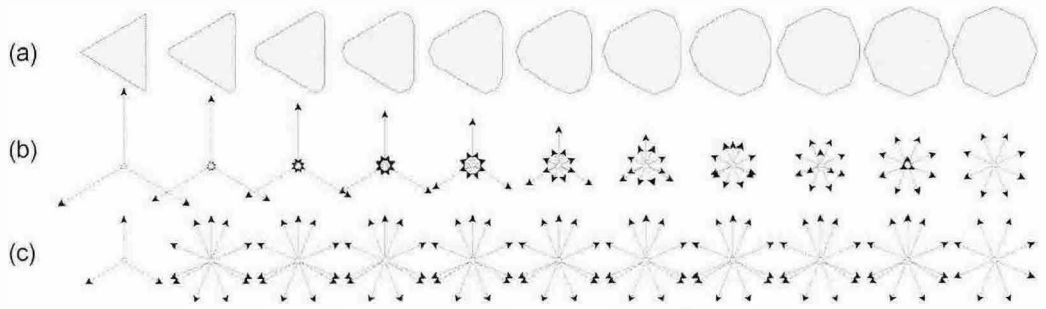


Fig. 4. (a) Smoothly changing polygons, (b) their direction maps, and (c) their normalized direction maps.

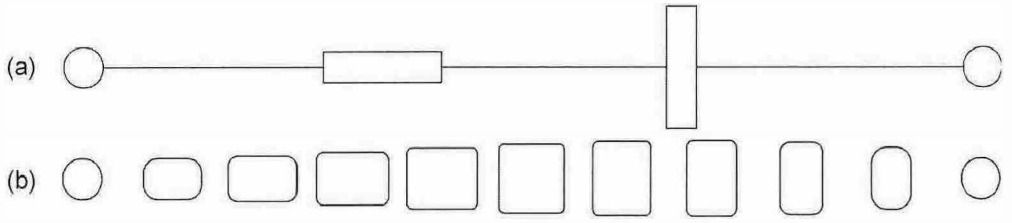


Fig. 5. (a) Control polygons: terminal polygons (both circles) and additional polygons (two quadrangle), (b) the generated sequence.

4. GC in Polygonal Mesh

In this section, we describe how to build a polygonal mesh of GC whose contours are blended by LIDM. For in-between shapes generated by LIDM, there exist two interesting geometric properties [8]: (1) all the shapes have the same normalized direction maps over u; and (2) the number of vertices (or edges) of generated polygons is constant over u. Two properties hold unless some edges vanish or shrink after trimming. Based on these properties, it is straightforward to build triangular (or quad) meshes by connecting corresponding vertices from neighboring contours. (See Algorithm 2 below.)

Algorithm 2: LIDM_GC_POLY_MESH
Input: Two terminal contours and additional control contours: $\mathbf{C}=\{C_0, \ldots, C_m\}$; and
Scalar functions: $\mathbf{B}=\{b_0(t), \ldots, b_m(t)\}$; and
/* i.e., Bernstein polynomials */
The number of in-between contours including two terminals: $(n+1)$.
Output: A Polygonal mesh representing a GC surface: $\mathbf{M}$.

1. $D \leftarrow D_0+\cdots+D_m$;
   /* merge direction maps: $D_i=\mathrm{DM}(C_i)$ */
2. $l \leftarrow |D|$;
   /* l: the number of direction vectors $\{\vec{d}_i\}$ in D*/
3. $C_{prev} \leftarrow \mathrm{LIDM}(D, \mathbf{B}, 0)$;
   /* the initial contour $C(0)$ */
4. **For** $i=1$ **to** n
5. $t \leftarrow t_i = \Delta t * i$; /* $\Delta t = 1/n$ */
6. $C_{curr} \leftarrow C(t) = \mathrm{LIDM}(D, \mathbf{B}, t)$;
   /* Assuming $|\mathrm{DM}(C(t))|$ is invariant. */
7. $\mathbf{M}_i \leftarrow \phi$;
8. **For** $j=1$ **to** l
   /* Construct a sub-mesh $\mathbf{M}_i$ connecting C_{prev} and C_{next} */
9. $M_j \leftarrow$ two triangle (or a quadrangle) constructed using the corresponding 4 vertices: j-th and $(j+1)$-th vertices of C_{prev} and C_{next};
10. $\mathbf{M}_i \leftarrow \mathbf{M}_i \cup M_j$;
11. $\mathbf{M} \leftarrow \mathbf{M} \cup \mathbf{M}_i$;
12. $C_{prev} \leftarrow C_{next}$;

In the above algorithm, $\mathrm{DM}(C_i)$ represents the construction operation of a direction map for the contour C_i. When the above algorithm generates one intermediate contour $C(t_i)$, it evaluates one LIDM operation for each parameter value $t=t_i$. Note that, however, direction map merging is computed just once

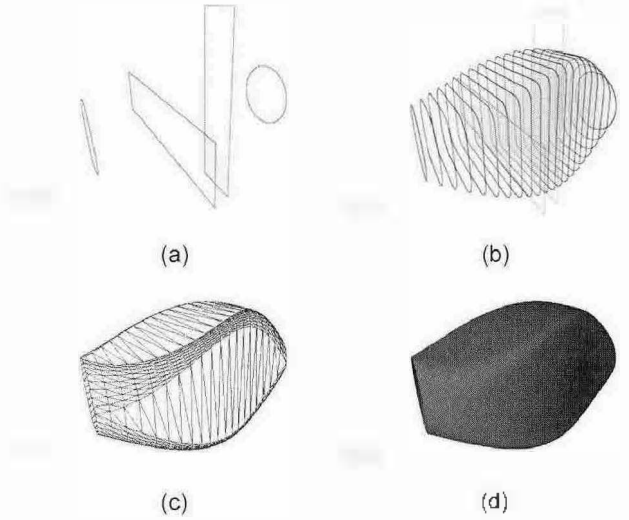


Fig. 6. Build a polygonal mesh of GC: (a) control polygons (the same as in Fig. 5) are arranged on a straight line, (b) in-between contours generated by LIDM, (c) the generated polygonal mesh, and (d) shaded result of (c).

during the whole execution since the correspondences are assumed static. Each sub-mesh $\mathbf{M}_i$ is generated by connecting two consecutive contours. The edge connection rules are simple. For example, to generate two triangles: (1) connect two starting points, (2) two end points of corresponding edge, and (3) choose one diagonal. The final mesh $\mathbf{M}$ is generated by combining all the sub-meshes. Fig. 6 shows an example of execution steps of Algorithm 2.

5. GC in Developable Surface Patches

In this section, we describe how to generate developable surface patches representing GC. A developable surface is a special type of ruled surface, where all the points from one ruling have the same tangent plane [3]. Specially, a developable surface can be unfolded (developed) into a plane without stretching or tearing. Hence, it has a wide-range of applications in manufacturing based on sheet metal-like materials. The recent works shows that a developable surface has a nice structure of controllability [2] and a neat representation into NURBS [11].

If a ruling direction is fixed over the entire patch, it generates a cylindrical developable surface. In our application, a direction vector becomes a ruling whose direction is invariant. Hence, every direction vector (or an edge of a polygonal contour) corresponds to one developable surface patch bounded by two boundary curves. Moreover, boundary curves are defined by vertices of control contours.

Algorithm 3: LIDM_GC_DEV_SURF
Input: Two terminal contours and additional control contours: $\mathbf{C}=\{C_0, \ldots, C_m\}$; and
Output: l control points set for profile curves: $\mathbf{P}=\{\mathbf{P}_1, \ldots, \mathbf{P}_l\}$. /* $\mathbf{P}_i=\{P_{i,0}, \ldots, P_{i,m}\}$ */
1. $D \leftarrow D_0+\cdots+D_m$;
   /* merge directions maps: $D_i=\mathrm{DM}(A_i)$ */
2. $l \leftarrow |D|$;
   /* l: the number of direction vectors $\{\vec{d}_i\}$ in D */
3. **For** i=1 **to** l
   /* For each profile curve F_i defined by a direction vector $\vec{d}_i$ */
4. $\vec{d}_{curr} \leftarrow$ the i-th direction vector of D;
5. **For** j=0 **to** m
   /* Find the control point set $\mathbf{P}_i$ for F_i */
6. $\vec{d} \leftarrow$ find a direction vector satisfies two conditions: (1) the counter-clockwise-nearest direction vector from $\vec{d}_{curr}$; and (2) its group id is j;
7. $P_{i,j} \leftarrow$ the end point of $\vec{d}$;
8. $\mathbf{P} \leftarrow \mathbf{P} \cup \{\mathbf{P}_i\}$;

Algorithm 3 finds a group of control points representing every pair of boundary curves (referred to as *profile curves*, here) per a direction vector. The interesting property is that, if we apply Bernstein polynomials $B_i^m(t)$ of degree m as scaling factors for the group scaling operation, the developable surface is a Bézier surface of degree $(1, m)$. For example, i-th developable surface patch $S_i(u, v)$ defined by $2(m+1)$ control points (i.e., $\{P_{i,0}, \ldots, P_{i,m}\}$ and $\{P_{i+1,0}, \ldots, P_{i+1,m}\}$) found in Algorithm 3 has the following tensor product form:

$$S_i(u,v)=\sum_{k=i}^{i+1}\sum_{j=0}^{m} P_{k,j}B_{k-i}^{1}(u)B_j^m(v) \tag{5}$$

Actually, the type of two boundary curves in a single developable surface patch is defined by how we blend contours using a certain scaling factors.

Fig. 7 shows an example of execution steps of Algorithm 3. In Fig. 7 control contours in pink are the same as in Fig. 5 and 6: (a) three consecutive green line segments represents a control polyline for a profile curve; (b) a developable surface patch is defined by consecutive profiles curves (in light blue). Moreover, its ruling (a straight-line segment in light blue) is the morphing edge sweeps along profile curves; (c) all the control polylines found; (d) all the control polylines

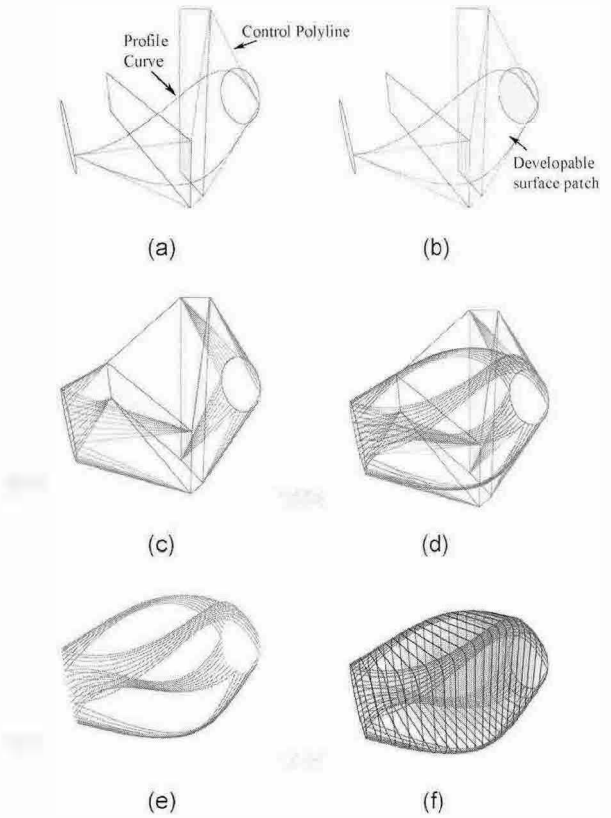


Fig. 7. Computing control points of developable surface patches: (a) two sets of control points and profile curves defining a developable surface patch, (b) a direction vector (i.e., a ruling) sweeps along profile curves, (c) all the control points for each profile curve, (d) parametric evaluation of profile curves, (e) evaluated profile curves, and (f) parametric evaluation of contours.

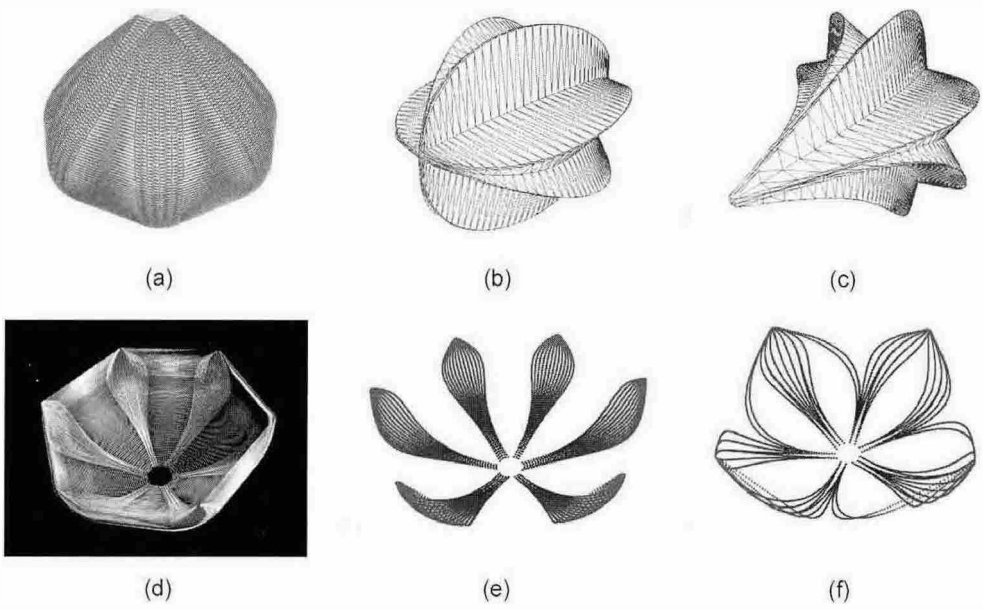


Fig. 8. Examples of GC generated using proposed algorithms.

and corresponding profile curves; and (e) all the profile curves. For display, we can adopt existing shading algorithms for Bézier surfaces. If we want to generate in-between contours, we can connect the points from every profile curves evaluated at the same value in sequential order, as in Fig. 7-(d). Note that, however, it is not easy to generate a parametric form of profile curves based on contour-wise evaluation as in Algorithm 2.

6. Result and Discussion

Algorithms 2 and 3 are of complexities $O(n \cdot l)$ and $O(l \cdot m)$, respectively. (n: the number of evaluated contours, l: the number of direction vectors, m: the number of control contours) The overall computation of proposed methods is fast enough to be implemented in interactive geometric design applications.

Fig. 8 illustrates surface representation of generalized cylinders defined by two cross-sectional polygons and some of additional control shapes arranged on a straight-line spine: (a) an onion-like shape in mesh representation, (b)-(c) more complex examples using non-convex control contours, (d) mesh representation of a bowl, and (e)-(g) design of flowers using profiles curves of developable surface patches. Note that (d) and (e) were modeled using the same contours.

Acknowledgements

This work has been supported in part by grant No. A1-03-0021-00 (Development on Collaborative Product Commerce Technologies) of Korean Ministry of Information and Communication.

References

[1] Akman, V. and Arslan, A. (1992), "Sweeping with all graphical ingredients in a topological picturebook," *Computer & Graphics*, **16**, 273-281.

[2] Bodduluri, R. M. C. and Ravani, B. (1993), "Design of developable surfaces using duality between plane and point geometries," *Computer-Aided Design*, **25**(10), 621-632.

[3] Carmo, M.P. do (1976), *Differential Geometry of Curves and Surfaces*, Prentice-Hall.

[4] Chang, T.-I., Lee, J.-H., Kim, M.-S. and Hong, S. J. (1998), "Direct manipulation of generalized cylinders based on B-spline motion," *The Visual Computer*, **14**, 228-239.

[5] Gansca, I., Bronsvoort, W. F., Coman, G. and Tambulea, L. (2002), "Self-intersection avoidance and integral properties of generalized cylinders," *Computer Aided Geometric Design*, **19**(9), 695-707.

[6] Kim, M.-S., Park, E.-J. and Lee, H.-Y. (1994), "Modelling and animation of generalized cylinders with variable radius offset space curves," *Journal of Visualization and Computer Animation*, **5**, 189-207.

[7] Klok, F. (1986), "Two moving coordinate frames for sweeping along a 3D trajectory," *Computer Aided Geometric Design*, **3**, 217-229.

[8] Lee, J.-H., (2003), Geometric Properties of Morphing based on Direction Map: (II) Characteristics of Intermediate Shapes, Preprint.

[9] Lee, J.-H., Lee, J. Y., Kim, H. and Kim, H.-S. (2003), "Interactive Control of Geometric Shape Morphing based on Minkowski Sum," *Transactions on SCCE*, **4**, 317-380.

[10] Lee, J.-H., Kim, H. and Kim, H.-S. (2003), "Efficient Computation and Control of Geometric Shape Morphing based on Direction Map," *Transactions on SCCE*, Accepted.

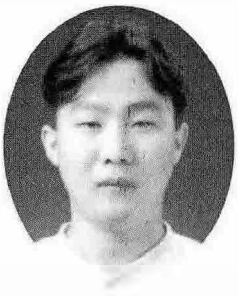
[11] Pottman, H. and Wallner, J. (1999), "Approximation algorithms for developable surfaces," *Computer Aided Geometric Design*, **16**, 539-556.

[12] Rossignac, J. and Kaul, A. (1994), AGRELs and BIPs: Metamorphosis as a Bezier curve in the space of Polyhedra, *EUROGRAPHICS '94*, C179-C184.
[13] Shapira, M. and Rappoport, A. (1995), "Shape blending using the star-skeleton representation," *IEEE Computer Graphics & Applications*, **15**(2), 44-50.
[14] Voogt, E. de, Helm, A. van der and Bronsvoort, W. F. (2000), "Ray tracing deformed generalized cylinders," *The Visual Computer*, **16**, 197-207.

Joo-Haeng Lee received his BS, MS, and PhD in Computer Science from POSTECH, Korea, in 1994, 1996 and 1999, respectively. He joined ETRI (Electronics and Telecommunications Research Institute), Korea in 1999, and is a senior member of research staff of Software Robot Research Team, ETRI. His research interests include geometric modeling and processing for various applications such as computer graphics, virtual reality, computer-aided design, mobile robots, and knowledge visualization.

Hyoung-Sun Kim received the M.S. degree in Computer Engineering from Kwangwoon University, Seoul, Korea, in 1991, and is a PhD student in Computer Engineering of Daejeon University, Korea, since 2001. He had worked for Systems Engineering Research Institute (SERI) from 1986 to 1998. He joined ETRI in 1998 and has worked for the software development related to Concurrent Engineering such as CPC. Currently, he is principal researcher in Software Robot Research Team, ETRI. His research interests include Concurrent Engineering, Collaborative Product Commerce, Distributed Computing, Information Security, and Distributed Database.

Hyun Kim received the B.S., M.S. and Ph.D. degrees in the department of mechanical design and manufacturing from Hanyang University, Seoul, Korea, in 1984, 1987 and 1997, respectively. He had worked for Systems Engineering Research Institute (SERI) from 1990 to 1998. He joined ETRI in 1998 and has worked for the software development related to engineering design such as CAD/CAM/CAE. Currently, he is a project leader in Software Robot Research Team, ETRI. His research interests include Concurrent Engineering, Web-enabled CAD, Virtual Engineering and Collaborative Product Commerce.



Joo-Haeng Lee

Hyun Kim

Hyoung-Sun Kim