Free-Form Deformations via Sketching and Manipulating Scalar Fields

Jing Hua Hong Qin
Computer Science, SUNY @ Stony Brook
{jinghua—qin}@cs.sunysb.edu

ABSTRACT

This paper presents a novel Scalar-field based Free-Form Deformation (SFFD) technique founded upon general flow constraints and implicit functions. In contrast to the traditional lattice-based FFD driven by parametric geometry and spline theory, we employ scalar fields as embedding spaces instead. Upon the deformation of the scalar field, the vertices will move accordingly, which result in free-form deformations of the embedded object. The scalar field construction, sketching, and manipulation are both natural and intuitive. By tightly coupling self-adaptive subdivision and mesh optimization with SFFD, versatile multi-resolution free-form deformations can be achieved because our algorithm can adaptively refine and improve the model on the fly to improve the mesh quality. We can also enforce various constraints on embedded models, which enable our technique to preserve the shape features and facilitate more sophisticated design. Our system demonstrates that SFFD is very powerful and intuitive for shape modeling. It significantly enhances traditional FFD techniques and facilitates a larger number of shape deformations.

Categories and Subject Descriptors

I.3.5 [**Computer Graphics**]: Computational Geometry and Object Modeling; I.3.6 [**Computer Graphics**]: Methodology and Techniques—*interaction techniques*

General Terms

Algorithms, Design

Keywords

Deformations, Interaction Techniques, Scalar Fields

1. INTRODUCTION

Free-Form Deformation (FFD) is an important technique in shape modeling, animation, and simulation. FFD-based techniques applied on an existing object are an appealing alternative to traditional modeling since they are independent of objects' representations. Various FFD techniques have been proposed during the past two decades [18, 5, 4, 12, 13]. However, there are several difficulties

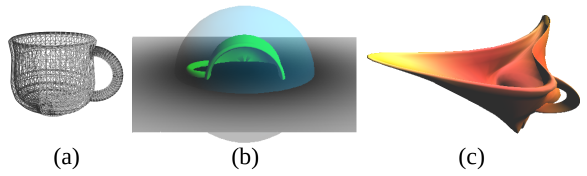

Figure 1: An object (a) is embedded in a scalar field (b) and deformed by altering the scalar field (c). (b) shows a cross-section of the entire scalar field (see the plane) and a single level set (see the transparent surface). (c) shows the deformed object.

associated with current FFD techniques. First, performing deformations with complex control lattices is extremely time-consuming in general. Although the axis-based FFD approaches and the directly manipulated FFD approaches [4, 12] are relatively intuitive and efficient, those methods can only offer limited deformations. Second, the control lattice is less flexible and cannot have arbitrary topology easily. It is difficult to afford a large number of deformation types. Third, the traditional FFD operation is generally a single operation applied on static models. The refinement and optimization can only occur before or after deformation. If a low curvature and low resolution region is not subdivided prior to the deformation, the model is no longer capable of representing the deformation accurately.

We propose a novel free-form deformation technique, or SFFD, based on vertex flow constraints and implicit functions, which employs a scalar field as the embedding space. Users can interactively sketch a scalar field of an implicit function via a mouse to embed an entire model or a part of the model. The embedding space based on scalar field is of diverse type, which implicitly defines a complicated geometry and an arbitrary topology. Upon the deformation of the embedding space (i.e., the modification of scalar field), the vertices will move according to the enforced flow constraints, which results in free-form deformations of the embedded object. The deformation velocity of each vertex on the model is very general and can adopt any user-desired constraints easily. Our SFFD technique generalizes traditional FFD technique and affords a larger number of shape deformations. The scalar field construction, sketching, and manipulation are more natural and easy-to-use than previous FFD techniques.

Furthermore, in order to represent deformations more accurately, the embedded models are equipped with self-optimization capability. Adaptive subdivision and mesh optimization are tightly coupled with SFFD, supporting versatile multi-resolution free-form deformations. Since our SFFD can be an evolution process, it allows self-adaptive refinement and mesh improvement to interleave with shape deformation throughout the SFFD operation. Our algorithm can adaptively subdivide the model in regions that require high resolution. As the SFFD deforms an object, the curvature of the affected surface is checked to see if subdivision is necessary. We also incorporate various constraints on embedded models, which enable our technique to facilitate feature-based design. Our results demonstrate that the proposed FFD technique is very useful and powerful for shape editing and solid design.

2. RELATED WORK

Sederberg and Parry [18] pioneered the FFD concept on solid geometry. However, FFD can be accomplished only with a parallelpiped lattice structure. Coquillart [5] developed the Extended Free-Form Deformation, or EFFD, as an extension of Sederberg and Parry's technique, which uses non-parallelepiped 3D lattices. MacCracken and Joy [13] presented a free-form deformation technique, which uses arbitrary lattices, namely, Catmull-Clark subdivision volumes. This technique allows a variety of deformable regions to be defined, and thus a broader range of shape deformations can be generated. However, the lattice space definition is time-consuming and difficult. This technique requires a great deal of CPU time and memory. Later, Jin *et. al.* [10] proposed a constrained local deformation technique based on generalized metaballs. Singh and Fiume [19] presented *wires* for interactive, geometric deformation. Crespin [6] presented a free-form deformation technique with the use of deformation primitives. Note that, the combination of deformations with the use of blending functions is counter-intuitive and need be tuned for each different type of deformations. Our paper introduces a new SFFD technique by establishing deformation methods defined on scalar fields, which avoids parameterization process.

As for interactive design based on scalar field, Bloomenthal *et. al.* [2] used skeleton methods to construct implicit surfaces. Blobby model, also known as soft object [22], is another popular technique for the design of implicit surfaces. Recently, Raviv and Elber [16] presented a 3D interactive sculpting paradigm that employed a set of scalar uniform trivariate B-spline functions as object representations. Schmitt and Pasko [17] presented an approach for constructive modeling of FRep solids defined by real-valued functions using 4D uniform rational cubic B-spline volumes as primitives. Turk and O'Brien [20] introduced new techniques for modeling with interpolating implicit surfaces.

Our work is also related to level-set approaches. Osher and Sethian first presented the level set methods [15]. 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. In graphics, Desbrun *et. al.* [7] and Breen *et. al.* [3] used the level set methods for shape morphing. Most recently, Museth *et. al.* and [14] and Bærentzen *et. al.* [1] presented a level set framework for interactively editing implicit surfaces. 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 be first converted into volumetric representation and represented as a single iso-surface.

3. SFFD ALGORITHM

In general, surfaces expressed by an implicit form can be formulated as $\{\mathbf{X} \in \mathbb{R}^3 | f(\mathbf{X}) = c\}$. The function f is called the implicit function, or the field function, which defines a scalar field. The related level set (also called the iso-surface) is corresponding to an iso-value c. The function f may be of any mathematical expression. 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 FFD embedding space, which wraps a *to-be-deformed* object. Note that, fundamentally different from traditional FFD space, the SFFD space is an implicit function based on scalar field instead of a lattice-based geometric object. Figure 2(a) shows a 2D implicit function and Figure 2(b) is the scalar field corresponding to the function. Deforming the scalar field to the one shown in Figure 2(c), an embedded 2D object is deformed accordingly. Figure 1 shows a 3D deformation example. The teacup model is embedded inside a 3D scalar field. After changing the scalar field, the teacup is significantly deformed.

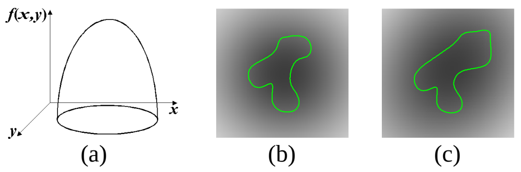

Figure 2: (a) shows an implicit function graph and (b) shows its corresponding scalar field. A 2D object (a free-form planar curve) is embedded in the 2D scalar field as shown in (b) and deformed by changing the embedding space as shown in (c).

Now let us overview our idea of applying scalar field of implicit function to perform free-form deformations on 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 the vertices of that embedded part. Then during the FFD process the vertices are always constrained on the level sets where they originally reside by enforcing vertex-flow constraints. Once users deform the scalar field, the vertex will move accordingly, which results in free-form deformation of the embedded object.

Since the SFFD enforces that the relocation position of a vertex $\mathbf{X}(t)$ of the deformed object remains on the same level set when scalar-field space is deformed, the trajectory of the vertex can be represented as $\{\mathbf{X}(t) | f(\mathbf{X}(t), t) = c\}$. The derivative of $f(\mathbf{X}(t), t)$ yields

$$\frac{d f(\mathbf{X}(t),t)}{d\mathbf{X}} = \frac{\partial f(\mathbf{X}(t),t)}{\partial \mathbf{X}} \frac{d\mathbf{X}(t)}{dt} + \frac{\partial f(\mathbf{X}(t),t)}{\partial t} = 0, \qquad (1)$$

where $\frac{\partial f(\mathbf{X}(t),t)}{\partial \mathbf{X}}$ is the gradient at $\mathbf{X}$. To simplify the notation, we represent the gradient using $\nabla \mathbf{f}$, and abbreviate $f(\mathbf{X}(t),t)$ to f. Therefore, (1) can be re-written as follows:

$$\nabla \mathbf{f} \cdot \frac{d\mathbf{X}(t)}{dt} + \frac{\partial f}{\partial t} = 0. \qquad (2)$$

Note that $\frac{d\mathbf{X}(t)}{dt}$ and $\frac{\partial f}{\partial \mathbf{X}}$ are both vectors. Therefore, there is an ambiguity and the solution for the vertex velocity from (2) is not unique. Only from this flow constraint, the velocity $\frac{d\mathbf{X}(t)}{dt}$ could not be uniquely solved. Dividing $\frac{d\mathbf{X}(t)}{dt}$ into $(\mathbf{v_n}, \mathbf{v_t}, \mathbf{v_w})$, where $\mathbf{n} = \frac{\nabla \mathbf{f}}{\|\nabla \mathbf{f}\|}$ represents the unit principle normal vector of the iso-surface of the scalar field, $\mathbf{t}$ represents the unit tangent vector, and $\mathbf{w}$ represents the unit binormal vector, we know that only the normal velocity, $\mathbf{v_n}$, is perpendicular to the constraint line. So the dot product in (2) only retains the item containing $\mathbf{v_n}$. Therefore, we can obtain the velocity of the normal flow, $\mathbf{v_n} = -\frac{1}{\|\nabla \mathbf{f}\|} \frac{\partial f}{\partial t} \mathbf{n}$. This normal flow scheme is essentially the basis of level set methods, which has been widely used in computer vision, and level-set based applications [7, 3] for tracking the target objects. The normal flow scheme only considers the evolution velocities along the normals of iso-surfaces. It is good for minimizing the similarity between the active model and target model. However, the intermediate deformations are not natural and are not addressed in the aforementioned work. In essence, the main task of level set methods is to design the normal velocities $\mathbf{v_n}$ to evolve the function f.

Instead, our objective is to provide a general FFD technique. We compute vertex velocities $\mathbf{v}$ 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 $\frac{d\mathbf{X}(t)}{dt}$ 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 $\mathbf{v}$. In order to obtain the unique solution from the flow constraint equation (2) and also maintain the smooth motion of the deformed model during the SFFD process, we add a smoothness constraint into the model. The vertex velocity variation inside a local region is minimized. This gives rise to minimizing the following objective function:

$$E = \int (\nabla \mathbf{f} \cdot \mathbf{v} + \frac{\partial f}{\partial t})^2 + \lambda(\|\nabla \mathbf{v}\|)^2 d\mathbf{x}, \qquad (3)$$

where λ is a Lagrangian multiplier. By discretizing the above objective function, (3) can be minimized iteratively. Considering a vertex k and its neighboring vertex set $\mathbf{Q}_k$ in an optimized mesh, $\mathbf{Q}_k = \{j | \overrightarrow{jk} \in \mathbf{M}\}$, where $\mathbf{M}$ 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 Section 4. The error of the flow constraint approximation is as follows,

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

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

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

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

$$E = \sum_k (c(k) + \lambda s(k)). \qquad (4)$$

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. That the derivative of E with respect to $v_x(k)$, $v_y(k)$, and $v_z(k)$ is equal to zero yields the following equations and (v_x, v_y, v_z) can be solved afterwards.

$$(\lambda + (\frac{\partial f}{\partial x})^2) v_x(k) + \frac{\partial f}{\partial x}\frac{\partial f}{\partial y} v_y(k) + \frac{\partial f}{\partial x}\frac{\partial f}{\partial z} v_z(k) = \lambda \bar{v}_x(k) - \frac{\partial f}{\partial x}\frac{\partial f}{\partial t},$$
$$(\lambda + (\frac{\partial f}{\partial y})^2) v_y(k) + \frac{\partial f}{\partial x}\frac{\partial f}{\partial y} v_x(k) + \frac{\partial f}{\partial y}\frac{\partial f}{\partial z} v_z(k) = \lambda \bar{v}_y(k) - \frac{\partial f}{\partial y}\frac{\partial f}{\partial t},$$
$$(\lambda + (\frac{\partial f}{\partial z})^2) v_z(k) + \frac{\partial f}{\partial x}\frac{\partial f}{\partial z} v_x(k) + \frac{\partial f}{\partial y}\frac{\partial f}{\partial z} v_y(k) = \lambda \bar{v}_z(k) - \frac{\partial f}{\partial z}\frac{\partial f}{\partial t},$$

where $(\bar{v}_x(k), \bar{v}_y(k), \bar{v}_z(k))$ is the average velocity $\bar{\mathbf{v}}(k)$ of all the neighboring vertices in $\mathbf{Q}_k$, $\bar{\mathbf{v}}(k) = \frac{1}{\|\mathbf{Q}_k\|} \sum_{j \in \mathbf{Q}_k} \mathbf{v}(j)$. Solving the above equation we can obtain the following iterative solution,

$$[v_x^{k+1}, v_y^{k+1}, v_z^{k+1}]^\top = [\bar{v}_x^k, \bar{v}_y^k, \bar{v}_z^k]^\top - \mu [\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}]^\top, \qquad (5)$$

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

Based on the above formulations, we have designed a dynamic continuous SFFD 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. Altering 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 where they originally resided.

At each time step Δt,

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

The SFFD can be an evolution process, which 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 be 1, the loop almost meets the stop conditions with only one time step. So the deformation is just like traditional free-form deformation of the static model. If the model needs to be extensively refined or optimized in order 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 as 0.1. Displaying the above sequence continuously, we can get an animation showing the dynamic deformation process.

4. SFFD MODEL OPTIMIZATION

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

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 deformation. Our method can generate additional triangles on the fly and only in regions that require more subdivision. Low curvature regions that were not previously subdivided still get chance to be refined during the deformation. As the SFFD deforms an object, the curvature of the affected surface is checked to see if subdivision is necessary. Triangles are added only when this criterion is met. Thus, the local, adaptive subdivision scheme used in this paper only adds the triangles in regions that require additional subdivision and, in addition, all deformed regions are checked to ensure that subdivision occurs on the appropriate regions. We also incorporate a complimentary decimation process which 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. This self-adaptive refinement strategy supports multi-resolution deformation of existing models.

Since our free-form deformation 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. There is much research work [8, 21] conducted in this field, producing several valuable algorithms.

We employ three mesh improvement operations, which include edge-split, edge-collapse, and edge-swap. Conditions under which the operations are valid are discussed in [8]. 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 keep increasing 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 solver for minimizing the objective function (4).

Ordering the operations this way seems to produce the best mesh at the end of the mesh improvement steps [21]. 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, making it appropriate for use in our scalar-field based free-form 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 SFFD operation.

5. SFFD SPACE CONSTRUCTION

In order to let users easily use our SFFD technique to perform free-form deformations on existing polygonal objects, our SFFD technique is equipped with three approaches for scalar field construction and deformation. We will detail them as follows.

Dynamic spline-based scalar fields [9] 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. In essence, the function is a hierarchical organization of the N scalar B-spline patches and has dynamic property since its coefficients are time-varying. Users can directly manipulate the scalar values to evolve the function. Please refer to [9] for the full detail about dynamic manipulation of spline-based volumetric implicit functions. Figure 3 shows an example using dynamic spline-based implicit functions.

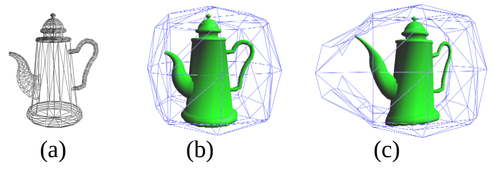

Figure 3: The teapot model (a) is embedded in the scalar field defined by a dynamic spline-based implicit function (b) and deformed by changing the scalar field as shown in (c) (We only show a single level set using coarse mesh in (b) and (c)).

Our system also allows users to interactively define skeletons, then the ***skeleton-based scalar field*** is generated as blending of field functions g_i of a set of skeletons s_i, $f(x,y,z) = \sum_{i=1}^{N} g_i(x,y,z)$, where the skeletons s_i can be any geometric primitive admitting a well defined distance function: points, curves, parametric surfaces, simple volumes, etc. The field functions g_i are decreasing functions of the distance to the associated skeleton: $g_i(x,y,z) = G_i(d(x,y,z,s_i))$, where $d(x,y,z,s_i)$ is the distance between (x,y,z) to s_i, and G_i can be defined for instance by pieces of polynomials.

We can enforce global and local control of an underlying scalar field in three separate ways: (1) defining or manipulating of 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.

We also employ the ***sketching*** technique for scalar field construction and modification. In this approach the manipulated scalar field is actually a signed/unsigned distance field. The sketched contour is the silhouette of the zero level set of the resulted distance field. This sketching technique can greatly ease the editing of scalar fields for designers. 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 using open curves, the resulted distance field is unsigned, neither interior nor exterior defined. The plane containing the 2D curve is called the drawing plane. Our system extrudes the contour along the perpendicular direction of the drawing plane until it meets the bounding space or a user-specified bounding region.

In practice, we only store the 2D distance field on the drawing plane since other slices of the 3D distance field along the perpendicular direction are exactly the same. 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, then assign the perpendicular foot's distance value to the point. This method avoids to compute the entire 3D distance field. 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. 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 4 corner distance values stored per quad using standard bilinear interpolation.

6. SFFD OPERATIONS

Given the novel SFFD technique and the simple, intuitive, and efficient scalar field construction and deformation approaches, users can easily perform various SFFD operations on existing models.

For ***bending*** operations, users may first draw a set of skeletons to define the source scalar field. Figure 4(a)(b) shows a bending operation on a 3D nozzle model. In that operation, a curve skeleton is used. The user just simply sketches a straight line segment near the center of the object and a bending curve one to define two distance fields. The object is then deformed according to the difference of these two fields.

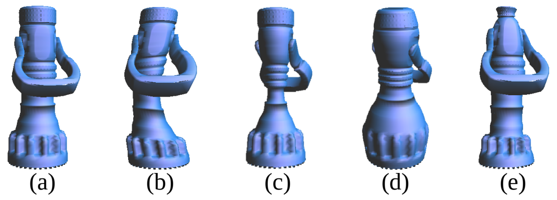

Figure 4: (a) Original model; (b) bending; (c) shrinking the middle part; (d) inflation on both ending parts; (e) shrinking the part near the bottom and the top parts.

For ***shrinking*** and ***inflation***, users can define two sets of skeletons (namely, a source one and a target one) to define scalar fields. The embedded object will inflate or shrink according to the field deformation from the source to target one. Or users can just modify some parameters of the source skeletons to perform the deformation. For examples shown in Figure 4(c-e), the user employs Gaussian blobs as skeletons, the nozzle model inflates or shrinks in several forms.

Users can perform ***tapering*** operations by simply sketching strokes. As shown in Figure 5, 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 and target distance fields are generated locally based on these two strokes. Then the localized front part is then tapered according to the field change.

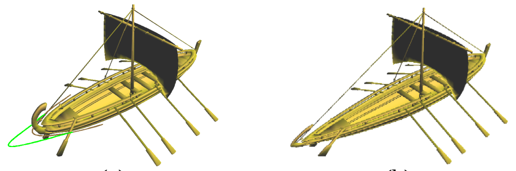

Figure 5: Tapering on the ship model.

For ***squeezing*** operations, users may first sketch a closed stroke to define a distance field and then modified this distance field by adding another stroke. In this operation, we only perform deformation on those vertices with positive distance values. For an example as shown in Figure 6(a), 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 Section 5. Then the user sketches another stroke (in green) to deform the distance field. The embedded dinosaur is then squeezed as shown in Figure 6(b) according to the SFFD. Users can also completely sketch another new stroke to define a target distance field instead of creating it by modifying the existing stroke.

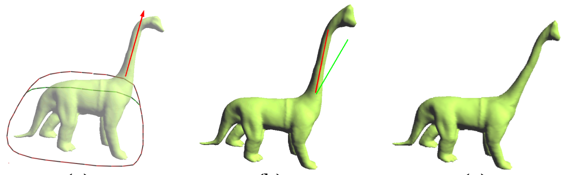

Figure 6: A dinosaur is deformed with the body squeezed (b), the neck stretched (b), and the position of the neck moved (c).

For ***stretching***, users can use force-based tools [9] to directly drag the embedding scalar field, which is equivalent to drag the embedded object directly. In essence, force-based tools alter the scalar field along the force vector, which result in the stretching effect of the embedding model along the force vector. For an example shown in Figure 6, the user picks up a vertex around the bottom of the dinosaur's neck then drags along the arrow to produce a stretching deformation on the dinosaur's neck.

Users can alter the part of an object to another location by using ***moving*** operations. Please see Figure 6(b)(c) for an example. 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 Figure 6(c) due to this field change.

Users can also apply ***embossing*** and ***engraving*** operations on an object easily. In these operations, users can paint a grey scale image or use an existing one to define an embedding space on the object. Figure 7(a) shows an image of global map. A surface S is obtained by projecting the sampling points along the object's normals with corresponding grey scale values. Our system then generate a local distance field in the region of interest according to the surface. The scalar values are the shortest distances to the surface. The image is similar to the concept of displacement map [11]. The difference is that we do not need to explicitly map the displacement to each point of the base model. In embossing and engraving operations, the source scalar field around the base model is initialized to zero everywhere. So 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. As shown in Figure 7(b), this image is embossed onto a ”soap” shape by using the sketch-based space construction approach together with the SFFD technique. If we reverse the distance field along the opposite direction, we can easily perform the engraving operations on the models.

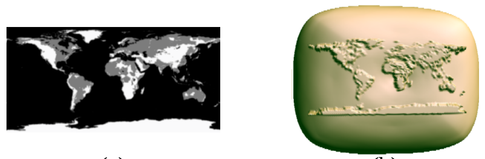

Figure 7: (a) shows a grey scale image, which is used to define a scalar field. (b) shows the deformed object.

It is very easy to use SFFD to create sharp creases on the embedded models, which is generally difficult when using traditional FFD. Users can perform ***creasing*** operations by simply sketching strokes. As shown in Figure 8, the user makes two shape creases on the both sides of the deformed cup model. The user first localizes the region where the creases will be formed. Then the user 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 other two open curves shown in green. The target unsigned distance fields are generated based on these two curves in the same regions. The SFFD forms the sharp creases according to the field change from the source one to target one.

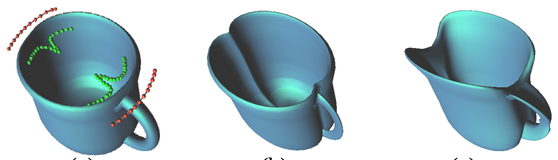

Figure 8: (a) shows the original model and the sketched strokes around the both sides of the cup. The red ones are strokes for generating the source distance field, while the green ones are strokes for generating the target distance field. (b) shows two sharp creases formed on the both sides of the cup. (c) Further deformation using two analytic implicit functions.

We can locally perform the aforementioned SFFD operations on polygonal models by only allowing the movement of the vertices, where the scalar value evaluates within a specified scope of the iso-values. For other vertices, they will not move during the deformation. Users can specify any implicit function to localize the part of the models to be deform. In our prototype system, we provide users three types of primitives to localize regions, which include rectangular box, cylinder, and sphere. Combining these primitives, users are able to localize any part of embedded models.

Our technique provides a general SFFD mechanism in the sense that it can easily accommodate various geometric constraints. To enforce constraints, we can simply augment the original objective function E with the constraint energy E_c, $E_n = E + \gamma \cdot E_c$, where E_n denotes the new, overall objective function and E_c denotes the additional cost term introduced by added constraints. Note that E_c can be a linear combination of several cost functionals. A standard implicit iterative method is employed to numerically compute the minimization of the overall objective function. 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. 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 purpose in SFFD operations.

With all the available operations and constraints, we perform some interesting deformations on the mannequin model as shown in Figure 9. During the deformations, some facial features are maintained by enforcing constraints such as normal and curvature constraints. In Figure 9(a)(b), we drop the smoothness constraints in (3) during the deformation, only allowing the vertex velocities along the specified directions, in order to produce a bump-like hair effect.

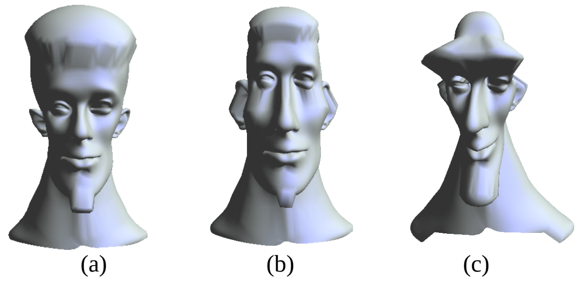

Figure 9: The mannequin model is deformed by using the available SFFD operations and enforcing constraints.

7. CONCLUSION

We have articulated a novel scalar-field based free-form deformation, or SFFD, methodology based on flow constraints and scalar-field functions. The new SFFD paradigm is fundamentally different from traditional FFD techniques because we employ scalar fields as FFD embedding spaces. A scalar field based embedding space is of diverse type and its space definition is much more simple, yet both powerful and intuitive for solid modeling applications. In our prototype system, we developed several easy-to-use techniques for efficiently constructing and manipulating the space as well as flexibly interacting with various geometric shapes. With SFFD method, users can directly sketch a scalar field of an implicit function to control the deformation of any embedded model. In addition, the embedding space can be of complicated geometry and an arbitrary topology. The deformation velocity for any model point can be either very general or constrained subject to any user-specified requirement. Therefore, our SFFD technique affords a larger number of shape deformation types. Furthermore, the embedded model has self-adaptive optimization capability throughout the SFFD process in order to accommodate versatile deformations, maintain the mesh quality, and preserve shape features. We have conducted a large number of experiments which demonstrate that our new SFFD technique is powerful, flexible, natural, and intuitive for shape design in solid modeling, animation, and interactive graphics.

8. REFERENCES

[1] J. Bærentzen and N. Christensen. Volume sculpting using the level-set method. In *Proceedings of International Conference on Shape Modelling and Applications*, pages 175–182, 2002.

[2] J. Bloomenthal and B. Wyvill. Interactive techniques for implicit modeling. In *Proceedings of 1990 Symposium on Interactive 3D Graphics*, pages 109–116, 1990.

[3] D. E. Breen and R. T. Whitaker. A level-set approach for the metamorphosis of solid models. *IEEE Trans. on Visualization and Computer Graphics*, 7(2):173–192, 2001.

[4] Y.-K. Chang and A. P. Rockwood. A generalized de casteljau approach to 3D free-form deformation. In *SIGGRAPH '94*, pages 257–260, 1994.

[5] S. Coquillart. Extended free-form deformation: A sculpting tool for 3D geometric modeling. In *SIGGRAPH 90 Proceedings*, pages 187–196, 1990.

[6] B. Crespin. Implicit free-form deformations. In *Proceedings of Implicit Surfaces '99*, pages 17–23, 1999.

[7] M. Desbrun and M. P. Cani. Active implicit surface for animation. In *Proceedings of Graphics Interface*, pages 143–150, 1998.

[8] H. Hoppe, T. Derose, T. Duchamp, J. McDonald, and W. Stuetzle. Mesh optimization. In *SIGGRAPH '93*, pages 19–26, 1993.

[9] J. Hua and H. Qin. Haptics-based volumetric modeling using dynamic spline-based implicit functions. In *Proceedings of IEEE Symposium on Volume Visualization and Graphics*, pages 55–64, 2002.

[10] X. Jin, Y. F. Li, and Q. Peng. General constrained deformations based on generalized metaballs. *Computer & Graphics*, 24(2):219–231, 2000.

[11] V. Krishnamurthy and M. Levoy. Fitting smooth surfaces to dense polygon meshes. In *SIGGRAPH '96*, pages 313–324, 1996.

[12] F. Lazarus, S. Coquillart, and P. Jancene. Axial deformations: An intuitive deformation technique. *Computer-Aided Design*, 26(8):607–612, 1994.

[13] R. MacCracken and K. I. Joy. Free-form deformations with lattices of arbitrary topology. In *SIGGRAPH 96 Proceedings*, pages 181–188, 1996.

[14] K. Museth, D. E. Breen, R. T. Whitaker, and A. H. Barr. Level set surface editing operators. In *SIGGRAPH '02*, pages 330–338, 2002.

[15] S. Osher and J. Sethian. Fronts propagating with curvature-dependent speed: Algorithms based on hamilton-jacobi formulations. *Journal of Computational Physics*, 79:12–49, 1988.

[16] A. Raviv and G. Elber. Three dimensioinal freeform sculpting via zero sets of scalar trivariate functions. In *Proceedings of Solid Modeling '99*, pages 246–257, 1999.

[17] B. Schmitt, A. Pasko, and C. Schlick. Constructive modeling of frep solids using spline volumes. In *Proceedings of Solid modeling '01*, pages 321–322, 2001.

[18] T. Sederberg and S. R. Parry. Free-form deformation of solid geometric models. In *SIGGRAPH '86*, pages 151–160, 1986.

[19] K. Singh and E. Fiume. Wires: a geometric deformation technique. In *SIGGRAPH '98*, pages 405–414, 1998.

[20] G. Turk and J. F. O'Brien. Modelling with implicit surfaces that interpolate. *ACM Trans. on Graphics*, 21(4):855–873, 2002.

[21] W. Welch. *Serious putty: Topological design for variational curves and surfaces*. PhD thesis, Carnegie Mellon University, June 1995.

[22] G. Wyvill, C. McPheeters, and B. Wyvill. Data structure for soft objects. *The Visual Computer*, 2(4):227–234, 1988.