Achieving developability of a polygonal surface by minimum deformation: a study of global and local optimization approaches

Charlie C. L. Wang*

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

Shatin, N.T., Hong Kong

E-mail: cwang@acae.cuhk.edu.hk; Tel: (852) 2609-8052

Kai Tang

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

Clear Water Bay, N.T., Hong Kong

E-mail: mektang@ust.hk, Tel: (852) 2358-8656

Abstract

Surface developability is required in a variety of applications in product design, such as clothing, ship hulls, automobile parts, etc. However, most current geometric modeling systems using polygonal surfaces ignore this important intrinsic geometric property. This paper investigates the problem of how to minimally deform a polygonal surface to attain developability, or the so called developability-by-deformation problem. In our study, this problem is first formulated as a global constrained optimization problem, and a penalty function based numerical solution is proposed for solving this global optimization problem. Next, as an alternative to the global optimization approach which usually requires lengthy computing time, we present an iterative solution based on a local optimization criterion which achieves near real-time computing speed. Both approaches preserve the topology and continuity of the original polygonal surface in the case when more than one individual polygonal patches comprise the surface. Experimental examples are provided to demonstrate the functionality of the proposed two approaches as well as their comparison in terms of computing cost, effectiveness of attaining developability, dimensional difference between the surfaces before and after the optimization, and other important aspects.

Keywords: developable surface, polygonal mesh, assembled patches, deformation, optimization.

* Corresponding author

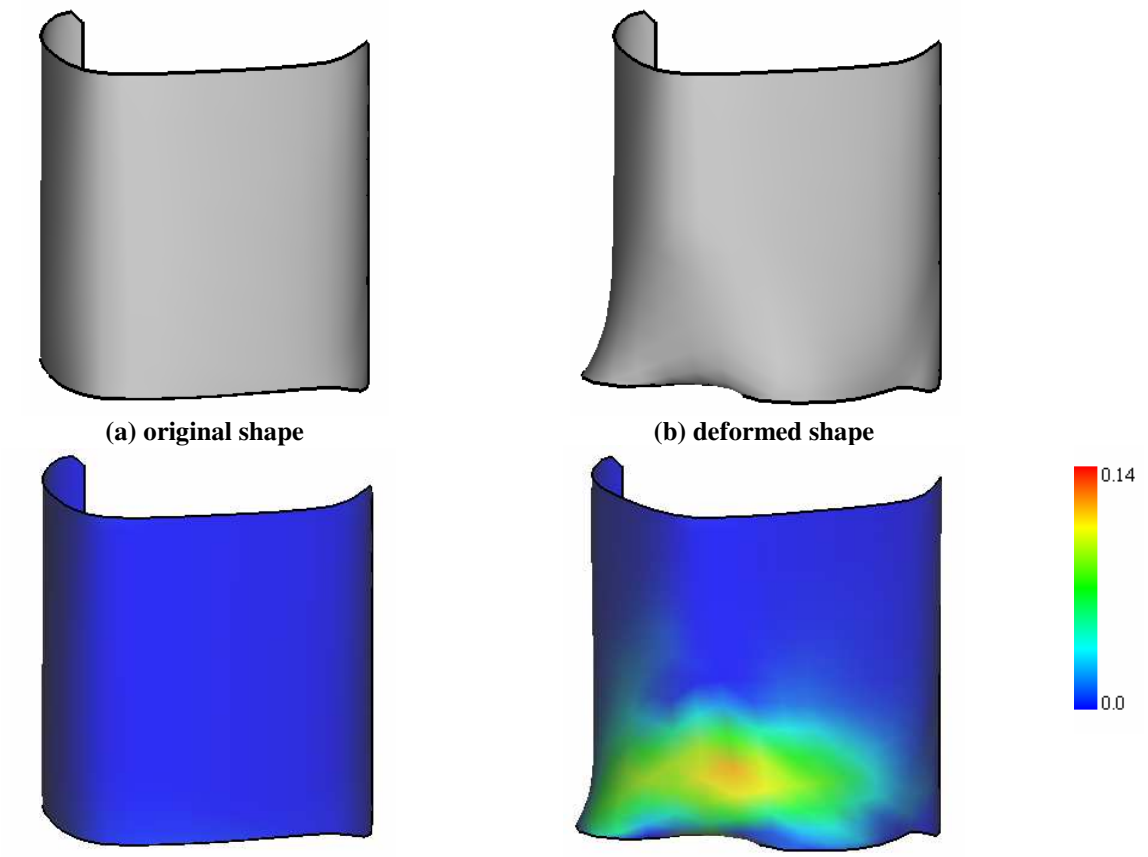
1. Introduction

Developability is an important intrinsic property of a surface. Informally, a surface is developable if it can be flattened onto a plane without any distortion [1]. This is a highly desired property in sheet manufacturing industry, where the stretch or compression in the sheet material should be avoided, as they make the product more prone to damage since internal strains and stresses are generated. As an example shown in Fig. 1a, the original design of the shell has a developable shape which can be bent or rolled by a metal sheet. After deformed by Wires [2], its shape becomes the one as shown in Fig. 1b, which is non-developable. The elastic energy maps of the shell surface before and after the deformation are given in Fig. 1c and Fig. 1d respectively. As clearly shown, a great amount of elastic energy is generated if the newly designed shell is to be manufactured by metal sheet. This requirement exists in many applications (e.g., clothing, ship hulls, ducts, shoes, aircraft and automobile parts). In this paper, we investigate and propose algorithms for solving the problem of how to deform a non-developable surface, in the form of assembled polygonal mesh patches, into a developable one while at the same time minimizing the difference between the two surfaces.

Our algorithms work on polygonal mesh patches which have become a widely accepted standard in most computer graphics applications. Triangular meshes are especially preferred due to their algorithmic simplicity, numerical robustness, and efficient display. The advantage of switching from spline-based surface representation to mesh representation is mainly due to the fact that algorithms for polygonal meshes usually can work on shapes with arbitrary topology and do not suffer from the severe restrictions which stem from the rigid algebraic structure of polynomial surfaces. More and more commercial modeling systems have included the polygonal mesh based module, and more and more applications are developed based on mesh representation.

The proposed technique is new – no prior research on developability optimization of polygonal surfaces has been found in literature. In our approach, the surface developability problem is formulated as a constrained optimization problem. The problem is first solved numerically by a penalty function based optimization scheme, which is a global approach. The continuity is preserved between the assembled patches. The global optimization is very time-consuming even after the gradients of the objective function have already been calculated locally. Therefore, as an alternative, we further present a local optimization solution in which the vertices on the surface are moved along their normal directions iteratively. The magnitude of each movement is actually derived from a locally defined objective function. This local approach enjoys a great advantage of faster computing speed as compared to its global counter-part and can be integrated into modeling systems to preserve the developability of assembled surface patches in real-time during the entire design process. Different from most existing surface

modeling solutions concerned with developability, our solutions, both global and local approaches, are more of bottom-up nature – we take as input an arbitrary (non-developable) surface in the form of a set of assembled polygonal mesh patches and output a *developable* polygonal mesh that deviates minimally from the original surface.



(a) original shape **(b) deformed shape**

(c) elastic energy map of the original duct **(d) elastic energy map of the newly shaped duct**

Fig. 1 Example I – a deformed shell leads to stretch in manufacturing

The paper is organized as follows. After reviewing the related work, the necessary mathematical formulations about the developability of a polygonal mesh are given in section 3, where the developability-by-deformation problem is formulated as a constrained optimization problem. In section 4, the details of a penalty-function-based solution are presented that numerically solves this constrained optimization problem. To overcome the usually lengthy computing time required by the proposed numerical solution, as an alternative, in section 5 we reformulate the problem as a local optimization problem and propose a much quicker numerical algorithm to solve the local optimization problem. In section 6 we then provide some experimental examples to illustrate the functionality of the proposed solutions as well as their comparison. Finally, the paper is concluded in section 7 and we offer some pointers to potential future research in this area.

2. Related Work

Over the past decade, mesh-processing techniques, such as mesh simplification [3-5] and mesh fairing [6-10], have been improved significantly. Apart from fundamental mesh processing algorithms, many new freeform modeling approaches have also been developed. The SKETCH system [11] rapidly constructs an approximate shape via direct mark based interaction. The Teddy system [12] constructs a rounded freeform mesh model by finding the chordal-axis of the user input 2D closed stroke to build a smooth surface around the axis. Other approaches construct mesh surfaces by use of implicit surfaces [13, 14]. Suzuki et al. [15] presented a 3D mesh-dragging method for intuitive, efficient geometric modeling of free-form polygonal models; this method is based on an adaptive remeshing procedure. With their method, the user can drag a part of a triangular mesh and change its position and orientation. Other interactive modeling research results were reported for the multi-resolution presentation of models; for example, Zorin et al. [16] built a scalable interactive multi-resolution editing system based on mesh refinement and coarsification algorithms, and based on Zorin's approach Khodakovsky and Schroder [17] developed an algorithm that can modify the fine level shape of a surface. However, in all the above approaches, the developability of the processed polygonal mesh surface is not considered. Our paper considers the developability property of the given polygonal surface and converts the original non-developable surface into a developable one.

Developable surfaces have been studied for a long time. The definition of a developable surface (cf. [1]) is derived from a ruled surface: for a ruled surface, $X(t,v)=\alpha(t)+v\beta(t)$, it is developable if β, $\frac{d\beta}{dt}$ and $\frac{d\alpha}{dt}$ are coplanar for all points on X (where $\alpha(t)$ is the *base curve* and $\beta(t)$ is the *director curve* of $X(t,v)$). The simplest examples of developable surfaces are cylinders and cones, and a simple and representative example of non-developable surfaces is a sphere. Every surface enveloped by a one-parameter family of planes is a developable surface. The key concept in characterizing the developability is Gaussian curvature which is the product of the maximum and minimum normal curvatures at a given point [1]. In general, a surface is developable if and only if the *Gaussian curvature* of every point on it is zero – this is the constraint that we want to preserve during the surface optimization. Research related to *Computer Aided Geometric Design*, in particular those concerning the design and approximation of developable surfaces, can be found in [18-27]. Most of them are in terms of NURBS or its special case – B-spline or Bézier surfaces [18-24]. Aumann [18] proposed the condition under which a developable Bézier surface can be constructed with two boundary curves. The boundary curves in his approach are restricted to lie in parallel planes; the projection of the boundary curves on the x-y plane must be a rectangle. Chalfant and Maekawa [19] presented a method to design developable B-

spline surfaces where boundary curves do not necessarily lie in parallel planes. In the work of Frey and Bindschadler [20], the results of Aumann are extended by generalizing the degree of the directions. Their system requires solving non-linear system equations to find the Bézier control points. Chu and Séquin [21] recently proposed a new method to design a developable Bézier patch. In their method, after one boundary curve is freely specified, five more degrees of free are available for a second boundary curve of the same degree. In the work of [22-24], the approximation methods are used to design developable B-Spline surfaces based on projective geometry. Other approaches are based on alternative perspective: Redont [25] constructs developable surfaces by specifying tangent planes along a geodesic of a surface, Randrup [26] approximates a given surface by cylinders in its Gaussian image, and Park et al. [27] design developable surfaces by the methods from optimal control theory.

All the work in the above references tried to use developable surfaces to (approximately) construct the shape of a product. There are also some surface flattening approaches [28-37] in literature. They usually adopt nonlinear programming techniques to find an optimized flattened result with respect to the given 3D surface. Shimada and Tada [28] presented a generic surface development algorithm. This algorithm is based on a meshed surface. In their algorithm, a dynamic programming method is used to develop a curved surface. An objective curved surface is decomposed into regions of adjacent strips. Then each region is developed, in turn, into a flattened shape. The whole shape is derived by solving a multi-stage decision process. Parida and Mudur [29] gave an algorithm to develop complex surfaces. Their algorithm first obtains an approximate planar surface, and then reorients cracks and overlapping parts in the developed plane to satisfy orientation constraints. The algorithm of Parida and Mudur might generate many cracks and calculation errors. McCartney et al. [30] flatten a triangulated surface by minimizing the strain energy in the 2D pattern. The surface is first triangulated using Delaunay triangulation. Then the triangles are transformed onto a 2D plane. However, there are some flattened triangles that cannot preserve their length relationship with respect to the triangles on the surface. This length differences are measured as strain energy. If the strain energy is zero, that means the flattened triangle preserve their length relationships with the original triangles on the surface, i.e. no deformation occurs. Thus, iterative method is applied to minimize this strain energy in the 2D pattern. The endpoints of the triangles are moved in orthogonal directions by trial to obtain smaller energy in each iteration. Wang et al. [31] improve McCartney's algorithm by using a spring-mass system. This guides the endpoints to approach better positions by the force of springs and the computational speed of the minimization is improved. The accuracy of the flattening can also be controlled by using the spring constant. Sheffer and de Sturler [33, 34] presented a texture mapping algorithm

that causes small mapping distortion. Their algorithm consists of two steps: 1) using the Angle Based Flattening (ABF) parameterization method to provide a continuous (no foldovers) mapping, which concentrates on minimizing the angular distortion of the mapping so leads to relatively large linear distortion; 2) to reduce the linear distortion, an inverse mapping from the plane to the result of ABF is computed to improve the parameterization – the improved result has low length distortion. The methods presented in [38, 39] handle the problem in a reverse way by fitting a 2D patch onto a 3D surface. However, even if an optimized flattened 2D shape is obtained, warping a sheet of such a 2D shape into the given 3D shape usually leads to stretching if the given surface itself is non-developable. Therefore, the essential solution is to let the surface itself be developable.

As alluded earlier, we propose to convert the Gaussian curvature of every point on the assembled mesh patches to zero during an optimization process. However, since differential geometry analyzes surfaces in the continuum domain, the traditional equations for calculating the Gaussian curvature cannot be applied to a mesh surface directly. A discrete Gaussian curvature computing method is needed. After Calladine (1984) firstly formulated the discrete Gaussian curvature in [40], Kobbelt et al. [41] gave the formulas of discrete Gaussian curvature based on the fact that a mesh can be interpreted as an approximation of a smooth surface. The idea in [41] is to discretize the formulation for defining the Gaussian curvature on a smooth surface based on a theorem by Rodrigues [1]. In a similar way, Sheffer [42] gave another discrete Gaussian curvature approximation, which is scale independent. In our approach, we utilize the formula of Kobbelt et al. [41] to derive the developability of a polygonal mesh surface.

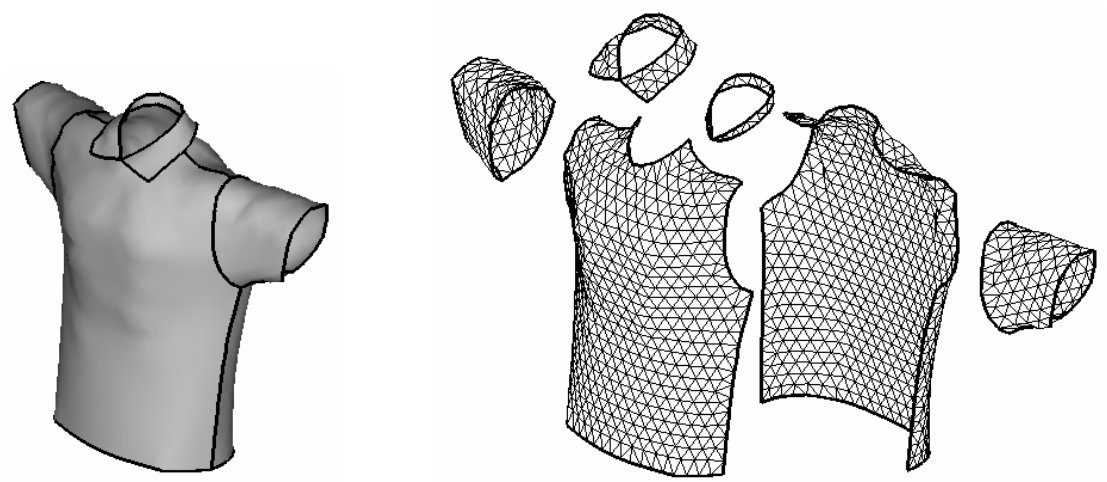
3. Mathematical Formulation

This section gives the necessary mathematical formulation based on which our optimization algorithms will be developed.

3.1 Representation of assembled polygonal patches

A polygonal patch M is defined as a pair (K,V), where K is a simplicial complex specifying the connectivity of the vertices, edges, and faces (in other words, the topological graph of M), and $V=\{v_1,\cdots,v_m\}$ is the set of vertices defining the shape of the polyhedral patch in $\Re^3$. The above definition follows the notation in [43]. In this paper, to simplify the algorithm, every polygonal face in M is subdivided into triangles by the *Constrained Delaunay Triangulation* (CDT) [44] of a planar contour. If the contour of a polygonal face is not coplanar, we project the vertices of this face onto its least-square plane to apply the CDT. No new vertex is inserted; and then the triangulation on the vertices before projection can be obtained by

maintaining the same connectivity of CDT result on the least-square plane. From K, it is straightforward for our algorithm to fetch the adjacent nodes, edges, and faces of a triangular node in constant time. The object considered in our approach is denoted by O which is a collection of assembled polygonal patches M_i, i.e., $O = M_1 \cup M_2 \cup \cdots \cup M_m$. Each surface patch M_i is a two-manifold in the form of a piecewise linear triangular mesh. The given polygonal patches are usually assembled together by sharing some common triangular edges (as illustrated in Fig. 2). In the following, the developability of a polygonal patch is first studied locally, and then its global developability function is defined.



(a) assembled surface patches **(b) mesh representation of each patch**

Fig. 2 Assembled polygonal surface patches

3.2 Developability of a polygonal mesh patch

By theorems from differential geometry, one can easily detect whether a surface is developable according to its overall Gaussian curvature [1] – "the Gaussian curvature of a developable surface is identically *zero* at every regular point". However, Gaussian curvature is not well defined mathematically on a piecewise linear polygonal mesh surface. Thus, the following proposition is needed for this purpose.

Proposition 3-1 At any internal point of a developable piecewise linear surface, the summed inner angle is identically 2π.

Proof. For a point q_i on a developable piecewise linear surface patch M, if θ_j is an inner angle adjacent to q_i before flattening and θ_j^F is the corresponding inner angle flattened on the 2D plane, as illustrated in Fig. 3, the inner angles satisfy $\theta_j = \theta_j^F$ since the surface at this point can be flattened without stretching or

overlapping. In the 2D plane, $\sum_j \theta_j^F$ equals 2π for an internal vertex. When M is developable, which demands $\theta_j = \theta_j^F$ at every point on M, we have $\sum_j \theta_j = 2\pi$.

The approximation Gaussian curvature formula in [41] on an internal triangular node q_i is

$$\kappa_{q_i} = \frac{2\pi - \sum_j \theta_j}{\frac{1}{3}\sum_j A_j}, \tag{1}$$

where θ_j are the inner angles incidental at q_i, and A_j are the corresponding triangle areas. When utilizing the above approximation of Gaussian curvature to detect the developability of the given patch M, by the theorem of differential geometry, we have $\kappa_{q_i} = 0$, which also leads to $\sum_j \theta_j = 2\pi$. Q.E.D.

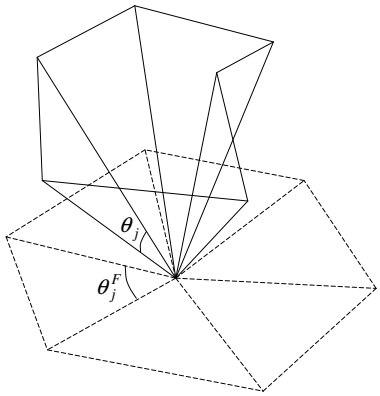


Fig. 3 The inner angles before and after flattening the triangles around a vertex

For an internal vertex, we call it a *developable point* when $\sum_j \theta_j = 2\pi$ is satisfied at this point; otherwise, it is called a *non-developable point*. Using Proposition 3-1, we can detect whether a given mesh patch M is developable by checking every internal vertex. However, simply stating whether a surface is developable or not is insufficient for identifying the degree of developability of the surface. Thus, we define the *developability function* on a tessellated surface as follows.

Definition 3-2 The developability function of a tessellated surface M is defined as

$$D[M] = \frac{1}{A}\sum_i \delta(2\pi - \theta_{sum}(q_i))A_{q_i} \tag{2}$$

where $\delta(t)$ is the impulse function, $A_{q_i} = \frac{1}{3}\sum_j A_j$ is the sum of the areas of the incidental triangles at a vertex q_i on M, and A is the area of M. $\theta_{sum}(q_i)$ is either the sum of inner angles incidental at q_i when q_i is an internal vertex, or set to 2π if q_i is on the boundary of M.

The developability function is actually a weighted sum of the discrete Gaussian curvature given in Eq(1). The value of the developability function gives a progressive estimate of the developability of a surface. When $D[M]=1$, all internal vertices on this surface are developable points; in other words, M is developable. When $D[M]=0$, it means that we cannot find any developable point on the surface – M is absolutely non-developable. For any $D[M]\in(0,1)$, there are some developable points on M. The larger the value of $D[M]$, the more developable the surface M is.

3.3 Constrained optimization

For a given polygonal patch M with *n* vertices and $D[M]<1$, the problem we are to solve here is to find an optimized M^* with the same topology as M but with different vertex positions. The M^* should be developable (i.e., $D[M^*]=1$), and the difference between M^* and M should be minimized since the shape of M is what the designer desires. Therefore, we formulate the problem as a *constrained optimization problem*

$$\textbf{\textit{min}}\ (M - M^*)\ \textbf{subject to}\ D[M^*]=1. \tag{3}$$

In the definition of the developability function, there is an impulse function which may lead to irregularity during the optimization. Here, we define a new *developability detect function* $G[\cdots]$ to take place of the developability function $D[\cdots]$ as

$$G[M^*]=\sum_i (g(q_i(M^*)))^2 \tag{4}$$

where $q_i(M^*)$ is the position of a triangular vertex $q_i \in M^*$, and the function $g(q_i)$ is the *vertex developability detect function* given as

$$g(q_i)=\begin{cases} 2\pi-\sum_k \theta_k & (q_i \notin \mathrm{B}) \\ 0 & (q_i \in \mathrm{B}) \end{cases} \tag{5}$$

where B is the set of triangular vertices on the boundary of the given mesh patch M^*. It is not hard to verify that when $G[M^*]=0$, the sum of the inner angles at every internal vertex equals 2π, hence $D[M^*]=1$ is satisfied. Thus, we replace the developability constraint by this new one and the constrained optimization problem is redefined as

$$\textbf{\textit{min}}\ (M - M^*)\ \textbf{subject to}\ G[M^*] = 0. \tag{6}$$

It is important to state that the optimization formulation of Eq. (6) pertains to a single patch M_i on the embedded object O. Since O is usually made of several surface patches assembled together, the continuity constraint should also be added when these patches are optimized individually. This will be discussed when the details of the algorithm is presented.

4. Global Optimization

A penalty function based scheme is presented in this section that solves the constrained optimization problem of Eq. (6). This is a global optimization (i.e., all vertices move at the same time at an iteration step). Two essential tasks need to be embarked upon: the numerical solution of the optimization itself and the continuity preservation among the patches during the optimization process, as entailed separately next.

4.1 Penalty function based scheme

By definition of the constrained optimization problem (Eq.(6)), we attempt to minimize the surface discrepancy between M and M^*. An elastic energy $E(M^*)$ is defined below to quantify the difference,

$$E(M^*) = \sum_j \left(\left\| q_{j,s} q_{j,e} \right\| - l_j^0 \right)^2 \tag{7}$$

where j is the index of a triangular edge, $q_{j,s} \in M^*$ and $q_{j,e} \in M^*$ are the vertices of the edge, and l_j^0 is the length of the triangular edge j on M. This energy function simulates a spring network in which every spring follows along a triangular edge on M^*. The energy measures the change of length on every triangular edge between M^* and M. Thus, the constrained optimization problem is redefined as

$$\textbf{\textit{min}}\ E(M^*)\ \textbf{subject to}\ G[M^*] = 0. \tag{8}$$

Eq. (8) can be converted into an unconstrained optimization problem by adding the constraint as a penalty term to the objective function [45]. As a result, the objective function to be optimized becomes

$$J(M^*) = E(M^*) + \frac{\rho}{2}(G(M^*))^2 \tag{9}$$

where ρ is the coefficient to balance the weight between $E(M^*)$ and $G(M^*)$. The choice of ρ is by no means trivial; for a smaller ρ, the computing procedure converges slowly to $G[M^*] = 0$; when ρ is large, on the other hand, the shape of the surface after optimization usually deviates too much from the one before the optimization. For any starting optimization point M^0, the procedure begins to minimize $J(M^0)$ with

$\rho = \frac{1}{n_e (G[M^0])^2} \sum_j (l_j^0)^2$, where n_e is the number of triangular edges. After applying the conjugate gradient method to minimize the value of $J(M)$ with a fixed number of iteration steps (which is empirical and is 5 in our implementation), we obtain a new point M^1. Then, we use M^1 as a starting guess for the minimum of $J(M)$ with $\rho = \frac{1}{n_e (G[M^1])^2} \sum_j (l_j^0)^2$ and obtain M^2, and so on. In actual computation, we stop the process either when the constraint violation is less than a given threshold or when changes in $J(M)$ become insignificant. This optimization procedure guarantees the convergence. Since our objective function (Eq.(9)) is in a quadratic form, with a fixed ρ, the conjugate gradient procedure will converge to a minimum near the initial value. This follows what we expected to minimize the difference of M and M^*. With the value of $G[M^i]$ becomes smaller and smaller, ρ increases accordingly, so the surface evolves to more and more developable during the computing (i.e., $G[M] \to 0$). Theoretically, we arrive at the developable patch M^* in the limit as ρ tends to infinity.

When using the gradient-based method to minimize $J(M)$, we need to compute the gradients of J with respect to q_i. First of all, we have

$$\frac{\partial J}{\partial q_i} = \frac{\partial E}{\partial q_i} + \rho G \frac{\partial G}{\partial q_i}. \tag{10}$$

Analytically, $\frac{\partial E}{\partial q_i} = \frac{\partial}{\partial q_i} \sum_j \left(\left\| q_i q_j \right\| - l_{ij}^0 \right)^2$, where q_j are the vertices adjacent to q_i, and l_{ij}^0 are the original length between q_i and q_j. Thus, we obtain

$$\frac{\partial E}{\partial q_i} = \sum_j 2 \left(\left\| q_i q_j \right\| - l_{ij}^0 \right) \frac{q_i - q_j}{\left\| q_j q_i \right\|}. \tag{11}$$

For $\frac{\partial G}{\partial q_i}$, since it is very complex (with more than 40 terms), we compute it numerically using the central difference equation $\frac{\partial G}{\partial q_i} = \frac{G(q_i + h) - G(q_i - h)}{2h}$. When the position of a vertex q_i is changed, of the terms in G, only the $g(\cdots)$ s with respect to q_i and its adjacencies will incur changes. Thus, to reduce the computing time of $\frac{\partial G}{\partial q_i}$, we adopt the following equation to determine it numerically,

$$\frac{\partial G}{\partial q_i} = \frac{G_P(q_i + h) - G_P(q_i - h)}{2h} \tag{12}$$

where $G_P(q_i) = (g(q_i))^2 + \sum_j (g(q_j))^2$ with q_j being the adjacent vertices to q_i, and h is a small constant (the determination method of h according to the value of $G_P(q_i)$ is from [46]).

In the above formulas, the gradients of J with respect to the vertex positions of M are computed locally, so the computing time is reduced. Now, we can compute the optimized J with respect to M by a conjugate gradient method which includes the iterative process of computing gradients at current state and searching for an optimum state along the conjugate direction [45]. The unnecessary details of the conjugate gradient method are omitted here. The terminal condition of the conjugate gradient method is chosen to be $\frac{\left\| G[M^i] - G[M^{i-1}] \right\|}{G[M^0]} < \eta$ where $G[M^i]$ is the value of the constraint function in the ith iteration (current value), $G[M^0]$ is the value of the constraint function before optimization, and η is a small threshold number (we choose $\eta = 0.01\%$ in our testing examples). Similar to other iterative solutions, a maximum iteration number $N_{\max}$ is used in our system as another stop criterion – the numerical iteration stops after it has iterated $N_{\max}$ steps.

4.2 Continuity preservation

In the object O consisting of assembled mesh patches M_i ($i = 1, \cdots, m$), a vertex shared by more than one patches is called an *assembling vertex*. All other assembly constraints, e.g., different kinds of fixed tolerance, need to be converted into the information of coincident assembling vertices and their related linked vertex sets. Associated with an assembling vertex q_p, we define a *linked vertex set* L_{q_p} which contains all the mesh vertices in O coincidental at q_p; also, for any vertex $q_q \in L_{q_p}$, there is the associated linked vertex set L_{q_q} where we have $q_p \in L_{q_q}$. The cardinality of the linked vertex set of a vertex is exactly the number of patches sharing the vertex. By means of these linked vertex sets, the connectivity information of assembled patches is stored. However, this connectivity is ignored when the shape of every $M_i \in O$ is being optimized individually – for two coincidental triangular nodes belonging to two different patches, their positions are adjusted independently since the gradients of J respect to them might be different; so cracks will appear at places where two patches originally met.

The numerical scheme then needs to be enhanced to take into consideration of preserving the G^0 continuity of O**.** The basic idea is to make the linked vertices consistent during the optimization. To achieve this consistency, the formulas of computing gradients at the assembling vertices are modified. When changing the

position of an assembling vertex q_a, the positions of vertices in L_{q_a} should be maintained the same as q_a. Thus, the gradient of E with respect to q_a relates to not only $\sum\left(\left\|q_a q_j\right\|-l_{aj}^0\right)^2$ but also all the other terms $\sum\left(\left\|q_p q_q\right\|-l_{pq}^0\right)^2$ ($q_q \in L_{q_a}$) in E, where $q_a q_j$ are the incident edges at q_a, and $q_p q_q$ are the edges with one endpoint $q_q \in L_{q_a}$. Thus, the gradient is modified to become

$$\frac{\partial E}{\partial q_a}=\frac{\partial}{\partial q_a}\sum_j\left(\left\|q_a q_j\right\|-l_{aj}^0\right)^2=\sum_j 2\left(\left\|q_a q_j\right\|-l_{aj}^0\right)\frac{q_a-q_j}{\left\|q_j q_a\right\|}, \tag{13}$$

where q_j are either the vertices adjacent to q_a or the adjacent vertices to a vertex in L_{q_a}. Also, the gradient of G with respect to q_a should be changed to

$$\frac{\partial G}{\partial q_a}=\frac{G_{PA}(q_a+h)-G_{PA}(q_a-h)}{2h}, \tag{14}$$

where $G_{PA}(q_a)=(g(q_a))^2+\sum_q(g(q_q))^2+\sum_j(g(q_j))^2$ with q_j being either the adjacent vertices of q_a or the adjacent vertices of q_q ($q_q \in L_{q_a}$). When calculated with the above prescribed method, the gradients of the linked vertices become consistent with each other. Therefore, while searching for the optimum along the conjugate direction, the updating of their positions is also kept consistent, which in turn then ensures the G^0 continuity.

5. Local Optimization

Although the penalty function based global optimization gives a high quality result, its computing speed is usually very slow which cannot satisfy the requirement of real-time design activities. In this section, the developability-by-deformation problem is reformulated as a local optimization problem and an algorithm is given that iteratively updates the position of vertices to achieve a developable mesh.

5.1 Reformulation of the problem

Recall the original definition of the constrained optimization problem (Eq.(6)), our objective is to modify a given mesh M into a developable one M^* while minimizing the difference between M and M^*. Let us consider only one vertex q on the given mesh M, where $g(q)\neq 0$. The basic idea of local optimization is moving q along its normal direction n_q (which is the average normal of q's adjacent faces) to find a new position $q^*=q+\delta n_q$ with $g(q^*)=0$; at the same time, the movement scale δ must be kept as small as

possible in order to minimize the surface change. Therefore, the global optimization problem is decomposed into a combination of local optimizations on triangular vertices. On a vertex q, the problem is defined as

$$\textbf{\textit{min}}\ \delta^2\ \textbf{subject to}\ T(\delta) = 0\,, \tag{15}$$

with $T(\delta) = (g(q + \delta n_q))^2 + \sum_j (g(q_j))^2$ with q_j being the adjacent vertices to q_i. When q moves, it affects not only the developability at q itself but also that of all its adjacent vertices. Thus, the constraint $T(\delta)$ of local optimization on q is set on both the vertex q and its neighbors. When using the *Lagrange Multiplier Method* to solve the above optimization problem (Eq. 15), the Lagrange function can be written as

$$L = \delta^2 + \lambda T(\delta) \tag{16}$$

where λ is the Lagrange multiplier. By setting $\frac{\partial L}{\partial \delta} = 0$ and $\frac{\partial L}{\partial \lambda} = 0$, we obtain the following equations:

$$2\delta + \lambda \frac{dT}{d\delta} = 0\,, \tag{17}$$

$$T(\delta) = 0\,. \tag{18}$$

After replacing $T(\delta)$ in (18) with a linear approximation based on T's Taylor series

$$T(\delta) \approx T(\delta_0) + \dot{T}(\delta_0)(\delta - \delta_0)\,,$$

the following equation of updating δ is obtained

$$\delta = \delta_0 - \frac{T(\delta_0)}{\dot{T}(\delta_0)}\,. \tag{19}$$

From (17), we have $\lambda = -\frac{2\delta}{\dot{T}(\delta)}$, so by the property of the Lagrange method of constrained optimization [45], if $\delta \neq 0$ and $\dot{T}(\delta) \neq 0$, the iteration of Eq. (19) converges to the minimum (if $\dot{T}(\delta) = 0$, we just simply fix the vertex). Now that the magnitude of the update of an individual vertex at an iteration step is decided by Eq. (19), next we need a mechanism by which the order of the local optimization on the vertices can be determined. The square of the vertex developability detect function $g(q_i)$ as defined in Eq. (5) presents itself to be a natural choice and is adopted in our system.

5.2 Outline of the algorithm

Our local optimization algorithm is built around vertex selection and vertex position updating. As mentioned earlier in section 3.1, our system represents a model by an adjacency graph structure, which includes

vertices, edges, and faces, as well as the connection relationship among them; they are all explicitly represented and linked together. Each vertex maintains a list of the edges of which it is a member. The overall algorithm is outlined below by *Algorithm* **LocalDevelopabilityOptimize**(*O*).

Algorithm **LocalDevelopabilityOptimize**(*O*)

Input: a given object *O* represented as a set of polygonal mesh patches

Output: the optimized polygonal mesh patches

1. Compute the vertex developability detect function $g(q)$ at each vertex q on the given mesh patches;
2. Compute the unit normal $\boldsymbol{n}$ of each vertex q on O;
3. Place all vertices in a maximum heap H keyed on the $[g(\cdots)]^2$ measure – the vertex with the maximum $[g(\cdots)]^2$ is placed at the top of H;
4. $j \leftarrow 0$;
5. **Do** {
6. Select the vertex q at the top of H and update its movement scale δ along its unit normal $\boldsymbol{n}$ according to Eq.(17);
7. Update the $[g(\cdots)]^2$ cost of q and its adjacent vertices to reflect the movement δ on q – this will change the locations of these vertices in H;
8. $j \leftarrow j+1$;
9. } **while** ((the $[g(\cdots)]^2$ of the vertex at top of H is greater than ε) **and** ($j < N_{\max}$));
10. Update the positions of all the vertices by their movement scales;
11. Update the normal vectors of all the vertices on O;
12. **return**.

We elaborate the above algorithm by addressing the following questions.

Surface difference control

In the above algorithm, the difference between the optimized mesh and the given mesh is not controlled. Such a control can be added when updating the vertex q at the top of H – in our implementation, we just simply set $\delta = \delta_T$ if $\delta > \delta_T$ and truncate δ to $-\delta_T$ when $\delta < -\delta_T$, where δ_T is the given difference tolerance. It calls to pay the special attention to the unit normal at each vertex – it remains unchanged in the entire iterative process and is updated only once at the end of the process – every vertex moves along its original normal passing through its original position during the iteration. If all vertices move by δ_T along their original normal directions, the result would be identical to an offset surface of the given surface (δ_T is the value of offset).

Therefore, the optimized mesh is controlled between the $+\delta_T$ and $-\delta_T$ offset surfaces of the given surface. The smaller the tolerance δ_T, the closer the optimized mesh patches are to the original surface, and the slower the optimization algorithm converges. On the opposite, a larger tolerance δ_T will result in a faster convergence but at a cost of larger deviation from the original surface.

Terminal conditions of iterations

During the iteration of algorithm **LocalDevelopabilityOptimize**(*O*), the $[g(\cdots)]^2$ value of the vertex at top of the heap decreases while the step number of iteration, *j*, increases. These two factors are utilized to control the terminal condition of the iteration. Which of the two takes effect depends crucially on the given tolerance δ_T – when the value of δ_T is large enough, the given polygonal patches can be fully optimized, we stop at $g(q) \le \varepsilon$ for the top element q in heap *H*; on the other hand, a too small δ_T will stingingly limit the movement of each vertex and the optimization (Eq. (15)) would dwell at certain level and has to be stopped by the maximum number of iterations $N_{\max}$ criterion.

Continuity preservation

By definition, the $[g(\cdots)]^2$ value of any point on the boundary of *O* is zero; as a result, it will not be moved during the optimization (note that a vertex with zero $[g(\cdots)]^2$ is put at the bottom of heap *H*). However, one still faces the continuity problem if an assembling vertex is interior to some patch (see Fig. 4). We resolve this problem by the simplest way – all the assembling vertices remain fixed during the optimizing process. For an interior assembling vertex, if its $[g(\cdots)]^2 > 0$, its developability is enhanced via adjusting the positions of its adjacent vertices. The reason, why the developability at an internal assembling vertex can be achieved by perturbing the neighboring non-assembling vertices, is that: by the definition of function $T(\delta)$ in eq.(15), $T(\delta) = (g(q + \delta n_q))^2 + \sum_j (g(q_j))^2$, where q_j are the adjacent vertices to the moved vertex, the movement of a vertex is not only measurement on the developability at this vertex but also on the vertices around it. Therefore, when perturbing the non-assembling vertices around an assembling vertex, the developability at the assembling vertex is also increased.

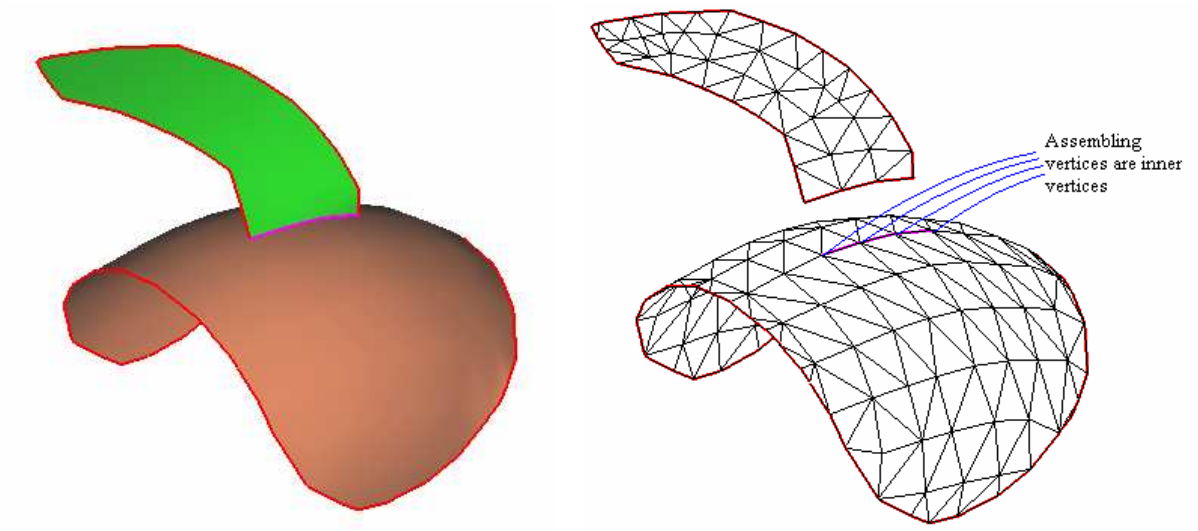
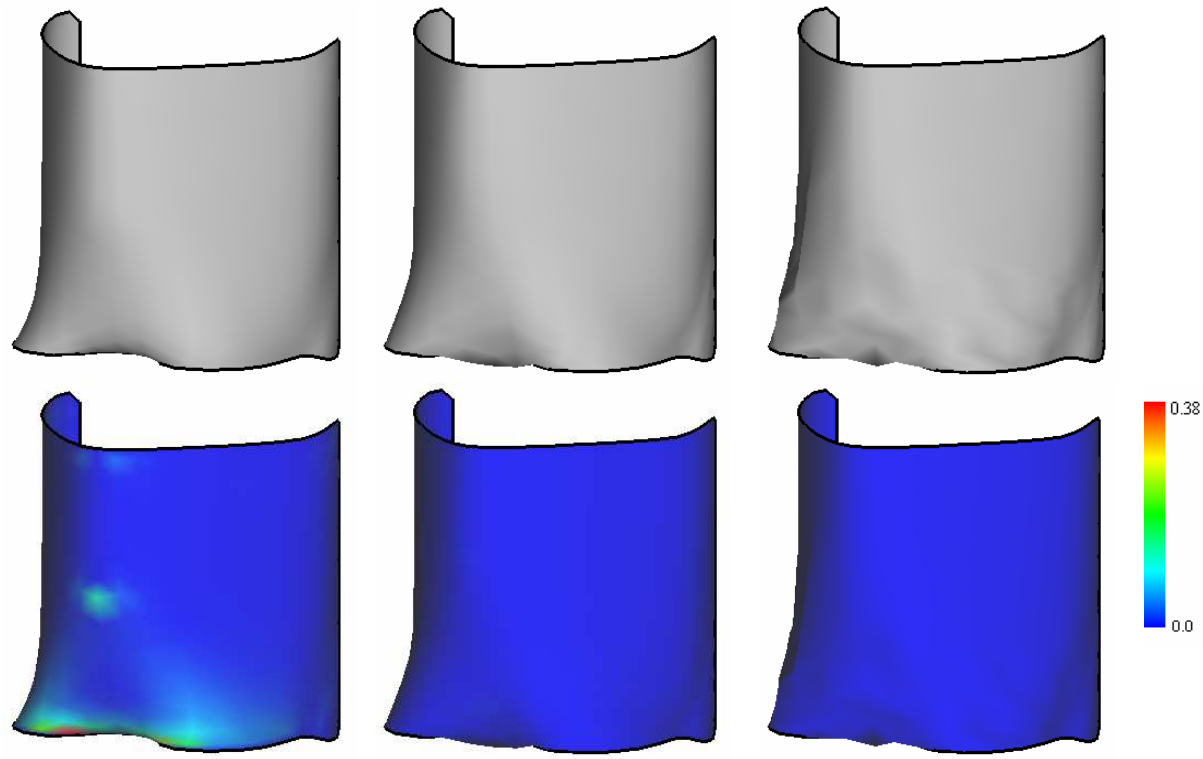


Fig. 4 Interior assembling vertices



(a) given surface and its Gaussian map **(b) results of global optimization** **(c) results of local optimization** ($\delta_T = 1.0$)

Fig. 5 Example I – global vs. local optimization

6. Experimental Results

In this section, we give some experimental examples to demonstrate the functionality of both the global and local optimization approaches, as well as their comparison. In the first example, Example I, which was originally shown in Fig. 1, we applied both the global and local optimizations to the original surface. The surface before optimization and its Gaussian map are given in Fig. 5a (in a Gaussian map, the color of a point

represents its $[g(\cdots)]^2$ value). The resultant surfaces after both the global and local optimizations are shown in Fig. 5b and 5c respectively, where for the local optimization of Fig. 5c the tolerance δ_T is set to 1.0. Both the global and local optimization approaches achieve fully optimized results within the required maximum iteration steps. As seen in the figures, the global optimization gives a smoother resultant surface. This is because in a global optimization all the vertices move together, while the local approach moves vertices one by one. Therefore, the original smoothness of the given surface is not maintained by the local optimization approach. The following examples, Example II and III, also verify this point.

Example II is the surface of a shoe upper layer; since it is usually manufactured from a planar leather sheet, the surface is desired to be developable. Fig. 6 displays the optimization results. In this particular case, neither the global nor the local approach can achieve the full optimum i.e., both of them were stopped by the maximum iteration step criterion (for the local optimization the difference tolerance δ_T is set to 0.38).

Example III comes from the application of apparel industry. The assembled polygonal patches of a pair of short pants are constructed in three-dimensional space; each patch must be developable since it will come from its corresponding 2D pattern in manufacturing. The Gaussian map of the original surface (Fig. 7a) shows that the original design incurs severe non-developability. The result surfaces after the optimizations are shown in Fig. 7b and 7c. Unlike the first two examples, in this case, the local optimization approach, with $\delta_T = 2.4$, achieves a fully developable result while the global approach fails to do so within 200 iteration steps.

The computational statistics of Example I, II, and III is given in Table 1. Implemented by a program written in C++ and running on a standard PC, the local optimization approach is seen to be able to generate the desired result in near real-time; on the other hand, the global optimization approach usually takes several minutes to reach a result with a decent level of surface developability. As expected, the converge speed of the local optimization crucially depends on the difference tolerance δ_T. For a properly chosen δ_T, the local optimization can converge quickly; otherwise, the iterative process is stopped by the maximum iteration criterion $N_{\max}$. In the most extreme case of $\delta_T = 0$, no any improvement can be made as the original surface is fixed. After experimenting with a variety of test examples, it is observed that a $\delta_T = \overline{L}$ in general achieves satisfactory improved developability while maintaining reasonably well the dimensions of the original surface, where $\overline{L}$ is the average length of the triangular edges on the given polygonal mesh patches. Thus, in all the three examples, δ_T is set to be $\overline{L}$.

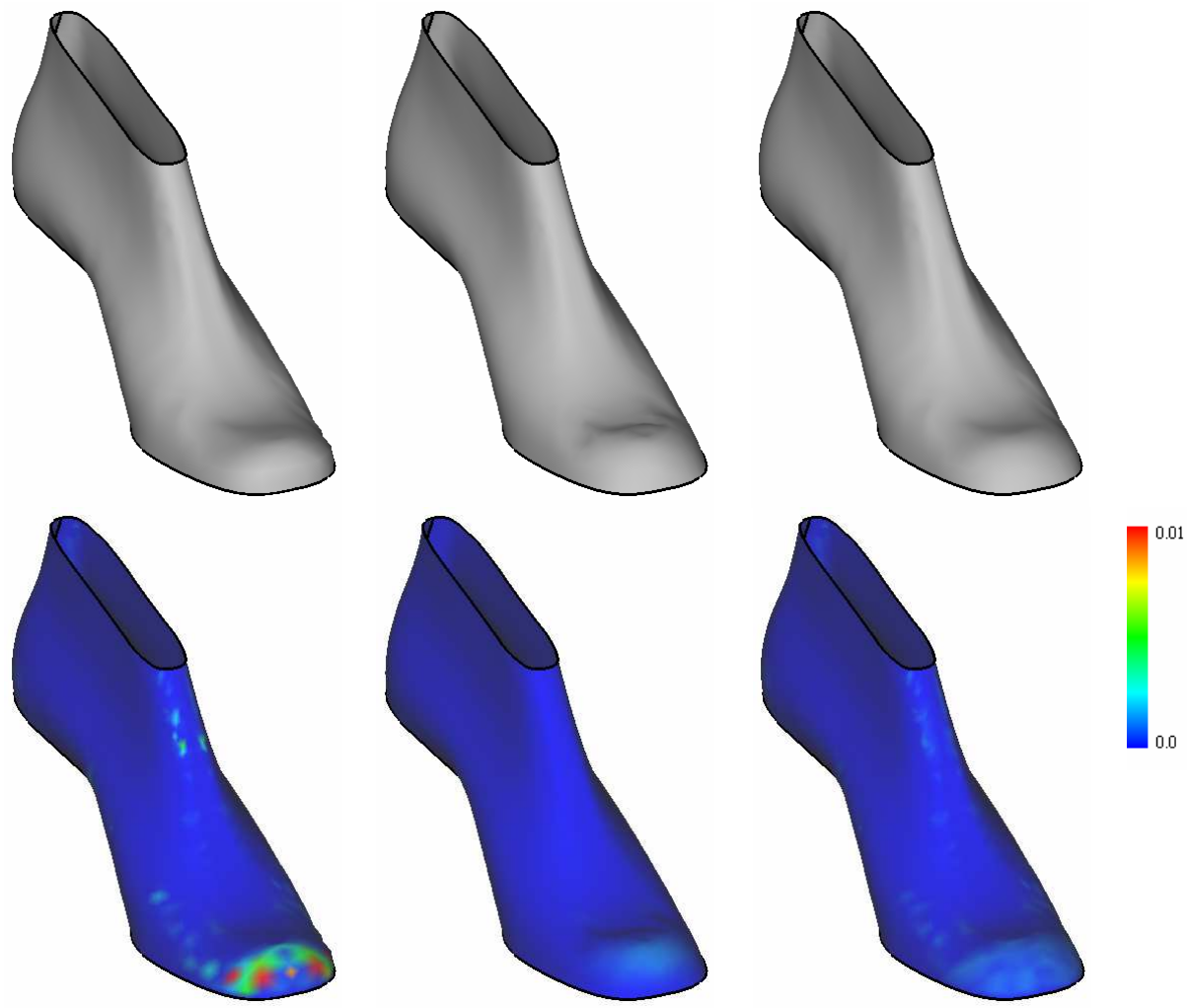
In addition to the Gaussian map, the *distance error map*, in which the colors represent the distances of vertices to the original surface, is utilized in our system to compare the results from global and local approaches. The distance error maps for the given three examples are shown in Fig. 8. As revealed from the figure, the maximum distance error from the global optimization approach generally is smaller than the one generated from the local approach. This phenomenon can be explained by noting that, by its nature, in the local optimization, only a small subset of the vertices with high $[g(\cdots)]^2$ values will be moved, whereas in the global optimization all the vertices are moved in sync at each iteration step. As a result, to achieve a same level of overall developability, certain vertices often need to be moved by larger distances in the local optimization than their counter-parts in the global optimization, due to the restraint on the moveable vertices in the local optimization.

The distance error map is also adopted to study the effect of the difference tolerance δ_T on the level of optimization in the local optimization approach. In Fig. 9, it is evidently seen that enlarging δ_T increases the freedom of movement for vertices in the local optimization which in turn enhances the optimization result. Viewed from another perspective, the distance error map together with the Gaussian map of the final optimized surface serve to "measure" the level of non-developability of the original surface: smaller errors on the distance error map but larger values on the Gaussian map indicate an "easy" conversion from the original non-developable surface to a highly developable surface with minimum deviation, while the opposite combination implies a "difficult" task – large discrepancies have to be resulted on the final surface if high degree of developability is desired. The maps of principal curvatures - $|\kappa_{\min}|$ and $|\kappa_{\max}|$ of the surfaces in Fig. 9 are also listed in Fig. 10. It is easy to find that the principal curvatures are increased with the enlargement of δ_T, so more wrinkles occurs. However, for example when $\delta_T = 1.0$, the places with large $|\kappa_{\min}|$ have a corresponding small value of $|\kappa_{\max}|$ (see Fig. 10). That's why more wrinkles can still give a result with better developability.

Table 1 Computational Statistics

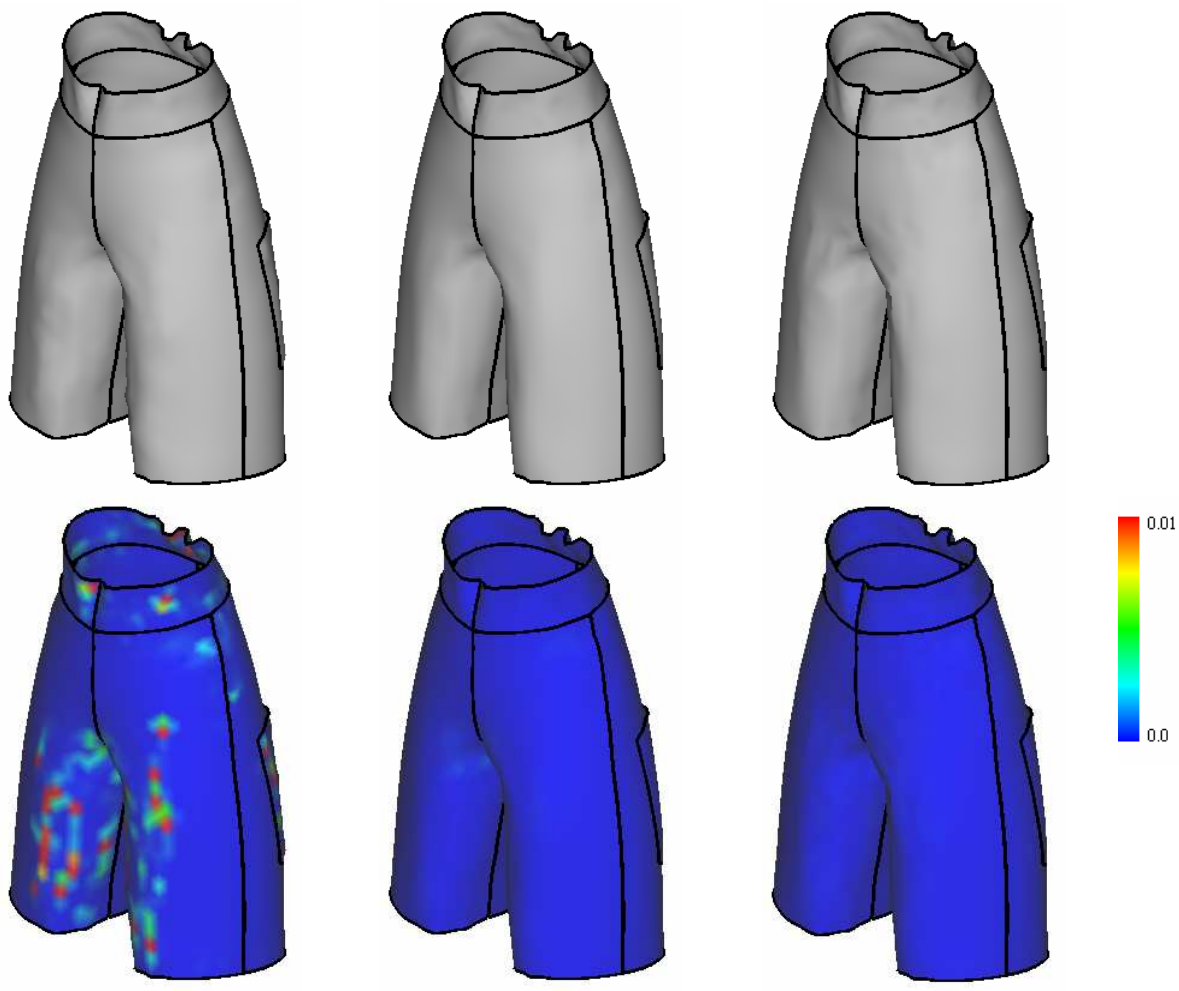
Example	Vertex number	Optimize Approach	Time Cost	Result Figure	$g^0_{\max}$	$g^*_{\max}$	Terminal Condition
I	517	Global	46s	Fig. 5b	0.14	2.1×10^{-3}	$\frac{\|G[M^i]-G[M^{i-1}]\|}{G[M^0]}\le 0.01\%$
		Local	1s	Fig. 5c		9.9×10^{-5}	$g^*_{\max}\le 10^{-4}$
II	3047	Global	390s	Fig. 6b	0.093	1.1×10^{-3}	$N_{\max}=200$
		Local	23s	Fig. 6c		1.0×10^{-3}	$N_{\max}=500000$
III	3016	global	310s	Fig. 7b	0.19	1.3×10^{-3}	$N_{\max}=200$
		local	1s	Fig. 7c		9.9×10^{-5}	$g^*_{\max}\le 10^{-4}$

*All tested on a PIII 900 PC with a program written in C++ with 1) $\eta=0.01\%$ and $N_{\max}=200$ for the global optimization approach; and 2) $\varepsilon=10^{-4}$ and $N_{\max}=500000$ for the local optimization approach.



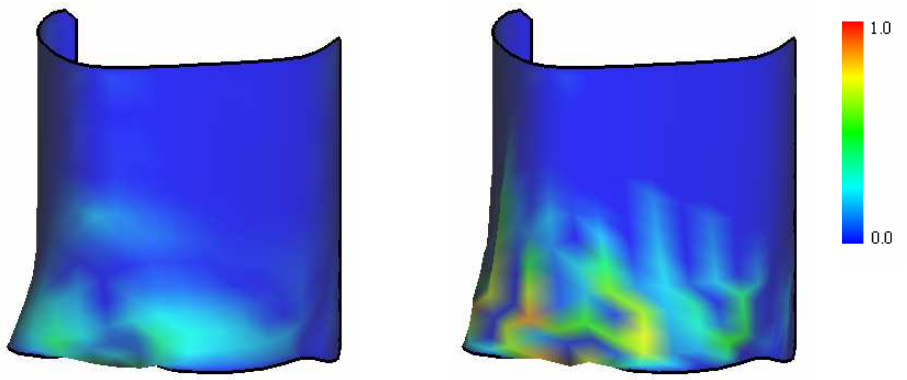
(a) given surface and its Gaussian map **(b) result of global optimization** **(c) result of local optimization** ($\delta_T=0.38$)

Fig. 6 Example II – the surface of a shoe last



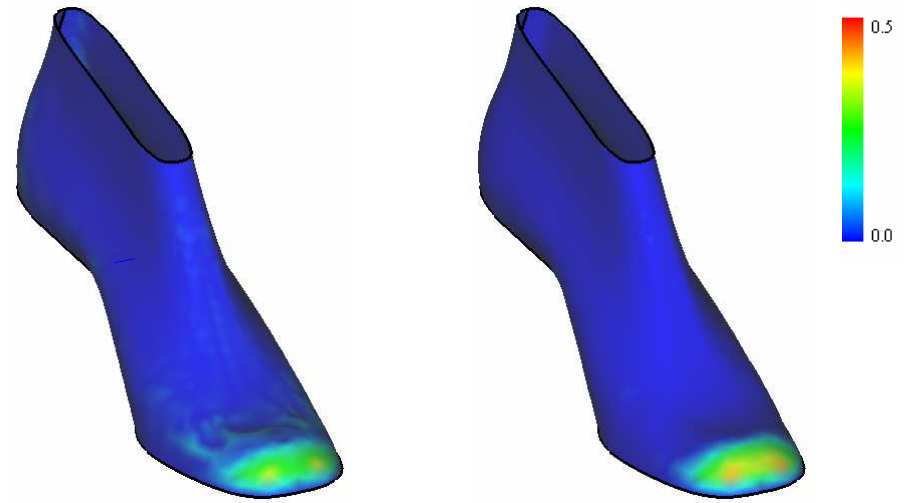
(a) the original pants and their Gaussian map | **(b) result of global optimization** | **(c) result of local optimization** ($\delta_T = 2.4$)

Fig. 7 Example III – a pair of short pants with multiple patches



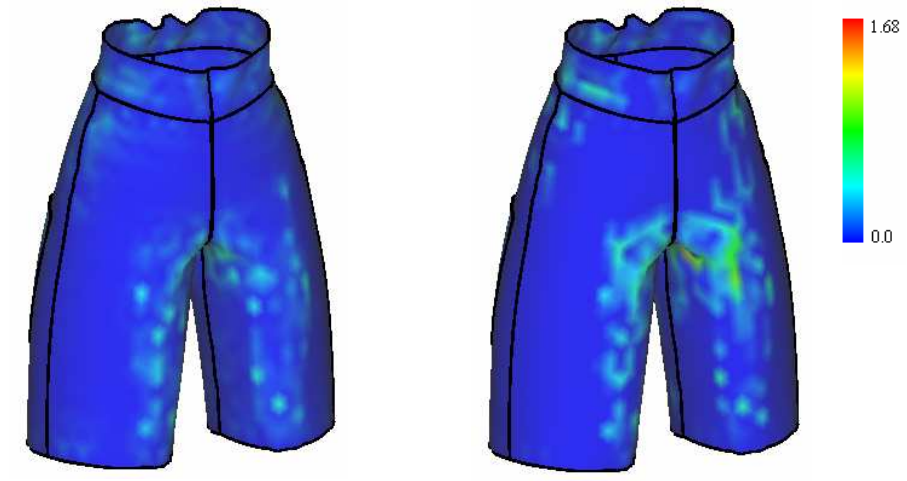
global optimization result **local optimization result**

(a) Example I



global optimization result **local optimization result**

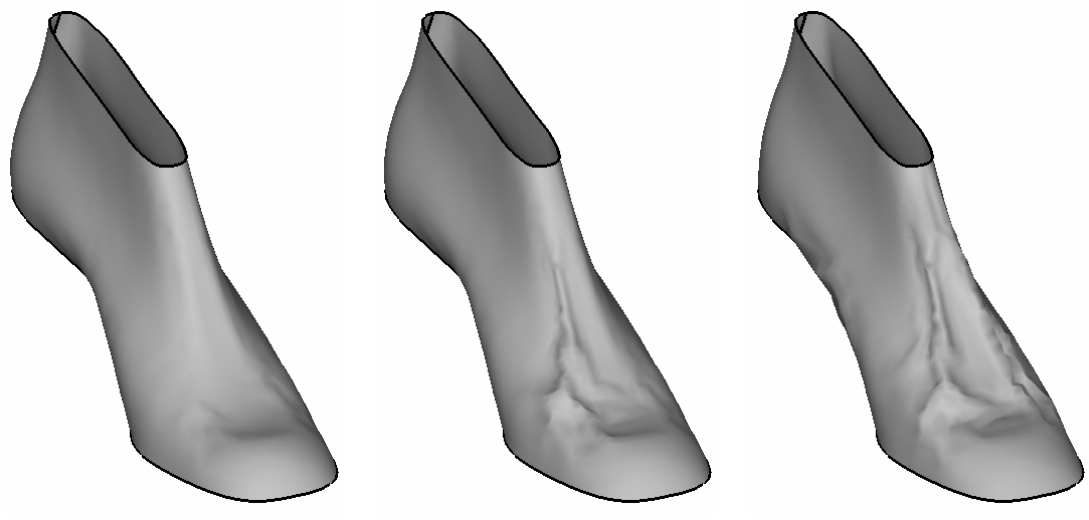
(b) Example II



global optimization result **local optimization result**

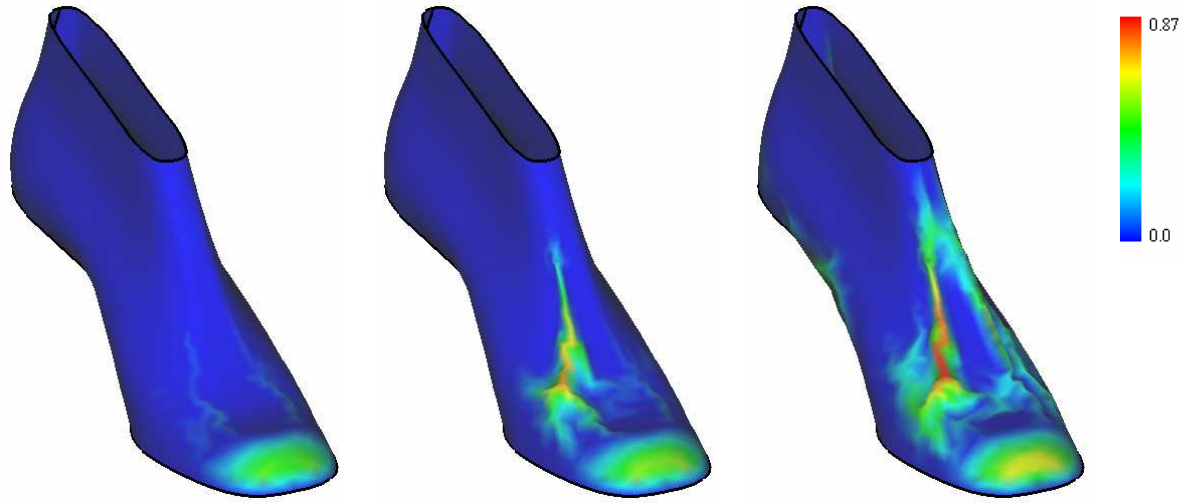
(c) Example III

Fig. 8 The distance error maps of global and local optimization results



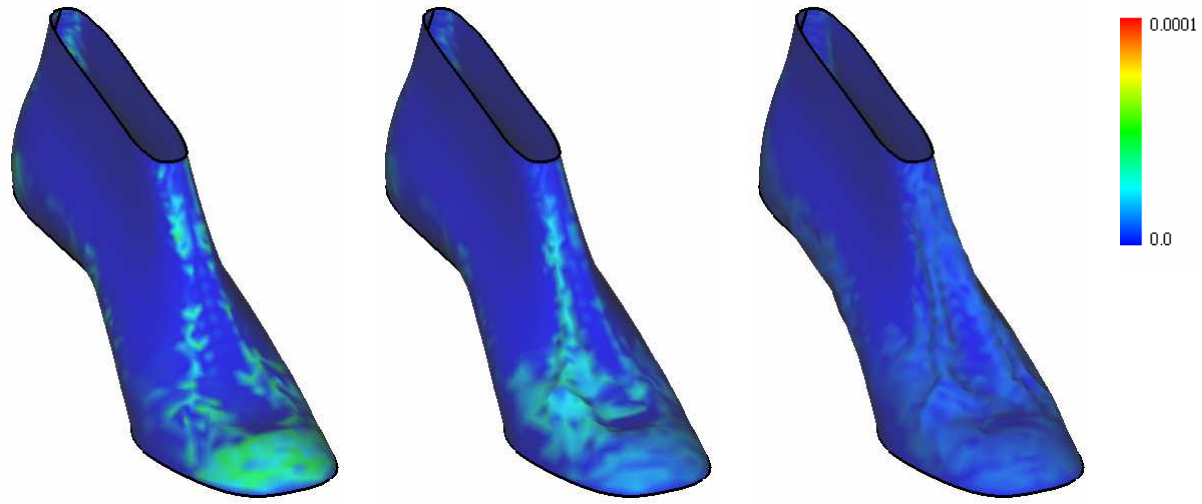
$\delta_T = 0.5$ $\delta_T = 0.75$ $\delta_T = 1.0$

(a) resultant surfaces



$\delta_T = 0.5$ $\delta_T = 0.75$ $\delta_T = 1.0$

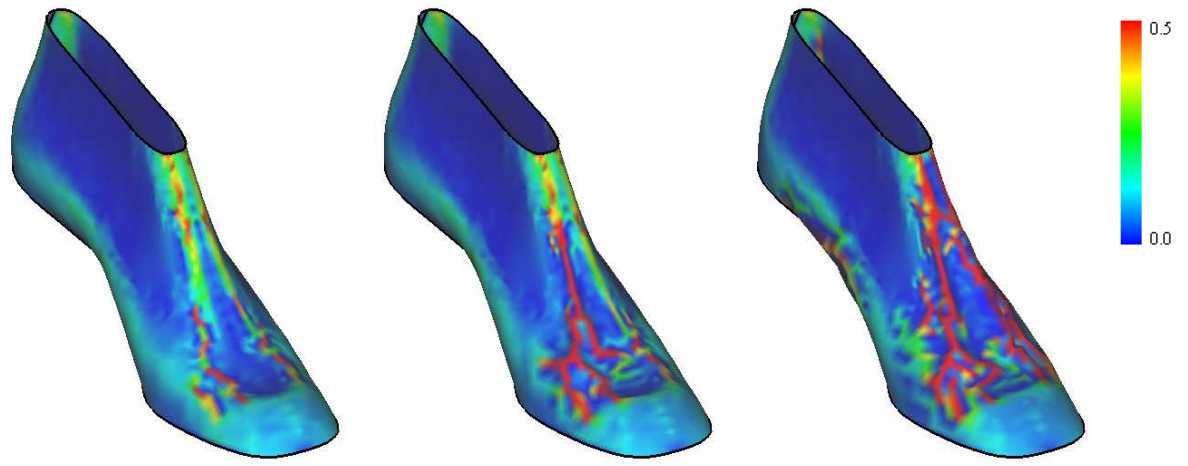
(b) distance error maps of the resultant surfaces



$\delta_T = 0.5$ $\delta_T = 0.75$ $\delta_T = 1.0$

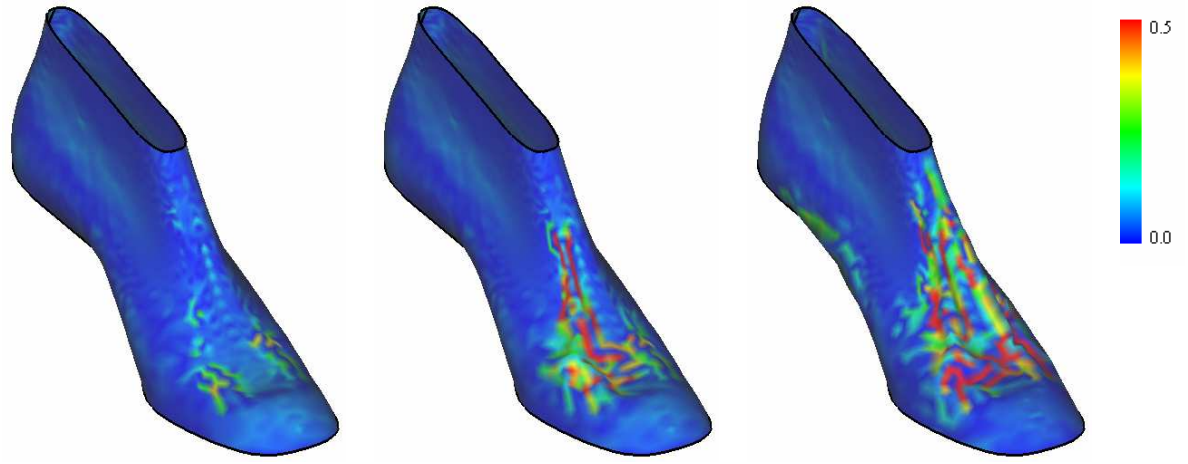
(c) Gaussian maps of the resultant surfaces

Fig. 9 Increasing δ_T leads to larger distance errors but better developability



$\delta_T = 0.5$ $\delta_T = 0.75$ $\delta_T = 1.0$

(a) $|\kappa_{\min}|$ of the resultant surfaces



$\delta_T = 0.5$ $\delta_T = 0.75$ $\delta_T = 1.0$

(b) $|\kappa_{\max}|$ of the resultant surfaces

Fig. 10 Increasing δ_T leads to more wrinkles (larger principal curvature) but better developability

7. Summary and Discussion

The focus of this paper is the so called developability-by-deformation problem – how to deform a given non-developable polygonal surface into a developable one with minimum change. Because developability of a surface is often a strongly required attribute in a diversity of engineering applications, a practical solution to this problem is highly needed. We contribute by proposing two numerical solutions to the developability-by-deformation problem. Both approaches are based on the principle of energy minimization which seeks to minimize the amount of deformation while at the same time maximizes the degree of developability of the surface. The two differ with each other in how this minimization is formulated as well as the way the vertices on the polygonal mesh are moved during the minimization process: while the first approach formulates the minimization as a global constrained optimization in which all the vertices move simultaneously at each iteration step, the second approach is of local optimization nature where only one vertex is moved at a time

based on a locally defined optimization criterion. Experimental examples are provided to demonstrate the functionality of the proposed two approaches as well as their comparison in terms of computing cost, effectiveness of attaining developability, dimensional difference between the surfaces before and after the optimization, and other important aspects.

Both solutions can be integrated into a geometric modeling system for product design where surface developability is obliged. Owing to its better ability of maintaining the smoothness of the original surface due to its global nature, the first solution, the global optimization approach, can be used for those applications where the smoothness and quality of the product surface are emphasized. On the other hand, the local optimization based solution may better suit situations where real-time computation – such as in computer graphics simulation – is demanded. Another potential application of the local optimization solution is in wrinkle design, such as in shoe manufacturing, where wrinkles are sometimes deliberately designed to be formed during the manufacturing process of the shoe (for fashion and aesthetic purpose).

On possible future work, as mentioned above, since the smoothness of the original surface is not preserved during the local optimization, one potential topic is how to add smoothing terms into the local updating operator to enforce the required smoothness on the surface. Also, in our current implementation of the local optimization, the surface continuity of multiple patches is preserved by simply fixing all the assembling vertices during the optimizing iteration – this obviously limits the degree of freedom of vertex movements. Thus, study of a better and more flexible continuity preserving method in the local optimization approach is another worthy further work. The topology of the original polygonal surface can be preserved by enforcing continuity across the boundaries of triangular patches. However, self-intersection might occur after re-positioning the vertices, especially in the case of the local optimization with large δ_T. Another possible further work is thus to integrate the self-collision detection and responding algorithm into the strategy of vertex movement.

References

[1] do Carmo M.P., *Differential Geometry of Curves and Surfaces*, Englewood Cliffs, N.J.: Prentice-Hall, 1976.

[2] Singh K., Fiume E., "Wires: a geometric deformation technique", *SIGGRAPH 1998 Conference Proceedings*, ACM., pp.405–414, 1998, New York, NY, US.

[3] Schroeder W.J., Zarge J.A., and Lorensen W.E., "Decimation of triangle meshes", *Computer Graphics*, v26, n2, pp.65-70, 1992, USA.

[4] Garland M., and Heckbert P.S., “Surface simplification using quadric error metrics”, *SIGGRAPH 97 Conference Proceedings*, pp. 209-16, 1997.

[5] Wu J.-H., Hu S.-M., Tai C.-L., and Sun J.-G., “An effective feature-preserving mesh simplification scheme based on face constriction”, *Proceedings Ninth Pacific Conference on Computer Graphics and Applications*, Pacific Graphics 2001, Tokyo, Japan, 16-18 Oct., 2001. IEEE Comput. Soc. 2001, pp.12-21. Los Alamitos, CA, USA.

[6] Taubin G., “A signal processing approach to fairing surface design”, *SIGGRAPH 95 Conference Proceedings*, ACM., 1995, pp.351-58, New York, USA.

[7] Kobbelt L., Campagna S., Vorsatz J., and Seidel H.P., “Interactive multi-resolution modeling on arbitrary meshes”, *SIGGRAPH 98 Conference Proceedings*, ACM., 1998, pp.105-114, New York, USA.

[8] Desbrun M., Meyer M., Schroder P, and Barr A.H., “Implicit fairing of irregular meshes using diffusion and curvature flow”, *SIGGRAPH 99 Proceeding*, ACM., 1999, pp.409-416, New York, USA.

[9] Schneider R., and Kobbelt L., “Generating fair meshes with G^1 boundary conditions”, *Geometric Modeling and Processing Conference Proceedings*, 2000.

[10] Schneider R., and Kobbelt L., “Geometric fairing of irregular meshes for free-form surface design”, *Computer Aided Geometric Design*, vol. 18, no. 4, 2001, pp. 359-379.

[11] Zeleznik R.C., Herndon K.P., and Hughes J.F., “SKETCH: An interface for sketching 3D scenes”, *SIGGRAPH 96 Proceeding*, ACM., 1996, pp.163-170, New York, USA.

[12] Igarashi T., Matsuoka S., and Tanaka H., “Teddy: a sketching interface for 3D freeform design”, *SIGGRAPH 99 Proceeding*, ACM., 1999, pp.409-416, New York, USA.

[13] Bloomenthal J., and Wyvill B., “Interactive techniques for implicit modeling”, *1990 Symposium on Interactive 3D Graphics*, pp. 109-116, 1990.

[14] Markosian L., Cohen J. M., Crulli T., and Hughes J., “Skin: a constructive approach to modeling free-form shapes”, *SIGGRAPH 99 Conference Proceedings*, ACM., 1999, pp.393-400, New York, USA.

[15] Suzuki H., Sakurai Y., Kanai T., and Kimura F., “Interactive mesh dragging with an adaptive remeshing technique”, *Visual Computer*, vol.16, no.3-4, 2000, pp.159-76, Springer-Verlag, Germany.

[16] Zorin D., Schroder P., and Sweldens W., “Interactive Multiresolution Mesh Editing”, *SIGGRAPH 97 Proceeding*, ACM., 1997, New York, USA.

[17] Khodakovsky A., and Schroder P., “Fine level feature editing for subdivision surfaces”, *Proceedings Fifth Symposium on Solid Modeling and Applications*, Jun. 1999, pp.203-11. N.Y.: ACM Press, USA.

[18] Aumann G., “Interpolation with developable Bézier patches”, *Computer Aided Geometric Design*, vol.8, pp.409-420, 1991.

[19] Maekawa T., and Chalfant J., “Design and tessellation of B-spline developable surfaces”, *ASME Transaction Journal of Mechanical Design*, vol.120, pp.453-461, 1998.

[20] Frey W.H., and Bindschadler D., “Computer aided design of a class of developable Bézier surfaces”, *GM Research Publication 1993*; GMR-8057.

[21] Chu C.H., and Séquin C.H., “Developable Bézier patches: properties and design”, *Computer-Aided Design*, 34(7): 511-527, 2002.

[22] Hoschek J., and Pottmann H., “Interpolation and approximation with developable B-spline surfaces”, In M. D. hlen, T. Lyche, and L. L. Schumaker, editors, *Mathematical Methods for Curves and Surfaces*, pages 255-264. Vanderbilt University Press, Nashville, TN, 1995.

[23] Chen H.Y., Lee I.K, Leopoldseder S., Pottmann H., Randrup T., and Wallner J., “On surface approximation using developable surfaces”, *Graphical Models & Image Processing*, 61(2): 110-124, 1999.

[24] Pottmann H., and Wallner J., “Approximation algorithms for developable surfaces”, *Computer Aided Geometric Design*, 16(6): 539-556, 1999.

[25] Redont P., “Representation and deformation of developable surfaces”, *Computer-Aided Design*, 21(1): 13-20, 1989.

[26] Randrup T., “Approximation of surfaces by cylinders”, *Computer-Aided Design*, 30(10): 807-812, 1998.

[27] Park F.C., Yu J., and Chun C., “Design of developable surfaces using optimal control”, *ASME Transaction Journal of Mechanical Design*, vol.124, pp.602-608, 2002.

[28] Shimada T., and Tada Y., “Approximate transformation of an arbitrary curved surface into a plane using dynamic programming”, *Computer Aided Design*, 23(2): 155-159, 1991.

[29] Parida L., and Mudur S.P., “Constraint-satisfying planar development of complex surfaces”, *Computer-Aided Design*, 25(4): 225-232, 1993.

[30] McCartney J., Hinds B.K., and Seow B.L., “The flattening of triangulated surfaces incorporating darts and gussets”, *Computer-Aided Design*, 31(4): 249-260, 1999.

[31] Wang C.C.L., Smith S.S.F., and Yuen M.M.F., “Surface flattening based on energy model”, *Computer-Aided Design*, 34(11): 823-833, 2002.

[32] Yu G., Patrikalakis N.M., and Maekawa T., "Optimal development of doubly curved surfaces", *Computer Aided Geometric Design*, vol.17, pp.545-577, 2000.

[33] Sheffer A., and de Sturler E., "Parameterization of faceted surfaces for meshing using angle based flattening", *Engineering with Computers*,17(3): 326-337, 2000.

[34] Sheffer A., and de Sturler E., "Smoothing an overlay grid to minimize linear distortion in texture mapping", *ACM Transactions on Graphics*, 21(4): 874-890, 2002.

[35] Azariadis P.N., and Aspragathos N.A.,"On using planar developments to perform texture mapping on arbitrarily curved surfaces", *Computers & Graphics*, vol.24, pp.539-554, 2000.

[36] Azariadis P.N., and Aspragathos N.A., "Geodesic curvature preservation in surface flattening through constrained global optimization", *Computer-Aided Design*, vol.33, pp.581-591, 2001.

[37] Azariadis P.N., Nearchou A., and Aspragathos N.A., "An evolutionary algorithm for generating planar developments of arbitrarily curved surfaces", *Computers in Industry*, 47(3): 357-368, 2002.

[38] Aono M., Denti P., Breen D.E., and Wozny M.J., "Fitting a woven cloth model to a curved surface: dart insertion", *IEEE Computer Graphics & Applications*, 16(5): 60-70., 1996.

[39] Aona M., Breen D.E., and Wozny M.J., "Modeling methods for the design of 3D broadcloth composite parts", *Computer-Aided Design*, 33(13): 989-1007, 2001.

[40] Calladine C.R., "Gaussian curvature and shell structures", *Mathematics of Surfaces* (*Proceedings of a Conference*), pp.179-96, Oxford, UK, 1986.

[41] Kobbelt L.P., Bischoff S., Botsch M., Kähler K., Rössl C., Schneider R., and Vorsatz J., "Geometric modeling based on polygonal meshes", *EUROGRAPHICS 2000 Tutorial*.

[42] Sheffer A., "Spanning tree seams for reducing parameterization distortion of triangulated surface", *SMI 2002: International Conference on Shape Modelling and Applications*.

[43] Hoppe H., DeRose T., Duchamp T., McDonald J., and Stuetzle W., "Mesh optimization", *SIGGRAPH 93 Proceeding*, ACM., 1993, pp. 19-26, New York, USA.

[44] Ganapathy S., Dennehy T.G., "A new general triangulation method for planar contours", *Computer Graphics*, vol.16, no.3, pp.69-75, 1982.

[45] Belegundu A.D., and Chandrupatla T.R., *Optimization Concepts and Applications in Engineering*, Upper Saddle River, N.J.: Prentice Hall, 1999.

[46] Moreton H.P., and Sequin C.H., "Functional optimization for fair surface design", *ACM Computer Graphics*, vol.26, no.2, July 1992, pp.167-76, USA.