Vis Comput (2010) 26: 853–863
DOI 10.1007/s00371-010-0467-5

ORIGINAL ARTICLE

A fully geometric approach for developable cloth deformation simulation

Ming Chen · Kai Tang

Published online: 8 April 2010

Abstract We present a new method for simulation of inextensible cloth subjected to a conservative force (e.g., the gravity) and collision-free constraint. Traditional algorithms for cloth simulation are all physically-based in which cloth is treated as an elastic material with some stiffness coefficient(s). These algorithms break down ultimately if one tries to set this stiffness coefficient to infinite which corresponds to inextensible cloth. The crux of the method is an algorithm for interpolating a given set of arbitrary points or space curves by a smooth developable mesh surface. We formulate this interpolation problem as a mesh deformation process that transforms an initial developable mesh surface, e.g., a planar figure, to a final mesh surface that interpolates the given points (called anchor points). During the deformation process, all the triangle elements in the intermediate meshes are kept isometric to their initial shapes, while the potential energy due to the conservative force is reduced gradually. The collision problem is resolved by introducing *dynamic anchor points* owing to the collision during the deformation. Notwithstanding its simplicity, the proposed method has shown some promising efficacy for simulation of inextensible cloth.

Keywords Developable surface · Cloth simulation · Collision · Mesh deformation

M. Chen · K. Tang (✉)
Mechanical Department, Hong Kong University of Science and Technology, Hong Kong, Hong Kong
e-mail: mektang@ust.hk

M. Chen
e-mail: mecm@ust.hk

1 Introduction

Most clothing materials in our daily life are extremely stiff and hence should be considered inextensible: They do not stretch or compress. During a cloth deformation, this inextensibility means that any intermediate shape of the deformed cloth must be developable or nearly developable, i.e., it should be isometric or quasi-isometric to its original in-plane deformation-free shape. A realistic cloth solver thus should be able to handle this inextensible deformation.

As most existing algorithms [5, 7, 9, 25, 26, 29–33] are physically-based, a large elastic moduli or stiff springs are usually used to avoid overstretching in cloth. But this strategy would cause the condition number of the implicit system to inevitably grow with the elastic material stiffness and as a result significantly degrade the numerical stability and performance. Thus, for performance and stability consideration, most of elastic model-based methods generally allow some perceptible stretch deformation within a 10% change rate of edge lengths [8, 12, 25], which may make the final shape unrealistic (compare 0% and 15% in Fig. 1). For a broad survey on elastic physically-based cloth simulation, please refer to [10].

Only recently did a few noticeable works specifically target the developability issue in cloth simulation. A few schemes were proposed for lessening this difficulty and trying to enforce the length preservation. For a detailed and up-to-date review on these schemes, please refer to [17]. These schemes either try to use better numerical integration methods to overcome large stiffness or employ some forms of iterative process to reenforce the length preservation constraint. They, however, are remedial in nature and do not guarantee developability on the cloth.

Liu et al. [21] used Hamiltonian mechanics to model the dynamic behavior of cloth under a conservative force

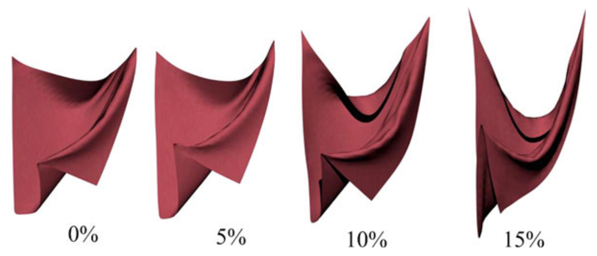

Fig. 1 Four final deformed shapes of different strain change rates: allowing large change rates (10% or more) will result in unrealistic deformation shapes

(the gravity), where the lengths of edges in the mesh are imposed as *hard* geometric constraints in the Lagrangian. Their model can ensure developability on the mesh surface. However, for an n-triangle mesh only $O(\sqrt{n})$ degrees of freedom are available for use in the integration; as a result, *locks* tend to appear during the integration and significant mesh-dependent artifacts often form on the final shape. More critically, these problems cannot be resolved by increasing n. Goldenthal et al. [17] presented an elaborate numerical solver for cloth simulation that builds on a framework of constrained Lagrangian mechanics. Since a quad-elements mesh is needed, their solver cannot be applied to a triangle mesh. Most recently, the isometric deformation problem is alleviated by English and Bridson [14], and the proposed method uses nonconforming elements in cloth simulation and the strain rate can be guaranteed to be below 1%. But the permission of nonconformity among triangles markedly increases the degree of freedom of the triangles and special rendering treatment is needed to eliminate the discontinuity due to nonconformity. Both methods in [14, 17] are complicated out of programming aspects.

In order to solve this problem, we introduce a robust and simple-to-implement method for the deformation of inextensible cloth that is subjected to a conservative force (e.g., the gravity) and collision-free requirement. The method can ensure a quasi-isometric deformation (close to zero). The crux of the method is an algorithm for interpolating an arbitrary set of 3D points by a smooth developable mesh surface. The whole algorithm is in principle an iterative process based on energy minimization, which is solved by Gauss-Newton iteration. The proposed algorithm formulates the minimization as a continuous shape deformation process that starts from some initial developable mesh surface (e.g., a planar figure) and ends on the final mesh surface that satisfies both the interpolation and developability requirements. Our algorithm is motivated by recent progress in mesh editing that employs the Laplacian operator [1, 2]. In a mesh editing environment that relies on the Laplacian operation, a mesh surface is continuously deformed while the Laplacian coordinates at all the vertices are being preserved as much as possible. Our algorithm works in a similar but broader sense: An initial developable mesh surface is continuously deformed until the three conditions are all satisfied: (1) all the designated vertices on the mesh surface (to be referred to as *anchor vertices*) coincide with their target positions; (2) the total gravitation potential energy of the mesh surface, if considered, can no longer be further reduced; and (3) the mesh surface is free of collision (including self-collision). During the entire deformation process, the lengths of all the edges on the triangle elements in the mesh are preserved. In a nutshell, in the deformation process, all three requirements (1)–(3) as well as the length preservation constraint are first linearized, which results in a large and overdetermined linear system, and then subsequently solved by an efficient least-squares method. By iteratively performing this procedure, the original mesh surface gradually and smoothly deforms to its target, while in all the time maintaining its developability. The contributions of this paper are: (1) treating the potential gravity field and collision avoidance as constraints, which provides an alternative way for the treatment of them and (2) ensuring the quasi-isometric deformation (close to zero), which is difficult to obtain by traditional methods. Although all the testing examples are presented as the final state of the cloth, it can be extended to dynamically simulate cloth if all the intermediate states of collision-free are collected as valid frames.

2 Related works

Intuitively, developable surfaces are those that can be unfolded into the plane with no length or area distortion. Developable surfaces are closely related to ruled surfaces, where a ruled surface is defined as the trajectory of a line (called a *ruling*) when it moves in space along two prescribed directrix curves [13, 24]. It is well known that a G^2 surface is developable if and only if it is a ruled surface whose normals are constant along each ruling [24]. Such a surface has a distinct characteristic: its normals on the Gaussian sphere form a continuous curve and is sometimes called a *torsal developable surface* [27]. A G^1 surface is sometimes called a *composite developable surface* if it is a union of some torsal developable surfaces. From differential geometry, a ruled surface is developable if and only if its Gaussian curvature is zero everywhere on the surface. In the following, we give a brief review of research work related to our developable surface interpolation task.

2.1 Exact surface interpolation

Almost all the works in exact surface interpolation deal with only torsal developable surfaces and are restricted to four-sides patches. Basically, an exact surface—parametric or algebraic—is sought to interpolate the given points or curves. Bézier or B-spline surfaces are the most used ones and the developability is enforced by nonlinear constraints [3, 4, 11, 19, 23]. To facilitate the modeling task, some novel algebraic tools were proposed. Pottmann [24] used a dual space approach to define a plane-based control interface for modeling developable patches. Bo and Wang [6] presented an interactive modeling scheme that can be used to simulate paper bending in computer animation. Notwithstanding its mathematical rigor and elegance, exact surface interpolation is considered ineffective and impractical for general interpolation, due to the extremely stringent and nonlinear constraints required.

2.2 Mesh surface interpolation

Frey [15, 16] introduced the concept of boundary triangulation—where all the triangles have their vertices lie on two 3D polylines—and presented a simple modeling method for generating a discrete (nearly) developable surface interpolating a given closed polyline. However, the method is restricted to height-field surfaces, i.e., surfaces must be monotone with respect to the XY-plane. The resulting surface depends on the choice of projection direction. Moreover, only a single polyline can be interpolated. Inspired by Frey's work, Wang and Tang [35] presented an approach that is capable of finding the most developable boundary triangulation interpolating two arbitrary polylines. Their method formulates the problem as a deterministic search problem and uses Dijkstra algorithm to perform the optimization, which is fast and robust. Nonetheless, their method can only generate a (discrete) torsal surface. Recently, based on boundary triangulation and convex hull principle, Rose et al. [27] gave an algorithm that is able to obtain a discrete (nearly) developable surface interpolating an arbitrary closed polyline. Though the algorithm can be expanded to more than one polyline, the resulting mesh usually is not smooth, that is, where two torsal surfaces meet does not have continuous normals.

2.3 Developable surface approximation

There exist a number of algorithms that aim at approximating a given mesh surface by one or more developable surfaces, continuous, or discrete. Wang and Tang [34] suggested deforming a given mesh to minimize its total Gaussian curvature, where a gradient-based search method is adopted for the minimization. Their algorithm, however, is not suitable for interpolation, since interpolation constraints were not considered in the deformation process. Mitani and Suzuki [22] introduced a method that approximates an arbitrary mesh surface by triangular strips, and the latter can be unfolded into their planar counterparts. Despite its usefulness in some particular applications, e.g., modeling paper craft toys; this method is not suitable for our interpolation problem, as the resulting mesh surface is only G^0—the places where triangular strips meet do not have continuous normals. Due to the same reason, the scheme introduced by Shatz et al. [28] is not suitable for us either, since their method approximates a mesh surface by conics which again cause discontinuity in normals. The algorithm proposed by Julius et al. [18], which is based on the Lloyd's segmentation scheme, partitions a mesh surface into (nearly) developable charts. The original mesh surface though is required to already interpolate the given boundary constraints. In their work primarily applicable to architectural design, Liu et al. [20] presented a development algorithm based on the use of planar quad strips. Despite its promise as a modeling and design tool, this algorithm nevertheless is not capable of dealing with general developable surface interpolation problem.

3 Preliminaries

The input to our developable cloth simulation algorithm consists of three items: a mesh model M, a set Φ of points in space that a subset Ψ of the vertices of M, called *anchor vertices*, must eventually coincide with, and a set of *obstacles* (represented as mesh models), $\Gamma = \{C_i : i = 1, 2, \ldots\}$, which the final shape of M must clear. Mesh M is defined by its vertices set $V = \{v_1, \ldots, v_n\}$, the edges set $E = \{e_i\}$, and the faces set F (in other words, the topological graph of M). Each vertex v_i in V is represented by its x, y and z coordinates, i.e., $v_i = [x_i, y_i, z_i]$. Set $\Phi = \{P_1, P_2, \ldots, P_m\}$ specifies m points in space that the anchor vertices on the final deformed mesh are required to coincide with, respectively, with $m \ll n$. Hereinafter, without loss of generality, it will be assumed that the coincidence correspondence specifies that, on the final deformed mesh, vertex v_i must coincide with P_i for $i = 1, 2, \ldots, m$. The final deformed mesh will be denoted as M' and the intermediate meshes in the deformation process from a given initial mesh M^0 to M' are M^i for $i = 1, 2, \ldots, N$ for some N, with $M^N = M'$. Each edge e in E is associated with an in-plane deformation-free length, to be denoted as $\ell_0(e)$. The following entities/terminologies are defined.

Isometric deformation In our inextensible cloth simulation problem, a stronger goal is sought: we want to preserve the lengths of the edges on M; in other words, the final mesh M' should be isometric to the in-plane deformation-free shape of M. To gauge this isometry, we adopt the summation:

$$E_L = \sum_{e \in E} \left| \ell(e) - \ell_0(e) \right|^2, \tag{1}$$

where $\ell(e)$ is the length of edge e on the current mesh M^i and $\ell_0(e)$ the original in-plane deformation-free length of e. A zero E_L obviously implies a zero area change, but not vice versa. It is the former that our iterative deformation process will try to minimize.

Measure of interpolation error The final interpolation constraint requires that vertex v_i of M' identifies with P_i, that is, $v_i = P_i$, for $i = 1, 2, \ldots, m$. The total interpolation error $E_I = \sum_{i=1}^{m} \|v_i - P_i\|^2$ of an intermediate M^k measures as a whole how close the vertices $\{v_i : i = 1, 2, \ldots, m\}$ are to their target positions $\{P_i : i = 1, 2, \ldots, m\}$. The deformation process gradually decreases the total interpolation error E_I until it becomes zero.

Measure of gravitation potential energy This measure, E_G, is used for the purpose of simulating the effect of gravity. Provided that E_I and E_L are minimized, a realistic shape of mesh M should be the one whose gravitation potential energy is the lowest as well. Assuming that our setting is such that the intermediate M is always above the ground Z = 0, the E_G is defined as: $E_G = (\sum_{i=m+1}^{n} m_i |z_i|)^2 = (\sum_{i=m+1}^{n} m_i z_i)^2$, where m_i is the mass associated with vertex v_i which is simply a third of the mass of sum of the masses of the incident triangles of v_i. Note that this potential energy excludes those of anchor vertices since they are considered fixed on the final shape M', and thus should not influence the deformation of M in the iterative process.

The final weighted form Combining all the three measures together, the final weighted total energy is $E_W = w_1 E_L + w_2 E_I + w_3 E_G$, where the three weights w_1, w_2, and w_3 are greater than 0. The deformation process, starting from the initial M^0, gradually moves the vertices in V to reduce E_W and eventually reaches the state when all three E_L, E_I, and E_G are minima. Therefore, we aim at solving the following minimization problem:

$$\arg\min_V (w_1 E_L + w_2 E_I + w_3 E_G) \tag{2}$$

4 Collisions and dynamic anchor points

In the deformation process, an intermediate mesh M^k may collide with some obstacles in Γ. Traditional treatment of collisions is to associate a dynamic repulsion force with the offending vertex that will prevent it from entering the obstacle or, if it is already inside the obstacle, push it out of the obstacle (c.f. [8]). Since in our geometric method no force would be used, the collision must be treated differently. Our solution is *dynamic anchor points*. An intermediate mesh M^k is allowed to collide with the obstacles. However, once a vertex v_i becomes an offending one, it will be marked as a *temporary anchor vertex* with a *temporary target position* P_i'. The P_i' is the position of v_i in the last iteration before v_i enters the obstacle. Vertex v_i then will participate in the ensuing deformation as a normal anchor vertex. Once v_i is found to be again clear of the obstacles, it will *immediately* be unmarked and becomes a normal vertex again. Let Ω represent the set of these dynamic anchor points. In our solution, all the *dynamic anchor vertices* are treated equally as those permanent *anchor vertices* in Ψ. Accordingly, the interpolation error E_I in (2) is then changed to

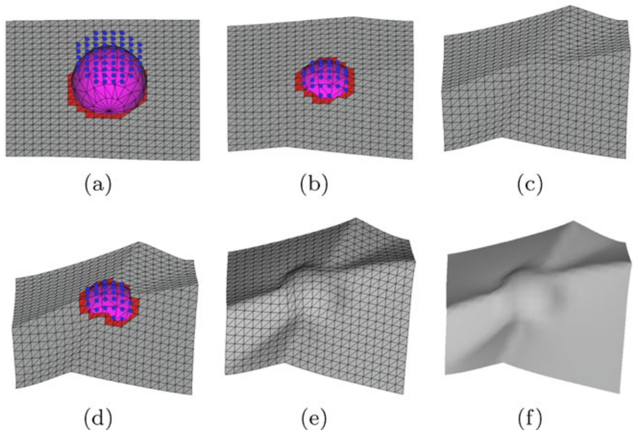

Fig. 2 Schematic of the use of dynamic anchor points in prevention of collisions: (**a**), (**b**), and (**d**) are three sequent intermediate states in which collision occurs but the number of dynamic anchor points (*red points*) are reduced gradually; (**c**) and (**e**) are the two sequent intermediate states in which all the inside points are dragged outside of the sphere and meanwhile its potential energy is lowered gradually. (These collision free states are valid frames if dynamic cloth simulation is required)

$$E_I = \sum_{i=1}^{m} \|v_i - P_i\|^2 + \sum_{v_i \in \Omega} \|v_i - P_i'\|^2. \tag{3}$$

To illustrate how these dynamic anchor points work, Fig. 2 displays a first few of intermediate shapes M^k after applying our cloth simulation algorithm to an initially flat piece of cloth with a spherical obstacle underneath. There are no permanent anchor points, so $\Psi = \emptyset$. The dynamic anchor points on M^k are colored red while their associated temporary target positions are colored blue. Notice the dynamic nature of Ω. After the iteration of Fig. 2(c), Ω became empty. However, the deformation process continually tried to lower the potential energy E_G, which caused some vertices again to become dynamic anchor points (i.e., the red triangles in Fig. 2(d)), and the iteration continued to try to pull these dynamic anchor vertices toward their temporary target positions, until once again the set Ω becomes empty, i.e., Fig. 2(e) (Fig. 2(f) is the shaded image of Fig. 2(e)). This iterative process then continued, with new equilibriums emerging and gone, until the total energy E_W was minimized (not shown in the figure). Note that if dynamic simulation is required the intermediate state of Fig. 2(c) and Fig. 2(e) can be valid frames.

The self-collision is treated in a similar manner, but with difference. There is no such term as “inside or outside” when describing self-collision, and perhaps self-intersection is a more proper term here. A popular way of dealing with self-collision in most cloth simulators is to backtrack to the last state immediately before the occurrence of the self-collision, add a repulsion force to the involved vertices, and then resume the integration but with a much smaller time step (refer to [8] and the references therein). We adopted the similar strategy, but this time using dynamic anchor points instead of repulsion force. Explicitly, once self-intersection is detected in M^k, we will: (1) restore to a previous M^l (l is not necessarily $k-1$); (2) mark those vertices of the triangles that intersect each other on M^k as dynamic anchor vertices whose temporary target positions are the vertices' corresponding positions in M^l; and (3) resume the minimization process from M^l. These temporary anchor vertices will be removed from Ω in the ensuing iterations once they are found free of self-intersection.

To expedite the intersection checking between M and the obstacles and also among M itself, the obstacles are defined as an aggregate of convex triangle meshes, while M is partitioned into a set of convex regions—a region on an M^k is *convex* if all the edges in it have the same sign of curvature (the sign of curvature of an edge is decided depending on whether its dihedral angle is smaller or larger than π). This convex partitioning is *dynamically* computed after each iteration. Since M is a surface but not a volume, a convex region on it might intersect itself. However, in our experiments, such situation is found to be extremely rare. Actually the results in all the test examples given in this paper are obtained with the assumption that self-intersection can never occur in any convex region of M.

5 Linearization and least-square solution

The deformation process deforms M while aims at minimizing the three total energies E_L, E_I, and E_G. Fittingly, this minimization can be cast in the following form:

$$\min \sum_{v_i \in V-\Psi} m_i z_i \tag{4}$$

subject to:

$$\delta(e) = \ell(e) - \ell_0(e) = 0, \quad e \in E \tag{5}$$

$$\lambda(v_i) = v_i - T(v_i) = \mathbf{0}, \quad v_i \in \Psi + \Omega \tag{6}$$

where $T(v_i)$ represents the target position of anchor vertex v_i, be it permanent or temporary.

A tempting approach to solving the above constrained optimization problem is to post the constraints as penalty functions with Lagrangian multipliers and merge them with the objective function into a single function which will then be solved iteratively using some gradient-based or heuristic method. However, through our experiments this approach was found to be unsuitable, primarily due to two concerns. First, because the set Ω is dynamic, so are the constraints associated with it in (6); this makes the search process very unstable. Then even more critically, while minimum gravitational potential energy is desired, our primary objective is to uphold the length-preservation and satisfy the interpolation and collision-free requirements, i.e., (5) and (6)—treating them as weighted penalty functions may diminish this precedence.

Our solution is to treat the minimization of E_G also as a constraint like (5) and (6) and iteratively solve the three. To devise a suitable form of constraint for the minimization of E_G, we adopted the idea of *maximum allowable displacement*. Consider a vertex $v_i \in V - \Psi$, let $l_1, l_2, \ldots, l_j$ be the lengths of its incident edges in the current M^k. With the one-ring neighboring vertices of v_i unchanged, lowering v_i would alter the sum $\sum_{i=1}^{j} l_i$. We define the γ_i of v_i to be the displacement in the $-Z$-direction for v_i before the sum $\sum_{i=1}^{j} l_i$ would be changed by ϕ percent, where ϕ is prespecified (15 in our implementation). The corresponding constraint of v_i for E_G then is given as

$$\Delta z_i = -\gamma_i \tag{7}$$

The left-hand sides of all the three equations (5)–(7) are functions of v_i. Trying to directly solve (5)–(7) is doomed to fail, as these are nonlinear and over-determined equations. Our approach is to iteratively linearize them and then solve the resultant linear system using the powerful least-squares method. Let $X = [x_1, x_2, \ldots, x_n, y_1, y_2, \ldots, y_n, z_1, z_2, \ldots, z_n]^T$ be the column vector representing the XYZ coordinates of the n vertices. In a nutshell, let X_0 be the solution to the (5)–(7) at the current iteration. We amend X by a small difference δX, that is, $X' = X_0 + \delta X$, such that X' is better than X_0 in (globally) reducing E_W. This process is then repeated with X' as the new X_0 until a satisfactory solution is obtained for the three equations.

5.1 Linearization

Taylor expansion was used for the linearization of (5) (the other two equations are already linear). We did not, however, expand it directly with respect to $[x_i, y_i, z_i]$ owing to the concern that such a direct expansion may cause inefficiency problem, as the δX must be extremely small in every iteration in order for the iteration to converge. For an edge e, its length $\ell(e)$ is determined by its two vertices $v_{i1} = [x_{i1}, y_{i1}, z_{i1}]$ and $v_{i2} = [x_{i2}, y_{i2}, z_{i2}]$, i.e., $\ell(e) = \|v_{i1} - v_{i2}\|$. Consider the change of the length along the Z-axis (the analysis for the X- and Y-axes is similar).

Suppose z_{i1} and z_{i2} are both updated to $z_{i1} + \Delta z_{i1}$ and $z_{i2} + \Delta z_{i2}$, respectively. We expand $\delta(e)$ in terms of the difference $(z_{i1} - z_{i2})$ instead of z_{i1} and z_{i2} themselves, and establish for (5) the following Taylor expansion:

$$\delta(\Delta z_{i1} - \Delta z_{i2}) = \delta(0) + \frac{(z_{i1} - z_{i2})}{\ell(e)}(\Delta z_{i1} - \Delta z_{i2}) + O\big(|\Delta z_{i1} - \Delta z_{i2}|^2\big)$$

Obviously, in general, the magnitude of $(\Delta z_{i1} - \Delta z_{i2})$ is much smaller than that of Δz_{i1} or Δz_{i2} themselves, which was found in our experiments to be extremely helpful in enhancing the efficiency of the program. Note that the final entire system equation (11) will have the same form as we expand (8) directly with respect to z, but after calculating the final Δz_i value, we only need to check that the value $(\Delta z_{i1} - \Delta z_{i2})$ instead of Δz_i is small enough and it can ensure that the linearization is accurate enough.

For the other two equations (6) and (7), similarly, they can be reformulated to be in relation to Δx, Δy, and Δz. (Actually (7) is already in the Δz form.) The three constraint equations (5)–(7); therefore, are linearized respectively into the following three linear equations (in Z-direction only):

$$(\Delta z_{i1} - \Delta z_{i2})\frac{(z_{i1} - z_{i2})}{\ell(e)} = \ell_0(e) - \ell(e), \quad e \in E \tag{8}$$

$$\Delta z_i = \big(z\big(T(v_i)\big) - z_i\big), \quad v_i \in \Psi + \Omega \tag{9}$$

$$\Delta z_i = -\gamma_i, \quad v_i \in V - \Psi \tag{10}$$

where we use $z(P)$ to denote the Z coordinate of point P ($x(P)$ and $y(P)$ are similarly defined). Note that of the above three, (8) and (9) are defined for X and Y as well.

It is interesting to note that a more intuitive form for the interpolation constraint, as compared to (6), would be the L_2 measure $\lambda(v_i) = \|v_i - T(v_i)\|^2$, since it is the 3D distance that matters. From our trials, however, it is found that replacing (6) by this L_2 form would receive no improvement in the quality of the final mesh (in terms of E_W), while using (6) tremendously reduced the computing time (since it is already linear), and improved the numerical stability as well.

5.2 Least squares solution

Now that (8)–(10) are all linear in Δz_i, letting $\Delta Z = [\Delta z_1, \Delta z_2, \ldots, \Delta z_n]^T$, we construct a (large) sparse system of linear equations as:

$$L\Delta Z = \begin{pmatrix} A \\ B \\ C \end{pmatrix} \Delta Z = \boldsymbol{b} \quad \Leftrightarrow$$

$$\begin{pmatrix} [w_1 & -w_1 & 0 & 0] \\ [0 & 0 & w_2 & 0] \\ [0 & 0 & 0 & w_3] \end{pmatrix} \begin{pmatrix} \begin{bmatrix} \Delta z_{i1} \\ \Delta z_{i2} \end{bmatrix} \\ [\Delta z_i] \\ [\Delta z_i] \end{pmatrix} \tag{11}$$

$$= \begin{pmatrix} [w_1 \frac{(\ell_0 - \ell(e))\ell(e)}{((z_{i1} - z_{i2}))}] \\ [w_2(z(T(v_i)) - z_i)] \\ [-w_3\gamma_i] \end{pmatrix}$$

where A is an $n_1 \times n$ matrix (n_1 is the number of edges in E), B an $(m + m_1) \times n$ matrix (m_1 is the number of *temporary* anchor vertices in Ω), and C an $n_2 \times n$ matrix ($n_2 = n - m - m_1$), and $\boldsymbol{b}$ an $(n + n_1)$ column vector. Similar linear systems involving ΔX and ΔY for the x- and y-coordinates are also established, with the exception that the submatrix C is no longer applicable and the size of column vector $\boldsymbol{b}$ will be shrunk to $n_1 + m + m_1$. The weights w_1, w_2, and w_3 are specified by the user or automatically determined by the program, which balance the weight among the three energies during the deformation. Their values play an important role in program efficiency as well as the final result, and the strategy for their choice will be explained in the following section.

Obviously, the linear system (11) is overdetermined ($n + n_1 > n$ for z and $n_1 + m + m_1 > n$ for x and y), and hence any attempt at finding the exact solution would be futile. We solve it in the least-squares sense as:

$$\Delta Z = \big(L^T L\big)^{-1} L^T \boldsymbol{b} \tag{12}$$

It is noted that the matrix $L^T L$ is sparse, positive definite and symmetric. Therefore, it is possible to compute a Cholesky factorization of $L^T L$, which helps solve (11) more efficiently; that is, $L^T L = R^T R$, where R is an upper-triangular sparse matrix. After the factorization, ΔZ is obtained by the back substitutions:

$$R^T S = L^T \boldsymbol{b}, \qquad R\Delta Z = S$$

In our implementation, we find that major computing time is taken in solving (11). When the problem is very large, it will be more efficient to use iterative techniques such as the conjugate gradient method to solve (11). For conjugate method, at each iteration, matrix vector multiplication with $L^T L$ is performed, as matrix L is very sparse, thus optimization on $L^T L$ and $L^T \mathbf{b}$ will significantly improve the program performance.

6 The final algorithm

The final algorithm, **UpdateMesh**, for the optimization problem of (2) is of recursive type: it takes as input the x, y and z coordinates, X_0, and the dynamic anchor vertices Ω of the current mesh of M, makes an amendment to X_0 and Ω, and calls itself again with the new X_0 and Ω, until the termination criteria are met. In addition to X_0 and Ω, the system maintains a mesh X^f (and its Ω set Ω^f) that is the *most recent* M that is free of self-intersection. We also use X_0^{-1} to denote the counterpart of the current X_0 in the previous iteration.

UpdateMesh (X_0, Ω)

```
Begin
Step 1:  If (Termination) then Exit
Step 2:  Determine weights $w_1, w_2$ and $w_3$
Step 3:  Linearize (5) and establish (8)–(10) for $X, Y$, and $Z$
Step 4:  Construct the linear system (11) for $X, Y$, and $Z$
Step 5:  Use the least-squares algorithm as specified in (12) to
         solve the overdetermined linear system from Step 4 for
         $X, Y$, and $Z$ independently (the results are in $\Delta X, \Delta Y$,
         and $\Delta Z$, respectively)
Step 6:  $X_0 \longleftarrow X_0 + [\Delta X^T, \Delta Y^T, \Delta Z^T]^T$
Step 7:  For every $v \in \Omega$, Do {
             If $v$ was but no longer is a self-intersection vertex,
             remove it from $\Omega$;
             If $v$ was but no longer is an obstacle-collision vertex,
             remove it from $\Omega$ }
Step 8:  For every $v \notin \Omega$, Do {
             If $v$ is a self-intersection vertex, add it to $\Omega$ with
             $T(v)$ being the coordinates of $v$ in $X^f$;
             If $v$ is an obstacle-collision vertex, add it to $\Omega$ with
             being the coordinates of $v$ in $X_0^{-1}$ }
Step 9:  If ($\Omega$ contains no self-intersection vertices), Do {
             $X^f \leftarrow X_0$
             $\Omega^f \leftarrow \Omega$ }
Step 10: If (originally $\Omega$ contained no self-intersection vertices
         but some self-intersection vertex is added to $\Omega$ in
         Step 8), Do {
             $X_0 \leftarrow X^f$
             $\Omega \leftarrow \Omega^f$ }
Step 11: Go to Step 1
End
```

Like any iterative algorithm, ours has some important *system control parameters*, i.e., the three weights w_1, w_2, and w_3. As seen in the algorithm, these weights are not static—they are decided dynamically in every iteration. Conceivably, a too large w_1 may diminish the influence of E_I and E_G in the total energy E_W and consequently adversely affects the conversion rate of the interpolation. On the other hand, if w_1 is too small, the anchor vertices might quickly move to their target positions while the energy E_L still remains large. The algorithm **AdustParameter** given next calculates the dynamic adjustment of the three, which is called in Step 2 in **UpdateMesh**. The three weights are calculated as percentages that always sum to 1—thus, from the concern of numerical stability, in our implementation they are multiplied by a common factor 500 in (11). To facilitate gauging the progress of the energy reduction, the program calculates and maintains three status variables r_L, r_I, and r_G which are respectively the *average reduction rates* of E_L, E_I, and E_G over the last l iterations (where $l = 5$ in our implementation). Additionally, to prevent over bias, the three weights are all capped by some maxima, i.e., $w_1 < w_1^{\max}$, $w_2 < w_2^{\max}$, and $w_3 < w_3^{\max}$, with $w_1^{\max} > w_2^{\max} > w_3^{\max}$ (80%, 60%, and 40%, respectively, in our system).

AdjustParameter

```
Begin
Step 1.1: If ($r_L <$ minimum_rate_of_$E_L$ and $E_L > \epsilon_L$), Do
              $w_1 \leftarrow 1.75 \cdot w_1$
Step 1.2: If ($r_I <$ minimum_rate_of_$E_I$ and $E_I > \epsilon_I$), Do
              $w_2 \leftarrow 1.5 \cdot w_2$
Step 1.3: If ($r_G <$ minimum_rate_of_$E_G$ and $E_G > \epsilon_G$), Do
              $w_3 \leftarrow 1.25 \cdot w_3$
Step 2:       $w_i \leftarrow \frac{w_i}{w_1+w_2+w_3}$
Step 3.1: If ($w_1 > w_1^{\max}$), Do
              $w_1 \leftarrow w_1^{\max}$
Step 3.2: If ($w_2 > w_2^{\max}$), Do
              $w_2 \leftarrow w_2^{\max}$
Step 3.3: If ($w_3 > w_3^{\max}$), Do
              $w_3 \leftarrow w_3^{\max}$
Step 4:   If (any of $w_1$, $w_2$, or $w_3$ was altered in Step 3), Do
              $w_i \leftarrow \frac{w_i}{w_1+w_2+w_3}$
End
```

In **AdjustParameter**, the three thresholds ϵ_L, ϵ_I, and ϵ_G are prespecified system constants that gauge the success of the minimizations of the three energies, respectively. In Steps 1.1–1.3, for each weight, we first check if the most recent (average) reduction speed of its corresponding energy is satisfactory and, if not, enlarge it by a factor. The strict decreasing order of the factors, 1.75, 1.5, and 1.25, helps uphold our precedence of the minimization of the three types of energy—note that even if all the three get enlarged, eventually w_1 will become more and more dominant over the other two, so will w_2 over w_3. The three numbers minimum_rate_of_E_L, minimum_rate_of_E_I, and minimum_rate_of_E_G are also

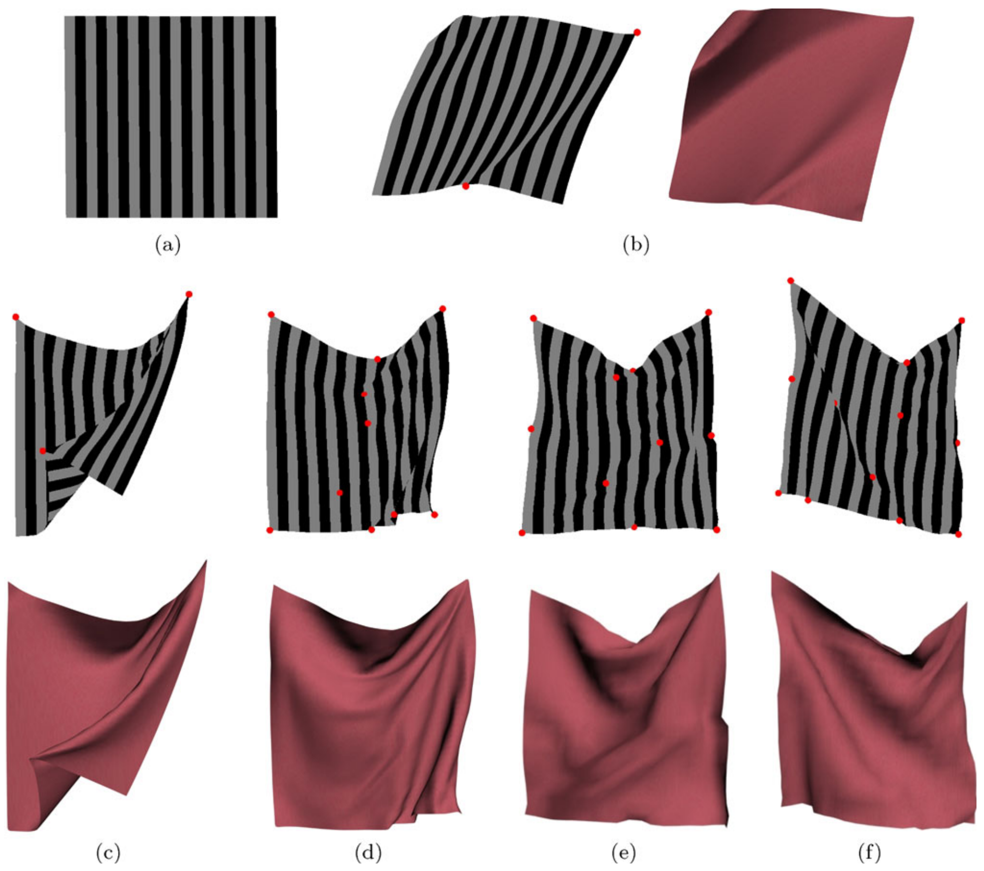

Fig. 3 Example I (draping cloth without obstacles): the original piece of flat cloth (**a**); the deformed cloth with three anchor vertices but ignoring the gravity (**b**); and the deformed cloth subjected to different patterns of anchor vertices and also the gravity (**c**)–(**f**). The anchor vertices are colored *red*

preset system constants. The three possibly updated weights are then normalized in Step 2, and then checked against their maximally allowed values in Step 3 (note that at most one of the three can be altered in Step 3). Finally, in case there is a change to one of the three weights in Step 3, they are once again normalized in Step 4.

The terminal condition (Step 1 in **UpdateMesh**) is relatively simple in our current implementation—either (Condition A) when *both* energies E_L and E_I are smaller than their respective thresholds *and* E_G has not been improved over a certain period (15 iterations in our implementation) or (Condition B) when the number of iterations has reached a certain preset limit (200 in our implementation). Before updating vertices V after iteration k, we will check the length edge change, if any edge length change exceeds preset threshold (in our implementation, it is 0.5%), we will restore to M^{k-1} and then double the value of w_1 to continue. As the length change rate is guaranteed small during deformation, the residual value after linearization can be regarded as zero, which will let our algorithm converge quadratically to solve (2).

In practice, for some irregular mesh models, zero-length change may result in lock problem. Once lock occurs, we will allow a little more edge length change by reducing the length preservation weight, i.e., w_1, to solve this problem.

7 Experimental results and discussion

A prototype of the proposed developable cloth simulation algorithm has been implemented and a batch of examples has been tested. All the tests were performed on a P4, 3.0 GHz, and 1 GB RAM PC. In this section, several test examples are given. To gauge the efficacy of the proposed algorithm, aside from E_I, E_L, and E_G, the area change ratio E_A is introduced, which is defined as $E_A = |Area(M') - Area(M^0)|/Area(M^0)$. All the related statistics data is listed in Table 1.

The first example, shown in Fig. 3, shows the application of our algorithm to a classic cloth simulation problem, the draping of a square piece of cloth under gravity. The initial cloth lies in a plane parallel to XY. Different patterns of anchor vertices and their target positions are then applied and the corresponding final deformed shapes of the cloth are depicted in Fig. 3(c)–(f). As a comparison, Fig. 3(b) shows the final shape of the cloth that is subjected to the same anchor constraints as in Fig. 3(c) but the gravity is ignored, with their respective gravity potential change $\sum \Delta z_i$—$+12956$ and -3929—given in Table 1.

Table 1 Computing statistics of the test examples

Example	Number of vertices	Running time	Result figure	E_I	E_L	$\sum \Delta z_i$	E_A	Iteration times
I	961	28.33 s	Fig. 3(b)	6.14×10^{-4}	2.33×10^{-3}	12,959	2.00×10^{-3}	56
		53.28 s	Fig. 3(c)	8.40×10^{-4}	4.74×10^{-3}	−3,929	2.25×10^{-3}	92
		18.62 s	Fig. 3(d)	7.66×10^{-3}	2.11×10^{-3}	−6,949	1.91×10^{-3}	36
		12.55 s	Fig. 3(e)	7.11×10^{-3}	2.12×10^{-3}	−6,410	2.19×10^{-3}	23
		15.82 s	Fig. 3(f)	6.26×10^{-3}	2.31×10^{-3}	−16,888	1.93×10^{-3}	32
II	923	31.60 s	Fig. 4(a)	1.16×10^{-4}	2.14×10^{-3}	−32,721	1.71×10^{-3}	158
		40.20 s	Fig. 4(b)	2.55×10^{-4}	2.63×10^{-3}	−25,532	2.53×10^{-3}	200
		40.11 s	Fig. 4(c)	3.12×10^{-4}	2.31×10^{-3}	−21,508	1.85×10^{-3}	200
		41.30 s	Fig. 4(d)	7.31×10^{-4}	2.37×10^{-3}	−18,976	2.32×10^{-3}	200
		41.30 s	Fig. 4(e)	1.31×10^{-4}	2.33×10^{-3}	−28,745	2.11×10^{-3}	200
III	1361	21.10 s	Fig. 5(b)	1.09×10^{-5}	2.56×10^{-3}	−3,622	7.14×10^{-3}	32
		24.83 s	Fig. 5(c)	3.11×10^{-5}	3.52×10^{-3}	3,011	2.15×10^{-3}	38
		114.90 s	Fig. 5(d)	7.70×10^{-4}	3.34×10^{-3}	4,771	3.13×10^{-3}	176
		69.55 s	Fig. 5(e)	6.28×10^{-5}	3.17×10^{-3}	3,125	2.07×10^{-3}	107
		84.41 s	Fig. 5(f)	5.10×10^{-5}	4.15×10^{-3}	2,160	3.12×10^{-3}	132

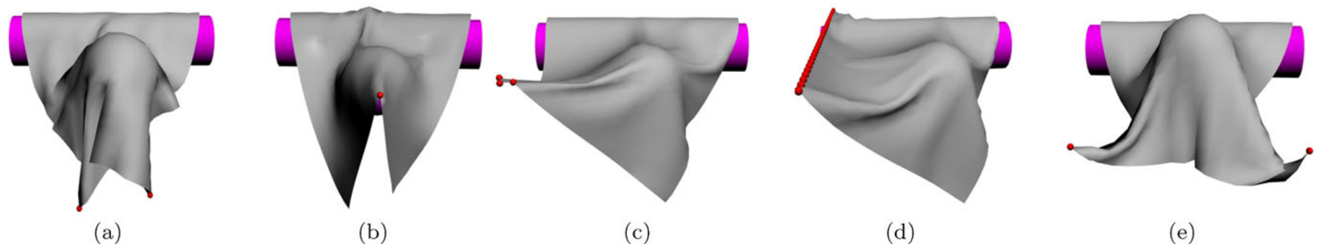

Fig. 4 Example II (draping cloth with obstacles): the final deformation shapes subjected to some different patterns of anchor vertices and target positions and the initial mesh model is a square cloth, which drops over obstacles with the effect of gravitation potential energy. All the permanent anchor vertices are colored *red*

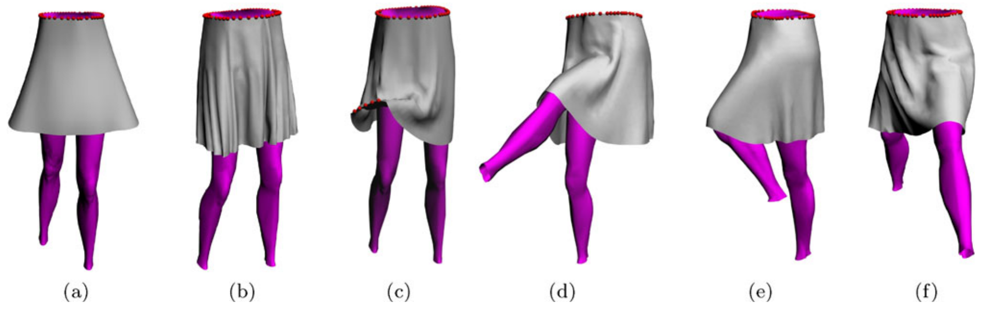

Fig. 5 Example III (skirt model deformation): the initial shape (**a**); the deformation shapes with different postures and anchor points and target positions (**b**)–(**f**)

The last two examples seek to find the final steady deformation shapes under various geometric and anchor point constraints. From Fig. 4 and Fig. 5, one can find that all the final shape is free of collision and the statistics given in Table 1 show that all the deformed shapes are isometric to their initial shapes, i.e., their E_L and E_A remain zero (within tolerance) throughout the entire deformation process.

8 Conclusions

We have presented a new approach to the computer simulation of inextensible cloth. Specific advantages of this new method include:

- It is purely geometric and does not involve stiffness coefficients or elastic moduli in the problem formulation;

consequently, it is able to, at least theoretically, achieve 100% inextensibility on the cloth.
- The algorithm is extremely simple and its implementation is easy and straightforward.
- The algorithm is free of special handling of any types of geometric degeneracy or artifacts, and hence is robust and numerically stable.
- The collision-avoidance and boundary constraints are handled by the same procedure, which further simplifies the algorithm and enhances its numerical stability.

However, notwithstanding our initial promising test results, the proposed method has several unsolved key issues. First, it can handle only conservative forces, and thus is not suitable when friction or other types of nonconservative forces are considered. Second, by its nature, the method is only able to find one *final* (i.e., steady) shape of the cloth, but not its dynamic behavior (as a function of *time t*). Third, the material properties of the cloth are not considered—note that even if being 100% inextensible the cloth can have varying stiffness along the shear direction. Finally, it is yet not clear how our method will fare in extreme geometric situations, e.g., when the cloth must incur deep and crowded folds. We hope that the work reported in this paper can at least spur the research in cloth simulation along this new direction.

References

1. Au, O.K.C.: Differential techniques for scalable and interactive mesh editing. Ph.D. thesis, Hong Kong University of Science and Technology (2007)
2. Au, O.K.C., Fu, H., Tai, C.L., Cohen-Or, D.: Handle-aware isolines for scalable shape editing. ACM Trans. Graph. **26**(3), 83:1–83:10 (2007)
3. Aumann, G.: Interpolation with developable Bézier patches. Comput. Aided Geom. Des. **20**(8–9), 601–619 (2003)
4. Aumann, G.: Degree elevation and developable Bézier surfaces. Comput. Aided Geom. Des. **21**(7), 661–670 (2004)
5. Baraff, D., Witkin, A.: Large steps in cloth simulation. In: Computer Graphics Proceedings. Annual Conference Series, pp. 43–54 (1998)
6. Bo, P., Wang, W.: Geodesic-controlled developable surfaces for modeling paper bending. Comput. Graph. Forum **26**(3), 329–338 (2007) (EuroGraphics 2007)
7. Breen, D.E., House, D.H., Wozny, M.J.: Predicting the drape of woven cloth using interacting particles. In: Computer Graphics Proceedings. Annual Conference Series, pp. 365–372 (1994)
8. Bridson, R., Fedkiw, R., Anderson, J.: Robust treatment of collisions, contact and friction for cloth animation. ACM Trans. Graph. **21**(3), 594–603 (2002)
9. Choi, K.J., Ko, H.S.: Stable but responsive cloth. In: Computer Graphics Proceedings. Annual Conference Series, pp. 604–611 (2002)
10. Choi, K.J., Ko, H.: Research problems in clothing simulation. Comput. Aided Des. **37**(6), 585–592 (2005)
11. Chu, C., Séquin, C.: Developable Bézier patches: Properties and design. Comput. Aided Des. **34**(7), 511–527 (2002)
12. Desbrun, M., Schröder, P., Barr, A.: Interactive animation of structured deformable objects. In: Graphics Interface, pp. 1–8 (1999)
13. doCarmo, M.: Differential Geometry of Curves and Surfaces. Prentice-Hall, Englewood Cliffs (1976)
14. English, E., Bridson, R.: Animating developable surfaces using nonconforming elements. ACM Trans. Graph. **27**(3), 66:1–66:5 (2008)
15. Frey, W.: Boundary triangulations approximating developable surfaces that interpolate a closed space curve. Int. J. Found. Comput. Sci. **13**, 285–302 (2002)
16. Frey, W.: Modeling buckled developable surface by triangulation. Comput. Aided Des. **36**(4), 299–313 (2004)
17. Goldenthal, R., Harmon, D., Fattal, R., Bercovier, M., Grinspun, E.: Efficient simulation of inextensible cloth. ACM Trans. Graph. **26**(3), 49:1–49:7 (2007)
18. Julius, D., Kraevoy, V., Sheffer, A.: D-charts: Quasi-developable mesh segmentation. Comput. Graph. Forum **24**(3), 581–590 (2005)
19. Lang, J., Röschel, O.: Developable (1,n)-Bézier surfaces. Comput. Aided Geom. Des. **9**(4), 291–298 (1992)
20. Liu, Y., Pottmann, H., Wallner, J., Yang, Y.L., Wang, W.: Geometric modeling with conical meshes and developable surfaces. ACM Trans. Graph. **25**(3), 681–689 (2006)
21. Liu, Y., Tang, K., Joneija, A.: Modeling dynamic developable meshes by Hamilton principle. Comput. Aided Des. **39**(9), 719–731 (2007)
22. Mitani, J., Suzuki, H.: Making papercraft toys from meshes using strip-based approximate unfolding. ACM Trans. Graph. **23**(3), 259–263 (2005)
23. Pottmann, H., Farin, G.E.: Developable rational Bézier and b-spline surfaces. Comput. Aided Geom. Des. **12**(5), 513–531 (1995)
24. Pottmann, H., Wallner, J.: Computational Line Geometry. Springer, Berlin (2001)
25. Provot, X.: Deformation constraints in a mass-spring model to describe rigid cloth behavior. In: Graphics Interface, pp. 147–154 (1995)
26. Provot, X.: Collision and self-collision handling in cloth model dedicated to design garment. In: Graphics Interface, pp. 177–189 (1997)
27. Rose, K., Sheffer, A., Wither, J., Cani, M.P., Thibert, B.: Developable surfaces from arbitrary sketched boundaries. In: Proceedings of the Fifth Eurographics Symposium on Geometry Processing, vol. 257, pp. 163–172 (2007)
28. Shatz, I., Tal, A., Leifman, G.: Paper craft models from meshes. Vis. Comput. **22**(9), 825–834 (2006)
29. Terzopoulos, D., Platt, J., Barr, A., Fleischert, K.: Elastically deformable models. In: Proceedings of ACM SIGGRAPH, vol. 21, pp. 205–214 (1987)
30. Volino, P., Magnenat-Thalmann, N.: An evolving system for simulating clothes on virtual actors. IEEE Trans. Graph. Appl. **16**(5), 42–51 (1996)
31. Volino, P., Magnenat-Thalmann, N.: Comparing efficiency of integration methods for cloth simulation. In: Proceedings of Computer Graphics International 2001, pp. 265–272 (2001)
32. Volino, P., Magnenat-Thalmann, N.: Stop-and-go cloth draping. Vis. Comput. **23**(8), 669–677 (2007)
33. Volino, P., Courchesne, M., Magnenat-Thalmann, N.: Versatile and efficient techniques for simulating cloth and other deformable objects. In: Proceedings of ACM SIGGRAPH, vol. 29, pp. 137–144 (1995)
34. Wang, C.C.L., Tang, K.: Achieving developability of a polygonal surface by minimum deformation: a study of global and local optimization approaches. Vis. Comput. **20**(8–9), 521–539 (2004)
35. Wang, C.C.L., Tang, K.: Optimal boundary triangulations of an interpolating ruled surface. ASME J. Comput. Inform. Sci. Eng. **5**(4), 291–301 (2005)

Ming Chen received the B.S. and a Master degree in mechanical engineering from the Huazhong University of Science and Technology, P.R. China in 2001 and 2004, respectively. He is currently a research association in CAD/CAM Lab, Hong Kong University of Science and Technology, Hong Kong, P.R. China. Has also worked as a software specialist and consultant in the CAD/CAM/ERP software industry for years. His research interests are about solid modeling and numerical optimization.

Kai Tang received the B.Eng. degree in mechanical engineering from the Nanjing Institute of Technology, Nanjing, China, in 1982 and the M.Sc. degree in information and control engineering and a Ph.D. in computer engineering from the University of Michigan, Ann Arbor, in 1986 and 1990, respectively. He is currently a faculty member in the Department of Mechanical Engineering, Hong Kong University of Science and Technology (HKUST), Hong Kong. Before joining HKUST in 2001. He had worked for more than 13 years in the CAD/CAM and IT industries. His research interests concentrate on designing efficient and practical algorithms for solving real-world computational, geometric, and numerical problems. He is a member of the IEEE.