The Visual Computer

Scalar-field-guided adaptive shape deformation and animation

Jing Hua,
Hong Qin

Department of Computer Science, State University of New York at Stony Brook, NY 11794-4400, USA
E-mail: {jinghualqin}@cs.sunysb.edu

Published online: 9 December 2003

In this paper, we propose a novel scalar-field-guided adaptive shape deformation (SFD) technique founded on PDE-based flow constraints and scalar fields of implicit functions. Scalar fields are used as embedding spaces. Upon deformation of the scalar field, a corresponding displacement/velocity field will be generated accordingly, which results in a shape deformation of the embedded object. In our system, the scalar field creation, sketching, and manipulation are both natural and intuitive. The embedded model is further enhanced with self-optimization capability. During the deformation we can also enforce various constraints on embedded models. In addition, this technique can be used to ease the animation design. Our experiments demonstrate that the new SFD technique is powerful, efficient, versatile, and intuitive for shape modeling and animation.

Key words: Shape deformations – Scalar fields – Interaction techniques

Correspondence to: Jing Hua

1 Introduction

Efficient and intuitive shape deformation techniques are vital to the success of geometric modeling, computer animation, physical simulation, and other visual computing areas. Free-form deformation (FFD) is one popular technique that meets this need. In essence, an object deformation based on FFD is conducted indirectly by deforming the enclosing space in which the object is defined and embedded. Every vertex of the object has a unique parameterization that initially defines its position in the space. When the space is altered, it gives rise to the shape deformation of its embedded object based on the initial parameterization. One appealing advantage of FFD-based techniques over other traditional modeling/editing methods is that they can be applied to any geometric object since the surrounding space is independent of objects' mathematical representations. Therefore, designers do not need to worry about the underlying (perhaps complicated) geometric formulation and topological structure of an embedded object when making the desired deformations. Various FFD techniques have been proposed during the past two decades [2, 7–9, 19, 20, 28].

Despite many attractive properties, such as the natural control of space definition, there are several difficulties associated with the current FFD techniques. First, while very useful for coarse-scale deformations of an object, the technique can be time consuming to use for finer-scale deformations [29], where a very dense and customized control lattice shape is usually required [20]. When dealing with a large number of complex control lattices (which define the space), this tends to be cumbersome and counterproductive. Designers have to construct the complex control lattice to define the space embedding the object and alter the large number of control lattices to deform the space to make the desired deformations of the object. Although the axis-based FFD approaches and the directly manipulated FFD approaches [2, 7, 19] are relatively intuitive and efficient, they can offer only limited deformations. Second, the control lattice is less flexible and cannot easily have very complex topology. It is difficult to perform a large number of distinct deformation types. Third, the traditional FFD operation is generally a single operation applied to static models. Refinement and optimization can only occur before or after the deformations. For instance, if a low curvature region is not subdivided prior to the deformation, the model is no longer capable of representing the deformation accurately. To alleviate this prob-

The Visual Computer (2004) 20:47–66
Digital Object Identifier (DOI) 10.1007/s00371-003-0225-z

lem, intermediate steps have to be introduced during the deformation steps.

In this paper, we propose a novel adaptive shape deformation technique, or SFD, founded on PDE-based flow constraints and scalar fields of implicit functions, that employs a scalar field as the embedding space. Users can interactively sketch a scalar field of an implicit function via a mouse or a 3D haptic interface to embed either an entire model or a part of the model. The embedding space based on scalar field is of diverse types, which implicitly defines a complicated geometry and an arbitrary topology. Upon deformation of the embedding space (i.e., the modification of scalar field), the vertices of the embedded object throughout the entire space will move in accordance with the enforced flow constraints, which result in FFDs of the embedded object. The deformation velocity of each vertex on the model is very general and can easily adopt any user-desired constraints. Our SFD technique greatly generalizes the traditional FFD approaches and affords a larger number of shape deformations. The SFD space construction, sketching, and manipulation are more natural and easy to use than previous FFD techniques based on parametric geometry.

Furthermore, to represent deformations more accurately, the embedded models are equipped with self-optimization capability. Adaptive subdivision and mesh optimization are tightly coupled with SFD, supporting versatile multiresolution deformations. Since our SFD can be treated as a time-evolving process (rather than a single operation), it allows self-adaptive refinement and mesh improvement to interleave with shape deformation throughout the SFD process. Our algorithm adaptively subdivides the model into regions that require high resolution. As the SFD deforms an object, the curvature of the affected surface is computed and checked to see if subdivision is necessary. The mesh improvement operations, such as the edge-split operation, edge-collapse operation, edge-swap operation, and Laplacian mesh smoothing, are then invoked to optimize neighborhood shapes, which equalize edge lengths, allowing vertices to distribute themselves more evenly during a SFD run. We also incorporate various constraints on embedded models that enable our technique to facilitate feature-based design. Our results demonstrate that the proposed SFD technique is useful and powerful for shape editing and animation design.

2 Related work

2.1 Free-form deformation

A deformation technique developed by Barr [2] uses a set of hierarchical transformations for deforming an object. This technique uses the surface normal vector of the undeformed surface and a transformation matrix to calculate the normal vector of an arbitrarily deformed smooth surface. Unfortunately, this technique restricts the possible definitions of the deformable space to that of a single coordinate system and the ways in which the space can be altered. Sederberg and Parry [28] proposed to deform solid geometry in a free-form manner. The vertices on the object embedded in the original lattice structure are mapped to the deformed lattice using a trivariate Bézier spline. However, the FFD can be accomplished only with a parallelpiped lattice structure. Coquillart [8] developed the extended free-form deformation, or EFFD, as an extension of Sederberg and Parry's technique, which uses nonparallelepiped 3D lattices. The goal of this technique is to change the shape of an existing surface either by bending it along an arbitrarily shaped curve or by adding randomly shaped bumps to it. Chang and Rockwood's approach [7] deforms an object by repeatedly applying affine transformations in space. But this technique also limits the ways in which the space surrounding the curve can be altered. MacCracken and Joy [20] presented a FFD technique that uses arbitrary lattices, namely, Catmull-Clark subdivision volumes. An underlying model can be deformed by establishing positions of model points within the converging sequence of lattices and then tracking the new positions of these points within the deformed sequence of lattices. This technique allows a variety of deformable regions to be defined and thus a broader range of shape deformations to be generated. However, the lattice space definition is laborious and difficult. This technique requires a great deal of CPU time and memory in general.

Recently, Singh and Fiume [29] presented *wires* for interactive, geometric deformation. The manipulation of wires deforms the surface of an associated object near the curves. Crespin [10] presented a FFD technique with the use of deformation primitives. Each of them results in a deformation on an associated part of an object. The blending functions associated with primitives are then used to combine the

deformation introduced by each primitive. However, the combinations for global deformations and local deformations are different. Jin et al. [16] proposed a constrained local deformation technique based on generalized metaballs. Schmitt et al. [27] presented a shape-driven technique for functionally defined heterogeneous volumetric objects. Fundamentally, the aforementioned four approaches all employed implicit functions associated with deformation primitives (such as curve, point, metaballs, and so on) to configure a specific, static pointwise function mapping to achieve FFD. Unlike those approaches, our paper introduces a new SFD technique by establishing deformation methods defined on entire scalar fields instead of building a mapping function from deformation primitives, even though we also employ similar primitives to construct scalar fields. In this way, a larger number of deformation types can be achieved. As the scalar field is modified, a deformation of the space is created and a corresponding velocity field will be generated according to PDE-based flow constraints, which evolves the shape of the embedded object or part of the object to deform over time. Therefore, during the deformation process, the self-optimization process can be substantially evolved to maintain the model quality.

2.2 Level set methods and implicit modeling

Our work is also partially related to level set approaches. The level set method was first presented in [23]. Level set models are deformable implicit surfaces where the deformation of the surface is controlled by a speed function in the level set partial differential equation (PDE). Level set methods have been successfully applied in image processing, computer vision, and visualization. In computer graphics, Desbrun et al. [11] and Breen et al. [6] used this method for shape morphing, and Whitaker [32] employed this technique for 3D reconstruction. More recently, Museth et al. [22] and Bærentzen et al. [1] presented a level set framework for interactively editing implicit surfaces, where they defined a collection of speed functions that produce a set of surface editing operators. The speed functions describe the velocity at each vertex on the evolving surface in the direction of the surface normal only. For level set methods, the essential problem is to construct implicit functions or implicit models based on application-oriented speed functions. Note that models to be deformed must first be converted into volumetric representation and represented as a single isosurface.

To date, most algorithms only focus on a single level set, such as implicit surface applications [4] or level set approaches. The isosurface property of implicit functions has not been fully utilized. Our SFD approach is the first method to provide deformations of an existing model by constraining the vertices of the model to stay on their initial, yet different, level sets of an underlying implicit function.

For interactive design of scalar fields, Bloomenthal et al. [5] used skeleton methods to construct implicit surfaces. Each skeletal element is associated with a locally defined implicit function. Individual functions are blended to form an implicit surface using a polynomial weighting function that can be controlled by users. The blobby model [3], also known as soft object [34], is another popular technique for the design of implicit surfaces. Implicit functions are also used to represent volumes. Recently, Raviv and Elber [25] presented a 3D interactive sculpting paradigm that employed a set of scalar uniform trivariate B-spline functions as object representations. Users can indirectly sculpt objects to a desirable shape by directly modifying relevant scalar control coefficients of the underlying functions with virtual sculpting tools. Schmitt et al. [26] presented an approach to constructive modeling of FRep solids [24] defined by real-valued functions using 4D uniform rational cubic B-spline volumes as primitives. Hua and Qin [13, 14] presented a haptics-based modeling system founded on dynamic spline-based implicit functions and physics-based modeling to directly manipulate any level set of their implicit objects. Turk and O'Brien [30] introduced new techniques for modeling with interpolating implicit surfaces.

3 SFD space definition using scalar field

From a mathematical point of view, there are basically two approaches to formulating geometric surfaces: parametric forms and implicit forms. Implicit forms treat coordinates as functional arguments rather than as functional values. In general, surfaces expressed by an implicit form can be formulated as:

$$\{\boldsymbol{X} \in \Re^3 | f(\boldsymbol{X}) = c\}. \tag{1}$$

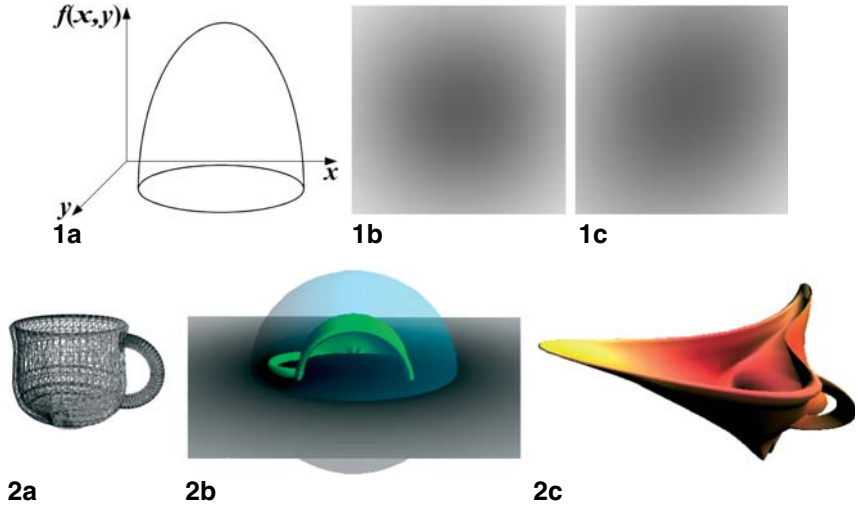


Fig. 1. Panel **a** shows an implicit function graph as the illustrative height function, and panel **b** shows its corresponding scalar field as the intensity image. A 2D object (a free-form planar curve or even an arbitrarily defined contour) can be embedded in the 2D scalar field as shown in **b** and deformed by altering the embedding space as shown in **c**
Fig. 2. A polygonal object (**a**) is embedded in a scalar field (**b**) and deformed by altering the scalar field (**c**). Note that only **b** shows a cross-sectional view of the scalar field (see the cutting plane) and a single level set (see the light-blue transparent sphere). Panel **c** shows the deformed object, where the color indicates the scalar value distribution over the object. Large scalar values in the extremities are indicated by *red*, while regions with small scalar values are colored *yellow*

Indeed, when c is 0, we say that f implicitly defines a locus, called an implicit surface. The function f is called the implicit function, or the field function, and defines the scalar field. The implicit surface is sometimes called the zero level set of f. The related level set (also called an isosurface) corresponds to an isovalue c. The function f may be of any mathematical expression containing polynomials or nonpolynomials. It may also be an arbitrary procedural process (i.e., a black box function) that produces a scalar value for a given point in space.

In our work, the scalar field can be used as SFD embedding space, which wraps a *to-be-deformed* object. Note that, in strong contrast to traditional FFD space, the SFD space is an implicit-function-based scalar field instead of a lattice-based geometric object. Figure 1a shows a 2D implicit function graph, and Fig. 1b is the scalar field corresponding to the function. There is a 2D object (in green) embedded in the scalar field. Note that the green-colored object (in Fig. 1) can be arbitrarily defined, and it need not be a certain level set of the SFD encompassing space. When the scalar field is deformed (Fig. 1c), the 2D object is deformed accordingly. Figure 2 shows a 3D example. The teacup model is embedded inside a 3D scalar field. After altering the scalar field, the teacup is significantly deformed. In this paper, we give users three ways to construct and manipulate scalar fields. We will discuss them in Sect. 6.

4 Scalar-field-guided shape deformation

Let us briefly overview our idea of applying scalar fields of implicit function to perform deformations on any existing polygonal models. First, we embed an entire model or a part of the model into a scalar field and calculate the scalar values at all vertices of that embedded part. Then, during the deformation process the vertices are always constrained on the level sets where they originally reside by enforcing PDE-based flow constraints. Once users deform the embedding space (i.e., the scalar field), the vertices will move accordingly, which results in FFD of the embedded object.

Since the SFD enforces the policy that the relocated position of a vertex $\boldsymbol{X}(t)$ of the deformed object must

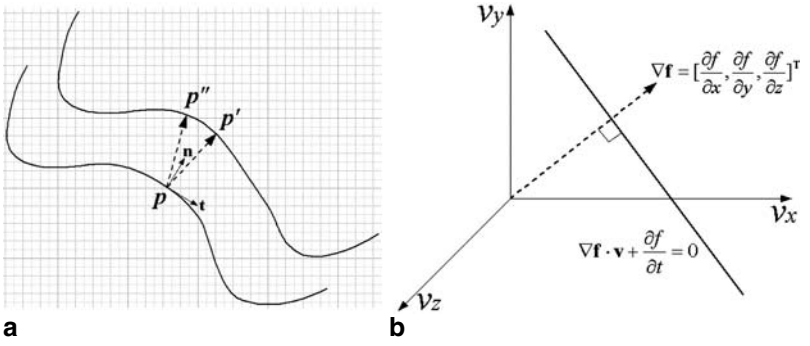


Fig. 3. a Ambiguity in determining the vertex motion. **b** *Straight line* of flow constraint and the normal velocity component along the gradient vector

remain on the same level set when scalar field space is deformed, the trajectory of the vertex can be represented as follows:

$$\{\boldsymbol{X}(t) | f(\boldsymbol{X}(t), t) = c\}. \tag{2}$$

The derivative of $f(\boldsymbol{X}(t), t)$ yields

$$\frac{df(\boldsymbol{X}(t), t)}{dt} = \frac{\partial f(\boldsymbol{X}(t), t)}{\partial \boldsymbol{X}} \frac{d\boldsymbol{X}(t)}{dt} + \frac{\partial f(\boldsymbol{X}(t), t)}{\partial t} = 0, \tag{3}$$

where $\frac{\partial f(\boldsymbol{X}(t), t)}{\partial \boldsymbol{X}}$ is the gradient at $\boldsymbol{X}$. To simplify the notation, we represent the gradient using $\nabla \boldsymbol{f}$ and abbreviate $f(\boldsymbol{X}(t), t)$ as f. Therefore, Eq. 3 can be rewritten as follows:

$$\nabla \boldsymbol{f} \cdot \frac{d\boldsymbol{X}(t)}{dt} + \frac{\partial f}{\partial t} = 0. \tag{4}$$

Note that $\frac{d\boldsymbol{X}(t)}{dt}$ and $\frac{\partial f}{\partial \boldsymbol{X}}$ are both vectors. Therefore, there is an ambiguity, and the solution for the vertex velocity from Eq. 4 is not unique. As shown in Fig. 3a, the evolution of p at the next time step could be p' or p''. The velocity $\frac{d\boldsymbol{X}(t)}{dt}$ cannot be solved uniquely with the flow constraint alone. As shown in Fig. 3b, the solutions for the velocity are restricted to the straight line of the flow constraint equation, $\nabla \boldsymbol{f} \cdot \frac{d\boldsymbol{X}(t)}{dt} + \frac{\partial f}{\partial t} = 0$. Dividing $\boldsymbol{v}$ into $(\boldsymbol{v_n}, \boldsymbol{v_t}, \boldsymbol{v_w})$, where $\boldsymbol{n} = \frac{\nabla \boldsymbol{f}}{\|\nabla \boldsymbol{f}\|}$ represents the unit principal normal vector of the isosurface of the scalar field, $\boldsymbol{t}$ represents the unit tangent vector, and $\boldsymbol{w}$ represents the unit binormal vector, we know that only the normal velocity, $\boldsymbol{v_n}$, is perpendicular to the constraint line. So the dot product in Eq. 4 retains only the item containing $\boldsymbol{v_n}$. Therefore, we can obtain the velocity of the normal flow as follows:

$$\boldsymbol{v_n} = -\frac{1}{\|\nabla \boldsymbol{f}\|} \frac{\partial f}{\partial t} \boldsymbol{n}. \tag{5}$$

This normal flow scheme is essentially the basis of level set methods and has been used widely in sampling implicit surfaces [33], computer vision [21], and level-set-based applications [6, 11] for tracking the target objects. The normal flow scheme considers only the evolution velocities along the normals of isosurfaces. It is good for minimizing the similarity between the active model and the target model. However, the intermediate deformations are not natural and are not dealt with or addressed in the aforementioned work. In essence, the aforementioned work designed the normal velocities $\boldsymbol{v_n}$ to evolve the function f.

By contrast, our objective is to provide a general SFD technique. We compute vertex velocities $\boldsymbol{v}$ based on the change of f. The motion of the embedded object inside the scalar field should be natural, versatile, and without strong limitation. Therefore, we need to consider the velocities along three linearly independent directions simultaneously. In this paper, we consider the general velocity $\frac{d\boldsymbol{X}(t)}{dt}$ along the three coordinate axes, x, y, z, of the 3D space. The general velocity is also represented using (v_x, v_y, v_z) or $\boldsymbol{v}$. To obtain the unique solution from the flow constraint Eq. 4 and also maintain the smoothness motion of

the deformed model during the SFD process, we add a further smoothness constraint over the model. The vertex velocity variation inside a local region is minimized. This gives rise to minimizing the following objective function:

$$E = \int \left(\nabla \boldsymbol{f} \cdot \boldsymbol{v} + \frac{\partial f}{\partial t} \right)^2 + \lambda (\|\nabla \boldsymbol{v}\|)^2 d\boldsymbol{x}, \tag{6}$$

where λ is a Lagrange multiplier.

By discretizing the above objective function, Eq. 6 can be minimized iteratively. Consider a vertex k and its adjacent neighboring vertex set $\boldsymbol{Q}_k$ in an optimized mesh,

$$\boldsymbol{Q}_k = \{ j | \overrightarrow{jk} \in \boldsymbol{M} \},$$

where $\boldsymbol{M}$ denotes a set of all the edges of the embedded model. Note that the mesh of the model will undergo an optimization process as described in Sect. 5. The error of the flow constraint approximation is as follows:

$$c(k) = \left(\frac{\partial f}{\partial x} v_x(k) + \frac{\partial f}{\partial y} v_y(k) + \frac{\partial f}{\partial z} v_z(k) + \frac{\partial f}{\partial t} \right)^2.$$

The smoothness of the motion of the local region can be computed according to the velocity difference between the vertex, k, and its adjacent neighboring ones, $j \in \boldsymbol{Q}_k$.

$$s(k) = \frac{1}{\|\boldsymbol{Q}_k\|} \sum_{j \in \boldsymbol{Q}_k} [(v_x(k) - v_x(j))^2 + (v_y(k) - v_y(j))^2 + (v_z(k) - v_z(j))^2],$$

where $\|\boldsymbol{Q}_k\|$ denotes the number of vertices in $\boldsymbol{Q}_k$. Therefore,

$$E = \sum_k (c(k) + \lambda s(k)). \tag{7}$$

The solution, satisfying $\frac{\partial E}{\partial v_x(k)} = 0$, $\frac{\partial E}{\partial v_x(k)} = 0$, and $\frac{\partial E}{\partial v_x(k)} = 0$, can minimize the above objective function E. The derivatives of E with respect to $v_x(k)$, $v_y(k)$, and $v_z(k)$ are as follows:

$$\frac{\partial E}{\partial v_x(k)} = 2 \left(\frac{\partial f}{\partial x} v_x(k) + \frac{\partial f}{\partial y} v_y(k) + \frac{\partial f}{\partial z} v_z(k) + \frac{\partial f}{\partial t} \right) \times \frac{\partial f}{\partial x} + 2\lambda (v_x(k) - \overline{v}_x(k)),$$

$$\frac{\partial E}{\partial v_y(k)} = 2 \left(\frac{\partial f}{\partial x} v_x(k) + \frac{\partial f}{\partial y} v_y(k) + \frac{\partial f}{\partial z} v_z(k) + \frac{\partial f}{\partial t} \right) \times \frac{\partial f}{\partial y} + 2\lambda (v_y(k) - \overline{v}_y(k)),$$

$$\frac{\partial E}{\partial v_z(k)} = 2 \left(\frac{\partial f}{\partial x} v_x(k) + \frac{\partial f}{\partial y} v_y(k) + \frac{\partial f}{\partial z} v_z(k) + \frac{\partial f}{\partial t} \right) \times \frac{\partial f}{\partial z} + 2\lambda (v_z(k) - \overline{v}_z(k)),$$

where $(\overline{v}_x(k), \overline{v}_y(k), \overline{v}_z(k))$ is the average velocity, $\overline{\boldsymbol{v}}(k)$, of all the adjacent neighboring vertices in $\boldsymbol{Q}_k$,

$$\overline{\boldsymbol{v}}(k) = \frac{1}{\|\boldsymbol{Q}_k\|} \sum_{j \in \boldsymbol{Q}_k} \boldsymbol{v}(j). \tag{8}$$

Therefore, we can obtain the following equations, and (v_x, v_y, v_z) can be solved afterwards. To abbreviate the equations, we omit (k) unambiguously.

$$\left(\lambda + \left(\frac{\partial f}{\partial x} \right)^2 \right) v_x + \frac{\partial f}{\partial x} \frac{\partial f}{\partial y} v_y + \frac{\partial f}{\partial x} \frac{\partial f}{\partial z} v_z = \lambda \overline{v}_x - \frac{\partial f}{\partial x} \frac{\partial f}{\partial t},$$

$$\left(\lambda + \left(\frac{\partial f}{\partial y} \right)^2 \right) v_y + \frac{\partial f}{\partial x} \frac{\partial f}{\partial y} v_x + \frac{\partial f}{\partial y} \frac{\partial f}{\partial z} v_z = \lambda \overline{v}_y - \frac{\partial f}{\partial y} \frac{\partial f}{\partial t},$$

$$\left(\lambda + \left(\frac{\partial f}{\partial z} \right)^2 \right) v_z + \frac{\partial f}{\partial x} \frac{\partial f}{\partial z} v_x + \frac{\partial f}{\partial y} \frac{\partial f}{\partial z} v_y = \lambda \overline{v}_z - \frac{\partial f}{\partial z} \frac{\partial f}{\partial t}.$$

By solving the above equation, we can obtain the following iterative solution:

$$[v_x, v_y, v_z]^\top = [\overline{v}_x, \overline{v}_y, \overline{v}_z]^\top - \mu \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \right]^\top, \tag{9}$$

where $\mu = \frac{\frac{\partial f}{\partial x} \overline{v}_x + \frac{\partial f}{\partial y} \overline{v}_y + \frac{\partial f}{\partial z} \overline{v}_z + \frac{\partial f}{\partial t}}{\lambda + \left(\frac{\partial f}{\partial x} \right)^2 + \left(\frac{\partial f}{\partial y} \right)^2 + \left(\frac{\partial f}{\partial z} \right)^2}$.

Based on the above formulations, we have designed a dynamic continuous SFD algorithm as follows.

First, generate a desired scalar field that embeds a model to be deformed and initialize the velocities (v_x, v_y, v_z) at all the vertices as 0. After the user alters the scalar field, the embedded model begins to deform. During the deformation process, perform the following loop until all the vertices reach the level set that they originally resided at.
At each time step Δt, do the following:

1. Update the scalar field $f(\boldsymbol{X}, t + \Delta t)$ at all the vertices;
2. Deduce $\frac{\partial f}{\partial t} = (f(\boldsymbol{X}, t + \Delta t) - f(\boldsymbol{X}, t))/\Delta t$;
3. Calculate $\frac{\partial f}{\partial \boldsymbol{X}}$ with finite differences;
4. Compute $\overline{\boldsymbol{v}}$ according to current vertex velocities;
5. Deduce $\boldsymbol{v}$ according to Eq. 9;
6. Update vertices' positions by $\boldsymbol{X}_{t+\Delta t} = \boldsymbol{X}_t + \gamma \cdot \boldsymbol{v} \cdot \Delta t$;
7. Perform SFD model optimization (Sect. 5);
8. If $f(\boldsymbol{X}_{t+\Delta t}, t + \Delta t) \approx f(\boldsymbol{X}, t)$, terminate; otherwise, advance to next time step and repeat the above steps.

The SFD essentially is an evolution process that allows self-adaptive refinement and mesh improvement interleaved with the model deformation at each iteration. In this algorithm, the γ is a step size of the vertex evolution, which can be specified by users. This parameter controls how many iterations it takes to meet the stop conditions. When γ is set to 1, the loop almost meets the stop conditions with only one time step. So the deformation is very much like traditional FFD of the static model in the sense that it meets the final deformed shape in a single displacement step. The refinement and optimization can only occur before or after the deformation. If the model needs to be extensively refined or optimized to represent the shape deformation accurately during a single deformation operation, the value of γ should be set much smaller. Thus the model is more likely to be refined and optimized during the deformation. Usually we set γ to 0.1. Displaying the above sequence continuously, we can obtain an animation showing the dynamic deformation process.

5 SFD model optimization

5.1 Subdivision-based self-adaptive refinement

Existing adaptive subdivision methods propose only to add triangles in high curvature areas and prevent subdividing of low curvature regions during deformations. While this is also the main intent of our method, those adaptive subdivision schemes apply the subdivision to a static model. This means that if a low curvature region not previously subdivided begins to deform, the model is no longer capable of representing the deformation accurately.

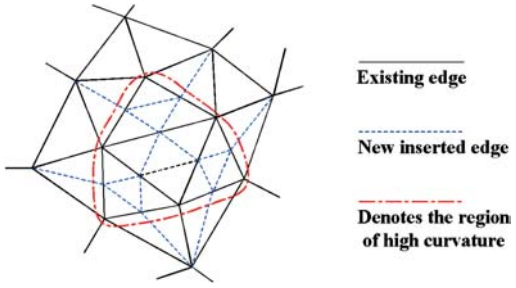


Fig. 4. Local adaptive subdivision in a high curvature region

We interleave shape evolution and self-adaptive refinement within a single deformation. Therefore, it allows the embedded model to represent the deformation more accurately during a deformation. Our method can generate additional triangles on the fly and only in regions that require more subdivision. Low curvature regions not previously subdivided still have the opportunity to be refined during the deformation. As the SFD deforms an object, the curvature of the affected surface is computed and checked to see if subdivision is necessary. Triangles are added only when this criterion is met. Thus our method adds the triangles only in regions that require additional subdivision, and in addition all deformed regions are checked to ensure that subdivision occurs on the appropriate regions. The local subdivision scheme we used in this paper is shown as Fig. 4.
We also incorporate a complementary decimation process that merges faces in nearly planar areas and thereby reduces the polygon–mesh complexity (i.e., the number of vertices, edges, and faces). We trigger decimation by testing the deviation between surface normals at edge endpoints.
With the help of the subdivision and the decimation, this self-adaptive refinement strategy easily supports multiresolution deformation of existing models.

5.2 Mesh improvement

Since our FFD can be a continuous evolution process, we are able to control the dynamic model throughout the deformation process. The mesh quality can be improved and maintained at each time step. In this paper, we consider three issues: a good vertex distribution, a proper vertex density, and a good aspect ratio of the triangles. Much research has been conducted in this field, e.g., [12, 31], producing several valuable algorithms.

We employ three mesh improvement operations: edge-split, edge-collapse, and edge-swap. Conditions under which the operations are valid are discussed in [12]. First, edge-split and edge-collapse are used to keep an appropriate node density. An edge-split is triggered if the edge is too long. Similarly, if any two neighboring vertices are too close to each other, the edge connecting these two vertices will be collapsed. Essentially, these two operations split long edges and delete crowded vertices to ensure a proper vertex density. Then edge-swap is used to ensure a good aspect ratio of the triangles. We swap an edge if doing so will increase the minimum inner angle within its adjacent faces. Repeated applications of this swap operation always increase the minimum inner angle and hence result in a constrained Delaunay triangulation at the end of the procedure. We also try to keep vertices uniformly distributed by performing Laplacian smoothing over the triangulated surface. In practice, these mesh optimization steps are interleaved with the shape deformation iterations so that a good computational mesh is always present. This also helps the iterative solve for minimizing the objective function Eq. 7.

Ordering the operations this way seems to produce the best mesh at the end of the mesh improvement steps. The edge-swap operation can clean up after the simple edge-split and edge-collapse operations, and the mesh smoothing is then invoked to optimize neighborhood shapes. The method of maintaining a good computational mesh over a triangulated surface is iterative and incremental. This makes it appropriate for use in our scalar-field-guided shape deformations, in which shape can change gradually over time. As shown in the flow of the algorithm, interleaving shape evolution and mesh optimization tends to equalize edge lengths, allowing vertices to distribute themselves more evenly during a SFD run.

6 SFD Space construction and deformation

To enable users to employ our SFD technique fexibly to perform shape deformations of existing polygonal objects, our SFD technique is equipped with three stable, easy-to-use approaches for scalar field construction and deformation. We will detail them as follows. Note that topological changes of isosurfaces of scalar fields do not always lead to self-intersections of the embedded models. With our scalar field construction and deformation approaches, scalar fields do not change dramatically, and there are fewer opportunities to cause self-intersections of embedded models (we refer to self-intersections as the "unstable scenarios"). When self-intersections do happen, a robust method for dealing with topological changes is desired. In addition, when manipulating the scalar field with these approaches, the resulting scalar field is continuous and bears the same codomain of the original one. Therefore, we can always compute a unique solution for velocities of all the vertices from Eq. 9.

6.1 Sketch-based distance fields

Sketch-based methods for shape design can facilitate the rapid creation of approximate shapes. Zeleznik et al. [35] showed how a gesture-based modeler could be used to simplify conventional CSG-like shape creation. Teddy [15] extended this to more free-form models, getting much of its power from its "inflation" operation and from an elegant collection of gestures for attaching additional parts to a shape, cutting the shape, and deforming it. Most recently, Karpenko et al. [17] presented free-form sketching with variational implicit surfaces. We employ this conceptual design for scalar field construction and modification. More precisely, in this approach the manipulated scalar field is actually a signed/unsigned distance field. The sketched stroke, or contour, is the silhouette of the zero level set of the resulting distance field. This sketching technique can greatly ease the editing of scalar fields for designers/modelers.

Strokes are gathered from the mouse as a collection of points. In our system, the input strokes are 2D curves and can be open or closed. If open curves are used, the resulting distance field is unsigned, i.e., neither interior nor exterior is defined. The plane containing the 2D curve is called the drawing plane. Unlike the technique used in [15, 17], the constructed

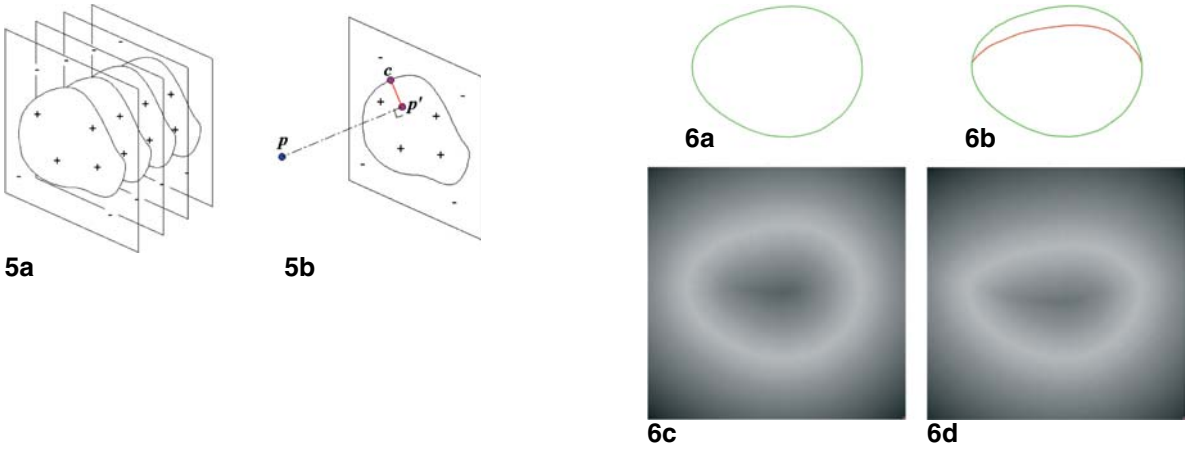


Fig. 5. a A contour is extruded along the perpendicular direction of the drawing plane. **b** The distance value of a point in the space is equal to the distance value of its perpendicular foot on the drawing plane

Fig. 6. The embedding space can be generated by sketching contours and modified by adding another stroke. **c** and **d** are the resulting distance fields on the drawing plane corresponding to **a** and **b**, respectively

zero isosurface does not need to be rounded. Therefore, the inflation in our system is simply for extruding the contour along the perpendicular direction of the drawing plane, as shown in Fig. 5a, until it meets the bounding space or a user-specified bounding region.

In practice, we store only the 2D distance field on the drawing plane since other slices of the 3D distance field along the perpendicular direction are exactly the same (as shown in Fig. 5a). Therefore, to obtain the distance value of any point in the space, we can simply project the point onto the drawing plane along the perpendicular direction of the drawing plane and then assign the perpendicular foot's distance value to the point (Fig. 5b). This method obviates the need to compute the entire 3D distance field. As a result, the computational expense is greatly reduced.

The Euclidean distance to a closed contour can be calculated and stored for each discrete point in an image (Fig. 6c,d). We calculate the distance at each required point using methods such as chamfer distance transforms and vector distance transforms, which propagate known distances throughout the image. An interpolation function is used to determine the distance from any point located within the quad bounded by distances at known grid points. In practice, distance values within a quad are reconstructed from the four corner distance values stored per quad using standard bilinear interpolation.

Figure 6 demonstrates one of the sketching operations in our system. A designer draws a 2D closed stroke as shown in Fig. 6a. Then, a distance field on the drawing plane (Fig. 6c) is generated based on the stroke. To modify the distance field, the designer may draw a stroke starting on the silhouette of the zero level set of the distance field, briefly leaving that silhouette and then returning to it. This sketching results in the deformation of the distance field. Figure 6d shows the deformed distance field.

6.2 Skeleton-based scalar fields

Our system also allows users to interactively sketch skeletons. Then, the scalar field is generated as the blending of field functions g_i of a set of skeletons $s_i (i = 1, \cdots, N)$,

$$f(x, y, z) = \sum_{i=1}^{N} g_i(x, y, z), \tag{10}$$

where the skeletons s_i can be any geometric primitive admitting a well-defined distance function: points, curves, parametric surfaces, simple volumes,

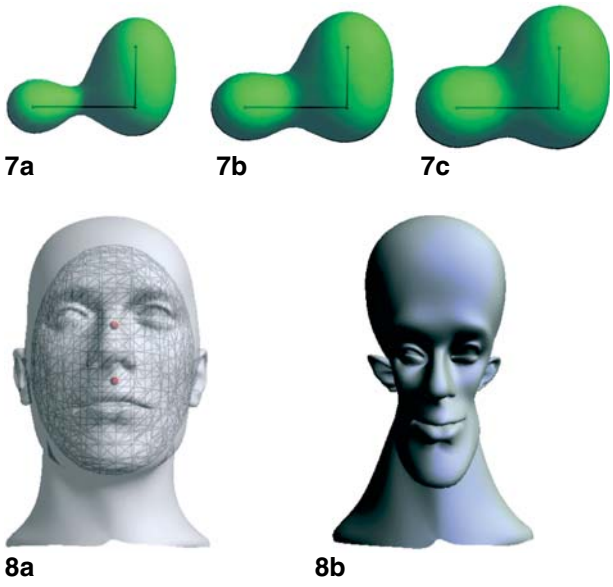


Fig. 7. A scalar field is defined by blending skeletons. Panels **a**, **b**, and **c** show three different isosurfaces that correspond to three different isovalues

Fig. 8. The mannequin model is deformed by manipulating point skeletons. The *mesh* indicates an isosurface of the skeleton-based scalar field

etc. The field functions g_i are decreasing functions of the distance to the associated skeleton,

$$g_i(x, y, z) = G_i(d(x, y, z, s_i)), \tag{11}$$

where $d(x, y, z, s_i)$ is the distance between (x, y, z) and s_i, and G_i can be defined by pieces of polynomials or by more sophisticated anisotropic functions. Figure 7 shows three isosurfaces of a skeleton-based scalar field. The field functions associated with the skeletons are Gaussian functions.

Users can interactively sketch skeletons by a mouse or a 3D pointing device and then define a scalar field according to these skeletons. In general, each skeletal element is associated with a locally defined implicit function; individual functions are blended using a polynomial weighting function that can be controlled by designers. Therefore, designers may enforce global and local control of an underlying scalar field in three separate ways: (1) defining or manipulating the skeleton, (2) defining or adjusting those implicit functions defined for each skeletal element, and (3) defining a blending function to weight the individual implicit functions. In our system, we offer designers the first two ways to modify the scalar field since they are very easy and intuitive and require minimal specialized knowledge. When a designer modifies the scalar field, the embedded objects are deformed according to our SFD algorithm. Figure 8 shows an example. The user defines two point skeletons. The field functions of the skeletons are general Gaussian functions (note that other user-specified blending functions can be straightforwardly incorporated into our system without any difficulties). By repositioning these two skeletons and reducing the strength of the Gaussian functions, the mannequin model is deformed as shown in Fig. 8b.

6.3 Dynamic spline-based scalar fields

Dynamic spline-based implicit functions [14] utilize scalar trivariate B-spline functions as the underlying shape primitives. Different scalar B-spline patches defined over the 3D working space are collected to form a volumetric implicit function that can be used to represent spaces of complicated geometry and arbitrary topology,

$$f(x, y, z) = \sum_{i=1}^{N} s_i(\boldsymbol{T}_i(x, y, z)), \tag{12}$$

where $\boldsymbol{s}_i$ represents the i-th scalar B-spline patch and $\boldsymbol{T}_i$ is an affine transformation that transforms a point in the Euclidian space into the parametric domain

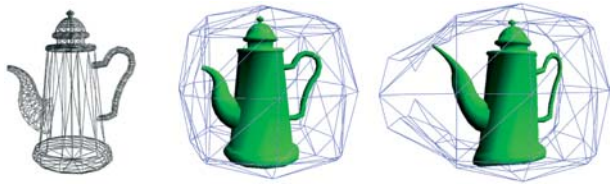


Fig. 9. The teapot polygonal model (**a**) is embedded in the scalar field defined by a dynamic spline-based implicit function (**b**) and deformed by changing the scalar field (**c**). (We only show a single level set using coarse mesh in **b** and **c**)

Fig. 10. The sofa model (**a**) is deformed through direct manipulation of a scalar field using a point force tool (**b**)

of the i-th patch. In essence, Eq. 12 is a hierarchical organization of the N scalar B-spline patches. For details about scalar B-spline expressions and the spline-based volumetric implicit functions, please refer to [13].

The function dynamically changes since its coefficients are time varying. Users can directly manipulate the scalar values to evolve the function. Figure 9 shows a polygonal model embedded in a scalar field defined by a dynamic spline-based implicit function and deformed by manipulating the scalar field.

Please refer to [14] for complete details about dynamic manipulation of spline-based volumetric implicit functions. Here we briefly review the dynamic manipulation mechanism so that this paper is self-contained. Direct and dynamic manipulation of scalar fields is founded on the above spline-based volumetric implicit functions and powerful physics-based modeling. The versatility of the volumetric modeling permits designers to easily modify arbitrary scalar fields, while the inherent physical properties can offer an intuitive mechanism for direct manipulation. The manipulated scalar field can have complicated geometry and arbitrary topologies. Designers can create any scalar fields from scratch and then deform them. The system provides a large number of virtual sculpting tools including geometric tools and force tools. Whenever a sculpting tool is used on the scalar field, the scalar values in the affected regions will be modified correspondingly. When using geometric tools, the designer starts with a simple primitive such as a cube or sphere and gradually constructs a more complex model through successive transformations or a combination of multiple primitives. When using force tools, the designer can directly drag a point or subset of points to change the scalar field. The system will automatically reconstruct the volumetric implicit function to represent the new, modified scalar field undergoing deformation. Therefore, when designers manipulate the scalar field with the system tools, the embedded object inside the scalar field will be deformed according to the change of the scalar field. Figure 10 shows the FFD of a polygonal object (sofa) via dynamic manipulation of the scalar field defined by the spline-based volumetric implicit functions. The user employs a point force tool to directly manipulate the scalar field as if he/she were directly manipulating the embedded model.

7 SFD operations

Given the novel SFD technique and the simple, intuitive, and efficient scalar field construction and deformation approaches, users can easily perform various SFD operations on existing models. For specific SFD operations, the chosen scalar field manipulation approach may have some advantages over the other two available approaches from the perspective of user interaction. However, for end users, those SFD tools are transparent. They do not need to make

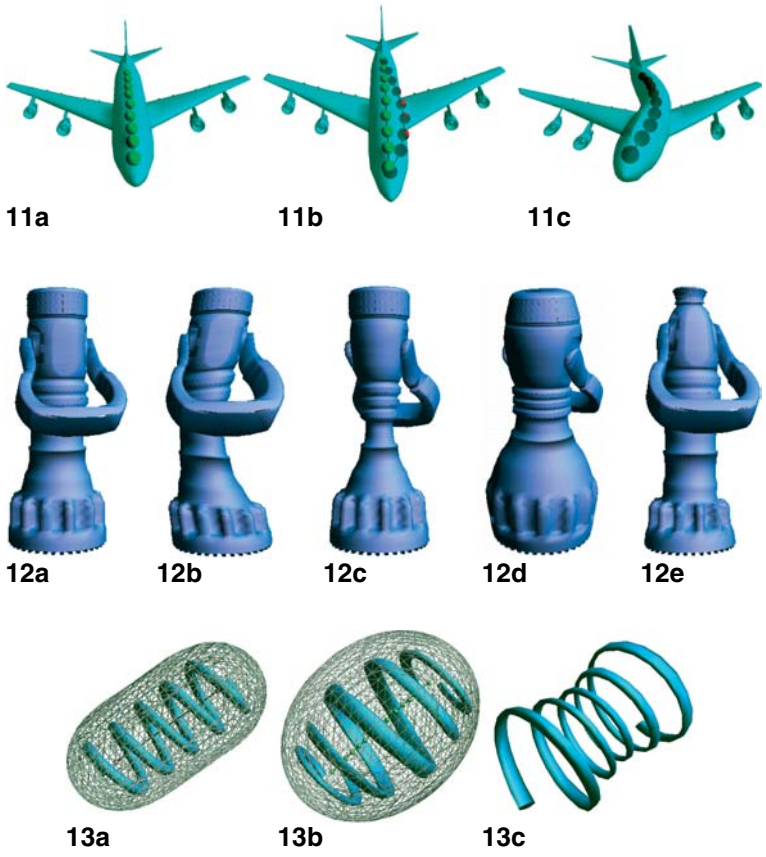


Fig. 11. a shows an airplane model with a skeleton. **b** shows a bending skeleton specified by a designer. **c** shows the resulting bending deformation
Fig. 12. Free-form deformations of a 3D nozzle model. **a** Original model. **b** Bending. **c** Shrinking the middle part. **d** Inflation on both ends. **e** Shrinking the bottom and top
Fig. 13. Inflation and shrinking of a spring model. The meshes show two isosurfaces with the same isovalue in two scalar fields in **a** and **b**, respectively

the choice of which one to use for a specific operation. We equip those SFD tools with the best scalar field manipulation approach.

For ***bending*** operations, users may first draw a set of skeletons to define the source scalar field. As shown in Fig. 11a, the user draws a set of points (blobs) to define the scalar field. Then the user moves the positions of several points, as shown in Fig. 11b. The airplane will bend due to this scalar field modification. Figure 12b shows another bending operation on a 3D nozzle model. In that operation, the user employs a curve skeleton instead. The user simply sketches a straight line segment near the center of the object and a bending curve to define two distance fields. The object is then deformed according to the difference of these two fields.

For ***shrinking and inflation***, users can define two sets of skeletons (namely, a source set and a target set) to define scalar fields. The embedded object will inflate or shrink according to the field deformation from the source to the target. Alternatively, users can just modify some parameters of the source skeletons to perform the deformation. For the examples shown in Fig. 12c–e, the nozzle model inflates or shrinks in several ways. Also as shown in Fig. 13, the user employs Gaussian blobs as skeletons. The mesh surface in Fig. 13a shows an isosurface of the defined source scalar field. Figure 13b shows the deformed

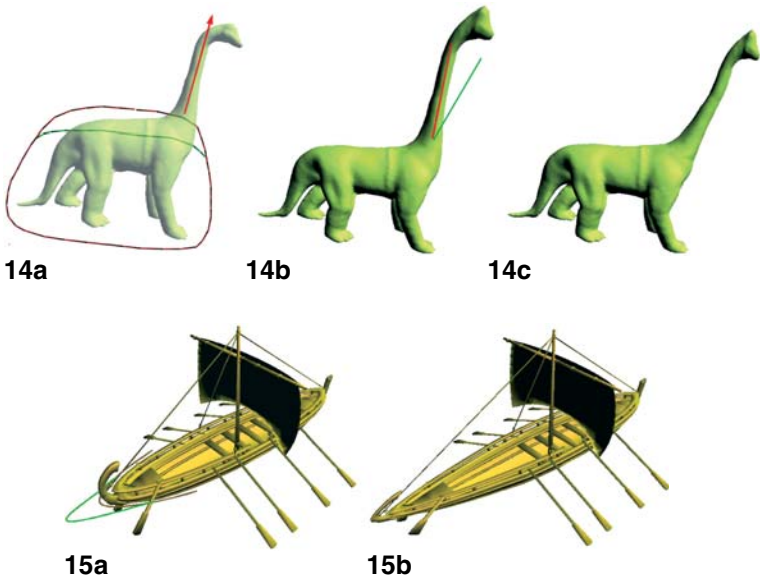


Fig. 14. A dinosaur (**a**) is deformed with the body squeezed (**b**) and the neck stretched (**b**). The user also changes the position of the dinosaur's neck (**c**)
Fig. 15. Tapering on the ship model

object corresponding to the altered scalar field. The mesh surface denotes an isosurface extracted from the target scalar field with the same isovalue of the one shown in Fig. 13a. Figure 13c shows a further deformed model that shrinks the center part of the object shown in Fig. 13b.

For ***squeezing*** operations, users may first sketch a closed stroke to define a distance field and then modify this distance field by adding another stroke. In this operation, we perform deformation only on those vertices with positive distance values. For example, as shown in Fig. 14a, the user first sketches a closed stroke (in red) around the body of the dinosaur. The corresponding distance field based on the sketched stroke is generated using the method described in Sect. 6.1. Then the user sketches another stroke (in green) to deform the distance field. The embedded dinosaur is then squeezed as shown in Fig. 14b according to the SFD. Users can also completely sketch another new stroke to define a target distance field instead of creating it by modifying the existing stroke.

For ***stretching***, users can use force-based tools [14] to directly drag the embedding scalar field, which is equivalent to dragging the embedded object directly. In essence, force-based tools alter the scalar field along the force vector, which results in the stretching effect of the embedded model along the force vector. For the example shown in Fig. 14a and b, the user selects a vertex around the bottom of the dinosaur's neck and then drags it along the arrow to produce a stretching deformation on the dinosaur's neck.

Users can move part of an object to another location by using ***moving*** operations (e.g., Fig. 14b and c). The user sketches a straight line segment near the center line of the dinosaur's neck, which defines a source distance field. Then the user draws another stroke, with a small rotation, to define a target field. The localized dinosaur's neck is then moved as shown in Fig. 14c due to this field change. We will discuss the localization operation in Sect. 8. Currently no physical properties are associated with SFD models. Hence the rigidity issue goes beyond the scope of this paper, though it is a legitimate research concern.

Users can perform ***tapering*** operations by simply sketching strokes. As shown in Fig. 15, the ship model is tapered on the front part. The user first sketches an open stroke (shown in red). Then the user sketches another one (shown in green). The source

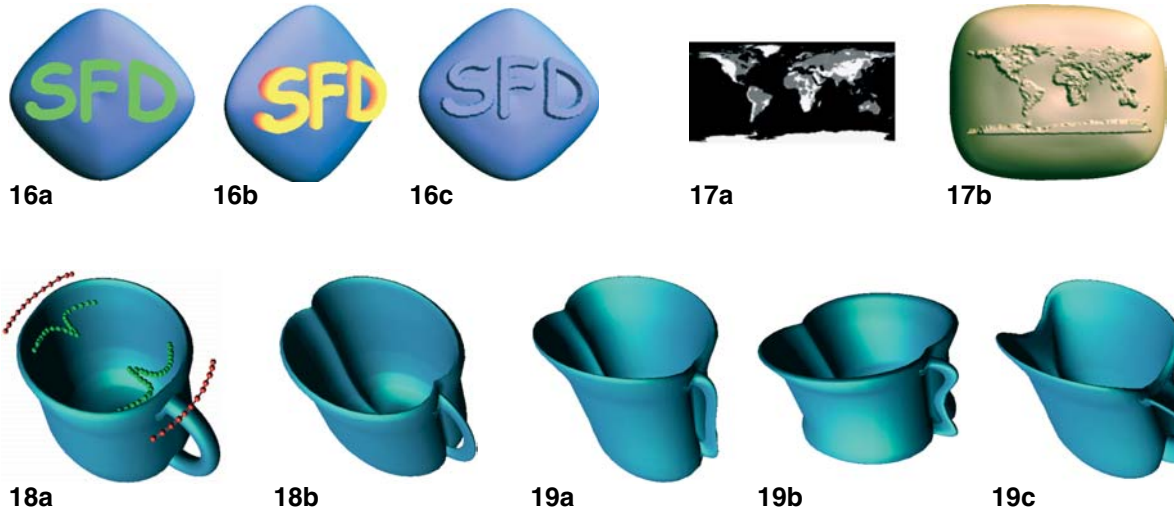


Fig. 16. a The user paints an image on an object. **b** A scalar field can be generated according to the user-painted image and specified height values. **c** The object is deformed by the generated scalar field

Fig. 17. Panel **a** shows a grayscale image. A scalar field can be generated from any grayscale image. The gray values of the image are used as projective distances along the normals to produce a surface. The scalar value inside the space is the shortest distance to the generated surface. Panel **b** shows the deformed object

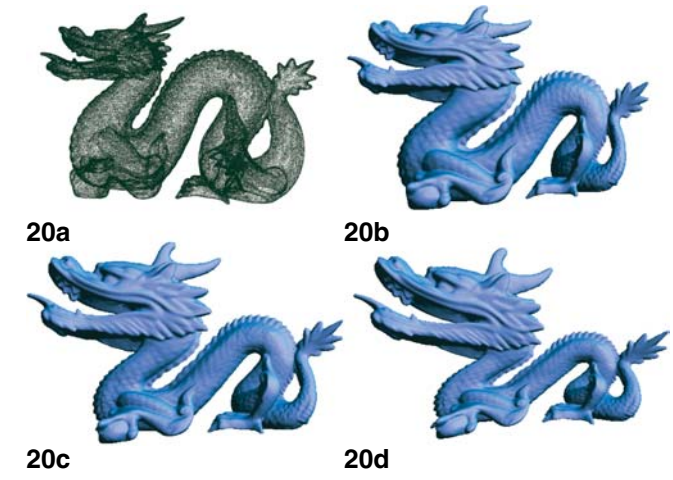
Fig. 18. Creasing operation. **a** Original model and sketched strokes around both sides of the cup. The *red* strokes are for generating the source distance field, while the *green* strokes are for generating the target distance field. **b** Two sharp creases formed on both sides of the cup

Fig. 19. Deformation using two analytic implicit functions with different parameter configurations. The original model is the deformed polygonal cup model with creases as shown in Fig. 18b

and target distance fields are generated locally based on these two strokes. The localized front part is then tapered according to the field change.

Users can also easily apply ***embossing*** and ***engraving*** operations on an object. In these operations, users can paint a grayscale image or use an existing one to define an embedding space on the object. For example, as shown in Fig. 16, the designer paints a binary image directly on a 3D object. Then a surface S is obtained by projecting the sampling points of the image along the normal directions with user-specified distances. Our system then generates a local distance field in the region of interest according to the surface. Figure 16a shows letters painted on an object by the designer. Figure 16b shows the generated embedding space according to the specified height, where the color denotes the distance value. The red color indicates the longer distance inside the space, and the yellow color indicates shorter distance. In embossing and engraving operations, the source scalar field around the base model is initialized to zero everywhere. Therefore, the deformation based on the zero source scalar field and the locally constructed distance field will produce embossing and engraving effects on the embedded models. Figure 16c shows the result after performing the SFD embossing.

Figure 17 shows another embossing example using an existing image. Figure 17a is an image of a global map. Here, a surface S is obtained by projecting the sampling points along the object's normals with corresponding grayscale values. Our system then generates a local distance field, where the scalar values are the shortest distances to the surface. The source scalar field is also initialized to zero everywhere. As shown in Fig. 17b, this image is embossed onto a "soap" shape by using this space construction approach together with our SFD technique, i.e., the same technique as used in the example of Fig. 16. If we project the distance field



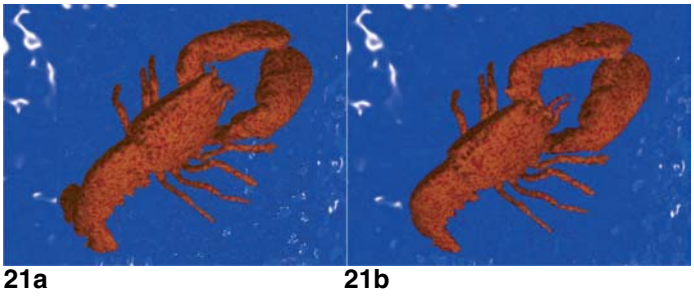
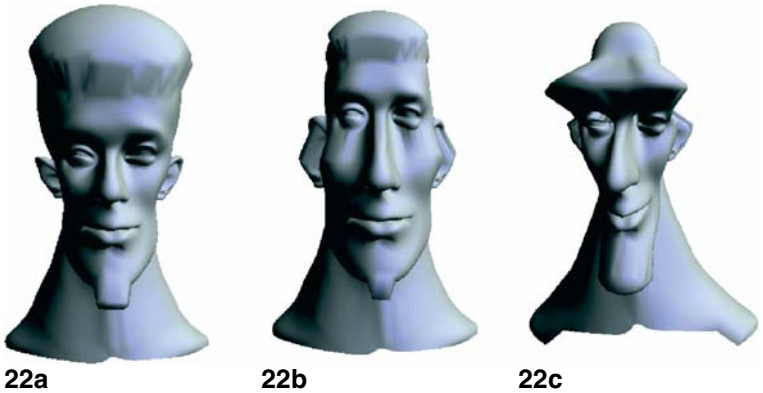


Fig. 20. Deformation of a point-set object. The dragon model contains 437 645 vertices. **a** Original point set, deformed several times using the SFD operations. The deformed results are shown in **b**, **c**, and **d**
Fig. 21. Deformation of a volumetric object. The lobster dataset is a CT scanned dataset. **a** Original isosurface. **b** Deformed isosurface with bending tail

along the reverse direction, we can also easily perform engraving operations on the models. Note that the used image is similar to the concept of the displacement map [18]. The difference is that we do not need to explicitly map the displacement to each point of the base model. The resolution or configuration of the displacement map can be different from that of the base surfaces. In our approach, once the distance field is generated, it can guide the deformation of the base surface and then produce the embossing or engraving effects. The mesh of the base surface is optimized and refined during the deformation to represent the detail of *to-be-embossed* features.

It is very easy to use SFD to make sharp creases on the embedded models, which is theoretically difficult with traditional FFD. Users can perform ***creasing*** operations by simply sketching strokes. As shown in Fig. 18, the user makes two shape creases on both sides of the deformed cup model. The user first localizes the region where the creases will be formed and then sketches two open strokes (shown in red) for generating source distance fields. Two unsigned distance fields are computed based on these two strokes in the localized regions, which are near the side of the cup model. Further, the user sketches two other open curves, shown in green. The target unsigned distance fields are generated based on these two curves in the same regions. The SFD forms the sharp creases according to the field change from the source one to the target one. Note that the crease can be very sharp. During the creasing deformation



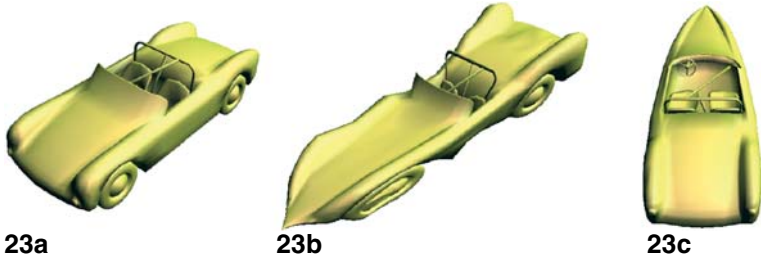


Fig. 22. The mannequin model is deformed by using the available deformation operations along with the use of point, curve, normal, and curvature constraints during the design process. In **a** and **b**, we produce a bumpy hair-like effect
Fig. 23. Free-form deformations on the car model with the use of available deformation operations and constraining methods. **a** Original model. **b** Deformed model. **c** Boat-like model deformed from **a**

some vertices connected by very short edges are collapsed at the top of the crease so that the top of the sharpest crease is just composed of a series of edges. In general, this cannot be done using traditional FFD methods.

Generally, our SFD technique can support any type of deformation provided that the appropriate source scalar field and target field are given. Hence users can define two arbitrary implicit functions analytically to embed an object to be deformed. We call one a source implicit function, the other a target implicit function. With SFD, the embedded object will be deformed according to scalar fields of the specified functions. As shown in Fig. 19, the object is deformed with the use of an ellipsoid function as the source function and a hyperboloid function of one sheet as the target function. With different parameter configurations users can easily produce the deformations (Fig. 19).

The SFD operations can be applied to point-set objects and volumetric datasets in the same fashion. Figure 20 shows an example of deformation of a point-set object. Before deformation, the point-set object needs to undergo preprocessing. For each vertex of the object, the preprocessing finds the neighboring vertices within a specified distance to the vertex. This information will be used to calculate the average speed among the vertices within a local region (Eq. 8). The point-set surface is rendered using splatting. Figure 21 shows an example of deformation of a volumetric object. We apply the SFD operation on the underlying grid of the volumetric dataset. At every time step, the operation first results in the change of positions of the underlying grid. But the associated density val-

Table 1. Run time for SFD operations on existing polygonal models. The number of vertices of the lobster volumetric model denotes the size of its underlying sampling grid

Model	No. of vertices	No. of faces	Update time(ms)
Airplane	2965	5706	0.12
Car	3307	6556	0.14
Cup	3749	7494	0.16
Ship	4892	9510	0.23
Nozzle	5845	9877	0.28
Mannequin	6737	13 408	0.33
Dinosaur	23 984	47 904	1.25
Dragon	437 645	N/A	25.58
Lobster	$104 \times 104 \times 34$	N/A	20.44

ues do not change. Therefore, the uniform volumetric dataset becomes an unstructured one. Then we employ a trilinear interpolation and resampling process to distribute the unstructured scalar field to a uniform one. The scalar field can be visualized and the isosurface of the volumetric object deformed.

8 Enforcing geometric constraints

Our technique provides a general deformation mechanism in the sense that it can easily accommodate various geometric constraints on the embedded models. Adding geometric constraints can make our technique more powerful and facilitate feature-based design. To enforce constraints, we can simply augment the original objective function E with the constraint energy E_c,

$$E_n = E + \rho E_c, \tag{13}$$

where E_n denotes the new, overall objective function and E_c denotes the additional energy term introduced by added constraints. Note that E_c can be a linear combination of several energy functionals. The following list contains discretized versions of their corresponding continuous forms to polygonal meshes.

Position constraint

$$E_{pos} = (p_i - p_i')^2, (p_i \in \boldsymbol{P}), \tag{14}$$

where p_i represents a point before a deformation, p_i' represents the same point after a deformation, and $\boldsymbol{P}$ denotes the point set of the object.

Curve constraint

$$E_{curv} = \sum_{p_i \in C} (p_i - p_i')^2, \tag{15}$$

where C denotes a constraint curve specified by the designer. In the implementation, C is discretized into a set of points.

Normal constraint

$$E_{norm} = \sum_{p_i \in S} (N(p_i) - N(p_i'))^2, \tag{16}$$

where $N(p_i)$ denotes the normal at point p_i and S denotes a set of points inside the region where the normal is to be preserved.

Curvature preserving

$$E_{curvature} = \sum_{p_i \in S} (K(p_i) - K(p_i'))^2, \tag{17}$$

where $K(p_i)$ denotes the total curvature at point p_i and S denotes a region where the curvature is to be preserved.

When the locations where the user specifies constraints do not have vertices of the embedded model, the new vertices at those locations will be inserted. Then the model is optimized several times with those constrained vertices fixed.

A standard implicit iterative method can be used to numerically compute the minimization of the overall objective function. The advantage of this approach is that it is relatively general and can offer an accurate, stable solution even for very large systems. Therefore, it is well suited for our purposes in SFD operations. The gradient used in this minimization process is numerically approximated using the central difference of the overall objective function for the current position of the model vertex with a very small perturbation.

We can also perform the local SFD operation on polygonal models by only allowing the movement of the vertices, where the scalar value evaluates within a specified scope of the isovalues. For other vertices, they will not move during the deformations. Users can specify any implicit function to localize the part of the models to be deformed. In our prototype system, we provide users with three types of primitives to localize regions, which include rectangular box, cylinder, and sphere. By combining these

primitives, users are able to localize any part of embedded models.

With all the available operations and constraining methods, we perform some interesting deformations on the mannequin model, as shown in Fig. 22. The mannequin model is significantly deformed to produce some interesting examples. During the deformations, some facial features are maintained through the use of constraints. In Fig. 22a and b, we drop the smoothness constraints in Eq. 6 during the deformation and allow only the vertex velocities along the specified directions to produce a bump-like hair effect. Figure 23 shows another deformation example on the car model. Figure 23b shows a new racing car model, rebuilt (deformed) from the old model as shown in Fig. 23a. Figure 23c shows a boat deformed directly from the car model shown in Fig. 23a.

9 Implementation

Our system is written in Microsoft Visual C++ and the rendering is implemented using OpenGL. In our system, we incorporate the Phantom 1.0 device from Sensable Technologies into our user interface. During the deformation process, users can use a mouse or the Phantom to manipulate the scalar field. When directly manipulating spline-based scalar fields using force tools [14], users can even feel the deformation force applied to the deformable model, which results in the deformation.

We have experimented with our SFD technique on a wide range of PCs with no special hardware. Those deformations can be performed on a relatively low-end system in real time. Unlike subdivision-based FFD techniques, our SFD technique does not require a large amount of memory. From a theoretical point of view, the timing achieved depends essentially on the number of vertices of the models and the evaluation time of the scalar fields at those vertices. On a 2.2-GHz PC with 1 GB RAM we have performed the same inflation deformations on the models. In this test, the used scalar field is a simple point skeleton admitting a Gaussian field function. We have examined the timings achieved for a single deformation iteration (i.e., one run from step 1 to step 6 of our algorithm described in Sect. 4). The recorded time, which is the CPU time of the single deformation iteration minus the rendering time,is shown in Table 1.

10 Conclusion

In this paper, we have articulated a novel scalar-field-guided shape deformation, or SFD, methodology founded on PDE-based flow constraints and scalar field functions. The new SFD paradigm is fundamentally different from traditional FFD techniques in that we employ scalar fields as SFD embedding spaces. A scalar-field-based embedding space is of diverse types and its space definition is much simpler yet both powerful and intuitive for various visual modeling applications. In our prototype system, we have developed a suite of easy-to-use techniques for efficiently constructing and manipulating the space as well as flexibly interacting with various geometric shapes. With the SFD method, users can directly sketch the scalar field of an implicit function via a mouse or a 3D haptic interface to control the deformation of any embedded model. In addition, the embedding space can be of complicated geometry and arbitrary topology. The deformation velocity for any model point can be either very general or constrained subject to user-specified requirements. Therefore, our SFD technique supports a larger number of shape deformation types than other techniques. Furthermore, the embedded model has self-adaptive and optimization capabilities throughout the SFD process to accommodate versatile deformations, maintain the mesh quality, and preserve shape features. We have conducted a large number of experiments that demonstrate that our new SFD technique is powerful, flexible, natural, and intuitive for shape modeling and geometric design in animation and interactive graphics.

Our current system does not deal with deformations where parameterization, such as torsion, is used. But this could be easily solved by introducing additional parameterized variables into pure scalar fields. At present, we are exploring new research in several directions. First, the continuity between the deformed part and the undeformed part is difficult to control. Manual specification from users has been used frequently in traditional FFD techniques. It is, however, much more challenging in our system because SFD is essentially a dynamic process. Second, our SFD technique is potentially useful for creating animated characters in the film industry and shortening the animation design cycle. With a few key frames generated from our SFD technique, it is also possible to use our SFD technique to obtain the “in-between” ones containing the entire motion sequence of ob-

jects. If we further incorporate physical properties and other relevant constraints, realistic simulation and animation can be achieved. Third, in these applications, preventing self-intersections or performing correct topology changes during deformations should also be considered. Finally, we would like to enhance all the deformation operations with haptics, so that the user can have realistic, physical, and force feedback when manipulating SFD models.

Acknowledgements. The models are courtesy of the 3D CAFE, the Graphics and Imaging Laboratory of the University of Washington, and the Large Geometric Models Archive of the Georgia Institute of Technology. This research was supported in part by NSF ITR Grant IIS-0082035, NSF Grant IIS-0097646, and an Alfred P. Sloan Fellowship.

References

1. Bærentzen J, Christensen N (2002) Volume sculpting using the level-set method. In: Proceedings of the international conference on shape modelling and applications, Banff, Canada, 17–22 May 2002. IEEE Press, Los Alamitos, CA, pp 175–182
2. Barr AH (1984) Global and local deformations of solid primitives. In: Proceedings of SIGGRAPH '84, Minneapolis, United States, 23–27 July 1984. ACM SIGGRAPH, New York, pp 47–58
3. Blinn J (1982) Generalization of algebraic surface drawing. ACM Trans Graph 1(3):235–256
4. Bloomenthal J, Bajaj C, Blinn J, Cani-Gascuel M-P, Rockwood A, Wyvill B, Wyvill G (1997) Introduction to implicit surfaces. Morgan Kaufmann, San Francisco
5. Bloomenthal J, Wyvill B (1990) Interactive techniques for implicit modeling. In: Proceedings of the symposium on interactive 3D graphics, Snowbird, Uhta, United States, 13–15 March 1990. Comput Graph 25(2):109–116
6. Breen DE, Whitaker RT (2001) A level-set approach for the metamorphosis of solid models. IEEE Trans Vis Comput Graph 7(2):173–192
7. Chang Y-K, Rockwood AP (1994) A generalized de casteljau approach to 3D free-form deformation. In: Proceedings of SIGGRAPH '94, Orlando, Florida, United States, 24–29 July 1994. ACM SIGGRAPH, New York, pp 257–260
8. Coquillart S (1990) Extended free-form deformation: a sculpting tool for 3D geometric modeling. In: Proceedings of SIGGRAPH '90, Dallas, Texas, United States, 6–10 August 1990. ACM SIGGRAPH, New York, pp 187–196
9. Coquillart S, Jancéne P (1991) Animated free-form deformation: an interactive animation technique. In: Proceedings of SIGGRAPH '91, Las Vegas, Nevada, United States, 28 July – 2 August 1991. ACM SIGGRAPH, New York, pp 23–26
10. Crespin B (1999) Implicit free-form deformations. In: Proceedings of Implicit Surfaces '99, Bordeaux, France, 13–15 September 1999, pp 17–23
11. Desbrun M, Cani-Gascuel M-P (1998) Active implicit surface for animation. In: Proceedings of Graphics Interface, Halifax, Canada, 18–20 June 1998, pp 143–150
12. Hoppe H, Derose T, Duchamp T, McDonald J, Stuetzle W (1993) Mesh optimization. In: Proceedings of SIGGRAPH '93, Anaheim, California, United States, 1–6 August 1993. ACM SIGGRAPH, New York, pp 19–26
13. Hua J, Qin H (2001) Haptic sculpting of volumetric implicit functions. In: Proceedings of the 9th Pacific conference on computer graphics and applications, Tokyo, Japan, 16–18 October 2001. IEEE Press, Los Alamitos, CA, pp 254–264
14. Hua J, Qin H (2002) Haptics-based volumetric modeling using dynamic spline-based implicit functions. In: Proceedings of the symposium on volume visualization and graphics, Boston, Massachusetts, United States, 28–29 October 2002. ACM Press, New York, pp 55–64
15. Igarashi T, Matsuoka S, Tanaka H (1999) Teddy: a skeching interface for 3D freeform design. In: Proceedings of SIGGRAPH '99, Los Angeles, California, United States, 8–13 August 1999. ACM SIGGRAPH, New York, pp 409–416
16. Jin X, Li Y, Peng Q (2000) General constrained deformations based on generalized metaballs. Comput Graph 24(2):219–231
17. Karpenko O, Hughes JF, Raskar R (2002) Free-form sketching with variational implicit surfaces. In: Proceedings of Eurographics 2002, Saarbrücken, Germany, 2–6 September 2002. Comput Graph Forum 21(3):585–594
18. Krishnamurthy V, Levoy M (1996) Fitting smooth surfaces to dense polygon meshes. In: Proceedings of SIGGRAPH '96, New Orleans, Louisiana, United States, 4–9 August 1996. ACM SIGGRAPH, New York, pp 313–324
19. Lazarus F, Coquillart S, Jancene P (1994) Axial deformations: an intuitive deformation technique. Comput Aided Des 26(8):607–612
20. MacCracken R, Joy KI (1996) Free-form deformations with lattices of arbitrary topology. In: Proceedings of SIGGRAPH '96, New Orleans, Louisiana, United States, 4–9 August 1996. ACM SIGGRAPH, New York, pp 181–188
21. Malladi R, Sethian JA, Vemuri BC (1995) Shape modeling with front propagation: a level set approach. IEEE Trans Patt Anal Mach Intell 17(2):158–175
22. Museth K, Breen DE, Whitaker RT, Barr AH (2002) Level set surface editing operators. In: Proceedings of SIGGRAPH '02, San Antonia, Texas, United States, 21–26 July 2002. ACM SIGGRAPH, New York, pp 330–338
23. Osher S, Sethian JA (1988) Fronts propagating with curvature-dependent speed: algorithms based on hamilton-jacobi formulations. J Comput Phys 79:12–49
24. Pasko A, Adzhiev V, Sourin A, Savchenko V (2001) Function representation in geometric modeling: concepts, implementation and applications. Vis Comput 11(8):429–446
25. Raviv A, Elber G (1999) Three dimensioinal freeform sculpting via zero sets of scalar trivariate functions. In: Proceedings of the 5th ACM symposium on solid modeling and applications, Los Angeles, California, United States, 8–13 August 1999. ACM Press, New York, pp 246–257
26. Schmitt B, Pasko A, Schlick C (2001) Constructive modeling of frep solids using spline volumes. In: Proceedings of the 6th ACM symposium on solid modeling and applications, Los Angeles, California, United States, 12–17 August 2001. ACM Press, New York, pp 321–322

27. Schmitt B, Pasko A, Schlick C (2003) Shape-driven deformations of functionally defined heterogeneous volumetric objects. In: Proceedings of GRAPHITE, Melbourne, Australia, 11–14 February 2003, pp 321–322
28. Sederberg T, Parry SR (1986) Free-form deformation of solid geometric models. In: Proceedings of SIGGRAPH '86, Dallas, Texas, United States, 18–22 August 1986. ACM SIGGRAPH, New York, pp 151–160
29. Singh K, Fiume E (1998) Wires: a geometric deformation technique. In: Proceedings of SIGGRAPH '98, Orlando, Florida, United States, 19–24 July 1998. ACM SIGGRAPH, New York, pp 405–414
30. Turk G, O'Brien JF (2002) Modelling with implicit surfaces that interpolate. ACM Trans Graph 21(4):855–873
31. Welch W (1995) Serious putty: topological design for variational curves and surfaces. PhD thesis, Carnegie Mellon University, Pittsburgh
32. Whitaker R (1998) A level-set approach to 3D reconstruction from range data. Int J Comput Vision 29(3):203–231
33. Witkin A, Heckbert P (1994) Using particles to sample and control implicit surfaces. In: Proceedings of SIGGRAPH '94, Orlando, Florida, United States, 24–29 July 1994. ACM SIGGRAPH, New York, pp 269–278
34. Wyvill G, McPheeters C, Wyvill B (1988) Data structure for soft objects. Vis Comput 2(4):227–234
35. Zeleznik RC, Herndon K, Hughes J (1996) Sketch: an interface for sketching 3D scences. In: Proceedings of SIGGRAPH '96, New Orleans, Louisiana, United States, 4–9 August 1996. ACM SIGGRAPH, New York, pp 163–170

JING HUA is a Ph.D. candidate in computer science at the State University of New York (SUNY) at Stony Brook, where he is also a research assistant in the SUNYSB Center for Visual Computing (CVC). He received his B.E. (1996) in electrical engineering from the Huazhong University of Science and Technology in Wuhan, P.R. China and his M.E. (1999) in pattern recognition and artificial intelligence from the Institute of Automation, Chinese Academy of Sciences in Beijing, P.R. China. In 2001, he received his M.S. in computer science from SUNY Stony Brook. His research interests include geometric and physics-based modeling, scientific visualization, interactive 3D graphics, human-computer interface, and computer vision. He is a student member of IEEE and ACM. For more information see http://www.cs.sunysb.edu/∼jinghua.

DR. HONG QIN is an associate professor of computer science at SUNY Stony Brook, where he is also a member of the SUNYSB Center for Visual Computing. He received his B.S. (1986) and his M.S. (1989) in computer science from Peking University in Beijing, P.R. China and his Ph.D. (1995) in computer science from the University of Toronto. From 1989 to 1990 he was a research scientist at North China Institute of Computing Technologies. From 1990 to 1991 he was a Ph.D. candidate in computer science at the University of North Carolina at Chapel Hill and from 1996 to 1997 he was an assistant professor of computer and information science and engineering at the University of Florida. In 1997, Dr. Qin was awarded the NSF CAREER Award from the National Science Foundation (NSF) and in September 2000 was awarded a newly established NSF Information Technology Research (ITR) grant. In December 2000, he received a Honda Initiation Grant Award and in April 2001 was selected as an Alfred P. Sloan Research Fellow by the Sloan Foundation. He is a member of ACM, IEEE, SIAM, and Eurographics.