Visual Comput (2005) 21: 279–288
DOI 10.1007/s00371-005-0282-6

ORIGINAL ARTICLE

Ye Duan
Jing Hua
Hong Qin

Interactive shape modeling using Lagrangian surface flow

Published online: 24 May 2005

Ye Duan (✉)
Department of Computer Science
University of Missouri at Columbia
Columbia, MO 65203, U.S.A.
e-mail: duanye@missouri.edu

Jing Hua
Department of Computer Science
Wayne State University
Detroit, MI 48202, U.S.A.
e-mail: jinghua@cs.wayne.edu

Hong Qin
Department of Computer Science
State University of New York at Stony Brook
Stony Brook, NY 11794, U.S.A.
e-mail: qin@cs.sunysb.edu

Abstract In this paper, we propose a new shape-modeling paradigm based on the concept of Lagrangian surface flow. Given an input polygonal model, the user interactively defines a distance field around regions of interest; the locally or globally affected regions will then automatically deform according to the user-defined distance field. During the deformation process, the model can always maintain its regularity and can properly modify its topology by topology merging when collisions between two different parts of the model occur. Comparing with level-set based methods, our algorithm allows the user to work directly on existing polygonal models without any intermediate model conversion. Besides closed polygonal models, our algorithm also works for mesh models with open boundaries. Within our framework, we developed a number of shape-modeling operators including blending, cutting, drilling, free-hand sketching, and mesh warping. We applied our algorithm to a variety of examples that demonstrate the usefulness and efficacy of the new technique in interactive shape design and surface deformation.

Keywords Shape modeling · Partial differential equations (PDEs) · Geometric surface flow · Distance field

1 Introduction

Mesh models have been prevalently used in geometric modeling, computer graphics, and visualization. Creating interesting and elegant mesh models is a very active research area. The advancement of 3D laser range-scanning technology has created a huge demand for efficient algorithms to handle meshes that consist of millions or even billions of vertices. To date, many algorithms for editing polygonal meshes have been developed. In general, they are either static, geometric algorithms, or dynamic, physics-based algorithms. Static, geometric algorithms often require a lot of user interventions that are tedious and laborious in general. Physics-based algorithms are more intuitive for the user to manipulate. However, conventional physics-based algorithms are relatively slow and do not scale well to large-scale models.

Recently, researchers have begun to apply the concept of geometric surface flow for the purpose of mesh processing – for example, in the work of Taubin [23], Desbrun et al. [4], Kobbelt et al. [10, 19], and Ohtake et al. [15, 16]. Besides its strong mathematical foundations in areas such as partial differential equations (PDEs) and calculus of variation, the main appeal of a geometric surface-flow-based algorithm is its simplicity: all of the computations can be conducted by local iterative equations. Hence, in principle, geometric surface-flow-based algorithms are very suitable for the processing of very large-scale polygonal meshes.

Nonetheless, most of the existing surface-flow-based algorithms have mainly been focused on problems such as mesh-fairing, mesh-smoothing and mesh optimization.

To the authors' best knowledge, the idea of applying surface flow directly for mesh editing was first proposed by Ohatake et al. [15]. They demonstrated some interesting embossing and engraving results achieved by the surface flow. However, in their work, no topology change is allowed. Most recently, Museth et al. [14] proposed the use of a level-set-based surface flow for implicit surface editing. Instead of directly working on the meshes, they first convert the polygonal meshes into a signed distance field. The shape is then implicitly manipulated by applying certain level-set-based editing tools on the distance field. Finally, an isosurface extraction algorithm such as the marching cube algorithm [12] is employed to convert the implicitly edited shape back into polygonal meshes. Implicit, level-set methods such as the one proposed by Museth et al. [14] have become very popular recently, mainly because they easily handle topology changes. However, the efficiency, user-interactivity, and multiresolution capability of level-set models is usually not as good as explicit models. Most importantly, open manifolds with boundaries remain challenging for implicit representations.

Our work further expands the basic principle presented by Ohatake et al. [15] with much more editing flexibility. In addition to local embossing and engraving effects, our technique also supports operations such as blending, cutting, drilling as well as free-hand sketching, etc. Starting with any polygonal model, the user interactively defines a distance field around regions of interest in the mesh model; the locally or globally affected regions of the model will then automatically deform according to the user-defined distance field. The deformation behavior of the model is guided by a PDE-based surface flow. During the deformation process, the model will always maintain its regularity and can properly modify its topology by topology merging when collisions between two different parts of the model occur.

The rest of the paper is organized as follows: after reviewing the related works in Sect. 2, we will introduce the PDE-based surface flow and its numerical simulation in Sects. 3 and 4, respectively. Various kinds of surface-flow-based mesh editing operations will be discussed in Sect. 5. Our collision detection and topology modification schemes are explained in Sects. 6 and 7, respectively. Finally, we conclude the paper and point out some future research directions in Sect. 8.

2 Related work

2.1 Distance-field-based shape modeling

A distance field is a scalar function that specifies the minimum distance to a shape, where the distance may be assigned a sign to distinguish between the inside and outside of the shape. The distance field is an effective representation of shape. It has been used to generate swept volumes [20], shape deformation [3, 22], offset surfaces [17], and to morph between surface models [2, 17]. Regularly sampled distance fields have drawbacks because of their size and limited resolution. Frisken et al. [5] overcame this limitation by proposing adaptively sampled distance fields (ADF). ADFs consists of adaptively sampled distance values organized in a spatial hierarchy of data structures. Octree-based ADFs were later incorporated into a prototype sculpting system called "Kizamu" [18] developed by Perry and Frisken. Recently, Shapiro et al. [21] proposed the use of R-functions to construct distance fields with guaranteed differential properties.

Since the shapes in these distance-field-based techniques are all implicitly defined, an isosurface extraction algorithm, such as the marching-cube algorithm [12], is always needed to convert the implicit shape to explicit polygonal representations. Hence, the quality of the model is always limited by the resolution of the underlying grid. In this paper, we propose a new algorithm that can create meshes of much higher quality (through the use of mesh optimization and Laplacian smoothing). More importantly, the new algorithm takes advantages of both the distance field representation and the explicit polygonal representation and allows the user to directly work on explicit polygonal models (both closed and open) without any intermediate model conversion.

Recently, Hua et al. proposed a scalar-field-based free-form deformation (FFD) technique [7], which shares some similarities with the mesh-warping operation we propose in this paper. However, the underlying mathematical foundations of these two works are quite different. We treat the mesh warping as an initial value problem that fits into the Lagrangian surface flow framework naturally. In contrast, Hua et al. elaborate on a free-form deformation technique based on the idea of "optical-flow" from computer vision and implement it as a constraints-based optimization problem. More importantly, unlike our approach, their work can not handle topology modifications.

2.2 Interactive mesh generation

Extensive literature exists in interactive mesh-generation. Welch and Witkin [24] proposed a free-form surface design paradigm, where they used a triangle mesh to approximate the underlying smooth variational surface. Markosian et al. [13] presented a particle-based surface representation with which a user can interactively sculpt free-form surfaces. It resembled blobby modeling in its constructive approach. Igarashi et al. [8] developed a more general, free-form modeling system, receiving much of its power from its "inflation" operation and from an elegant collection of gestures for attaching additional parts to a shape, cutting a shape, etc. Nevertheless, most of these mesh-generation techniques are intended for "start-from-scratch" shape-sketching. Our approach, on the other

hand, allows users to start from existing, complicated mesh models and create new models via the available, easy-to-use operations in our system.

3 PDE-based surface flow

The general formulation of geometric surface flow is a nonlinear initial-value partial differential equation:

$$\frac{\partial \boldsymbol{S}(\boldsymbol{p})}{\partial t} = F(t, \boldsymbol{k}, \boldsymbol{k}', \boldsymbol{f} \cdots)\boldsymbol{N}(\boldsymbol{p}, t), \qquad (1)$$
$$\boldsymbol{S}(p, 0) = \boldsymbol{S}_0(\boldsymbol{p}),$$

where F is the velocity function, t is the time parameter, $\boldsymbol{k}$ and $\boldsymbol{k}'$ are the surface curvature and its derivative at the point $\boldsymbol{p}$, respectively, and $\boldsymbol{f}$ is the external force. $\boldsymbol{S}_0(\boldsymbol{p})$ is the initial shape of the model. $\boldsymbol{N}$ is the unit direction vector, and oftentimes, it is the surface normal vector. The velocity function F in Eq. 1 is application-dependent – it can be either directly provided by the user, or more generally obtained as a gradient descent flow by the Euler-Lagrange equation of some underlying energy functionals based on the calculus of variation. One of the important PDEs we used for shape-editing (Sects. 5.1 and 5.2) is the mean curvature flow:

$$\frac{\partial \boldsymbol{S}}{\partial t} = (gv + g\|\boldsymbol{H}\|)\boldsymbol{N}, \qquad (2)$$

where $\boldsymbol{H}$ is the mean curvature of the surface and is acting as a smoothing constraint. v is the constant velocity, which will enable the convex initial shape to capture non-convex, arbitrarily complicated shapes and prevent the model from getting stuck in local minima during the evolution process. In addition, the smoothing effect of the mean curvature term $\boldsymbol{H}$ is inversely proportional to the magnitude of v, i.e., the smaller the v is, the smoother the mean curvature flow of Eq. 2 will be and vice versa. $\boldsymbol{N}$ is the unit normal of the surface. g is a data-dependent, monotone, non-increasing, non-negative weight function that is used for interacting the model with the data, and will stop the deformation of the model when it reaches the shape boundary. For volumetric data, the weight function g can be defined as the commonly used 3D edge detector:

$$g(\boldsymbol{S}) = \frac{1}{1 + \|\nabla(I(\boldsymbol{S}))\|^2}, \qquad (3)$$

where I is the volumetric density function, and ∇ is the gradient function. If the density function $I(\boldsymbol{S})$ is in fact a distance field, then $g(\boldsymbol{S})$ can be directly defined as:

$$g(\boldsymbol{S}) = I(\boldsymbol{S}). \qquad (4)$$

There are, in general, two approaches to numerically simulate PDEs such as Eqs. 1 and 2: an explicit Lagrangian approach or an implicit level-set approach. In this paper, we take the Lagrangian approach, i.e., the geometry and topology of the model are always explicitly represented throughout the simulation process. The advantage of explicit simulation of surface flow is that the user can directly interact with the polygonal models without any intermediate conversion steps, while the challenge is that we need to explicitly maintain the model regularity and to be able to handle collision detection and topology modification accurately during the deformation process. We will discuss issues related to the explicit simulating of surface flow such as mesh regularity in the next section. Our approaches of handling collision detection and topology modifications are described in Sects. 6 and 7, respectively.

4 Surface flow simulation

4.1 Model regularity

To ensure that the numerical simulation of the surface flow proceed smoothly, we must maintain the regularity of the model such that the model has a good node distribution, a proper node density, and a good aspect ratio of the triangles. This is achieved by the incorporation of the following three techniques: mesh optimization, Laplacian smoothing, and local refinement. Note that these techniques are applied only to regions of the polygonal model that are activated (i.e., that are deforming) at the current time step.

4.1.1 Mesh optimization

There are three commonly used mesh optimization operations [6]: edge-splitting, edge-collapsing, and edge-swapping. Edge-splitting and edge-collapsing are used to keep an appropriate node density. An edge split is triggered if the edge length is bigger than the maximum edge-length threshold. Similarly, an edge will be collapsed if its length is smaller than the minimum edge length threshold. Edge swapping is used to ensure a good aspect ratio of the triangles. This can be achieved by forcing the average valence to be as close to 6 as possible [9]. An edge is swapped if and only if the quantity $\sum_{\boldsymbol{p}\in\Delta}(valence(\boldsymbol{p}) - 6)^2$ is minimized after the swapping, where Δ represents the four vertices in the two adjacent triangles of the current edge.

4.1.2 Laplacian smoothing

Laplacian operator, in its simplest form, moves repeatedly at each mesh vertex by a displacement equal to a positive scale factor times the average of the neighboring vertices. Consider a mesh vertex $\boldsymbol{p}$ and its neighbors $\boldsymbol{Q}_1, \cdots, \boldsymbol{Q}_n$; the Laplacian operator $\boldsymbol{L}$ is

$$\boldsymbol{L}(\boldsymbol{p}) = \frac{1}{n}\sum_{i=1}^{n}(\boldsymbol{Q}_i - \boldsymbol{p}). \qquad (5)$$

The tangential Laplacian operator is used to maintain a good node distribution and is defined as:

$$\boldsymbol{T}(\boldsymbol{p}) = c[\boldsymbol{L} - (\boldsymbol{L} \cdot \boldsymbol{N})\boldsymbol{N}], \tag{6}$$

where $\boldsymbol{N}$ is the surface normal vector at vertex $\boldsymbol{p}$, and c is a positive constant between 0 and 1, set to 0.1 in our experiment.

4.1.3 Local refinement

In order to control the smoothness of the model as well as the size of each triangle during the model deformation phase, we must allow the model to be able to dynamically increase its degrees of freedom during the deformation. This is achieved by using local refinement. If the area of a triangle is larger than a certain user-defined threshold, then this triangle will be subdivided into four smaller triangles by splitting the triangle at the midpoints of its three edges.

4.2 Step size estimation

The surface evolution process of Eq. 1 can be numerically approximated using a simple, explicit iterative equation:

$$\boldsymbol{S}(\boldsymbol{p}, t + \Delta t) = \boldsymbol{S}(\boldsymbol{p}, t) + F(\boldsymbol{p}, t)\boldsymbol{N}(\boldsymbol{p}, t)\Delta t. \tag{7}$$

When advancing the model, we must enforce a constraint on the size of the time step Δt. In particular, the time step Δt must satisfy the CFL condition, also known as the Courant-Friedrichs-Lewy stability criterion; i.e., the velocity of change must be strictly restrained by the minimum detail in the system. In our system, this condition is

$$\Delta t \leq \frac{m_e}{M_F}, \tag{8}$$

where m_e is the minimum edge length of the mesh, and M_f is the maximum magnitude of the velocity F obtained by Eq. 1. Before each deformation step, we will calculate the velocity F at each vertex point and determine the maximum magnitude of the velocity M_F. A proper time step can then be estimated from Eq. 8.

4.3 Mean curvature approximation

The mean curvature vector $\boldsymbol{H}$ of Eq. 2 is approximated by the discrete curvature estimator proposed by Desbrun et al. [4]:

$$\boldsymbol{H} = \frac{1}{4A} \sum_{j \in N_1(i)} (cot\,\alpha_j + cot\,\beta_j)(\boldsymbol{x}_i - \boldsymbol{x}_j), \tag{9}$$

where $\boldsymbol{x}_j$ is one of the vertices at the one-neighborhood $N_1(i)$ of $\boldsymbol{x}_i$. α_j and β_j are the two angles opposite the edge connecting the two vertices $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$, and A is the sum of the areas of the triangles having $\boldsymbol{x}_i$ as a common vertex.

5 Surface-flow-based shape modeling

Based on the explicit surface flow introduced in the previous section (Eqs. 1 and 2, and 3), we have developed a number of surface-editing operators, including blending, drilling, sketching, etc. In general, there are five main steps during a typical interactive shape-editing process:

1. First, the user selects a polygonal model to be modified and loaded in. The system then automatically creates an embedding distance field (such as an implicit cube or sphere) that encloses the selected polygonal model.
2. The user interactively defines a local distance field at regions of interest; the corresponding regions of the model that are affected by the user-defined distance field are activated by the system.
3. The user selects the desired type of surface editing operators (e.g., blending, drilling, sketching, etc.), the system then automatically conducts the corresponding Boolean operations such as union or difference between the user-defined distance field and the embedding distance field.
4. The activated regions of the model start to deform according to the user-defined distance field. The deformation is governed by the Lagrangian surface flow introduced in the previous sections (Sects. 3 and 4).
5. During the deformation process, the system automatically detects the potential collisions between different parts of the model and changes the topology if necessary (Sects. 6 and 7).

Figure 1 illustrates the whole shape-editing process. Here, the user wants to add a handle to a goblet. Figure 1(a) shows the original mesh model of the goblet, along with the user-defined local distance field (shown in red color) of the handle. The affected regions of the goblet model are then activated and start to grow. Figures 1(b)–1(d) are the three snapshots of the growing process. To maintain the mesh quality, model relaxation techniques such as mesh optimization, Laplacian smoothing and local refinement (Sect. 4.1) are conducted on-the-fly. Figure 1(e) is the final shape of the goblet with an added handle after a topology merge operation is conducted. Figure 1(f) is the close-up view of the newly added handle to highlight the good mesh quality. In the following subsections, we will explain each surface editing operators in more detail.

5.1 Blending, drilling and cutting

The blending operation is conducted by doing a Boolean-like union operation between the embedding distance field

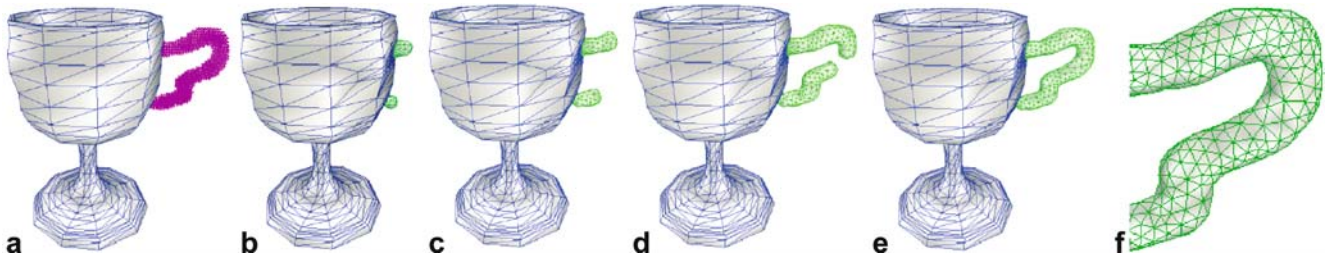

Fig. 1. Adding a handle to the goblet

of the mesh model and the user-defined distance field. The distance field can be created by implicit primitives such as cylinders, spheres, etc., or it can be defined by volumetric datasets. For example, the input of Fig. 2 is a polygonal model of a mannequin head. The user placed an implicit torus (shown in cyan) on the top of the head (Fig. 2(a)), and a union operation is selected to create the base of the crown. The locally affected regions of the head will be activated (shown in red in Fig. 2(a)), start to grow (Fig. 2(b)), and finally stop (Fig. 2(c)). To create the top part of the crown, another three primitives (two half-tori and an ellipsoid) are added in iteratively (shown in cyan) and are recovered by the surface flow one at a time. Figure 2(e) shows the final shape of the model. Note that the nontrivial topology has been correctly represented by three topology merge operations (one merge operation to create the first half-torus, another two merge operations to create the second half-torus).

Figure 3(e) shows another example. Here, a volumetric dataset of a spring is placed on the back of the mannequin head model. The affected region of the model is activated and starts to deform according to the volumetric spring dataset. Figure 3(f) shows the modified shape. The mannequin head model is not a closed model (it has a big opening in the neck); level-set methods cannot directly work on such kinds of models.

The drilling operation is implemented in a similar fashion as the blending operation. Here, instead of a union operation, a subtraction is conducted between the embedding distance field of the mesh model and the user-defined distance field. For example, in Fig. 3, the user wants to subtract an implicitly defined cylinder (shown in cyan in Fig. 3(a)) from the top of the head. The locally affected regions of the head are then activated and start to shrink (shown in red in Fig. 3(b)). Later on, a collision is detected between the two shrinking regions of the model, and a topology merge operation is conducted, creating a hole in the model (Fig. 3(c)).

Because of the convenient inside/outside properties, distance fields can also be used to conduct mesh-cutting operations very easily. The user first creates a distance field among regions of interest of the mesh model, and decides whether to cut regions of the model that are inside or outside of the distance field. The system will then automatically remove the corresponding region from the model. For example, in Fig. 6, an implicit cube is placed on the back of the bunny. Vertices that are inside the implicit cube are removed, and a coin slot is created. In order to ensure that the newly created openings align well with the boundary of the distance field after a mesh-cutting operation, all the vertices around the new openings are projected to the boundary of the distance field using the following operator:

$$\boldsymbol{p} = \boldsymbol{p} - I(\boldsymbol{p})\nabla I(\boldsymbol{p}), \tag{10}$$

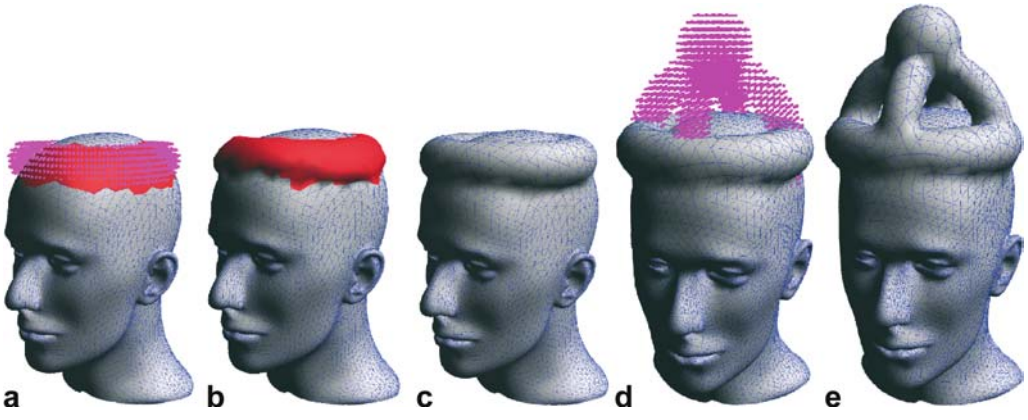

Fig. 2. The crowned mannequin

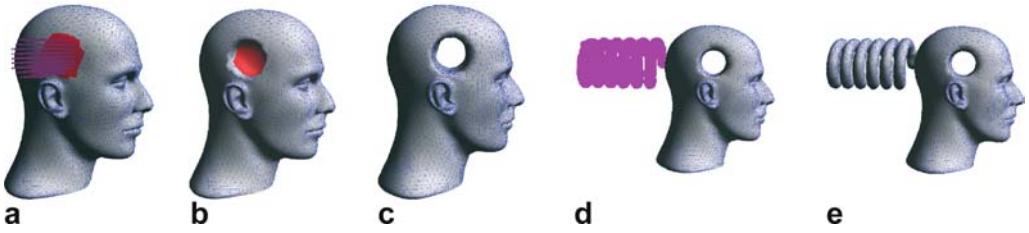

Fig. 3. The mannequin holed and wired

where $\boldsymbol{p}$ is the current vertex position, I is the distance field function, and ∇ is the gradient function.

5.2 Free-hand sketching

Our system also supports interactive free-hand sketching for mesh manipulation. The user can draw some free-hand strokes either directly on the mesh or on the stem away from the mesh, using a mouse or a 3D pointing device. Strokes are then densely sampled by the system to create a local distance field $d(x, y, z)$ as a combination of Gaussian blobs ω_i that are assigned evenly at each point p_i:

$$d(x, y, z) = \sum_{i=1}^{N} \omega_i(x, y, z). \tag{11}$$

The affected regions of the underlying mesh model will then be activated and start to deform according to the corresponding distance field generated by the strokes.

Figure 4 shows an example. The input is a polygonal model of a goblet (Fig. 4(a)). Now, the user wants to add two handles on both sides of the goblet. So he or she simply draws two curved strokes on both sides of the goblet. A distance field is then created by summing up the Gaussian functions that are assigned at each point (shown as red dots and green dots in Fig. 4(a)) on the two strokes. The original goblet mesh will grow along the track of the strokes to form a champion trophy as show in Fig. 4(b). A detailed growing process of the right handle of the trophy is shown in Fig. 1.

The starfish in Fig. 5 is also created using free-hand sketching. Starting from a simple sphere-like polygonal model in the center, the user iteratively draws five curved strokes away from the original mesh. In the same fashion as the in previous example, the curved strokes are first converted to distance fields. The original mesh will then grow along the newly defined distance fields and create the five legs of the starfish.

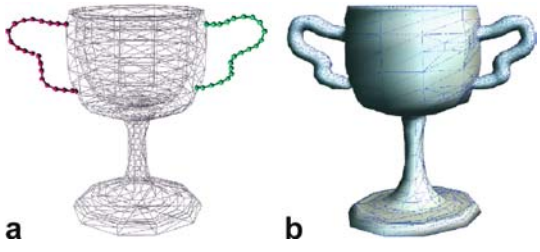

Fig. 4. The champion trophy

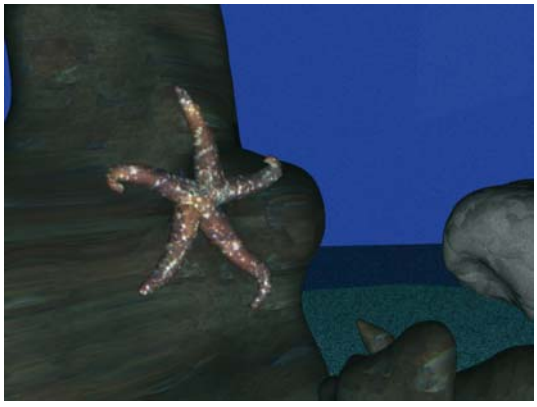

Fig. 5. The starfish

The user can also directly draw strokes on the top of an existing mesh. For example, a $ sign is embossed on the bunny-bank model (Fig. 6) by directly sketching two strokes on the mesh. The Gaussian blobs are assigned evenly at each sampling point. Engraving effect can also be achieved by simply setting the deformation direction as the opposite of the normal vector of each vertex in the affected region.

5.3 Mesh-warping

In addition to local Boolean operations, our algorithm can be used to create some free-form deformations using a new surface flow called *mesh-warping*. The basic idea of mesh warping is as follows: (1) first embed (partially or globally) the input mesh model into an underlying distance field; (2) the user interactively changes the underlying distance field (e.g., by strokes); (3) the mesh model will then deform its shape according to the change in its embedding distance field.

Suppose that the embedding distance field is $\phi : \mathrm{Re}^3 \rightarrow \mathrm{Re}$. For each vertex p, the isovalue of the current vertex p is k_p. Assuming that the isovalues of all the vertices remain the same before and after the deformation, i.e., in the level-set notation, the vertex with position $\boldsymbol{x}(t)$ remains on the same level-set of the embedding distance field ϕ, we

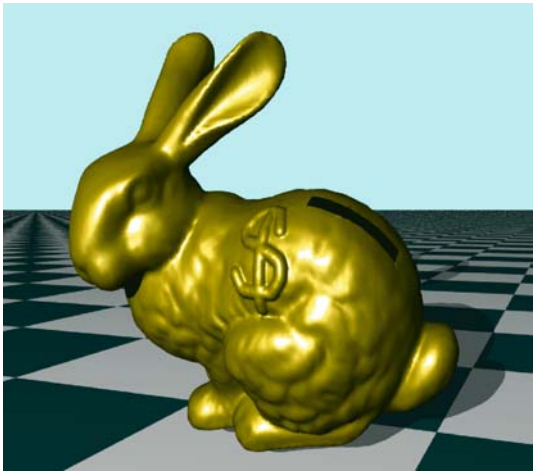

Fig. 6. The bunny-bank

have

$$\phi(\boldsymbol{x}(t), t) = k_p. \tag{12}$$

Differentiating both sides of Eq. 12 with respect to time t, we have

$$\frac{\partial \phi(\boldsymbol{x}, t)}{\partial t} + \nabla \phi(\boldsymbol{x}, t) \cdot \frac{d\boldsymbol{x}}{dt} = 0. \tag{13}$$

If we assume the deformation of the model is along the normal direction of the isocontour, then

$$\frac{d\boldsymbol{x}}{dt} = F(\boldsymbol{x})\boldsymbol{N}, \tag{14}$$

where

$$\boldsymbol{N} = -\frac{\nabla \phi(\boldsymbol{x})}{\|\nabla \phi(\boldsymbol{x})\|}. \tag{15}$$

Note that this assumption is very natural for the modeling of the deformation behavior of changing from one shape to another. For a more detailed explanation on this process, please refer to the work published by Breen et al. [1]. The negative sign of the normal is obtained by the common assumption that the scalar value of the distance field is positive inside and negative outside.

Replacing $\frac{d\boldsymbol{x}}{dt}$ with Eq. 14, Eq. 13 becomes

$$\frac{\partial \phi(\boldsymbol{x}, t)}{\partial t} + \nabla \phi(\boldsymbol{x}, t) \cdot F(\boldsymbol{x}) \cdot \boldsymbol{N} = 0. \tag{16}$$

Plugging Eq. 15 into Eq. 16, we obtain

$$\frac{\partial \phi(\boldsymbol{x}, t)}{\partial t} = F(\boldsymbol{x}) \cdot \|\nabla \phi(\boldsymbol{x}, t)\|.$$

Hence,

$$F(\boldsymbol{x}) = \frac{\partial \phi(\boldsymbol{x}, t)}{\partial t} \cdot \frac{1}{\|\nabla \phi(\boldsymbol{x}, t)\|}. \tag{17}$$

Finally,

$$\begin{cases} \frac{d\boldsymbol{x}}{dt} = \frac{\partial \phi(\boldsymbol{x}, t)}{\partial t} \cdot \frac{1}{\|\nabla \phi(\boldsymbol{x}, t)\|} \cdot \boldsymbol{N} \\ \boldsymbol{N} = -\frac{\nabla \phi(\boldsymbol{x})}{\|\nabla \phi(\boldsymbol{x})\|} \end{cases} \tag{18}$$

Equation 18 is the PDE used in the new mesh-warping flow, which is used to guide the deformation of the model during the mesh-warping process. Note that the derivative and the gradient of the scalar function ϕ can be numerically approximated by finite difference methods. The deformation of the vertex will stop when the scalar value of the distance field at the current vertex position is very close to its original scalar value.

Figure 7 shows an example: an initial mesh model of an engine is embedded in a distance field (Fig. 7(a), where the mesh shows a single level set of the embedding distance field). Now the embedding distance field is partially inflated (Fig. 7(b)), this causes the engine model to be locally inflated as well. Finally, Fig. 7(c) shows a further deformed engineering model, where the front part of the model shrinks.

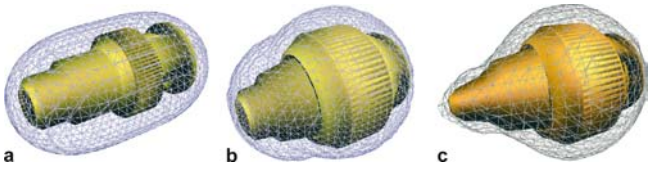

Fig. 7. Mesh-warping of an engineering model

6 Particle-based collision detection

One key challenge in simulating continuous surface evolution of an explicit deformable model is in performing collision detection such that surface interpenetrations can be detected and handled properly. There has been considerable research on the problem of collision detection; in general, existing methods employ either an object-oriented bounding volume method or a domain-oriented spatial decomposition method. The idea behind these approaches is to approximate the objects (with bounding volumes) or to decompose the space they occupy (using decomposition) in order to reduce the number of pairs of objects or primitives that need to be checked for contact. We propose a hybrid approach that can detect potential collisions between different parts of the surface both accurately and efficiently by combining the advantages of both the spatial decomposition method and the bounding volume method. A spatial decomposition method (a uniform occupancy grid) is used for fast collision-rejection between vertices that are located at non-neighboring grid cells. A bounding volume method (bounding spheres)

is used to detect potential collisions between vertices that are located within the same or neighboring grid cells.

We consider the object as a particle system (connected by edges) that is bounded by partially overlapping spheres of radius r centered at each particle (i.e., vertices of the object). Potential collisions between different regions of the object are then detected by potential collisions between particles. Since our model always maintains explicit maximum/minimum thresholds for the edge length, the radius r can be pre-calculated. A potential collision is detected if the distance between any two non-adjacent vertices is smaller than r. To further speed up the performance, a uniform occupancy grid is superimposed on the domain space for fast collision-rejection. Each vertex of the object will belong to a grid cell, and each grid cell will store the index/pointer of the vertices that belong to the current grid cell. The size of the grid cell is decided by the radius r of the bounding sphere so that collisions can only occur between vertices in the same cells or between neighboring cells. At the beginning of each deformation step, the occupancy grid needs to update its vertex information. This process can be done locally, since only a few vertices will move at each deformation step, and it usually takes constant time, or at most $O(n)$.

Figure 8 shows a 2D illustration of the collision-detection scheme. Here, the two moving curves are bounded by partially overlapping circles of radius r (shown as dark circles in Fig. 8(a)) centered on each particle. Several time-steps later, the distance $||P - Q||$ between particle P and Q becomes smaller than r (Fig. 8(b)) and a collision is detected. These two vertices (P and Q) are deactivated and sent to the following topology modification step.

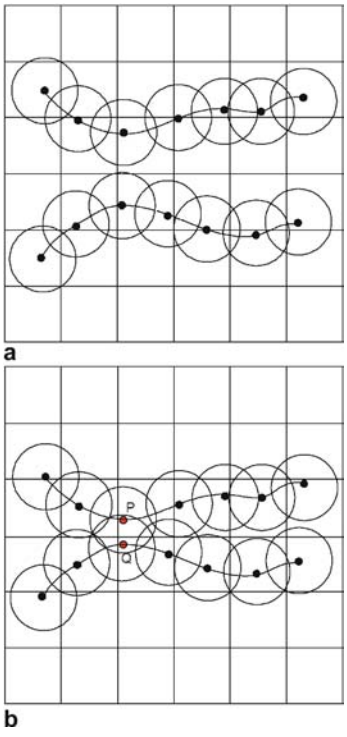

Fig. 8. Particle-based collision detection

7 Topology modification

There are two types of topology modifications: topology merging (i.e., axial melting) and topology splitting (i.e. axial constriction). In our current system, only topology merging is implemented, and it is conducted in a sequential fashion, i.e., at most one topology merge operation can occur at any time. Moreover, to ensure the correctness of the algorithm, a topology merge operation can occur only after all of the vertices of the model become deactivated (i.e., do not move anymore). There are three steps in the topology-merging operation:

1. Find a pair of merging vertices and align their one-neighborhoods to face each other.
2. Put the two one-neighborhoods into correspondence.
3. Reconnect the two one-neighborhoods.

7.1 Find a pair of center vertices and align their one-neighborhoods

The first step of the algorithm is to pick the best pair of merging vertices to serve as the center vertices. Specifically, we will calculate the inner products of the normal vectors of all pairs of merging vertices and choose the pair with the smallest inner product. If the angular deviation of the normal vectors of this pair of vertices is smaller than a certain threshold (e.g., 30 degrees), they will be picked as the two center vertices. In the rare case that the angular deviations of the normal vectors of all pairs of vertices are larger than the threshold, to ensure the robustness of the algorithm, no topology merge operation will be allowed to happen. Instead, the merging regions of the model (i.e., regions of the model that contain merging vertices) will be locally refined one or more times by Loop's subdivision scheme [11] until one pair of merging vertices is selected as the two center vertices (i.e., the angular deviation of the normal vectors of these two vertices is smaller than the threshold). Because of the well-known smoothing effect of Loop's scheme, one level of refinement is usually sufficient.

Next, all of the one-neighborhood vertices of these two center vertices are projected onto the plane passing the center vertex and are perpendicular to the vector connecting these two center vertices. This way, the two one-neighborhoods will face towards each other exactly (Fig. 9(a)–(b)).

7.2 Put the two one-neighborhoods into correspondence

After the two one-neighborhoods are aligned to face each other, they will be put into correspondence by the following procedure: iteratively refine the one-neighborhood that has fewer vertices by splitting its longest edge until both of the two one-neighborhoods have the same number of vertices, then choose an alignment that minimizes the sum of squared distances between corresponding vertices of the two one-neighborhoods. For example, in Fig. 9(b), originally, the one-neighborhood of vertex A has five nodes ($\{A_1, A_2, A_3, A_4, A_5\}$), and the one-neighborhood of vertex B has six nodes ($\{B_1, B_2, B_3, B_4, B_5, B_6\}$). To make these two one-neighborhoods have the same number of nodes, we first find the longest edge of the one-neighborhood of vertex A, which is the edge between nodes A_1 and A_2, and then split this edge into two edges and insert a new node in between. Finally, we put these two sets of vertices into correspondence by finding the alignment that minimizes the sum of squared distances between nodes. In Fig. 9(c), vertices $\{A_1, A_2, \cdots, A_6\}$ correspond to $\{B_1, B_2, \cdots, B_6\}$, respectively.

7.3 Reconnect the two one-neighborhoods

After the two sets of one-neighborhood vertices are put into correspondence, each vertex is connected with its corresponding vertex in the opposite one-neighborhood. The two center vertices and all of their incident edges are removed (Fig. 9(d)). The newly created quadrilaterals are further triangulated by splitting each quadrilateral into two triangles along one of its diagonals (Fig. 9(e)). The aforementioned model-relaxation operations (Sect. 4.1) can quickly smooth out any artifacts that may result from the matching procedure once the topology-merging operation has been completed. In particular, if the merging regions of the model had been locally refined (Sect. 7.1), after the topology merge, the edge collapse operation (Sect. 4.1) will automatically collapse all of the edges in the merging regions whose lengths are smaller than the user-defined minimum edge length.

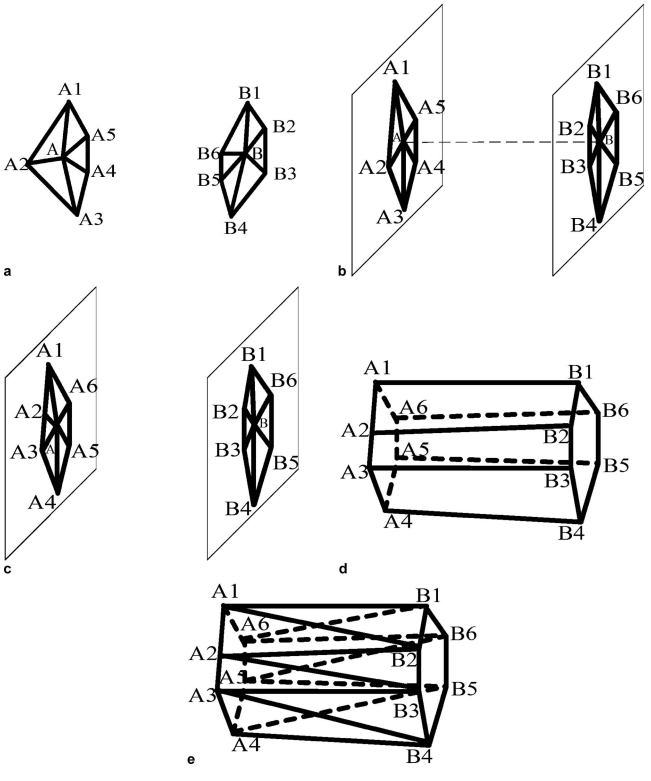

Fig. 9. Topology merge

8 Discussion

In this paper, we propose a new shape modeling algorithm that integrates the PDE-based geometric surface flow commonly used in implicit, level-set methods with explicit polygonal models. Given an input polygonal model, the user can directly manipulate the mesh with the help of an implicit distance field. Unlike the level-set approaches, no intermediate conversion stage is necessary. In addition, our algorithm can work with both closed and open meshes. During the surface evolution process, our algorithm will always ensure model regularity and can properly modify the model topology by a topology-merging operation whenever necessary.

Several further improvements are possible: although it works well in our experiments, the topology-modification algorithm still has some limitations (e.g., it cannot handle topology-splitting, and only one topology merge can occur at any time). In the future, we would like to develop a more general topology modification algorithm that does not have these constraints. Second, we would like to incorporate level-of-detail (LOD), multi-resolution capability into our free-hand sketching operator (Sect. 5.2) so that more refined and more precise models can be created. Besides the currently used Gaussian blobs, other types of distance field functions might also be used to recover sharp features. Third, since all of the computations are conducted locally, our algorithm is actually quite efficient – all of the examples shown in this paper are done in a few seconds on a Pentium 4 Notebook PC with a 1.6 GHZ CPU with 512 MB of RAM. Nonetheless, our current code is not optimized; we would like to further improve our code and take advantage of the recent advancements in graphics-oriented high-end workstations to achieve real-time performance. This, in turn, will make

our system more suitable in a semi-immersive virtual reality environment such as a workbench and would definitely help users gain a much better understanding of the 3D shape geometry and perform the direct geometric deformation through user-immersion.

Acknowledgement This research was supported in part by the NSF ITR grant IIS-0082035, the NSF grant IIS-0097646, and the Alfred P. Sloan Fellowship. The mannequin head model is provided by Dr. Hughes Hoppe and the University of Washington. Other mesh models are provided by 3DCafe.com. The spring volumetric dataset is provided by Professor Arie Kaufman and the State University of New York at Stony Brook.

References

1. Breen DE, Whitaker RT (2001) A level-set approach for the metamorphosis of solid models. IEEE Trans Visual Comput Graph 7(2):173–192
2. Coher-Or D, Levin D, Solomovici A (1997) Three-dimensional distance field metamorphosis. ACM Trans Graph
3. Crespin B (1999) Implicit free-form deformations. In: Implicit Surfaces '99 Workshop, Eurographics, pp 17–24
4. Desbrun M, Meyer M, Schroder P, Barr AH (1999) Implicit fairing of irregular meshes using diffusion and curvature flow. In: SIGGRAPH '99 Proceedings, pp 317–324
5. Frisken S, Perry R, Rockwood A, Jones T (2000) Adaptively sampled distance fields: a general representation of shape for computer graphics. In SIGGRAPH '00 Proceedings, pp 249–254
6. Hoppe H, DeRose T, Duchamp T, McDonald J, Stuetzle W (1993) Mesh optimization. In: SIGGRAPH '93 Proceedings, pp 19–26
7. Hua J, Qin H (2004) Scalar-field guided adaptive shape deformation and animation. Vis Comput 20(1):47–66
8. Igarashi T, Matsuoka S, Tanaka H (1999) Teddy: a skeching interface for 3D freeform design. In: SIGGRAPH 99 Proceedings, pp 409–416
9. Kobbelt L, Bareuther T, Seidel H-P (2000) Multiresolution shape deformations for meshes with dynamic vertex connectivity. In: Eurographics '00 Proceedings, pp 249–260
10. Kobbelt L, Campagna S, Vorsatz J, Seidel H-P (1998) Interactive multi-resolution modeling on arbitrary meshes. In: SIGGRAPH '98 Proceedings, pp 105–114
11. Loop C (1987) Smooth subdivision surfaces based on triangles. Dissertation, University of Utah
12. Lorensen WE, Cline HE (1987) Marching cubes: a high resolution 3D surface construction algorithm. In: SIGGRAPH '87 Proceedings, pp 163–169
13. Markosian L, Cohen JM, Crulli T, Hughes J (1999) Skin: a constructive approach to modeling free-form shapes. In: SIGGRAPH 99 Proceedings, pp 393–400
14. Museth K, Breen DE, Whitaker RT, Barr AH (2002) Level set surface editing operators. In: SIGGRAPH 02 Proceedings, pp 330–338
15. Ohatake Y, Belyaev A (2001) Mesh optimization for polygonized isosurfaces. In: Proceedings of Eurographics, pp 368–376
16. Ohatake Y, Belyaev A, Pasko A (2001) Dynamic meshes for accurate polygonization of implicit surfaces with sharp features. In Proceedings of Shape Modeling International 2001, pp 74–81
17. Payne B, Toga A (1992) Distance field manipulation of surface models. IEEE Computer Graphics and Applications, pp 65–71
18. Perry R, Frisken S (2001) Kizamu: A system for sculpting digital characters. In SIGGRAPH '01 Proceedings, pp 47–56
19. Schneider R, Kobbelt L (2001) Geometric fairing of irregular meshes for free-form surface design. Comput Aided Geom Des 18(4):359–379
20. Schroeder W, Lorensen W, Linthicum S (1994) Implicit modeling of swept surfaces and volumes. In: IEEE Visualization '94 Proceedings, pp 40–45
21. Shapiro V, Tsukanov I (1999) Implicit functions with guaranteed di(r)erential properties. In: Proceedings of the Fifth ACM Symposium on Solid Modeling and Applications
22. Singh K, Parent R (1995) Implicit function based deformations of polyhedral objects. In: Implicit Surfaces '95 Workshop, Eurographics, pp 113–128, 1995.
23. Taubin G (1995) A signal processing approach to fair surface design. In: SIGGRAPH '95 Proceedings, pp 351–358
24. Welch W, Witkin A (1994) Free-form shape design using triangulated surfaces. In: SIGGRAPH '94 Proceedings, pp 247–256