Modeling Generalized Cylinders using Direction Map Representation

Joo-Haeng Lee

Electronics and Telecommunications Research Institute (ETRI), joohaeng@etri.re.kr

ABSTRACT

For generalized cylinders (GC) defined by contours of discrete curves, we propose two algorithms to generate GC surfaces (1) in polygonal meshes and (2) in cylindrical type of developable surface patches. To solve the contour blending problem of generalized cylinder, the presented algorithms have adopted the algorithms and properties of LIDM (linear interpolation by direction map) that interpolate geometric shapes based on direction map merging and group scaling operations. Moreover, we propose an algorithm to develop generated developable surface patches on a plane. Proposed algorithms are fast to compute and easy to implement.

Keywords: generalized cylinder, direction map representation, 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 curve and a sequence of contour curves. Hence, there are abundant research works presenting how to represent GC surfaces as a polyhedral mesh [1], Bezier [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 a smooth orientation change is simply achieved by embedding a contour on the normal plane [3] of the Frenet frame [4],[6],[14]. For better results, rotation minimizing frame can optimize 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 fundamental steps for contour blending is to set up correspondences between features of adjacent key-frame contours. However, in most of 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 generally 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 representing different postures of a dancer [12], we can not specify semantic constraint such as "an arm should not be morphed to a leg" based on geometric intelligence of previous algorithms. In addition, most of morphing techniques can not express interpolating path in the form of parametric curves. If we consider contours as key-frames, the parametric representation of interpolating path is critical to generate the surfaces of a GC.

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. [9] 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) [8],[11], but more generalized and computationally efficient.

In addition, we present methods to construct GC surface with (1) polygonal mesh and (2) cylindrical type of developable surface patches using the geometric properties of LIDM. The overall computation of proposed methods is fast enough to be applied in interactive geometric design applications. We also propose the algorithm to develop generated developable surface patches on a plane.

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 matters 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 explain how to develop generated developable surface patches on a plane. In Section 7, we demonstrate the example results. This template file contains all relevant information to process papers into the right format for the annual CAD conference proceedings. For ease of paper preparation, authors may want to cut and paste the relevant sections into this document.

2. PARAMETRIC REPRESENTATION OF GC

Let a contour curve be placed on a *contour plane*. For a spine curve parameterized as $\mathbf{K}(u) \in \mathbb{R}^3$, we can choose a contour plane $\mathbf{N}(u_0) \in \mathbb{R}^3$ as the normal plane at $\mathbf{K}(u_0)$, which is intrinsically defined as a part of the 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, $\mathbf{N}(u_0)$ is spanned by its normal and binormal vectors, $\mathbf{n}_x(u)$ and $\mathbf{n}_y(u)$, and its local origin is placed at $\mathbf{K}(u_0)$.

When a 2D contour curve $\bar{\mathbf{C}}_{u_0} = (x_{u_0}(v), y_{u_0}(v))$ is embedded in $\mathbf{N}(u_0)$, it has the following parametric form in 3D:

$$\mathbf{C}_{u_0}(v) = x_{u_0}(v) \cdot \mathbf{n}_x(u_0) + y_{u_0}(v) \cdot \mathbf{n}_y(u_0) \tag{1}$$

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

$$\mathbf{C}_u(v) = x_u(v) \times \mathbf{n}_x(u) + y_u(v) \times \mathbf{n}_y(u) \tag{2}$$

$$\circ \quad \mathbf{C}(u,v) \tag{3}$$

$$= x(u,v) \times \mathbf{n}_x(u) + y(u,v) \times \mathbf{n}_y(u). \tag{4}$$

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

$$\begin{aligned} \mathbf{S}(u,v) &= \mathbf{C}(u,v) + \mathbf{K}(u) \\ &= x_u(v) \times \mathbf{n}_x(u) + y_u(v) \times \mathbf{n}_y(u) + \mathbf{K}(u). \end{aligned} \tag{5}$$

In the above equation, we assume that a pair of coordinate functions $(x_u(v), y_u(v))$ is defined at every point $\mathbf{K}(u)$ of the spine curve; however, a human designer could not 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 simple parametric form as follows:

$$\mathbf{S}(u,v) = x_u(v) \times \mathbf{n}_x + y_u(v) \times \mathbf{n}_y + \mathbf{K}(u). \quad (6)$$

This is the typical case when the spine curve is a straight line segment. Specially, the developable surface examples in Section 5 and 6 are of this type.

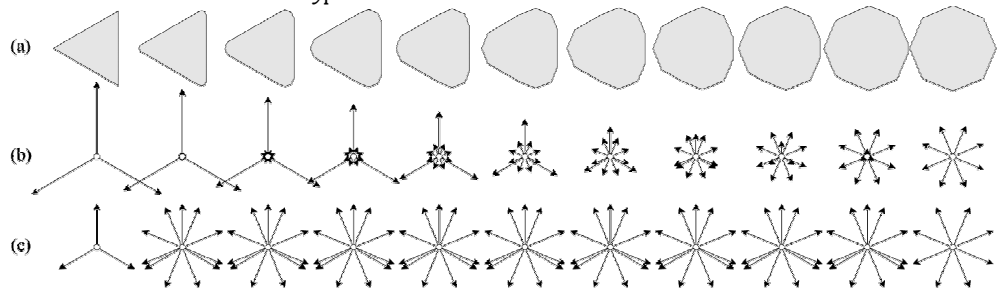


Fig. 1. An example output of LIDM (see Algorithm 1.): (a) smoothly changing polygons, (b) their direction maps, and (c) their normalized direction maps.

3. CONTOUR BLENDING BY LIDM

Recently, Lee et al. [9] 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 of LIDM is a sequence of interpolated polygons—one parameter family of polygons in a parametric form. Specially, the automatic correspondences work quite well for relatively simple shapes rather than the complex ones of character animation applications. Note that GC does not require so complex contours as in character animations. Hence, we propose to adopt LIDM as a contour blending method 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. A group of consecutive direction vectors may represent a geometric feature such as pocket, chamfer, and fillet. We assume that, in LIDM, the direction of any direction vector is invariant although its length may change. Hence, (1) the sequence of directions and (2) lengths of individual direction vectors are key factors determining the actual shape of a feature.

We can generate a new polygon by merging two direction maps representing different shapes. This step is analogous with blending features from different polygons. When merging direction maps, we change neither the direction nor the length of any direction vector; rather, the sequence of direction vectors is newly generated after merging with a certain geometric correspondence rule. Among various feature correspondence strategies, convolution merging is closely related to Minkowski sum or convolution operations, where vertex-wise feature correspondences are set up by geometric rules [9]. Note that LIMS (linear interpolation by Minkowski sum [8],[11]) is a special case of LIDM.

To generate in-between shapes continuously, we have to smoothly change the degree of dominating features. In LIDM, this is accomplished by changing the length of direction vectors using *group scaling* operation where a scaling factor is assigned to a group of direction vectors. The examples of scaling factors are Bézier or blossom basis functions. Using both merging and group scaling operations, Lee et al. proposed interpolation algorithm (for details, we refer readers to [9]) as follows:

Algorithm 1. LIDM

Input (1) A merged direction map:

$$D \neg D_0 + \mathrm{L} + D_m ;$$

(2) A set of scalar functions for the group scaling operation: $\mathbf{s}(t) = \{s_0(t), \ldots s_m(t)\}$; and

(3) The blending parameter: t.

Output A newly generated contour: $\mathbf{C}(t)$.

Procedure LIDM(D, $\mathbf{s}$, t; $\mathbf{C}(t)$)

Fig. 1. is an example output of LIDM algorithm applied for two direction maps of a triangle and an octagon: (a) a sequence of polygons changing from a triangle to an octagon; (b) by group scaling operations, dominating direction vectors become longer; however, (c) when the lengths are normalized into one, it is easy to find that the individual directions are invariant throughout the sequence.

4. GC IN POLYGONAL MESHES

In this section, we describe how to build a polygonal mesh of GC whose contours are blended by LIDM. For

the in-between shapes generated by LIDM, there exist two interesting geometric properties [9]: (1) all the in-between shapes have the same normalized direction maps; and (2) the number of vertices (or edges) of generated polygons is constant. These 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. (When correspondences should be made for direction maps of trimmed contours, more sophisticated rules should be applied to handle exceptions. We skip details here.)

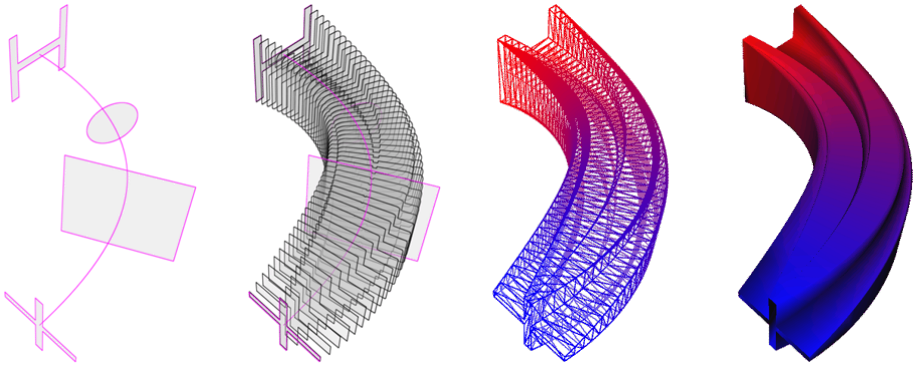


Fig. 2. An example result of LIDM-GC-Mesh (see Algorithm 2.): (a) control contours (in pink) arranged on a spine line, (b) blended contours (in gray), (c) the generated polygonal mesh, and (d) shaded result of (c)

Algorithm 2. LIDM-GC-Mesh

Input (1) Two terminal contours and additional control contours: $\mathbf{C} = \{\mathbf{C}_0,\ldots,\mathbf{C}_m\}$;

(2) A set of scalar functions (i.e., Bernstein polynomials): $\mathbf{B} = \{b_0(t),\ldots,b_m(t)\}$; and

(3) The number of in-between contours to be generated: n.

Output A polygonal mesh interpolating given control contours: $\mathbf{M}$.

Procedure

1. $D \neg D_0 + \mathrm{L} + D_m$; /* to merge directions maps: $D_i = \mathrm{DM}(A_i)$ */
2. $l \neg |D|$; /* the number of direction vectors in the merged direction map */
3. $\mathbf{C}_{\mathrm{prev}} \neg \mathbf{C}(0) = \mathrm{LIDM}(D, \mathbf{b}, 0)$; /* to compute the initial contour */
4. **For** $i = 1$ **to** $(n+1)$
5. $t_i \neg t = \mathrm{D}t * i$; /* $\mathrm{D}t = 1/(n+1)$ */
6. $\mathbf{C}_{\mathrm{curr}} \neg \mathbf{C}(t) = \mathrm{LIDM}(D, \mathbf{b}, t_i)$; /* assuming $|\mathrm{DM}(\mathbf{C}(t))|$ is invariant. */
7. $\mathbf{M}_i \neg \text{Æ}$; /* the initial value of the i-th sub-mesh */
8. **For** $j = 1$ **to** l /* to construct a sub-mesh $\mathbf{M}_i$ connecting $\mathbf{C}_{\mathrm{prev}}$ and $\mathbf{C}_{\mathrm{curr}}$ */
9. $\mathbf{m}_j \neg$ two triangle (or a quadrangle) constructed using the corresponding 4 vertices: j-th and $(j+1)$-th vertices of $\mathbf{C}_{\mathrm{prev}}$ and $\mathbf{C}_{\mathrm{curr}}$;
10. $\mathbf{M}_i \neg \mathbf{M}_i \text{ È } \mathbf{m}_j$; /* incremental addition of mesh elements */
11. $\mathbf{M} \neg \mathbf{M} \text{ È } \mathbf{M}_i$;
12. $\mathbf{C}_{\mathrm{prev}} \neg \mathbf{C}_{\mathrm{next}}$; /* incremental addition of sub-mesh */

In Algorithm 2., each execution of LIDM at t_i generates one intermediate contour $\mathbf{C}(t_i)$. Note that, however, direction map merging is computed just once during the whole execution since the feature correspondences are assumed to be static.

Each sub-mesh $\mathbf{M}_i$ Mi is generated by connecting two consecutive contours, $\mathbf{C}_{\text{prev}}$ and $\mathbf{C}_{\text{curr}}$. 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 sequentially combining all the sub-meshes $\mathbf{M}_i$ (See Algorithm 2.)

Fig. 2. illustrates the execution of LIDM-GC-Mesh algorithm. Fig. 2(a). shows four control contours (in pink) arranged on a spine curve. In Fig. 2(b)., gray contours are blended ones generated by LIDM. By connecting the result of Fig. 2(b)., LIDM-GC-Mesh generates a mesh surface (Fig. 2(c)). For display, we can apply standard mesh shading to the generated mesh (Fig. 2(d)).

Algorithm 3. LIDM-GC-Dvlp

Input Two terminal contours and additional control contours: $\mathbf{C} = \{\mathbf{C}_0, \ldots, \mathbf{C}_m\}$;

Output Sets of control points defining boundary curves of developable surface patches: $\mathbf{P} = \{\mathbf{P}_1, \ldots, \mathbf{P}_l\}$.

where $\mathbf{P}_i = \{\mathbf{p}_{i,0}, \ldots, \mathbf{p}_{i,m}\}$ and $\mathbf{P}_i$ defines a boundary curve $\mathbf{F}_i$.

Procedure

1. $D \neg D_0 + \mathrm{L} + D_m$; /* merge directions maps: $D_i = \mathrm{DM}(A_i)$ */
2. $l \neg |D|$; /* the number of direction vectors in D */
3. **for** $i = 1$ **to** l /* for each profile curve $\mathbf{F}_i$ defined by a direction vector $\mathbf{d}_i$ */
4. $\mathbf{d}_i \neg$ the i-th direction vector of D;
5. $\mathbf{P}_i \neg$ Æ;
6. **for** $j = 0$ **to** m /* find each set of control points $\mathbf{P}_i$ defining $\mathbf{F}_i$ */
7. $\mathbf{d} \neg$ find a direction vector of D that following two conditions:
   (1) the counter-clockwise nearest direction vector from $\mathbf{d}_i$ in D (including $\mathbf{d}_i$); and
   (2) its group id is j; /* selecting a sing vertex from $\mathbf{C}_j$ */
8. $\mathbf{p}_{i,j} \neg$ the vertex of $\mathbf{C}_j$ corresponding to the end point of $\mathbf{d}$;
9. $\mathbf{P}_i \neg \mathbf{P}_i \cup \{\mathbf{p}_{i,j}\}$ /* finding each set of control points */
10. $\mathbf{P} \neg \mathbf{P} \cup \{\mathbf{P}_i\}$; /* finding all the sets of control points */

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 (or 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 in NURBS [10].

When every control contour has the same orientation (i.e., not on the normal plane of Frenet frame), corresponding edges of blended contours are parallel to each other since they are constructed by the same direction vector whose direction is invariant over blending. Hence, each direction vector $\mathbf{d}_i$ (moving along a certain directrix curve) defines one developable surface patch $\mathbf{S}_i$ of cylindrical type. Note that, if a ruling direction is fixed over a directrix, it is a cylindrical developable surface.

Each developable surface patch $\mathbf{S}_i$ is bounded by two boundary curves: $\mathbf{F}_{i-1}(u)$ and $\mathbf{F}_i(u)$. Let these curves be *profile curves*. Each profile curve $\mathbf{F}_i(u)$ is identical to the sweep trajectory of a vertex of a blended contour. Based on the properties of merged direction map and group scaling operation [9], each vertex of a blended contour is defined by blending some vertices $\{\mathbf{p}_{i,0},\ldots,\mathbf{p}_{i,m}\}$ of control contours $\{\mathbf{C}_0,\ldots,\mathbf{C}_m\}$ with the same blending function used in group scaling operation.

For example, when applying Bernstein polynomials as blending functions, a profile curve is represented as follows:

$$\mathbf{F}_i(u) = \sum_{j=0}^{m} B_j^m(u) \cdot \mathbf{p}_{i,j} \ . \qquad (7)$$

Actually, the type of boundary curve (or surface) is defined by how we blend contours using a certain scaling factors such as blossom or NURBS basis functions.

It is clear that profile curves satisfy the following condition:

$$\langle \mathbf{F}_i(u) - \mathbf{F}_{i-1}(u), \mathbf{d}_i \rangle = 0 \quad . \tag{8}$$

This leads to the following relation for every pair of neighboring control points:

$$\langle \mathbf{p}_{i,j} - \mathbf{p}_{i-1,j}, \mathbf{d}_i \rangle = 0 \, . \tag{9}$$

Hence, control points of $\mathbf{F}_{i-1}(u)$ and $\mathbf{F}_i(u)$ are identical except the pairs whose direction $(\mathbf{p}_{i,j} - \mathbf{p}_{i-1,j}) \neq 0$ is parallel to $\mathbf{d}_i$.

Using this property, Algorithm 3. can find control points of each profile curve regardless of convexity of control contours.

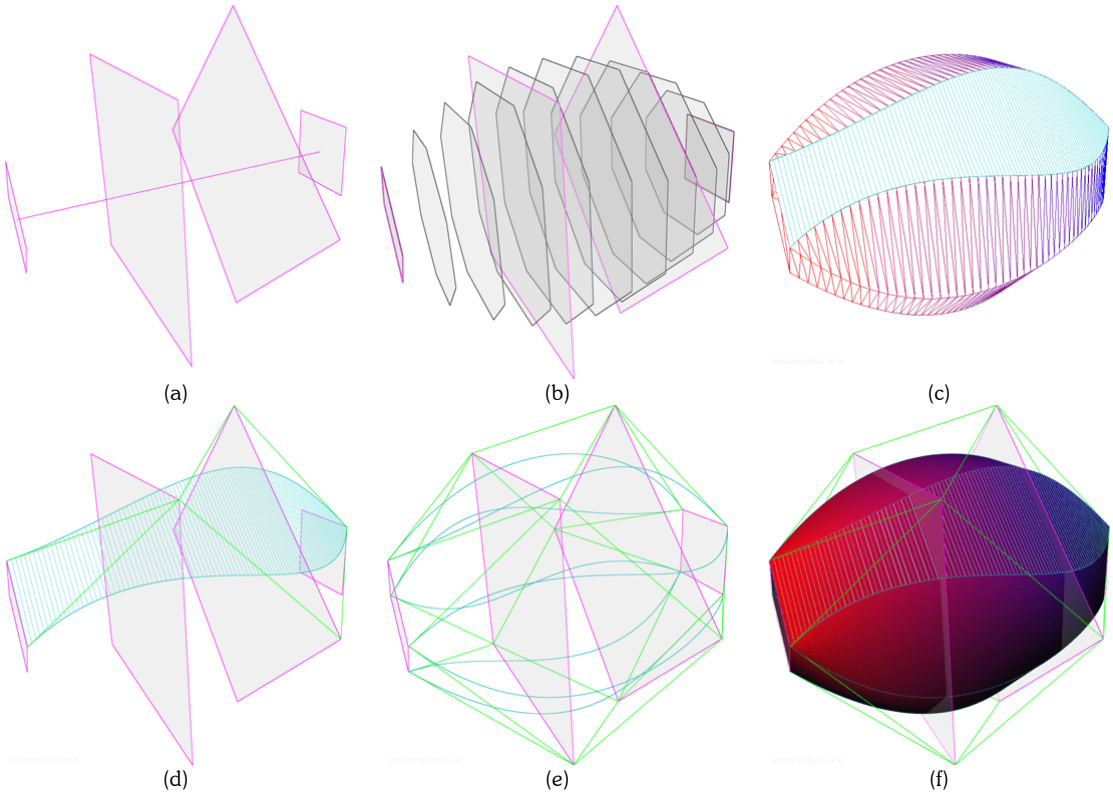


Fig. 3. An example result of LIDM-GC-DvIp (see Algorithm 3.): (a) control contours; (b) blended contours (in gray); (c) meshes (in gradient color) and a developable surface patch (in cyan); (d) control points (connected by green lines) of boundary curves; (e) the control points of all boundary curves; and (f) developable surface patches and their control points.

If we apply Bernstein polynomials of degree *m* as scaling factors for the group scaling operation, each developable surface can be represented as Bézier surface of degree (*m*, 1) as follows:

$$\mathbf{S}_i(u,v) = (1-v)\mathbf{F}_{i-1}(u) + v\mathbf{F}_i(u) \tag{10}$$

$$= (1-v)\sum_{j=0}^{m} B_j^m(u)\cdot\mathbf{p}_{i-1,j} + v\sum_{j=0}^{m} B_j^m(u)\cdot\mathbf{p}_{i,j}$$

$$= \sum_{j=0}^{m}\sum_{k=0}^{1} B_j^m(u)B_k^1(u)\cdot\mathbf{p}_{i-1+k,j}, \tag{11}$$

where $i \in \{1,\ldots,l\}$.

Fig. 3. shows an example result of where each patch is a Bézier surface of degree (3, 1). In Fig. 3(a)., four control contours—in this case, quadrangles of different sizes—are placed on a straight line with the same orientation. In Fig. 3(b)., blended contours (in gray) are generated by LIDM-GC-Mesh algorithm using Bernstein polynomials as blending functions (i.e., scalar functions of group scaling operation) In Fig. 3(c)., when a mesh

(in gradient color) is displayed, we can easily observe an approximated developable surface patch (in cyan) bounded by two boundary curves. (Note that, although LIDM-GC-Mesh is used in Fig. 3(b). for illustration, it has no dependency on LIDM-GC-Dvlp.)

In Fig. 3(d)., for each developable surface patch, we can select a set of control points (connected by green line) of each boundary curve among the vertices of control contours using LIDM-GC-Dvlp algorithm. In Fig. 3(e)., after we find all the control points, we can evaluate the boundary curves as Bézier curves. (Evaluated boundary curves give an impression of the GC shape.) Otherwise, as in Fig. 3(f)., we can adopt existing shading algorithms for Bézier surfaces.

Note that LIDM-GC-Dvlp is more powerful than LIDM-GC-Mesh. If we want to generate in-between contours using LIDM-GC-Dvlp, we can connect the points from every boundary curves evaluated at the same value in sequential order. For example, a contour at the spine curve $\mathbf{K}(u)$ is defined by following vertices: $\{\mathbf{F}_1(u), \mathbf{F}_2(u), ..., \mathbf{F}_l(u)\}$. However, it is not easy to generate a parametric form of boundary curves using LIDM-GC-Mesh since it is based on contour-wise evaluations.

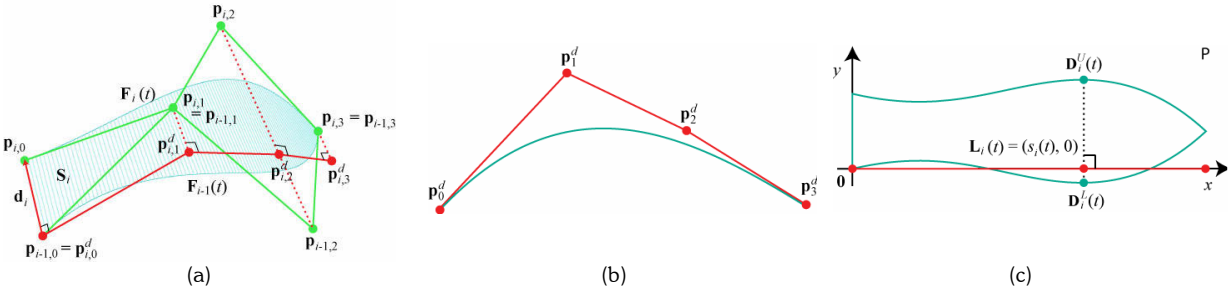


Fig. 4. Development of a developable surface patch on a plane: (a) finding the control points (connected in red line) of a normal directrix curve, (b) the normal directrix and its control points on a plane normal to the direction vector $\mathbf{d}_i$, and (c) developing on a plane whose coordinate axis are defined by the normal directrix and direction vector.

6. PLANE DEVELOPMENT OF GC

In this section we explain plane development algorithm for the developable surface patches generated by LIDM-GC-Dvlp. In addition, we provide examples experimented with real paper patches.

In Fig. 4(a)., a developable surface patch $\mathbf{S}_i$ (the same one introduced in Fig. 3(d).) is defined by the ruling parallel to direction vector $\mathbf{d}_i$ along a certain directrix curve.

For plane development, we define a special directrix curve $\mathbf{F}_i^d(t)$ for each developable surface patch $\mathbf{S}_i$ that is normal to $\mathbf{d}_i$ Such a directrix curve is referred to as *normal directrix*, which is the trace of principal direction along the maximum normal curvature We can derive control points $\mathbf{p}_{i,j}^d$ (connect in red lines in Fig. 4(a).) of $\mathbf{F}_i^d(t)$ as follows:

$$\mathbf{d}_i = \mathbf{p}_{i,j} - \mathbf{p}_{i-1,j}, \quad (\mathbf{d}_i \neq 0) \tag{12}$$

$$\mathbf{p}_{i-1,0} = \mathbf{p}_{i,0}^d, \tag{13}$$

$$\mathbf{p}_{i-1,j} = \mathbf{p}_{i,j}^d + h_{i,j}^L \cdot \mathbf{d}_i, \tag{14}$$

$$\mathbf{p}_{i,j} = \mathbf{p}_{i,j}^d + h_{i,j}^U \cdot \mathbf{d}_i \tag{15}$$

$$= \mathbf{p}_{i+1,j}^d + h_{i+1,j}^L \cdot \mathbf{d}_{i+1}, \tag{16}$$

where h_*^L and h_*^U are scalar values.

Based on the planarity of $\mathbf{p}_{i,j}^d$, the following property holds:

$$0 = \left\langle (\mathbf{p}_{i-1,j}^d - \mathbf{p}_{i-1,0}^d), \mathbf{d}_i \right\rangle \tag{17}$$

$$= \left\langle (\mathbf{p}_{i-1,j} - h_{i,j}^L \cdot \mathbf{d}_i - \mathbf{p}_{i-1,0}), \mathbf{d}_i \right\rangle \tag{18}$$

Using above property, we can sequentially derive the following equations:

$$h_{i,j}^{L} = \frac{\left\langle (\mathbf{p}_{i-1,j} - \mathbf{p}_{i-1,0}), \mathbf{d}_i \right\rangle}{\left\langle \mathbf{d}_i, \mathbf{d}_i \right\rangle} \tag{19}$$

$$\mathbf{p}_{i,j}^{d} = \mathbf{p}_{i-1,j} - h_{i,j}^{L} \cdot \mathbf{d}_i \tag{20}$$

$$h_{i,j}^{U} = \frac{\left\langle (\mathbf{p}_{i,j} - \mathbf{p}_{i,j}^{d}), \mathbf{d}_i \right\rangle}{\left\langle \mathbf{d}_i, \mathbf{d}_i \right\rangle} \tag{21}$$

Now, we can derive the profile curve as follows:

$$\mathbf{F}_{i-1}(t) = \sum_{j=0}^{m} B_j^3(t) \cdot \mathbf{p}_{i-1,j} \tag{22}$$

$$= \sum_{j=0}^{m} B_j^3(t) \cdot \left(\mathbf{p}_{i,j}^{d} + h_{i,j}^{U} \cdot \mathbf{d}_i \right) \tag{23}$$

$$= \sum_{j=0}^{m} B_j^3(t) \cdot \mathbf{p}_{i,j}^{d} + \left(\sum_{j=0}^{m} B_j^3(t) \cdot h_{i,j}^{L} \right) \cdot \mathbf{d}_i$$

$$= \mathbf{F}_i^{d}(t) + H_i^{L}(t) \cdot \mathbf{d}_i \tag{24}$$

Similarly, we can derive the neighboring profile curve as follows:

$$F_i(t) = F_i^{d}(t) + H_i^{U}(t) \cdot d_i \tag{25}$$

$$F_i(t) = F_{i+1}^{d}(t) + H_{i+1}^{L}(t) \cdot d_{i+1} \tag{26}$$

Let the development function satisfy the following two conditions:

(i) When we develop the normal directrix curve $\mathbf{F}_i^{d}(t)$ on a plane P, it becomes a straight line $\mathbf{L}_i(t)$ on the x-axis of P such that

$$F_i(\mathbf{F}_i^{d}(t)) = \mathbf{L}_i(t) = (s_i(t), 0), \tag{27}$$

where

$$s_i(t) = \int_0^t \left\| \dot{\mathbf{F}}_i^{d}(t) \right\| dt. \tag{28}$$

(ii) When we develop the ruling vector $\mathbf{d}_i$ on P, it becomes a vector on the y-axis of P:

$$F_i(\mathbf{d}_i) = \mathbf{y}_i = (0, \|\mathbf{d}_i\|). \tag{29}$$

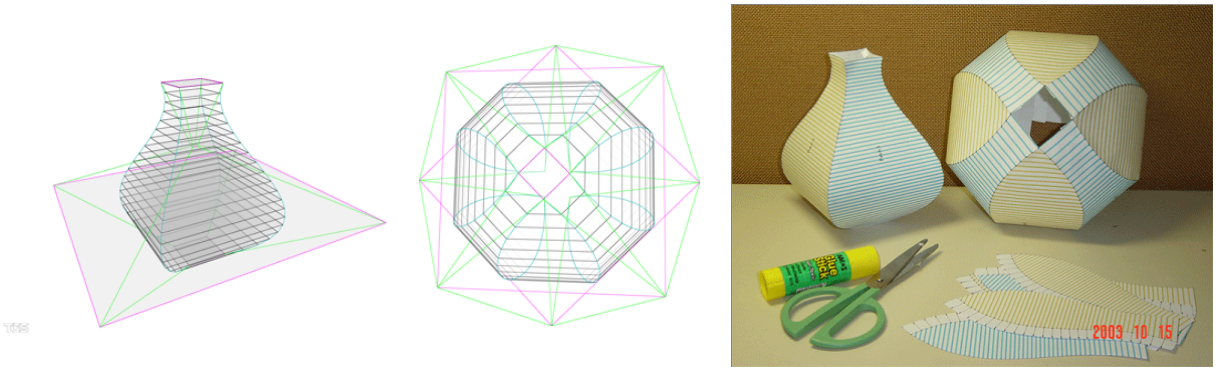
Algorithm 4. LIDM-GC-Plane-Dvlp

Input (1) Output of LIDM-GC-Dvlp of Algorithm 3: $\mathbf{P}$; and
(2) Number of samples for curve length approximation: n.

Output A set of polygons representing planar boundary curves of developable surface patches:
$\mathbf{C}^p = \{\mathbf{C}_0^p, \ldots, \mathbf{C}_l^p\}$ where $\mathbf{C}_i^p$ is composed of a list of planar vertices $\{\mathbf{v}_{i,j}\}$.

Procedure

1. **for** $i = 1$ **to** l /* for each direction vector as the ruling of a developable surface patch */
2. **for** $j = 0$ **to** m /* to find the ruling (or direction vector) of the i-th patch */
3. $\mathbf{d}_i \neg \mathbf{p}_{i,j} - \mathbf{p}_{i-1,j}$;
4. **if** $|\mathbf{d}_i| \neq 0$ **then break**;
5. **for** $j = 0$ **to** m /* find each set of control points $\mathbf{P}_i$ defining $\mathbf{F}_i$ */
6. $h_{i,j}^{L} \leftarrow \dfrac{\left\langle (\mathbf{p}_{i-1,j} - \mathbf{p}_{i-1,0}), \mathbf{d}_i \right\rangle}{\left\langle \mathbf{d}_i, \mathbf{d}_i \right\rangle}$; /* $\mathbf{h}_i^{L} = \{h_{i,0}^{L}, \ldots, h_{i,m}^{L}\}$ */
7. $\mathbf{p}_{i,j}^{d} \leftarrow \mathbf{p}_{i-1,j} - h_{i,j}^{L} \cdot \mathbf{d}_i$; /* $\mathbf{P}_i^{d} = \{\mathbf{p}_{i,0}^{d}, \ldots, \mathbf{p}_{i,m}^{d}\}$ */
8. $h_{i,j}^{U} \leftarrow \dfrac{\left\langle (\mathbf{p}_{i,j} - \mathbf{p}_{i,j}^{d}), \mathbf{d}_i \right\rangle}{\left\langle \mathbf{d}_i, \mathbf{d}_i \right\rangle}$; /* $\mathbf{h}_i^{U} = \{h_{i,0}^{U}, \ldots, h_{i,m}^{U}\}$ */
9. **for** $j = 0$ **to** n /* to get the vertices of $\mathbf{C}_i^p = \{\mathbf{v}_{i,j}\}$ */
10. $t \leftarrow \Delta t * j$; /* $\Delta t = 1/n$ */
11. $x \leftarrow$ CurveLengthBezierCurve3D$(t, \mathbf{P}_i^{d})$; /* get the curve length of $\mathbf{F}_i^{d}(t)$ from 0 to t */
12. $y^{L} \leftarrow |\mathbf{d}_i| \cdot$ BezierCurve1D$(t, \mathbf{h}_i^{L})$; /* one dimensional Bézier curve */
13. $\mathbf{v}_{i,j} \neg (x, y^{L})$; /* the vertex in the lower developed boundary: $\mathbf{D}_i^{L}(t)$ */
14. $y^{U} \leftarrow |\mathbf{d}_i| \cdot$ BezierCurve1D$(t, \mathbf{h}_i^{U})$; /* one dimensional Bézier curve */
15. $\mathbf{v}_{i,(2n+1-j)} \neg (x, y^{U})$; /* the vertex in the upper developed boundary: $\mathbf{D}_i^{U}(t)$ */



(a) (b) (c)

Fig. 5. Examples of plane development based on LIDM-GC-Plane-Dvlp (see Algorithm 4.): Computer-aided design of (a) a bottle and (b) a bowl; and (c) corresponding physical models assembled using paper patches.

When we develop $\mathbf{F}_{i-1}(t)$ on P using F_i, it becomes a planar curve $\mathbf{D}_i^L(t)$ of the following coordinates of P :

$$F_i(\mathbf{F}_{i-1}(t)) = \mathbf{D}_i^L(t) = \mathbf{L}_i(t) + H_i^L(t)\cdot\mathbf{y}_i \quad (30)$$

$$= (s_i(t), 0) + H_i^L(t)\cdot(0, \|\mathbf{d}_i\|) \quad (31)$$

$$= (s_i(t), H_i^L(t)\cdot\|\mathbf{d}_i\|). \quad (32)$$

Similarly, we can develop $\mathbf{F}_{i+1}(t)$ on P as $\mathbf{D}_i^U(t)$ as follows:

$$F_i(\mathbf{F}_i(t)) = \mathbf{D}_i^U(t) \quad (33)$$

$$= (s_i(t), H_i^U(t)\cdot\|\mathbf{d}_i\|), \quad (34)$$

$$F_{i+1}(\mathbf{F}_i(t)) = \mathbf{D}_{i+1}^L(t) \quad (35)$$

$$= (s_{i+1}(t), H_{i+1}^L(t)\cdot\|\mathbf{d}_{i+1}\|). \quad (36)$$

LIDM-GC-Plane-Dvlp of Algorithm 4. can be used to compute plane development of developable surface patches that are generated by LIDM-GC-Dvlp of Algorithm 3. Note that, however, LIDM-GC-Plane-Dvlp can not produce the exact boundary curves on a plane since there is no closed form solution to compute the length of an arbitrary parametric curve as in Eqn. (28). Hence, we have to approximate the curve lengths using a known method. For example, we adopted the circular arc method of Vincent et al. [13] in CurveLengthBezierCurve3D algorithm which is used in the body of Algorithm 4.

Fig. 5. shows real examples of plane development of GC's. Fig. 5(a). is a bottle and Fig. 5(b). is a bowl. (They are defined by four simple control contours to simplify the paper manipulation.) Fig. 5(c). is a picture of physical models made of real paper development of developable surface patches of Fig. 5(a)-(b). After designing the GC's using LIDM-GC-Dvlp, the result of plane development using LIDM-GC-Plane-Dvlp was printed on papers automatically, and then cut with scissors and glued, of course, manually.

7. DISCUSSION

In this paper, we proposed three algorithms regarding GC based on direction map representations: (1) LIDM-GC-Mesh, (2) LIDM-GC-Dvlp, (2) LIDM-GC-Plane-Dvlp. The first algorithm generates polygonal meshes of GC using the previous method called LIDM. This is a simple procedure of $O(n\cdot l)$ where n and l are the number of blended contours and the number of direction vectors in a merged direction map, respectively.

The second algorithm generates cylindrical developable surface patches of GC whose contour planes have the same orientation. This algorithm has the complexity of $O(m\cdot l)$ where m is the number of control contours. It sequentially generates control points of each developable surface patch.

The third algorithm computes plane development for the result of the second algorithm. Although it is simple two-fold loop of $O(n\cdot l)$, its performance depends on the computation of curve length.

The implementation of proposed algorithms is straightforward, and their overall computations are fast enough to be implemented in interactive geometric design applications.

As a further work, the supported developable surface patches should include cone and tangent envelop types.

8. ACKNOWLEDGEMENT
This work has been supported in part by grant no. A1-03-0021-00 of Korean Ministry of Information and Communication. The preliminary result of this work was presented in *IJCC Workshop on Digital Engineering*, Hyatt Hotel Jeju, Korea, in August 21, 2003.

9. REFERENCES

[1] Akman, V. and Arslan, A., Sweeping with all graphical ingredients in a topological picturebook, *Computer & Graphics*, 16, 1992, pp 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), pp 621-632.

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

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

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

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

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

[8] Lee, J.-H., Lee, J.Y., Kim, H. and Kim, H.-S., Interactive Control of Geometric Shape Morphing based on Minkowski Sum, *Transactions on SCCE*, 7(4), 2002, pp 317-380. http://joohaeng.etri.re.kr/pub/ToSCCE-2002-LIMS.pdf

[9] Lee, J.-H., Kim, H. and Kim, H.-S., Efficient Computation and Control of Geometric Shape Morphing based on Direction Map, *Transactions on SCCE*, 8(4), 2003, pp 243-253. http://joohaeng.etri.re.kr/pub/TR-2003-LIDM.pdf

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

[11] Rossignac, J. and Kaul, A., AGRELs and BIPs: Metamorphosis as a Bezier curve in the space of Polyhedra, in *Proceedings of EUROGRAPHICS '94*, 1994; C179-C184.

[12] Shapira, M. and Rappoport, A., Shape blending using the star-skeleton representation, *IEEE Computer Graphics & Applications*, 15(2), 1995, pp 44-50.

[13] Vincent, S. and Forsey, D., Fast and accurate parametric curve length computation, *Journal of Graphics Tools*, 6(4), 2001, pp 29-40.

[14] Voogt, E. de, Helm, A. van der and Bronsvoort W.F., Ray tracing deformed generalized cylinders, *The Visual Computer*, 16, 2000, pp 197-207.