Wen-Yong Gong
Institute of Mathematics,
Jilin University,
Changchun 130012, China
e-mail: gongwenyong@gmail.com

Yong-Jin Liu
TNList,
Department of Computer
Science and Technology,
Tsinghua University,
Beijing 100084, China
e-mail: liuyongjin@tsinghua.edu.cn

Kai Tang
Department of Mechanical Engineering,
Hong Kong University of
Science and Technology,
Hong Kong 00852, China
e-mail: mektang@ust.hk

Tie-Ru Wu
Institute of Mathematics,
Jilin University,
Changchun 130012, China
e-mail: wutr@jlu.edu.cn

Variational Discrete Developable Surface Interpolation

Modeling using developable surfaces plays an important role in computer graphics and computer aided design. In this paper, we investigate a new problem called variational developable surface interpolation (VDSI). For a polyline boundary P, different from previous work on interpolation or approximation of discrete developable surfaces from P, the VDSI interpolates a subset of the vertices of P and approximates the rest. Exactly speaking, the VDSI allows to modify a subset of vertices within a prescribed bound such that a better discrete developable surface interpolates the modified polyline boundary. Therefore, VDSI could be viewed as a hybrid of interpolation and approximation. Generally, obtaining discrete developable surfaces for given polyline boundaries are always a time-consuming task. In this paper, we introduce a dynamic programming method to quickly construct a developable surface for any boundary curves. Based on the complexity of VDSI, we also propose an efficient optimization scheme to solve the variational problem inherent in VDSI. Finally, we present an adding point condition, and construct a G^1 continuous quasi-Coons surface to approximate a quadrilateral strip which is converted from a triangle strip of maximum developability. Diverse examples given in this paper demonstrate the efficiency and practicability of the proposed methods.
[DOI: 10.1115/1.4026291]

1 Introduction

Developable surfaces can be unfolded into a plane without any stretching and distortions [1]. Many researchers have paid much attention to the design of developable surfaces [2–7]. Even before the modern time, developable surfaces already played an important role in human's life. For example, a popular way of making the hull of a boat in those days was to bend wood or bamboo strips and piece them together, which literally restricted the final hull shape to be developable. Today developable surfaces have extensive applications in many fields of engineering and manufacturing such as architecture [8], garment industry [9,10], sheet metal industry, and so on. In computer aided design (CAD) and computer graphics (CG), developable surfaces are also ubiquitous, including architecture and industrial surface design [5].

Discrete developable surfaces are developable triangular or quadrilateral patches that maximize a discrete developability measure $m(tp)$ [6]. The research on discrete developable surfaces, as far as practical computer aided manufacturing (CAD/CAM) is concerned, deals with how to design a quadrilateral or triangular mesh surface that meets the given geometric constraints while at the same time is exactly $(m(tp) = 0)$ or quasi $(m^* = \arg\min\{m(tp)\} > 0)$ developable. Parametric surfaces are not considered a viable choice for the task of complex free-form developable shape design, due to their stringent and inflexible requirements on the geometric constraints. Depending on specific applications, this task can be further categorized into two groups: approximation and interpolation. Although both seek as much developability as possible on the final shape, in the former the satisfaction of the geometric constraints is desired but not strictly enforced, whereas in the latter meeting the interpolation constraints is strictly required (at least theoretically). For example, clothing simulation falls into the first group since it is mainly used for the purpose of rough estimate or examination. On the other hand, when designing the shape of a blank holder surface, the geometric constraints (interpolation of the given punch lines/curves) must be met as strictly as possible, since a punch line usually has a matching part and, if the final blank holder surface does not contain this punch line, a mismatch would occur.

Sometimes a given geometric constraint is not completely rigid but allowed to vary within certain bound. In such a case, the task, which is more challenging, is to modify the constraint within the bound such that a better (in terms of developability) interpolating surface can be obtained. For example, when designing the shape of an architectural structure, the design intended could be the boundary curve and maybe some key interior points; this boundary curve, however, is allowed to vary within a small bound. Without being too rigorous on terminology, we will refer this task as the *variational developable surface interpolation* (VDSI).

The VDSI, at least in the prescribed setting, to the best of our knowledge has not been studied before. Existing approaches of interpolating or approximating developable surfaces are not suitable for VDSI. In this paper, we formulate the VDSI into an optimization problem with constraints and develop an optimization algorithm to solve it. The presented method can generate a better developable surface to interpolate a boundary curve which could be varied within a bound.

2 Background

In this section, we mainly give preliminaries of discrete developable surfaces, and then review some related works.

2.1 Fundamentals. Developable surfaces are a subset of ruled surfaces. A ruled surface S is a family of straight lines. The surface is formed by a straight line sweeping out along a space curve. Thus, for every point $p \in S$, there exists a straight line which passes through p and also lies in S. The straight line is called a ruling. Let $\alpha(u)$ and $\beta(u)$ be two space curves, mathematically a ruled surface has the following parameterization representation:

Contributed by the Design Engineering Division of ASME for publication in the Journal of Computing and Information Science in Engineering. Manuscript received February 27, 2012; final manuscript received December 16, 2013; published online February 26, 2014. Editor: Bahram Ravani.

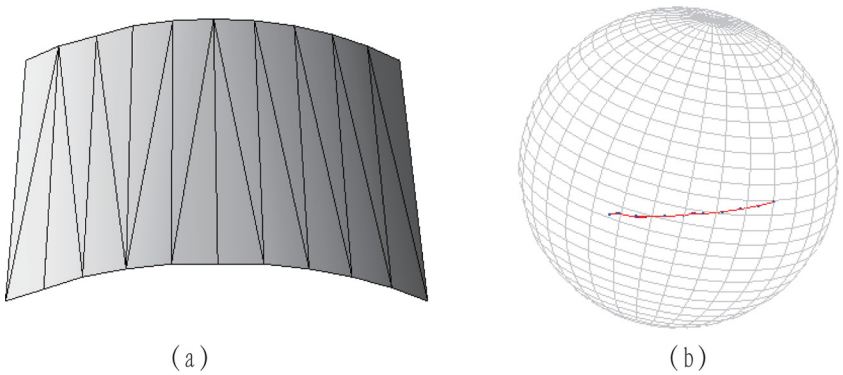


Fig. 1 (*a*) Discrete developable surface. (*b*) Normal map

$$S(u,v) = \alpha(u) + v\beta(u),$$

where $\alpha(u)$ is called the base curve and $\beta(u)$ the director curve. If we fix u, a straight line or a ruling is formed. As u continuously varying, we could imagine the generation process of ruled surfaces.

A surface is flat or developable if it has vanishing Gaussian curvature everywhere. Since Gaussian curvature measures how much a surface deviates from a Euclidean plane, the developable surface can be unfolded into a plane without any distortions and tearing [1]. As ruled surfaces have the above parameterization representation, they are said to be developable if $\det(\alpha'(u)\beta(u)\beta'(u)) = 0$. Actually developable surfaces also have intuitive geometric description. A ruled surface is developable if and only if all points of a ruling have common tangent plane. Thus the normal map of a developable surface is one-dimensional [11] (see Fig. 1 for an illustration).

For a discrete developable surface (i.e., a triangular mesh) interpolating a closed polyline, an edge in the mesh is called a bank or boundary edge if its two ending points are neighboring vertices on the polyline, otherwise it is called a bridge. In the following, we formally describe two special triangulations of a closed polyline: *boundary bridge triangulation* (BT) and *boundary triangulation* [6,10]. If the vertices of the triangulation belong to that of the polyline only, and any triangle in the triangulation has at least one bank edge, then the triangulation is called a boundary bridge triangulation. While a boundary triangulation is still required to have only vertices on the polyline, its triangles, however, are allowed to comprise bridge edges only. By construction, a (discrete) boundary bridge triangulation or boundary triangulation is always developable, since both can be flatten onto a plane without any area or length distortions. In Fig. 2, a boundary bridge triangulation and a boundary triangulation are generated for a same closed polyline and flattened into the plane. A triangulation, however, may not approximate a smooth developable surface in the limit, unless the majority of its interior edges are locally convex [12]. An interior edge is said to be locally convex if it lies on a convex hull of itself and its four neighboring vertices on the polyline.

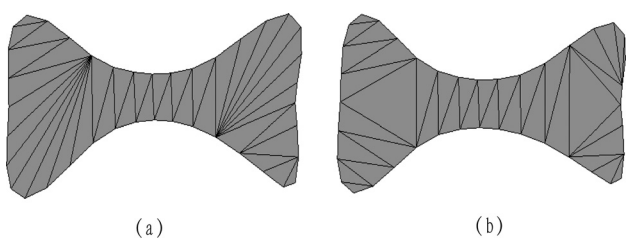


Fig. 2 Boundary bridge triangulation (*a*) and boundary triangulation (*b*) generated by the same closed polyline

2.2 Related Work. Since developable surfaces have Gaussian curvature $K = 0$ at every point, they are isometric to a plane in R^3. This important property is desired by CAD and CG. Hence various methods of modeling developable surfaces are proposed by researchers.

Many researchers pay attention to developing new algorithms to design developable surfaces. The work [13,14] is the first to construct developable surfaces using computer from different considerations. The former constructs developable surfaces for a given directrix and a given generator direction, and the latter interpolates two boundary curves to construct developable surfaces. As it is well known, Bézier or B-Spline surfaces are widely used in surface modeling and industry product design. Thus many algorithms are proposed to represent developable surfaces using Bezier or B-Spline form [15,16]. The original work of using the dual space transformation to design developable surfaces was proposed in Ref. [17]. Some related works were also presented in Refs. [16] and [18]. On the other hand, Aumann [19] took advantage of the de Casteljau algorithm to model developable Bézier surfaces. Despite their mathematically rigor and elegance, the analytic or parametric form is not suitable for practical developable surface design due to the non-linear characteristics.

In 3D computer graphics, triangular mesh surfaces are a commonly used means to represent 3D models. They have many advantages such as discrete representation, easy to compute and store, and so on. Recently there has been a flux of research work in the modeling and design of developable triangular meshes that exactly interpolate one or two boundary curves.

Frey's work [12] not only explicitly discussed the problem of interpolating a closed 3D curve by quasi-developable mesh surface but also provided a practical solution to the problem. In his work, he assumed that the final mesh surface must be monotone with respect to a known plane-the XY plane, and thus there is a one-to-one mapping between the interpolating mesh and its projection in the XY plane. Since a developable surface must be one or an assembly of ruled surfaces which are made of (infinite) line segments, he proposed the plausible idea of *boundary triangulation* (i.e., triangulations whose vertices can only fall on the given curves). Since the computation is fairly time-consuming, it cannot deal with developable surface design task with sufficiently large number of sample points. Wang and Tang [6] proposed a method to generate developable surfaces interpolating boundary curves using a weighted graph. For the special structure of the weighted graph, their method can achieve a global optimal solution in linear time. Observing that a ruling of a developable surface lies on a local convex hull of the surface, Rose et al. [5] presented an approach to the discrete developable surface interpolation task. But their work has several disadvantages. First, the convex hull computation is time-consuming and numerically unstable. Second, the final solution needs to be chosen by users. Recently, Liu et al. [4] proposed a global optimal solution to the optimal triangulation (OT) of a closed boundary using a dynamic programming

method. Their method is fairly fast and can support more than one closed polyline.

On the other hand, there exist many algorithms as well for the task of approximating a surface, smooth, or discrete, by a discrete or smooth developable surface. In the work [20], Robson et al. minimize the Gaussian curvature of a surface to reach an approximation surface with maximal developability, and they simulate garment design based on sketch. Observing the sum of angles incident with an interior point of 2D meshes is equal to 2π, Wang [21,22] presented to approximate developable surfaces with flattenable meshes. In general, however, approximation methods have not been very successful in practice due to two main reasons. The first is that they require the Gaussian curvature of input mesh to be small enough, otherwise these methods may fail. Another reason is that the approximation result usually is not able to satisfy the geometric constraints precisely.

As a balance of interpolation and approximation, the proposed VDSI formulation and our solution strive to fulfill the gap between the two. Our main contribution lies in that we are the first to formulate this problem, and then develop an optimization method to solve it. Furthermore, we present an adding point condition, and the condition can be used to guide and construct a G^1 continuous quasi-Coons surface to approximate a quadrilateral strip, which is converted from a maximal developable triangle strip.

The rest of this paper is organized as follows. In Sec. 3, a dynamic programming-based solution for the discrete developable surface interpolation problem is described in detail. Subsequently, we formulate the VDSI as an optimization problem, and develop an optimization method to solve it. In Sec. 5, we present an adding point condition, and construct a G^1 continuous quasi-Coons surface to approximate a quadrilateral strip, which is converted from a maximal developable triangle strip. Extensive examples and applications are also presented in Sec. 6. At last we conclude the whole paper.

3 Preliminary

In this section, we review a method proposed in Refs. [4] and [23] by describing their key ingredients of generating developable surfaces for interpolating polyline boundaries by using the dynamic programming idea. Since a developable surface must be a ruled surface, boundary triangulations are a natural way of developable interpolation. However, the number of different boundary triangulations could be exponentially large [5] in terms of the sample points on the boundaries, which makes an exhaustive search impractical. According to the recursive nature of dynamic programming, we could realize the global optimization, which then is able to tremendously reduce the search time and make it practically useful.

Assuming P is a closed polyline approximating a simple closed boundary curve in 3D, and its vertex set which is also denoted as P consist of $P_i (i = 1, \ldots, m)$ with counter-clockwise order. A line segment P_iP_j, to be denoted as $\langle i,j\rangle$, is called a rung if it is entirely inside P except its two ending points (see Figs. 3(*a*) and 3(*b*)). A rung is free-twist if every point in the rung has a common tangent plane. A (discrete) surface is developable if all rungs are twist-free. For better lucidity of description, when a rung $\langle i,j\rangle$ is referred, it is assumed that $i < j$. For any rung $l = \langle i,j\rangle$, we use $w(l)$ to measure its twist

$$w(l) = 1 - \frac{t_i \times (P_j - P_i)}{\|P_j - P_i\|} \cdot \frac{t_j \times (P_i - P_j)}{\|P_i - P_j\|} \tag{1}$$

where t_i and t_j represent the unit tangent vectors at P_i and P_j, respectively. For any triangulation T of P, let $W(T) = \sum w(e)$ denote the total twist of T, with the summation over all the edges in T. In particular, let $D(P)$ represent the minimum of all the total twists of P over all the possible triangulations of P, and the particular T^* that realizes $D(P)$ will be called an OT of P.

As P is simple, P is divided into two halves by any rung $l = \langle i,j\rangle$; let $P(l^+)$ (shaded area in Figs. 3(*a*) and 3(*b*)) denote the half that has the vertex P_{i+1}, and $P(l^-)$ the other half. (Follow the C convention, we use"$i+$" and"$i-$" to denote" $i+1$" and" $i-1$", respectively.) Now, suppose a rung l is an edge in an OT of P, to be referred to as an optimal rung. Then it is obvious that the equation below is true,

$$D(P) = D(l^+) + D(l^-) + w(l) \tag{2}$$

The above equation holds for any valid OT of P. Now we define a particular type of triangulation called *bridge triangulation*: in a BT, every triangle has at most two rungs. Every triangle in a BT calls a bridge triangle (see Fig. 3(*a*) for an illustration). For any optimal BT of $D(P(l^+))$, it must have either rung $\langle i,j-\rangle$ or $\langle i+,j\rangle$ as an edge. The following recursive propagation rule for a optimal rung $l = \langle i,j\rangle$ can be easily verified:

$$\begin{aligned} D(P(\langle i,j\rangle^+)) = \mathrm{Min}\{&D(P(\langle i+,j\rangle)^+) + w(\langle i+,j\rangle), D(P(\langle i,j-\rangle)^+) \\ &+ w(\langle i,j-\rangle)\} \end{aligned} \tag{3}$$

Accordingly, the recursive rule for $D(P(\langle i,j\rangle^-))$ can be similarly defined and is omitted here for brevity. Note that this recursion terminates when $j = i + 2$ (mod m).

Now, consider an arbitrary vertex P_i. Let L set $L(i)$ represent the set of rungs emanating from P_i (there are less than $m-3$ of them and in most cases much less). Any valid triangulation of P must contain either a rung in $L(i)$ or the rung $\langle i-,i+\rangle$. Therefore, by applying Eq. (2) to every rung in $L(i)$ and also $\langle i-,i+\rangle$, and picking the minimal $D(P)$ of the results, the corresponding triangulation is an optimal BT.

In order to avoid the exponential running time, dynamic programming algorithm is adopted to implement the rule (3).

The optimal triangulation obtained through the application of rules (2) and (3) is limited to bridge triangulation only every triangle must have an edge of P. Although most practical surfaces do not have or even try to avoid large flat regions due to various reasons (e.g., aesthetics), some do, at least in the initial design stage. Now we want to develop a similar recursive algorithm for the general optimal triangulation problem allowing triangles to have bridge edges only. Toward this goal, we make the following assumption: for the polyline P, there is a plane B such that any triangulation of P is monotone with respect to B (which also implies that the projection of the triangulation in B is inside the projection of P in B, $B(P)$). This assumption is in general valid in practice, e.g., for automobile parts or architectural designs. Let $l = \langle i,j\rangle$ be a rung, a vertex P_k is said to be visible from l if both $\langle i,k\rangle$ and $\langle k,j\rangle$ are rungs. Because C is simple, if p_k is visible, then the triangle $P_iP_jP_k$ lies entirely inside $B(P)$ except the three vertices. Let $V(i)$ denote the set of indices of the vertices visible from P_i, the following recursion rule then is naturally established:

$$\begin{aligned} D(P(\langle i,j\rangle^+)) = \mathrm{Min}\{&D(P(\langle i,k\rangle^+)) + w(\langle i,k\rangle) + D(P(\langle k,j\rangle^+)) \\ &+ w(\langle k,j\rangle) : k \in V(i) \cap V(j)\} \end{aligned} \tag{4}$$

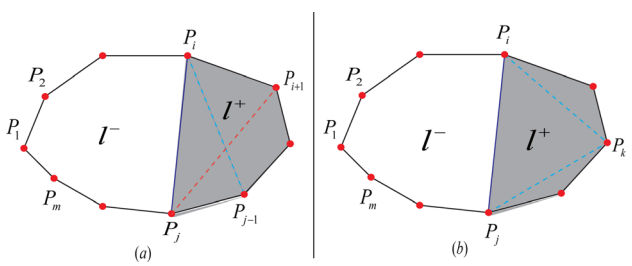


Fig. 3 Two halves $P(l^+)$ and $P(l^-)$ by a rung $\langle i,j\rangle$. (*a*) Two bridge triangles from $\langle i,j\rangle$. (*b*) Visibility vertex P_k of rung $\langle i,j\rangle$.

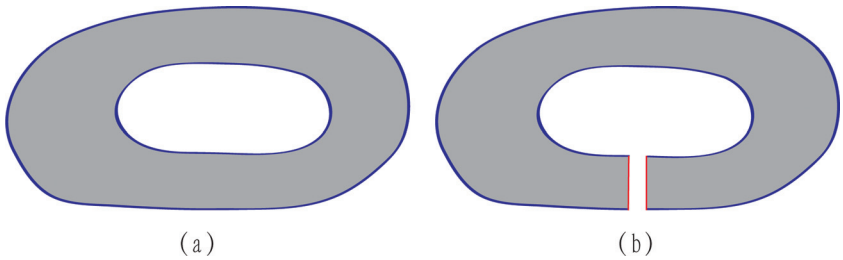


Fig. 4 A multiple connected region (*a*) and a cut (*b*)

It is obvious that the idea of using the L set of an arbitrary vertex still holds. Thus, by applying Eq. (2) to all the rungs in an L set and using rule (4) instead of rule (3) in the recursion, an optimal triangulation can be found.

Finally, for a multiple connected region formed by more than one polyline (see Fig. 4(*a*)), we propose to insert some artificial "cuts" to convert the region into a simple one to which the just prescribed optimal triangulation method can be applied. The main difficulty is how to determine a proper cut. Fortunately, since any ruling of a developable surface lies on a local convex hull of the surface [5], we could seek the cuts according to this property. For any vertex V on the boundaries, let $L(V)$ be the L set of V, and we choose as the cut a rung l from $L(V)$ which lies on its local convex hull, e.g., the red line segment in Fig. 4.

4 Variational Developable Surface Interpolation

In this section, we describe the formulation and optimization method of VDSI in detail.

4.1 Catmull–Rom Splines. Catmull–Rom splines [24] are cubic space interpolating curves passing through given control points. The tangent at each point P_i of a Catmull–Rom spline is related to its previous and next point on the spline, and it is calculated by $\tau(P_{i+1} - P_{i-1})$. Note that the beginning point P_1 and ending point P_m is not well defined. Let $\tau(P_2 - P_1)$ and $\tau(P_m - P_{m-1})$ be their tangent, respectively.

The parameter τ describes the bending when the spline interpolates a point. The larger the parameter τ, the more bending the Catmull–Rom spline at a point (see Fig. 5 for an illustration).

Suppose there are four control points P_{i-2}, P_{i-1}, P_i, and P_{i+1}, they determine a Catmull–Rom spline $R(u)$. In Figs. 5(*a*) and 5(*b*), four points define two Catmull–Rom splines with different τ. Since Catmull–Rom spline is a cubic space curve, the curve is determined by four equations: $R(0) = P_{i-1}$, $R(1) = P_i$, $R'(0) = \tau(P_i - P_{i-2})$, and $R'(1) = \tau(P_{i+1} - P_{i-1})$. Consequently, we could formulate the Catmull–Rom spline by the following matrix form:

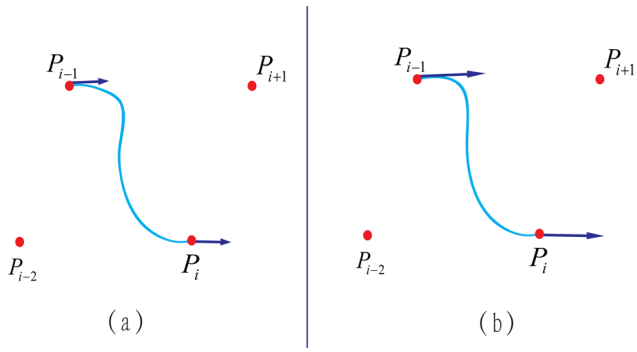


Fig. 5 Two Catmull–Rom splines with $\tau = 0.5$ in (*a*) and $\tau = 1.0$ in (*b*)

$$R(u) = U^T M P,$$

where U and P are two column vectors $U = (1, u, u^2, u^3)^T$, $P = (P_{i-2}, P_{i-1}, P_i, P_{i+1})^T$, while M is a 4×4 matrix,

$$M = \begin{pmatrix} 0 & 1 & 0 & 0 \\ -\tau & 0 & \tau & 0 \\ 2\tau & \tau - 3 & 3 - 2\tau & -\tau \\ -\tau & 2 - \tau & \tau - 2 & \tau \end{pmatrix}$$

Catmull–Rom spline has many useful properties. The spline passes through all control points. The spline is C^1 continuous. It means that the tangent direction and magnitude has no discontinuities.

The tangent at a point is calculated by the previous and next point, which, as we will see in the following context, can make our optimization function be just associated with points on boundary curves and facilitate our calculations. Furthermore the C^1 continuity is enough for our applications. Thus we adopt the Catmull–Rom spline to interpolate the boundary points and calculate tangent vectors.

In addition, there are two considerations for us to favor the Catmull–Rom spline rather than other types of splines such as B-spline. The first one is the computational complexity. Since the Catmull–Rom spline passes through its control points, and the tangent at each control point is easily calculated, thus using this spline can reduce the computation and make the optimization function (Sec. 4.2) more compact. On the contrast, B-splines do not pass through their control points, and we have to rely on inverse calculation to find them, which is not convenient in this work. Crossing phenomenon (see Fig. 6) also stops us from using the B-spline. In Ref. [4], they utilize the B-spline to interpolate boundary points and resample the spline, but crossing is caused. Our method does not cause crossing.

4.2 Variational Developable Surface Interpolation. Interpolation and approximation of curves and surfaces always play a central role in computer aided geometric design. Generally interpolation requires that curves or surfaces pass through all given points strictly, while in the latter this is not strictly required. The VDSI, however, is different from either—we allow some points to vary within a bound while the rest must be strictly interpolated; it is a hybrid of the two.

Let C be a smooth closed boundary curve which is approximated by a closed polygon P. In Fig. 7, the red dashed line is a smooth boundary curve, while its approximation polygon $P = \{P_1, P_2, \ldots, P_m\}$ is represented by the blue vertices. Some of the vertices (specified by user) in P are allowed to move within a certain bound in space, e.g., point P_i in Fig. 7. Our objective is to seek a better (in terms of developability) developable surface interpolating a given boundary curve. Accordingly, the twist

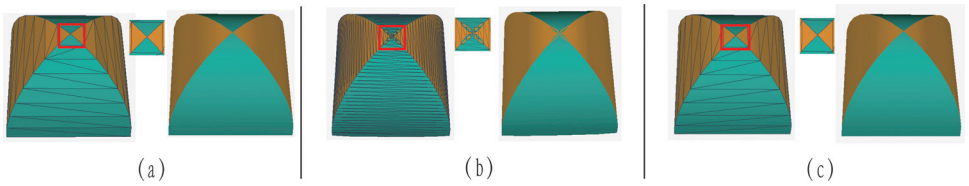


Fig. 6 A tent model is generated by different methods. (*a*) Wang and Tang's result [6]. (*b*) The result of Liu et al. [4]. (*c*) Our result.

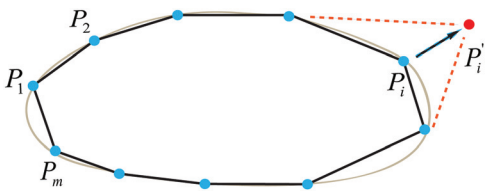


Fig. 7 Modify a polygon boundary by moving its vertices within a bound. Adjusting the vertex P_i finally obtains P_i'.

measurement $w(l)$ of a rung l could be viewed as the function of some points.

Without loss of generality, suppose the varied points of P_1, P_2, $\cdots$, P_n $(n \le m)$ are moved and their respective new positions are points $P_1', P_2', \ldots, P_n'$. For a prescribed bound ε, the optimization function can be formulated as

$$W(T^*) = \text{Min} \sum_{l \in T} w(l) \quad \text{s.t.}$$
$$\sum_{i=1}^{n} \|P_i - P_i'\| \le \varepsilon$$
$$P_j = P_j' \qquad j = n+1, \cdots, m$$

where T is any triangulation of variable point set.

We have given the specific formulation of $w(l)$ in Eq. (1); nevertheless calculations of unit tangents are not dealt with so far. For calculating the unit tangent, many methods take advantage of quadric or cubic polynomials or splines to fit a curve and estimate the tangent. In order to compute the tangent t_i at P_i, for example, a quadric curve $f(t) = a_0 + a_1 t + a_2 t^2$ is used to interpolate three points P_{i-1}, P_i and P_{i+1} such that $f(0) = P_{i-1}, f(1/2) = P_i$ and $f(1) = P_{i+1}$, where a_i is a 3×1 vector. Then the tangent can be easily computed as $t_i = f'(1/2)/\|f'(1/2)\|$. Considering the high non-linearity of the optimization function, if we make use of the above method to compute the tangent, the optimization function will become more complex. It is desirable that the optimization function is just related to points and does not involve other complex computations. According to the early discussion of Catmull–Rom splines, the tangent t_i of a point P_i can be computed by its next point P_{i+1} and previous point P_{i-1}. In light of the definition of Catmull–Rom splines, t_i can thus be formulated as follows:

$$t_i = \frac{P_{i+1} - P_{i-1}}{\|P_{i+1} - P_{i-1}\|}$$

Note that there is no relationship between t_i and τ.

For the rung $l = \langle i, j \rangle$, the following formulation holds if we substitute t_i and t_j in (1):

$$w(l) = 1 - \frac{(P_{i+1} - P_{i-1}) \times (P_j - P_i)}{\|P_{i+1} - P_{i-1}\|\|P_j - P_i\|} \cdot \frac{(P_{j+1} - P_{j-1}) \times (P_i - P_j)}{\|P_{j+1} - P_{j-1}\|\|P_i - P_j\|}$$

where $\times$ represents the cross product. Therefore, the optimization function can be represented as follows:

$$W(T^*) = \text{Min} \sum_{l \in T} \left(1 - \frac{(P_{i+1}' - P_{i-1}') \times (P_j' - P_i')}{\|P_{i+1}' - P_{i-1}'\|\|P_j' - P_i'\|} \cdot \frac{(P_{j+1}' - P_{j-1}') \times (P_i' - P_j')}{\|P_{j+1}' - P_{j-1}'\|\|P_i' - P_j'\|} \right) \quad \text{s.t.}$$
$$\sum_{i=1}^{n} \|P_i - P_i'\| \le \varepsilon$$
$$P_j = P_j' \qquad j = n+1, \ldots, m$$

Obviously the optimization function just relies on points lying on a boundary curve, thus it can be viewed as a function of P_1, $P_2, \ldots, P_m$. Next we will discuss how to solve the complex optimization problem in detail.

4.3 Optimization Method. In this subsection, we propose an iterative scheme to find the optimal solution of VDSI. For a given triangulation T of P, the equality constraints of the optimization problem can be ignored as long as the points P_j' is substituted by $P_j (j = n+1, \ldots, m)$. Thus we consider the following optimization problem:

$$\text{Min} \sum_{l \in T} \left(1 - \frac{(P_{i+1}' - P_{i-1}') \times (P_j' - P_i')}{\|P_{i+1}' - P_{i-1}'\|\|P_j' - P_i'\|} \cdot \frac{(P_{j+1}' - P_{j-1}') \times (P_i' - P_j')}{\|P_{j+1}' - P_{j-1}'\|\|P_i' - P_j'\|} \right)$$
$$\text{s.t.} \qquad \sum_{i=1}^{n} \|P_i - P_i'\| - \varepsilon \le 0$$

Let $P_j = (x_j, y_j, z_j), P_j' = (x_j', y_j', z_j')$, and also assuming $\mathbf{x} = (x_1', y_1', z_1', \ldots, x_n', y_n', z_n')$, thus the above objective function and inequality constraint can be viewed as two multivariate functions $f(\mathbf{x})$ and $g(\mathbf{x}) - \varepsilon$. Then we can solve the problem by using the Zoutendijk's feasible direction method [25]. Given an initial feasible point $\mathbf{x}^{(1)} = (x_1, y_1, z_1, \ldots, x_n, y_n, z_n)$, the iterative procedure can be formulated as follows:

Step 1: Solve the linear programming problem

$$\begin{aligned} &\text{Min} \quad z \\ &\text{s.t.} \quad \nabla^T f(\mathbf{x}^{(k)}) \cdot \mathbf{d} - z \le 0, \\ &\qquad \nabla^T g(\mathbf{x}^{(k)}) \cdot \mathbf{d} - z \le 0, \\ &\qquad -1 \le d_i \le 1, \quad i = 1, \ldots, 3n \end{aligned}$$

Label the solution $\mathbf{d}^{(k)}$ and $z^{(k)}$. If $z^{(k)} \le 0$, the iteration terminates, since no further improvement is possible.

Step 2: Find a $\alpha_{\max} > 0$, s.t.

$$\alpha_{\max} = \text{Max}\{\alpha : g(\mathbf{x}^{(k)} + \alpha \cdot \mathbf{d}^{(k)}) - \varepsilon \le 0\}$$

If $\alpha_{\max} \le 0$, set $\alpha_{\max} = \infty$.

Table 1 Algorithm of Solving VDSI

Input: $T^{(1)}$, $\varepsilon' \in R$
Output: A triangulation T^* of P

1. For a triangulation $T^{(k)}$, solve the optimization problem through Step 1 to Step 4, and obtain a new point set P'
2. Triangulate P' using the Dynamic programming method, and denote the new triangulation $T^{(k+1)}$
3. If $|W(T^{k+1}) - W(T^k)| \le \epsilon'$, the iteration is terminated, and otherwise go to 1

Step 3: Solve the 1-D search problem

$$\begin{aligned} \text{Min} \quad & f(\mathbf{x}^{(k)} + \alpha \mathbf{d}^{(k)}) \\ \text{s.t.} \quad & 0 \le \alpha \le \alpha_{\max} \end{aligned}$$

Label the solution $\alpha^{(k)}$.
Step 4: Set $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha^{(k)}\mathbf{d}^{(k)}$, and then go to Step 1.

In the above algorithm, $f(\mathbf{x})$ is dependent on the triangulation T, thus the x_i' component of $\nabla f(\mathbf{x})$ can be written as $(\partial f/\partial x_i') = \sum_{l \in T} \sum_{P_i' \in l} (df/dx_i')$, where $P_i' \in l$ means P_i' is an endpoint of l. The y_i' and z_i' component of $\nabla f(\mathbf{x})$ has a similar representation. In this implementation, we take $g(x) = \sum_{i=1}^{n} \|P_i - Pi'\|^2$. Assuming P is a point set with counter-clockwise order, and assuming $T^{(1)}$ be a triangulation of P constructed by the dynamic programming method introduced in Sec. 3, we give the algorithm of solving VDSI in Table 1. In fact, the algorithm uses a two-step iteration. First by fixing a triangulation, a new point set is obtained by solving an optimization problem; then the new point set is triangulated by dynamic programming. The procedure proceeds until the prescribed accuracy is satisfied.

5 Quasi-Coons Surface Approximation With G^1 Continuity

In this section, we present a fairly simple and significant adding point rule which can guarantee the G^1 continuity when we approximate a quadrilateral strip by using a quasi-Coons surface. Noticing we use the algorithm proposed by Liu et al. [4] to convert a triangle strip into a quadrilateral strip.

Taking a quadrilateral strip in Fig. 8 for example, black points represent the added points, and the red points are the original points. Two boundary polylines are approximated by two (blue) curves $L_1(t)$, $L_2(t)$. Exactly, $L_1(t)$ ($L_2(t)$) consists of four Bézier curves $L_{1j}(t)$ ($L_{2j}(t)$) ($j = 1, 2, 3, 4$).

In this paper, we approximate each quadrilateral by a quasi-Coons patch, and all quasi-Coons patches form a quasi-Coons surface which approximates the quadrilateral strip with G^1 continuity. Assuming $t, u \in [0, 1]$, we now give the definition of the quasi-Coons patch as follows:

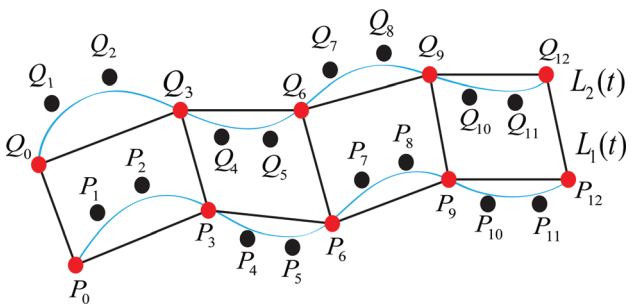


Fig. 8 The diagram of adding point rule

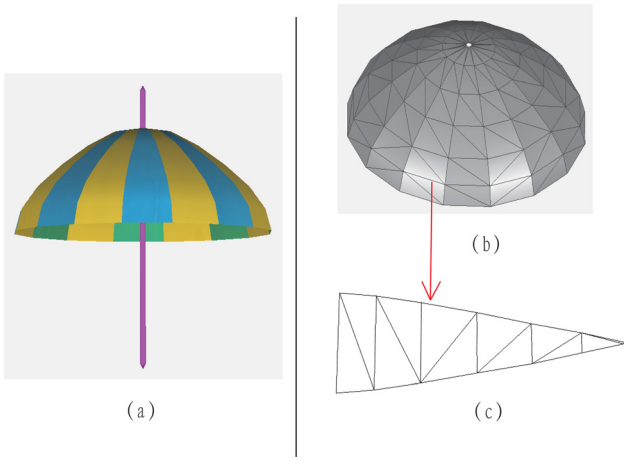


Fig. 9 (*a*) An umbrella model. (*b*) Wire umbrella model. (*c*) Unfolded planar mesh of an optimal triangulation.

$$S_j(t,u) = (1-u)L_{1j}(t) + uL_{2j}(t), \quad j = 1,2,3,4$$

Along the common curve $S_1(1,u) = S_2(0,u)$, two quasi-Coons patches $S_1(t,u), S_2(t,u)$ is G^1 if the following condition holds:

$$\frac{\partial S_2}{\partial t}(0,u) = \lambda(u)\frac{\partial S_1}{\partial t}(1,u) \tag{5}$$

According to $L_1(t)$ and $L_2(t)$, $S_1(t, u)$ and $S_2(t, u)$ are reformulated, then

$$(1-u)L_{12}'(0) + uL_{22}'(0) = \lambda(u)[(1-u)L_{11}'(1) + uL_{21}'(1)]$$

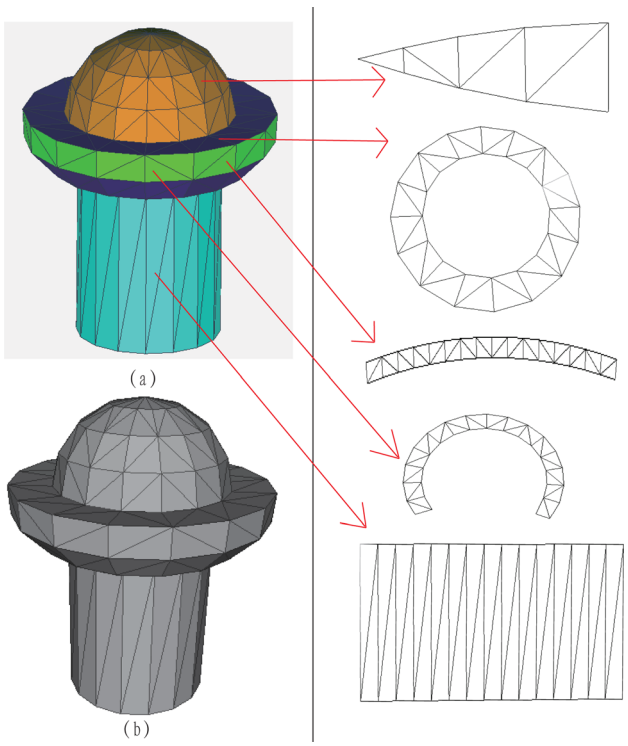


Fig. 10 (*a*) A tower model is made up of five optimal triangulation strips. (*b*) The solid model of the tower. Right column: unfolded planar meshes of several optimal triangulation strips.

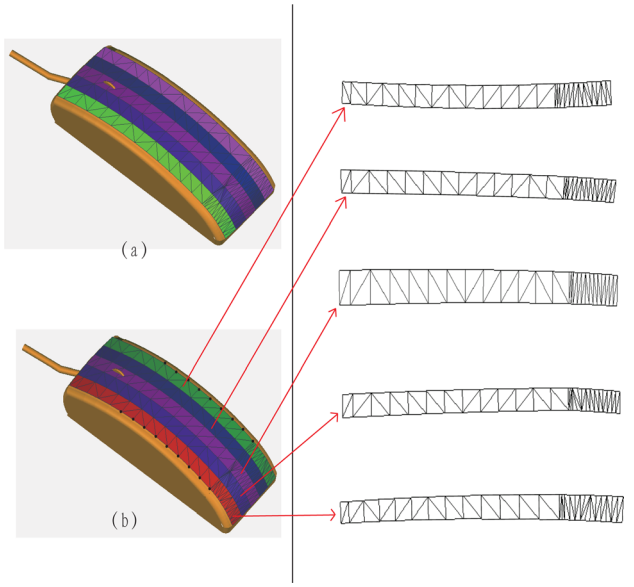


Fig. 11 (*a*) The top surface of a mouse is made up of five optimal triangulation strips. (*b*) The variational developable surface interpolation, and the black points represent the varying points. Right column: unfolded planar meshes.

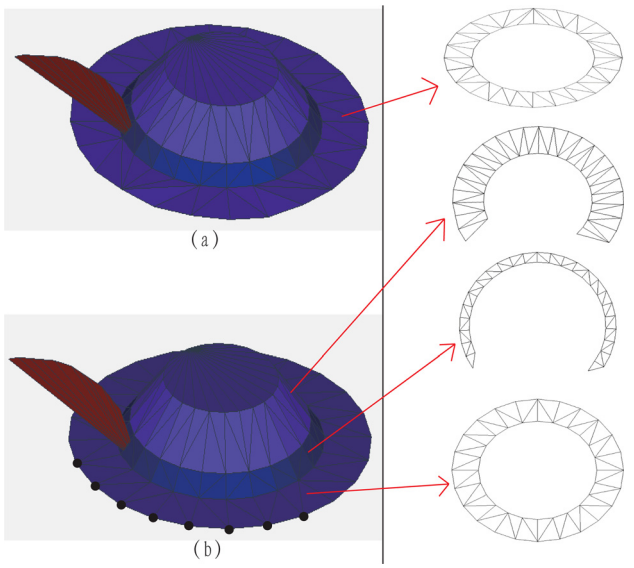


Fig. 12 (*a*) A hat model consists of three optimal triangulation strips (except the feather model and top surface). (*b*) The variational developable surface interpolation, and the black points represent the varying points. Right column: unfolded planar meshes.

Furthermore, we have

$$(1-u)(P_4-P_3)+u(Q_4-Q_3) = \lambda(u)[(1-u)(P_3-P_2)+u(Q_3-Q_2)] \quad (6)$$

Let $u=0$ and $u=1$, respectively, and denote $\lambda_1=\lambda(0)$ and $\lambda_2=\lambda(1)$, then

$$P_4-P_3=\lambda_1(P_3-P_2), \quad Q_4-Q_3=\lambda_2(Q_3-Q_2)$$

Equation (6) is reformulated and arranged according to above two equalities, then

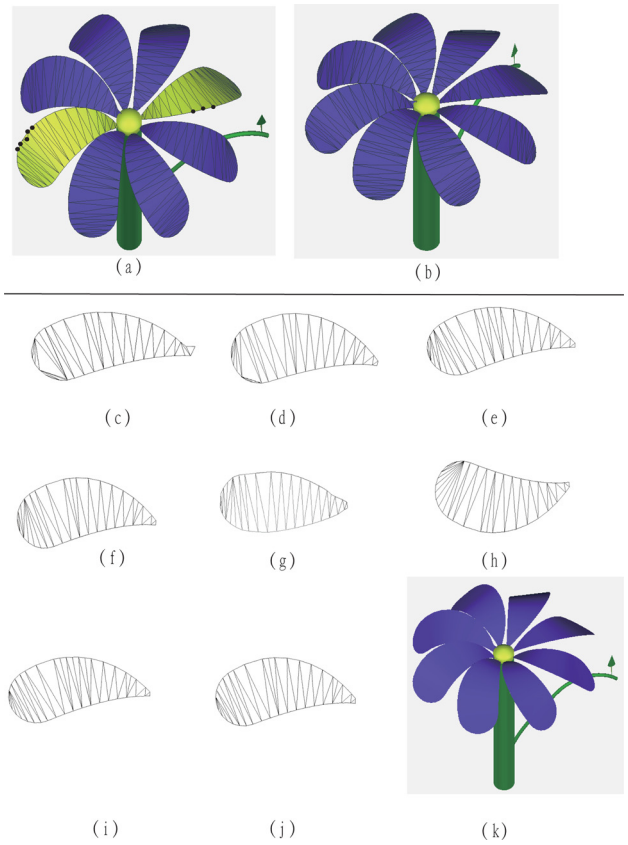


Fig. 13 (*a*) A flower model. (*b*) The flower model after varying several points. (*c*)–(*j*) Unfolded planar meshes corresponding to the optimal triangulation strips in (*b*). (*k*) The solid flower model.

$$(1-u)[\lambda_1-\lambda(u)](P_3-P_2)=u[\lambda(u)-\lambda_2](Q_3-Q_2)$$

Obviously P_3–P_2 is proportional to Q_3–Q_2, and denote $Q_3-Q_2=\alpha(P_3-P_2)$. When $P_3-P_2\neq 0$ and $Q_3-Q_2\neq 0$, $\lambda(u)$ can be easily computed.

$$\lambda(u)=\frac{(1-u)\lambda_1+\alpha u\lambda_2}{(\alpha-1)u+1}$$

According to the definition of the quasi-Coons patch and the derivation above, we achieve an important result to guarantee the G^1 continuity.

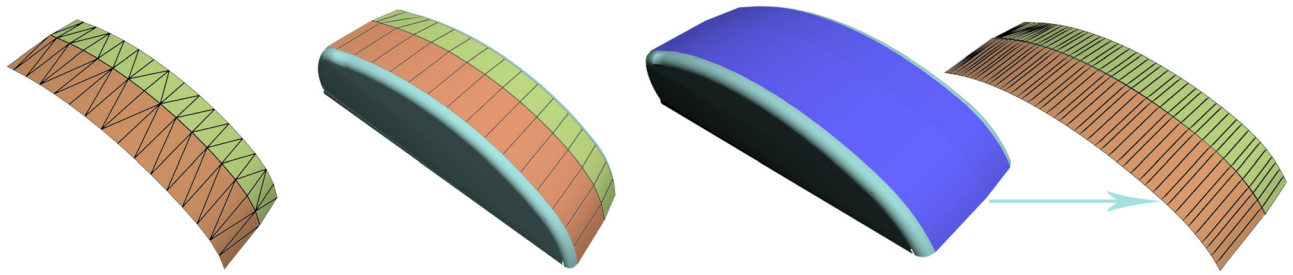
Property 1. The quasi-Coons surface, which approximates a quadrilateral strip and is formed by an array of quasi-Coons patches, is G^1 if the added points (black points in Fig. 8) satisfy one of the following two rules of adding points:

$$P_{i+1}-P_i=\lambda(P_i-P_{i-1}) \quad Q_{i+1}-Q_i=\lambda(_i-Q_{i-1})$$

or

$$P_{i+1}-P_i=\lambda_1(P_i-P_{i-1})$$
$$Q_{i+1}-Q_i=\lambda_2(Q_i-Q_{i-1})$$
$$Q_i-Q_{i-1}=\alpha(P_i-P_{i-1})$$

where $\lambda_1,\lambda_2>0$.



(a) (b) (c)

Fig. 14 (*a*) The top surface of a mouse is made up of two optimal triangulation strips. (*b*) Two strips in (*a*) are converted into two quadrilateral strips. (*c*) The top surface is approximated by two G^1 continuous quasi-Coons surfaces.

6 Experiments and Applications

We have implemented the algorithms proposed in this paper using C++. In all the test examples, we take the terminating accuracy of $\varepsilon' = 10^{-6}$. In this section, we first model two 3D geometric objects using several optimal triangulation strips constructed by the dynamic programming method. Then the VDSI is verified on some models, and the results show that our optimization method is valid.

For a given set of boundary polylines, different methods would generate different results. In Fig. 6, we compare three results generated by two methods in [4,6], respectively, and our method. Obviously on the top part of the tent model, the method by Liu et al. (Fig. 6(*b*)) appears crossing but Wang and Tang (Fig. 6(*a*)) and ours (Fig. 6(*c*)) does not.

As shown in Fig. 9, we construct an umbrella model (except the handle) using the proposed dynamic programming algorithm. The umbrella consists of 16 optimal triangulation strips. Figures 9(*a*) and 9(*b*) are the solid and wire models, respectively, and Fig. 9(*c*) is the unfolded planar mesh of an optimal triangulation strip. We do not display all unfolded planar meshes of strips since all strips have similar triangulations.

We also generate a tower model using several optimal triangulation strips as shown in Fig. 10. The tower model consists of five parts, and especially the semi-sphere on the top of the tower has 16 optimal triangulation strips. In the right column of Fig. 10, some unfolded planar meshes corresponding to optimal triangulation strips are illustrated.

To obtain a better developability of a geometric object, we apply the VDSI model. Notice that the varying points in VDSI should not be shared by two or more triangular strips, since otherwise the final model may have some gaps.

We model a top surface of a mouse by using five optimal triangulation strips (Fig. 11(*a*)). In order to obtain a better developability, we prescribe some varying points (black points in Fig. 11(*b*)) and take advantage of the optimization method in Sec. 4, and finally the top surface with better developability is achieved (Fig. 11(*b*)). Notice that we take $\varepsilon = 10.5$ in this case. The right column of Fig. 11 shows unfolded planar meshes corresponding to the optimal triangulation strips in Fig. 11(*b*).

In Fig. 12, a hat model is made up of five triangular strips, and three of the strips are optimal triangulation strips (except the feather and top surface). In Fig. 12(*b*), the black points indicate the varying points of VDSI, and the optimal triangulation in Fig. 12(*b*) has a better developability and an excellent appearance compared with those in Fig. 12(*a*). The right column in this figure illustrates the corresponding unfolded planar meshes. In addition, we take the varying bound $\varepsilon = 16.5$ in the optimization method.

We construct a flower shown in Fig. 13 which has two parts: rachis and petals. Each petal of the flower model is an optimal triangulation. The yellow petals in Fig. 13(*a*) are performed with the VDSI optimization method for the indicated black points, and we take $\varepsilon = 10$ (for the front yellow petal) and $\varepsilon = 5$ (for the back yellow petal). Figures 13(*c*)–13(*j*) are unfolded planar meshes corresponding to strips in Fig. 13(*b*). A solid model of the flower is presented in Fig. 13(*k*).

Table 2 Statistic data of three models

Model	♯tri	♯str	D(T^1)	D(T^*)	Time (s)
Mouse	260	5	0.3069452	0.3050914	8.5
Hat	144	3	0.84143407	0.8396112	3.7
Flower	365	8	0.7865463	0.7822923	7.2

In Fig. 14, two optimal triangulation strips (Fig. 14(*a*)) are converted into corresponding quadrilateral strips (Fig. 14(*b*)), then we construct two G^1 continuous quasi-Coons surfaces to approximate the quad strips according to our adding point rule in Sec. 5. Note that two Bézier boundary curves are sampled (original vertices are preserved) with the same parameter setting. And we connect the samples with the same parameter. The final result is shown in Fig. 14(*c*).

Finally, the statistic data of models in Figs. 11–13 are summarized in Table 2. In the table ♯tri and ♯str represent the number of triangles and strips, respectively. D(T^1) means the initial total twist of the triangulation generated by our dynamic programming-based optimization solution, and D(T^*) means the total twist after performing the VDSI. Once the boundary polylines are specified, the models in all the examples are generated in a few seconds, on a Core2Duo 1.8 GHz laptop computer with 2GB memory.

These examples demonstrate the feasibility and practicability of our optimization method combining dynamic programming for VDSI. Furthermore, they show the vast applications of developable surfaces.

7 Conclusions

In this paper, we investigate the construction problem on VDSI which so far has no literatures to address this problem. Observing the recursive property of boundary triangulation, the dynamic programming method is applied to triangulate boundaries. In this paper, an improved dynamic programming method is designed to guarantee the global optimization and has a low complexity of running time. With the usage of dynamic programming, a global optimization solution to VDSI using an iterative scheme is proposed. We do not prove the convergence of our method, but all calculations in this paper are convergent. At last, we present a method to approximate a discrete quadrilateral strip with G^1 continuous quasi-Coons surface. Diverse examples are presented to demonstrate the usefulness of the proposed method.

Acknowledgment

This work was supported in part by Natural Science Foundation of China (Grant Nos. 61272228, 61322206, and 61373003), and National High-tech R&D Program of China (863 Program) (Grant

No. 2012AA011801). The work of Y. J. Liu was supported in part by the Program for NCET-11-0273.

References

[1] Carmo, M. P. D., 1976, *Differential Geometry of Curves and Surfaces*, Vol. 1, Prentice-Hall, Englewood Cliffs, NJ.

[2] Frey, W. H., 2004, "Modeling Buckled Developable Surfaces by Triangulation," Comput.-Aided Des., **36**(4), pp. 299–313.

[3] Liu, Y. J., Tang, K., and Joneja, A., 2007, "Modeling Dynamic Developable Meshes by the Hamilton Principle," Comput.-Aided Des., **39**(9), pp. 719–731.

[4] Liu, Y. J., Tang, K., Gong, W. Y., and Wu, T. R., 2011, "Industrial Design Using Interpolatory Discrete Developable Surfaces," Comput.-Aided Des., **43**(9), pp. 1089–1098.

[5] Rose, K., Sheffer, A., Wither, J., Cani, M. P., and Thibert, B., 2007, "Developable Surfaces from Arbitrary Sketched Boundaries," Proceedings of the Fifth Eurographics Symposium on Geometry Processing, pp. 163–172.

[6] Wang, C. C. L., and Tang, K., 2005, "Optimal Boundary Triangulations of an Interpolating Ruled Surface," ASME J. Comput. Inf. Sci. Eng., **5**(4), pp. 291–301.

[7] Zeng, L., Liu, Y. J., Chen, M., and Yuen, M. F., 2012, "Least Squares Quasi-Developable Mesh Approximation," Comput. Aided Geom. Des., **29**(7), pp. 565–578.

[8] Shelden, D. R., 2002, "Digital Surface Representation and the Constructability of Gehry's Architecture," PhD thesis, Massachusetts Institute of Technology, Cambridge, MA.

[9] Liu, Y. J., Zhang, D. L., and Yuen, M. M., 2010, "A Survey on CAD Methods in 3D Garment Design," Comput. Ind., **61**(6), pp. 576–593.

[10] Tang, K., and Wang, C. C. L., 2005, "Modeling Developable Folds on a Strip," J. Comput. Inf. Sci. Eng., **5**(1), pp. 35–47.

[11] Pottmann, H., and Wallner, J., 2010, *Computational Line Geometry*, Springer Verlag, Berlin, Germany.

[12] Frey, W. H., 2002, "Boundary Triangulations Approximating Developable Surfaces that Interpolate a Closed Space Curve," Int. J. Found. Comput. Sci., **13**(2), pp. 285–302.

[13] Gurunathan, B., and Dhande, S. G., 1987, "Algorithms for Development of Certain Classes of Ruled Surfaces," Comput. Graphics, **11**(2), pp. 105–112.

[14] Weiss, G., and Furtner, P., 1988, "Computer-Aided Treatment of Developable Surfaces," Comput. Graphics, **12**(1), pp. 39–51.

[15] Chu, C. H., and Séquin, C. H., 2002, "Developable Bezier Patches: Properties and Design," Comput.-Aided Des., **34**(7), pp. 511–527.

[16] Pottmann, H., and Farin, G., 1995, "Developable Rational Bezier and B-spline Surfaces," Comput. Aided Geom. Des., **12**(5), pp. 513–531.

[17] Bodduluri, R. M. C., and Ravani, B., 1993, "Design of Developable Surfaces Using Duality Between Point and Plane Geometries," Comput. Aided Des., **25**(10), pp. 621–632.

[18] Park, F. C., Yu, J., Chun, C., and Ravani, B., 2002, "Design of Developable Surfaces Using Optimal Control," ASME J. Mech. Des., **124**(4), pp. 602–608.

[19] Aumann, G., 2003, "A Simple Algorithm for Designing Developable Bézier Surfaces," Comput. Aided Geom. Des., **20**(8–9), pp. 601–619.

[20] Robson, C., Maharik, R., Sheffer, A., and Carr, N., 2011, "Context-Aware Garment Modeling from Sketches," Comput. Graphics, **35**(3), pp. 604–613.

[21] Wang, C. C. L., 2008, "Flattenable Mesh Surface Fitting on Boundary Curves," ASME J. Comput. Inf. Sci. Eng., **8**(2), p. 021006.

[22] Wang, C. C. L., 2008, "Towards Flattenable Mesh Surfaces," Comput.-Aided Des., **40**(1), pp. 109–122.

[23] Liu, Y. J., Lai, Y. K., and Hu, S. M., 2009, "Stripification of Free-Form Surfaces with Global Error Bounds for Developable Approximation," IEEE Trans. Autom. Sci. Eng., **6**(4), pp. 700–709.

[24] Catmull, E., and Rom, R., 1974, "A Class of Local Interpolating Splines," Comput. Aided Geom. Des., R. E. Barnhill and R. F. Reisenfeld, eds. Academic Press, New York, **74**, pp. 317–326.

[25] Zoutendijk, G., 1960, *Methods of Feasible Directions: A Study in Linear and Non-linear Programming*, Elsevier Publishing Co., Amsterdam.